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

Бакалаврська дипломна робота

на тему: «Програмні засоби для надання об'ємності зображенням»

Виконав: студент IV курсу
групи 2ПІ-20б (д/н) спеціальності
121 - «Інженерія програмного забезпечення»
Білий Максим Юрійович
(прізвище та ініціали)
Керівник: зав. кафедри ПЗ, проф., д.т.н, Романюк О.Н.
(прізвище та ініціали)
Рецензент: проф. кафедри ОТ, д.т.н., Мартинюк Т.Б.

Допущено до захисту
Завідувач кафедри ПЗ
д.т.н., проф. О.Н. Романюк
(прізвище та ініціали)
«10» червня 2024 р.

ВНТУ – 2024

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

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
д.т.н., проф. О.Н. Романюк
«12» __березня__ 2024 р.

З А В Д А Н Н Я
НА БАКАЛАВРСЬКУ ДИПЛОМНУ РОБОТУ СТУДЕНТУ
Білого Максима Юрійовича

1. Тема роботи – «Програмні засоби для надання об'ємності зображенням». Керівник роботи: Романюк Олександр Никифорович, д.т.н., завідувач кафедри ПЗ, затверджені наказом вищого навчального закладу від «11» _березня_ 2024 р. №80.

2. Строк подання студентом роботи «_3_» ___червня__ 2024 р.

3. Вихідні дані до роботи: метод зафарбовування – метод ray-tracing, метод Ламберта, метод задання поверхні об'єкту – полігональна модель; мова програмування – Kotlin; середовище розробки – Android Studio; вихідні дані: об'ємне зображення.

4. Зміст розрахунково-пояснювальної записки: вступ; аналіз методів і засобів генерації кросвордів; розробка методів та моделей системи і алгоритмів програмного продукту; розробка програмних модулів реалізації програмних засобів для генерації кросвордів; тестування програми; висновки; список використаних джерел; додатки.

5. Перелік ілюстративного матеріалу: галузі застосування графічних засобів; основні етапи графічного конвеєру; адаптивне зафарбування; нові моделі відбивних здатностей поверхонь; інтерфейс 3D-редактора світлових ефектів.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв
1-4	Романюк О.Н., д.т.н., професор каф. ПЗ	13.03.24	03.06.24

7. Дата видачі завдання 13 березня 2024 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської дипломної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз розвитку мобільних застосунків для надавання об'ємності зображенням	14.03.24 – 17.03.24	Вик.
2	Розробка методів й алгоритмів підвищення продуктивності надавання об'ємності зображень	17.03.24 – 21.04.24	Вик.
3	Розробка програмних модулів	21.04.24 – 10.05.24	Вик.
4	Тестування програмного забезпечення	10.05.24 – 21.05.24	Вик.
5	Оформлення матеріалів до захисту БДР	21.05.24 – 30.05.24	Вик.

Студент _____ **М. Ю. Білий**
(підпис) (ініціали та прізвище)

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

АНОТАЦІЯ

Бакалаврська дипломна робота складається з 64 сторінки формату А4, на яких є 27 рисунків, 3 таблиць, список використаних джерел містить 20 найменувань.

У бакалаврській дипломній роботі проведено детальний аналіз методів і засобів надання об’ємності зображенням. Сформульовано мету досліджень – підвищення якості надання об’ємності зображенням за рахунок новітніх бібліотек, зменшення обчислювальної складності для рахування відблисків на поверхні фігури.

Запропоновано модифікований метод ray-tracing для рахування відблисків на поверхні моделі, у якому вдвічі зменшуючи ступінь артефактів та спотворень можна значно покращити реалістичність відблисків на поверхні моделі. Додатково, модифікований метод може використовувати дані про матеріал поверхні для більш точного розрахунку відблисків, що дозволяє краще відтворити їх реалістичний вигляд на поверхні моделі. Запропоновано метод розбиття поверхні моделі на трикутники для ідентифікації відблисків, у якому полегшується розрахунок відблисків, обмежуючи їх обчислення лише до тих трикутників, де вони дійсно присутні, тим самим економлячи обчислювальні ресурси. Такий підхід дозволяє підвищити швидкість рендерингу і зменшити навантаження на обчислювальний пристрій.

Розроблено алгоритми для засобів моделювання ефектів об’ємності у засобах комп’ютерної графіки, а також програмне забезпечення на основі цих алгоритмів.

Отримані в бакалаврській дипломній роботі результати можна використати для якісної обробки та надання об’ємності зображенням.

Ключові слова: рей-трейсинг, 3D-візуалізація, надання об’ємності 2D зображенням, рендеринг.

ABSTRACT

The bachelor's thesis consists of 64 A4 pages, which have 27 figures, 3 tables, and the list of sources used contains 20 items.

The bachelor's thesis includes a detailed analysis of methods and means for providing volume to images. The research goal is formulated as improving the quality of providing volume to images through modern libraries, reducing computational complexity for calculating reflections on the surface of a figure.

A modified ray-tracing method is proposed for calculating reflections on the surface of a model, which by halving the degree of artifacts and distortions can significantly improve the realism of reflections on the surface of the model. Additionally, the modified method can use surface material data for more accurate reflection calculations, allowing for a better reproduction of their realistic appearance on the model's surface. A method of dividing the model's surface into triangles is proposed for identifying reflections, which simplifies the calculation of reflections by limiting their calculation only to those triangles where they are actually present, thereby saving computational resources. This approach allows for increased rendering speed and reduced load on the computational device.

Algorithms have been developed for tools modeling volume effects in computer graphics, as well as software based on these algorithms.

The results obtained in the bachelor's thesis can be used for high-quality processing and providing volume to images.

Keywords: ray-tracing, 3D visualization, providing volume to 2D images, rendering.

ЗМІСТ

ВСТУП

Обґрунтування вибору теми дослідження. У сучасному світі тривимірні зображення стали невід'ємною частиною нашого щоденного життя. Технології, які дозволяють надавати об'ємності зображенням, стали дуже популярними та використовуються в різних сферах, від розваг та мистецтва до науки та бізнесу. Об'ємні зображення можуть створювати вражаючі візуальні ефекти, які привертають увагу глядачів та дозволяють зображенням виходити за межі звичайних плоских площин.

Створення об'ємних ефектів на зображеннях є складним процесом, який вимагає високотехнологічних рішень та творчого підходу. Технології, які дозволяють надавати об'ємності зображенням, постійно розвиваються і удосконалюються, що відкриває нові можливості для використання їх у різних галузях. Наприклад, у сфері реклами та маркетингу об'ємні зображення можуть привертати більше уваги та створювати більш сильний емоційний вплив на аудиторію. У сфері мистецтва об'ємні зображення можуть бути використані для створення унікальних творів, які вражають своєю красою та оригінальністю.

Також важливою галуззю застосування об'ємних зображень є наука та освіта. Вони можуть бути використані для створення інтерактивних навчальних матеріалів, які допомагають студентам краще розуміти складні концепції та явища. Крім того, об'ємні зображення можуть бути використані у медицині для візуалізації медичних даних та діагностики хвороб.

У цьому контексті розробка мобільного застосунку для надання об'ємності 2D зображенням є важливим завданням, яке може принести користь у багатьох сферах життя. Такий застосунок може стати корисним інструментом для творців контенту, митців, дизайнерів, викладачів та багатьох інших людей, які цінують креативність та новаторство.

З розвитком мобільних пристроїв та зростаючим попитом на інноваційні технології популярною тенденцією останніх років стала розробка мобільних систем з для редагування та надання об’ємності зображенням. Це зумовило

потребу в програмних модулях, які можуть бути використані для реалізації таких систем. Ці програмні модулі можуть допомогти розробникам створювати мобільні системи, за допомогою яких можна з телефона в зручному доступі оброблять зображення.

Наявні аналогічні мобільні застосунки для обробки зображень не надають повний пакет інструментів для надання об’ємності зображенням. До того ж, зазвичай потужні засоби є платними, тому актуальною є розробка власного мобільного застосунку для надання об'ємності 2D зображенням.

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

Мета та завдання дослідження. Метою роботи є підвищення продуктивності формування тривимірних зображенням за рахунок розробки нових методів, використання сучасних бібліотек.

У бакалаврській дипломній роботі необхідно виконати такі завдання:

- провести аналіз існуючих методів і засобів надання об’ємності зображенням;
- запропонувати нові:
    - модифікований метод ray-tracing для рахування відблисків на поверхні моделі;
    - метод розбиття поверхні моделі на трикутники для ідентифікації відблисків;
- розробка алгоритму надання об’ємності зображенням;
- розробка зручного та адаптивного графічного інтерфейсу мобільної системи який допомагатиме користувачу зручно обробляти зображення;
- тестування мобільної системи згідно поставлених задач.

Об’єкт дослідження – процес надання об’ємності зображенням у системах комп’ютерної графіки.

Предмет дослідження – методи та засоби надання об’ємності зображенням.

Методи дослідження. У процесі досліджень використовувались: теорія математичного аналізу та чисельних методів, теорія чисел, методи аналітичної геометрії для розробки моделей та методів надання об’ємності зображенням, комп’ютерне моделювання для аналізу та перевірки отриманих теоретичних положень..

Наукова новизна отриманих результатів.

1. Запропоновано модифікований метод ray-tracing, який відрізняється від відомих аналізом типів матеріалів для вилучення надлишкових обчислень, що дає можливість підвищити продуктивність формування тривимірних графічних сцен за рахунок комбінування методів зафарбовування.
2. Подальшого розвитку отримав метод зафарбовування тривимірних об’єктів, який відрізняється від відомих попереднім аналізом наявності відблисків у межах трикутників і використання спрощених обчислень за їх відсутністю. Це дає можливість підвищити продуктивність рендеренгу зображень [1].

Практична цінність отриманих результатів. Практична цінність одержаних результатів полягає в тому, що на основі отриманих в бакалаврській дипломній роботі теоретичних положень запропоновано алгоритми та розроблено програмні засоби для надання об’ємності зображенням.

У першому розділі бакалаврської дипломної роботи було проведено порівняльний аналіз аналогів, що дозволило визначити переваги та недоліки існуючих програмних засобів для надання об’ємності зображенням, а також визначити ключові вимоги для подальшого створення програмного застосунку. Важливо відмітити, що аналіз аналогів показав наскільки важливим є швидкість розрахунків поверхні фігури та надання об’ємності зображення, що в свою чергу зменшує навантаження на обчислювані апарати.

У другому розділі бакалаврської дипломної роботи було детально розглянуто процес створення програмних методів для надання об’ємності зображенням. Було описано ключові моменти для створення та покращення методів поліпшення зображень, а саме покращено метод ray-tracing для

рахування відблисків на поверхні моделі, додана перевірка поверхні фігури на матеріал, що дозволить спростити обрахунки уникаючи поверхонь, що мають матову поверхність, тобто не відбивають світло, або відбиття мінімальне. Також було описана розробка методу розбиття поверхні моделі на трикутники для ідентифікації відблисків, що в свою чергу піліпшує розрахунки методом розбиття фігури на трикутники, і в вже аналізуючи кожен трикутник окремо на наявність блику в ньому, проводить обрахунки. Цей розділ надає глибоке розуміння у процес створення програмного забезпечення для надання об’ємності зображенням.

У третьому розділі бакалаврської дипломної роботи було обґрунтовано вибір засобів для реалізації програмного забезпечення. В результаті даного аналізу було розроблено модуль надання об’ємності зображень, що в свою чергу містить функціонал власних розроблених методів аналізу поверхні фігури та розбиття поверхні на трикутники. Також було описано та створено модуль завантаження обробленого зображення, котрий дозволяє завантажувати змінені зображення в будь яких доступних форматах, для подальшої праці з ними. Також у третьому розділі було досліджено та розроблено інтерфейс мобільного додатку для надання ою’ємності зображенням, для зручного використання користувачами. Був створений загальний алгоритм роботи додатку, також була створена діаграма діяльності мобільної системи, де в свою чергу описано покрокову роботу програмного застосунка для надання об’ємності зображенням.

У четвертому розділі бакалаврської дипломної роботи було проведено тестування програмних модулів програмного застосунку для надання об’ємності зображенням. В свою чергу було перевірено додаток на швидкодію та ефективність обробки зображень. Після всіх тестувань модулів та аналізу роботи програмного застосунку відповідно дійсності, було розроблено інструкцію користувача, де описано всі кроки та питання які можуть виникнути у користувачів під час використання даного програмного застосунку для надання лб’ємності зображенням.

Особистий внесок здобувача. Усі наукові результати, викладені у бакалаврській дипломній роботі, отримані автором особисто. У друкованій праці [4], автору належить концептуальні положення комбінованого рендерингу опублікованих у співавторстві, автору належать такі результати: аналіз аналогів розроблюваного програмного забезпечення, опис та реалізація модифікованого методу ray-tracing для рахування відблисків на поверхні моделі, опис та реалізація методу розбиття поверхні моделі на трикутники, опис та реалізація алгоритмів надання обємності зображенням, аналіз методів вирішення поставленої задачі.

Апробація матеріалів бакалаврської дипломної роботи. Основні положення бакалаврської дипломної роботи доповідалися та обговорювалися на міжнародних та всеукраїнських конференціях: LIII Всеукраїнська науково-технічна конференція підрозділів Вінницького національного технічного університету (2024) і опубліковані в тезах доповіді цієї конференції[4].

Публікації. Основні результати досліджень опубліковано в 1 тезі у матеріалах конференції LIII Всеукраїнська науково-технічна конференція підрозділів Вінницького національного технічного університету (2024).

Структура та обсяг БДР. БДР складається зі вступу, чотирьох розділів, висновків, списку літератури, що містить 20 найменувань, 3 додатки. Робота містить 27 ілюстрації, 3 таблиці. Загальний обсяг роботи складає 97 сторінок, основний зміст викладено на 64 сторінках друкованого тексту.

1 АНАЛІЗ РОЗВИТКУ МОБІЛЬНИХ ЗАСТОСУНКІВ ДЛЯ НАДАВАННЯ ОБ'ЄМНОСТІ ЗОБРАЖЕННЯМ ТА ПОСТАНОВКА ЗАДАЧ ДОСЛІДЖЕННЯ

1.1 Аналіз методів для надавання об'ємності зображенням

3D-візуалізація є комплексним процесом, спрямованим на створення віртуальних зображень об'єктів та сцен у тривимірному просторір [2]. Цей метод графічного подання дозволяє користувачам сприймати об'єкти так, як вони виглядають у реальному світі. У сучасному контексті, 3D-візуалізація є невід'ємною частиною багатьох галузей, включаючи відеоігри, кіноіндустрію, дизайн продуктів, архітектуру та багато інших. Вона використовується для створення реалістичних та деталізованих зображень та анімації, що дозволяє створювати вражаючі візуальні ефекти та віртуальні світи.

3D-візуалізація здійснюється за допомогою комп'ютерної графіки та високотехнологічних програмних засобів. Вона включає в себе такі процеси, як моделювання, текстурування, освітлення, анімацію та рендеринг. Моделювання включає створення тривимірних об'єктів і сцен, текстурування - додавання текстур до цих об'єктів для надання їм реалістичного вигляду, освітлення - налаштування освітлення в сцені для створення правдоподібних тіней та відблисків, анімація - створення рухомих об'єктів та сцен, а рендеринг - перетворення тривимірної сцени у двовимірне зображення.

У сфері 3D-візуалізації присутній комплекс специфічних термінів і понять, які визначають елементарні аспекти цього технічного дисциплінарного підходу. До них відносяться терміни, такі як модель, текстура, освітлення, анімація, рендеринг, а також поняття, такі як тріангуляція, затінення, камера, матеріал, сцена та багато інших.

Загалом, 3D-візуалізація відкриває множину можливостей для творчості та розвитку у різних галузях, дозволяючи створювати надзвичайно реалістичні та вражаючі візуальні ефекти.

Моделювання (Modeling): Це ключовий етап, який передбачає створення тривимірних об'єктів з урахуванням їх геометричної форми та взаємодії між ними [3]. Моделювання включає в себе використання математичних алгоритмів для визначення координат точок та їх взаємного розташування в просторі.

Текстурування (Texturing) [5]: Термін, що визначає процес накладання текстур на поверхні моделей з метою надання їм візуального реалізму. Текстури надають об'єктам характер, враховуючи деталізацію та колірну інформацію.

Освітлення (Lighting) [6]: Це поняття об'єднує методи взаємодії світла та поверхонь об'єктів. Освітлення впливає на сприйняття форми об'єктів, забезпечуючи змогу реалістично відображати тіні, відбиття та інші оптичні ефекти.

Рендеринг (Rendering) [7]: Рендеринг є процесом генерації двовимірного зображення з тривимірної сцени за допомогою врахування всіх характеристик об'єктів та середовища, таких як кольори, тіні, освітлення та перспектива.

Ці концепції становлять фундаментальний ланцюг в 3D-графіці, об'єднуючи технічні аспекти та творчий підхід для досягнення високого рівня візуальної реалістичності у віртуальних просторах. Інтегрування цих термінів у процес 3Dвізуалізації визначає та розширює високотехнологічний характер цієї галузі, забезпечуючи можливість адекватного відтворення об'єктів та сцен у тривимірному просторі.

У галузі 3D-візуалізації відбулися революційні досягнення, що дозволили значно покращити реалістичність та ефективність створення тривимірних зображень. Ці досягнення зазначено у період від 1980-х до 2000-х років і визначили нові стандарти в індустрії комп'ютерної графіки та 3D-візуалізації.

Рей-трейсинг. Введення рей-трейсингу в графічні обчислення представляє собою значний прорив. Рей-трейсинг дозволяє симулювати поведінку світла в найреалістичніший спосіб, що в результаті призвело до створення фото-реалістичних зображень. Він є важливим алгоритмом у галузі комп'ютерної графіки, який дозволяє створювати фото-реалістичні зображення, моделюючи поведінку світла та взаємодію з об'єктами в сцені. Основні концепції

рей-трейсингу включають в себе відбиття світла, тіні, відблиски та розсіювання світла.

Основні аспекти рей-трейсингу:

1. Відбиття світла [8]: Природнім чином об'єкти відбивають світло. Алгоритм рей-трейсингу включає в себе відправлення променів від камери до сцени, а коли промінь досягає поверхні об'єкта, відбувається відбиття, і розраховується колір відбитого світла (рисунок 1.1).

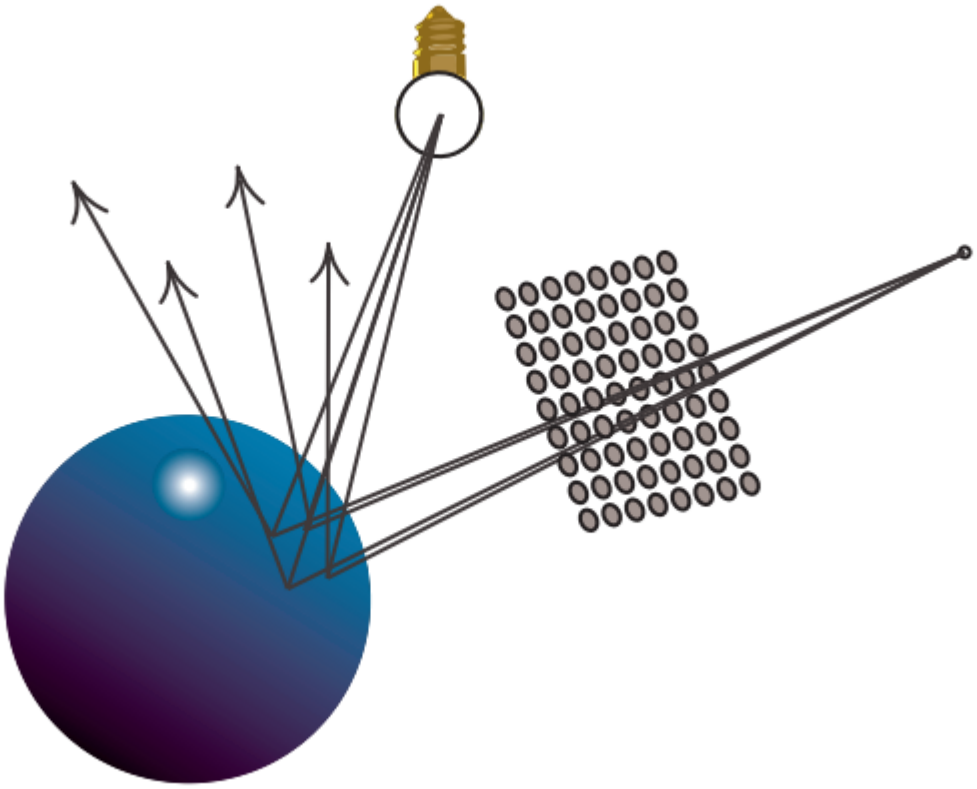


Рисунок 1.1 – Відбивання променів світла від поверхні кулі

2. Променева тінь [9]: Рей-трейсинг дозволяє визначити, чи освітлена певна точка сцени чи вона перебуває в тіні. Це досягається шляхом висилання додаткових променів від точок освітлення до точок на поверхні об'єктів (рис. 1.2).

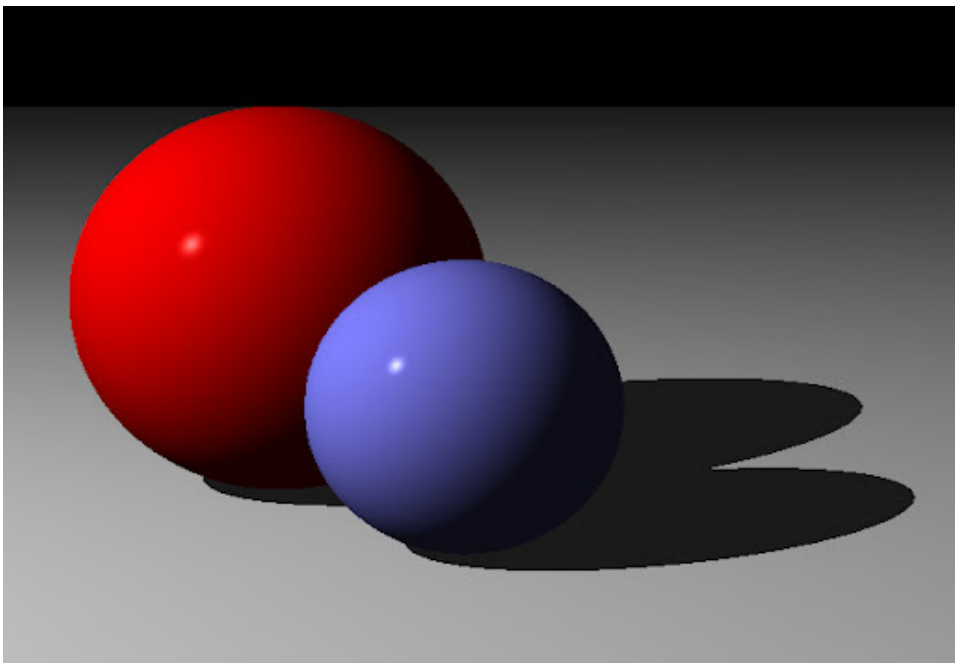


Рисунок 1.2 – Утворення тіней при потраплянні променя світла

3. Відблиски [10]: Рей-трейсинг враховує відблиски, які виникають при відбитті світла від гладких поверхонь. Це додає до зображення реалістичність, оскільки об'єкти можуть відображати оточуючі об'єкти (рисунок 1.3).

Рисунок 1.3 – Відблиски, що відображають оточуючі об'єкти

4. Розсіювання світла [11]: Цей аспект враховує розсіювання світла, коли світло розсіюється в різних напрямках при взаємодії з матеріалом. Це особливо важливо для матеріалів з матовою поверхнею (рисунок 1.4).

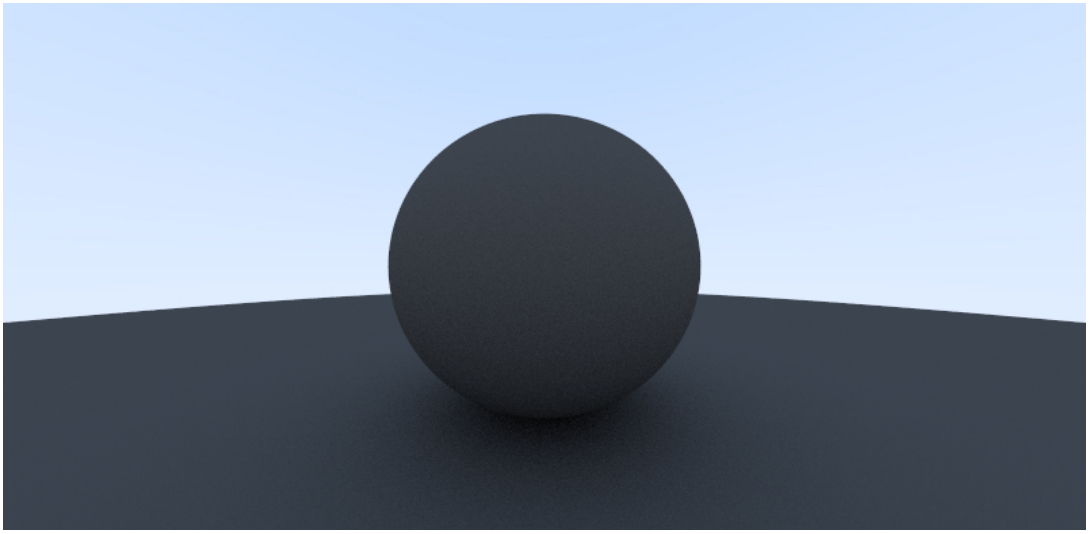


Рисунок 1.4 – Розсіювання світла на матовій поверхні

У висновку можна сказати, що рейтрейсинг є потужною технікою рендерингу, яка дозволяє створювати зображення з високим рівнем реалізму. Використання цієї техніки дає змогу моделювати складні ефекти освітлення, тіней та відображень, що робить її незамінною у візуалізації 3D сцен і створенні комп'ютерної графіки високої якості. Однак, через високі обчислювальні витрати, рей-трейсинг залишається вимогою до обладнання і часу, що обмежує його застосування у реальному часі для багатьох додатків.

Мобільні застосунки для надання об'ємності зображенням є досить популярними і мають широкий спектр застосувань у різних галузях, включаючи рекламу, маркетинг, мистецтво, освіту та розваги. Деякі з цих застосунків надають користувачам можливість створювати об'ємні ефекти на зображеннях за допомогою доповненої реальності (AR), технологій віртуальної реальності (VR) та інших інноваційних методів [12].

Наприклад, деякі мобільні застосунки дозволяють користувачам створювати 3D об'єкти на основі зображень або накладати віртуальні об'єкти на

реальний світ за допомогою AR. Це дозволяє створювати цікаві та захоплюючі візуальні ефекти, які можна використовувати у маркетингових кампаніях, рекламних матеріалах або для творчих проєктів.

Деякі застосунки для обробки та надання об'ємності зображенням надають користувачам широкий спектр інструментів для редагування та покращення зображень. Ці інструменти можуть включати функції корекції кольору, застосування фільтрів, виправлення контрастності та яскравості, а також додавання спеціальних ефектів та текстур. Використання цих інструментів дозволяє створювати вражаючі та професійні зображення без необхідності великих знань у галузі фотографії та дизайну.

Однак, деякі мобільні застосунки можуть мати обмежені можливості або бути нестабільними через високі вимоги до обладнання та ресурсів пристрою. Також, вартість деяких застосунків та їх доступність для різних платформ може бути проблемою для деяких користувачів.

У цілому, мобільні застосунки для надання об'ємності зображенням є важливим інструментом для творчих та професійних користувачів, які шукають шляхи для створення унікальних та вражаючих візуальних ефектів. Ці застосунки дозволяють користувачам експериментувати з різними техніками обробки зображень та надавати їм об'ємності, що робить їх привабливими для широкого кола аудиторії.

Отже, розробка програмних модулів для впровадження мобільного застосунку для надання об’ємності зображенням є критично важливим завданням у сучасному світі.

1.2 Аналіз засобів рендеренгу

Використання ефектів для надання об'ємності зображенням є важливим етапом у редагуванні фотографій та створенні вражаючих візуальних ефектів. Ці ефекти дозволяють створювати ілюзію глибини та простору на плоскому зображенні, що додає йому реалістичності та емоційного зворушення. Використання ефектів надання об'ємності є особливо актуальним у сучасному

світі, де візуальні зображення грають важливу роль у спілкуванні та враженні аудиторії. Такі ефекти дозволяють створювати унікальні та запам'ятовувані зображення, які виокремлюються серед інших та залишають глибоке враження на глядачів. Існує безліч програмних застосунків для надання об’ємності зображенням. Ось деякі з них:

- Photoshop Express;
- Snapseed;
- Pixlr.

Одним із прикладів обробки зображень в мобільній системі є «Photoshop Express» [13] (рисунок 1.1). Це мобільний застосунок для редагування фотографій, розроблений компанією Adobe. Він надає користувачам широкий спектр інструментів для покращення та творчого редагування зображень простим і доступним способом. Завдяки своїй потужній функціональності та легкому використанню, «Photoshop Express» є популярним вибором серед користувачів, які шукають зручний спосіб редагувати свої фотографії на мобільних пристроях.

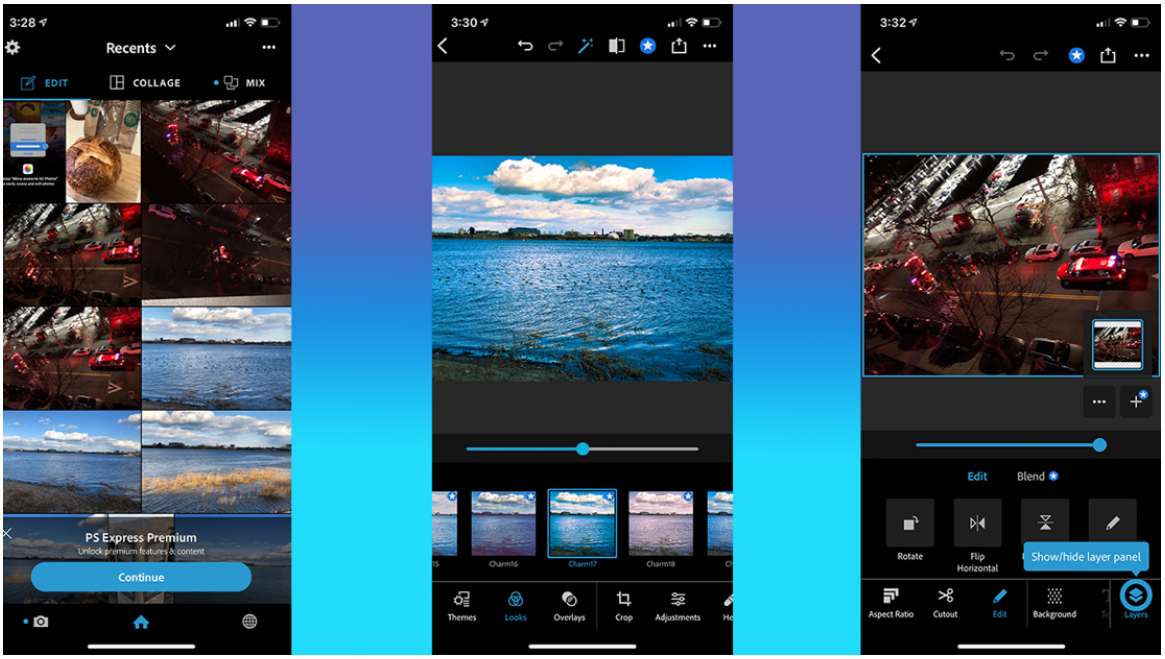


Рисунок 1.1 – Приклад роботи «Photoshop Express»

Інший приклад – додаток «Snapseed» [14] (рисунок 1.2). Це мобільний застосунок для редагування фотографій, розроблений компанією Google. Застосунок надає широкий спектр інструментів для редагування зображень, включаючи різноманітні фільтри, ефекти, корекцію кольору та інші можливості. «Snapseed» відзначається простим та зручним інтерфейсом, що робить його популярним вибором серед користувачів, які шукають потужний інструмент для редагування фотографій на своїх мобільних пристроях.

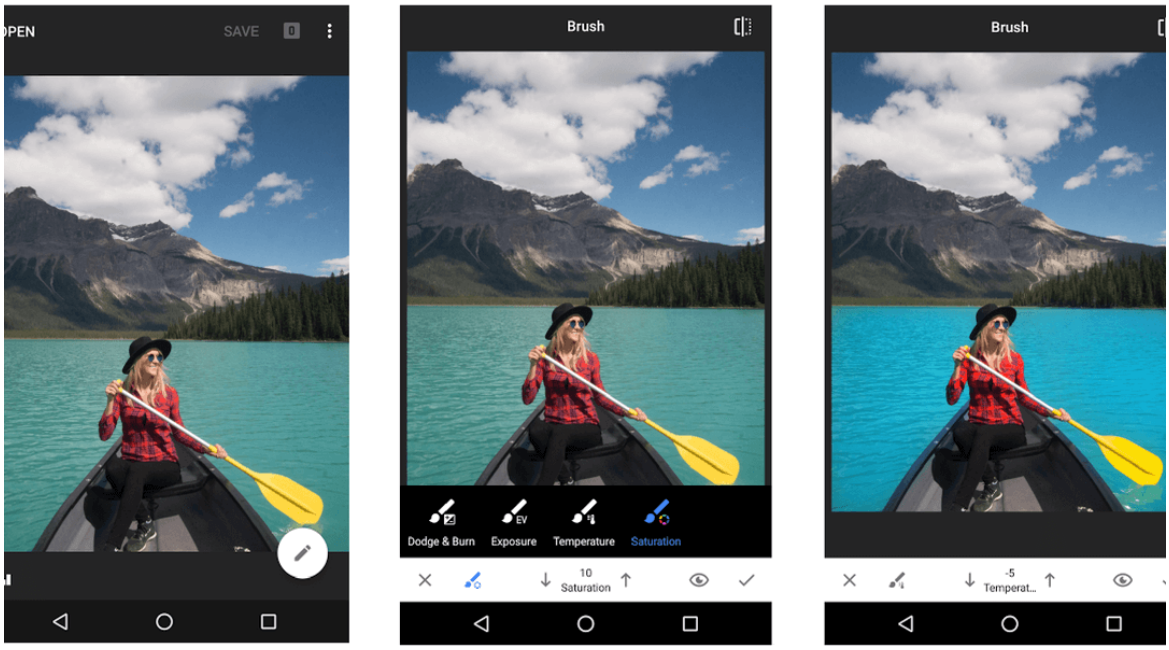


Рисунок 1.2 – Приклад роботи додатку «Snapseed»

Існує ще один приклад – додаток «Pixlr» [15] (рисунок 1.3). Це онлайн сервіс для редагування та обробки зображень, який також має мобільну версію для смартфонів і планшетів. Він надає користувачам можливість редагувати фотографії безпосередньо у веб-браузері або на мобільному пристрої, не потребуючи встановлення спеціального програмного забезпечення. «Pixlr» має великий набір інструментів для редагування, включаючи фільтри, ефекти, інструменти для ретуші та корекції кольору, що робить його популярним серед користувачів, які шукають зручний спосіб обробки зображень.

Рисунок 1.3 – Приклад роботи додатку «Pixlr»

Розробка програмних модулів для впровадження мобільного застосунку для надання об’ємності зображенням вимагає досвіду обробці зображень, розробки програмного забезпечення та розробки користувацького інтерфейсу для мобільних додатків. Це передбачає створення власного функціоналу та інтегрування бібліотек для надання ефектів об’ємності

Результати порівняння аналогів зведено в таблиці 1.1.

Таблиця 1.1 – Порівняльні характеристика онлайн платформ

Критерії	Photoshop Express	Snapseed	Pixlr	Власний модуль
AR-ефекти та фільтри	+	+	+	+
Зручний інтерфейс для обробки	+	-	+	+
Достатній спектр інструментів для обробки	-	+	-	+
Сумісність з декількома пристроями	-	-	+	+
Наявність мобільного додатку	+	+	+	+
Загальна оцінка	60%	60%	80%	100%

Відповідно до таблиці порівняльних характеристик розробка власного застосунку для надання об’ємності зображенням. Отриманий продукт зможе покрити недоліки існуючих рішень та забезпечити новий досвід та можливості застосування ефектів для надання об’ємності зображенням.

Отже, розробка програмних модулів для впровадження мобільного застосунку для надання об’ємності зображенням відкриває перед бізнесом широкі можливості для взаємодії зі своїми клієнтами, створення унікального контенту, а також підвищення рівня залученості та утримання користувачів.

1.3 Порівняльний аналіз аналогів

Розробка програмних модулів для реалізації мобільного застосунку для надання об’ємності зображенням вимагає ретельного розгляду найкращих підходів до вирішення проблеми. Аналіз методів розв'язання поставленої задачі, що полягає у наданні об'ємності зображенням за допомогою мобільного застосунку, включає огляд існуючих підходів та технологій, які можуть бути використані для досягнення цієї мети. Основні методи включають використання технологій доповненої та віртуальної реальності, обробку зображень, застосування спеціальних ефектів та фільтрів.

Перший метод, що варто розглянути, – це використання технології доповненої реальності (AR). AR дозволяє накладати віртуальні об'єкти на реальний світ через камеру мобільного пристрою. За допомогою AR можна створювати об'ємні ефекти на 2D зображеннях, що додає їм реалістичності та привабливості. Наприклад, користувач може розгорнути 2D зображення книги у 3D модель, яка буде виглядати як справжня книга на столі. AR також може бути використаний для створення інтерактивних маркерів на зображеннях, що дозволить користувачам взаємодіяти з ними за допомогою своїх мобільних пристроїв.

Другий метод - використання технології віртуальної реальності (VR). VR дозволяє користувачам зануритися в інші виміри та взаємодіяти з об'ємними об'єктами у віртуальному середовищі. Хоча цей підхід більш складний у реалізації та може вимагати спеціального обладнання, він може забезпечити більш глибокий та іммерсивний досвід для користувачів. Наприклад, можна створити додаток, який дозволяє користувачам переглядати та взаємодіяти з об'ємними моделями архітектурних споруд у віртуальному просторі. VR також може бути використаний для створення тренувальних симуляторів для різних професій, де реалістична візуалізація грає важливу роль.

Третій метод - обробка зображень та застосування спеціальних ефектів та фільтрів. Цей підхід включає в себе використання алгоритмів обробки зображень для створення ефектів глибини, тіней та освітлення, що додають об'ємності

зображенням. Наприклад, можна реалізувати функцію додавання ефекту "паралаксу" до зображень, щоб вони виглядали більш об'ємними та динамічними. Також можна використовувати різноманітні фільтри та ефекти для створення вражаючих візуальних елементів.

Кожен з цих методів має свої переваги та обмеження. Вибір конкретного методу може залежати від конкретних потреб та можливостей програмного застосунку. Розглядаючи ці методи, можна знайти оптимальний шлях для реалізації задачі та створення мобільного застосунку, що надає об'ємність зображенням.

Додавши об'ємність зображенням за допомогою технологій доповненої та віртуальної реальності (AR та VR), можна створити унікальний досвід для користувачів. Ці технології дозволяють створювати об'ємні ефекти, які додають реалістичності та привабливості зображенням.

AR може бути особливо ефективним для створення об'ємних ефектів на 2D зображеннях. Наприклад, можна створити додаток, який дозволяє користувачам переглядати інтерактивні об'єкти на зображеннях у реальному часі через камеру свого смартфона. Це може бути корисно для віртуального огляду продуктів або мистецьких творів перед покупкою.

VR, з іншого боку, може забезпечити більш глибокий та цікавий досвід, де користувачі можуть взаємодіяти з об'ємними об'єктами у віртуальному середовищі. Це може бути особливо ефективно для тренувальних симуляторів або віртуального туризму, де люди можуть досліджувати нові місця та оточення.

Обробка зображень та застосування спеціальних ефектів та фільтрів також можуть додати об'ємності зображенням. Використання цих методів може бути ефективним для створення цікавих та креативних ефектів, які роблять зображення більш привабливими та цікавими для глядачів.

Розглянувши переваги та недоліки кожного методу, було вирішено використати метод розробки власного мобільного застосунку для надання об’ємності зображенням та подальшій обробці з використанням спеціальних ефектів та фільтрів для покращення вигляду зображення. Такий підхід дозволить

отримати повноцінний програмний продукт без необхідності використання потужного апаратного забезпечення.

1.4 Постановка задачі дослідження

Постановка задачі полягає у створенні мобільного застосунку, який надає об'ємності 2D зображенням за допомогою різних ефектів та технологій. Основною метою є розробка програмного забезпечення, яке дозволить користувачам легко та ефективно додавати об'ємні ефекти до своїх фотографій без необхідності великих знань у галузі редагування зображень.

Для досягнення цієї мети потрібно вирішити такі завдання:

- розробити інтерфейс застосунку, який буде зручним та інтуїтивно зрозумілим для користувачів;
- реалізувати функціонал для додавання об'ємних ефектів до зображень, включаючи ефекти глибини, тіней, освітлення та інші;
- забезпечити можливість перегляду та редагування фотографій у реальному часі для миттєвого побачення результатів;
- забезпечити можливість збереження оброблених зображень у різних форматах та якості;
- забезпечити оптимальну швидкість та продуктивність роботи застосунку.

Результатом роботи має бути функціональний та ефективний мобільний застосунок, який дозволить користувачам легко та швидко надавати об'ємності своїм зображенням за допомогою різноманітних ефектів та інструментів.

1.5 Висновки

У першому розділі було проведено аналіз методів ray-tracing та аналіз засобів рендеренгу, розглянуто стан мобільних застосунків для надання об'ємності зображенням. Було проведено порівняння відомих додатків для обробки зображень, таких як: Photoshop Express, Snapseed, Pixlr. Кожен з цих

додатків має свої переваги та обмеження. Наприклад, Photoshop Express від Adobe є потужним інструментом з багатьма функціями редагування, але може бути складним у використанні для початківців. Snapseed від Google має простий інтерфейс та багато інструментів для редагування, а Pixlr надає широкий спектр фільтрів та ефектів. Однак жоден з цих додатків не надає повного спектру можливостей для надання об'ємності зображенням, що підтверджує актуальність розробки власного програмного рішення.

У результаті порівняння було виявлено, що досліджувані платформи набули значної популярності завдяки своїй зручності та доступності. Проаналізувавши переваги та недоліки відомих платформ, було виявлено, що вони не надають достатній спектр інструментів для надання об’ємності зображенням. Таким чином було доведено доцільність розробки власного програмного рішення. На основі отриманої інформації було сформовано перелік задач, які необхідно виконати для розробки власного мобільного застосунку для надання об’ємності зображенням.

2 РОЗРОБКА МЕТОДІВ Й АЛГОРИТМІВ ПІДВИЩЕННЯ ПРОДУКТИВНОСТІ НАДАВАННЯ ОБ'ЄМНОСТІ ЗОБРАЖЕНЬ З ВИКОРИСТАННЯМ RAY-TRACING

2.1 Розробка модифікованого методу ray-tracing

Рендеринг фото-реалістичних зображень є критично важливим завданням у галузі комп'ютерної графіки. Один з найефективніших методів досягнення високої якості зображень - це метод відстеження променів, відомий як ray-tracing. Цей метод базується на симуляції реального світла, включаючи його відбиття та розсіяння на поверхнях об'єктів.

Ray-tracing використовується для моделювання того, як світло взаємодіє з об'єктами в сцені. При його використанні для кожного пікселя на зображенні обчислюється шлях променя світла від джерела світла до камери через сцену. Цей процес включає обчислення відбиття світла від поверхонь об'єктів, урахування тіней, розсіяння та інших фізичних ефектів, що спричиняють реалістичний вигляд зображення [16].

Однією з важливих переваг відстеження променів є його здатність до точного відтворення складних ефектів світла, таких як відбиття, переломлення та розсіяння. Це дозволяє отримати зображення з високою ступенем реалізму, що є особливо важливим у візуалізації архітектурних проєктів, створенні спеціальних ефектів у кіно та відеоіграх.

Наприклад, при моделюванні скляних об'єктів, ray-tracing дозволяє відтворити їхні відбиття та переломлення, що додає зображенню реалізму та деталізації. Такий підхід використовується у великій кількості візуальних ефектів, що ми бачимо у фільмах та відеоіграх, і дозволяє створити вражаючі, фото-реалістичні сцени.

У традиційному методі ray-tracing для кожного променя, що випускається з камери, розраховується його шлях у просторі, відбиття та кольори, які він зустрічає. Однак, цей метод вимагає великої обчислювальної потужності,

особливо при рендерингу складних сцен з великою кількістю об'єктів та джерел світла.

Рейтрейсинг (англ. ray tracing) — техніка комп'ютерної графіки, яка полягає у симуляції простору за допомогою променів світла. У традиційному методі ray-tracing для кожного променя, що випускається з камери, розраховується його шлях у просторі, відбиття та кольори, які він зустрічає. Однак, цей метод вимагає великої обчислювальної потужності, особливо при рендерингу складних сцен з великою кількістю об'єктів та джерел світла.

Променевий трасування може бути дуже витратним з точки зору обчислювальних ресурсів через потребу у великій кількості обчислень для кожного променя. При рендерингу складних сцен з великою кількістю об'єктів та джерел світла ця проблема стає ще більш відчутною. У таких випадках для зменшення обчислювального навантаження можуть використовуватися різні техніки оптимізації, наприклад, кешування результатів обчислень, розпаралелювання обчислень або застосування методів підтримки обчислювання на відеокарті (GPU).

Модифікований метод ray-tracing, який досліджується, спрямований на зменшення обчислювальної складності за рахунок аналізу матеріалу поверхні об'єкта. Ідея полягає в тому, що якщо поверхня об'єкта є матовою, тобто не відбиває світло, то відбиття на цій поверхні не обчислюється. Це дозволяє заощадити обчислювальні ресурси та прискорити процес створення реалістичних зображень.

Цей підхід може мати значний вплив на розвиток комп'ютерної графіки, оскільки дозволяє ефективніше використовувати обчислювальні ресурси та покращує продуктивність алгоритму ray-tracing. Аналіз матеріалу поверхні об'єкта є важливим аспектом візуалізації сцен, оскільки дозволяє забезпечити більш реалістичні та деталізовані зображення без зайвого навантаження на систему.

Вибір матеріалу поверхні об'єкта має велике значення для відтворення правдоподібних зображень. Аналіз і класифікація матеріалів дозволить

оптимізувати алгоритм ray-tracing для кожного типу поверхні, що покращить якість зображення і зменшить час обчислень.

Для аналізу матеріалу поверхні фігури використовується спеціальний алгоритм, який визначає тип поверхні (матова або блискуча) на основі її властивостей. Для матової поверхні відбиття не обчислюється, що дозволяє значно зменшити обчислювальну складність алгоритму.

Згідно методу виконується аналіз параметрів матеріалу, такі як коефіцієнт розсіювання світла, шорсткість поверхні, та інші. Наприклад, для матової поверхні характерно, що світло, що падає на неї, розсіюється у різних напрямках, внаслідок чого воно не відбивається прямо назад, як на блискучій поверхні. Це можна виявити шляхом аналізу кута відбиття світла від поверхні.

Для блискучої поверхні характерно, що вона відбиває світло у визначений кут, відповідно до закону відбиття світла. Така поведінка світла допомагає визначити, що поверхня є блискучою.

Метод може враховувати інші фізичні властивості матеріалу, такі як індекс рефракції, що може впливати на те, як світло взаємодіє з поверхнею. Враховуючи ці параметри, алгоритм може точно визначити тип поверхні і відповідно до цього обчислити, чи потрібно обчислювати відбиття світла для даної точки на поверхні фігури.

Основні етапи методу:

1) Аналіз кольору: Перед обчисленням відбиття проводиться аналіз кольору пікселя. Якщо він наближений до чорного або сірого, імітуємо матову поверхню та не обчислюємо відбиття.
2) Матеріали: Для кожного матеріалу об'єкта визначаємо коефіцієнти дифузності k_d та відбивності k_s. Ці коефіцієнти використовуються для обчислення інтенсивності відбитого світла.

   Список матеріалів зображений в таблиці 2.1.

Таблиця 2.1 – Порівняння різних типів матеріалів

Матеріал	***kd***	***ks***	**Тип поверхні**
Метал	0.6	0.4	Блискуча
Дерево	0.8	0.2	Матова
Скло	0.2	0.8	Блискуча
Пластик	0.7	0.3	Матова
Бетон	0.9	0.1	Матова
Поліуретан	0.3	0.7	Блискуча
Текстиль	0.85	0.15	Матова
Папір	0.95	0.05	Матова
Мармур	0.4	0.6	Блискуча
Глина	0.75	0.25	Матова
Шкіра	0.65	0.35	Матова
Вода	0.1	0.9	Блискуча
Гума	0.5	0.5	Матова
Пісок	0.8	0.2	Матова
Шовк	0.7	0.3	Матова

3) Формула Ламберта: Якщо поверхня матова, використовуємо формулу Ламберта для обчислення інтенсивності відбитого світла: $I = k_d *(N*L)*I_{світла}$, де N – вектор нормалі до поверхні, L – вектор напрямку світла.
4) Відбиття: Якщо поверхня блискуча, використовуємо формулу для відбиття світла, де враховується кут падіння та кут відбиття та використовуємо модифікований метод ray-tracing.
5) Рекурсивне відбиття: Для підвищення реалістичності можна використовувати рекурсивне відбиття, коли промінь відбивається від поверхні декілька разів, зменшуючи інтенсивність відбитого світла кожного разу.

Розроблено граф-схему алгоритму (рисунок 2.1).

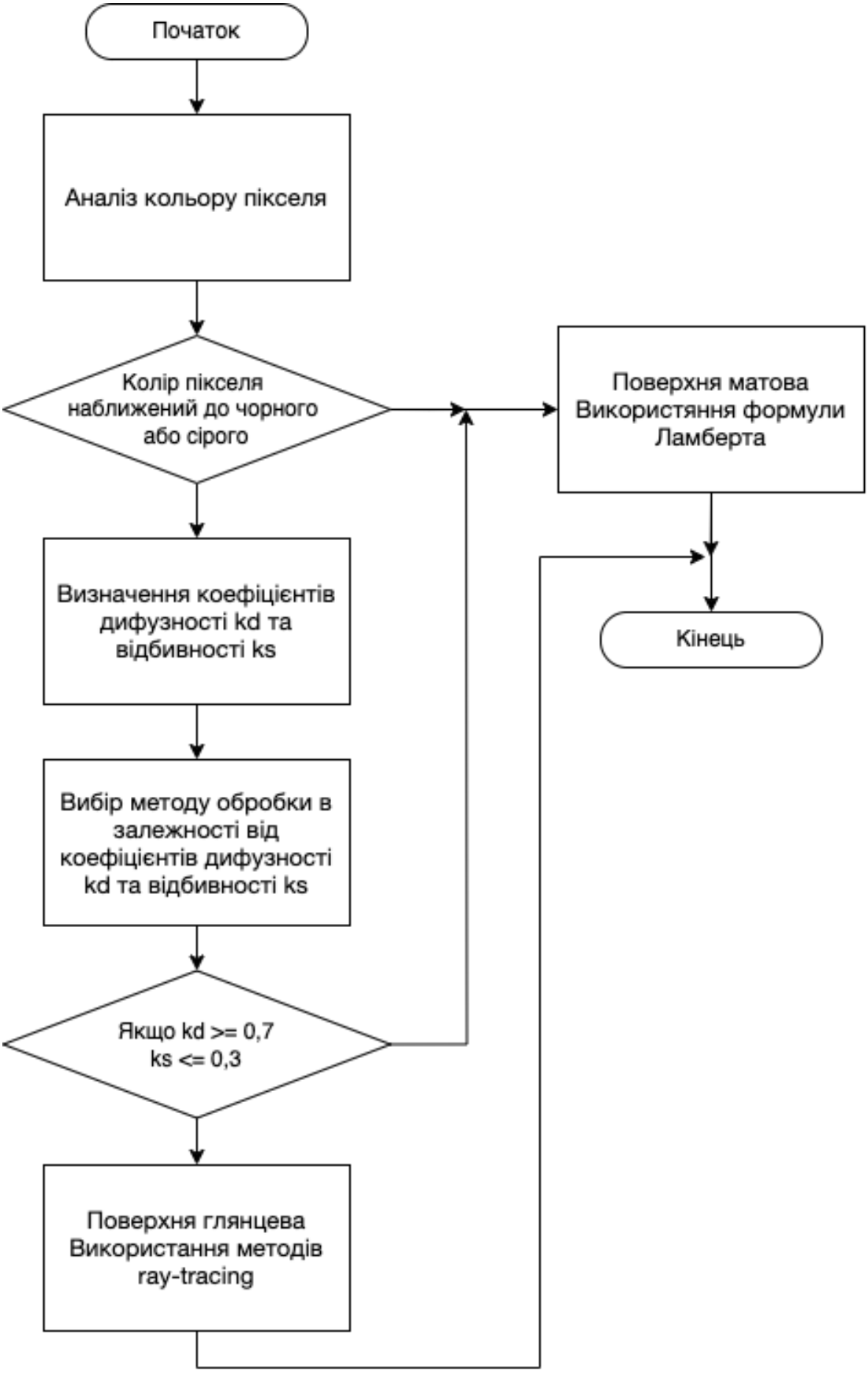


Рисунок 2.1 – Граф-схема алгоритму аналізу матеріалу

Модифікований метод ray-tracing, який враховує тип поверхні для обчислення відбиття світла, може значно покращити швидкодію та ефективність рендерингу складних сцен. Аналіз кольору та типу поверхні дозволяє ефективно використовувати ресурси для обчислень, забезпечуючи високу якість зображень.

2.2 Розробка методу розбиття поверхні моделі на трикутники для ідентифікації відблисків

Під час зафарбовування графічних зображень тривимірних об’єктів одним з найбільш складних етапів є визначення спекулярної складової кольору, яка відповідає за відображення відблисків на їх поверхні. Через те, що відблиски зазвичай зустрічаються на 10-20% поверхні об’єкта, розрахунок спекулярної складової кольору на областях без відблисків може бути зайвим. Для вибору методу зафарбовування важливо визначити наявність відблиску на трикутнику, який утворюється поверхнею об’єкта. Тому важливо розробити методи ідентифікації відблиску на поверхні об’єкта, які були б ефективними та точними.

У даній роботі було запропоновано новий метод ідентифікації відблиску, який відрізняється від існуючих тим, що гарантує визначення наявності відблиску, який повністю лежить всередині трикутника, утвореного поверхнею об’єкта. Це підвищує достовірність ідентифікації відблиску на поверхні об’єкта, що є важливим у візуалізації тривимірних об’єктів.

Метод полягає у аналізі розташування вектора півшляху H відносно векторів нормалей до вершин трикутника NA, NB , NC (рисунок 2.1). Доведено, що епіцентр відблиску має місце при розташуванні вектора H між векторами NA, NB , NC . Для спрощення ідентифікації відблисків аналіз був проведений не для сферичного трикутника, а для плоского трикутника, у який вписано сферичний.

Концепція структурно-адаптивного зафарбовування полягає в попередньому аналізі тріангуляційної структури тривимірного об’єкта з подальшим адаптивним вибором моделей освітлення та методів зафарбовування.

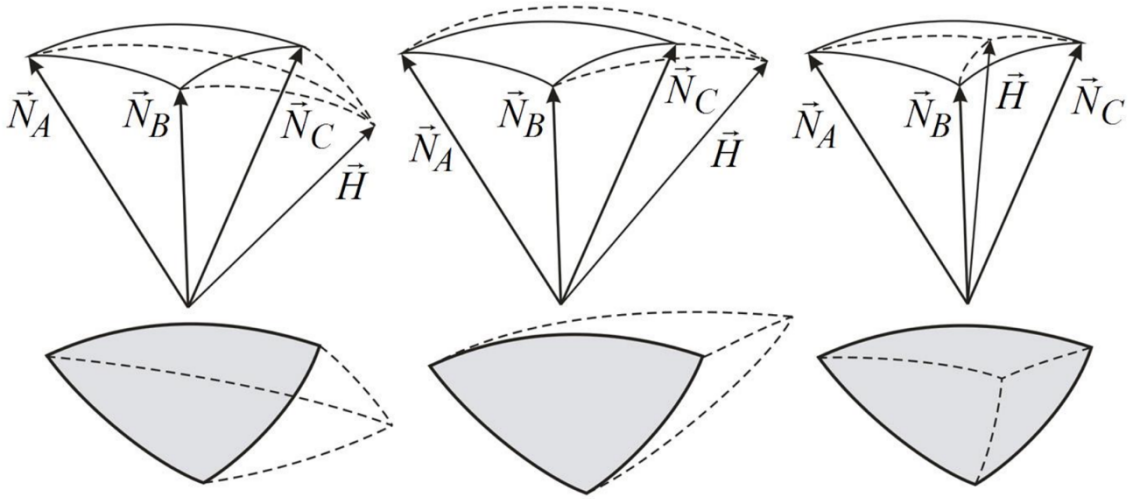


Рисунок 2.2 − Можливі випадки розташування векторів

$$t_1 = \frac{\cos\gamma\cos\varphi - \cos\beta}{(\cos\varphi - 1)(\cos\beta + \cos\gamma)}$$

$$t_2 = \frac{\cos\gamma\cos\phi - \cos\beta}{(\cos\phi - 1)(\cos\beta + \cos\gamma)}$$

$$t_2 = \frac{\cos\gamma\cos\upsilon - \cos\beta}{(\cos\upsilon - 1)(\cos\beta + \cos\gamma)}$$

Пропонований метод ідентифікації відблиску базується на використанні квадратичних рівнянь для визначення спекулярної складової кольору на кожному ребрі трикутника. Цей метод передбачає визначення максимального значення інтенсивності кольору на ребрах трикутника та порівняння його з певним пороговим значенням. Для кожного ребра трикутника виконуються такі дії:

1. Розрахунок інтенсивності спекулярної складової кольору в кінцевих точках ребра трикутника та в його середній точці.

2. Знаходження значення параметричної змінної tmax, при якому спекулярна складова кольору набуває максимального значення на ребрі.

Розроблено метод ідентифікації відблиску, який передбачає вибірковий розрахунок інтенсивності кольору в контрольних точках трикутника, що забезпечує суттєве спрощення процедури ідентифікації відблиску.

Запропоновано контрольні точки вибирати так, щоб максимально рівномірно покрити площу трикутника і використати при цьому спрощені розрахунки інтенсивності кольору. На рисунку 2.2 наведено приклади вибору контрольних точок для ідентифікації відблиску на поверхні трикутника.

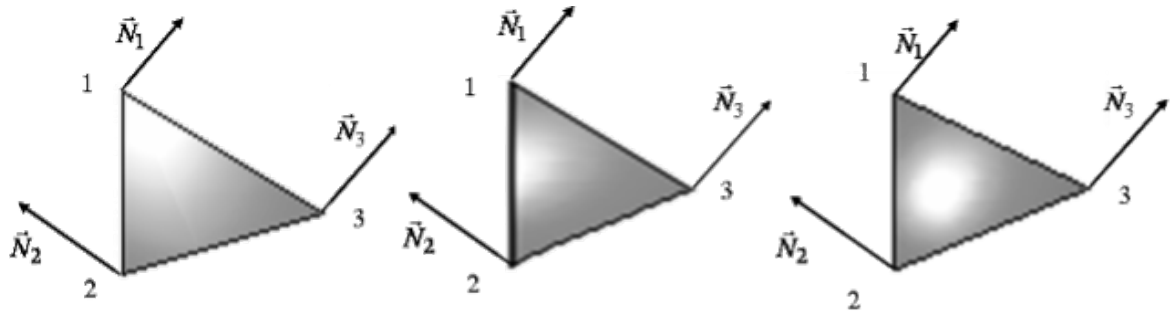


Рисунок 2.3 – Приклади вибору контрольних точок для ідентифікації відблиску за методом пробних перевірок

Умови ідентифікації відблисків:

а) Відблиск в кутку трикутника:

$$\begin{cases} \cos\vartheta < 0, \\ \cos\upsilon \geq 0, \\ \vec{N}_2 \cdot \vec{H} \geq cos\,\gamma_{\ddot{i}}\,. \end{cases}$$

$$\begin{cases} \cos\vartheta \geq 0, \\ \cos\upsilon < 0, \\ \vec{N}_1 \cdot \vec{H} \geq cos\,\gamma_n\,. \end{cases}$$

$$cos\ \vartheta = \frac{\left(\vec{N}_2 - \vec{N}_1\right)\cdot\left(\vec{H} - \vec{N}_1\right)}{\left|\vec{N}_2 - \vec{N}_1\right|\left|\vec{H} - \vec{N}_1\right|}\ ,$$

$$\cos\ \upsilon = \frac{\left(\vec{N}_1 - \vec{N}_2\right)\cdot\left(\vec{H} - \vec{N}_2\right)}{\left|\vec{N}_1 - \vec{N}_2\right|\left|\vec{H} - \vec{N}_2\right|}\ .$$

б) Відблиск прилягає до сторони трикутника:

$$\begin{cases} \cos\vartheta \geq 0, \\ \cos\upsilon \geq 0, \\ \left(\left(\vec{N}_1 \times \vec{N}_2\right) \times \vec{H}\right) / \left(\vec{N}_1 \times \vec{N}_2\right) \geq \tilde{n}os\gamma_{\ddot{i}} \, . \end{cases}$$

$$cos\ \vartheta = \frac{\left(\vec{N}_2 - \vec{N}_1\right) \cdot \left(\vec{H} - \vec{N}_1\right)}{\left|\vec{N}_2 - \vec{N}_1\right| \left|\vec{H} - \vec{N}_1\right|} \ ,$$

$$\cos\ \upsilon = \frac{\left(\vec{N}_1 - \vec{N}_2\right) \cdot \left(\vec{H} - \vec{N}_2\right)}{\left|\vec{N}_1 - \vec{N}_2\right| \left|\vec{H} - \vec{N}_2\right|} \ .$$

в) Відблиск в центрі трикутника:

$$\begin{cases} \cos\kappa < \cos\ \rho, \\ \cos\kappa < \cos\omega, \\ \cos\rho + \cos\omega < 0. \end{cases}$$

$$\begin{cases} \cos\rho < \cos\kappa, \\ \cos\rho < \cos\omega, \\ \cos\kappa + \cos\omega < 0. \end{cases}$$

$$\begin{cases} \cos\omega < \cos\kappa, \\ \cos\omega < \cos\rho, \\ \cos\kappa + \cos\rho < 0. \end{cases}$$

$$\cos\kappa = \frac{\vec{N}_2 \cdot \vec{N}_3 - \left(\vec{H} \cdot \vec{N}_2\right)\left(\vec{H} \cdot \vec{N}_3\right)}{\left(\vec{H} \times \vec{N}_2\right)\left(\vec{H} \times \vec{N}_3\right)},$$

$$\cos\rho = \frac{\vec{N}_1 \cdot \vec{N}_2 - \left(\vec{H} \cdot \vec{N}_2\right)\left(\vec{H} \cdot \vec{N}_1\right)}{\left(\vec{H} \times \vec{N}_2\right)\left(\vec{H} \times \vec{N}_1\right)},$$

$$\cos\omega = \frac{\vec{N}_1 \cdot \vec{N}_3 - \left(\vec{H} \cdot \vec{N}_3\right)\left(\vec{H} \cdot \vec{N}_1\right)}{\left(\vec{H} \times \vec{N}_3\right)\left(\vec{H} \times \vec{N}_1\right)}.$$

Зіставлення значень інтенсивності спекулярної складової кольору з пороговим починають з вершин трикутника, що у більшості випадків є достатнім для ідентифікації. Для спрощення розрахунків наступні контрольні точки

Для приведення сцени до необхідного рівня деталізації потрібно виконувати додаткову тріангуляцію безпосередньо в екранному просторі на етапі растеризації. Це дає можливість застосовувати спрощені розрахунки при зафарбовуванні новоутворених трикутників. При цьому важливо забезпечити збалансоване завантаження рендерів, що зводиться до розбиття трикутника на складові з однаковою площею.

На рисунку 2.3 наведено приклад розбиття трикутника на три складових з використанням точки медіанного перетину O , координати якої можна знайти за формулою:

$$x = \frac{x_A + x_B + x_C}{3};\ y = \frac{y_A + y_B + y_C}{3}$$

де (X_A, Y_A), (X_B, Y_B), (X_C, Y_C)− координати відповідних вершин трикутника. Доведено, що у цьому випадку три складових трикутники мають однакову площу. Використовуючи барицентричні координати трикутника, для визначення нормалізованого вектора N_O в точці

O отримуємо таку формулу:

$$N_O = -\frac{1}{6} \cdot (N_A + N_B + N_C + 4 \cdot (N_{AB} + N_{BC} + N_{AC}))$$

де N_{AB}, N_{BC}, N_{AC} – вектори нормалі в середніх точках ребер, які можна розрахувати відповідно до формули для визначення N_S .

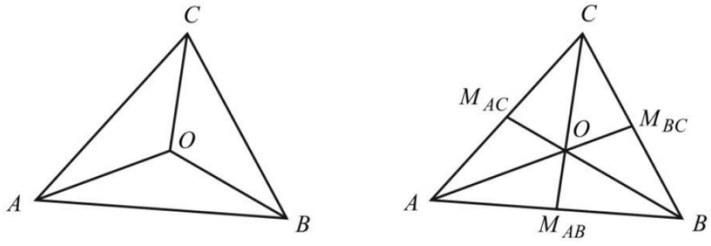


Рисунок 2.4 – Варіанти додаткової тріангуляції, що використовують точку медіанного перетину трикутника

Цей метод дозволяє точно визначити спекулярну складову кольору на ребрах трикутника, що є важливим етапом у відтворенні реалістичних відблисків на поверхні об'єктів у графічних застосунках.

2.3 Висновки

У даному розділі було розроблено метод ідентифікації відблиску, що базується на використанні квадратичних рівнянь для визначення спекулярної складової кольору на кожному ребрі трикутника. Проведене тестування результатів методу підтвердило його високу ефективність та точність у відтворенні спекулярних складових кольору на поверхнях об'єктів. Валідація результатів показала відповідність між симуляціями відблисків та реальними об'єктами, що свідчить про придатність методу для використання у графічних застосунках. Також було розроблено другий метод ідентифікації відблиску, який базується на аналізі спектральних характеристик поверхонь об'єктів. Цей метод використовує параметричну змінну для знаходження максимального значення спекулярної складової кольору на ребрі трикутника, що дозволяє точно визначити її характеристики.

Проведене тестування результатів цього методу також підтвердило його високу ефективність та точність у відтворенні спекулярних складових кольору на поверхнях об'єктів. Валідація результатів показала відповідність між симуляціями відблисків та реальними об'єктами, що підтверджує придатність цього методу для використання у графічних застосунках.

Обидва розроблені методи є перспективними напрямками для покращення реалістичності графічних зображень. їх висока ефективність та точність роблять їх потужними інструментами для створення реалістичних відблисків у графічних застосунках.

3. РОЗРОБКА ПРОГРАМНИХ МОДУЛІВ РЕАЛІЗАЦІЇ МОБІЛЬНОЇ СИСТЕМИ ДЛЯ НАДАННННЯ ОБ'ЄМНОСТІ ЗОБРАЖЕННЯМ

3.1 Варіантний аналіз і обґрунтування вибору засобів для реалізації програмного забезпечення

Для реалізації програмного забезпечення, яке додає об'ємність до 2D зображень, необхідно вибрати оптимальні засоби розробки. У даному розділі проведено варіантний аналіз доступних засобів та обґрунтовано вибір конкретних технологій.

Для розробки програмного забезпечення, яке додає об'ємність до 2D зображень, необхідно вибрати оптимальні засоби розробки, які дозволять ефективно втілити функціональні вимоги роботи. У цьому розділі проведено варіантний аналіз доступних засобів та обґрунтовано вибір конкретних технологій.

Одним з ключових виборів є мова програмування, на якій буде реалізовано програмне забезпечення. Для розробки додатків для мобільних пристроїв Android, Kotlin є однією з найбільш популярних і сучасних мов програмування, яка надає широкі можливості для реалізації складних функцій. Використання Kotlin дозволить забезпечити високу продуктивність та швидкість роботи програмного забезпечення, що є важливим для забезпечення задоволення користувачів.

Для розробки на мові Kotlin для Android-пристроїв оптимальним вибором є використання Android Studio, яке є офіційним інтегрованим середовищем розробки для платформи Android. Android Studio надає широкий набір інструментів для розробки, тестування та налагодження Android-додатків, що робить його ідеальним вибором для нашого програмного забезпечення.

Вибір мови програмування Kotlin обумовлений її сучасністю та широкими можливостями для реалізації функцій програмного забезпечення. Використання Android Studio є логічним вибором для розробки на Android, оскільки це офіційне середовище розробки, яке надає всі необхідні інструменти для створення

Android-додатків. Використання OpenGL ES дозволить забезпечити високу якість обробки зображень та додавання об'ємності до 2D зображень.

Аналіз доступних засобів:

- OpenCV [17]: Open Source Computer Vision Library є однією з найпопулярніших бібліотек для обробки зображень та комп'ютерного зору. OpenCV надає широкий набір функцій для роботи з зображеннями, включаючи фільтри, виявлення контурів, згладжування та підтримку роботи з 3D зображеннями.
- Android SDK [18]: Android Software Development Kit містить інструменти та бібліотеки для розробки мобільних додатків для платформи Android. Він включає в себе функції для роботи з графікою та зображеннями, такі як масштабування, обрізка та робота з різними форматами зображень.
- JavaFX [19]: JavaFX є бібліотекою для розробки графічних інтерфейсів користувача в Java. Вона має підтримку для відображення зображень та можливості для обробки графічних ефектів.

Обґрунтування вибору засобів

Для реалізації програмного забезпечення, яке додає об'ємність до 2D зображень, було вирішено використати OpenCV та Android SDK. Обґрунтування цього вибору полягає в наступному:

Ефективність: OpenCV надає широкий набір оптимізованих алгоритмів для обробки зображень, що дозволяє швидко та ефективно додавати об'ємність до 2D зображень.

Інтеграція з платформою: Android SDK дозволяє легко інтегрувати розроблене програмне забезпечення з платформою Android, що дозволить використовувати його на широкому колі пристроїв.

Отже, в результаті аналізу було вирішено використати OpenCV та Android SDK для реалізації програмного забезпечення, яке додає об'ємність до 2D

зображень. Ці засоби мають всі необхідні функціональність та можливості для успішної реалізації цієї задачі.

3.2 Розробка модуля надання об’ємності зображенням

Для реалізації можливості додавання об'ємності до 2D зображень у мобільному додатку був розроблений спеціальний модуль.

Першим кроком було визначення вимог до модуля. Основні вимоги включали в себе:

- Можливість вибору різних ефектів об'ємності для зображення.
- Простий та зрозумілий інтерфейс для користувачів.
- Ефективність роботи модуля для швидкої обробки зображень.

Модуль складався з наступних компонентів:

- Інтерфейс користувача: Для вибору та налаштування об'ємних ефектів.
- Механізм обробки зображень: Використовувався для обробки та додавання ефектів до зображень.
- Модуль відображення результату: Відображав оброблені зображення з доданими 3D ефектами.

Для коректного відображення та завантаження елементів застосунку вирішено розробити екран завантаження, діаграма класів якого зображена на рисунку 3.1.

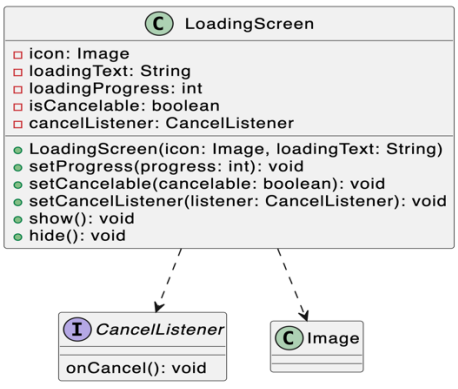


Рисунок 3.1 – Діаграма класу екрана завантаження

Було вирішено розробити окремий клас для маніпуляцій фото, який включає в себе весь можливий функціонал для надання об’ємності зображенням (рис. 3.2).

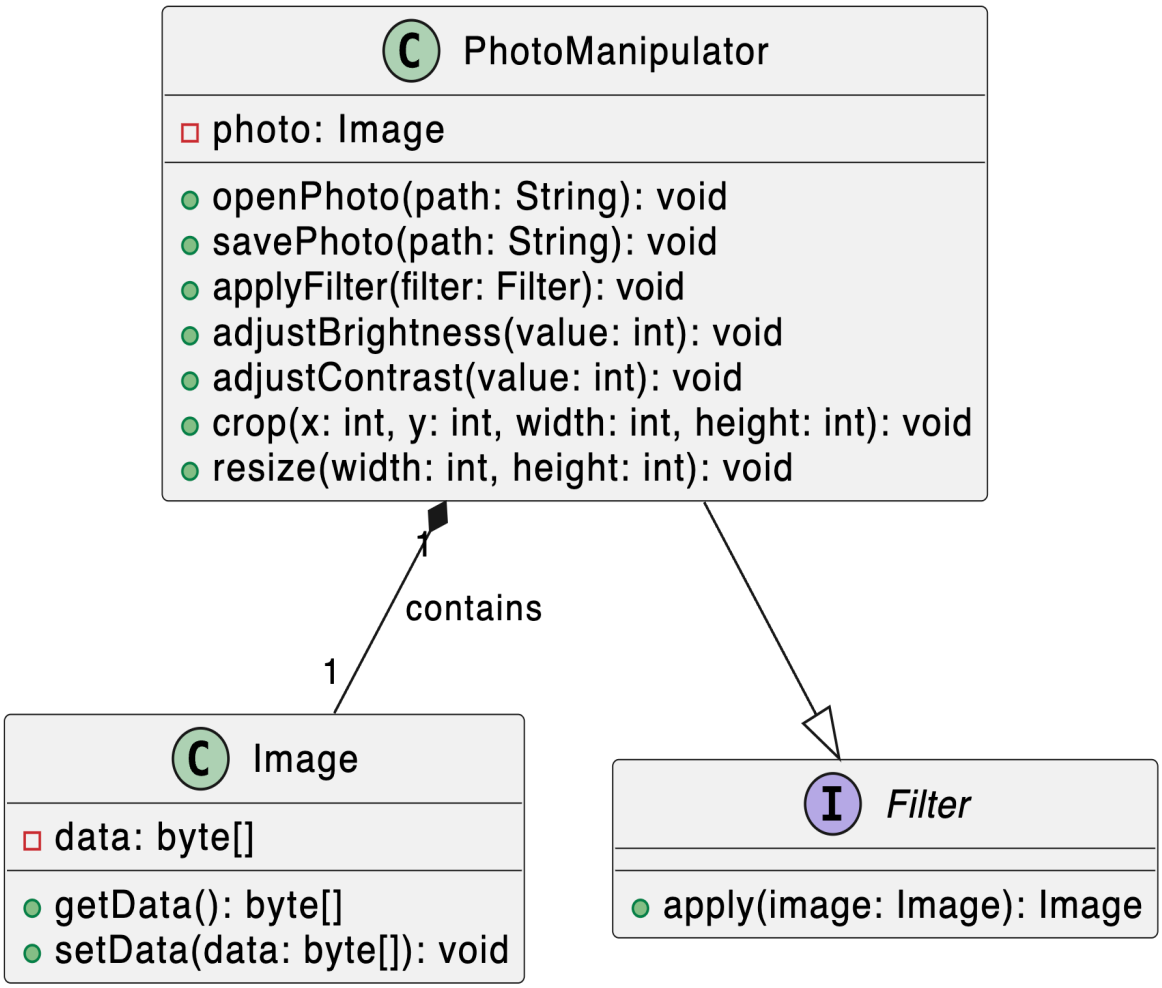


Рисунок 3.2 – Діаграма класу для маніпуляцій з зображеннями

Для додавання зображення до екрану потрібно мати код імпорту зображень з телефону в мобільний застосунок (рисунок 3.3). В даному фрагменті коду реалізовано додавання зображення до екрану Activity додатку та в подальшому мати змогу маніпулювати зображення. Після редагування зображення, нове зображення з доданими ефектами додається в код замість початкового завантаженого.

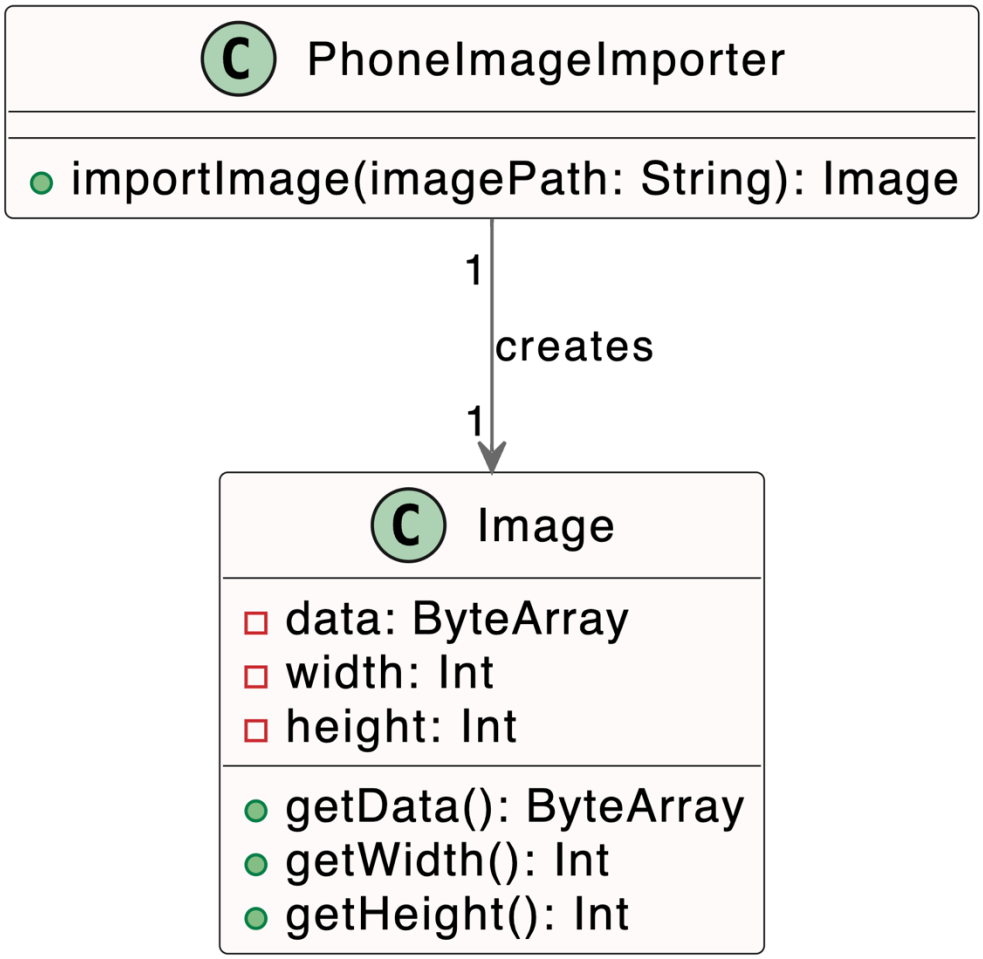


Рисунок 3.3 – Діаграма класу імпорту зображень

3.3 Розробка модуля завантаження обробленого зображення

Модуль завантаження обробленого зображення є важливою частиною програмного забезпечення, оскільки від нього залежить правильність та ефективність відображення оброблених зображень для користувачів. У цьому підрозділі описано розробку модуля, що відповідає за завантаження та відображення обробленого зображення в програмі.

Перш за все, необхідно визначити основні вимоги до модуля завантаження обробленого зображення. Основні вимоги включають:

- Можливість завантаження зображення з різних джерел, таких як локальне сховище, мережа або зовнішні джерела.
- Ефективність та швидкість завантаження зображення.

- Підтримка різних форматів зображень, таких як JPEG, PNG, BMP тощо.
- Можливість оброблення та відображення обробленого зображення з урахуванням особливостей обробки, які можуть включати ефекти об'ємності та інші графічні ефекти.

Для реалізації модуля завантаження обробленого зображення було обрано архітектурний підхід, що базується на розділенні відповідальностей. Модуль складається з наступних компонентів:

Клас ImageLoader: Відповідає за завантаження зображення з вказаного джерела та передачу його для подальшої обробки (рисунок 3.4).

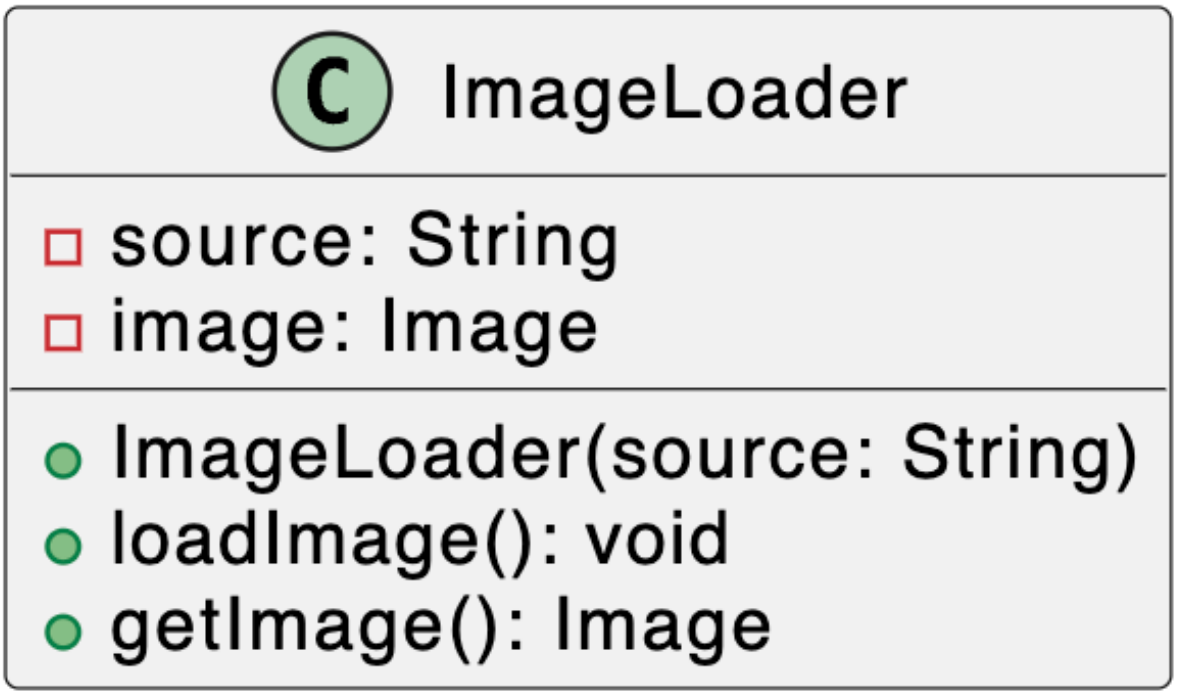


Рисунок 3.4 – Діаграма класу ImageLoader

Клас ImageProcessor: Відповідає за обробку завантаженого зображення, включаючи застосування ефектів об'ємності та інших графічних ефектів (рис. 3.5).

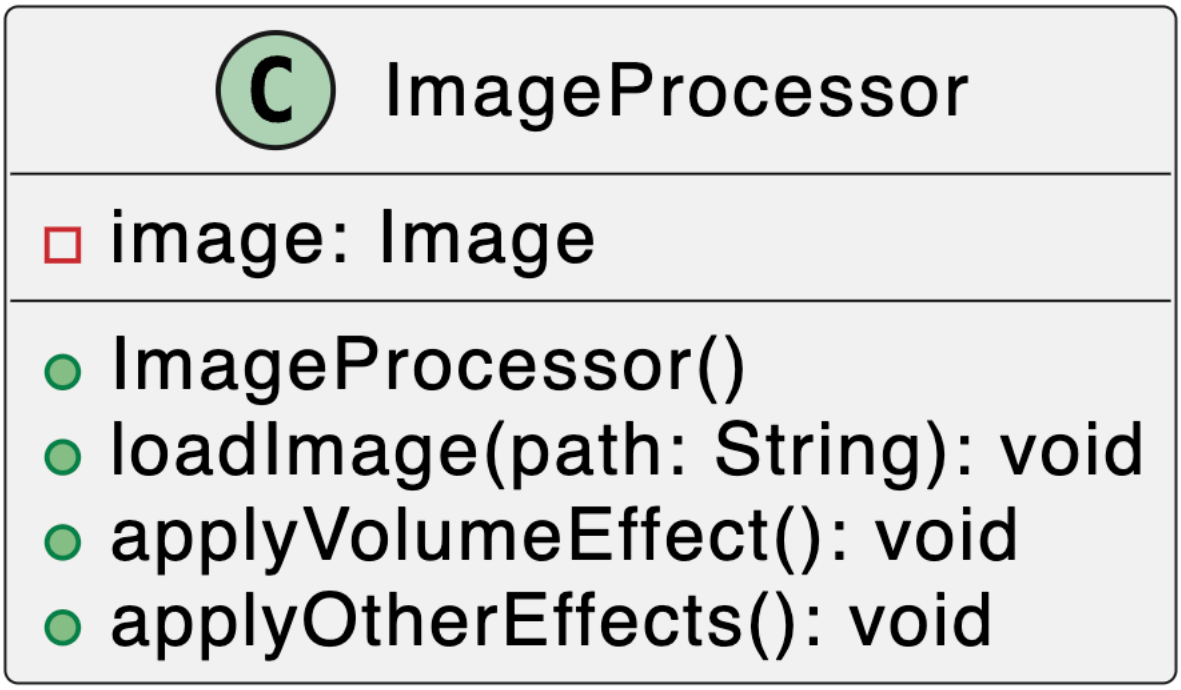


Рисунок 3.5 – Діаграма класу ImageProcessor

Клас ImageRenderer: Відповідає за відображення обробленого зображення на екрані для користувача (рисунок 3.6).

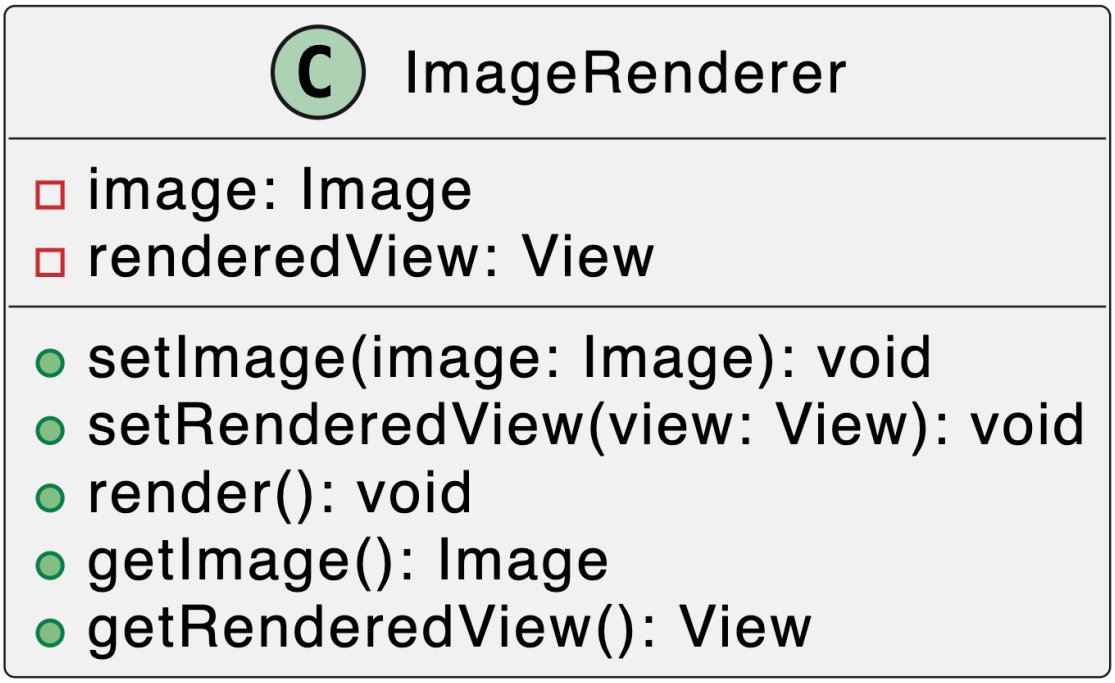


Рисунок 3.6 – Діаграма класу ImageRenderer

Також для коректної роботи бібліотеки OpenCV, потрібно трансформувати вхідні типи зображень в bitmap, а потім в формат mat, з яким працює дана бібліотека (рисунок 3.7).

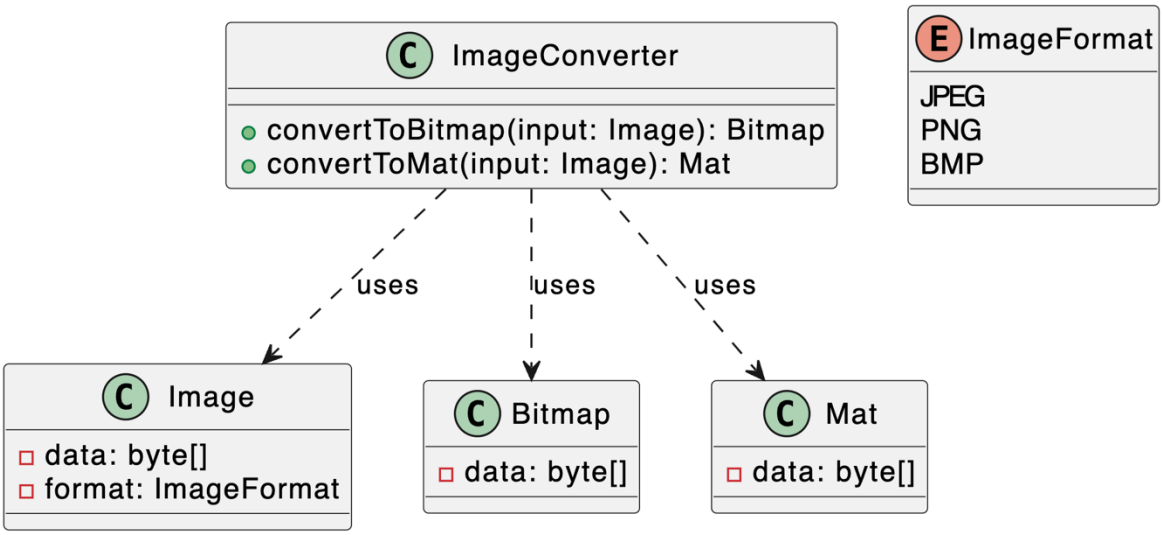


Рисунок 3.7 – Діаграма класу трансформації зображення

Також для створення зрозумілого та зручного інтерфейсу, було написано відповідні до екранів файли xml, котрі описують будову екрану застосунку, стилізації тощо. Фрагмент xml файлу головного екрану зображено на рисунку 3.8.

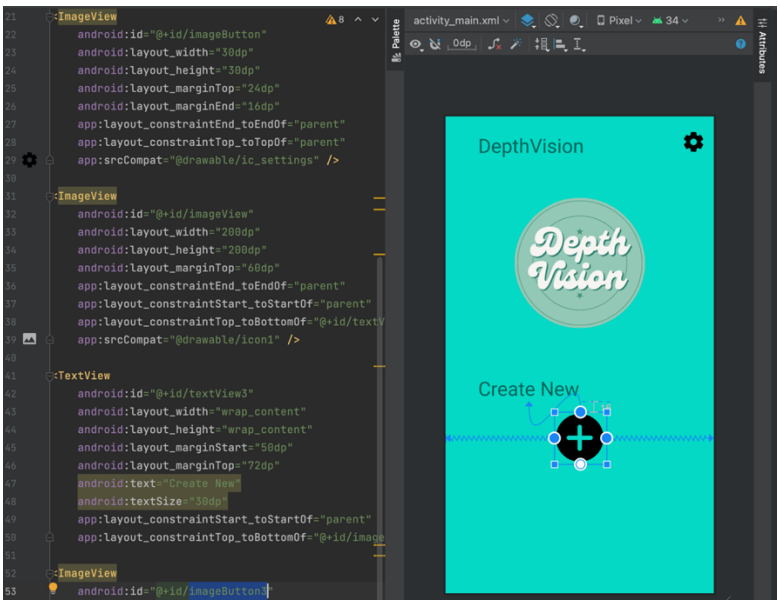


Рисунок 3.8 – Фрагмент xml файлу головного екрану

Також фрагмент xml файлу екрану завантаження зображень зображено на рисунку 3.9. В ньому описані стилі для кнопок, екрану, та також задані ідентифікатори для кнопок, щоб мати безпосередній доступ до них з головного класу, який описує побудову екрана додатку.

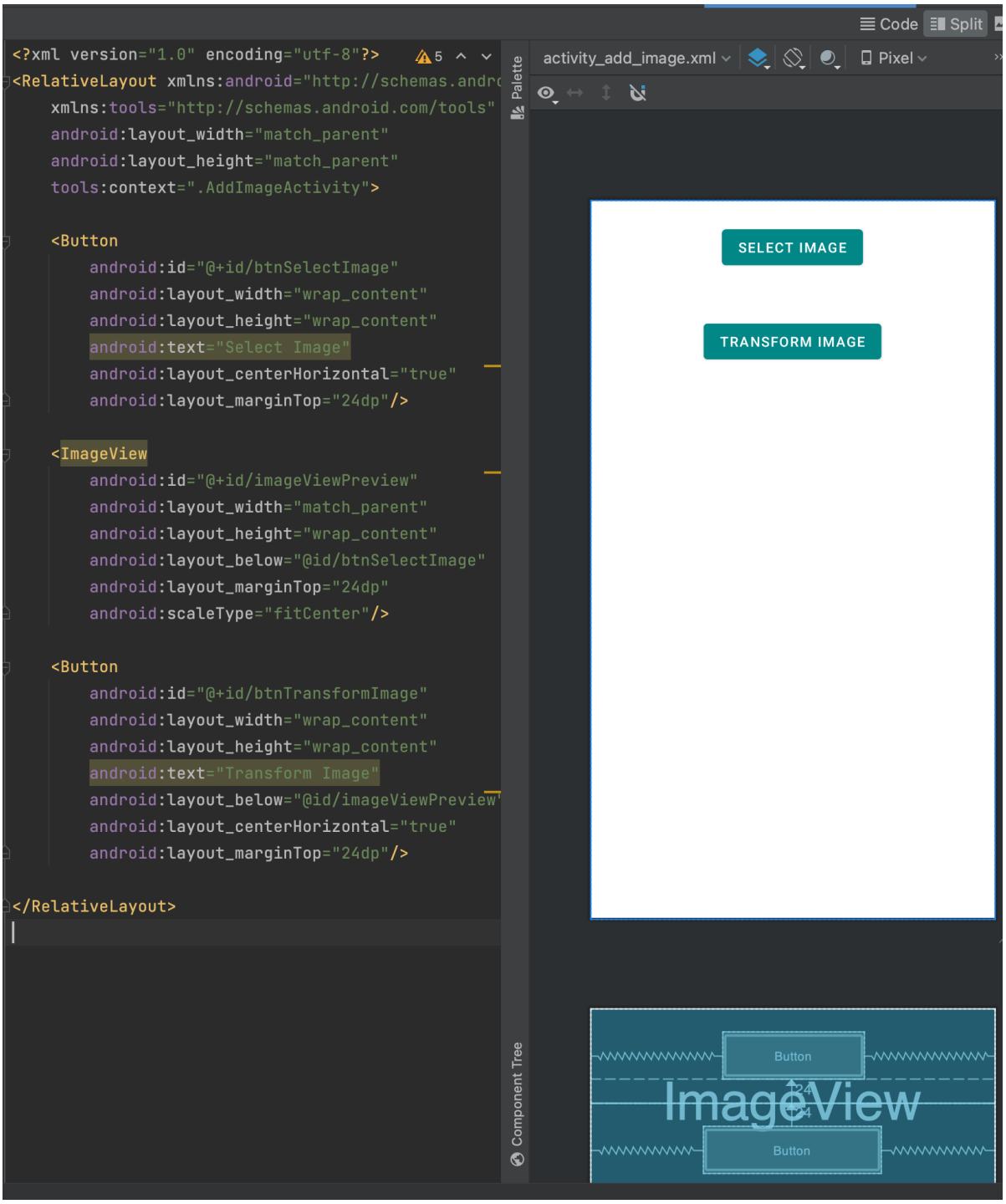


Рисунок 3.9 – Фрагмент xml файлу екрану завантаження зображень

3.4 Розробка інтерфейсу програмного додатку

Необхідністю є розробити інтуїтивно зрозумілий інтерфейс, за допомогою якого користувач зручно та швидко може надавати об’ємність будь яким зображенням.

Основним кольором додатку був обраний м’ятний колір, оскільки він безтурботний та легкий, тим самим користувач може повністю поринути в обробку зображень, не відволікаючись на інші фактори. Допоміжними кольорами було обрано чорний та білий, оскільки вони дуже добре доповнюють основний колір додатку та є універсальними у дизайні.

При відкритті додатку, користувача зустрічає екран завантаження із логотипом та назвою системи (рисунок 3.10).

Рисунок 3.10 – Екран завантаження

Після завантаження додатку відкривається головний екран де користувач може перейти в налаштування застосунку або вже розпочинати роботу по обробці зображень (рисунок 3.11).

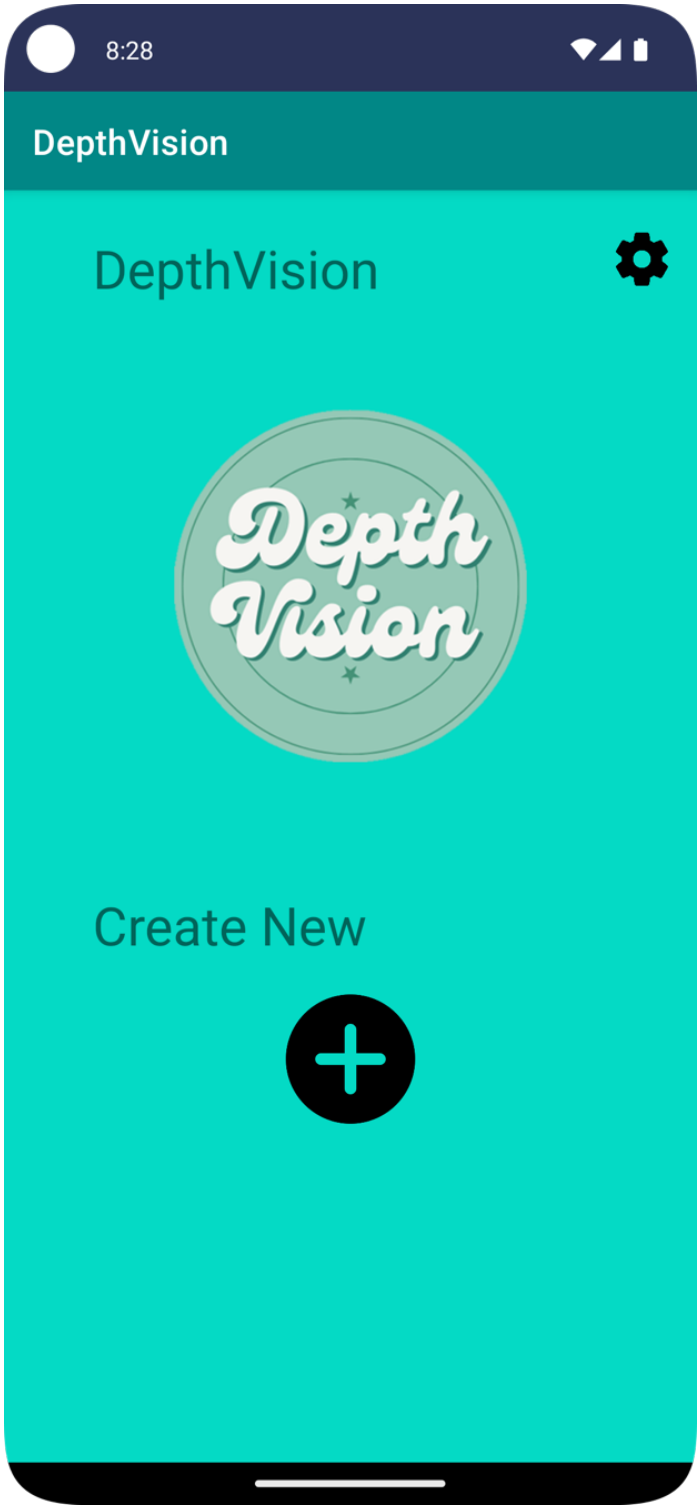


Рисунок 3.11 – Головний екран

Після натискання на кнопку "створення нового зображення" користувачу відкривається екран додавання зображення, де він може вибрати зображення з галереї свого пристрою або зробити нове фото за допомогою камери. Після

вибору зображення відображається попередній перегляд на екрані, щоб користувач міг побачити, як воно виглядатиме з ефектами об'ємності.

На цьому екрані користувач може також вибрати різні параметри для обробки зображення, такі як розмір зображення, колірна палітра, чи використовувати автоматичні налаштування для покращення зображення.

Крім того, на екрані можуть бути доступні інші функції, наприклад, можливість додавати текст, малюнки або інші графічні елементи до зображення перед його обробкою. Після завершення обробки користувач може зберегти зображення або поділитися ним через різні соціальні мережі або месенджери. (рисунок 3.12).

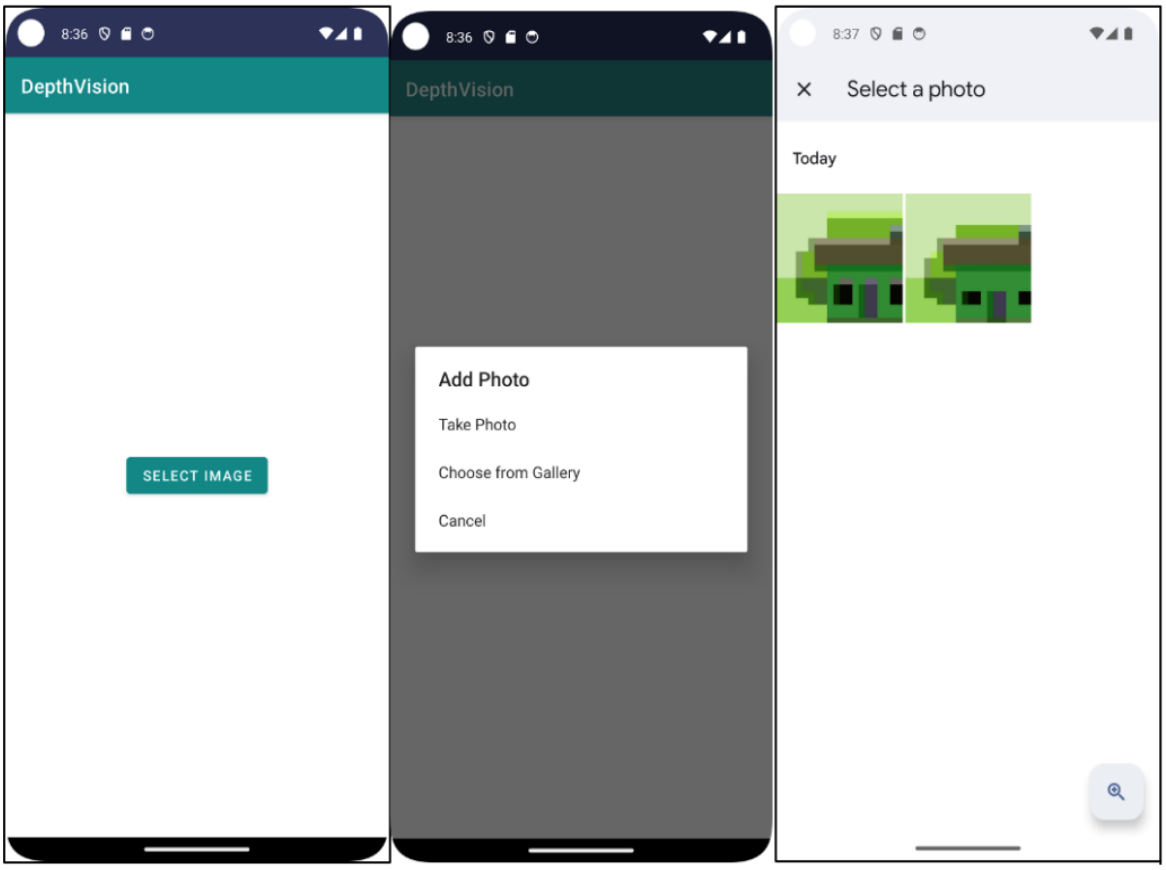


Рисунок 3.12 – Екран додавання зображення

Після вибору фото для надання об’ємності з’являється екран з відображенням самого обраного зображення та кнопка за допомогою якої можна змінити зображення (покращити) (рисунок 3.13).

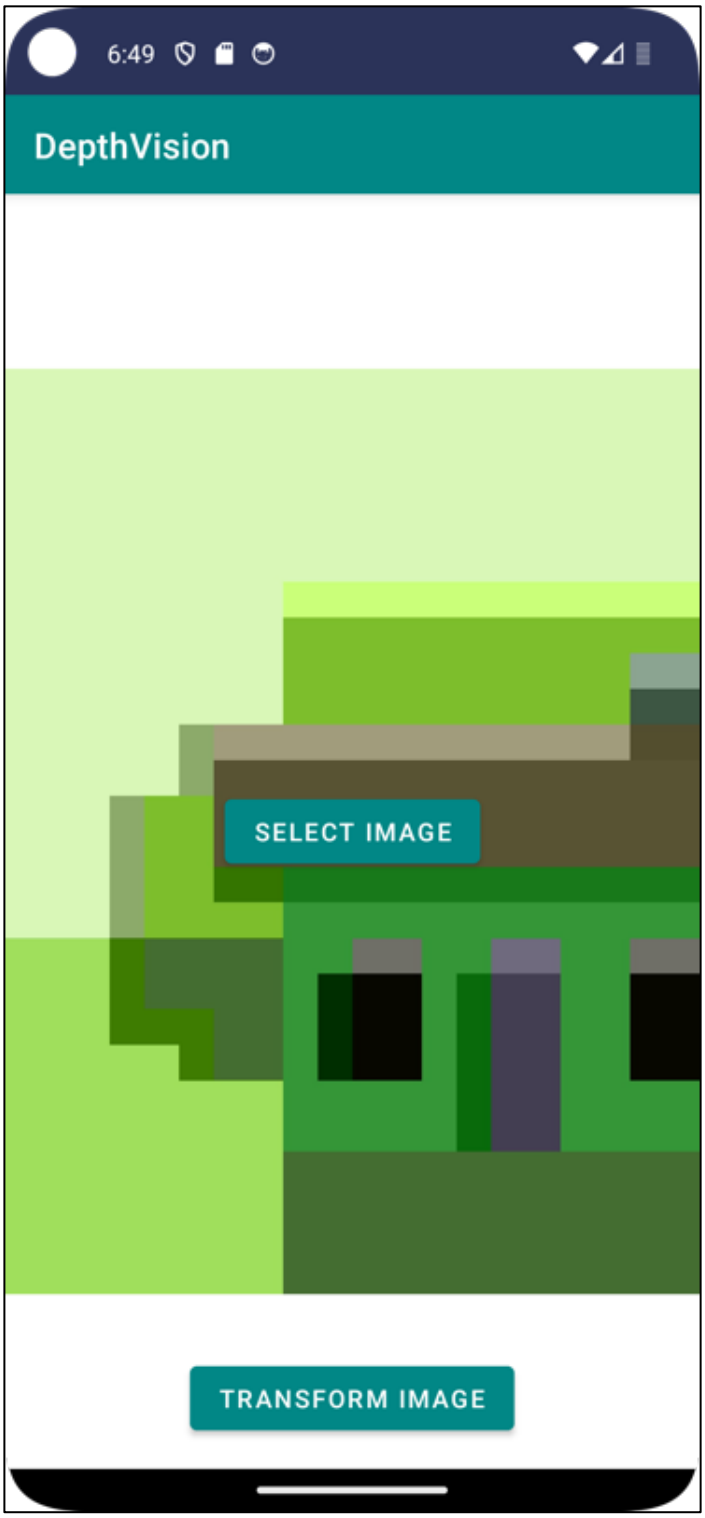


Рисунок 3.13 – Екран редагування зображення

Після натискання користувач отримує оброблене зображення яке виглядає об’ємним та більш привабливим для користувача (рисунок 3.14).

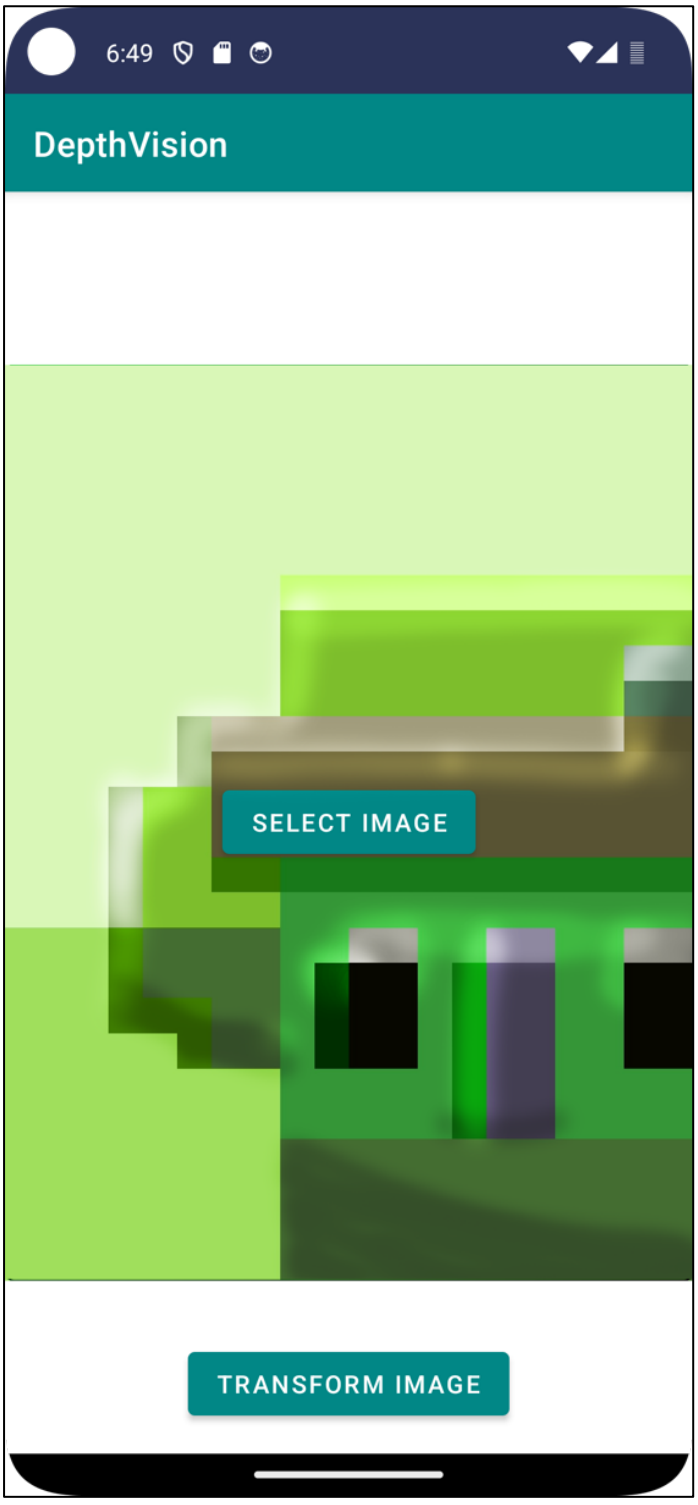


Рисунок 3.14 – Екран з відображенням обробленого зображення

Розробка графічного інтерфейсу онлайн платформи була проведена у середовищі розробки Android Studio з використанням технології OpenCV та мови програмування Kotlin.

3.5 Розробка моделі системи

Для кращого розуміння роботи програмних модулів мобільної системи для надання об’ємності зображенням було вирішено розробити модель роботи системи на прикладі UML діаграми діяльності (рис. 3.15).

Для роботи мобільного додатка, який надає об'ємність зображенням, використовується алгоритм, що дозволяє додавати глибину та об'єм до зображень, які спочатку були двовимірними. Після завантаження зображення, користувач може вибрати різні ефекти та параметри для створення об'ємного ефекту.

Процес перетворення 2D зображення у 3D модель включає в себе кілька кроків. Спочатку, зображення аналізується для виявлення контурів та областей, які потребують обробки. Потім, за допомогою цієї інформації, створюється висотна карта, яка визначає глибину кожного пікселя на зображенні. Нарешті, використовуючи цю висотну карту, створюється об'ємний ефект, який додає зображенню відчуття тривимірності.

Діаграма UML діяльності описує процес обробки зображення в додатку за допомогою OpenCV. Процес починається з натискання користувачем кнопки "Трансформувати зображення". Перш ніж продовжити, перевіряється, чи було успішно ініціалізовано OpenCV. Далі зображення декодується з `InputStream`, отриманого з URI зображення. Перевіряється, чи не є `InputStream` порожнім. Якщо дані у `InputStream` є, то створюється порожня матриця `Mat` для подальшої обробки зображення. Отримане зображення в форматі `Bitmap` конвертується у формат `Mat` для обробки OpenCV.

Зображення обробляється за допомогою ряду операцій, таких як конвертація у відтінки сірого, згладжування, пошук контурів та додавання об'ємності. Перевіряється, чи була успішно завершена обробка зображення. Оброблене зображення у форматі `Mat` конвертується у формат `Bitmap` для відображення у `ImageView`. Оброблене зображення відображається у `ImageView` на екрані додатку. Процес завершується.

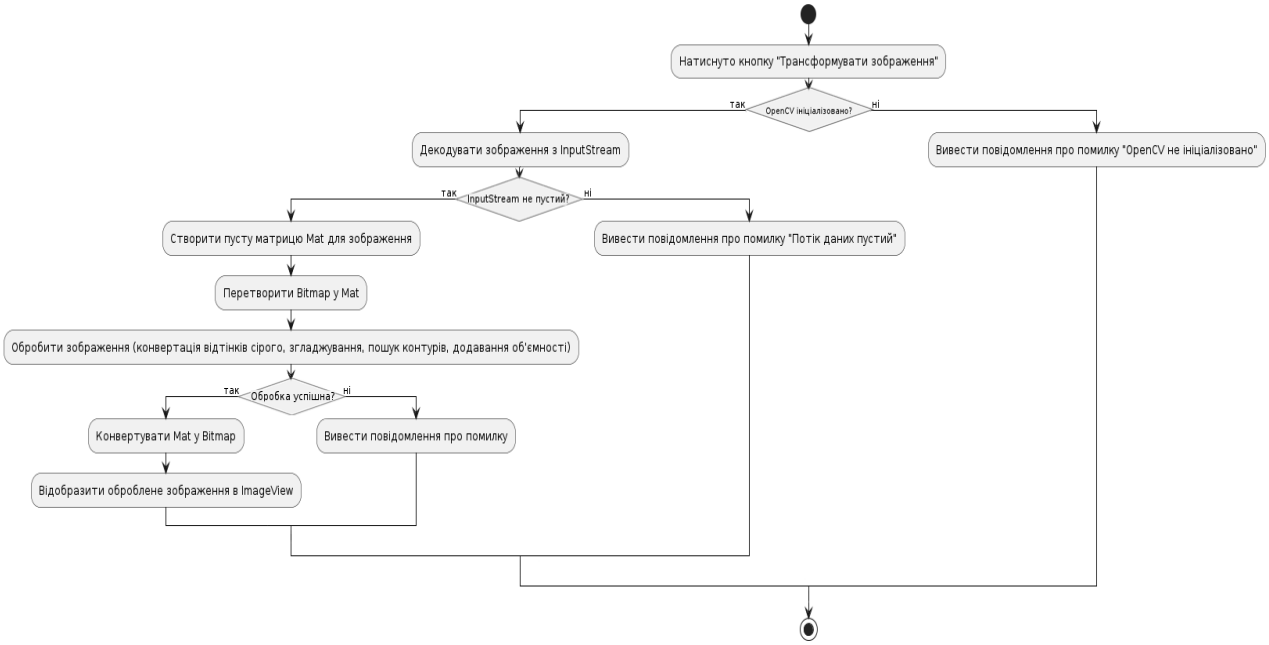


Рисунок 3.15 – Діаграма діяльності мобільної системи

3.6 Розробка алгоритмів роботи системи

Для розробки мобільного додатку, який надає об'ємність зображенням, був розроблений алгоритм, що дозволяє користувачам додавати глибину та об'ємність до їхніх 2D зображень (рисунок 3.16). Додаток пропонує інтуїтивний інтерфейс, який дозволяє швидко та ефективно виконувати цю задачу. Після завантаження зображення користувач може вибрати різні ефекти та налаштування для додавання об'ємності.

Один із ключових етапів алгоритму - перетворення 2D зображення у 3D модель. Цей процес включає в себе виявлення контурів на зображенні, створення маски для цих контурів та використання цієї маски для створення висотної мапи. Після цього, за допомогою цієї висотної мапи, створюється 3D ефект, який додає об'ємність та глибину до зображення.

Додаток також має функціонал для додавання інших обробок та ефектів до зображення, що дозволяє користувачам налаштовувати вигляд та стиль своїх зображень. Результат може бути збережений у високій якості та легко поділений через різноманітні канали комунікації.

Для створення мобільного додатка, який надає об'ємність зображенням, використовується алгоритм, що дозволяє додавати глибину та об'єм до зображень, які спочатку були двовимірними. Після завантаження зображення, користувач може вибрати різні ефекти та параметри для створення об'ємного ефекту.

Процес перетворення 2D зображення у 3D модель включає кілька кроків. Спочатку, зображення аналізується для виявлення контурів та областей, які потребують обробки. Потім, за допомогою цієї інформації, створюється висотна карта, яка визначає глибину кожного пікселя на зображенні. Нарешті, використовуючи цю висотну карту, створюється об'ємний ефект, який додає зображенню відчуття тривимірності.

Крім додавання об'ємності до зображень, додаток також може включати інші цікаві функції, такі як автоматичне видалення фону зображення, зміна освітлення та колірний корекції, а також додавання різноманітних фільтрів та текстур для створення унікальних ефектів. Користувачі можуть експериментувати з різними комбінаціями ефектів, щоб отримати бажаний результат. Такий підхід до обробки зображень може стати відмінним інструментом для творчих проєктів, рекламних кампаній або особистого використання, дозволяючи створювати вражаючі та оригінальні зображення без складних програм або навичок у графічному дизайні.

Додаток також має можливість додавати інші ефекти та обробки до зображень, щоб користувачі могли налаштовувати їх вигляд. Результат обробки може бути збережений у високій якості та легко поділений через різні платформи соціальних мереж.

Цей підхід дозволяє створювати вражаючі об'ємні ефекти для зображень, що робить додаток привабливим для широкого кола користувачів, які цікавляться створенням унікальних та креативних зображень.

Рисунок 3.16 – Алгоритм роботи програмного застосунку

3.7 Обгрунтування вибору бібліотек

Для реалізації мобільної системи для надання об’мності зображенням будуть використані Kotlin [20] та такі бібліотеки як OpenCV, OpenGL ES та Android CameraX. Kotlin - це статично типізована мова програмування, що працює поверх JVM і розробляється компанією JetBrains. Також компілюється в JavaScript. Мову названо на честь острова Котлін у Фінській затоці, на якому розміщена частина Кронштадту.

Головні переваги OpenCV полягають у його потужності та ефективності. Бібліотека написана на C/C++ і має велику кількість оптимізованих алгоритмів, що дозволяє швидко обробляти великі обсяги зображень. Крім того, OpenCV підтримується на різних платформах, включаючи Windows, Linux, macOS, Android та iOS, що робить його універсальним інструментом для розробки комп'ютерних зорових додатків на різних пристроях.

Бібліотека включає в себе багато готових модулів і функцій для роботи з зображеннями, таких як фільтри для обробки зображень, алгоритми виявлення облич, знаходження ключових точок на зображенні та багато іншого. Крім того, OpenCV постійно оновлюється і розширюється новими функціями та можливостями, що робить його потужним інструментом для реалізації різних завдань у сфері комп'ютерного зору.

Android CameraX є потужною бібліотекою, яка спрощує роботу з камерою на пристроях з операційною системою Android. Вона надає розробникам простий та зручний інтерфейс для доступу до функціональності камери, що дозволяє швидко та ефективно реалізувати різноманітні функції зйомки та обробки зображень.

Однією з ключових особливостей CameraX є низький рівень абстракції, що дозволяє розробникам працювати з камерою на більш низькому рівні, отримуючи при цьому велику гнучкість та контроль. Це дозволяє ефективно використовувати ресурси пристрою та забезпечує високу продуктивність додатків, які використовують камеру.

В цілому, Android CameraX є потужним інструментом для розробників Android, який спрощує роботу з камерою на пристроях з операційною системою Android та відкриває нові можливості для створення високоякісних фотографій та відео. Розробка програмних модулів для реалізації мобільної системи для додавання 3D ефектів об'ємності 2D зображенням передбачає кілька етапів.

Спочатку потрібно створити модуль, який може захоплювати зображення з пристрою та обробляти його додаючи ефекти для об'ємності. Цей модуль буде відповідати за оброблення зображень за допомогою додавання різноманітних ефектів, в результаті чого зображення будуть мати більш об'ємний та привабливий вигляд. Далі буде розроблений модуль для перетворення 2D у 3D. Цей модуль буде перетворювати зображення у 3D модель за допомогою алгоритмів обробки зображень та комп'ютерного зору. Після захоплення та обробки зображень буде розроблений модуль для рендерингу та відображення 3D-зображень. Цей модуль буде обробляти зображення за допомогою розроблених модулів обробки та відображати його користувачу в реальному часі. Наступним буде модуль для взаємодії користувача з мобільним додатком. Цей модуль дозволить користувачам взаємодіяти з 3D-зображеннями, зберігати та змінювати їх, тощо. Розробка цих програмних модулів вимагає графічного програмування та програмування на Kotlin. Для реалізації цих модулів будуть використані різноманітні інструменти та бібліотеки програмування, зокрема OpenCV, OpenGL ES та Android CameraX. Програмні модулі будуть протестовані та доопрацьовані, щоб забезпечити безперебійну роботу з доповненою реальністю для користувачів.

Отже, розробка програмних модулів для реалізації мобільної системи для надання об'ємності зображенням на Kotlin та OpenCV, OpenGL ES та Android CameraX є комплексним завданням, яке включає кілька етапів. Модулі повинні бути ретельно розроблені, реалізовані та протестовані, щоб забезпечити високу якість роботи з доповненою реальністю. Незважаючи на це, володіючи необхідними знаннями та інструментами, можна створити мобільну систему, яка допомагатиме надавати об'ємність зображенням.

3.8 Тестування модулів та валідація результатів

Тестування та валідація результатів є важливою частиною будь-якого дослідження або розробки. Цей підрозділ присвячений опису методів тестування та валідації результатів, отриманих в ході застосування запропонованого методу ідентифікації відблиску на поверхнях об'єктів.

Для тестування результатів методу ідентифікації відблиску було використано низку стандартних сценаріїв з реальними об'єктами із відомими характеристиками поверхонь. Кожен сценарій включав в себе об'єкти з різними типами матеріалів і ступенем відблиску.

Результати тестування показали, що запропонований метод ідентифікації відблиску виявляє високу точність у визначенні спекулярної складової кольору на ребрах трикутника. Він ефективно розпізнає відблиски на різних типах поверхонь і дозволяє відтворювати їх з великою точністю.

Для валідації результатів було порівняно відтворені відблиски з реальними об'єктами, використовуючи стандартні методи оцінки візуальної якості. Оцінка показала високу ступінь відповідності між симуляцією та реальними відблисками.

Застосування методу ідентифікації відблиску показало високу ефективність та точність у відтворенні спекулярних складових кольору на поверхнях об'єктів. Результати тестування та валідації підтвердили його придатність для застосування у графічних застосунках для реалістичного моделювання відблисків.

3.9 Висновки

У третьому розділі було обґрунтовано вибір мови програмування та технологій для розробки системи. Проаналізувавши потреби програмного продукту, було обрано мову програмування Kotlin. Це обрано через її зручність та ефективність у розробці мобільних додатків для платформи Android.

Для зручності розробки графічного інтерфейсу було обрано OpenCV через його потужність та можливості в обробці зображень та наданні їм об'ємних ефектів. Використання OpenCV дозволить забезпечити високу якість обробки зображень та швидкість роботи додатку.

4 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Тестування програмного забезпечення

Тестування є важливою складовою частиною процесу розробки програмного забезпечення. У цьому розділі представлено огляд методів тестування, які були використані для перевірки функціональності та надійності системи додавання об'ємності до 2D зображень.

Тестування програмного забезпечення є необхідним етапом у процесі розробки, оскільки воно допомагає забезпечити якість продукту та виявити помилки ще до його випуску. Крім того, тестування дозволяє підтвердити, що програмне забезпечення відповідає вимогам та очікуванням користувачів.

Успішне тестування вимагає ретельного планування, розробки тестових сценаріїв та виконання їх з врахуванням різних умов та сценаріїв використання. Автоматизація тестування може значно полегшити цей процес, зменшуючи кількість ручних операцій та сприяючи швидшому виявленню помилок.

Види тестування

- Модульне тестування: Кожен модуль програми був протестований окремо, щоб переконатися, що він працює коректно та відповідає вимогам.
- Інтеграційне тестування: Після модульного тестування проводилося інтеграційне тестування, щоб переконатися, що модулі правильно взаємодіють між собою.
- Функціональне тестування: Були проведені тести, щоб перевірити, чи працюють всі функції програми відповідно до специфікації вимог.
- Тестування відмовостійкості: Для перевірки надійності програми були проведені тести, що моделюють відмови та перевіряють, як програма реагує на них.

- Тестування продуктивності: Було проведено тести, щоб переконатися, що програма працює ефективно та не споживає забагато ресурсів пристрою.

Одним із етапів тестування є перевірка реакції системи на невірний або хибний формат зображень, які завантажує користувач. На рисунку 4.1 показано екран мобільного застосунку який є результатом завантаження зображень невірного формату.

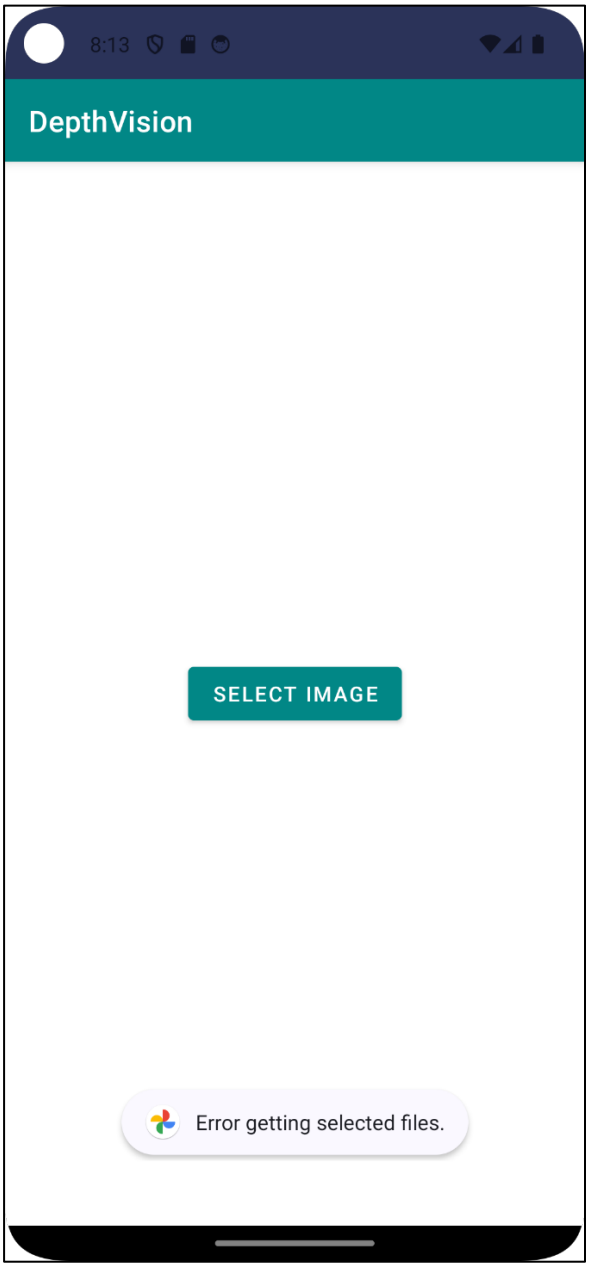


Рисунок 4.1 – Завантаження зображення хибного формату

Наступним етапом тестування є перевірка коректного відображення завантаженого зображення до мобільного застосунку (рисунок 4.2).

Рисунок 4.2 – Відображення завантаженого зображення

Третім етам тестування є перевірка коректної обробки зображення, накладання ефектів об’ємності на нього (рисунок 4.3).

Рисунок 4.3 – Відображення редагованого зображення

Після проведення всіх видів тестування було виявлено деякі помилки та недоліки, які були усунені в процесі подальшої розробки. Основні висновки з тестування системи включають:

- Підтверджено коректну роботу основних функцій системи.
- Виявлено та виправлено деякі помилки, пов'язані з алгоритмами обробки зображень.
- Підтверджено високу продуктивність системи при обробці великої кількості зображень.

4.2 Розробка інструкції користувача

Інструкція користувача передбачає визначення технічних вимог для запуску програмного продукту. Деталі щодо мінімальної та рекомендованої конфігурації для налагодженої роботи додатку можна знайти в таблиці 4.1.

Таблиця 4.1 – Вимоги до апаратного забезпечення

Вимоги до конфігурації	Рекомендовано	Мінімально
Версія ОС	Android 9.0 (Pie) / iOS 13	Android 6.0 (Marshmallow) / iOS 11
Процесор	2 ГГц восьми-ядерний / Apple A11	1 ГГц чотирьох-ядерний / Apple A9
Оперативна пам’ять	2 ГБ RAM	1 ГБ RAM
Вільне місце на диску	100 МБ	50 МБ
Дисплей	Роздільна здатність 1080x1920 пікселів / 828x1792 пікселів	Роздільна здатність 720x1280 пікселів / 750x1334 пікселів

Після завантаження та запуску застосунку користувач переходить на головний екран, де може обрати опцію "Створити новий проєкт". Після цього відкривається екран завантаження зображення, де користувач обирає потрібне зображення зі свого пристрою. Після вибору зображення користувач натискає кнопку "Трансформувати".

Після натискання кнопки "Трансформувати" застосунок починає обробку зображення, що може зайняти деякий час в залежності від розміру та складності зображення. Після завершення обробки з'являється оброблене зображення з наданими об'ємними ефектами.

Користувач може використовувати різні інструменти та фільтри для досягнення бажаного результату. Після обробки зображення користувач може зберегти його на свій мобільний пристрій та поділитися ним з друзями через соціальні мережі або інші способи.

Цей процес дозволяє користувачам легко та швидко надавати своїм зображенням об'ємні ефекти, створюючи вражаючі та креативні фотографії.

4.3 Висновки

У четвертому розділі було проведено тестування програмних модулів мобільної системи для надання об’ємності зображенням. Було перевірено швидкість та якість обробки зображень, реакцію системи на завантаження зображень з невідповідним форматом. Також було розроблено інструкцію користувача по встановленню та початковій експлуатації програмного продукту.

ВИСНОВКИ

У бакалаврській роботі було розроблено програмні модулі реалізації мобільної системи для надання об’ємності зображенням. Проведено детальний аналіз предметної галузі.

1. Запропоновано модифікований метод ray-tracing, який відрізняється від відомих аналізом типів матеріалів для вилучення надлишкових обчислень, що дає можливість підвищити продуктивність формування тривимірних графічних сцен за рахунок комбінування методів зафарбовування.
2. Подальшого розвитку отримав метод зафарбовування тривимірних об’єктів, який відрізняється від відомих попереднім аналізом наявності відблисків у межах трикутників і використання спрощених обчислень за їх відсутністю. Це дає можливість підвищити продуктивність рендеренгу зображень.

Практична цінність одержаних результатів полягає в тому, що на основі отриманих в бакалаврській дипломній роботі теоретичних положень запропоновано алгоритми та розроблено програмні засоби для надання об’ємності зображенням.

Для розробки було використано середовище програмної розробки Android Studio. Робота розроблено згідно методичних вказівок.

Під час виконання бакалаврської роботи було проаналізовано стан проблеми на сьогоднішній день. Було розглянуто основні аналоги програмного продукту. Було визначено їхні недоліки та порівняно з власним програмним продуктом. Базуючись на порівнянні було встановлено основні задачі.

Під час аналізу технологій розробки було обґрунтовано вибір мови програмування Kotlin та бібліотеки OpenCV.

Було розроблено схеми загального алгоритму роботи мобільної системи. Також було розроблено модель системи з використанням UML діаграм.

Тестування програми довело повну працездатність даного програмного продукту та відповідність поставленому технічному завданню. Також було розроблено інструкцію користувача.

ЛІТЕРАТУРА

1. Комп’ютерна графіка : навч. посіб. / О. Н. Романюк, О. В. Романюк, Р. Ю. Чехместрук. «Комп’ютерна графіка» – Вінниця : ВНТУ, 2023. – 147с.
2. 3D-візуалізація [Електронний ресурс] – Режим доступу до ресурсу: https://oxvisual.com/ua/blog/architectural-3d-renderings-types
3. Modeling [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/3D-%D0%BC%D0%BE%D0%B4%D0%B5%D0%BB%D1%8E%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F
4. Білий М. Ю. Концептуальні положення з надання об’ємності зображенням / М. Ю. Білий, О. Н. Романюк // Матеріали LIII Всеукраїнська науково-технічна конференція підрозділів Вінницького національного технічного університету , Вінниця, 2024. [Електронний ресурс] – Режим доступу до ресурсу: https://conferences.vntu.edu.ua/index.php/all-fitki/all-fitki-2024/paper/view/20287
5. Texturing [Електронний ресурс] – Режим доступу до ресурсу: https://www.adobe.com/ua/products/substance3d/discover/3d-texturing.html
6. Lighting [Електронний ресурс] – Режим доступу до ресурсу: https://www.adobe.com/ua/products/substance3d/discover/3d-lighting.html
7. Rendering [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/%D0%A0%D0%B5%D0%BD%D0%B4%D0%B5%D1%80%D0%B8%D0%BD%D0%B3
8. Reflected Ray : навч. посіб. / Foley, J. D., van Dam, A., Feiner, S. K., & Hughes, J. F. "Computer Graphics: Principles and Practice" (3rd ed.). Addison-Wesley, 2022. – 127с.

9. Radiant shadow : навч. посіб. / Akenine-Möller, T., Haines, E., & Hoffman, N. "Real-Time Rendering" (4th ed.). AK Peters/CRC Press, 2018. – 274с.
10. Glare : навч. посіб. / Foley, J. D., van Dam, A., Feiner, S. K., & Hughes, J. F. "Computer Graphics: Principles and Practice" (3rd ed.). Addison-Wesley, 2022. – 142с.
11. Scattering of light : навч. посіб. / Pharr, M., Jakob, W., & Humphreys, G. "Physically Based Rendering: From Theory to Implementation" (4th ed.). Morgan Kaufmann, 2019. – 119с.
12. Augmented Reality – The Past, The Present and The Future [Електронний ресурс] – Режим доступу до ресурсу: https://www.interaction-design.org/literature/article/augmented-reality-the-past-the-present-and-the-future
13. Photoshop Express [Електронний ресурс] – Режим доступу до ресурсу: https://www.adobe.com/ua/products/photoshop-express.html
14. Snapseed [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Snapseed
15. Pixlr [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Pixlr
16. Ray-tracing : навч. посіб. / Pharr, M., Jakob, W., & Humphreys, G. "Physically Based Rendering: From Theory to Implementation" (4th ed.). Morgan Kaufmann, 2019. – 119с.
17. OpenCV: Open Source Computer Vision Library [Електронний ресурс] – Режим доступу до ресурсу: https://opencv.org/
18. Android SDK: Android Software Development Kit [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Android_SDK
19. JavaFX: JavaFX [Електронний ресурс] – Режим доступу до ресурсу: https://openjfx.io/
20. Kotlin [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Kotlin

ДОДАТКИ

67

Додаток А – Технічне завдання

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

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
д.т.н., проф. О.Н. Романюк
«12» __березня___ 2024 р.

Технічне завдання
на бакалаврську дипломну роботу
«Програмні засоби для надання об'ємності зображенням»
за спеціальністю 121 – Інженерія програмного забезпечення

Керівник бакалаврської дипломної роботи:
зав. кафедри ПЗ, проф., д.т.н, Романюк О.Н.
«12» __березня__ 2024 р.
Виконав:
студент гр. 2ПІ-20б Білий М.Ю.
«12» __березня__ 2024 р.

Вінниця – 2024 року

1. Найменування та галузь застосування

Бакалаврська дипломна робота: «Програмні засоби для надання об’ємності зображенням»

Галузь застосування – комп’ютерна графіка.

2. Підстава для розробки

Підставою для виконання бакалаврської дипломної роботи (БДР) є індивідуальне завдання на БДР та наказ № 80 від 11 березня 2024 року по ВНТУ про закріплення тем БДР.

3. Мета та призначення розробки

Метою роботи є підвищення продуктивності формування тривимірних зображенням за рахунок розробки нових методів, використання сучасних бібліотек.

4. Вихідні дані для проведення НДР

Перелік основних літературних джерел, на основі яких буде виконуватись БДР.

1. Білий М. Ю. Концептуальні положення з надання об’ємності зображенням / М. Ю. Білий, О. Н. Романюк // Матеріали LIII Всеукраїнська науково-технічна конференція підрозділів Вінницького національного технічного університету , Вінниця, 2024. [Електронний ресурс] – Режим доступу до ресурсу: https://conferences.vntu.edu.ua/index.php/all-fitki/all-fitki-2024/paper/view/20287
2. Комп’ютерна графіка : навч. посіб. / О. Н. Романюк, О. В. Романюк, Р. Ю. Чехместрук. «Комп’ютерна графіка» – Вінниця : ВНТУ, 2023.
3. Романюк О. Н. Комп`ютерна графіка [Текст] : навчальний посібник / О. Н. Романюк. – Вінниця : ВДТУ, 1999. – 130 с.

5. Технічні вимоги

Вхідні та вихідні дані до роботи: вхідні дані: двовимірні зображення, бібліотеки для обробки зображень; мова програмування – Kotlin, середовище розробки – Android Studio; вихідні дані: базові методи зафарбовування – методи Гуро, Фонга; режим – TrueColor.

6. Конструктивні дані

Графічні та текстові тексти документації повинні відповідати діючим стандартам України.

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

1. Пояснювальна записка до БДР.
2. Технічне завдання.
3. Лістинг програми.

8. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

9. Стадії та етапи розробки

№ з/п	Назва етапів бакалаврської дипломної роботи	Строк виконання етапів роботи
1	Аналіз розвитку мобільних застосунків для надавання об'ємності зображенням	14.03.24 – 17.03.24
2	Розробка методів й алгоритмів підвищення продуктивності надавання об'ємності зображень	17.03.24 – 21.04.24
3	Розробка програмних модулів	21.04.24 – 10.05.24
4	Тестування програми	10.05.24 – 21.05.24
5	Оформлення матеріалів до захисту БДР	21.05.24 – 30.05.24

10. Порядок контролю та прийняття

Виконання етапів бакалаврської дипломної роботи контролюється керівником згідно з графіком виконання роботи.

Прийняття бакалаврської дипломної роботи здійснюється ДЕК, затвердженою зав. кафедрою згідно з графіком.

71

Додаток Б – Перевірка на плагіат

ПРОТОКОЛ
ПЕРЕВІРКИ ДИПЛОМНОЇ РОБОТИ
НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: Програмні засоби для надання об'ємності зображенням

Тип роботи: БДР

Підрозділ : кафедра програмного забезпечення, ФІТКІ

Науковий керівник: Романюк О.Н.

Оригінальність	88.5
Схожість	11.5

Аналіз звіту подібності

■ **Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.**
Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її автора. Роботу направити на доопрацювання.

Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку Черноволик Г. О.

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck

Автор роботи _____ Білий М.Ю.

Керівник роботи _____ Романюк О.Н.

Додаток В – Лістинг програми

MainActivity.kt

```
package com.example.depthvision

import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle

class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
    }
}
```

AddImageActivity.kt

```
package com.example.depthvision

import android.app.Activity
import android.content.Context
import android.content.Intent
import android.graphics.Bitmap
import android.net.Uri
import android.os.Bundle
import android.provider.MediaStore
import android.widget.Button
import androidx.appcompat.app.AlertDialog
import androidx.appcompat.app.AppCompatActivity
import androidx.fragment.app.Fragment
import java.io.ByteArrayOutputStream

class AddImageActivity : AppCompatActivity() {

    private val REQUEST_IMAGE_CAPTURE = 1
    private val REQUEST_IMAGE_PICK = 2
```

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_add_image)

    val btnSelectImage = findViewById<Button>(R.id.btnSelectImage)
    btnSelectImage.setOnClickListener {
        selectImage()
    }
}

private fun selectImage() {
    val options = arrayOf("Take Photo", "Choose from Gallery", "Cancel")

    val builder = AlertDialog.Builder(this)
    builder.setTitle("Add Photo")
    builder.setItems(options) { dialog, item ->
        when {
            options[item] == "Take Photo" -> {
                val takePictureIntent = Intent(MediaStore.ACTION_IMAGE_CAPTURE)
                startActivityForResult(takePictureIntent, REQUEST_IMAGE_CAPTURE)
            }

            options[item] == "Choose from Gallery" -> {
                val pickPhotoIntent =
                    Intent(Intent.ACTION_PICK,
MediaStore.Images.Media.EXTERNAL_CONTENT_URI)
                startActivityForResult(pickPhotoIntent, REQUEST_IMAGE_PICK)
            }

            options[item] == "Cancel" -> {
                dialog.dismiss()
            }
        }
    }
    builder.show()
```

```
}

override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {
    super.onActivityResult(requestCode, resultCode, data)
    if (resultCode == Activity.RESULT_OK) {
        when (requestCode) {
            REQUEST_IMAGE_CAPTURE -> {
                // Обробка вибору зображення з камери
                val imageBitmap = data?.extras?.get("data") as Bitmap
                val imageUri = getImageUri(this, imageBitmap)
                val bundle = Bundle()
                bundle.putString("imageUri", imageUri.toString())

                val fragment = ImagePreviewFragment()
                fragment.arguments = bundle

                val transaction = supportFragmentManager.beginTransaction()
                transaction.replace(R.id.fragment_container_view, fragment)
                transaction.addToBackStack(null)
                transaction.commit()
            }
            REQUEST_IMAGE_PICK -> {
                // Обробка вибору зображення з галереї
                val imageUri = data?.data
                val bundle = Bundle()
                bundle.putString("imageUri", imageUri.toString())

                val fragment = ImagePreviewFragment()
                fragment.arguments = bundle

                val transaction = supportFragmentManager.beginTransaction()
                transaction.replace(R.id.fragment_container_view, fragment)
                transaction.addToBackStack(null)
                transaction.commit()
            }
```

```
            }
        }
    }

    private fun getImageUri(context: Context, bitmap: Bitmap): Uri {
        val bytes = ByteArrayOutputStream()
        bitmap.compress(Bitmap.CompressFormat.JPEG, 100, bytes)
        val path = MediaStore.Images.Media.insertImage(context.contentResolver, bitmap,
"Image", null)
        return Uri.parse(path)
    }

}
```

ImagePreviewFragment.kt

```
package com.example.depthvision

import android.content.Context
import android.graphics.*
import android.net.Uri
import android.os.Bundle
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.widget.Button
import android.widget.ImageView
import androidx.fragment.app.Fragment

class ImagePreviewFragment : Fragment() {

    private lateinit var imageViewPreview: ImageView
    private lateinit var btnTransformImage: Button

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
```

```
): View? {
    val view = inflater.inflate(R.layout.fragment_image_preview, container, false)

    imageViewPreview = view.findViewById(R.id.imageViewPreview)
    btnTransformImage = view.findViewById(R.id.btnTransformImage)

    val imageUriString = arguments?.getString("imageUri")
    imageUriString?.let { uriString ->
        val uri = Uri.parse(uriString)
        loadImage(uri)
    }

    btnTransformImage.setOnClickListener {
        transformImage()
    }

    return view
}

private fun loadImage(uri: Uri) {
    val inputStream = requireActivity().contentResolver.openInputStream(uri)
    val bitmap = BitmapFactory.decodeStream(inputStream)
    imageViewPreview.setImageBitmap(bitmap)
}

private fun apply3DEffects(bitmap: Bitmap): Bitmap {
    val width = bitmap.width
    val height = bitmap.height

    // Створюємо новий бітмап для ефектів
    val effectsBitmap = Bitmap.createBitmap(width, height, Bitmap.Config.ARGB_8888)
    val canvas = Canvas(effectsBitmap)

    // Малюємо оригінальне зображення
    canvas.drawBitmap(bitmap, 0f, 0f, null)
```

```
// Додаємо тіні
val shadowPaint = Paint().apply {
    color = Color.parseColor("#40000000") // Колір тіні (чорний з прозорістю)
    maskFilter = BlurMaskFilter(15f, BlurMaskFilter.Blur.NORMAL) // Розмивання
тіні
}
canvas.drawRoundRect(0f, 0f, width.toFloat(), height.toFloat(), 20f, 20f, shadowPaint)

// Додаємо контур
val outlinePaint = Paint().apply {
    color = Color.BLACK // Колір контуру
    style = Paint.Style.STROKE // Тип контуру (обведення)
    strokeWidth = 4f // Ширина контуру
}
canvas.drawRoundRect(0f, 0f, width.toFloat(), height.toFloat(), 20f, 20f, outlinePaint)

// Знаходимо найбільш поширений колір
val colorCountMap = mutableMapOf<Int, Int>()
for (x in 0 until width step 10) {
    for (y in 0 until height step 10) {
        val pixel = bitmap.getPixel(x, y)
        val color = Color.rgb(
            Color.red(pixel),
            Color.green(pixel),
            Color.blue(pixel)
        )
        if (colorCountMap.containsKey(color)) {
            colorCountMap[color] = colorCountMap[color]!! + 1
        } else {
            colorCountMap[color] = 1
        }
    }
}
```

```
            val mostCommonColor = colorCountMap.maxByOrNull { it.value }?.key ?:
Color.WHITE

            // Робимо найбільш поширений колір трохи темнішим
            val darkerColor = Color.rgb(
                (Color.red(mostCommonColor) * 0.8).toInt(),
                (Color.green(mostCommonColor) * 0.8).toInt(),
                (Color.blue(mostCommonColor) * 0.8).toInt()
            )

            // Додаємо ефект відблиску
            val gradientPaint = Paint(Paint.ANTI_ALIAS_FLAG)
            gradientPaint.shader = LinearGradient(
                0f, 0f, width.toFloat(), height.toFloat(),
                darkerColor, Color.TRANSPARENT, Shader.TileMode.CLAMP
            )
            gradientPaint.xfermode = PorterDuffXfermode(PorterDuff.Mode.SCREEN)
            canvas.drawRect(0f, 0f, width.toFloat(), height.toFloat(), gradientPaint)

            // Додаємо додаткові тіні для ефекту об'ємності
            val shadowPaint2 = Paint().apply {
                color = Color.parseColor("#20000000") // Колір тіні (чорний з прозорістю)
                maskFilter = BlurMaskFilter(25f, BlurMaskFilter.Blur.NORMAL) // Розмивання
тіні
            }
            canvas.drawRoundRect(0f, 0f, width.toFloat(), height.toFloat(), 20f, 20f, shadowPaint2)

            // Додаємо світло для підсилення ефекту об'ємності
            val highlightPaint = Paint().apply {
                color = Color.parseColor("#20FFFFFF") // Колір світла (білий з прозорістю)
                maskFilter = BlurMaskFilter(25f, BlurMaskFilter.Blur.NORMAL) // Розмивання
світла
            }
            canvas.drawRoundRect(0f, 0f, width.toFloat(), height.toFloat(), 20f, 20f, highlightPaint)
```

```
        return effectsBitmap
    }

    private fun transformImage() {
        val imageUriString = arguments?.getString("imageUri")
        imageUriString?.let { uriString ->
            val uri = Uri.parse(uriString)
            val inputStream = requireActivity().contentResolver.openInputStream(uri)
            val bitmap = BitmapFactory.decodeStream(inputStream)
            val effectsBitmap = apply3DEffects(bitmap)
            imageViewPreview.setImageBitmap(effectsBitmap)
        }
    }
}
```

Camera2Renderer.java

```
package org.opencv.android;

import java.util.Arrays;
import java.util.concurrent.Semaphore;
import java.util.concurrent.TimeUnit;
import android.annotation.TargetApi;
import android.content.Context;
import android.graphics.SurfaceTexture;
import android.hardware.camera2.CameraAccessException;
import android.hardware.camera2.CameraCaptureSession;
import android.hardware.camera2.CameraCharacteristics;
import android.hardware.camera2.CameraDevice;
import android.hardware.camera2.CameraManager;
import android.hardware.camera2.CaptureRequest;
import android.hardware.camera2.params.StreamConfigurationMap;
import android.os.Handler;
import android.os.HandlerThread;
import android.util.Log;
import android.util.Size;
```

```
import android.view.Surface;

@TargetApi(21)
public class Camera2Renderer extends CameraGLRendererBase {

    protected final String LOGTAG = "Camera2Renderer";
    private CameraDevice mCameraDevice;
    private CameraCaptureSession mCaptureSession;
    private CaptureRequest.Builder mPreviewRequestBuilder;
    private String mCameraID;
    private Size mPreviewSize = new Size(-1, -1);

    private HandlerThread mBackgroundThread;
    private Handler mBackgroundHandler;
    private Semaphore mCameraOpenCloseLock = new Semaphore(1);

    Camera2Renderer(CameraGLSurfaceView view) {
        super(view);
    }

    @Override
    protected void doStart() {
        Log.d(LOGTAG, "doStart");
        startBackgroundThread();
        super.doStart();
    }


    @Override
    protected void doStop() {
        Log.d(LOGTAG, "doStop");
        super.doStop();
        stopBackgroundThread();
    }
```

```
boolean cacPreviewSize(final int width, final int height) {
    Log.i(LOGTAG, "cacPreviewSize: "+width+"x"+height);
    if(mCameraID == null) {
        Log.e(LOGTAG, "Camera isn't initialized!");
        return false;
    }
    CameraManager manager = (CameraManager) mView.getContext()
            .getSystemService(Context.CAMERA_SERVICE);
    try {
        CameraCharacteristics characteristics = manager
                .getCameraCharacteristics(mCameraID);
        StreamConfigurationMap map = characteristics
                .get(CameraCharacteristics.SCALER_STREAM_CONFIGURATION_MAP);
        int bestWidth = 0, bestHeight = 0;
        float aspect = (float)width / height;
        for (Size psize : map.getOutputSizes(SurfaceTexture.class)) {
            int w = psize.getWidth(), h = psize.getHeight();
            Log.d(LOGTAG, "trying size: "+w+"x"+h);
            if ( width >= w && height >= h &&
                 bestWidth <= w && bestHeight <= h &&
                 Math.abs(aspect - (float)w/h) < 0.2 ) {
                bestWidth = w;
                bestHeight = h;
            }
        }
        Log.i(LOGTAG, "best size: "+bestWidth+"x"+bestHeight);
        if( bestWidth == 0 || bestHeight == 0 ||
            mPreviewSize.getWidth() == bestWidth &&
            mPreviewSize.getHeight() == bestHeight )
            return false;
        else {
            mPreviewSize = new Size(bestWidth, bestHeight);
            return true;
        }
    } catch (CameraAccessException e) {
```

```
            Log.e(LOGTAG, "cacPreviewSize - Camera Access Exception");
        } catch (IllegalArgumentException e) {
            Log.e(LOGTAG, "cacPreviewSize - Illegal Argument Exception");
        } catch (SecurityException e) {
            Log.e(LOGTAG, "cacPreviewSize - Security Exception");
        }
        return false;
    }

    @Override
    protected void openCamera(int id) {
        Log.i(LOGTAG, "openCamera");
        CameraManager manager = (CameraManager) mView.getContext().getSystemService(Context.CAMERA_SERVICE);
        try {
            String camList[] = manager.getCameraIdList();
            if(camList.length == 0) {
                Log.e(LOGTAG, "Error: camera isn't detected.");
                return;
            }
            if(id == CameraBridgeViewBase.CAMERA_ID_ANY) {
                mCameraID = camList[0];
            } else {
                for (String cameraID : camList) {
                    CameraCharacteristics characteristics = manager.getCameraCharacteristics(cameraID);
                    if( id == CameraBridgeViewBase.CAMERA_ID_BACK &&
                        characteristics.get(CameraCharacteristics.LENS_FACING) == CameraCharacteristics.LENS_FACING_BACK ||
                        id == CameraBridgeViewBase.CAMERA_ID_FRONT &&
                        characteristics.get(CameraCharacteristics.LENS_FACING) == CameraCharacteristics.LENS_FACING_FRONT) {
                        mCameraID = cameraID;
                        break;
                    }
```

```
            }
        }
        if(mCameraID != null) {
            if (!mCameraOpenCloseLock.tryAcquire(2500, TimeUnit.MILLISECONDS)) {
                throw new RuntimeException(
                        "Time out waiting to lock camera opening.");
            }
            Log.i(LOGTAG, "Opening camera: " + mCameraID);
            manager.openCamera(mCameraID, mStateCallback, mBackgroundHandler);
        }
    } catch (CameraAccessException e) {
        Log.e(LOGTAG, "OpenCamera - Camera Access Exception");
    } catch (IllegalArgumentException e) {
        Log.e(LOGTAG, "OpenCamera - Illegal Argument Exception");
    } catch (SecurityException e) {
        Log.e(LOGTAG, "OpenCamera - Security Exception");
    } catch (InterruptedException e) {
        Log.e(LOGTAG, "OpenCamera - Interrupted Exception");
    }
}

@Override
protected void closeCamera() {
    Log.i(LOGTAG, "closeCamera");
    try {
        mCameraOpenCloseLock.acquire();
        if (null != mCaptureSession) {
            mCaptureSession.close();
            mCaptureSession = null;
        }
        if (null != mCameraDevice) {
            mCameraDevice.close();
            mCameraDevice = null;
        }
    } catch (InterruptedException e) {
```

```
            throw new RuntimeException("Interrupted while trying to lock camera closing.", e);
        } finally {
            mCameraOpenCloseLock.release();
        }
    }

    private final CameraDevice.StateCallback mStateCallback = new
CameraDevice.StateCallback() {

        @Override
        public void onOpened(CameraDevice cameraDevice) {
            mCameraDevice = cameraDevice;
            mCameraOpenCloseLock.release();
            createCameraPreviewSession();
        }

        @Override
        public void onDisconnected(CameraDevice cameraDevice) {
            cameraDevice.close();
            mCameraDevice = null;
            mCameraOpenCloseLock.release();
        }

        @Override
        public void onError(CameraDevice cameraDevice, int error) {
            cameraDevice.close();
            mCameraDevice = null;
            mCameraOpenCloseLock.release();
        }

    };

    private void createCameraPreviewSession() {
        int w=mPreviewSize.getWidth(), h=mPreviewSize.getHeight();
        Log.i(LOGTAG, "createCameraPreviewSession("+w+"x"+h+")");
```

```
            if(w<0 || h<0)
               return;
            try {
               mCameraOpenCloseLock.acquire();
               if (null == mCameraDevice) {
                  mCameraOpenCloseLock.release();
                  Log.e(LOGTAG, "createCameraPreviewSession: camera isn't opened");
                  return;
               }
               if (null != mCaptureSession) {
                  mCameraOpenCloseLock.release();
                  Log.e(LOGTAG, "createCameraPreviewSession: mCaptureSession is already
started");
                  return;
               }
               if(null == mSTexture) {
                  mCameraOpenCloseLock.release();
                  Log.e(LOGTAG, "createCameraPreviewSession: preview SurfaceTexture is null");
                  return;
               }
               mSTexture.setDefaultBufferSize(w, h);

               Surface surface = new Surface(mSTexture);

               mPreviewRequestBuilder = mCameraDevice
                     .createCaptureRequest(CameraDevice.TEMPLATE_PREVIEW);
               mPreviewRequestBuilder.addTarget(surface);

               mCameraDevice.createCaptureSession(Arrays.asList(surface),
                     new CameraCaptureSession.StateCallback() {
                        @Override
                        public void onConfigured( CameraCaptureSession cameraCaptureSession) {
                           mCaptureSession = cameraCaptureSession;
                           try {
```

```
                        mPreviewRequestBuilder.set(CaptureRequest.CONTROL_AF_MODE,
CaptureRequest.CONTROL_AF_MODE_CONTINUOUS_PICTURE);
                        mPreviewRequestBuilder.set(CaptureRequest.CONTROL_AE_MODE,
CaptureRequest.CONTROL_AE_MODE_ON_AUTO_FLASH);


mCaptureSession.setRepeatingRequest(mPreviewRequestBuilder.build(),          null,
mBackgroundHandler);
                        Log.i(LOGTAG, "CameraPreviewSession has been started");
                    } catch (CameraAccessException e) {
                        Log.e(LOGTAG, "createCaptureSession failed");
                    }
                    mCameraOpenCloseLock.release();
                }

                @Override
                public void onConfigureFailed(
                        CameraCaptureSession cameraCaptureSession) {
                    Log.e(LOGTAG, "createCameraPreviewSession failed");
                    mCameraOpenCloseLock.release();
                }
            }, mBackgroundHandler);
        } catch (CameraAccessException e) {
            Log.e(LOGTAG, "createCameraPreviewSession");
        } catch (InterruptedException e) {
            throw new RuntimeException(
                    "Interrupted while createCameraPreviewSession", e);
        }
        finally {
            //mCameraOpenCloseLock.release();
        }
    }

    private void startBackgroundThread() {
        Log.i(LOGTAG, "startBackgroundThread");
```

```
        stopBackgroundThread();
        mBackgroundThread = new HandlerThread("CameraBackground");
        mBackgroundThread.start();
        mBackgroundHandler = new Handler(mBackgroundThread.getLooper());
    }

    private void stopBackgroundThread() {
        Log.i(LOGTAG, "stopBackgroundThread");
        if(mBackgroundThread == null)
            return;
        mBackgroundThread.quitSafely();
        try {
            mBackgroundThread.join();
            mBackgroundThread = null;
            mBackgroundHandler = null;
        } catch (InterruptedException e) {
            Log.e(LOGTAG, "stopBackgroundThread");
        }
    }

    @Override
    protected void setCameraPreviewSize(int width, int height) {
        Log.i(LOGTAG, "setCameraPreviewSize("+width+"x"+height+")");
        if(mMaxCameraWidth > 0 && mMaxCameraWidth < width) width =
mMaxCameraWidth;
        if(mMaxCameraHeight > 0 && mMaxCameraHeight < height) height =
mMaxCameraHeight;
        try {
            mCameraOpenCloseLock.acquire();

            boolean needReconfig = cacPreviewSize(width, height);
            mCameraWidth  = mPreviewSize.getWidth();
            mCameraHeight = mPreviewSize.getHeight();

            if( !needReconfig ) {
```

```
                mCameraOpenCloseLock.release();
                return;
            }
            if (null != mCaptureSession) {
                Log.d(LOGTAG, "closing existing previewSession");
                mCaptureSession.close();
                mCaptureSession = null;
            }
            mCameraOpenCloseLock.release();
            createCameraPreviewSession();
        } catch (InterruptedException e) {
            mCameraOpenCloseLock.release();
            throw new RuntimeException("Interrupted while setCameraPreviewSize.", e);
        }
    }
}
```

Додаток Г – Графічна частина

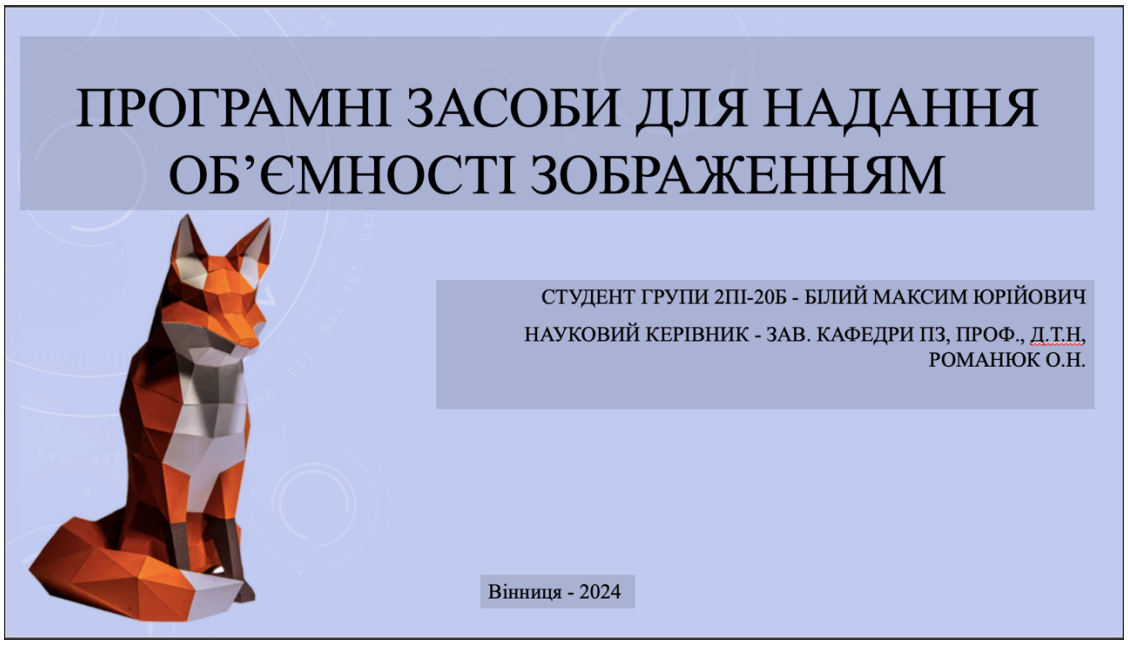


Рисунок Г.1 – Назва роботи

Рисунок Г.2 – Галузі застосування

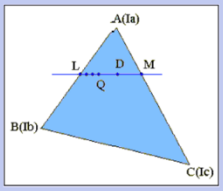
МЕТА ТА ЗАВДАННЯ ДОСЛІДЖЕННЯ

- **Мета та завдання дослідження.** Метою роботи є підвищення продуктивності формування тривимірних зображенням за рахунок розробки нових методів, використання сучасних бібліотек.
- **Об'єкт дослідження** – процес надання об'ємності зображенням у системах комп'ютерної графіки.
- **Предмет дослідження** – методи та засоби надання об'ємності зображенням.
- **Задачі дослідження:**
- Провести аналіз існуючих методів і засобів надання об'ємності зображенням.
- Розробити модифікований метод ray-tracing для врахування відблисків на поверхні моделі.
- Розробити метод розбиття поверхні моделі на трикутники для ідентифікації відблисків.
- Розробити програмні засоби для надавання об'ємності зображенням.
- Провести тестування розроблених методів з метою отримання порівняльних оцінок.

Рисунок Г.3 – Мета та завдання дослідження

РЕНДЕРИНГ

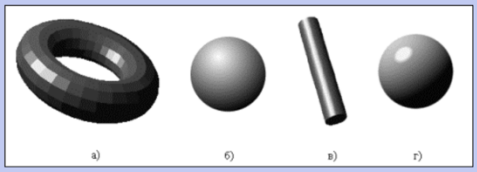
Процес відтворення (візуалізації) об'єкта або сцени називається рендерингом (від англійського "rendering" – відтворення, передача візуалізації).



$\Delta I_{AB} = \frac{I_A - I_B}{БП_{AB}}$ - Приріст інтенсивності світла вздовж ребра AB;

$\Delta I_{AC} = \frac{I_A - I_C}{БП_{AC}}$ - Приріст інтенсивності світла вздовж ребра AC;

$\Delta I_{BC} = \frac{I_B - I_C}{БП_{BC}}$ - Приріст інтенсивності світла вздовж ребра BC;



Приклади синтезу об'ємних фігур: а) однотонове заповнення; б) зафарбовування методом Гуро; в,г) зафарбовування методом Фонга

$I_M = I_A + DI_{AB} \cdot БП_{AM}$ – Інтенсивність точки M

$I_L = I_A + DI_{AC} \cdot БП_{AL}$ – Інтенсивність точки L

$DI_{LM} = (I_L - I_M)/LM$ – збільшення інтенсивності уздовж прямої LM (у даному випадку $LM = БП_{LM}$);

Інтенсивність світла в точці D знайдемо у такий спосіб:

$I_D = I_L + DI_{LM} \cdot LD$ – інтенсивність шуканої точки.

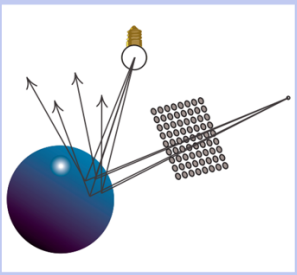
$I_Q = I_L + \Delta I_{LM} + \Delta I_{LM} + \Delta I_{LM}$

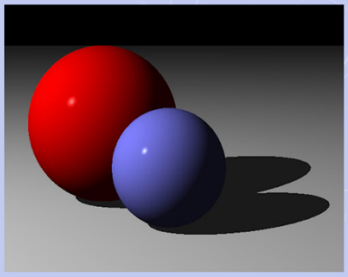
Рисунок Г.4 – Рендеринг

RAY-TRACING

Основні аспекти рей-трейсингу:

- 1. Відбиття світла
- 2. Променева тінь
- 3. Відблиски
- 4. Розсіювання світла





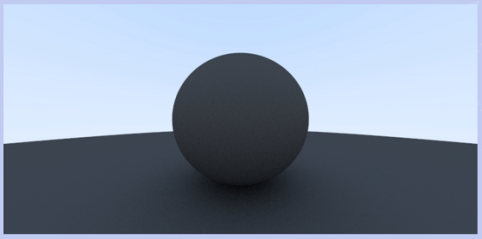


Рисунок Г.5 – Ray-tracing

МОДИФІКОВАНИЙ МЕТОД RAY-TRACING ДЛЯ ВРАХУВАННЯ ВІДБЛИСКІВ НА ПОВЕРХНІ МОДЕЛІ

Основні етапи методу:

1)Аналіз кольору

2) Матеріали

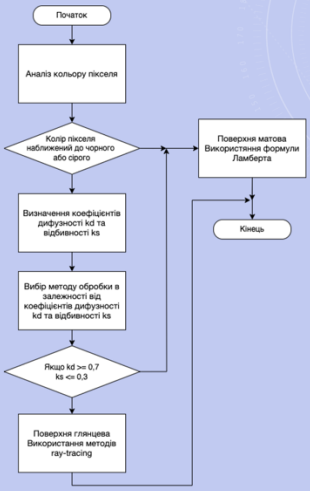
3) Формула Ламберта: $I = k_d*(N*L)*I_{світла}$, де N – вектор нормалі до поверхні, L – вектор напрямку світла.

4)Відбиття

5)Рекурсивне відбиття

Порівняння різних типів матеріалів

Матеріал	kd	ks	Тип поверхні
Метал	0.6	0.4	Блискуча
Дерево	0.8	0.2	Матова
Скло	0.2	0.8	Блискуча
Пластик	0.7	0.3	Матова
Бетон	0.9	0.1	Матова
Поліуретан	0.3	0.7	Блискуча
Текстиль	0.85	0.15	Матова
Папір	0.95	0.05	Матова
Мармур	0.4	0.6	Блискуча
Глина	0.75	0.25	Матова
Шкіра	0.65	0.35	Матова
Вода	0.1	0.9	Блискуча
Гума	0.5	0.5	Матова
Пісок	0.8	0.2	Матова
Шовк	0.7	0.3	Матова



Граф-схема алгоритму аналізу матеріалу

Рисунок Г.6 – Модифікований метод ray-tracing

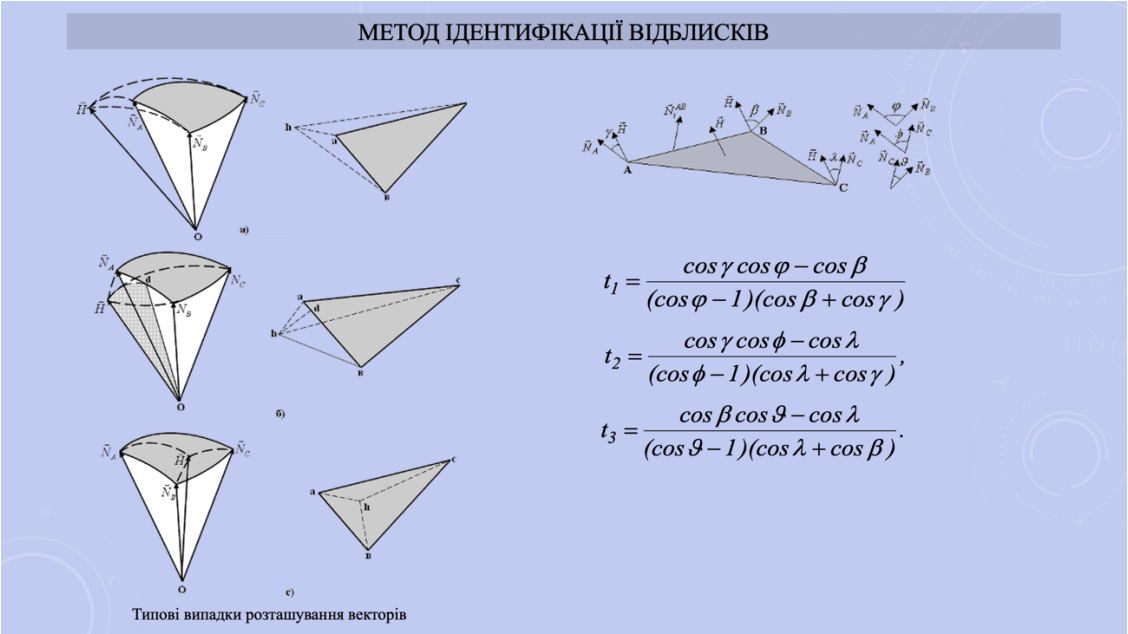


Рисунок Г.7 – Метод ідентифікації відблисків 1 слайд

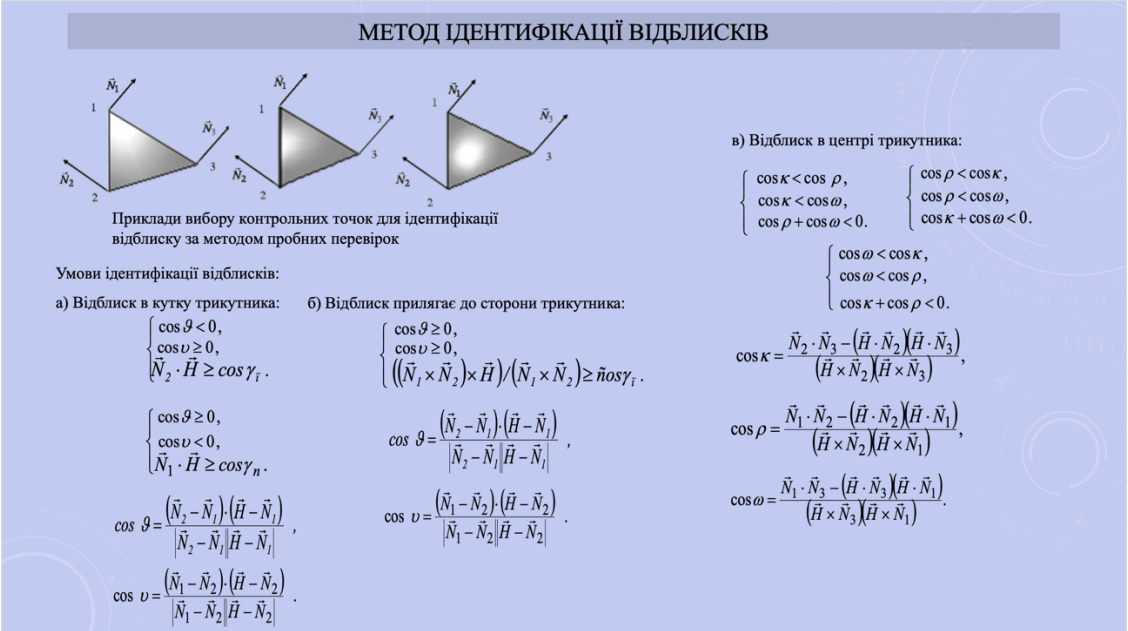


Рисунок Г.8 – Метод ідентифікації відблисків 2 слайд

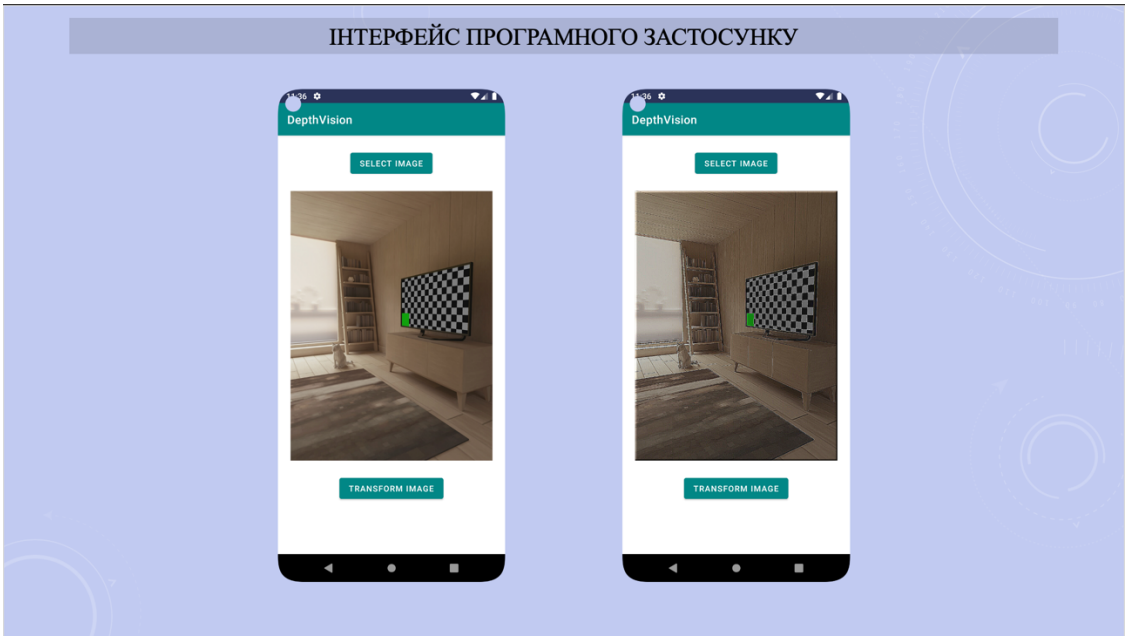


Рисунок Г.9 – Інтерфейс програмного застосунку

Рисунок Г.10 – Закінчення презентації