Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Високопродуктивні програмно-апаратні засоби для рендерингу графічних сцен»

08-54.БКР.007.00.000 ПЗ

ПОЯСНЮВАЛЬНА ЗАПИСКА

Виконав: студент 4 курсу, групи 1КІ-23мс
спеціальності 123 – Комп'ютерна інженерія
освітня програма – Комп'ютерна інженерія
(шифр і назва спеціальності)
Дембіцький О.М.
(прізвище та ініціали)

Керівник: к.т.н., доц., доцент кафедри ОТ
Черняк О. І.
(прізвище та ініціали)

Рецензент:
Романюк О. Н.
(прізвище та ініціали)

Допущено до захисту
завідувач кафедри ОТ
д.т.н., проф. Азаров О. Д.
«18» 06 2025 р.

Вінниця ВНТУ – 2025 рік

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії Кафедра обчислювальної техніки
Освітньо-кваліфікаційний рівень бакалавр
Галузь знань – 12 – Інформаційні технології
Спеціальність – 123 – Комп'ютерна інженерія
Освітньо-професійна програма – Комп'ютерна інженерія

ЗАТВЕРДЖУЮ
Завідувач кафедри ОТ
д.т.н., проф. Азаров О. Д.

_____2025року

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

Дембіцькому Олександру Миколайовичу

1 Тема роботи: «Високопродуктивні програмно-апаратні засоби для рендерингу графічних сцен», керівник роботи Черняк О. І. к.т.н., доцент кафедри ОТ затверджені наказом вищого навчального закладу від "21" березня 2025 року № 97

2 Термін подання студентом роботи 13.05.2025 р.

3 Вихідні дані до роботи:

Базовий метод зафарбовування – метод Фонга; моделі освітлення – моделі Бліна, Фонга ; графічний режим – TrueColor; вихідні дані для зафарбовування – напрямки векторів нормалей до вершин трикутників; дані про розташування джерела світла та спостерігача; коефіцієнт спекулярності поверхні; координати вершин трикутників; максимальна розрядність для задання адрес вершин трикутників –12; вихідні дані – кінцеве зображення, зафарбоване згідно методу Фонга в поєднанні з моделлю освітлення Бліна або Ламбертова.

4 Зміст розрахунково-пояснювальної записки

Вступ. Аналіз методів і засобів рендерингу. Прискорене визначення нормалізованих векторів. Програмно-апаратна реалізація методів зафабування. Практична реалізація рендеренгу в системах комп'ютерної графіки. Висновки. Перелік джерел. Додатки.

5 Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) Графічний конвеєр. Основні етапи рендерингу. Порівняльна таблиця методів зафарбовування. Структурні схеми пристроїв для рендерингу. Граф-схеми алгоритмів

6 Консультанти розділів роботи надано в табл. 1

Таблиця 1 — Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Спеціальна частина	к.т.н., доц., доцент кафедри ОТ Черняк О.І.	21.03.2025	30.05.2025
Нормоконтроль	асист. Швець С. І.	14.05.2025	30.05.2025

6 Дата видачі завдання 21.03.2025 р.

Календар видачі завдань наведено в Таб. 2

Таблиця 2 — Календарний план

№	Назва та зміст етапу	Термін виконання		Примітка
		початок	закінчення	
1.	Вибір, узгодження та затвердження теми БКР	21.03.2025	22.03.2025	Вик.
2.	Аналіз літературних джерел. Попередня розробка основних розділів	22.03.2025	30.03.2025	Вик.
3.	Розробка технічного завдання (ТЗ)	1.04.2025	08.04.2025	Вик.
4.	Аналіз вирішення поставленої задачі на даному етапі	09.04.2025	18.04.2025	Вик.
5.	Розробка програмно-апаратної реалізації методів зафарбування	19.04.2025	29.04.2025	Вик.
6.	Вивчення пристрою для визначення спекулярної складової кольору	30.03.2025	12.05.2025	Вик.
7.	Практична реалізація рендерингу	12.05.2025	27.05.2025	Вик.
9.	Оформлення пояснювальної записки та графічної частини	27.04.2025	12.05.2025	Вик.
10.	Нормоконтроль	13.05.2025	13.05.2025	Вик.
11.	Попередній захист БКР, доопрацювання, рецензування БКР	13.05.2025	01.05.2025	Вик.
12.	Захист БКР ЕК	02.06.2025	04.06.2025	Вик.

Студент _____ Дембіцький О.М.

Керівник роботи _____ Черняк О.І.

АНОТАЦІЯ

УДК 004.92

Дембіцький О.М. Високопродуктивні програмно-апаратні засоби для рендерингу графічних сцен, бакалаврська дипломна робота зі спеціальності 123 — Комп'ютерна інженерія, освітня програма — Комп'ютерна інженерія. Вінниця: ВНТУ, 2025. 96 с.

На укр. мові. Бібліогр.: рис.: 32; ліст.: 3; посил.: 21.

Доведено, що рендеринг є важливим етапом формування тривимірної графіки, який забезпечує візуальну реалістичність зображень шляхом моделювання освітлення, відбиття та тіней, і саме він найбільш ресурсоємний у графічному конвеєрі. Обгрунтовано, що програмно-апаратна реалізація дозволяє значно підвищити швидкодію рендерингу, поєднуючи гнучкість програмного забезпечення з паралелізмом і енергоефективністю апаратних засобів. Запропоновано нову модель відбивної здатності поверхні, що зменшує обчислювальну складність за рахунок зниження ступеня в аналітичних виразах. Розроблено ефективні алгоритми нормалізації векторів, зокрема з використанням бінарного поділу та поліноміальної інтерполяції другого ступеня, які скорочують час обчислень у 2–2.4 раза при незначних похибках. Було реалізовано апаратний модуль для визначення спекулярної складової кольору, що дозволяє винести найзатратніші обчислення на рівень спеціалізованої логіки. Створено структурні схеми пристроїв та алгоритми для апаратної частини, що реалізують етапи растеризації, нормалізації, освітлення та текстурування в реальному часі. Сформовано критерії оцінки ефективності програмно-апаратного рендерингу.

Ключові слова: рендеринг, модель освітлення, комп'ютерна графіка, системи рендерингу, растеризація, нормалізація

ABSTRACT

Dembitsky O.M. High-performance software and hardware for rendering graphic scenes

Bachelor's thesis in the specialty 123 — Computer Engineering, educational program — Computer Engineering. Vinnytsia: VNTU, 2025. 97 p.

In Ukrainian. Bibliography: fig.: 32; list.: 3; ref.: 21.

It is proven that rendering is an important stage of three-dimensional graphics formation, which provides visual realism of images by modeling lighting, reflection and shadows, and it is the most resource-intensive in the graphics pipeline. It is substantiated that software-hardware implementation allows to significantly increase the rendering speed, combining the flexibility of software with parallelism and energy efficiency of hardware. A new model of surface reflectivity is proposed, which reduces computational complexity by reducing the degree in analytical expressions. Effective algorithms for vector normalization have been developed, in particular using binary division and polynomial interpolation of the second degree, which reduce the computation time by 2–2.4 times with insignificant errors. A hardware module has been implemented to determine the specular component of color, which allows to transfer the most expensive calculations to the level of specialized logic. Structural diagrams of devices and algorithms for the hardware part that implement the stages of rasterization, normalization, lighting, and texturing in real time have been created. Criteria for evaluating the effectiveness of software-hardware rendering have been formed.

Keywords: rendering, lighting model, computer graphics, rendering systems, rasterization, normalization

ЗМІСТ

ВСТУП

Актуальність теми, рендеринг є основою для створення візуальних зображень, що сприяє точному та реалістичному представленню даних у різноманітних галузях, надаючи користувачам можливість взаємодіяти з цифровими об'єктами та сценаріями.

Підвищення продуктивності рендерингу є важливим аспектом у розвитку сучасної комп'ютерної графіки та візуальних ефектів, оскільки він дозволяє значно зменшити час, необхідний для створення високоякісних зображень або анімацій, а також покращує ефективність роботи з великими обсягами даних

Програмно-апаратна реалізація відіграє важливу роль у досягненні високої ефективності та оптимізації процесів в різноманітних сферах, від обробки сигналів і рендерингу до розробки систем для вбудованих додатків і високопродуктивних обчислень. Такий підхід поєднує переваги програмного забезпечення (гнучкість і можливість оновлень) та апаратного забезпечення (висока швидкість і енергоефективність), що дозволяє створювати більш ефективні, надійні та масштабовані рішення.

Програмно-апаратні рішення можуть сприяти зменшенню вартості виготовлення та часу на розробку нових продуктів. Апаратні компоненти можна повторно використовувати в різних проєктах, що дозволяє скоротити витрати на розробку і зменшити ризики, пов'язані з створенням нових рішень.

Таким чином, доцільність програмно-апаратної реалізації полягає в тому, щоб поєднати переваги як апаратних, так і програмних компонентів, що дозволяє створювати високо ефективні, швидкі, енергоефективні та гнучкі рішення для різноманітних застосувань, від рендерингу і криптографії до обробки даних у реальному часі та вбудованих систем.

Зв’язок роботи з науковими програмами, планами, темами. Робота виконувалася згідно плану виконання наукових досліджень на кафедрі обчислювальної техніки.

Метою дослідження є підвищення продуктивності рендерингу за рахунок програмно-апаратної реалізації

Основними задачами дослідження є такі:

— провести аналіз існуючих методів і засобів рендерингу для визначення напрямків підвищення їх продуктивності за рахунок програмно-апаратної реалізації;

— запропонувати нові методи рендерингу;

— розробити алгоритми для програмно-апаратної реалізації рендерингу;

— розробити програмні та апаратнв компоненти на основі запропонованих алгоритмів ;

— провести експериментальні дослідження розроблених засобів рендерингу.

Об'єкт дослідження — процес кінцевої візуалізації (рендерингу) тривимірних об'єктів у системах комп'ютерної графіки.

Предмет дослідження — методи та засоби програмно-апаратної реалізації рендерингу тривимірних графічних зображень.

У процесі досліджень використовувались **такі методи дослідження** теорія чисел та чисельних методів, теорія алгоритмів, методи аналітичної геометрії для розробки методів і засобів програмно-апаратної реалізації рендерингу; комп'ютерне моделювання для аналізу та перевірки отриманих теоретичних положень.

Новизна отриманих результатів:

1.Запропоновано нову модель відбивної здатності поверхні, яка відрізняється від відомою меншою степенню, що дало можливість зменшити час рендерингу.

2.Запропоновані нові методи розділення програмних і апаратних функцій, що дозволяють підвищити швидкодію формування тривимірних графічних зображень.

Практичне значення отриманих результатів полягає в тому, що на основі отриманих в бакалаврській кваліфікаційній роботі теоретичних положень запропоновано алгоритми та розроблено програмно-апаратні програмні засоби рендерингу для комп'ютерних систем високореалістичної візуалізації тривимірних зображень.

Особистий внесок здобувача. Усі наукові результати, викладені у БКР , отримані автором особисто. У друкованих працях, опублікованих у співавторстві, автору належать такі результати: перспективи розвитку методу Фонга [1]; модифікація методу Фонга [2]; метод підвищення реалістичності методу Гуро [3]; аналіз інновацій в GPU[4];
методи розподілення програмних і апаратних функцій [5].

Апробація матеріалів бакалаврської роботи. Основні положення БДР доповідалися та обговорювалися на II Міжнародній науковій конференції «Наукові горизонти XXI століття: мультидисциплінарні дослідження» (Ужгород, 2025), Всеукраїнській науково-практичній Інтернет-конференції «Автоматизація та комп’ютерно- інтегровані технології у виробництві та освіті: стан, досягнення, перспективи розвитку (Черкаси, 2025).

Публікації. Основні результати досліджень опубліковано в 5 наукових працях, у матеріалах конференцій.

1. АНАЛІЗ МЕТОДІВ І ЗАСОБІВ РЕНДЕРИНГУ

1.1. Основні етапи графічного конвеєра

Графічний конвеєр [6-7] (рис. 1.1) - це послідовність етапів, які виконуються для перетворення [8] тривимірної сцени у двовимірне зображення на екрані. Першим етапом є опис сцени, що включає задання геометрії об'єктів, матеріалів, світла та камер. Далі виконується теселяція [9, 10] — розбиття складних поверхонь на прості примітиви, зазвичай трикутники, що спрощує подальші обчислення. Потім застосовуються афінні перетворення, зокрема масштабування, обертання та перенесення об'єктів у світовій системі координат. На етапі видових перетворень сцена трансформується у координатну систему камери, визначаючи, що саме потрапляє в зону видимості. Далі йде оброблення вершин, у якому відбувається трансформація кожної вершини (наприклад, у віконну систему координат), а також розрахунок освітлення і нормалей. Після цього виконується формування каркасної моделі, що означає побудову полігональних структур з використанням відповідних вершин. Далі відбувається оброблення полігонів, включно з розсіченням, відкиданням або відбором примітивів згідно з правилами фруструму. Наступний етап — растеризація, яка перетворює полігони у пікселі на екрані. Під час видалення невидимих ліній та поверхонь система визначає, які фрагменти мають бути показані, а які — заблоковані іншими об'єктами. Далі накладаються текстури — процес текстурування, що додає деталізацію та колір. Потім йде зафарбовування, яке враховує текстури, освітлення, шейдери та прозорість для отримання кольору кожного пікселя. На завершення виконується фінальна обробка [11, 12]: застосовуються ефекти, як-от розмиття руху, глибина різкості, згладжування країв та тонова компресія, після чого зображення виводиться на екран.

Етап зафарбовування [12] є одним із найтрудомісткіших у графічному конвеєрі, оскільки саме на ньому здійснюється інтенсивна обробка кожного фрагмента зображення. Після того як тривимірні об'єкти перетворено на двовимірні пікселі через растеризацію, система повинна виконати індивідуальні обчислення

для кожного з цих фрагментів. Для екрана стандартної роздільної здатності 1920×1080 це понад два мільйони фрагментів на кадр, а при 60 кадрах на секунду — більше 120 мільйонів фрагментів, які потребують обробки щосекунди (рис.1.1).

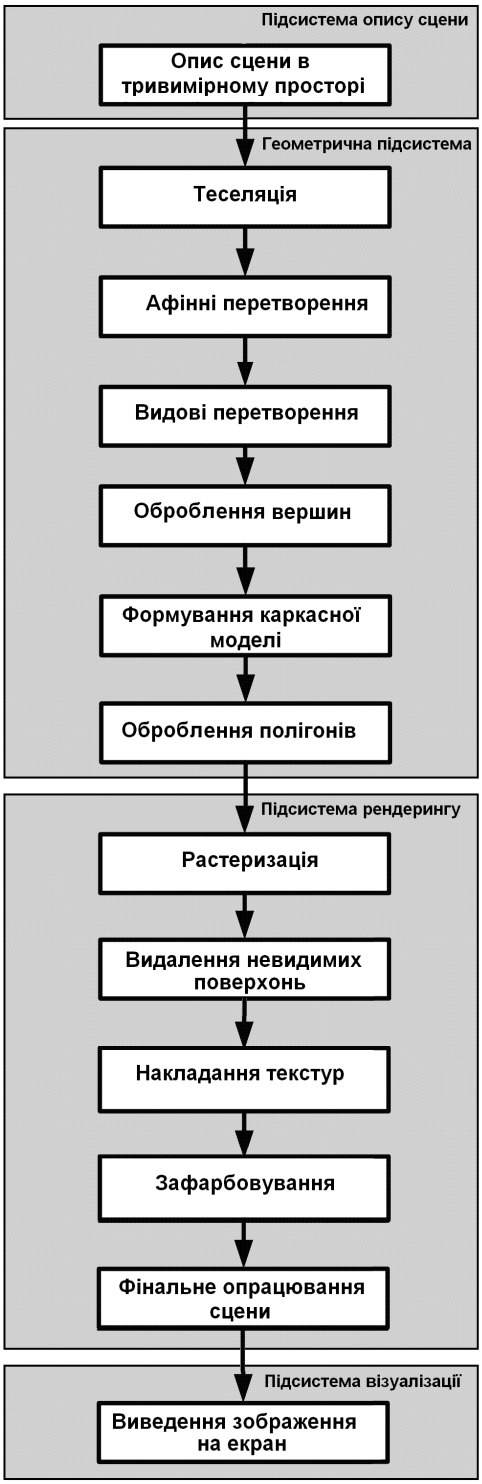


Рисунок 1.1 Етапи формування тривимірного зображення

Кожен фрагментний рейдер [13-16] виконує складні обчислення: розрахунок

нормалей, інтерполяцію координат, освітлення, доступ до текстур, обробку прозорості та ефектів самосвітіння. У випадку фізично обґрунтованого рендерингу [6] застосовуються ще складніші моделі, зокрема Cook-Torrance BRDF, що вимагають виконання операцій добутків, ділення, степеневих функцій та квадратних коренів.

Якщо в сцені присутні кілька джерел світла, то для кожного з них повторюються окремі розрахунки освітлення для кожного фрагмента, що експоненційно збільшує обчислювальне навантаження.

Сучасні відеокарти, такі як NVIDIA RTX або AMD RDNA, мають десятки тисяч потокових процесорів, з яких значна частина призначена саме для паралельної обробки фрагментів. У рушіях Unity або Unreal [16] профайлер показує, що саме фрагментні шейдери найчастіше є основними споживачами обчислювальних ресурсів.

Навіть у найпростішому прикладі, де фрагмент лише звертається до текстури та виконує базову формулу освітлення, сумарне навантаження в мільйонах операцій на секунду перевищує обробку геометрії у кілька разів. Усе це свідчить про те, що рендеринг є не лише ключовим для візуальної якості, а й найвимогливішим до ресурсів під час рендерингу сцени.

Наведемо таблицю часу виконання етапу зафарбовування у графічному конвеєрі, який залежить від складності сцени, роздільної здатності, типу освітлення, кількості текстур і моделі рендерингу. Приблизні середні значення на фрагмент або кадр, виміряні в мікросекундах або мілісекундах подано в таблиці 1.1

1.2. Методи зафарбовування у рендерингу

Методи зафарбовування у рендерингу є основним етапом обчислення остаточного вигляду тривимірної сцени на зображенні, оскільки саме вони визначають, як світло взаємодіє з поверхнями об'єктів. Основна задача методів зафарбовування полягає у визначенні кольору пікселя або фрагмента на основі фізичних або емпіричних моделей освітлення. Існує кілька основних підходів до зафарбовування, кожен з яких має свої переваги, недоліки та сфери застосування.

Таблиця 1.1 — Орієнтовні оцінки часу виконання на один кадр

Етап графічного конвеєра	Складність обробки	Орієнтовний час (мс/кадр)	Частка від загального часу (%)
1.Опис сцени	Низька	<0.1 мс/кадр	<0.1%
2.Теселяція	Середня	від 1 до 2 мс/кадр	від 2% до 5%
3.Афінні перетворення	Низька	0.5 мс/кадр	від 1% до 2%
4.Видові перетвореня	Низька	0.2 мс/кадр	<1%
5.Обробка вершин (Vertex Shader)	Середня	від 1 до 2 мс/кадр	5%
6.Формуваня примітивів	Низька	0.2 мс/кадр	<1%
7.Clipping, Culling	Середня	від 0.5 до 1 мс/кадр	2%
8.Растеризація	Середня	від 2 до 3 мс/кадр	5%
9.Видалення невидимих фрагментів	Середня	від 1 до 2 мс/кадр	від 3% до 5%
10.Зафарбування (Shading)	Висока	від 15 до 50 мс/кадр	від 40% до 70%
11.Текстурування	Висока	від 5 до 15 мс/кадр	від 10% до 20%
12.Фінальнаобробка (PostFX)	Висока	від 3 до 5 мс/кадр	від 10% до 15%
Усього		25-75 мс/кадр	100%

Його суть полягає у тому, що кожен полігон об'єкта (найчастіше трикутник або чотирикутник) зафарбовується єдиним кольором, який визначається на основі

обчислення освітлення в одній точці цього полігона — зазвичай у центрі або у якійсь із вершин. Отже, всі пікселі, що належать одному полігону, мають однакове значення кольору та яскравості.

Алгоритм плоского зафарбовування складається з декількох основних етапів. Спочатку для кожного полігона обчислюється нормаль до поверхні. Оскільки полігон у плоскому зафарбовуванні розглядається як цілісна одиниця, нормаль є сталою для всіх його точок і визначається через векторний добуток двох його сторін (ребер). Далі використовується певна модель освітлення (найчастіше базова ламбертова модель дифузного відбиття), яка розраховує інтенсивність світла, що падає на полігон, з урахуванням нормалі полігона, напрямку світла, кольору матеріалу та інших параметрів. Після цього результат застосовується до всіх пікселів цього полігона під час растеризації.

Основною перевагою плоского зафарбовування є його надзвичайна обчислювальна простота та висока швидкість виконання. Свого часу це дозволяло отримувати тривимірні сцени навіть на слабких обчислювальних системах або в режимах, де потрібна максимальна продуктивність при мінімальних затратах ресурсів. Візуально результат плоского зафарбовування має характерний "гранований" вигляд: окремі полігони легко помітні, а поверхня виглядає "ламаною" навіть на об'єктах, які в реальності повинні бути гладкими.

Серед недоліків плоского зафарбовування варто відзначити відсутність плавності переходів між полігонами. Оскільки зміна освітлення відбувається тільки на межі полігонів, об’єкти виглядають "східчасто", особливо якщо геометрія є малополігональною. Це погано підходить для рендерингу реалістичних сцен, де потрібні плавні градієнти освітлення і деталізація складних поверхонь.

Технічно реалізація плоского зафарбовування не вимагає інтерполяції кольорів або нормалей під час растеризації: під час малювання кожного трикутника просто використовується єдине значення кольору, розраховане за його нормаллю. Це дозволяє мінімізувати обчислення і зробити рендеринг надзвичайно ефективним. У класичних графічних API, таких як OpenGL, для плоского зафарбовування навіть існує спеціальний режим роботи растеризатора.

Плоске зафарбовування може мати художню цінність. Воно часто використовується у стилізованій графіці, у low-poly мистецтві або в ситуаціях, де необхідно підкреслити структуру об'єкта або створити спеціальний естетичний ефект. Наприклад, у сучасних іграх і візуальних проєктах плоске зафарбовування може використовуватись для створення унікального, впізнаваного візуального стилю.

Отже, плоске зафарбовування є базовим методом, який не тільки має історичне значення у розвитку тривимірної графіки, а й продовжує знаходити застосування у певних сферах мистецької графіки, а також у розробці технічних прототипів та ресурсозберігаючих застосунків.

Метод Гуро [3, 6] є класичним способом інтерполяційного зафарбовування в тривимірній комп'ютерній графіці, який було запропоновано Анрі Гуро у 1971 році. Основною метою цього методу є згладжування переходів освітлення між сусідніми полігонами для досягнення більш реалістичного вигляду поверхонь без різких контрастів між ними, що було характерним для простого плоского зафарбовування. уть методу Гуро полягає у тому, що інтенсивність освітлення обчислюється тільки у вершинах полігонів, а потім ця інтенсивність лінійно інтерполюється всередині полігона під час растеризації. Сам розрахунок освітлення в методі Гуро здійснюється на основі певної моделі освітлення, наприклад, класичної моделі Фонга, але із суттєвим спрощенням: замість обчислення освітлення в кожній точці поверхні, воно обчислюється лише у ключових точках (вершинах).

Розглянемо етапи роботи методу Гуро детально. На першому кроці для кожної вершини тривимірної моделі обчислюється нормаль до поверхні. Якщо модель складається з плоских полігонів (наприклад, трикутників), то нормаль вершини визначається як усереднення нормалей всіх сусідніх полігонів, що з’єднані з цією вершиною. Такий підхід забезпечує плавність переходів освітлення між різними гранями. На другому кроці для кожної вершини за допомогою обраної моделі освітлення розраховується яскравість або колір у цій вершині, враховуючи положення джерел світла, положення камери та властивості матеріалу об'єкта. На третьому етапі при растеризації кожного трикутника проводиться лінійна

інтерполяція значень яскравості або кольору між вершинами для кожного пікселя всередині трикутника.

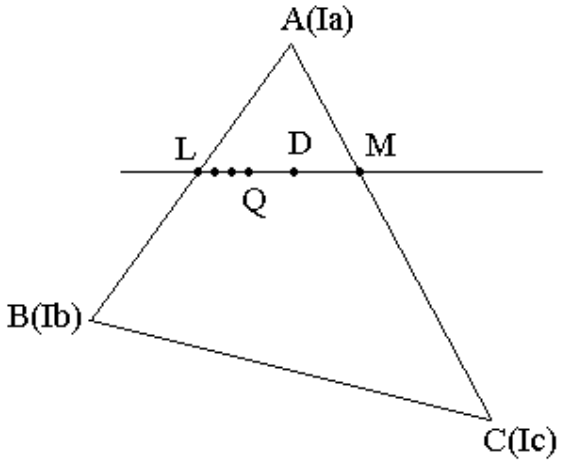


Рисунок 1.2 — Визначення інтенсивності кольору в рядку растеризації

Переваги методу Гуро полягають у його простоті та ефективності. Оскільки дорогі обчислення освітлення виконуються лише у вершинах, а не для кожного пікселя окремо, цей метод є значно швидшим за більш точні методи, такі як зафарбовування Фонга. Він добре працює для об'єктів із досить щільною сіткою, де відстань між вершинами невелика, що дозволяє забезпечити достатньо плавні переходи.

Проте метод має і певні недоліки. Найсуттєвіший з них полягає у тому, що інтерполяція базується на яскравості, а не на нормалях. Через це висококонтрастні ефекти, такі як дзеркальні відблиски, і повинні бути різкими і точковими, можуть бути втрачені або розмиті, оскільки їхня максимальна інтенсивність може не потрапити у вершину полігона і, відповідно, не враховуватись у загальній інтерполяції.

Математично інтерполяція в методі Гуро здійснюється шляхом двоетапного лінійного процесу. Спочатку обчислюється інтерполяція інтенсивності уздовж кожної сторони трикутника, що з'єднує вершини, а потім між цими проміжними значеннями здійснюється інтерполяція для кожного пікселя на скан-лінії. Таке рішення дуже добре підходить для апаратної реалізації, наприклад, у ранніх

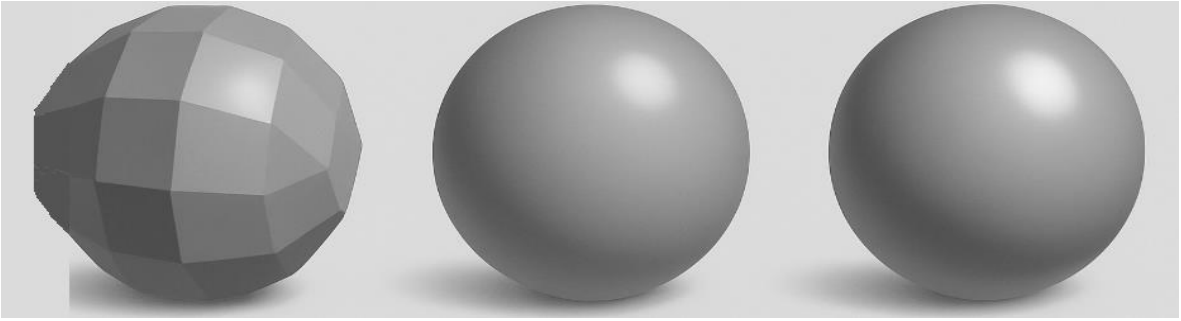
графічних прискорювачах.

Підсумовуючи, метод Гуро є історично важливим кроком у розвитку комп'ютерної графіки, який дозволив перейти від жорстких, грубих моделей освітлення до значно реалістичнішого зображення об'єктів при збереженні розумного балансу між якістю та обчислювальними витратами. Сьогодні метод Гуро рідше використовується у своєму початковому вигляді, поступившись місцем більш точним методам зафарбовування нормалей, однак він залишається важливим для розуміння принципів освітлення у рендерингу

Метод Фонга [1-2, 6, 11, 16-22] є одним із класичних і фундаментальних підходів до зафарбовування в комп’ютерній графіці, який запропонував Буй Туонг Фонг (Bui Tuong Phong) у 1973 році. Він був розроблений як удосконалення методу Гуро для кращого відображення реалістичних ефектів освітлення на поверхнях, зокрема для точнішого відтворення блискучих відблисків і плавності переходів освітлення.

Суть методу Фонга полягає у тому, що замість інтерполяції яскравості між вершинами, як це робить метод Гуро, інтерполюються нормалі поверхні, а саме освітлення обчислюється для кожного пікселя або фрагмента окремо на основі інтерпольованої нормалі. Завдяки цьому можна досягти значно вищої точності освітлення, зокрема чітких і правильних спекулярних ефектів, навіть на великих трикутниках, без потреби щільної геометричної сітки.

Процес реалізації методу Фонга включає кілька етапів. На першому кроці для кожної вершини об'єкта обчислюється нормаль, аналогічно до інших методів — зазвичай як середнє значення нормалей суміжних полігонів для забезпечення згладженості. На другому кроці у процесі растеризації ці нормалі лінійно інтерполюються по всій поверхні трикутника для кожного пікселя (або фрагмента). І нарешті, у кожній точці інтерпольована нормаль нормалізується (перетворюється в одиничний вектор), після чого для неї локально обчислюється модель освітлення.



а) плоске зафарбовування б) метод Гуро в) метод Фонга

Рисунок 1.4 Приклади зафарбовування

У класичній реалізації методу Фонга використовується спрощена емпірична модель освітлення, яка складається з трьох основних компонентів: дифузної, спекулярної та амбієнтної складових.

Амбієнтне освітлення [6, 17] (Ambient Light) моделює глобальне розсіяне світло, яке заповнює всю сцену і не має конкретного напрямку. Воно забезпечує базову видимість об’єктів навіть у тіні. Його внесок є просто постійним множенням певного коефіцієнта амбієнтного освітлення на базовий колір матеріалу.

Дифузне освітлення[6, 17] (Diffuse Light) моделює розсіювання світла на шорстких поверхнях. Його інтенсивність залежить від косинуса кута між напрямком світла і нормаллю поверхні (закон Ламберта). Математично це обчислюється як скалярний добуток нормалізованої нормалі і нормалізованого вектора напрямку до джерела світла.

Спекулярне освітлення [1, 2, 11] (Specular Light) моделює віддзеркалення світла на гладких поверхнях, тобто блиск. У класичній моделі Фонга для обчислення спекулярної складової використовується напрямок відбиття світла від поверхні і напрямок на камеру (глазовий вектор). Чим ближче ці два вектори співпадають, тим яскравіший спекулярний відблиск. Інтенсивність спекулярного світла зменшується за експоненціальним законом залежно від кута відхилення, піднятого до певного ступеня (званого коефіцієнтом блискучості).

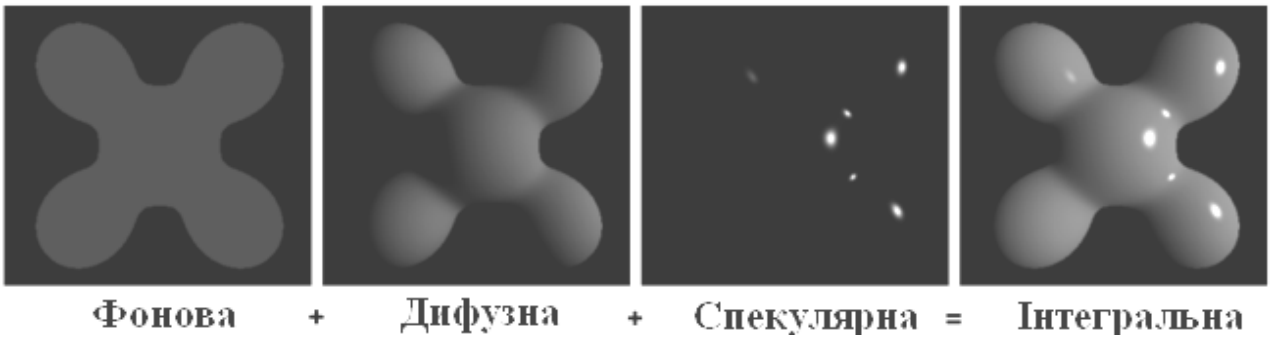
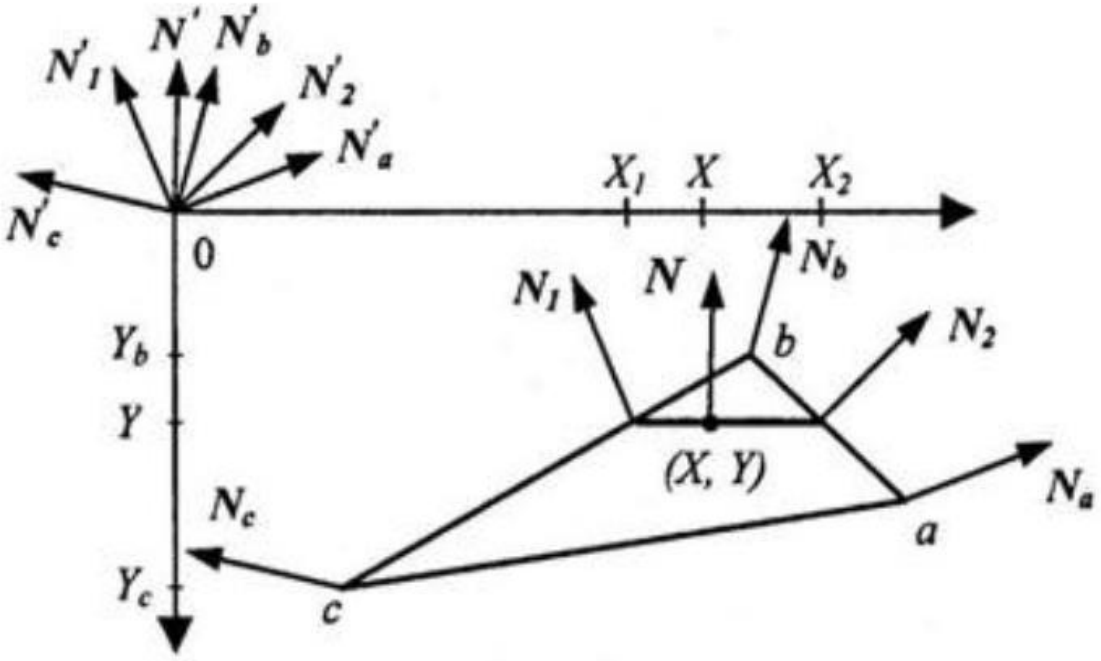


Рисунок 1.5.- Складові кольору



Де вектори:

N — нормаль до поверхні в точці рендерингу (нормалізована),

L — нормалізований вектор напрямку на джерело світла,

R — вектор відбиття світла від поверхні,

V — вектор напрямку на спостерігача (камеру),

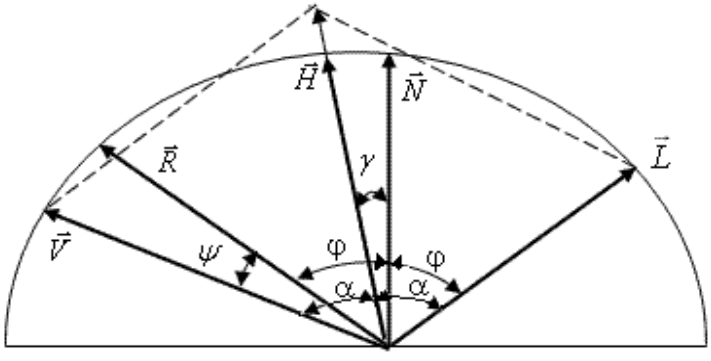


Рисунок 1.6. Визначення спекулярної складової в методі Фонга

Математично освітлення в методі Фонга можна описати так:

$$I = I_{ambient} + I_{diffuse} + I_{specular} \quad (6)$$

Де

$I_{diffuse} = k_d * (N \bullet L) * I_{light}$

$I_{specular} = k_s * (R \bullet V)^n * I_{light}$

де,

N — нормаль до поверхні в точці рендерингу (нормалізована);

L — нормалізований вектор напрямку на джерело світла;

R — вектор відбиття світла від поверхні;

V — вектор напрямку на спостерігача (камеру);

k_d — коефіцієнт дифузного відбиття;

k_s — коефіцієнт спекулярного відбиття;

n — показник блиску;

I_{light} — інтенсивність світла.

Метод Фонга має суттєві переваги: він забезпечує візуально набагато кращу якість, ніж метод Гуро, з чіткими та реалістичними блискучими відблисками, плавними переходами освітлення і природним виглядом шорстких і глянцевих матеріалів. Його широко використовували в класичних ігрових рушіях, візуалізаціях та на початку розвитку тривимірної графіки у фільмах.

Втім, метод має й певні недоліки. Він є значно обчислювально дорожчим, оскільки вимагає розрахунку освітлення для кожного фрагмента (пікселя), а також регулярної нормалізації векторів під час інтерполяції нормалей, що накладає додаткові вимоги до апаратних ресурсів, особливо у старіших системах. Саме через ці обмеження у реальному часі виникли оптимізовані версії, наприклад, Blinn-Phong модель, яка замінює обчислення вектора відбиття на менш затратні обчислення середнього вектора між напрямком на камеру і напрямком на джерело світла.

У підсумку, метод Фонга став важливим кроком у розвитку тривимірної комп'ютерної графіки, заклавши основи для подальшого розвитку фізично коректних моделей освітлення. Він навчив розділяти різні типи освітлення та окремо обчислювати їхній внесок, що стало фундаментом для більш складних моделей, таких як Cook-Torrance або сучасні PBR-системи.

Фізично коректні методи зафарбовування [6, 11] (Physically Based Shading або PBS, також часто зустрічається термін Physically Based Rendering — PBR) представляють собою сучасний підхід до рендерингу поверхонь, який прагне моделювати взаємодію світла і матеріалів максимально наближено до реальних фізичних процесів. Головна ідея фізично коректного зафарбовування полягає в тому, щоб замінити емпіричні або наближені моделі освітлення (на кшталт моделей Фонга чи Бліна-Фонга) на точніші обчислення, що базуються на законах фізики, таких як закон збереження енергії, закон Френеля, модель мікрофасетної структури поверхні тощо.

У фізично коректних методах [11] використовуються складніші функції відбиття світла — так звані BRDF [6, 11, 15] (Bidirectional Reflectance Distribution Function) — які описують, як світло відбивається від поверхні залежно від кута падіння і кута спостереження. Правильний BRDF повинен задовольняти дві основні умови: симетричність (обмін напрямком падіння і спостереження не змінює результату) та закон збереження енергії (відбиття не може перевищувати кількість вхідної енергії).

Класичним прикладом BRDF [6, 8], який використовується в PBR-системах, є модель Кука-Торренса (Cook-Torrance BRDF). Вона враховує три основні фізичні компоненти: геометричне згасання (G), розподіл мікрофасетів (D) і коефіцієнт Френеля (F).

Функція F (Fresnel) моделює зміну кількості відбитого світла в залежності від кута падіння: чим більший кут між напрямком світла і нормаллю поверхні, тим сильніше поверхня відбиває світло (ефект особливо помітний у воді, склі, металах). Функція D (Normal Distribution Function) визначає розподіл мікрофасетів на поверхні — тобто, як багато маленьких дзеркал розташовані під різними кутами,

що визначає шорсткість або гладкість матеріалу. Функція G (Geometry or Shadowing-Masking Function) враховує ефекти самозатінення і маскування мікрофасетів. Окремо в фізично коректному зафарбовуванні важливу роль відіграє розділення матеріалів на металічні (conductors) і діелектричні (dielectrics). Метали мають специфічні властивості відбиття: вони повністю поглинають падаюче світло в поверхневих шарах і дають інтенсивне дзеркальне відбиття, тоді як діелектрики (наприклад, пластик, дерево, шкіра) частково заломлюють світло всередину матеріалу.

Методи фізично коректного зафарбовування також вимагають правильного обліку глобального освітлення (Global Illumination), оскільки світло в реальному світі відбивається багаторазово між поверхнями. Для цього використовуються технології на зразок ієрархічного трасування променів, фотонних карт або різноманітних методів апроксимації непрямого освітлення.

Перевагами фізично коректного зафарбовування є висока реалістичність сцени, узгодженість вигляду під різними умовами освітлення, можливість автоматичного масштабування якості (наприклад, шляхом зниження роздільності текстур або використання mipmapping) і полегшення художньої роботи завдяки стандартизованим параметрам. Недоліками є вища обчислювальна вартість і необхідність ретельного калібрування матеріалів і джерел світла для досягнення оптимального результату.

Наведемо таблицю порівняльного аналізу методів зафарбовування табл. 2.2

Таблиця 2.2 — Порівняльний аналіз методів зафарбовування

Умови рендерингу	Тип шейдера	Складність текстур і освітлення	Орієнтовний час зафарбовування (на фрагмент)	Орієнтовний час (на кадр, 1920×1080, 60 FPS)
Проста сцена, flat shading, без текстур	Basic Fragment Shader	Низька	~0.03–0.05 мкс	~5–10 мс
Середня сцена, Phong shading, 1 джерело світла	Medium Shader	Середня	~0.08–0.12 мкс	~15–20 мс
Сцена з PBR, BRDF, кілька текстур, кілька світел	PBR Shader	Висока	~0.2–0.4 мкс	~35–50 мс
Велика сцена, з	Complex	Дуже висока	~0.5–1.0 мкс	~60–80 мс

Умови рендерингу	Тип шейдера	Складність текстур і освітлення	Орієнтовний час зафарбовування (на фрагмент)	Орієнтовний час (на кадр, 1920×1080, 60 FPS)
тінями, AO, SSR, прозорістю	Shader			

У таблиці-– Flat shading — простий однотонний колір поверхні;–Phong shading — класична модель з обчисленням нормалей на фрагмент; – PBR — фізично обґрунтоване зафарбовування з BRDF і текстурними мапами. – AO, SSR — ambient occlusion, screen space reflections.

Показники актуальні для середнього GPU 2022–2023 рр., -NVIDIA RTX 3060–3080.

1.3 Критерії оцінювання програмно-апаратної реалізації рендерингу

Критерії оцінювання програмно-апаратної реалізації рендерингу охоплюють як технічні параметри, так і якісні характеристики результату. Насамперед, оцінюється продуктивність системи, що включає час рендерингу кадру, кількість кадрів за секунду (FPS)і загальну затримку обробки. Важливою є якість зображення, яка визначається реалістичністю освітлення, точністю текстур, наявністю або відсутністю артефактів, згладжуванням країв і загальним візуальним сприйняттям. Апаратна ефективність визначається тим, наскільки повно та стабільно задіяні ресурси GPU [4, 11], CPU або FPGA, а також рівнем енергоспоживання системи. Значну роль відіграє масштабованість — можливість запуску рендерингу на різних апаратних конфігураціях, включаючи мобільні пристрої чи серверні системи.

Гнучкість та модульність реалізації оцінюються за здатністю системи до зміни шейдерів, впровадження нових візуальних ефектів або оновлення алгоритмів без модифікації апаратного забезпечення. Підтримка сучасних графічних стандартів, таких як OpenGL, Vulkan, DirectX, а також PBR, HDR, VR — також є

критичним критерієм, оскільки визначає здатність системи до роботи з актуальними технологіями. Стійкість до помилок передбачає надійну роботу системи в умовах високих навантажень або при обробці некоректних вхідних даних. У випадку вбудованих рішень важливою є енергоефективність — вона оцінюється через енергоспоживання на кадр або сцену. Враховується також режим рендерингу — чи підтримується робота в реальному часі, чи система розрахована лише на попередній рендеринг. Нарешті, інтерфейсна взаємодія й інтерактивність визначають, наскільки оперативно система реагує на дії користувача чи зміни сцени, що є критичним для застосувань у віртуальній реальності, ігровій індустрії та науковій візуалізації.

Оптимальний розподіл програмних і апаратних функцій у системах рендерингу базується на принципі делегування обчислювально складних, паралельних завдань апаратному забезпеченню, тоді як програмна частина виконує високорівневе керування, логіку та адаптацію до змін. Апаратна реалізація охоплює ті функції, які вимагають великої обчислювальної потужності та паралельної обробки. До них належать: растеризація, обчислення шейдерів (вершинних, фрагментних), фізично коректне освітлення, трасування променів у реальному часі, згладжування, обробка текстур, міпмапінг, постобробка (глибина, тіні, ефекти), а також паралельне виконання матричних операцій. Також до апаратних функцій відносяться стиснення і декомпресія графічних даних, вивід зображення на дисплей через спеціалізовані контролери.

Програмна частина реалізує гнучку логіку керування, включаючи створення ієрархії сцени, завантаження та керування ресурсами (моделями, шейдерами, текстурами), обробку анімацій, фізику руху, інтерфейс користувача, події введення, менеджмент пам’яті, адаптацію рівнів деталізації, а також забезпечення взаємодії з графічними API (OpenGL, Vulkan, DirectX). Програмне забезпечення також відповідає за багатоплатформену сумісність, налагодження, збирання статистики та оптимізацію рендеру.

У деяких випадках застосовуються гібридні рішення, де частина функцій виконується апаратно, а частина — програмно. Наприклад, шейдери

програмуються на CPU, але виконуються на GPU; динамічне освітлення може виконуватись на GPU, а попереднє обчислення освітлення — програмно; нейронні мережі можуть бути частково реалізовані апаратно (через Tensor Cores) і частково програмно. Таким чином, всі функції, які вимагають низької затримки та високої продуктивності, доцільно реалізовувати апаратно, тоді як адаптивні та логічні завдання — програмно. Такий підхід забезпечує високу швидкодію, гнучкість і масштабованість сучасних систем рендерингу.

Доповнюючи викладене, важливо зазначити, що ефективність програмно-апаратної реалізації рендерингу також залежить від правильного проектування архітектури системи. Однією з ключових стратегій є попереднє розділення сценового рендерингу на етапи, що можуть бути обчислені незалежно та паралельно, що дозволяє максимально використати переваги апаратного прискорення. Наприклад, тесселяція та обробка вершин і фрагментів реалізуються через програмовані шейдерні блоки на GPU, що дає змогу досягати високого ступеня паралельності.

Важливу роль відіграє правильна організація буферів обміну між CPU та GPU. Передавання великих обсягів даних через шину пам’яті повинно бути мінімізовано шляхом кешування, використання динамічних буферів або обчислень безпосередньо на GPU. Для цього розробляються спеціальні алгоритми попередньої підготовки даних: наприклад, сортування світильників за важливістю або групування об’єктів сцени у зони видимості (застосування Z-катів, оклюзії).

У сучасних системах все частіше використовуються спеціалізовані апаратні модулі, зокрема для трасування променів [6, 11] що дозволяє значно покращити реалізм освітлення, відбиттів і тіней при мінімальному навантаженні на CPU. У той же час, розрахунок складних логічних залежностей між об’єктами сцени, управління подіями, сценаріями користувача, а також інтеграція з іншими системами (наприклад, з фізичними рушіями чи мережевими протоколами) залишається в межах програмної частини, реалізованої через універсальні мови програмування (C++, Python тощо).

Для вбудованих або енергообмежених систем (наприклад, AR/VR гарнітур

або мобільних пристроїв) особливе значення має перенесення обчислювальних навантажень з CPU на низькоспоживальні графічні модулі або FPGA, які можуть виконувати жорстко задані графічні завдання з мінімальною затримкою. У таких випадках навіть частина програмних рішень (наприклад, попереднє декодування текстур або обробку аудіосигналів) переноситься в апаратну логіку для економії енергії.

Загалом, грамотний розподіл програмних і апаратних функцій не тільки підвищує швидкодію рендерера, але й дозволяє досягти балансу між гнучкістю, масштабованістю, якістю візуалізації та енергоефективністю, що особливо важливо в реальному часі та в інтерактивних середовищах.

Крім зазначених аспектів, варто також враховувати вплив обраної графічної платформи й архітектури на розподіл функцій. Наприклад, у консолях нового покоління (PlayStation 5, Xbox Series X) архітектура системи спроектована таким чином, щоб більшість рендерингових обчислень виконувались безпосередньо на GPU, тоді як CPU зосереджується на загальній логіці гри, штучному інтелекті та сценарній взаємодії. Завдяки використанню спільної пам’яті (Unified Memory Architecture) затрати на переміщення даних між процесорами мінімізуються, що ще більше зміщує акцент на апаратну обробку.

У випадку персональних комп’ютерів і робочих станцій, де використовується дискретна відеокарта, важливу роль відіграє оптимізація драйверів і середовища виконання — саме вони визначають, які частини обчислень будуть ефективніше реалізовані на GPU. Наприклад, у середовищах на базі Vulkan чи DirectX 12 розробник має більший контроль над низькорівневою архітектурою, і може вручну керувати завантаженням графічного конвеєра.

Окремий напрям — це програмно-апаратні рендерери у хмарних середовищах, де основне навантаження виконується на графічних серверах, а кінцевому користувачу передається вже готове відео (так званий cloud rendering). У таких системах критично важливо програмне планування ресурсів, балансування навантаження між GPU та CPU, оптимізація черг задач та асинхронна обробка. Це дозволяє одночасно обслуговувати багато користувачів із різними вимогами до

якості та продуктивності.

Використання FPGA або спеціалізованих прискорювачів (наприклад, Intel Xeon Phi), які дозволяють реалізовувати власні рендеринг-алгоритми апаратно з високою точністю та мінімальною затримкою. Наприклад, у медичній томографії або візуалізації CFD-симуляцій може використовуватись апаратна фільтрація, реконструкція зображень або розгортка тривимірних об'єктів у реальному часі.

Загалом, тренди розвитку програмно-апаратної реалізації рендерингу свідчать про поступовий перехід до гібридних і адаптивних архітектур, де межа між програмною логікою та апаратним прискоренням є динамічною та налаштовується залежно від умов виконання, навантаження, типу сцени та платформи. Це забезпечує гнучкість у проєктуванні систем, підвищення якості візуалізації й ефективне використання доступних обчислювальних ресурсів.

2. ПРИСКОРЕНЕ ВИЗНАЧЕННЯ НОРМАЛІЗОВАНИХ ВЕКТОРІВ

2.1 Застосування методу бінарного поділу для визначення одиничних векторів

Інтерполяція (рис. 2.1) нормалізованих векторів між вектором $\vec{N}_a$ і вектором $\vec{N}_b$ рядка растеризації виконують згідно виразу [11] (2.1)

$$\vec{N}(w) = \vec{N}_a \frac{\sin((1-w)\psi)}{\sin\psi} + \vec{N}_b \frac{\sin(w\psi)}{\sin\psi}, \qquad (2.1)$$

де $w \in [0,1]$ - параметрична змінна,

ψ - кут між вектором $\vec{N}_a$ в початковій і вектором $\vec{N}_b$ в кінцевих точках рядка пастеризації (рис. 2.1).

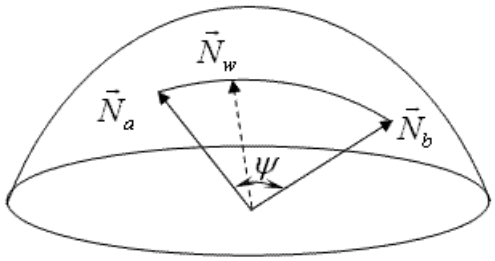


Рис. 2.1. Вектори $\vec{N}_a, \vec{N}_b$ і потоковий $\vec{N}_w$ одиничний вектор

Розрахуємо вектор $\vec{N}_{(1/2)}$, який має кут $\psi/2$ між векторами $\vec{N}_a$ і $\vec{N}_b$

$$\vec{N}_{(1/2)} = \vec{N}_a \frac{\sin(\frac{1}{2}\cdot\psi)}{\sin\psi} + \vec{N}_b \frac{\sin(\frac{1}{2}\cdot\psi)}{\sin\psi} = \frac{\vec{N}_a + \vec{N}_b}{2\cos\frac{\psi}{2}} = \frac{\vec{N}_a + \vec{N}_b}{\sqrt{2\cdot(1+\cos\psi)}}.$$

Визначемо одиничний вектор $\vec{N}_{(1/2^2)}$, який утворює однакові кути між векторами $\vec{N}_a, \vec{N}_{1/2}$.

$$\vec{N}_{(1/2^2)} = \frac{\vec{N}_a + \vec{N}_{1/2}}{\sqrt{2(1+\cos\frac{\psi}{2})}} = \frac{\vec{N}_a + \vec{N}_{1/2}}{\sqrt{2(1+\sqrt{\frac{1+\cos\psi}{2}})}} = \frac{\vec{N}_a + \vec{N}_{1/2}}{\sqrt{2+\sqrt{2(1+\cos\psi)}}}.$$

Нехай $z_{2^n} = \sqrt{2(1+\cos\frac{\psi}{2^{n-1}})}$. Тоді

$$\vec{N}_{(1/2^2)} = \frac{\vec{N}_a + \vec{N}_b}{\sqrt{2 \cdot (1 + \cos\psi)}} = \frac{\vec{N}_a + \vec{N}_{1/2}}{\sqrt{2 + z_1}}.$$

Визначимо

$$z_{2^{n+1}} = \sqrt{2(1 + \cos\frac{\psi}{2^{n-2}})} = \sqrt{2(1 + \sqrt{\frac{1 + \cos\frac{\psi}{2^{n-1}}}{2}})} = \sqrt{2 + 2\sqrt{1 + \cos\frac{\psi}{2^{n-1}}}} = \sqrt{2 + z_{2^n}}.$$

Тоді

$$\vec{N}_{(1/2^{n+1})} = \frac{\vec{N}_a + \vec{N}_{1/2^n}}{\sqrt{2(1 + \cos\frac{\psi}{2^{n+1}})}} = \frac{\vec{N}_a + \vec{N}_{1/2^n}}{z_{2^{n+1}}} = \frac{\vec{N}_a + \vec{N}_{1/2^n}}{\sqrt{2 + z_{2^n}}}. \qquad (2.2)$$

У процесі кожної ітерації обчислення знаменника зводиться до виконання однієї операції додавання та одного добування квадратного кореня.

В результаті першої ітерації відбувається зміна кута між нормалями векторів від початкового значення нуль до 90^0, тому $\sqrt{2} \le z_{2^n} \le 2$. Значення $\frac{1}{\sqrt{2 + z_{2^n}}}$ задамо рядом Чебішева . У разі застосування полінома першого порядку

$$\frac{1}{\sqrt{2 + z_{2^n}}} \approx -0{,}07 \cdot z_{2^n} + 0{,}64.$$

У такому випадку найбільше значення абсолютної похибки апроксимації становить не більше ніж 0,0005, тоді як відносна похибка не перевищує 0,12 %. Для випадку другої степені

$$\frac{1}{\sqrt{2 + z_{2^n}}} \approx 0{,}014 \cdot z^2_{2^n} - 0{,}119 \cdot z_{2^n} + 0{,}681.$$

максимальна абсолютна похибка апроксимації не перевищує, а відносна 0,004 %. Першу формулу доцільно використовувати для екранів із невеликою роздільною здатністю, для яких трикутники, які складають поверхню тривимірного об’єкту, мають незначні розміри.

Найбільша абсолютна похибка апроксимації не перевищує $2 \cdot 10^{-5}$, а відносна

становить лише 0,004 %. Такий вираз доцільно застосовувати для дисплеїв із низькою роздільною здатністю, де розміри трикутників, що формують поверхню 3D-об'єкта, є досить малими.

Проведений аналіз продемонстрував, що застосування останньої апроксимаційної формули дозволяє скоротити час обчислення вектора у 2,4 раза порівняно з традиційним методом реалізації.

Розглянемо знаходження векторів нормалей за умови, що відомо кут ψ між векторами нормалей у початковій і кінцевій точках рядка трикутника.

Якщо виконати n етапів кут між сусідніми векторами буде дорівнювати $\psi / 2^n$, тому, якщо вважати вектор $\vec{N}_a$ початковим, то

Якщо здійснити n етапів, то кут між суміжними одиничними векторами становитиме $\psi / 2^n$. Тому

$$\vec{N}_{(1/2^n)} = \frac{\vec{N}_A + \vec{N}_{1/2^{n-1}}}{2\cos\frac{\psi}{2^n}}.$$

Упродовж першого кроку ітераційного процесу

$$\vec{N}_{(1/2)} = \frac{\vec{N}_A + \vec{N}_B}{2\cos\frac{\psi}{2}}.$$

Визначемо одиничні вектори $\vec{N}_{(1/4)}$ і $\vec{N}_{(1/8)}$:

$$\vec{N}_{(1/4)} = \frac{\vec{N}_A + \vec{N}_{(1/2)}}{2\cos\frac{\psi}{4}} = \frac{\vec{N}_A + \frac{(\vec{N}_A + \vec{N}_B)}{2\cos\frac{\psi}{2}}}{2\cos\frac{\psi}{4}} = \frac{\vec{N}_A \cdot (1 + 2\cos\frac{\psi}{2}) + \vec{N}_B}{4\cos\frac{\psi}{2}\cos\frac{\psi}{4}}.$$

$$\vec{N}_{(1/8)} = \frac{\vec{N}_A + \vec{N}_{(1/4)}}{2\cos\frac{\psi}{8}} = \frac{\vec{N}_A + \frac{\vec{N}_A + \vec{N}_{(1/2)}}{2\cos\frac{\psi}{4}}}{2\cos\frac{\psi}{8}}.$$

Враховуючи аналітичний вираз для визначення $\vec{N}_{(1/2)}$, одержуємо

$$\vec{N}_{(1/8)} = \frac{\vec{N}_A \cdot (1 + 2\cos\frac{\psi}{2} + 4\cos\frac{\psi}{2}\cos\frac{\psi}{4}) + N_B}{8\cos\frac{\psi}{2}\cos\frac{\psi}{4}\cos\frac{\psi}{8}}.$$

Знайдемо $\vec{N}_{(1/4)} = \frac{a\vec{N}_A + \vec{N}_B}{b}$, де $a = 1 + 2\cos\frac{\psi}{2}$ і $b = 4\cos\frac{\psi}{2}\cos\frac{\psi}{4}$, то

$$\vec{N}_{(1/8)} = \frac{(a+b)\cdot\vec{N}_A + N_B}{2\cdot b\cdot\cos\frac{\psi}{8}}.$$

Як видно з останньої формули, на кожній ітерації знаменник збільшується в результаті множення на $2\cos\frac{\psi}{2^n}$, а для знаходження чисельника використовуються операнди, необхідні для реконструкції попереднього векторного значення.

Знайдемо $\vec{N}_{(1/2^n)}$

$$\vec{N}_{(1/2^n)} = \frac{a\vec{N}_A + \vec{N}_B}{b}.$$

Тоді

$$\vec{N}_{(1/2^{n+1})} = \frac{\vec{N}_A + \vec{N}_{(1/2^n)}}{2\cdot\cos\frac{\psi}{2^{n+1}}} = \frac{\vec{N}_A + \frac{a\cdot\vec{N}_A + \vec{N}_B}{b}}{2\cdot\cos\frac{\psi}{2^{n+1}}} \frac{(a+b)\cdot\vec{N}_A + \vec{N}_B}{2\cdot b\cdot\cos\frac{\psi}{2^{n+1}}}.$$

Загалом, на основі отриманих формул, маємо наступний запис

$$\vec{N}_{(1/2^{n+1})} = \frac{\vec{N}_A \cdot L_{(1/2^n)} + \vec{N}_B}{M_{(1/2^n)}},$$

де $M_{(1/2^n)} = 2\cdot M_{(1/2^{n-1})}\cdot\cos\frac{\psi}{2^n}$, $L_{(1/2^n)} = (L_{(1/2^{n-1})} + M_{(1/2^{n-1})})$,

$a = L_{(1/2^n)}$, $b = M_{(1/2^n)}$, $M_{(1)} = 1$, $L_{(1)} = 0$.

З алгоритмічної позиції доцільно реалізовувати бінарне розділення кута в заданому напрямі, наприклад, зліва направо, з подальшим визначенням одиничних векторів у точках, симетричних відносно потокової. Запропонований підхід до

нормалізації, який ґрунтується на бінарному поділі кута між нормалями, виявляється ефективним при малій довжині рядків у процесі растеризації трикутників (наприклад, для $m \le 16$), оскільки в таких умовах практично не виникає суттєвого розгалуження обчислень. Отримані результати було використано як основу для подальшої розробки альтернативних методів нормалізації векторів.

2.2 Застосування поліноміальної інтерполяції другого ступеня для нормалізації векторів.

Розглянемо використання квадратичної інтерполяції [8] для знаходження нормалізованих векторів нормалей [6, 17] за умови, що відомо одиничні вектори нормалей у початковій та кінцевій точках і-того рядка растерізації трикутника. Проміжні значення векторів нормалей у рядку растеризації трикутника знайдемо за формулою

Розглянемо застосування квадратичної інтерполяції для обчислення одиничних векторів. Будемо важати, що задано нормалізовані вектори в кінцевій точках *i*-го РР трикутника. Вектори нормалей для проміжних точок цього рядка можуть бути визначені за такою формулою:

$$\vec{N}_{i,t} = \vec{G}_i \cdot t^2 + \vec{P}_i \cdot t + \vec{Q}_i . \qquad (2.3)$$

Введемо такі позначення- $\vec{N}_{i,l}$, $\vec{N}_{i,p}$, $\vec{N}_{i,c}$ - відповідно одиничні вектори у лівій, правій та середній точках РР .

За умови, що $t = 0$ $\vec{N}_{i,l} = \vec{Q}_i$. Якщо $t = 1$, то $\vec{N}_{i,p} = \vec{G}_i + \vec{P}_i + \vec{Q}_i$. Посередині РР $t = 1/2$, тому $N_{i,c} = \frac{\vec{G}_i}{4} + \frac{\vec{P}_i}{2} + \vec{Q}_i$. Знайдемо невідомі члени такої системи координат

$$\begin{cases} \vec{N}_{i,l} = \vec{Q}_i, \\ \vec{N}_{i,p} = \vec{G}_i + \vec{P}_i + \vec{Q}_i, \\ N_{i,c} = \frac{\vec{G}_i}{4} + \frac{\vec{P}_i}{2} + \vec{Q}_i, \end{cases}$$

$$\vec{G}_i = 2\cdot\vec{N}_{i,p} - 4\cdot\vec{N}_{i,c} + 2\cdot\vec{N}_{i,l}, \quad \vec{P}_i = 4\cdot\vec{N}_{i,c} - \vec{N}_{i,p} - 3\cdot\vec{N}_{i,l}, \quad \vec{Q}_i = \vec{N}_{i,l}.$$

Нехай $1/\sqrt{2(1+\vec{N}_{i,l}\cdot\vec{N}_{i,p})}$ дорівнює h_i, тоді:

$$\vec{N}_{i,c} = h_i(\vec{N}_{i,l} + \vec{N}_{i,p}),$$

$$\vec{G}_i = 2(1-2\cdot h)\cdot(\vec{N}_{i,l} + \vec{N}_{i,p}), \quad \vec{P}_i = (4h-3)\cdot(\vec{N}_{i,l} + \vec{N}_{i,p}) + 2\cdot\vec{N}_{i,p}, \quad \vec{Q}_i = \vec{N}_{i,l}.$$

Виходячи з наведених математичних співвідношень, робимо висновок, що $\vec{P}_i = (\vec{N}_{i,p} - \vec{N}_{i,l}) - \vec{G}_i$.

Використаємо ряд Чебішева.

Позначимо величину $\vec{N}_{i,l}\cdot\vec{N}_{i,p} = cos\psi_i$ як r, де ψ_i кут між векторами $\vec{N}_{i,l}$ та $\vec{N}_{i,p}$. Вираз можна наближено подати у вигляді полінома Чебишева другого ступеня. При цьому максимальна абсолютна похибка не перевищує $0{,}002$. На рисунку 2.3 представлено графік відносної похибки апроксимації, яка залишається в межах від 0 до 0,36 %. Застосування даної апроксимації дає змогу скоротити час обчислення нормалі в центральній точці РР більш ніж у $2{,}5$ раза.

Якщо для значення $1/\sqrt{2(1+r)}$ використати апроксимацію поліному третьої степені, то

$1/\sqrt{2(1+r)} \approx -0{,}059r^3 + 0{,}193r^2 - 0{,}341r + 0{,}707$. У цьому випадку найбільша абсолютна похибка не перевищує $0{,}00038$, тоді як відносна становить лише $0{,}054$ %.

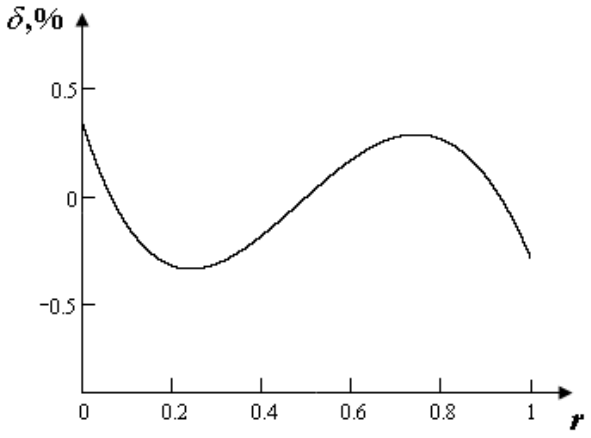


Рис. 2.3. Графік залежності відносної похибка визначення $\frac{1}{\sqrt{2(1+r)}}$ поліномом Чебишева другого порядку

У бакалаврській кваліфікаційній роботі запропоновано інтерактивний алгоритм для розрахунку векторів нормалей у РР

За умови, що $t = t + \Delta t,$

$$\vec{N}_{i,t+\Delta t} = \vec{G}_i \cdot (t + \Delta t)^2 + \vec{P}_i \cdot (t + \Delta t) + \vec{Q}_i = \vec{N}_{i,t} + \Delta t \cdot (\vec{G}_i (2 \cdot t + \Delta t) + \vec{P}_i).$$

Введемо таке позначення- $W_{i,t} = \Delta t \cdot (\vec{G}_i (2 \cdot t + \Delta t) + \vec{P}_i)$. Тоді для $t = t + \Delta t$:

$$W_{i,t+\Delta t} = \Delta t \cdot (\vec{G}_i (2 \cdot (t + \Delta t) + \Delta t) + \vec{P}_i) = \Delta t \cdot (G_i (2 \cdot t + \Delta) + P_i) + 2G_i \cdot \Delta t^2$$

З урахуванням формули для $W_{i,t}$ знаходимо:

$$W_{i,t+\Delta t} = W_{i,t} + 2G_i \cdot \Delta t^2.$$

Враховуючи, що $2G_i \cdot \Delta t^2$ є константою у межах одного РР трикутника, можна стверджувати, що в процесі обчислення одиничних векторів використовуються виключно мікрооперації накопичувального додавання. Аналіз показав, що при програмному виконанні час розрахунку нормалей для типового трикутника скоротився у 3,6 раза порівняно з традиційним підходом.

2.3. Апаратна нормалізації векторів нормалей

Для визначення одиничеих векторів використовують такий вираз [6, 8]

$$\vec{N} = \frac{x}{\sqrt{x^2 + y^2 + z^2}} \vec{i} + \frac{y}{\sqrt{x^2 + y^2 + z^2}} \vec{j} + \frac{z}{\sqrt{x^2 + y^2 + z^2}} \vec{k}.$$

При проектуванні засобів для нормалізації векторів необхідно враховувати, що нормовані вектори нормалей у подальшому використовуються для обчислення спекулярної й дифузної складової кольору. Важливо, щоб сформовані при цьому тривимірні зображення були візуально ідентичні відносно еталонних. Для аналізу спекулярної складової кольору можливо використання менш точних методів визначення векторів нормалей, однак при цьому важливо, щоб зменшення точності

не призвело до артефактного вилучення відблисків. Слід зазначити, що при наближеному обчисленні векторів нормалей використання складної моделі освітлення при відсутності візуально ідентифікованого відблиску не є критичною, хоча й впливає на швидкодію формування тривимірної сцени. У підрозділі 2.1 запропоновано підхід до прискореного визначення векторів нормалей, які розміщено в середній точці цифрового сегменту, на які розбито рядок растеризації. Вектор, які розраховано на потоковій ітерації, можна використати для наступної ітерації. При цьому розмір цифрового сегменту зменшується вдвічі. На кожній ітерації знаходять значення z_{2^i}, яке для векторів $\vec{N}_1$ і $\vec{N}_2$ дорівнює $2\cdot(\vec{N}_1\cdot\vec{N}_2)$. Множення на 2 замінюють монтажним зсувом у сторону старших розрядів.

На рис. 2.4 зображено структуру для визначення скалярного добутку двох нормованих векторів, тобто для знаходження $cos\psi$. Ортогональні складові векторів, які відповідають одній координатній осі, перемножуються й додаються в суматорі Sm.

Розглянемо формування векторів нормалей згідно виразу - $\vec{N}_{(1/2^{n+1})}=(\vec{N}_a+\vec{N}_{1/2^n})/\sqrt{2+z_{2^n}}$ з використанням апроксимації $1/\sqrt{2+z_{2^n}}$ багаточленом другої степені. На рис. 2.5 зображено структуру пристрою. Регістр $Rg1$, блоки множення $\otimes_1, \otimes_2$, суматори $Sm1, Sm3$

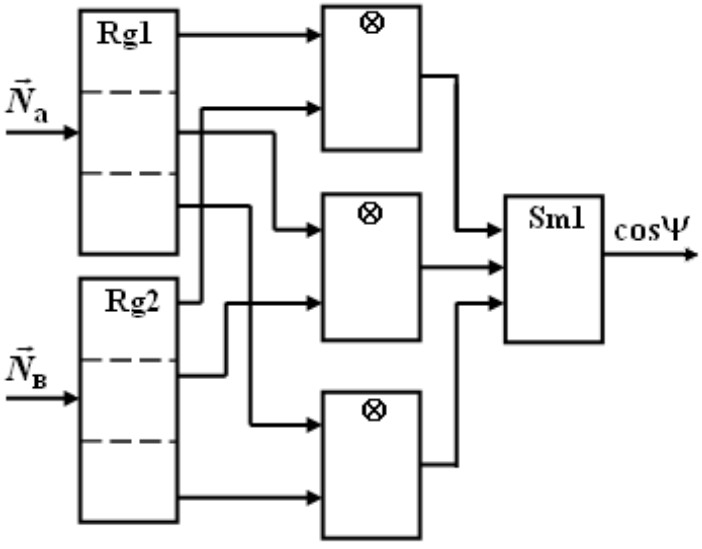


Рис. 2.5. Структурна схема блоку для визначення скалярного добутку двох векторів

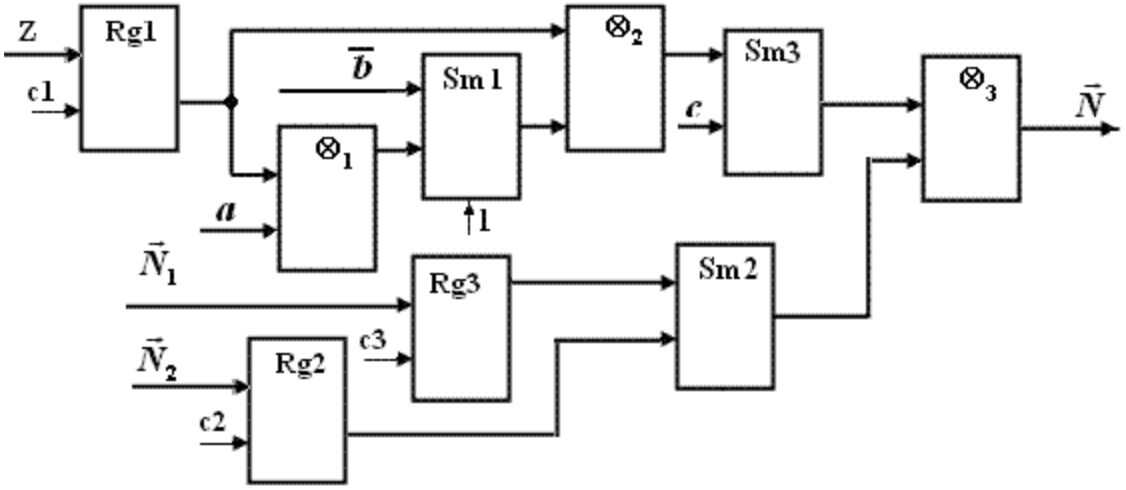
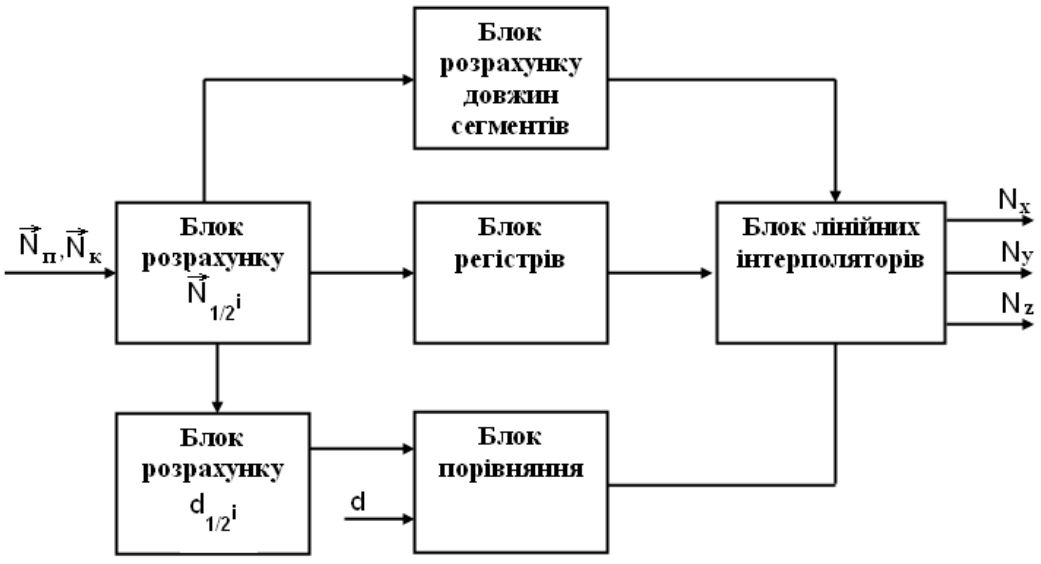


Рис. 2.6. Структурна схема блоку для визначення вектору

$$\vec{N}_{(1/2^{n+1})} = \frac{\vec{N}_a + \vec{N}_{1/2^n}}{\sqrt{2 + z_{2^n}}}$$

використовуються для обчислення $1/\sqrt{2+z_{2^n}} \approx 0{,}014 \cdot z^2_{2^n} - 0{,}119 \cdot z_{2^n} + 0{,}681$. На виході суматору $Sm2$ формується сума векторів $\vec{N}_1, \vec{N}_2$, а на виході третього блоку множення вектор, значення якого дорівнює $(\vec{N}_1 + \vec{N}_2)/\sqrt{2+z_{2^n}}$.

На рис. 2.7 зображено структуру для організації адаптивної нормалізації векторів.



2.7. Структура пристрою для адаптивної нормалізації векторів нормалей

При використанні адаптивної нормалізації векторів згідно формули $\vec{N}_{(1/2^{n+1})} = (\vec{N}_a + \vec{N}_{1/2^n})/\sqrt{2+z_{2^n}}$ легко визначити вектори нормалей у кінцевих точках цифрових сегментів. Знаменник наведеного виразу використовується блоком розрахунку похибки $d_{(1/2^i)}$, яка порівнюється з заданою. У випадку, коли $d_{(1/2^i)} \le d$, формується сигнал, який дозволяє роботу блоку лінійних інтерполяторів по формуванню векторів нормалей по їх значенням у кінцевих точках сегменту. Блок розрахунку довжин сегментів визначає їх потокові значення шляхом поділу попереднього значення пополам. У найпростішому випадку для цього можна використати зсувний регістр. Оскільки $d_{(1/2^i)}/d_{(1/2^{i+1})} \approx 4$, то похибку d можна розрахувати для першої ітерації і у подальшому використовувати мікрооперацію зсуву. Блоки розрахунку $\vec{N}_{1/2^i}$ і $d_{1/2^i}$ можуть бути реалізовані і програмним шляхом (наприклад, шейдером).

При використанні квадратичної інтерполяції для знаходження нормалізованих векторів нормалей суттєво зменшується час розрахунку. При цьому для кожного рядка растеризації необхідно знайти нормалізований вектор нормалі в середній точці рядка растеризації трикутника. Проміжні значення векторів нормалей у рядку растеризації трикутника знаходять за формулою див. розділ 2.2

$$\vec{N}_{i,t} = \vec{G}_i \cdot t^2 + \vec{P}_i \cdot t + \vec{Q}_i, \text{ де}$$

$$\vec{G}_i = 2(1-2\cdot h)\cdot(\vec{N}_{i,l} + \vec{N}_{i,p}), \quad \vec{P}_i = (4h-3)\cdot(\vec{N}_{i,l} + \vec{N}_{i,p}) + 2\cdot\vec{N}_{i,p}, \quad \vec{Q}_i = \vec{N}_{i,l}.$$

Структурну схему блоку для визначення векторів $\vec{G}_i$, $\vec{P}_i$, $\vec{Q}_i$ зображено на рис. 2.8. Він достатньо простий і включає тільки суматори.

Визначення потокового вектору $\vec{N}_{i,t}$ здійснюється блоком, структурну схему якого подано на рис. 2.9, а.

Перед початком перетворення регістр Rg обнуляється. Шляхом послідовного додавання до вмісту нагромаджувального суматору значення Δt

визначається значення змінної t. Добуток $t \cdot t = t^2$ формується на виході другого блоку множення. На виходах блоків множення $\otimes_1, \otimes_3$ знаходять відповідно $\vec{P}_i \cdot t$ і $\vec{G}_i \cdot t^2$. Обчислення завершують після формування передостаннього вектору в рядку сканування трикутника.

На рис. 2.9, б зображено структурну схему для реалізації виразу 2.17 у якій задіяно всього один блок множення і суматор. У пристрої реалізовано конвеєрний принцип обчислення.

У початковий момент часу в регістр заноситься значення $\vec{P}_i$, а на другий вхід мультиплексора Мх$_1$ подається вектор $\vec{G}_i$.

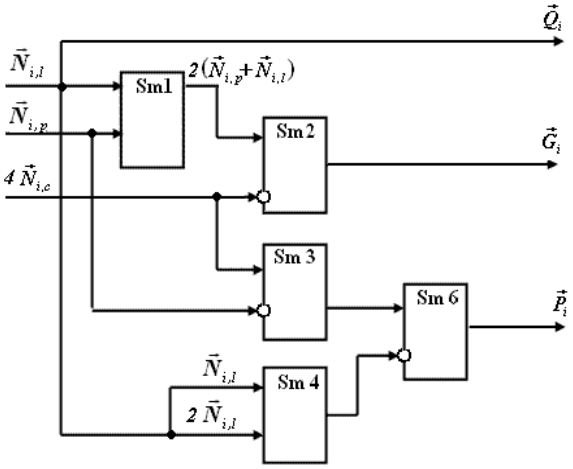


Рис. 2.8. Структурна схема блоку визначення векторів $\vec{G}_i$, $\vec{P}_i$, $\vec{Q}_i$

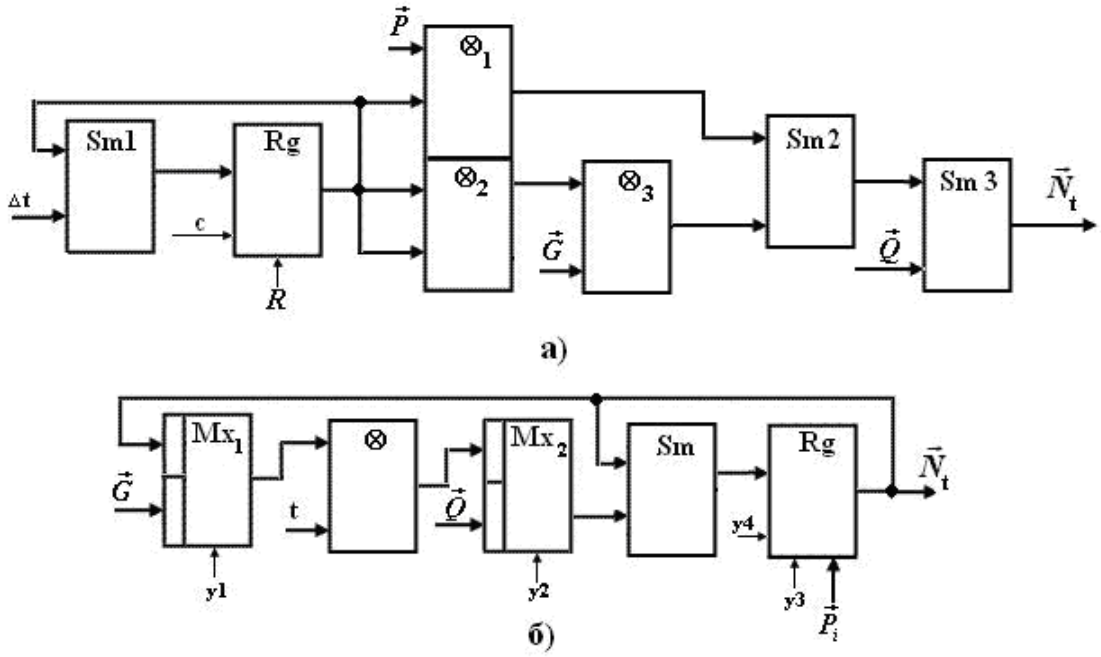


Рис. 2.9. Структурні схема блоків визначення векторів нормалей з використанням квадратичної інтерполяції

На виході блока множення знаходять добуток $\vec{G}_i \cdot t$, який додається до вмісту нагромаджувального сматору, на виході якого формується значення $\vec{G} \cdot t + \vec{P}$. Цей операнд через мультиплексор Мх$_1$ подається на вхід блока множення, на виході якого формується добуток $(\vec{G} \cdot t + \vec{P}) \cdot t$, який заноситься в регістр. У наступний проміжок часу на вхід суматору через мультиплексор Мх$_2$ поступає значення $\vec{Q}$, яке додається до $(\vec{G} \cdot t + \vec{P}) \cdot t$, що й утворює вектор $\vec{N}_t$. Блок формування змінної t не відрізняється від його реалізації на рис. 2.9, а.

З метою спрощення апаратної нормалізації векторів нормалей у розділі 2..2 використано метод кінцевих різниць.

На рис. 2.10 зображено структурну схему блоку розрахунку обчислення нормалізованих векторів нормалей. Два нагромаджувальні суматори ($Rg1$, $Sm1$), ($Rg2$, $Sm2$) використовуються для знаходження відповідно $W_{i,t+\Delta t}$ і $\vec{N}_{i,t+\Delta t}$. Інші блоки задіяно для розрахунку $W_{i,0}$ і $2G_i \cdot \Delta t^2$. На рис. 2.11. зображено один із можливих реалізацій блоку для розрахунку цих операндів, який має більш просту реалізацію за рахунок послідовного виконання дій. Блок включає регістр RG,

суматор Sm, блок множення та блок ключів BK. Для знаходження W_0 у регістр заноситься компонента вектора $\vec{G}$, а блок ключів блокується, тобто на його виході формується нулевий операнд. На виході блока множення формується добуток $G_i \cdot \Delta t$, який запам'ятовується в регістрі. Через блок ключів на вхід суматору подається $\vec{P}_i$, що забезпечує формування суми $\vec{G}_i \cdot \Delta t + \vec{P}_i$, яка заноситься в регістр і в наступному такті множиться на Δt, тобто на виході суматору отримуємо значення $W_0 = (\vec{G} \cdot \Delta t + \vec{P}) \cdot \Delta t$.

Для знаходження $G_i \cdot \Delta t^2$ блок ключів блокується, а тому добуток $G_i \cdot \Delta t$ після першого такту множиться на Δt. Шляхом монтажного зсуву в сторону старших розрядів отримуємо значення $2 \cdot G_i \cdot \Delta t^2$

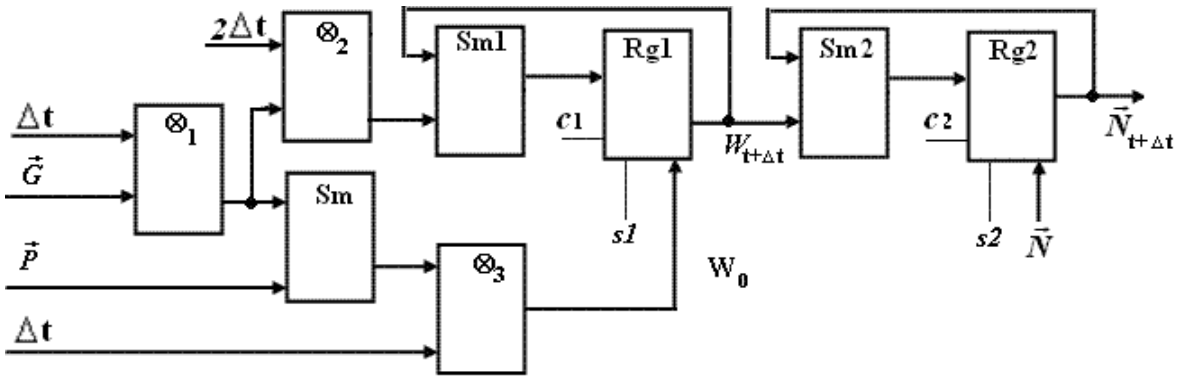


Рис. 2.10. Структурна схема блоку визначення векторів нормалей

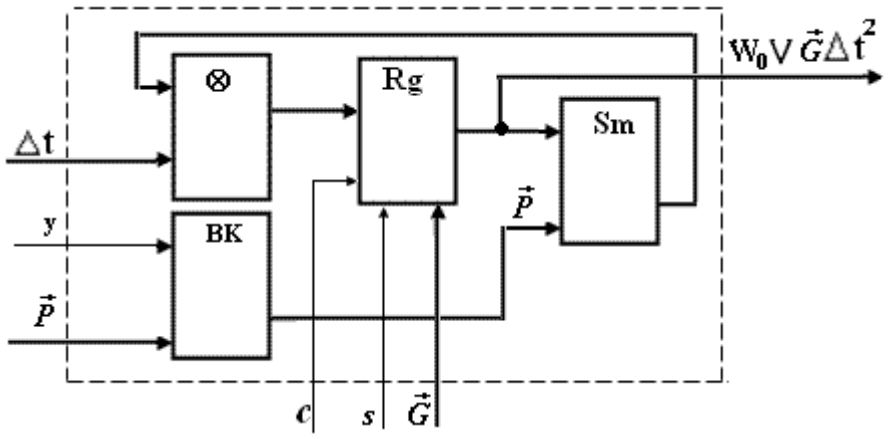


Рис. 2.11 Структурна схема блоку визначення W_0 і $\vec{G} \cdot \Delta t^2$

3. ПРОГРАМНО-АПАРАТНА РЕАЛІЗАЦІЯ МЕТОДІВ ЗАФАРБОВУВАННЯ

3.1. Апаратно-орієнтований метод рендерингу на основі нової моделі відбивної здатності поверхні

Ряд методів зафарбовування оперують не зі значенням $\cos^n \gamma$, а з кутом γ. У зв’язку із цим актуальним є питання розробки моделі ДФВЗ, яка визначається через

кут γ. Зрозуміло, що така функція для простоти апаратної реалізації повинна мати невисоку степінь.

Розглянемо апроксимацію функції $\cos^n \gamma$ функцією $cos^k(a\gamma)$ за умови, що $k < n$, $k = const$. Вибір такої функції обумовлений таким:

— для обох функцій у якості твірної використовується функція косинуса;

— при збільшенні n зменшується розмір відблиску та досягається більша сконцентрованість інтенсивності світла в області його епіцентру, при використанні функції $\cos^k(a\gamma)$ цього можна досягти за рахунок зміни параметра a, дійсно при збільшенні a зменшується період функції $\cos^k(a\gamma)$, а, як наслідок, залежність $I_l \cdot k_s cos^k(a\gamma)$ забезпечить більш високу швидкість затухання інтенсивності світла від епіцентру відблиску;

— за теоремою Муавра [8] $cos(a\gamma) + i\,sin(a\gamma) = (cos\,\gamma + i\,sin\,\gamma)^a$.

Розклавши біном і взявши дійсні частини від обох сторін, а також замінивши парні степені $\sin\gamma$ з рівняння $(sin^2\gamma)^m = (1 - cos^2\gamma)^m$, можна зробити висновок, що $cos(a\gamma)$ є багаточленом ступеня a від $cos\,\gamma$. У зв'язку із цим, можна констатувати, що $cos^k(a\gamma)$ є багаточленом степені ak. Таким чином, змінюючи параметр a, можна корегувати степінь $cos^k(a\gamma)$.

Для визначення a можна запропонувати кілька підходів. Розглянемо перший із них.

Припустимо, що

$$cos^n\gamma = cos^k(a\gamma) \quad \text{при} \quad -\frac{\pi}{2a} \le \gamma \le \frac{\pi}{2a}. \qquad (3.1)$$

Помножимо ліву та праву частину останнього рівняння на $\sin\gamma$. Тоді

$$\int_0^{\frac{\pi}{2}} cos^n\gamma \cdot sin\gamma \;\; d\gamma = \int_0^{\frac{\pi}{2a}} cos^k(a\gamma) \cdot sin\gamma \;\; d\gamma.$$

Останнє рівняння можна використати для визначення параметра a, задавши

конкретне значення k.

Наприклад, при $k = 2$

$$\int_0^{\frac{\pi}{2}} \cos^n \gamma \cdot \sin\gamma \;\; d\gamma = \int_0^{\frac{\pi}{2a}} \cos^k a\gamma \cdot \sin\gamma \;\; d\gamma, \quad \frac{1}{n+1} = \frac{2 \cdot a^2 \cdot (1 - \cos\frac{\pi}{2 \cdot a}) - 1}{4 \cdot a^2 - 1}.$$

При $k = 4$

$$\int_0^{\frac{\pi}{2}} \cos^n \gamma \cdot \sin\gamma \;\; d\lambda = \int_0^{\frac{\pi}{4a}} \cos^k (a\gamma) \cdot \sin\gamma \;\; d\gamma,$$

$$\frac{1}{n+1} = \frac{(4 \cdot \cos(\frac{\pi}{4 \cdot a}) \cdot a^2 - 2 \cdot a \cdot \sin(\frac{\pi}{4 \cdot a}) - \cos(\frac{\pi}{4 \cdot a}) - 4 \cdot a^2 + 2)}{2 \cdot (1 - 4 \cdot a^2)}.$$

Із наведених формул для заданих n і k можна визначити значення a. Приведений підхід до визначення a приводить до громіздких формул, що обумовлює доцільність застосування такого підходу тільки для невеликих значень k.

Розглянемо другий підхід для визначення a.

Розкладемо ліву та праву частину виразу (3.1) у ряд Тейлора [8] та обмежимося першими двома членами:

$$1 - \frac{1}{2} \cdot n \cdot \gamma^2 = 1 - \frac{1}{2} \cdot k \cdot a^2 \cdot \gamma^{\;2}.$$

Звідси, $a = \sqrt{n/k}$, а сама ДФВЗ має такий вигляд $\cos^k\left(\sqrt{\frac{n}{k}} \cdot \gamma\right)$.

3.2. Пристрій для визначення спекулярної складової кольору

У підрозділі 3.1 розроблено метод зафарбовування, особливість якого полягає у вилученні з обчислювального процесу трудомісткої процедури нормалізації векторів нормалей за рахунок використання кутової інтерполяції. На рис. 3.1 зображено основні етапи розрахунку спекулярної (дифузної) складових інтенсивностей кольору на рівні блоків, які можуть бути реалізовані програмним

або апаратним шляхом.

По векторам нормалей $\vec{N}_A$ і $\vec{N}_B$ розраховують кут між ними, а потім $\Delta\phi$. Це передбачає виконання операцій арккосинуса та ділення. По значенням векторів нормалей $\vec{N}_A, \vec{N}_B, \vec{H}$ визначається $\cos\beta$. Перераховані дії доцільно виконувати програмним шляхом, оскільки вони достатньо трудомісткі. Подальші дії легко реалізувати апаратним шляхом. Для кожної точки рядка растеризації трикутника визначається косинус кута ν, який є вхідним параметром для розрахунку ДФВЗ. По значення дистрибутивної функції визначають дифузну та спекулярну складові кольору.

На рис. 3.2 зображено структуру блоку для апаратного обчислення $\cos\nu$ за формулою (3.23), який включає нагромаджувальний суматор PSm, регістри $RG1, RG2$, блок постійної пам'яті $Prom$ для зберігання значень $\cos x$, блок множення.

У початковий момент часу в регістри $RG1, RG2$ заносяться відповідно значення $\Delta\phi$, $\cos\beta$, а в нагромаджувальний суматор - ψ. В останньому в кожний тактовий момент часу розраховується значення аргументу $\psi - t\cdot\Delta\phi$, яке поступає на вхід блоку постійної пам'яті для знаходження $\cos(\psi - t\cdot\Delta\phi)$.

Нагромаджувальний суматор працює в режимі віднімання, для чого на його вхід переносу p_v подають рівень логічної одиниці. На виході блока множення знаходять $\cos\nu = \cos\beta\cdot\cos(\psi - t\cdot\Delta\phi)$. Обчислення завершують за умови, що $\psi - t\cdot\Delta\phi < 0$, про що можна судити по знаку вихідного переносу p_{vv}.

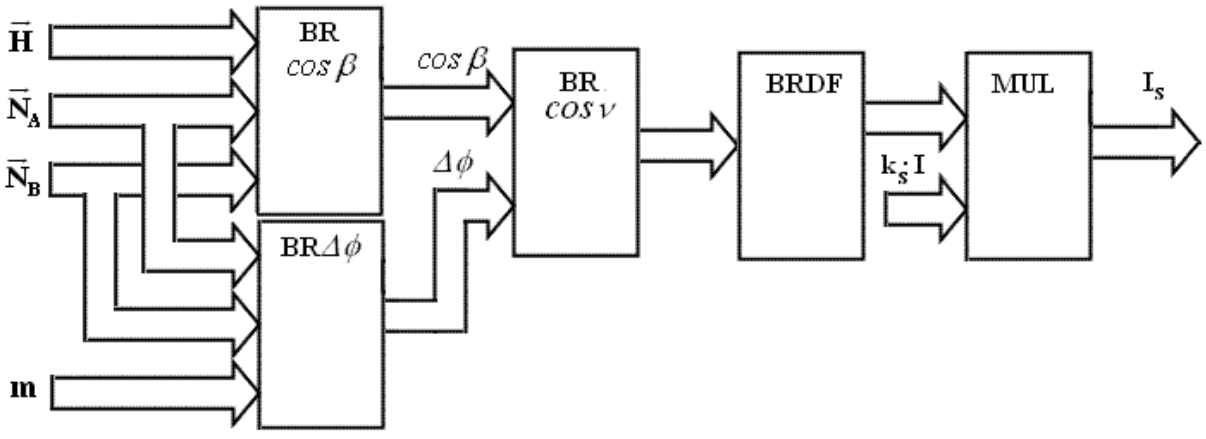


Рис. 3.1. Структура пристрою визначення спекулярної складової кольору

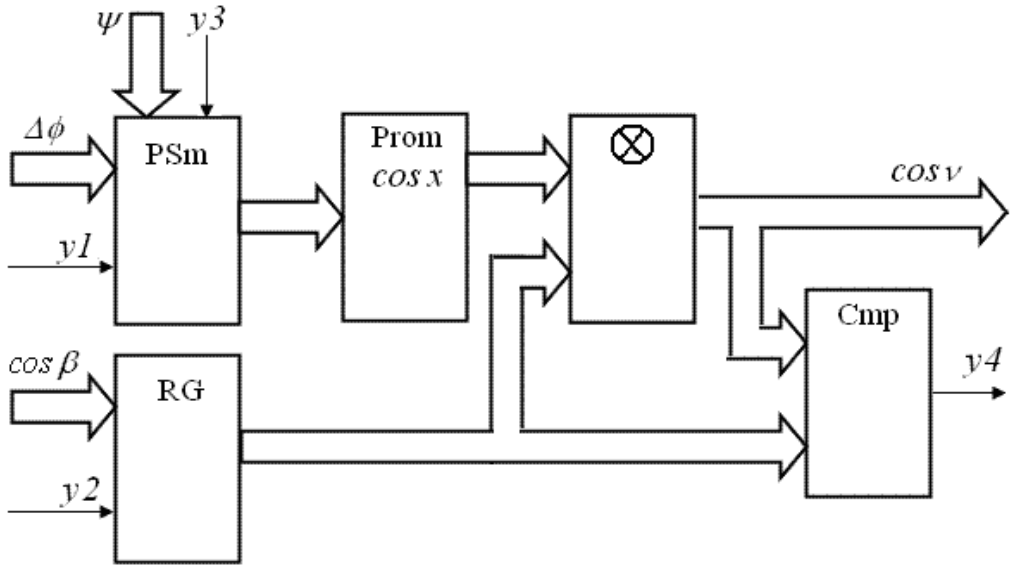


Рис. 3.2 Структурна схема блоку визначення $cos\nu$

У подальшому нагромаджувальний суматор установлюють у стан $\psi := \phi - \psi$ і повторюють дії для правої частини рядка растеризації.

У запропонованому методі необхідно по значенню $\vec{N}_A \cdot \vec{N}_B = cos\,\phi$ визначати $\phi = arccos\ (\vec{N}_A \cdot \vec{N}_B)$. Розрахунок арккосинуса на програмному рівні вимагає значних затрат часу, тому для доцільно його знаходити апаратно. Було проведено розклад функції $arccos(x)$ по багатого членам Чебішева.

Порівняно з розкладом у ряд Тейлора в цьому випадку має місце суттєвий виграш у швидкості сходження (розклад функції $arccos(x)$ у ряд Тейлора, який включає три члени (багатогочлен третьої степені), повільно сходиться біля границі ділянки $|x| \le 1$ і вже на проміжку $0 \le x < 0{,}8$ має відносну похибку апроксимації, більшу за 6 %.).

Аналіз показав, що для апаратної реалізації найбільш доцільно використати багатогочлен другої степні. Це пояснюється тим, що при використанні кусково-

лінійної апроксимації має місце відносно велика похибка, для зменшення якої використовують велику кількість інтервалів апроксимації. Це призводить до необхідності зберігання великої кількість коефіцієнтів розкладу. При використанні поліному третьої степені збільшується кількість блоків множення або кількість ітерацій за умови послідовної реалізації.

У таблиці 3.1 наведено значення коефіцієнтів A, B, C розкладу функції $arccos(x)$ по багатогочленам Чебішева за умови, що відносна похибка не більша 0,2%. Багатогочлен має такий вигляд $arccos(x) \approx Ax^2 + Bx + C$.

На рис. 3.3 наведено структурну схему пристрою для розрахунку арккосинуса кута за умови використання квадратного багаточлена.

Пристрій включає схему порівняння Cmp, мультиплексор MS, блок множення, регістр RG, нагромаджувальний суматор PSm, лічильник $CT2$. На вході X_0 монтажним шляхом фіксуються коефіцієнти A, B, C, а на вхід X подається значення косинуса кута. У початковий момент часу лічильник обнуляють а в нагромаджувальний суматор через мультиплексом заносять значення C. Лічильник інкрементують, що забезпечу передачу на вихід мультиплексора MS значення коефіцієнта A, який перемножується зі значенням X.

Таблиця 3.1 — Значення коефіцієнтів розкладу

x	A	B	C
0─0,5	1,57	-0,972	-0,144
0,5─0,75	1,439	-0,447	-0,677
0,75─0,9	0,454	2,154	-2,397
0,9─0,96	-5,529	15,47	-9,806
0,96─1	-33,838	74,547	-40,535

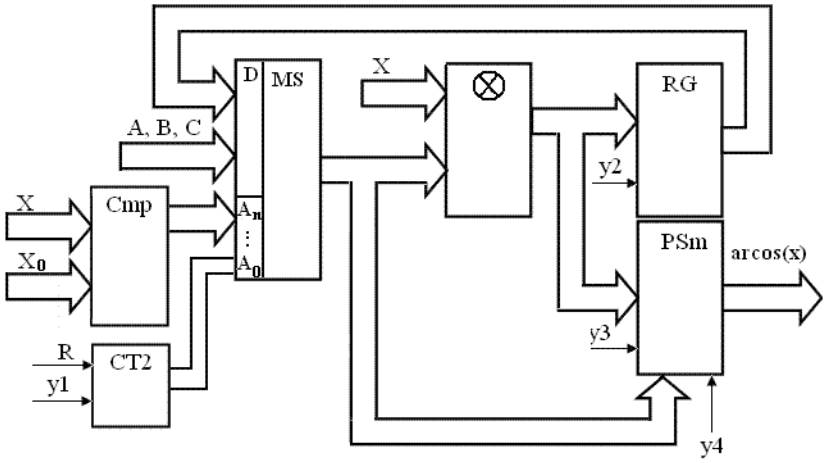


Рис. 3.3. Структурна схема пристрою для визначення $arccos(x)$

Отриманий добуток записують у регістр. Після цього стан лічильника збільшують на одиницю, що забезпечить передачу через мультиплексор на вхід блоку множення значення $A \cdot x$ з виходу регістра. На виході блока множення отримують значення $A \cdot x^2$, яке подають на вхід нагромаджувального суматору. Після виконання мікрооперацій додавання та запису в нагромаджувальному суматорі зберігається значення $Ax^2 + C$. Уміст лічильника збільшують на одиницю. Це призведе передачу через мультиплексор MS значення B, яке перемножується зі значенням x. Отриманий операнд $A \cdot x$ додається до вмісту нагромаджувального суматору, в якому зберігається шукане значення $arccos(x) \approx Ax^2 + Bx + C$. Слід зазначити, що запропоновану структуру можна використати для апаратного визначення арккосинуса за умови використання поліномів любої степені з тією відмінність, що зросте розрядність лічильника та число каналів передачі мультиплексора.

3.3. Використання для рендерингу тривимірних об’єктів сферично-кутової інтерполяції

Для зменшення складності обчислювального процесу при рендерингу використовують сферично-кутову інтерполяцію . На рис. 3.4 наведено приклад визначення одиничних векторів з використанням сферично-кутової інтерполяції

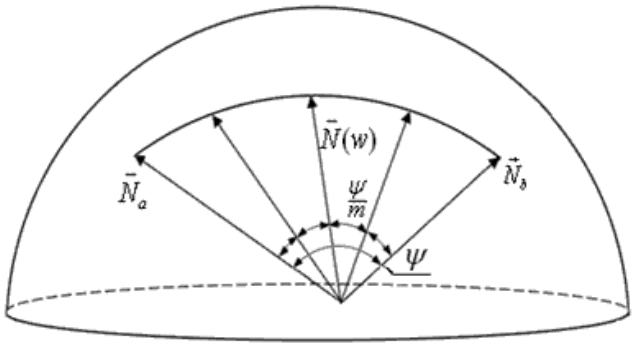


Рис. 3.4. Визначення нормалізованих векторів з використанням сферично-кутова інтерполяція

В [8] доведено, що

$$\vec{N}(t+1)=2\vec{N}(t)\cdot\cos\varphi-\vec{N}(t-1).$$

(3.1)

На основі останнього виразу можна дійти висновку, що одиничний вектор нормалі при використанні сферично-кутової інтерполяції можливо визначати, спираючись лише на два попередні його значення.

Проведений аналіз показав, що під час програмної реалізації час обчислення нормалей для середнього трикутника скоротився у 1.7 раза порівняно з звичайною реалізацією.

Отже, можна стверджувати про помітне зростання продуктивності процесу зафарбовування поверхні

Розглянемо програмно-апаратну реалізація зафарбовування.

Розрахунок векторів нормалей для точок на ребрах трикутника, а також інтенсивностей дифузної складової кольору та добутку $\vec{H}\cdot\vec{N}$ для перших двох точок рядка растеризації виконують на програмному рівні, а для решти точок - на апаратному рівні.

Структурна схема пристрою, який використовується для розрахунку дифузної складової кольору, наведено на рис. 3.5 Пристрій включає регістри *RG1-RG3*, лічильник *CT*, блок множення *MUL*, суматор *Sm*

Для розрахунку інтенсивності дифузної складової кольору третьої точки рядка

растерізації в регістри $RG1, RG3$ заносять значення інтенсивностей дифузної складової кольору $I(t-1)_d$, $I(t)_d$ перших двох точок рядка растеризації, а в регістр $RG2$ - $cos(\varphi/m) \approx (cos\varphi - 1)/m^2 + 1.$ З виходу регістра $RG3$ значення інтенсивності $2 \cdot I(t)_d$, отримане монтажним зсувом, надходить у блок множення MUL, де перемножується з $cos(\varphi/m)$, значення якого зберігається в регістрі $RG2$. З виходу пристрою множення отриманий добуток надходить на вхід суматора Sm, де від нього віднімається інтенсивність $I(t-1)_d$, яка зберігається в регістрі $RG1$.

На виході суматора Sm отримується значення інтенсивності дифузної складової кольору $I(t+1)_d$ третьої точки в рядку растеризації. Інтенсивність $I(t)_d$ із регістра RG3 записується у регістр $RG1$, а $I(t+1)_d$ - у регістр $RG3$.

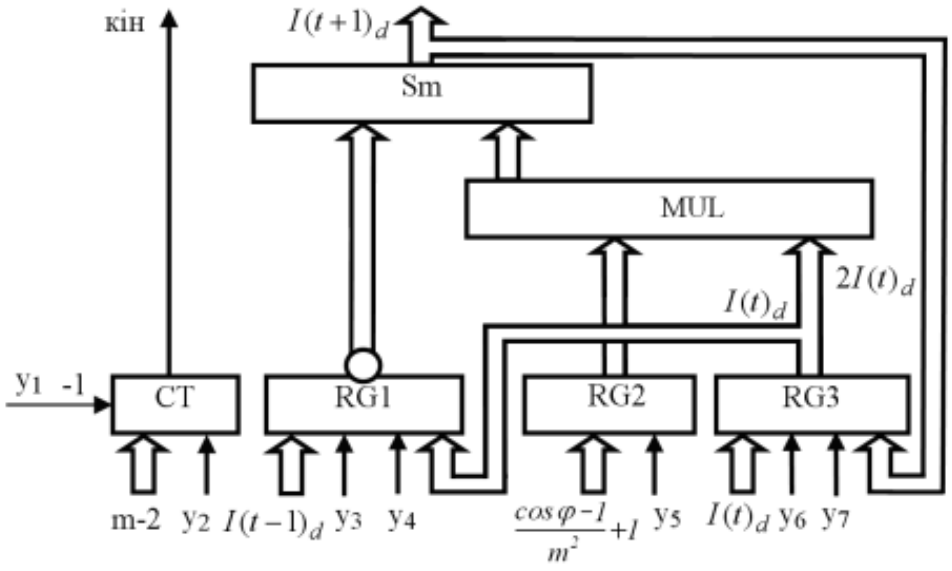


Рис. 3.5. Структурна схема пристрою для визначення дифузної складової кольору з використанням сферично-кутової інтерполяції

Після вказаних дій здійснюється розрахунок інтенсивності дифузної складової кольору для наступної точки рядка растеризації.

Робота блока по формуванню рядка растеризації трикутника закінчується після розрахунку інтенсивності дифузної складової кольору для останньої точки рядка растеризації. Про це свідчить сигнал “Кін” на виході лічильника *CT*, в який у циклі підготування заносять значення довжини рядка растеризації трикутника, зменшене на два.

Після визначення інтенсивності дифузної складової кольору потокової точки вміст лічильника *CT* декрементують.

Такий же блок використовується й для знаходження добутку $\vec{H} \cdot \vec{L}$ за винятком того, що регістр для зберігання $cos(\varphi / m)$ виконується спільним для розрахунку обох складових.

3.4 Апаратна реалізація рендерингу з використання косинус –квадратичної функції

Розглянемо програмно-апаратну реалізацію пристрою рендерингу, що використовує модель освітлення (3.2) $\left(\frac{n}{2}(cos\gamma - 1) + 1\right)^2$. Структурну схему пристрою для визначення інтенсивності спекулярної складової кольору з використання моделі $\left(\frac{n}{2}(cos\gamma - 1) + 1\right)^2$ зображено на рис. 3.6.

Пристрій містить блок постійної пам’яті *PROM*, регістри *RG1- RG5*, суматори *Sm1-Sm2*, блоки множення $MUL1 - MUL4$, схему порівняння *CMP*, блок ключів *BK*.

У регістри *RG1, RG2* записуються значення $cos\gamma$ і коефіцієнта *n* спекулярності поверхні, а у регістри *RG3- RG5* -значення інтенсивності *R, G, B* складових кольору, помножених на коефіцієнт k_s дзеркального відбиття поверхні.

На виході *Sm1* формується значення $(cos\gamma - 1)$, яке перемножується блоком *MUL1* зі значенням $n/2$, яке отримують монтажним зсувом операнду, що зберігається в регістрі *RG2*. До отриманого значення додається одиниця з використанням блока *MUL2* і підноситься до квадрату, тобто отримують значення

ДФВЗ, що дорівнює $\left(\frac{n}{2}(cos\gamma - 1) + 1\right)^2$. Значення коефіцієнта спекулярності n зі регістру $RG2$ поступає на вхід блоку постійної пам'яті, де зберігаються відповідні граничні значення ДФВЗ, після яких обчислення функції недоцільно.

Порівняння граничного значення ДФВЗ із розрахованим виконується схемою порівняння, яка формує відповідний сигнал для блоку ключів BK. Для отримання $-R, G, B$ складових спекулярної інтенсивності кольору значення дистрибутивної функції з виходу блоків ключів перемножують на операнди, які зберігаються в регістрах $RG3$- $RG5$.

Апаратна реалізація пристрою, у якому використовується ДФВЗ $W_3 \approx \left(\frac{7 \cdot n}{16} \cdot (cos\gamma - 1) + 1\right)^2$, додатково потребує всього одного суматору (див. рис. 3.7) , скільки множення на 8 і ділення на 16 легко реалізувати монтажним зсувом.

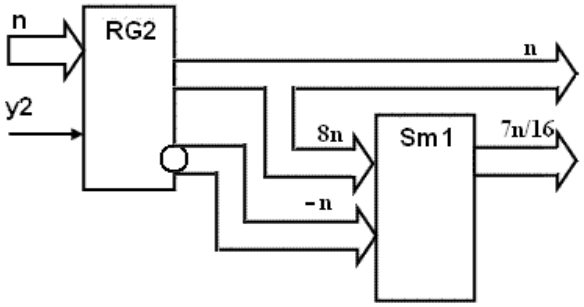


Рис. 3.7-Структурна схема блоку розрахунку значення $7n/16$

4. ПРАКТИЧНА РЕАЛІЗАЦІЯ РЕНДЕРИНГУ В СИСТЕМАХ КОМП'ЮТЕРНОЇ ГРАФІКИ

4.1.Програмна реалізація рендерингу Фонга

У бакалаврській кваліфікаційній роботі реалізовано алгоритм Фонга для відображення поверхневих елементів (найчастіше поверхня розбивається на трикутники, тому алгоритм застосовувався саме до трикутників). Алгоритм Фонга є більш точним порівняно з відомим алгоритмом Гуро, оскільки він враховує кривизну поверхні під час растеризації трикутника, однак програє йому за швидкістю (у будь-якому випадку можна сказати, що прогрес в одному аспекті завжди призводить до певних втрат в іншому; наприклад, досягнення високої швидкості може супроводжуватися погіршенням якості зображення, і навпаки).

Принцип роботи алгоритму Фонга [1, 2, 6] полягає у такому: початкові дані включають тривимірні координати вершин трикутника, вектори нормалей до поверхні у цих вершинах та три кольори (колір випромінювання поверхні, колір дифузії та колір джерела світла). Спочатку нормалізуються вектори нормалей.

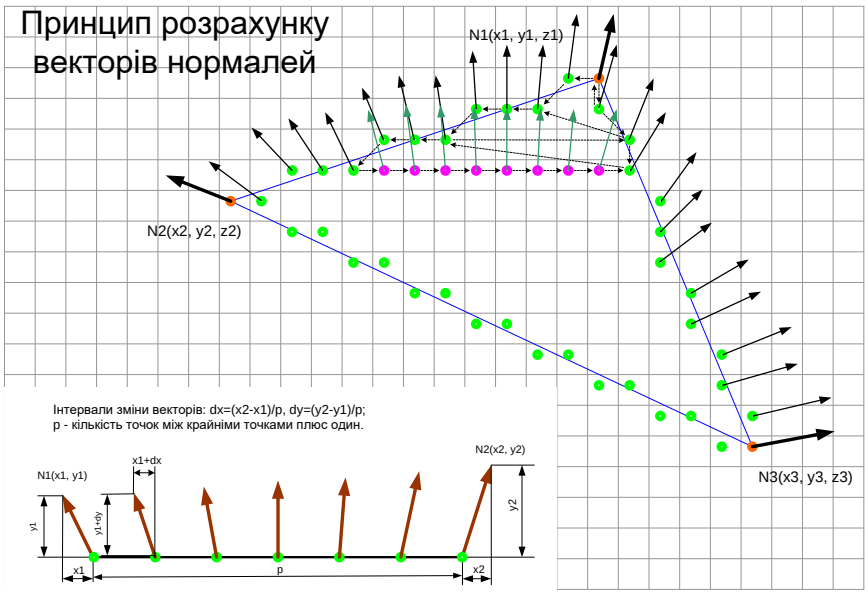


Рис. 4.1- Принцип розрахунку векторів

Потім визначається вершина з найменшим значенням координати Y (буде обрана для побудови). Обчислюються прирости по кожній з граней трикутника

(ΔX, ΔY, ΔZ для кожної сторони). Далі визначаються більші та менші прирости для кожної грані трикутника, враховуючи тільки прирости за координатами X та Y, оскільки на екрані монітора ми працюємо в двовимірному просторі, а координата Z необхідна для визначення косинуса кута між вектором нормалі та вектором, який є сумою векторів світла та спостерігача. Грань, що виходить з вершини з найменшим значенням Y і має більший приріст по Y, обирається як основна (довша сторона). Інша з цих двох граней буде коротшою. Грань, що залишилась, є нижньою стороною трикутника.

Інтерполяція всіх відрізків прямих здійснюється за допомогою алгоритму Обідника-Пєтуха [5], який використовує функцію оцінювання. Для застосування цього алгоритму необхідно визначити початкові значення функції оцінювання для кожної з граней трикутника. Окрім цього, для подальших розрахунків слід визначити, на яку величину змінювати координату Z. Це робиться шляхом ділення ΔZ на більший приріст за відповідною гранню, результатом чого буде дробове значення — крок зміни координати Z.

Аналогічно обчислюються кроки зміни компонентів векторів нормалей. Для цього різниця між відповідними компонентами векторів ділиться на більший приріст. Далі відбувається розрахунок координат точок по гранях трикутника та по внутрішній частині: інтерполяція починається від довшої сторони, і, коли здійснюється переміщення по координаті Y (вертикальний чи діагональний крок), переходять до інтерполяції коротшої прямої. Після завершення інтерполяції коротшої прямої повертаються до інтерполяції довшої.

Крім того, аналогічним чином інтерполюється інша частина трикутника, де замість коротшої сторони використовується нижня, а довша залишається без змін. При інтерполяції кожного відрізка кількість кроків відповідає більшому приросту цього відрізка. Під час кожного кроку алгоритму змінюється координата, для якої приріст більший, а іншу координату змінюють залежно від значення оціночної функції на поточному етапі. Важливим моментом є зміна координати Z: ця координата змінюється на дробове значення під час кожного кроку, тому для обчислень використовують тільки цілу частину числа, не відкидаючи дробову

частину для подальших операцій.

Розглянемо приклад, коли вектори, нормаль, світло та камера мають довільні напрямки.

Вхідні параметри:

Нормаль до поверхні:

$N = (0, 0, 1)$

Напрямок на світло:

$L = (0.577, 0.577, 0.577)$ (нормалізований вектор у напрямку по діагоналі, тобто $1/\sqrt{3}$)

Напрямок до камери:

$V = (0, 0, 1)$

Коефіцієнти:

$ka = 0.1$, $kd = 0.7$, $ks = 0.5$

Інтенсивність світла:

$Ia = 0.2$, $Id = 0.8$, $Is = 1.0$

Показник блиску:

$n = 10$

Виконати такі кроки алгоритму:

Крок 1. Навколишнє освітлення:

$I_{ambient} = ka \times Ia = 0.1 \times 0.2 = 0.02$

Крок 2. Дифузне освітлення:

$N \cdot L = (0, 0, 1) \cdot (0.577, 0.577, 0.577) = 0.577$

$I_{diffuse} = kd \times Id \times \max(0, N \cdot L) = 0.7 \times 0.8 \times 0.577 \approx 0.323$

Крок 3. Дзеркальне освітлення:

Відбитий вектор R:

$R = 2(N \cdot L)N - L$

$\rightarrow R = 2(0.577)(0,0,1) - (0.577, 0.577, 0.577)$

$\rightarrow R = (0, 0, 1.154) - (0.577, 0.577, 0.577)$

$\rightarrow R = (-0.577, -0.577, 0.577)$

Скалярний добуток R · V = (−0.577, −0.577, 0.577) · (0, 0, 1) = 0.577

I_specular = ks × Is × max(0, R · V)^n = 0.5 × 1.0 × (0.577)^10 ≈ 0.5 × 0.003 ≈ 0.0015

Крок 4. Загальна інтенсивність:

I_total = I_ambient + I_diffuse + I_specular

I_total = 0.02 + 0.323 + 0.0015 ≈ 0.3445

Результат:

Інтенсивність кольору ≈ 0.3445

На рис. 4.2 — 4.8 наведено граф-схеми складових рендерингу

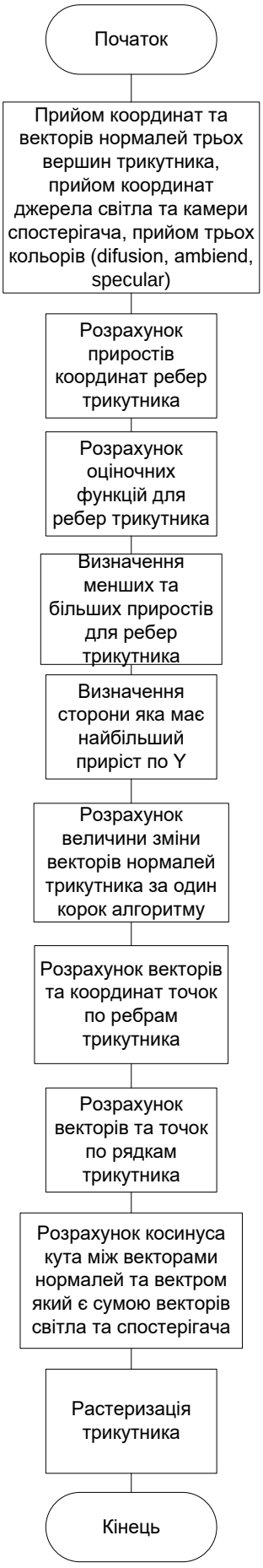


Рисунок 4.2 – Граф-схема алгоритму роботи програми

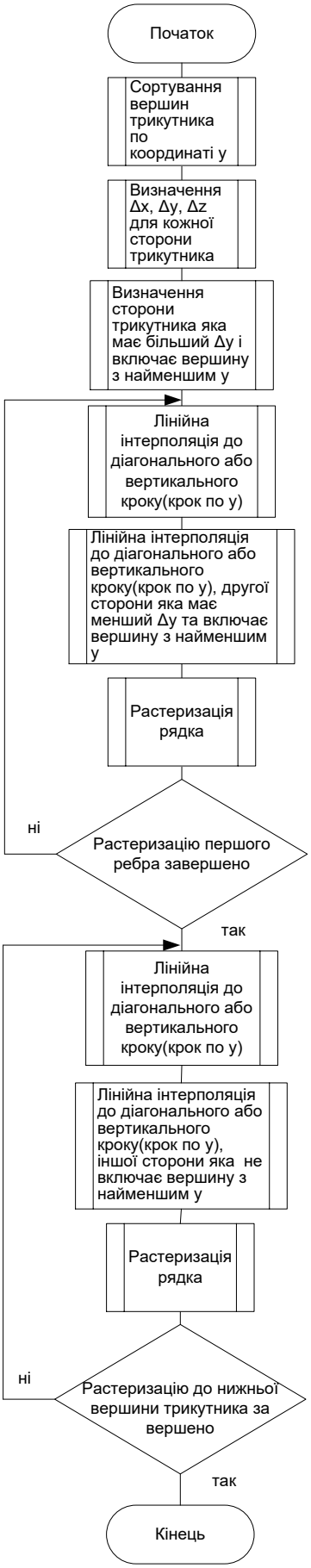


Рисунок 4.3 – Граф-схема алгоритму растеризації трикутника

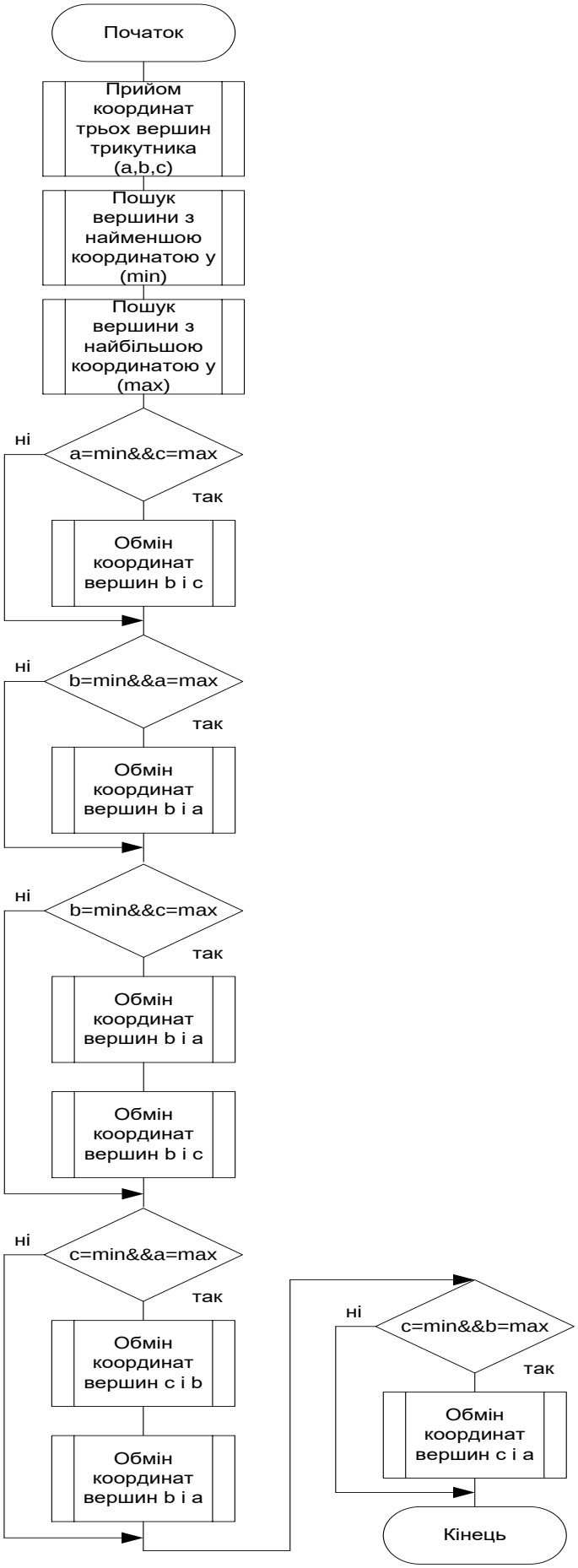


Рисунок 4.4– Граф-схема сортування верши трикутника

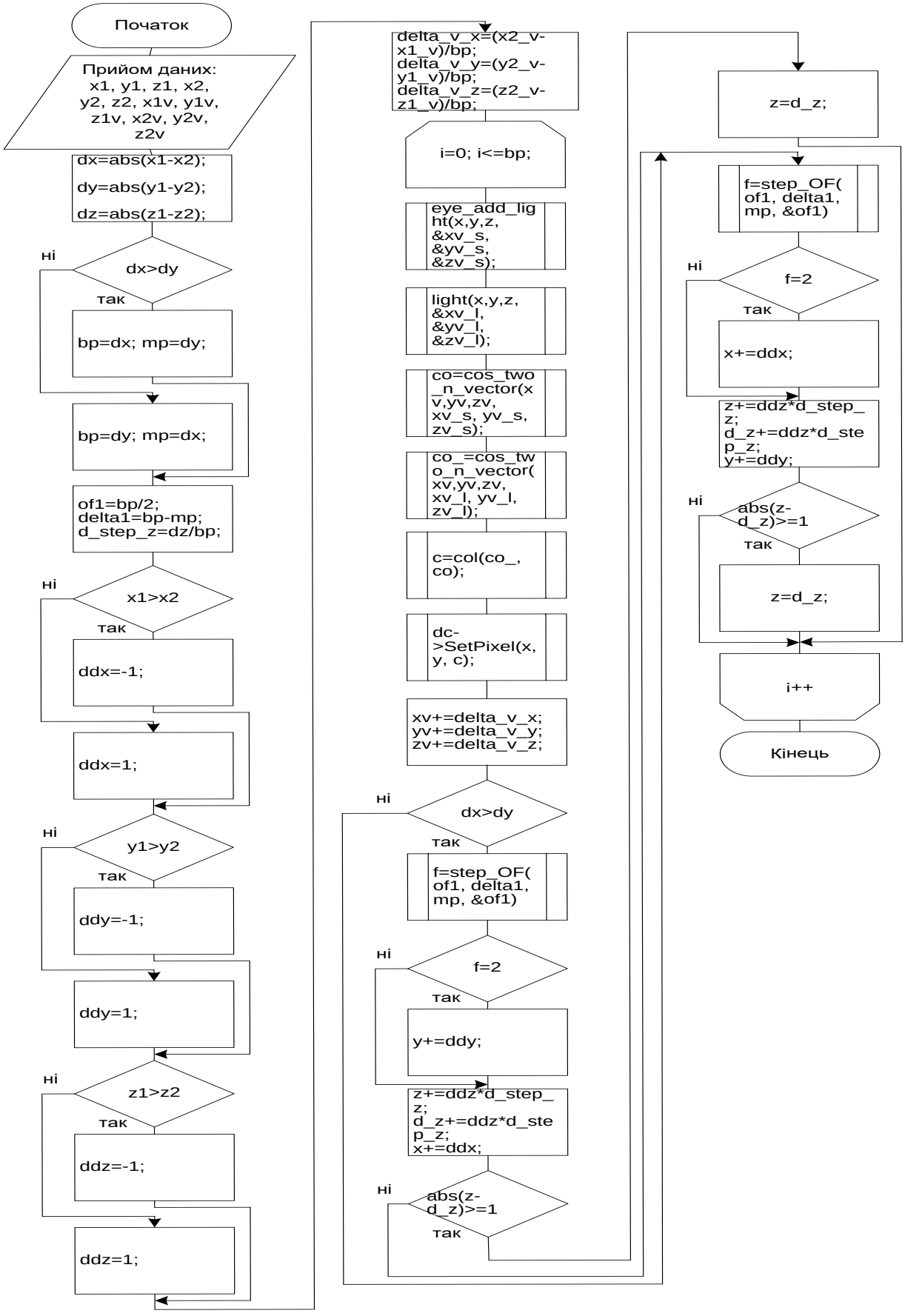


Рисунок 4.5 – Граф-схема підпрограми інтерполяції відрізка прямої

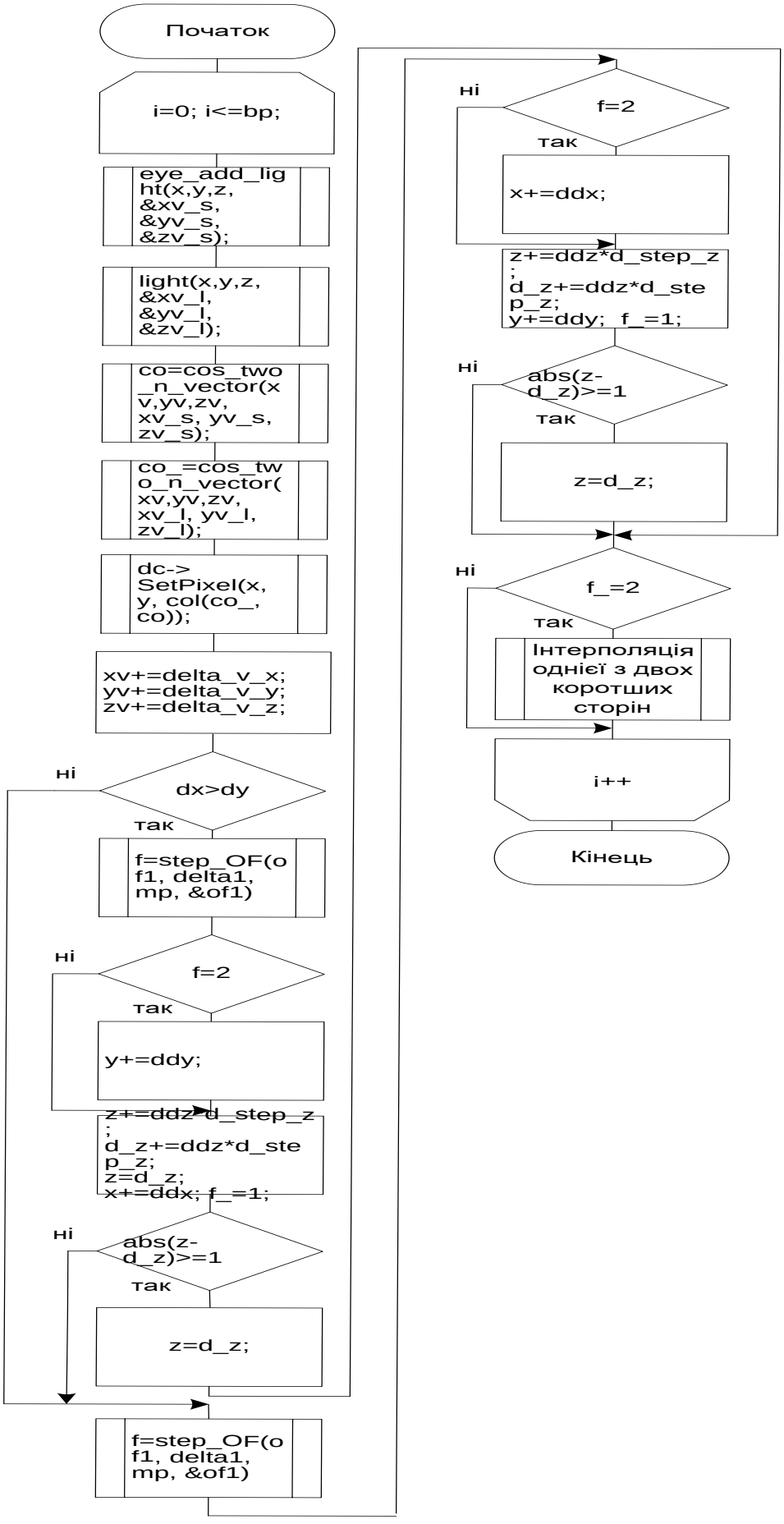


Рисунок 4.6 - Граф-схема алгоритму інтерполяції до діагонального кроку

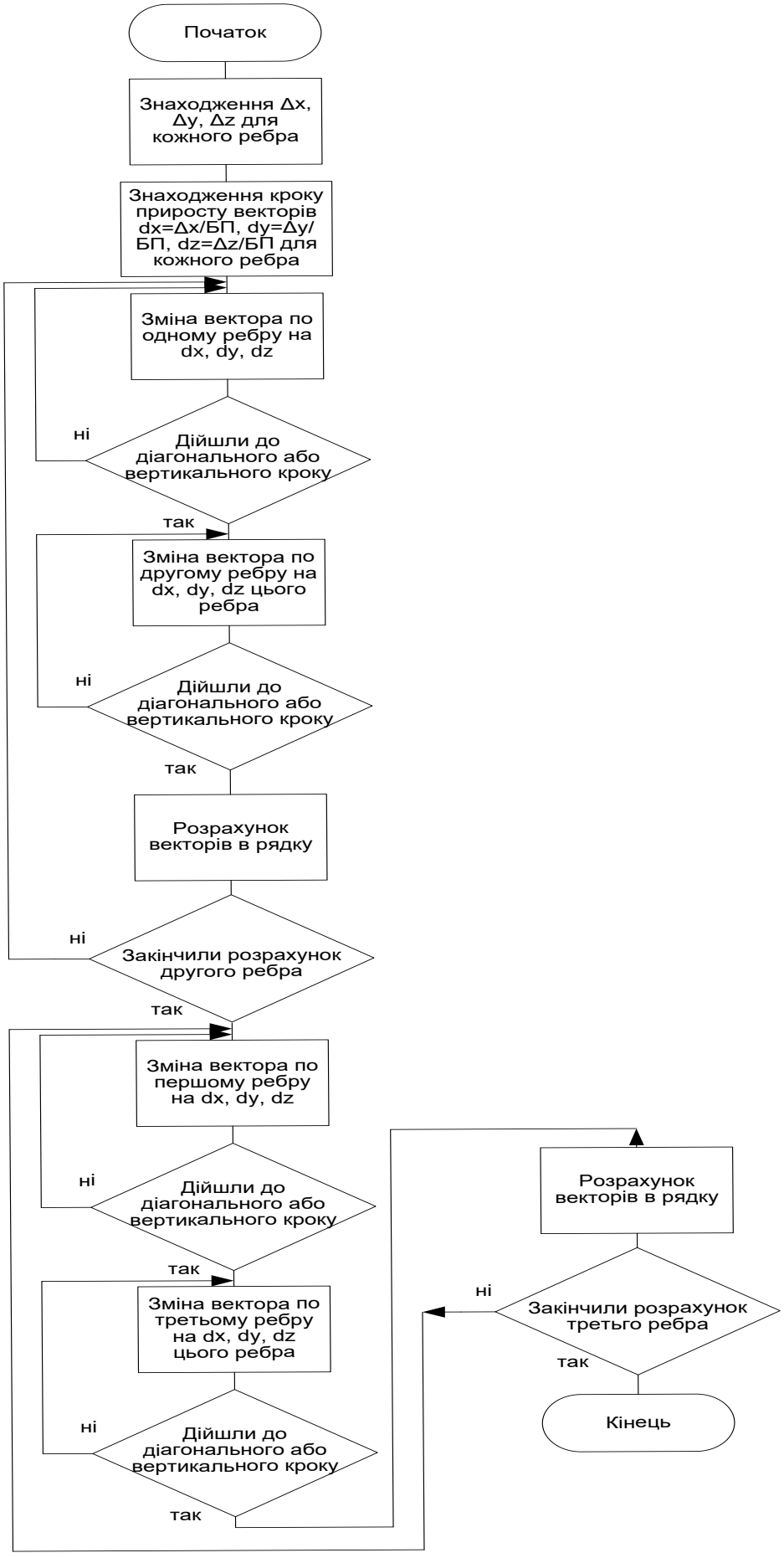


Рисунок 4.7 - Граф-схема розрахунку векторів нормалей

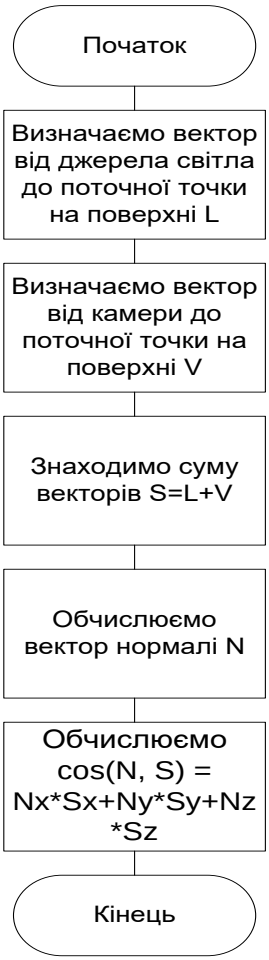


Рисунок 4.8 - Граф-схема алгоритму визначення cosφ

На рис. 4.9 наведено інтерфейс розробленої програми. У правому вікні відображається фігура, які сформовано як еталонну. Порівняння сформованого та еталонного зображення здійснювалося за MNSE (Mean Normalized Squared Error)

$$MNSE = (1 / N) * \Sigma [(I_i - \bar{O}_i)^2 / (I_i - \bar{I})^2]$$

де:

— N — кількість пікселів або точок у зображенні,

— I_i — значення інтенсивності освітлення для пікселя і в зображенні,

— $\bar{O}_i$ — значення інтенсивності передбаченого освітлення для пікселя і,

— $\bar{I}$ — середнє значення інтенсивності освітлення для всіх пікселів.

Ця формула використовується для вимірювання відносної похибки між передбаченими та фактичними інтенсивностями освітлення, зокрема в задачах

оцінки якості рендерингу або порівняння методів освітлення в комп'ютерній графіці.

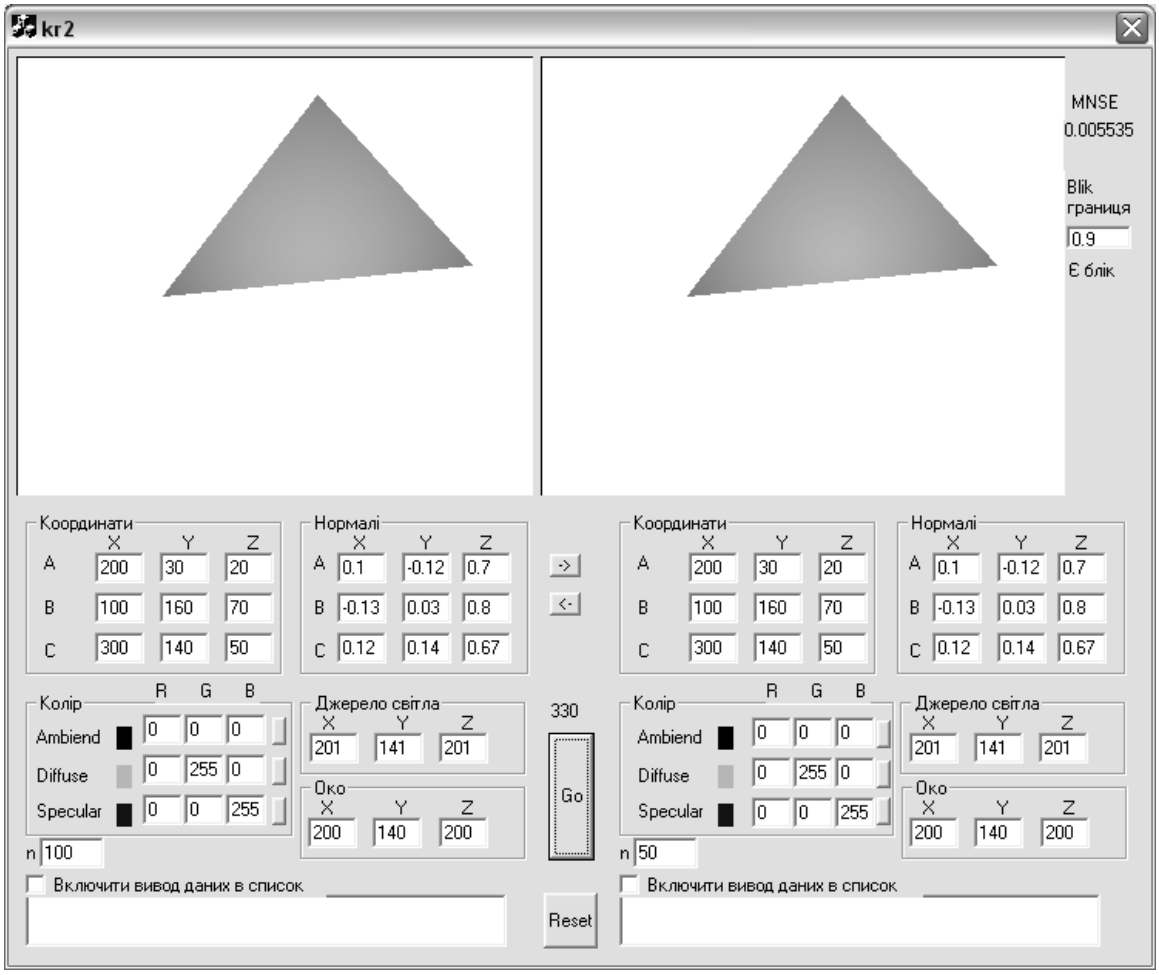


Рис. 4.9 –Інтерфейс користувача

Залежно від значення MNSE, можна оцінити якість рендерингу: коли MNSE близько до 0, це означає ідеальну точність рендерингу, де фактичні і передбачені значення освітлення або інтенсивності співпадають, що свідчить про дуже високу точність і мало помилок у відтворенні зображення. Якщо MNSE знаходиться близько до 1, це може вказувати на помітні відхилення між фактичними і передбаченими значеннями, що свідчить про те, що рендеринг не зовсім точний і можуть бути проблеми з відтворенням освітлення або текстур. Якщо ж MNSE перевищує 1, це вказує на значні помилки в рендерингу, що може бути результатом серйозних недоліків у моделюванні освітлення, обробці текстур чи інших

компонентів рендерингу, а також свідчить про наявність артефактів або невірно налаштованих параметрів у рендеринг-алгоритмі.

4.2 Структурна схема системи зафарбовування тривимірних графічних об’єктів.

Система зафарбовування поверхонь графічних зображень використовують для визначення кольору або відтінку кожної частини тривимірної поверхні в 3D-графіці. Метою є забезпечення візуальної реалістичності поверхонь при рендерингу, відтворенні світлових ефектів та інших властивостей матеріалів. Зазвичай це включає розрахунки інтенсивності кольору для кожного пікселя або елемента поверхні, що залежить від таких факторів, як освітлення, типи матеріалів, кути огляду та нормалі. Освітлення визначається джерелами світла (напрямок, яскравість, колір) і властивостями поверхні щодо відбиття світла (дифузне та дзеркальне відбиття), а також використанням моделей освітлення, таких як Фонг, Гуро, Ламберт тощо. Матеріали можуть бути матовими або глянцевими, що впливає на спосіб відображення світла, а текстури додають додаткові деталі на поверхню, наприклад, шорсткість або блиск. Основні етапи реалізації зафарбовування включають обчислення нормалей для кожної вершини, які визначають орієнтацію поверхні в просторі, використання моделей освітлення для розрахунку відбитого світла, застосування шейдерів для розрахунку кольору пікселів, накладання текстур для додавання деталей, а також пост-обробку, яка додає додаткові ефекти, як-от розмиття, підсвічування або корекція кольору для покращення візуального ефекту.

На рис. 4.10 зображено структуру підсистеми зафарбовування тривимірних графічних об’єктів.

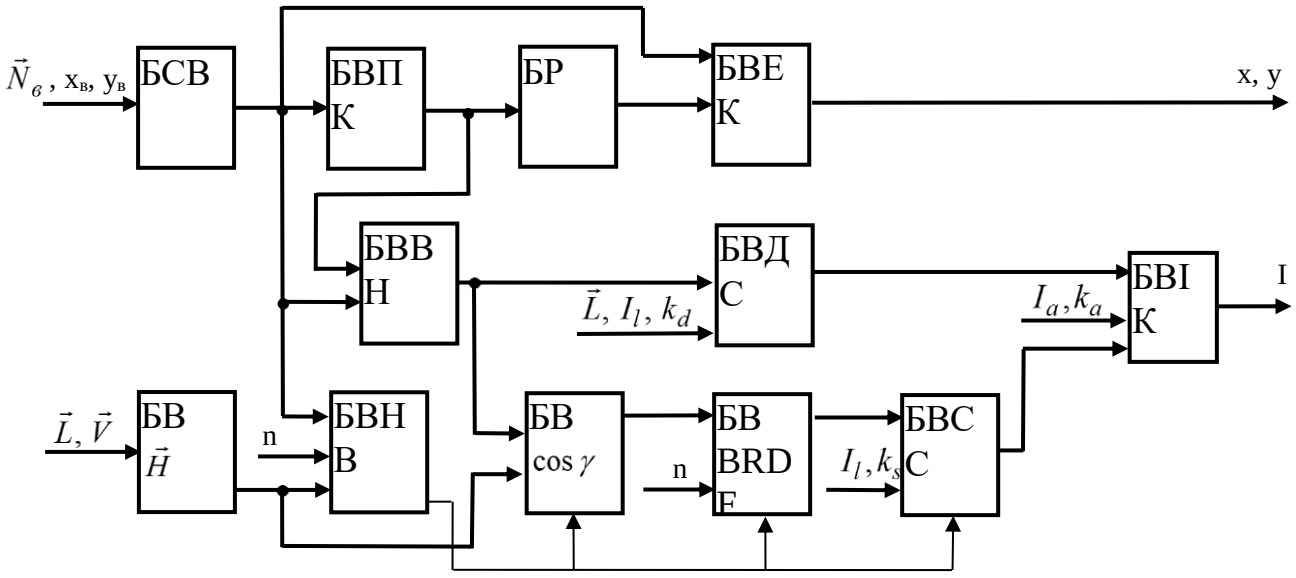


Рис. 6.27 Структура підсистеми зафарбовування тривимірних графічних об’єктів

Підсистема включає блоки: сортування вершин – БСВ; визначення приростів координат – БВПК; растеризації – БР; визначення екранних координат – БЕК; визначення векторівнормалей уздовж ребер трикутника – БВВН; визначення вектора $\vec{H}$ - БВ $\vec{H}$; визначення наявності відблиску – БВНВ; визначення $\cos\gamma$ - БВ $\cos\gamma$; визначення BRDF – БВ BRDF; визначення дифузної складової кольору – БВДС; визначення спекулярної складової кольору – БВСС; визначення інтенсивності кольору - БВІК.

У БВ $\vec{H}$ по напрямку джерела світла $\vec{L}$ та розташуванню спостерігача $\vec{V}$ розраховують HW – вектора $\vec{H}$. У БВНВ визначають наявність зони відблиску в межах трикутника, а у БВВН - вектори нормалей уздовж усіх ребер трикутника. У БВДС розраховують по рекурентним співвідношенням дифузну складову кольору потокової точки, а у БВ $\cos\gamma$ - добуток вектора нормалі $\vec{N}$ до потокової точки рядка растеризації і HW-вектора $\vec{H}$. По отриманим значенням $\cos\gamma$ та коефіцієнта спекулярності n у БВBRDF визначають значення двопроменевої дистрибутивної функції. Спекулярну складову кольору розраховують у БВСС. У БВІК по значенням розсіяної, дифузної та спекулярної

складових кольору визначають підсумкову інтенсивність кольору точки.

Характерна особливість роботи підсистеми зафарбовування полягає в тому, що при відсутності відблиску або його частини в межах трикутника на виході БВНВ формується сигнал, який призупиняє роботу блоків, задіяних на визначення спекулярної складової кольору. У цьому випадку розраховують тільки дифузну складову кольору, що суттєво підвищує продуктивність зафарбовування.

Визначення наявності відлиску в межах трикутника за адаптивним методом найбільш доцільно здійснювати на програмному рівні.

Для розрахунку дифузної складової кольору та добутку $\vec{H} \cdot \vec{N}$ для внутрішніх точок трикутника, в якому спостерігається відблиск або його частина, можна використати сферично-кутову інтерполяцію векторів нормалей .

ВИСНОВКИ

1. Доведено, що рендеринг є важливим етапом формування тривимірної графіки, який забезпечує візуальну реалістичність зображень шляхом моделювання освітлення, відбиття та тіней, і саме він найбільш ресурсоємний у графічному конвеєрі.
2. Обгрунтовано, що програмно-апаратна реалізація дозволяє значно підвищити швидкодію рендерингу, поєднуючи гнучкість програмного забезпечення з паралелізмом і енергоефективністю апаратних засобів.
3. Запропоновано нову модель відбивної здатності поверхні, що зменшує обчислювальну складність за рахунок зниження ступеня в експоненційних виразах.
4. Розроблено ефективні алгоритми нормалізації векторів, зокрема з використанням бінарного поділу та поліноміальної інтерполяції другого ступеня, які скорочують час обчислень у 2–2.4 раза при незначних похибках.
5. Метод Фонга обґрунтовано як основний для реалізації освітлення, оскільки він забезпечує плавні переходи освітленості, реалістичні спекулярні ефекти та є базою для подальших PBR-моделей.
6. Було реалізовано апаратний модуль для визначення спекулярної складової кольору, що дозволяє винести найзатратніші обчислення на рівень спеціалізованої логіки.
7. Створено структурні схеми пристроїв та алгоритми для апаратної частини, що реалізують етапи растеризації, нормалізації, освітлення та текстурування в реальному часі.
8. Сформовано критерії оцінки ефективності програмно-апаратного рендерингу, серед яких: FPS, затримка, апаратне завантаження, масштабованість, енергоефективність, підтримка сучасних API.
9. Результати роботи апробовані на конференціях та опубліковані в 5 збірниках конференцій.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Романюк О. Н., Завальнюк Є. К., Дембіцький О. М., Черняк О. І., Романюк О. В. Перспективи розвитку методу Фонга для формування реалістичних графічних зображень // Освіта, наука та виробництво: розвиток та перспективи : матеріали X Всеукр. наук.-метод. конф., 24 квіт. 2025 р. – Шостка, 2025. – С. 242–247.

2. Черняк О. І., Дембіцький О. М., Романюк О. Н. Модифікації методу Фонга для формування тривимірних зображень // Інформаційні технології – 2025 : зб. тез XII Всеукр. наук.-практ. конф. молодих учених, 15 трав. 2025 р., м. Київ. – Київ : Київ. столич. ун-т ім. Бориса Грінченка, 2025. – С. 252–254. – ISSN 2664-2638.

3. Романюк О. Н., Черняк О. І., Дембіцький О. М., Романюк О. В. Підвищення реалістичності зафарбовування за методом Гуро // Інформаційні технології: наука, техніка, технологія, освіта, здоров’я : тези доп. XXXIII міжнар. наук.-практ. конф. MicroCAD-2025, 14–17 трав. 2025 р. / за ред. проф. Сокола Є. І. – Харків : НТУ «ХПІ», 2025. – С. 1632.

4. Романюк О. Н., Романюк О. В., Майданюк В. П., Дембіцький О. М. Нововведення в сучасних графічних процесорах // Освіта, наука та виробництво: розвиток та перспективи : матеріали X Всеукр. наук.-метод. конф., 24 квіт. 2025 р. – Шостка, 2025. – С. 223–225.

5. Романюк О. Н., Черняк О. І., Дембіцький О. М. Особливості програмно-апаратної реалізації рендерингу // Освіта, наука та виробництво: розвиток та перспективи : матеріали X Всеукр. наук.-метод. конф., 24 квіт. 2025 р. – Шостка, 2025. – С. 226–230.

6. Möller T., Haines E., Hoffman N. Real-Time Rendering. 4th ed. Boca Raton : CRC Press, 2018. 1045 p.

7. Glatzel B. 3D Graphics Rendering Cookbook: A comprehensive guide to exploring rendering algorithms in modern OpenGL and Vulkan. Birmingham : Packt Publishing, 2021. 452 p.

8. Lengyel E. Mathematics for 3D Game Programming and Computer Graphics. 4th ed. Boston : Cengage Learning, 2022. 600 p.

9. Gonzalez J. Blender 2.8 for Technical Drawing: Render 2D drawings for architecture, engineering, and design. Independently published, 2020. 200 p.

10. Селбі Е. Анімація. Тривимірні комп’ютерні зображення / пер. з англ. В. Заєць. – Київ : ArtHuss, 2023. – 240 с.

11. Pharr M., Jakob W., Humphreys G. Physically Based Rendering: From Theory to Implementation. 4th ed. Cambridge : The MIT Press, 2023. 1266 p.

12. Gambetta G. Computer Graphics from Scratch: A Programmer’s Introduction to 3D Rendering. San Francisco : No Starch Press, 2021. 248 p.

13. Castorina M. Mastering Graphics Programming with Vulkan: Develop a Modern Rendering Engine from First Principles to State-of-the-Art Techniques. Birmingham : Packt Publishing, 2021. 450 p.

14. Маценко В. Г. Обчислювальна геометрія та комп’ютерна графіка : навч. посіб. / В. Г. Маценко ; М-во освіти і науки України, Чернів. нац. ун-т ім. Юрія Федьковича. – Чернівці : Чернів. нац. ун-т ім. Юрія Федьковича, 2023. – 440 с

15. Kosarevsky S. Vulkan 3D Graphics Rendering Cookbook: Implement Expert-Level Techniques for High-Performance Graphics with Vulkan. Birmingham : Packt Publishing, 2020. 452 p.

16. Lengyel E. Foundations of Game Engine Development, Volume 2: Rendering. Lincoln : Terathon Software LLC, 2019. 300 p.

17. Романюк О. Н., Романюк О. В., Чехместрук Р. Ю. Комп’ютерна графіка : електронний навч. посіб. [Електрон. ресурс]. – Вінниця : ВНТУ, 2023. – 147 с.

18. Карпюк Л. В., Татарченко Г. О., Білошицька Н. І. Комп’ютерна графіка у будівництві : навч. посіб. – Сєвєродонецьк : Вид-во СНУ ім. В. Даля, 2020. – 180 с.

19. Пічугін М. Ф., Канкін І. О., Воротніков В. В. Комп’ютерна графіка : навч. посіб. – Київ : Центр навч. літ., 2019. – 346 с.

20. Козяр М. М., Фещук Ю. В. Комп’ютерна графіка: AutoCAD : навч. посіб. – Херсон : Олді-плюс, 2020. – 304 с.

21. Царенко М. О. Комп’ютерна графіка : навч. посіб. – Одеса : ПНПУ ім. К. Д. Ушинського, 2020. – 120 с.

ДОДАТОК А

Міністерство освіти і науки України

Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерно інженерії

Кафедра обчислювальної техніки

ЗАТВЕРДЖУЮ

Завідувач кафедри

ОТ д.т.н., проф.

Азаров О. Д.

« » 2025 року

ТЕХНІЧНЕ ЗАВДАННЯ

на виконання бакалаврської дипломної роботи "«Високопродуктивні програмно-апаратні засоби для рендерингу графічних сцен»

08-54.БДР.007.00.000 ТЗ

Науковий керівник: к.т.н.

доцент

Кафдри ОТ

_____Черняк О. І.

Студент групи 1КІ-23мс

_____Дембіцький О.М.

м. Вінниця – 2025

1 Підстава для виконання бакалаврської дипломної роботи (БДР)

1.1 Рендеринг тривимірних графічних сцен вимагає великих обчислювальних витрат, що передбачає актуальність його програмно-апаратної реалізації.

1.2 Наказ про затвердження теми БДР від «20» березня 2025 року № 80.

2 Мета БДР і призначення розробки

2.1 Мета роботи —підвищення продуктивності рендерингу за рахунок програмно-апаратної реалізації

2.2 Призначення розробки — розробка програмно-апаратних засобів для високопродуктивних графічних сцен

3 Вихідні дані для виконання БДР

Базовий метод зафарбовування – метод Фонга; моделі освітлення – моделі Бліна, Фонга ; графічний режим – TrueColor; вихідні дані для зафарбовування – напрямки векторів нормалей до вершин трикутників; дані про розташування джерела світла та спостерігача; коефіцієнт спекулярності поверхні; координати вершин трикутників; максимальна розрядність для задання адрес вершин трикутників –12; вихідні дані – кінцеве зображення, зафарбоване згідно методу Фонга в поєднанні з моделлю освітлення Бліна або Ламбертова.

4 Вимоги до виконання БДР

Результати повинні відповідати стандартам візуалізації. Програмна підтримка планування задач на GPU. Система повинна підтримувати BDRF-моделі (Bidirectional Reflectance Distribution Function)

5 Етапи БДР та очікувані результати

Етапи роботи та очікувані результати наведено у Таблиці А.1.

Таблиця А.1 — Етапи БДР

№	Назва та зміст етапу	Термін виконання	Примітка
1.	Постановка задачі роботи	21.03.25	Початок вик.
2.	Аналіз методів і засобів рендерингу	22.03-23.03.25	Розділ 1
3.	Основні етапи графічного конвеєра	24.03-26.03.25	Розділ 1
4.	Методи зафарбування у рендерингу	27.03-02.04.25	Розділ 1
5.	Критерії оцінювання програмно-апаратної реалізації рендерингу	03.04-07.04.25	Розділ 1
6.	Прискорення визначення нормалізованих векторів	08.04-13.04.25	Розділ 2
7.	Застосування методу бінарного поділу для визначення одиночних векторів	14.04-18.04.25	Розділ 2
8.	Апаратна нормалізація векторів нормалей	19.04.-25.04.25	Розділ 2
9.	Програмно-апаратна реалізація методів зафарбування	26.03.-30.04.25	Розділ 3
10	Пристрій для визначення спекулярної складової кольору	30.03-12.04.25	Розділ 3
11.	Практична реалізація рендерингу в системах комп’ютерної графіки	12.04-27.04.25	Розділ 4
12.	Оформлення пояснювальної записки до бакалаврської дипломної роботи	27.04-12.05.25	ПЗ, графіч. матеріал
13.	Перевірка якості виконання бакалаврської роботи	13.05.25	Оформлені документи

6 Матеріали, що подаються до захисту БДР

До захисту подаються: пояснювальна записка БДР, ілюстративні матеріали, протокол попереднього захисту БДР на кафедрі, відгук наукового керівника, анотації до БДР українською та іноземною мовами, довідка про відповідність оформлення БДР діючим вимогам.

7 Порядок контролю виконання та захисту БДР

Виконання етапів графічної та розрахункової документації БДР контролюється науковим керівником згідно зі встановленими термінами. Захист БДР відбувається на засіданні Екзаменаційної комісії, затвердженої наказом ректора.

8 Вимоги до оформлювання та порядок виконання БДР

8.1 При оформлювання БКР використовуються:

— ДСТУ 3008: 2015 «Звіти в сфері науки і техніки. Структура та правила оформлювання»;

— ДСТУ 8302: 2015 «Бібліографічні посилання. Загальні положення та правила складання»;

— міждержавний ГОСТ 2.104-2006 «Єдина система конструкторської документації. Основні написи»;

— методичні вказівки до виконання бакалаврських кваліфікаційних робіт зі спеціальності 123 – «Комп’ютерна інженерія» (освітня програма «Комп’ютерна інженерія»). Кафедра обчислювальної техніки ВНТУ 2023;

— документами на які посилаються у вище вказаних.

8.2 Порядок виконання БКР викладено в «Положення про кваліфікаційні роботи на першому (бакалаврському) рівні вищої освіти СУЯ ВНТУ-03.02.02- П.001.01:21»

ДОДАТОК Б

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Назва роботи: Високопродуктивні програмно-апаратні засоби для рендерингу графічних сцен

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

Підрозділ _____кафедра обчислювальної техніки_____
(кафедра, факультет, навчальна група)

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism ____2,5___%

Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне)

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

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

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

Експертна комісія:

_____Азаров О.Д., зав. каф. ОТ _____ _____
(прізвище, ініціали, посада) (підпис)

____Крупельницький Л.В., доц. каф ОТ_____ _____
(прізвище, ініціали, посада) (підпис)

Особа, відповідальна за перевірку _____ ___Захарченко С.М.____
(підпис) (прізвище, ініціали)

З висновком експертної комісії ознайомлений(-на)

Керівник _____ _____Черняк О. І._____
(підпис) (прізвище, ініціали, посада)

Здобувач _____ _____Дембіцький О. М._____
(підпис) (прізвище, ініціали)

ДОДАТОК В

Лістигн модуля Fong.java

```
public class AdaptivGourauAndFong extends idx3d_Rasterizer
{
   private boolean bTypeRasterizer=false; //true - Guoro, false - Fong.
   private double m12, m13, m23;
   private double q_1, q_2, q_3;
   private double p_1, p_2, p_3;
   private double opt_m=0.9;
   private double opt_q=0.004;
   private double opt_p=0.008;
   private idx3d_Vector h;
   int i1, i2, i3;
   float ii12r, ii12g, ii12b;
   float ii13r, ii13g, ii13b;
   float ii23r, ii23g, ii23b;
   float liir, liig, liib;
   float riir, riig, riib;
   float lir, lig, lib;
   float rir, rig, rib;
   float i4r, i4g, i4b;
   //End for Gouraund
   //for Fong
   private idx3d_Vector v4;
   private idx3d_Vector vi13, vi12, vi23;
   private idx3d_Vector lvi, rvi;
   private idx3d_Vector lv, rv;
   //End for Fong
   public boolean selectMetod()
   {
      m12=p1.n.x*p2.n.x+p1.n.y*p2.n.y+p1.n.z*p2.n.z;
      m13=p1.n.x*p3.n.x+p1.n.y*p3.n.y+p1.n.z*p3.n.z;
      m23=p2.n.x*p3.n.x+p2.n.y*p3.n.y+p2.n.z*p3.n.z;
      if(Math.abs(m12)>opt_m ||
            Math.abs(m13)>opt_m ||
            Math.abs(m23)>opt_m) return false;
      else
      {
         h = idx3d_Vector.add(scene.light[0].v, scene.defaultCamera.lookat);
         q_1=p1.n.x*h.x+p1.n.y*h.y+p1.n.z*h.z;
         q_2=p2.n.x*h.x+p2.n.y*h.y+p2.n.z*h.z;
         q_3=p3.n.x*h.x+p3.n.y*h.y+p3.n.z*h.z;
         if(Math.abs(q_1-q_2)>opt_q ||
```

```
                Math.abs(q_1-q_3)>opt_q ||
                Math.abs(q_2-q_3)>opt_q) return false;
            else
            {

p_1=p1.n.x*scene.light[0].v.x+p1.n.y*scene.light[0].v.y+p1.n.z*scene.light[0].v.z;

p_2=p2.n.x*scene.light[0].v.x+p2.n.y*scene.light[0].v.y+p2.n.z*scene.light[0].v.z;

p_3=p3.n.x*scene.light[0].v.x+p3.n.y*scene.light[0].v.y+p3.n.z*scene.light[0].v.z;
                if(Math.abs(p_1-p_2)>opt_p ||
                   Math.abs(p_1-p_3)>opt_p ||
                   Math.abs(p_2-p_3)>opt_p) return false;
                else return true;
            }
        }
        //return true;
    }
    public AdaptivGourauAndFong()
    {
    }
    protected void initRender()
    {
        bTypeRasterizer=selectMetod();
     if(bTypeRasterizer)
     {
       //for Gouraund
        h = idx3d_Vector.add(scene.light[0].v, scene.defaultCamera.lookat);
        int dy12 = Math.abs(p1.y-p2.y);
        int dy13 = Math.abs(p3.y-p1.y);
        int dy23 = Math.abs(p3.y-p2.y);
        i1 = calcColor(p1.n2);
        i2 = calcColor(p2.n2);
        i3 = calcColor(p3.n2);
        if (dy12 != 0)
        {
            ii12r = ((float)(idx3d_Color.getRed(i2) - idx3d_Color.getRed(i1))) /
(float)dy12;
            ii12g = ((float)(idx3d_Color.getGreen(i2) - idx3d_Color.getGreen(i1))) /
(float)dy12;
            ii12b = ((float)(idx3d_Color.getBlue(i2) - idx3d_Color.getBlue(i1))) /
(float)dy12;
        }
        if (dy13 != 0)
        {
```

```
        ii13r = ((float)(idx3d_Color.getRed(i3) - idx3d_Color.getRed(i1))) /
(float)dy13;
        ii13g = ((float)(idx3d_Color.getGreen(i3) - idx3d_Color.getGreen(i1))) /
(float)dy13;
        ii13b = ((float)(idx3d_Color.getBlue(i3) - idx3d_Color.getBlue(i1))) /
(float)dy13;
      }
      if (dy23 != 0)
      {
        ii23r = ((float)(idx3d_Color.getRed(i3) - idx3d_Color.getRed(i2))) /
(float)dy23;
        ii23g = ((float)(idx3d_Color.getGreen(i3) - idx3d_Color.getGreen(i2))) /
(float)dy23;
        ii23b = ((float)(idx3d_Color.getBlue(i3) - idx3d_Color.getBlue(i2))) /
(float)dy23;
      }
      i4r = (dy12*ii13r) + idx3d_Color.getRed(i1);
      i4g = (dy12*ii13g) + idx3d_Color.getGreen(i1);
      i4b = (dy12*ii13b) + idx3d_Color.getBlue(i1);
      //End for Gouraund
    }
    else
    {
      //for Fong
       h = idx3d_Vector.add(scene.light[0].v, scene.defaultCamera.lookat);
      int dx12 = Math.abs(p1.x-p2.x);
      int dx13 = Math.abs(p3.x-p1.x);
      int dx23 = Math.abs(p3.x-p2.x);
      int dy12 = Math.abs(p1.y - p2.y);
      int dy13 = Math.abs(p3.y - p1.y);
      int dy23 = Math.abs(p3.y - p2.y);
      vi13 = new idx3d_Vector((p3.n2.x - p1.n2.x) / dy13,
                        (p3.n2.y - p1.n2.y) / dy13,
                        (p3.n2.z - p1.n2.z) / dy13);
      vi12 = new idx3d_Vector((p2.n2.x - p1.n2.x) / dy12,
                        (p2.n2.y - p1.n2.y) / dy12,
                        (p2.n2.z - p1.n2.z) / dy12);
      vi23 = new idx3d_Vector((p3.n2.x - p2.n2.x) / dy23,
                        (p3.n2.y - p2.n2.y) / dy23,
                        (p3.n2.z - p2.n2.z) / dy23);
      v4 = idx3d_Vector.add(idx3d_Vector.scale(dy12, vi13), p1.n2);
      //End for Fong
    }
  }
  protected void prerenderFirstPart()
```

```
{
 if(bTypeRasterizer)
 {
   //for Gouraund
   lir = idx3d_Color.getRed(i1);
   lig = idx3d_Color.getGreen(i1);
   lib = idx3d_Color.getBlue(i1);
   rir = idx3d_Color.getRed(i1);
   rig = idx3d_Color.getGreen(i1);
   rib = idx3d_Color.getBlue(i1);
   if (dx > 0)
   {
      liir = ii12r;
      liig = ii12g;
      liib = ii12b;
      riir = ii13r;
      riig = ii13g;
      riib = ii13b;
   }
   else
   {
      riir = ii12r;
      riig = ii12g;
      riib = ii12b;
      liir = ii13r;
      liig = ii13g;
      liib = ii13b;
   }
   //End for Gouraund
 }
   else
 {
   //for Fong
   lv = p1.n2;
   rv = p1.n2;
   if (dx > 0)
   {
      lvi = vi12;
      rvi = vi13;
   }
   else
   {
      lvi = vi13;
      rvi = vi12;
   }
```

```
    //End for Fong
  }
 }
protected void prerenderSecondPart()
{
 if(bTypeRasterizer)
 {
    //for Gouraund
    if (dx > 0)
    {
       lir = idx3d_Color.getRed(i2);
       lig = idx3d_Color.getGreen(i2);
       lib = idx3d_Color.getBlue(i2);
       rir = i4r;
       rig = i4g;
       rib = i4b;
       liir = ii23r;
       liig = ii23g;
       liib = ii23b;
       riir = ii13r;
       riig = ii13g;
       riib = ii13b;
    }
    else
    {
       rir = idx3d_Color.getRed(i2);
       rig = idx3d_Color.getGreen(i2);
       rib = idx3d_Color.getBlue(i2);
       lir = i4r;
       lig = i4g;
       lib = i4b;
       riir = ii23r;
       riig = ii23g;
       riib = ii23b;
       liir = ii13r;
       liig = ii13g;
       liib = ii13b;
    }
    //End for Gouraund
 }
 else
 {
    //for Fong
    if (dx > 0)
    {
```

```
        lv = p2.n2;
        rv = v4;
        lvi = vi23;
        rvi = vi13;
    }
    else
    {
        lv = v4;
        rv = p2.n2;
        lvi = vi13;
        rvi = vi23;
    }
    //End for Fong
 }
}
protected void prerenderLine()
{
 if(bTypeRasterizer)
 {
    //for Gouraund
    renderLine();
    lir += liir;
    lig += liig;
    lib += liib;
    rir += riir;
    rig += riig;
    rib += riib;
    //End for Gouraund
 }
 else
 {
    //for Fong
    renderLine();
    lv = idx3d_Vector.add(lv, lvi);
    rv = idx3d_Vector.add(rv, rvi);
    //End for Fong
 }
}
public String getRasterizerName()
{
    return "Adaptiv Gouraud And Fong";
}
protected void renderLine()
{
 if(bTypeRasterizer)
```

```
    {
      //for Gouraund
      int cdx = xR - xL;
      if (cdx == 0) return;
      float iir = 0;
      float iig = 0;
      float iib = 0;
      float cir = 0;
      float cig = 0;
      float cib = 0;
      float lastr = lir;
      float lastg = lig;
      float lastb = lib;
      boolean bReset = true;
      for (x=xL;x<xR;x++)
      {
         pos=x+offset;
         if (z<zBuffer[pos])
         {
            if (rir==255 && rig==255 && rib==255 )
            {
               return;
            }
            if (bReset)
            {
               cdx = xR - x;
               if ((xR - x)>1)
               {
                  iir = (rir - lir) / cdx;
                  iig = (rig - lig) / cdx;
                  iib = (rib - lib) / cdx;
                  bReset = false;
               }
               cir = lir;
               cig = lig;
               cib = lib;
            }
            screen.p[pos]=0xFF000000 | idx3d_Color.getColor(Math.round(cir > 255 ?
255 : cir), Math.round(cig > 255 ? 255 : cig), Math.round(cib > 255 ? 255: cib));
//             screen.p[pos]=0xFF000000 | idx3d_Color.getColor(255, 255, 255);
            zBuffer[pos]=z;
            if (useIdBuffer) idBuffer[pos]=currentId;
            cir += iir;
            cig += iig;
            cib += iib;
```

```
        }
        else
            bReset = true;
    }
    z += dz;
    //End for Gouraund
  }
  else
  {
    //for Fong
    int cdx = xR - xL;
    if (cdx==0)
        return;
    for (int x=xL; x<xR; x++)
    {
        pos=x+offset;
        if (z<zBuffer[pos])
        {
            idx3d_Vector cv = idx3d_Vector.GetVectorOnPos(lv, rv, xL, xR, x);
            float angle = idx3d_Vector.angle(scene.light[0].v, cv);
            if (angle < 0) angle = 0;
            c = idx3d_Color.scale(color, (int)(angle*255));
            angle = idx3d_Vector.angle(h, cv);
            if (angle < 0) angle = 0;
            angle = (float)Math.pow(angle, n);
            int c2 = idx3d_Color.scale(0xFFFFFF, (int)(angle*255));
            c = idx3d_Color.add(c, c2);
            screen.p[pos]=0xFF000000|c;
            zBuffer[pos]=z;
            if (useIdBuffer) idBuffer[pos]=currentId;
        }
        z+=dz;
        nx+=dnx;
        ny+=dny;
    }
    //End for Fong
  }
}
//for Gouraund
private int calcColor(idx3d_Vector vector)
{
    float angle = idx3d_Vector.angle(scene.light[0].v, vector);
    if (angle < 0) angle = 0;
    int c = idx3d_Color.scale(color, (int)(angle*255));
    angle = idx3d_Vector.angle(h, vector);
```

```
        if (angle < 0) angle = 0;
        angle = (float)Math.pow(angle, n);

        int c2 = idx3d_Color.scale(0xFFFFFF, (int)(angle*255));
        c = idx3d_Color.add(c, c2);
        return c;
    }
    //End for Gouraund
    protected void beforeRenderTriangle()
    {
    }
}
```

ДОДАТОК Г

Лістинг модуля NormalVector.java

```
public class NormalVector extends idx3d_Rasterizer
{
   private idx3d_Vector h, v4;
   private idx3d_Vector vi13, vi12, vi23;
   private idx3d_Vector lvi, rvi;
   private idx3d_Vector lv, rv;
   int n_2;
   private idx3d_Vector N_2 ;
   int Nl,Nr, count_n;
   private idx3d_Vector dN_i,Na,Nb;
   private RET Ret[];
   public NormalVector()
   {
   }
   protected void initRender()
   {
      h = idx3d_Vector.add(scene.light[0].v, scene.defaultCamera.lookat);
      int dx12 = Math.abs(p1.x-p2.x);
      int dx13 = Math.abs(p3.x-p1.x);
      int dx23 = Math.abs(p3.x-p2.x);
      int dy12 = Math.abs(p1.y-p2.y);
      int dy13 = Math.abs(p3.y-p1.y);
      int dy23 = Math.abs(p3.y-p2.y);
      vi13 = new idx3d_Vector((p3.n2.x - p1.n2.x) / dy13,
                        (p3.n2.y - p1.n2.y) / dy13,
                        (p3.n2.z - p1.n2.z) / dy13);
      vi12 = new idx3d_Vector((p2.n2.x - p1.n2.x) / dy12,
                        (p2.n2.y - p1.n2.y) / dy12,
                        (p2.n2.z - p1.n2.z) / dy12);
      vi23 = new idx3d_Vector((p3.n2.x - p2.n2.x) / dy23,
                        (p3.n2.y - p2.n2.y) / dy23,
                        (p3.n2.z - p2.n2.z) / dy23);
      v4 = idx3d_Vector.add(idx3d_Vector.scale(dy12, vi13), p1.n2);
   }
   protected void prerenderFirstPart()
   {
      lv = p1.n2;
      rv = p1.n2;
      if (dx > 0)
      {
         lvi = vi12;
         rvi = vi13;
      }
```

```
        else
        {
            lvi = vi13;
            rvi = vi12;
        }
    }
    protected void prerenderSecondPart()
    {
        if (dx > 0)
        {
            lv = p2.n2;
            rv = v4;
            lvi = vi23;
            rvi = vi13;
        }
        else
        {
            lv = v4;
            rv = p2.n2;
            lvi = vi13;
            rvi = vi23;
        }
    }
    protected void prerenderLine()
    {
        norm();
        renderLine();
        lv = idx3d_Vector.add(lv, lvi); // приращение вектора по горизонтали в лево
        rv = idx3d_Vector.add(rv, rvi); // приращение вектора по горизонтали в
право
    }
     protected idx3d_Vector dN(int n, int k,idx3d_Vector N_p,idx3d_Vector N_g
)//для промежних ненормованых векторів
     {
      int n2=1, k2=1;
      n2<<=n;
      k2<<=k;
      idx3d_Vector Ngp= idx3d_Vector.sub(N_g,N_p);
      return new idx3d_Vector((n2*Ngp.x/k2),(n2*Ngp.y/k2),(n2*Ngp.z/k2));
     }
      protected void norm()
    {
        idx3d_Vector Nbuf12,N12;
        Na=lv; Nb=rv;
        int st=Math.abs(xR - xL);
```

```
      for(n_2=1;n_2<st;)      // пошук числа 2 в степені к
      {
        n_2<<=1;
      }
      if(st!=n_2)
      {              //знаходження кінцевого вектора
        idx3d_Vector buf= idx3d_Vector.sub(Nb,Na);
        N_2=new idx3d_Vector((buf.x*n_2/st),(buf.y*n_2/st),(buf.z*n_2/st));
        N_2.normalize();
        Nb=N_2;
      }
      double cos_W = idx3d_Vector.angle(Na,Nb);
      double z_12=Math.pow((2*(1+cos_W)),0.5);
      Nbuf12 = idx3d_Vector.add(Na,Nb);
      N12= new
idx3d_Vector((float)(Nbuf12.x/z_12),(float)(Nbuf12.y/z_12),(float)(Nbuf12.z/z_12)
);       N12.normalize();//середина
 double di=0;
      double zi=z_12;
      idx3d_Vector Ni=N12;
      Nl=xL;
      count_n=0;
      Ret=new RET[n_2];
      RET Wet= GetSegment( Na, Ni,Nb ,n_2/2,zi,n_2/2); //поиск интервалов
      Ret[count_n]=Wet;
    for(int k=0;k<count_n;k++)
    {
      int min=n_2,point=0;
      for(int i=k;i<=count_n;i++)
      {
        int retx=Ret[i].GetX();
        if(min>retx)
        {
           min=retx;
           point=i;
        }
      }
      RET buf= Ret[k];
      Ret[k]=Ret[point];
      Ret[point]=buf;
    }
    if(count_n>2)
     Ret[count_n+1]= new RET(Nb,n_2);
  }
  public RET GetSegment(idx3d_Vector NA,idx3d_Vector Ni,idx3d_Vector NB ,
```

```
int n_2,double zi,int R)
   {
        double di=1-(Math.pow((2+zi),0.5)/2); //максимальная ошибка
        if(di<0.02 )
        {
         return new RET(Ni,R);
        }
        double zi1 = Math.pow((2+zi),0.5);
        idx3d_Vector  NLi= idx3d_Vector.add(NA,Ni);
        idx3d_Vector  Li=new
idx3d_Vector((float)(NLi.x/zi1),(float)(NLi.y/zi1),(float)(NLi.z/zi1));
        idx3d_Vector  NRi= idx3d_Vector.add(NB,Ni);
        idx3d_Vector  Ri=new
idx3d_Vector((float)(NRi.x/zi1),(float)(NRi.y/zi1),(float)(NRi.z/zi1));
        int dN=n_2/2;
       if(dN>=2)
        {
         Ret[count_n++]= GetSegment(NA, Li,Ni , dN, zi1,R-dN);//влево
         Ret[count_n++]=GetSegment(Ni, Ri,NB , dN, zi1,R+dN);//вправо
        }
     return new RET(Ni,R);
   }
   public String getRasterizerName()
   {
      return "NormalVector";
   }
   protected void renderLine()
   {
      int cdx = Math.abs( xR - xL);
     if (cdx==0)
         return;
      int i=0,k=0,xi=0;
      idx3d_Vector Np=Na,Ng;
      idx3d_Vector dN,Ni;
      Ng=Ret[k].GetVector();
      dN= idx3d_Vector.sub(Ng,Np);
      dN.x*=n_2/cdx;
      dN.y*=n_2/cdx;
      dN.z*=n_2/cdx;
      for (int x=xL; x<xR; x++)
      {
       if(count_n>2)
         if(Ret[k].GetX()<xi )
         {
           Np=Ret[k].GetVector();
```

```
            Ng=Ret[k+1].GetVector();
            i=0;
            k++;
            dN= idx3d_Vector.sub(Ng,Np);
            dN.x*=n_2/cdx;
            dN.y*=n_2/cdx;
            dN.z*=n_2/cdx;
          }
            dN.x*=i;
            dN.y*=i;
            dN.x*=i;
          Ni=idx3d_Vector.add(Np,dN);
          i++;
          xi++;
          pos=x+offset;
          if (z<zBuffer[pos])
          {
              idx3d_Vector cv=Ni;
             float angle = idx3d_Vector.angle(scene.light[0].v, cv);
             if (angle < 0) angle = 0;
             c = idx3d_Color.scale(color, (int)(angle*255));
             angle = idx3d_Vector.angle(h, cv);
             if (angle < 0) angle = 0;
             angle = (float)Math.pow(angle, n);
             int c2 = idx3d_Color.scale(0xFFFFFF, (int)(angle*255));
             c = idx3d_Color.add(c, c2);
             screen.p[pos]=0xFF000000|c;
             zBuffer[pos]=z;
             if (useIdBuffer) idBuffer[pos]=currentId;
          }
          z+=dz;
          nx+=dnx;
          ny+=dny;
       }
    }
    protected void beforeRenderTriangle()
    {
    }
}
```

ДОДАТОК Д

Лістинг модуля GouraudRasterizer4.java

```
public class GouraudRasterizer4 extends idx3d_Rasterizer
{
   private idx3d_Vector h;
   int i1, i2, i3;
   float ii12r, ii12g, ii12b;
   float ii13r, ii13g, ii13b;
   float ii23r, ii23g, ii23b;
   float liir, liig, liib;
   float riir, riig, riib;
   float lir, lig, lib;
   float rir, rig, rib;
   float i4r, i4g, i4b;
   double Pr[];
   double Pg[];
   double Pb[];
   private idx3d_Vector vA, vB, vC,nA, nB, nC ;
   double X1,X2,X3;
   double Y1,Y2,Y3;
   double X12,X23,X13;
   double Y12,Y23,Y13;
   public GouraudRasterizer4()
   {
   }
   protected void initRender()
   {
      h = idx3d_Vector.add(scene.light[0].v, scene.defaultCamera.lookat);
      vA = new idx3d_Vector(p1.n2);
      vB = new idx3d_Vector(p2.n2);
      vC = new idx3d_Vector(p3.n2);
      double cos_1 = idx3d_Vector.angle(vA, h);//  cos(Na H)
      double cos_2 = idx3d_Vector.angle(vB, h);//  cos(Nb H)
      double cos_3 = idx3d_Vector.angle(vC, h);//  cos(Nc H)
      double cos_4 = idx3d_Vector.angle(vA, vB);//  cos(Na Nb)
      double cos_5 = idx3d_Vector.angle(vA, vC);//  cos(Na Nc)
      double cos_6 = idx3d_Vector.angle(vC, vB);//  cos(Nc Nb)
      double t1=(cos_1*cos_4 - cos_2)/((cos_4 - 1)*(cos_2 + cos_1)) ;
      double t2=(cos_1*cos_5 - cos_3)/((cos_5 - 1)*(cos_3 + cos_1)) ;
      double t3=(cos_2*cos_6 - cos_3)/((cos_6 - 1)*(cos_3 + cos_2)) ;
      X1=Math.abs(p1.x+t1*(p2.x-p1.x));
      X2=Math.abs(p1.x+t2*(p3.x-p1.x));
      X3=Math.abs(p2.x+t3*(p3.x-p2.x));
      Y1=Math.abs(p1.y+t1*(p2.y-p1.y));
```

```
        Y2=Math.abs(p1.y+t2*(p3.y-p1.y));
        Y3=Math.abs(p2.y+t3*(p3.y-p2.y));
         ////////////////////////////
        int   v1 = calcColor(p1.n2);
        int   v2 = calcColor(p2.n2);
        int   v3 = calcColor(p3.n2);
         ////////////////////////////
        double Fr[][]={{0,0,0,0,0,1,idx3d_Color.getRed(v1)},
                    {X2*X2,X2*Y2,Y2*Y2,X2,Y2,1,idx3d_Color.getRed(v2)},
        {X2*X2/4,X2*Y2/4,Y2*Y2/4,X2/2,Y2/2,1,(idx3d_Color.getRed(v1)+idx3d_Co
        lor.getRed(v2))/2},{X3*X3,X3*Y3,Y3*Y3,X3,Y3,1,idx3d_Color.getRed(v3)},
        X3*X3/4,X3*Y3/4,Y3*Y3/4,X3/2,Y3/2,1,(idx3d_Color.getRed(v1)+idx3d_Col
        or.getRed(v3))/2},{(X2+X3)*(X2+X3)/4,(X2+X3)*(Y2+Y3)/4,(Y2+Y3)*(Y2+
        Y3)/4,(X2+X3)/2,(Y2+Y3)/2,1,(idx3d_Color.getRed(v2)+idx3d_Color.getRed(v
        3))/2}
        };
        Kramer sist= new Kramer();
        Pr=new double[6];
        Pr= sist.GetKramer(Fr,6,7);
        double Fg[][]={{0,0,0,0,0,1,idx3d_Color.getGreen(v1)},
                    {X2*X2,X2*Y2,Y2*Y2,X2,Y2,1,idx3d_Color.getGreen(v2)},
        {X2*X2/4,X2*Y2/4,Y2*Y2/4,X2/2,Y2/2,1,(idx3d_Color.getGreen(v1)+idx3d_
        Color.getGreen(v2))/2},{X3*X3,X3*Y3,Y3*Y3,X3,Y3,1,idx3d_Color.getGree
        n(v3)},
        {X3*X3/4,X3*Y3/4,Y3*Y3/4,X3/2,Y3/2,1,(idx3d_Color.getGreen(v1)+idx3d_
        Color.getGreen(v3))/2},
        {(X2+X3)*(X2+X3)/4,(X2+X3)*(Y2+Y3)/4,(Y2+Y3)*(Y2+Y3)/4,(X2+X3)/2,(
        Y2+Y3)/2,1,(idx3d_Color.getGreen(v2)+idx3d_Color.getGreen(v3))/2}
        };
        Pg=new double[6];
        Pg= sist.GetKramer(Fg,6,7);
        double Fb[][]={{0,0,0,0,0,1,idx3d_Color.getBlue(v1)},
                    {X2*X2,X2*Y2,Y2*Y2,X2,Y2,1,idx3d_Color.getBlue(v2)},
        {X2*X2/4,X2*Y2/4,Y2*Y2/4,X2/2,Y2/2,1,(idx3d_Color.getBlue(v1)+idx3d_C
        olor.getBlue(v2))/2},
                    {X3*X3,X3*Y3,Y3*Y3,X3,Y3,1,idx3d_Color.getBlue(v3)},
    {X3*X3/4,X3*Y3/4,Y3*Y3/4,X3/2,Y3/2,1,(idx3d_Color.getBlue(v1)+idx3d_Color.
        getBlue(v3))/2},
        {(X2+X3)*(X2+X3)/4,(X2+X3)*(Y2+Y3)/4,(Y2+Y3)*(Y2+Y3)/4,(X2+X3)/2,(
        Y2+Y3)/2,1,(idx3d_Color.getBlue(v2)+idx3d_Color.getBlue(v3))/2}
        };
        Pb=new double[6];
        Pb= sist.GetKramer(Fb,6,7);
      protected void prerenderFirstPart()
      {
```

```
}
protected void prerenderSecondPart()
{
}
protected void prerenderLine()
{
   renderLine();
}
public String getRasterizerName()
{
   return " --------- ";
}
protected void renderLine()
{
   int cdx = xR - xL;
   if (cdx == 0) return;
   for (x=xL;x<xR;x++)
   {

      double Ixyr=Pr[5]*x*x+Pr[4]*x*y+Pr[3]*y*y+Pr[2]*x+Pr[1]*y+Pr[0];
      double Ixyg=Pg[5]*x*x+Pg[4]*x*y+Pg[3]*y*y+Pg[2]*x+Pg[1]*x+Pg[0];
      double Ixyb=Pb[5]*x*x+Pb[4]*x*y+Pb[3]*y*y+Pb[2]*x+Pb[1]*y+Pb[0];

    float   cir = (float)Ixyr;
    float   cig = (float)Ixyg;
    float   cib = (float)Ixyb;

      if(cir<0){ cir=0;}
      if(cig<0){ cig=0;}
      if(cib<0){ cib=0;}
      pos=x+offset;
      if (z<zBuffer[pos])
      {
         screen.p[pos]=0xFF000000 | idx3d_Color.getColor(Math.round(cir > 255 ?
   255 : cir), Math.round(cig > 255 ? 255 : cig), Math.round(cib > 255 ? 255: cib));
         zBuffer[pos]=z;
         if (useIdBuffer) idBuffer[pos]=currentId;
      }
   }
}
private int calcColor(idx3d_Vector vector)
{
   float angle = idx3d_Vector.angle(scene.light[0].v, vector);
   if (angle < 0) angle = 0;
   int c = idx3d_Color.scale(color, (int)(angle*255));
```

```
        angle = idx3d_Vector.angle(h, vector);
        if (angle < 0) angle = 0;
        angle = (float)Math.pow(angle, n);
        int c2 = idx3d_Color.scale(0xFFFFFF, (int)(angle*255));
        c = idx3d_Color.add(c, c2);
        return c;
    }
    protected void beforeRenderTriangle()
    {
    }
}
```

ДОДАТОК Е

Графічна частина

Високопродуктивно програмно-апаратні засоби для рендерингу графічних сцен

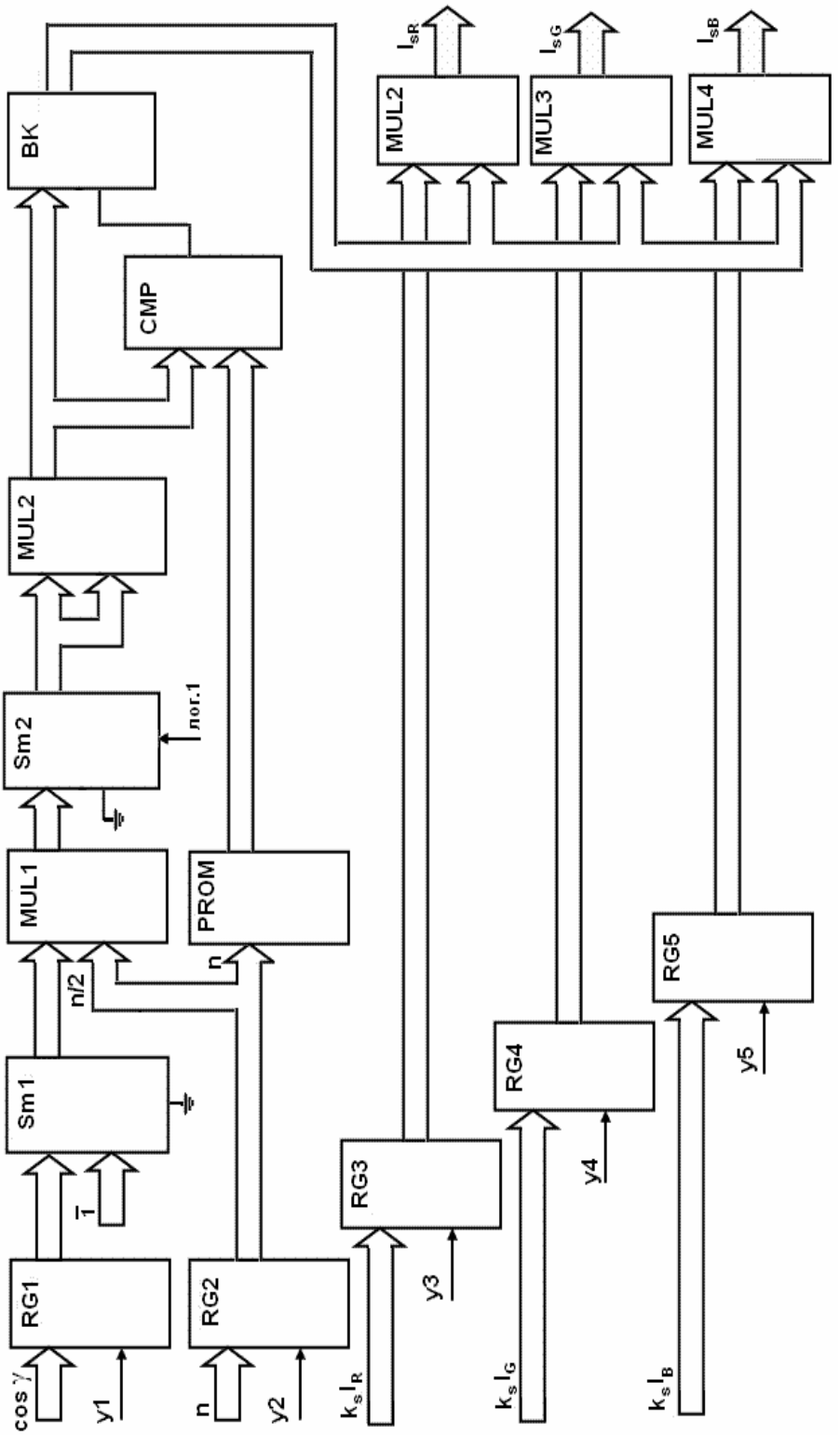


Рис. 3.6 Структурна схема блоку дли визначення спекулярної складової кольору з використанням ДФВЗ типу $n/2(\cos\gamma-1)+1)^2$