

УДК 004.92

DOI [https://doi.org/10.15589/znп2026.1\(503\).2.12](https://doi.org/10.15589/znп2026.1(503).2.12)**MODELS OF COLOR AND TONE TECHNOLOGIES FOR FILTERING GRAPHIC IMAGES USING WebGL AND THE FABRICJS****МОДЕЛІ КОЛЬОРОВИХ ТА ТОНАЛЬНИХ ТЕХНОЛОГІЙ ФІЛЬТРАЦІЇ ГРАФІЧНИХ ЗОБРАЖЕНЬ З ВИКОРИСТАННЯМ WebGL AND THE FABRICJS****Roman I. Kordiuk¹**

roman.i.kordiuk@lpnu.ua

ORCID: 0009-0008-6082-901X

Volodymyr K. Ovsyak^{1,2}

vovsyak@ukr.net

ORCID: 0000-0001-9295-284X

Р. І. Кордюк¹,

аспірант

В. К. Овсяк^{1,2},

докт. техн. наук, професор

^{1,2}*Lviv Polytechnic National University, Lviv*²*Kielce University of Technology, Kielce, Poland*^{1, 2}*Національний університет «Львівська політехніка», м. Львів*²*Кельцький технологічний університет, м. Кельце, Польща*

Abstract. A model of color and tonal filtering technologies for raster images in a web environment is described. The model is based on WebGL hardware acceleration and the filter-system architecture of the FabricJS library. It is oriented toward a formalized, per-component description of channel transformations (R, G, B, A) without considering the spatial neighborhood of a pixel, and is intended to ensure consistent execution of color correction, tonal correction, and image stylization operations for high-resolution images in browser applications, with reproducible results and stable algorithm behavior during interactive editing.

Purpose: to investigate and formalize a modular model of color and tonal filtering technologies for raster images in a browser environment based on WebGL and the FabricJS filter-system architecture.

Method: filters are represented as parameterized per-component RGBA transformations without spatial pixel neighborhoods; the architecture is built around BaseFilter and BlendFilter to unify WebGL execution (shaders, textures, framebuffers, swap logic), enable operator composition, and cache intermediate results; the practical implementation is developed in TypeScript/Angular and integrated with FabricJS.

Results: key transformation classes are described (4×5 color matrices, LUT/tonal curves, automatic brightness/contrast corrections, black-and-white conversion, exposure correction, and tinting) together with their mathematical models and parameterization; a GPU resource-management and texture/framebuffer caching model is proposed; the filtering subsystem is integrated into a hierarchical tree-like collage model; experimental observations confirm high performance and reproducible rendering during interactive high-resolution editing.

Scientific novelty: the browser GPU-filtering technology is improved by formalizing a modular WebGL+FabricJS architecture that ensures correct sequential composition (BaseFilter/BlendFilter), unified GPU execution, and reuse of intermediate states via caching.

Practical importance: the model serves as an engineering framework for building and extending web-oriented image and collage editors with GPU-accelerated effects, predictable quality, and scalable functionality without modifying the core.

Key words: WebGL; FabricJS; GPU acceleration; color correction; tonal correction; tonal curves; LUT; 4×5 matrix; texture caching; browser-based editors; image filtering.

Анотація. Описано модель кольорових і тональних технологій фільтрації графічних зображень у веб-середовищі, засновану на апаратному прискоренні WebGL та архітектурі системи фільтрів бібліотеки FabricJS. Модель орієнтована на формалізований опис покомпонентних перетворень каналів (R, G, B, A) без урахування просторової околиці пікселя та призначена для узгодженого виконання операцій кольорокорекції, тональної корекції й стилізації зображень високої роздільної здатності в браузерних додатках із відтворюваними результатами та стабільною поведінкою алгоритмів під час інтерактивного редагування.

Мета: дослідити та формалізувати модульну модель кольорових і тональних технологій фільтрації графічних зображень у браузерному середовищі на основі WebGL та архітектури системи фільтрів бібліотеки FabricJS.

Методика: фільтри подано як параметризовані оператори покомпонентних перетворень каналів (R, G, B, A) без урахування просторової околиці пікселя; архітектуру реалізовано через базові компоненти BaseFilter і BlendFilter з уніфікацією WebGL-виконання (шейдери, текстури, framebuffer, swap-логіка), підтримкою композиції операторів та кешування проміжних результатів; практичну реалізацію виконано засобами TypeScript і фреймворку Angular з інтеграцією у FabricJS.

Результати: описано класи перетворень (матриці 4×5, LUT і тональні криві, автоматичні корекції яскравості/контрасту, чорно-біле перетворення, експозиційна корекція, тонування) та наведено їх математичні моделі й параметризацію; запропоновано модель керування GPU-ресурсами і кешування текстур/кадробуферів; підсистему фільтрації інтегровано в ієрархічну деревовидну модель колажів; експериментальні спостереження підтверджують високу швидкодію та стабільну відтворюваність результатів під час інтерактивного редагування зображень високої роздільної здатності.

Наукова новизна: удосконалено технологію браузерної GPU-фільтрації шляхом формалізації модульної архітектури WebGL+FabricJS, що забезпечує коректну послідовну композицію перетворень (BaseFilter/BlendFilter), уніфіковане виконання на GPU та повторне використання проміжних станів через кешування.

Практична значимість: запропонована модель є інженерним каркасом для розроблення та розширення вебо-орієнтованих редакторів зображень і колажів із GPU-прискореними ефектами, прогнозованою якістю та масштабуванням функціоналу без модифікації ядра.

Ключові слова: WebGL; FabricJS; GPU-прискорення; кольорова корекція; тональна корекція; тональні криві; LUT; матриця 4×5; кешування текстур; браузерні редактори; фільтрація зображень.

ПОСТАВНОВКА ЗАДАЧІ

Інтерактивне опрацювання графічних зображень у веб-середовищі набуло значного поширення завдяки розвитку апаратного прискорення та впровадженню стандартизованих графічних інтерфейсів програмування, доступних безпосередньо в браузері. Використання HTML5 Canvas API [16] та WebGL API [17] дає змогу реалізовувати операції фільтрації, кольорокорекції та тональних перетворень із залученням обчислювальних ресурсів графічного процесора, що забезпечує виконання складних піксельних операцій у реальному часі. У поєднанні із сучасними JavaScript-бібліотеками та фреймворками це сприяло активному розвитку браузерних редакторів зображень, зокрема Adobe Photoshop [1], Canva [2], Pixlr [21] та Fotor [5].

Наукові дослідження останніх років підтверджують ефективність застосування GPU-прискорення для опрацювання растрових зображень у веб-браузерах. У працях Меєр Т. та ін. [15], Овенс Дж. Д. та ін. [19], Янг С. та ін. [29] показано, що використання шейдерних перетворень забезпечує паралельне опрацювання піксельних даних і дає змогу зменшити обчислювальні витрати порівняно з класичними CPU-реалізаціями. Роботи Гонзалес Р. та ін. [7], Ян А. [10], Прат В. [22] узагальнюють теоретичні основи ключових операцій фільтрації – лінійних кольорних перетворень, тональних корекцій та таблиць відповідності, тоді як специфікації GLSL ES і WebGL формалізують практичні засоби реалізації таких перетворень у браузері [26], [27].

Разом із тим більшість наявних онлайн-редакторів зображень орієнтовані насамперед на прикладний функціонал і використовують внутрішні або частково закриті архітектурні рішення, що ускладнює їх формалізований аналіз та подальше розширення [1], [2], [5],

[21]. Навіть у випадку відкритих бібліотек документація зазвичай зосереджується на описі окремих фільтрів без узагальненого представлення архітектури системи кольорових і тональних перетворень та принципів побудови складних послідовностей [3], [4].

Бібліотека FabricJS є одним із поширених інструментів для роботи з canvas-графікою у веб-середовищі та підтримує як CPU-, так і GPU-орієнтовані режими застосування фільтрів [3], [4]. Вона надає засоби розширення системи фільтрації, однак архітектурні аспекти організації базових кольорових і тональних фільтрів, а також логіка їх послідовного застосування залишаються недостатньо формалізованими в наукових публікаціях, що й зумовлює актуальність цього дослідження. У роботах [11–13] досліджено архітектурні технології побудови інтерактивних графічних моделей у веб-середовищі, зокрема питання структурної організації об'єктів та узгодженості операцій опрацювання.

АНАЛІЗ ОСТАННІХ ДОСЛІДЖЕНЬ ТА ПУБЛІКАЦІЙ

У дослідженні Меєр Т. та ін. [15] автори запропонували практичний метод GPU-прискореної фільтрації у браузері шляхом перехоплення викликів WebGL і модифікації процесу відтворення на рівні JavaScript. У такій постановці фільтрацію зручно формалізувати як піксельне перетворення у фрагментному шейдері, де для кожних координат (x, y) вихідний колір визначається параметризованою функцією:

$$c_{out}(x, y) = F(c_{in}(x, y); \theta) \quad (1)$$

де: $c = [R, G, B, A]^T$ – вектор RGBA, θ – параметри фільтра (матриця, коефіцієнти, текстури/маски), $F(c_{in}; \theta)$ – оператор фільтра (shader-функція).

У специфікації The Khronos Group [26] формалізовано мову GLSL ES та типові операції над векторами і матрицями, що безпосередньо використовуються

у кольорових фільтрах. Уніфікованим поданням лінійних кольорних перетворень може слугувати матрична модель:

$$[R', G', B', A']^T = M \cdot [R, G, B, A, 1]^T, M \in \mathbb{R}^{4 \times 5} \quad (2)$$

де: R, G, B, A – вхідні канали, R', G', B', A' – вихідні канали, M – матриця перетворення розміру 4×5 (остання колонка реалізує зсув/bias через множення на 1).

Ця модель є основою для широкого класу кольорових перетворень: регулювання яскравості, контрасту, насиченості, тонування та стилізації, що робить її універсальним інструментом для побудови систем кольорових фільтрів у вебсередовищі. Матричне подання дозволяє агрегувати кілька послідовних кольорних перетворень в одну операцію множення, зменшуючи кількість проходів GPU.

У монографії Джейн А. [10] наведено класичні підходи до тональної корекції та статистичних методів аналізу зображення. Для задач автонормалізації інтенсивностей у межах діапазону $[x_{\min}, x_{\max}]$ використовують лінійне відображення:

$$T(x) = \frac{x - x_{\min}}{x_{\max} - x_{\min}} \quad (3)$$

де: x – вхідна інтенсивність, $x_{\min}$, $x_{\max}$ – оцінені межі діапазону, T(x) – нормалізоване значення (з подальшим clamp до допустимого інтервалу).

У дослідженні Овенс Дж. [19] виконано огляд GPGPU-технологій, у якому підкреслено ключові фактори ефективності паралельних перетворень: інтенсивність обчислень і пропускну здатність пам'яті.

У монографії Матсуда К. та ін. [14] наведено системний виклад програмування WebGL і матричних перетворень геометрії, що є важливим для коректного відображення текстурованих прямокутників при фільтрації.

У монографії Гонзалес Р. [7] подано фундаментальні моделі цифрової фільтрації, що застосовуються і в GPU-реалізаціях. Монографії Джейн А. [10] та Прат В. [22] детально розглядають методи підсилення різкості та тональні корекції, що часто реалізуються через покомпонентні перетворення каналів. Окремим напрямом є прискорені кольорні перетворення на базі LUT, що набули розвитку в сучасних підходах до фото-покращення, зокрема із застосуванням 3D-LUT та інтерполяційних схем [28], [30].

У дослідженні Джо Й. [24] автори запропонували LUT-орієнтований підхід, у якому виходи навченої моделі переносяться у таблицю відповідності, а під час застосування виконуються швидкі звертання до попередньо обчислених значень.

У дослідженні Янг С. та ін. [28] автори розглянули SepLUT як separable image-adaptive LUT для покращення зображень у реальному часі, що підтверджує практичну доцільність LUT-операторів для браузерних GPU-процедур.

У стандарті IEC 61966-2-1:1999 [8] визначено перетворення sRGB↔linear, що напряму впливає на

коректність багатьох кольорових перетворень (контраст, експозиція, змішування). У рекомендації ITU-R BT.709-6 [9] формалізовано вагове перетворення RGB→Y (luma), що використовується у фільтрах яскравості та авто-корекціях.

У дослідженні Отсу Н. [18] автори запропонували метод Otsu для вибору порогу за гистограмою через максимум міжкласової дисперсії, що є релевантним для AutoLevels-подібних процедур.

ВІДОКРЕМЛЕННЯ НЕВИРІШЕНИХ РАНІШЕ ЧАСТИН ЗАГАЛЬНОЇ ПРОБЛЕМИ

Незважаючи на наявність численних досліджень у сфері цифрової обробки зображень і практичних реалізацій WebGL-фільтрації, відсутня цілісна формалізована модель покомпонентних (R, G, B, A) перетворень, інтегрована в архітектуру браузерних систем із забезпеченням відтворюваності результатів. Існуючі веборієнтовані редактори зазвичай реалізують фільтри як окремі ефекти без математично узгодженої композиції операторів, що ускладнює прогнозованість багатоступеневої обробки. Недостатньо дослідженими залишаються механізми керування GPU-ресурсами, зокрема моделі кешування текстур і кадробуферів для стабільної роботи під час інтерактивного редагування зображень високої роздільної здатності. Отже, невирішеною частиною проблеми є розроблення модульної формалізованої моделі браузерної GPU-фільтрації, яка поєднує коректну композицію перетворень, уніфіковане WebGL-виконання та кероване повторне використання проміжних результатів.

Мета дослідження – дослідити й описати моделі кольорових і тональних технологій фільтрації графічних зображень у браузерному середовищі на основі WebGL та бібліотеки FabricJS, а також проаналізувати архітектурні принципи побудови системи фільтрів і особливості реалізації покомпонентних перетворень каналів.

Для досягнення зазначеної мети визначено такі основні завдання дослідження:

- проаналізувати сучасні наукові та прикладні технології кольорової та тональної фільтрації графічних зображень у браузерному середовищі, зокрема з використанням апаратного прискорення WebGL, що дасть змогу визначити їхні переваги, обмеження та особливості реалізації у веборієнтованих системах графічного опрацювання;
- створити архітектуру системи кольорових та тональних фільтрів бібліотеки FabricJS, визначити призначення базових компонентів та принципи побудови моделей перетворень, що забезпечують виконання покомпонентних кольорних та тональних корекцій;
- розробити модель технології комбінування кольорових та тональних фільтрів у межах єдиного процесу опрацювання зображень та визначити умови коректності їх послідовного застосування на основі відомих моделей цифрового опрацювання зображень;

– виконати аналіз та реалізувати базові кольорові та тональні фільтри засобами WebGL, описати логіку їх застосування та взаємодії у процесі опрацювання растрових зображень у браузері.

МАТЕРІАЛИ ТА МЕТОДИ ДОСЛІДЖЕННЯ

Моделі кольорових та тональних технологій фільтрації графічних зображень та алгоритми їх застосування у браузерному середовищі засобами інформаційних технологій і систем. Для побудови моделі фільтрації використано метод подання фільтрів як параметризованих операторів покомпонентного опрацювання пікселів із можливістю їх послідовного комбінування в межах єдиного процесу опрацювання зображення. Для розроблення архітектури системи фільтрів та визначення правил їх коректного застосування використано теорію алгоритмів, а для систематизації типів перетворень – технології цифрового опрацювання зображень (колірні корекції, тональні перетворення, матричні моделі).

Практичну реалізацію створеної моделі виконано шляхом застосування фреймворку Angular, мов програмування TypeScript і HTML, canvas-технологій через бібліотеку FabricJS, а також WebGL як технологію апаратно прискореної фільтрації. Реалізація ґрунтується на уніфікованому інтерфейсі фільтра та базових класах BaseFilter і BlendFilter, що забезпечують повторне використання спільних методів ініціалізації ресурсів, параметризації та застосування перетворень.

Для організації програмної структури та реактивного оновлення стану використано патерни проектування Strategy, Composite та Observer, що дає змогу ефективно керувати змінами параметрів фільтрів і відображенням результатів у режимі реального часу. Кешування проміжних результатів та керування WebGL-текстурами зменшує накладні витрати при повторному застосуванні однакових перетворень.

Об'єкт дослідження – технології кольорової та тональної фільтрації графічних зображень у браузерному середовищі з використанням апаратного прискорення GPU.

Предмет дослідження – моделі кольорових і тональних технологій фільтрації графічних зображень, реалізовані на основі WebGL та бібліотеки FabricJS, архітектура системи фільтрів і принципи покомпонентних перетворень каналів.

ОСНОВНИЙ МАТЕРІАЛ. АРХІТЕКТУРА ТА ПРИНЦИПИ РОБОТИ МОДЕЛЕЙ ФІЛЬТРАЦІЇ

Модель архітектури системи фільтрації побудована як модульна ієрархія операторів опрацювання, орієнтована на розширюваність, повторне використання та ефективне виконання у вебсередовищі. Кожен фільтр описує локальне перетворення зображення або пари зображень, тоді як поєднання та порядок застосування фільтрів формуються у вигляді послідовності опрацювання з проміжними результатами.

Система реалізована як набір *FabricJS*-сумісних класів: кожен фільтр представлено TypeScript-модулем і реєструється в *FabricJS* (через *fabric.util.createClass*) як *fabric.Image.filters.**. Базовий клас *BaseFilter* є обгорткою над *fabric.Image.filters*. *BaseFilter* та узгоджує WebGL-виконання (шейдери, framebuffer/текстури, swar-логіка, область застосування) з єдиним API редактора.

1. Принцип модульності та контракт фільтра

Кожен фільтр реалізується як незалежний модуль із чітко визначеним контрактом:

- *вхідні дані*: зображення I або пара (I_b, I_s) , а також набір параметрів θ ;
- *вихідні дані*: зображення I' тієї ж або заданої розмірності;
- *детермінованість*: однакові вхідні дані та параметри завжди формують ідентичний результат;

Формально послідовність виконання фільтрації описується моделлю композиції:

$$I^{(0)} = I, I^{(k+1)} = F_k(I^{(k)}; \theta_k), I' = I^{(n)}, \quad (4)$$

де: I – вхідне зображення, $I^{(k)}$ – проміжний результат після k -го кроку, I' – вихідне зображення, $F_k(I^{(k)}; \theta_k)$ – k -й фільтр (оператор опрацювання), θ_k – параметри k -го фільтра, n – кількість фільтрів у ланцюгу опрацювання, k – індекс кроку (проходу).

Такий підхід дозволяє будувати складні ефекти як композиції простіших операторів та виконувати оптимізації (наприклад, агрегацію сумісних перетворень в один крок).

2. Багатопрхідне опрацювання та проміжні результати

Для алгоритмів, що потребують проміжних стадій (локальний контраст, саяво, автоматичні корекції), система підтримує багатопрхідну модель опрацювання:

$$I \rightarrow I^{(1)} \rightarrow I^{(2)} \rightarrow \dots \rightarrow I^{(n)}, \quad (5)$$

де: $I^{(t)}$ – результат після t -го проходу, n – загальна кількість проходів.

У GPU-режимі кожен прохід реалізується рендерингом у проміжні текстури з подальшим повторним використанням ресурсів.

3. Дворежимність виконання фільтрації: Canvas2D та WebGL

Під час ініціалізації бібліотеки FabricJS виконується задання режиму фільтрації. Можна використовувати одночасно тільки один з двох наведених режимів:

– *Canvas2D (CPU)*: прості операції, прямий доступ до піксельних даних, продуктивність обмежується послідовністю обчислень.

– *WebGL (GPU)*: паралельне виконання фрагментних шейдерів; ефективно для великих зображень, згорток, масок, багатопрхідних ефектів, потребує керування текстурами, синхронізації проходів і контролю пам'яті.

4. Управління ресурсами та кешування

Для зменшення накладних витрат інтерактивного редагування система застосовує:

- Кешування текстур і framebuffer;
- Мінімізацію повторних алокацій пам'яті;
- Повторне використання проміжних результатів за незмінних параметрів;

Практично це реалізується через:

- *Contexture* як обгортку WebGL-текстури (посилання + розміри + контекст);
- *GLMemoryManager* для виділення/звільнення GPU-ресурсів (наприклад, *releaseTexture()* для тимчасових текстур);
- *Cache* (локальний кеш фільтра) та *TextureCache* (глобальний кеш проміжних результатів);

Формально кешування можна подати як правило заміни обчислення:

$$\text{if } key(F, \theta, I) = \text{const} \Rightarrow I' = \text{Cache}[key], \quad (6)$$

де: $\mathcal{F}$ – оператор (фільтр), θ – параметри оператора, I – вхідне зображення/стан, $key(F, \theta, I)$ – функція формування ключа стану (ідентифікатора) для кешу, $\text{Cache}[key]$ – кешований результат для заданого ключа, I' – вихідне зображення.

5. Інтеграція з деревовидною моделлю колажів з поперечними зв'язками

Система фільтрації інтегрована в деревовидну модель колажів як частина обчислювального графа. Фільтри можуть застосовуватися на рівні окремих вузлів композиції, результати узгоджуються з масками, альфа-каналами та правилами накладання. Відтак, фільтрація є елементом моделі функціонування редактора, а не зовнішньою процедурою постопрацювання.

Узагальнена піксельна модель

Нехай піксель:

$$p = (R, G, B, A), R, G, B, A \in [0, 1], \quad (7)$$

фільтр задає відображення:

$$p' = \mathcal{F}(p, x), \quad x = (x, y), \quad (8)$$

де: p – вектор поточного пікселя, R, G, B, A – компоненти кольору та альфа в нормованому діапазоні $[0, 1]$, $(R, G, B, A)^T$ – операція транспонування, p' – вихідний (опрацьований) піксель, $\mathcal{F}(p, x)$ – оператор фільтрації, θ – параметри фільтра, $x = (x, y)$ – координати пікселя.

Надалі $\text{clamp}(z, 0, 1)$ означає обрізання значення до $[0, 1]$:

$$\text{clamp}(z, 0, 1) = \min(1, \max(0, z)). \quad (9)$$

На рис. 1 наведено графічне зображення у початковому вигляді без накладених фільтрів з розмірами 1920×2400 пікселів. На основі поданого зображення проведено часові заміри виконання накладання кольорових та тональних фільтрів у режимі WebGL.

1) BaseFilter

Мета: базовий клас сумісності з FabricJS; уніфікація WebGL-виконання, областей застосування, серіалізації та кешу. Принцип: підготовка framebuffer

і target texture, застосування шейдера, swap source/target, кешування проміжного результату. Для зменшення накладних витрат інтерактивного редагування система застосовує кешування текстур і framebuffer, мінімізацію повторних алокацій пам'яті та повторне використання проміжних результатів за незмінних параметрів. Формально правило кешування (6).



Рис. 1. Зображення без накладених фільтрів / Pic. 1. Image without applied filters

2) BlendFilter



Рис. 2. Приклад накладення BlendFilter (час виконання = 0.05 секунди) / Pic. 2. Example of BlendFilter applied (execution time = 0.05 seconds)

Мета: накладання результату на базовий шар з параметрами *blendAlpha*, *blendMode*, опційною маскою.

Принцип: спочатку обчислюється C_s (результат), потім виконується змішування з базою C_b . Модель (узагальнена):

$$C' = (1 - \alpha)C_b + \alpha B(C_b, C_s), \alpha \in [0, 1], \quad (10)$$

де: C' – результатний колір, C_b – колір базового шару (оригінал), C_s – колір “верхнього шару” (результат фільтра або проміжний стан), α – коефіцієнт інтенсивності (*blendAlpha*), $B(C_b, C_s)$ – оператор режиму змішування (*blendMode*), $[0, 1]$ – допустимий діапазон параметра α .

3) NestedFilter



Рис. 3. Приклад накладення NestedFilter [BlurFilter([blur, blur]), AdjustCurvesFilter(MasterCurve = [{x: 0, y: 0}, {x: 128, y: (128 + (brightness - 50) * 75 / 50)}, {x: 255, y: 255}])](час виконання = 0.0389 секунди) / Pic. 3. Example of NestedFilter [BlurFilter([blur, blur]), AdjustCurvesFilter(MasterCurve = [{x: 0, y: 0}, {x: 128, y: (128 + (brightness - 50) * 75 / 50)}, {x: 255, y: 255}])] applied (execution time = 0.0389 seconds)
a: (blur: 25, brightness: 50)
b: (blur: 50, brightness: 75)
c: (blur: 50, brightness: 25)

Мета: об'єднання групи підфільтрів subFilters у єдиний ефект (багатопрхідний модуль).

Принцип: послідовне застосування $F_1 \dots F_n$ з коректним веденням лічильника проходів.

Модель:

$$I' = (F_n \circ F_{n-1} \circ \dots \circ F_1)(I), \quad (11)$$

де: I – вхідне зображення, I' – вихідне зображення, F_i – i -й підфільтр у контейнері, “ $\circ$ ” – композиція операторів (застосування справа наліво), n – кількість підфільтрів у NestedFilter.

4) PaletteMapFilter



Рис. 4. Приклад накладення PaletteMapFilter (час виконання = 0.032 секунди) / Pic. 4. Example of PaletteMapFilter applied (execution time = 0.032 seconds)

Мета: LUT-перетворення (криві/авто-корекції) з попереднім обчисленням таблиць відповідностей.

Принцип: обчислити LUT $\rightarrow$ застосувати LUT покомпонентно.

Модель:

$$R' = \text{LUT}_R(R), G' = \text{LUT}_G(G), B' = \text{LUT}_B(B), A' = A, \quad (12)$$

де: R, G, B, A – вхідні компоненти пікселя, R', G', B', A' – вихідні компоненти, $\text{LUT}_R, \text{LUT}_G, \text{LUT}_B$ – таблиці відповідності (look-up tables) для каналів, $\text{LUT}(x)$ – відображення “вхід $\rightarrow$ вихід” (зазвичай у дискретній шкалі 0...255 або нормованій 0...1).

5) AdjustCurvesFilter



Рис. 5. Приклад накладення AdjustCurvesFilter для ефекту інвертування (час виконання = 0.0238 секунди, задання параметрів Master кривій масиву [{x: 0, y: 255}, {x: 255, y: 0}]) / Pic. 5. Example of applying AdjustCurvesFilter for invert effect (execution time = 0.0238 seconds, setting the Master curve array parameters [{x: 0, y: 255}, {x: 255, y: 0}])

Мета: нелінійна корекція тону (Master/R/G/B) через криві $\rightarrow$ LUT.

Принцип: контрольні точки $\rightarrow$ інтерполяція $\rightarrow$ LUT $\rightarrow$ застосування LUT.

Модель:

$$x' = C(x), x \in [0, 255], R' = C_R(R), G' = C_G(G), B' = C_B(B), \quad (13)$$

де: x – вхідна інтенсивність у шкалі 0...255, x' – вихідна інтенсивність, $C(x)$ – тональна крива (Master), C_R, C_G, C_B – криві для каналів R/G/B, (R, G, B) – вхідні значення каналів (0...255 або нормовані), R', G', B' – вихідні значення каналів.

6) AutoLevelsFilter



Рис. 6. Приклад накладення AutoLevelsFilter (час виконання = 0.0658 секунди) / Pic. 6. Example of AutoLevelsFilter applied (execution time = 0.0658 seconds)

Мета: автоматичне вирівнювання рівнів/контрасту на основі статистик/гістограм.

Принцип: оцінити $(x_{\min}, x_{\max}) \rightarrow$ застосувати рісесвіс-нормалізацію (як LUT).

Модель:

$$T(x) = \begin{cases} 0, & x < x_{\min}, \\ 255 \cdot \frac{x - x_{\min}}{x_{\max} - x_{\min}}, & x_{\min} \leq x \leq x_{\max}, x \in [0, 255], \\ 255, & x > x_{\max}, \end{cases} \quad (14)$$

де: x – вхідна інтенсивність (0...255), $T(x)$ – скоригована інтенсивність, $x_{\min}$ – нижній поріг (мінімум за

статистикою/гістограмою), $x_{\max}$ – верхній поріг (максимум за статистикою/гістограмою), 0 та 255 – межі вихідного діапазону після нормалізації.

7) AutoLuminanceFilter



Рис. 7. Приклад накладення AutoLuminanceFilter (час виконання = 0.0291 секунди) / Pic. 7. Example of AutoLuminanceFilter applied (execution time = 0.0291 seconds)

Мета: автоматична корекція яскравості з гамма-адаптацією до середнього значення μ . **Принцип:** обчислити $\mu \rightarrow$ оцінити $\gamma \rightarrow$ застосувати гамма-перетворення (через LUT).

Модель:

$$\gamma = \frac{\ln(0.5)}{\ln(\mu/255)}, T(x) = 255 \left(\frac{x}{255} \right)^\gamma, \quad (15)$$

де: μ – середня яскравість (у шкалі 0...255), γ – параметр гамма-корекції,

x – вхідна інтенсивність (0...255), $T(x)$ – вихідна інтенсивність після гамма-перетворення, 255 – масштабування до 8-бітної шкали.

8) AutoFixFilter



Рис. 8. Приклад накладення AutoFixFilter (час виконання = 0.0467 секунди) / Pic. 8. Example of AutoFixFilter applied (execution time = 0.0467 seconds)

Мета: авто-корекція на основі гістограм (режими збереження балансу/вирівнювання).

Принцип: аналіз гістограми $\rightarrow$ формування LUT $\rightarrow$ застосування LUT.

Класична модель через CDF:

$$\text{LUT}(x) = 255 \cdot \text{CDF}(x), \text{CDF}(x) = \frac{1}{N} \sum_{i=0}^x h(i), \quad (16)$$

де: $h(i)$ – гістограма (кількість пікселів з інтенсивністю i), i – значення інтенсивності (0...255), N – загальна кількість пікселів (нормувальний

множник), $\text{CDF}(x)$ – кумулятивна функція розподілу до рівня x , $255 \cdot \text{CDF}(x)$ – операція взяття цілої частини (floor), $\text{LUT}(x)$ – таблиця відповідності для перетворення інтенсивностей.

9) AutoFix2Filter



Рис. 9. Приклад накладення AutoFix2Filter (час виконання = 0.0718 секунди) / Pic. 9. Example of AutoFix2Filter applied (execution time = 0.0718 seconds)

Мета: композиційний авто-пресет як послідовність базових операторів.

Принцип: реалізація як NestedFilter: AutoLevels $\rightarrow$ LocalContrast $\rightarrow$ ColorMatrix (типова схема).

Модель:

$$\mathcal{F}_{\text{AutoFix2}} = \mathcal{F}_{\text{AutoLevels}} \circ \mathcal{F}_{\text{LocalContrast}} \circ \mathcal{F}_{\text{ColorMatrix}}, \quad (17)$$

де: $\mathcal{F}_{\text{AutoFix2}}$ – оператор авто-пресета, $\mathcal{F}_{\text{AutoLevels}}$ – оператор автоматичних рівнів, $\mathcal{F}_{\text{LocalContrast}}$ – оператор локального контрасту, $\mathcal{F}_{\text{ColorMatrix}}$ – оператор колірної матриці, “ $\circ$ ” – композиція операторів.

10) ColorMatrixFilter

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}, \begin{bmatrix} 0.75 & 0 & 0 & 0 & 0 \\ 0 & 0.75 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

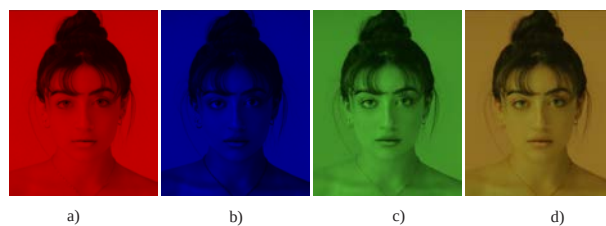


Рис. 10. Приклад накладення ColorMatrixFilter (a, b, c, d) з різними коефіцієнтами зсуву та їх матриці (час виконання 0.0123_м секунди) / Pic. 10. Example of ColorMatrixFilter applied (a, b, c, d) with different shift coefficients and their matrices (execution time 0.0123_м seconds)

Мета: універсальне афінне перетворення кольору (корекції/стилізації).

Принцип: застосування матриці 4×5 до $[R, G, B, A, 1]^T$.

Модель:

$$\begin{bmatrix} R' \\ G' \\ B' \\ A' \end{bmatrix} = \begin{bmatrix} m_{00} & m_{01} & m_{02} & m_{03} & m_{04} \\ m_{10} & m_{11} & m_{12} & m_{13} & m_{14} \\ m_{20} & m_{21} & m_{22} & m_{23} & m_{24} \\ m_{30} & m_{31} & m_{32} & m_{33} & m_{34} \end{bmatrix} \begin{bmatrix} R \\ G \\ B \\ A \\ 1 \end{bmatrix}, \quad (18)$$

де: R, G, B, A – вхідні компоненти пікселя, R', G', B', A' – вихідні канали після перетворення, $M_{4 \times 5}$ – матриця афінного перетворення кольору, m_{ij} – елементи матриці (коефіцієнти змішування/зсуву), “1” у векторі – однорідна компонента для доданка зсуву (bias).

11) SimpleColorMatrixFilter



Рис. 11. Приклад накладення SimpleColorMatrixFilter (час виконання = 0.0134_м секунди) / Pic. 11. Example of SimpleColorMatrixFilter applied (execution time = 0.0134_м seconds)

a: (brightness: -20, saturation: 20, contrast: 40),
b: (brightness: -30, saturation: 30, contrast: 60),
c: (brightness: -40, saturation: 40, contrast: 80)

Мета: спрощений ColorMatrixFilter (пресетні/ типові матриці).

Принцип і модель: як у ColorMatrixFilter (31), але з обмеженим набором параметрів/матриць.

12) MultiplyColorMatrixFilter

$$\begin{bmatrix} 1.6 & 0 & 0 & 0 & 0 \\ 0 & 1.6 & 0 & 0 & 0 \\ 0 & 0 & 1.6 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}, \begin{bmatrix} 1.8 & 0 & 0 & 0 & 0 \\ 0 & 1.8 & 0 & 0 & 0 \\ 0 & 0 & 1.8 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

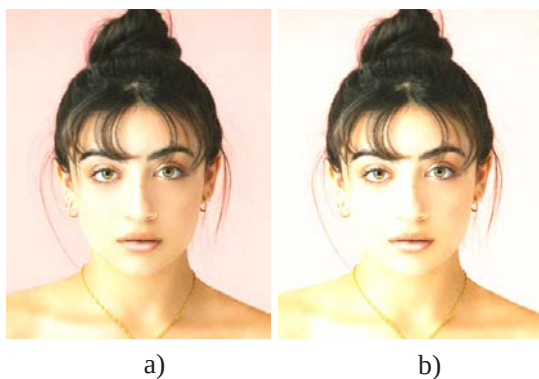


Рис. 12. Приклад накладення MultiplyColorMatrixFilter (a, b) з різними коефіцієнтами множника та їх матриці (час виконання 0.0105_м секунди) / Pic. 12. Example of MultiplyColorMatrixFilter applied (a, b) with different multiplier coefficient and their matrices (execution time 0.0105_м seconds)

Мета: злиття кількох ColorMatrix-перетворень в одне через множення.

Принцип: композицію двох афінних перетворень подати як один оператор.

Модель:

$$M_{\text{total}} = M_2 \cdot M_1, P' = M_{\text{total}}[p; 1], \quad (19)$$

де: M_1, M_2 – матриці окремих кольорних перетворень (формату 4×5 або еквівалентного), M_{total} – зведена (агрегована) матриця, “.” – матричне множення, $[p; 1]$ – вхідний вектор пікселя з однорідною компонентою, p' – вихідний піксель після зведеного перетворення.

13) BWFilter



Рис. 13. Приклад накладення BWFilter (час виконання = 0.0573 секунди) / Pic. 13. Example of BWFilter applied (execution time = 0.0573 seconds)

Мета: чорно-біле перетворення з нормалізацією ваг каналів.

Принцип: вагова сума каналів із нормалізацією коефіцієнтів.

Модель:

$$\tilde{w}_i = \frac{w_i f_i}{w_R f_R + w_G f_G + w_B f_B}, i \in \{R, G, B\},$$

$$Y = \tilde{w}_R R + \tilde{w}_G G + \tilde{w}_B B, R' = G' = B' = Y, A' = A, \quad (20)$$

де: w_i – базова вага каналу i , f_i – множник/коефіцієнт налаштування для каналу i , $\tilde{w}_i$ – нормалізована вага, R, G, B, A – вхідні компоненти, Y – скаляр яскравості (luma), R', G', B' – вихідні компоненти після перетворення в сірий, A' – вихідна альфа (без змін).

14) ExposureFilter



Рис. 14. Приклад накладення ExposureFilter (час виконання = 0.0325 секунди) / Pic. 14. Example of ExposureFilter applied (execution time = 0.0325 seconds) (levelMin = 63.5, levelMax = 191.5, exposure = 71, contrast = 50)

Мета: експозиційна корекція (масштабування інтенсивності).

Принцип: множення каналів на 2° з обрізанням до $[0, 1]$.

Модель:

$$R' = \text{clamp}(R \cdot 2^e, 0, 1), G' = \text{clamp}(G \cdot 2^e, 0, 1),$$

$$B' = \text{clamp}(B \cdot 2^e, 0, 1), A' = A, \quad (21)$$

де: R,G,B,A – вхідні компоненти (0...1), R',G',B',A' – вихідні компоненти, e – параметр експозиції, 2^e – коефіцієнт підсилення/послаблення, clamp($B \cdot 2^e, 0, 1$) – обрізання до [0,1].

15) TintFilter

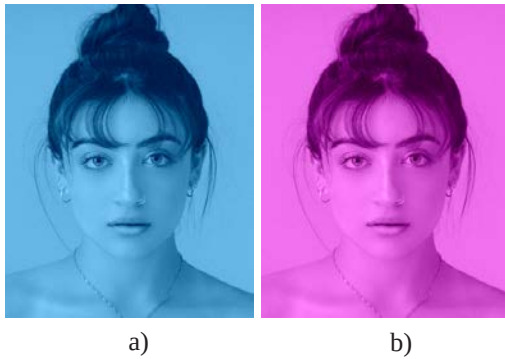


Рис. 15. Приклад накладення TintFilter (час виконання = 0.041_м секунди) / Pic. 15. Example of TintFilter applied (execution time = 0.041_м seconds)
a: (color = #80CFFF),
b: (color = #FF80FC)

Мета: тонування зображення заданим кольором T.

Принцип: лінійна інтерполяція між C та T.

Модель:

$$C' = (1 - \alpha)C + \alpha T, \quad (22)$$

де: C – вхідний колір (вектор RGB), T – колір тону (target tint), α – інтенсивність тонування, C' – вихідний колір після інтерполяції.

16) SimpleTintFilter



Рис. 16. Приклад накладення SimpleTintFilter (час виконання = 0.0457_м секунди) / Pic. 16. Example of SimpleTintFilter applied (execution time = 0.0457_м seconds)
color = #6C785C, a: channel = RED, b: channel = GREEN, c: channel = BLUE

Мета: спрощене тонування (preset/обмежені параметри).

Принцип і модель: як у TintFilter (35).

На рис. 17 подано діаграму класів підсистеми модульної фільтрації зображень у браузерному редакторі (WebGL + FabricJS), яка демонструє ієрархію наслідування та зв'язки між основними компонентами системи.

Отримані результати демонструють, що реалізована підсистема кольорових та тональних фільтрів на базі WebGL + FabricJS забезпечує одночасно формальну коректність (чітко визначені оператори: LUT/криві, матриця 4x5, автоматичні корекції, тонування/експозиція) та інтерактивну ефективність (покомпонентне опрацювання без повернення даних на CPU, кешування результатів і повторне використання текстур).

ОБГОВОРЕННЯ

РЕЗУЛЬТАТИ ДОСЛІДЖЕННЯ

У роботі описано практики браузерного редагування зображень у продукті Photoshop [1], де ключовим є поєднання інтерактивності та якості ефектів.

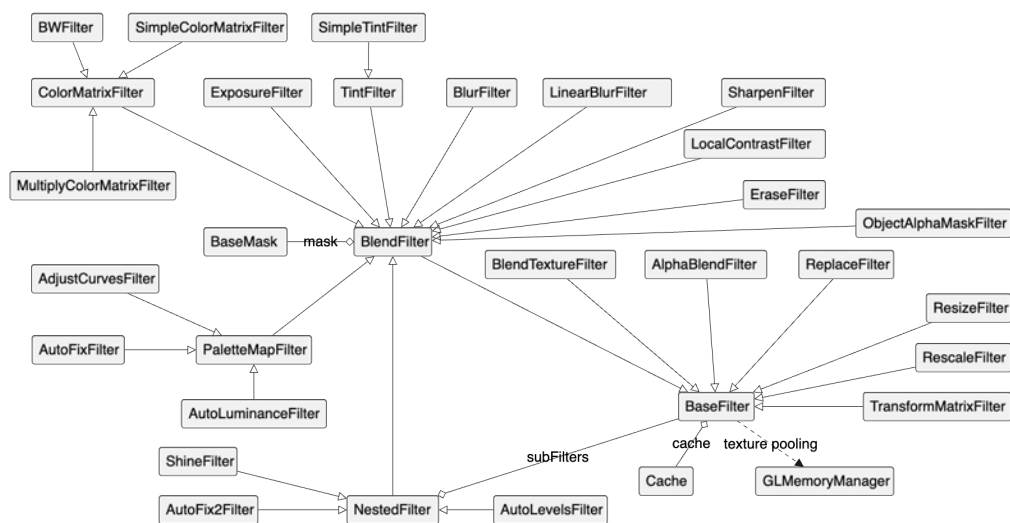


Рис. 17. Діаграма класів підсистеми модульної фільтрації зображень у браузерному редакторі (WebGL + FabricJS) / Pic. 17. Class diagram of the modular image filtering subsystem in a browser-based editor (WebGL + FabricJS)

У розроблених моделях кольорових і тональних фільтрів задані параметри та правила комбінування, що підвищує відтворюваність поведінки системи при розширенні набору ефектів.

У роботі Canva [2] наведено веборієнтовану модель застосування фоторедагувальних операцій, орієнтованих на масовий користувацький сценарій. У нашій реалізації фокус зміщено на керованість параметрів кольорових перетворень і на формальну коректність послідовностей, що особливо важливо при інтеграції фільтрації в модель колажу.

У документації FabricJS [3] описано загальні принципи застосування фільтрів у FabricJS та можливість виконання у різних режимах. У розробленій моделі цей рівень узагальнення доповнено модульною архітектурою кольорових і тональних фільтрів, механізмами керування проміжними результатами та правилами коректного комбінування.

У матеріалах Fotor [5] представлено приклад прикладного онлайн-редактора, де користувацькі ефекти подаються як набір типових фотооперацій. Порівняння з нашою роботою підкреслює перевагу формалізованої моделі: при наявності математично визначених операторів легше забезпечити однаковість результатів для різних комбінацій ефектів.

У стандарті IEC 61966-2-1:1999 [8] визначено перетворення $sRGB \leftrightarrow linear$, що напряму впливає на коректність кольорових перетворень (контраст, експозиція, тонування). Наші результати підтверджують необхідність явної фіксації колірної простору в системі фільтрації, оскільки одна і та сама формула дає різні візуальні результати у $linear$ і $gamma$ -просторах.

У рекомендації ITU-R BT.709-6 [9] формалізовано вагове перетворення $RGB \rightarrow Y$ (luma), що використовується у фільтрах яскравості, авто-корекціях і в BWFilter. У нашій моделі це важливо для коректної побудови статистик і для узгодженого застосування тональних перетворень без небажаного зміщення кольорів.

У дослідженні Отсу Н. [18] автори запропонували метод Otsu для вибору порогу за гистограмою через максимум міжкласової дисперсії. Це є релевантним для нашої інтерпретації AutoLevels-подібних процедур: замість фіксованих евристик доцільно використовувати відтворювані критерії вибору порогів, особливо коли фільтр є частиною складної послідовності перетворень.

У дослідженні Джо Й. [24] автори запропонували LUT-орієнтований підхід, у якому виходи навченої моделі переносяться у таблицю відповідності. У контексті нашої системи це є додатковим аргументом на користь модульності: LUT може виступати як компонента, яка отримує таблицю як параметр, а спосіб її побудови може бути взаємозамінним.

У дослідженні Янг С. [28] автори розглянули SepLUT як separable image-adaptive LUT для

покращення зображень у реальному часі, що прямо узгоджується з нашим підходом, де аналітична частина може формувати $T(x)$, після чого виконується швидка піксельна трансформація. У дослідженні Джанг Й. [30] автори запропонували 3D-LUT підхід із семантичними компенсаціями, що підкреслює перспективність розширення нашої системи.

У підсумку порівняння з прикладними вебредакторами Adobe Photoshop [1], Canva [2], Fotor [5], Photopea [20], Pixlr [21] показує, що такі системи ефективні як готові продукти, але не надають відкритої архітектурної моделі для перенесення у власні моделі систем та інформаційних технологій. Натомість результати нашого дослідження формують інженерну основу: модульність, прогнозованість, керованість і можливість розширення системи кольорових та тональних фільтрів.

Отже, внаслідок виконаної роботи можна сформулювати такі наукову новизну та практичну значущість результатів дослідження.

Наукова новизна отриманих результатів – удосконалено технологію браузерної GPU-фільтрації шляхом розроблення та формалізації модульної архітектури системи кольорових та тональних фільтрів на основі WebGL і FabricJS, що передбачає представлення фільтрів як параметризованих операторів покомпонентних перетворень каналів із чітким зв'язком вхід/вихід, введення ієрархії базових класів (BaseFilter/BlendFilter) для послідовної композиції, інтеграцію LUT-перетворень та автоматичних корекцій, а також модель кешування проміжних результатів для зменшення повторних алокацій та забезпечення стабільної інтерактивної швидкодії.

Практична значущість результатів дослідження – розроблена модель є основою для створення й розширення інженерних веборієнтованих редакторів зображень і колажів, у яких потрібні GPU-прискорені кольорові та тональні ефекти, повторне використання модулів, прогнозована якість результатів та масштабування функціоналу без модифікації ядра. Система може застосовуватися у професійній діяльності дизайнерів, фотографів та контент-менеджерів, забезпечуючи інтерактивність і можливість інтеграції у вебплатформи з використанням Angular/TypeScript і FabricJS.

ВИСНОВКИ

У результаті дослідження розроблено, формалізовано та програмно реалізовано моделі кольорових та тональних технологій фільтрації графічних зображень у браузерному середовищі на основі WebGL і бібліотеки FabricJS. За результатами виконаного дослідження можна зробити такі основні висновки.

Встановлено, що наявні прикладні вебредактори та частина відкритих реалізацій переважно не надають прозорої формалізації архітектури GPU-фільтрації кольорових перетворень, правил комбінування ефектів

та управління проміжними станами, що ускладнює відтворюваний аналіз та перенесення технологій.

Розроблено та описано модульну систему кольорових і тональних фільтрів як композицію параметризованих операторів із визначеним зв'язком вхід/вихід, що забезпечує коректне послідовне застосування перетворень та уніфікує виконання в межах спільного інтерфейсу FabricJS-сумісних класів.

Створено архітектурну реалізацію на основі базових класів BaseFilter і BlendFilter із підтримкою двох режимів виконання (Canvas2D та WebGL), що забезпечує відтворюваність обчислень та модульність системи кольорових перетворень.

Реалізовано моделі інформаційних технологій покомпонентних перетворень каналів: LUT/

криві, афінні колірні матриці 4×5, автоматичні корекції яскравості/контрасту, чорно-біле перетворення, експозиційна корекція та тонування.

Обґрунтовано та створено модель кешування проміжних результатів кольорових перетворень (повторне використання текстур, мінімізація алокацій), що знижує накладні витрати при повторних застосуваннях фільтрів у реальному часі.

Зверифіковано коректність та практичну придатність створених моделей програмною реалізацією з використанням TypeScript, Angular, FabricJS, HTML5 Canvas і WebGL, що підтвердило можливість побудови повноцінної браузерної системи кольорових і тональних перетворень із формалізованими правилами комбінування ефектів.

REFERENCES

- [1] Adobe Photoshop. (n.d.). Photoshop.adobe.com. Retrieved from: <https://photoshop.adobe.com/discover>
- [2] Canva. (2025). Canva. Retrieved from: <https://www.canva.com/>
- [3] FabricJS. (n.d.). Fabric.js documentation. URL: <https://fabricjs.com/docs/>
- [4] FabricJS. (n.d.). fabric.js (source repository). Retrieved from: <https://github.com/fabricjs/fabric.js>
- [5] Fotor. (n.d.). Online photo editor. Retrieved from: <https://www.fotor.com/features/photo-editor/>
- [6] Franke, L., & Haehn, D. (2020). Modern Scientific Visualizations on the Web. *Informatics*, 7(4), 37. Retrieved from <https://doi.org/10.3390/informatics7040037>
- [7] Gonzalez, R. C., & Woods, R. E. (2018). *Digital image processing* (4th ed.). Pearson.
- [8] IEC. (1999). IEC 61966-2-1:1999: Multimedia systems and equipment — Colour measurement and management — Part 2-1: Default RGB colour space — sRGB.
- [9] ITU-R. (2015). Recommendation ITU-R BT.709-6: Parameter values for the HDTV standards for production. International Telecommunication Union.
- [10] Jain, A. K. (1989). *Fundamentals of digital image processing*. Prentice Hall.
- [11] Kordiuk, R. I. (2025). TREE-LIKE COLLAGE MODEL OF INFORMATION SYSTEM. *Scientific Notes of Taurida National V. I. Vernadsky University*, 2(3), 229–240. Retrieved from <https://doi.org/10.32782/2663-5941/2025.3.2/31>
- [12] Kordiuk, R. I., & Ovsia, O. V. (2025). Derevopodibna model kolazhiv z poperechnymy zviazkamy [Tree-like model of collages with cross-links]. *Scientific Bulletin of UNFU*, 35(4), 123–133. Retrieved from <https://doi.org/10.36930/40350414>
- [13] Kordiuk, R. I. (2025). Model opratsiuvannia kolazhiv informatsiinoi systemy [Model for processing collages of an information system]. *Visnyk KhNTU*, 2(2(93)), 202–217. Retrieved from <https://doi.org/10.35546/kntu2078-4481.2025.2.2.25>
- [14] Matsuda, K., & Lea, R. (2013). *WebGL programming guide: Interactive 3D graphics programming with WebGL*. Addison-Wesley Professional.
- [15] Meyer, T., Gabi Dreo Rodosek, & Haehn, D. (2024). WebGL-based Image Processing through JavaScript Injection. 1–5. Retrieved from <https://doi.org/10.1145/3665318.3677163>
- [16] Mozilla. (n.d.). Canvas API. MDN Web Docs. Retrieved from https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API
- [17] Mozilla. (n.d.). WebGL API. MDN Web Docs. Retrieved from https://developer.mozilla.org/en-US/docs/Web/API/WebGL_API
- [18] Otsu, N. (1979). A Threshold Selection Method from Gray-Level Histograms. *IEEE Trans. Syst., Man, Cybern.*, 9(1), 62–66. Retrieved from <https://doi.org/10.1109/tsmc.1979.4310076>
- [19] Owens, J. D. et al. (2007). A Survey of General-Purpose Computation on Graphics Hardware. *Computer Graphics Forum*, 26(1), 80–113. Retrieved from <https://doi.org/10.1111/j.1467-8659.2007.01012.x>
- [20] Photopea. (n.d.). Photopea online photo editor. Retrieved from <https://www.photopea.com/>
- [21] Pixlr. (n.d.). Pixlr E (advanced photo editor). Retrieved from <https://pixlr.com/editor/>
- [22] Pratt, W. K. (2007). *Digital image processing: PIKS scientific inside* (4th ed.). Wiley-Interscience.
- [23] Reinhard, E., Stark, M., Shirley, P., & Ferwerda, J. A. (2002). Photographic tone reproduction for digital images. *ACM Trans. Graph.* Retrieved from <https://doi.org/10.1145/566570.566575>
- [24] Jo, Y., & Kim, S. J. (2021). Practical Single-Image Super-Resolution Using Look-Up Table. 691–700. Retrieved from <https://doi.org/10.1109/cvpr46437.2021.00075>
- [25] The Khronos Group. (n.d.). OpenGL ES 2.0 Specification. Retrieved from https://registry.khronos.org/OpenGL/specs/es/2.0/es_full_spec_2.0.pdf
- [26] The Khronos Group. (n.d.). The OpenGL ES Shading Language (GLSL ES) Specification. Retrieved from https://registry.khronos.org/OpenGL/specs/es/2.0/GLSL_ES_Specification_1.00.pdf
- [27] The Khronos Group. (n.d.). WebGL 2.0 Specification. Retrieved from <https://registry.khronos.org/webgl/specs/latest/2.0/>
- [28] Yang, C., Jin, M., Xu, Y., Zhang, R., Chen, Y., & Liu, H. (2022). SepLUT: Separable Image-adaptive Lookup Tables for Real-time Image Enhancement. *ArXiv* (Cornell University). Retrieved from <https://doi.org/10.48550/arxiv.2207.08351>

- [29] Yao, Z., Ma, Z., Liu, Y., Sani, A. A., & Aparna Chandramowliswaran. (2018). Sugar. 519–534. Retrieved from <https://doi.org/10.1145/3173162.3173186>
- [30] Zhang, Y., Xu, W., & Lyu, C. (2024). Semantic-aware enhancement: Integrating semantic compensation with 3D-LUT for low-light image enhancement. *Eng. Appl. Artif. Intell.*, 130, 107793. Retrieved from <https://doi.org/10.1016/j.engappai.2023.107793>

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

- [1] Adobe Photoshop. *Photoshop.adobe.com*. URL: <https://photoshop.adobe.com/discover>
- [2] Canva. *Canva*. 2025. URL: <https://www.canva.com/>
- [3] FabricJS. *Fabric.js documentation*. URL: <https://fabricjs.com/docs/>
- [4] FabricJS. *fabric.js (source repository)*. URL: <https://github.com/fabricjs/fabric.js>
- [5] Fotor. *Online photo editor*. URL: <https://www.fotor.com/features/photo-editor/>
- [6] Franke L., Haehn D. Modern Scientific Visualizations on the Web. *Informatics*. 2020. Vol. 7, No. 4. P. 37. DOI: <https://doi.org/10.3390/informatics7040037>
- [7] Gonzalez R. C., Woods R. E. *Digital image processing*. 4th ed. Pearson, 2018.
- [8] IEC 61966-2-1:1999. *Multimedia systems and equipment – Colour measurement and management – Part 2-1: Default RGB colour space – sRGB*. IEC, 1999.
- [9] Recommendation ITU-R BT.709-6: Parameter values for the HDTV standards for production. International Telecommunication Union, 2015.
- [10] Jain A. K. *Fundamentals of digital image processing*. Prentice Hall, 1989.
- [11] Kordiuk R. I. Tree-like collage model of information system. *Scientific Notes of Taurida National V. I. Vernadsky University*. 2025. Vol. 2, No. 3. P. 229–240. DOI: <https://doi.org/10.32782/2663-5941/2025.3.2/31>
- [12] Кордюк Р. І., Овсяк О. В. Деревоподібна модель колажів з поперечними зв'язками. *Scientific Bulletin of UNFU*. 2025. Т. 35, № 4. С. 123–133. DOI: <https://doi.org/10.36930/40350414>
- [13] Кордюк Р. І. Модель опрацювання колажів інформаційної системи. *Вісник ХНТУ*. 2025. Т. 2, № 2(93). С. 202–217. DOI: <https://doi.org/10.35546/kntu2078-4481.2025.2.2.25>
- [14] Matsuda K., Lea R. *WebGL programming guide: Interactive 3D graphics programming with WebGL*. Addison-Wesley Professional, 2013.
- [15] Meyer T., Gabi Dreio Rodosek, Haehn D. WebGL-based Image Processing through JavaScript Injection. 2024. P. 1–5. DOI: <https://doi.org/10.1145/3665318.3677163>
- [16] Mozilla. *Canvas API*. MDN Web Docs. URL: https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API
- [17] Mozilla. *WebGL API*. MDN Web Docs. URL: https://developer.mozilla.org/en-US/docs/Web/API/WebGL_API
- [18] Otsu N. A Threshold Selection Method from Gray-Level Histograms. *IEEE Transactions on Systems, Man, and Cybernetics*. 1979. Vol. 9, No. 1. P. 62–66. DOI: <https://doi.org/10.1109/tsmc.1979.4310076>
- [19] Owens J. D. et al. A Survey of General-Purpose Computation on Graphics Hardware. *Computer Graphics Forum*. 2007. Vol. 26, No. 1. P. 80–113. DOI: <https://doi.org/10.1111/j.1467-8659.2007.01012.x>
- [20] Photopea. *Photopea online photo editor*. URL: <https://www.photopea.com/>
- [21] Pixlr. *Pixlr E (advanced photo editor)*. URL: <https://pixlr.com/editor/>
- [22] Pratt W. K. *Digital image processing: PIKS scientific inside*. 4th ed. Wiley-Interscience, 2007.
- [23] Reinhard E., Stark M., Shirley P., Ferwerda J. A. Photographic tone reproduction for digital images. *ACM Transactions on Graphics*. 2002. DOI: <https://doi.org/10.1145/566570.566575>
- [24] Jo Y., Kim S. J. Practical Single-Image Super-Resolution Using Look-Up Table. 2021. P. 691–700. DOI: <https://doi.org/10.1109/cvpr46437.2021.00075>
- [25] The Khronos Group. *OpenGL ES 2.0 Specification*. URL: https://registry.khronos.org/OpenGL/specs/es/2.0/es_full_spec_2.0.pdf
- [26] The Khronos Group. *The OpenGL ES Shading Language (GLSL ES) Specification*. URL: https://registry.khronos.org/OpenGL/specs/es/2.0/GLSL_ES_Specification_1.00.pdf
- [27] The Khronos Group. *WebGL 2.0 Specification*. URL: <https://registry.khronos.org/webgl/specs/latest/2.0/>
- [28] Yang C., Jin M., Xu Y., Zhang R., Chen Y., Liu H. SepLUT: Separable Image-adaptive Lookup Tables for Real-time Image Enhancement. *ArXiv*. 2022. DOI: <https://doi.org/10.48550/arxiv.2207.08351>
- [29] Yao Z., Ma Z., Liu Y., Sani A. A., Chandramowliswaran A. Sugar. 2018. P. 519–534. DOI: <https://doi.org/10.1145/3173162.3173186>
- [30] Zhang Y., Xu W., Lyu C. Semantic-aware enhancement: Integrating semantic compensation with 3D-LUT for low-light image enhancement. *Engineering Applications of Artificial Intelligence*. 2024. Vol. 130. P. 107793. DOI: <https://doi.org/10.1016/j.engappai.2023.107793>

© Кордюк Р. І., Овсяк В. К.

Дата першого надходження статті до видання: 17.02.2026

Дата прийняття статті до друку після рецензування: 12.03.2026

Дата публікації (оприлюднення) статті: 22.05.2026



Стаття поширюється на умовах ліцензії відкритого доступу (CC BY 4.0)