Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра програмного забезпечення

Бакалаврська кваліфікаційна робота

на тему: «Методи та програмні засоби високопродуктивного рендерингу Гуро»

Виконав: студент 4 курсу
групи 2ПІ-21б
спеціальності
121 – Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)
Барасій О. А.
(прізвище та ініціали)
Керівник: д.т.н., проф. каф. ПЗ Романюк О. Н.
(прізвище та ініціали)
Рецензент: к.т.н., доц. каф. ПЗ Снігур А. В.
(прізвище та ініціали)

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

Барасій Ользі Андріївні

1. Тема роботи – «Методи та програмні засоби високопродуктивного рендерингу Гуро»

Керівник роботи: Романюк Олександр Никифорович, д.т.н., проф. 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: Метод зафарбовування – метод Гуро; розмір координатного простору 600x600; кольоровий режим – truecolor; тип полігональної моделі – трикутна; середовище розробки PyCharm 2024.2.1; мова розробки Python; операційна система – Windows 10.

4. Зміст розрахунково-пояснювальної записки: вступ; аналіз методів зафарбовування поверхонь; розробка методів для високопродуктивного та якіснішого формування зображень методом Гуро рендерингу; розробка програмних засобів для покращеного зафарбовування Гуро; тестування програмного 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.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-4	Романюк О.Н., д.т.н., професор 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 методів зафарбовування поверхонь	25.03.25 – 30.03.25	вик
2	Розробка методів для підвищення продуктивності та реалістичності формування зображень методом гуро рендерингу	31.03.25 – 20.04.25	вик
3	Розробка програмних засобів для покращеного зафарбовування Гуро	21.04.25 – 30.04.25	вик
4	Тестування програмного застосунку	01.05.25 – 10.05.25	вик
5	Оформлення матеріалів до захисту БКР	11.05.25 – 30.05.25	вик

Студент _____ Барасій О. А.
(підпис) (прізвище та ініціали)

Керівник бакалаврської кваліфікаційної роботи _____ Романюк О.Н.
(підпис) (прізвище та ініціали)

АННОТАЦІЯ

УДК

Барасій О. А. Методи та засоби високопродуктивного рендерингу Гуро : бакалаврська кваліфікаційна робота зі спеціальності 121 Інженерія програмного забезпечення, освітня програма – інженерія програмного забезпечення. Вінниця: ВНТУ, 2025. 102 с.

На укр. мові. Бібліогр. : 23 назв ; рис. : 29; табл. 2.

У бакалаврській кваліфікаційній роботі було розроблено методи та програмні засоби для підвищення якості, продуктивності та реалістичності рендеригу Гуро.

Обґрунтовано необхідність розробки власних підходів, спрямованих на підвищення швидкості обчислень, а також покращення якості та реалістичності зображень при їх рендерингу методом Гуро.

Розроблено методи, що дозволяють ефективно задіяти доступні ресурси графічних процесорів та зменшити час обробки графічних зображень.

Для реалізації роботи використано мову програмування Python. Для розробки програмного забезпечення обрано середовище розробки PyCharm.

На основі отриманих у ході виконання бакалаврської кваліфікаційної роботи теоретичних напрацюваннь, запропоновано алгоритми та реалізовано програмні засоби для швидкісного та більш якісного і реалістичного зафарбовування полігональних моделей за методом Гуро рендерингу.

Ключові слова: метод Гуро, рендеринг, смуги Маха, модифікація методу Гуро.

UDC

Barasiy O. A. Methods and tools of accelerated Gouro rendering: bachelor's qualification work in the specialty 121 Software Engineering, educational program - Software Engineering. Vinnytsia: VNTU, 2025. 102 p.

In Ukrainian. Bibliography: 23 titles; fig.: 29; table 2.

In the bachelor's diploma work, methods and software applications were developed to accelerate and improve the quality and realism of Gouro rendering.

The need to develop our own approaches aimed at increasing the speed of calculations, as well as improving the quality and realism of images when rendering them using the Gouro method was substantiated.

Methods were developed that allow for the effective use of available resources of graphics processors and reduce the time for processing graphic images.

The Python programming language was used to implement the work. The PyCharm development environment was chosen for software development.

Based on the theoretical developments obtained during the bachelor's thesis, algorithms have been proposed and software tools have been implemented for fast, high-quality and realistic painting of polygonal models using the Gouraud rendering method.

Keywords: Gouraud method, rendering, Mach bands, modification of the Gouraud method.

ЗМІСТ

ВСТУП

Обґрунтування вибору теми дослідження. У наш час комп’ютерна графіка є важливою складовою багатьох галузей: від відеоігор, анімації та віртуальної реальності до медицини, архітектури та промисловості. Одним із ключових процесів у цій сфері є рендеринг. Існує багато методів рендерингу, кожен з яких має свої особливості, переваги та недоліки. Серед них метод зафарбовування Гуро. Він є одним із класичних підходів до обчислення освітлення на поверхнях, та зафарбовування об’єктів. Цей метод забезпечує плавні переходи яскравості між вершинами об’єкта, завдяки чому створюється ілюзія гладкості. Він був запропонований Анрі Гуро, але не втратив актуальності і сьогодні, зокрема у системах з обмеженими ресурсами. Причина в його простоті та відносно високій швидкості роботи, що особливо важливо для додатків реального часу або пристроїв із обмеженими ресурсами.

Не зважаючи на появу складніших алгоритмів, таких як зафарбовування Фонга або трасування променів, метод Гуро продовжує широко використовуватись у багатьох програмних та апаратних рішеннях завдяки своїй обчислювальній простоті та швидкості. Це особливо важливо в умовах реального часу, де швидкість рендерингу безпосередньо впливає на користувацький досвід, наприклад, у мобільних застосунках або вбудованих системах.

З розвитком апаратного забезпечення та появою нових бібліотек і фреймворків для графічного програмування, з’являються нові можливості для оптимізації класичних алгоритмів. Це відкриває простір для дослідження методів покращення продуктивності зафарбовування Гуро без втрати візуальної якості.

Тож зважаючи на сучасні тенденції, де обчислювальні ресурси використовуються з максимальною ефективністю, постає необхідність не просто застосовувати класичні методи рендерингу, а й удосконалювати їх з точки зору продуктивності, якості та реалістичності. Нинішній процес рендерингу методом Гуро характеризується значними обчислювальними витратами. Саме тому дослідження ефективних реалізацій цього алгоритму, зниження обчислювальних

витрат зі збереженням якості, а також вивчення відповідних інструментів та програмних засобів, є актуальним та значущим.

Метод Гуро часто виступає як базовий приклад при вивченні графічних конвеєрів, шейдерів, а також технік зафарбовування й освітлення, і широко застосовується для створення ескізів у 3D Max.

Сучасні розробники дедалі частіше стикаються з викликами впровадження 3D-графіки в мобільні додатки, веб-сервіси, а також у системи доповненої реальності. У цьому контексті критично важливо мати не лише теоретичні знання візуалізації, а й уміння ефективно оптимізувати графічні алгоритми під різноманітні апаратні платформи.

Саме тому обрана тема є актуальною та важливою для розвитку комп'ютерної графіки в сучасних програмних системах.

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 та реалістичності зафарбовування Гуро за рахунок розробки нових методів і засобів.

Основними задачами дослідження є:

– проведення аналізу існуючих методів і засобів рендеригу з метою виявлення їх оптимізації, підвищення якості та реалістичності;

– запропонувати:

– методи підвищення якості рендеригу Гуро;

– методи усунення візуальних артефактів, що виникають при реалізації зафарбовування методом Гуро;

– розробити алгоритми, програмні модулі та систему візуалізації з урахуванням розроблених методів;

– виконати експериментальну перевірку ефективності розроблених методів.

Об’єкт дослідження – процес кінцевої візуалізації Гуро рендерингу у системах комп’ютерної графіки.

Предмет дослідження – методи та засоби високопродуктивного рендерингу Гуро.

Методи дослідження. У процесі досліджень використовувались: теорія чисельних методів, теорія чисел для реалізації лінійної інтерполяції кольору вздовж сторін і внутрішньої частини полігону, лінійна алгебра для опису геометрії двовимірних фігур, аналітична геометрія для точного задання положення полігонів у 2D-просторі, визначення площинних координат, комп’ютерне моделювання для перевірки коректності алгоритму та оцінки його продуктивності.

Наукова новизна отриманих результатів:

1. Подальшого розвитку отримав метод Гуро, який відрізняється від відомого, використанням методу Фонга для обчислення інтенсивності кольору на межах цифрових сегментів, на які розбитий рядок растеризації, з подальшим визначенням інтенсивності кольору, що дозволило підвищити продуктивність формування графічних сцен.

2. Вперше запропоновано метод підвищення реалістичності рендеригу Гуро, особливість якого полягає в усуненні смуг Маха, за рахунок адаптивної зміну кольору на суміжних ребрах трикутників, що дозволило підвищити реалістичність.

3. Подальшого розвитку отримав метод Гуро, який відрізняється від відомого розбиттям вихідного трикутника на чотири складові, однакової площі, та орієнтації. При цьому зафарбовується лише одна складова з подальшою трансформацією отриманих результатів на інші.

Практичне значення отриманих результатів полягає в тому, що на основі отриманих в бакалаврській кваліфікаційній роботі роботі теоретичних положень запропоновано алгоритми та розроблено програмні засоби високопродуктивного рендерингу Гуро для комп’ютерних систем візуалізації зображень.

Особистий внесок здобувача. У друкованих працях, опублікованих у співавторстві, автору належать такі результати: аналіз реалізації методу Гуро у відеокартах [1], особливості вбудованих відеокарт [2], метод додаткової триангуляції[3].

Апробація матеріалів бакалаврської кваліфікаційної роботи. Основні положення БДР доповідалися та обговорювалися на Міжнародних і Всеукраїнських конференціях: XVIII Всеукраїнська науково-практична WEB конференція аспірантів, студентів та молодих вчених «Компютерні інтелектуальні системи та мережі» (Кривий Ріг, 2025), I Всеукраїнська науково-практична конференція «Науковий пошук: сучасні проблеми інформаційних технологій, математики, фізики, астрономії, управління та освіти» (Житомир, 2025), IV Міжнародна науково-практична конференція «Інформаційні технології в освіті та науці» (Мелітополь, 2025).

Публікації. Основні результати досліджень опубліковано в 3 наукових працях у матеріалах конференцій, 1 авторське свідоцтво про реєстрацію авторського права на комп’ютерну програму.

1 АНАЛІЗ МЕТОДІВ ЗАФАРБОВУВАННЯ ПОВЕРХОНЬ

1.1 Основні етапи графічного конвеєра

Базовою концепцією комп’ютерної графіки, яка описує послідовність обробки даних для генерації тривимірної сцени на двовимірному екрані є графічний конвеєр. Це послідовний процес, що виконується або програмно, або апаратно відеокартою, і включає серію етапів, кожен з яких виконує окремі завдання перетворення, від геометричного опису сцени до остаточного зображення (рисунок 1.1) [4].

Рисунок 1.1 – Основні етапи графічного конвеєра

Графічний конвеєр умовно можна розділити на 4 блоки:

– програмний додаток, який містить опис 3D-сцени: об’єкти, освітлення, камеру, матеріали, передає дані для подальшої обробки;

– геометричний блок, що виконує всі просторові перетворення: трансформує об'єкти, переводить координати у вигляд камери, виконує проєкцію, обчислює нормалі. Готує сцену до виводу на екран;

– блок растеризації, який перетворює геометричні примітиви у пікселі, визначає кольори, застосовує текстури, видаляє непомітні частини;

– блок візуалізації, що формує кінцеве зображення для виводу на екран користувача.

Кожен блок поділяється на етапи. Програмний блок містить один етап:

– опис тривимірного простору сцени, що формує початкові дані про об’єкти, освітлення та камеру в 3D-сцені.

Геометричний блок містить сім етапів:

– теселяція – це розбиття складних поверхонь на простіші геометричні примітиви;

– тріангуляція – це перетворення багатокутників на набір трикутників для спрощення обробки;

– афінні перетворення, що включають масштабування, обертання, зсув об'єктів у просторі;

– видові перетворення, які включають трансформація координат сцени з урахуванням положення камери;

– проєкція вершин, тобто перетворення 3D-координат у 2D-координати екрану;

– відсікання невидимих ліній, тобто усунення частин об’єктів, що виходять за межі видимої області;

– визначення векторів нормалей, яке полягає в обчислені напрямків поверхонь для подальшого освітлення й зафарбовування.

Блок растеризації містить три етапи:

– зафарбовування шляхом визначення кольору кожного пікселя з урахуванням освітлення, нормалей і матеріалів;

– Текстурування, а саме накладання зображень на поверхні об’єктів;

– тидалення невидимих поверхонь, тобто усунення частин об’єктів, які перекриті іншими ближчими до камери.

Блок візуалізації містить один, заключний етап графічного конвеєра, а саме:

– відтворення кінцевого зображення, що забезпечує формування зображення, яке буде показано на екрані користувача.

Серед цих блоків блок растеризації, а саме його етап зафарбовування є одним з найважливіших у графічному конвеєрі, оскільки саме на цьому етапі визначається остаточний вигляд кожного пікселя зображення: його колір, яскравість, ефекти освітлення, текстурування тощо. Він напряму впливає на візуальну якість сцени та її реалістичність. У сучасних графічних системах цей етап виконується фрагментним шейдером, який може бути надзвичайно складним за логікою й ресурсозатратним у виконанні.

На практиці сцена у високоякісній графіці може містити мільйони фрагментів. Якщо кожному з них потрібно обчислити повноцінне освітлення, це призводить до величезного навантаження на графічний процесор. Особливо це стосується складних моделей з великою кількістю трикутників, де кожен фрагмент може мати унікальні властивості.

Для зафарбовування об'єктів використовуються наступні методи зафарбовування: плоского, Гуро, Фонга, трасування променів. Всі ці методи базуються на використанні інформації, що зберігається у вершинах полігонів. Метод плоского зафарбовування передбачає обчислення інтенсивності освітлення лише для однієї точки полігону, після чого весь полігон зафарбовується цим значенням. Метод Фонга виконує інтерполяцію нормалей усередині полігону з подальшим розрахунком освітлення для кожного пікселя. Метод Гуро передбачає обчислення інтенсивностей освітлення у точках полігону, виходячи з інтенсивностей, визначених у його вершинах.

Отже, оскільки етап зафарбовування є одним із найресурсоємніших у графічному конвеєрі, актуальною залишається розробка методів та програмних засобів, що дозволяють скоротити обчислювальні витрати цього етапу без значної втрати якості зображення, забезпечуючи тим самим баланс між продуктивністю та реалістичністю візуалізації.

1.2 Аналіз методів рендерингу

Рендеринг є невід'ємною частиною процесу візуалізації комп'ютерної графіки, що передбачає створення зображень на основі 3D-моделей. Існують різні методи рендерингу, кожен з яких має свої сильні та слабкі сторони.

Метод Гуро – це алгоритм зафарбовування поверхонь у комп’ютерній графіці, який забезпечує візуально реалістичне відтворення об’єктів [5]. Він полягає в тому, що кожній вершині моделі присвоюється колір, а потім ці значення кольору плавно інтерполюються по площині полігонів. Таке рішення дає змогу швидко отримувати гладкі переходи відтінків, але для його ефективної реалізації в програмних засобах високопродуктивного рендерингу виникають певні технічні виклики. На рисунку 1.2 зображено приклад зафарбовування методом Гуро.

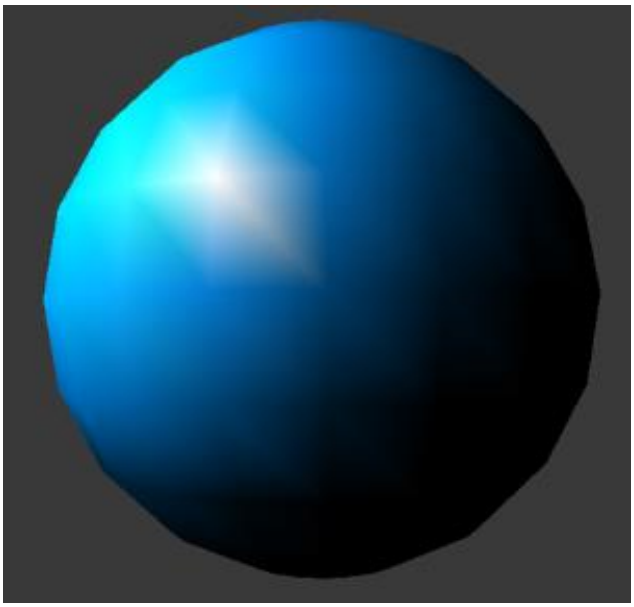

Рисунок 1.2 – Рендеринг методом Гуро

Оскільки освітлення обчислюється лише для вершин, а не для кожного пікселя, метод Гуро значно швидший за методи, які враховують освітлення на кожному пікселі. Оскільки метод інтерполює тільки освітлення на вершинах, він не враховує складні деталі освітлення на криволінійних або вигнутих поверхнях. Це може призводити до появи артефактів або вигляду різких переходів між

полігонами, тобто появи смуг Маха, особливо на об'єктах з високою кривизною. Завдяки простоті й швидкості виконання метод Гуро широко застосовується в тривимірній графіці, особливо при генерації динамічних та інтерактивних зображень, де питання оптимізації його продуктивності залишається вкрай актуальним.

Метод зафарбовування Фонга [6] – це техніка зафарбовування поверхонь у комп'ютерній графіці, яка є вдосконаленням методу Гуро. На рисунку 1.3 зображено приклад зафарбовування методом Фонга.

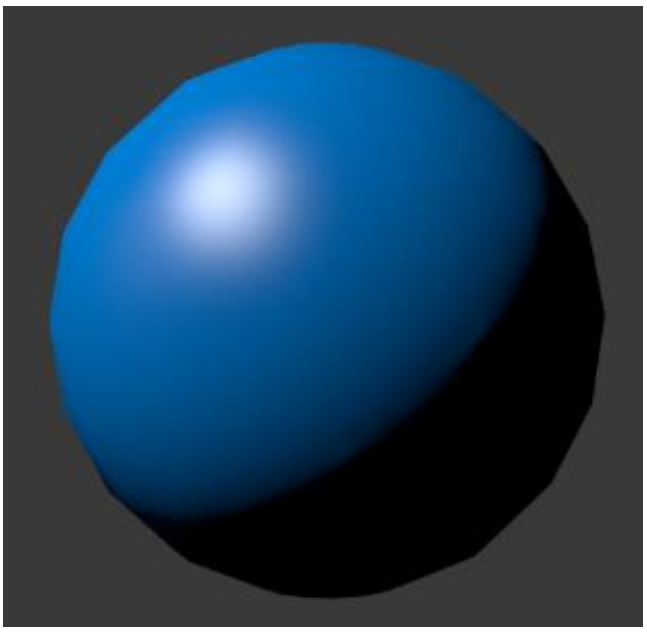

Рисунок 1.3 – Рендеринг методом Фонга

Метод Фонга дозволяє досягти більш реалістичних зображень, ніж плоске зафарбовування або метод Гуро, особливо для відображення блискучих поверхонь і складних світлових ефектів. Він розроблений для покращення обробки світла та затінення на тривимірних об'єктах. Метод Фонга базується на інтерполяції нормалей поверхні, а не кольорів, як у методі Гуро. Це дозволяє досягти більш точного відображення освітлення на кожному пікселі поверхні. Метод Фонга вимагає більше обчислювальних ресурсів порівняно з методом Гуро, оскільки для кожного пікселя необхідно обчислювати освітлення на основі інтерпольованих нормалей. Це може знижувати продуктивність, особливо в

реальному часі. Параметри спекулярного освітлення в моделі Фонга залежать від кута між спостерігачем і поверхнею, що може призводити до варіацій ефектів блиску, якщо поверхня має складну форму або відображає світло від різних джерел.

Метод плоского зафарбовування– це один з найпростіших методів затінення в комп'ютерній графіці, який застосовується для надання кольору поверхням тривимірних моделей. На рисунку 1.4 зображено приклад зафарбовування пласким методом.

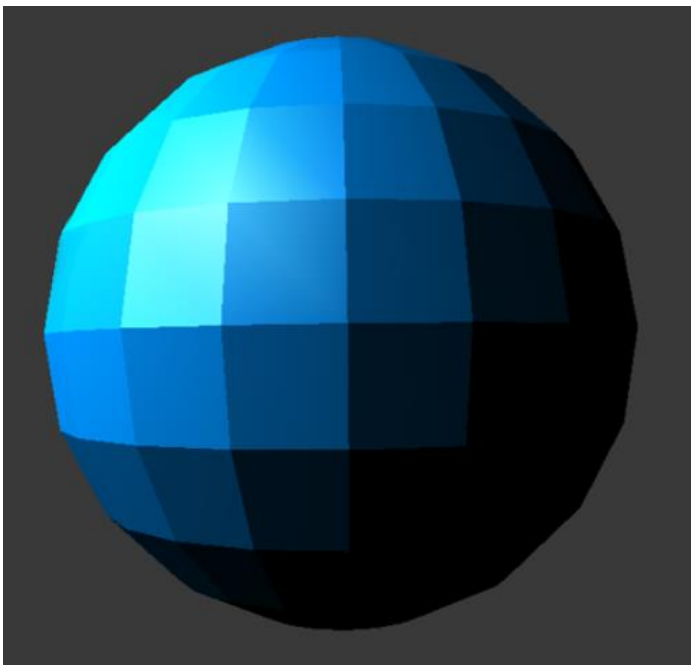

Рисунок 1.4 – Рендеринг методом плаского зафарбовування

У цьому методі кожен полігон моделі покривається єдиним кольором, що визначається для всієї поверхні. Оскільки метод застосовує один колір до всієї поверхні полігону, переходи між сусідніми полігонами виглядають дуже різкими і неприродними. Це може призвести до зламу на зображенні, де кожен полігон виглядає окремим шматком. Плоске зафарбовування не враховує вигини та деталі поверхні об'єктів. Це означає, що відсутні плавні переходи між тінями і освітленням, що робить зображення менш реалістичним порівняно з більш складними методами затінення. Цей метод не підходить для створення

фотореалістичних зображень або складних сцен, де важливі деталі освітлення та текстур. Плоске зафарбовування найкраще підходить для використання в ситуаціях, коли важлива швидкість рендерингу і простота, наприклад, для попереднього перегляду або в простих 3D-сценах. Метод забезпечує високу швидкість і простоту реалізації, але створює різкі, неприродні переходи та підходить лише для сценаріїв з обмеженими обчислювальними ресурсами, наприклад попередньої візуалізації або старих апаратних платформ.

Метод трасування променів [7] – це метод, що базується на симуляції шляху світла від ока глядача до джерел освітлення, враховуючи відбивання, заломлення та розсіювання на поверхнях. Завдяки такій детальній фізичній моделі він забезпечує надзвичайно високу якість зображення. На рисунку 1.5 зображено приклад зафарбовування методом трасування променів.

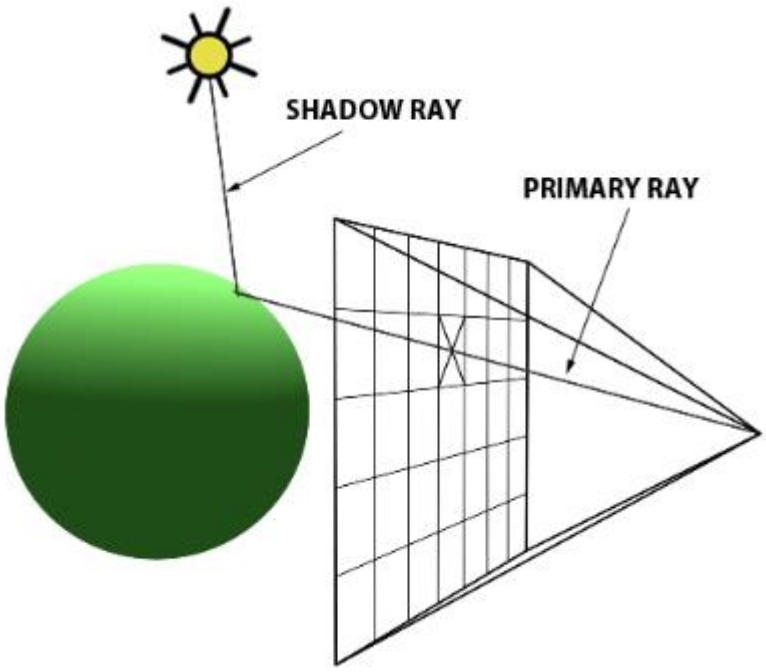

Рисунок 1.5 – Рендеринг методом трасування променів

Найбільшим недоліком методу є його велика обчислювальна складність. Кожен промінь, що відбивається або заломлюється, повинен бути трасований знову. Це може призвести до великої кількості обчислень, особливо в сценах з багатьма об’єктами та світловими джерелами. Потрібно трасувати багато

променів для кожного пікселя, а також враховувати всі ефекти, такі як тіні, відбиття, заломлення, що збільшує кількість обчислень у кілька разів. Технічні вимоги до апаратних засобів для ефективного використання трасування променів значно вищі, ніж для інших методі. Для досягнення високої якості зображення в реальному часі потрібен високопродуктивний графічний процесор і часто спеціалізоване обладнання. Для трасування променів може бути потрібно значно більше пам'яті та збереження інформації про сцену, що збільшує вимоги до системи. У випадку недостатньо великої кількості променів або обмеженого часу на обробку кадру, можуть виникати шуми або артефакти в зображеннях, що погіршують якість фінального результату. Це може бути вирішене за допомогою різних методів постобробки, але це збільшує обчислювальну складність. Метод трасування променів є потужним інструментом для досягнення фотореалістичних зображень, але його основними обмеженнями є висока обчислювальна складність і вимоги до ресурсів, що робить його менш придатним для реального часу.

Метод трасування променів з обмеженнями – це спрощений варіант методу трасування променів, який зазвичай використовується для рендерингу 3D-сцен і має менш складні обчислювальні вимоги порівняно з класичним трасуванням променів. Основна ідея методу полягає в тому, що для кожного пікселя на екрані створюється промінь, який проходить через сцену від камери, поки не зустрічає об'єкт. Технологія забезпечує ефективне і швидке обчислення для простих сцен, але її можливості з обмеженнями, зокрема у відтворенні складних світлових ефектів. Один з головних недоліків методу трасування променів з обмеженнями полягає в тому, що він не може точно відтворити складні світлові ефекти, такі як відбиття, заломлення або м'які тіні. У результаті зображення може виглядати менш реалістично порівняно з використанням більш складних методів рендерингу, таких як трасування променів з відбиттями. На рисунку 1.6 зображено приклад зафарбовування методом трасування променів з обмеженнями.

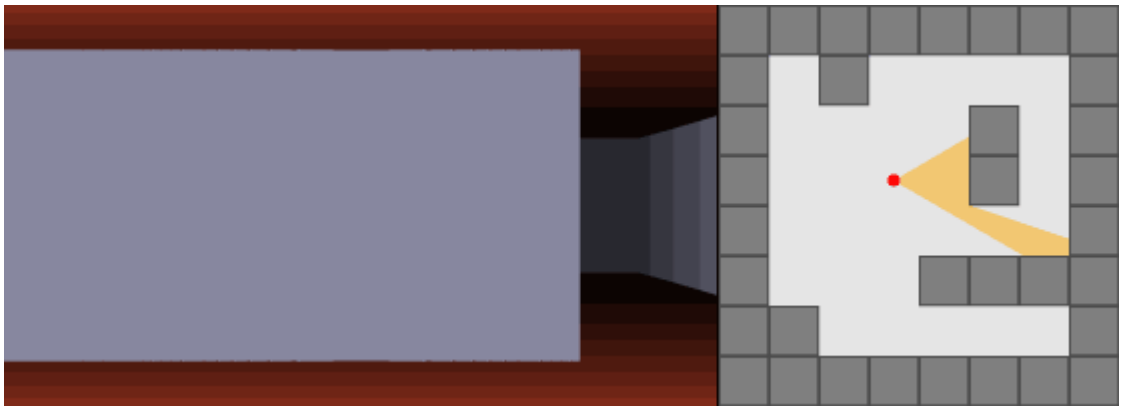

Рисунок 1.6 – Рендеринг методом трасування променів з обмеженнями

Оскільки метод не обчислює всі складні ефекти освітлення, він може давати менш детальні та менш реалістичні результати, особливо для сцен з відблисками, прозорими матеріалами чи складними тінями. Метод є ефективним та швидким підходом для рендерингу простих сцен, де важлива швидкість обробки і висока реалістичність. Однак, через обмеження у відображенні складних світлових ефектів і відсутність відбиттів, цей метод не підходить для високоякісної фотореалістичної візуалізації. Тому його найкраще застосовувати в реальних часах або в ситуаціях, де потрібна достатня швидкість, але не потрібна максимальна якість зображення.

Радіальне трасування – це метод рендерингу, який фокусується на моделюванні розповсюдження світла в сцені з врахуванням взаємодії між поверхнями. Основною метою цього методу є досягнення високоякісного освітлення, зокрема для створення природних тіней, м’яких переходів світла та кольору на різних поверхнях. Радіальне трасування широко використовується в області високоякісної графіки, зокрема для створення фотореалістичних зображень в архітектурній візуалізації, кінематографічній графіці та інших додатках, де важлива точність світлових ефектів і освітлення. Радіальне трасування вимагає значних обчислювальних ресурсів, оскільки для кожної сцени потрібно вирішити систему рівнянь для всіх поверхонь, що робить цей метод не дуже ефективним для реального часу. Враховуючи велику кількість обчислень, цей метод не підходить для складних, детальних сцен з великою

кількістю об'єктів, особливо в реальному часі. На рисунку 1.7 зображено приклад зафарбовування методом радіального трасування.

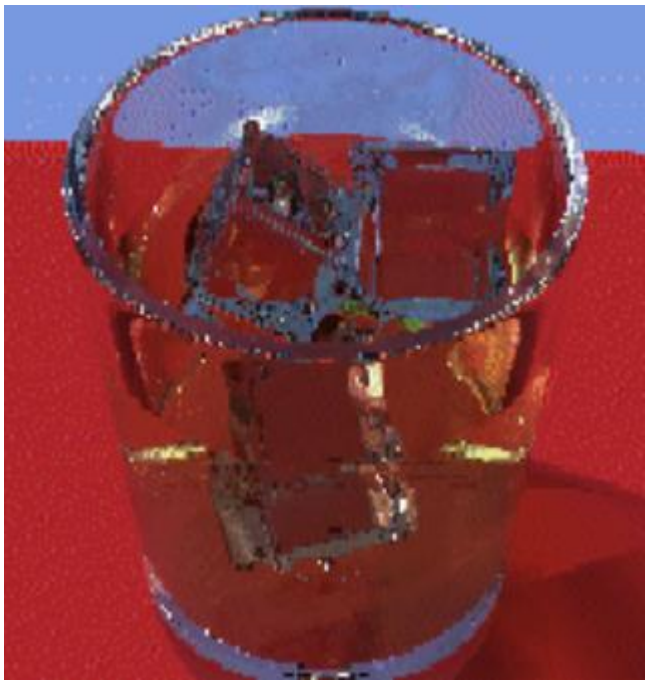

Рисунок 1.7 – Рендеринг методом радіального трасування

Перераховані методи рендерингу використовують у різних ситуаціях залежно від вимог до якості зображення та наявних обчислювальних потужностей.

Метод Гуро є оптимальним вибором для тих випадків, де необхідно досягти покращеної якості зображення в порівнянні з плоским зафарбовуванням, при цьому мінімізуючи додаткові обчислювальні витрати. Цей підхід особливо доречний для додатків, які вимагають гарної якості рендерингу, але працюють з обмеженими ресурсами, таких як мобільні ігри та інтерактивні програми. З іншого боку, метод Фонга є кращим для задач, де потрібна висока деталізація зображень, наприклад, для моделювання блискучих поверхонь або складних світлових ефектів.

В умовах сучасних вимог до одночасного збільшення швидкодії та візуального реалізму алгоритм Гуро залишається надзвичайно затребуваним. Подальша розробка інструментів і методик на його основі здатна суттєво

підвищити ефективність рендерингу. Крім того, поєднання цих підходів із потужностями сучасних обчислювальних платформ, включно з багатопоточними обчисленнями та апаратними акселераторами, відкриває нові можливості для формування високоякісних зображень у режимі реального часу.

1.3 Аналіз особливостей реалізації методу рендерингу Гуро

У процесі зафарбовування поверхні її поділяють на окремі трикутні площини. Для кожної такої площини задаються координати вершин та відповідні значення інтенсивності кольору. Одним із найпоширеніших підходів до реалізації зафарбовування за методом Гуро є алгоритм скануючого рядка. У межах цього алгоритму інтерполяція інтенсивностей кольору виконується спочатку вздовж ребер трикутника, а згодом уздовж горизонтальних рядків растеризації між відповідними ребрами.

Зафарбовування Гуро – це метод інтерполяції в комп’ютерній графіці, що дозволяє отримувати плавні градієнти освітлення на поверхнях, представлених у вигляді полігональних сіток з плоскими гранями. Суть підходу полягає в тому, що складні розрахунки кольорових компонентів виконуються лише для вершин полігона, а потім їхні результати лінійно інтерполюються всередині кожного трикутника. Завдяки цьому відеокарти можуть швидко відтворювати гладке світло без великих витрат ресурсів на обчислення для кожного пікселя.

Для реалізації цього методу виконуються такі кроки:

1. Геометрична обробка, коли для кожного вершини полігонального трикутника обчислюються просторові координати, нормалі та інші необхідні атрибути, після чого розраховується значення освітлення в цих точках відповідно до обраної моделі освітлення.

2. Інтерполяція кольору. Після завершення обробки вершин настає черга кроку растеризації, де трикутники перетворюються на пікселі екрану. Значення кольорів, отримані в вершинах, плавно поширюються по площі трикутника, так що кожен піксель всередині набуває відтінків, обчислених як комбінація кольорів його вершин.

3. Фрагментна обробка. Цей крок передбачає виконання остаточної обробки кожного фрагмента всередині трикутника. Після рівномірного розподілу кольору по його площі, за потреби, накладаються текстурні зображення, а в завершення можна додати додаткові візуальні прийоми.

4. Виведення зображення. Фінальним кроком є перетворення оброблених фрагментів на пікселі з врахуванням глибини, щоб зпопередити проблеми перекриття об’єктів на екрані.

На рисунку 1.8 зображено полігональний трикутник з позначеними вершинами A, B, C з інтенсивностями Ia, Ib та Ic.

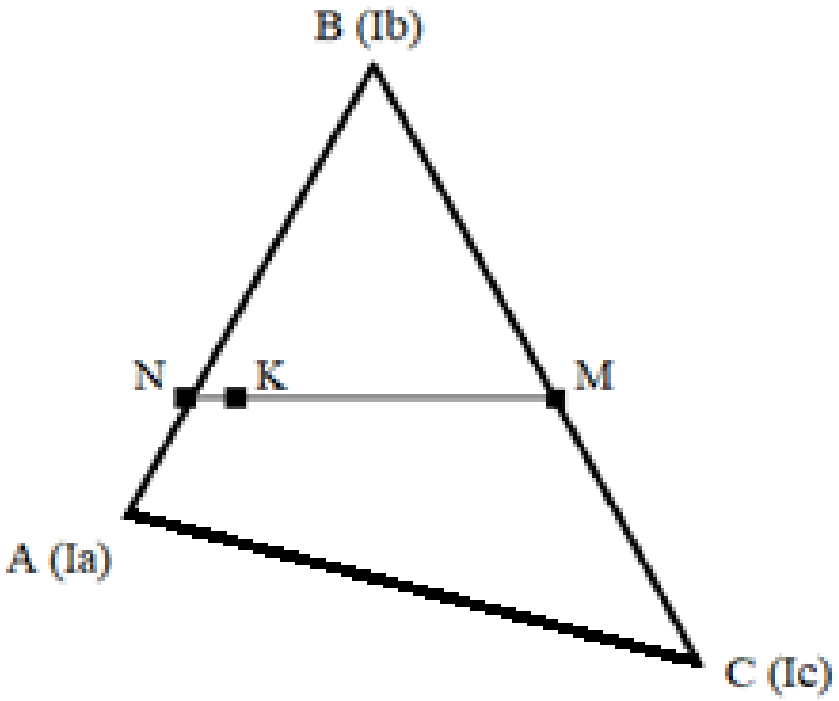

Рисунок 1.8 – Трикутний плігон із заданою на вершинах інтенсивністю

На основі значень інтенсивності кольору на вершинах (I_B, I_A, I_C) обчислюється значення інтенсивності кольору для інших пікселей трикутника:

$$\Delta I_{AB} = (I_B - I_A)/БП_{AB} \quad (1.1)$$

$$\Delta I_{BC} = (I_B - I_C)/БП_{BC} \quad (1.2)$$

$$I_N = I_B + \Delta I_{AB} \times БП_{BN} \quad (1.3)$$

У формулах 1.1-1.3 БП – більший прирост координат Δx, Δy заданого відрізка.

Аби визначити інтенсивність кольору у точці K, необхідно спершу розглянути лінію растеризації (NM), що проходить через цю точку. Для обчислення приросту інтенсивності в точці M використовується формула 1.4. Аналогічною буде формула для обчислення приросту інтенсивності в точці N.

$$I_M = I_B + \Delta I_{BC} \times БП_{BM} \quad (1.4)$$

Приріст інтенсивності вздовж лінії растеризації розраховується за формулою 1.5.

$$\Delta I_{NM} = (I_N - I_M)/БП_{NM} \quad (1.5)$$

Аби знайти інтенсивність кольору точки K використовується формула 1.6.

$$I_K = I_N + \Delta I_{NM} \times БП_{NM} \quad (1.6)$$

За даним прикладом знаходиться інтенсивність кольорів для всіх точок, які знаходяться на лінії растеризації, й аналогічно для будь-якої точки, що знаходиться в межах заданого полігону. Похибка буде мінімальною за умови поділу поверхні на прямокутні трикутники, чиї катети орієнтовані вздовж координатних осей, оскільки в такій конфігурації найбільша змінна катета збігається з його довжиною.

Під час зафарбовування трикутника сканувальна лінія поступово охоплює кожну точку полігона. Зміни, що відбуваються вздовж сканувальної лінії та між сусідніми лініями, визначаються величиною одного пікселя. Такий

підхід дає змогу замінити множення на більш ефективну операцію накопичувального додавання:

$$I_K = I_N + 3 \times \Delta I_{NM} \qquad (1.7)$$

На рисунку 1.9 показано приклад рендерингу трикутника з інтенсивностями світла в його вершинах: $IA = 5$, $IB = 9$ та $IC = 1$. Отримані значення світлових інтенсивностей відображені на підрисунку і обчислені за наведеними вище формулами.

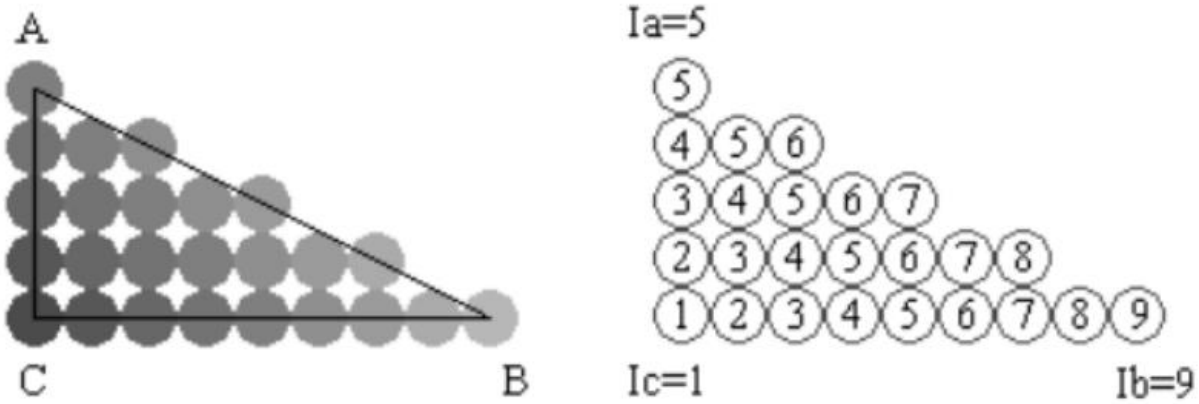

Рисунок 1.9 – Приклад зафарбовування методом Гуро

Під час детального аналізу об'єкта, зафарбованого методом Гуро, помітний характерний артефакт, а саме, смуги Маха (рисунок 1.10). Вони виникають через посилення контрасту між сусідніми відтінками на межах полігонів. Такий ефект свідчить про обмеження методу, адже він не завжди здатен згладити перепади яскравості на стиках багатокутних поверхонь [8].

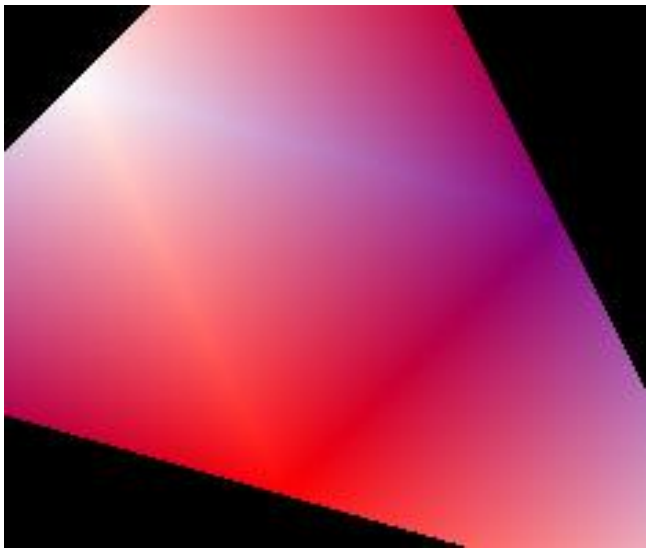

Рисунок 1.10 – Смуги Маха

На ринку представлено чимало фреймворків і бібліотек для створення зображень, однак далеко не всі вони забезпечують достатню точність і швидкодію при реалізації алгоритму зафарбовування Гуро. Більшість існуючих рішень розповсюджується на комерційній основі, що звужує коло потенційних користувачів. Проведений огляд наявних технологій демонструє необхідність створення власного високопродуктивного механізму зафарбовування багатокутних моделей за Гуро, а розробка спеціалізованого програмного забезпечення для цієї мети є ключовим кроком до підвищення оперативності та ефективності рендерингу.

1.4 Висновки

У першому розділі було виконано аналіз етапів графічного конвеєра, методів зафарбовування.

Окрім цього проведено аналіз послідовних кроків графічного конвеєра від формулювання сцени та виконання геометричних трансформацій до власне рендерингу та остаточного відображення кадру. Було виявлено, що найбільш ресурсоємним і важливим етапом у графічному конвеєрі є зафарбовування, оскільки воно вимагає точного визначення просторових координат кожної точки поверхні та обчислення її колірної інтенсивності.

Було встановлено характерні особливості методу зафарбовування Гуро, а також обґрунтовано розробку методів покращення якості, продуктивності та реалістичності методу зафарбовування Гуро.

Визначено мету та задачі дослідження.

2 РОЗРОБКА МЕТОДІВ ДЛЯ ПІДВИЩЕННЯ ПРОДУКТИВНОСТІ ТА РЕАЛІСТИЧНОСТІ МЕТОДУ ГУРО

2.1 Модифікація методу Гуро шляхом додаткової триангуляції трикутника

Триангуляція – це процес розбиття площини на скінченну кількість трикутників, які не накладаються один на одного та мають спільні сторони й вершини, утворюючи безперервне з'єднання між собою [9]. У загальному вигляді задача триангуляції зводиться до з’єднання заданих точок ребрами, які не перетинаються між собою. Оптимальною вважається така триангуляція, для якої загальна довжина всіх ребер є найменшою серед усіх можливих варіантів триангуляцій, побудованих на основі одного й того ж набору вхідних точок.

Прикладом оптимальної триангуляції є трикутник Серпінського [10]. Для тріангуляції Серпінського першого порядку в трикутнику ABC (рисунок 2.1) проведено середні лінії: KL, LM, KM.

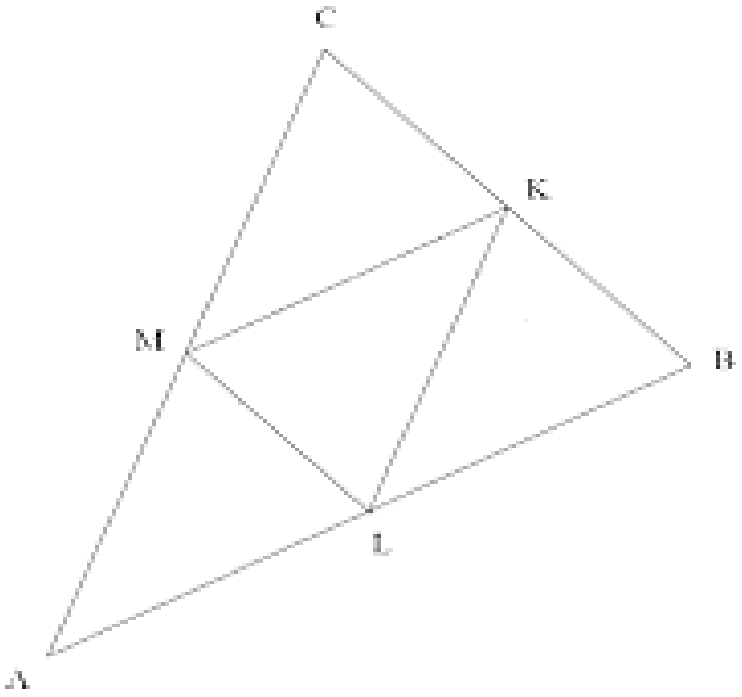

Рисунок 2.1 – Трикутник Серпінського

Доведемо рівність трикутників AML і LKB. Задані трикутники є рівними, тому що їх відповідні сторони рівні:

$$AM=KL=0.5\times AC \quad (2.1)$$

$$LM=KB=0.5\times CB \quad (2.2)$$

$$AL=LB=0.5\times AB \quad (2.3)$$

Таким самим чином доводиться рівність усіх інших трикутників, утворених середніми лініями KL, LM, KM. У результаті отримано чотири рівні трикутники та один з яких є дзеркально відображеним (рисунок 2.2).

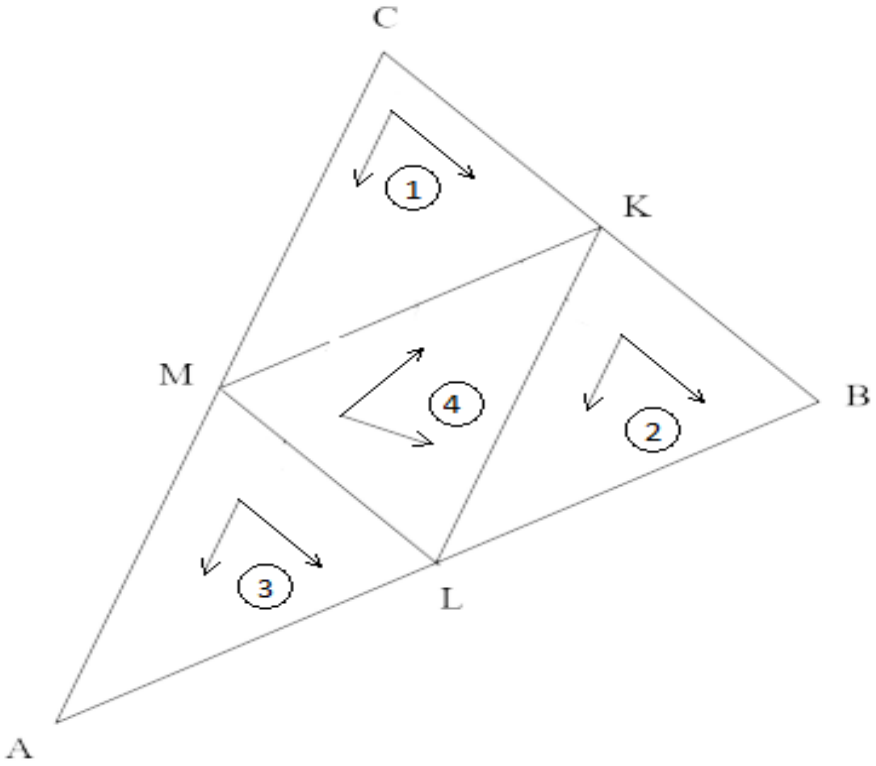

Рисунок 2.2 – Напрямки відображення трикутників

З огляду на те, що зміна кольорової інтенсивності всередині трикутника залишається сталою, можна виконати обчислення лише для одного з утворених трикутників, після чого використати ці ж результати для інших. Інакше кажучи, достатньо здійснити растеризацію тільки верхнього трикутника CMK, а потім

аналогічним чином обробити решту внутрішніх трикутників. Варто зауважити, що трикутники 1, 2 і 3 мають однаковий напрямок заливки, тоді як для трикутника 4 цей напрямок є зворотним.

Зафарбовування завжди починається з верхнього трикутника. Спершу для кожної вершини трикутника за відомими значеннями інтенсивностей визначаються константи зсуву.

Для вершини M:

$$\begin{cases} \Delta X = (X_A - X_C)/2 \\ \Delta Y = (Y_A - Y_C)/2 \\ \Delta I = (I_A - I_C)/2 \end{cases} \quad (2.4)$$

Для вершини K:

$$\begin{cases} \Delta X = (X_B - X_C)/2 \\ \Delta Y = (Y_B - Y_C)/2 \\ \Delta I = (I_B - I_C)/2 \end{cases} \quad (2.5)$$

Для вершини C:

$$\begin{cases} \Delta X = (X_A - X_B)/2 \\ \Delta Y = (Y_A - Y_B)/2 \\ \Delta I = (I_A - I_B)/2 \end{cases} \quad (2.6)$$

Застосування даного методу дозволяє скоротити час зафарбовування вчетверо: малий трикутник займає лише чверть площі оригіналу, а обробка решти сегментів відбувається паралельно й не додає часових витрат.

Оскільки для тріангуляції початкового трикутника в дискретній системі використовуються середини його сторін, то дійсна середина сторони трикутника може не потрапити в точку на сітці зображення, якщо прирости координат не є парними. Для подолання цієї проблеми необхідно, щоб приріст за X, Y для

кожної пари точок був парним. Для цього для будь-якої пари точок має бути дотримана хоча б одна з вимог:

– X для усіх точок парний, Y теж парний;

– X для усіх точок парний, Y непарний;

– X для усіх точок непарний, Y парний;

– X для усіх точок непарний, Y теж непарний.

Щоб задовольнити ці вимоги, у молодший розряд координат X та Y кожної точки сітки записують «0» для вирівнювання за парністю або «1» для вирівнювання за непарністю.

На рисунку 2.3 наведено приклад зафарбовування об’єкта згідно запропонованого методу.

Рисунок 2.3 – Зафарбовування об’єкта з використанням додаткової триангуляції

Нормалізація для досягнення даних вимог може викликати спотворення структури зображення, оскільки для її реалізації буде проводитися зміна координат точок сітки, проте значення інтенсивності залишаться незмінними.

Через дискретну природу зображень, обмежену точність алгоритмів їх побудови та фізіологічні можливості людського зору, операція нормалізації полігональної сітки не спричинить помітних або суттєвих спотворень об’єктів.

У такому разі похибка обчислень залишається на рівні класичного методу рендерингу. Проте варто врахувати, що нормалізація виявляє свої обмеження при роботі з дуже дрібними деталями, що є типовою складністю для більшості рендерингових алгоритмів.

2.2 Використання методу Фонга для підвищення реалістичності методу Гуро

У процесі зафарбовування графічних примітивів важливо забезпечити високу швидкість обробки графіки при збереженні прийнятного рівня якості. Для цього можна розбити лінію растеризації на цифрові сегменти, кратні степеням двійки за методом Фонга [11], а в проміжних пікселях цих сегментів використовувати метод Гуро для визначення інтенсивності кольору (рисунок 2.4).

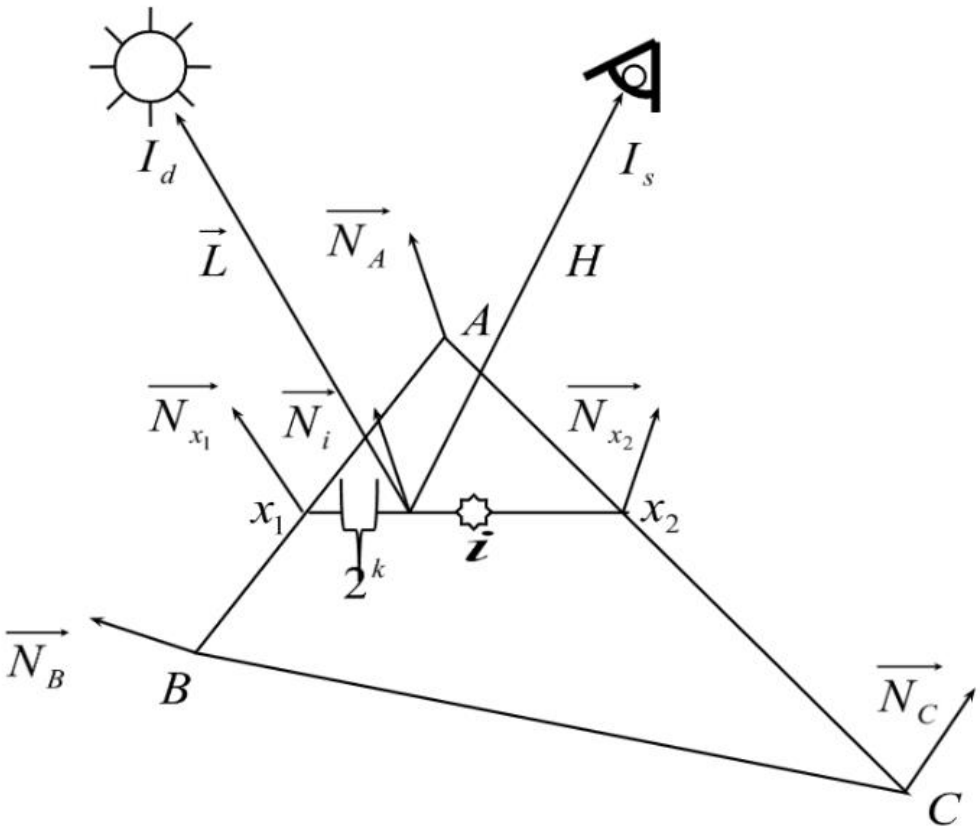

Рисунок 2.4 – Визначення параметрів зафарбовування за Фонгом

Як наслідок, обчислювальні операції значно спрощуються, оскільки множення або ділення на такі значення можна замінити простим побітовим зсувом.

Отже ряди растеризації трикутника розбито на хвилі довжиною 2^k, після чого визначено точні значення кольорових інтенсивностей у їхніх кінцевих точках, а потім, використовуючи лінійну інтерполяцію між цими граничними значеннями, обчислюються інтенсивності для всіх внутрішніх точок сегментів (рисунок 2.4).

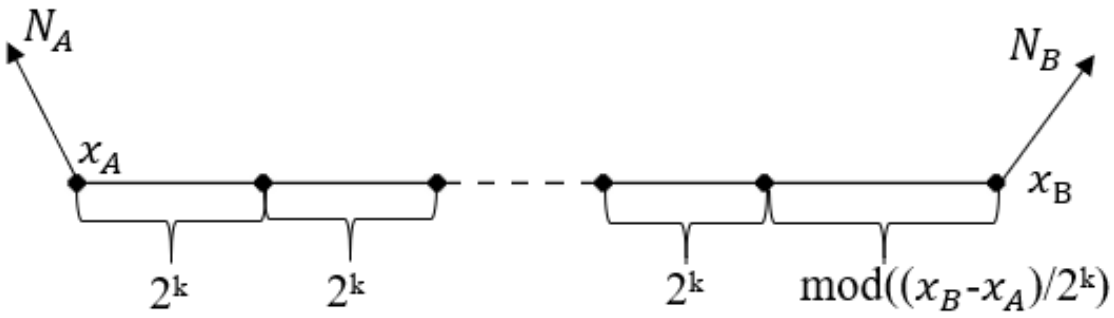

Рисунок 2.5 – Рядок растеризації, поділений на цифрові сегменти

Якщо довжина скан-рядка дорівнює x_B-x_A, то крок зміни вектора нормалі ΔN обчислюється:

$$\Delta N_{AB} = (N_B\text{-}N_A) / (x_B\text{-}x_A), \qquad (2.7)$$

де N_A і N_B – це вектори нормалей в правій та лівій точках ряду растеризації відповідно.

Для сегментів, довжина яких 2^k, приріст буде визначатися:

$$\Delta N = \Delta N_{AB} \times 2^k = (N_B\text{-}N_A)\frac{2^k}{x_B - x_A}, \qquad (2.8)$$

За допомогою лінійної інтерполяції легко отримати нормалі в крайніх точках кожного цифрового сегменту. Зокрема, у правій кінцевій точці n-го сегмента вектор нормалі обчислюється як значення вектора нормалі в початковій точці сегмента, При цьому множення на 2^k можна замінити послідовними додаваннями для зниження обчислювальних витрат. Для кожного сегмента інтенсивність кольору легко знаходиться за методом Гуро.

На рисунку 2.6 наведено приклад зображення, зафарбованого з використанням запропонованого методу підвищення реалістичності методу Гуро.

Рисунок 2.6 – Рендеринг зображення методом Гуро з використанням методу Фонга для підвищення реалістичності

Також приклад зображення, зафарбованого з використанням запропонованого методу підвищення реалістичності методу Гуро наведено на рисунку 2.7.

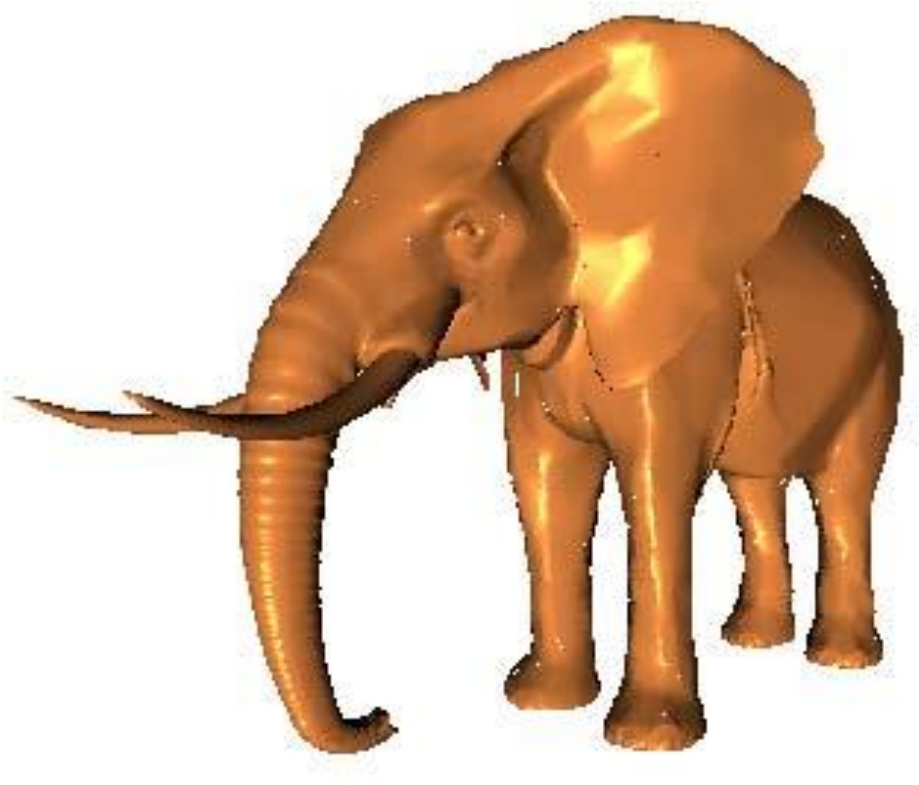

Рисунок 2.7 – Рендеринг зображення методом Гуро з підвищеною методом Фонга реалістичністю

2.3 Метод усунення смуг Маха

Смуги Маха – ілюзорні артефакти, які спричиняють посилення країв на гранях, що містять розрив градієнта інтенсивності світла. Вони виникають через особливості сприйняття людським оком (рисунок 2.8).

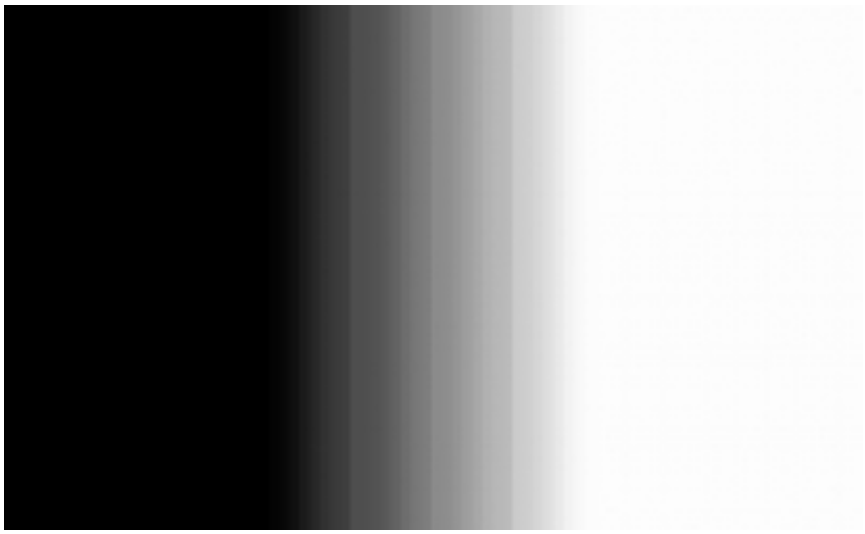

Рисунок 2.8 – Ефект смуг Маха

Тож сприйняття є важливим фактором, який слід враховувати при створенні та зафарбовуванні об’ємних зображень методами комп’ютерної графіки. Межі трикутників, що стають видимими при зафарбовуванні методом Гуро, приписуються феномену, названому на честь психолога Ернеста Маха, смугам Маха, майже всіма дослідженими текстами комп’ютерної графіки [5, 9, 12, 13 (першоджерело)], і навіть самим Анрі Гуро. Вирішення проблеми появ смуг Маха приписують методу рендерингу Фонга, хоча сам Фонг зазначав, що смуги все ще видно в деяких ситуаціях екстремальної геометрії.

Існують моделі зорової системи, які імітують процеси людського сприйняття й пояснюють підсилення контрасту біля країв об’єктів. Ранні підходи припускали наявність спеціалізованих нейронів, чутливих до орієнтації ліній, розміру, глибини, кольору та руху, і вважали, що латеральне гальмування, коли сусідні нейрони взаємно пригнічують одне одного, відповідальне за виникнення ефекту смуг Маха [14]. Дослідження 90-х років показують, що видимість цих смуг більше залежить від фазової характеристики сигналу в частотній області, а не від звичайного градієнту інтенсивності. Існує також протилежний ефект – коли тонкі темні лінії роблять однорідну ділянку темнішою, а світлі лінії – світлішою. Поєднана активність численних нейронів дає глобальні візуальні наслідки, що, зокрема, використовуються для створення відчуття глибини в випадкових точкових стереограмах.

Метод Фонга використовує інтерполяцію нормалей поверхні для отримання плавних переходів кольорів і ефективного усунення смуг Маха. Також, застосування синусоїдальної функції інтерполяції при побудові колірного простору дозволяє ще більше зменшити видимість цих артефактів. Отже, існують стратегії інтерполяції, які допомагають досягти бажаного рівня гладкості зображення.

Середня видимість смуг Маха залежить від кольору й є характерною для наступних кольорів: синього, зеленого, блакитного, червоного, пурпурного, жовтого та білого (рисунок 2.9).

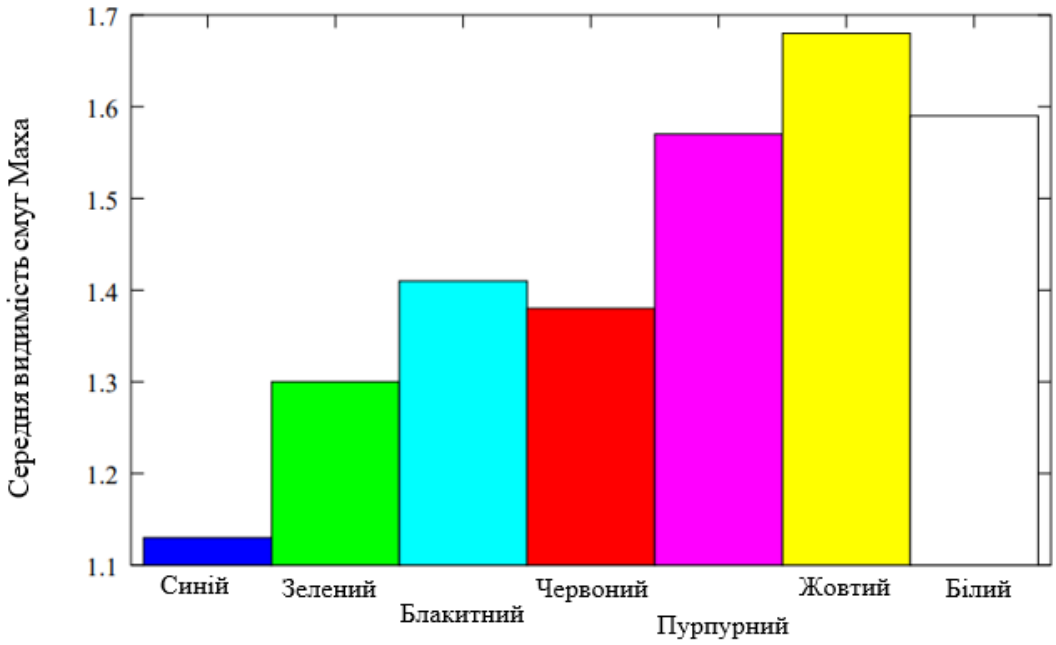

Рисунок 2.9 – співвідношення між видимістю смуг Маха та кольорами

Видимість смуг Маха можна зменшити, маніпулюючи змінними факторами, такими як:

– ширина області вимірювання, а саме відрізок, над яким оцінюється інтенсивність. Ця ширина задається як частка вікна w, у межах якого виконується інтерполяція. Інтерполяційна смуга завжди розташована посеред вікна, яке візуально займає кут приблизно 10° при відстані 50 см від екрану;

– глибина неперервності, яка позначається n у формулі інтерполяції. Параметр t змінюється від 0 до 1 вздовж ширини w смуги. Функція гарантує, що розриви з’являються лише на межах (t=0 та t=w) і лише після n-го порядку диференціації. Чим більше значення n, тим плавнішим буде градієнт інтенсивності:

$$I_{n,w}(t) = \begin{cases} 0 & t < 0 \\ \begin{cases} \frac{t}{w} & n = 1 \\ 3\left(\frac{t}{w}\right)^2 - 2\left(\frac{t}{w}\right)^3 & n = 2 \\ 10\left(\frac{t}{w}\right)^3 - 15\left(\frac{t}{w}\right)^4 + 6\left(\frac{t}{w}\right)^5 & n = 3 \\ 35\left(\frac{t}{w}\right)^4 - 84\left(\frac{t}{w}\right)^5 + 70\left(\frac{t}{w}\right)^6 - 20\left(\frac{t}{w}\right)^7 & n = 4 \end{cases} & 0 \le t < w \\ 1 & w \le t \end{cases} \quad (2.9)$$

На рисунку 2.10 показані перші чотири варіанти цієї функції, побудовані на основі нормалізованої площі під кривими Без’є ступеня 2n – 1.

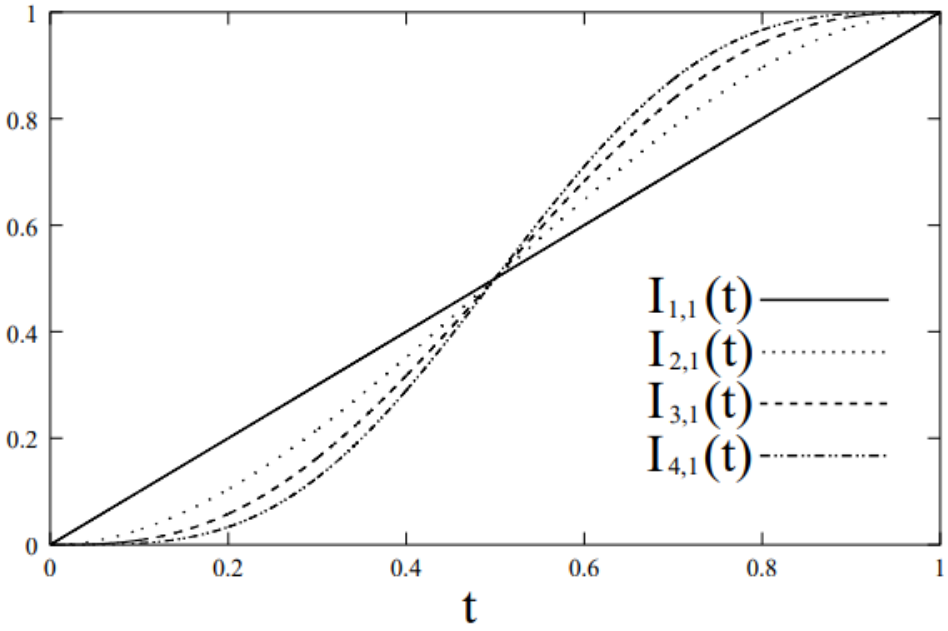

Рисунок 2.10 – Перші чотири рівні функції інтерполяції

Видимість смуг Маха падає при збільшенні ширини області вимірювання, а також плавності інтерполяційної функції. Смуги Маха стають помітними при використанні затінення Гуро, особливо в місцях, де сходяться різні інтенсивності освітлення.

Щоб зменшити цей ефект, можна застосувати більш складні інтерполяційні функції, які забезпечують плавніший перехід між градієнтами. Ціною використання цього способу буде збільшення часу зафарбовування Другим варіантом є використання інтерполяції високого порядку лише в вузькій смужці вздовж граней полігона де ефект Маха найпомітніший, а для внутрішньої частини залишити класичне Гуро-затінення. У реалізації другого варіанту спочатку визначають «кордонну зону» вздовж ребер трикутника для інтерполяції з корекцією Смужок Маха, а потім для решти площі полігона застосовують звичайне Гуро-затінення (рисунок 2.11).

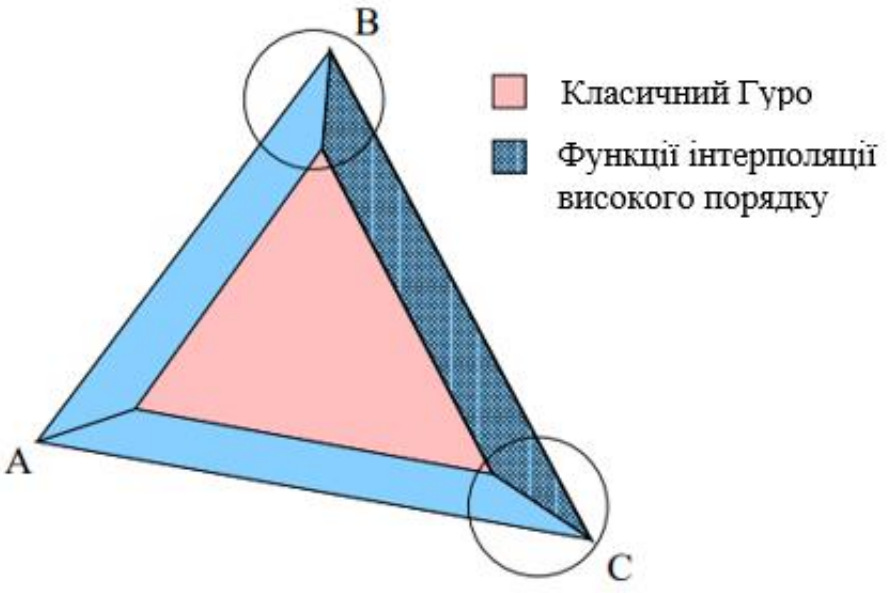

Рисунок 2.11 – Корегування смуг Маха комбінуванням способів зафарбовування

Ширину кордонної зони можна гнучко налаштувати для отримання найкращих результатів. Щоб виявити ці області, обчислюють мінімальне значення ваги в рівнянні:

$$I_P = w_A I_A + w_B I_B + w_C I_C \qquad (2.10)$$

Якщо цей мінімум опускається нижче заданого порогового значення, вершину з найменшою вагою вважають тією, що лежить навпроти границі регіону.

Функція затінення вздовж прикордонної смуги повинна забезпечувати принаймні сім рівнів неперервності в точці t=0 на ребрі трикутника і плавно переходити до Гуро-інтерполяції в точці t=6. Відомо, що похідні Гуро-функції до n-го порядку існують і можуть бути використані для узгодження на рівні межі. Зовні трикутника є відомим лише значення інтенсивності I_E, тому всі вищі похідні в точці t=0 слід зафіксувати рівними нулю, щоб гарантувати суцільність і стикування обох частин функції:

$$J_{n,b}(t) = tI_{n,b}(t)(I_A - I_E) + I_E, \quad (2.11)$$

де $I_{n,b}(t)$, визначена у формулі 2.9, демонструє бажаний результат.

Інтенсивність точки P в межах межі області протилежній вершині A задається формулою:

$$I_P = J_{n,b}(w_A) \quad (2.12)$$

Для ідеального узгодження градієнтів інтенсивності на спільних ребрах сусідніх трикутників потрібні складні обчислення, які враховують, як змінюється освітлення в обох полігонах. Натомість можна застосувати наближення, зберігаючи основну ідею Гуро рендерингу. Спочатку необхідно обчислити інтенсивності всіх вершин перед початком інтерполяції. Тоді значення інтенсивності в сусідніх точках (наприклад, у вершині D на рисунку 2.12) вже будуть відомі до растеризації полігонів.

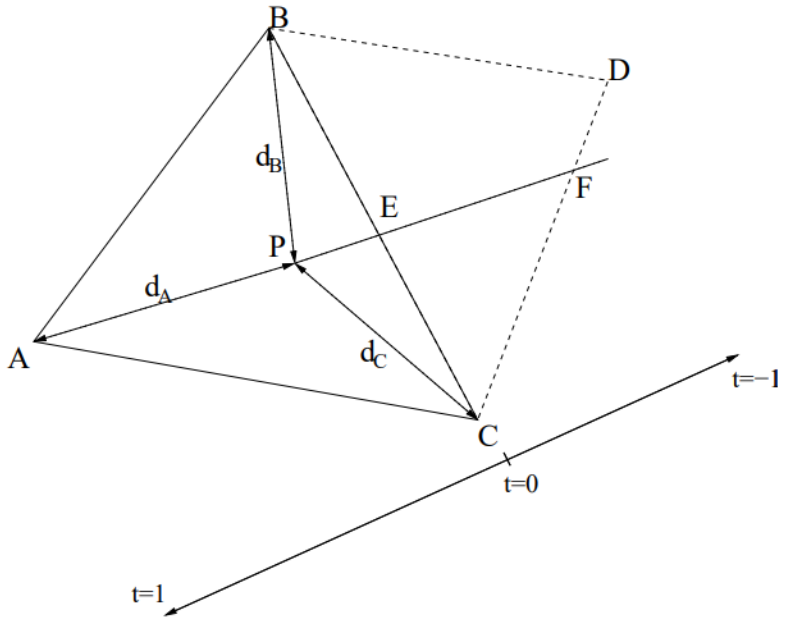

Рисунок 2.12 – Розрахунок інтенсивності в точці P

Інтенсивність I_F визначається лінійною інтерполяцією між інтенсивностями крайніх вершин, причому вага кожної з них залежить від відстані до цієї точки вздовж спільного ребра. Приклад обчислення для випадку, коли $w_B > w_C$, наведено у рівняннях:

$$a = \frac{2w_B}{w_B + w_C} - 1 \tag{2.13}$$

$$f = \frac{2a}{1+a} \tag{2.14}$$

$$I_F = fI_B + (1-f)I_D \tag{2.15}$$

Коли сусідній трикутник не є точним дзеркальним відображення, а лише наближеним, помилка в обчисленні I_F збільшується зі зростанням відхилення вершини D від її ідеального положення. Щоб забезпечити коректний градієнт уздовж межі, інтерполяційну формулу можна модифікувати так:

$$J_{n,b}(t) = tI_{n,b}(t)(I_A - I_E) + tI_{n,b}(b-t)\left(\frac{I_F - I_E}{2}\right) + I_E \tag{2.16}$$

Кордонні зони не мають фіксованої ширини – вони звужуються до рядків на кінцях . Тому параметр «ширина межі», який використовувався вище, слід коригувати під час зафарбовування точок, що розташовані безпосередньо поблизу вершини трикутника.

Навколо кожної вершини виникають особливі умови, які потребують окремої уваги. Вершина з максимальною інтенсивністю освітлення створює локальний екстремум, що підсилює ефект смуг Маха. Причина в тому, що інтерполяційні градієнти стикаються під різними кутами, а вплив сусідніх вершин у цій зоні зменшується.

Щоб узгодити кути стиків градієнтів у суміжних трикутниках, вводять вагу на основі евклідової відстані від вершини. Оскільки довжини двох ребер, що сходяться в одній точці, можуть відрізнятися, застосовують еліптичне спотворення, але воно часом призводить до зсуву контурів уздовж спільного ребра.

Цей зсув можна виправити, помноживши початкові лінійні ваги на косинус кута між ребрами, що дає чітку стиковку. Оновлені ваги вершин обчислюють із урахуванням відстані до вершини:

$$d_A(P) = 1 - \frac{|AP|^2}{\left[|AB|\frac{w_B^2}{w_B^2+w_C^2} + |AC|\frac{w_C^2}{w_B^2+w_C^2})\right]^2} \qquad (2.17)$$

Вагові коефіцієнти, розраховані за відстанню, не нормалізуються і можуть набувати від'ємних значень поблизу протилежних ребер. Проте це практично не впливає на результат, оскільки така поведінка проявляється лише в невеликій зоні довкола кожної вершини.

Пікселі, затінені за Гуро, демонструють різкі зміни яскравості навіть на невеликій відстані від грані полігона. Саме це створює враження, ніби краї трикутника світліші за його центр. У випадку Фонгового зафарбовування такий контраст значно менший. За класичною локальною моделлю освітлення, інтенсивність у довільній точці P обчислюється за формулою:

$$I_P = I_{amb} + k_{diff}(N \times L) + k_{spec}(N \times S)^n, \qquad (2.18)$$

де N – нормалізоване нормальне векторне поле в цій точці, а L та S – нормалізовані вектори напрямку на джерело світла й на спостерігача, а решта параметрів описує властивості матеріалу та джерела світла.

Віддзеркалене відбиття є незначним поза зоною блиску, тому для подальшого виведення її можна опустити. А взаємозв’язок між двома методами зафарбовування можна спостерігати в наступному:

$$I_{Guard} = w_A I_A + w_B I_B + w_C I_C \simeq I_{amb} + k_{diff}(N_{Phong})L \quad (2.19)$$

$$I_{Phong} = I_{amb} + k_{diff}\left(\frac{N_{Phong}}{|N_{Phong}|}\right)L \simeq \frac{I_{Guard}}{d}, \quad (2.20)$$

де $N_{Phong} = w_A N_A + w_B N_B + w_C N_C$ зважений вектор нормалі, який потрібно нормалізувати перед тим, як застосовувати у розрахунках освітлення.

Розподіл яскравості при затіненні за методом Фонга можна подати у вигляді формули, що базується на масштабованих коефіцієнтах, залежних лише від лінійного зважування та характеристик вершин. Це наближення є найбільш ефективним у зонах, де переважає розсіяне освітлення. Для областей, де помітні впливи навколишнього середовища або дзеркальні відбиття, можна побудувати подібні моделі з урахуванням цих факторів.

На рисунку 2.13 порівнюються результати застосування зафарбовування Гуро з фіксованою прикордонною зоною та стандартного шейдера.

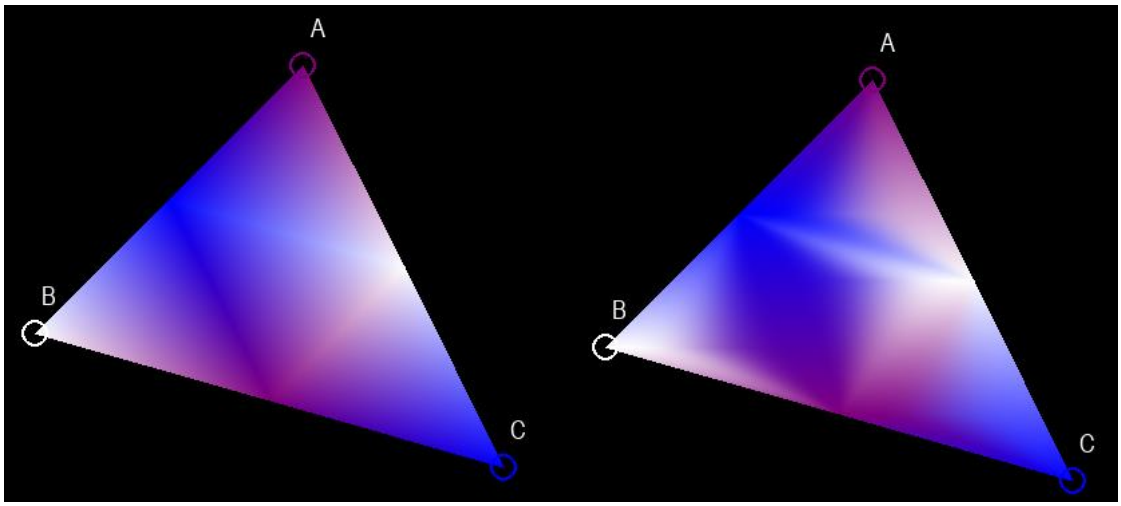

Рисунок 2.13 – Виправлене граничне зафарбовування

Незважаючи на корекцію, смуги Маха все ще проявляються, особливо там, де сусідні трикутники мають різко відмінні градієнти. Зі збільшенням рівня неперервності інтерполяційної функції смуги стають ширшими й розмитішими, а їхня частота спадає, проте краї трикутників лишаються помітними. Крім того, налаштування широких прикордонних областей призводить до нових артефактів уздовж ребер, у вигляді великих ділянок однорідної яскравості, що робить стики копланарних трикутників добре помітними. Включення вершинної корекції значно згладжує краї й зменшує ці ефекти.

2.4 Висновки

У другому розділі модифіковано метод Гуро шляхом додаткової тріангуляції, використано ітераційні методи Фонга, що забезпечить вищу продуктивність рендерингу зображень. Також розроблено методи усунення смуг Маха, що виникають при реалізації зафарбовування методом Гуро. Досягнено висновків, що:

– чим вища неперервність інтерполяційної функції, тим менш помітні смуги, але тим ширші та розмитіші вони стають.

– вершинна корекція суттєво зменшує артефакти, однак повністю їх не усуває;

– використання комбінованого зафарбовування дозволяє досягнути компромісу між якістю та продуктивністю.

3 РОЗРОБКА ПРОГРАМНИХ ЗАСОБІВ ДЛЯ ПОКРАЩЕНОГО ЗАФАРБОВУВАННЯ ГУРО

3.1 Обґрунтування вибору засобів для реалізації

Для створення програми зафарбовування полігонів методом Гуро було обрано мову програмування Pyton. В таблиці 3.1 коротко наведено порівняльний аналіз мов програмування, які також можуть бути використані для реалізації.

Таблиця 3.1 – Порівняльний аналіз мов програмування

Критерій	Python	C++	Java	JavaScript
Простота використання	+	-	-	+
Продуктивність	+	+	-	-
Підтримка паралельних обчислень	+	+	+	-
Швидке інтерактивне тестування	+	-	-	-
Інтеграція з іншими мовами	+	+	+	+

Python [15] – це універсальна мова програмування високого рівня, яка робить акцент на читабельності коду завдяки суттєвим відступам.

Вона використовує динамічну типізацію та автоматичне збирання сміття і підтримує різні парадигми: структурне, об’єктно-орієнтоване та функціональне програмування. Завдяки багатій стандартній бібліотеці Python часто називають «мовою з усім необхідним у комплекті».

Розробку почав Гвідо ван Россум наприкінці 1980-х як наступницю мови ABC, а перший реліз Python 0.9.0 відбувся в 1991 році. У 2000-му вийшла версія 2.0, а значна модернізація Python 3.0, яка не повністю зберегла зворотну сумісність, побачила світ у 2008. Останньою гілкою Python 2 стала версія 2.7.18, випущена в 2020 році.

Python нерухомо утримує високі позиції в рейтингах популярності мов програмування та здобув особливе визнання в спільноті машинного навчання.

Python має один з найбільш простих синтаксисів в порівнянні з іншими мовами програмування, що дозволяє розробникам зосереджуватися на логіці програми, а не на технічних деталях. Завдяки високому рівню абстракції, Python дозволяє швидко прототипувати та експериментувати з алгоритмами, що важливо для методів, таких як зафарбовування методом Гуро.

Python зазвичай повільніший за C++ через інтерпретовану природу, але це можна компенсувати, використовуючи високопродуктивні бібліотеки, написані на C. Оскільки Python зручний для експериментування, він може бути ідеальним для прототипування, де швидкість розробки важливіша за максимальну продуктивність.

Python має величезну кількість бібліотек для наукових обчислень, роботи з графікою та паралельних обчислень. Це робить Python дуже зручним для графічних і обчислювальних задач. Python підтримує паралельні обчислення через бібліотеки, що дозволяє ефективно розподіляти завдання на кілька ядер процесора. Однак через глобальний інтерпретатор Python паралельність в деяких випадках може бути обмежена, що знижує ефективність.

Python надає найкращі умови для швидкого прототипування та тестування завдяки інтерактивному середовищу. Це дозволяє легко тестувати окремі частини алгоритмів і змінювати їх на ходу.

З огляду на всі перераховані критерії, Python є найкращим вибором для реалізації методу Гуро через свою простоту, доступність бібліотек, можливість використання високопродуктивних функцій через бібліотеки на C, а також легкість інтеграції з іншими мовами та середовищами. Завдяки цьому, Python

дозволяє швидко розробляти і тестувати складні алгоритми для рендерингу та ефективно використовувати ресурси для досягнення високої продуктивності при рендерингу об'ємних зображень.

Середовищем програмування створення програми зафарбовування полігонів методом Гуро з використанням мови програмування Pyton було обрано PyCharm. В таблиці 3.2 коротко наведено порівняльний аналіз середовищ програмування, які також можуть бути використані для реалізації.

Таблиця 3.2 – Порівняльний аналіз середовищ програмування

Критерій	PyCharm	Visual Studio Code	Eclipse	Spyder
Вбудована підтримка графіки	+	-	-	+
Інтелектуальне автодоповнення	+	+	-	+
Вбудована інтеграція з Git	+	-	+	+
Вбудована підтримка тестування	+	-	-	+
Продуктивність	+	+	-	-

PyCharm [16] – це найпотужніше середовище для розробки на Python, створене спеціально для цієї мови. Його головними перевагами є глибока інтеграція з мовою Python, розвинене автодоповнення коду, зручне налагодження, потужна підтримка сторонніх бібліотек, а також інструменти для візуалізації та аналізу. PyCharm має все необхідне для ефективної роботи з

графічними алгоритмами та математичними обчисленнями. Єдиним недоліком є досить високе споживання ресурсів, однак це виправдовується функціональністю.

VS Code [17] – це легкий і дуже популярний редактор коду, який можна налаштувати під будь-яку мову програмування, зокрема й Python. Підтримка Python реалізується через розширення, і хоча ця підтримка досить якісна, вона поступається глибині інтеграції PyCharm. Також для графіки потрібна ручна інсталяція бібліотек та додаткових плагінів.

Eclipse історично створений для Java, і підтримка Python через плагін PyDev є вторинною. Це середовище підходить для користувачів, які вже працюють в Eclipse, однак воно не надає зручних інструментів для роботи з графікою, і налаштування середовища потребує більше часу. Крім того, автодоповнення та відлагодження Python-коду тут реалізовані гірше, ніж у PyCharm.

Spyder – це наукове середовище, популярне серед дослідників, зокрема в галузі машинного навчання. Воно має зручний інтерфейс для написання та налагодження наукових скриптів, інтегровано з IPython, підтримує Matplotlib та NumPy. Spyder добре підходить для чисельних обчислень і візуалізації, однак не є найзручнішим середовищем для структурованої розробки великих проєктів або алгоритмів графічного рендерингу. Крім того, воно не має такої глибини автодоповнення та перевірки помилок, як PyCharm.

Отже, PyCharm є найкращим вибором серед усіх інтегрованих середовищ розробки для реалізації зафарбовування методом Гуро завдяки своїй повній підтримці Python, наявності потужних інструментів для роботи з графікою, тестуванням та налагодженням. Крім того, зручний інтерфейс і багатофункціональні можливості роблять його ідеальним для розробки складних алгоритмів, що потребують високої точності та оптимізації.

3.2 Розробка інтерфейсу програмного застосунку

Графічний інтерфейс користувача – це середовище взаємодії, у якому командами керують не текстові рядки, а візуальні компоненти: вікна, кнопки, іконки, меню й панелі інструментів. Завдяки наочним елементам він істотно знижує навантаження на пам’ять, оскільки відпадає потреба запам’ятовувати суворо структуровані команди.

Графічний інтерфейс користувача забезпечує прямий доступ до графіки, мультимедіа та інших елементів, а також підтримує контекстні меню для швидкого виконання дій. Найважливіша його властивість – миттєвий візуальний зворотний зв’язок, який підтверджує користувачеві успішність виконаних операцій. Графічний інтерфейс користувача дозволяє:

– інтуїтивно сприймати функції програми через наочні піктограми та кнопки;

– відображати складні графічні й мультимедійні об’єкти без додаткових команд;

– використовувати контекстні меню для швидкого доступу до потрібних дій;

– отримувати миттєвий зворотний зв’язок: користувач завжди бачить результат своєї операції на екрані.

Графічний інтерфейс користувача програмного застосунку для реалізації методу Гуро з покращенням якості та реалістичності повинен забезпечувати зручне керування параметрами об’єку візуалізації зі сторони користувача. Це робиться з метою надати користувачу можливість дослідити візуалізацію методу зафарбовування Гуро та впливу модифікацій на цей метод.

Графічний інтерфейс користувача програмного застосунку буде складатися з одного, головного, вікна. В головному вікні представлено всі функції, виконує програмний застосунок.

На рисунку 3.1 зображено модель інтерфейсу користувача.

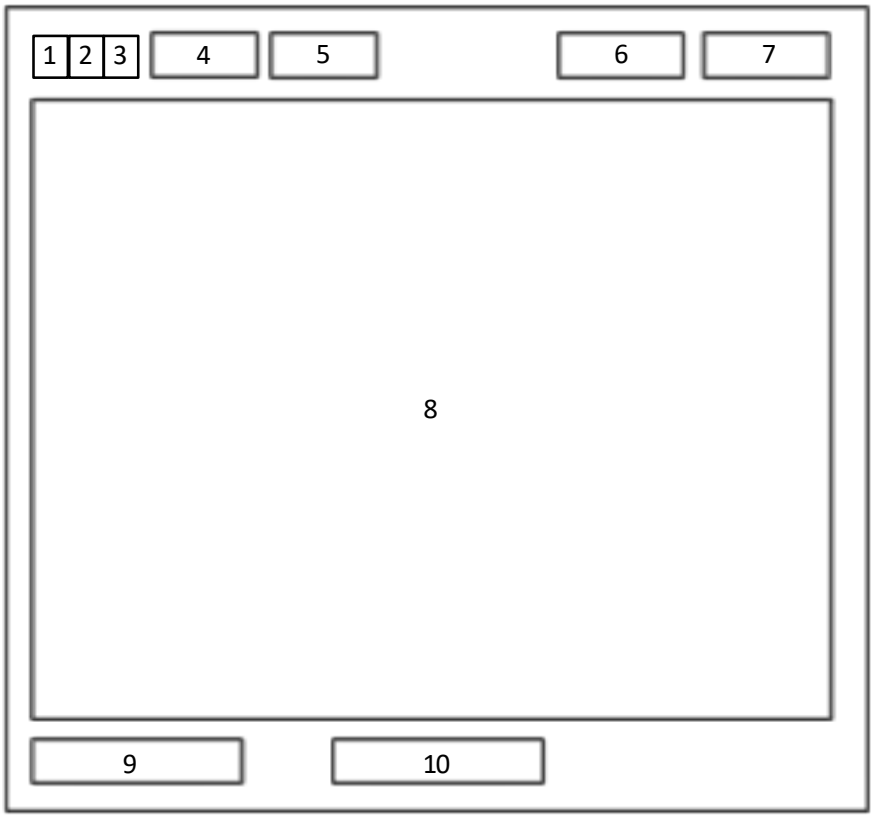

Рисунок 3.1 Модель інтерфейсу програмного застосунку

Елементи головного вікна, що позначені на рисунку 4.1 мають наступні значення:

1. Кнопка для зміни кольору вершини A трикутника для Гуро рендерингу.

2. Кнопка для зміни кольору вершини B трикутника для Гуро рендерингу.

3. Кнопка для зміни кольору вершини C трикутника для Гуро рендерингу.

4. Випадний список для вибору кольору вершини, чия кнопка активна в даний момент.

5. Кнопка для застосування змін кольорів вершин.

6. Кнопка для виконання зафарбовування модифікованим методом Гуро. При натисканні цієї кнопки функція зміни положення вершин трикутника для рендерингу блокується.

7. Кнопка для очищення робочої області екрану. Після натискання на екрані залишаються вершини, функція зміни їх положення розблоковується.

8. Робоча область програмного застосунку.

9. Кнопка для виконання зафарбовування трикутника канонічним методом Гуро.

10. Checkbox для реалізації методу зниження видимості смуг Маха при рендерингу Гуро.

Наявність в інтерфейсі можливості обрати метод зафарбовування (класичний Гуро, модифікований Гуро, Гуро з мінімалізацією смуг Маха) надає можливість помітити різницю між різними варіантами модифікацій методу Гуро рендерингу та класичним зафарбовуванням за методом Гуро.

3.3 Алгоритми роботи програмного застосунку

Програмний застосунок покращеного рендерингу Гуро включає три основні функції:

– функція рендерингу канонічним методом Гуро;

– функція рендерингу модифікованим методом Гуро;

– функція модифікації рендерингу методом Гуро для зменшення видимості смуг Маха.

Реалізація алгоритму рендерингу канонічним методом Гуро проходила через чотири взаємопов’язані етапи:

1. Для кожної вершини полігонального трикутника було визначено її просторові координати, нормальний вектор та інші атрибути, після чого за заданою моделлю освітлення обчислювалися інтенсивності у цих точках.

2. Далі відбувався процес сканування трикутника. На цьому етапі колірні значення, розраховані для вершин, рівномірно поширено по всій його площі, так що кожен піксель отримував власний відтінок як зважену суму вершинних кольорів.

3. Після заповнення фрагментів кольором здійснено фрагментну обробку.

4. Результати перетворено на кінцевий растер.

На рисунку 3.2 зображено блок-схему для функції рендерингу трикутника методом Гуро.

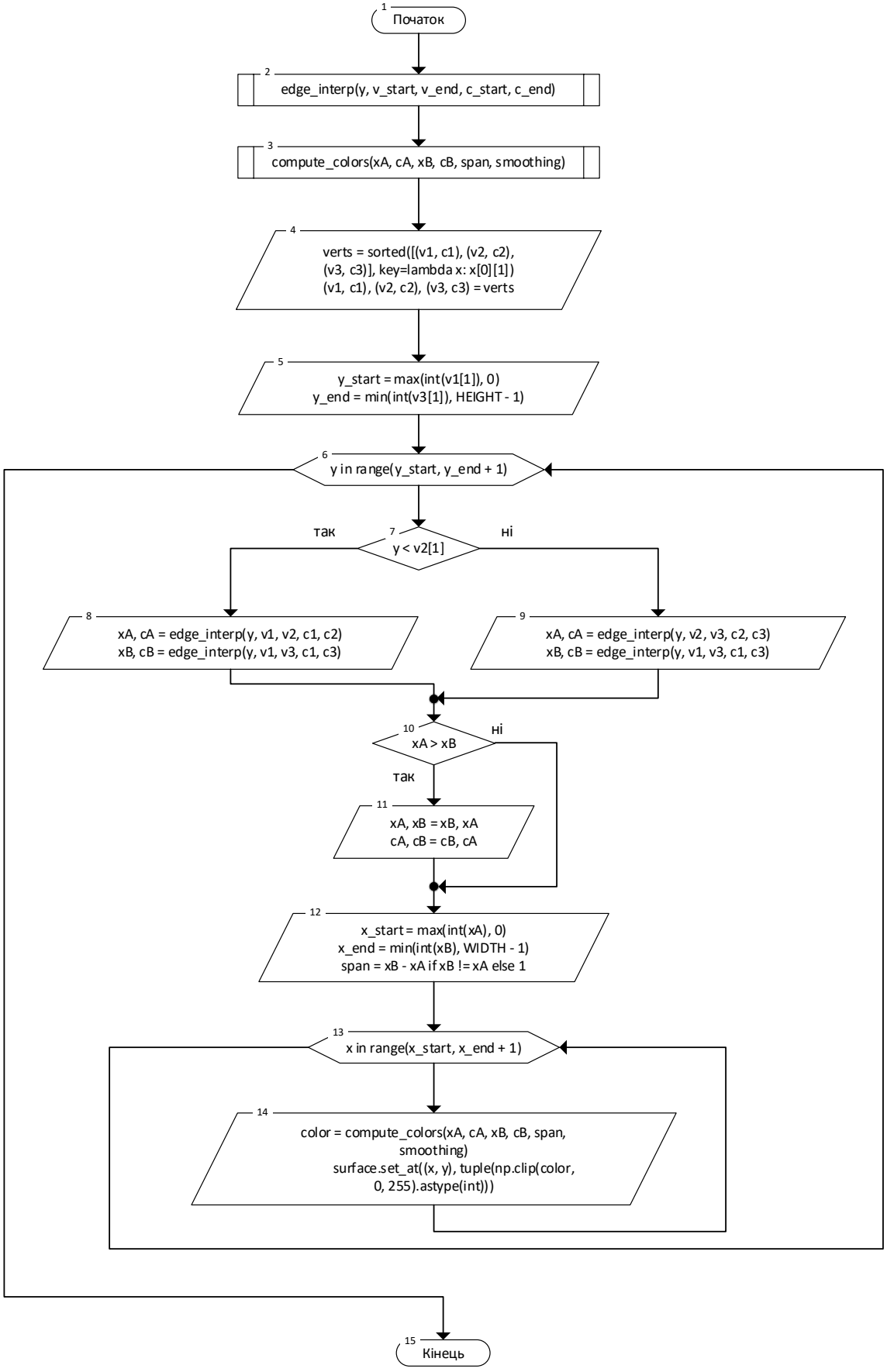

Рисунок 3.2 – Блок-схема алгоритму рендерингу трикутника методом Гуро

Функція рендерингу модифікованим методом Гуро виконана з використанням тріангуляції трикутника, інтерполяції кольорів і нормалей, а також покращення якості й продуктивності методу шляхом використання формул Фонга. Процес триангуляції розбиває великий трикутник на чотири менші трикутники, що дозволяє ефективніше обчислювати освітленість на кожній площині. При цьому результати обчислень для одного трикутника можна використовувати для інших.

Для кожної сторони трикутника інтерполюється координата X і колір на основі пікселів на кожній горизонтальній лінії. Аналогічно виконується інтерполяція нормалей. Для кожної піксельної лінії обчислюються нормалі в точках і відбувається інтерполяція між ними. Метод інтерполяції нормалей використовує крок зміни нормалі ΔN між двома точками за допомогою формули $\Delta N = \Delta N_{AB} \times 2^k$. Це дає змогу для кожного сегмента обчислювати нормаль у проміжних точках, значно зменшуючи обчислювальні витрати.

Застосовується метод інтерполяції кольорів Гуро для визначення інтенсивності кольору в кожній точці трикутника. Для кожної піксельної лінії інтерполюється колір, зокрема, за допомогою лінійної інтерполяції між двома кінцевими точками, які мають різні кольори. Модифікація коду дозволяє додавати додаткову вагу до кольорів, беручи до уваги косинусну вагу між напрямком до пікселя та напрямком нормалей до кожної вершини трикутника.

Використання формули Фонга дозволяє розбивати лінії растеризації на цифрові сегменти, що кратні степеням двійки, з подальшим застосуванням методу Гуро для визначення інтенсивності кольору в середині цих сегментів. Це дозволяє значно зменшити обчислювальні витрати.

Використання лінійної інтерполяції для нормалей та кольорів дозволяє отримати більш рівномірне освітлення на поверхні трикутника. Модифікація дозволяє точно відображати зміну кольору та освітлення без видимих артефактів.

На рисунку 3.3 зображено блок-схему для функції модифікованого рендерингу методом Гуро.

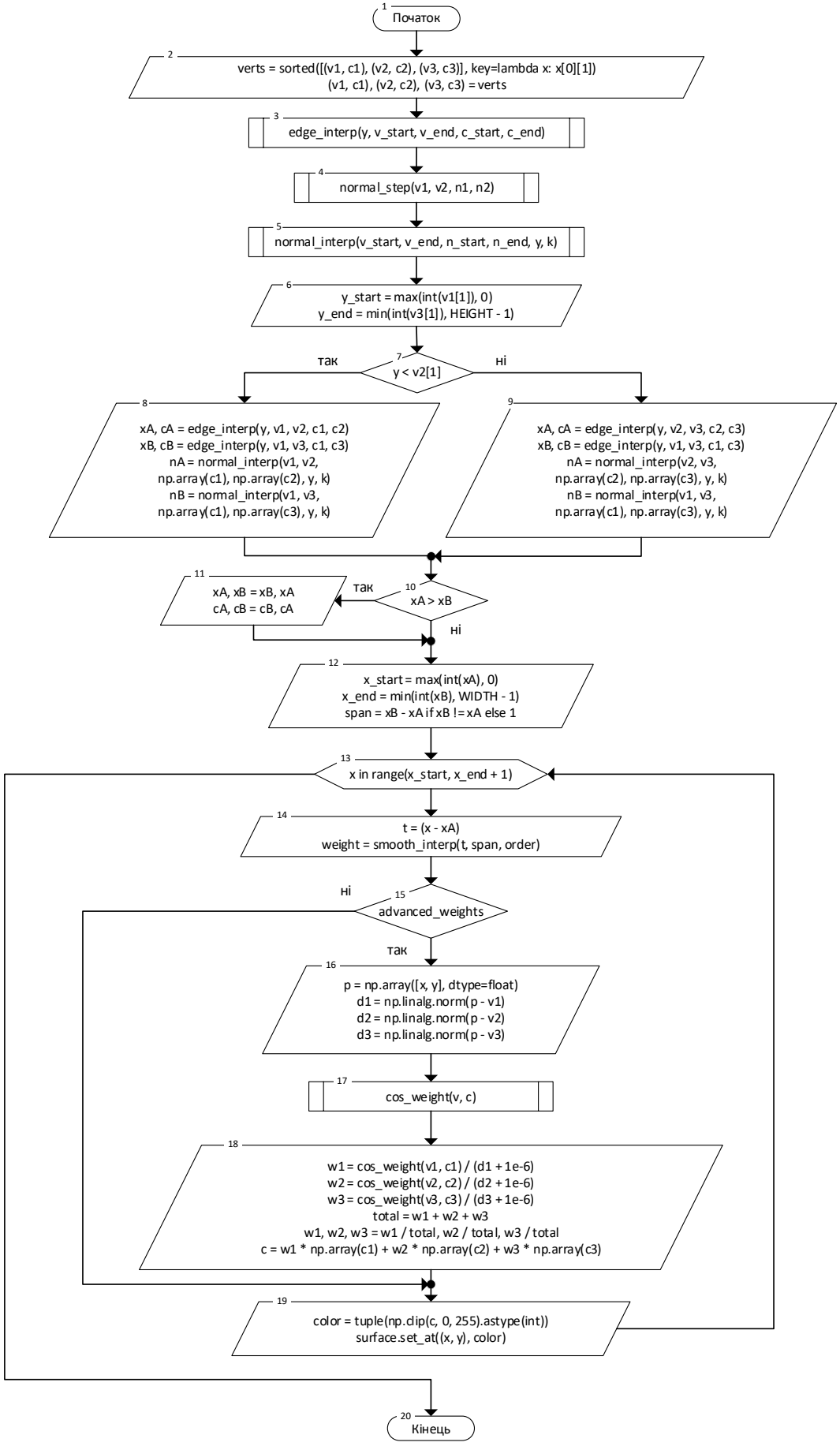

Рисунок 3.2 – блок-схема алгоритму модифікованого методу Гуро

Функція модифікації рендерингу методом Гуро для зменшення видимості смуг Маха реалізує згладжену інтерполяцію, яка використовується для зменшення видимості смуг Маха.

На рисунку 3.3 зображено блок-схему функції модифікації рендерингу методом Гуро для зменшення видимості смуг Маха

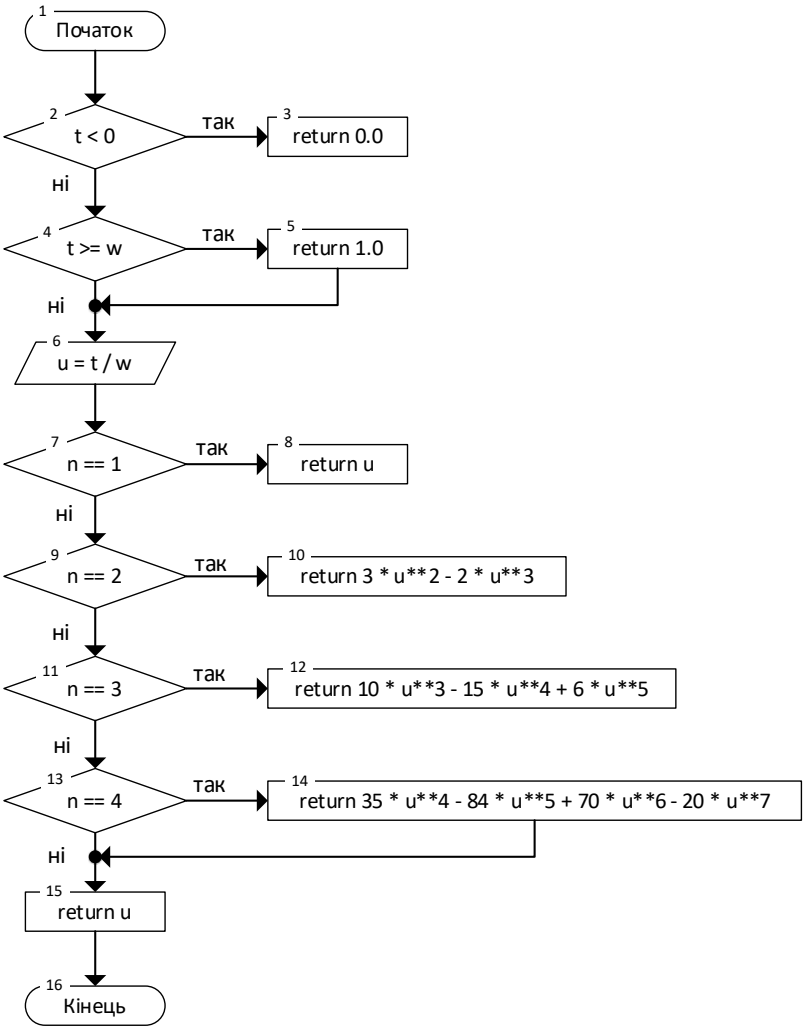

Рисунок 3.3 – алгоритм модифікації рендерингу методом Гуро для зменшення видимості смуг Маха

Згладжування досягається завдяки використанню більш складних функцій інтерполяції, які забезпечують м'якші переходи між кольорами:

– для високих значень n, наприклад 3 або 4, інтерполяція забезпечує більш плавні переходи, що ефективно згладжує різкі зміни інтенсивності

кольору, усуваючи ефект смуг Маха, який може бути помітним при лінійній інтерполяції;

– для низьких значень n, наприклад, 1 або 2, інтерполяція залишається менш складною, але все одно дає кращий результат, ніж проста лінійна інтерполяція, зменшуючи видимість цих смуг.

3.4 Висновки

У третьому розділі обґрунтовано вибір мови та середовища програмування для розробки програмного застосунку покращеного рендерингу методом Гуро.

Розроблено схему інтерфейсу для програмного застосунку.

Описано алгоритми зафарбовування зображень методом Гуро; модифікації рендерингу методом Гуро, а також алгоритм зменшення видимості смуг Маха при рендерингу методом Гуро.

4 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАСТОСУНКУ

4.1 Опис методів тестування програмного застосунку

Тестування програмного забезпечення є критичним етапом розробки, який забезпечує високу якість, стабільність і відповідність програми вимогам замовника. Воно дозволяє виявити помилки в логіці, коді чи інтерфейсі до впровадження системи, що сприяє своєчасному їх усуненню та економії ресурсів на подальших етапах. Завдяки тестуванню підтверджується, що програмне забезпечення виконує всі передбачені функції за специфікацією, підвищується довіра користувачів і замовників, а також оцінюється зручність використання. Ефективне тестування сприяє зниженню загальних витрат проєкту, оскільки дефекти, виявлені на ранніх стадіях, обходяться значно дешевше в усуненні. Таким чином, тестування є невід’ємною складовою життєвого циклу ПЗ, що гарантує його готовність до надійної експлуатації в реальних умовах.

Існує кілька основних методів тестування програмних застосунків, кожен з яких покликаний перевірити різні аспекти їхньої роботи.

Модульне тестування [18] – це базовий рівень тестування програмного забезпечення, що полягає у перевірці найменших частин програми, таких як окремі функції, методи або класи, ізольовано від інших частин системи. Його метою є впевнитися, що кожен окремий елемент виконує свою функцію правильно. Для кожного модуля створюється тестовий сценарій із заданими вхідними даними та очікуваним результатом, після чого результат виконання порівнюється з очікуванням. Найчастіше модульні тести автоматизуються і стають частиною процесу безперервної інтеграції. Для реалізації тестів використовують спеціальні фреймворки. Модульне тестування дозволяє швидко виявити помилки ще на етапі розробки, полегшує рефакторинг коду, сприяє створенню чистої архітектури та забезпечує автоматизовану перевірку роботи програми. Модульне тестування є ключовим інструментом для забезпечення надійності та стабільності програмного продукту.

Інтеграційне тестування [19] – це етап процесу тестування програмного забезпечення, під час якого перевіряється взаємодія між окремими модулями або компонентами системи. Після того як кожен модуль було протестовано окремо за допомогою модульного тестування, їх об'єднують у більші підсистеми або в єдину програму, щоб переконатися, що модулі правильно взаємодіють між собою. Основна мета інтеграційного тестування полягає у виявленні помилки в інтерфейсах між компонентами, а також збоїв, що можуть виникати під час обміну даними чи виклику методів між модулями. Існує кілька підходів до проведення інтеграційного тестування. Інкрементне коли модулі додаються поступово, неінкрементне коли все з’єднується одразу, а також методи зверху вниз, знизу вгору або комбіновані. Під час тестування можуть застосовуватись спеціальні заглушки та драйвери, що імітують поведінку ще не реалізованих або не готових модулів. Інтеграційне тестування дозволяє своєчасно виявити проблеми, які неможливо було зафіксувати при ізольованому тестуванні окремих компонентів, і тим самим забезпечити коректну роботу всієї системи в цілому.

Системне тестування [20] полягає в комплексній перевірці готового програмного продукту в умовах, максимально наближених до реального використання, і спрямоване на оцінку його поведінки як єдиного цілого. Під час цього етапу перевіряють усі функціональні та нефункціональні вимоги від коректної роботи бізнес-логіки й взаємодії з іншими системами до продуктивності, безпеки та зручності використання. Тестувальники моделюють реальні сценарії, що включають введення користувацьких даних, роботу з базами даних, мережеву взаємодію та обмін даними з зовнішніми сервісами, а також перевіряють сумісність системи з різними платформами, браузерами або операційними системами. Метою є виявити будь-які помилки, збої чи невідповідності, які могли залишитися непоміченими на попередніх рівнях тестування, і переконатися, що система забезпечує необхідну стабільність, масштабованість і безперервну роботу в реальних умовах експлуатації.

Тестування за принципом «чорного ящика» [21] передбачає перевірку програмного застосунку з точки зору користувача або зовнішнього середовища без урахування внутрішньої структури коду та алгоритмів. Цей підхід дозволяє сфокусуватися на функціональності системи та перевірити, чи відповідає вона вимогам замовника та специфікації. Під час тестування чорного ящика складаються тест-кейси на основі вимогних документів, сценаріїв використання та прикладів бізнес-логіки, після чого аналізується, чи видає програма очікувані результати в різних ситуаціях. Основна перевага методики полягає в тому, що вона не залежить від технологій чи мови програмування, які були використані для створення продукту, а також дає змогу виявити недоліки в інтерфейсі та взаємодії компонентів з боку кінцевого користувача. Однак тестування чорного ящика не гарантує виявлення всіх помилок у внутрішній логіці, тому його зазвичай поєднують із методами, що досліджують структуру коду.

Тестування «білого ящика» [22] передбачає всебічний аналіз внутрішньої структури та логіки програмного коду з метою перевірки коректності кожного його елемента. На відміну від методів, що оцінюють лише зовнішню поведінку системи, тут тестувальник має повний доступ до вихідних текстів, алгоритмів і схем архітектури, що дозволяє створювати тестові випадки на основі знання умовних переходів, циклів, гілок і обробки винятків. Основна мета такого тестування забезпечення високого рівня покриття коду: перевірка всіх логічних гілок, рядків, шляхів виконання та умов. Тестувальник використовує інструменти статичного аналізу, профайлінгу та спеціалізовані фреймворки для юніт-тестування щоб автоматизувати запуск тестів і збирати метрики покриття. Перевага цього підходу полягає в тому, що він дозволяє виявити приховані помилки на етапі розробки задовго до інтеграції модулів у загальну систему. Проте такий підхід вимагає глибокого знання системи та додаткового часу на створення й підтримку тестових сценаріїв, тому його часто комбінують із зовнішніми методами для досягнення балансу між ефективністю й комплексністю.

Тестування продуктивності [23] спрямоване на оцінку швидкодії та стійкості програмного застосунку під різними рівнями навантаження і в реальних умовах експлуатації. Воно включає кілька підвидів: навантажувальне тестування, стрес-тестування, тестування витривалості; а також тестування пікових навантажень. Під час тестування продуктивності вимірюються метрики часу відгуку, пропускної здатності, використання процесора, оперативної пам’яті та інших системних ресурсів. Результати аналізуються для виявлення «вузьких місць» у архітектурі або реалізації неефективних алгоритмів, блокувань у роботі з базою даних чи витоків пам’яті і служать основою для оптимізації коду, переконфігурації середовища або масштабування інфраструктури. Таке тестування дозволяє гарантувати, що програмний продукт буде стабільно працювати навіть під навантаженням, відповідатиме очікуванням користувачів і витримає пік активності без критичних збоїв.

Юзабіліті-тестування [24] полягає в перевірці програмного застосунку з погляду зручності та інтуїтивності його використання кінцевим користувачем: за допомогою реальних або змодельованих сценаріїв роботи дослідник спостерігає, як освоюють інтерфейс, виконують типові завдання і долають труднощі, фіксуючи час виконання операцій, кількість помилок та рівень задоволеності. У процесі проводять як формальні експерименти з чіткими метриками й протоколами, так і неформальні сеанси, коли користувач озвучує свої думки під час роботи. Аналіз отриманих результатів дозволяє виявити вузькі місця в навігації, недостатньо зрозумілий інтерфейс або неочікувану поведінку елементів, що спричиняє плутанину. Зібрані дані стають основою для оптимізації дизайну: уточнюють розташування кнопок, спрощують тексти підказок, переробляють структуру меню та інші компоненти, щоб скоротити кількість кроків до виконання завдання, підвищити ефективність і задоволеність користувачів. Таким чином, юзабіліті-тестування забезпечує створення зрозумілих, доступних і комфортних у використанні інтерфейсів, що безпосередньо впливає на успішність продукту та його сприйняття аудиторією.

Отже, для виконання тестування програмного застосунку покращеного Гуро рендеренгу, обрано використання тестування методом «білого ящика».

4.2 Тестування програмного застосунку

Тестування проводиться методом «білого ящика» з використанням операційної системи Windows 10.

Перш за все протестовано роботу програми у разі екстремально малої відстані між вершинами трикутника, який буде зафарбовуватися. На рисунку 4.1 зображено реакцію програмного застосунку у разі занадто близького розміщення вершин трикутника одна до одної.

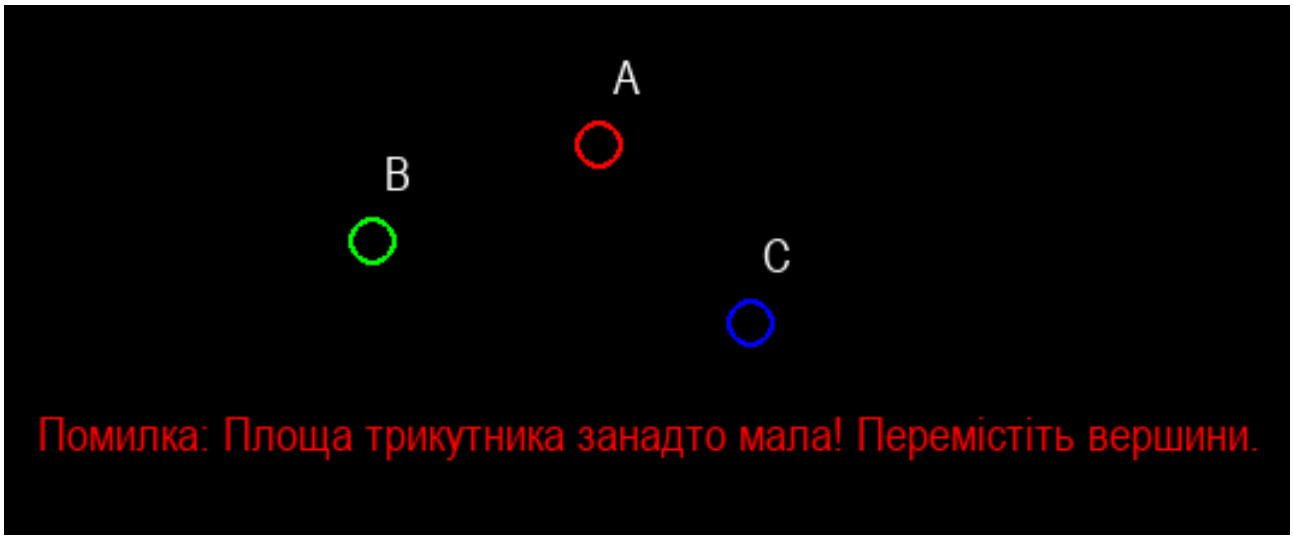

Рисунок 4.1 – Реакція програми на занадто близьке розміщення вершин трикутника

У разі некоректного розміщення вершин трикутника, коли відстань між вершинами є екстремально малою, програмний застосунок виводить повідомлення про помилку з текстом «Помилка: Площа трикутника занадто мала! Перемістіть вершини.» на центр робочої області застосунку. Це допомагає користувачу зрозуміти, що при такому розміщені вершин трикутника результат роботи програми буде некоректним, та дає зрозуміти які дії потрібно виконати для правильного відпрацювання програмного застосунку.

На рисунку 4.2 зображено вікно програмного застосунку за достатнього віддалення вершин трикутника одна від одної.

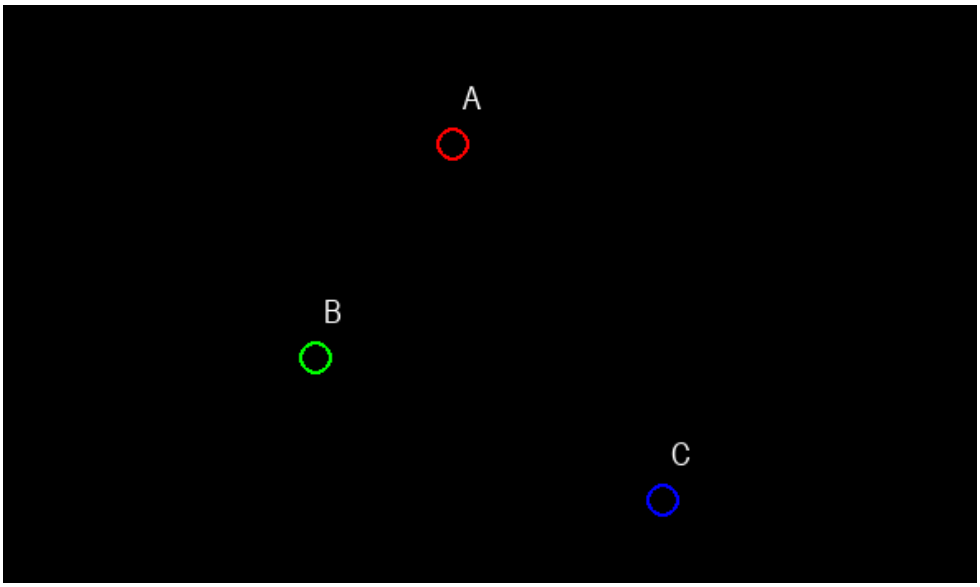

Рисунок 4.2 – Вікно програмного застосунку

За достатньої відстані між вершинами програмний застосунок не відтворює повідомлення про помилку.

Було проведено функціональне тестування створеного програмного забезпечення, основною метою якого є перевірка правильності виконання ключових функцій застосунку. У межах цього етапу перевірялися такі функціональні можливості, як зміна кольорів вершин трикутника, зафарбовування методами канонічного Гуро, модифіковного Гуро та методу Гуро модифікованим для зменшення видимості смуг Маха.

Програмний застосунок дозволяє змінювати кольори вершин трикутника з допомогою кнопок, розташованих у верхньому лівому куті. При натисканні кнопки «Колір» з’являється випадний список, з якого обирається новий колір для обраної вершини. Для застосування змін кольорів вершин натискається кнопка «Застосувати». На рисунку 4.3 зображено результат виконання зміни кольору вершини «А» з червоного на білий.

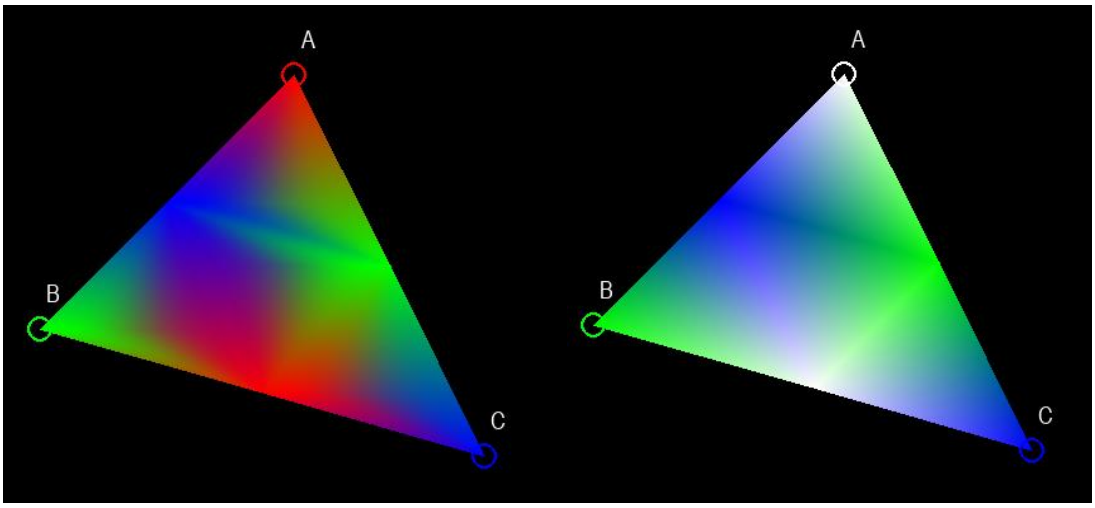

Рисунок 4.3 – Результат тестування зміни кольору вершин

В можливості програмного застосунку входить рендеринг трикутника класичним методом Гуро. Для цього слід натиснути кнопку «Гуро (класичн.)», яка викликає функцію draw_triangle_canonical(v1, v2, v3, c1, c2, c3, surface).

Функція draw_triangle_canonical(v1, v2, v3, c1, c2, c3, surface) зафарбовує трикутник використовуючи канонічний метод Гуро рендерингу. На рисунку 4.4 зображено результат класичного Гуро рендерингу трикутника.

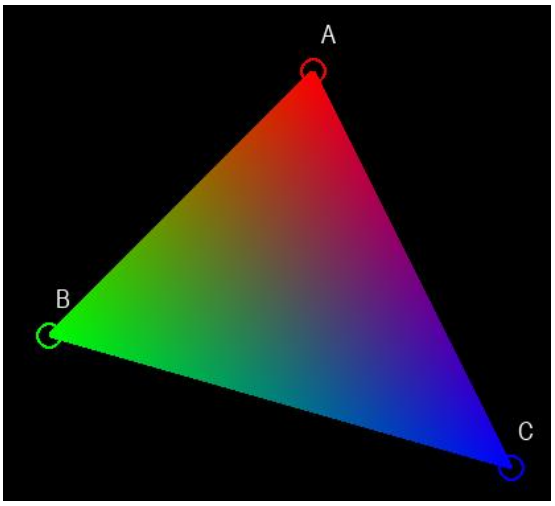

Рисунок 4.4 – Результат тестування виконання класичного рендерингу Гуро

Також в програмному застосунку реалізовано зафарбовування з використанням модифікованого методу Гуро. Для даної функції модифікація полягає в тріангуляції трикутника, а також використання методу Фонга для інтерполяції нормалей, а обчислення інтенсивності кольору сегментів згідно методу Гуро.

Для виконання зафарбовування модифікованим методом Гуро необхідно натиснути кнопку «Зафарбувати» у правій верхній частині екрану. Натискання на цю кнопку викликає функцію draw_triangle(v1, v2, v3, c1, c2, c3, surface, k=3, order=3, smoothing=True, advanced_weights=False), яка зафарбовує трекутник методом Гуро, модифікованим вже зазначеними способами.

На рисунку 4.5 зображено результат модифікованого рендерингу методом Гуро.

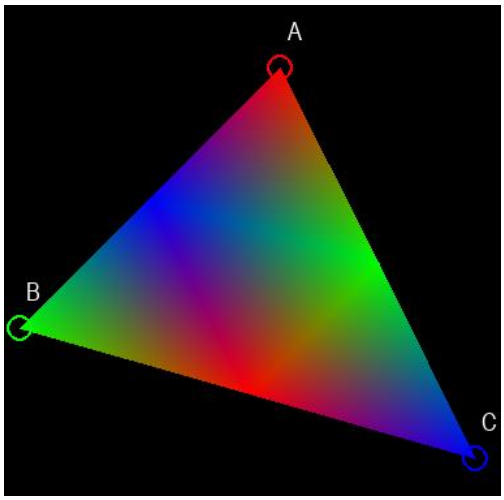

Рисунок 4.5 – Результат тестування виконання модифікованого рендерингу Гуро

При виконанні рендерингу модифікованим методом Гуро помітні смуги Маха на стиках трикутникув, отриманих в результаті триангуляції.

Для того аби зменшити видимість смуг Маха в програмному застосунку додано кнопку «Згладжування», натискання якої активує функцію smooth_interp(t, w, n), яка забезпечує згладжування смуг Маха за рахунок

використання принципу високопорядкової інтерполяції та зменшення контрастності біля ребер полігона.

На рисунку 4.6 наведено результат виконання функції зменшення видимості смуг Маха.

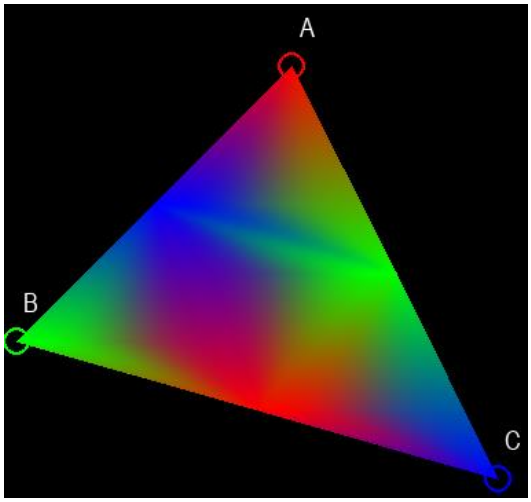

Рисунок 4.6 – Результат тестування методу зменшення смуг Маха

Також в програмному застосунку реалізовано очищення робочої області шляхос натискання кнопки «Очистити» (рисунок 4.7), яка прибирає зафарбовування трикутника.

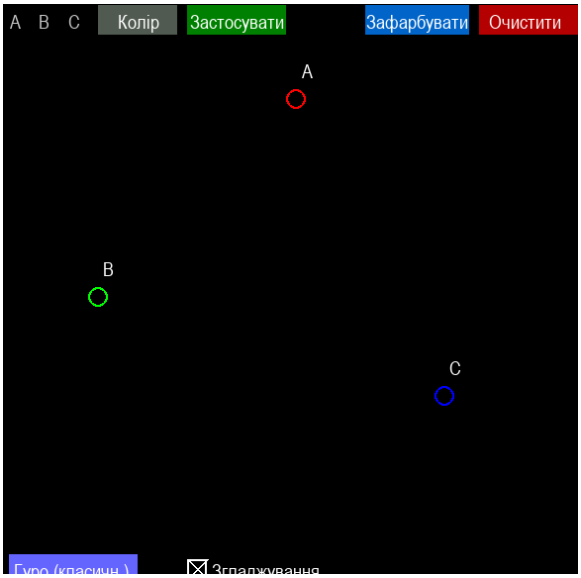

Рисунок 4.7 – Тестування функції очищення робочої зони

4.3 Розробка інструкції користувача

При розробці програмного забезпечення для генерації зображень покращеним методом Гуро обов’язковою складовою є створення повної документації для користувача. Така інструкція слугує основою для швидкого опанування програми й раціонального використання її можливостей. Вона містить докладний опис функцій, поради з налаштувань для досягнення максимального ефекту та покрокові вказівки для отримання оптимальних результатів.

Інструкція користувача потребує визначення технічних вимог до пристрою для запуску програмного застосунку.

Мінімальна конфігурація для запуску:

– операційна система Windows 7;

– процесор AMD Ryzen 3;

– оперативна пам’ять 2ГБ;

– інтегрований графічний процесор Intel HD Graphics;

– 100 МБ вільного місця на диску.

Рекомендована конфігурація для запуску:

– операційна система Windows 10;

– процесор AMD Ryzen 5;

– оперативна пам’ять 8ГБ;

– інтегрований графічний процесор AMD Radeon RX 560;

– 200 МБ вільного місця на диску.

Інформація про мінімальні та рекомендовані технічні вимоги допомагає користувачам зрозуміти чи підходить їх пристрій для запуску програмного застосунку, а також зрозуміти, чи зможуть вони комфортно використовувати програмний застосунок.

Розроблений програмний застосунок містить наступні функції:

– рендеринг трикутника класичним методом Гуро;

– рендеринг трикутника модифікованим методом Гуро;

– рендеринг трикутника методом Гуро з модифікованими покращеннями, які зменшують видимість смуг Маха;

– перетягування кожної з трьох вершин трикутника мишею для зміни форми трикутника;

– динамічне оновлення координат вершин у реальному часі;

– призначення кольорів кожній з наявних вершин трикутника;

– керування станом робочої області.

Після запуску програмного застосунку перед користувачем відкривається вікно застосунку. Щоб змінити положення вершин на робочій області, необхідно мишкою навестись на потрібну вершину та перетягнути її в бажану частину робочої області (рисунок 4.8).

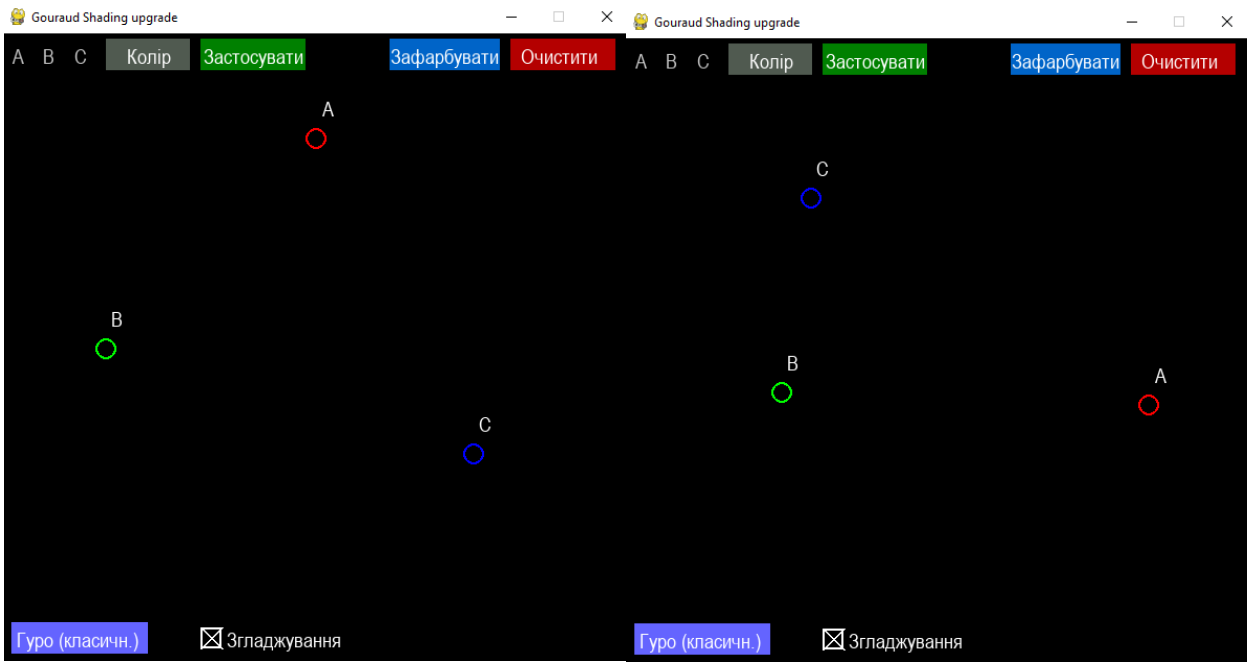

Рисунок 4.8 – Переміщення вершин трикутника в межах робочої області

Щоб змінити колір вершини трикутника, необхідно клацнути мишкою по назві вершини в лівому верхньому куту головного вікна програмного застосунку (рисунок 4.9).

Після цього назва вершини почне підсвічуватися жовтим, це означає що можна перейти до вибору нового кольору для вершини.

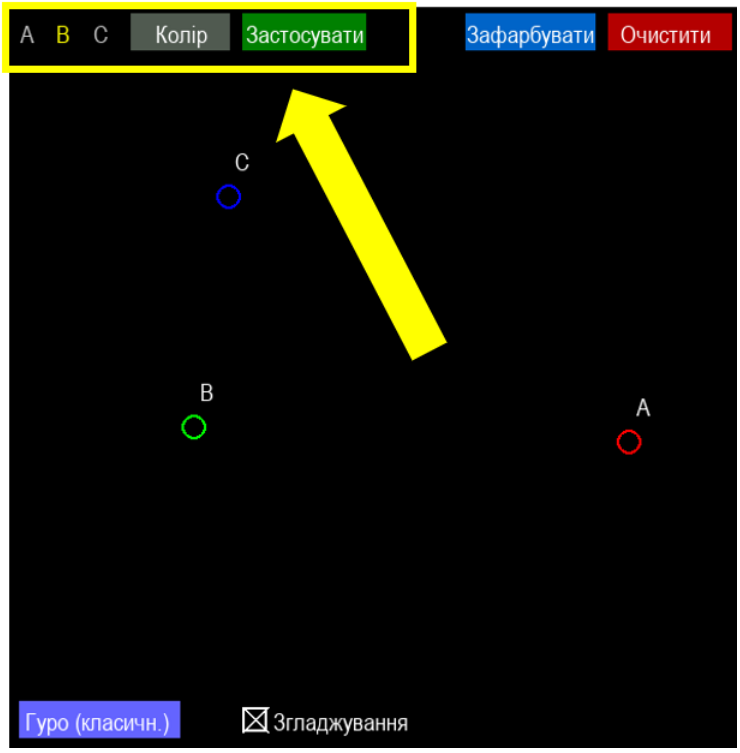

Рисунок 4.9 – Розташування кнопок для зміни кольорів вершин трикутника

Аби задати вершині новий колір необхідно натиснути на кнопку «Колір», після чого з’явиться випадний список з доступними для вибору кольорами (рисунок 4.10).

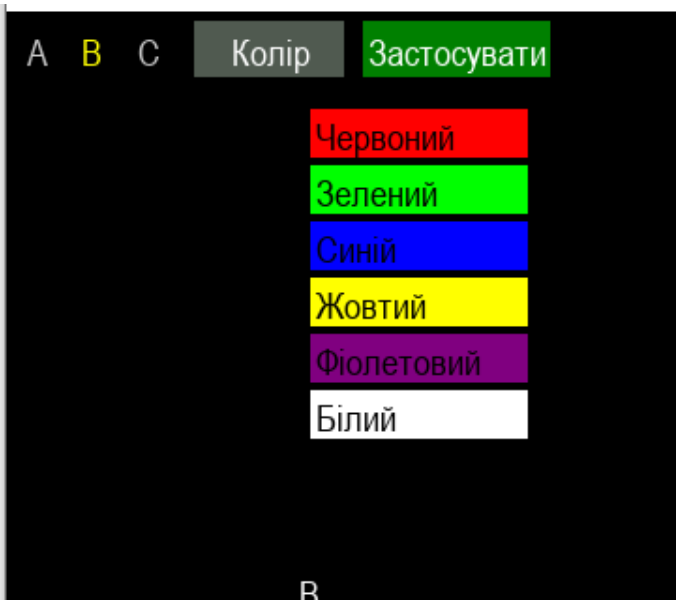

Рисунок 4.10 – Список доступних кольорів

Після вибору нових кольорів для всіх бажаних вершин, аби застосувати зміни, необхідно натиснути кнопку «Застосувати» і зміни відобразяться на робочій області програмного застосунку (рисунок 4.11).

Рисунок 4.11 – Змінені кольори вершин трикутника

Для того, щоб зафарбувати трикутник класичним методом зафарбовування Гуро потрібно натиснути кнопку «Гуро (класичн.)», що знаходиться у лівому нижньому кутку. Після зафарбовування функція переміщення вершин блокується до натискання кнопки «Очистити», що знаходиться в правому верхньому кутку програмного застосунку.

Аби зафарбувати трикутник модифікованим методом Гуро необхідно натиснути кнопку «Зафарбувати» (рисунок 4.12).

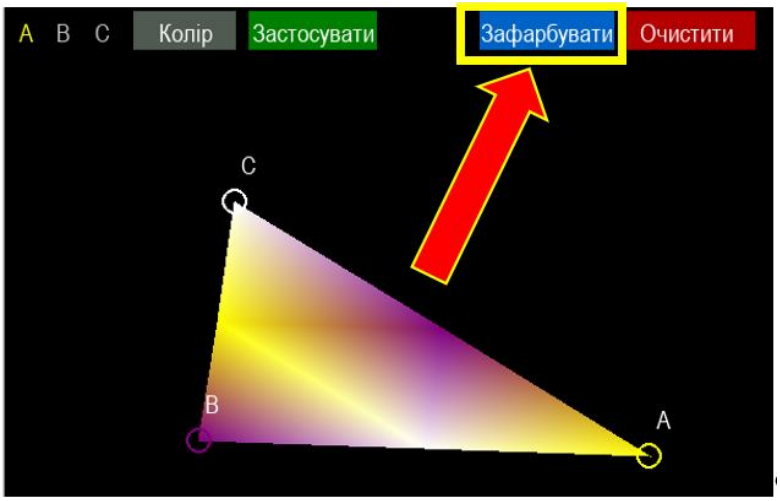

Рисунок 4.12 – Розташування кнопки «Зафарбувати»

Для того щоб зменшити видимість смуг Маха потрібно натиснути на квадратик «Згладжування» (рисунок 4.13), а щоб не використовувати методи зменшення смуг Маха достатньо повторно натиснути на квадратик.

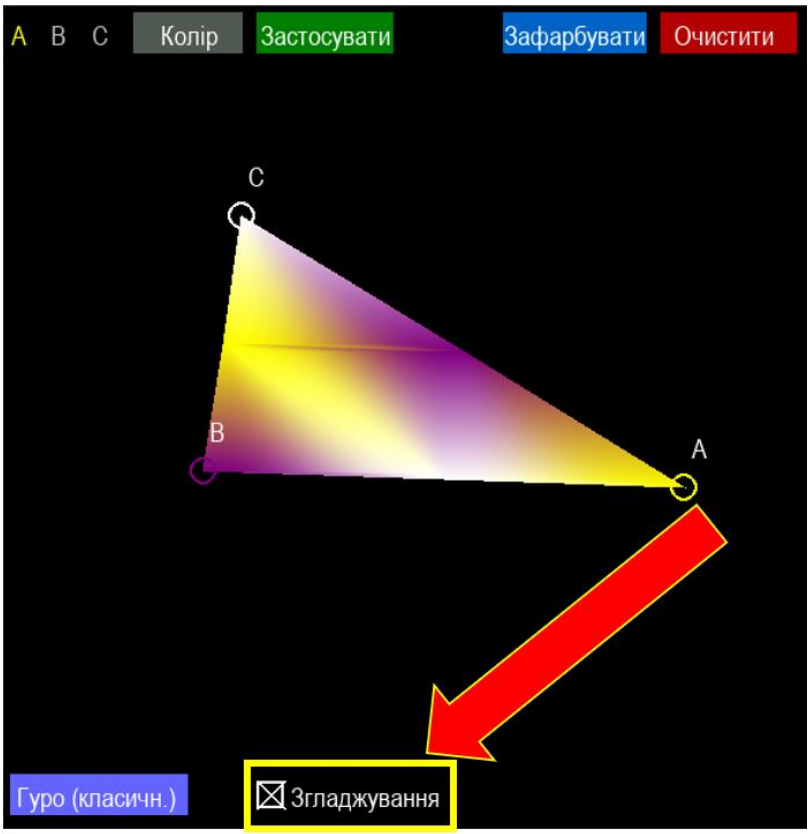

Рисунок 4.13 – Розташування кнопки «Згладжування»

Кольори вершин можна змінювати на будь-якому етапі в той час як переміщення точок доступне лише коли трикутник не зафарбований.

4.4 Висновки

У четвертому розділі було проведено тестування програмного застосунку.

Розроблено інструкцію користувача для експлуатації програмного застосунку.

ВИСНОВКИ

У результаті бакалаврської кваліфікаційної роботи були розроблені модифікації для підвищення якості та реалістичності зафарбовуванням методом Гуро. Модифікацію реалізовано з використанням тріангуляції трикутника, інтерполяції кольорів і нормалей, а також покращення якості й продуктивності методу шляхом використання формул Фонга. Також для зменшення видимості смуг Маха використано більш складні функції інтерполяції, які забезпечують м'якші переходи між кольорами, запропоновано комбінований підхід у вигляді роздільного зафарбовування прикордонної зони та центру трикутника.

При аналізі мов програмування, які підтримують можливість створення графічного інтерфейсу, для розробки програмного застосунку було обрано мову програмування Python. Середовищем для розробки програмного застосунку було обрано PyCharm.

У першому розділі подано огляд усіх етапів графічного конвеєра та методів зафарбовування. Ретельно розглянуто послідовність обробки сцени. З’ясовано, що найбільше обчислювальне навантаження припадає на стадію зафарбовування, оскільки вона вимагає точного визначення просторових координат точок поверхні та відповідних значень інтенсивності. Окрему увагу приділено особливостям алгоритму Гуро та обґрунтовано доцільність створення методів його прискорення для покращення продуктивності й реалістичності візуалізації.

Другий розділ присвячено удосконаленню алгоритму Гуро шляхом застосування додаткової тріангуляції та впровадження ітеративних підходів за методом Фонга для підвищення швидкодії рендерингу. Також запропоновано рішення для усунення смуг Маха, що виникають під час затінення за Гуро.

У третьому розділі наведено аргументи на користь вибору мови програмування та середовища розробки для реалізації покращеного методу Гуро. Створено графічний інтерфейс користувача. Також описано три ключові алгоритми, реалізовані у програмному забезпеченні.

У четвертому розділі розглянуто методи тестування, застосовані для перевірки коректності роботи програмного забезпечення для рендерингу трикутника з використанням покращеного алгоритму Гуро. Описано основні типи тестування: модульне, інтеграційне, системне, продуктивності, юзабіліті, а також тестування за методиками «чорного» та «білого ящика». Для даного застосунку обрано підхід «білого ящика». Перевірено роботу програми за різних сценаріїв: некоректне розташування вершин, зміна кольорів, рендеринг класичним і модифікованим методом Гуро з урахуванням триангуляції, усунення смуг Маха та очищення сцени. Результати тестування підтвердили стабільність і правильність виконання основних функцій. Також створено інструкцію користувача, де описано можливості програми та технічні вимоги до її запуску.

Бакалаврську кваліфікаційну роботу реалізовано згідно методичних вказівок [25].

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Романюк О. Н., Барасій О. А. Особливості реалізації методу Гуро у відеокартах. *Комп'ютерні інтелектуальні системи та мережі* : зб. матеріалів доп. учасн. XVIII всеукраїнської науково-практичної web конференції аспірантів, студентів та молодих вчених. Кривий Ріг: КНУ, 2025. с. 33-35.

2. Романюк О. Н., Барасій О. А. Особливості вбудованих відеокарт. *Науковий пошук: сучасні проблеми інформаційних технологій, математики, фізики, астрономії, управління та освіти* : зб. матеріалів доп. учасн. І всеукраїнської науково-практичної конференції. Житомир : ЖДУ імені Івана Франка, 2025. с. 23-25.

3. Романюк О. Н., Барасій О. А. Тріангуляція Серпінського для зменшення кількості обчислень при Гуро рендерингу. *Комп'ютерні інтелектуальні системи та мережі* : зб. матеріалів доп. учасн. IV Міжнародна науково-практична конференція «Інформаційні технології в освіті та науці». Мелітополь: МДПУ, 2025. с. 20-22.

4. Основи програмування шейдерів мовою GLSL. Частина 2 – Що таке шейдери, графічний конвеєр та растеризація. [Електронний ресурс] – Режим доступу до ресурсу: https://captaingpu.medium.com/основи-програмування-шейдерів-мовою-glsl-532560e22cee

5. Gouraud shading [Електронний ресурс] – Режим доступу до ресурсу: https://graphics.fandom.com/wiki/Gouraud_shading

6. Д. Трофімов, «Особливості створення шейдерних програм для реалізації освітлення», у Інженерія програмного забезпечення і передові інформаційні технології (SoftTech-2023): матеріали IV Міжнар. наук.-практ. конф. молодих вчених та студентів, присвячених 125-й річниці КПІ ім. Ігоря Сікорського, Київ, 9–11 трав. 2023, Київ: КПІ ім. Ігоря Сікорського, ІПІ ФІОТ, 2023, с. 58-59.

7. Технологія трасування променів: що це, навіщо потрібно і чим відрізняється від звичайного RTX? [Електронний ресурс] – Режим доступу до

ресурсу: https://ek.ua/ua/post/5066/298-ray-tracing-technology-what-is-it-why-is-it-needed-and-how-is-it-different-from-regular-rtx/

8. Razzali, N., Shafry, M., Rahim, M., Kumoi, R., Daman, D. Gouraud shading to improve the appearance of neuronal morphology visualization. International Journal of Biometrics and Bioinformatics. 2022. Vol. 4, №. 2. P. 52-62

9. Романюк О. Н., Романюк О. В., Чехместрук Р. Ю. Комп'ютерна графіка: електронний навчальний посібник / О. Н. Романюк, О. В. Романюк, Р. Ю. Чехместрук. – Вінниця, 2023. – [Електронний ресурс]. – Режим доступу: https://pdf.lib.vntu.edu.ua/books/2024/Romanjuk_2023_147.pdf

10. Тріангуляція по точках [Електронний ресурс] – Режим доступу до ресурсу: https://lola.artbooks.cx.ua/articles/trianguljacija-ploshhini-po-tochkah.html

11. І. Ковальчук, «Методи триангуляції фракталів на прикладі трикутника Серпінського», у *Матеріали Всеукр. наук.-техн. конференції студентів і молодих вчених*, Харків, 15–17 квіт. 2024, Харків: ХНУ ім. В. Н. Каразіна, 2024, с. 112-115.

12. Phong Shading Computer Graphics [Електронний ресурс] – Режим доступу до ресурсу: https://www.geeksforgeeks.org/phong-shading-computer-graphics/

13. Bishop, Gary, and David M. Weimer. "Fast phong shading." ACM Siggraph Computer Graphics 20.4 (1986): 103-106.

14. Матсенко, В.Г. Обчислювальна геометрія та комп'ютерна графіка: навчальний посібник, Чернівці: Чернівецький національний університет імені Юрія Федьковича, 2023, 440 с.

15. O. Romanyuk, O. Romanyuk, P. Mykhaylov, R. Chekhmestruk, T. Korobeinikova, and S. Romanyuk, "Features of the computational process organization of initial parameters determination for shading," in Proc. 12th Int. Conf. Advanced Computer Information Technologies (ACIT), Sep. 2022, pp. 22–26.

16. Python Introduction [Електронний ресурс] – Режим доступу до ресурсу: https://www.w3schools.com/python/python_intro.asp

17. Your code editor. Redefined with AI. [Електронний ресурс] – Режим доступу до ресурсу: https://code.visualstudio.com

18. PyCharm The only Python IDE you need [Електронний ресурс] – Режим доступу до ресурсу: https://www.jetbrains.com/pycharm/

19. Модульне тестування [Електронний ресурс] – Режим доступу до ресурсу: https://qalight.ua/baza-znaniy/modulne-testuvannya/

20. Що таке інтеграційне тестування? Глибоке занурення в типи, процеси та впровадження [Електронний ресурс] – Режим доступу до ресурсу: https://www.zaptest.com/uk/що-таке-інтеграційне-тестування-глиб

21. Що таке системне тестування? Типи з прикладом [Електронний ресурс] – Режим доступу до ресурсу: https://www.guru99.com/uk/system-testing.html

22. Яка різниця між Black Box & White Box Testing? [Електронний ресурс] – Режим доступу до ресурсу: https://surli.cc/ewccjp

23. Рекомендації щодо тестування продуктивності [Електронний ресурс] – Режим доступу до ресурсу: https://learn.microsoft.com/uk-ua/power-platform/well-architected/performance-efficiency/performance-test

24. Що таке юзабіліті тестування: розгорнутий огляд для розробників програмного забезпечення [Електронний ресурс] – Режим доступу до ресурсу: https://mate.academy/blog/qa/usability-testing/

25. 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

72

ДОДАТОК А – ТЕХНІЧНЕ ЗАВДАННЯ

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

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

Технічне завдання
на бакалаврську кваліфікаційну роботу «Методи та програмні засоби високопродуктивного рендеригу Гуро»
за спеціальністю
121 – Інженерія програмного забезпечення

Керівник бакалаврської кваліфікаційної роботи:
д.т.н., проф. Романюк О.Н.
«21» 03 2025 року.
Виконала:
студентка гр. 2ПІ-21б Барасій О. А.
«21» 03 2025 року.

м. Вінниця – 2025 рік

1. Найменування та галузь застосування

Бакалаврська кваліфікаційна робота: «Методи та засоби високопродуктивного рендеригу Гуро».

Галузь застосування – системи комп’ютерної графіки.

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 та реалістичності рендерингу Гуро за рахунок розробки нових методів і засобів.

4. Вихідні дані для проведення НДР

1. Романюк О. Н., Романюк О. В., Чехместрук Р. Ю. Комп’ютерна графіка: електронний навчальний посібник / О. Н. Романюк, О. В. Романюк, Р. Ю. Чехместрук. – Вінниця, 2023. – [Електронний ресурс]. – Режим доступу: https://pdf.lib.vntu.edu.ua/books/2024/Romanjuk_2023_147.pdf

2. Methods of Reducing the Visibility of Mach Bands during Gouraud Shading [Електронний ресурс] – Режим доступу до ресурсу: https://www.cs.ru.ac.za/research/groups/vrsig/pastprojects/041machbands/paper01.pdf

3. Матсенко, В.Г. Обчислювальна геометрія та комп’ютерна графіка: навчальний посібник, Чернівці: Чернівецький національний університет імені Юрія Федьковича, 2023, 440 с.

5. Технічні вимоги

Метод зафарбовування – метод Гуро; розмір координатного простору 600x600; кольоровий режим – truecolor; тип полігональної моделі – трикутна; середовище розробки PyCharm 2024.2.1; мова розробки Python; операційна система – Windows 10.

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 методів зафарбовування поверхонь	25.03.25 – 30.03.25
2	Розробка методів для високопродуктивного та якіснішого формування зображень методом Гуро рендерингу	31.03.25 – 20.04.25
3	Розробка програмних засобів для покращеного зафарбовування Гуро	21.04.25 – 30.04.25
4	Тестування програмного застосунку	01.05.25 – 10.05.25
5	Оформлення матеріалів до захисту БКР	11.05.25 – 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.

75

ДОДАТОК Б – ПРОТОКОЛ ПЕРЕВІРКИ НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

ПРОТОКОЛ
ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ
НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: Методи та засоби високопродуктивного рендеригу Гуро.

Тип роботи: БКР

Підрозділ : кафедра програмного забезпечення, ФІТКІ

Науковий керівник: д.т.н., проф. каф. ПЗ Романюк О.Н.

Оригінальність	98 %
Схожість	2 %

Особа, відповідальна за перевірку _____ Черноволик Г. О.

Ознайомлені з повним звітом подібності, який був згенерований системою StrikePlagiarism.

Автор роботи _____ Барасій О. А.

Керівник роботи _____ Романюк О.Н.

ДОДАТОК В – ЛІСТИНГ ПРОГРАМИ

```
import pygame
import numpy as np

pygame.init()

WIDTH, HEIGHT = 600, 600
screen = pygame.display.set_mode((WIDTH, HEIGHT))
pygame.display.set_caption("Gouraud Shading upgrade")
font = pygame.font.SysFont("Arial", 20)

def triangle_area(v1, v2, v3):
    return abs(v1[0]*(v2[1] - v3[1]) + v2[0]*(v3[1] - v1[1]) + v3[0]*(v1[1] - v2[1])) /
2

def calculate_area(v1, v2, v3):
    return 0.5 * abs(v1[0] * (v2[1] - v3[1]) + v2[0] * (v3[1] - v1[1]) + v3[0] * (v1[1] -
v2[1]))

def interpolate(v0, v1, t):
    return v0 + t * (v1 - v0)

def enforce_parity(coord):
    x, y = int(coord[0]), int(coord[1])
    if (x % 2) != (y % 2):
        x += 1
    if (x % 2) != (y % 2):
        y += 1
    return np.array([x, y], dtype=float)
```

```
def midpoint_with_parity(v1, v2):
    mid = (v1 + v2) / 2
    return enforce_parity(mid)

def smooth_interp(t, w, n):
    if t < 0:
        return 0.0
    if t >= w:
        return 1.0
    u = t / w
    if n == 1:
        return u
    elif n == 2:
        return 3 * u**2 - 2 * u**3
    elif n == 3:
        return 10 * u**3 - 15 * u**4 + 6 * u**5
    elif n == 4:
        return 35 * u**4 - 84 * u**5 + 70 * u**6 - 20 * u**7
    return u

def draw_triangle(v1, v2, v3, c1, c2, c3, surface, k=3, order=3, smoothing=True,
advanced_weights=False):
    verts = sorted([(v1, c1), (v2, c2), (v3, c3)], key=lambda x: x[0][1])
    (v1, c1), (v2, c2), (v3, c3) = verts

    def edge_interp(y, v_start, v_end, c_start, c_end):
        if v_end[1] == v_start[1]:
            return v_start[0], np.array(c_start)
        t = (y - v_start[1]) / (v_end[1] - v_start[1])
```

```
    x = interpolate(v_start[0], v_end[0], t)
    c = interpolate(np.array(c_start), np.array(c_end), t)
    return x, c

def normal_step(v1, v2, n1, n2):
    delta_n = (n2 - n1) / (v2[1] - v1[1])
    return delta_n

def normal_interp(v_start, v_end, n_start, n_end, y, k):
    delta_n = normal_step(v_start, v_end, n_start, n_end)
    offset = int(y - v_start[1])
    shifted = offset << k
    return n_start + delta_n * shifted

y_start = max(int(v1[1]), 0)
y_end = min(int(v3[1]), HEIGHT - 1)

for y in range(y_start, y_end + 1):
    if y < v2[1]:
        xA, cA = edge_interp(y, v1, v2, c1, c2)
        xB, cB = edge_interp(y, v1, v3, c1, c3)
        nA = normal_interp(v1, v2, np.array(c1), np.array(c2), y, k)
        nB = normal_interp(v1, v3, np.array(c1), np.array(c3), y, k)
    else:
        xA, cA = edge_interp(y, v2, v3, c2, c3)
        xB, cB = edge_interp(y, v1, v3, c1, c3)
        nA = normal_interp(v2, v3, np.array(c2), np.array(c3), y, k)
        nB = normal_interp(v1, v3, np.array(c1), np.array(c3), y, k)

    if xA > xB:
```

```
    xA, xB = xB, xA
    cA, cB = cB, cA
    nA, nB = nB, nA

x_start = max(int(xA), 0)
x_end = min(int(xB), WIDTH - 1)
span = xB - xA if xB != xA else 1

for x in range(x_start, x_end + 1):
    t = (x - xA)
    weight = smooth_interp(t, span, order) if smoothing else t / span
    c = interpolate(cA, cB, weight)

    if advanced_weights:
        p = np.array([x, y], dtype=float)
        d1 = np.linalg.norm(p - v1)
        d2 = np.linalg.norm(p - v2)
        d3 = np.linalg.norm(p - v3)

        def cos_weight(v, c):
            dir_vec = p - v
            if np.linalg.norm(dir_vec) == 0 or np.linalg.norm(c) == 0:
                return 1.0
            return np.dot(dir_vec / np.linalg.norm(dir_vec), c / np.linalg.norm(c))

        w1 = cos_weight(v1, c1) / (d1 + 1e-6)
        w2 = cos_weight(v2, c2) / (d2 + 1e-6)
        w3 = cos_weight(v3, c3) / (d3 + 1e-6)
        total = w1 + w2 + w3
        w1, w2, w3 = w1 / total, w2 / total, w3 / total
```

```
                c = w1 * np.array(c1) + w2 * np.array(c2) + w3 * np.array(c3)

            color = tuple(np.clip(c, 0, 255).astype(int))
            surface.set_at((x, y), color)

def draw_triangle_canonical(v1, v2, v3, c1, c2, c3, surface, smoothing=True):
    def edge_interp(y, v_start, v_end, c_start, c_end):
        if v_end[1] == v_start[1]:
            return v_start[0], np.array(c_start)
        t = (y - v_start[1]) / (v_end[1] - v_start[1])
        x = interpolate(v_start[0], v_end[0], t)
        c = interpolate(np.array(c_start), np.array(c_end), t)
        return x, c

    def compute_colors(xA, cA, xB, cB, span, smoothing):
        t = (x - xA) / span
        weight = smooth_interp(t, span, 3) if smoothing else t
        return interpolate(cA, cB, weight)

    verts = sorted([(v1, c1), (v2, c2), (v3, c3)], key=lambda x: x[0][1])
    (v1, c1), (v2, c2), (v3, c3) = verts

    y_start = max(int(v1[1]), 0)
    y_end = min(int(v3[1]), HEIGHT - 1)

    for y in range(y_start, y_end + 1):
        if y < v2[1]:
            xA, cA = edge_interp(y, v1, v2, c1, c2)
            xB, cB = edge_interp(y, v1, v3, c1, c3)
        else:
```

```
            xA, cA = edge_interp(y, v2, v3, c2, c3)
            xB, cB = edge_interp(y, v1, v3, c1, c3)

        if xA > xB:
            xA, xB = xB, xA
            cA, cB = cB, cA

        x_start = max(int(xA), 0)
        x_end = min(int(xB), WIDTH - 1)
        span = xB - xA if xB != xA else 1

        for x in range(x_start, x_end + 1):
            color = compute_colors(xA, cA, xB, cB, span, smoothing)
            surface.set_at((x, y), tuple(np.clip(color, 0, 255).astype(int)))

# Вершини
vA = np.array((300, 100), dtype=float)
vB = np.array((100, 300), dtype=float)
vC = np.array((450, 400), dtype=float)

# Кольори для вершин
colors = {
    "Червоний":     (255, 0, 0),
    "Зелений":      (0, 255, 0),
    "Синій":        (0, 0, 255),
    "Жовтий":       (255, 255, 0),
    "Фіолетовий":   (128, 0, 128),
    "Білий":        (255, 255, 255)
}
```

```
vertex_colors = {"A": (255, 0, 0), "B": (0, 255, 0), "C": (0, 0, 255)}
pending_vertex_colors = vertex_colors.copy()

# Кольори для фону
bg_colors = {
    "Чорний":    (0, 0, 0),
    "Білий":     (255, 255, 255),
    "Сірий":     (128, 128, 128),
    "Синій":     (0, 0, 255),
    "Зелений":   (0, 255, 0)
}
background_color = bg_colors["Чорний"]

# Стан взаємодії
selected_vertex = None
dragging_vertex = None
dropdown_open = False
bg_dropdown_open = False
vertex_radius = 10
rendering_enabled = False
canonical_render = False
drag_enabled = True
smoothing_enabled = True

# Кнопки та області взаємодії
dropdown_button     = pygame.Rect(100, 5, 80, 30)
apply_button        = pygame.Rect(190, 5, 100, 30)
render_button       = pygame.Rect(370, 5, 105, 30)
clear_button        = pygame.Rect(485, 5, 100, 30)
smoothing_checkbox  = pygame.Rect(190, 565, 20, 20)
```

```
canonical_button    = pygame.Rect(10, 560, 130, 30)

# Випадаючі елементи для вибору кольору вершин
dropdown_items = [pygame.Rect(160, 50 + i * 30, 120, 30) for i in
range(len(colors))]

# Випадаючі елементи для вибору кольору фону
bg_dropdown_button = pygame.Rect(10, 40, 100, 30)
bg_dropdown_items  = [pygame.Rect(10, 80 + i * 30, 120, 30) for i in
range(len(bg_colors))]

vertex_buttons = {}

def draw_ui():

   # Малюємо назви вершин A, B, C
    x_offset = 10
    for name in ["A", "B", "C"]:
        col = (255, 255, 0) if selected_vertex == name else (200, 200, 200)
        lbl = font.render(name, True, col)
        rect = lbl.get_rect(topleft=(x_offset, 10))
        screen.blit(lbl, rect)
        vertex_buttons[name] = rect
        x_offset += 30

    # Кнопка для випадаючого списку кольорів вершин
    pygame.draw.rect(screen, (80, 90, 80), dropdown_button)
    screen.blit(font.render("Колір", True, (255, 255, 255)), (dropdown_button.x + 20,
dropdown_button.y + 5))
```

```
    # Кнопки Застосувати, Зафарбувати, Очистити
    pygame.draw.rect(screen, (0, 128, 0), apply_button)
    screen.blit(font.render("Застосувати", True, (255, 255, 255)), (apply_button.x + 3, apply_button.y + 5))

    pygame.draw.rect(screen, (0, 100, 200), render_button)
    screen.blit(font.render("Зафарбувати", True, (255, 255, 255)), (render_button.x + 1, render_button.y + 5))

    pygame.draw.rect(screen, (180, 0, 0), clear_button)
    screen.blit(font.render("Очистити", True, (255, 255, 255)), (clear_button.x + 10, clear_button.y + 5))

    # Визначаємо колір для рамки та хрестика чекбоксу залежно від фону
    if background_color in [bg_colors["Зелений"], bg_colors["Білий"]]:
        box_col = (0, 0, 0)
        text_col = (0, 0, 0)
    else:
        box_col = (255, 255, 255)
        text_col = (255, 255, 255)

    # Малюємо рамку чекбоксу
    pygame.draw.rect(screen, box_col, smoothing_checkbox, 2)

    # Якщо увімкнено, малюємо хрестик тим же кольором
    if smoothing_enabled:
        pygame.draw.line(screen, box_col, smoothing_checkbox.topleft, smoothing_checkbox.bottomright, 2)
        pygame.draw.line(screen, box_col, smoothing_checkbox.topright, smoothing_checkbox.bottomleft, 2)
```

```
    screen.blit(font.render("Згладжування", True, text_col), (smoothing_checkbox.x
+ 25, smoothing_checkbox.y))

    # Кнопка для класичного Гуро
    pygame.draw.rect(screen, (100, 100, 255), canonical_button)
    screen.blit(font.render("Гуро (класичн.)", True, (255, 255, 255)),
(canonical_button.x + 5, canonical_button.y + 5))

    # Кнопка для вибору кольору фону
    pygame.draw.rect(screen, (80, 90, 80), bg_dropdown_button)
    screen.blit(font.render("Колір фону", True, (255, 255, 255)),
(bg_dropdown_button.x + 5, bg_dropdown_button.y + 5))

    # Якщо відкрито список кольорів вершин
    if dropdown_open:
        for i, (name, col) in enumerate(colors.items()):
            r = dropdown_items[i]
            pygame.draw.rect(screen, col, r)
            pygame.draw.rect(screen, (0, 0, 0), r, 2)
            screen.blit(font.render(name, True, (0, 0, 0)), (r.x + 5, r.y + 5))

    # Якщо відкрито список кольорів фону
    if bg_dropdown_open:
        for i, (name, col) in enumerate(bg_colors.items()):
            r = bg_dropdown_items[i]
            pygame.draw.rect(screen, col, r)
            pygame.draw.rect(screen, (0, 0, 0), r, 2)
            screen.blit(font.render(name, True, (0, 0, 0)), (r.x + 5, r.y + 5))

def draw_label(pos, name, col):
```

```
    lbl = font.render(name, True, (255, 255, 255))
    screen.blit(lbl, (pos[0] + 5, pos[1] - 40))
    pygame.draw.circle(screen, col, pos.astype(int), vertex_radius, 2)

running = True
while running:

    # Заливаємо фон обраним кольором
    screen.fill(background_color)
    area = triangle_area(vA, vB, vC)

    if area < 5000:
        error_message = font.render(
            "Помилка: Площа трикутника занадто мала! Перемістіть вершини.",
            True, (255, 0, 0)
        )

        screen.blit(
            error_message,
            (WIDTH // 2 - error_message.get_width() // 2, HEIGHT // 2)
        )

    else:
        if rendering_enabled:
            vA_adj = enforce_parity(vA)
            vB_adj = enforce_parity(vB)
            vC_adj = enforce_parity(vC)

            if canonical_render:
                draw_triangle_canonical(
```

```
            vA_adj, vB_adj, vC_adj,
            vertex_colors["A"],
            vertex_colors["B"],
            vertex_colors["C"],
            screen
        )

    else:
        mAB = midpoint_with_parity(vA_adj, vB_adj)
        mBC = midpoint_with_parity(vB_adj, vC_adj)
        mCA = midpoint_with_parity(vC_adj, vA_adj)

        draw_triangle(
            vA_adj, mAB, mCA,
            vertex_colors["A"], vertex_colors["C"], vertex_colors["B"],
            screen, smoothing=smoothing_enabled
        )

        draw_triangle(
            vB_adj, mBC, mAB,
            vertex_colors["B"], vertex_colors["A"], vertex_colors["C"],
            screen, smoothing=smoothing_enabled
        )

        draw_triangle(
            vC_adj, mCA, mBC,
            vertex_colors["C"], vertex_colors["B"], vertex_colors["A"],
            screen, smoothing=smoothing_enabled
        )
```

```
            draw_triangle(
                mAB, mBC, mCA,
                vertex_colors["C"], vertex_colors["A"], vertex_colors["B"],
                screen, smoothing=smoothing_enabled
            )

    # Малюємо маркери та підписи вершин
    draw_label(vA, "A", vertex_colors["A"])
    draw_label(vB, "B", vertex_colors["B"])
    draw_label(vC, "C", vertex_colors["C"])

    # Малюємо UI
    draw_ui()
    pygame.display.flip()

    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False

        elif event.type == pygame.MOUSEBUTTONDOWN:
            mx, my = event.pos

            # Вибір вершини A/B/C для зміни кольору або перетягування
            for name, rect in vertex_buttons.items():
                if rect.collidepoint(mx, my):
                    selected_vertex = name

            # Переключення випадаючого списку кольорів вершин
            if dropdown_button.collidepoint(mx, my):
```

```
            dropdown_open = not dropdown_open
            bg_dropdown_open = False  # закриваємо список кольорів фону

        # Переключення випадаючого списку кольорів фону
        elif bg_dropdown_button.collidepoint(mx, my):
            bg_dropdown_open = not bg_dropdown_open
            dropdown_open = False  # закриваємо список кольорів вершин

        # Застосувати нові кольори вершин
        elif apply_button.collidepoint(mx, my):
            vertex_colors = pending_vertex_colors.copy()

        # Зафарбувати (звичайний спосіб)
        elif render_button.collidepoint(mx, my):
            rendering_enabled = True
            drag_enabled = False
            canonical_render = False

        # Очистити екран (прибрати затінення)
        elif clear_button.collidepoint(mx, my):
            rendering_enabled = False
            drag_enabled = True

        # Перемикач згладжування
        elif smoothing_checkbox.collidepoint(mx, my):
            smoothing_enabled = not smoothing_enabled

        # Класичний Гуро
        elif canonical_button.collidepoint(mx, my):
            rendering_enabled = True
```

```
            drag_enabled = False
            canonical_render = True

        # Якщо випадаючий список кольорів вершин відкритий — вибираємо колір для вершини
        elif dropdown_open:
            for i, rect in enumerate(dropdown_items):
                if rect.collidepoint(mx, my) and selected_vertex:
                    key = list(colors.keys())[i]
                    pending_vertex_colors[selected_vertex] = colors[key]
                    dropdown_open = False
                    break

        # Якщо випадаючий список кольорів фону відкритий — вибираємо колір фону
        elif bg_dropdown_open:
            for i, rect in enumerate(bg_dropdown_items):
                if rect.collidepoint(mx, my):
                    key = list(bg_colors.keys())[i]
                    background_color = bg_colors[key]
                    bg_dropdown_open = False
                    break

        # Перетягування вершини (початок)
        elif drag_enabled:
            for name, v in [("A", vA), ("B", vB), ("C", vC)]:
                if np.linalg.norm(np.array((mx, my)) - v) < vertex_radius:
                    dragging_vertex = name
                    break
```

```
        elif event.type == pygame.MOUSEMOTION and dragging_vertex and
drag_enabled:

            # Оновлюємо позицію вершини під час перетягування
            if dragging_vertex == "A":
                vA[:] = event.pos
            elif dragging_vertex == "B":
                vB[:] = event.pos
            elif dragging_vertex == "C":
                vC[:] = event.pos

        elif event.type == pygame.MOUSEBUTTONUP:
            # Завершення перетягування
            dragging_vertex = None

pygame.quit()
```

ДОДАТОК Г – ІЛЛЮСТРАТИВНА ЧАСТИНА

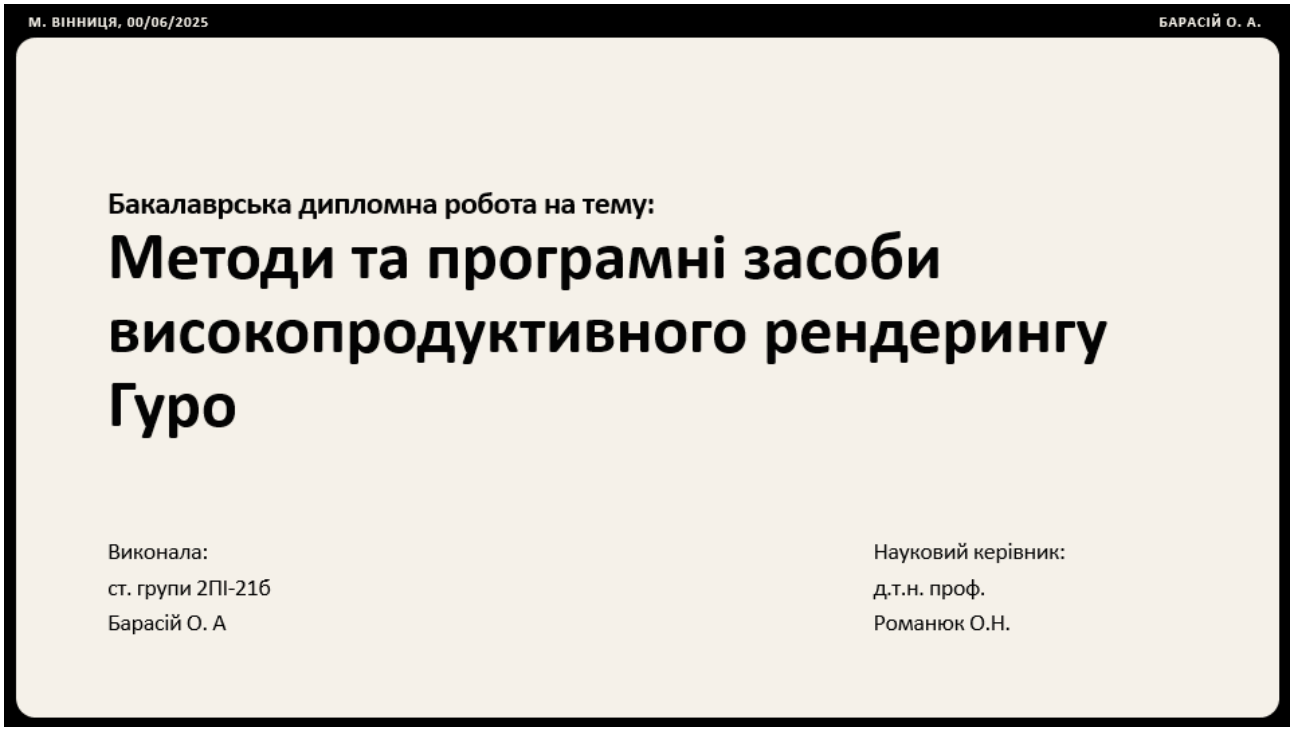

Рисунок Г.1 – Титульний слайд

Рисунок Г.2 – Галузі застосування графічних засобів

Рисунок Г.3 – Основні етапи графічного конвеєра

Рисунок Г.4 – Аналіз методів зафарбовування

Рисунок Г.5 – Мета і завдання дослідження

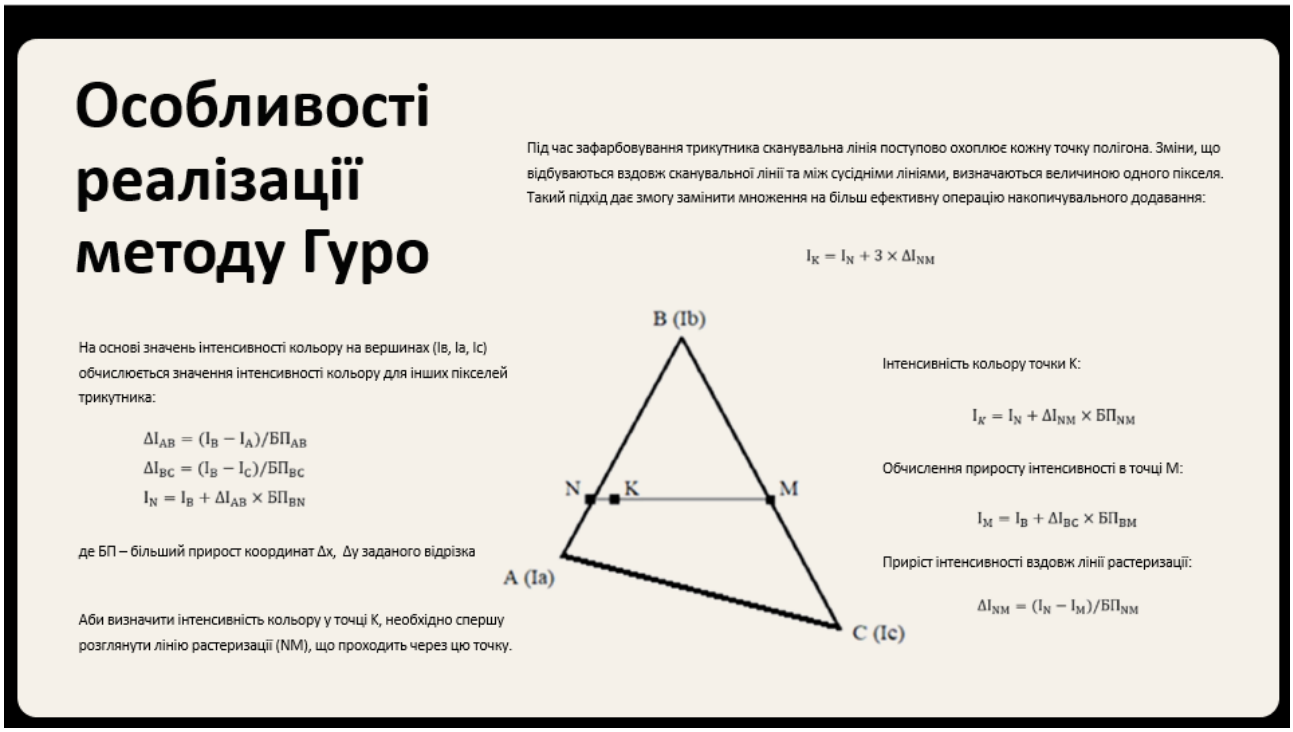

Рисунок Г.6 – Особливості реалізації методу Гуро

Рисунок Г.7 – Модифікація методу шляхом триангуляції трикутника

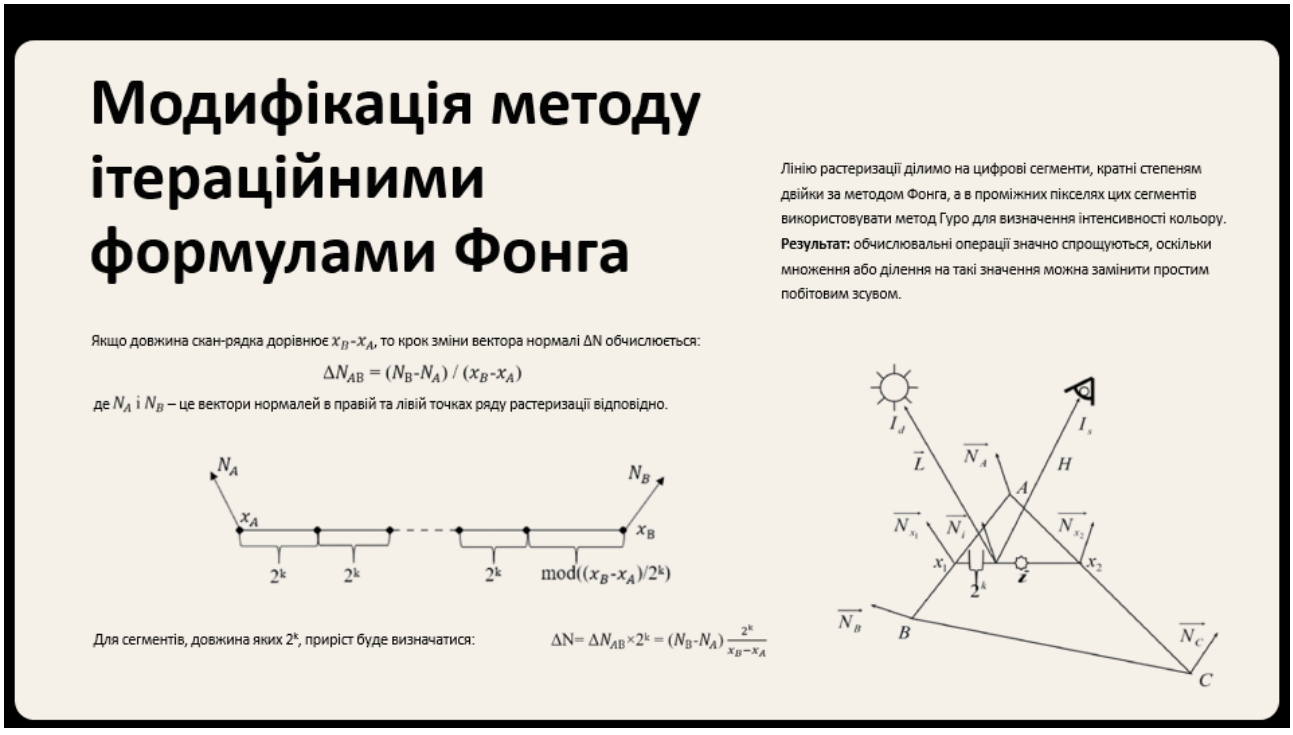

Рисунок Г.8 – Модифікація методу ітераційними формулами Фонга

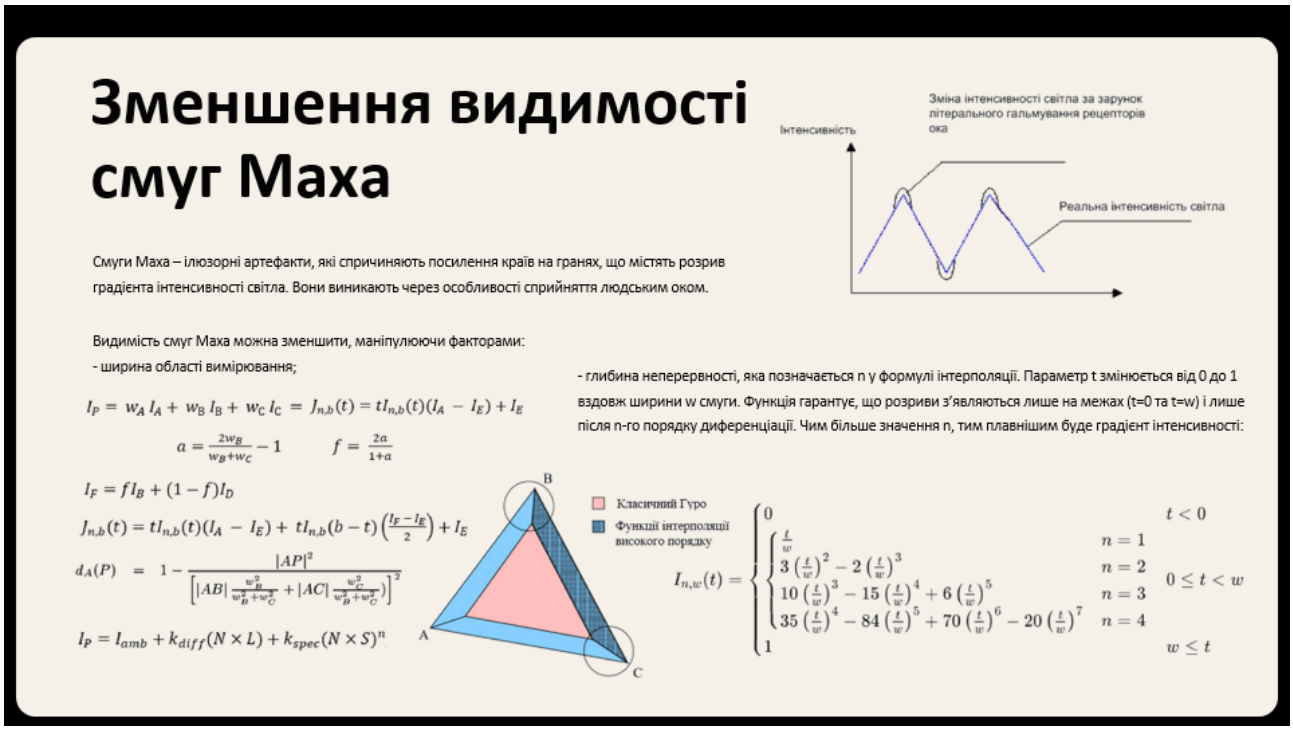

Рисунок Г.9 – Зменшення видимості смуг Маха

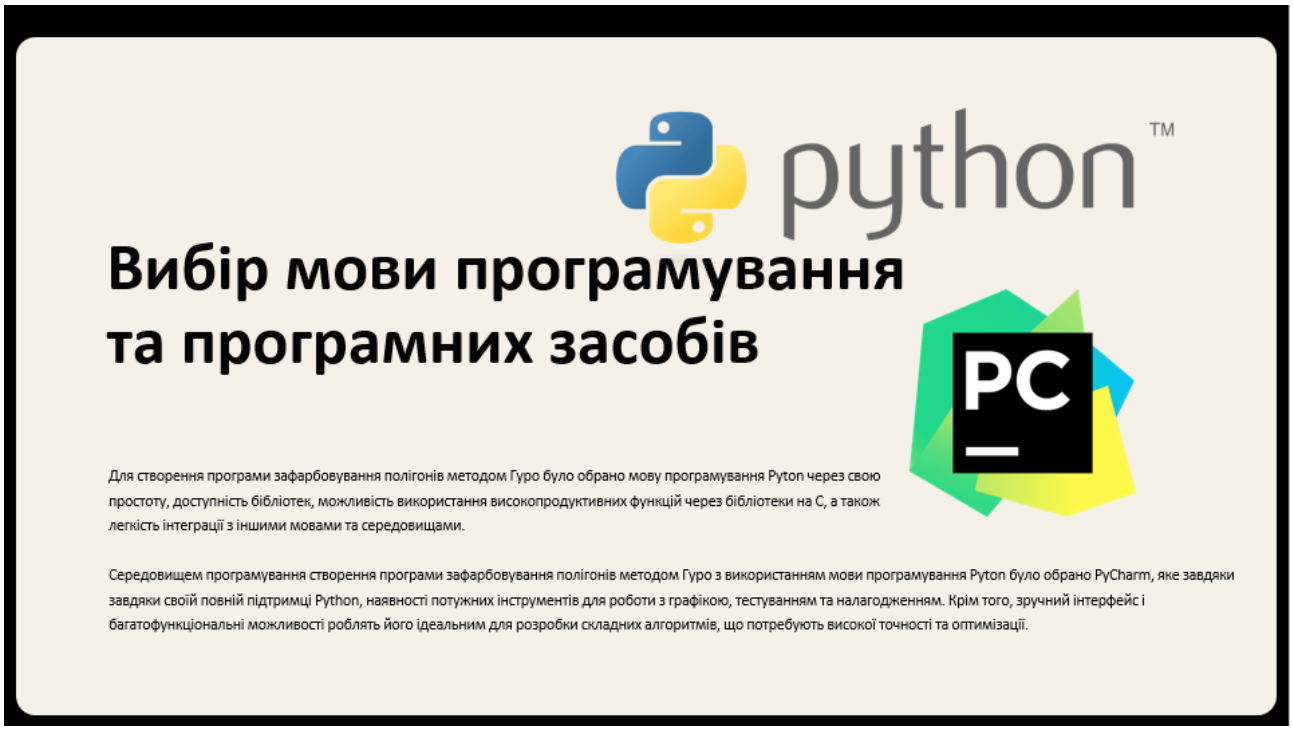

Рисунок Г.10 – Обрані мова програмування та програмний засіб

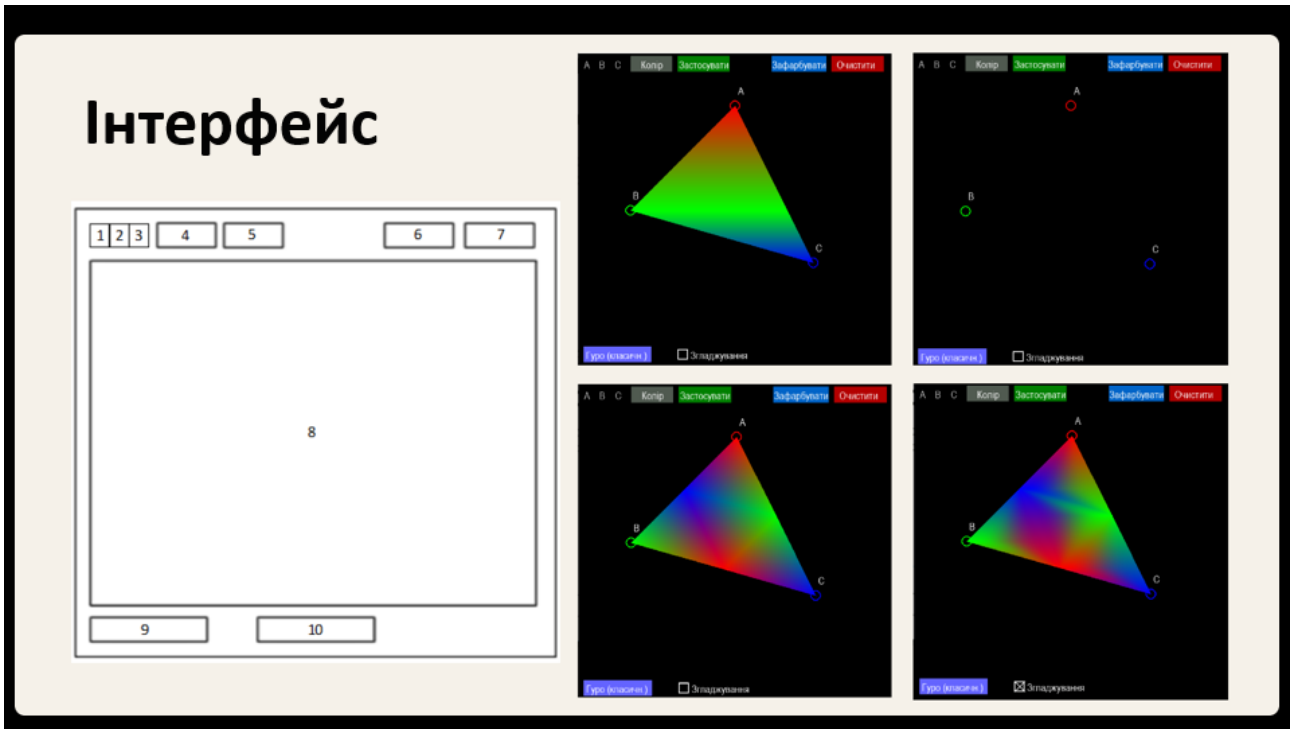

Рисунок Г.11 – Інтерфейс програмного застосунку

Рисунок Г.12 – Блок-схема зафарбовування класичним методом Гуро

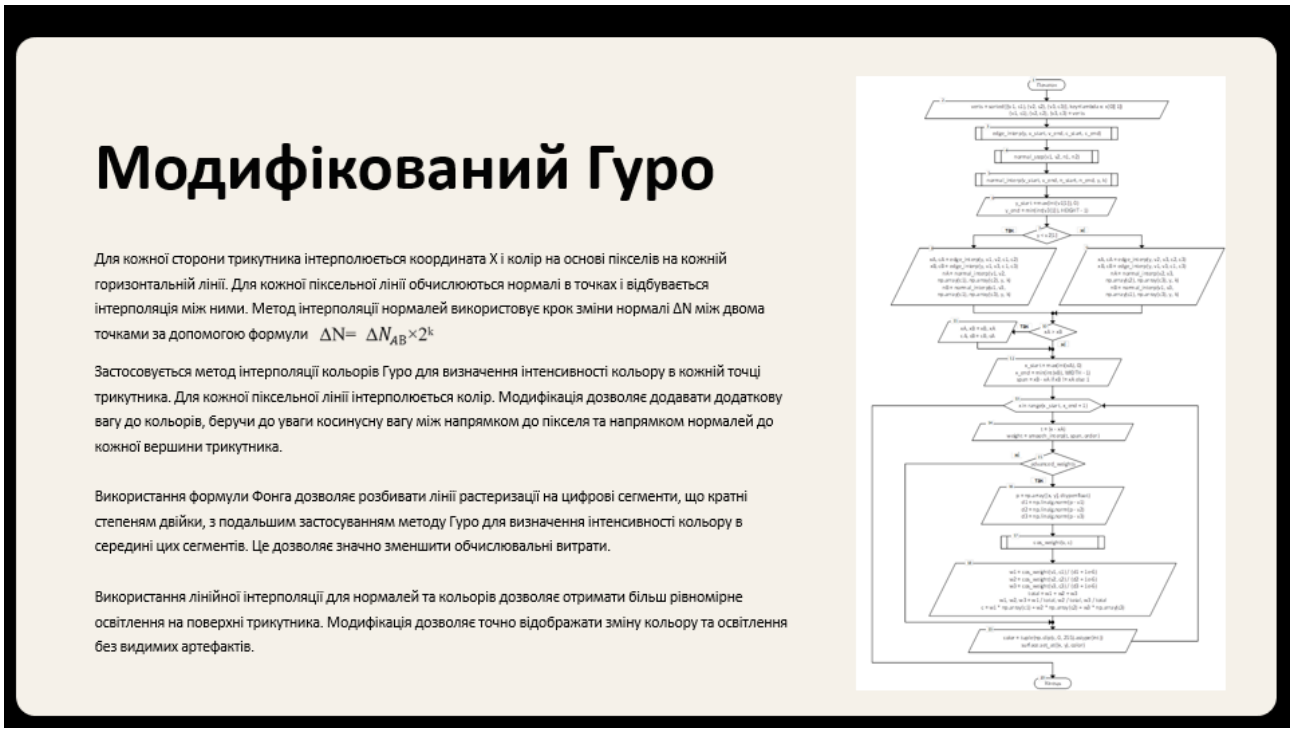

Рисунок Г.13 – Блок-схема зафарбовування модифікованим методом Гуро

Рисунок Г.14 – Блок-схема методу для згладжування смуг Маха

Рисунок Г.15 – Наукова новизна, практична цінність

Рисунок Г.16 – Заключний слайд