Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра програмного забезпечення

Бакалаврська кваліфікаційна робота

на тему: «Розробка бібліотеки візуальних ефектів для використання у графічних та ігрових застосунках»

Виконав: студент 4 курсу
групи 5ПІ-21б спеціальності
121 – Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)
Гаврилюк Вадим Павлович
(прізвище та ініціали)
Керівник: к.т.н., доц. каф. ПЗ Ткаченко О. М.
(прізвище та ініціали)
Рецензент: к.т.н., доц. каф. ОТ Кадук О. В.
(прізвище та ініціали)

Допущено до захисту
Завідувач кафедри ПЗ
д.т.н., проф. Романюк О.Н.
«09» 06 2025 р.

ВНТУ – 2025

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення
Рівень вищої освіти перший бакалаврський
Галузь знань 12 – Інформаційні технології
Спеціальність 121 – Інженерія програмного забезпечення
Освітньо-професійна програма – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
Романюк О.Н.
« 21 » березня 2025 р.

З А В Д А Н Н Я
НА БАКАЛАВРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Гаврилюку Вадиму Павловичу

1. Тема роботи – «Розробка бібліотеки візуальних ефектів для використання у графічних та ігрових застосунках»

Керівник роботи: Ткаченко Олександр Миколайович, к.т.н., доц. кафедри ПЗ, затверджені наказом вищого навчального закладу від 20 березня 2025 р. №97.

2. Строк подання студентом роботи 2 червня 2025 р.

3. Вихідні дані до роботи: тип програми – десктопний застосунок; модель розробки – інкрементна; засоби організації даних програми – бібліотеки NumPy, Pygame, PyOpenCL; вхідні дані: базові графічні ресурси, параметри налаштування ефектів; вихідні дані: візуальні ефекти, анімації; середовище розробки – PyCharm; мова програмування – Python.

4. Зміст розрахунково-пояснювальної записки: вступ; аналіз стану технологій візуальних ефектів для ігрових застосунків; розробка методів та алгоритмів програмного застосунку; розробка застосунку; тестування застосунку; висновки; список використаних джерел; додатки.

5. Перелік графічного матеріалу: титульний слайд; актуальність теми;

мета, об'єкт та предмет дослідження; задачі дослідження, наукова новизна, практична цінність одержаних результатів; порівняльний аналіз аналогів, ….

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада Консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-4	Ткаченко О.М., к.т.н., доцент кафедри ПЗ	24.03.25	01.06.25

7. Дата видачі завдання 24 березня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз стану технологій візуальних ефектів для ігрових застосунків	25.03.25 - 05.04.25	вик
2	Розробка методів та алгоритмів програмного застосунку	06.04.25 - 18.04.25	вик
3	Варіантний аналіз та вибір мови програмування розробки	19.04.25 - 21.04.25	вик
4	Розробка застосунку	22.04.25 - 17.05.25	вик
5	Тестування застосунку	18.05.25 - 25.05.25	вик
6	Оформлення матеріалів до захисту БКР	26.05.25 - 30.05.25	вик

Студент _____ Гаврилюк В.П.
(підпис) (прізвище та ініціали)

Керівник бакалаврської кваліфікаційної роботи _____ Ткаченко О.М.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

УДК 004:005.9

Гаврилюк В.П. Розробка бібліотеки візуальних ефектів для використання у графічних та ігрових застосунках : бакалаврська кваліфікаційна робота зі спеціальності 121 Інженерія програмного забезпечення, освітня програма – інженерія програмного забезпечення. Вінниця : ВНТУ, 2025. 90 с.

На укр. мові. Бібліогр. : 28 назв.; рис.: 36; табл.: 3.

У бакалаврській кваліфікаційній роботі проведено детальний аналіз поточного стану технологій у сфері візуальних ефектів. Розглянуто сучасні аналоги програмних засобів, визначено їх переваги та недоліки, а також здійснено порівняння з власною розробкою. На основі результатів порівняння сформульовано основні задачі, вирішення яких реалізовано в межах проєкту.

У ході роботи було розроблено алгоритми програми. Також описано вхідні та вихідні дані системи, а також метод накладання динамічних ефектів на статичне зображення, метод багатопроцесорного експорту анімацій з адаптивним розподілом ресурсів.

Програмне забезпечення розроблено із використанням середовища розробки PyCharm та мови програмування Python. У процесі розробки використовувались сучасні бібліотеки та інструменти, такі як NumPy, Pygame, PyOpenCL. Проведено тестування функціональних та графічних компонентів програмної бібліотеки, розробленої в рамках бакалаврської кваліфікаційної роботи. Також надано документацію та інструкції для розробників щодо використання бібліотеки візуальних ефектів у ігрових застосунках.

У результаті виконання роботи розроблено застосунок, що дозволяє створювати візуальні ефекти для ігрових та графічних застосунків та застосовувати існуючі.

Ключові слова: паралелізація, ігрові застосунки, візуальні ефекти, динамічні ефекти.

ABSTRACT

UDC 004:005.9

Havrylyuk V.P. Development of a library of visual effects for use in graphic and game applications: bachelor's qualification work in specialty 121 Software Engineering, educational program - Software Engineering. Vinnytsia: VNTU, 2025. 90 p.

In Ukrainian. Bibliography: 28 titles; fig.: 36; tab.: 3.

The bachelor's qualification work provides a detailed analysis of the current state of technologies in the field of visual effects. Modern analogues of software tools are considered, their advantages and disadvantages are identified, and a comparison with the company's own development is also made. Based on the results of the comparison, the main tasks are formulated, the solution of which is implemented within the project.

In the course of the work, the program algorithms were developed. The input and output data of the system are also described, as well as the method of applying dynamic effects to a static image, the method of multiprocessor export of animations with adaptive resource allocation.

The software was developed using the PyCharm development environment and the Python programming language. Modern libraries and tools such as NumPy, Pygame, PyOpenCL were used in the development process. Functional and graphic components of the software library developed as part of the bachelor's qualification work were tested. Documentation and instructions for developers on using the visual effects library in game applications are also provided.

As a result of the work, an application was developed that allows you to create visual effects for game and graphic applications and apply existing ones.

Keywords: parallelization, game applications, visual effects, dynamic effects.

ЗМІСТ

ВСТУП

Обґрунтування вибору теми дослідження. Сучасний етап розвитку ігрової індустрії вимагає постійного вдосконалення графічних технологій, зокрема в області візуальних ефектів. Однією з важливих задач у розробці візуальних ефектів для ігрових застосунків є досягнення оптимального балансу між візуальною якістю та продуктивністю [1].

Традиційні методи створення візуальних ефектів часто обмежені заздалегідь визначеними шаблонами та недостатньою гнучкістю налаштувань [2]. У цьому контексті, використання модульних бібліотек, здатних генерувати різноманітні ефекти з можливістю тонкого налаштування параметрів, відкриває нові перспективи для підвищення якості графічного представлення, що має значущий потенціал для розвитку як в ігровій індустрії, так і в суміжних областях.

Сучасні візуальні ефекти є найреалістичнішими, відображають фізичні процеси, точно передають світлові та колірні особливості об'єктів, підлягають модифікації для зміни візуального представлення. Бібліотека візуальних ефектів є багатофункціональним інструментом для розробників, дозволяє істотно знизити необхідний обсяг ручного кодування порівняно з існуючими методами [3].

Важливість підвищення продуктивності визначається суттєвим внеском у підвищення плавності роботи ігрових застосунків на різних апаратних конфігураціях. У ігровій індустрії якісні візуальні ефекти відіграють важливу роль у створенні атмосфери та емоційного впливу на гравців. Наприклад, ефективне застосування ефектів частинок та світла може бути критичним для переконливого відтворення природних явищ або магічних здібностей персонажів [4].

Крім того, більшість комерційних бібліотек візуальних ефектів орієнтовані на великі студії розробки з відповідними бюджетами, залишаючи поза увагою потреби незалежних розробників та малих команд. Ліцензійні обмеження та високі витрати на підтримку таких застосунків створюють додаткові бар'єри для входу в ігрову індустрію. Водночас, відкриті рішення часто характеризуються недостатньою документацією, обмеженою підтримкою спільноти та відсутністю регулярних

оновлень, що ускладнює їх інтеграцію в сучасні ігрові проєкти.

Власна розробка дозволяє створити спеціалізоване рішення, адаптоване під конкретні вимоги цільової аудиторії, з урахуванням сучасних стандартів розробки та можливостей підвищення продуктивності. Такий підхід забезпечує повний контроль над архітектурою системи, можливість швидкого реагування на зміни технологічних вимог та створення унікальних функціональних можливостей, які відрізняють продукт від існуючих аналогів. Власна бібліотека може бути оптимізована під специфічні сценарії використання, що дозволяє досягти кращого співвідношення якості візуалізації та продуктивності порівняно з універсальними рішеннями. Тому актуальною є власна розробка модульної бібліотеки візуальних ефектів.

Зв'язок роботи з науковими програмами, планами, темами. Робота виконувалася згідно плану виконання наукових досліджень на кафедрі програмного забезпечення.

Мета та завдання дослідження. Метою роботи є підвищення продуктивності обчислень за рахунок раціонального розподілу задач між доступними процесорними ядрами, а також реалістичності за рахунок динамічного відтворення змін на зображенні.

Основними задачами дослідження є:

- розробка алгоритмів генерації візуальних ефектів, які забезпечать високу якість та оптимальну продуктивність на різних апаратних конфігураціях;

- розробка модуля системи частинок, який буде спроможний генерувати, відтворювати та керувати різноманітними ефектами на основі частинок;

- розробка модуля графічного редактора, який буде зручним у використанні для розробників ігор, забезпечуючи зручність і швидкість налаштування параметрів ефектів;

- інтеграція різних модулів в єдину бібліотеку, яка буде використовуватись для створення візуальних ефектів у ігрових застосунках.

Об'єктом дослідження є процес створення та відтворення візуальних ефектів у ігрових застосунках.

Предметом дослідження є методи та засоби розробки оптимізованих візуальних ефектів для ігрових застосунків.

Методи дослідження. У процесі досліджень використовувались: методи комп'ютерної графіки для відтворення візуальних ефектів; методи математичної статистики для оцінювання ступеню підвищення продуктивності; методи об'єктно-орієнтованого програмування для розробки модулів бібліотеки; методи комп'ютерного моделювання для аналізу й експериментальної перевірки теоретичних положень.

Наукова новизна отриманих результатів.

1. Подальшого розвитку отримав метод накладання динамічних ефектів на статичне зображення, який, на відміну від відомих, використовує інтелектуальний аналіз вмісту зображення для автоматичного визначення глибини та перспективи, що дозволяє інтегрувати частинки та візуальні елементи з урахуванням структури сцени без необхідності ручного маскування або налаштування параметрів для кожного типу зображень.

2. Подальшого розвитку отримав метод багатопроцесорного експорту анімацій з адаптивним розподілом ресурсів, який, на відміну від відомих, впроваджує динамічне балансування навантаження з аналізом складності кожного кадру та інтелектуальним розподілом задач між доступними процесорними ядрами, що забезпечує оптимальне використання обчислювальних ресурсів системи.

Практична цінність отриманих результатів. Практична цінність розробки полягає у створенні алгоритмічного та програмного забезпечення, яке дозволяє отримати суттєве прискорення рендеренгу анімації шляхом ефективного використання всіх доступних процесорних ядер, навіть у системах з нерівномірним навантаженням.

Особистий внесок здобувача. Усі наукові результати, викладені у бакалаврській кваліфікаційній роботі, отримані автором особисто. У роботах, опублікованих у співаторстві, автору належать: розробка бібліотеки візуальних ефектів для графічних та ігрових застосунків [5], розробка методу багатопроцесорного експорту анімацій з адаптивним розподілом ресурсів [6],

розробка методу накладання динамічних ефектів на статичне зображення з інтелектуальним аналізом сцени [7].

Апробація результатів роботи. Описані у бакалаврській кваліфікаційній роботі положення доповідались на конференції Молодь в науці: дослідження, проблеми, перспективи (МН-2025) .

Публікації. Результати роботи опубліковано в матеріалах конференції Молодь в науці: дослідження, проблеми, перспективи (МН-2025) Вінницького національного технічного університету.

Аналіз. Пояснювальна записка до бакалаврської кваліфікаційної роботи складається з 4 розділів та містить 28 джерел посилання.

1 АНАЛІЗ РОЗВИТКУ ТЕХНОЛОГІЙ ВІЗУАЛЬНИХ ЕФЕКТІВ У ІГРОВИХ ЗАСТОСУНКАХ ТА ПОСТАНОВКА ЗАДАЧ ДОСЛІДЖЕННЯ

1.1 Аналіз стану технологій візуальних ефектів для ігрових застосунків

Сучасний етап розвитку ігрової індустрії вимагає постійного вдосконалення методів візуалізації та графічних ефектів, зокрема в області спеціальних візуальних ефектів. Традиційні методи створення візуальних ефектів, хоча і залишаються важливими, часто обмежені жорсткими рамками конкретних ігрових рушіїв та недостатньою гнучкістю налаштувань. У цьому контексті, використання модульних бібліотек, здатних генерувати різноманітні ефекти з можливістю тонкого налаштування параметрів, стає дедалі більш поширеним.

Одним із ключових аспектів розвитку цих технологій є зростання потужності графічних процесорів та еволюція графічних API. Сучасні GPU та технології рендерингу дозволяють швидко та ефективно обробляти складні візуальні ефекти в реальному часі, що є необхідним для створення переконливих візуальних ефектів у ігрових застосунках.

Модульні бібліотеки візуальних ефектів надають значні переваги порівняно з традиційними підходами до розробки графіки.

Головною перевагою використання модульних бібліотек для створення візуальних ефектів є те, що вони забезпечують високий рівень гнучкості та ревикористання коду. Це дозволяє розробникам швидко створювати та налаштовувати різноманітні ефекти без необхідності написання великої кількості унікального коду для кожного ефекту.

Також, сучасні бібліотеки візуальних ефектів можуть включати різні типи ефектів, такі як системи частинок, постобробка зображення, світлові ефекти, деформації об'єктів тощо. Це дозволяє отримувати більш комплексне візуальне представлення та забезпечує можливість використання різноманітних технік для досягнення бажаного результату.

Ще однією перевагою використання модульних бібліотек у області візуальних ефектів є оптимізація продуктивності. В ігровій індустрії баланс між візуальною

якістю та продуктивністю відіграє важливу роль у створенні успішних продуктів. Зокрема, ефективне управління ресурсами, такими як пам'ять GPU та процесорний час, може бути критичним для забезпечення плавної роботи гри на різних апаратних конфігураціях.

Отже, аналіз стану технологій візуальних ефектів для ігрових застосунків підтверджує важливість переходу до більш гнучких та оптимізованих методів створення візуальних ефектів у ігровій індустрії та суміжних сферах. Розробка програмних модулів для впровадження бібліотеки візуальних ефектів стає необхідністю для покращення якості графічного представлення та підвищення продуктивності. Звернення до досвіду та ресурсів спеціалізованих компаній у розробці графічних програмних засобів сприятиме вирішенню складних завдань і дозволить створювати інноваційні рішення, які підвищують користувацький досвід та забезпечують конкурентні переваги. Враховуючи постійний розвиток технологій у цій області, важливо бути в курсі останніх досягнень у сфері комп'ютерної графіки та тенденцій для успішної адаптації та забезпечення актуальності ігрових продуктів та їх візуальної складової.

1.2 Порівняльний аналіз аналогів

Використання бібліотек візуальних ефектів стає все більш поширеним в останні роки, і ці технології інтегруються в різні ігрові рушії та системи, включаючи як великі AAA-проєкти, так і інді-розробки. Розробка програмних модулів бібліотеки візуальних ефектів для ігрових застосунків забезпечить гнучкість, оптимізацію та різноманітність візуального представлення та допоможе швидше створювати переконливий ігровий досвід. До найбільш відомих застосунків у цій галузі відносять:

- Unity VFX Graph;
- Unreal Engine Niagara;
- PopcornFX;
- Notch Builder;
- Shadertoy.

Одним із прикладів потужної бібліотеки візуальних ефектів є Unity VFX Graph (рис. 1.1) – інструмент для створення та налаштування візуальних ефектів, розроблений Unity Technologies. Він дозволяє створювати складні системи частинок та візуальні ефекти за допомогою нодового редактора, що спрощує процес створення та налаштування ефектів.

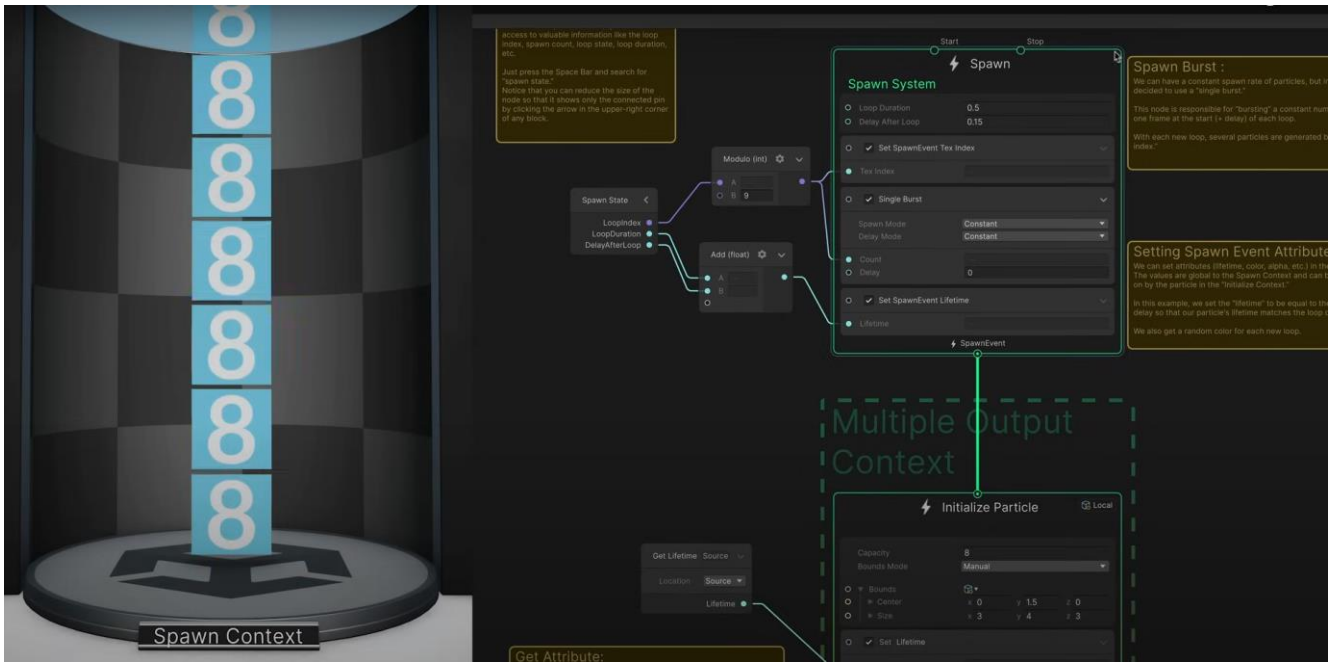


Рисунок 1.1 – Приклад нодового редактору ефектів Unity VFX Graph

Інший приклад – Unreal Engine Niagara (рис. 1.2), є потужною системою для створення візуальних ефектів, інтегрованою в Unreal Engine [8]. Вона надає користувачам можливість створювати деталізовані та інтерактивні ефекти з урахуванням різних параметрів, таких як фізика частинок, взаємодія зі світлом і оточенням. Цей інструмент часто використовується у великих ігрових проєктах, фільмах та віртуальній реальності.

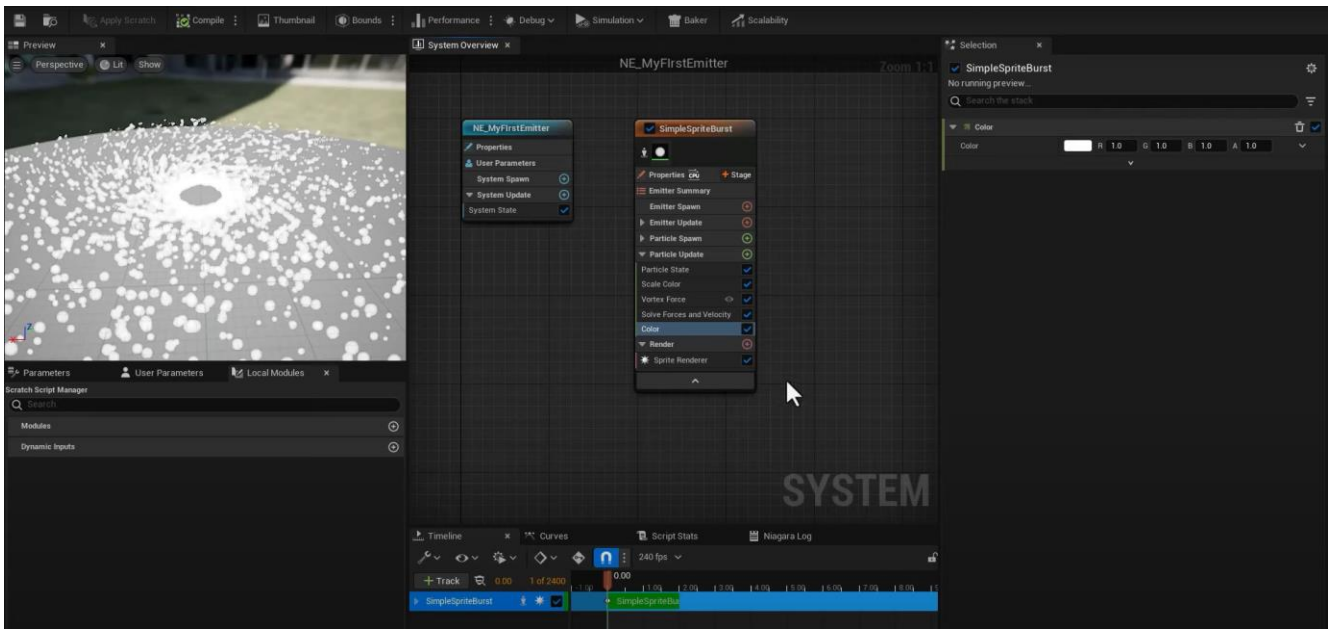


Рисунок 1.2 – Приклад створення ефекту за допомогою системи ефектів Niagara в Unreal Engine

PopcornFX (рис. 1.3) є іншою бібліотекою, спеціально розробленою для створення високопродуктивних візуальних ефектів для ігор [9]. Цей інструмент використовується для генерації та підвищення продуктивності складних систем частинок, забезпечуючи високу продуктивність і якість візуального представлення. Варто зазначити, що дане програмне забезпечення підтримує інтеграцію з різними ігровими рушіями. PopcornFX має розвинену вузлову систему редагування, яка дозволяє користувачеві створювати складні ефекти без необхідності глибокого занурення в програмування. Кожен ефект може включати емітери, модифікатори, деформації, фізичні взаємодії та кастомні матеріали, що дає змогу досягти високого рівня деталізації та художньої гнучкості. Важливо, що ефекти можна попередньо переглядати в режимі реального часу, що пришвидшує процес ітеративної розробки та налагодження.

Окрім потужного редактора, PopcornFX надає розробникам можливість розширювати функціональність за допомогою скриптів, API та SDK для інтеграції в індивідуальні ігрові рушії. Програма активно застосовується в індустрії розробки ігор для створення візуальних ефектів, таких як вибухи, магія, погодні явища,

динамічний дим, вогонь тощо. Завдяки своїй продуктивності та масштабованості PopcornFX вважається одним із найефективніших інструментів для роботи з візуальними ефектами у великих проєктах.

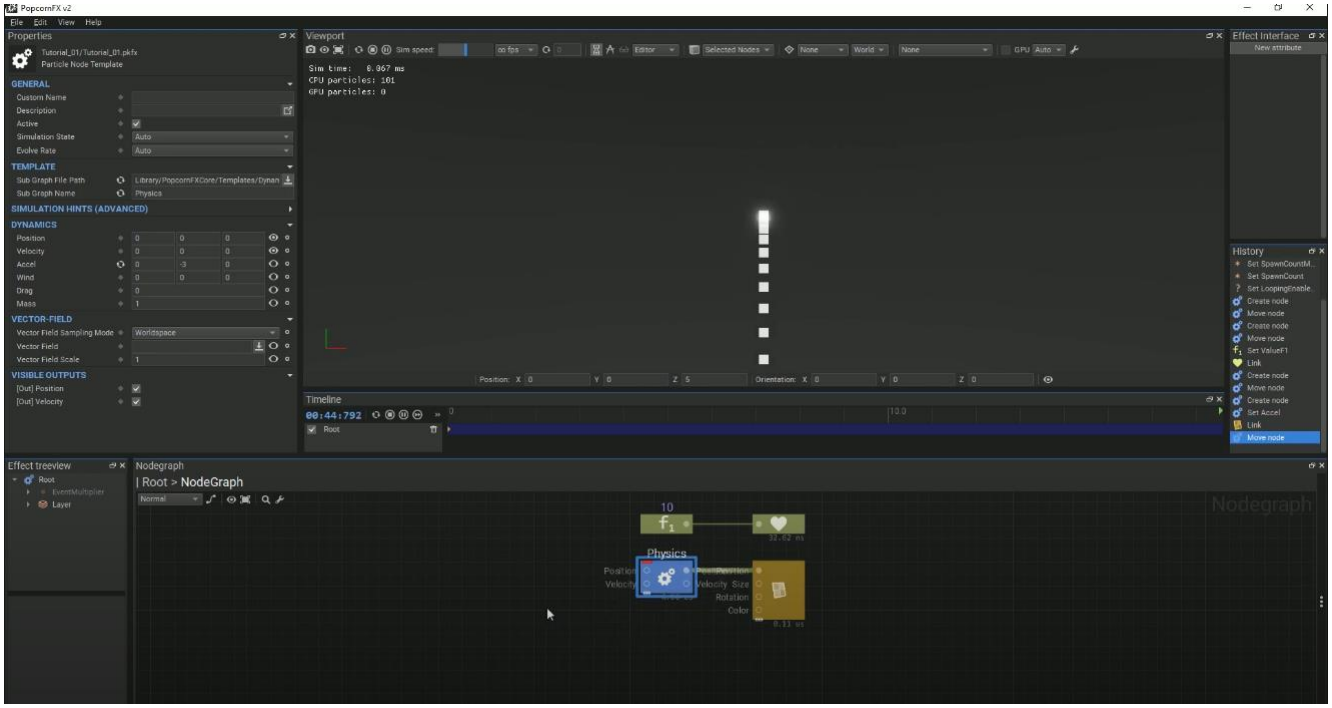


Рисунок 1.3 – Приклад роботи застосунку PopcornFX

Notch Builder (рис. 1.4) є програмним засобом, призначеним для створення інтерактивних візуальних ефектів для різних медіа, включаючи ігри та інтерактивні інсталяції [10]. Цей інструмент дозволяє розробникам створювати реалістичні та складні візуальні ефекти, які можуть бути використані для створення атмосфери та візуального стилю в ігрових застосунках.

Notch Builder вирізняється з-поміж інших застосунків завдяки поєднанню можливостей реального часу та інтерактивного управління параметрами ефектів, що дозволяє миттєво бачити зміни без повторного рендерингу сцени. Серед ключових функцій — підтримка анімації, генерація систем частинок, робота з відео, 3D-об’єктами та інтеграція з MIDI-контролерами для живої взаємодії. Завдяки цьому Notch активно використовується не лише в ігровій індустрії, а й у сфері сценографії, VJ-інсталяцій та візуального супроводу живих подій.

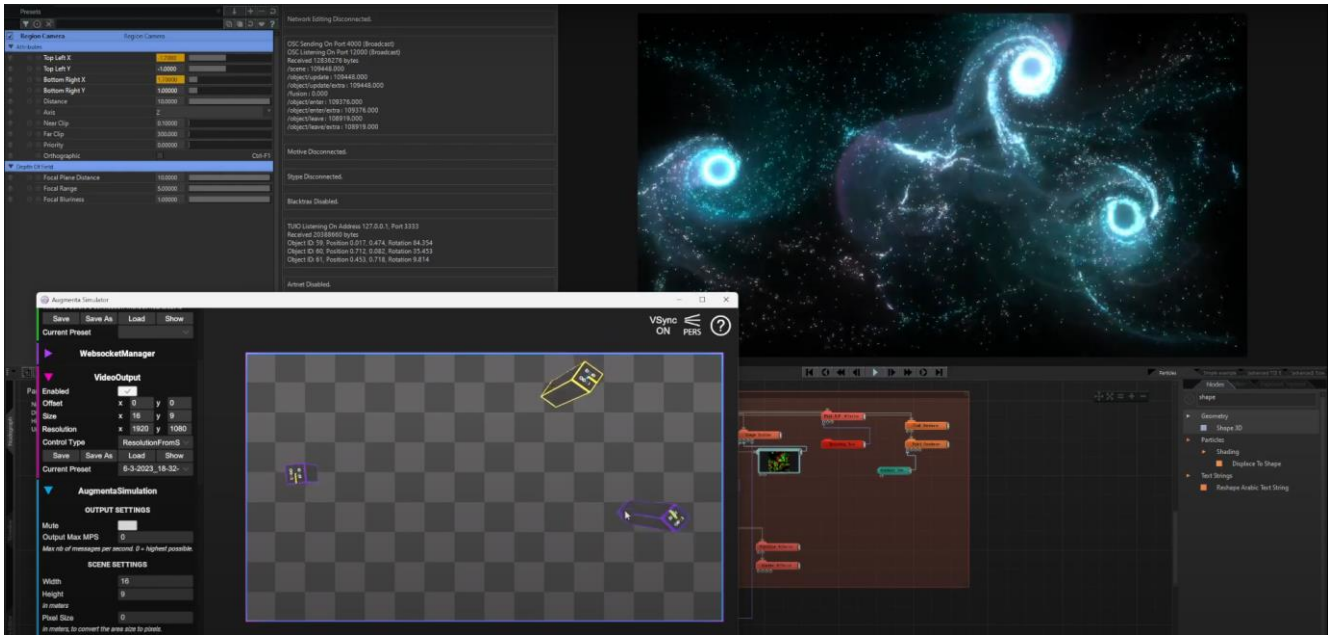


Рисунок 1.4 – Приклад роботи застосунку Notch Builder

Shadertoy (рис. 1.5) є онлайн-платформою для створення та обміну шейдерами та візуальними ефектами на основі GLSL [11]. Цей інструмент використовується для експериментування з різними техніками візуалізації та створення прототипів ефектів, що можуть бути пізніше інтегровані в ігрові проєкти.

Shadertoy забезпечує розробникам зручне середовище для роботи безпосередньо у веббраузері, дозволяючи створювати фрагментні шейдери в реальному часі з миттєвим візуальним зворотним зв’язком. Платформа підтримує інтерактивність через введення даних користувача, звукові реакції, відеопотоки та інші джерела, що робить її особливо корисною для тестування складних графічних ефектів. Крім того, Shadertoy виконує важливу роль у навчанні та поширенні знань у сфері графічного програмування завдяки великій спільноті та відкритому доступу до тисяч прикладів шейдерів.

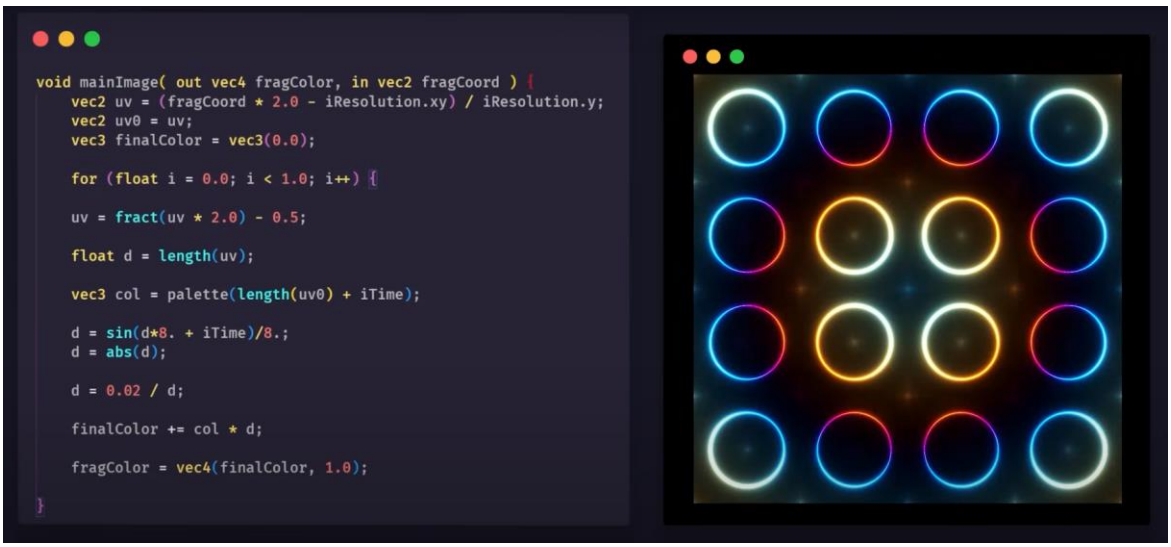


Рисунок 1.5 – Приклад коду та ефект у Shadertoy

Розробка програмних модулів бібліотеки візуальних ефектів для ігрових застосунків вимагає досвіду роботи з графічними API, шейдерами, системами частинок та підвищенням продуктивності [12]. Для цього необхідне створення модульної системи, яка буде здатна генерувати різноманітні візуальні ефекти, а також забезпечувати їх ефективне відтворення в реальному часі. Складова інтерфейсу користувача включає розробку зручного редактора, який дозволить розробникам ігор налаштовувати та комбінувати різні ефекти.

У рамках розробки важливо передбачити підтримку масштабованої архітектури, що дозволяє легко додавати нові типи ефектів без суттєвих змін у вже реалізованих модулях. Такий підхід забезпечує гнучкість під час розробки ігрових сцен з урахуванням змін вимог до візуалізації. Кожен модуль бібліотеки має бути відповідальним за окремий тип ефекту, зокрема: частинки, постобробка, об’ємні ефекти, світлові промені, що полегшує підтримку та масштабування системи. Крім того, використання уніфікованих шейдерів дозволяє забезпечити однакову якість рендерингу на різних апаратних платформах.

Редактор ефектів відіграє ключову роль у взаємодії з бібліотекою, оскільки дозволяє візуально налаштовувати параметри ефектів у режимі реального часу. Це

прискорює процес ітераційної розробки, дозволяючи миттєво оцінювати зміни без перекомпіляції коду. Такий підхід є особливо ефективним у прототипуванні ігрових сцен, де художники та дизайнери можуть самостійно працювати з візуальними ефектами без залучення програмістів. Редактор також може бути оснащений функцією експорту у власний формат (наприклад, .vfx), що дозволяє інтегрувати налаштовані ефекти у зовнішні рушії або проєкти.

Результати порівняння аналогів зведено в табл. 1.1.

Таблиця 1.1 – Порівняльна характеристика аналогів

Критерії	Unity VFX Graph	Unreal Engine Niagara	PopcornFX	Notch Builder	Shadertoy	Власна бібліотека
Підвищення продуктивності	+	+	+	-	-	+
Крос-платформна сумісність	-	-	+	-	+	+
Модульна архітектура	+	+	+	-	-	+
Простота інтеграції	-	-	+	-	+	+
Гнучкість налаштувань	+	+	-	+	+	+
Загальна оцінка	60%	60%	80%	40%	60%	100%

Відповідно до таблиці порівняльних характеристик розробка власних програмних модулів бібліотеки візуальних ефектів для ігрових застосунків є доцільною. Отриманий продукт зможе покрити недоліки існуючих застосунків та

забезпечити розширені можливості для створення та налаштування візуальних ефектів у ігрових застосунках.

Отже, розробка програмних модулів бібліотеки візуальних ефектів відкриває перед розробниками ігор широкі можливості для покращення візуальної складової та підвищення продуктивності. Завдяки гнучким та оптимізованим модулям, розробники можуть створювати більш переконливі та атмосферні візуальні ефекти, що є критичним для успішності ігрових проєктів на сучасному конкурентному ринку.

1.3 Аналіз методів розв’язання задачі

Розробка програмних модулів бібліотеки візуальних ефектів для ігрових застосунків вимагає комплексного підходу до вирішення завдань, пов'язаних з генерацією, відтворенням та оптимізацією різноманітних візуальних ефектів[11]. Існують різні методи, кожен з яких має свої переваги та недоліки. У цьому розділі проаналізовано найбільш ефективні методи розробки таких програмних модулів.

Один з основних аспектів розробки бібліотеки візуальних ефектів – це створення системи частинок, яка є основою багатьох візуальних ефектів. Для цього можна використовувати CPU-based підхід, який передбачає обчислення поведінки частинок на центральному процесорі. Однак такий підхід має певні недоліки. Він створює значне навантаження на CPU та має обмежену масштабованість при роботі з великою кількістю частинок. Тому такий підхід не варто застосовувати як основний.

Інший підхід полягає у використанні GPU-based системи частинок [13], яка базується на використанні обчислювальних шейдерів (compute shaders) для паралельної обробки даних частинок. Цей метод дозволяє обробляти більшу кількість частинок одночасно, розподіляючи обчислювальне навантаження на графічний процесор. Застосування GPU-based системи дозволяє отримати високу продуктивність та можливість створення складних ефектів у реальному часі. Однак цей метод має обмеження щодо сумісності з деякими мобільними платформами та старішими відеокартами, які не підтримують обчислювальні шейдери. Тому

використання цього методу буде ефективним лише за умови наявності альтернативної реалізації для платформ з обмеженими можливостями.

Найефективнішим підходом є використання гібридної системи, яка поєднує переваги CPU та GPU підходів [14] та додатково використовує інстансинг (instancing) для оптимізації відтворення великої кількості подібних об'єктів. Такий підхід дозволяє динамічно адаптувати продуктивність системи залежно від доступних ресурсів та цільової платформи. Додатково, використання процедурної генерації текстур та шейдерних ефектів є важливим елементом у розробці бібліотеки візуальних ефектів. Цей процес дозволяє скоротити обсяг використовуваної пам'яті та забезпечує високу гнучкість у налаштуванні ефектів. Такий підхід гарантує оптимальну продуктивність та візуальну якість ефектів [15]. Однак, він має недолік – підвищену складність розробки та необхідність глибокого розуміння як CPU, так і GPU оптимізацій.

Розглянувши переваги та недоліки кожного методу, було вирішено використовувати гібридний підхід з модульною архітектурою та підтримкою різних рівнів деталізації (LOD), що забезпечить найкращу продуктивність та гнучкість у розробці бібліотеки візуальних ефектів. Більше того, такий метод розробки дозволить створити універсальну бібліотеку з унікальним функціоналом, яка може бути інтегрована в різні ігрові рушії та адаптована до різних апаратних конфігурацій.

1.4 Постановка задач дослідження

Проаналізувавши переваги та недоліки існуючих бібліотек візуальних ефектів для ігрових застосунків, було визначено наступні завдання, які необхідно виконати для успішної розробки програмних модулів:

- розробити архітектуру модульної бібліотеки візуальних ефектів, яка забезпечить гнучкість та можливість розширення функціоналу;
- розробити алгоритми генерації та відтворення різних типів візуальних ефектів (системи частинок, світлові ефекти, деформації об'єктів, постобробка зображення);

- розробити алгоритми підвищення продуктивності, включаючи систему рівнів деталізації (LOD), кулінг (culling) та інші техніки оптимізації;
- розробити модуль графічного редактора, який буде зручним у використанні для розробників ігор, забезпечуючи інтуїтивний інтерфейс для налаштування параметрів ефектів;
- реалізувати механізми інтеграції з популярними ігровими рушіями (Unity, Unreal Engine);
- забезпечити підтримку різних графічних API (DirectX, OpenGL, Vulkan) для забезпечення крос-платформності;
- провести тестування бібліотеки на різних апаратних конфігураціях та в різних ігрових сценаріях.

1.5 Висновки

У першому розділі було розглянуто стан технологій візуальних ефектів для ігрових застосунків. Були розглянуті бібліотеки та інструменти такі як Unity VFX Graph, Unreal Engine Niagara, PopcornFX, Notch Builder та Shadertoy.

Проаналізувавши переваги та недоліки існуючих бібліотек, було сформульовано завдання для подальшої розробки власної бібліотеки візуальних ефектів. Ці завдання включають розробку архітектури модульної бібліотеки, алгоритмів генерації та відтворення різних типів ефектів, механізмів підвищення продуктивності та розробку графічного редактора для налаштування параметрів. Таким чином було доведено доцільність розробки власного програмного рішення. На основі отриманої інформації було сформовано перелік задач, які необхідно виконати для розробки власних програмних модулів.

2 РОЗРОБКА МЕТОДІВ ТА АЛГОРИТМІВ СИСТЕМИ

2.1 Аналіз вхідних та вихідних даних системи

Основними вхідними даними для бібліотеки візуальних ефектів є статичні зображення, які завантажуються користувачем для подальшої обробки та накладання ефектів. Система підтримує широкий спектр форматів зображень, включаючи PNG, JPEG, JPG та інші поширені формати. Завантаження зображень реалізовано через метод load_image() класу PixelFX, який може приймати шлях до файлу зображення як аргумент.

Важливими вхідними даними також є параметри ефектів, які визначають їхню поведінку і зовнішній вигляд. Ці параметри структуровані у вигляді словників, де кожен параметр має свій тип (слайдер, колір, прапорець, випадаючий список) та значення. Наприклад, для ефекту "магічний портал" вхідними параметрами є розмір, яскравість, швидкість обертання, кольорова схема, прозорість та інші [16].

Система також приймає взаємодію користувача з інтерфейсом як вхідні дані через події миші та клавіатури. Ці дані обробляються функціями handle_event() різних класів, зокрема Button та Interface. Взаємодія користувача може змінювати стан елементів інтерфейсу, активувати функції і змінювати параметри ефектів у реальному часі.

Для ефектів, що вимагають інтерактивності, таких як InteractiveEffect, важливими вхідними даними є координати курсора миші, які впливають на поведінку частинок та інших елементів ефекту. Ці дані постійно оновлюються під час руху миші та обробляються у функції update() відповідного ефекту.

При експорті анімації система приймає додаткові вхідні дані: формат експорту, тривалість анімації, частоту кадрів, якість та інші параметри, які впливають на кінцевий результат.

У процесі обробки вхідного зображення система створює буферну поверхню buffer_surface для рендерингу ефектів, не змінюючи оригінальне зображення. Це дозволяє зберегти оригінал і працювати з копією для накладання різних ефектів.

Для генерації ефектів система використовує різні алгоритми візуалізації.

Наприклад, ефект дощу (RainEffect) обробляє масив "крапель", оновлюючи їхні позиції та відображаючи їх на буферній поверхні. Ефект снігу (SnowEffect) імітує рух сніжинок з урахуванням їх фізичних властивостей, таких як швидкість падіння та коливання.

Обробка даних відбувається в кілька етапів. Спочатку у функції update() класу PixelFX оновлюється стан кожного ефекту на основі минулого часу. Потім у функції render() ефекти послідовно застосовуються до буферної поверхні. Система також включає механізм збору статистики продуктивності, яка відображається користувачу.

Для систем з підтримкою графічного процесора, процес обробки даних може використовувати графічний процесор для прискорення обчислень. Система перевіряє доступність графічного процесора через функцію check_gpu() і, якщо це можливо, використовує GPU-прискорення через функції, такі як process_with_pytorch(), process_with_opencl() або process_with_opencv().

При експорті анімації система генерує набір кадрів на основі часової прогресії ефектів.

Основними вихідними даними системи є візуалізація ефектів у реальному часі на екрані користувача. Це реалізується через відображення буферної поверхні buffer_surface на основну поверхню, яка представлена змінною screen або surface.

Для збереження результатів система генерує кілька типів вихідних даних:

1.Статичні зображення у форматі PNG через функцію save_image(), який зберігає поточний стан буферної поверхні у файл.

2.Анімовані GIF-файли через функцію export_gif(), який створює серію кадрів на основі прогресії ефектів і об'єднує їх у GIF-анімацію. Система оптимізує GIF-файли, регулюючи кількість кольорів та розмір залежно від заданих параметрів якості.

3.Відеофайли через функцію export_video(), яка дозволяє зберігати анімацію у відеоформаті з вибором кодека, бітрейту та інших параметрів.

Окрім файлів, система також генерує метадані для експортованих ефектів у форматі JSON. Ці метадані містять інформацію про назву ефекту, версію, параметри

та інші дані, необхідні для імпорту ефекту в зовнішні системи або для відтворення ефекту в майбутньому.

Додатковими вихідними даними є інформаційні повідомлення про стан системи, такі як успішний експорт, помилки або попередження. Ці повідомлення відображаються користувачу через графічний інтерфейс і допомагають йому розуміти результати операцій.

Варто відзначити, що система виконує складну трансформацію даних: від статичного зображення до динамічної анімації. Ця трансформація відбувається через накладання часозалежних ефектів, які змінюють візуальне представлення зображення в часі.

Важливою особливістю трансформації є збереження основної структури та змісту зображення при додаванні динамічних елементів. Наприклад, при застосуванні ефекту дощу, система не змінює саме зображення, а лише накладає напівпрозорі краплі, які рухаються по поверхні зображення.

Трансформація також включає адаптацію ефектів до характеристик зображення. Наприклад, система може автоматично визначати розмір і масштаб ефектів залежно від розміру вхідного зображення, забезпечуючи пропорційне відображення.

У результаті трансформації даних користувач отримує не лише візуально привабливий ефект, але й файли, які можна використовувати в інших проектах, таких як веб-сайти, презентації або ігри, розширюючи можливості статичних зображень і додаючи їм динамічності та емоційного впливу.

2.2 Розробка методу накладання динамічних ефектів на статичне зображення

Метод накладання динамічних ефектів на статичне зображення дозволяє трансформувати звичайні статичні зображення в анімовані сцени без необхідності створення покадрової анімації. Цей метод є одним із головних елементів бібліотеки візуальних ефектів.

Користувач може завантажити будь-яке зображення і застосувати до нього

різноманітні динамічні ефекти – від природних явищ (дощ, сніг, туман) до магічних ефектів (світіння, портали, електричні розряди) і спецефектів (частинки вогню, інтерактивні елементи). Ефекти при цьому органічно інтегруються із зображенням, враховуючи його особливості, глибину та освітлення.

Система забезпечує контроль над параметрами ефектів через інтерфейс з візуальним відображенням змін у реальному часі, а також можливість експорту результату в різноманітні формати.

Послідовність виконання роботи методу:

1. Завантаження зображення:

- Підтримка різноманітних форматів зображень (PNG, JPG, TIFF, тощо).
- Автоматичне масштабування зображення для оптимальної продуктивності.
- Аналіз зображення для визначення областей, придатних для ефектів.

2. Вибір та налаштування ефекту:

- Вибір категорії ефекту (частинки, освітлення, спецефекти).
- Вибір конкретного ефекту (дощ, сніг, вогонь, магічний портал, тощо).
- Налаштування специфічних параметрів через панель властивостей.

3. Інтелектуальне поєднання ефекту із зображенням:

- Автоматичне визначення фронтальних та фонових елементів зображення.
- Аналіз освітлення сцени для реалістичного накладання ефектів.
- Налаштування глибини та масштабу ефекту відповідно до перспективи зображення.

4. Візуалізація:

- Динамічний рендеринг ефекту на зображенні з оновленням в реальному часі.
- Інтерактивне управління властивостями ефекту з миттєвим відображенням змін.
- Використання багатопоточної обробки для забезпечення плавності анімації.

5. Налаштування ефекту:

- Маскування зон, де ефект не повинен відображатися.

• Налаштування взаємодії ефекту з елементами зображення.

• Коригування швидкості, інтенсивності та інших параметрів для досягнення бажаного результату.

6. Налаштування часової анімації:

• Використання часової шкали для контролю поведінки ефекту у часі.

• Налаштування циклічності та тривалості ефекту.

• Додавання ключових кадрів для зміни параметрів впродовж анімації.

7. Експорт результату:

• Вибір формату експорту (GIF, відео, PNG-послідовність, спрайт-атлас).

• Налаштування параметрів якості та підвищення продуктивності.

• Експорт з використанням багатопроцесорної обробки для пришвидшення.

Метод накладання динамічних ефектів на статичні зображення спрощує процес створення анімованого контенту. Традиційно, перетворення статичного зображення в анімацію вимагало б використання складних програм анімації, глибоких знань ротоскопії та значних часових витрат. Розроблений метод у свою чергу надає інструмент, який дозволяє будь-якому користувачеві швидко створювати анімовані ефекти.

2.3 Розробка методу багатопроцесорного експорту анімацій з адаптивним розподілом ресурсів

Метод багатопроцесорного експорту анімацій, на відміну від традиційних систем, які обробляють кадри послідовно, використовує повну потужність сучасних багатоядерних процесорів, розподіляючи обробку кадрів між кількома паралельними процесами.

Метод працює за принципом паралельних обчислень, де процес генерації кожного кадру анімації виконується як окреме завдання. Після завершення всіх завдань результати збираються та об'єднуються в остаточний анімований файл. Паралельна обробка дозволяє досягти прискорення порівняно з традиційними послідовними методами, залежно від кількості доступних процесорних ядер.

Особливістю цього методу є адаптивний розподіл ресурсів, який автоматично

визначає оптимальну кількість процесів залежно від доступних системних ресурсів та складності ефекту, що експортується. Це забезпечує максимальну продуктивність без перевантаження системи.

Послідовність виконання роботи методу:

1.Аналіз системних ресурсів:

•Визначення кількості доступних ядер процесора.

•Оцінка доступної системної пам'яті.

•Встановлення оптимальної кількості робочих процесів.

2.Підготовка даних для паралельної обробки:

•Створення копії оригінального зображення та ефектів для безпечного використання в паралельних процесах.

•Розбиття загальної кількості кадрів на окремі завдання.

•Формування аргументів для кожного процесу через функцію args_list.

3.Створення та запуск пулу процесів:

•Ініціалізація пулу процесів через конструкцію with multiprocessing.Pool() as pool.

•Розподіл завдань між процесами за допомогою методу pool.map().

•Запуск паралельного виконання функції export_frame_worker для кожного кадру.

4.Обробка кадрів у паралельних процесах:

•Виконання функції create_frame() для генерації кожного кадру.

•Застосування ефектів до базового зображення.

•Повернення готового кадру та його індексу.

5.Збір результатів та їх впорядкування:

•Сортування отриманих кадрів за індексом для збереження правильної послідовності.

•Перетворення кадрів у формат, придатний для збереження (Image.fromarray).

•Підготовка метаданих анімації (тривалість кадру, цикличність).

6.Запис анімації у файл:

•Вибір формату збереження (GIF або відео).

•Застосування оптимізації залежно від обраного формату.

•Збереження анімації з встановленими параметрами.

7.Аналіз та звітування про продуктивність:

•Вимірювання загального часу виконання та обчислення швидкості обробки (кадрів/секунду).

•Відображення прогресу та оцінки часу до завершення.

•Генерація звіту про успішність експорту.

Метод багатопроцесорного експорту анімацій з адаптивним розподілом ресурсів скорочує час очікування при експорті складних анімацій. Адаптивність методу забезпечує оптимальне використання ресурсів комп'ютера, автоматично визначаючи баланс між швидкістю обробки та навантаженням на систему. Це дозволяє ефективно працювати як на високопродуктивних робочих станціях, так і на менш потужних системах, адаптуючи процес до наявних ресурсів.

2.5 Розробка діаграм та алгоритмів роботи системи

Для кращого розуміння роботи модулів програмних засобів бібліотеки візуальних ефектів було розроблено модель роботи системи на прикладі UML діаграми діяльності (рис. 2.1).

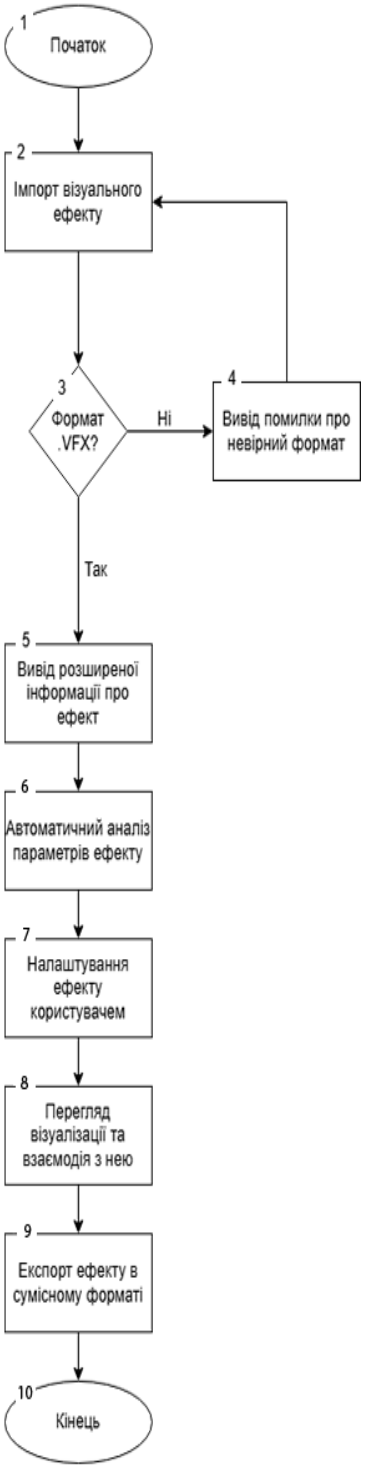


Рисунок 2.1 – Діаграма діяльності програми управління ефектами

Для початку взаємодії із програмним застосунком проводиться імпортування візуального ефекту у форматі .vfx, у разі іншого формату програма видаватиме помилку.

Наступним кроком буде перевірка структури ефекту на відповідність до

стандартів бібліотеки. Якщо структуру певних елементів ефекту не вдасться визначити (через особливості імпортованого файлу), програма повідомить про це у робочому середовищі і виведе перелік функцій, які можуть бути обмежені. У разі повної відповідності структури, програмний застосунок надає розширену інформацію про імпортований ефект.

Після успішного імпортування візуального ефекту, програмний застосунок автоматично проводить аналіз параметрів ефекту, визначає ключові точки анімації та налаштування частинок за алгоритмами бібліотеки. Далі, користувач може переходити до налаштування візуального ефекту, змінюючи параметри кольору, розміру, швидкості та інші характеристики.

Після завершення налаштування, користувач може переглянути візуалізацію ефекту у віртуальному робочому середовищі, взаємодіяти з нею (змінювати кут огляду, масштаб, відображення певних елементів) та експортувати отриманий ефект у форматі, сумісному з популярними ігровими рушіями такими як Unity, Unreal Engine або Godot.

Розробка діаграми діяльності була виконана з урахуванням специфіки роботи бібліотеки візуальних ефектів та потреб розробників ігрових застосунків.

Для розробки програмного застосунку бібліотеки візуальних ефектів для ігрових застосунків необхідно розробити загальний алгоритм, згідно якого будуть працювати програмні модулі (рис. 2.2).

Згідно даного алгоритму, першим ділом, відбувається завантаження робочого середовища програми, яке надає користувачу доступ до функціоналу розробленої бібліотеки візуальних ефектів. Далі, користувачу необхідно імпортувати візуальний ефект у форматі .vfx. Обравши файл ефекту, програмний застосунок запускає модуль перевірки імпортованого файлу на наявність дефектів. При наявності дефектів або невірному форматі програмний застосунок надасть користувачу інформацію про виявлені проблеми або виведе повідомлення про невірний формат файлу.

При відсутності дефектів програмний засіб виконає модуль розбору компонентів ефекту, для аналізу структури візуального ефекту. Після чого

виконується модуль перетворення компонентів у систему частинок, що необхідно для коректної візуалізації ефекту в ігрових рушіях. Наступним кроком буде визначення параметрів ефекту, таких як швидкість частинок, їх розмір, колір, прозорість, час життя та інші характеристики.

Після визначення всіх необхідних параметрів відбувається генерація візуального ефекту в робочому середовищі для попереднього перегляду. Останнім етапом є збереження готового ефекту для використання в популярних ігрових рушіях, таких як Unity, Unreal Engine чи Godot.

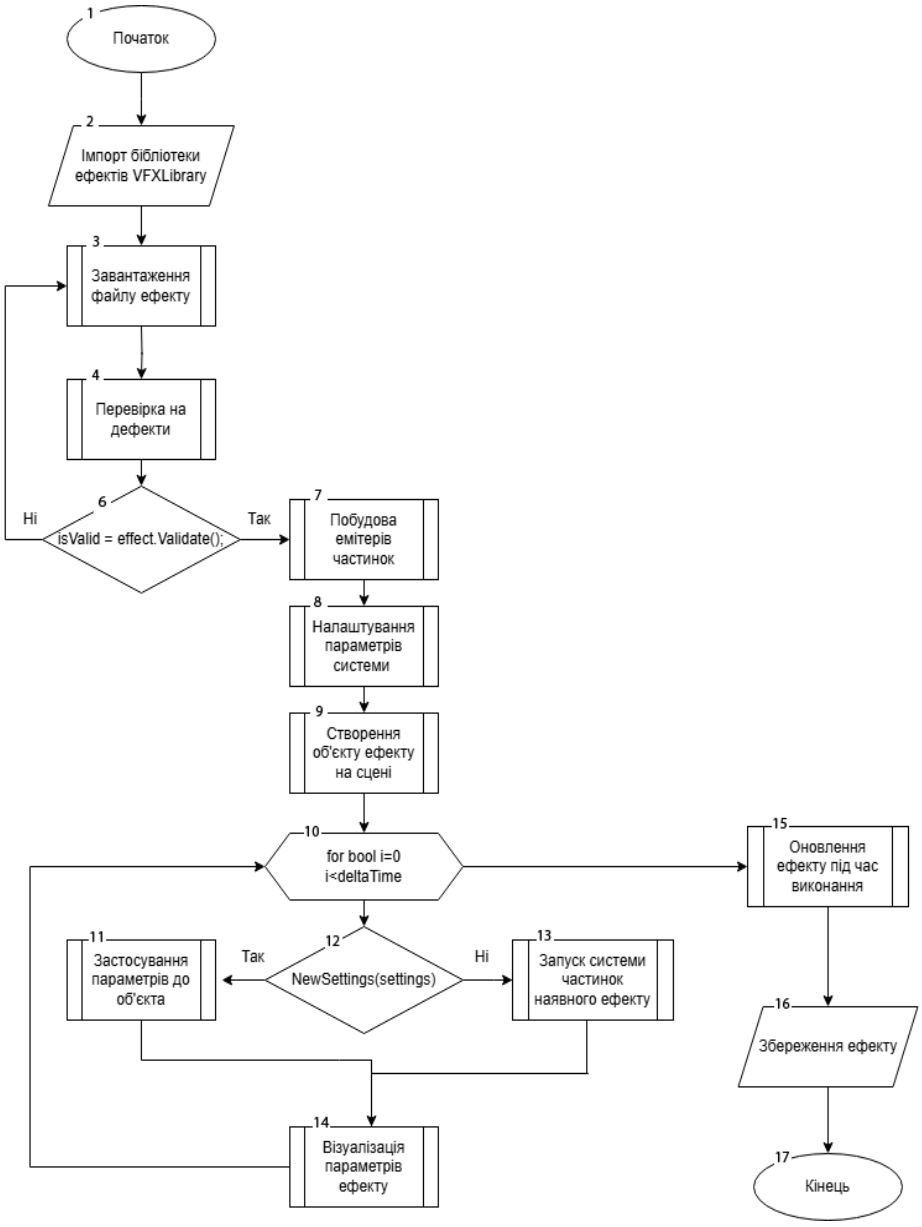


Рисунок 2.2 – Алгоритм додавання ефекту на сцену

Діаграма станів кнопки handle_event показує внутрішню машину станів об’єкта Button, що відповідає за обробку мишиних подій у класі pixelfx.ui.Button. Є три основні стани: Normal (штатний стан, коли курсор поза кнопкою), Hover (курсор знаходиться над кнопкою) і Pressed (ліва кнопка миші натиснута всередині меж кнопки). Перехід Normal → Hover відбувається при події MOUSEMOTION, коли курсор заходить на область кнопки; Hover → Normal на події MOUSEMOTION із координатами поза прямокутником; Hover → Pressed — при MOUSEBUTTONDOWN в межах кнопки; Pressed → Hover — при MOUSEBUTTONUP всередині, що також викликає прив’язану дію action(); Pressed → Normal — при відпусканні кнопки поза межами; і остаточний перехід Hover → Normal або Pressed → Normal і при звичайному MOUSEBUTTONUP поза кнопкою. Така модель гарантує коректну зміну візуального стилю та виклик колбеку в потрібний момент. Діграма станів наведена на рисунку 2.3.

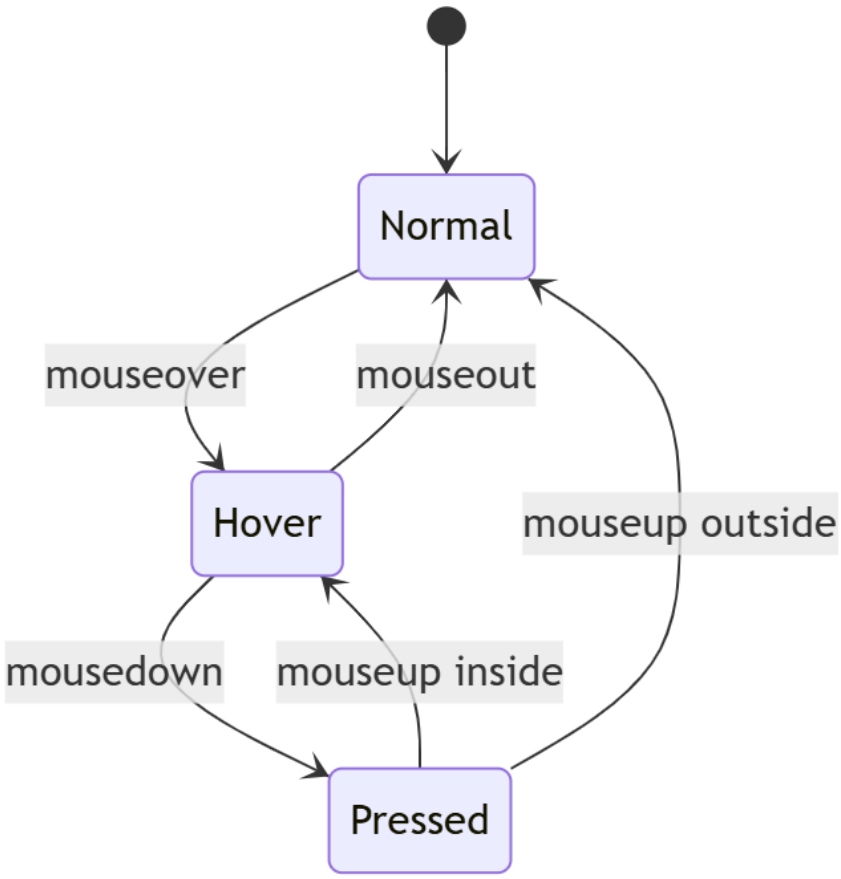


Рисунок 2.3 — Діаграма станів кнопки handle_event

Діаграма послідовності основного циклу роботи системи зображено

взаємодію між точками входу програми (main()), об’єктом інтерфейсу (Interface), ядром обробки кадру (PixelFX), менеджером пресетів і переходів (EffectManager) та власне наборами ефектів (Effects). Спочатку в main() створюється Interface передаючи йому посилання на pixelfx. Далі відбувається завантаження зображення через pixelfx.load_image(...). Після цього починається нескінченний цикл: у кожній ітерації для кожної події викликається Interface.process_event(event), яка проходить крізь всі кнопки. Потім виконується PixelFX.update() — всередині він рахує delta_time, делегує оновлення менеджеру ефектів (EffectManager.update(delta_time)), а той, у разі переходу між пресетами, оновлює прогрес анімаційного переходу і застосовує кожен ефект. Далі EffectManager.render(buffer_surface) малює буфер з усіма ефектами, після чого main() викликає Interface.render(screen), виводячи результат на екран і обчислюючи FPS, які передаються назад до PixelFX для статистики. Діаграма послідовності основного циклу роботи системи наведена на рисунку 2.4.

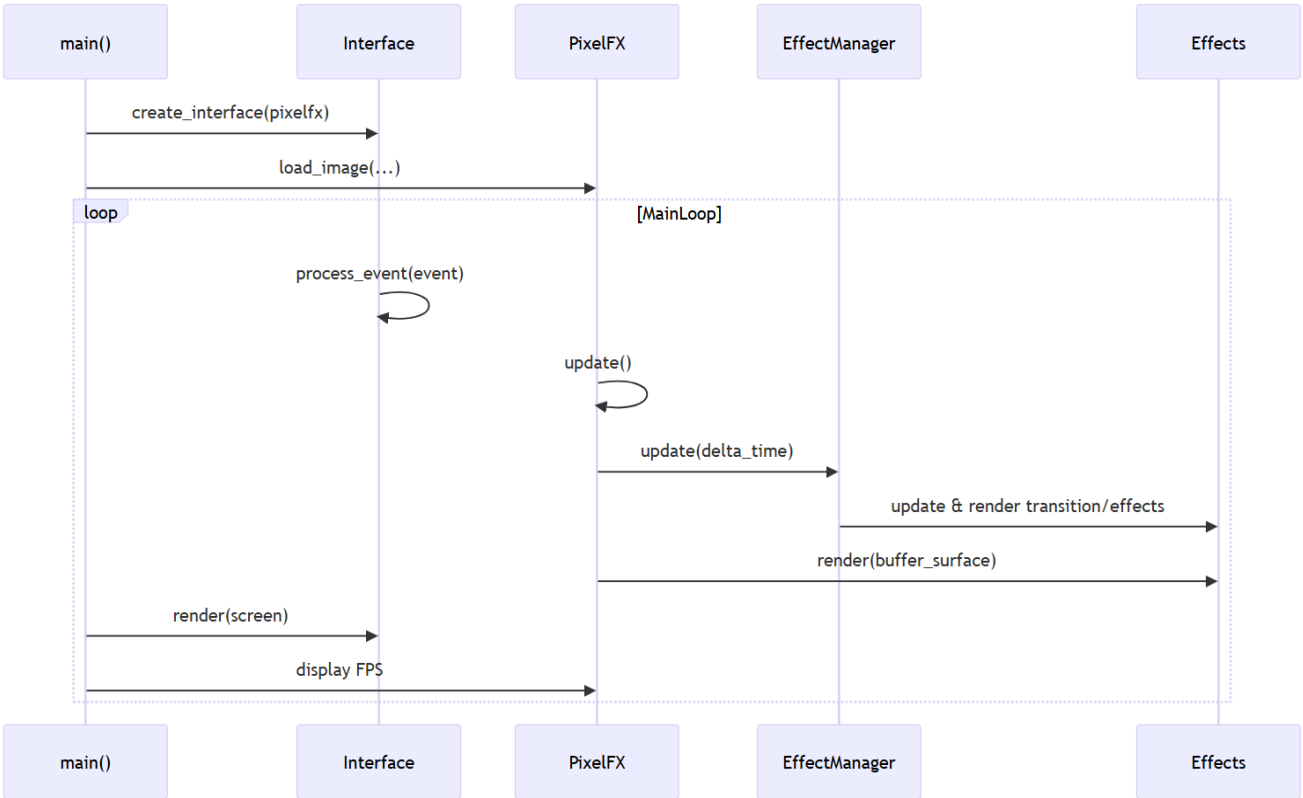


Рисунок 2.4 — Діаграма послідовності основного циклу системи

Діаграма класів редактора демонструє статичну структуру головного класу

VFXEditor (модуль верхнього рівня редактора ефектів). Він має атрибути налаштувань вікна (screen_width, screen_height), словник усіх доступних даних ефектів (effects_data) і список кнопок головного меню (main_menu_buttons). До функцій належать run() (головний цикл), _handle_event(event) (обробка кліків/натискань) та _render() (рендеринг поточного вікна). Зі стрілок видно, що VFXEditor асоціюється із класом Interface (відображення панелей і кнопок), з PixelFX (ядром обробки зображень) і опосередковано через нього — з EffectManager та конкретними класами ефектів. Діаграма класів редактора наведена на рисунку 2.5.

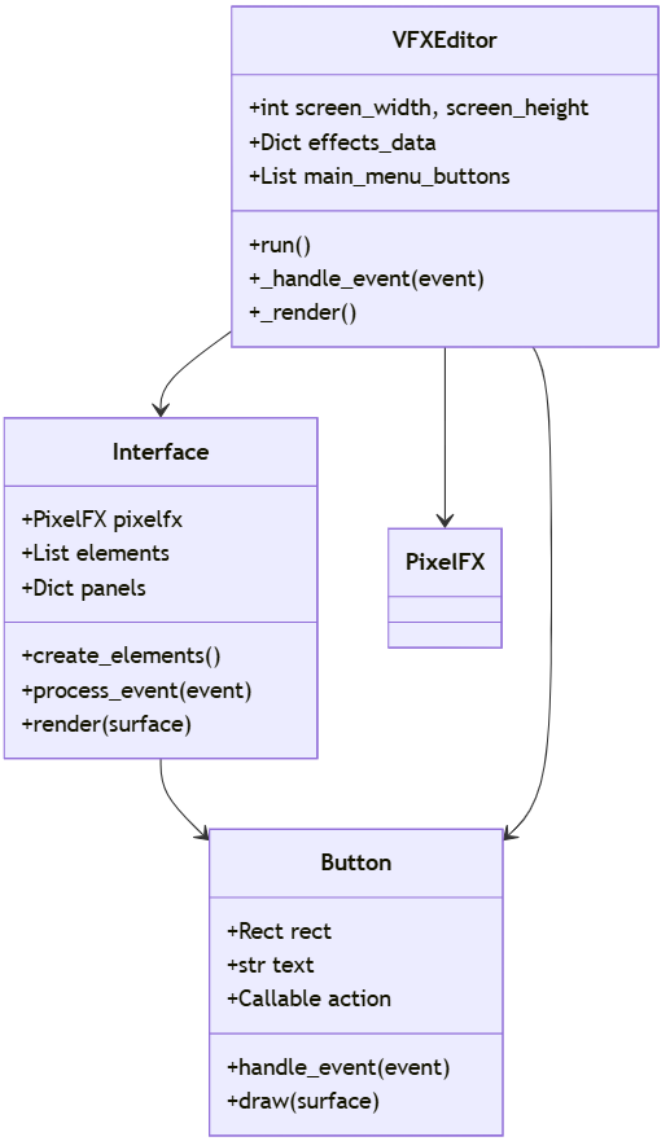


Рисунок 2.5 — Діаграма класів редактора

У діаграмі класів ядра системи центральне місце належить PixelFX: цей клас

містить посилання на основну поверхню Pygame (surface), оригінальне зображення (original_image), NumPy-масив (numpy_image), список активних ефектів та пресетів, а також прапори use_gpu та use_multithreading. Функції цього класу: load_image(), apply_preset(name), update(), render() та export_gif(). Стрілки показують, що для керування пресетами й переходами використовується EffectManager, для окремих типів обчислень можливе залучення OpenCLContext, а конкретні ефекти (RainEffect, SnowEffect, GlowEffect, InteractiveEffect) успадковують загальну інтерфейсну модель set_image_size, set_progress, update, render.

Діаграма класів ядра системи наведена на рисунку 2.6.

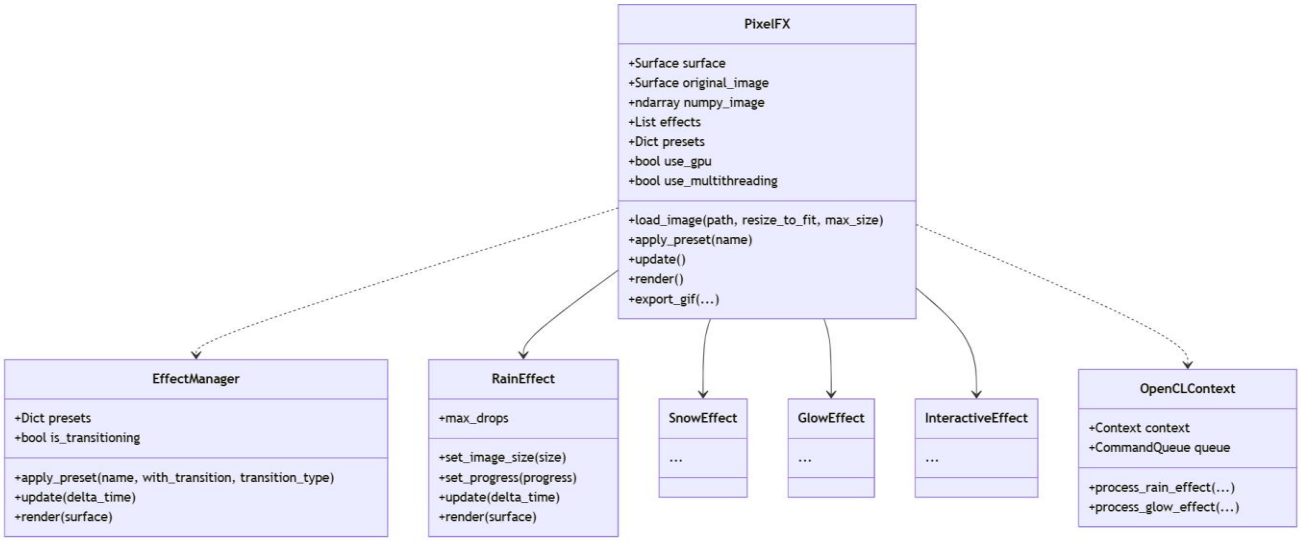


Рисунок 2.6 — Діаграма класів ядра системи

Алгоритм експорту GIF спершу перевіряє наявність бібліотеки PIL (Pillow). Якщо вона відсутня — виводиться повідомлення з інструкцією встановлення. Інакше обчислюються параметри анімації: загальна кількість кадрів total_frames = duration/1000*fps, тривалість одного кадру frame_duration = 1000/fps та створюється список копій ефектів для анімації. Далі у циклі for i in 0…total_frames−1 на кожному кроці обчислюється прогрес i/total_frames, створюється поверхня кадру, на неї покладаються всі ефекти через set_progress, update, render, після чого результат конвертується в об’єкт PIL Image, масштабується та оптимізується за кольорами.

Після завершення циклу перший кадр зберігається у GIF з параметрами save_all=True, append_images=frames[1:], optimize=True, duration=frame_duration, loop=0. Наприкінці функція повертає True (успіх) або, у разі будь-якої помилки, ловить виняток, виводить трасу та повертає False. Блок-схема алгоритму експорту GIF наведено на рисунку 2.7.

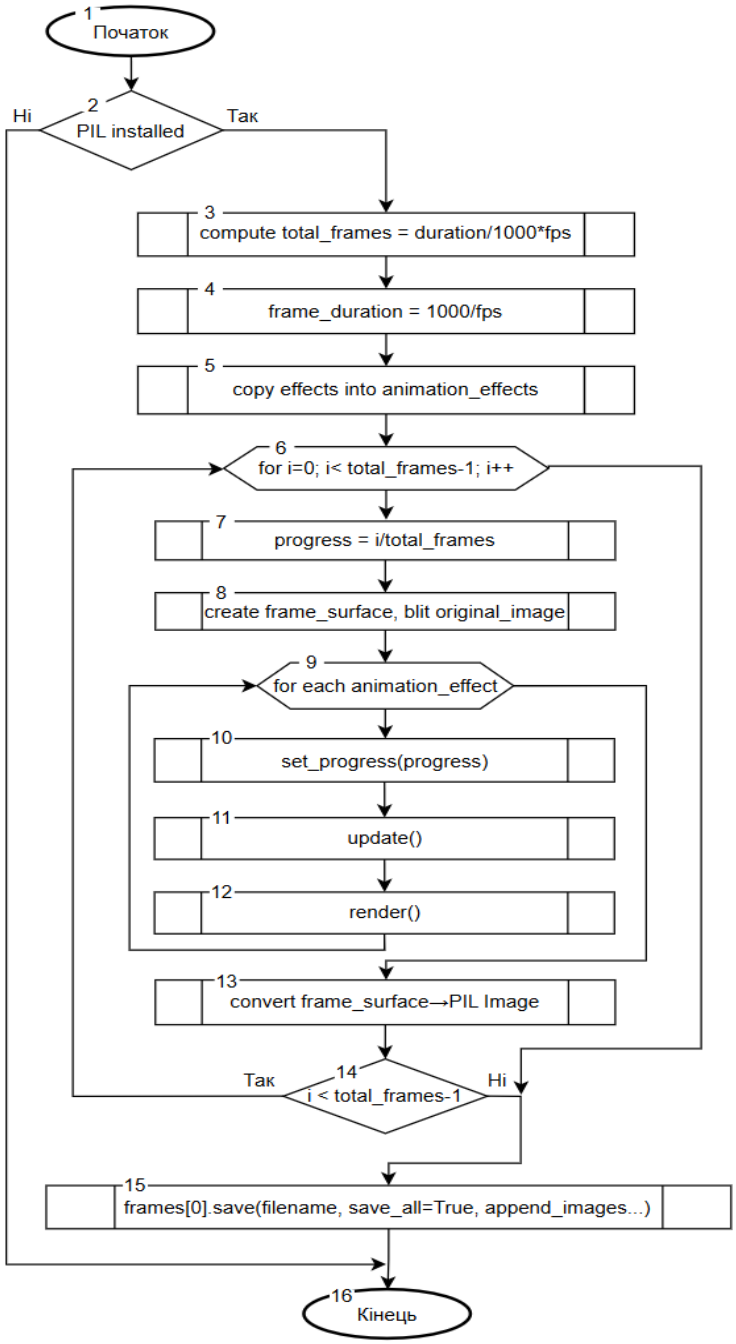


Рисунок 2.7 — Блок-схема алгоритму експорту GIF

2.6 Висновки

У другому розділі надано детальний опис алгоритмів програми. Також описано вхідні та вихідні дані системи, а також метод накладання динамічних ефектів на статичне зображення, метод системи інтерактивних частинок, метод багатопроцесорного експорту анімацій з адаптивним розподілом ресурсів.

3 РОЗРОБКА БІБЛІОТЕКИ ВІЗУАЛЬНИХ ЕФЕКТІВ

3.1 Варіантний аналіз і обґрунтування вибору засобів для реалізації програмного забезпечення

Розглянемо такі мови програмування, як: Python, C++, JavaScript.

Python [17] є високорівневою інтерпретованою мовою програмування з динамічною типізацією, що відзначається надзвичайно читабельним синтаксисом і простотою освоєння. Особливо цінною для розробки бібліотеки візуальних ефектів є розвинена екосистема наукових та графічних бібліотек, таких як NumPy, Pygame, Pillow (PIL) і OpenCV. Python дозволяє швидко прототипувати складні функції та ефекти завдяки високому рівню абстракції, а його мультипарадигмальність (підтримка об'єктно-орієнтованого, функціонального та процедурного програмування) забезпечує гнучкість при проектуванні архітектури програми. Незважаючи на відносно нижчу продуктивність порівняно з компільованими мовами, Python може ефективно задіяти можливості GPU через інтеграцію з бібліотеками як PyTorch і модулем multiprocessing для паралельної обробки, що критично важливо для обробки зображень та створення анімацій.

C++ [18] є мовою програмування, що поєднує парадигми структурного, об'єктно-орієнтованого та узагальненого програмування. Мова забезпечує високу продуктивність та низькорівневий доступ до апаратних ресурсів, що робить її традиційним вибором для розробки графічних застосунків з високими вимогами до швидкодії. Для створення бібліотеки візуальних ефектів C++ пропонує потужні можливості у поєднанні з такими бібліотеками як OpenGL, SFML або SDL для графіки, а також OpenCV для обробки зображень. Сильною стороною C++ є можливість оптимізації під конкретну апаратну платформу та ефективне використання багатопоточності. Однак розробка на C++ вимагає більше часу через складніший синтаксис, ручне керування пам'яттю та більш тривалий цикл компіляції, що уповільнює процес ітеративної розробки та експериментування з новими ефектами.

JavaScript [19] є мовою програмування, яка традиційно використовується для

веб-розробки, але з розвитком Node.js та фреймворків як Electron набула популярності і для створення десктопних застосунків. Для розробки бібліотеки візуальних ефектів JavaScript пропонує потужні можливості через Canvas API, WebGL для апаратно-прискореної графіки та бібліотеки як Three.js для 3D-візуалізації. Основними перевагами JavaScript є широка доступність (працює в будь-якому браузері), великий екосистемний ресурс пакетів NPM та порівняно швидкий цикл розробки з моментальним відображенням змін. Проте JavaScript має обмеження у високонавантажених обчисленнях через свою однопоточну природу (хоча Web Workers частково вирішують цю проблему), а також менш розвинену підтримку для наукових обчислень та обробки зображень порівняно з Python, що може ускладнити реалізацію складних алгоритмів для візуальних ефектів.

З метою порівняння цих мов програмування, для визначення найкращої мови для розробки бібліотеки візуальних ефектів, було сформовано таблицю 3.1.

Таблиця 3.1 — Порівняння мов програмування

Характеристика	Python	C++	JavaScript
Екосистема бібліотек для обробки зображень	+	+	-
Нативна підтримка багатопроцесорності	+	+	-
Доступність GPU-прискорення	+	+	+
Кросплатформеність	+	-	+
Підтримка математичних і векторних операцій	+	+	-
Вбудовані структури даних для роботи з графікою	+	-	-
Автоматичне керування пам'яттю	+	-	+
Сумарний бал	7	4	3

Python є оптимальним вибором для розробки бібліотеки візуальних ефектів завдяки поєднанню переваг:

- екосистеми бібліотек для обробки зображень (NumPy, Pygame, Pillow, OpenCV), що дозволяє швидко реалізовувати складні алгоритми;

- високої швидкості розробки завдяки читабельному синтаксису та

динамічній типізації;

- ефективному вирішенню питань продуктивності через інтеграцію з multiprocessing для паралельної обробки та бібліотеками для GPU-прискорення.

3.2 Розробка модуля обробки зображень

Модуль обробки зображень і основний цикл (PixelFX у pixelfx.core) Це серце бібліотеки, яке відповідає за завантаження, зберігання, оновлення й рендеринг зображень із накладеними ефектами. У конструкторі PixelFX налаштовуються параметри апаратного прискорення (GPU, OpenCL), багатопоточності, ініціалізуються внутрішні буфери й статистика FPS. Функція load_image(path, resize_to_fit, max_size) намагається прочитати файл через OpenCV для швидшої роботи, із масштабуванням в разі потреби, або, за відсутності OpenCV, через Pygame; результатом стають поверхня original_image і масив numpy_image. У циклі update() рахується дельта-час між кадрами, здійснюється або багатопотокове, або послідовне оновлення всіх активних ефектів, із заміром часу кожного, а статистика накопичується для виводу [20]. Потім render() копіює оригінал у буфер, рендерить усі ефекти зверху, відмальовує буфер на екран і, за бажанням, виводить лічильник FPS та середній час рендерингу кожного ефекту. Окремою функцією export_gif(...) генерується анімація в GIF: розрахунок кадрів, застосування ефектів із прогресом анімації, конвертація в PIL, оптимізація кольорів і збереження в один файл.

PixelFX є основним класом бібліотеки, що інкапсулює завантажену поверхню, зображення, список активних ефектів, пресетів і налаштувань рендерингу. У конструкторі зберігаються параметри (use_gpu, device, use_multithreading), ініціалізуються порожні списки effects та presets, створюється структура статистики. Якщо включено багатопоточність — створюється пул ThreadPoolExecutor. Якщо потрібне OpenCL-прискорення — намагаються імпортувати та ініціалізувати OpenCLContext. Після цього екземпляр готовий до завантаження зображень та застосування ефектів у циклі оновлення/рендеру.

Блок-схема процесу ініціалізації та завантаження зображення наведена на

рисунку 3.1.

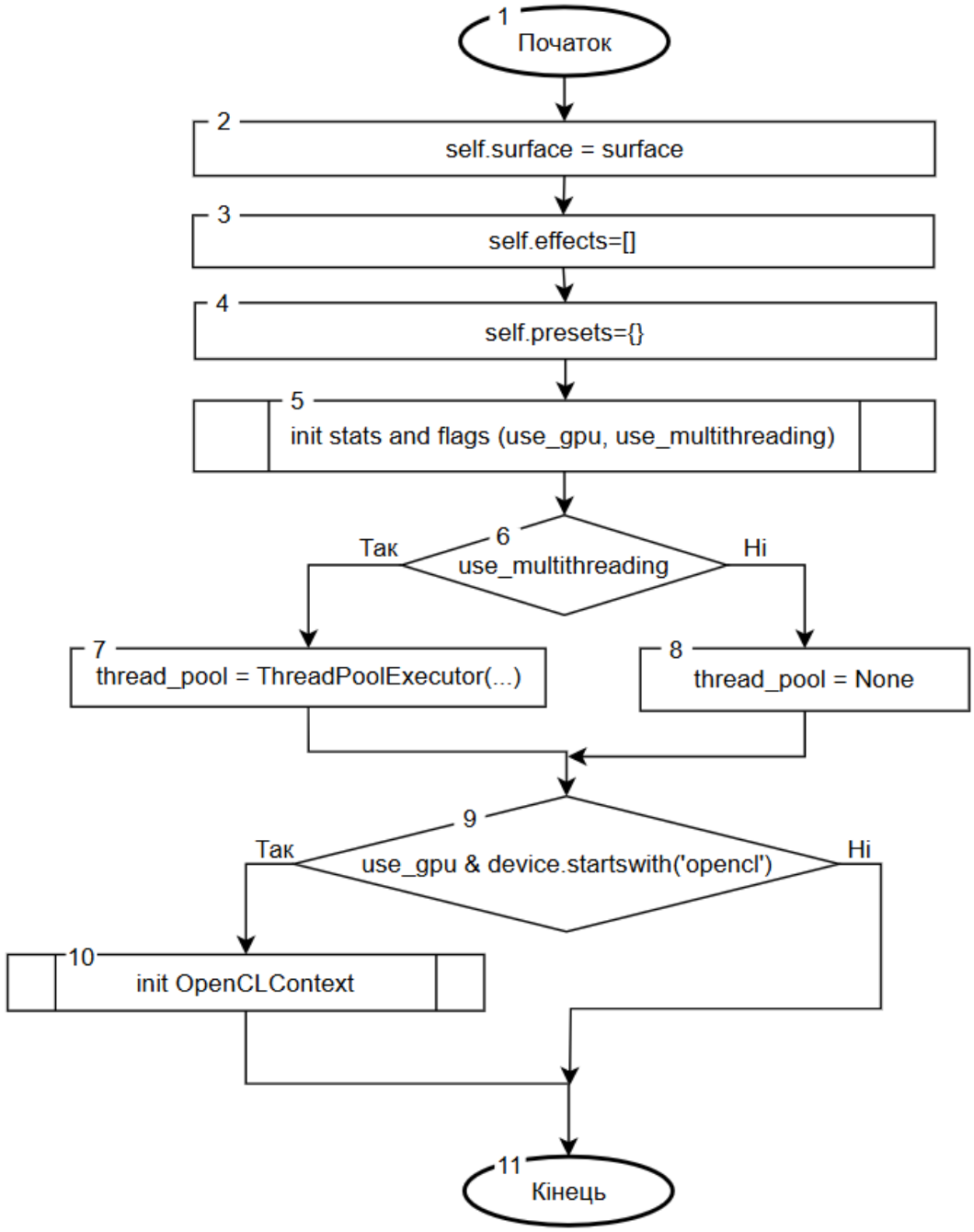


Рисунок 3.1 — Блок-схема процесу ініціалізації та завантаження зображення

Функція load_image(path, resize_to_fit, max_size) завантажує зображення з диску у об’єкт PixelFX. Спочатку він намагається використовувати OpenCV (cv2.imread), а при успішному читанні конвертує в RGB, зберігає як NumPy-масив

і створює поверхню Pygame. Опціонально виконує плавне зменшення до max_size. Якщо OpenCV не встановлено або читання не вдалось, резервний варіант через pygame.image.load робить аналогічну обробку. У кінці і в першому, і в другому шляхах створюється buffer_surface для рендерингу ефектів, і функція повертає True при успіху або False при будь-якій помилці. Блок-схема процесу завантаження зображення наведена на рисунку 3.2.

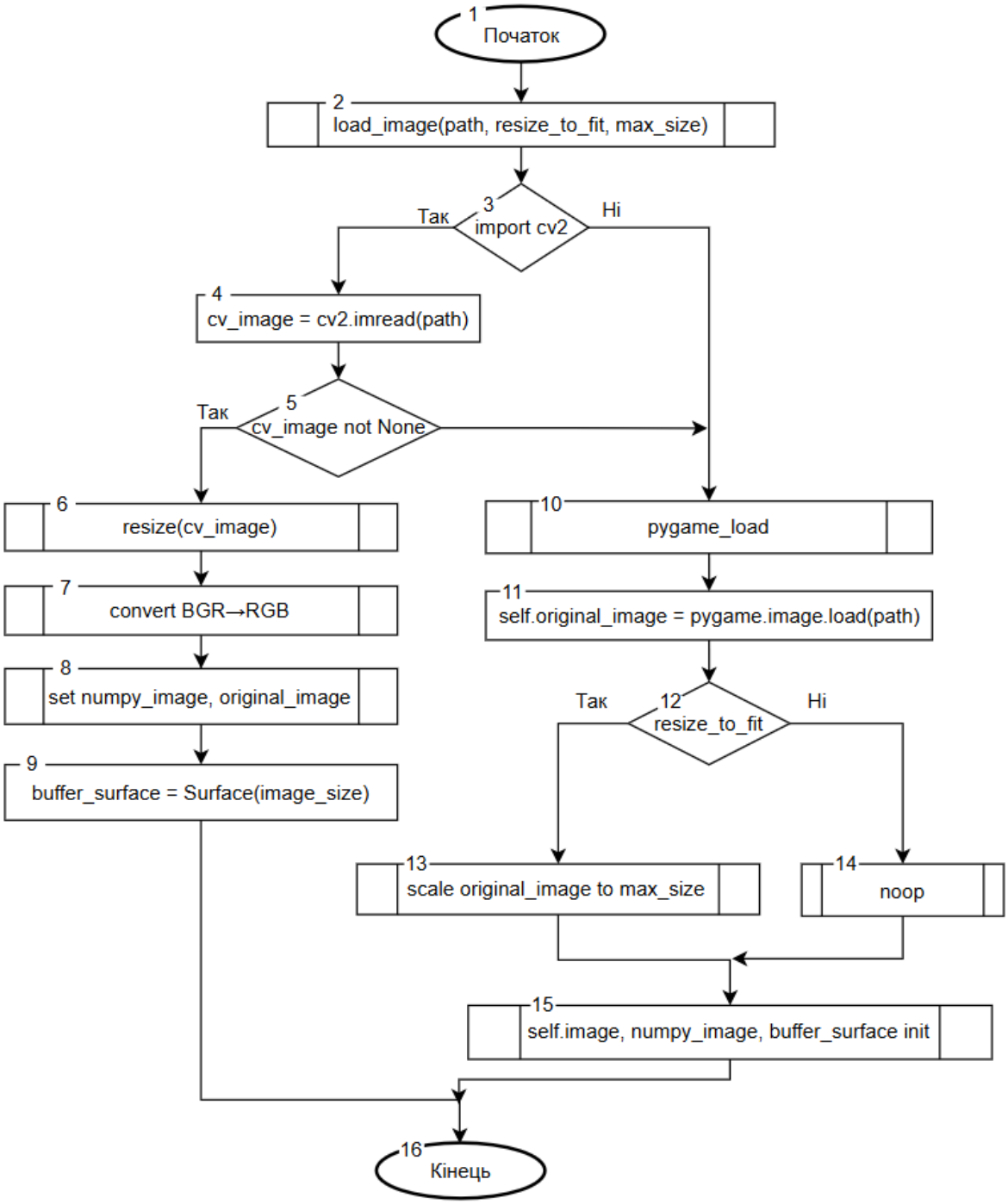


Рисунок 3.2 — Блок-схема процесу завантаження зображення

Функція update() відповідає за оновлення всіх активних ефектів у циклі анімації. Він фіксує поточний час, обчислює delta_time (мс від останнього кадру) і, якщо увімкнено багатопоточність та є кілька ефектів, відправляє кожен виклик effect.update(delta_time) у пул потоків, чекаючи завершення всіх задач. Інакше виконує послідовне оновлення, заміряючи час виконання кожного ефекту та зберігаючи його для статистики. Таким чином реалізовано гнучкість між продуктивністю та простотою обробки.

Блок-схема функції оновлення ефектів наведена на рисунку 3.3.

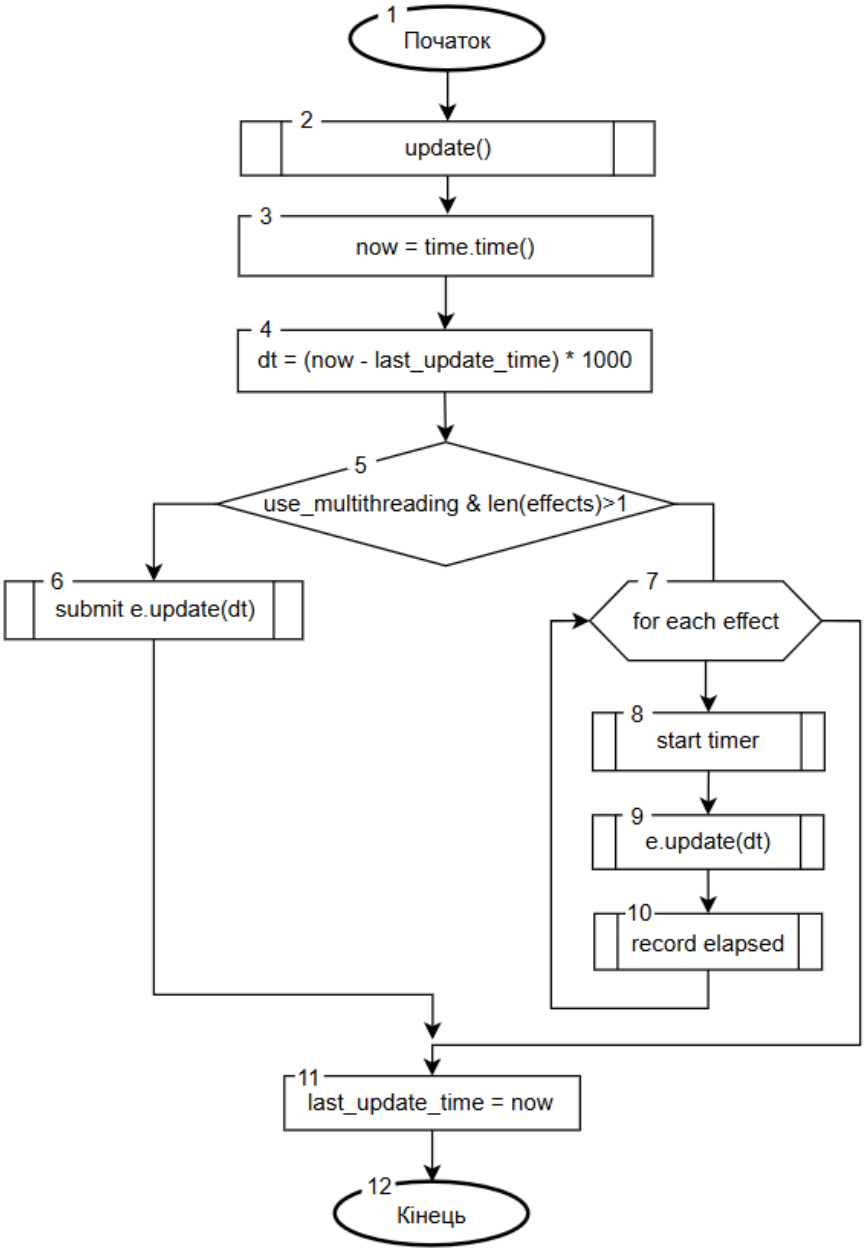


Рисунок 3.3 - Блок-схема функції оновлення ефектів

Функція render() малює на екрані результат застосування усіх ефектів до оригінального зображення. Спочатку перевіряє, чи завантажений кадр, потім копіює його у buffer_surface. Далі послідовно викликає effect.render(buffer_surface) для кожного активного ефекту, заміряючи час. Після цього буфер виводиться в головний surface. Якщо включене відображення статистики, викликається render_stats() — малює лічильник FPS і середній час рендерингу кожного ефекту

поверх кадру. Блок-схема процесу рендерингу ефектів наведена на рисунку 3.4.

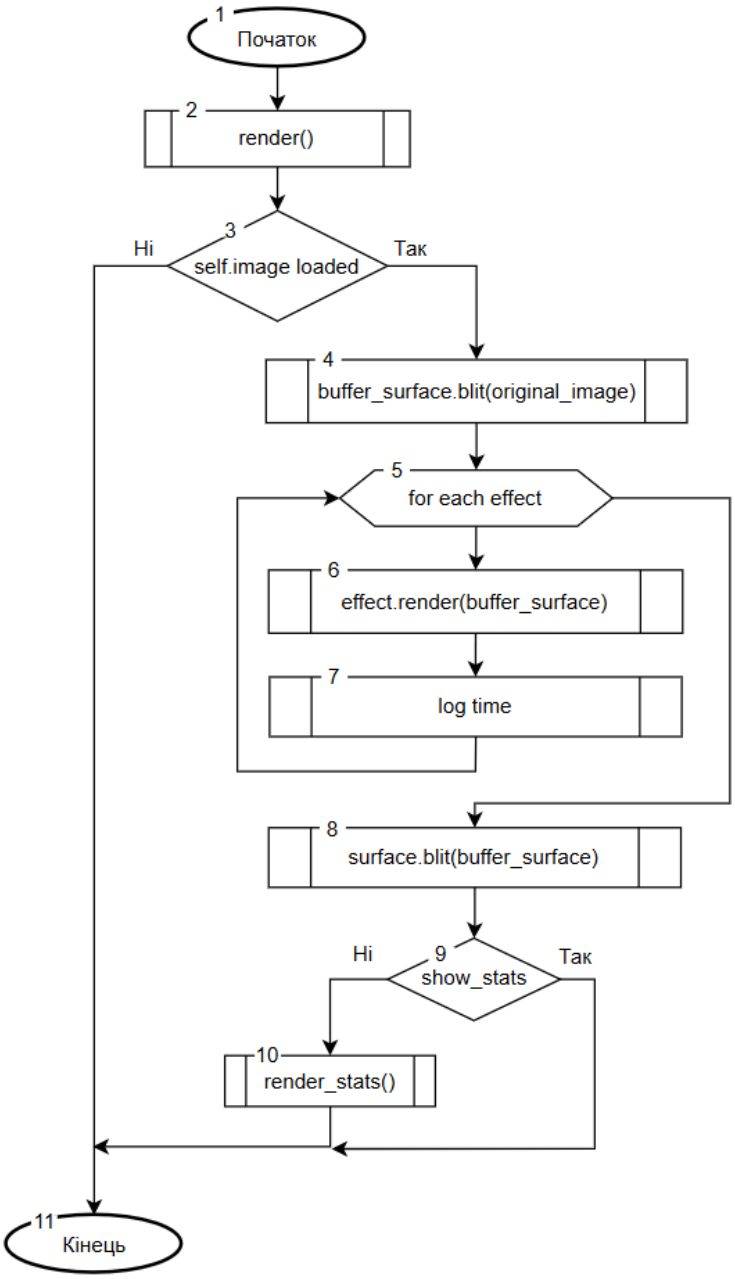


Рисунок 3.4 — Блок-схема процесу рендерингу ефектів

Модуль обробки зображень виступає ядром бібліотеки, поєднуючи завантаження й масштабування зображень, мультипотокове або GPU-прискорене оновлення ефектів, збирання статистики продуктивності, рендеринг у реальному часі та експорт анімацій у GIF. Його чіткий, модульний дизайн полегшує інтеграцію з різними ефектами та забезпечує високу продуктивність навіть із великою

кількістю одночасно активних об’єктів.

3.3 Розробка модуля керування пресетами і переходами

Цей модуль координує застосування попередньо налаштованих наборів ефектів із підтримкою плавних анімаційних переходів. Об’єкт EffectManager зберігає словник пресетів (назва → список об’єктів ефектів), поточний пресет і стан переходу, включно з його тривалістю та прогресом. Функція apply_preset(name, with_transition, transition_type) перевіряє, чи потрібно робити перехід: якщо ні — просто замінює ефекти; якщо так — копіює поточний кадр у тимчасову поверхню, встановлює новий набір ефектів і створює один із трьох об’єктів переходу (FadeTransition, WipeTransition, ZoomTransition) із початковим прогресом 0. Далі у кожному кадрі update(delta_time) нарощує прогрес переходу, доки він не дійде до 1.0, а render(surface) викликає конкретний перехідний ефект, щоб змішати старий і новий кадри за допомогою затухання, зсуву чи масштабування.

EffectManager керує пресетами ефектів у бібліотеці PixelFX і підтримує плавні переходи між ними. Функція apply_preset(name, with_transition, transition_type) перевіряє наявність пресету з вказаною назвою. Якщо перехід не потрібен або програма тільки-но запустилась, просто призначає список ефектів. В іншому випадку він копіює поточний буфер кадру у тимчасову поверхню (temp_surface), встановлює новий пресет, створює відповідний об’єкт переходу (FadeTransition, WipeTransition або ZoomTransition) на основі аргумента transition_type, ініціалізує transition_progress = 0 і ставить прапор is_transitioning = true, щоб у наступних кадрах почалася анімація змішування. Блок-схема процесу застосування пресету з переходом наведена на рисунку 3.5.

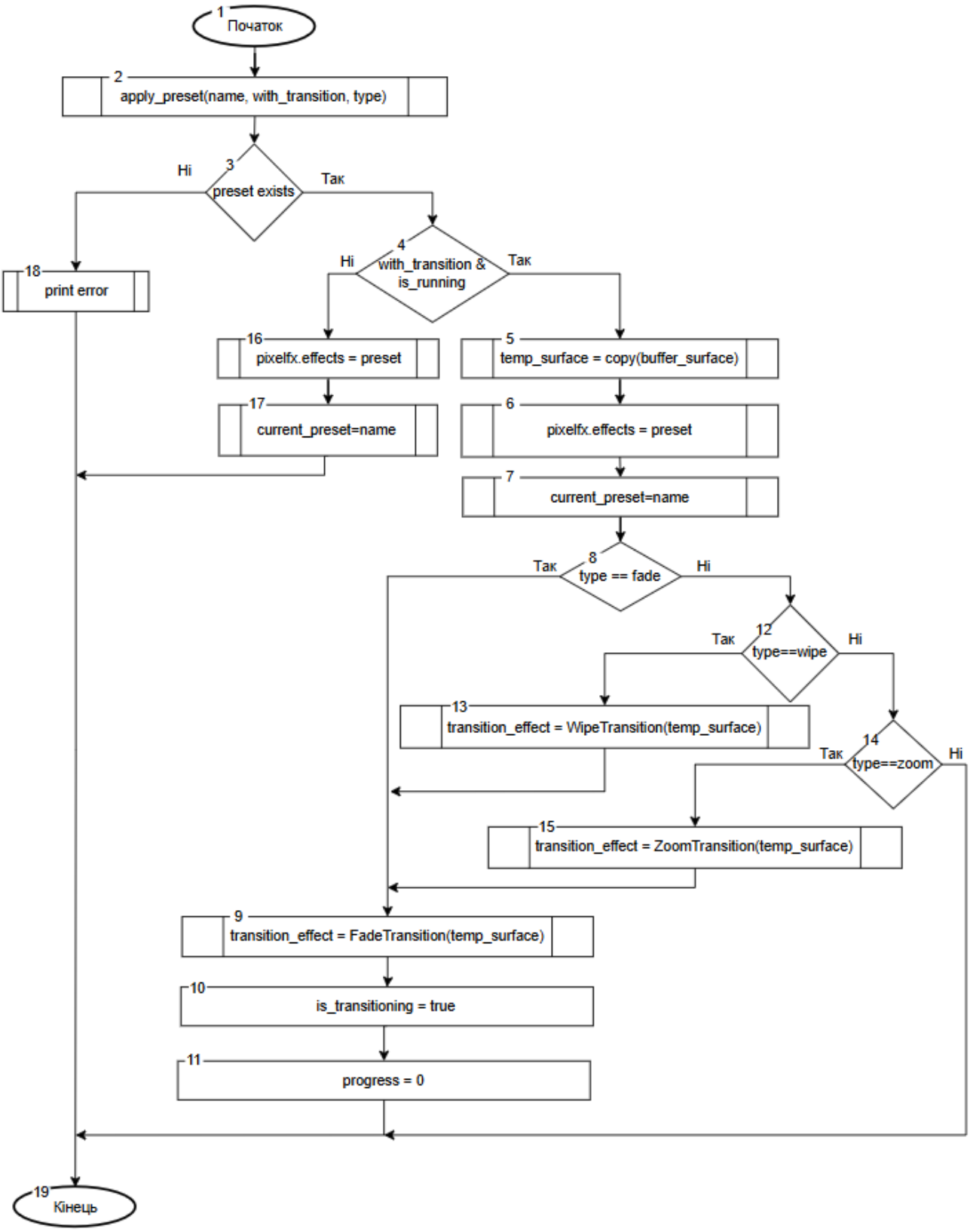


Рисунок 3.5 - Блок-схема процесу застосування пресету з переходом

Функція update(delta_time) у EffectManager відповідає за кроки анімаційного переходу між пресетами. Якщо is_transitioning вимкнений, нічого не відбувається.

Інакше він додає до transition_progress нормалізовану величину часу (відношення delta_time до загальної тривалості transition_duration). Коли прогрес досягає або перевищує 1.0, перехід завершується: прапор is_transitioning вимикається, а transition_progress фіксується на 1.0, що сигналізує про завершення анімації. Блок-схема процесу оновлення стану переходу наведена на рисунку 3.6.

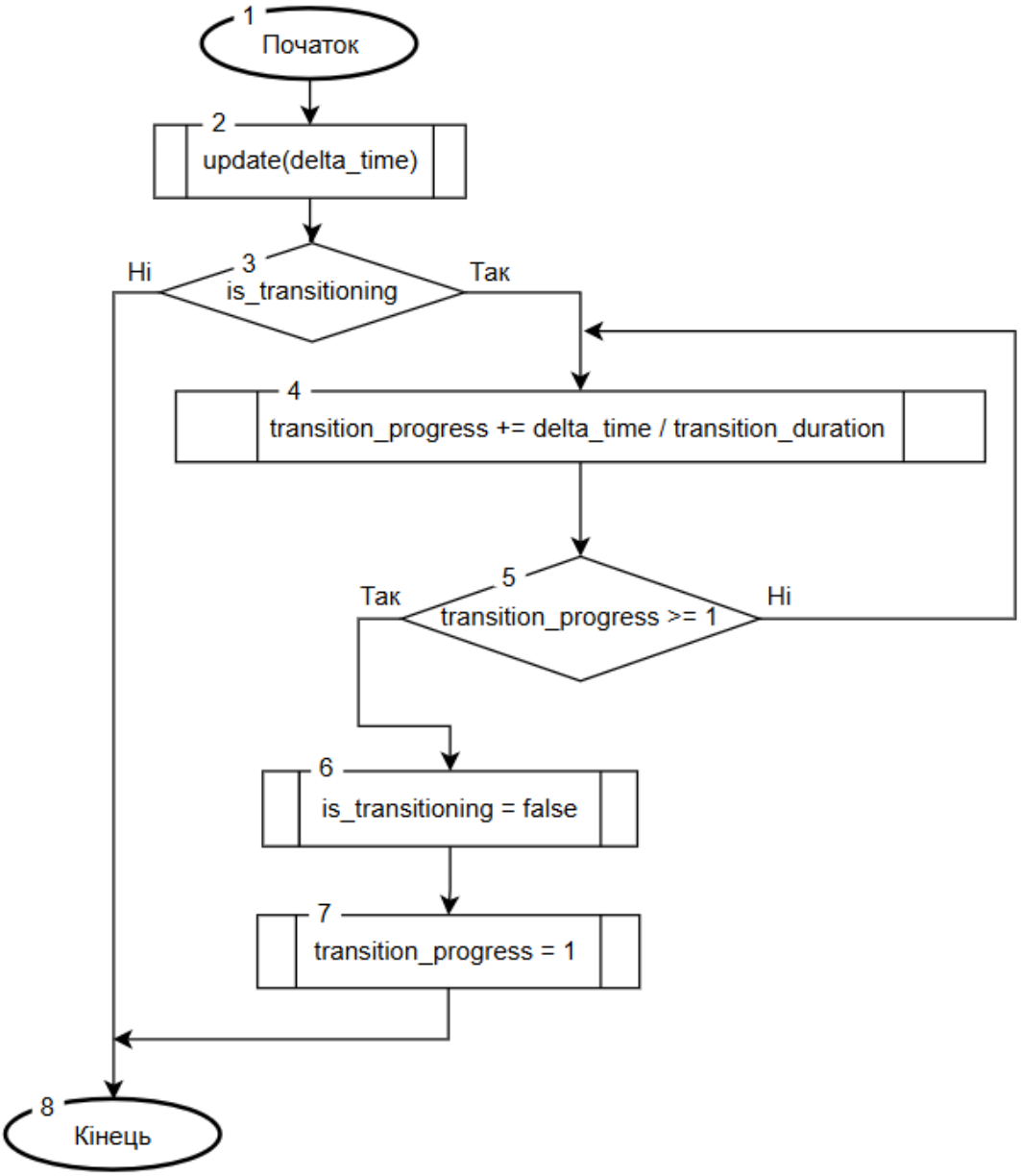


Рисунок 3.6 - Блок-схема процесу оновлення стану переходу

Функція render(surface) класа EffectManager малює сам перехід поверх уже застосованих ефектів. Якщо перехід не активний (is_transitioning == false) або

відсутній об’єкт transition_effect, функція одразу повертається. Інакше він викликає transition_effect.render(surface, transition_progress), передаючи поточний прогрес, щоб цей об’єкт відмалював ефект затухання, змивання або масштабування на основі його власної логіки. Блок-схема процесу малювання переходу наведена на рисунку 3.7.

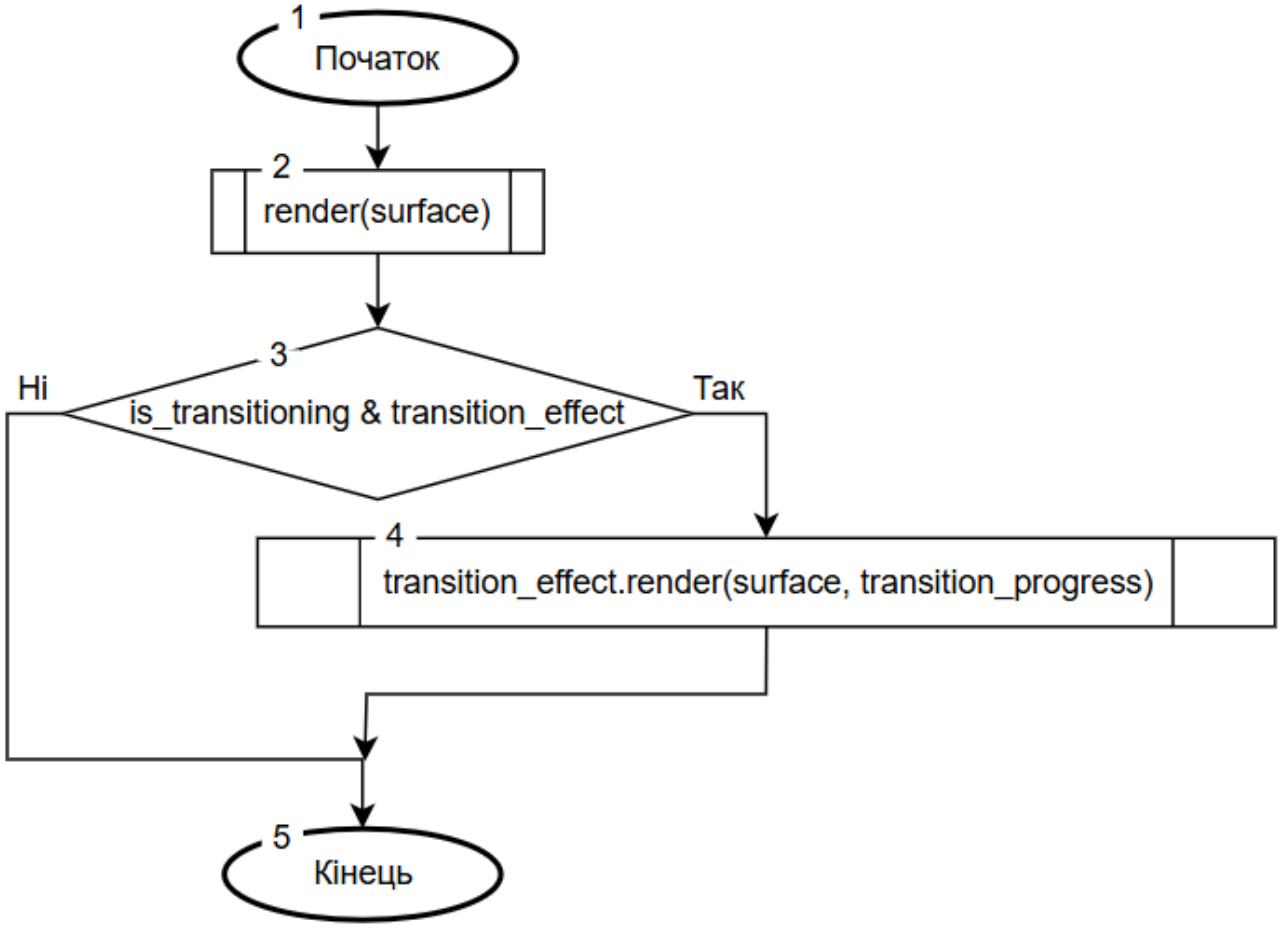


Рисунок 3.7 - Блок-схема процесу малювання переходу

Модуль керування ефектами та переходами узагальнює механізм створення, застосування та плавної анімації між пресетами ефектів, розділяючи відповідальність із класом PixelFX. Він чітко окреслює, як і коли починати перехід, як оновлювати його прогрес і як малювати змішаний кадр, що дозволяє додавати нові типи переходів і налаштовувати їх незалежно від основної логіки рендерингу.

3.4 Розробка модуля інтерфейсу користувача

Цей модуль відповідає за побудову й управління графічними елементами управління у вікні Pygame. Він містить клас Button, який інкапсулює логіку відображення прямокутника з текстом, трьома основними візуальними станами (штатний, наведення, натискання) та колбеком на клік. Основний клас Interface збирає разом усі такі кнопки, створює напівпрозору панель управління у верхній частині екрана та динамічно розташовує на ній елементи: кнопки пресетів, контролі швидкості GIF, кнопки експорту й збереження. При зміні розміру вікна Interface оновлює власну панель і перестворює всі кнопки, щоб їхні координати відповідали новим розмірам. Функція process_event передає кожну подію миші й клавіатури всім кнопкам, а render відповідає за малювання панелей і відтворення поточного стану кожного елемента на поверхні.

Клас Button відповідає за відображення інтерактивної кнопки у Pygame: він знає власний прямокутник (rect), текст, кольори для різних станів і функцію-обробник action. У цій функції спочатку перевіряється тип отриманої події. Якщо це переміщення миші (MOUSEMOTION), то за допомогою перевірки поточної позиції курсору відносно прямокутника кнопки вирішується, чи перейти в стан наведення (hover), чи в штатний стан (normal). Якщо ж подія — натискання лівої кнопки миші (MOUSEBUTTONDOWN з button == 1), і курсор знаходиться над прямокутником кнопки, то стан змінюється на «натиснено» (pressed). Коли надходить подія відпускання кнопки (MOUSEBUTTONUP), функція перевіряє, чи попередній стан був pressed. Якщо так, і курсор досі над кнопкою, спрацьовує прив’язана дія action(), після чого стан переходить у hover; в інших випадках стан скидається в normal. Така послідовна перевірка дозволяє коректно відслідковувати всі можливі взаємодії користувача з кнопкою.

Блок-схема послідновності обробки подій у функції handle_event() класу Button наведена на рисунку 3.8.

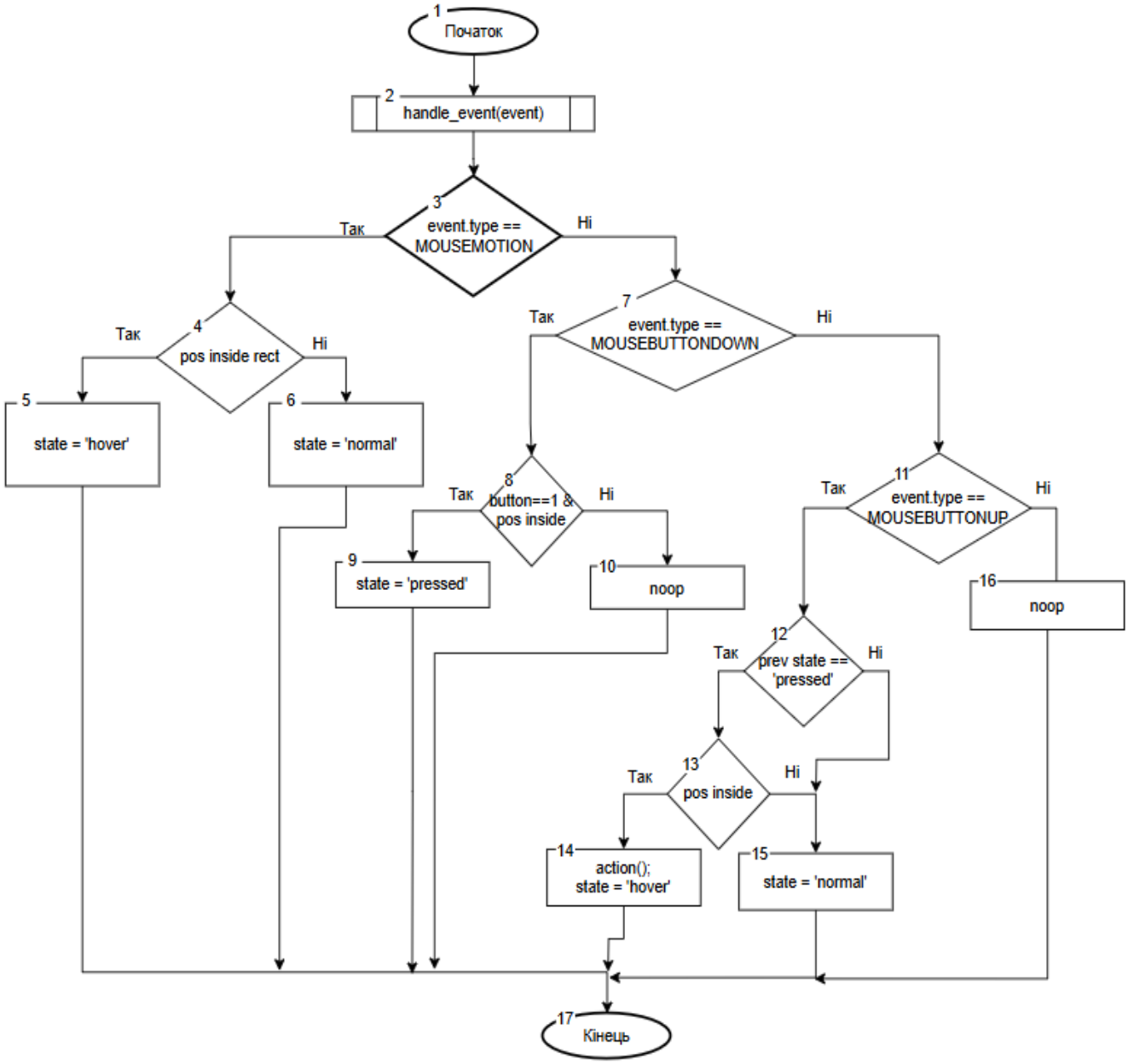


Рисунок 3.8 - Блок-схема послідновності обробки подій у функції handle_event() класу Button

Клас Interface формує графічний інтерфейс для керування застосунком PixelFX: він збирає всі кнопки й панелі та відповідає за їх розташування. Функція create_elements викликається під час ініціалізації або при зміні розмірів вікна і виконує такі кроки: визначає розміри екрану, створює напівпрозору верхню панель, динамічно генерує кнопки для кожного пресету, додає підпис «GIF Speed» і кнопки регулювання швидкості анімації, а також кінцеві кнопки «Export GIF» і «Save PNG», розташовані у правому куті.

Блок-схема процесу побудови елементів інтерфейсу наведена на рисунку 3.9.

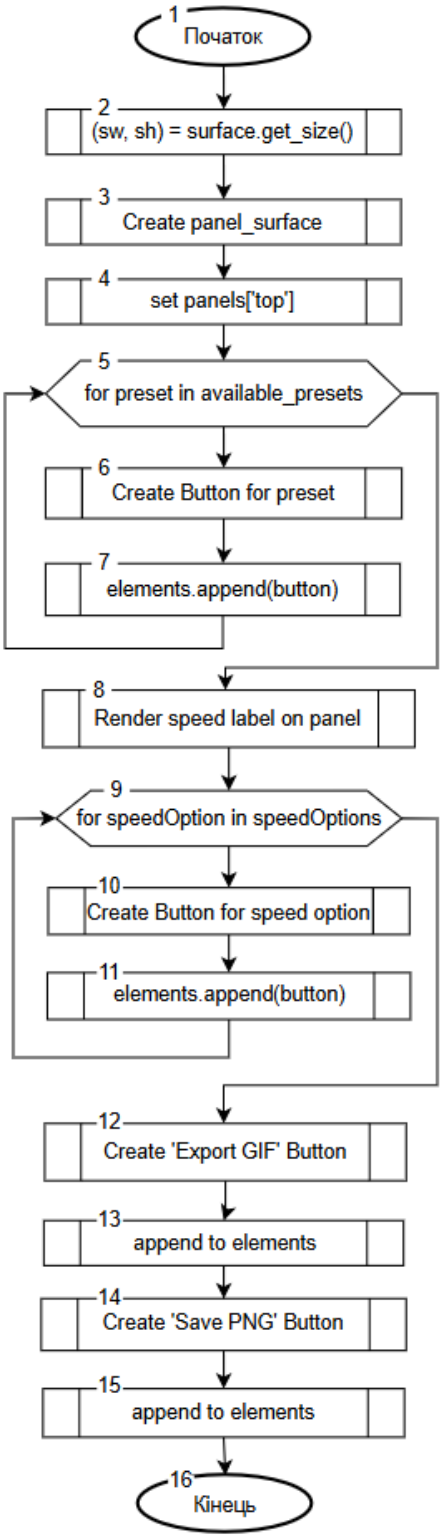


Рисунок 3.9 - Блок-схема процесу побудови елементів інтерфейсу

Модуль інтерфейсу користувача реалізує всю взаємодію користувача з бібліотекою через візуальні елементи Pygame — кнопки, панелі та динамічне

розташування. Він ізолює складну логіку обробки подій і малювання інтерфейсу в окремі класи `Button` та `Interface`, що спрощує підтримку та розширення функціональності без втручання в ядро обробки кадрів.

3.5 Висновки

У третьому розділі було проведено порівняльний аналіз мов програмування розробки бібліотеки візуальних елементів та обрано мову Python. Було розроблено модуль обробки зображень, модуль керування пресетами і переходами, модуль інтерфейсу користувача.

4 ТЕСТУВАННЯ ПРОГРАМИ

4.1 Тестування системи

Тестування програмного забезпечення є важливим етапом розробки, який має завдання перевірити роботу бібліотеки візуальних ефектів та забезпечення її стабільності, якості та відповідності функціональним вимогам. Основна задача тестування — виявлення помилок у програмних модулях, дефектів взаємодії між ефектами, а також перевірка відповідності поведінки бібліотеки очікуваним сценаріям у рамках графічного або ігрового застосунку.

Перед початком тестування на комп'ютер встановлюється тестова версія програмного модуля, розробленого для середовищ Windows або macOS, із попередньо налаштованими сценами та конфігураціями ефектів. Також підготовлюється технічна документація, зокрема план тестування, специфікації API та опис підтримуваних параметрів ефектів. Процес тестування включає кілька основних етапів.

На першому етапі перевіряється коректність роботи всіх реалізованих ефектів (системи частинок, постобробка, освітлення, туман, тощо).

Перевіряється:

- відображення візуальних ефектів у вікні попереднього перегляду;
- динамічна реакція ефектів на зміну параметрів у реальному часі;
- правильне збереження налаштувань у форматах .vfx та .json;
- адекватна реакція на невалідні або неповні конфігурації;
- стабільність при активації кількох ефектів одночасно.

Другий етап перевіряє здатність бібліотеки реагувати на критичні ситуації:

- імпорт файлу ефекту у неправильному форматі (наприклад, .txt замість .vfx);
- навмисно пошкоджені конфігурації (відсутність ключових параметрів);
- відсутність прив'язки ефекту до сцени;
- надмірно висока інтенсивність або розмір ефекту, що виходить за межі екрану.

У результаті тестування таких сценаріїв, бібліотека повинна повідомити

користувача про помилку, запобігти аварійному завершенню роботи та запропонувати альтернативні дії (наприклад, імпорт коректного файлу або скидання параметрів до дефолтних).

Тестування застосунку розпочинається з його головного вікна. Воно містить кнопки для переходу у редактор зображення, попередній перегляд ефектів, галерею шаблонів та пресетів, налаштування застосунку.

Головне вікно застосунку наведена на рисунку 4.1.

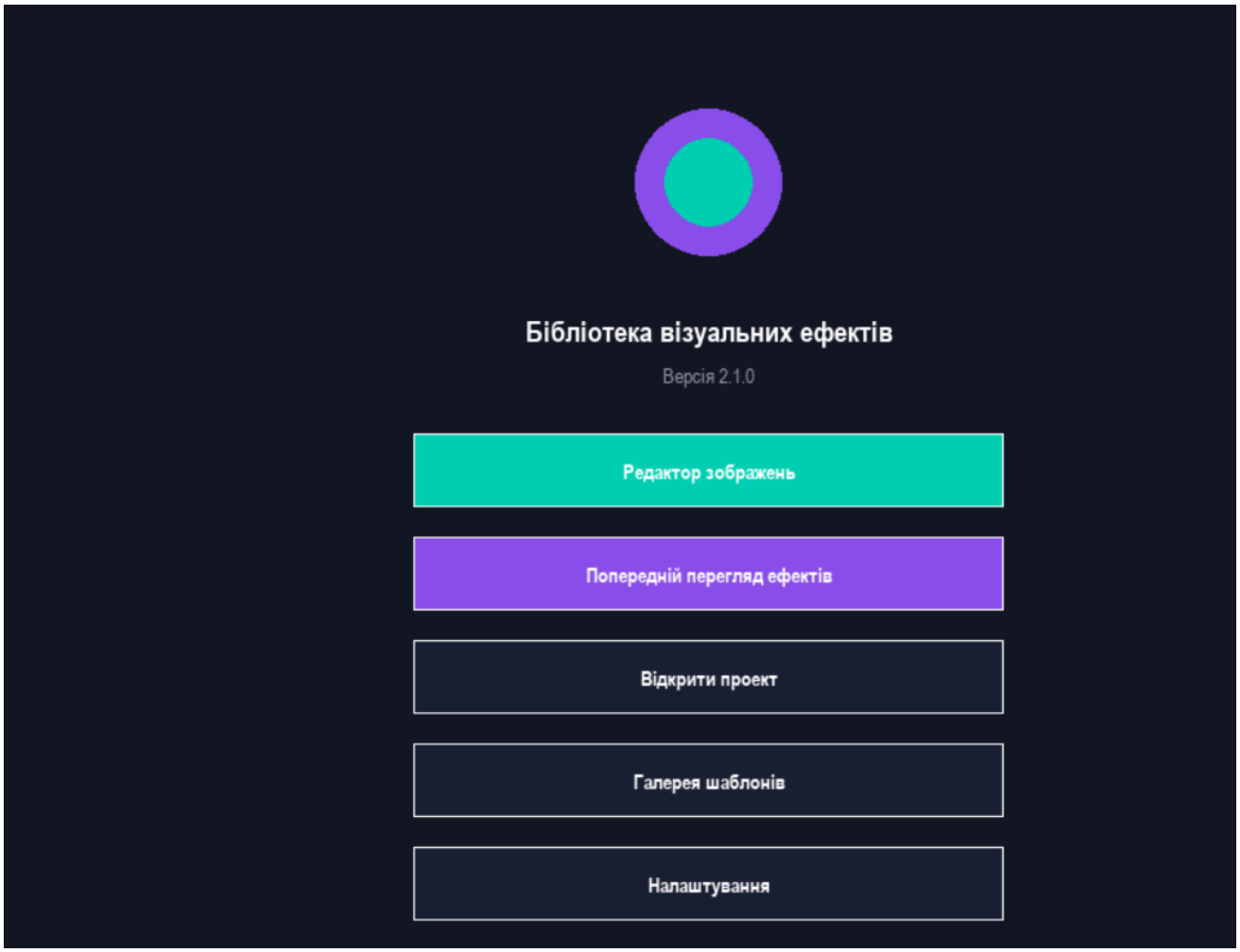


Рисунок 4.1 — Головне вікно програми

Перейдемо до попереднього перегляду ефектів. Цей модуль дозволяє створювати візуальні ефекти для ігрових застосунків, налаштовувати їх, зберігати у системі або експортувати для подальшого використання. Початковий стан модуля

попереднього перегляду ефектів наведено на рисунку 4.2.

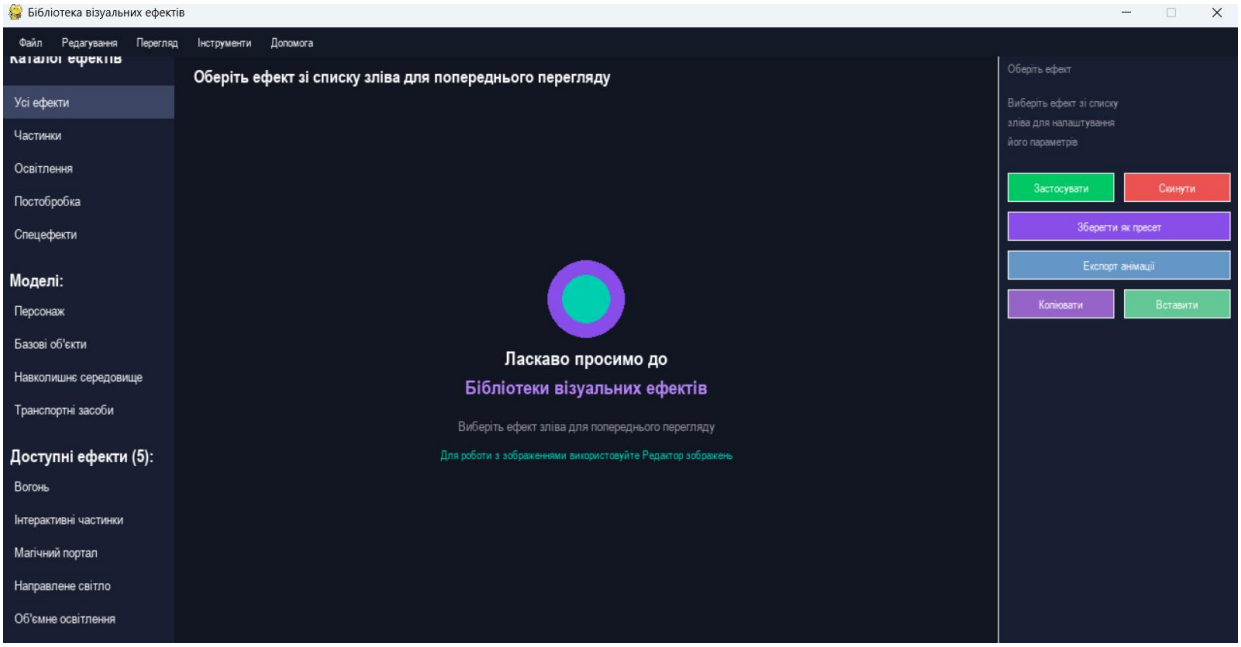


Рисунок 4.2 — Початковий стан модуля створення візуальних ефектів

Створимо ефект вогню. Його стан зі значеннями за замовчуванням наведено на рисунку 4.3. У правій частині міститься панель налаштувань, яка дозволяє задавати інтенсивність, колір, розмір частинок та швидкість руху.

Рисунок 4.3 — Створення ефекту вогню

Змінимо параметри вогню, збільшивши інтенсивність та зменшивши розмір частинок. Результат наведено на рисунку 4.4.

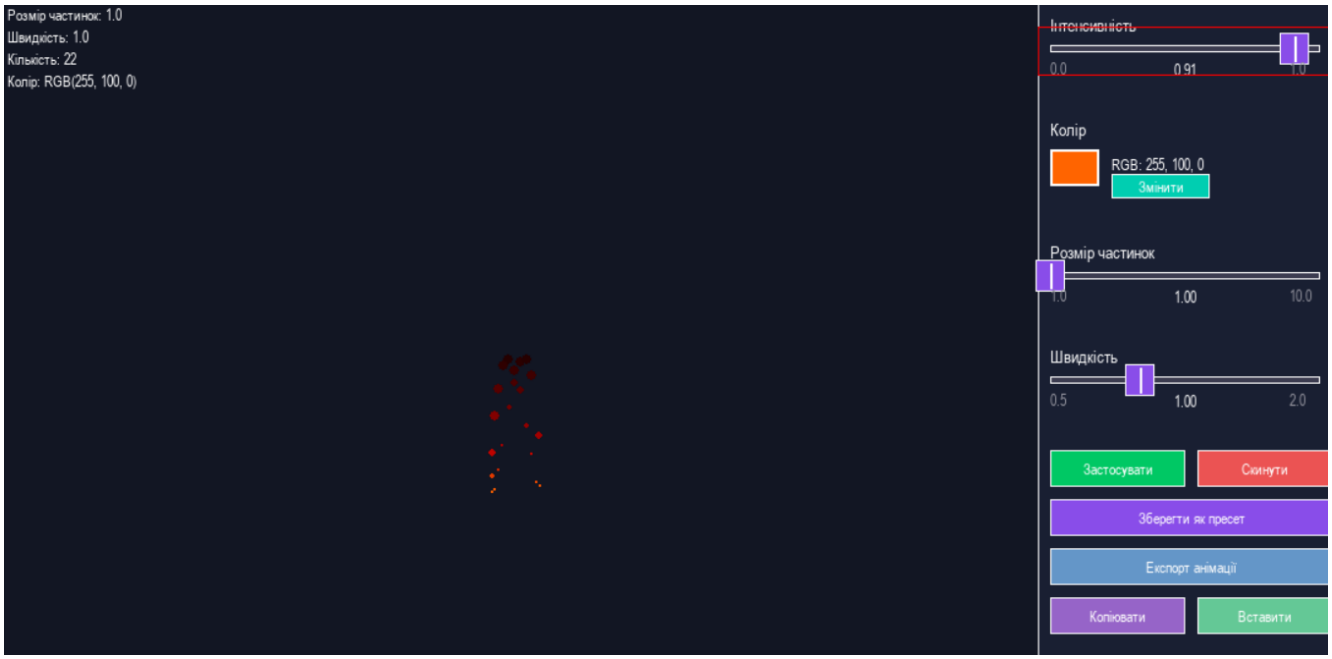


Рисунок 4.4 — Змінені параметри ефекту вогню

Тепер створимо ефект інтерактивних частинок. Цей ефект змінює свою поведінку залежно від руху курсором користувачем. Деякі частинки віддаляються, а деякі рухаються за курсором, створюючи унікальний ефект з рухом частинок. Ефект інтерактивних частинок наведено на рисунку 4.5.

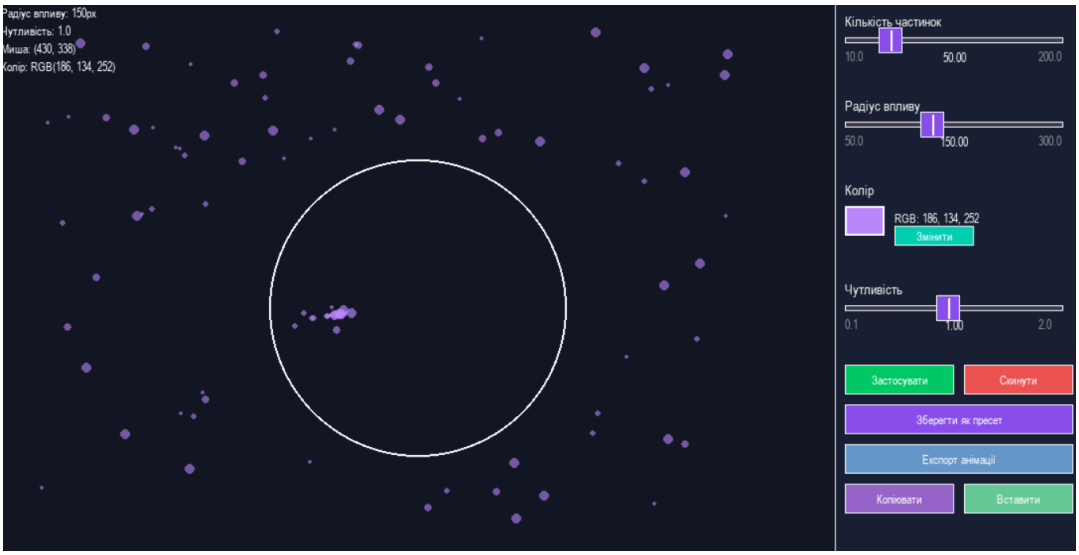


Рисунок 4.5 — Ефект інтерактивних частинок

Збережемо цей ефект, натиснувши на відповідну кнопку у правій частині

екрану. Відповідне сповіщення про збереження ефекту наведено на рисунку 4.6.

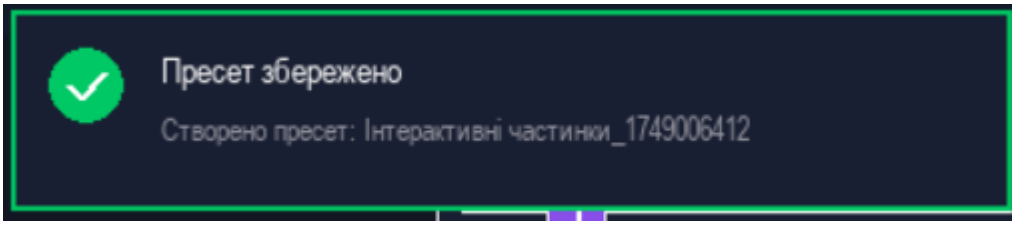


Рисунок 4.6 — Сповіщення про збереження ефекту

На рисунку 4.7 наведено ефект магічного порталу.

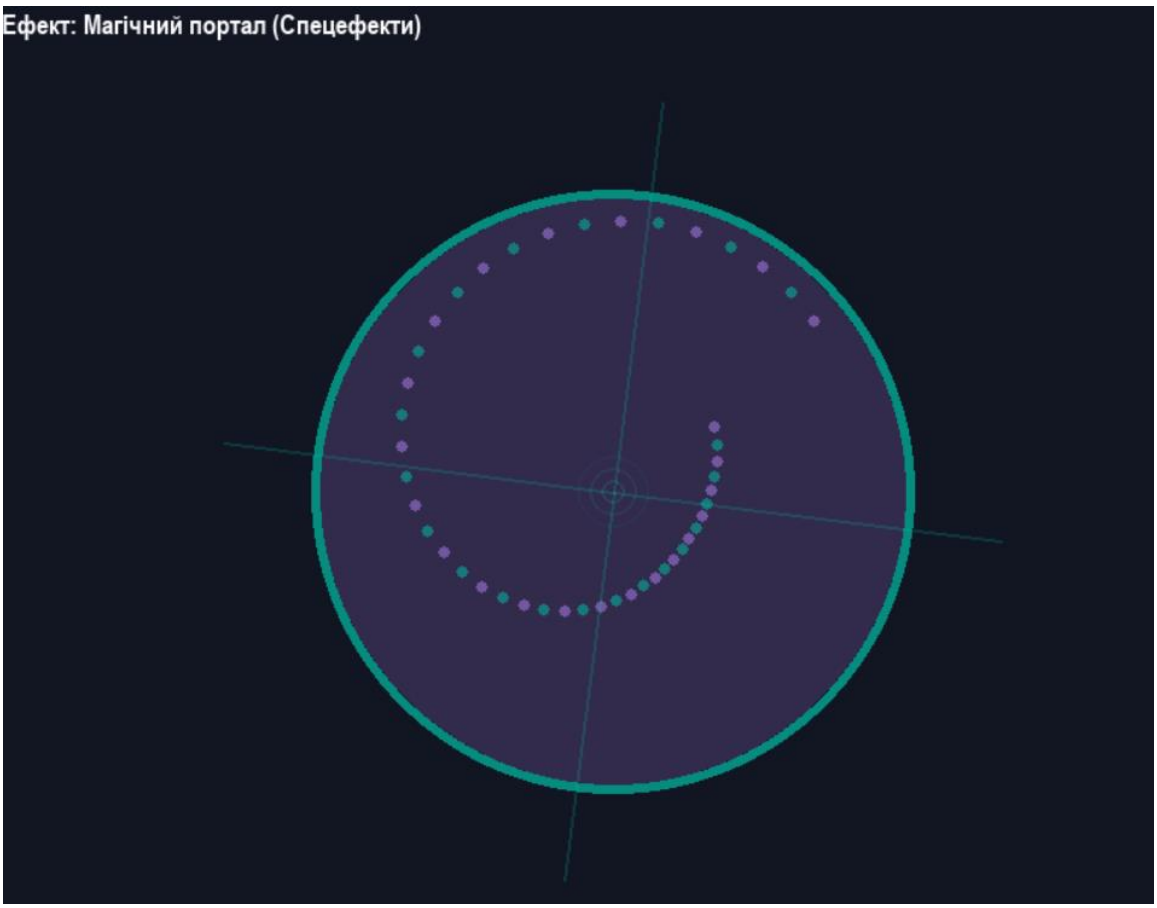


Рисунок 4.7 — Ефект магічного порталу

На рисунку 4.8 наведено модель персонажа з ефектами, який являє собою політ частинок навколо персонажа.

Рисунок 4.8 — Модель персонажа з ефектами

Перейдемо в галерею шаблонів та ефектів. В цьому розділі містять уже готові шаблони (рисунок 4.9).

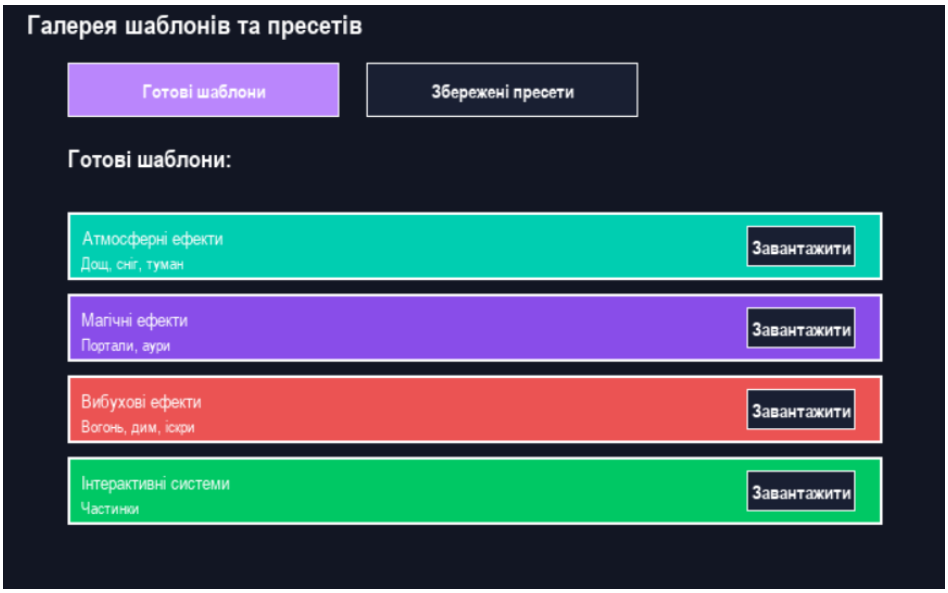


Рисунок 4.9 — Список готових шаблонів візуальних ефектів

Перейдемо у вкладку збережених власних візуальних ефектів (рисунок 4.10).

Тут міститься перелік усіх збережених ефектів, що були створені перед цим з відповідними параметрами.

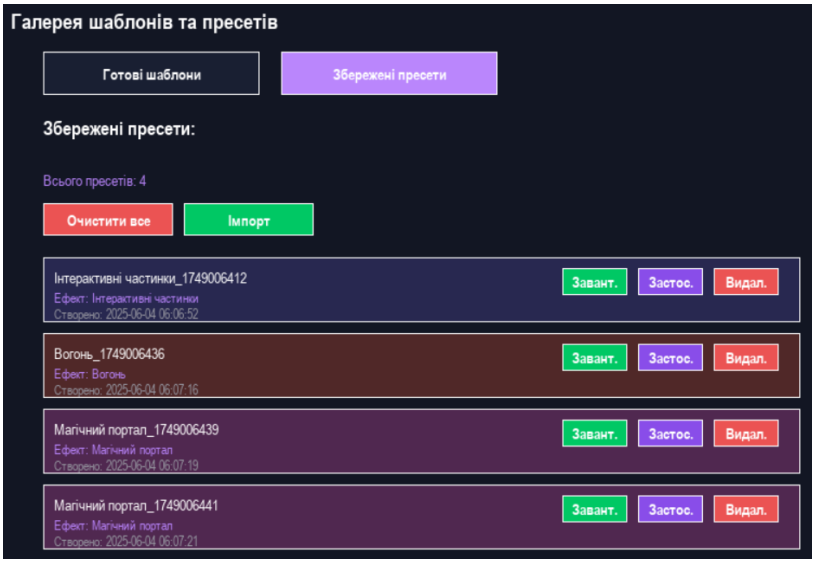


Рисунок 4.10 — Збережені власні візуальні ефекти

Перейдемо в розділ редактора зображень. Спершу необхідно завантажити зображення у редактор за допомогою стандартного меню вибору файлів у Windows. Початковий стан редактора зображень наведено на рисунку 4.11.

Рисунок 4.11 — Початковий стан редактора зображень

Накладемо на це зображення ефект дощу. Особливість цього ефекту дощу

полягає у тому, що краплі не просто пролітають через зображення, а враховують те, що на них зображено, розпізнають об’єкти, і краплі дощу при зіткненні з об’єктом поводять себе, як справжній дощ, «стікаючи» по об’єкту.

З метою тестування, краплям було задано різні кольори залежно від їх поточної поведінки: синій колір — крапля у процесі падіння, червоний колір — крапля вдарилася і відскочила від об’єкта, жовті бризки - утворилися від удару з об’єктом, зелена крапля - почала стікати по схилу об’єкта, сіра крапля - «поглинулася» або осіла.

Перший стан роботи ефекту дощу наведено на рисунку 4.12, другий стан — на рисунку 4.13.

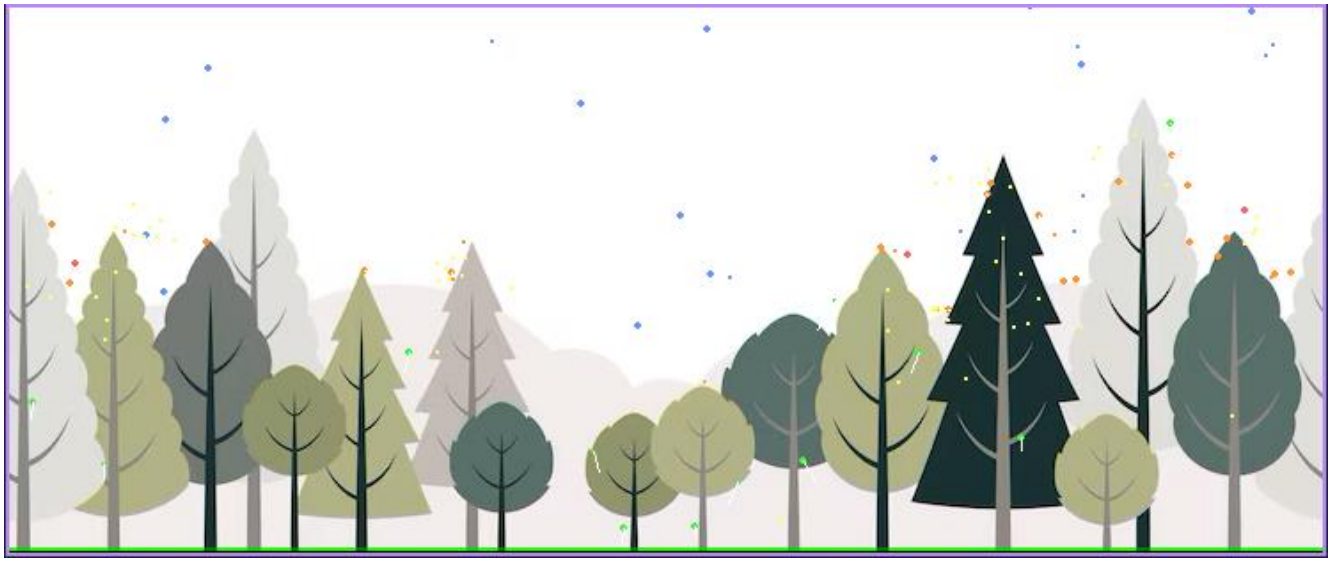


Рисунок 4.12 — Перший стан роботи ефекту дощу

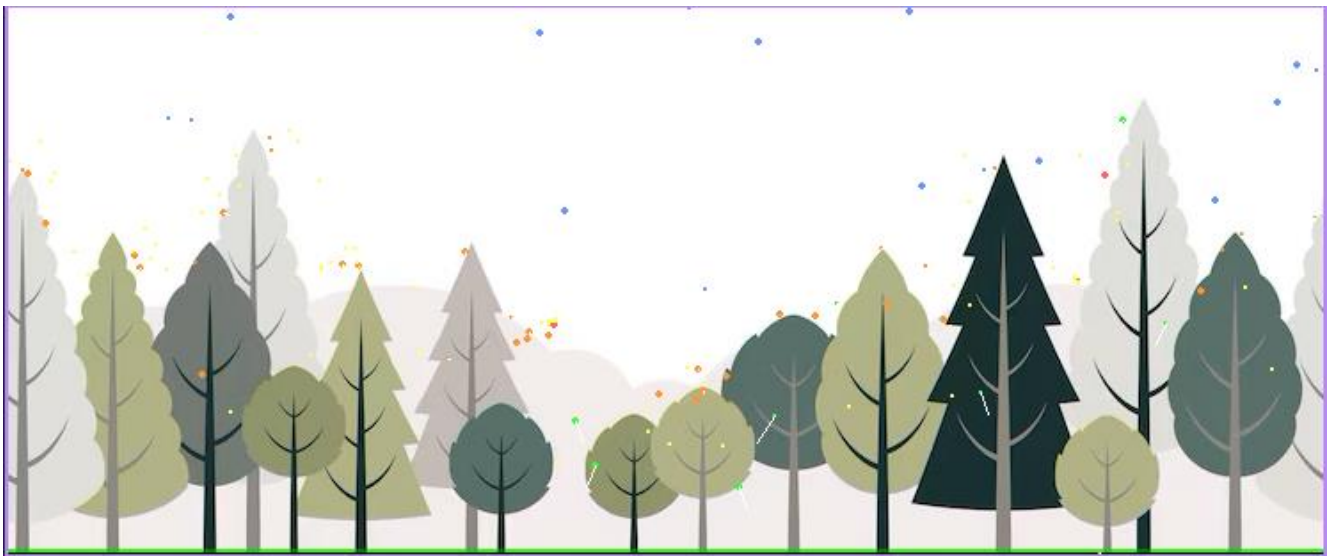


Рисунок 4.13 — Другий стан роботи ефекту дощу

На рисунку 4.14 продемонстровано логи роботи ефекту дощу з описом поведінки крапель.

```
Колізія на (509.1, 141.6)
Поверхня: steep_slope, bounce #3
Стікання по схилу: v=(2.94, 4.44)
Колізія на (153.0, 91.9)
Поверхня: steep_slope, bounce #1
Стікання по схилу: v=(3.69, 5.19)
Колізія на (162.0, 117.8)
Поверхня: flat, bounce #1
Відскок від плоскої поверхні: vy=-2.50
Колізія на (615.0, 109.3)
Поверхня: wall, bounce #2
Відскок від стінки: vx=0.09
Після update: 189 крапель
  Стани: {'bounced': 25, 'spreading': 41, 'falling': 30, 'flowing': 10, 'splash': 83}
=== _render() викликано для view: 'image_editor' ===
Колізія на (88.9, 160.4)
Поверхня: wall, bounce #3
Відскок від стінки: vx=2.59
Колізія на (162.1, 116.6)
Поверхня: wall, bounce #2
Відскок від стінки: vx=0.02
Колізія на (545.1, 112.9)
Поверхня: wall, bounce #1
Відскок від стінки: vx=0.29
Колізія на (615.1, 114.2)
Поверхня: flat, bounce #3
Після update: 186 крапель
```

Рисунок 4.14 — Логи з описом поведінки крапель дощу

Отриману анімацію можна експортувати у форматі .gif. Експортована анімація у застосунку для перегляду зображень у Windows наведена на рисунку 4.15.

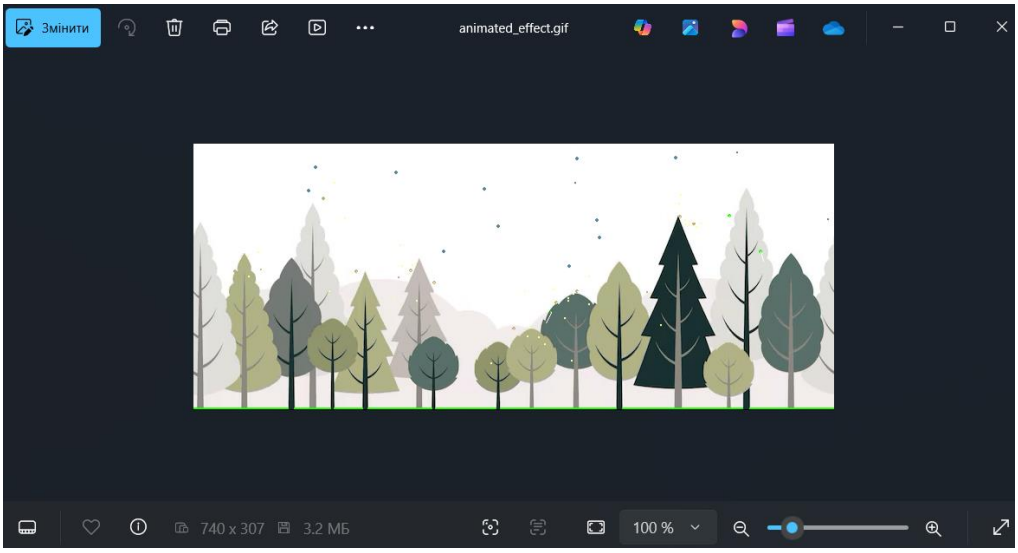


Рисунок 4.15 — Експортована GIF-анімація

У рамках додаткового тестування було проведено оцінку швидкості реакції системи на дії користувача, а також часу, необхідного для автоматизованого виконання антропометричних вимірювань та їх візуалізації у тривимірному просторі. За результатами тестування встановлено, що середній час відгуку інтерфейсу на дії користувача не перевищує 0,5 секунди, що забезпечує оперативний зворотний зв’язок та підвищує зручність взаємодії з програмним забезпеченням.

4.2 Розробка інструкції користувача

Інструкція користувача містить визначення технічних вимог для встановлення та запуску бібліотеки візуальних ефектів. Вимоги до апаратного забезпечення подано в таблиці 4.1. У ній зазначено мінімальні та рекомендовані параметри системи, включаючи тип процесора, обсяг оперативної пам’яті, графічну підсистему, обсяг вільного місця на диску та підтримувану версію операційної системи.

Таблиця 4.1 – Вимоги до апаратного забезпечення для коректної бібліотеки візуальних ефектів

Параметр	Рекомендована конфігурація	Мінімальна конфігурація
Процесор	Intel Core i7	Intel Core i3
Оперативна пам’ять	8 ГБ RAM	4 ГБ RAM
Вільне місце на диску	15 ГБ	15 ГБ
Відеокарта	NVIDIA GeForce GTX 1660 Ti або аналог	NVIDIA GeForce GTX 1050
Операційна система	Windows 10 або macOS Big Sur	Windows 7 або macOS Mojave
Монітор	≥1920x1080, діагональ 21" або більше	≥1280x720, діагональ 15" або більше

Після інсталяції та запуску програмного забезпечення користувач потрапляє на вітальний екран бібліотеки візуальних ефектів. Після завантаження робочого середовища відкривається доступ до основного функціоналу.

Початковий етап роботи складається з імпорту візуального ефекту у внутрішньому форматі бібліотеки .vfx. Після вибору файлу автоматично активується модуль перевірки структури ефекту на відповідність вимогам бібліотеки. У разі виявлення конфігураційних помилок користувач отримує повідомлення з детальною інформацією. Якщо структура є коректною — відображається короткий опис ефекту.

Далі користувач має змогу працювати з елементами головного меню, зокрема «Довідка», «Про програму» або перейти безпосередньо до редагування. Натискання кнопки запуску відкриває редактор візуальних ефектів, де активується модуль параметризації — користувач може змінювати ключові характеристики ефекту: колір, розмір, тривалість, інтенсивність тощо.

Після редагування ефекту користувач може переглянути результат у режимі попереднього візуального відтворення на сцені. За потреби можна активувати вкладку «Композиція ефектів» для поєднання кількох візуальних елементів в один сценарій.

Після завершення налаштувань користувач має змогу зберегти конфігурацію

або експортувати ефект у формат .vfx чи .json для подальшої інтеграції в ігрові рушії (наприклад, Unity або Unreal Engine). Це забезпечує повноцінну підтримку розробки та повторного використання візуальних ефектів у різних проєктах.

4.3 Висновки

У четвертому розділі було проведено тестування програмних модулів бібліотеки візуальних ефектів для ігрових застосунків. Перевірка охоплювала як стандартні сценарії використання, так і типові помилкові ситуації — зокрема, обробку некоректних вхідних даних, файлів у непідтримуваних форматах, а також моделювання некоректних конфігурацій ефектів.

ВИСНОВКИ

У результаті виконання бакалаврської кваліфікаційної роботи була розроблена бібліотека візуальних ефектів, призначена для використання у графічних та ігрових застосунках. Реалізація програмного забезпечення виконувалась із використанням середовища розробки PyCharm. Бакалаврську кваліфікаційну роботу реалізовано згідно методичних вказівок [26].

У процесі виконання роботи було проведено детальний аналіз поточного стану технологій у сфері візуальних ефектів. Розглянуто сучасні аналоги програмних засобів, визначено їх переваги та недоліки, а також здійснено порівняння з власною розробкою. На основі результатів порівняння сформульовано основні задачі, вирішення яких реалізовано в межах проєкту.

Під час аналізу технологій програмної реалізації було обґрунтовано вибір мов програмування Python.

У першому розділі бакалаврської квалфікаційної роботи розглянуто поточний стан технологій у сфері візуальних ефектів, важливість підвищення продуктивності та задачі, пов'язані з традиційними методами. У розділі також проведено порівняння запропонованого рішення з існуючими системами та окреслено завдання проєкту.

У другому розділі надано детальний опис алгоритмів програми. Також описано вхідні та вихідні дані системи, а також метод накладання динамічних ефектів на статичне зображення, метод системи інтерактивних частинок, метод багатопроцесорного експорту анімацій з адаптивним розподілом ресурсів.

У третьому розділі було проведено порівняльний аналіз мов програмування розробки бібліотеки візуальних елементів та обрано мову Python. Було розроблено модуль обробки зображень, модуль керування пресетами і переходами, модуль інтерфейсу користувача.

У четвертому розділі проведено тестування функціональних та графічних компонентів програмної бібліотеки, розробленої в рамках бакалаврської кваліфікаційної роботи.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Shreiner D., Sellers G., Kessenich J., Licea-Kane B. OpenGL Programming Guide (The Red Book). 9th Edition. Addison-Wesley, 2016.

2. Lengyel E. Mathematics for 3D Game Programming and Computer Graphics. 4th Edition. Cengage Learning, 2021.

3. Watt A., Policarpo F. Advanced Game Programming: A Programmer's Guide to 3D Graphics. CRC Press, 2014.

4. Engel W. (Ed.). GPU Pro: Advanced Rendering Techniques. A.K. Peters, 2010.

5. Гаврилюк В.П., Ткаченко О.М. Розробка бібліотеки візуальних ефектів для графі-чних та ігрових застосунків / В.П. Гаврилюк, О.М. Ткаченко // Матеріали Міжнародної науково-практичної інтернет-конференції «Молодь в науці: дослідження, проблеми,перспективи (МН-2025)». – Вінниця: ВНТУ, 2025 [Електронний ресурс] – Режим доступу: https://conferences.vntu.edu.ua/index.php/mn/mn2025/paper/view/24944

6. Гаврилюк В.П., Ткаченко О.М. Розробка методу багатопроцесорного експорту анімацій з адаптивним розподілом ресурсів / В.П. Гаврилюк, О.М. Ткаченко // Матеріали Міжнародної науково-практичної інтернет-конференції «Молодь в науці: дослідження, проблеми,перспективи (МН-2025)». – Вінниця: ВНТУ, 2025 [Електронний ресурс] – Режим доступу: https://conferences.vntu.edu.ua/index.php/mn/mn2025/paper/view/25440

7. Гаврилюк В.П., Ткаченко О.М. Розробка методу накладання динамічних ефектів на статичне зображення з інтелектуальним аналізом сцени / В.П. Гаврилюк, О.М. Ткаченко // Матеріали Міжнародної науково-практичної інтернет-конференції «Молодь в науці: дослідження, проблеми,перспективи (МН-2025)». – Вінниця: ВНТУ, 2025 [Електронний ресурс] – Режим доступу: https://conferences.vntu.edu.ua/index.php/mn/mn2025/paper/view/25441

8. Unity Technologies. Visual Effect Graph Documentation [Електронний ресурс] – Режим доступу: https://docs.unity3d.com/Manual/VisualEffectGraph.html

9. Epic Games. Niagara Visual Effects System Documentation [Електронний

ресурс] – Режим доступу: https://docs.unrealengine.com/en-US/niagara/index.html

10. PopcornFX. Real-Time Particle FX Middleware [Електронний ресурс] – Режим доступу: https://www.popcornfx.com

11. ShaderToy. GLSL Online Shader Editor and Community [Електронний ресурс] – Режим доступу: https://www.shadertoy.com

12. Khronos Group. OpenGL 4.6 Specification [Електронний ресурс] – Режим доступу: https://www.khronos.org/registry/OpenGL/specs/gl/glspec46.core.pdf

13. NVIDIA. NVIDIA Nsight Graphics [Електронний ресурс] – Режим доступу: https://developer.nvidia.com/nsight-graphics

14. "VFX Tools that are Used in Games" [Електронний ресурс] – Режим доступу: https://www.mad-vfx.com/blogs/vfx-tools-that-are-used-in-games?

15. RenderDoc. Graphics Debugger [Електронний ресурс] – Режим доступу: https://renderdoc.org

16. Microsoft. Visual Studio Documentation [Електронний ресурс] – Режим доступу: https://learn.microsoft.com/en-us/visualstudio/

17. Romaniuk O.N et al. Method of procedural texturing. Innovative research and prospects for the development of science and technology in the XXI century. Rivne, 2023.

18. Лучано Рамальо. Fluent Python. Clear, Concise, and Effective Programming. Farnham: O'Reilly Media, 2024. 1012c.

19. Marius Bancila. Modern C++ Programming Cookbook: Master Modern C++ with comprehensive solutions for C++23 and all previous standards. Birmingham: Packt Publishing, 2024. 816c.

20. David Flanagan. JavaScript: The Definitive Guide: Master the World's Most-Used Programming Language 7th Edition. Farnham: O'Reilly Media, 2020. 704c.

21. Romaniuk O.N et al. Interrelation between the vectors of normal to the surface, observer vector and light source vector for rendering tasks. Kharkiv: Sciconf, 2022.

22. Eberly D.H. 3D Game Engine Architecture: Engineering Real-Time Applications with Wild Magic. Morgan Kaufmann, 2006.

23. Pharr M., Jakob W., Humphreys G. Physically Based Rendering: From Theory to Implementation. 3rd ed. Morgan Kaufmann, 2016.

24. Unreal Engine Developer Resources [Електронний ресурс] – Режим доступу: https://dev.epicgames.com/

25. Unity Learn Platform [Електронний ресурс] – Режим доступу: https://learn.unity.com/

26. OpenGL SuperBible. Comprehensive Tutorial and Reference. 7th ed. Pearson Education, 2015.

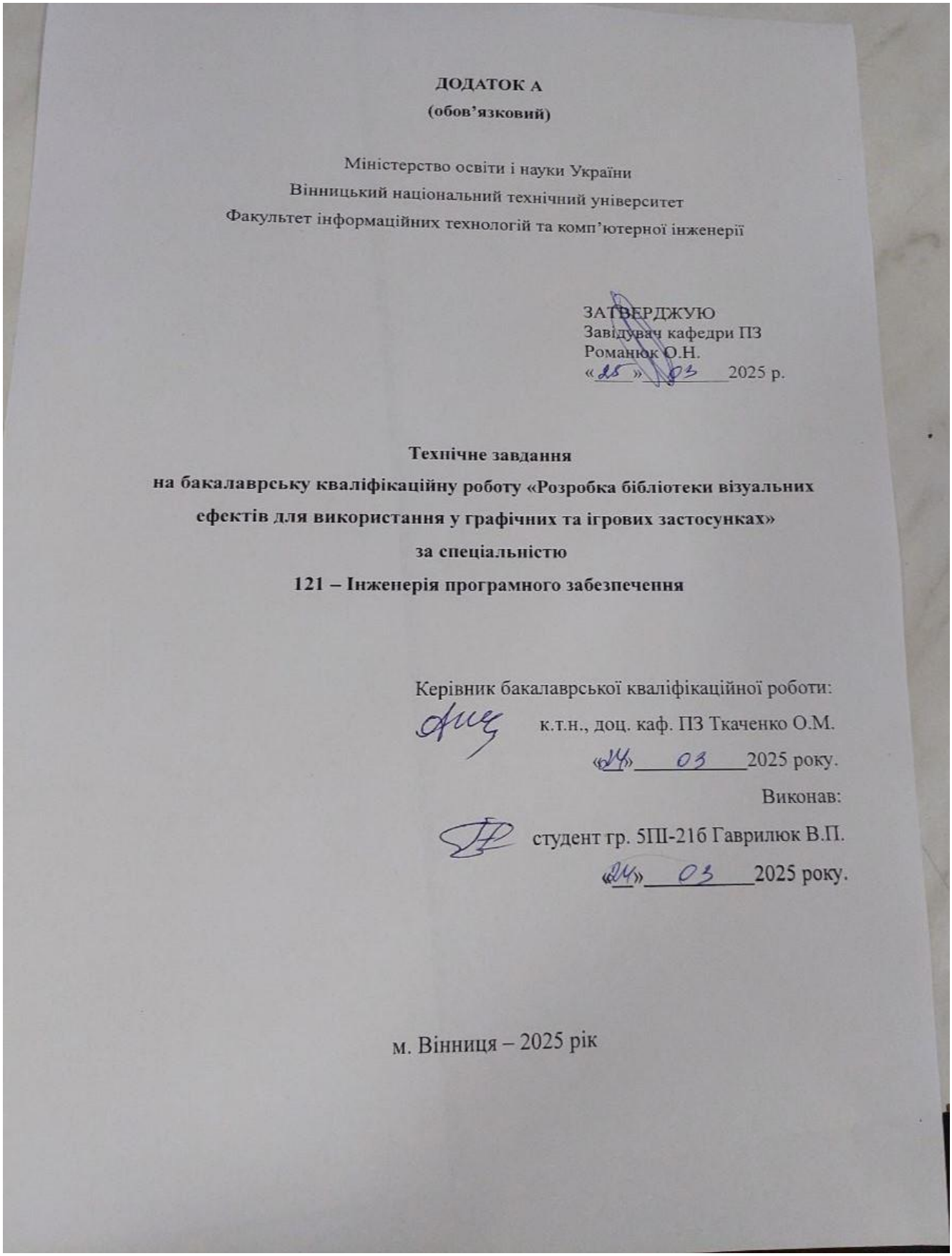
27. ISO/IEC/IEEE 29119-2:2013 – Software Testing – Part 2: Test Processes.

28. Методичні вказівки до виконання бакалаврських кваліфікаційних робіт для студентів спеціальності 121 "Інженерія програмного забезпечення" / Уклад. О. Н. Романюк, Г. О. Черноволик – Вінниця: ВНТУ, 2022. – 52с.

ДОДАТКИ

ДОДАТОК А

(обов'язковий)



ДОДАТОК А
(обов'язковий)

Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
Романюк О.Н.
«25» 03 2025 р.

Технічне завдання
на бакалаврську кваліфікаційну роботу «Розробка бібліотеки візуальних ефектів для використання у графічних та ігрових застосунках»
за спеціальністю
121 – Інженерія програмного забезпечення

Керівник бакалаврської кваліфікаційної роботи:
к.т.н., доц. каф. ПЗ Ткаченко О.М.
«24» 03 2025 року.

Виконав:
студент гр. 5ПІ-21б Гаврилюк В.П.
«24» 03 2025 року.

м. Вінниця – 2025 рік

1. Найменування та галузь застосування

Бакалаврська кваліфікаційна робота: «Розробка бібліотеки візуальних ефектів для використання у графічних та ігрових застосунках».

Галузь застосування – десктоп-технології.

2. Підстава для розробки.

Завдання на роботу, яке затверджене на засіданні кафедри програмного забезпечення – протокол № 18 від 18 лютого 2025 р.

3. Мета та призначення розробки.

Метою роботи є підвищення продуктивності обчислень за рахунок раціонального розподілу задач між доступними процесорними ядрами, а також реалістичності за рахунок динамічного відтворення змін на зображенні.

4. Вихідні дані для проведення НДР

1. Shreiner D., Sellers G., Kessenich J., Licea-Kane B. OpenGL Programming Guide (The Red Book). 9th Edition. Addison-Wesley, 2016.

2. PopcornFX. Real-Time Particle FX Middleware [Електронний ресурс] – Режим доступу: https://www.popcornfx.com

3. Лучано Рамальо. Fluent Python. Clear, Concise, and Effective Programming. Farnham: O'Reilly Media, 2024. 1012c.

4. OpenGL SuperBible. Comprehensive Tutorial and Reference. 7th ed. Pearson Education, 2015.

5. Технічні вимоги

Вхідні дані — базові графічні ресурси, параметри налаштування ефектів.

Вихідні дані — візуальні ефекти, анімації.

6. Конструктивні вимоги.

Інтерфейс програми поовинен відповідати естетичним та ергономічним вимогам, повинна бути зручною в обслуговуванні та керуванні.

Графічна та текстова документація повинна відповідати діючим стандартам України.

7. Перелік технічної документації, що пред'являється по закінченню робіт:

–пояснювальна записка до бакалаврської кваліфікаційної роботи;

–технічне завдання;

–лістинги програми.

8. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

9. Стадії та етапи розробки

№ з/п	Назва етапів бакалаврської кваліфікаційної роботи	Строк виконання етапів роботи
1	Аналіз стану технологій візуальних ефектів для ігрових застосунків	25.03.25 - 05.04.25
2	Розробка методів та алгоритмів програмного застосунку	06.04.25 - 18.04.25
3	Варіантний аналіз та вибір мови програмування розробки	19.04.25 - 21.04.25
4	Розробка застосунку	22.04.25 - 17.05.25
5	Тестування застосунку	18.05.25 - 25.05.25
6	Оформлення матеріалів до захисту БКР	26.05.25 - 30.05.25

10. Порядок контролю та прийняття.

Усі етапи бакалаврської роботи контролюються науковим керівником згідно плану по виконанню роботи. Прийняття бакалаврської кваліфікаційної роботи здійснюється ДЕК, затвердженою зав. кафедрою згідно з графіком захисту.

ДОДАТОК Б

(обов'язковий)

ДОДАТОК Б

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Назва роботи: Розробка бібліотеки візуальних ефектів для використання у графічних та ігрових застосунках

Тип роботи: Бакалаврська кваліфікаційна робота
(бакалаврська кваліфікаційна робота / магістерська кваліфікаційна робота)

Підрозділ кафедра програмного забезпечення, ФІТКІ
(кафедра, факультет, навчальна група)

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism 5,9 %

Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне)

■ Запозичення, виявлені у роботі, є законними і не містять ознак плагіату, фабрикації, фальсифікації. Роботу прийняти до захисту

☐ У роботі не виявлено ознак плагіату, фабрикації, фальсифікації, але надмірна кількість текстових запозичень та/або наявність типових розрахунків не дозволяють прийняти рішення про оригінальність та самостійність її виконання. Роботу направити на доопрацювання.

☐ У роботі виявлено ознаки плагіату та/або текстових маніпуляцій як спроб укриття плагіату, фабрикації, фальсифікації, що суперечить вимогам законодавства та нормам академічної доброчесності. Робота до захисту не приймається.

Експертна комісія:

(прізвище, ініціали, посада) (підпис)

(прізвище, ініціали, посада) (підпис)

Особа, відповідальна за перевірку Черноволик Г. О.

З висновком експертної комісії ознайомлений(-на)

Керівник (підпис) Ткаченко О.М. (прізвище, ініціали, посада)

Здобувач (підпис) Гаврилюк В.П. (прізвище, ініціали)

ДОДАТОК В
(довідниковий)

```
class VFXEditor:
    """Головний клас редактора візуальних ефектів"""

    def __init__(self, screen_width=1280, screen_height=720):
        """Ініціалізація редактора візуальних ефектів"""
        print("=== ІНІЦІАЛІЗАЦІЯ VFXEditor ===")

        # Ініціалізація pygame
        pygame.init()
        pygame.font.init()

        # ОБОВ'ЯЗКОВО встановлюємо ці змінні ПЕРШИМИ
        self.has_loaded_image = False
        self.loaded_image_path = None

        print(f"Початкові значення: has_loaded_image={self.has_loaded_image}")

        # Налаштування вікна
        self.screen_width = screen_width
        self.screen_height = screen_height
        self.screen = pygame.display.set_mode((screen_width, screen_height))
        pygame.display.set_caption("Бібліотека візуальних ефектів")

        # Завантаження шрифтів
        self.fonts = {
            'title': pygame.font.SysFont('Arial', 20, bold=True),
            'subtitle': pygame.font.SysFont('Arial', 18, bold=True),
            'regular': pygame.font.SysFont('Arial', 14),
            'small': pygame.font.SysFont('Arial', 12),
            'button': pygame.font.SysFont('Arial', 14, bold=True)
        }

        # Стан програми
        self.running = True
        self.current_view = 'main_menu'
        self.selected_category = 'Усі ефекти'
        self.selected_effect = None
        self.effects_data = self._load_effects_data()

        self.dragging_slider = None
        self.property_controls = {}
        self.action_buttons = {}
        self.dropdown_open = None
        self.copied_settings = None
        self.saved_presets = {}

        # Категорії ефектів
```

```
self.effect_categories = [
    'Усі ефекти',
    'Частинки',
    'Освітлення',
    'Постобробка',
    'Спецефекти'
]

# Доступні ефекти
self.available_effects = [
    'Вогонь',
    'Інтерактивні частинки',
    'Магічний портал',
    'Направлене світло',
    "Об'ємне освітлення"
]

# Моделі (окремо від ефектів)
self.model_categories = [
    'Персонаж',
    'Базові об\'єкти',
    'Навколишнє середовище',
    'Транспортні засоби'
]

# Поточний режим (ефекти або моделі)
self.current_mode = 'effects'  # 'effects' або 'models'

# Області інтерфейсу
self.sidebar_width = 180
self.properties_width = 250
self.main_area = pygame.Rect(
    self.sidebar_width, 30,
    self.screen_width - self.sidebar_width - self.properties_width,
    self.screen_height - 60
)

# Повідомлення
self.error_message = None
self.notification = None
self.dropdown_menu = None  # Для випадаючих меню

# Інформація про продуктивність
self.performance_info = {
    'fps': 60,
    'cpu': 12,
    'gpu': 24,
    'ram': 256
}

# Система інтерактивних частинок
self.interactive_particles = InteractiveParticleSystem()
```

```
# Налаштування експорту
self.export_settings = {
    'format': 'gif',
    'duration': 3.0,
    'fps': 30,
    'output_path': './exported_animation.gif'
}

# Стан слайдерів для кожного ефекту
self.dragging_slider = None
self.slider_states = {}

# Ініціалізація PixelFX
self.pixel_fx = PixelFX(use_multithreading=True, use_gpu=True)

# Створення розширеного PixelFX
self.enhanced_pixel_fx = EnhancedPixelFX(
    pixel_fx_instance=self.pixel_fx,
    use_multithreading=True,
    max_workers=4
)

# Завантаження розширених пресетів
self._load_enhanced_presets()
self.effect_manager = EffectManager(self.enhanced_pixel_fx)

# Завантаження базових пресетів
self._load_default_presets()

# Ініціалізація фільтрованих ефектів
self.filtered_effects = self.available_effects.copy()

# Ініціалізація UI
self._init_ui()

# Створення ImageEditor ПІСЛЯ ініціалізації всіх необхідних атрибутів
try:
    print("Створення ImageEditor...")
    self.image_editor = ImageEditor(self)
    print("ImageEditor створено успішно")
except Exception as e:
    print(f"КРИТИЧНА ПОМИЛКА створення ImageEditor: {e}")
    import traceback
    traceback.print_exc()
    self.image_editor = None

if hasattr(self, 'image_editor') and self.image_editor:
    # Інтеграція інтелектуальних ефектів
    integrate_fixed_image_editor(self)

    # Додаємо багатопроцесорний експорт
    integrate_advanced_multiprocessor_export(self)
```

```
        print("□ Інтелектуальні ефекти та розширений експорт інтегровано!")

    # Додаткові методи завантаження для сумісності
    self.main_menu_buttons = [
        {
            'text': '□ Редактор зображень',
            'icon': '□',
            'action': lambda: self.load_image_dialog()
        },
        {
            'text': 'Створити новий ефект',
            'icon': '+',
            'action': lambda: self.set_view('editor')
        },
        {
            'text': 'Відкрити проект',
            'icon': 'O',
            'action': lambda: self.set_view('file_browser')
        },
        {
            'text': 'Галерея шаблонів',
            'icon': 'G',
            'action': lambda: self.set_view('templates')
        },
        {
            'text': 'Налаштування',
            'icon': 'S',
            'action': lambda: self.set_view('settings')
        }
    ]

    print("=== Ініціалізація VFXEditor завершена ===")

def _load_enhanced_presets(self):
    """Завантаження розширених пресетів згідно з документацією"""

    # Пресет "Магічні ефекти" (з документації модулі.docx)
    magic_effects = []
    if hasattr(self, 'interactive_particles'):
        magic_effects.append(self.interactive_particles)

    self.enhanced_pixel_fx.create_preset_enhanced(
        "Магічні ефекти",
        magic_effects,
        "Інтерактивні частинки та магічні портали з плавними переходами",
        "Спецефекти"
    )

    # Пресет "Атмосферні ефекти"
    atmospheric_effects = []
    self.enhanced_pixel_fx.create_preset_enhanced(
        "Атмосферні ефекти",
        atmospheric_effects,
```

```
        "Дощ, сніг, туман та інші погодні ефекти з реалістичною анімацією",
        "Природні"
    )

    # Пресет "Світлові ефекти"
    light_effects = []
    self.enhanced_pixel_fx.create_preset_enhanced(
        "Світлові ефекти",
        light_effects,
        "Направлене світло та об'ємне освітлення з динамічними тінями",
        "Освітлення"
    )

    print("Завантажено розширені пресети згідно з документацією")

def apply_preset_with_transition(self, preset_name: str, transition_type: str = 'fade'):
    """Застосування пресету з анімованим переходом (з документації)"""
    print(f"Застосування пресету '{preset_name}' з переходом '{transition_type}'")

    # Перевіряємо, чи завантажено зображення
    if not self.has_loaded_image:
        self.error_message = {
            'title': 'Потрібно зображення',
            'message': 'Спочатку завантажте зображення для застосування ефектів'
        }
        return

    success = self.enhanced_pixel_fx.apply_preset_with_transition(
        preset_name,
        transition_type,
        transition_duration=1500  # 1.5 секунди згідно з документацією
    )

    if success:
        self.notification = {
            'title': 'Пресет застосовано',
            'message': f'Активовано "{preset_name}" з переходом {transition_type}'
        }

        # Встановлюємо відповідний ефект як обраний
        if preset_name == "Магічні ефекти":
            self.selected_effect = "Інтерактивні частинки"
        elif preset_name == "Атмосферні ефекти":
            self.selected_effect = "Вогонь"
        elif preset_name == "Світлові ефекти":
            self.selected_effect = "Направлене світло"

    else:
        self.error_message = {
            'title': 'Помилка',
            'message': f'Не вдалося застосувати пресет "{preset_name}"'
        }
```

```
# Додайте новий метод для завантаження зображень
def load_image_enhanced(self, image_path):
    """Завантаження зображення в розширений PixelFX"""
    # Встановлюємо поверхню для рендерингу
    buffer_surface = pygame.Surface((self.main_area.width, self.main_area.height))
    self.enhanced_pixel_fx.set_surface(buffer_surface)

    # Завантажуємо зображення
    success = self.enhanced_pixel_fx.load_image(
        image_path,
        resize_to_fit=True,
        max_size=(self.main_area.width, self.main_area.height)
    )

    if success:
        # Також завантажуємо в старий PixelFX для сумісності
        self.pixel_fx.load_image(image_path, resize_to_fit=True, max_size=(1920, 1080))

        self.notification = {
            'title': 'Зображення завантажено',
            'message': f'Файл завантажено успішно'
        }
        return True
    else:
        self.error_message = {
            'title': 'Помилка завантаження',
            'message': 'Не вдалося завантажити зображення'
        }
        return False

# Модифікуйте метод apply_preset_with_transition
def apply_preset_with_transition(self, preset_name: str, transition_type: str = 'fade'):
    """Застосування пресету з анімованим переходом"""
    success = self.enhanced_pixel_fx.apply_preset_with_transition(
        preset_name,
        transition_type,
        transition_duration=1500  # 1.5 секунди
    )

    if success:
        self.notification = {
            'title': 'Пресет застосовано',
            'message': f'Активовано "{preset_name}" з переходом {transition_type}'
        }
    else:
        self.error_message = {
            'title': 'Помилка',
            'message': f'Не вдалося застосувати пресет "{preset_name}"'
        }

# Додайте методи в меню
def show_presets_menu(self):
    """Показати меню пресетів"""
```

```
    self.current_view = 'presets_menu'

def show_transitions_menu(self):
    """Показати меню переходів"""
    self.current_view = 'transitions_menu'

def _render_main_area(self):
    """Оновлене відображення головної області з усіма моделями"""
    # Фон головної області
    pygame.draw.rect(self.screen, DARK_BLUE, self.main_area)

    # Заголовок
    title = self._get_main_area_title()
    title_text = self.fonts['subtitle'].render(title, True, WHITE)
    self.screen.blit(title_text, (self.sidebar_width + 20, 40))

    if self.selected_effect:
        # Відображаємо обраний ефект або модель
        if self.current_mode == 'effects':
            # Ефекти
            if self.selected_effect == "Інтерактивні частинки":
                self._render_interactive_particles()
            elif self.selected_effect == "Вогонь":
                self._render_fire_effect()
            elif self.selected_effect == "Магічний портал":
                self._render_magic_portal_effect()
            elif self.selected_effect == "Направлене світло":
                self._render_directional_light_effect()
            else:
                self._render_welcome_screen()

        elif self.current_mode == 'models':
            # Моделі
            if self.selected_effect == 'Персонаж':
                self._render_character_model()
            elif self.selected_effect == 'Базові об\'єкти':
                self._render_basic_objects_model()
            elif self.selected_effect == 'Навколишнє середовище':
                self._render_environment_model()
            elif self.selected_effect == 'Транспортні засоби':
                self._render_vehicle_model()
            else:
                self._render_welcome_screen()
    else:
        self._render_welcome_screen()

def _get_main_area_title(self):
    """Оновлений заголовок для головної області"""
    if self.selected_effect:
        if self.current_mode == 'models':
            model_descriptions = {
                'Персонаж': "Енергетичний персонаж з аурою",
                'Базові об\'єкти': "Геометричні об'єкти з частинками",
```

```
            'Навколишнє середовище': "Ландшафт з атмосферними ефектами",
            'Транспортні засоби': "Транспорт з реалістичними ефектами"
        }
        description = model_descriptions.get(self.selected_effect, "Модель з ефектами")
        return f"Модель: {self.selected_effect} - {description}"
    else:
        effect_category = "Невідома категорія"
        for category, effects in self.effects_data.items():
            if self.selected_effect in effects:
                effect_category = category
                break
        return f"Ефект: {self.selected_effect} ({effect_category})"
else:
    return "Оберіть ефект або модель зі списку зліва для попереднього перегляду"

def _render_basic_objects_model(self):
    """Відображення базових об'єктів з ефектами"""
    center_x = self.sidebar_width + self.main_area.width // 2
    center_y = self.screen_height // 2

    # Отримання параметрів з налаштувань
    if 'Моделі' in self.effects_data and 'Базові об\'єкти' in self.effects_data['Моделі']:
        params = self.effects_data['Моделі']['Базові об\'єкти']['params']
        object_count = int(params['кількість об\'єктів']['value'])
        object_size = params['розмір']['value']
    else:
        object_count = 3
        object_size = 1.0

    current_time = pygame.time.get_ticks() / 1000

    # Базові об'єкти: куб, сфера, піраміда
    objects = ['cube', 'sphere', 'pyramid']
    colors = [(100, 150, 255), (255, 150, 100), (150, 255, 100)]

    for i in range(min(object_count, len(objects))):
        obj_type = objects[i % len(objects)]
        obj_color = colors[i % len(colors)]

        # Позиція об'єкта
        angle = (i * 120) + current_time * 20  # Повільне обертання
        distance = 100
        obj_x = center_x + distance * np.cos(np.radians(angle))
        obj_y = center_y + distance * np.sin(np.radians(angle))

        # Розмір об'єкта
        base_size = int(40 * object_size)
        pulse_size = int(base_size + 10 * np.sin(current_time * 3 + i))

        if obj_type == 'cube':
            # Куб (квадрат з ефектом глибини)
            cube_rect = pygame.Rect(obj_x - pulse_size // 2, obj_y - pulse_size // 2, pulse_size, pulse_size)
            pygame.draw.rect(self.screen, obj_color, cube_rect)
```

```
            # Ефект глибини
            shadow_rect = pygame.Rect(obj_x - pulse_size // 2 + 5, obj_y - pulse_size // 2 + 5, pulse_size,
                                      pulse_size)
            shadow_color = tuple(c // 2 for c in obj_color)
            pygame.draw.rect(self.screen, shadow_color, shadow_rect)
            pygame.draw.rect(self.screen, obj_color, cube_rect)
            pygame.draw.rect(self.screen, WHITE, cube_rect, 2)

        elif obj_type == 'sphere':
            # Сфера з градієнтним ефектом
            pygame.draw.circle(self.screen, obj_color, (int(obj_x), int(obj_y)), pulse_size)

            # Внутрішнє світло
            light_color = tuple(min(255, c + 80) for c in obj_color)
            pygame.draw.circle(self.screen, light_color,
                               (int(obj_x - pulse_size // 3), int(obj_y - pulse_size // 3)), pulse_size // 2)
            pygame.draw.circle(self.screen, WHITE, (int(obj_x), int(obj_y)), pulse_size, 2)

        elif obj_type == 'pyramid':
            # Піраміда (трикутник)
            points = [
                (obj_x, obj_y - pulse_size),  # Верх
                (obj_x - pulse_size, obj_y + pulse_size // 2),  # Лівий низ
                (obj_x + pulse_size, obj_y + pulse_size // 2)  # Правий низ
            ]
            pygame.draw.polygon(self.screen, obj_color, points)
            pygame.draw.polygon(self.screen, WHITE, points, 2)

        # Енергетичні частинки навколо об'єктів
        for j in range(5):
            particle_angle = current_time * 100 + i * 50 + j * 72
            particle_distance = pulse_size + 20 + 10 * np.sin(current_time * 2 + j)
            particle_x = obj_x + particle_distance * np.cos(np.radians(particle_angle))
            particle_y = obj_y + particle_distance * np.sin(np.radians(particle_angle))

            particle_color = tuple(min(255, c + 50) for c in obj_color)
            pygame.draw.circle(self.screen, particle_color, (int(particle_x), int(particle_y)), 3)

    # Інформація про модель
    info_lines = [
        f"Об'єктів: {object_count}",
        f"Масштаб: {object_size:.1f}x",
        f"Тип: Базові геометричні форми",
        f"Ефекти: Енергетичні частинки"
    ]

    for i, line in enumerate(info_lines):
        info_text = self.fonts['small'].render(line, True, WHITE)
        self.screen.blit(info_text, (self.sidebar_width + 10, 45 + i * 18))

def _render_environment_model(self):
    """Відображення навколишнього середовища"""
```

```
center_x = self.sidebar_width + self.main_area.width // 2
center_y = self.screen_height // 2

# Отримання параметрів
if 'Моделі' in self.effects_data and 'Навколишнє середовище' in self.effects_data['Моделі']:
    params = self.effects_data['Моделі']['Навколишнє середовище']['params']
    atmosphere = params['атмосфера']['value']
else:
    atmosphere = 'Денна'

current_time = pygame.time.get_ticks() / 1000

# Фон залежно від атмосфери
if atmosphere == 'Денна':
    # Денне небо з хмарами
    sky_color = (135, 206, 235)  # Блакитний
    cloud_color = (255, 255, 255, 180)
    sun_color = (255, 255, 0)

    # Сонце
    sun_x = center_x + 200
    sun_y = center_y - 150
    sun_size = 30 + 5 * np.sin(current_time)

    # Сонячне сяйво
    for i in range(3):
        glow_size = sun_size + (i + 1) * 15
        glow_alpha = 100 - i * 30
        glow_surface = pygame.Surface((glow_size * 2, glow_size * 2), pygame.SRCALPHA)
        glow_color = (*sun_color[:3], glow_alpha)
        pygame.draw.circle(glow_surface, glow_color, (glow_size, glow_size), glow_size)
        self.screen.blit(glow_surface, (sun_x - glow_size, sun_y - glow_size))

    pygame.draw.circle(self.screen, sun_color, (int(sun_x), int(sun_y)), int(sun_size))

    # Хмари
    cloud_positions = [(center_x - 150, center_y - 100), (center_x + 100, center_y - 120),
                       (center_x - 50, center_y - 80)]
    for i, (cloud_x, cloud_y) in enumerate(cloud_positions):
        cloud_offset = 20 * np.sin(current_time * 0.5 + i)
        cloud_surface = pygame.Surface((100, 40), pygame.SRCALPHA)
        pygame.draw.ellipse(cloud_surface, cloud_color, (0, 0, 100, 40))
        self.screen.blit(cloud_surface, (cloud_x + cloud_offset, cloud_y))

elif atmosphere == 'Нічна':
    # Нічне небо з місяцем та зірками
    moon_color = (220, 220, 255)
    star_color = (255, 255, 255)

    # Місяць
    moon_x = center_x - 200
    moon_y = center_y - 150
    moon_size = 25
```

```
        pygame.draw.circle(self.screen, moon_color, (int(moon_x), int(moon_y)), moon_size)

        # Місячне сяйво
        glow_surface = pygame.Surface((moon_size * 4, moon_size * 4), pygame.SRCALPHA)
        glow_color = (*moon_color[:3], 80)
        pygame.draw.circle(glow_surface, glow_color, (moon_size * 2, moon_size * 2), moon_size * 2)
        self.screen.blit(glow_surface, (moon_x - moon_size * 2, moon_y - moon_size * 2))

        # Зірки
        star_positions = [
            (center_x - 100, center_y - 180), (center_x + 150, center_y - 160),
            (center_x - 250, center_y - 100), (center_x + 200, center_y - 80),
            (center_x - 180, center_y - 50), (center_x + 80, center_y - 200)
        ]

        for i, (star_x, star_y) in enumerate(star_positions):
            twinkle = 3 + 2 * np.sin(current_time * 3 + i)
            pygame.draw.circle(self.screen, star_color, (int(star_x), int(star_y)), int(twinkle))

    elif atmosphere == 'Туманна':
        # Туманна атмосфера
        fog_color = (200, 200, 200, 100)

        # Шари туману
        for i in range(5):
            fog_y = center_y - 100 + i * 40 + 20 * np.sin(current_time * 0.3 + i)
            fog_surface = pygame.Surface((self.main_area.width, 60), pygame.SRCALPHA)
            fog_alpha = 60 - i * 10
            current_fog_color = (*fog_color[:3], fog_alpha)
            fog_surface.fill(current_fog_color)
            self.screen.blit(fog_surface, (self.sidebar_width, int(fog_y)))

    # Ландшафт (пагорби)
    hill_points = []
    for x in range(0, self.main_area.width + 20, 20):
        hill_height = 50 + 30 * np.sin((x + current_time * 50) * 0.01)
        hill_points.append((self.sidebar_width + x, center_y + 150 - hill_height))

    # Додаємо кути для закриття
    hill_points.append((self.sidebar_width + self.main_area.width, center_y + 200))
    hill_points.append((self.sidebar_width, center_y + 200))

    hill_color = (34, 139, 34) if atmosphere == 'Денна' else (20, 60, 20)
    pygame.draw.polygon(self.screen, hill_color, hill_points)

    # Дерева
    tree_positions = [center_x - 100, center_x + 50, center_x + 200]
    for tree_x in tree_positions:
        tree_y = center_y + 100
        tree_sway = 5 * np.sin(current_time * 2 + tree_x * 0.01)

        # Стовбур
```

```
            trunk_color = (139, 69, 19)
            pygame.draw.rect(self.screen, trunk_color, (tree_x - 5, tree_y, 10, 40))

            # Крона
            crown_color = (0, 128, 0) if atmosphere == 'Денна' else (0, 80, 0)
            crown_points = [
                (tree_x + tree_sway, tree_y - 20),
                (tree_x - 25 + tree_sway, tree_y + 20),
                (tree_x + 25 + tree_sway, tree_y + 20)
            ]
            pygame.draw.polygon(self.screen, crown_color, crown_points)

        # Інформація
        info_lines = [
            f"Атмосфера: {atmosphere}",
            f"Ландшафт: Пагорби з деревами",
            f"Погода: {'Ясно' if atmosphere == 'Денна' else 'Туман' if atmosphere == 'Туманна' else 'Ясна ніч'}",
            f"Час доби: {'День' if atmosphere == 'Денна' else 'Ніч'}"
        ]

        for i, line in enumerate(info_lines):
            info_text = self.fonts['small'].render(line, True, WHITE)
            self.screen.blit(info_text, (self.sidebar_width + 10, 45 + i * 18))

    def _render_vehicle_model(self):
        """Відображення транспортних засобів"""
        center_x = self.sidebar_width + self.main_area.width // 2
        center_y = self.screen_height // 2

        # Отримання параметрів
        if 'Моделі' in self.effects_data and 'Транспортні засоби' in self.effects_data['Моделі']:
            params = self.effects_data['Моделі']['Транспортні засоби']['params']
            speed = params['швидкість']['value']
        else:
            speed = 5

        current_time = pygame.time.get_ticks() / 1000

        # Автомобіль
        car_x = center_x + 100 * np.sin(current_time * speed * 0.1)
        car_y = center_y

        car_color = (255, 0, 0)  # Червоний автомобіль

        # Основа автомобіля
        car_body = pygame.Rect(car_x - 40, car_y - 15, 80, 30)
        pygame.draw.rect(self.screen, car_color, car_body)
        pygame.draw.rect(self.screen, WHITE, car_body, 2)

        # Дах
        roof = pygame.Rect(car_x - 25, car_y - 25, 50, 20)
        pygame.draw.rect(self.screen, car_color, roof)
        pygame.draw.rect(self.screen, WHITE, roof, 2)
```

```
# Колеса
wheel_spin = current_time * speed * 2
for wheel_offset in [-25, 25]:
    wheel_center = (int(car_x + wheel_offset), int(car_y + 15))
    pygame.draw.circle(self.screen, (50, 50, 50), wheel_center, 10)
    pygame.draw.circle(self.screen, (100, 100, 100), wheel_center, 8)

    # Спиці коліс (обертаються)
    for spoke in range(4):
        spoke_angle = wheel_spin + spoke * 90
        spoke_x = wheel_center[0] + 6 * np.cos(np.radians(spoke_angle))
        spoke_y = wheel_center[1] + 6 * np.sin(np.radians(spoke_angle))
        pygame.draw.line(self.screen, WHITE, wheel_center, (int(spoke_x), int(spoke_y)), 2)

# Вихлопні гази (якщо швидкість висока)
if speed > 7:
    exhaust_particles = []
    for i in range(10):
        particle_x = car_x - 45 - i * 8
        particle_y = car_y + 10 + 5 * np.sin(current_time * 5 + i)
        particle_size = 5 - i * 0.4
        particle_alpha = 255 - i * 25

        if particle_size > 0 and particle_alpha > 0:
            particle_surface = pygame.Surface((int(particle_size * 2), int(particle_size * 2)), pygame.SRCALPHA)
            particle_color = (100, 100, 100, int(particle_alpha))
            pygame.draw.circle(particle_surface, particle_color,
                               (int(particle_size), int(particle_size)), int(particle_size))
            self.screen.blit(particle_surface,
                             (int(particle_x - particle_size), int(particle_y - particle_size)))

# Фари (світяться вночі або при високій швидкості)
if speed > 3:
    # Ліва фара
    headlight_surface = pygame.Surface((30, 20), pygame.SRCALPHA)
    light_color = (255, 255, 200, 150)
    pygame.draw.ellipse(headlight_surface, light_color, (0, 0, 30, 20))
    self.screen.blit(headlight_surface, (car_x + 40, car_y - 10))

    # Права фара
    self.screen.blit(headlight_surface, (car_x + 40, car_y - 5))

# Літак (у небі)
plane_x = center_x - 200 + 100 * np.sin(current_time * speed * 0.05)
plane_y = center_y - 150

plane_color = (200, 200, 255)

# Корпус літака
plane_body = pygame.Rect(plane_x - 30, plane_y - 5, 60, 10)
pygame.draw.rect(self.screen, plane_color, plane_body)
```

```
# Крила
wing = pygame.Rect(plane_x - 40, plane_y - 2, 80, 4)
pygame.draw.rect(self.screen, plane_color, wing)

# Хвіст
tail = pygame.Rect(plane_x - 35, plane_y - 15, 10, 20)
pygame.draw.rect(self.screen, plane_color, tail)

# Інверсійний слід
for i in range(15):
    trail_x = plane_x - 35 - i * 10
    trail_y = plane_y + 2 * np.sin(current_time + i * 0.5)
    trail_alpha = 255 - i * 17

    if trail_alpha > 0:
        trail_surface = pygame.Surface((8, 3), pygame.SRCALPHA)
        trail_color = (255, 255, 255, int(trail_alpha))
        trail_surface.fill(trail_color)
        self.screen.blit(trail_surface, (int(trail_x), int(trail_y)))

# Корабель (якщо є місце)
if self.main_area.height > 400:
    ship_x = center_x
    ship_y = center_y + 180

    ship_color = (139, 69, 19)

    # Корпус корабля
    ship_hull = pygame.Rect(ship_x - 50, ship_y, 100, 20)
    pygame.draw.rect(self.screen, ship_color, ship_hull)

    # Щогла
    pygame.draw.rect(self.screen, ship_color, (ship_x - 2, ship_y - 40, 4, 40))

    # Вітрило
    sail_sway = 10 * np.sin(current_time * 2)
    sail_points = [
        (ship_x, ship_y - 35),
        (ship_x + 30 + sail_sway, ship_y - 30),
        (ship_x + 25 + sail_sway, ship_y - 10),
        (ship_x, ship_y - 15)
    ]
    pygame.draw.polygon(self.screen, (255, 255, 255), sail_points)

    # Хвилі
    wave_y = ship_y + 20
    for wave_x in range(self.sidebar_width, self.sidebar_width + self.main_area.width, 20):
        wave_height = 5 * np.sin((wave_x + current_time * 100) * 0.02)
        pygame.draw.circle(self.screen, (0, 100, 200),
                           (wave_x, int(wave_y + wave_height)), 8)

# Інформація
info_lines = [
```

```
            f"Швидкість: {speed:.1f}",
            f"Транспорт: Автомобіль, Літак, Корабель",
            f"Ефекти: Вихлопи, Інверсійний слід, Хвилі",
            f"Анімація: Рух, Обертання коліс"
        ]

        for i, line in enumerate(info_lines):
            info_text = self.fonts['small'].render(line, True, WHITE)
            self.screen.blit(info_text, (self.sidebar_width + 10, 45 + i * 18))

    def _render_loaded_image_simple(self):
        """СПРОЩЕНЕ відображення зображення"""
        print("=== ПОЧАТОК ВІДОБРАЖЕННЯ ЗОБРАЖЕННЯ ===")

        try:
            # Отримуємо зображення
            image = self.enhanced_pixel_fx.original_image
            img_width, img_height = image.get_size()
            print(f"Розмір зображення: {img_width}x{img_height}")

            # ПРОСТО МАЛЮЄМО В ЛІВОМУ ВЕРХНЬОМУ КУТІ ГОЛОВНОЇ ОБЛАСТІ
            image_x = self.sidebar_width + 50
            image_y = 150

            print(f"Позиція для малювання: ({image_x}, {image_y})")

            # МАЛЮЄМО ЗОБРАЖЕННЯ
            self.screen.blit(image, (image_x, image_y))
            print("Зображення намальовано!")

            # ЧЕРВОНА РАМКА ДЛЯ НАОЧНОСТІ
            border_rect = pygame.Rect(image_x - 5, image_y - 5, img_width + 10, img_height + 10)
            pygame.draw.rect(self.screen, RED, border_rect, 3)
            print("Рамку намальовано!")

            # ТЕКСТ З ІНФОРМАЦІЄЮ
            info_text = f"ЗОБРАЖЕННЯ ВІДОБРАЖЕНО! {img_width}x{img_height}"
            info_surface = self.fonts['regular'].render(info_text, True, GREEN)
            self.screen.blit(info_surface, (image_x, image_y + img_height + 10))

            print("=== ВІДОБРАЖЕННЯ ЗАВЕРШЕНО ===")

        except Exception as e:
            print(f"ПОМИЛКА ВІДОБРАЖЕННЯ: {e}")
            import traceback
            traceback.print_exc()

            error_text = f"ПОМИЛКА ВІДОБРАЖЕННЯ: {str(e)}"
            error_surface = self.fonts['regular'].render(error_text, True, RED)
            self.screen.blit(error_surface, (self.sidebar_width + 50, 200))

    def _handle_mouse_motion(self, event):
        """Обробка руху миші для перетягування слайдерів"""
```

```
    if hasattr(self, 'dragging_slider') and self.dragging_slider:
        control, param_name = self.dragging_slider
        mouse_x = event.pos[0]
        self._update_slider_value(control, mouse_x)
        # Не виводимо лог тут, щоб не засмічувати консоль

def _handle_mouse_up(self, event):
    """Виправлена обробка відпускання миші"""
    if (hasattr(self, 'dragging_slider') and
            self.dragging_slider is not None):

        try:
            if (isinstance(self.dragging_slider, tuple) and
                    len(self.dragging_slider) == 2):
                control, param_name = self.dragging_slider
                print(f"Завершено перетягування слайдера: {param_name}")
            else:
                print("Завершено перетягування слайдера (невідомий)")
        except Exception as e:
            print(f"Помилка при завершенні перетягування: {e}")
        finally:
            # ОБОВ'ЯЗКОВО скидаємо режим перетягування
            self.dragging_slider = None
def _render_loaded_image(self):
    """Відображення завантаженого зображення"""
    print("_render_loaded_image викликано")

    if not hasattr(self, 'enhanced_pixel_fx') or not self.enhanced_pixel_fx.original_image:
        print("Немає зображення для відображення")
        return

    # Отримуємо розмір зображення
    img_width, img_height = self.enhanced_pixel_fx.original_image.get_size()
    print(f"Розмір зображення: {img_width}x{img_height}")

    # Центруємо зображення в головній області
    center_x = self.sidebar_width + self.main_area.width // 2
    center_y = 70 + (self.main_area.height - 70) // 2

    # Розрахунок позиції для центрування
    image_x = center_x - img_width // 2
    image_y = center_y - img_height // 2

    print(f"Позиція зображення: ({image_x}, {image_y})")

    # Відображення зображення
    try:
        self.screen.blit(self.enhanced_pixel_fx.original_image, (image_x, image_y))
        print("Зображення успішно відображено")

        # Рамка навколо зображення
        border_rect = pygame.Rect(image_x - 2, image_y - 2, img_width + 4, img_height + 4)
        pygame.draw.rect(self.screen, WHITE, border_rect, 2)
```

```
            # Інформація про зображення
            filename = getattr(self, 'loaded_image_path', 'Unknown').split('/')[-1].split('\\')[-1]
            info_text = self.fonts['small'].render(
                f"Розмір: {img_width}x{img_height} | {filename}",
                True, GRAY
            )
            info_rect = info_text.get_rect(center=(center_x, image_y + img_height + 20))
            self.screen.blit(info_text, info_rect)

        except Exception as e:
            print(f"Помилка відображення зображення: {e}")
            error_text = f"Помилка відображення: {str(e)}"
            error_surface = self.fonts['small'].render(error_text, True, RED)
            self.screen.blit(error_surface, (image_x, image_y))

    def _get_main_area_title(self):
        """Отримання заголовка для головної області"""
        if self.selected_effect:
            if self.current_mode == 'models':
                return f"Перегляд: Модель персонажа з ефектами"
            else:
                effect_category = "Невідома категорія"
                for category, effects in self.effects_data.items():
                    if self.selected_effect in effects:
                        effect_category = category
                        break
                return f"Ефект: {self.selected_effect} ({effect_category})"
        else:
            return "Оберіть ефект зі списку зліва для попереднього перегляду"

    def _render_image_loading_prompt(self):
        """Інструкції для завантаження зображення"""
        center_x = self.sidebar_width + self.main_area.width // 2
        center_y = self.screen_height // 2

        # ВЕЛИКИЙ ТЕКСТ
        main_text = self.fonts['title'].render("ЗАВАНТАЖТЕ ЗОБРАЖЕННЯ", True, WHITE)
        main_rect = main_text.get_rect(center=(center_x, center_y - 100))
        self.screen.blit(main_text, main_rect)

        # КНОПКА ЗАВАНТАЖЕННЯ
        load_button = pygame.Rect(center_x - 150, center_y - 50, 300, 60)
        pygame.draw.rect(self.screen, PURPLE, load_button)
        pygame.draw.rect(self.screen, WHITE, load_button, 3)

        button_text = self.fonts['title'].render("ЗАВАНТАЖИТИ ФАЙЛ", True, WHITE)
        button_rect = button_text.get_rect(center=load_button.center)
        self.screen.blit(button_text, button_rect)

        # КНОПКА ТЕСТУ
        test_button = pygame.Rect(center_x - 100, center_y + 30, 200, 40)
        pygame.draw.rect(self.screen, GREEN, test_button)
```

```
    pygame.draw.rect(self.screen, WHITE, test_button, 2)

    test_text = self.fonts['regular'].render("ТЕСТ ЗОБРАЖЕННЯ", True, WHITE)
    test_rect = test_text.get_rect(center=test_button.center)
    self.screen.blit(test_text, test_rect)

    # Зберігаємо для кліків
    self.quick_load_button = load_button
    self.test_image_button = test_button

def _sync_effects_with_enhanced_pixelfx(self):
    """Синхронізація ефектів з розширеним PixelFX"""
    if not self.enhanced_pixel_fx or not self.selected_effect:
        return

    print(f"Синхронізація ефекту: {self.selected_effect}")

    # Очищуємо поточні ефекти
    self.enhanced_pixel_fx.clear_effects()

    # Додаємо відповідний ефект
    if self.selected_effect == "Інтерактивні частинки":
        # Використовуємо існуючу систему частинок
        self.enhanced_pixel_fx.add_effect(self.interactive_particles)
        print("Додано інтерактивні частинки")

    elif self.selected_effect == "Вогонь":
        # Додаємо ефект вогню (потрібно створити)
        fire_effect = self._create_fire_effect()
        self.enhanced_pixel_fx.add_effect(fire_effect)
        print("Додано ефект вогню")

    elif self.selected_effect == "Магічний портал":
        # Додаємо магічний портал
        portal_effect = self._create_portal_effect()
        self.enhanced_pixel_fx.add_effect(portal_effect)
        print("Додано магічний портал")

def _create_fire_effect(self):
    """Створення ефекту вогню"""

    class FireEffect:
        def __init__(self):
            self.particles = []
            self.max_particles = 50
            self.width = 800
            self.height = 600
            self._initialize_particles()

        def _initialize_particles(self):
            import random
            self.particles = []
            for _ in range(self.max_particles):
```

ДОДАТОК Г

(обов’язковий)

ІЛЮСТРАТИВНА ЧАСТИНА

РОЗРОБКА БІБЛІОТЕКИ ВІЗУАЛЬНИХ ЕФЕКТІВ ДЛЯ ВИКОРИСТАННЯ У ГРАФІЧНИХ ТА ІГРОВИХ ЗАСТОСУНКАХ

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення

Бакалаврська кваліфікаційна робота на тему:
«Розробка бібліотеки візуальних ефектів для використання у графічних та ігрових застосунках»

Автор: ст.групи 5ПІ-21б Гаврилюк В.П.
Науковий керівник: к.т.н., доцент каф. ПЗ Ткаченко О.М.

Вінниця - 2025

Рисунок Г.1 – Титульний слайд

Актуальність теми

Сучасний етап розвитку ігрової індустрії вимагає постійного вдосконалення графічних технологій, зокрема в області візуальних ефектів. Однією з важливих задач у розробці візуальних ефектів для ігрових застосунків є досягнення оптимального балансу між візуальною якістю та продуктивністю [1].

Традиційні методи створення візуальних ефектів часто обмежені заздалегідь визначеними шаблонами та недостатньою гнучкістю налаштувань [2]. У цьому контексті, використання модульних бібліотек, здатних генерувати різноманітні ефекти з можливістю тонкого налаштування параметрів, відкриває нові перспективи для підвищення якості графічного представлення, що має значущий потенціал для розвитку як в ігровій індустрії, так і в суміжних областях.

Сучасні візуальні ефекти є найреалістичнішими, відображають фізичні процеси, точно передають світлові та колірні особливості об'єктів, підлягають модифікації для зміни візуального представлення. Бібліотека візуальних ефектів є багатофункціональним інструментом для розробників, дозволяє істотно знизити необхідний обсяг ручного кодування порівняно з існуючими методами [3].

Важливість оптимізації продуктивності визначається суттєвим внеском у підвищення плавності роботи ігрових застосунків на різних апаратних конфігураціях. У ігровій індустрії якісні візуальні ефекти відіграють важливу роль у створенні атмосфери та емоційного впливу на гравців. Наприклад, ефективне застосування ефектів частинок та світла може бути критичним для переконливого відтворення природних явищ або магічних здібностей персонажів [4].

Крім того, більшість комерційних бібліотек візуальних ефектів орієнтовані на великі студії розробки з відповідними бюджетами, залишаючи поза увагою потреби незалежних розробників та малих команд. Ліцензійні обмеження та високі витрати на підтримку таких застосунків створюють додаткові бар'єри для входу в ігрову індустрію. Водночас, відкриті рішення часто характеризуються недостатньою документацією, обмеженою підтримкою спільноти та відсутністю регулярних оновлень, що ускладнює їх інтеграцію в сучасні ігрові проєкти.

Власна розробка дозволяє створити спеціалізоване рішення, адаптоване під конкретні вимоги цільової аудиторії, з урахуванням сучасних стандартів розробки та можливостей оптимізації продуктивності. Такий підхід забезпечує повний контроль над архітектурою системи, можливість швидкого реагування на зміни технологічних вимог та створення унікальних функціональних можливостей, які відрізняють продукт від існуючих аналогів. Власна бібліотека може бути оптимізована під специфічні сценарії використання, що дозволяє досягти кращого співвідношення якості візуалізації та продуктивності порівняно з універсальними рішеннями. Тому актуальною є власна розробка модульної бібліотеки візуальних ефектів.

Рисунок Г.2 — Актуальність теми

Мета, об’єкт та предмет дослідження

Мета дослідження. Метою роботи є підвищення продуктивності обчислень за рахунок раціонального розподілу задач між доступними процесорними ядрами, а також реалістичності за рахунок динамічного відтворення змін на зображенні.
Об'єктом дослідження є процес створення та відтворення візуальних ефектів у ігрових застосунках.
Предметом дослідження є методи та засоби розробки оптимізованих візуальних ефектів для ігрових застосунків.

Рисунок Г.3 — Мета, об’єкт та предмет дослідження

Завдання дослідження

Основними задачами дослідження є:
- розробка алгоритмів генерації візуальних ефектів, які забезпечать високу якість та оптимальну продуктивність на різних апаратних конфігураціях;
- розробка модуля системи частинок, який буде спроможний генерувати, відтворювати та керувати різноманітними ефектами на основі частинок;
- розробка модуля графічного редактора, який буде зручним у використанні для розробників ігор, забезпечуючи зручність і швидкість налаштування параметрів ефектів;
- інтеграція різних модулів в єдину бібліотеку, яка буде використовуватись для створення візуальних ефектів у ігрових застосунках.

Рисунок Г.4 — Завдання дослідження

Наукова новизна

1. Подальшого розвитку отримав метод накладання динамічних ефектів на статичне зображення, який, на відміну від відомих, використовує інтелектуальний аналіз вмісту зображення для автоматичного визначення глибини та перспективи, що дозволяє інтегрувати частинки та візуальні елементи з урахуванням структури сцени без необхідності ручного маскування або налаштування параметрів для кожного типу зображень.
2. Подальшого розвитку отримав метод багатопроцесорного експорту анімацій з адаптивним розподілом ресурсів, який, на відміну від відомих, впроваджує динамічне балансування навантаження з аналізом складності кожного кадру та інтелектуальним розподілом задач між доступними процесорними ядрами, що забезпечує оптимальне використання обчислювальних ресурсів системи.

Рисунок Г.5 — Наукова новизна

Практична цінність отриманих результатів

Практична цінність розробки полягає у суттєвому прискоренні рендеренгу анімації шляхом ефективного використання всіх доступних процесорних ядер, навіть у системах з нерівномірним навантаженням.

Рисунок Г.6 — Практична цінність отриманих результатів

Порівняльний аналіз аналогів

Критерії	Unity VFX Graph	Unreal Engine Niagara	PopcornFX	Notch Builder	Shadertoy	Власна бібліотека
Підвищення продуктивності	+	+	+	-	-	+
Крос-платформна сумісність	-	-	+	-	+	+
Модульна архітектура	+	+	+	-	-	+
Простота інтеграції	-	-	+	-	+	+
Гнучкість налаштувань	+	+	-	+	+	+
Загальна оцінка	60%	60%	80%	40%	60%	100%

Рисунок Г.7 — Порівняльний аналіз аналогів

Використані технології

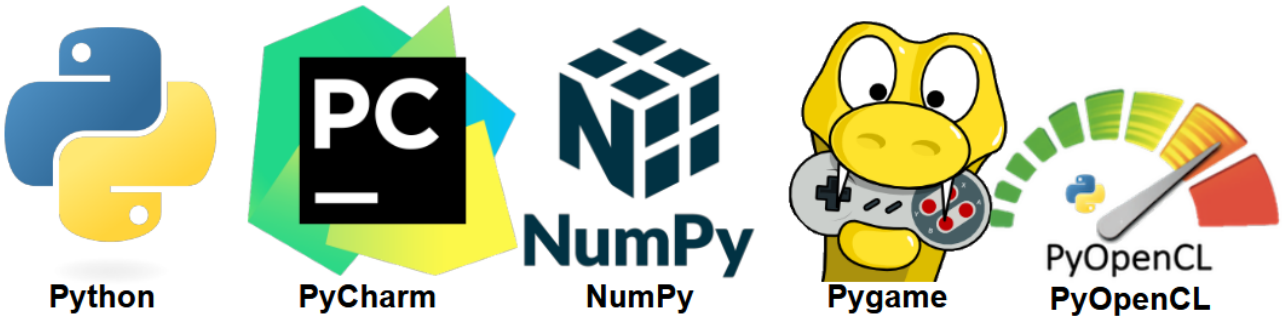


Рисунок Г.8 — Використані технології

Розробка методу накладання динамічних ефектів на статичне зображення

Послідовність виконання роботи методу:

1. Завантаження зображення:
   - Підтримка різноманітних форматів зображень (PNG, JPG, TIFF, тощо).
   - Автоматичне масштабування зображення для оптимальної продуктивності.
   - Аналіз зображення для визначення областей, придатних для ефектів.
2. Вибір та налаштування ефекту:
   - Вибір категорії ефекту (частинки, освітлення, спецефекти).
   - Вибір конкретного ефекту (дощ, сніг, вогонь, магічний портал, тощо).
   - Налаштування специфічних параметрів через панель властивостей.
3. Інтелектуальне поєднання ефекту із зображенням:
   - Автоматичне визначення фронтальних та фонових елементів зображення.
   - Аналіз освітлення сцени для реалістичного накладання ефектів.
   - Налаштування глибини та масштабу ефекту відповідно до перспективи зображення.
4. Візуалізація:
   - Динамічний рендеринг ефекту на зображенні з оновленням в реальному часі.
   - Інтерактивне управління властивостями ефекту з миттєвим відображенням змін.
   - Використання багатопоточної обробки для забезпечення плавності анімації.
5. Налаштування ефекту:
   - Маскування зон, де ефект не повинен відображатися.
   - Налаштування взаємодії ефекту з елементами зображення.
   - Коригування швидкості, інтенсивності та інших параметрів для досягнення бажаного результату.
6. Налаштування часової анімації:
   - Використання часової шкали для контролю поведінки ефекту у часі.
   - Налаштування циклічності та тривалості ефекту.
   - Додавання ключових кадрів для зміни параметрів впродовж анімації.
7. Експорт результату:
   - Вибір формату експорту (GIF, відео, PNG-послідовність, спрайт-атлас).
   - Налаштування параметрів якості та оптимізації.
   - Експорт з використанням багатопроцесорної обробки для пришвидшення.

Рисунок Г.9 - Розробка методу накладання динамічних ефектів на статичне зображення

Розробка методу багатопроцесорного експорту анімацій з адаптивним розподілом ресурсів

1. Аналіз системних ресурсів:
   - Визначення кількості доступних ядер процесора.
   - Оцінка доступної системної пам'яті.
   - Встановлення оптимальної кількості робочих процесів.
2. Підготовка даних для паралельної обробки:
   - Створення копії оригінального зображення та ефектів для безпечного використання в паралельних процесах.
   - Розбиття загальної кількості кадрів на окремі завдання.
   - Формування аргументів для кожного процесу через функцію args_list.
3. Створення та запуск пулу процесів:
   - Ініціалізація пулу процесів через конструкцію with multiprocessing.Pool() as pool.
   - Розподіл завдань між процесами за допомогою методу pool.map().
   - Запуск паралельного виконання функції export_frame_worker для кожного кадру.
4. Обробка кадрів у паралельних процесах:
   - Виконання функції create_frame() для генерації кожного кадру.
   - Застосування ефектів до базового зображення.
   - Повернення готового кадру та його індексу.
5. Збір результатів та їх впорядкування:
   - Сортування отриманих кадрів за індексом для збереження правильної послідовності.
   - Перетворення кадрів у формат, придатний для збереження (Image.fromarray).
   - Підготовка метаданих анімації (тривалість кадру, цикличність).
6. Запис анімації у файл:
   - Вибір формату збереження (GIF або відео).
   - Застосування оптимізації залежно від обраного формату.
   - Збереження анімації з встановленими параметрами.
7. Аналіз та звітування про продуктивність:
   - Вимірювання загального часу виконання та обчислення швидкості обробки (кадрів/секунду).
   - Відображення прогресу та оцінки часу до завершення.
   - Генерація звіту про успішність експорту.

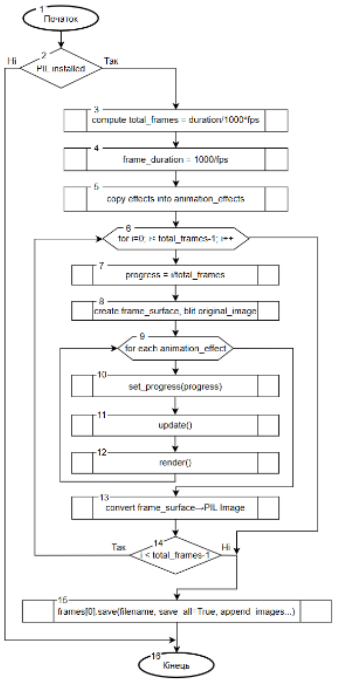


Рисунок Г.10 - Розробка методу багатопроцесорного експорту анімацій з адаптивним розподілом ресурсів

Тестування бібліотеки

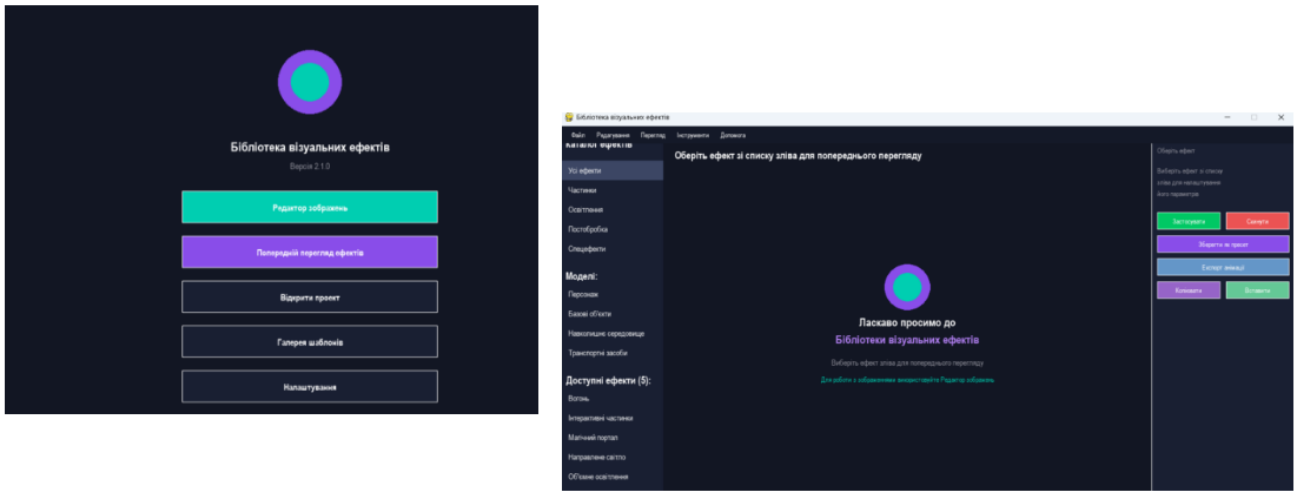


Рисунок Г.11 — Тестування модуля створення ефектів

Тестування бібліотеки

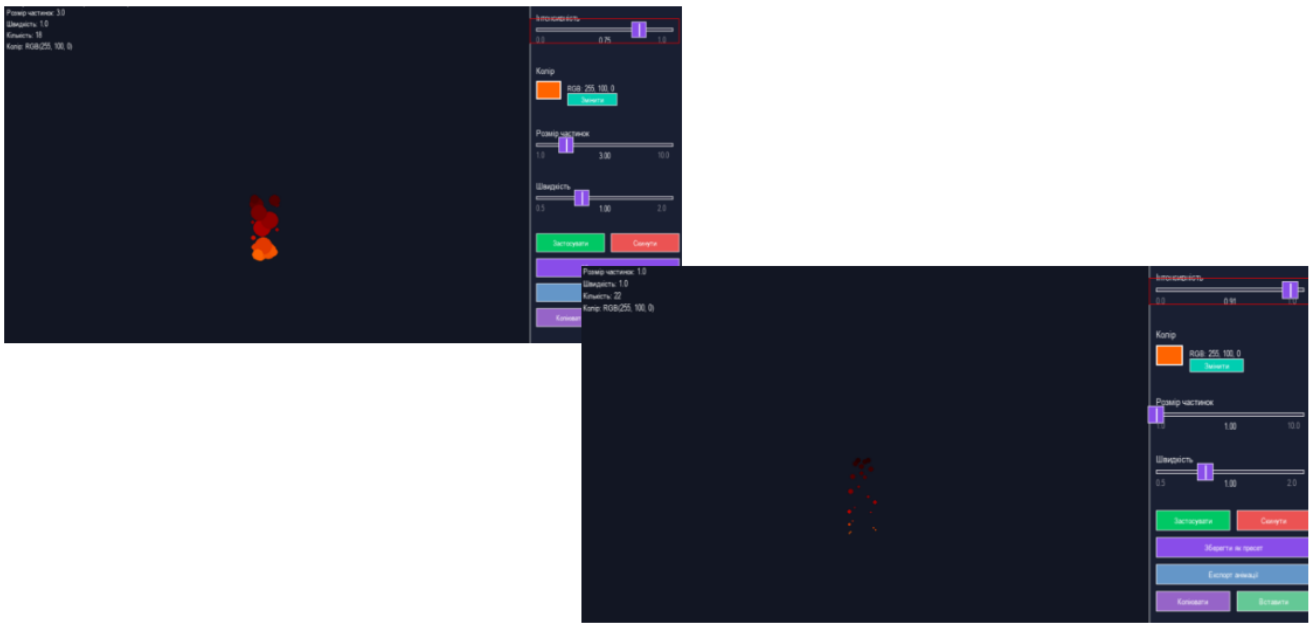


Рисунок Г.12 — Тестування створення ефекту

Тестування бібліотеки

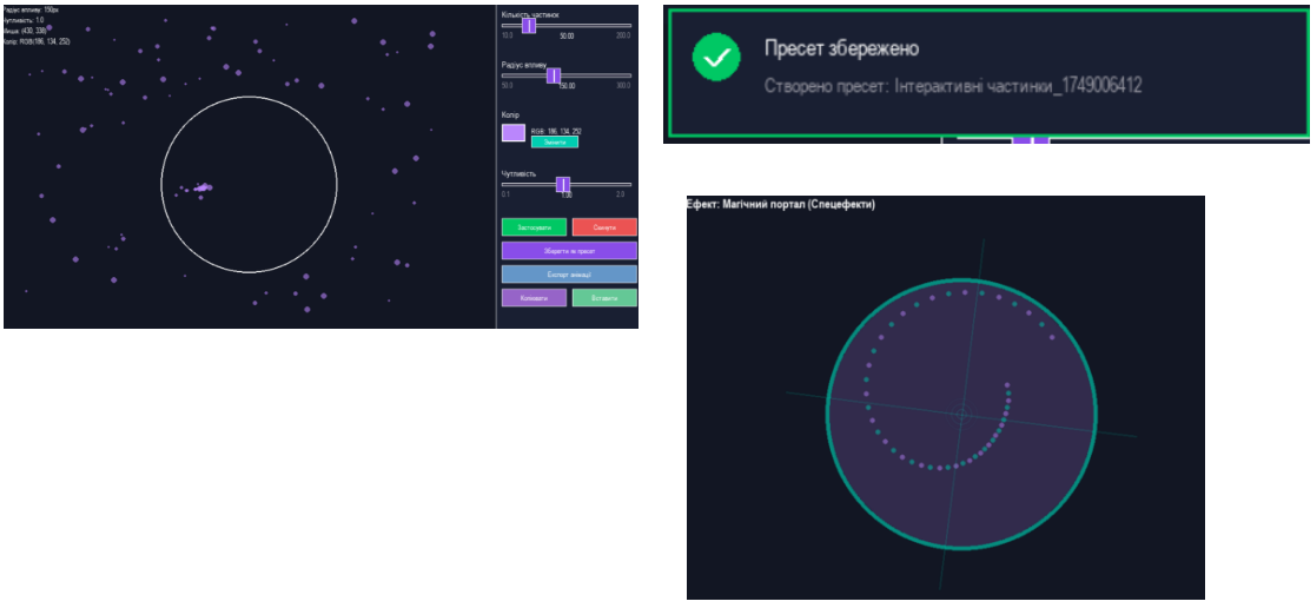


Рисунок Г.13 — Тестування роботи та збереження ефектів

Тестування бібліотеки

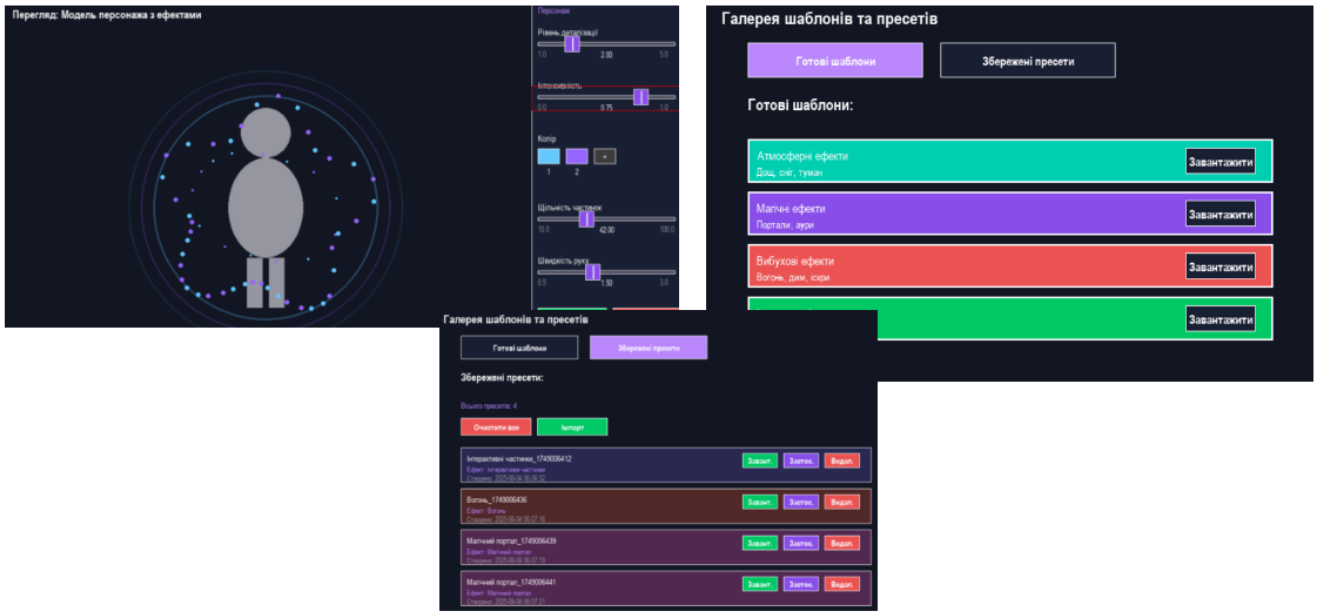


Рисунок Г.14 — Тестування меню збережених ефектів

Тестування бібліотеки

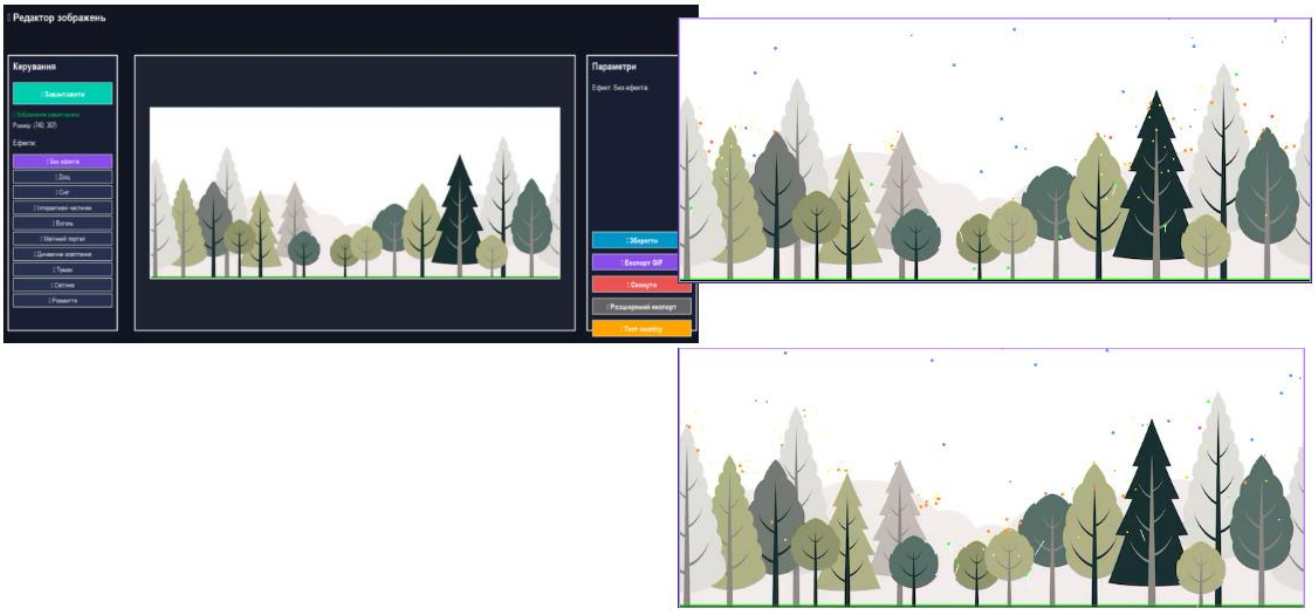


Рисунок Г.15 — Тестування накладання ефекту на зображення

Висновки

У результаті виконання бакалаврської кваліфікаційної роботи була розроблена бібліотека візуальних ефектів, призначена для використання у графічних та ігрових застосунках. Реалізація програмного забезпечення виконувалась із використанням середовища розробки PyCharm. Бакалаврську кваліфікаційну роботу реалізовано згідно методичних вказівок [26].

У процесі виконання роботи було проведено детальний аналіз поточного стану технологій у сфері візуальних ефектів. Розглянуто сучасні аналоги програмних засобів, визначено їх переваги та недоліки, а також здійснено порівняння з власною розробкою. На основі результатів порівняння сформульовано основні задачі, вирішення яких реалізовано в межах проєкту.

Під час аналізу технологій програмної реалізації було обґрунтовано вибір мов програмування Python.

У першому розділі бакалаврської квалфікаційної роботи розглянуто поточний стан технологій у сфері візуальних ефектів, важливість оптимізації продуктивності та задачі, пов'язані з традиційними методами. У розділі також проведено порівняння запропонованого рішення з існуючими системами та окреслено завдання проєкту.

У другому розділі надано детальний опис алгоритмів програми. Також описано вхідні та вихідні дані системи, а також метод накладання динамічних ефектів на статичне зображення, метод системи інтерактивних частинок, метод багатопроцесорного експорту анімацій з адаптивним розподілом ресурсів.

У третьому розділі було проведено порівняльний аналіз мов програмування розробки бібліотеки візуальних елементів та обрано мову Python. Було розроблено модуль обробки зображень, модуль керування пресетами і переходами, модуль інтерфейсу користувача.

У четвертому розділі проведено тестування функціональних та графічних компонентів програмної бібліотеки, розробленої в рамках бакалаврської кваліфікаційної роботи.

Рисунок Г.16 — Висновки

Апробація результатів роботи і публікації

Апробація результатів роботи. Описані у бакалаврській кваліфікаційній роботі положення доповідалися на міжнародній науково-практичній конференції «Молодь в науці: дослідження, проблеми, перспективи (МН-2025)».

Публікації.

1. Гаврилюк В.П., Ткаченко О.М. Розробка бібліотеки візуальних ефектів для графі-чних та ігрових застосунків / В.П. Гаврилюк, О.М. Ткаченко // Матеріали Міжнародної науково-практичної інтернет-конференції «Молодь в науці: дослідження, проблеми,перспективи (МН-2025)». – Вінниця: ВНТУ, 2025 [Електронний ресурс] – Режим доступу: https://conferences.vntu.edu.ua/index.php/mn/mn2025/paper/view/24944

2. Гаврилюк В.П., Ткаченко О.М. Розробка методу багатопроцесорного експорту анімацій з адаптивним розподілом ресурсів / В.П. Гаврилюк, О.М. Ткаченко // Матеріали Міжнародної науково-практичної інтернет-конференції «Молодь в науці: дослідження, проблеми,перспективи (МН-2025)». – Вінниця: ВНТУ, 2025 [Електронний ресурс] – Режим доступу: https://conferences.vntu.edu.ua/index.php/mn/mn2025/paper/view/25440

3. Гаврилюк В.П., Ткаченко О.М. Розробка методу накладання динамічних ефектів на статичне зображення з інтелектуальним аналізом сцени / В.П. Гаврилюк, О.М. Ткаченко // Матеріали Міжнародної науково-практичної інтернет-конференції «Молодь в науці: дослідження, проблеми,перспективи (МН-2025)». – Вінниця: ВНТУ, 2025 [Електронний ресурс] – Режим доступу: https://conferences.vntu.edu.ua/index.php/mn/mn2025/paper/view/25441

Рисунок Г.17 — Апробація результатів роботи і публікації