**Міністерство освіти і науки України**
**Тернопільський національний технічний університет імені Івана Пулюя**

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету )
Кафедра комп'ютерних наук
(повна назва кафедри)

# КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(назва освітнього ступеня)

на тему: Розробка вебзастосунку для фільтрації зображень методом згортки із використанням засобів WebGPU

Виконав: студент IV курсу, групи СН-43
спеціальності 122 Комп’ютерні науки
(шифр і назва спеціальності)

_____ Шимків І.М.
(підпис) (прізвище та ініціали)

Керівник _____ Фриз М.Є.
(підпис) (прізвище та ініціали)

Нормоконтроль _____ Липак Г.І.
(підпис) (прізвище та ініціали)

Завідувач кафедри _____ Боднарчук І.О.
(підпис) (прізвище та ініціали)

Рецензент _____ Варавін А.В.
(підпис) (прізвище та ініціали)

Тернопіль
2025

Міністерство освіти і науки України
**Тернопільський національний технічний університет імені Івана Пулюя**

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)

Кафедра комп'ютерних наук
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Боднарчук І.О.
(підпис) (прізвище та ініціали)
«_27_» ____січня____ 2025 р.

# ЗАВДАННЯ
## НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня Бакалавр
(назва освітнього ступеня)

за спеціальністю 122 Комп'ютерні науки
(шифр і назва спеціальності)

Студенту Шимкову Івану Михайловичу
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка вебзастосунку для фільтрації зображень методом згортки із використанням засобів WebGPU

Керівник роботи Фриз Михайло Євгенович, кандидат технічних наук, доцент кафедри КН
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від «*07*» *травня* 2025 року № *4/7-444*

2. Термін подання студентом завершеної роботи 27 червня 2025 р.

3. Вихідні дані до роботи Літературні та Інтернет-джерела, які застосовані для розробки вебзастосунку для фільтрації зображень методом згортки із використанням засобів WebGPU

4. Зміст роботи (перелік питань, які потрібно розробити)
Вступ. 1. Аналіз предметної області та постановка завдання. 1.1. Кольорові цифрові зображення. 1.1.1. Кольорові зображення у моделі RGBA. 1.1.2. Зв'язок моделей RGBA та RGB. 1.1.3. Монохромні зображення. 1.2. Метод згортки. 1.3. Паралельні обчислення із використанням GPU. 1.4. Обгрунтування вибору графічного API. 1.4.1. WebGL та його недоліки. 1.4.2. WebGPU як заміна WebGL. 1.5. Розгляд альтернатив вебзастосунку. 1.6. Постановка завдання. 1.7. Висновки до першого розділу. 2. Вибір технологій та проєктування інтерфейсу вебзастосунку. 2.1. Вибір фронтенд-фреймворку. 2.1.1. React. 2.1.2. Vue.js. 2.1.3. Svelte. 2.2. Додаткові інструменти. 2.2.1 SvelteKit. 2.2.2. Tailwind CSS. 2.3. Проєктування інтерфейсу вебзастосунку. 2.4. Проєктування структури вебзастосунку. 2.5. Висновки до другого розділу. 3. Розробка та демонстрація вебзастосунку. 3.1. Розробка вебзастосунку. 3.1.1. Розробка компонента заголовку. 3.1.2. Розробка компонента бічної панелі. 3.1.3. Розробка компонента попереднього перегляду. 3.1.4. Розробка зв'язуючого компонента. 3.2. Демонстрація вебзастосунку та результатів його роботи. 3.2.1. Фільтр простого розмиття. 3.2.2. Фільтр розмиття Гауса. 3.2.3. Фільтр тиснення. 3.2.4. Фільтр наведення контурів. 3.2.5. Оператори Превітта. 3.2.6. Оператори Собеля. 3.2.7. Фільтр наведення різкості. 3.3. Висновки до третього розділу. 4. Безпека життєдіяльноті, основи охорони праці. Висновки. Перелік джерел.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)
1. Вступ. 2. Підбір фронтенд-фреймворку. 3. Додаткові технології. 4. Проєктування інтерфейсу вебзастосунку. 5. Проєктування структури вебзастосунку. 6. Демонстрація інтерфейсу вебзастосунку. 7. Фільтр розмиття. 8. Фільтр тиснення. 9. Оператор Собеля. 10. Оператор Превітта. 11. Фільтр наведення контурів. 12. Фільтр наведення різкості.

6. Консультанти розділів роботи

| Розділ | Прізвище, ініціали та посада консультанта | Підпис, дата | |
|---|---|---|---|
| | | завдання видав | завдання прийняв |
| Безпека життєдіяльності, основи охорони праці | Сенчишин В.С., кандидат технічних наук, доцент кафедри МТ | | |

7. Дата видачі завдання 27 січня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

| № з/п | Назва етапів роботи | Термін виконання етапів роботи | Примітка |
|---|---|---|---|
| 1. | Ознайомлення з завданням до кваліфікаційної роботи | 27.01.2025 | *Виконано* |
| 2. | Підбір та опрацювання літературних джерел по темі кваліфікаційної роботи | 28.01.2025-08.02.2025 | *Виконано* |
| | | | |
| 3. | Виконання дослідження щодо розробки вебзастосунку | 09.02.2025-16.03.2025 | *Виконано* |
| | Розроблення вебзастосунку | | |
| | | | |
| 4. | Оформлення розділу «Аналіз предметної області та | 17.03.2025-21.03.2025 | *Виконано* |
| | постановка завдання» | | |
| 5. | Оформлення розділу «Вибір технологій та | 22.03.2025-04.04.2025 | *Виконано* |
| | проєктування вебзастосунку» | | |
| 6. | Оформлення розділу «Розробка та демонстрація | 05.04.2025-17.04.2025 | *Виконано* |
| | вебзастосунку» | | |
| 7. | Виконання завдання до підрозділу «Безпека життєдіяльності» | 18.04.2025-08.05.2025 | *Виконано* |
| 8. | Виконання завдання до підрозділу «Основи охорони | 09.05.2025-21.05.2025 | *Виконано* |
| | праці» | | |
| 9. | Оформлення кваліфікаційної роботи | 28.05.2025-10.06.2025 | *Виконано* |
| 10. | Нормоконтроль | 16.06.2025 | *Виконано* |
| 11. | Перевірка на плагіат | 18.06.2025-19.06.2025 | *Виконано* |
| 12. | Попередній захист кваліфікаційної роботи | 20.06.2025 | *Виконано* |
| 13. | Захист кваліфікаційної роботи | 29.06.2025 | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

Студент _____ Шимків І.М.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Фриз М.Є.
(підпис) (прізвище та ініціали)

# АНОТАЦІЯ

Розробка вебзастосунку для фільтрації зображень методом згортки із використанням засобів WebGPU // Кваліфікаційна робота освітнього рівня «Бакалавр» // Шимків Іван Михайлович // Тернопільський національний технічний університет імені Івана Пулюя, факультет комп'ютерно-інформаційних систем і програмної інженерії, кафедра комп'ютерних наук, група СН-43 // Тернопіль, 2025 // С. 62, рис. – 33, табл. – 0, кресл. – 0, додат. – 16, бібліогр. – 45.

***Ключові слова:*** вебзастосунок, фільтрація зображень, лінійна згортка, паралельні обчислення, WebGPU, Svelte, JavaScript.

Кваліфікаційна робота присвячена розробці вебзастосунку для фільтрації зображень методом лінійної згортки із використанням WebGPU.

У першому розділі кваліфікаційної роботи описано математичну модель кольорового зображення та стисло розглянуто операцію лінійної згортки. Висвітлено роль графічних процесорів у обчисленнях та обґрунтовано вибір WebGPU для розробки вебзастосунку. Розглянуто наявні альтернативні рішення. Проаналізовано їхні переваги та недоліки і виконано постановку завдання на основі одержаних відомостей.

У другому розділі кваліфікаційної роботи підібрано технології для розробки вебзастосунку. Виконано порівняння фронтенд-фреймворків та вибір найоптимальнішого варіанту. Описано використані додаткові технології. Спроєктовано користувацький інтерфейс вебзастосунку.

У третьому розділі кваліфікаційної роботи описано процес розробки вебзастосунку. Продемонстровано вигляд і роботу створеного вебзастосунку.

Об'єкт дослідження: процес використання графічного процесора для обчислень у середовищі веб.

Предмет дослідження: застосування WebGPU для лінійної фільтрації зображень.

# ANNOTATION

Development of a Web Application for Image Filtering Using Convolution Methods with WebGPU // Qualification work of the educational level «Bachelor» // Shymkiv Ivan // Ternopil Ivan Puluj National Technical University, Computer and Information Systems and Software Engineering Faculty, Computer Sciences Department, group SN-43 // Ternopil, 2025 // P. 62, fig. – 33, tabl. – 0, chair. – 0, annexes. – 16, references – 45.

***Keywords:*** web application, image filtering, linear convolution, parallel computing, WebGPU, Svelte, JavaScript.

The qualification work is dedicated to the development of a web application for image filtering using the linear convolution method with WebGPU.

The first section of the qualification work describes the mathematical model of a color image and briefly considers the linear convolution operation. The role of graphics processors in scientific computing is highlighted and the choice of WebGPU for developing a web application is justified. Existing alternative solutions are considered. Their advantages and disadvantages are analyzed and the task is formulated based on the information obtained.

In the second section of the qualification work, technologies for developing a web application are selected. Front-end frameworks are compared and the most optimal option is selected. Additional technologies used are described. The user interface of the web application is designed.

In the third section of the qualification work, the process of developing a web application is described. The appearance and operation of the created web application are demonstrated.

Research object: the process of using a graphics processor for computing in a web environment.

Research subject: using WebGPU for linear image filtering.

# ПЕРЕЛІК СКОРОЧЕНЬ

API (англ. Application Programming Interface) – прикладний програмний інтерфейс.

CPU (англ. Central Processing Unit) – центральний процесор.

CSS (англ. Cascading Style Sheets) – каскадні таблиці стилів.

GPU (англ. Graphical Processing Unit) – графічний процесор.

HTML (англ. HyperText Markup Language) – мова розмітки гіпертексту.

RGB (англ. Red, Green, Blue) – червоний, зелений, синій.

RGBA (англ. Red, Green, Blue, Alpha) – червоний, зелений, синій, альфа.

WGSL (англ. WebGPU Shading Language) – мова шейдерів WebGPU.

# ЗМІСТ

ВСТУП ..... 8
РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАВДАННЯ ..... 9
1.1 Кольорові цифрові зображення ..... 9
1.1.1 Кольорові зображення у моделі RGBA ..... 9
1.1.2 Зв’язок моделей RGBA та RGB ..... 11
1.1.3 Монохромні зображення ..... 12
1.2 Метод згортки ..... 13
1.3 Паралельні обчислення із використанням GPU ..... 14
1.4 Обґрунтування вибору графічного API ..... 15
1.4.1 WebGL та його недоліки ..... 15
1.4.2 WebGPU як заміна WebGL ..... 16
1.5 Розгляд альтернатив вебзастосунку ..... 18
1.6 Постановка завдання ..... 23
1.7 Висновки до першого розділу ..... 23
РОЗДІЛ 2. ВИБІР ТЕХНОЛОГІЙ ТА ПРОЄКТУВАННЯ ІНТЕРФЕЙСУ ВЕБЗАСТОСУНКУ ..... 25
2.1 Вибір фронтенд-фреймворку ..... 25
2.1.1 React ..... 25
2.1.2 Vue.js ..... 27
2.1.3 Svelte ..... 29
2.2 Додаткові інструменти ..... 31
2.2.1 SvelteKit ..... 31
2.2.2 Tailwind CSS ..... 31
2.3 Проєктування користувацького інтерфейсу вебзастосунку ..... 32
2.4 Проєктування структури вебзастосунку ..... 33
2.5 Висновки до другого розділу ..... 34
РОЗДІЛ 3. РОЗРОБКА ТА ДЕМОНСТРАЦІЯ ВЕБЗАСТОСУНКУ ..... 35

3.1 Розробка вебзастосунку ..... 35
3.1.1 Розробка компонента заголовку ..... 37
3.1.2 Розробка компонента бічної панелі ..... 39
3.1.3 Розробка компонента попереднього перегляду ..... 40
3.1.4 Розробка зв’язуючого компонента ..... 40
3.2 Демонстрація вебзастосунку та результатів його роботи ..... 41
3.2.1 Фільтр простого розмиття ..... 43
3.2.2 Фільтр розмиття Гауса ..... 44
3.2.3 Фільтр тиснення ..... 45
3.2.4 Фільтр наведення контурів ..... 46
3.2.5 Оператори Превітта ..... 47
3.2.6 Оператори Собеля ..... 49
3.2.7 Фільтр наведення різкості ..... 50
3.3 Висновки до третього розділу ..... 51
РОЗДІЛ 4. БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ ..... 52
4.1 Психологічні чинники небезпеки ..... 52
4.2 Показники ефективності та заходи щодо покращення умов охорони праці ..... 54
4.3 Висновки до четвертого розділу ..... 56
ВИСНОВКИ ..... 57
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ ..... 58
ДОДАТКИ

# ВСТУП

**Актуальність теми.** В процесі розвитку апаратного забезпечення дедалі більш зростають вимоги до програмного забезпечення загалом та до вебзастосунків зокрема. Це надає переваги, серед яких незалежність від користувацької платформи (що спрощує розробку) та більш тісна інтеграція між пристроями (що спрощує використання кінцевими користувачами).

**Мета і задачі дослідження.** Метою цієї кваліфікаційної роботи освітнього рівня «Бакалавр» є створення вебзастосунку, призначеного для фільтрації зображень методом лінійної згортки із використанням засобів WebGPU задля покращення продуктивності їх обробки.

Для виконання поставленої мети потрібно виконати такі завдання:

– проаналізувати наявні альтернативні рішення та визначити їхні переваги і недоліки;

– виконати постановку завдання до розробки вебзастосунку із урахуванням одержаних відомостей про альтернативи;

– дослідити принципи використання графічних процесорів у обчислювальних задачах;

– розглянути засоби, що надають доступ до обчислювальних ресурсів графічного процесора у браузерному середовищі та обрати технологію за результатами аналізу;

– обрати технології для розробки користувацького інтерфейсу вебзастосунків і дослідити їхні особливості та способи взаємодії.

**Практичне значення одержаних результатів.** Створений вебзастосунок дозволить підвищити продуктивність обробки зображень у середовищі веб та використати наявні обчислювальні ресурси ефективніше, адже графічні процесори стрімко розвиваються та набувають дедалі більшої потужності, яку доцільно використовувати за призначенням – для обробки графіки.

Таким чином створений вебзастосунок дозволить оптимально використовувати ресурси пристрою користувача.

# РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАВДАННЯ

## 1.1 Кольорові цифрові зображення

Дискретизація зображень – операція поділу неперервного зображення на частини із чітко визначеними межами. Розрізняють два типи дискретизації:

– рівномірна (поділ на однакові частини прямокутної форми);
– нерівномірна (поділ на частини довільної форми) [1].

При оцифруванні зображень застосовується також операція квантування, мета якої полягає у обмеженні кількості кольорів до скінченної. Квантування теж може бути рівномірним чи нерівномірним аналогічно до дискретизації [2].

Для рівномірного квантування можна використати довільне розбиття діапазону кольорів на скінченну кількість рівнопотужних класів із подальшою заміною кольорів зображення довільними фіксованими представниками кожного з них. Квантування задля спрощення приймемо неявним.

У межах роботи розглядатимемо тільки рівномірно дискретизовані та рівномірно квантовані зображення як найбільш поширені. Їхньою перевагою є зокрема і простота математичної моделі, адже рівний розмір частин дозволяє знехтувати ним.

### 1.1.1 Кольорові зображення у моделі RGBA

Для представлення кольорових зображень розглядатимемо модель RGBA, бо вона є однією із найпоширеніших та використовується графічними API для роботи із зображеннями з підтримкою прозорості [3]. Приклад кольорового зображення наведено на рисунку 1.1.

Рисунок 1.1 – Кольорове зображення

Зображення є прикладом лінійного сигналу, що дозволяє застосовувати до них техніки ЦОС (цифрової обробки сигналів) [4-7].

Нехай, $\mathbb{N} = \{x \in \mathbb{Z} \mid x \geq 0\}$ – множина натуральних чисел, $\mathbb{N}_n = \{x \in \mathbb{N} \mid x < n\}$ при ненульовому $n \in \mathbb{N}$. Тоді математичною моделлю двовимірного кольорового зображення у RGBA вважатимемо відображення із системи цілочислових координат у множину нормалізованих значень рівнів каналів:

$$I\colon \mathbb{N} \times \mathbb{N} \to [0,1]^c,\ (x,y) \mapsto z = (r,g,b,a), \tag{1.1}$$

де $c = 4$ – кількість каналів;

$r$ – інтенсивність червоного кольору;

$g$ – інтенсивність зеленого кольору;

$b$ – інтенсивність синього кольору;

$a$ – значення альфа-каналу (рівень непрозорості).

Пікселем називатимемо найменшу неподільну частину зображення, представлену елементом вищезгаданої системи координат.

Цифрове зображення – звуження $I|_{\mathbb{N}_w \times \mathbb{N}_h}$, де $w$ та $h$ – ширина і висота зображення у пікселях відповідно. Надалі без втрати загальності вважатимемо їх фіксованими, коли не вказано інше.

### 1.1.2 Зв'язок моделей RGBA та RGB

Іншою поширеною, але менш загальною моделлю є RGB, яка відрізняється від RGBA відсутністю альфа-каналу [8]. Кожному кольору в моделі RGB відповідає клас кольорів моделі RGBA із такими ж рівнями колірних каналів, тобто

$$(r_1, g_1, b_1, a_1) \sim (r_2, g_2, b_2, a_2) \Leftrightarrow (r_1, g_1, b_1) = (r_2, g_2, b_2). \qquad (1.2)$$

Таким чином задано відношення еквівалентності $\sim$, для якого комутує діаграма, наведена на рисунку 1.2.

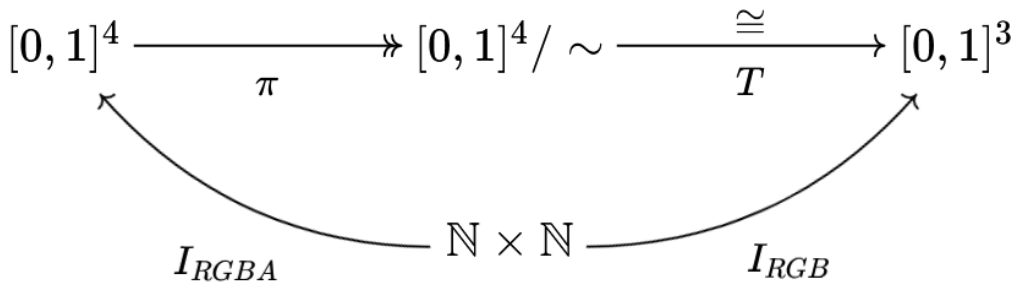


Рисунок 1.2 – Комутативна діаграма переходу від моделі RGBA до RGB

На вищенаведеному рисунку $\pi$ – канонічна проєкція, $T$ – відображення, яке співставляє клас кольорів RGBA їхньому еквіваленту в RGB:

$$T\colon [0,1]^4/\sim \to [0,1]^3,\ [(r,g,b,a)]_\sim \mapsto (r,g,b). \qquad (1.3)$$

Отже, за формулою (1.3) клас кольорів RGBA має єдине представлення у RGB.

### 1.1.3 Монохромні зображення

Зображення називаємо монохромним у випадку, коли воно містить лише один канал. Тоді воно складається із відтінків одного кольору.

Кольорове зображення у моделі RGB можна перетворити на монохромне (див. рисунок 1.3) за допомогою відображення

$$G\colon [0,1]^3 \to [0,1],\ (r,g,b) \mapsto 0.2126 \cdot r + 0.7152 \cdot g + 0.0722 \cdot b, \qquad (1.4)$$

де $r$ – інтенсивність червоного кольору;

$g$ – інтенсивність зеленого кольору;

$b$ – інтенсивність синього кольору.

Рисунок 1.3 – Монохромне зображення, одержане із кольорового

За формулою (1.4) можна обчислити значення яскравості кольору [9].

При розкладі зображення у RGB на колірні канали одержуємо три монохромні зображення (див. рисунок 1.4).

Рисунок 1.4 – Колірні канали зображення в RGB(-A)

Монохромні зображення використовуються у випадках, коли важлива лише інтенсивність кольору. Застосовуються, наприклад, у медицині. Зокрема для рентгенівських знімків, що дозволяє розглядати будову та структуру кісток [10].

### 1.2 Метод згортки

Операцією лінійної згортки називатимемо відображення, задане виразом

$$(I * K)_c(x, y) = \sum_{i=-r}^{r} \sum_{j=-r}^{r} K(i, j) \cdot \tilde{I}_c(x + i, y + j)\,, \tag{1.6}$$

де $K$ – дійсна матриця непарного порядку $(2r + 1)$ для $r \in \mathbb{N}\backslash\{0\}$;

$c$ – канал зображення;

$\tilde{I}$ – розширення зображення за межі системи координат.

Методом лінійної згортки можна застосовувати численну кількість фільтрів до зображення, використовуючи лише арифметичні операції суми й добутку, зокрема такі:

- розмиття;
- різкість;
- визначення країв;

– оператор Превітта;

– оператор Соболя.

Таким чином метод лінійної згортки є зручним інструментом у галузі цифрової обробки зображень [11]. До його переваг належить також простота програмної реалізації.

### 1.3 Паралельні обчислення із використанням GPU

Паралельні обчислення – це процес поділу завдання на підзавдання із подальшим одночасним виконанням їх множинною кількістю процесорів. При цьому розрізняють два типи обчислювальних середовищ – гомогенні та гетерогенні.

Гомогенними називають системи, де всі процесори однакового типу та мають подібні можливості. У цьому випадку кілька ідентичних процесорів працюють разом для паралельного виконання завдань. З іншого боку, гетерогенні обчислення включають систему, яка поєднує різні типи процесорів, такі як центральні процесори, графічні процесори, програмовані вентильні матриці або спеціалізовані прискорювачі.

Гомогенні обчислення можна обрати, коли робоче навантаження добре підходить для паралельного виконання на кількох ідентичних процесорах. Це спрощує програмування, оскільки один і той самий код може виконуватися на всіх процесорах, а балансування навантаження між процесорами є простим (на відміну від гетерогенних середовищ). Однорідні обчислення використовуються зокрема в кластерах високопродуктивних обчислень та розподілених системах, які потребують масштабної паралельної обробки [12].

У наукових задачах GPU виконують роль обчислювальних прискорювачів загального призначення, хоча початково були призначеними для обробки графічних даних.

Таке їх застосування можливе завдяки побудові гетерогенного середовища на основі CPU та GPU. Тобто CPU виконує основну програму, а GPU

використовується для виконання певних функцій, що потребують великої кількості обчислювальних ядер для ефективного виконання.

У той час, як CPU має одиниці або десятки обчислювальних ядер, GPU має тисячі. В обох випадках обчислювальні ядра можна використовувати паралельно. Для певних робочих навантажень, таких як обробка зображень, навчання штучних нейронних мереж та розв'язання диференціальних рівнянь, код, що використовує GPU, може значно перевершити продуктивність коду, що виконується тільки CPU. Алгоритми зі складною логікою виконання та розгалуженнями, як правило, краще працюють із CPU [13].

## 1.4 Обґрунтування вибору графічного API

Графічні API надають додаткам прямий доступ до GPU пристрою. Завдяки ним додатки можуть використовувати GPU для швидкого рендерингу складних сцен та паралельного виконання обчислювальних завдань [14].

### 1.4.1 WebGL та його недоліки

Значного поширення у веброзробці набув API WebGL (див. рисунок 1.5), заснований на OpenGL ES, адже початково GPU не використовувались вебсайтами для промальовування графічних елементів, та із розвитком технологій з’явилася така необхідність. Тому WebGL свого часу став проривом, надавши можливості та синтаксис, близькі до OpenGL, але у середовищі веб.


Рисунок 1.5 – Логотип WebGL

Попри це WebGL має значні недоліки, серед яких:

– синхронність (більшості) операцій;
– використання глобального стану;
– високий рівень абстракції.

Усі вищенаведені недоліки є прямим наслідком того, що WebGL створено на базі OpenGL ES. Та із цим же пов’язана ще одна суттєва проблема – відсутність підтримки обчислювальних шейдерів. Під проводом компанії Intel виконувалась робота над специфікацією WebGL 2.0 Compute, але через складність реалізації від ідеї було вирішено відмовитись, натомість зосередившись над більш низькорівневою та сучасною заміною WebGL – проектом WebGPU [15].

### 1.4.2 WebGPU як заміна WebGL

WebGPU (див. рисунок 1.6) – це новий низькорівневий графічний API, розроблюваний під проводом комітету W3C, що призначений для використання можливостей сучасних GPU [16].

Рисунок 1.6 – Логотип WebGPU

До його переваг належать:

– асинхронність операцій;
– імутабельність (незмінність) об’єктів;
– відсутність глобального стану;

– підтримка обчислювальних шейдерів;
– акцент на портативність;
– надання стеку викликів [17].

Асинхронність операцій є ключовою для досягнення високої продуктивності, тоді як імутабельність та відсутність глобального стану знижують ризик виникнення помилок: імутабельність спрощує розробку багатопотокових програм, а відсутність стану допомагає уникнути помилок, пов’язаних із його налаштуваннями.

Підтримка обчислювальних шейдерів натомість сприяє розвитку високопродуктивних обчислень у середовищі браузера. При використанні WebGL навіть операція додавання двох чисел потребує імітації обчислювального шейдера за допомогою двох інших типів – фрагментного та вершинного. Та окрім цього, потрібно також виконувати операцію графічного рендерингу, що не призначена для обчислень за дизайном. Тоді як у випадку WebGPU можна застосувати саме обчислювальний шейдер для цього (див. лістинг 1.1).

Лістинг 1.1 – WGSL-шейдер для додавання двох чисел

```
@group(0) @binding(0)
var<storage, read> inputBuffer: array<f32>;

@group(0) @binding(1)
var<storage, read_write> outputBuffer: array<f32>;

@compute @workgroup_size(1)
fn main(@builtin(global_invocation_id) id: vec3<u32>) {
  let a = inputBuffer[0];
  let b = inputBuffer[1];

  outputBuffer[0] = a + b;
}
```

В такому разі використовуватиметься саме спеціалізований обчислювальний конвеєр [18].

WebGPU використовує власну мову шейдерів WGSL та підтримує тільки її. З одного боку це можна вважати за недолік, та з іншого варто враховувати і те, що зусилля спільноти зосереджено саме на ній, завдяки чому її активно

розвивають. WGSL має синтаксис, заснований на Rust, але зберігає риси GLSL (OpenGL Shading Language).

GLSL є найбільш популярною мовою розробки шейдерів і початково використовувалась Vulkan, але накладає обмеження, пов’язані зокрема й зі зворотною сумісністю та є застарілою. Із цієї причини навіть у проектах, що використовують Vulkan, поступово витісняється мовою HLSL (High-Level Shading Language), розвиток якої продовжується [19].

Станом на період виконання цієї кваліфікаційної роботи WebGPU все ще є експериментальною технологією й підтримується не усіма браузерами [20], але набуває дедалі більшого поширення завдяки активному впровадженню зокрема у браузерах на базі Chromium, як-от Google Chrome та Microsoft Edge [21].

Отже, WebGPU навіть у експериментальному статусі вже є перспективною технологією, що після переходу до статусу Baseline, дозволятиме розробляти високопродуктивні графічні вебзастосунки. Тому саме цей API було вирішено обрати для виконання завдання.

### 1.5 Розгляд альтернатив вебзастосунку

Розглянемо такі наявні альтернативні вебзастосунки, призначені для застосування фільтрів методом лінійної згортки:

- Image Convolution Playground;
- setosa.io;
- PlanetCalc;
- Image Kernel Convolution Tool.

Розпочнемо із Image Convolution Playground (див. рисунок 1.7).

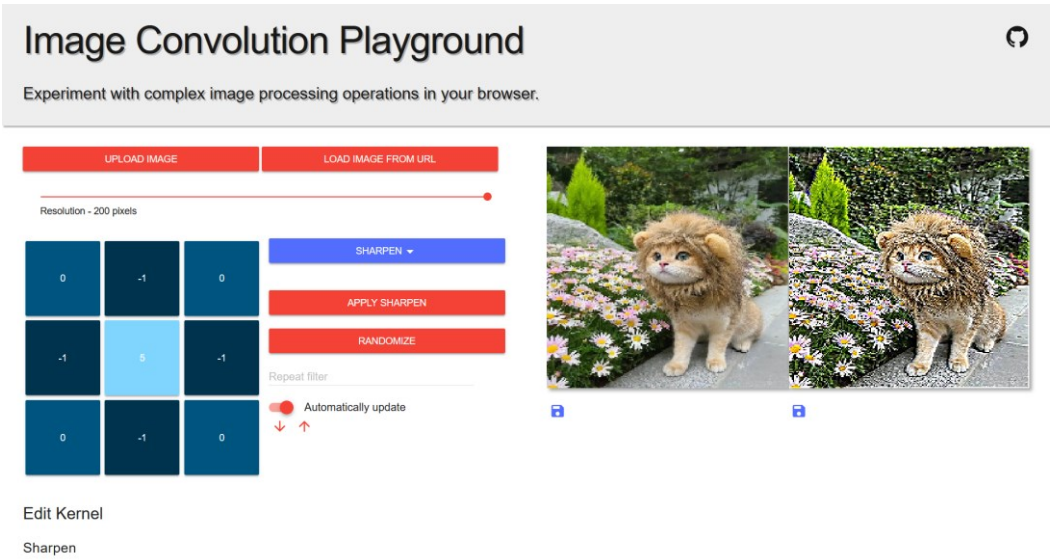


Рисунок 1.7 – Інтерфейс Image Convolution Playground

До переваг цього вебзастосунку можна врахувати можливість ручного вводу матриці ядра фільтра, але суттєвими недоліками є автоматична зміна розміру вихідного зображення та максимальна роздільна здатність результату, рівна 200x200. Вебзастосунок виконує обчислення засобами CPU та не використовує низькорівневі графічні API.

Приклад накладання фільтра різкості за допомогою Image Convolution Playground зображено на рисунку 1.8.

Рисунок 1.8 – Результат роботи Image Convolution Playground

На вищенаведеному рисунку можна бачити погіршення роздільної здатності зображення, порівняно із оригінальним (рис. 1.1).

Наступним розглянемо вебзастосунок на сайті setosa.io. Він дозволяє накладати такі фільтри:

– різкість;
– розмиття;
– тиснення;
– тотожність;
– оператор Собеля.

Результат обробки зображено на рисунку 1.9.


Рисунок 1.9 – Результат роботи setosa.io

До переваг вебзастосунку можна віднести наявність інформації про лінійну згортку. До недоліків – відсутність можливості завантажити результат обробки та перетворення кольорів у відтінки сірого. Це спрощує логіку обробки, оскільки потрібно враховувати тільки один канал, але спричиняє втрату інформації про кольори.

Тепер перейдемо до вебзастосунку PlanetCalc (див. рисунок 1.10).

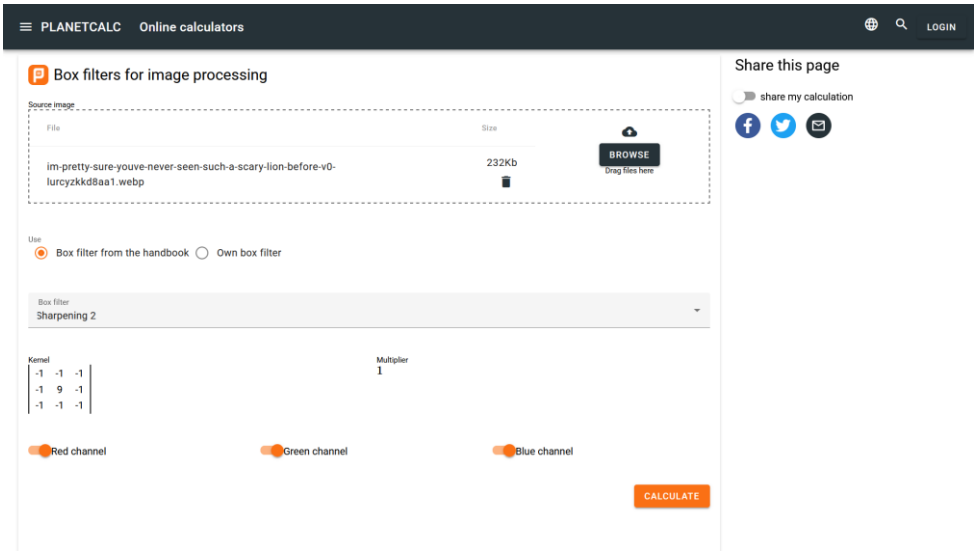


Рисунок 1.10 – Інтерфейс керування вебзастосунку PlanetCalc

До його переваг належать підтримка повної роздільної здатності зображення та фільтрація каналів зображення окремо.

До недоліків можна віднести відсутність автоматичного масштабування вихідного зображення та результату обробки, через що потрібно інтенсивно використовувати прокрутку.

Застосуємо фільтр різкості до синього та червоного каналів зображення (див. рисунок 1.11). При цьому зелений канал залишиться без змін.

Рисунок 1.11 – Результат фільтрації синього та червоного каналів у PlanetCalc

Далі розглянемо Image Kernel Convolution Tool (див. рисунок 1.12).

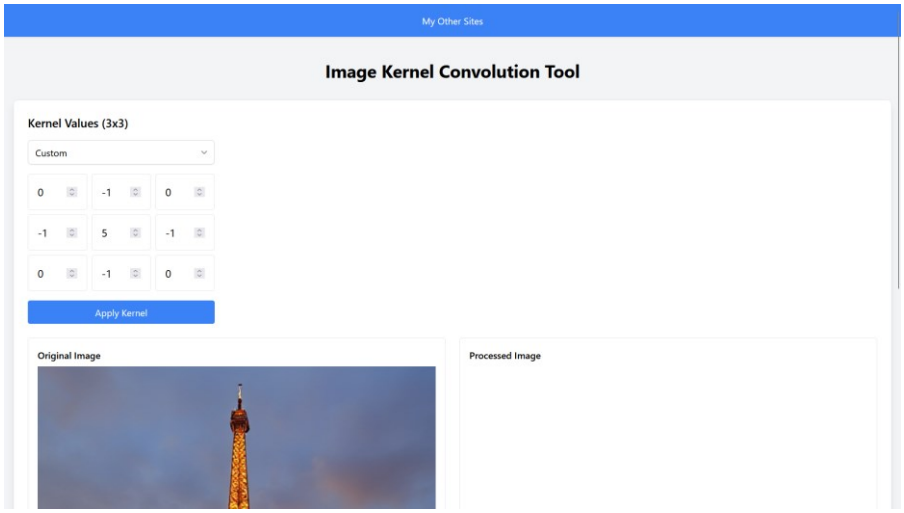


Рисунок 1.12 – Вигляд інтерфейсу Image Kernel Convolution Tool

Перевагою вебзастосунку є те, що він підтримує задання фільтра вручну, отже дозволяє використовувати довільну матрицю ядра фільтра третього порядку. Серед недоліків – неоптимальна розмітка сторінки, що змушує використовувати прокрутку.

Результат накладання фільтра наведення різкості зображено на рисунку 1.13.

Рисунок 1.13 – Результат роботи Image Kernel Convolution Tool

При виконанні обробки зображення було збережено його початкову роздільну здатність, що теж є перевагою. Проте усі розглянуті вебзастосунки не мають акценту на зручності інтерфейсу користувача, що також є важливим.

### 1.6 Постановка завдання

У ході виконання кваліфікаційної роботи ми створимо клієнтський вебзастосунок для фільтрації зображень методом лінійної згортки із використанням API WebGPU. Буде реалізовано такі фільтри:

- тотожність;
- просте розмиття;
- розмиття Гауса;
- тиснення;
- наведення контурів;
- наведення різкості;
- оператори Собеля у горизонтальному та вертикальному напрямках;
- оператори Превітта у горизонтальному та вертикальному напрямках.

При застосуванні фільтра має бути доступним попередній перегляд результату, а також потрібна можливість завантаження результату обробки. Окрім цього, додамо можливість повернутися до початкового стану зображення.

### 1.7 Висновки до першого розділу

У першому розділі кваліфікаційної роботи було розглянуто спочатку математичну модель кольорового зображення у RGBA як відображення із системи цілочислових координат у множину нормалізованих значень рівнів каналів, а далі зв’язок між нею й RGB. Після цього описано монохромні зображення та перетворення зображень із RGB на монохромні.

Далі стисло наведено визначення операції лінійної згортки зображень та можливості й переваги методу лінійної згортки.

Після цього описано коротко однорідні й неоднорідні обчислювальні середовища, а далі описано роль GPU в паралельних обчисленнях. Наступним кроком було порівняння WebGL та WebGPU і обґрунтування вибору WebGPU API для створення вебзастосунку.

Далі було розглянуто такі наявні альтернативи створюваного вебзастосунку:

- https://generic-github-user.github.io/Image-Convolution-Playground/src/;
- https://setosa.io/ev/image-kernels/;
- https://planetcalc.com/9313/;
- https://mystaticsite.com/kernelconvolution/.

На завершення було сформульовано завдання для розробки вебзастосунку на основі аналізу переваг і недоліків наявних альтернатив.

# РОЗДІЛ 2. ВИБІР ТЕХНОЛОГІЙ ТА ПРОЄКТУВАННЯ ІНТЕРФЕЙСУ ВЕБЗАСТОСУНКУ

Оскільки створюваний вебзастосунок не потребує обробки даних користувача сервером, його доцільно буде розробити повністю клієнтським – при переході користувача за адресою вебзастосунку він завантажуватиметься із сервера, а усі подальші дії оброблятимуться на боці клієнта. Тому розпочнемо із підбору фреймворку для створення інтерфейсу користувача.

## 2.1 Вибір фронтенд-фреймворку

Першим розглянемо React як найпопулярніший за даними 2024 StackOverflow Developer Survey [22].

### 2.1.1 React

React (див. рисунок 2.1) – бібліотека для створення користувацьких інтерфейсів вебзастосунків та мобільних додатків [23].

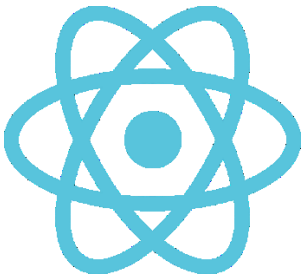

Рисунок 2.1 – Логотип React

React є не фреймворком, а саме бібліотекою. Він використовується разом з іншими бібліотеками для рендерингу в підтримуваних середовищах. Наприклад, React Native можна використовувати для створення мобільних додатків.

Для веброзробки React використовують разом із ReactDOM. Коли називаємо React фреймворком, маємо на увазі це поєднання.

Основна мета React – мінімізувати помилки при створенні інтерфейсів користувача. Це досягається за допомогою компонентів – самодостатніх, логічних фрагментів коду, які описують частину інтерфейсу. Ці компоненти можна об'єднувати для створення повноцінного інтерфейсу користувача, а React абстрагує значну частину опису логіки рендерингу, дозволяючи зосередитися на дизайні інтерфейсу [24].

До його переваг належать зокрема:

– простота використання;

– абстрагування за допомогою віртуального DOM (Document Object Model);

– найбільша екосистема серед фронтенд-фреймворків;

– модульність;

– розвинений інструментарій [25].

Серед недоліків можна відзначити:

– нестача якісної документації;

– розрізненість екосистеми;

– конфлікти бібліотек;

– не є повноцінним фреймворком [26].

Приклад класу компонента React наведено у лістингу 2.1.

Лістинг 2.1 – Приклад компонента React

```
class Foo extends Component {
  state = { age: 1 };

  render() {
    return (
      <div>
        Name: {this.props.name}, Age: {this.state.age}
      </div>
    );
  }
}
```

У цьому прикладі використано синтаксис JSX. Він є зручним для створення невеликих компонентів, тому на ідеях, використаних у React, було засновано також бібліотеки Preact, SolidJS та фреймворк Qwik [27].

Приклад ідентичного компонента для Preact наведено у лістингу 2.2.

Лістинг 2.2 – Приклад компонента Preact

```
class Foo extends Component {
  state = { age: 1 };

  render({ name }, { age }) {
    return (
      <div>
        Name: {name}, Age: {age}
      </div>
    );
  }
}
```

Можна бачити, що код відрізняється, але компоненти усе ж схожі.

Приклад вигляду інтерфейсу вебзастосунка

Тепер перейдемо до іншого популярного фреймворка – Vue.js.

### 2.1.2 Vue.js

Vue.js (див. рисунок 2.2) – це ще один із найпопулярніших фреймворків JavaScript для створення користувацьких інтерфейсів [22].

Рисунок 2.2 – Логотип Vue.js

Фреймворк базується на стандартних HTML, CSS та JavaScript і надає декларативну компонентну модель програмування, яка допомагає ефективно розробляти користувацькі інтерфейси [28].

Одними із ключових можливостей Vue.js є декларативний рендеринг (Vue.js має синтаксис шаблонів, який дозволяє декларативно описувати вивід HTML на основі стану JavaScript) та підтримка реактивності (зміни стану JavaScript відстежуються та за наявності DOM автоматично оновлюється).

До шляхів використання фреймворку належать:

– покращення статичного HTML без етапу побудови [28];

– вбудовування компонентів у довільну вебсторінку за допомогою Web Components API [29];

– SPA (Single-Page Application);

– SSR (Server-Side Rendering);

– SSG (Static Site Generation);

– настільні, мобільні та консольні додатки.

Приклад однофайлового компонента наведено у лістингу 2.3 [28].

Лістинг 2.3 – Приклад компонента Vue.js

```
<script setup>
import { ref } from 'vue'
const count = ref(0)
</script>

<template>
  <button @click="count++">Count is: {{ count }}</button>
</template>

<style scoped>
button {
  font-weight: bold;
}
</style>
```

До переваг Vue.js належать зокрема:

– простота;

– продуктивність;

– якість документації;
– кросплатформність;
– модульність
– абстрагування за допомогою віртуального DOM.

Серед недоліків фреймворку можна виділити:

– невелика спільнота;
– розрізненість підходів;
– обмежена екосистема [30].

І останнім розглянемо новий (порівняно із попередніми) фреймворк – Svelte.

### 2.1.3 Svelte

Svelte (див. рисунок 2.3) – це фреймворк побудови інтерфейсу користувача, який використовує компілятор, що дозволяє створювати лаконічні компоненти, які виконують мінімальну роботу в браузері, використовуючи мови HTML, CSS та JavaScript.

Рисунок 2.3 – Логотип Svelte

Як і інші фреймворки для інтерфейсу користувача, він дає змогу декларативно створювати додаток з компонентів, що поєднують розмітку, стилі та поведінку.

Ці компоненти компілюються в невеликі, ефективні модулі JavaScript, що усувають накладні витрати, традиційно пов'язані з фреймворками інтерфейсу користувача.

Можна створити весь додаток за допомогою Svelte (наприклад, використовуючи фреймворк додатків, такий як SvelteKit), або можливо додавати його поступово до існуючої кодової бази. Також можна створювати компоненти як окремі пакети, які працюють будь-де, за допомогою Web Components API [31].

Серед переваг фреймворку відзначимо такі:

– простота (найбільша серед розглянутих фронтенд-фреймворків);
– зручна модель реактивності;
– висока продуктивність;
– легкість інтеграції [32].

До недоліків належить ускладнення роботи із типами так званими рунами, які було додано у Svelte 5, які проте спрощують написання коду та роблять його лаконічнішим [33].

Приклад компонента наведено у лістингу 2.4.

Лістинг 2.4 – Приклад компонента Svelte

```
<script lang="ts">
  function greet() {
    alert('Welcome to Svelte!');
  }
</script>

<button onclick={greet}>click me</button>

<style>
  button {
    font-size: 2em;
  }
</style>
```

Можна бачити, що Svelte, подібно до Vue.js та React, підтримує створення однофайлових компонентів (у файлах *.svelte).

Для створення вебзастосунку використаємо Svelte, адже він є простим та швидким фреймворком, що стрімко набуває популярності.

## 2.2 Додаткові інструменти

Оскільки створюваний вебзастосунок є невеликим проєктом, доцільно буде скористатися додатковими інструментами, що допоможуть ефективніше виконати завдання.

### 2.2.1 SvelteKit

SvelteKit – це фреймворк для швидкої розробки надійних та продуктивних вебзастосунків за допомогою Svelte.

SvelteKit допомагає створювати вебзастосунки, дотримуючись сучасних принципів веброзробки та пропонуючи рішення для її поширених завдань. Серед його можливостей є зокрема:

- маршрутизація;
- оптимізація збірки;
- підтримка офлайн-режиму;
- попереднє завантаження вебсторінок перед навігацією;
- SSR;
- оптимізація зображень [34].

Таким чином SvelteKit надає багато можливостей для розробки стабільних та продуктивних вебзастосунків на основі фреймворку Svelte, тому використаємо його.

### 2.2.2 Tailwind CSS

Tailwind CSS – фреймворк мови CSS, який спрощує стилізацію, надаючи бібліотеку класів із попередньо заданими стилями CSS. Фреймворк створює стилі елементів HTML на основі їхньої належності до класів [35].

Таким чином спрощується процес розробки стилів оформлення елементів вебзастосунку, що дозволяє приділити більше уваги логіці його роботи.

## 2.3 Проєктування користувацького інтерфейсу вебзастосунку

Побудуємо діаграму варіантів використання вебзастосунку (див. рисунок 2.4).


Рисунок 2.4 – Діаграма варіантів використання

Інтерфейс вебзастосунку повинен містити такі елементи:

– кнопка вибору файлу зображення для обробки;

– випадаючий список фільтрів;

– кнопка для застосування фільтру;

– кнопка для завантаження результату обробки;

– область попереднього перегляду.

Доцільно буде додати такі елементи:

– кнопка скидання вигляду зображення до початкового;

– заголовок із логотипами та посиланнями на основні використані технології;

– статус виконання операції;

– матриця ядра обраного фільтра.

Вебзастосунок повинен пристосовуватися до розміру дисплею та зміни розміру вікна браузера. Тому використаємо підхід адаптивного дизайну (англ. responsive design). Його суть полягає у використанні одного макету, що міститиме гнучкі компоненти, які пристосовуватимуться до розмірів дисплею та відгукуватимуться на дії користувача, як-от зміна розміру вікна браузера [36].

Це дозволить використовувати вебзастосунок як при альбомній, так і при портретній (зокрема із мобільних пристроїв) орієнтаціях дисплею.

## 2.4 Проєктування структури вебзастосунку

Основним зв’язуючим компонентом вебзастосунку буде App.svelte, який міститиме логіку обробки зображень. При цьому App.svelte складатиметься із таких компонентів:

– Header.svelte, що міститиме заголовок вебзастосунку;

– Sidebar.svelte, у якому буде зосереджено елементи керування вебзастосунком та інформацію щодо статусу виконуваної операції;

– ImageCanvas.svelte, у якому розміщуватиметься область попереднього перегляду результату застосування фільтра до обраного зображення.

Header.svelte міститиме компоненти із логотипами та посиланнями на вебсайти основних використовуваних технологій:

– WebGPULogo.svelte;

– SvelteLogo.svelte.

Набір матриць ядер підтримуваних фільтрів виділимо у окрему бібліотеку filter-kernels.ts.

WGSL-шейдер, призначений для фільтрації зображень засобами GPU, доцільно буде виділити у окремий файл під назвою convolve.wgsl, адже він є самодостатнім та не залежить від інших компонентів.

На рисунку 2.5 зображено будову проєкту вебзастосунку на рівні структурних одиниць програмного коду.

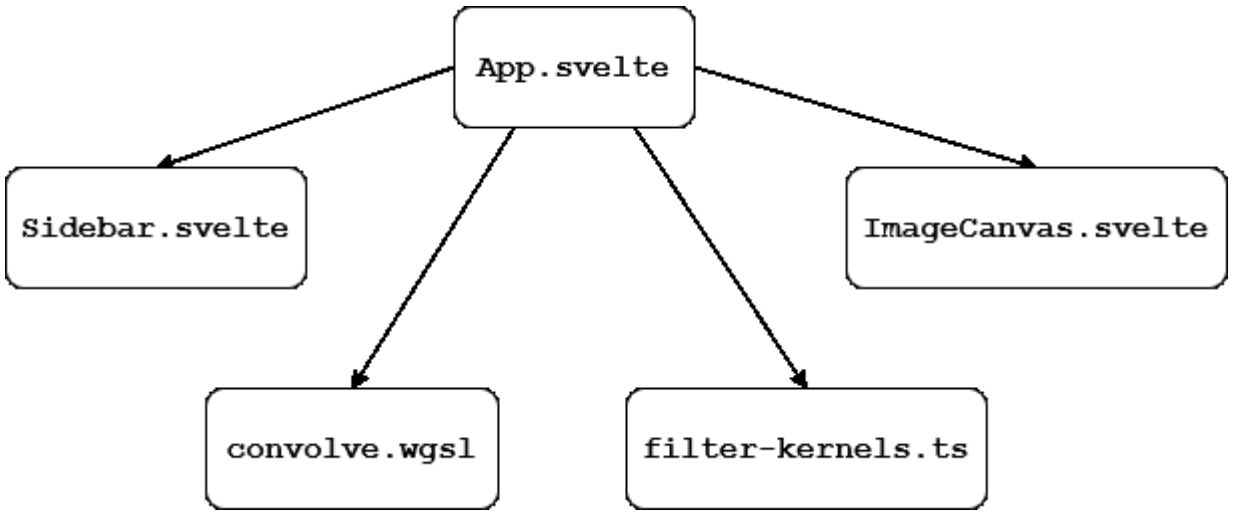


Рисунок 2.5 – Структура проєкту вебзастосунку

На вищенаведеному рисунку можна бачити, що компонент App.svelte є зв’язуючим компонентом вебзастосунку, та зосереджує усі інші.

## 2.5 Висновки до другого розділу

У другому розділі кваліфікаційної роботи було виконано підбір технологій для створення вебзастосунку згідно поставленого завдання.

Спочатку розглянуто фронтенд-фреймворки React, Vue.js та Svelte. У результаті їх порівняння було вирішено обрати Svelte як простий, зручний і продуктивний фреймворк, що набуває дедалі більшої популярності [22].

Після цього було обґрунтовано рішення застосувати SvelteKit як доповнення до Svelte, що дозволяє зосередитися на логіці роботи вебзастосунку та його можливостях. Задля ефективнішого створення стилів оформлення елементів було вирішено застосувати також і CSS-фреймворк Tailwind CSS.

Далі було спроєктовано вміст і розмітку користувацького інтерфейсу вебзастосунку, а після цього було виконано поділ вебзастосунку на структурні одиниці програмного коду.

# РОЗДІЛ 3. РОЗРОБКА ТА ДЕМОНСТРАЦІЯ ВЕБЗАСТОСУНКУ

## 3.1 Розробка вебзастосунку

Для того, щоби почати розробку вебзастосунку, потрібно створити новий проєкт із використанням шаблону SvelteKit за допомогою NPM (Node Package Manager):

```
npx sv create webgpu-svelte-conv
```

У результаті виконання вищенаведеної команди буде створено новий проєкт та надано вибір необхідних пакунків для додаткового встановлення із SvelteKit (див. рисунок 3.1).

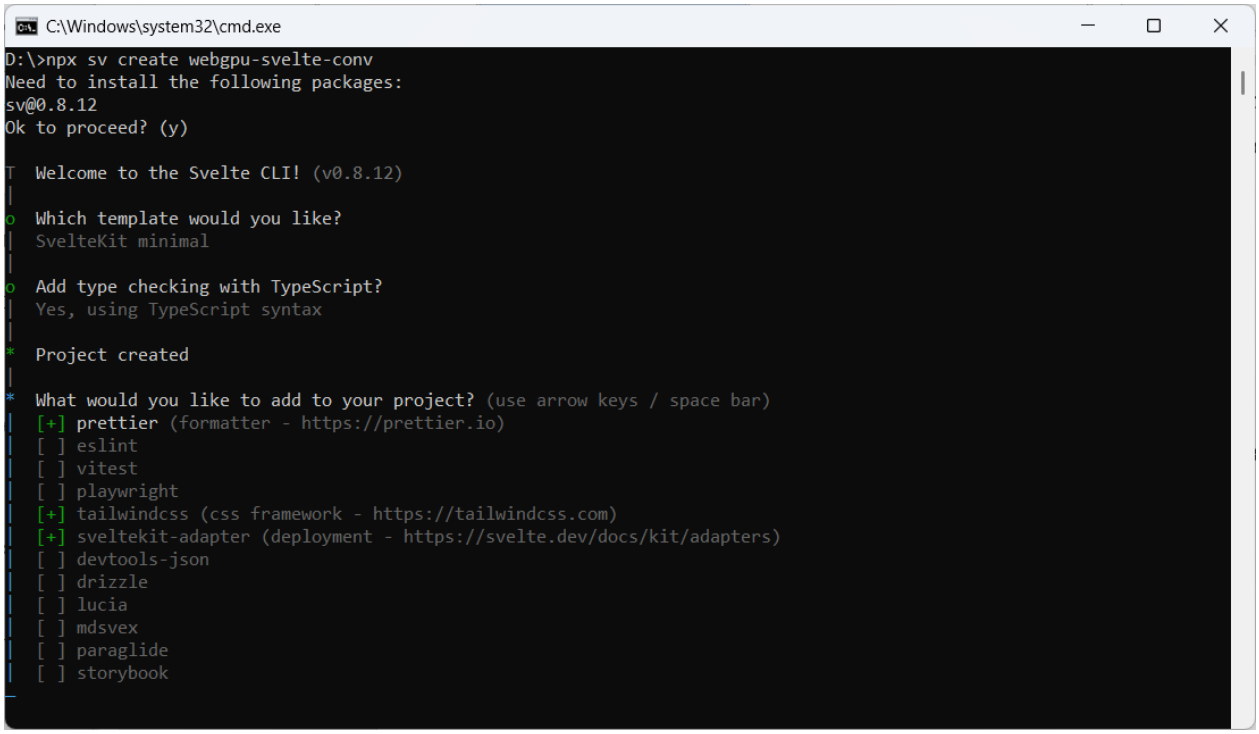


Рисунок 3.1 – Створення нового проєкту SvelteKit за допомогою NPM

На рисунку 3.2 зображено структуру основних директорій проєкту (див. рисунок 3.2).

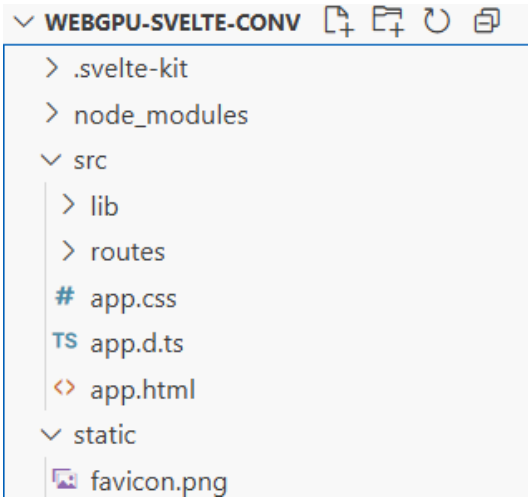


Рисунок 3.2 – Структура основних директорій створеного проєкту

У директорії .svelte-kit зберігаються налаштування та згенеровані при побудові проекту файли.

Директорія node_modules зберігає пакунки, необхідні для проекту, що були встановлені локально.

Основний код вебзастосунку розміщуватиметься у директорії src. При цьому директорія lib призначена для файлів власних JavaScript-бібліотек, що згодом імпортуватимуться за допомогою псевдоніму $lib, а директорія routes зберігатиме маршрути, тобто структуру посилань вебзастосунку.

Два перших відповідають за поведінку сторінки на боці клієнта, два останні – на боці сервера.

Директорія static призначена для вмісту, що повинен передаватись вебсервером у незміненому вигляді [37]. Прикладами є зокрема іконки та файли robots.txt.

Після створення ініціалізуємо проєкт, встановивши таким чином пакунки, вказані у package.json:

```
npm install
```

Результат виконання команди зображено на рисунку 3.3.

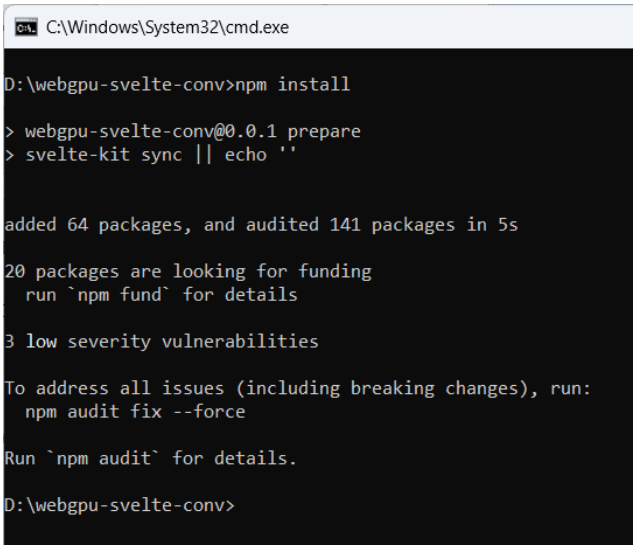


Рисунок 3.3 – Результат встановлення пакунків

Після створення та ініціалізації проєкту перейдемо до розробки компонентів вебзастосунку.

### 3.1.1 Розробка компонента заголовку

Заголовок вебзастосунку міститиме логотипи WebGPU (рис. 2.3) та Svelte (рис. 1.6) із посиланнями на специфікацію та на головну сторінку вебсайту фреймворку відповідно. Для того, щоби додати логотип WebGPU, створимо компонент WebGPULogo.svelte (див. лістинг 3.1).

Лістинг 3.1 – Код компонента WebGPULogo.svelte

```
<script>
  import webGPULogoUrl from '$lib/assets/webgpu.svg';
</script>

<div class="flex items-center">
  <a
    href="https://www.w3.org/TR/webgpu/"
    target="_blank"
    title="Powered by WebGPU"
    aria-label="WebGPU specification"
  >
    <img class="w-11 h-11" src={webGPULogoUrl} alt="WebGPU logo"
/>
  </a>
</div>
```

Після цього створимо аналогічний компонент для логотипу Svelte під назвою SvelteLogo.svelte (див. лістинг 3.2).

Лістинг 3.2 – Код компонента SvelteLogo.svelte

```
<script>
  import svelteLogoUrl from '$lib/assets/svelte.svg';
</script>

<div class="flex items-center">
  <a
    href="https://svelte.dev/"
    target="_blank"
    title="Built with Svelte"
    aria-label="Svelte website"
  >
    <img class="w-8 h-8" src={svelteLogoUrl} alt="Svelte logo" />
  </a>
</div>
```

Окрім логотипів основних використовуваних технологій, компонент Header.svelte має містити заголовок, а також дані про підтримку WebGPU. Код компонента наведено у лістингу 3.3.

Лістинг 3.3 – Код компонента Header.svelte

```
<script>
  import SvelteLogo from "$lib/components/SvelteLogo.svelte";
  import WebGPULogo from "$lib/components/WebGPULogo.svelte";

  let { webGPUSupported = false } = $props();
</script>

<header class="bg-white shadow-sm border-b">
  <div class="max-w-7xl mx-auto px-4 sm:px-6 lg:px-8">
    <div class="flex justify-between items-center h-16">
      <div class="flex items-center space-x-3">
        <WebGPULogo />
        <SvelteLogo />
      </div>
      <h1 class="text-xl font-semibold text-gray-900">WebGPU Image
Convolution</h1>
      <div class="flex items-center space-x-4">
        {#if webGPUSupported}
          <span class="text-sm text-green-600 bg-green-50 px-2 py-
1 rounded">
            Using GPU
          </span>
```

```
        {:else}
          <span class="text-sm text-red-600 bg-red-50 px-2 py-1
rounded">
            Using CPU
          </span>
        {/if}
      </div>
    </div>
  </div>
</header>
```

У вищенаведеному лістингу можна бачити, що компонент містить заголовок «WebGPU Image Convolution», а також дані про підтримку WebGPU: за наявності підтримки у правому куті виводиться напис «Using GPU», а за її відсутності – «Using CPU» на червоному.

Після цього перейдемо до створення бокової панелі із налаштуваннями фільтра.

### 3.1.2 Розробка компонента бічної панелі

У створюваному вебзастосунку бічна панель є одним із ключових елементів інтерфейсу користувача, адже повинна містити такі складові:

– кнопка для вибору файлу зображення;

– випадаючий список для вибору фільтра;

– кнопки для застосування обраного фільтра, для завантаження результату обробки та для скидання вигляду зображення до початкового;

– інформація про статус операції, виконуваної вебзастосунком;

– матриця ядра обраного фільтра.

Код компонента Sidebar.svelte наведено у лістингу А.1.

Саме бічна панель зосереджує елементи керування вебзастосунком, але окрім неї, має бути ще область попереднього перегляду, якій відведено основну частину інтерфейсу. Перейдемо до її створення.

### 3.1.3 Розробка компонента попереднього перегляду

Область попереднього перегляду призначена для показу результату застосування обраного фільтра до зображення.

Оскільки розмір зображення може бути більшим за роздільну здатність дисплею та не вміщуватись у відведену область інтерфейсу зокрема, то необхідно використовувати масштабування у таких випадках, але при цьому на розмірі результуючого зображення це не повинно відзначитись.

Окрім цього, у області попереднього перегляду варто вказувати дані про відсутність у ній зображення. Хоча зображення може співпадати із виведеним попередженням, ця інформація у більшості випадків усе ж може бути корисною.

Код компонента ImageCanvas.svelte наведено у лістингу А.2.

Розробивши програмний код елементів інтерфейсу, потрібно створити зв’язуючий компонент, який керуватиме логікою вебзастосунку.

### 3.1.4 Розробка зв’язуючого компонента

App.svelte є основним компонентом створюваного вебзастосунку, який керуватиме його логікою та інкапсулюватиме попередньо створені компоненти інтерфейсу користувача, що дозволить легко під’єднати одразу всі елементи інтерфейсу вебзастосунку (див. лістинг 3.4).

Лістинг 3.4 – Код +page.svelte головного маршруту вебзастосунку

```
<script>
  import App from './App.svelte';
</script>

<App />
```

Код компонента App.svelte наведено у лістингу Б.1.

Для виконання фільтрації зображень засобами GPU потрібен обчислювальний шейдер, виділимо його у файл convolve.wgsl (див. лістинг Б.2).

Задля зручності доповнення переліку фільтрів їх матриці було виділено у окрему бібліотеку filter-kernels.ts (див. лістинг Б.3).

## 3.2 Демонстрація вебзастосунку та результатів його роботи

Розглянемо спочатку вигляд інтерфейсу користувача створеного вебзастосунку, а потім перейдемо до тестування його роботи, спробувавши накласти на зображення підтримувані фільтри.

На рисунку 3.4 зображено початковий вигляд інтерфейсу створеного вебзастосунку.

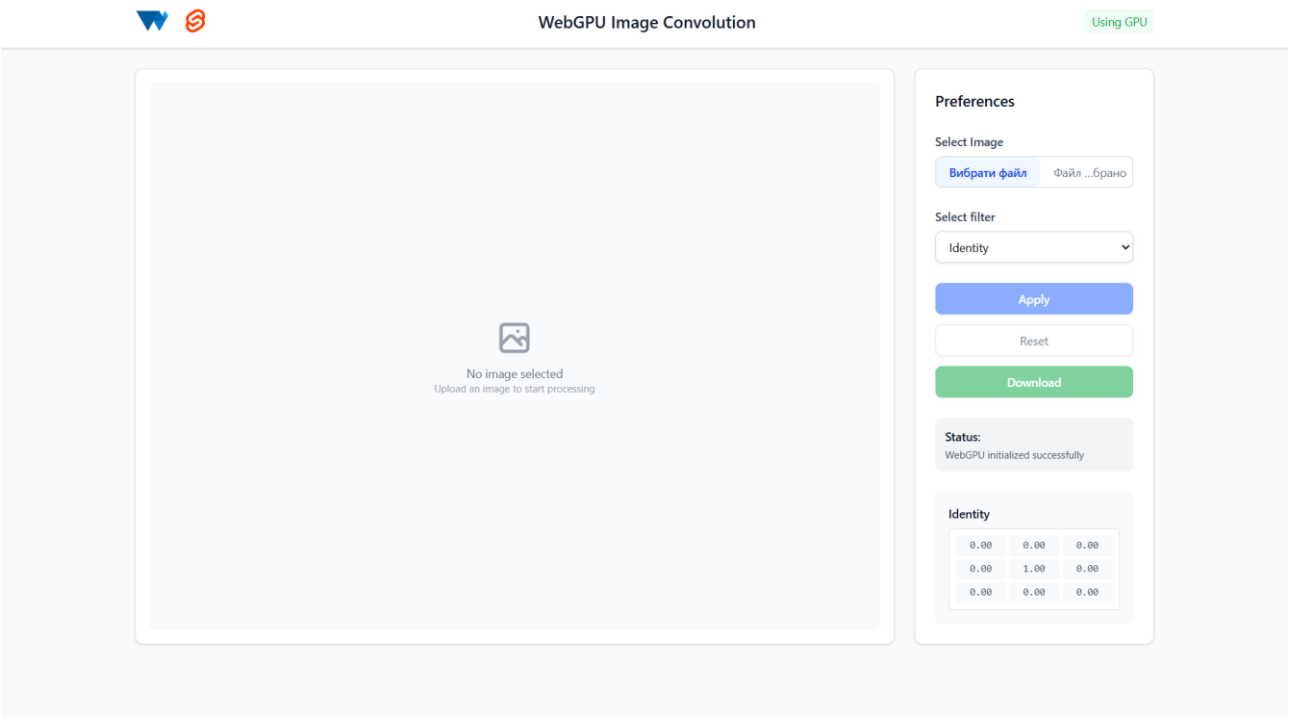


Рисунок 3.4 – Вигляд інтерфейсу вебзастосунку у альбомній орієнтації

На вищенаведеному рисунку зображено вигляд інтерфейсу користувача при альбомній орієнтації дисплею. Однак дисплей може також мати портретну орієнтацію і в цьому випадку для бічної панелі параметрів вебзастосунку та області попереднього перегляду не вистачить місця при розміщенні в рядок. Саме тому в цьому випадку потрібно використовувати розміщення у стовпчик (див. рисунок 3.5).

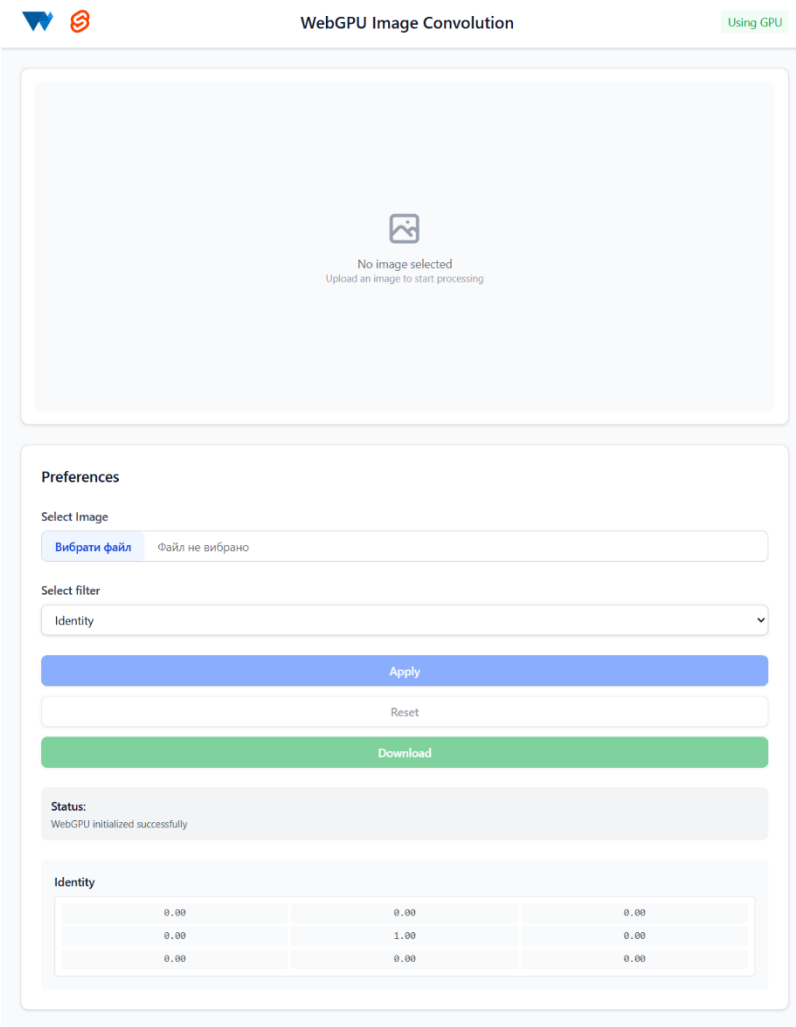


Рисунок 3.5 – Вигляд інтерфейсу вебзастосунку у портретній орієнтації

Можна бачити, що вебзастосунок відображає інформацію про статус підтримки WebGPU браузером. На вищенаведеному рисунку є напис «Using GPU», бо знімок екрану було зроблено в браузері Google Chrome Beta, який підтримує API WebGPU. Саме бета-версію браузера було використано тому, що стан реалізації API ближчий до актуальної версії специфікації. Поки API є експериментальним, до нього можуть бути внесені зміни, які порушують зворотну сумісність.

Проте станом на період виконання кваліфікаційної роботи Mozilla Firefox Stable узагалі не підтримує WebGPU [21], тому при використанні вебзастосунку обробка виконуватиметься засобами CPU, про що також буде повідомлено користувачу (див. рисунок 3.6).

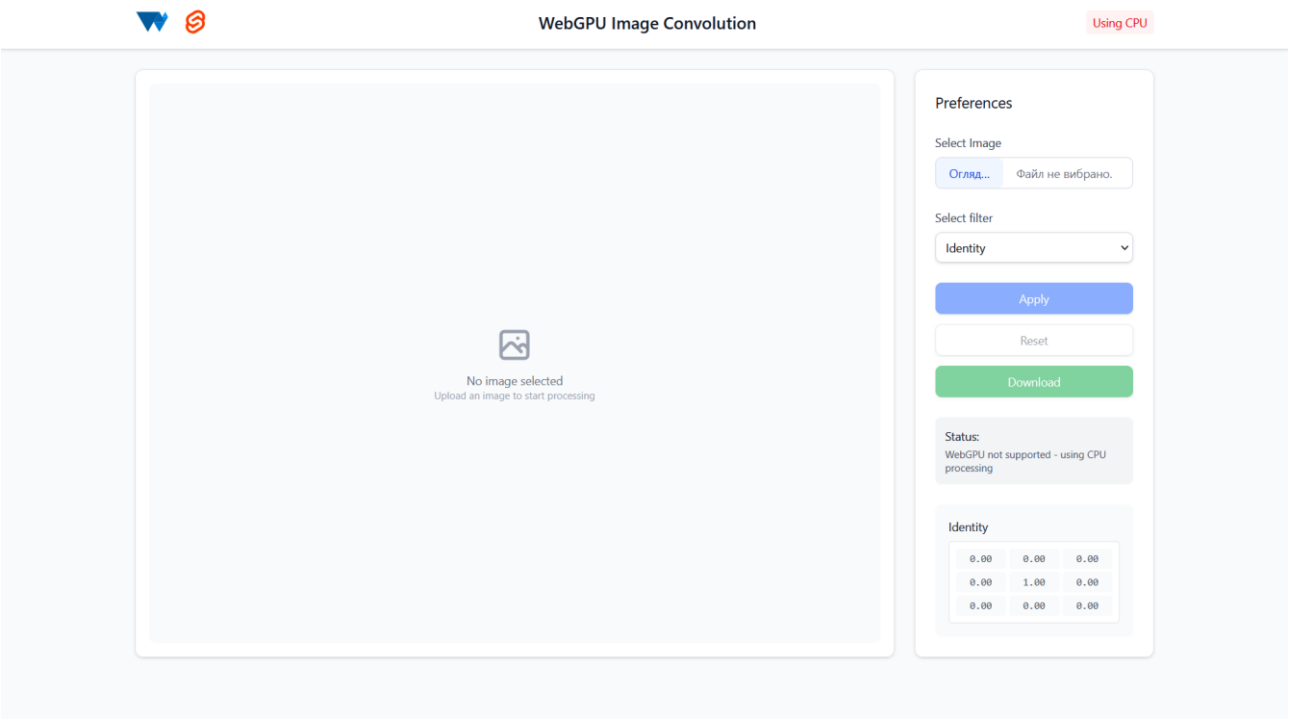


Рисунок 3.6 – Інтерфейс вебзастосунку за відсутності підтримки WebGPU

Перейдемо тепер до тестування вебзастосунку. Надалі $K$ – матриця ядра фільтра, що розглядається у межах пункту.

### 3.2.1 Фільтр простого розмиття

Вебзастосунок підтримує фільтр простого розмиття, матриця ядра якого виглядає так:

$$K = \begin{pmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{pmatrix}. \tag{3.1}$$

Вигляд інтерфейсу вебзастосунку при застосуванні фільтра простого розмиття зображено на рисунку 3.7.

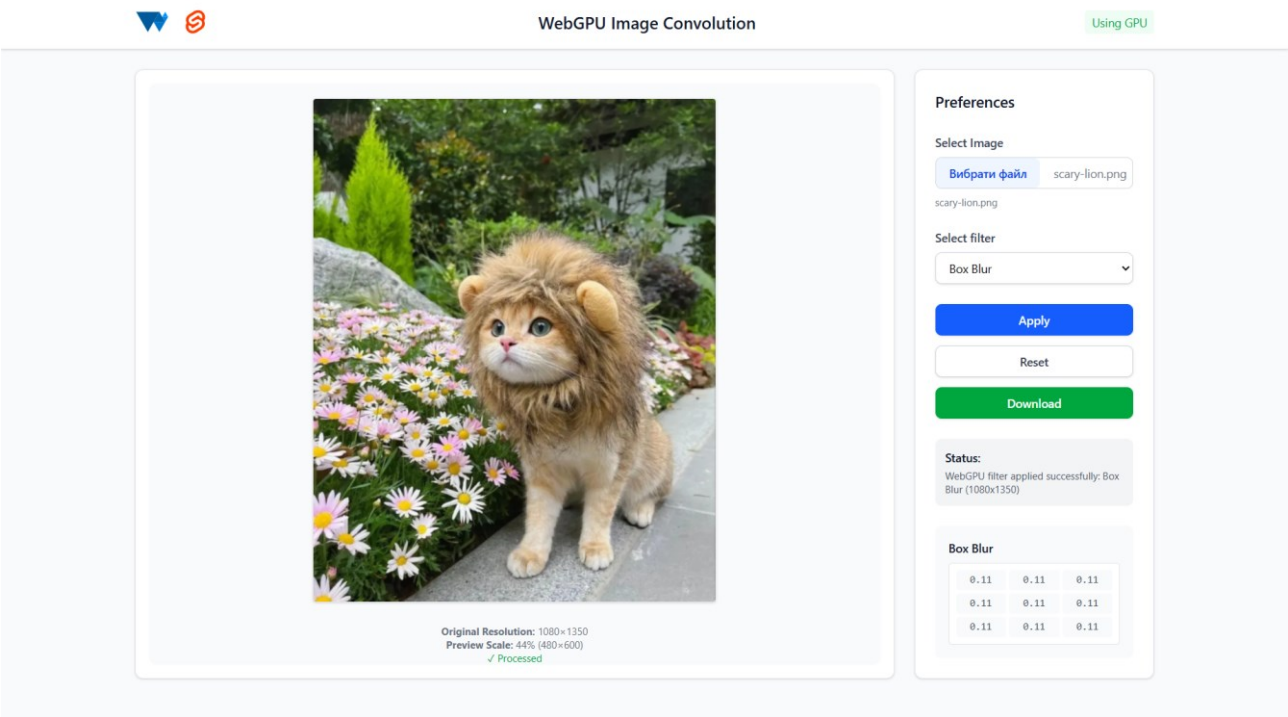


Рисунок 3.7 – Вигляд інтерфейсу при застосуванні фільтра простого розмиття

Оскільки вигляд інтерфейсу вебзастосунку при накладанні інших фільтрів буде аналогічним до зображеного на вищенаведеному рисунку, надалі демонструватимемо лише результат обробки.

### 3.2.2 Фільтр розмиття Гауса

Створений вебзастосунок дозволяє застосовувати ще один фільтр розмиття – розмиття Гауса, матриця ядра якого виглядає так:

$$K = \begin{pmatrix} 1/16 & 2/16 & 1/16 \\ 2/16 & 4/16 & 2/16 \\ 1/16 & 2/16 & 1/16 \end{pmatrix}. \quad (3.2)$$

Результат накладання фільтра розмиття Гауса зображено на рисунку 3.8.

Рисунок 3.8 – Результат застосування фільтра розмиття Гауса

Наступним використаємо фільтр тиснення.

### 3.2.3 Фільтр тиснення

Фільтр тиснення використовується для надання зображенню рельєфного вигляду [38]. Матриця ядра фільтра виглядає так:

$$K = \begin{pmatrix} -2 & -1 & 0 \\ -1 & 1 & 1 \\ 0 & 1 & 2 \end{pmatrix}. \quad (3.3)$$

Результат накладання фільтра тиснення зображено на рисунку 3.9.

Рисунок 3.9 – Результат застосування фільтра тиснення

Далі розглянемо фільтр наведення контурів.

### 3.2.4 Фільтр наведення контурів

Фільтр наведення контурів застосовується для відкидання усієї інформації про кольори за винятком необхідної для зображання меж об’єктів. Матриця ядра фільтра виглядає так:

$$K = \begin{pmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{pmatrix}. \tag{3.4}$$

Результат накладання фільтра наведення контурів зображено на рисунку 3.10.

Рисунок 3.10 – Результат застосування фільтра наведення контурів

Тепер перейдемо до операторів Превітта у горизонтальному та вертикальному напрямках.

### 3.2.5 Оператори Превітта

Оператори Превітта застосовуються для визначення горизонтальних або вертикальних меж об’єктів зображення [39]. Матриця ядра фільтра виглядає так:

$$K = \begin{pmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{pmatrix}. \quad (3.5)$$

Результат накладання фільтра зображено на рисунку 3.11.

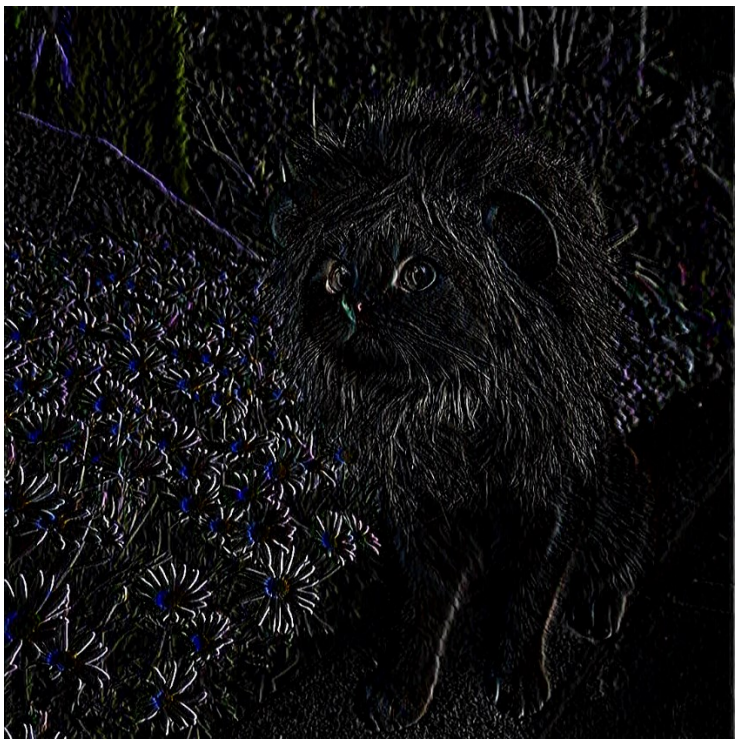

Рисунок 3.11 – Результат застосування горизонтального оператора Превітта

У випадку вертикального оператора Превітта матриця ядра фільтру є транспонованою до (3.5). Результат накладання фільтра зображено на рисунку 3.12.

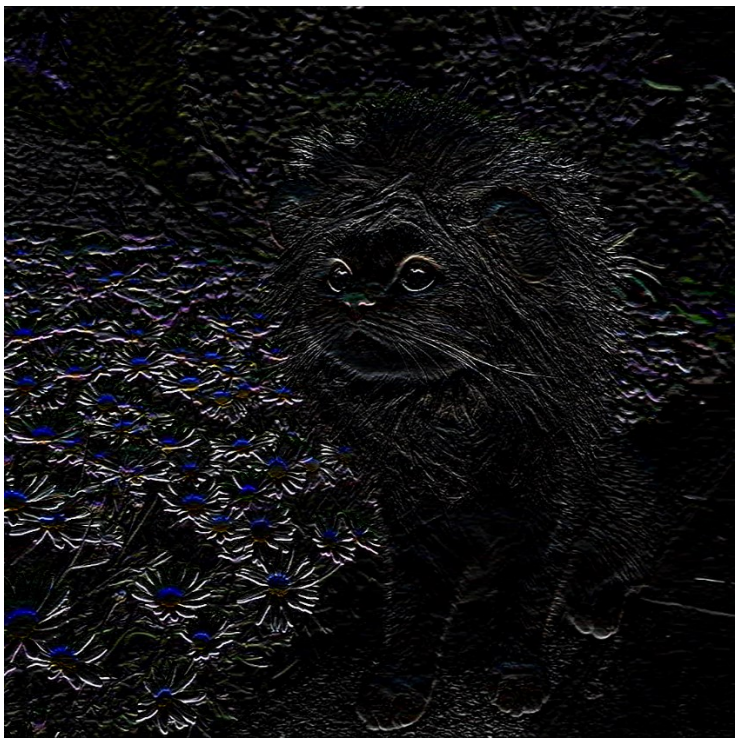

Рисунок 3.12 – Результат застосування вертикального оператора Превітта

Після цього розглянемо інший метод виявлення країв елементів зображення – оператори Собеля.

### 3.2.6 Оператори Собеля

Оператори Собеля, як і оператори Превітта, застосовуються для виявлення країв елементів зображення у горизонтальному та вертикальному напрямках [38]. Матриця ядра фільтра виглядає так:

$$K = \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix}. \tag{3.6}$$

Результат накладання фільтру зображено на рисунку 3.13.

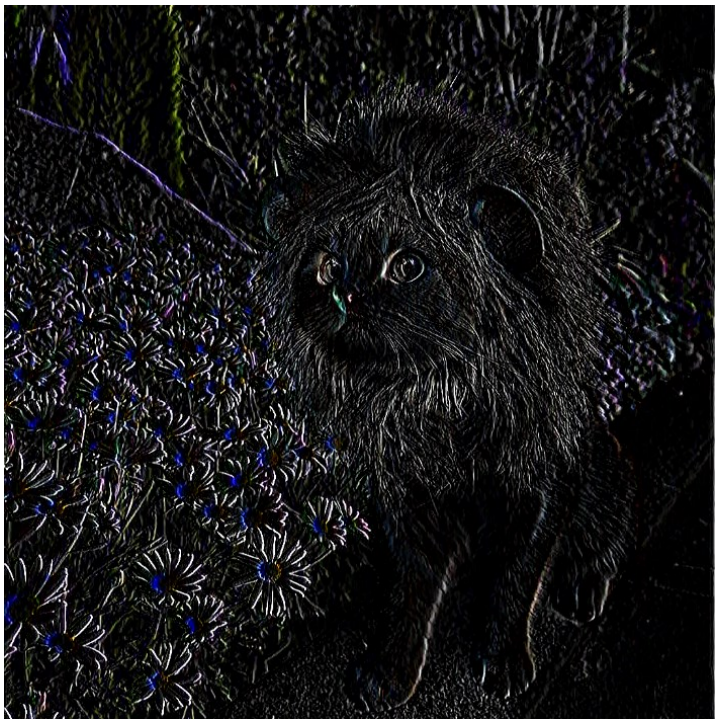

Рисунок 3.13 – Результат застосування горизонтального оператора Собеля

У випадку вертикального оператора Собеля матриця ядра фільтру є адитивно оберненою до транспонованої (3.5), тобто $-K^{\mathrm{T}}$. Результат накладання фільтра зображено на рисунку 3.14.

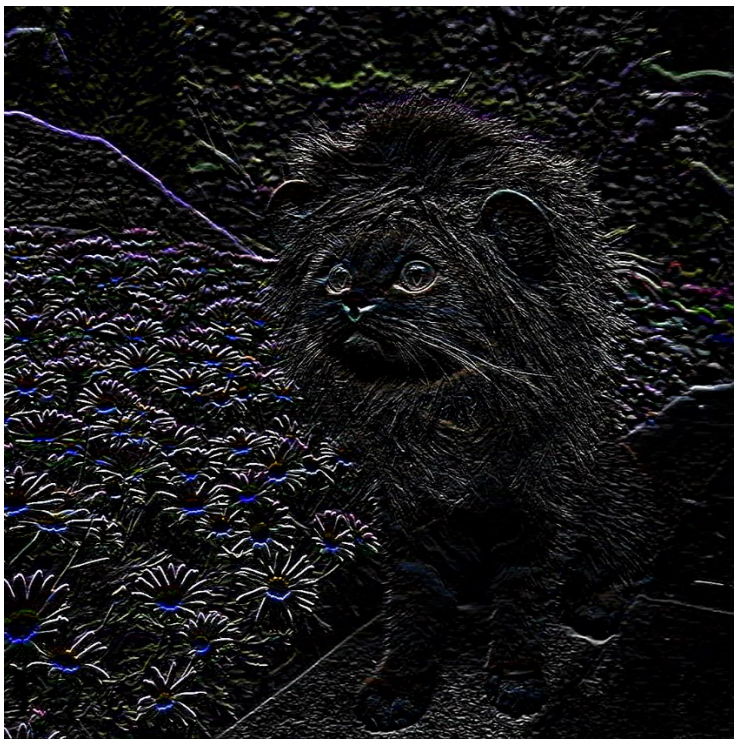

Рисунок 3.14 – Результат застосування вертикального оператора Собеля

Останнім розглянемо фільтр наведення різкості.

### 3.2.7 Фільтр наведення різкості

Фільтр наведення різкості виконує дію, протилежну до розмиття – робить більш помітною різницю між кольорами сусідніх пікселів зображення за її наявності. Матриця ядра фільтра виглядає так:

$$K = \begin{pmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{pmatrix}. \quad (3.7)$$

Результат накладання фільтра зображено на рисунку 3.15.

Рисунок 3.15 – Результат застосування фільтра наведення різкості

Усі фільтри накладаються вебзастосунком зі збереженням оригінального розміру зображення, що було однією із цілей роботи.

### 3.3 Висновки до третього розділу

У третьому розділі кваліфікаційної роботи було спочатку розглянуто основні характеристики створюваних компонентів вебзастосунку та уточнено вимоги до їхньої реалізації.

Після цього було продемонстровано вигляд та поведінку користувацького інтерфейсу вебзастосунку за умов як альбомної, так і портретної орієнтацій дисплею.

На завершення розділу було розглянуто результати застосування до зображення кожного із доданих фільтрів.

# РОЗДІЛ 4. БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

## 4.1 Психологічні чинники небезпеки

До психологічних чинників небезпеки належать:

- підвищена емоційність;
- втома;
- необережність;
- відсутність мотивації;
- недостатність досвіду [40].

Праця людини є функціональним процесом, у якому використовуються фізіологічні та психологічні якості працівника. В процес праці залучаються всі органи й системи організму людини – мозок, м'язи, судини, серце, легені та ін.

При цьому витрачається нервова та м'язова енергія. Крім того, в процесі праці активізуються усі психічні функції людини: сприймання, мислення, пам'ять, відчуття, уява, вольові якості, уважність, задоволення, зосередженість тощо.

У процесі праці людина сприймає і переробляє інформацію, в тому числі інформацію про наявність шкідливих і небезпечних чинників на робочому місці; приймає і реалізує рішення; осмислює доступні варіанти дій.

Психологічні можливості робітника не є постійними. Вони залежать від інформаційного навантаження, темпу роботи, напруження зорового та слухового аналізаторів, емоційного стану людини. Так, після конфліктних ситуацій чи виробничих невдач обсяг уваги різко знижується, порушується пам’ять [41, с. 99].

Імовірність наразитись на небезпеку підвищують періодичні функціональні зміни в нервовій системі або інших системах чи органах, що мають хворобливий характер. Такі зміни не означають непрацездатності, однак можуть чинити несприятливий вплив на людину з погляду на її безпеку (наприклад, головний біль, серцеві захворювання, та цукровий діабет).

Перебіг хвороби відбивається на поведінці людини як безпосередньо – у вигляді слабкості, недомагання, так і побічно – шляхом загального впливу на психіку (наприклад, пригніченість, депресія, роздратованість), підвищуючи тим самим імовірність наразитись на небезпеку.

Підвищення захищеності осіб, що страждають такими недугами, можна досягти перш за все шляхом постійних медичних оглядів та необхідного лікування. Важливо також не допускати таких осіб до робіт з підвищеною небезпекою.

Імовірність наразитись на небезпеку підвищують зокрема вади органів чуття, наприклад, часткова втрата зору, слуху. Дефекти органів чуття можуть мати різну вираженість, однак навіть мінімальні порушення впливають на працездатність людини.

Підвищують імовірність наразитись на небезпеку також і порушення зв’язку між сенсорними та руховими центрами вищих відділів нервової системи. Внаслідок таких порушень людина не здатна з необхідними швидкістю та точністю реагувати на зовнішні впливи.

Порушення узгодженості між сенсорними та моторними процесами відіграють значну роль у виникненні нещасних випадків. Вказані порушення можуть бути компенсовані в першу чергу завдяки правильному розподілу уваги [42].

Поряд з чинниками, що стійко підвищують індивідуальну імовірність наразитись на небезпеку, існують також чинники, які або проявляються лише в певні періоди трудового процесу, або впливають на поведінку людини протягом короткого часу (кількох годин чи навіть хвилин):

- недостатність досвіду;
- втома;
- необережність;
- емоційний вплив.

## 4.2 Показники ефективності та заходи щодо покращення умов охорони праці

Працездатність людини залежить від її вихідного функціонального стану та дії факторів навколишнього середовища взагалі і виробничого – зокрема. З цієї причини для її оцінки використовується система різних показників, які характеризують як кількісні і якісні результати роботи, так і функціональні стани працівника.

Для оцінки працездатності застосовуються три групи показників, що характеризують результати виробничої діяльності, фізіологічні зрушення і зміни у психічних функціях людини в процесі праці. Це виробничі, фізіологічні і психологічні показники.

До виробничих показників належать:

- продуктивність праці – виробіток продукції за одиницю часу;
- трудомісткість роботи – витрати часу на виробничу операцію;
- якість роботи (продукції) – наявність браку;
- втрати робочого часу і простої устаткування з вини працівника.

До фізіологічних показників належать:

- величина енергозатрат;
- частота пульсу, ударний і хвилинний об’єм крові;
- м’язова сила;
- м’язова витривалість;
- частота дихання, легенева вентиляція, коефіцієнт споживання кисню.

До психологічних показників відносяться:

- увага (концентрація, переключення, розподіл);
- мислення;
- пам’ять;
- сприймання;
- емоційно-вольове напруження [43].

Для аналізу рівня працездатності використовують в основному такі методи:

– аналіз продуктивності праці;

– визначення суб'єктивних показників;

– визначення фізіологічних показників.

Методи визначення суб’єктивних показників засновані на оцінюванні ступеня стомлення самою працюючою людиною за допомогою опитування й тестування.

Методи визначення фізіологічних показників дозволяють об'єктивно оцінити рівень працездатності. Сутність фізіологічних методів полягає у вивченні функціонального стану людини, що виконує роботу, у визначенні реакції різних систем організму на виконання даної роботи [41, с.104].

Підтримка працездатності на оптимальному рівні – основна мета раціонального режиму праці та відпочинку.

При встановленні оптимального режиму праці та відпочинку необхідно визначити параметри, що сприятимуть оптимальному використанню виробничих фондів і забезпечать найефективніший розвиток виробництва.

Режими праці та відпочинку побудовані для раціонального виробничого режиму, з тим щоб забезпечити нормальний перебіг технологічного процесу, виконання заданих обсягів виробництва [44].

До заходів щодо покращення умов охорони праці належать:

– приведення основних фондів до вимог нормативно-правових актів із охорони праці;

– усунення впливу на працівників небезпечних і шкідливих виробничих чинників або приведення їх рівня на робочих місцях вимогам нормативно-правових актів із охорони праці;

– проведення атестації робочих місць та аудиту з охорони праці, оформлення стендів, оснащення кабінетів, виставок, придбання необхідних нормативно-правових актів, наочних посібників, літератури, плакатів, відеофільмів, макетів, програмних продуктів тощо з питань охорони праці;

– проведення навчання і перевірки знань із охорони праці посадових осіб та інших працівників у процесі трудової діяльності, організація лекцій, семінарів та консультацій із зазначених питань;

– забезпечення працівників спецодягом, спецвзуттям та іншими засобами індивідуального захисту відповідно до норм, передбачених законодавством, або до норм, визначених колективним договором;

– забезпечення працівників, зайнятих на роботах із важкими та шкідливими умовами праці, лікувально-профілактичним харчуванням, молоком або рівноцінними харчовими продуктами, а також газованою солоною водою;

– проведення медичних оглядів працівників, зайнятих на важких роботах, роботах зі шкідливими чи небезпечними умовами праці або таких, де є потреба у професійному доборі, щорічного обов'язкового медичного огляду осіб віком до 21 року [45].

### 4.3 Висновки до четвертого розділу

Психологічні чинники суттєво впливають на безпеку праці, оскільки трудова діяльність вимагає залучення фізіологічних і психічних ресурсів людини. Зниження уваги, стомлення, емоційна нестабільність, хвороби, дефекти органів чуття та порушення взаємодії сенсорних і моторних центрів підвищують імовірність нещасних випадків.

Окрім постійних ризиків, існують і короткочасні, що також можуть призвести до небезпеки. Для підтримки безпечних умов праці необхідно застосовувати методи оцінювання працездатності – як суб'єктивні (опитування), так і фізіологічні (вивчення функціонального стану).

Основними заходами з охорони праці є приведення їх до нормативних вимог, усунення шкідливих чинників, регулярна атестація робочих місць, навчання персоналу, забезпечення засобами індивідуального захисту, проведення медичних оглядів та організація раціонального режиму праці й відпочинку.

# ВИСНОВКИ

Під час виконання цієї кваліфікаційної роботи було розроблено вебзастосунок, призначений для накладання на зображення фільтрів методом лінійної згортки із використанням нового низькорівневого графічного API, створеного в якості заміни WebGL – WebGPU.

Такий вибір дозволив керувати обчислювальними ресурсами та їхнім розподілом ефективно, а підтримка WebGPU обчислювальних шейдерів допомогла виконати завдання із використанням інструментів за призначенням.

У першому розділі кваліфікаційної роботи було сформульовано математичну модель кольорового зображення RGBA, після чого наведено визначення операції лінійної згортки зображень. Далі розглянуто роль GPU в паралельних обчисленнях та обґрунтовано вибір WebGPU для одержання доступу до ресурсів пристрою. Після цього розглянуто наявні альтернативні рішення та виконано постановку завдання на основі аналізу їхніх переваг і недоліків.

У другому розділі кваліфікаційної роботи було виконано підбір технологій для виконання поставленого завдання. Оскільки вебзастосунок є клієнтським, адже не потребує відправки даних на сервер після завантаження, було зосереджено увагу на виборі фронтенд-фреймворку, а також було обрано допоміжні інструменти, що інтегруються із ним. Після цього було виконано проєктування користувацького інтерфейсу вебзастосунку та його поділ на структурні одиниці програмного коду.

У третьому розділі кваліфікаційної роботи було описано процес розробки вебзастосунку та уточнено вимоги до реалізації його компонентів. Після цього було продемонстровано створений вебзастосунок та протестовано коректність його роботи.

Створений вебзастосунок дозволяє підвищити продуктивність лінійної фільтрації зображень та дає змогу оптимально використовувати обчислювальні ресурси клієнтського пристрою у середовищі веб.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Berkovich A. Understanding Image Sampling and Quantization in Digital Image Processing. *Akridata*. URL: https://akridata.ai/blog/image-sampling-quantization-digital-image-processing (date of access: 21.03.2025).

2. Concept of Quantization. *Tutorials Point*. URL: https://www.tutorialspoint.com/dip/concept_of_quantization.htm (date of access: 21.03.2025).

3. 3.10. Color Spaces and Encoding. *W3C*. URL: https://www.w3.org/TR/webgpu/#color-spaces (date of access: 22.03.2025).

4. RGB. *MDN Web Docs*. URL: https://developer.mozilla.org/en-US/docs/Glossary/RGB (date of access: 22.03.2025).

5. Recommendation ITU-R BT.709-6. Parameter values for the HDTV standards for production and international programme exchange. Official edition. 2015. 19 p. URL: https://www.itu.int/dms_pubrec/itu-r/rec/bt/R-REC-BT.709-6-201506-I!!PDF-E.pdf (date of access: 22.03.2025).

6. Umbaugh S. E. Digital Image Processing and Analysis: Computer Vision and Image Analysis. Taylor & Francis Group, 2023. 440 p.

7. A. Zaporozhets, Y. Kuts, B. Mlynko, M. Fryz, and L. Scherbak, “EEG Signal Classification Using Linear Process Model-Based Feature Extraction and Supervised Learning,” in Advanced System Development Technologies II. Studies in Systems, Decision and Control, M. Bezuglyi, N. Bouraou, V. Mykytenko, G. Tymchyk, and A. Zaporozhets, Eds., Cham: Springer Nature Switzerland, 2025, pp. 235–257. doi: 10.1007/978-3-031-82035-9_7.

8. Бабак В.П., Куц Ю.В., Мислович М.В., Фриз М.Є., Щербак Л.М. Об’єктно-орієнтована ідентифікація стохастичних шумових сигналів. Київ: Наукова думка, 2024. 240 с. https://doi.org/10.15407/978-966-00-1883-9.

9. V. Babak, A. Zaporozhets, Y. Kuts, M. Fryz, L. Scherbak. Noise signals: Modelling and Analyses. Cham: Springer Nature Switzerland, 2025. 222 p. DOI: https://doi.org/10.1007/978-3-031-71093-3

10. Бабак В. П., Марченко М. Є., Фриз. Б. Г. Теорія ймовірностей, випадкові процеси та математична статистика. К.: Техніка, 2004. 288 с.

11. Types of Convolution Kernels. *Geeks for Geeks*. URL: https://www.geeksforgeeks.org/deep-learning/types-of-convolution-kernels/ (date of access: 24.03.2025).

12. Flinders M., Smalley I. What is parallel computing?. *IBM*. URL: https://www.ibm.com/think/topics/parallel-computing (date of access: 24.03.2025).

13. Heterogeneous vs. Homogeneous Computing Environments. *Intel Corporation*. URL: https://www.intel.com/content/www/us/en/docs/sycl/introduction/latest/01-homogeneous-vs-heterogeneous.html (date of access: 25.03.2025).

14. Metal. *Apple Developer Documentation*. URL: https://developer.apple.com/documentation/Metal/ (date of access: 27.03.2025).

15. WebGL 2.0 Compute. *Khronos Registry*. URL: https://registry.khronos.org/webgl/specs/latest/2.0-compute/ (date of access: 28.03.2025).

16. Introduction. *W3C*. URL: https://www.w3.org/TR/WebGPU/#intro (date of access: 28.03.2025).

17. Beaufort F. From WebGL to WebGPU. *Chrome for Developers*. URL: https://developer.chrome.com/blog/from-webgl-to-webgpu (date of access: 28.03.2025).

18. GPUComputePipeline. *W3C*. URL: https://www.w3.org/TR/WebGPU/#gpucomputepipeline (date of access: 28.03.2025).

19. Galvan A. A Review of Shader Languages. *Alain.xyz*. URL: https://alain.xyz/blog/a-review-of-shader-languages (date of access: 28.03.2025).

20. Browser compatibility. *MDN Web Docs*. URL: https://developer.mozilla.org/en-US/docs/Web/API/WebGPU_API#browser_compatibility (date of access: 28.03.2025).

21. Implementation Status. *GitHub*. URL: https://github.com/gpuweb/gpuweb/wiki/Implementation-Status (date of access: 28.03.2025).

22. Web frameworks and technologies. *Stack Overflow Insights*. URL: https://survey.stackoverflow.co/2024/technology#2-web-frameworks-and-technologies (date of access: 04.04.2025).

23. React. *React*. URL: https://react.dev/ (date of access: 04.04.2025).

24. Getting started with React. *MDN Web Docs*. URL: https://developer.mozilla.org/en-US/docs/Learn_web_development/Core/Frameworks_libraries/React_getting_started (date of access: 05.04.2025).

25. Why Use React? Top Benefits for Web Development. *Netguru*. URL: https://www.netguru.com/blog/why-use-react (date of access: 19.04.2025).

26. Advantages and Disadvantages of ReactJS. *Tutorials Point*. URL: https://www.tutorialspoint.com/reactjs/reactjs_advantages_and_disadvantages.htm (date of access: 23.04.2025).

27. The Case for Signals: Why Everyone's Talking About Solid.js, Qwik, and Preact. *Medium*. URL: https://medium.com/@asierr/the-case-for-signals-why-everyones-talking-about-solid-js-qwik-and-preact-497218345a50 (date of access: 28.04.2025).

28. Vue.js. *Vue.js - The Progressive JavaScript Framework*. URL: https://vuejs.org/guide/introduction.html (date of access: 28.04.2025).

29. Web Components. *MDN Web Docs*. URL: https://developer.mozilla.org/en-US/docs/Web/API/Web_components (date of access: 29.04.2025).

30. The Pros and Cons of Vue.js. *AltexSoft*. URL: https://www.altexsoft.com/blog/pros-and-cons-of-vue-js/ (date of access: 30.04.2025).

31. Welcome to Svelte • Svelte Tutorial. *Svelte*. URL: https://svelte.dev/tutorial/svelte/welcome-to-svelte (date of access: 01.05.2025).

32. Corradini F. Svelte 5 2025 Review: Runes and Other Exciting New Features. *Scalable Path*. URL: https://www.scalablepath.com/javascript/svelte-5-review (date of access: 15.05.2025).

33. The Svelte Team. Introducing runes. *Svelte*. URL: https://svelte.dev/blog/runes (date of access: 02.05.2025).

34. Introduction. *Svelte*. URL: https://svelte.dev/docs/kit/introduction (date of access: 08.05.2025).

35. Installing with Vite - Installation. *Tailwind Labs Inc*. URL: https://tailwindcss.com/docs/installation/using-vite (date of access: 08.05.2025).

36. Responsive web design. *MDN Web Docs*. URL: https://developer.mozilla.org/en-US/docs/Learn_web_development/Core/CSS_layout/Responsive_Design (date of access: 08.05.2025).

37. Project structure. *Svelte*. URL: https://svelte.dev/docs/kit/project-structure (date of access: 08.05.2025).

38. Emboss. *Affinity Online Help*. URL: https://affinity.help/photo2/en-US.lproj/index.html?page=pages/Filters/filter_emboss.html&title=Emboss (date of access: 12.05.2025).

39. Timothy M. Edge Detection in Image Processing: An Introduction. *Roboflow Blog*. URL: https://blog.roboflow.com/edge-detection/ (date of access: 15.05.2025).

40. Психофізіологічні фактори небезпеки. *Pidru4niki*. URL: https://pidru4niki.com/70725/bzhd/psihofiziologichni_faktori_nebezpeki (дата звернення: 17.05.2025).

41. «Безпека життєдіяльності» / Т. Є. Стиценко та ін. Харків : ХНУРЕ, 2018. 336 с. URL: https://os.nure.ua/wp-content/uploads/2019/05/posibnik-bgd_2018_p.1.pdf (дата звернення: 24.06.2025).

42. Березюк О. В., Лемешев М. С. 2.3 Психологічні чинники небезпеки. *Безпека життєдіяльності*. URL: https://web.posibnyky.vntu.edu.ua/fmbt/berezyuk_bezpeka_zhittyediyalnosti/23.htm (дата звернення: 19.05.2025).

43. 6.3. Показники і методика оцінки працездатності людини. *Studentam.net.ua*. URL: https://studentam.net.ua/content/view/6446/86/ (дата звернення: 19.05.2025).

44. Методи підвищення працездатності. Реферат. *Освіта.ua*. URL: https://osvita.ua/vnz/reports/econom_pidpr/18931/ (дата звернення: 20.05.2025).

45. Розробляємо комплексні заходи з охорони праці - Головне управління Пенсійного фонду України в Луганській області. *ГУ Пенсійного фонду України в Луганській області*. URL: https://www.pfu.gov.ua/lg/357125-rozroblyayemo-kompleksni-zahody-z-ohorony-pratsi/ (дата звернення: 21.05.2025).

# ДОДАТКИ

**Додаток А**

## Програмний код компонентів бічної панелі та області попереднього перегляду

Лістинг А.1 – Код компонента Sidebar.svelte

```
<script>
  let {
    selectedFile = null,
    selectedFilter = $bindable('sharpening'),
    isProcessing = false,
    debugInfo = 'Ready to process images',
    filterKernels = {},
    onFileSelect = () => {},
    onApplyFilter = () => {},
    onResetCanvas = () => {},
    onDownloadImage = () => {}
  } = $props();
</script>

<aside class="bg-white rounded-lg shadow-sm border p-6">
  <h2 class="text-lg font-medium text-gray-900 mb-
6">Preferences</h2>

  <form class="space-y-6" onsubmit={(e) => e.preventDefault()}>
    <!-- File Input -->
    <div>
      <label for="file-upload" class="block text-sm font-medium
text-gray-700 mb-2">
        Select Image
      </label>
      <input
        id="file-upload"
        type="file"
        accept="image/*"
        onchange={onFileSelect}
        class="block w-full text-sm text-gray-500 file:mr-4
file:py-2 file:px-4 file:rounded-md file:border-0 file:text-sm
file:font-medium file:bg-blue-50 file:text-blue-700 hover:file:bg-
blue-100 border border-gray-300 rounded-md"
      />
      {#if selectedFile}
        <p class="mt-2 text-xs text-gray-500">
          {selectedFile.name}
        </p>
      {/if}
    </div>

    <!-- Filter Selection -->
    <div>
      <label for="filter-select" class="block text-sm font-medium
text-gray-700 mb-2">
        Select filter
```

```
      </label>
      <select
        id="filter-select"
        bind:value={selectedFilter}
        class="block w-full px-3 py-2 border border-gray-300
rounded-md shadow-sm focus:outline-none focus:ring-blue-500
focus:border-blue-500 text-sm"
      >
        {#each Object.entries(filterKernels) as [key, filter]}
          <option value={key}>{filter.name}</option>
        {/each}
      </select>
    </div>

    <!-- Action Buttons -->
    <div class="space-y-3">
      <button
        type="button"
        onclick={onApplyFilter}
        disabled={!selectedFile || isProcessing}
        class="w-full flex justify-center py-2 px-4 border border-
transparent rounded-md shadow-sm text-sm font-medium text-white
bg-blue-600 hover:bg-blue-700 focus:outline-none focus:ring-2
focus:ring-offset-2 focus:ring-blue-500 disabled:opacity-50
disabled:cursor-not-allowed"
      >
        {#if isProcessing}
          Processing...
        {:else}
          Apply
        {/if}
      </button>

      <button
        type="button"
        onclick={onResetCanvas}
        disabled={!selectedFile || isProcessing}
        class="w-full flex justify-center py-2 px-4 border border-
gray-300 rounded-md shadow-sm text-sm font-medium text-gray-700
bg-white hover:bg-gray-50 focus:outline-none focus:ring-2
focus:ring-offset-2 focus:ring-blue-500 disabled:opacity-50
disabled:cursor-not-allowed"
      >
        Reset
      </button>

      <button
        type="button"
        onclick={onDownloadImage}
        disabled={!selectedFile || isProcessing}
        class="w-full flex justify-center py-2 px-4 border border-
transparent rounded-md shadow-sm text-sm font-medium text-white
bg-green-600 hover:bg-green-700 focus:outline-none focus:ring-2
```

```
focus:ring-offset-2 focus:ring-green-500 disabled:opacity-50
disabled:cursor-not-allowed"
      >
        Download
      </button>
    </div>

    <!-- Status -->
    <div class="p-3 bg-gray-100 rounded-md">
      <h3 class="text-sm font-medium text-gray-900 mb-
1">Status:</h3>
      <p class="text-xs text-gray-600">{debugInfo}</p>
    </div>

    <!-- Filter Preview -->
    <div class="p-4 bg-gray-50 rounded-md">
      <h3 class="text-sm font-medium text-gray-900 mb-2">
        {filterKernels[selectedFilter]?.name || 'Unknown Filter'}
      </h3>
      <div class="text-xs text-gray-600">
        <div class="font-mono bg-white p-2 rounded border grid
grid-cols-3 gap-1 text-center">
          {#each (filterKernels[selectedFilter]?.kernel || []) as
value, i}
            <span class="p-1 bg-gray-50 rounded">
              {value.toFixed(2)}
            </span>
          {/each}
        </div>
      </div>
    </div>
  </form>
</aside>
```

Лістинг А.2 – Код компонента ImageCanvas.svelte

```
<script>
  let {
    canvasElement = $bindable(null),
    selectedFile = null,
    isProcessing = false,
    originalDimensions = { width: 0, height: 0 },
    previewScale = 1,
    processedImageData = null
  } = $props();
</script>

<main class="lg:col-span-3 bg-white rounded-lg shadow-sm border p-
4">
  <div class="h-full min-h-[400px] lg:min-h-[600px] flex flex-col
items-center justify-center bg-gray-50 rounded-lg relative">
    {#if isProcessing}
```

```
      <div class="absolute inset-0 bg-black bg-opacity-50 flex
items-center justify-center z-10 rounded-lg">
        <div class="text-white text-center">
          <div class="animate-spin rounded-full h-8 w-8 border-b-2
border-white mx-auto mb-2"></div>
          <p>Processing image...</p>
        </div>
      </div>
    {/if}

    {#if selectedFile}
      <div class="flex-1 flex items-center justify-center">
        <canvas
          bind:this={canvasElement}
          class="max-w-full max-h-full border border-gray-200
rounded shadow-sm bg-white"
        ></canvas>
      </div>

      {#if originalDimensions.width > 0}
        <div class="text-xs text-gray-500 mt-2 text-center">
          <strong>Original Resolution:</strong>
{originalDimensions.width}×{originalDimensions.height}
          {#if previewScale < 1}
            <br><strong>Preview Scale:</strong>
{Math.round(previewScale * 100)}%
({Math.round(originalDimensions.width *
previewScale)}×{Math.round(originalDimensions.height *
previewScale)})
          {/if}
          {#if processedImageData}
            <br><span class="text-green-600">✓ Processed</span>
          {/if}
        </div>
      {/if}
    {:else}
      <div class="text-center text-gray-500">
        <svg class="mx-auto h-12 w-12 text-gray-400" fill="none"
viewBox="0 0 24 24" stroke="currentColor">
          <path stroke-linecap="round" stroke-linejoin="round"
stroke-width="2" d="M4 16l4.586-4.586a2 2 0 012.828 0L16 16m-2-
2l1.586-1.586a2 2 0 012.828 0L20 14m-6-6h.01M6 20h12a2 2 0 002-
2V6a2 2 0 00-2-2H6a2 2 0 00-2 2v12a2 2 0 002 2z" />
        </svg>
        <p class="mt-2 text-sm">No image selected</p>
        <p class="text-xs text-gray-400">Upload an image to start
processing</p>
      </div>
    {/if}
  </div>
</main>
```

**Додаток Б**

# Програмний код зв’язуючого компонента вебзастосунку, шейдера згортки та набору матриць ядер фільтрів

Лістинг Б.1 – Код компонента App.svelte

```
<script>
  import Header from '$lib/components/Header.svelte';
  import ImageCanvas from '$lib/components/ImageCanvas.svelte';
  import Sidebar from '$lib/components/Sidebar.svelte';
  import { filterKernels } from '$lib/filter-kernels';
  import shaderCode from '$lib/shaders/convolve.wgsl?raw';

  let canvasElement = $state(null);
  let selectedFile = $state(null);
  let ctx = $state(null);
  let device = $state(null);
  let selectedFilter = $state('identity');
  let isProcessing = $state(false);
  let webGPUSupported = $state(false);
  let originalImageData = $state(null);
  let processedImageData = $state(null);
  let debugInfo = $state('Ready to process images');
  let previewScale = $state(1);
  let originalDimensions = $state({ width: 0, height: 0 });

  async function initWebGPU() {
    try {
      if (!navigator.gpu) {
        debugInfo = 'WebGPU not supported - using CPU processing';
        return;
      }

      const adapter = await navigator.gpu.requestAdapter({
        powerPreference: 'high-performance',
      });
      if (!adapter) {
        debugInfo = 'No WebGPU adapter found - using CPU
processing';
        return;
      }

      device = await adapter.requestDevice({
        requiredLimits: {
          maxBufferSize: 2147483648,
          maxStorageBufferBindingSize: 2147483644
        },
      });
      webGPUSupported = true;
      debugInfo = 'WebGPU initialized successfully';
    } catch (error) {
      console.error('WebGPU initialization failed:', error);
```

```
      debugInfo = `WebGPU failed: ${error.message} - using CPU
processing`;
    }
  }

  $effect(() => {
    initWebGPU();
  });

  function handleFileSelect(event) {
    const file = event.target.files[0];
    if (file && file.type.startsWith('image/')) {
      selectedFile = file;
      loadImageToCanvas(file);
    }
  }

  function loadImageToCanvas(file) {
    const reader = new FileReader();
    reader.onload = (e) => {
      const img = new Image();
      img.onload = () => {
        if (canvasElement) {
          // Store original dimensions
          originalDimensions = {
            width: img.width,
            height: img.height
          };

          // Store original image data using OffscreenCanvas
          const offscreenCanvas = new OffscreenCanvas(img.width,
img.height);
          const offscreenCtx = offscreenCanvas.getContext('2d');
          offscreenCtx.drawImage(img, 0, 0, img.width,
img.height);
          originalImageData = offscreenCtx.getImageData(0, 0,
img.width, img.height);
          processedImageData = null; // Reset processed data

          // Calculate preview scale
          const containerWidth = 800; // Max preview width
          const containerHeight = 600; // Max preview height

          const scaleX = containerWidth / img.width;
          const scaleY = containerHeight / img.height;
          previewScale = Math.min(scaleX, scaleY, 1); // Only
scale down for preview

          // Update preview display
          updatePreviewCanvas();

          debugInfo = `Image loaded at full resolution:
${img.width}x${img.height} pixels`;
        }
```

```
      };
      img.src = e.target.result;
    };
    reader.readAsDataURL(file);
  }

  function updatePreviewCanvas() {
    if (!originalImageData || !canvasElement) return;

    // Use processed data if available, otherwise use original
    const imageDataToShow = processedImageData ||
originalImageData;

    // Create temporary canvas with full-resolution data
    const tempCanvas = new
OffscreenCanvas(originalDimensions.width,
originalDimensions.height);
    const tempCtx = tempCanvas.getContext('2d');
    tempCtx.putImageData(imageDataToShow, 0, 0);

    // Set preview canvas size based on scale
    const previewWidth = Math.round(originalDimensions.width *
previewScale);
    const previewHeight = Math.round(originalDimensions.height *
previewScale);

    canvasElement.width = previewWidth;
    canvasElement.height = previewHeight;

    ctx = canvasElement.getContext('2d');
    ctx.clearRect(0, 0, previewWidth, previewHeight);

    // Draw scaled version to preview canvas
    ctx.drawImage(tempCanvas, 0, 0, previewWidth, previewHeight);
  }

  async function applyFilter() {
    if (!originalImageData) {
      debugInfo = 'No image loaded';
      return;
    }

    isProcessing = true;

    try {
      if (webGPUSupported && device) {
        debugInfo = 'Attempting WebGPU processing...';
        try {
          await applyWebGPUFilter();
        } catch (webgpuError) {
          console.warn('WebGPU failed, falling back to CPU:',
webgpuError);
          debugInfo = `WebGPU failed: ${webgpuError.message}.
Using CPU fallback...`;
```

```
          applyCPUFilter();
        }
      } else {
        debugInfo = 'Using CPU processing...';
        applyCPUFilter();
      }

      // Update preview with processed image
      updatePreviewCanvas();

    } catch (error) {
      console.error('All processing methods failed:', error);
      debugInfo = `Processing failed: ${error.message}`;
    } finally {
      isProcessing = false;
    }
  }

  function applyCPUFilter() {
    if (!originalImageData) return;

    const kernel = filterKernels[selectedFilter].kernel;
    const width = originalImageData.width;
    const height = originalImageData.height;
    const inputData = originalImageData.data;
    const outputData = new Uint8ClampedArray(inputData.length);

    // Apply convolution to full-resolution image
    for (let y = 0; y < height; y++) {
      for (let x = 0; x < width; x++) {
        const pixelIndex = (y * width + x) * 4;

        let r = 0, g = 0, b = 0;

        // Apply 3x3 convolution kernel
        for (let ky = -1; ky <= 1; ky++) {
          for (let kx = -1; kx <= 1; kx++) {
            const sampleX = Math.min(Math.max(x + kx, 0), width -
1);
            const sampleY = Math.min(Math.max(y + ky, 0), height -
1);
            const sampleIndex = (sampleY * width + sampleX) * 4;
            const kernelIndex = (ky + 1) * 3 + (kx + 1);
            const weight = kernel[kernelIndex];

            r += inputData[sampleIndex] * weight;
            g += inputData[sampleIndex + 1] * weight;
            b += inputData[sampleIndex + 2] * weight;
          }
        }

        // Clamp values and set output
        outputData[pixelIndex] = Math.min(Math.max(Math.round(r),
0), 255);
```

```
        outputData[pixelIndex + 1] =
Math.min(Math.max(Math.round(g), 0), 255);
        outputData[pixelIndex + 2] =
Math.min(Math.max(Math.round(b), 0), 255);
        outputData[pixelIndex + 3] = inputData[pixelIndex + 3]; //
Preserve alpha
      }
    }

    // Store the processed image data
    processedImageData = new ImageData(outputData, width, height);
    debugInfo = `CPU filter applied:
${filterKernels[selectedFilter].name} (${width}x${height})`;
  }

  async function applyWebGPUFilter() {
    if (!device || !originalImageData) {
      throw new Error('WebGPU not ready or no image data');
    }

    const width = originalImageData.width;
    const height = originalImageData.height;
    debugInfo = `Starting WebGPU processing: ${width}x${height}`;

    try {
      // Normalize image data
      const pixelCount = width * height * 4;
      const inputFloats = new Float32Array(pixelCount);
      for (let i = 0; i < pixelCount; i++) {
        inputFloats[i] = originalImageData.data[i] / 255.0;
      }
      debugInfo = `Converted ${pixelCount} pixels to float data`;

      // Create buffers with proper alignment
      const bufferSize = inputFloats.byteLength;
      const alignedSize = Math.ceil(bufferSize / 4) * 4;

      const inputBuffer = device.createBuffer({
        size: alignedSize,
        usage: GPUBufferUsage.STORAGE | GPUBufferUsage.COPY_DST,
        mappedAtCreation: false
      });

      const outputBuffer = device.createBuffer({
        size: alignedSize,
        usage: GPUBufferUsage.STORAGE | GPUBufferUsage.COPY_SRC,
        mappedAtCreation: false
      });

      const paramsBuffer = device.createBuffer({
        size: 8, // 2 * u32
        usage: GPUBufferUsage.UNIFORM | GPUBufferUsage.COPY_DST,
        mappedAtCreation: false
      });
```

```
      const kernelBuffer = device.createBuffer({
        size: 36, // 9 * f32
        usage: GPUBufferUsage.STORAGE | GPUBufferUsage.COPY_DST,
        mappedAtCreation: false
      });

      const resultBuffer = device.createBuffer({
        size: alignedSize,
        usage: GPUBufferUsage.COPY_DST | GPUBufferUsage.MAP_READ,
        mappedAtCreation: false
      });

      // Load into GPU
      device.queue.writeBuffer(inputBuffer, 0, inputFloats);
      device.queue.writeBuffer(paramsBuffer, 0, new
Uint32Array([width, height]));
      device.queue.writeBuffer(kernelBuffer, 0, new
Float32Array(filterKernels[selectedFilter].kernel));

      debugInfo = 'Data loaded into GPU buffers';

      // Create shader module and pipeline
      const shaderModule = device.createShaderModule({
        code: shaderCode,
        label: 'Convolution Compute Shader'
      });

      const computePipeline = device.createComputePipeline({
        layout: 'auto',
        compute: {
          module: shaderModule,
          entryPoint: 'main'
        },
        label: 'Convolution Pipeline'
      });

      // Create bind group
      const bindGroup = device.createBindGroup({
        layout: computePipeline.getBindGroupLayout(0),
        entries: [
          { binding: 0, resource: { buffer: inputBuffer } },
          { binding: 1, resource: { buffer: outputBuffer } },
          { binding: 2, resource: { buffer: paramsBuffer } },
          { binding: 3, resource: { buffer: kernelBuffer } }
        ],
        label: 'Convolution Bind Group'
      });

      // Execute compute shader
      const commandEncoder = device.createCommandEncoder({ label:
'Convolution Commands' });
      const computePass = commandEncoder.beginComputePass({ label:
'Convolution Pass' });
```

```
      computePass.setPipeline(computePipeline);
      computePass.setBindGroup(0, bindGroup);

      const workgroupsX = Math.ceil(width / 32);
      const workgroupsY = Math.ceil(height / 8);
      computePass.dispatchWorkgroups(workgroupsX, workgroupsY);
      computePass.end();

      // Copy result to readable buffer
      commandEncoder.copyBufferToBuffer(
        outputBuffer, 0,
        resultBuffer, 0,
        alignedSize
      );

      // Submit commands and wait
      const commandBuffer = commandEncoder.finish();
      device.queue.submit([commandBuffer]);
      await device.queue.onSubmittedWorkDone();

      debugInfo = `GPU processing complete, dispatched
${workgroupsX}x${workgroupsY} workgroups`;

      // Read result back
      await resultBuffer.mapAsync(GPUMapMode.READ);
      const mappedRange = resultBuffer.getMappedRange();
      const resultFloats = new Float32Array(mappedRange, 0,
pixelCount);

      debugInfo = `Reading back ${resultFloats.length} float
values`;

      // Convert back to uint8 with validation
      const resultBytes = new Uint8ClampedArray(pixelCount);
      for (let i = 0; i < pixelCount; i++) {
        const floatValue = resultFloats[i];
        if (isNaN(floatValue) || !isFinite(floatValue)) {
          resultBytes[i] = originalImageData.data[i];
        } else {
          resultBytes[i] = Math.round(Math.max(0, Math.min(1,
floatValue)) * 255);
        }
      }

      // Verify data
      const nonZeroPixels = resultBytes.filter(v => v > 0).length;
      debugInfo = `Converted to bytes, ${nonZeroPixels} non-zero
values`;

      if (nonZeroPixels === 0) {
        throw new Error('WebGPU processing resulted in blank
image');
      }
```

```
      // Store the processed image data
      processedImageData = new ImageData(resultBytes, width,
height);

      resultBuffer.unmap();
      debugInfo = `WebGPU filter applied successfully:
${filterKernels[selectedFilter].name} (${width}x${height})`;

      // GPU resources cleanup
      inputBuffer.destroy();
      outputBuffer.destroy();
      paramsBuffer.destroy();
      kernelBuffer.destroy();
      resultBuffer.destroy();

    } catch (error) {
      debugInfo = `WebGPU error: ${error.message}`;
      throw error;
    }
  }

  function resetCanvas() {
    if (!originalImageData) return;

    // Reset to original image data
    processedImageData = null;
    updatePreviewCanvas();
    debugInfo = 'Image reset to original';
  }

  function downloadImage() {
    if (!originalImageData) return;

    // Use processed data if available, otherwise use original
    const imageDataToDownload = processedImageData ||
originalImageData;

    // Create a full-resolution canvas for download
    const downloadCanvas = document.createElement('canvas');
    downloadCanvas.width = imageDataToDownload.width;
    downloadCanvas.height = imageDataToDownload.height;

    const downloadCtx = downloadCanvas.getContext('2d');
    downloadCtx.putImageData(imageDataToDownload, 0, 0);

    const link = document.createElement('a');
    link.download = `processed-image-${selectedFilter}.png`;
    link.href = downloadCanvas.toDataURL();
    link.click();

    debugInfo = `Downloaded full resolution:
${imageDataToDownload.width}x${imageDataToDownload.height}`;
  }
```

```
</script>

<div class="min-h-screen bg-gray-50 flex flex-col">
  <Header {webGPUSupported} />

  <!-- Main Content -->
  <div class="flex-1 max-w-7xl mx-auto w-full px-4 sm:px-6 lg:px-8
py-6">
    <div class="grid grid-cols-1 lg:grid-cols-4 gap-6 h-full">
      <ImageCanvas
        bind:canvasElement
        {selectedFile}
        {isProcessing}
        {originalDimensions}
        {previewScale}
        {processedImageData}
      />

      <Sidebar
        {selectedFile}
        bind:selectedFilter
        {isProcessing}
        {debugInfo}
        {filterKernels}
        onFileSelect={handleFileSelect}
        onApplyFilter={applyFilter}
        onResetCanvas={resetCanvas}
        onDownloadImage={downloadImage}
      />
    </div>
  </div>
</div>
```

Додаток Б.2 – Код шейдера convolve.wgsl

```
//Convolution shader
@group(0) @binding(0) var<storage, read> inputBuffer: array<f32>;
@group(0) @binding(1) var<storage, read_write> outputBuffer:
array<f32>;
@group(0) @binding(2) var<uniform> params: vec2<u32>;

struct Kernel {
  values: array<f32, 9>
}
@group(0) @binding(3) var<storage, read> kernel: Kernel;

@compute @workgroup_size(32, 8)
fn main(@builtin(global_invocation_id) global_id: vec3<u32>) {
  let x = global_id.x;
  let y = global_id.y;
  let width = params.x;
  let height = params.y;
```

```
  if (x >= width || y >= height) {
    return;
  }

  var r = 0.0;
  var g = 0.0;
  var b = 0.0;

  for (var ky = 0u; ky < 3u; ky++) {
    for (var kx = 0u; kx < 3u; kx++) {
      let offsetX = i32(kx) - 1;
      let offsetY = i32(ky) - 1;

      let sampleX = clamp(i32(x) + offsetX, 0, i32(width) - 1);
      let sampleY = clamp(i32(y) + offsetY, 0, i32(height) - 1);

      let sampleIndex = u32(sampleY) * width + u32(sampleX);
      let kernelIndex = ky * 3u + kx;
      let weight = kernel.values[kernelIndex];

      r += inputBuffer[sampleIndex * 4u + 0u] * weight;
      g += inputBuffer[sampleIndex * 4u + 1u] * weight;
      b += inputBuffer[sampleIndex * 4u + 2u] * weight;
    }
  }

  let pixelIndex = y * width + x;
  outputBuffer[pixelIndex * 4u + 0u] = clamp(r, 0.0, 1.0);
  outputBuffer[pixelIndex * 4u + 1u] = clamp(g, 0.0, 1.0);
  outputBuffer[pixelIndex * 4u + 2u] = clamp(b, 0.0, 1.0);
  outputBuffer[pixelIndex * 4u + 3u] = inputBuffer[pixelIndex * 4u
+ 3u]; // Preserve alpha
}
```

Лістинг Б.3 – Код бібліотеки filter-kernels.ts

```
// Filter kernels
export const filterKernels = {
  boxBlur: {
    name: 'Box Blur',
    kernel: [
      1/9, 1/9, 1/9,
      1/9, 1/9, 1/9,
      1/9, 1/9, 1/9
    ]
  },
  gaussianBlur: {
    name: 'Gaussian Blur',
    kernel: [
      1/16, 2/16, 1/16,
      2/16, 4/16, 2/16,
      1/16, 2/16, 1/16
    ]
```

```
},
emboss: {
  name: 'Emboss',
  kernel: [
    -2, -1, 0,
    -1, 1, 1,
    0, 1, 2
  ]
},
identity: {
  name: 'Identity',
  kernel: [
    0, 0, 0,
    0, 1, 0,
    0, 0, 0
  ]
},
outline: {
  name: 'Outline',
  kernel: [
    -1, -1, -1,
    -1, 8, -1,
    -1, -1, -1
  ]
},
prewittX: {
  name: 'Prewitt X',
  kernel: [
    1, 0, -1,
    1, 0, -1,
    1, 0, -1
  ]
},
prewittY: {
  name: 'Prewitt Y',
  kernel: [
    1, 1, 1,
    0, 0, 0,
    -1, -1, -1
  ]
},
sharpen: {
  name: 'Sharpen',
  kernel: [
    0, -1, 0,
    -1, 5, -1,
    0, -1, 0
  ]
},
sobelX: {
  name: 'Sobel X',
  kernel: [
    1, 0, -1,
    2, 0, -2,
```

```
      1, 0, -1
    ]
  },
  sobelY: {
    name: 'Sobel Y',
    kernel: [
      -1, -2, -1,
      0, 0, 0,
      1, 2, 1
    ]
  }
};
```