Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра програмного забезпечення

БАКАЛАВРСЬКА ДИПЛОМНА РОБОТА

на тему: Розробка методів і програмних засобів для високопродуктивного формування зображень з використанням методу Гуро

Виконав: студент IV курсу, групи 2ПІ-20б

спеціальності

121 – Інженерія програмного забезпечення

(шифр і назва напряму підготовки, спеціальності)

Комісарик Артур Сергійович

(прізвище та ініціали)

Керівник: д.т.н., професор каф. ПЗ Романюк О.Н.

(прізвище та ініціали)

Рецензент: к.т.н., доцент каф. ОТ Черняк О.І.

(прізвище та ініціали)

Допущено до захисту

Зав. кафедри ПЗ О.Н. Романюк

«___» _____ 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., професор
О.Н. Романюк _____
«12» березня 2024 р.

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

Комісарику Артуру Сергійовичу

1. Тема роботи – розробка методів і програмних засобів для високопродуктивного формування зображень з використанням методу Гуро.

Керівник роботи: Романюк Олександр Никифорович, д.т.н., професор, завідувач кафедри ПЗ, затверджені наказом вищого навчального закладу від 11 бе́резня 2024 р. № 80.

2. Строк подання студентом роботи 3 червня 2024.

3. Вихідні дані до роботи: базовий метод зафарбовування – метод Гуро; модель освітлення – модель Бліна; кольоровий режим TrueColor; вихідні дані для формування зображень за методом Гуро – координати вершин трикутників, інтенсивності кольору у вершинах трикутника.

4. Зміст текстової частини: вступ, аналіз методу Гуро, аналіз методів рендерингу, аналіз аналогів, визначення ключових аспектів та етапів процесу формування зображень за методом Гуро, розробка методу зафарбовування тривимірної моделі за Гуро, вибір методів оптимізації формування зображень за Гуро, розробка методу підвищення продуктивності рендерингу зображень, розробка моделі та алгоритмів роботи програмного забезпечення, розробка модуля зафарбовування тривимірної моделі за методом Гуро, розробка модуля високопродуктивного формування зображень, тестування програмного забезпечення, розробка інструкції користувача, висновки, список використаних джерел, додатки 63.

5. Перелік ілюстративного матеріалу: титульний аркуш, аналіз методів рендерингу, галузі застосування, мета та задачі дослідження, розробка методів

для високопродуктивного формування зображень за методом Гуро, граф-схеми алгоритмів, інтерфейс програмного забезпечення.

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 дипломна роботи	Строк виконання етапів роботи	Примітка
1	Аналіз розвитку технологій формування зображень з використанням методу Гуро	14.03.24 – 22.03.24	Вик.
2	Розробка методів для формування зображень з використанням методу Гуро	22.03.24 – 19.04.24	Вик.
3	Розробка алгоритмів роботи програмного забезпечення	19.04.24 – 10.05.24	Вик.
4	Розробка модулів програмного засобу	10.05.24 – 24.05.24	Вик.
5	Тестування програми	24.05.24 – 27.05.24	Вик.
6	Оформлення матеріалів до захисту БДР	27.05.24 – 03.06.24	Вик.

Студент _____ **А. С. Комісарик**
(підпис) (ініціали та прізвище)

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

АНОТАЦІЯ

УДК 00492

Комісарик А. С. Розробка методів і програмних засобів для високопродуктивного формування зображень з використанням методу ГУРО : бакалаврська дипломна робота зі спеціальності 121 – інженерія програмного забезпечення, освітня програма – інженерія програмного забезпечення. Вінниця: ВНТУ, 2024. 67 с.

На укр. мові. Бібліогр. : 20 назв ; рис. : 40; табл. 3.

У бакалаврській дипломній роботі було розроблено методи та програмні засоби для високопродуктивного формування зображень за методом Гуро. Було виконано обґрунтування розробки власних методів формування зображень за методом Гуро з метою підвищення продуктивності. Розроблено методи високопродуктивного формування зображень за методом Гуро використовуючи засоби тривимірного моделювання та розроблені методи, що дозволяють ефективно використовувати обчислювальні ресурси та зменшити час обробки графічних зображень

На основі отриманих в бакалаврській дипломній роботі теоретичних положень запропоновано алгоритми та розроблено програмні засоби для високопродуктивного формування зображень шляхом зафарбовування тривимірних об’єктів за методом Гуро.

Розроблені засоби вимірювання антропометричних параметрів можна використати для побудови високопродуктивних систем рендерингу.

Ключові слова: тривимірне моделювання, метод Гуро, рендеринг, продуктивність, програмне забезпечення.

ABSTRACT

UDC 00492

Komisaryk A. S. Development of methods and software tools for high-performance image formation using the GURO method : bachelor's thesis in specialty 121 - Software Engineering, educational program - Software Engineering. Vinnytsia: VNTU, 2024. 67 c.

In Ukrainian. Bibliography: 20 titles; Figures: 40; Table 3.

In the bachelor's thesis, methods and software tools for high-performance image formation using the Gouraud method were developed. The development of own methods of Gouraud imaging was justified in order to increase productivity. Methods for high-performance Gouraud imaging using three-dimensional modeling tools have been developed and methods have been developed that allow efficient use of computing resources and reduce the time of graphic image processing.

On the basis of the theoretical positions obtained in the bachelor's thesis, algorithms were proposed and software tools were developed for high-performance image formation by painting three-dimensional objects using the Gouraud method.

The developed tools for measuring anthropometric parameters can be used to build high-performance rendering systems.

Keywords: three-dimensional modeling, Gouraud method, rendering, performance, software.

ЗМІСТ

ВСТУП

Обґрунтування вибору теми дослідження.

У зв’язку із розвитком вимог до програмного забезпечення та швидкості формування зображень, у сучасному світі велика увага приділяється розвитку методів обробки та формування зображень. Особливо це стосується галузі комп'ютерної графіки та обробки зображень, де існує великий попит на програмні засоби, що забезпечують швидку та ефективну генерацію тривимірних зображень високої якості. Цей попит обумовлений потребою в реалістичних візуальних ефектах у відеоіграх, анімації, а також у наукових та медичних додатках, де важливо точно та швидко обробляти великі обсяги тривимірних даних [1].

На сьогоднішній день існує ряд технічних викликів, пов'язаних з формуванням зображень, що вимагають нових підходів та інструментів для їх вирішення. Велика увага приділяється підвищенню якості, продуктивності та швидкості формування зображень, а також зменшенню обсягу необхідних ресурсів.

Один з основних процесів у формуванні тривимірних зображень – це рендеринг [2]. Цей процес включає в себе складні операції, такі як розрахунок нормалізованих векторів, визначення кривизни поверхні та розрахунок складових кольору. Для досягнення реалістичності зображень використовуються традиційні методи рендерингу, які часто мають обмеження щодо швидкості та якості результатів. Використання методу ГУРО дозволяє подолати ці обмеження, забезпечуючи високу точність та продуктивність обробки зображень.

Метод Гуро (Gouraud shading) – [3] це техніка зафарбовування поверхні в комп'ютерній графіці, яка використовується для створення візуально реалістичних зображень. У цьому методі кожна вершина моделі має власний колір, а інтерполяція кольорів відбувається між цими точками. Особливість методу Гуро полягає в тому, що кольори визначаються для кожної вершини, і

потім вони інтерполюються вздовж кожного полігону на основі цих кольорів [4-5]. Однак існують певні технічні виклики, пов'язані з розробкою програмних засобів для високопродуктивного формування зображень з використанням методу ГУРО.

Метод Гуро отримав широке поширення в засобах тривимірної графіки. При формуванні динамічних зображень та зображень в інтерактивному режимі необхідна висока продуктивність зафарбовування. Тому питання підвищення продуктивності за методом Гуро є актуальним.

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

Мета та завдання дослідження. Метою дослідження є підвищення продуктивності методу Гуро, що дозволить ефективно використовувати обчислювальні ресурси та зменшити час обробки графічних зображень, що дає можливість формувати динамічні зображення в реальному маштабі часу.

Відповідно до поставленої мети в бакалаврській дипломній роботі потрібно вирішити такі **задачі**:

- виконати аналіз методу Гуро і засобів зафарбовування поверхонь тривимірних об’єктів;
- розробити методи підвищення продуктивності рендерингу Гуро;
- розробити алгоритм зафарбовування тривимірних моделей на основі методу Гуро;
- розробити алгоритми рендерингу, які забезпечать високу продуктивність у процесі формування зображень за Гуро;
- покращення процесу формування зображень за методом Гуро із забезпеченням високої продуктивності.
- тестування розробленого програмного застосунку.

Об'єкт дослідження – процес високопродуктивного формування зображень з використанням методу Гуро.

Предмет дослідження – методи та засоби високопродуктивного формування зображень з використанням методу Гуро.

Методи дослідження. У процесі дослідження застосовувалися: теорія чисел і чисельних методів; теорія алгоритмів; теорія інтерполювання функції; методи тривимірного моделювання, комп'ютерне моделювання для аналізу та перевірки отриманих теоретичних положень.

Наукова новизна отриманих результатів:

1. Вперше запропоновано метод високопродуктивного формування зображень за методом Гуро, особливість якого полягає у встановленні сталості приросту інтенсивності кольору для всіх рядків растеризації, що дозволить підвищити ефективність зафарбовування тривимірних моделей.

2. Вперше запропоновано метод розпаралелення зафарбовування за Гуро, особливість якого полягає у виконанні додаткової тріангуляції для отримання трикутників однакової площі з їх подальшим кореляційним тонуванням.

Практичне значення отриманих результатів полягає у розробці програмного забезпечення, яке дозволить формувати зображення високої якості за допомогою використання методу Гуро.

Особистий внесок здобувача. Усі наукові результати отримано здобувачем самостійно. У праці, опублікованих у співавторстві, студенту належать: [6] – проаналізовано використання методу Гуро у процесі формування тривимірних зображень.

Апробація матеріалів бакалаврської дипломної роботи. Основні положення БДР доповідалися та обговорювалися на Міжнародних конференціях: Матеріали міжнародної науково-практичної інтернет-конференції студентів аспірантів та молодих науковців «Молодь в науці: дослідження, проблеми, перспективи (МН-2024)» (м. Вінниця, 2024).

Публікації. За темою бакалаврської роботи надруковано 1 праця у матеріалах міжнародної конференції.

Структура та обсяг БДР. Бакалаврська дипломна робота складається зі вступу, чотирьох розділів, висновків, списку використаних джерел і додатків, у які винесено технічне завдання, протокол перевірки на плагіат, лістинг програмного забезпечення й ілюстративний матеріал до захисту роботи.

1 АНАЛІЗ РОЗВИТКУ ТЕХНОЛОГІЙ ФОРМУВАННЯ ЗОБРАЖЕНЬ З ВИКОРИСТАННЯМ МЕТОДУ ГУРО

1.1 Аналіз методів рендерингу

Рендеринг є критично важливим етапом візуалізації комп'ютерної графіки, що включає створення зображень з 3D-моделей. Існують численні методи рендерингу, кожен з яких має свої переваги та недоліки. Класичні методи, такі як растеризація і трасування променів, були домінуючими протягом останніх десятиліть.

Растеризація, як найпоширеніший метод рендерингу, має значний вплив на відтворення графіки у відеоіграх та інших реально-часових додатках. Цей підхід передбачає конвертацію тривимірних об'єктів у двовимірне зображення, розраховуючи кольори пікселів на основі їх положення та властивостей поверхонь. Однією з ключових переваг растеризації є її висока швидкість виконання, що забезпечує плавний рендеринг навіть у вимогливих графічних сценах. Проте, варто відзначити, що растеризація також має свої обмеження. У складних сценах з великою кількістю джерел світла та тіней, цей метод може призвести до недоліків у якості зображення. Наприклад, артефакти відсутності деталей або ненатуральні переходи між кольорами можуть виникнути при рендерингу складних сцен. Такі обмеження можуть впливати на реалізм відображення та сприйняття графіки користувачем. Незважаючи на ці обмеження, растеризація залишається популярним вибором завдяки своїй швидкості та ефективності, особливо в контексті відеоігор та інших додатків, де час відгуку грає критичну роль. Для досягнення оптимальних результатів у рендерингу, розробники постійно вдосконалюють техніки оптимізації та використовують додаткові методи, такі як розподілені обчислення та технології апаратного прискорення.

Трасування променів (ray tracing) є іншим підходом, який надає значно кращу якість зображення за рахунок точнішого моделювання поведінки світла.

Цей метод слідує шляхом світлових променів від камери до джерел світла, обчислюючи відбивання, заломлення та розсіювання на поверхнях об'єктів. Незважаючи на високу якість результатів, трасування променів вимагає значних обчислювальних ресурсів, що обмежує його застосування в реальному часі.

Рендеринг Гуро (Gouraud shading) є одним з найбільш відомих і широко використовуваних методів інтерполяційного затінення, призначених для підвищення якості зображень при мінімальних витратах ресурсів. Метод Гуро був розроблений Анрі Гуро у 1971 році і призначений для більш реалістичного рендерингу полігональних моделей. Гладке затінення Гуро полягає в інтерполяції кольорів та інтенсивностей світла між вершинами полігонів, що дозволяє створювати більш плавні переходи між кольорами на поверхні об'єктів. На кожній вершині обчислюється інтенсивність освітлення на основі нормалі вершини та розташування джерел світла. Ці інтенсивності потім інтерполюються вздовж ребер і всередині полігонів для кожного пікселя. Основною перевагою методу Гуро є його обчислювальна ефективність. Інтерполяція інтенсивностей замість обчислення освітлення для кожного пікселя значно знижує обсяг необхідних обчислень. Проте, метод має і свої недоліки. Наприклад, він може створювати артефакти на великих полігонах або при різких змінах освітлення, оскільки інтерполяція не враховує зміну геометрії всередині полігона.

Рендеринг Фонга (Phong shading) є вдосконаленням методу Гуро, розробленим Буй-Туан Фонгом. Він також використовує інтерполяцію, але замість інтерполяції кольорів інтерполюються нормалі поверхні. Для кожного пікселя обчислюється освітлення на основі інтерпольованих нормалей, що дозволяє отримати більш точні результати. Основною перевагою методу Фонга є його здатність створювати реалістичніші зображення, зокрема для блискучих поверхонь, завдяки кращій обробці відблисків. Цей метод також вирішує проблему артефактів, характерних для методу Гуро, забезпечуючи більш гладке і точне відображення освітлення на криволінійних поверхнях. Недоліком методу Фонга є підвищена обчислювальна складність порівняно з методом Гуро, що може вплинути на продуктивність в реальному часі.

Плоске затінення (Flat shading) є найпростішим методом затінення, при якому кожен полігон відображається з єдиним кольором. Інтенсивність світла обчислюється для однієї точки (зазвичай центру або вершини полігона) і застосовується до всього полігона. Плоске затінення має кілька переваг, серед яких висока швидкість виконання та простота реалізації. Однак, воно призводить до низької якості зображення, оскільки переходи між полігонами виглядають різкими і ненатуральними. Цей метод підходить для застосувань, де обчислювальні ресурси обмежені, і висока якість не є пріоритетом, наприклад, в попередній візуалізації або на старих апаратних платформах.

Кожен з розглянутих методів рендерингу має свої області застосування залежно від вимог до якості зображення та обчислювальних ресурсів. Растеризація залишається основним методом для реального часу завдяки своїй швидкості, тоді як трасування променів використовується в додатках, де якість зображення є критичною, таких як кінематографічна графіка та візуалізація для дизайну.

Метод Гуро є компромісним рішенням, яке дозволяє досягти кращої якості зображення в порівнянні з простою растеризацією, але без значного збільшення обчислювальних витрат. Він ідеально підходить для застосувань, де потрібна вища якість рендерингу при обмежених ресурсах, таких як мобільні ігри та інтерактивні додатки. Метод Фонга підходить для задач, що потребують високої якості зображення, зокрема для відображення блискучих поверхонь і складних світлових ефектів.

Отже, у сучасних умовах, де продуктивність та якість зображення є однаково важливими, метод Гуро залишається актуальним. Розробка нових методів та програмних засобів, що базуються на цих методах, може значно покращити ефективність рендерингу. Інтеграція цих методів з сучасними обчислювальними можливостями та технологіями, такими як паралельні обчислення та апаратні прискорювачі, дозволяє досягати високоякісних результатів у реальному часі.

1.2 Аналіз методу Гуро

У сучасному світі комп'ютерна графіка стає все більш популярною і використовується у різних галузях, включаючи бізнес, медицину, рекламу та індустрію розваг. Однією з ключових складових комп'ютерної графіки є тривимірна (3D) графіка, яка дозволяє відтворювати об'єкти у тривимірному просторі для отримання більш реалістичного зображення.

Використання тривимірної графіки надає більшу інтерактивність та реалізм графічним додаткам. Для відображення тривимірного об'єкта на двовимірному екрані необхідно провести процес, який називається рендерингом. Під час рендерингу кожен піксель зображення обчислюється для відтворення коліру та адреси, створюючи ілюзію об'ємності на плоскому екрані.

В сучасних графічних програмах найбільш поширеним є використання трикутників для представлення тривимірних об'єктів. Кожен трикутник визначається своїми вершинами та нормалями, які використовуються для обчислення інтенсивності кольору вздовж ребер. Одним з методів зафарбовування трикутників на тривимірній моделі є метод Гуро, який дозволяє отримати реалістичне зображення за рахунок інтерполяції кольорів між вершинами полігонів [6].

Метод ГУРО є технікою зафарбовування поверхні в комп'ютерній графіці, що дозволяє створювати візуально реалістичні зображення. У цьому методі кожна вершина моделі має свій власний колір, а інтерполяція кольорів відбувається між цими точками. Це дозволяє отримати плавне переходи кольорів і надає більшу реалістичність зображенням.

Головною перевагою використання методу Гуро для формування зображень є можливість отримання більш реалістичних зображень. Тривимірні моделі об'єктів дозволяють точніше відтворювати їхню форму, текстуру та колір, що призводить до створення більш живого та реалістичного відображення. Це особливо корисно в галузях, де потрібно детально досліджувати об'єкти або створювати візуально привабливі сцени, таких як відеоігри, анімація та медичні

додатки. Крім того, тривимірні моделі також дозволяють включати різноманітні типи даних, такі як геометричні виміри, текстури, колірні дані тощо. Це дозволяє отримувати більш комплексну інформацію про об'єкт та використовувати різноманітні методи аналізу, такі як аналіз освітлення, тіней та взаємодії об'єктів.

Розглянемо процес зафарбування трикутника на тривимірній моделі, за допомогою методу Гуро. Припустимо, що ми маємо трикутник з вершинами, координати яких позначені як A, B і C. Крім того, в кожній з цих вершин задаються значення інтенсивності світла: відповідно I_A, I_B, I_C. На основі цих значень ми розраховуємо інтенсивність кольору для всіх внутрішніх точок трикутника. Процес полягає у визначенні приросту інтенсивності світла на один піксель уздовж ребер трикутника AB, BC і AC:

$$\Delta I_{AB} = \frac{(I_A - I_B)}{\text{БП}_{AB}} \tag{1.1}$$

$$\Delta I_{AC} = \frac{(I_A - I_C)}{\text{БП}_{AC}} \tag{1.2}$$

$$\Delta I_{BC} = \frac{(I_B - I_C)}{\text{БП}_{BC}} \tag{1.3}$$

, де БП - більший з приростів координат Δx, Δy обраного відрізка прямої.

Розглянемо лінію растеризації, що проходить через точку D (рис. 1.1). Кінцевими точками цієї лінії будуть точки L і M. Визначимо прирости інтенсивності вздовж ребер. Для розрахунку інтенсивності кольору в точках M і L використаємо наступні формули:

$$I_M = I_A + \Delta I_{AB} * \text{БП}_{AM} \tag{1.4}$$

$$I_L = I_A + \Delta I_{AC} * \text{БП}_{AL} \tag{1.5}$$

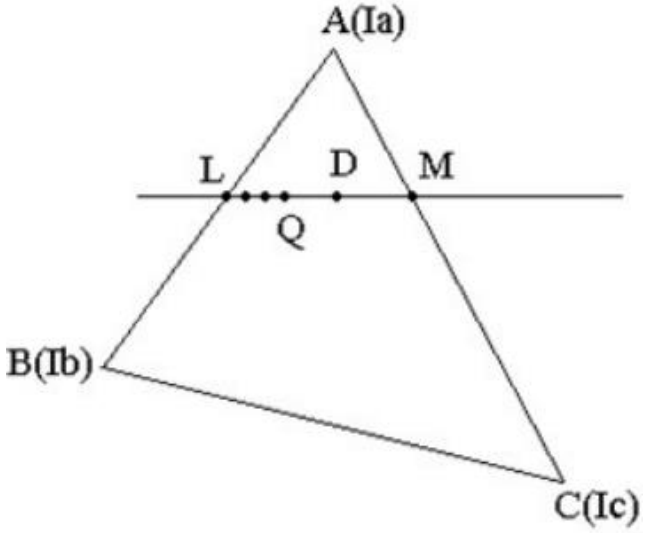


Рисунок 1.1 – Інтерполяція трикутника

Розрахуємо приріст інтенсивності вздовж лінії растеризації LM. Для цього використаємо наступну формулу:

$$\Delta I_{LM} = \frac{(I_L - I_M)}{|\overrightarrow{LM}|} \tag{1.6}$$

Наступним кроком знайдемо інтенсивність усіх точок, що знаходяться на лінії растеризації. Для прикладу знайдено інтенсивність кольору точки D:

$$I_D = I_L + \Delta I_{LM} * |\overrightarrow{LD}| \tag{1.7}$$

Інтенсивність будь-якої іншої точки, обмеженої заданим полігоном, визначається аналогічним чином. Найменша похибка досягається при розбивці поверхні на прямокутні трикутники з катетами, паралельними координатним осям, оскільки в цьому випадку найбільший приріст катета збігається з його довжиною.

При зафарбуванні всього трикутника сканувальна лінія послідовно проходить усі точки полігона. Приріст уздовж сканувальної лінії та між сусідніми сканувальними лініями в цьому випадку складає один піксель. Це

дозволяє замінити операцію множення операцією накопичувального додавання.

$$I_Q = I_L + 3 * \Delta I_{LM} \quad (1.8)$$

На рисунку 1.2, а наведено приклад рендерингу трикутника при I_A=5, I_B=9, I_C=1. Значення інтенсивностей світла отримані за описаними вище формулами (рис. 1.2 б).

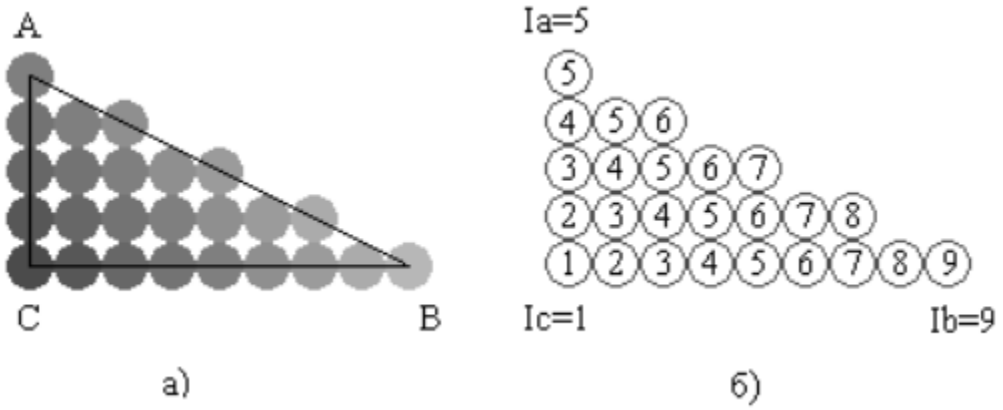


Рисунок 1.2 – Приклад зафарбовування трикутника

Хоча метод ГУРО надає багато переваг у формуванні зображень, існують деякі технічні виклики, пов'язані з його використанням. Одним із головних викликів є забезпечення швидкодії та ефективності обробки графічних об'єктів, особливо при роботі з великими обсягами даних чи в реальному часі. Це стає особливо важливим у вимірюваннях, де потрібно обробляти великі обсяги тривимірних даних з високою швидкістю та точністю.

На сьогоднішній день існують різні програмні засоби та бібліотеки для формування зображень, але не всі з них надають достатньо точного та ефективного функціоналу для використання методу ГУРО. Більшість існуючих рішень є платними, що ускладнює доступність для широкого кола користувачів.

Отже, аналіз стану технологій формування зображень з використанням методу ГУРО підтверджує важливість розробки високопродуктивного алгоритму зафарбовування тривимірних об’єктів за методом ГУРО. Розробка

програмних засобів для впровадження високопродуктивного формування зображень з використанням методу ГУРО стає необхідністю для покращення швидкості та продуктивності процесу рендерингу зображень.

1.3 Аналіз аналогів

У останні роки спостерігається зростаюча популярність технологій комп'ютерної графіки та рендерингу зображень з методом ГУРО. Розробка програмних програмних засобів для високопродуктивного формування зображень з використанням методу ГУРО забезпечить ефективність та швидкість рендерингу зображень високої роздільної якості та допоможе знизити вимоги до апаратних вимог. До найбільш відомих рішень відносять:

- Blender;
- Autodesk Maya;
- Cinema 4D;
- Unity;
- 3D Max.

Одним із прикладів використання методу ГУРО для формування зображень є програмний застосунок Blender (рис. 1.3) – [7] програмний засіб для моделювання, анімації та рендерингу тривимірних об'єктів.

Рисунок 1.3 – Приклад роботи застосунку Blender

Однією з його переваг є потужна та безкоштовна платформа для тривимірного моделювання, анімації та рендерингу. Крім того, Blender має активну спільноту користувачів, що сприяє обміну досвідом та навчанню. Однак для новачків може бути складним у використанні через велику кількість функцій та складний інтерфейс.

Інший приклад – застосунок Autodesk Maya (рис. 1.4), є однією з провідних програм для тривимірного моделювання та рендерингу. Вона використовується в багатьох галузях, включаючи кіноіндустрію та відеоігрову розробку [8]. Однією з переваг є її інтеграція з іншими програмними продуктами Autodesk, що спрощує роботу в команді. Проте, вона має високу ціну та вимоги до обладнання.

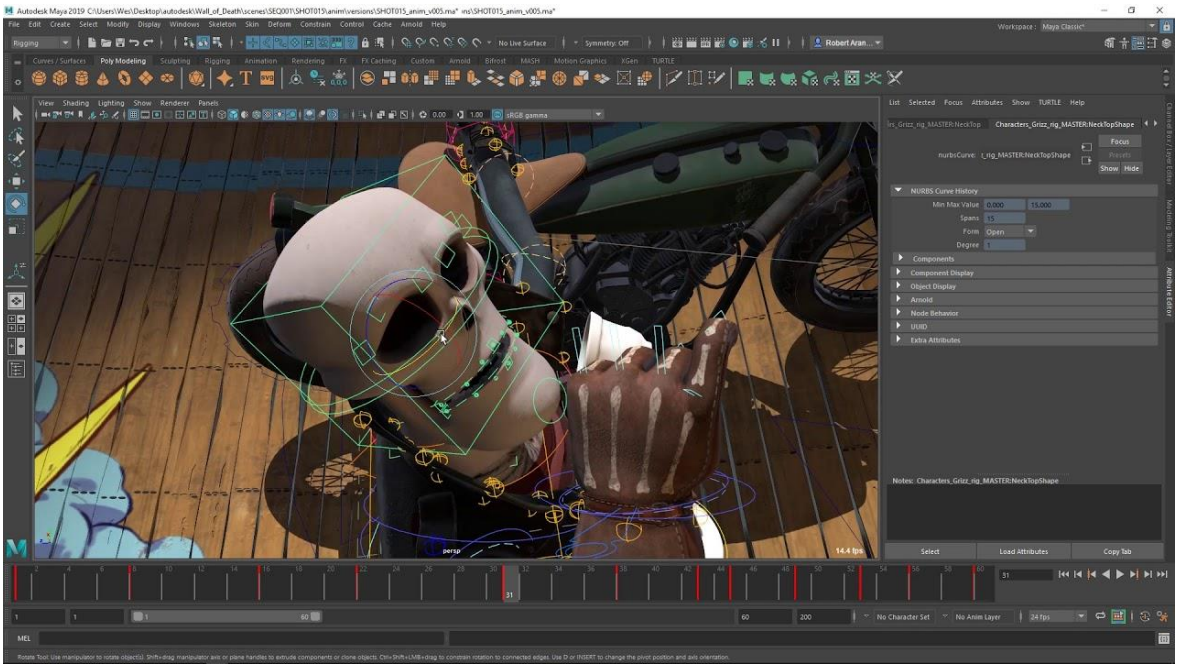


Рисунок 1.4 – Приклад роботи застосунку Autodesk Maya

Cinema 4D (рис. 1.5) є іншим популярним програмним засобом для тривимірного моделювання та анімації [9]. Даний програмний продукт відомий своєю легкістю використання та дружнім інтерфейсом, що робить його популярним серед початківців та професіоналів. Користувачі можуть швидко освоїти основні функції Cinema 4D та створювати якісні тривимірні моделі. Він також має велику кількість плагінів та розширень, що розширюють можливості

програми. Проте, в порівнянні з іншими програмами, Cinema 4D може бути менш потужним рішенням для задач рендерингу тривимірних сцен та обмеженою у функціональності, особливо для складних проектів, де потрібні великі обсяги обробки даних та додаткові можливості редагування.

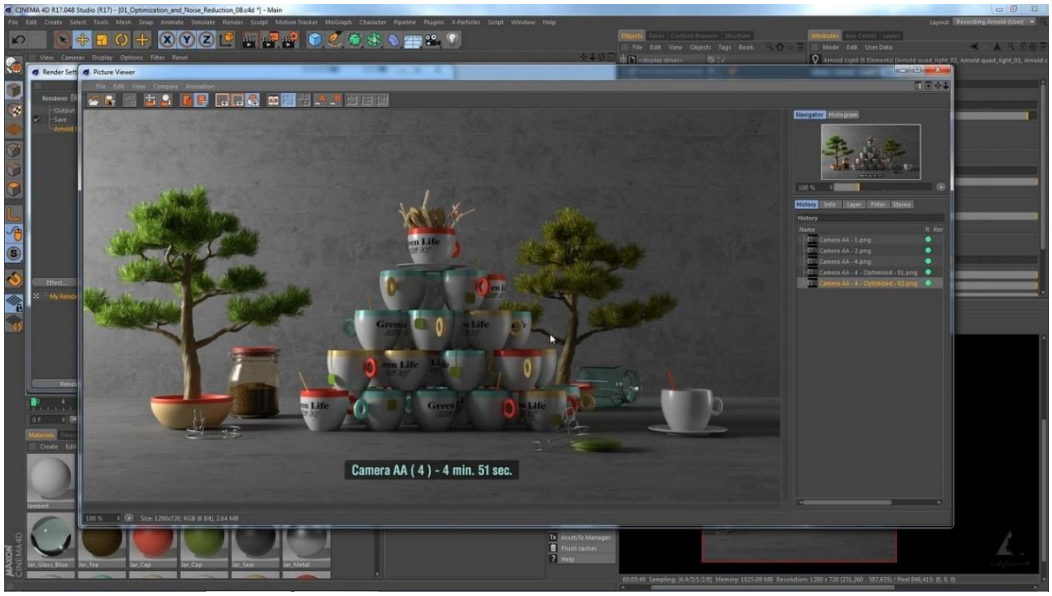


Рисунок 1.5 – Приклад роботи застосунку Cinema 4D

Unity (рис. 1.6) – [10] це ігровий двигун, але він також може бути використаний для тривимірного моделювання та рендерингу.

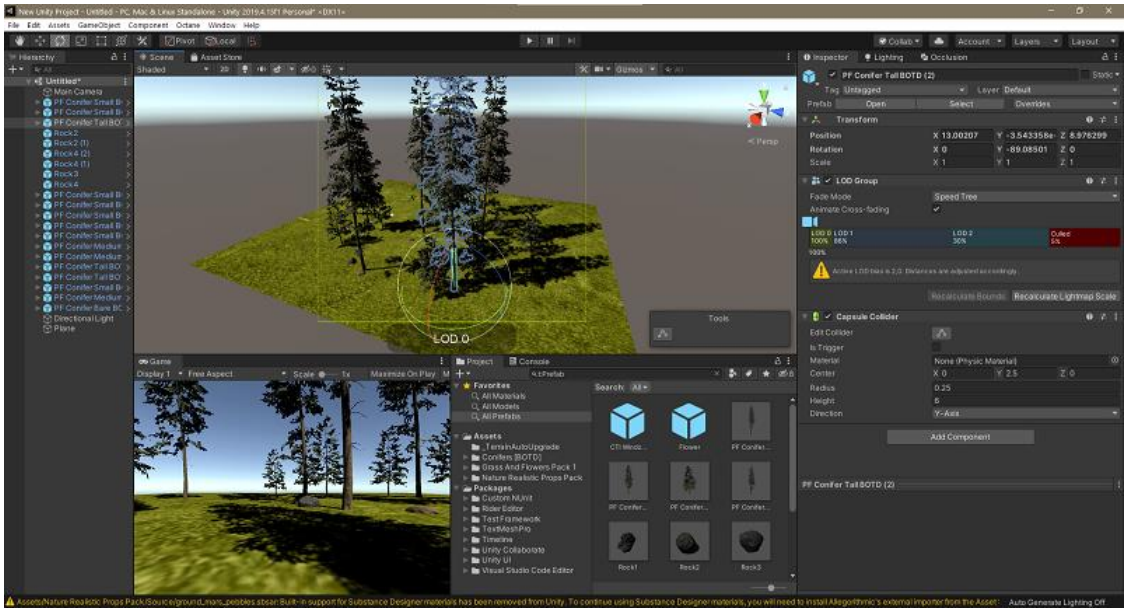


Рисунок 1.6 – Приклад роботи застосунку Unity

Він має власні інструменти для створення реалістичних тривимірних зображень, в тому числі підтримку методу ГУРО. Однак, порівняно з іншими програмами, Unity має обмежені можливості для складних тривимірних сцен.

3D Max (рис. 1.7) – [11] це потужний інструмент для тривимірного моделювання та рендерингу, що використовується в різних галузях, включаючи архітектуру, дизайн та ігрову розробку. Він має велику кількість інструментів та плагінів, що дозволяє реалізувати різноманітні проекти. Однак, він поширюється на платній основі та вимагає відповідного обладнання для ефективної роботи. Незважаючи на це, його висока функціональність та гнучкість роботи зробили його одним з найпопулярніших програмних засобів у сфері тривимірного моделювання і візуалізації.

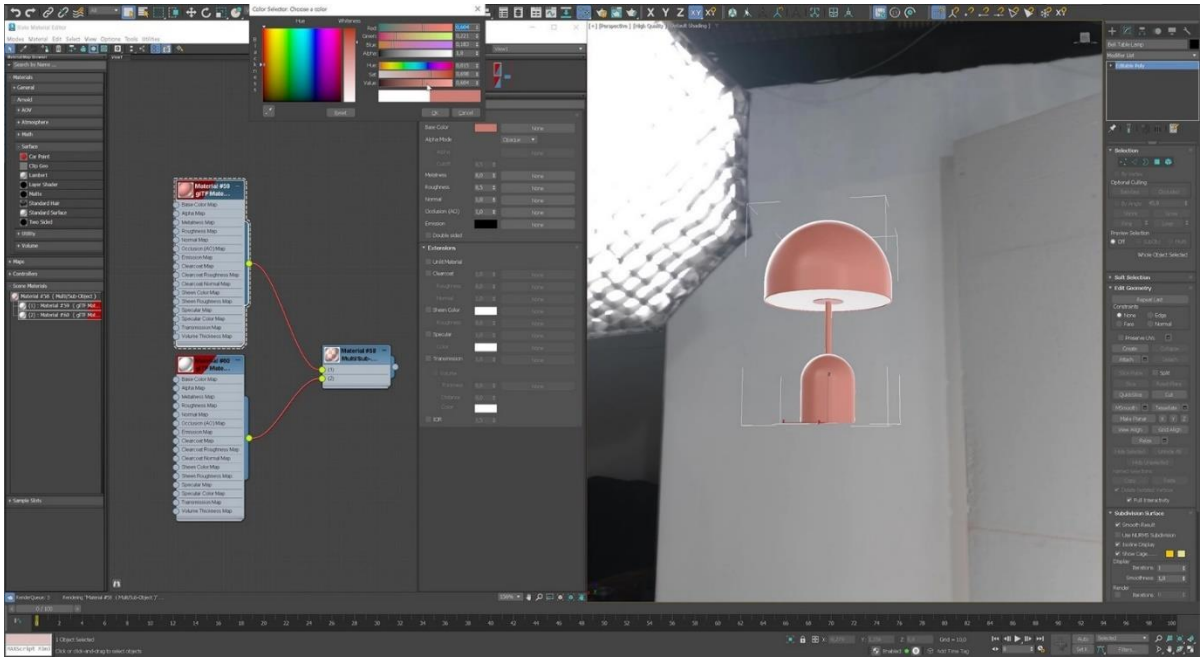


Рисунок 1.7 – Приклад роботи застосунку 3D Max

Розробка програмних засобів високопродуктивного формування зображень з використанням методу ГУРО вимагає досвіду роботи з тривимірним моделюванням, рендерингом, зафарбовування поверхонь тривимірних об’єктів, розробки програмного забезпечення та дизайну інтерфейсу користувача. Це передбачає створення внутрішньої системи, яка буде здатна зафарбовувати поверхні тривимірних об'єктів з використанням методу ГУРО та врахування

освітлення, а також виконувати високопродуктивне формування зображень. Складова інтерфейсу користувача передбачає розробку інтерфейсів, які дозволять користувачу взаємодіяти з розробленими модулями.

Результати порівняння аналогів зведено в табл. 1.1.

Таблиця 1.1 – Порівняльна характеристика аналогів

Критерії	Blender	Autodesk Maya	Cinema 4D	Unity	3D Max	Власний модуль
Безкоштовна ліцензія	+	-	-	+	-	+
Необхідність додаткового навчання для використання	-	-	-	-	-	+
Можливість рендерингу	+	+	+	-	+	+
Можливість зміни параметрів через засоби інтерфейсу	+	+	+	-	+	+
Низькі вимоги до апаратного забезпечення	-	-	-	-	-	+
Загальна оцінка	60%	40%	40%	20%	40%	100%

Відповідно до таблиці порівняльних характеристик розробка власних програмних засобів для високопродуктивного формування зображень з використанням методу ГУРО є доцільною. Отриманий продукт зможе покрити недоліки існуючих рішень та забезпечити розширені можливості для формування зображень тривимірних моделей з використанням методу ГУРО.

Проаналізувавши переваги та недоліки існуючих програмних засобів для високопродуктивного формування зображень з використанням методу ГУРО, було визначено наступні завдання, які необхідно виконати для успішної розробки програмного забезпечення:

- розробити алгоритм високопродуктивного формування зображень, що базується на методі ГУРО та забезпечить високу точність та роздільну здатність зображень;
- розробити модуль для зафарбовування тривимірної моделі за методом ГУРО, який буде спроможний використовувати тривимірні геометричні моделі;
- розробити модуль для високопродуктивного формування зображень, який буде спроможний проводити процес рендерингу тривимірного зображення;
- розробити модуль графічного інтерфейсу, який буде зручним у використанні для користувачів, забезпечуючи зручність і швидкість взаємодії з програмними засобами;
- виконати інтеграцію різних модулів в єдиний програмний засіб, який буде використовуватись для високопродуктивного формування зображень з використанням методу ГУРО.
- реалізувати програмний продукт з використанням методу ГУРО;
- автоматизація процесу формування зображень, забезпечуючи високу роздільну здатність та продуктивність і швидкість рендерингу;
- провести тестування програмного забезпечення згідно поставлених задач.

Отже, розробка програмних засобів високопродуктивного формування зображень з використанням методу ГУРО відкриває широкі можливості для покращення швидкості, якості роздільної здатності та зниження вимог до апаратних ресурсів. Завдяки точним та деталізованим тривимірним моделям тіла людини, медичні фахівці можуть здійснювати більш точне та індивідуалізоване вимірювання параметрів тіла, що є критичним для правильної діагностики та лікування захворювань.

1.4 Концептуальні основи процесу формування зображень

Метод Гуро (або алгоритм Гуро) є одним із методів комп'ютерної графіки,

використовуваним для створення реалістичних зображень шляхом апроксимації поверхонь об'єктів і обчислення їхнього освітлення. Визначення ключових аспектів та етапів процесу формування зображень за методом Гуро є важливою складовою розробки методів та програмних засобів для високопродуктивного формування зображень [12].

Першим ключовим аспектом є моделювання геометричних об'єктів. Це включає в себе створення математичних моделей поверхонь об'єктів, які складаються з вершин, ребер і граней. Крім того, для точного відображення форми об'єктів можуть використовуватися різноманітні методи, такі як криві Безьє, B-сплайни та інші.

Другий аспект полягає у реалізації алгоритму Гуро для обчислення інтенсивності освітлення в кожній точці поверхні. При розробці програмних засобів для високопродуктивного формування зображень за методом Гуро, потрібно враховувати ефективність алгоритму, адаптуючи його до конкретних умов застосування. Це включає оптимізацію обчислень, використання спеціалізованих структур даних для зберігання і обробки інформації про поверхні об'єктів та їхнє освітлення, а також розробку паралельних алгоритмів для використання багатоядерних архітектур.

Третій аспект – це управління освітленням і тінями. Для отримання реалістичних зображень необхідно враховувати вплив різних джерел світла на об'єкти та їхнє оточення. Це включає в себе моделювання різних типів світла (наприклад, директних, розсіяних, відбитих), а також розрахунок тіней, які кастують об'єкти один на одного та на оточуючі поверхні. Управління освітленням та тінями відіграє важливу роль у створенні реалістичних ілюстрацій.

Четвертий аспект – це оптимізація та підвищення продуктивності процесу рендерингу. При роботі з великими об'ємами даних та складними сценами, швидкодія алгоритмів має велике значення. Розробники повинні удосконалювати алгоритми та програмні засоби для забезпечення швидкості та ефективності обробки графічних об'єктів, використовуючи, наприклад, методи

оптимізації пам'яті та ресурсів, паралельні обчислення та використання апаратного прискорення.

П’ятий аспект – це зведення результуючого зображення. Після обчислення освітлення для кожної точки поверхні об'єктів і застосування інших ефектів, таких як тіні, текстури і т. д., отримується результуюче зображення. Цей процес може включати такі етапи, як зведення різних шарів зображення, фільтрація та підготовка для відображення на екрані.

Під час розробки методів та програмних засобів для високопродуктивного формування зображень за методом Гуро, необхідно враховувати не лише базові аспекти, а й ключові етапи процесу. Крім вже згаданих аспектів, ще декілька етапів важливі для успішного реалізації цього методу [13]:

1. Розрахунок нормалей до поверхонь об'єктів: цей етап передбачає визначення нормалей до кожної точки поверхні об'єкта. Для цього можуть використовуватися методи чисельного диференціювання або аналітичні формули, залежно від складності геометричних об'єктів. Формально, нормаль до поверхні в кожній точці може бути обчислена як векторна сума добутку векторів, які лежать в площині грані та нормалі до цієї грані.

2. Інтерполяція нормалей на гранях: після отримання нормалей до кожної точки поверхні, необхідно визначити нормали на гранях об'єктів. Це досягається шляхом інтерполяції нормалей вздовж граней від вершин до центру грані. Інтерполяція може проводитися за допомогою різних методів, таких як лінійна або квадратична інтерполяція, в залежності від потреб конкретного застосування.

3. Обчислення інтенсивності освітлення: після визначення нормалей до кожної точки поверхні та на гранях, можна переходити до обчислення інтенсивності освітлення в кожній точці. Цей процес включає в себе врахування різних факторів, таких як напрямок до джерел світла, кут між нормаллю та напрямом світла, а також властивості матеріалу поверхні.

4. Застосування моделей освітлення: на цьому етапі використовуються різні моделі освітлення, для обчислення інтенсивності освітлення. Ці моделі

враховують різні аспекти освітлення, такі як розсіяне, відбите та поглинуте світло, що дозволяє отримати більш реалістичні зображення.

5. Рендеринг: після отримання тривимірної моделі об'єкта і врахування його освітлення, виконується процес рендерингу. Цей етап включає процес перетворення тривимірної геометрії в двовимірне зображення, що складається з пікселів або точок, які зазвичай описуються за допомогою кольору та глибини.

6. Застосування ефектів та підготовка для відображення: на цьому етапі застосовуються різноманітні ефекти, такі як антиаліасинг, м'яке тіньове розмиття, глибина різкості тощо. Також виконується підготовка зображення для відображення на екрані або друку.

Узагальнюючи, ключові етапи процесу формування зображень за методом Гуро включають в себе розрахунок нормалей до поверхні об'єктів, інтерполяцію нормалей на гранях, обчислення інтенсивності освітлення та застосування моделей освітлення.

1.5 Висновки

У першому розділі було виконано аналіз методу Гуро та методів рендерингу тривимірних зображень, що показали потребу у розробці нових методів та програмних засобів, що будуть реалізувати високопродуктивне формування зображень з використанням методу Гуро.

Також було розглянуто стан програмних засобів для високопродуктивного формування зображень з використанням методу ГУРО. Були розглянуті програмні засоби такі як Blender, Autodesk Maya, Cinema 4D, Unity, 3D Max. Кожен з цих засобів має свої переваги і недоліки, що були проаналізовані.Проаналізувавши переваги та недоліки існуючих програмних засобів, було сформульовано завдання для подальшої розробки власного програмного забезпечення для формування зображень за Гуро. Було проаналізовано та виокремлено основні етапи та аспекти формування зображень за методом Гуро.

2 РОЗРОБКА МЕТОДІВ ДЛЯ ФОРМУВАННЯ ЗОБРАЖЕНЬ З ВИКОРИСТАННЯМ МЕТОДУ ГУРО

2.1 Розробка методу зафарбовування тривимірної моделі за Гуро

Розробка методу зафарбовування тривимірної моделі за Гуро включає кілька ключових етапів, які спрямовані на створення реалістичного візуального ефекту. Початково необхідно здійснити обробку тривимірної моделі, що включає в себе перетворення геометричних форматів та оптимізацію їх для подальшого використання.

Одним з важливих кроків є необхідність тріангуляції моделі, оскільки метод зафарбовування за Гуро базується на розбитті поверхні на трикутники. Це дозволяє виконати обчислення освітлення та затемнення для кожного трикутника окремо, що значно спрощує процес обробки.

Схема інтенсивності-інтерполяції, розроблена Гуро і зазвичай називається зафарбовуванням Гуро, відтворює поверхню полігону шляхом лінійної інтерполяції значення інтенсивності по поверхні. Значення інтенсивності для кожного полігону координуються зі значеннями інтенсивності сусідніх полігонів уздовж спільних ребер, таким чином усуваючи розриви інтенсивності, які можуть виникати при пласкому зафарбовуванні [14].

Поверхня кожного полігону тривимірної моделі рендериться за допомогою зафарбовування Гуро шляхом виконання наступних обчислень:

- визначення середнього одиничного вектора нормалі в кожній вершині полігону;
- застосування моделі освітлення до кожної вершини для визначення її інтенсивності;
- лінійна інтерполяція вершинних інтенсивностей кольору по поверхні полігону;
- у кожній вершині багатокутника отримуємо вектор нормалі, усереднюючи нормалі до поверхні всіх багатокутників, що дивляться на цю

вершину, як показано на рис 2.1.

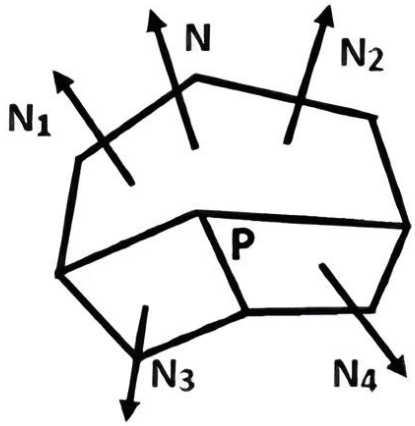


Рисунок 2.1 – Усереднені вектори нормалей

Таким чином, для довільного положення вершини V на тривимірній моделі, отримуємо одиничну нормаль до вершини, відповідно до формули:

$$N_v = \frac{\sum_{k=1}^{n} N_k}{|\sum_{k=1}^{n} N_k|} \tag{2.1}$$

Отримавши вершинні нормалі певного полігону, визначимо інтенсивність у вершинах за допомогою моделі освітлення:

$$\mathrm{I}_4 = \frac{y_4 - y_2}{y_1 - y_2} I_1 + \frac{y_1 - y_4}{y_1 - y_2} I_2 \tag{2.2}$$

Отримавши формулу визначення інтенсивності у точці, необхідним кроком є визначення інтерполяції інтенсивностей кольору вздовж ребер полігону на тривимірній моделі (рис. 2.2). Для цього необхідно обробляти кожен полігон окремо використовуючи процеси сканування та визначення інтенсивності кольору відповідно до моделі освітлення. Для кожної лінії сканування інтенсивність на перетині лінії сканування з краєм полігону лінійно інтерполюється від інтенсивності в кінцевих точках країв.

Наприклад на рисунку 2.2 ребро багатокутника з кінцевими вершинами в

позиціях 1 і 2 перетинається лінією сканування в точці 4. Швидкий метод отримання інтенсивності в точці 4 полягає в інтерполяції між інтенсивністю I_1 та I_2, використовуючи лише вертикальне зміщення лінії розгортки. Аналогічно, інтенсивність на правому перетині цієї лінії сканування (точка 5) інтерполюється зі значень інтенсивності у вершинах 2 і 3.

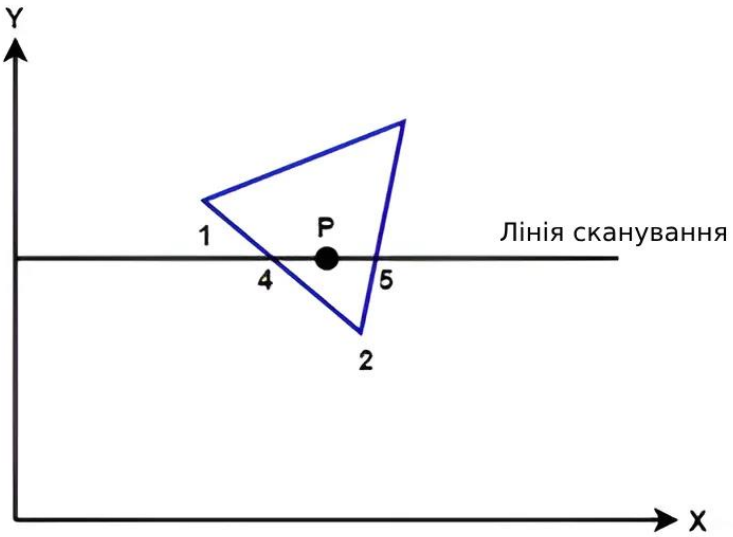


Рисунок 2.2 – Інтерполяції інтенсивностей вздовж ребер полігону

Після того, як ці граничні значення інтенсивності визначено для лінії сканування, внутрішня точка (наприклад, точка P) інтерполюється від граничних значень інтенсивності в точках 4 і 5 як:

$$I_P = \frac{x_5 - x_P}{x_5 - x_4} I_4 + \frac{x_P - x_4}{x_5 - x_4} I_5 \qquad (2.3)$$

Інкрементні обчислення використовуються для отримання послідовних значень інтенсивності країв між лініями сканування та для отримання послідовних значень інтенсивності вздовж лінії сканування, як показано на рисунку 2.3.

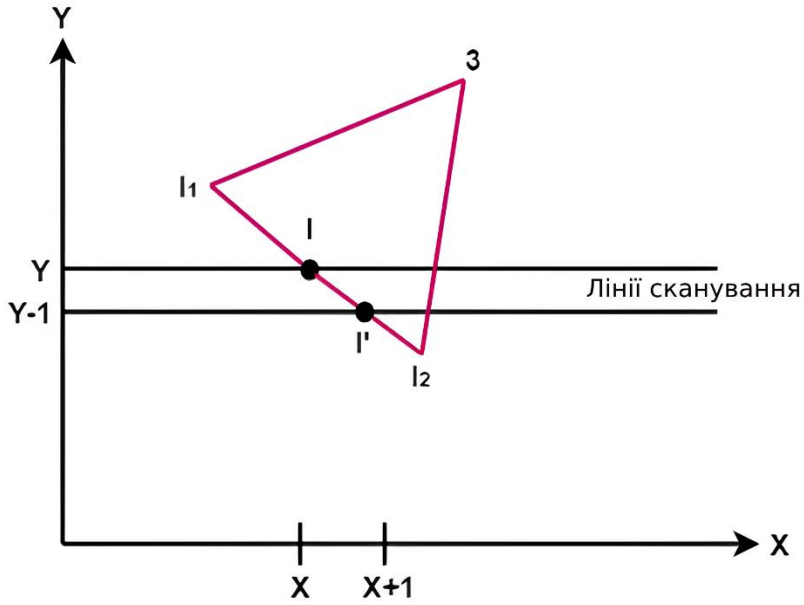


Рисунок 2.3 – Інкрементна інтерполяція значення інтенсивності вздовж краю полігону для послідовних ліній сканування.

Звідси, можна вивести формулу для обчислення інтенсивості кольору за методом Гуро на краю полігона(x, y):

$$\mathrm{I} = \frac{y - y_2}{y_1 - y_2} I_1 + \frac{y_1 - y}{y_1 - y_2} I_2 \qquad (2.4)$$

Отримавши формулу для визначення інтенсивності кольору на краю полігона, необхідної дією є отримання інтенсивністі вздовж цього краю для наступної лінії сканування (Y-1). Для цього використаємо наступну формулу:

$$I^1 = I + \frac{I_2 - I_1}{y_1 - y_2} \qquad (2.5)$$

Аналогічні обчислення використовуються для отримання інтенсивностей у послідовних горизонтальних позиціях пікселів уздовж кожної лінії сканування.

Якщо поверхня має бути відтворена в кольорі, інтенсивність кожного компонента кольору обчислюється у вершинах. Зафарбовування Гуро можна поєднати з алгоритмом прихованих поверхонь для заповнення видимих

полігонів уздовж кожної лінії розгортки. Приклад об'єкта, затіненого методом Гуро, показано на наступному рисунку:

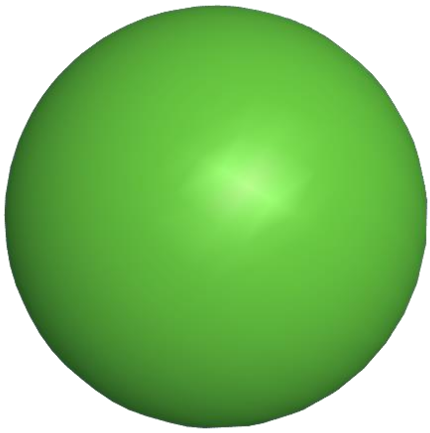


Рисунок 2.4 – Зафарбування 3Д моделі за методом Гуро

Затінення Гуро усуває розриви інтенсивності, пов'язані з моделлю постійного затінення, але має деякі інші недоліки. Відблиски на поверхні іноді мають аномальну форму, а лінійна інтерполяція інтенсивності може спричинити появу на поверхні яскравих або темних смуг інтенсивності, які називаються смугами збігу (Match bands).

2.2 Модифікації зафарбовування зображень за методом Гуро

Метод зафарбовування тривимірної моделі за Гуро, відомий також як інтерполяційне зафарбовування за Гуро, широко застосовується в комп'ютерній графіці для забезпечення плавних переходів кольорів на поверхнях полігонів. Основна концепція методу полягає у визначенні кольорів на вершинах полігонів та їх поступовій інтерполяції всередині полігону.

Теоретичні дослідження показали [15], що під час зафарбовування за Гуро зміни інтенсивності кольору ΔI_r та ΔI_b, як уздовж растрових ліній, так і по краях полігонів, залишаються постійними величинами. Це означає, що немає потреби повторно обчислювати ці зміни для кожної растрової лінії. Далі розглянемо

взаємозв'язок між змінами інтенсивності кольору вздовж країв полігонів та растрових ліній.

$$\Delta I_{\Gamma} = \frac{(I_{\mathrm{A}} - I_{\mathrm{C}}) * \Delta Y_{BC} - (I_B - I_{\mathrm{C}}) * \Delta Y_{AC}}{\Delta X_{AC} * \Delta Y_{BC} - \Delta X_{BC} * \Delta Y_{AC}} \quad (2.6)$$

$$\Delta I_{\mathrm{B}} = \frac{(I_B - I_{\mathrm{C}}) * \Delta Y_{BC} - (I_A - I_{\mathrm{C}}) * \Delta Y_{AC}}{\Delta X_{AC} * \Delta Y_{BC} - \Delta X_{BC} * \Delta Y_{AC}} \quad (2.7)$$

$$\Delta I_{\mathrm{B}} = \Delta I_{\Gamma} + \Delta I_{\mathrm{B}} \quad (2.8)$$

Було встановлено, що зміни інтенсивностей ΔI_H, ΔI_V та ΔI_D в горизонтальному, вертикальному та діагональному напрямках відповідно залишаються постійними. Інтенсивності кольорів у вершинах трикутника позначаються як I_{A}, I_B, I_C, що ілюструється на рисунку 2.5.

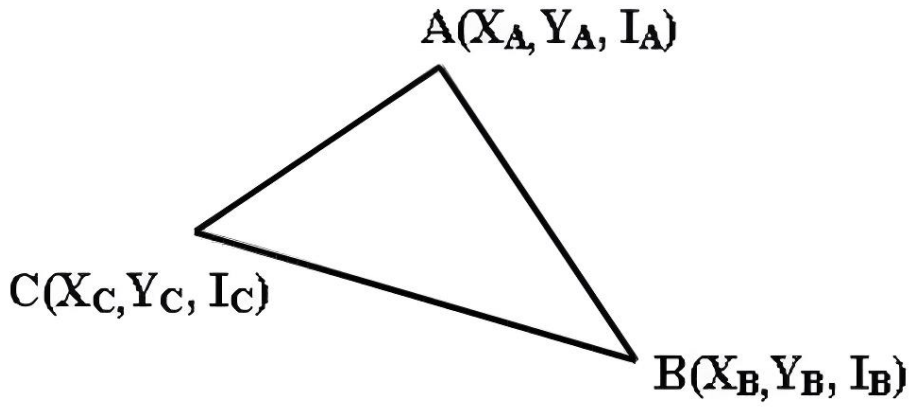


Рисунок 2.5 – Схема інтенсивності кольорів

Отримана властивість дає змогу розпаралелити рендеринг за методом Гуро використовуючи тріангуляцію Серпінського першого порядку. Для цього початковий трикутник ділиться на чотири частини за серединами кожного ребра, як показано на рисунку 2.6. У результаті отримуємо три ідентичні трикутники та один дзеркально відображений.

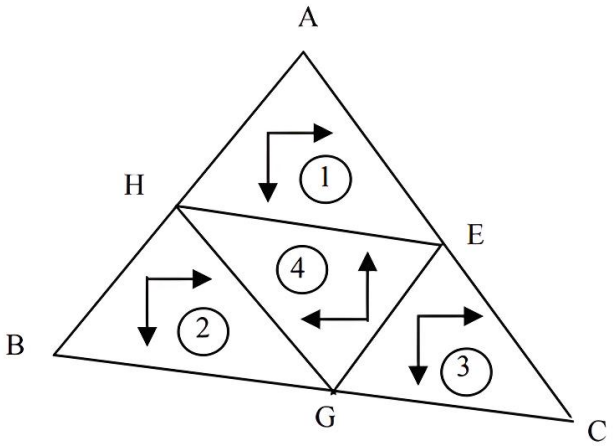


Рисунок 2.6 – Напрямки затінення трикутників

Беручи до уваги, що зміна інтенсивності кольору є сталою по всьому трикутнику, ми можемо виконати затінення шляхом обчислення лише для одного з нових трикутників, а потім застосувати ці результати до інших. Тобто, необхідно провести растеризацію тільки для верхнього трикутника 1, а потім повторити аналогічні кроки для всіх інших внутрішніх трикутників. Важливо зазначити, що трикутники 1, 2 і 3 мають однаковий напрямок зафарбовування, тоді як трикутник 4 зафарбовується в протилежному напрямку.

Перед початком растеризації слід визначити всі координати вершин трикутників. Аналогічно, код інтерполяції інтенсивності кольору для внутрішніх точок лівого верхнього трикутника можна використати для всіх інших трикутників, але з урахуванням їхніх відповідних вершин. Оскільки тріангуляція початкового трикутника здійснюється шляхом бісектрис його ребер, а зміна сторін вихідного трикутника не завжди є парним числом, виникає проблема розбиття трикутника без утворення нерастеризованих точок. Були розглянуті різні варіанти розбиття та обрано оптимальний [16].

Другий підхід для підвищення продуктивності зафарбовування за методом Гуро базується на окремому обробленні парних і непарних точок у кожному рядку сканування. У циклі підготовки інтенсивності кольорів I_1, I_2 обчислюються для першої та наступної точок на лінії сканування. Інтенсивності $I_{\text{РБ}}$ для перших точок рядків визначаються шляхом інтерполяції інтенсивностей кольорів вздовж базових ребер, тоді як I_2 обчислюється за допомогою наступної

формули:

$$I_2 = I_{\text{РБ}} + \Delta I_{\Gamma} \tag{2.9}$$

Інший метод покращення рендерингу за методом Гуро полягає у виявленні відблисків на межах трикутників та їх подальшій обробці. Припустимо, що вектори нормалі $\overrightarrow{N_0}$, $\overrightarrow{N_1}$, $\overrightarrow{N_2}$ визначені для вершин трикутника з координатами (x_0, y_0), (x_1, y_1), (x_2, y_2).

У комп'ютерній графіці часто вважають, що джерело світла і спостерігач розташовані на нескінченності, що означає, що вектор $\vec{L}$ має однаковий напрямок для всіх точок трикутника. Доведено, що в цьому випадку максимальні значення інтенсивності кольору знаходяться в точках на межах трикутника, які задовольняють рівнянню:

$$t_1 = \frac{\cos\gamma\cos\phi - \cos\beta}{(\cos\phi - 1)(\cos\beta + \cos\gamma)}, \tag{2.10}$$

$$t_2 = \frac{\cos\gamma\cos\phi - \cos\lambda}{(\cos\phi - 1)(\cos\lambda + \cos\gamma)}, \tag{2.11}$$

$$t_3 = \frac{\cos\beta\cos\vartheta - \cos\lambda}{(\cos\vartheta - 1)(\cos\beta + \cos\lambda)} \tag{2.12}$$

Відповідність кутів між векторами нормалей зображено на рисунку 2.7.

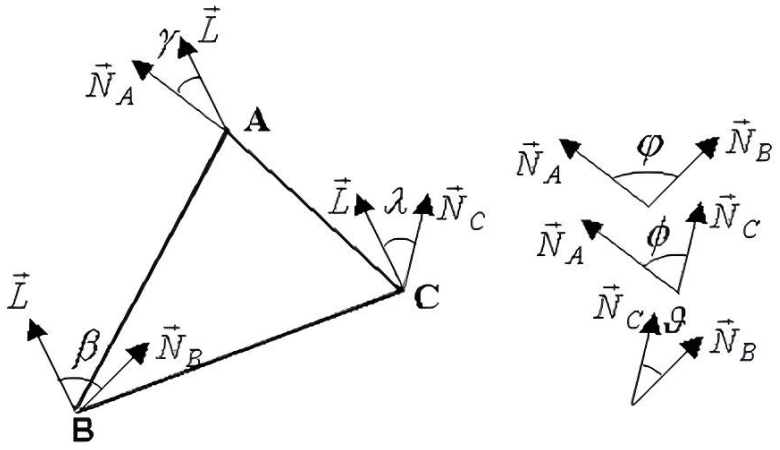


Рисунок 2.7 – Вектори нормалі трикутника ABC

Змінні t_1, t_2, t_3 є параметричними і змінюються в інтервалі від 0 до 1. Початковий трикутник можна розбити на декілька залежно від результату порівняння порогових значень з інтенсивністю кольору в точках t_1, t_2, t_3. Отримані трикутники далі зафарбовуються за методом Гуро. Такий підхід підвищує якість рендерингу Гуро завдяки визначенню того, чи перетинає світло один або декілька ребер трикутника. У класичному методі зафарбовування Гуро таке виявлення відсутнє, і відблиски усуваються, що призводить до суттєвих артефактів.

З метою підвищення продуктивності зафарбовування можна використовувати комбінацію методів як Гуро. Так, наприклад, можна розбити лінію розгортки на відрізки, довжина яких кратна ступеню двійки. В проміжних пікселях кожного відрізка визначаються значення інтенсивності кольору за методом Гуро. Довжина відрізків може адаптивно змінюватися і визначається кривизною поверхні та напрямком вектора $\vec{H}$. Під час обчислення як дзеркальної, так і дифузної інтенсивності кольору для точок на 3D-поверхні необхідно виконувати нормалізацію нормальних векторів. Це пояснюється тим, що косинуси кутів, які використовуються у функції затінення, можна легко знайти за внутрішнім добутком векторів одиничної довжини. Враховуючи, що нормалізація виконується для вектора спостерігача, вектора джерела світла та нормального вектора, актуальною є проблема зменшення обчислювальних зусиль для цієї ресурсномісткої процедури. Тому запропоновано метод адаптивної нормалізації нормального вектора. Нормалізація виконується лише у випадках значної зміни інтенсивності кольору.

2.3 Розробка методів розпаралелення зафарбовування за Гуро

У традиційному підході до реалізації рендерингу використовуються два лінійних інтерполятора для визначення координат точок, що утворюють ребра трикутника. На основі розрахункового значення приросту інтенсивності вздовж усіх ребер трикутника здійснюється кодування інтерполяції. Для визначення

координат внутрішніх точок трикутника застосовується лінійний інтерполятор по рядках.

Інтенсивність внутрішніх точок трикутника визначається за допомогою відповідного кодового інтерполятора, який можна поєднати з інтерполятором, що визначає інтенсивність ребер трикутника. Таким чином, класична реалізація рендерингу вимагає наявності двох лінійних інтерполяторів, одного кодового та одного рядкового (рис. 2.8).

Один з варіантів підходів (рис. 2.9) до паралельного зафарбовування полігону трикутника базується на методі зустрічних кодових інтерполяторів.

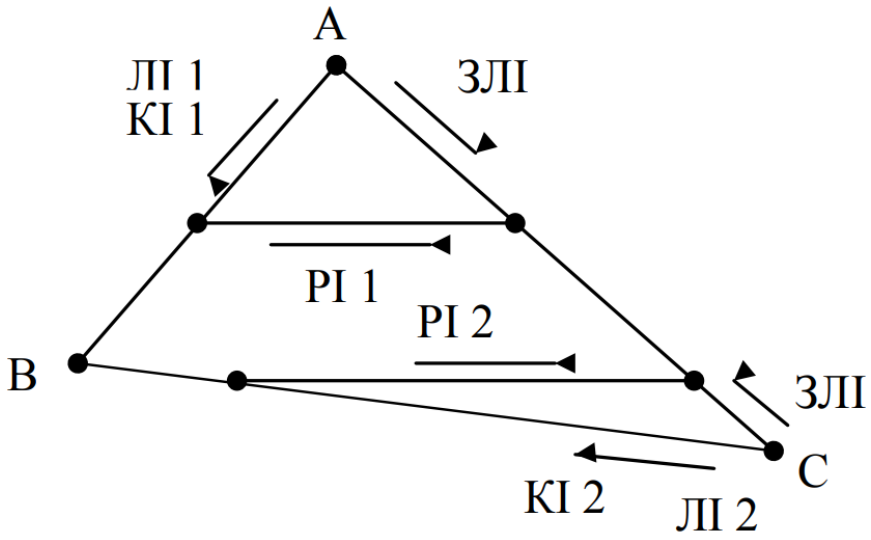


Рисунок 2.8 – Розподіл обчислювального процесу між інтерполяторами

Його суть полягає в тому, що значення інтенсивності в рядку трикутника, який сканується, обчислюються за допомогою двох кодових інтерполяторів, які працюють один назустріч одному, виходячи з початкової і кінцевої точок рядка.

Лінійна інтерполяція сторін трикутника здійснюється подібно до класичного методу рендеринга.

Запропонований принцип спрощує кодову інтерполяцію рядка, який сканується, удвічі та розподіляє похибку рендерингу між двома кодовими інтерполяторами.

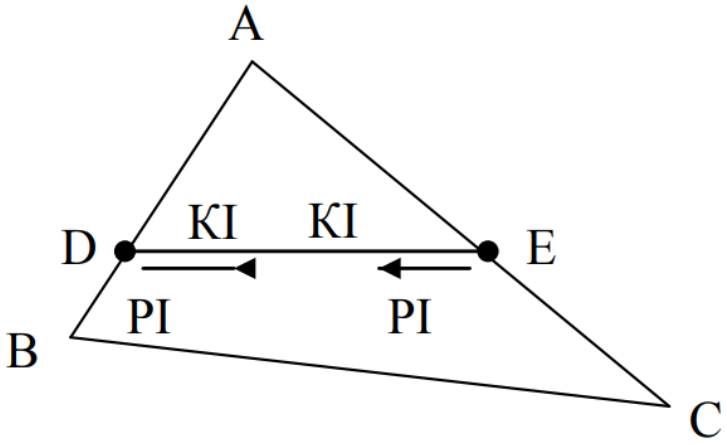


Рисунок 2.9 – Зафарбовування методом зустрічної кодової інтерполяції

Для реалізації цього підходу, на відміну від класичного методу, потрібно використовувати один кодовий інтерполятор і один інтерполятор рядка. Іншим способом прискорення процесу рендерингу є незалежний розрахунок інтенсивностей пікселів для парних і непарних точок рядка, який сканується (див. рис. 2.10).

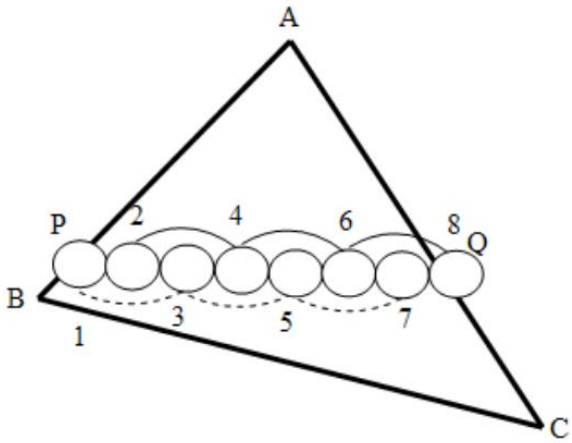


Рисунок 2.10 – Визначення інтенсивностей кольору в парних і непарних точках

Перший кодовий інтерполятор обчислює інтенсивність лише для непарних точок рядка трикутника, подвоюючи значення приросту інтенсивності у рядку сканування ($2\Delta I$). Другий кодовий інтерполятор визначає інтенсивність для парних точок рядка. Цей метод дозволяє подвоїти швидкість кодової інтерполяції сканованого рядка. Для цього підходу також потрібні додаткові

апаратні витрати: один кодовий інтерполятор і один рядковий інтерполятор.

Цей принцип може бути розширений на більшу кількість кодових інтерполяторів, оптимальна кількість яких зазвичай не перевищує чотирьох. Це пов'язано з тим, що розміри $2\Delta I, 3\Delta I, 4\Delta I$ можна обчислити за допомогою однієї операції зсуву та одного додавання. Визначення приросту інтенсивності для більшої кількості точок вимагає додаткових обчислювальних витрат, що, як правило, не є доцільними.

Для збільшення продуктивності рендерингу за методом Гуро пропонується, окрім розпаралелення, використовувати метод зафарбовування трикутника, який базується на аналізі вже існуючої теореми про середню лінію трикутника. У трикутнику ABC (рис. 2.11) проводяться три середні лінії KL, LM, KM. Середня лінія визначається як лінія, що проходить через середини сторін трикутника.

Властивості середньої лінії наступні:

а) вона паралельна одній із сторін трикутника;

б) її довжина дорівнює половині довжини цієї сторони трикутника, до якої вона паралельна.

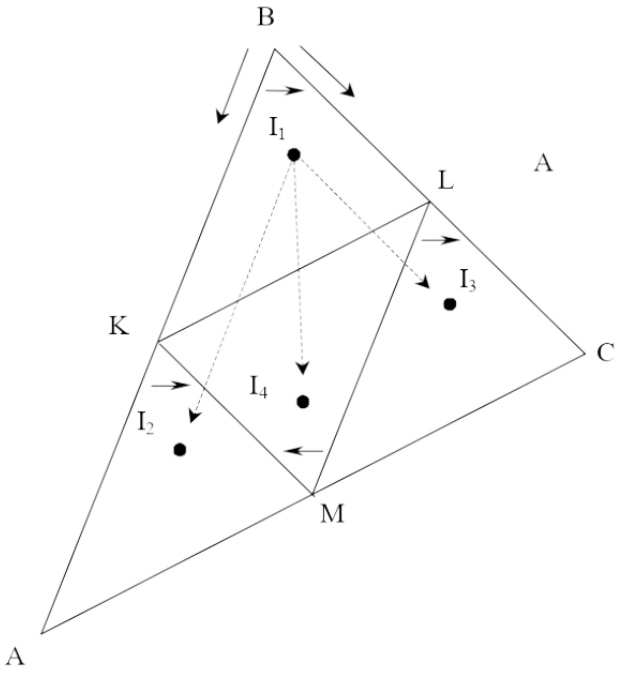


Рисунок 2.11 – Паралельний рендеринг трикутника

Для використання даного методу, необхідною дією є доведення, що отримані трикутники AKM і MLC є рівними. Для доведення рівності цих трикутників, розглянемо їхні сторони:

$$AM = MC = 0.5 * AC \tag{2.13}$$

$$AK = ML = 0.5 * AB \tag{2.14}$$

$$KM = LC = 0.5 * BC \tag{2.15}$$

Так як відповідні сторони розглянутих трикутників є рівними, то це доводить, що вони є рівними. Аналогічно трикутники AKM, MLC, KBL, LMK також є рівними.

Одним з найважливіших етапів, що дозволять підвищити продуктивність формування тривимірних зображень є прискорення продуктивності інтерполяції. Під час інтерполяції трикутника, приріст уздовж його сторони залишається сталим, що дозволяє припустити, що після інтерполяції кожна точка трикутника буде відрізнятися лише на певну константу від відповідної точки іншого трикутника. Тому пропонується зафарбовувати лише один отриманий трикутник трикутник, а інші зафарбовувати методом розмноження результату. Для забезпечення максимальної реалістичності першим будемо зафарбовувати верхній отриманий трикутник. Наступним етапом буде обчислення констант зсуву для кожної з трьох середніх точок (K,L,M):

K:

$$\begin{cases} \Delta X = \dfrac{(X_A - X_B)}{2} \\ \Delta Y = \dfrac{(Y_A - Y_B)}{2} \\ \Delta I = \dfrac{(I_A - I_B)}{2} \end{cases} \tag{2.16}$$

L:

$$\begin{cases} \Delta X = \dfrac{(X_C - X_B)}{2} \\ \Delta Y = \dfrac{(Y_C - Y_B)}{2} \\ \Delta I = \dfrac{(I_C - I_B)}{2} \end{cases} \quad (2.17)$$

M:

$$\begin{cases} \Delta X = \dfrac{(X_A - X_C)}{2} \\ \Delta Y = \dfrac{(Y_A - Y_C)}{2} \\ \Delta I = \dfrac{(I_A - I_C)}{2} \end{cases} \quad (2.18)$$

Використання даного методу сприяє значному прискоренню процесу зафарбування, приблизно в чотири рази, оскільки площа малого трикутника становить лише четверту частину від площі початкового трикутника. Крім того, зафарбування окремих частин трикутника, що залишилися, проводиться паралельно, що уникне зайвих витрат часу.

2.4 Висновки

У другому розділі бакалаврської дипломної роботи було проведено розробку методу зафарбовування тривимірної моделі за методом Гуро, обрано методи оптимізації формування зображень за Гуро та розроблено метод високопродуктивного рендерингу зображень, що буде використовувати поліпшений метод Гуро для зафарбовування поверхні тривимірних моделей та буде використовувати особливості розпаралелення рендерингу.

3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ВИСОКОПРОДУКТИВНОГО ФОРМУВАННЯ ЗОБРАЖЕНЬ

3.1 Розробка моделі та алгоритмів роботи програмного забезпечення

Для кращого розуміння роботи модулів програмних засобів високопродуктивного формування зображень з використанням методу ГУРО було вирішено розробити модель роботи системи на прикладі UML діаграми діяльності (рис. 3.1).

Згідно розробленої діаграми користувач для початку взаємодії із програмним застосунком повинен або імпортувати тривимірну модель у форматі .obj, або вибрати тривимірну модель із доступних базових моделей.

При виборі імпортування моделі, користувач переходить у стан імпортування тривимірної моделі. У іншому випадку виконується стан вибору базової моделі, для виконання операцій над нею.

Наступним станом після стану імпортування тривимірної моделі у робоче середовище розроблених програмних засобів є стан перевірки імпортованої моделі на відповідний тип розширення .obj. При вірному форматі розширення програмний застосунок дозволить доступ до наступних етапів роботи застоунку, у іншому випадку, програмний засіб видасть помилку.

Наступним кроком є аналіз тривимірної моделі, а саме збір інформацію про знайдені вершини та полігони. Після чого початкові стани імпортування і вибору базової моделі мають однакові функції.

Після цього відбувається відображення моделі у робочому середовищі програмного застосунку. Наступним кроком програмне забезпечення очікує вибір функції від користувача. Для користувача доступні наступні функції: показати каркас тривимірної моделі (відобразити контури полігонів), змінити колір зафарбовування, зафарбувати за методом ГУРО, змінити параметри згладжування, змінити параметри прозорості.

При виборі користувачем функції, програмний застосунок автоматично

застосовує розроблену функцію до тривимірної моделі та автоматично застосовує і відображає зміни на тривимірній моделі. Після чого користувач може виконати експорт зображення із тривимірної фігури. При виборі цієї функції програмне забезпечення переходить у стан рендерингу та зберігає створене зображення.

Після чого користувач може відкривати, переглядати та використовувати сформоване зображення.

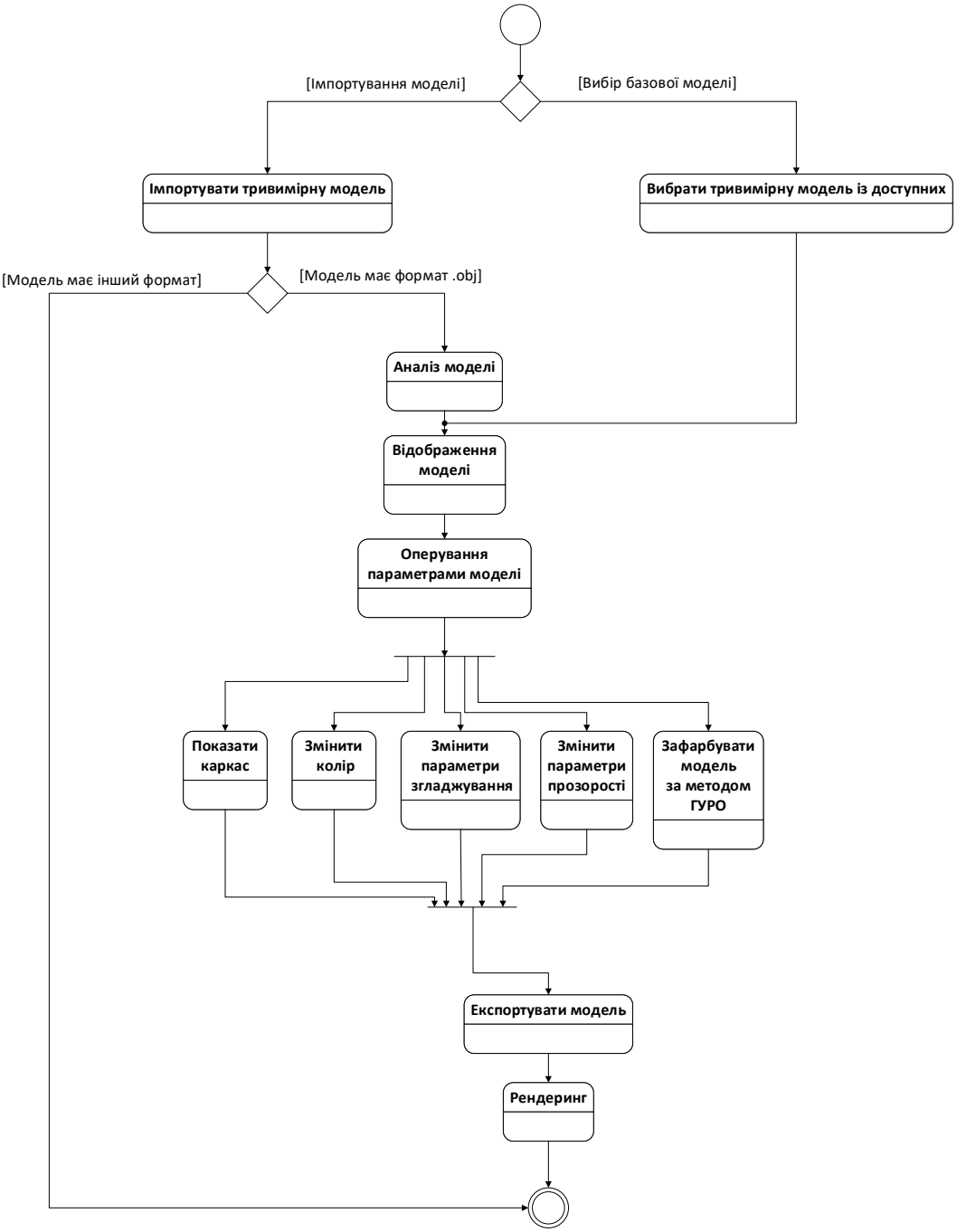


Рисунок 3.1 – Діаграма діяльності програмних засобів

Для розробки програмного застосунку, що реалізує функціонал високопродуктивного формування зображень за методом ГУРО необхідно розробити загальний алгоритм, згідно якого буде працювати програмний модуль (рис. 3.2). Згідно даного алгоритму, першим ділом, відбувається завантаження робочого середовища програми, яке надає користувачу доступ до функціоналу розробленого програмного застосунку.

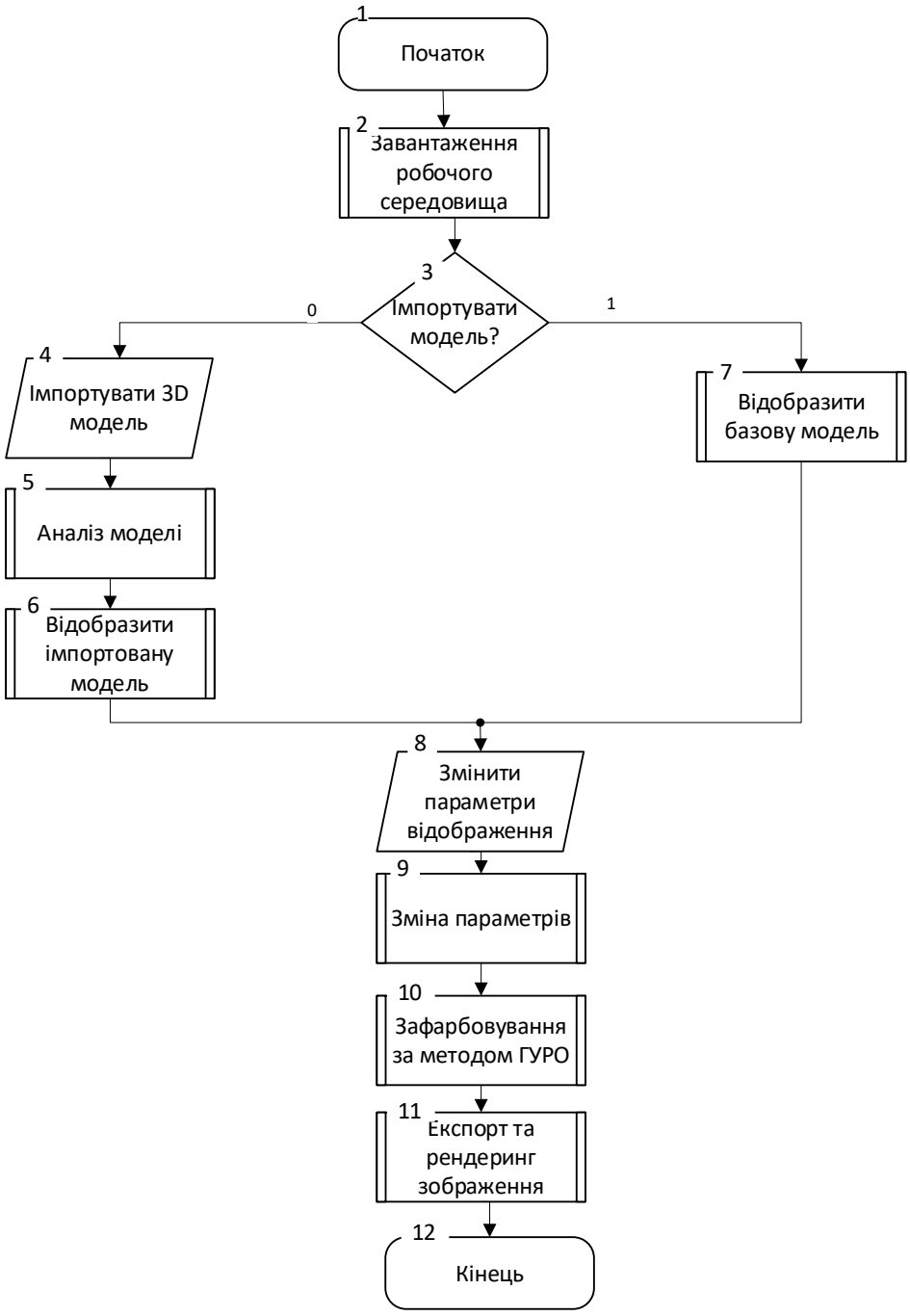


Рисунок 3.2 – Алгоритм роботи програмного застосунку

Наступним кроком користувач може імпортувати тривимірну модель, або ж обрати доступні базові тривимірні моделі з програмного застосунку. При виборі базових моделей програмний застосунок одразу відобразить тривимірну модель у робочу середовищі програми та буде очікувати на дії користувача, а саме вибір функцій.

При виборі імпортування власної тривимірної моделі, запускається модуль імпортування 3д моделі у якому користувач обирає тривимірну модель, що розміщена у файловій системі. Наступним кроком програмне забезпечення проводить аналіз тривимірної моделі та виконує тріангуляцію. Після чого відсортовує кожну вершину полігонів та формує множину точок. Після чого виконується відображення імпортованої моделі і програмний застосунок очікує на вибір функціоналу від користувача.

Наступним кроком користувача може обрати одну з реалізованих функцій, тим самим змінивши параметри для відображення тривимірної моделі. Також користувач може змінити колір зафарбовування та провести зафарбовування тривимірної фігури за методом ГУРО. Після зафарбовування фігури та застосування параметрів, користувач може експортувати сформоване зображення. Для цього програмне забезпечення запускає процес рендерингу та зберігає зображення у відповідну директорію.

Для кращого розуміння роботи модулів зафарбовування тривимірної моделі за методом ГУРО, було розроблено додатковий алгоритм (рис. 3.3). Згідного даного алгоритму першим етапом є завантаження даних обраної тривимірної моделі. Після чого виконується автоматична тріангуляція моделі з подальшим розбиттям отриманих трикутників на множину тривимірних вершин. Далі виконується сортування отриманої множини, шо виконується задля точного зафарбовування тривимірної моделі відповідно до позиції точки освітлення. Після чого виконується цикл, який проходить по усім вершинам у масиві та виконує процеси зафарбовування.

Першим кроком циклу є додавання позиці точки освітлення та виконання обчислення відстані і впливу джерела світла на вершину трикутника. Після чого

виконується обрахування векторів нормалей для відповідної вершини, що дозволить рівномірно зафарбувати вершину.

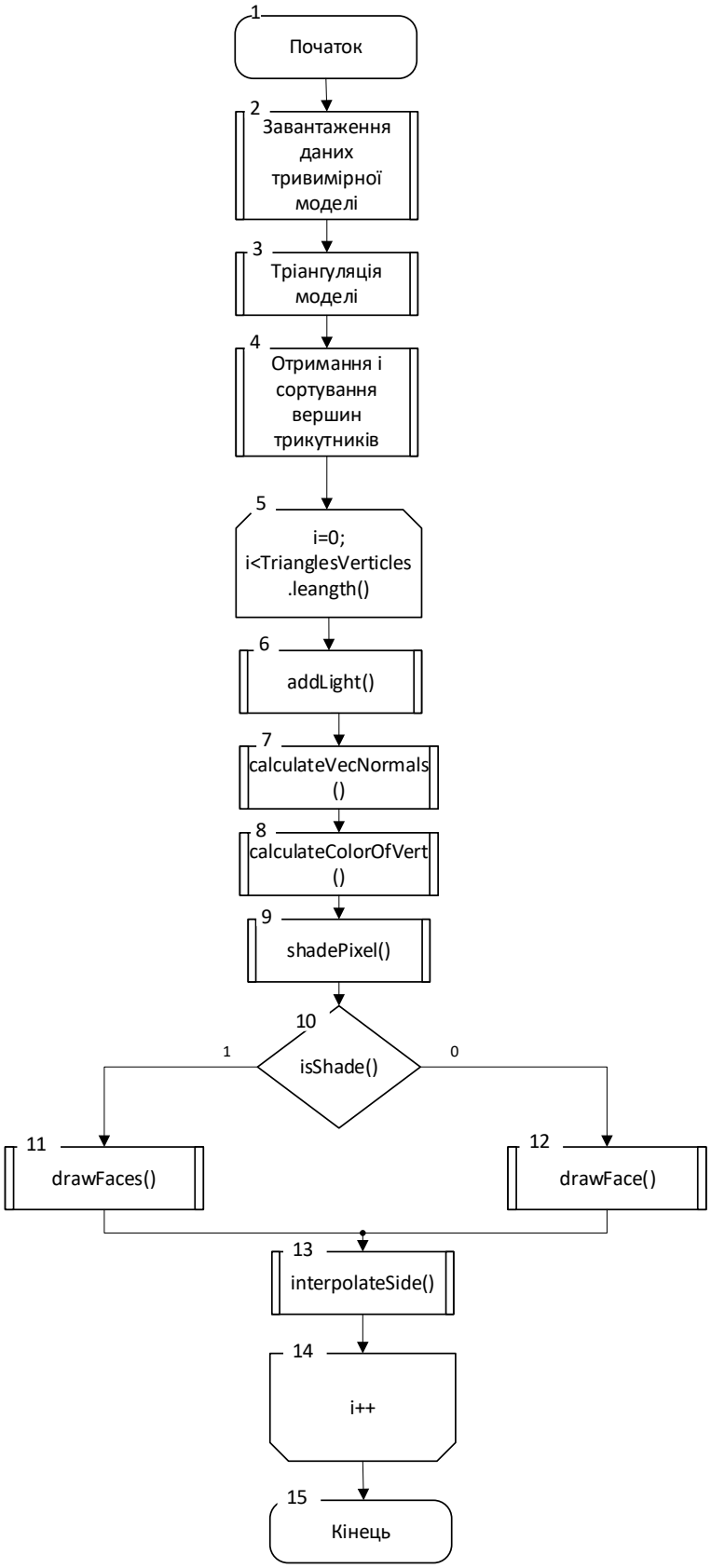


Рисунок 3.3 – Алгоритм зафарбовування тривимірної моделі

Далі виконується обрахування кольору вершини відповідно до джерела світла та отриманого вектору нормалі. Після обрахування складової кольору виконується модуль зафарбовування окремого пікселю вершини. Далі відбувається перевірка на доступність джерела світла до вектора нормалі, при умові що світло відображається на поверхні вектора нормалі, відбуваєть зафарбовування площини окремого трикутника з відображенням свіла. Розроблені діаграми створювались у ПЗ Visio [17].

3.2 Розробка модуля зафарбовування тривимірної моделі за методом ГУРО

При розробці модуля для зафарбовування тривимірної моделі за методом ГУРО важливо враховувати ряд технічних аспектів. В першу чергу, необхідно аналізувати особливості самої тривимірної моделі та виконати процес тріангуляції для визначення вершин, які будуть використовуватися під час зафарбовування. Для цього процесу реалізований клас mainDraw, який відповідає за визначення позицій усіх вершин, розташування джерела світла та перелік параметрів, необхідних для роботи з тривимірною моделлю, а також зафарбовування та відображення моделі.

Цей клас, показаний на рисунку 3.4, виконує ключові функції для ефективної роботи з тривимірною моделлю. Він ініціалізує необхідні параметри, здійснює обробку позицій вершин, враховує джерело світла та інші важливі параметри, що впливають на відображення та зафарбовування моделі. Такий підхід дозволяє ефективно управляти процесом зафарбовування та відображення тривимірної моделі з врахуванням всіх необхідних параметрів.

Після визначення необхідних параметрів було вирішено, задля досягнення процесу оптимізації, використати у якості площин для зафарбування не трикутники а чотирикутні полігони довільної форми. Для обрахування позиції вершин використовується відсортована множина усіх вершин тривимірної моделі.

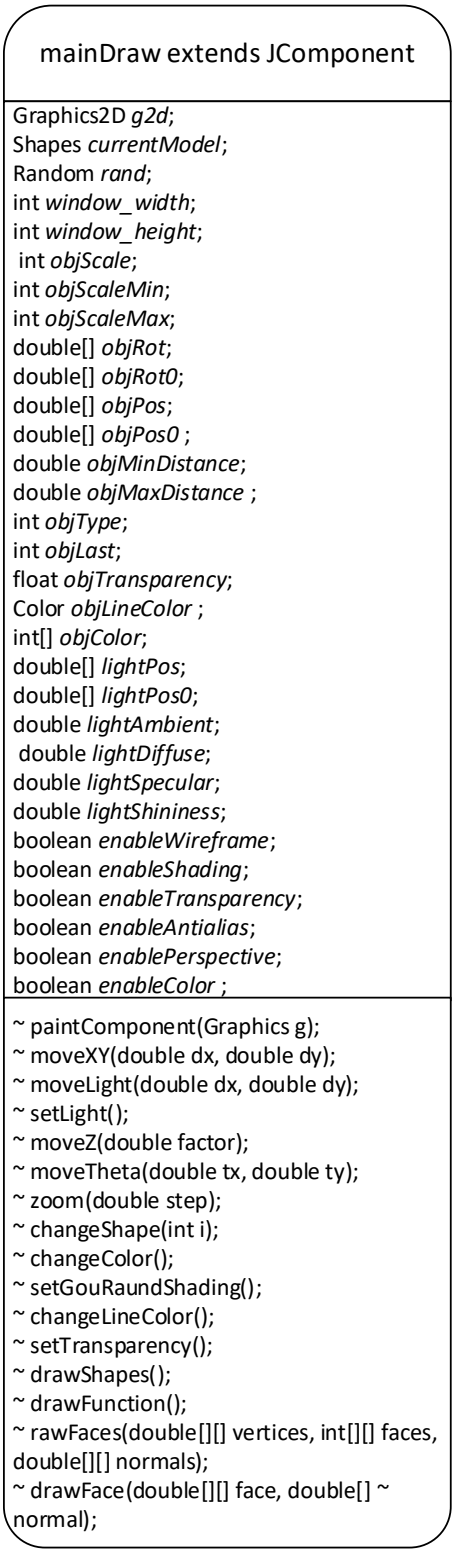


Рисунок 3.4 – Клас mainDraw

Після формування окремих множин для кожної координати, необхідною дією є визначення векторів нормалей, що реалізовано у методі drawFunction(рис. 3.5). Даний метод обраховує позиції чотирьох вершин полігона та вектор нормалі. Після чого викликає функцію зафарбовування полігона.

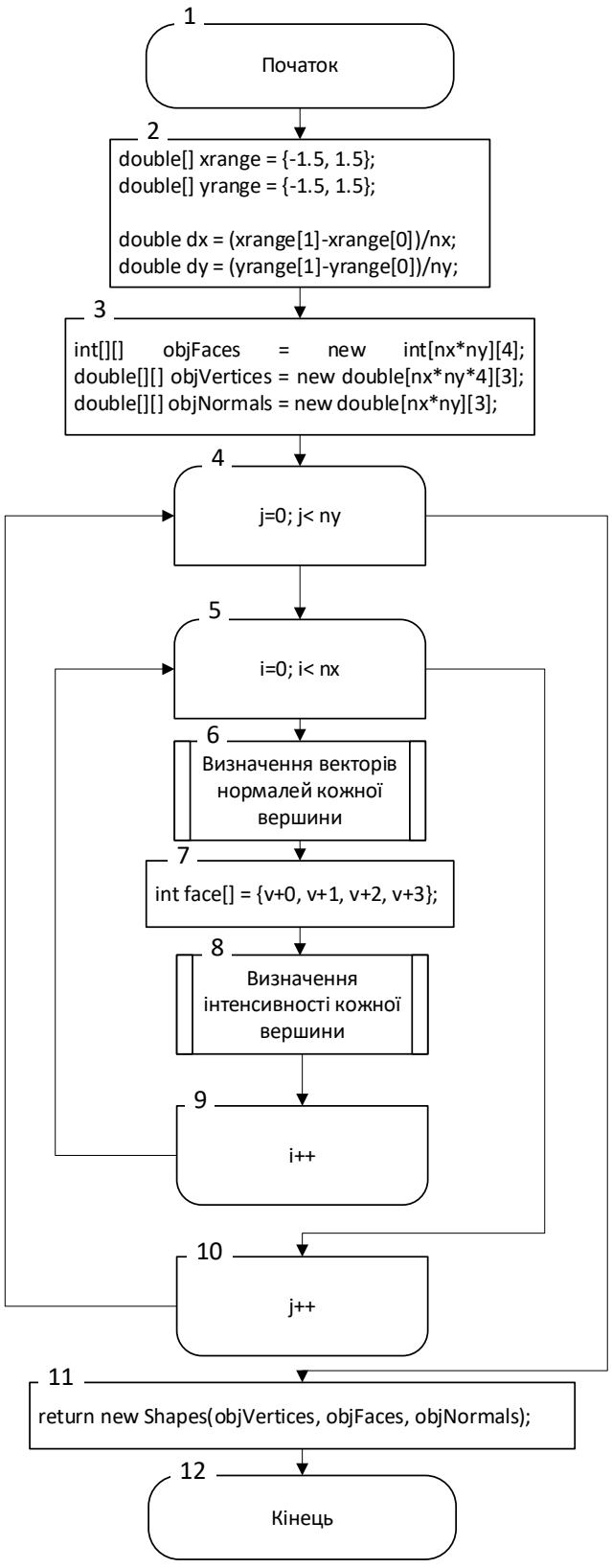


Рисунок 3.5 – Метод формування полігонів

Наступним кроком є відображення отриманих полігонів на тривимірній площині. Це необхідно для відображення та візуалізації процесу зафарбовування

та обрахування рівня затемнення та освітлення полігона. Для цього створюються точки на кутах полігонів та виконується записування даних про кожну вершину у відповідні масиви, а саме дані про позицію точки, вектори нормалі та множина позицій пікселів, що розміщені в полігоні. Далі виконується визначення точки освітлення та перевірка його впливу на пікселі полігона (рис. 3.6).

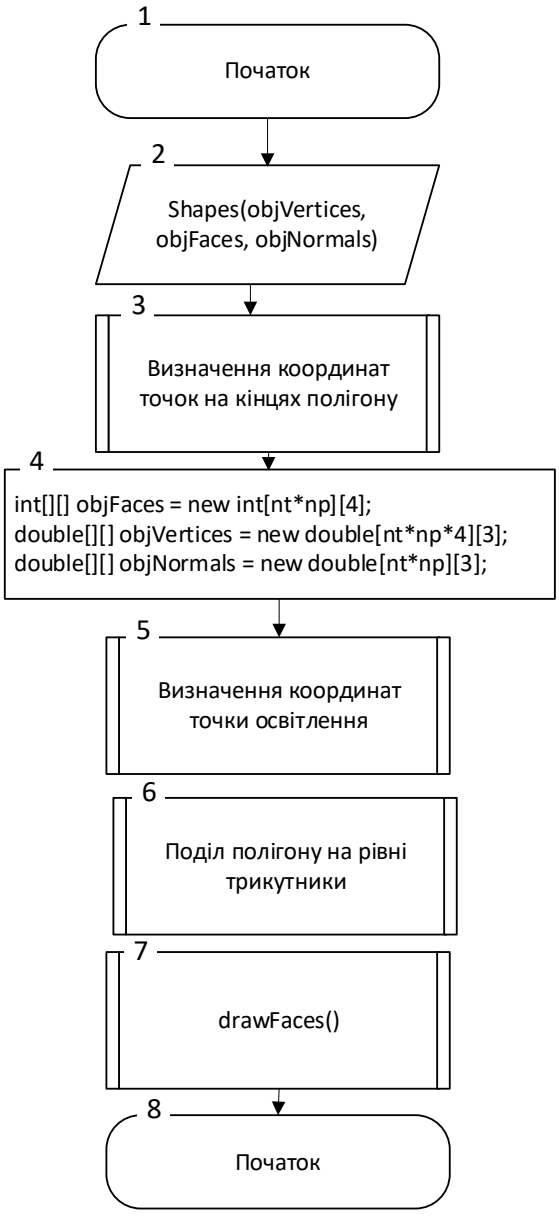


Рисунок 3.6 – Функція drawSphere

Після чого, при відсутності впливу освітлення викликається метод drawFaces (рис. 3.7), який виконує функцію зафарбовування площини відповідним кольором. Для цього він використовує позиції пікселів полігона та

вектори нормалі, для відображення світла та задання реалістичності його поширення на площині.

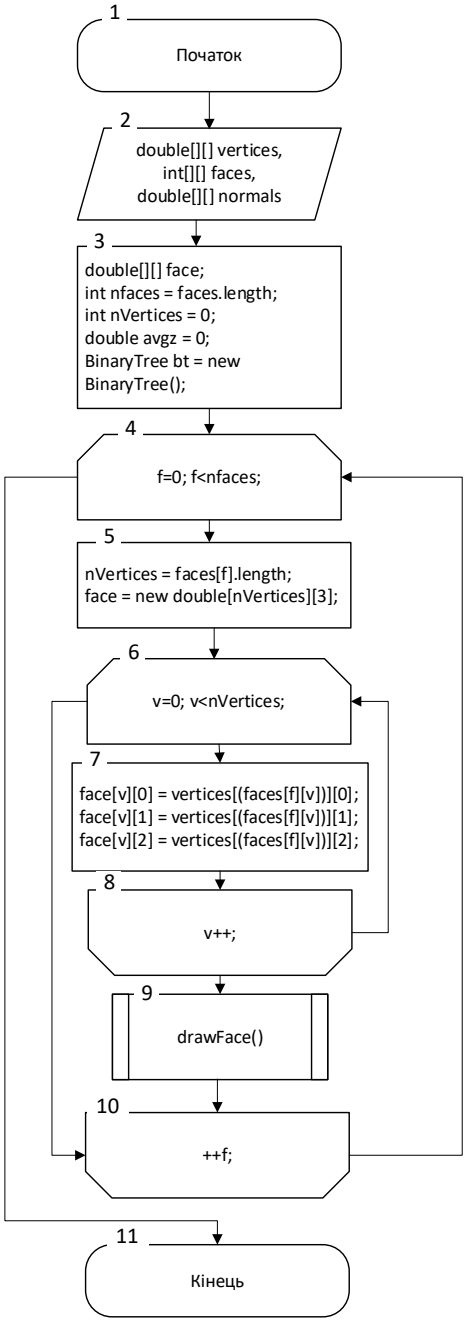


Рисунок 3.7 – Метод drawFaces

При наявності світла на площині полігона викликається метод drawFace (ри. 3.8), що виконує зафарбовування площини полігона враховуючи при цьому позицію світла, дифузну та спекулярну складову освітлення а також показник яскравості освітлення. Після чого у методі відбуваєть обрахування впливу

освітдення на кожний окремий піксель у площині та викликається метод shade (рис. 3.9). У даному методі виконується обрахування кольору кожного пікселю на основі позиції точки освітлення, складової кольору та яскравості точки освітлення. Для цього виконується визначення векторів нормалей та процесів інтерполяції для виконання поступового зафарбовування зі зменшенням або збільшенням впливу світла на поверхню полігона.

Рисунок 3.8 – Метод drawFace

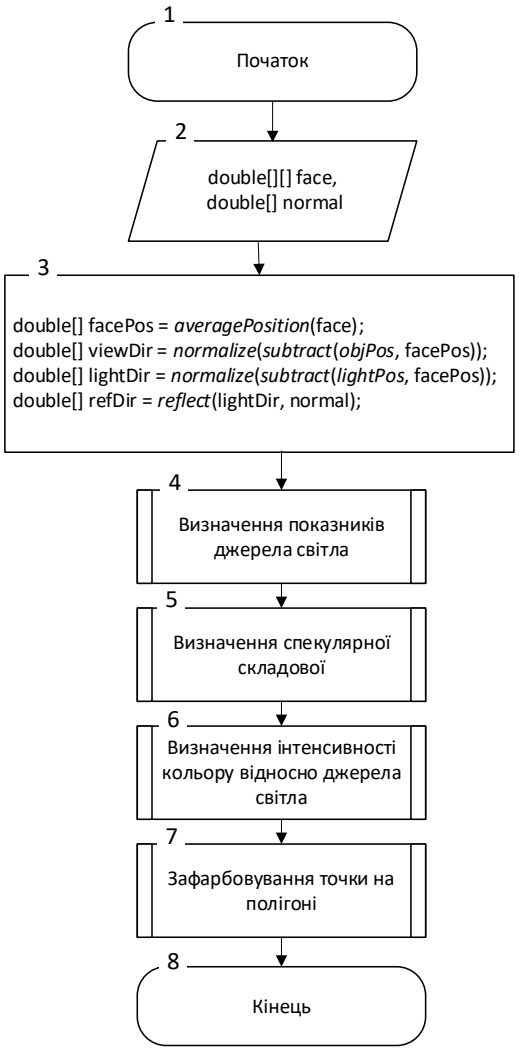


Рисунок 3.9 – Метод shade

3.3 Розробка модуля високопродуктивного формування зображень

Головною задачею модуля високопродуктивного формування зображень є точний та швидкий рендеринг зображення. Для реалізації цієї задачі було створено клас SimpleInterpreter. При створені і запуску класу автоматично виконується визначення позицій світла та зафарбовування тривимірної моделі (рис. 3.10).

Крім того було розроблено окрему функцію для рендерингу пікселів полігонів за методом ГУРО (рис. 3.11). Дана функція дозволяє реалістично передати зафарбовування за методом ГУРО, базуючись на позиції точки освітлення.

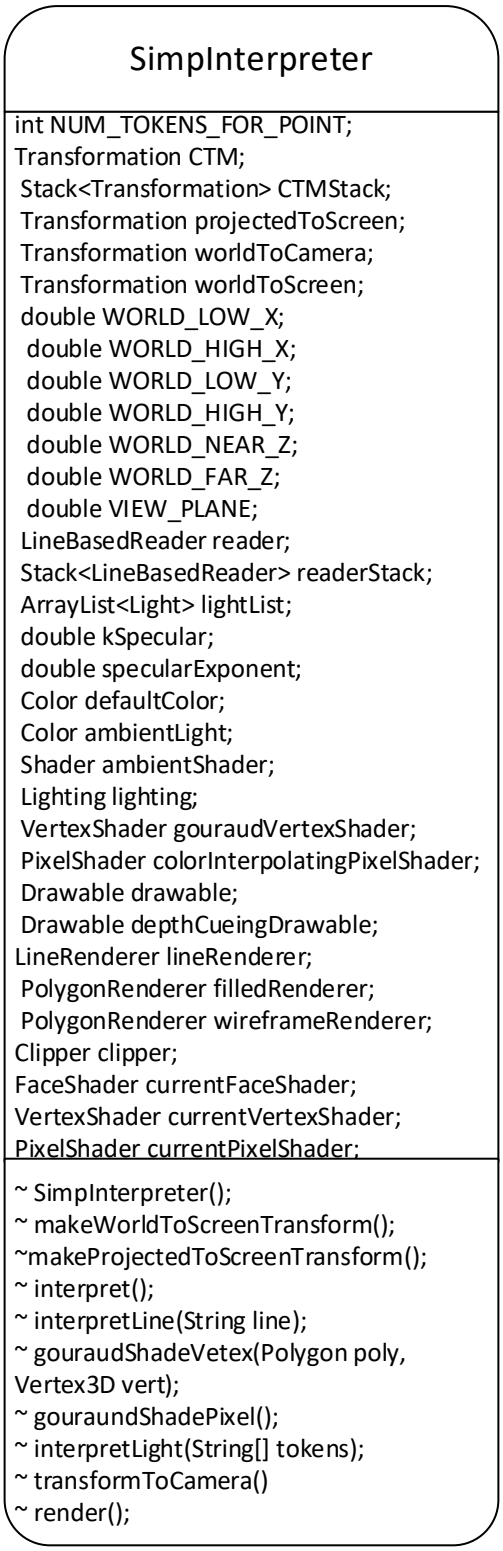


Рисунок 3.10 – Клас SimpInterpreter

Наступним кроком є інтерполяція за вектором нормалі відбиття світла. Для цього використовується метод класу interpretLight, який обчислює відбиття світла відповідно до положення камери.

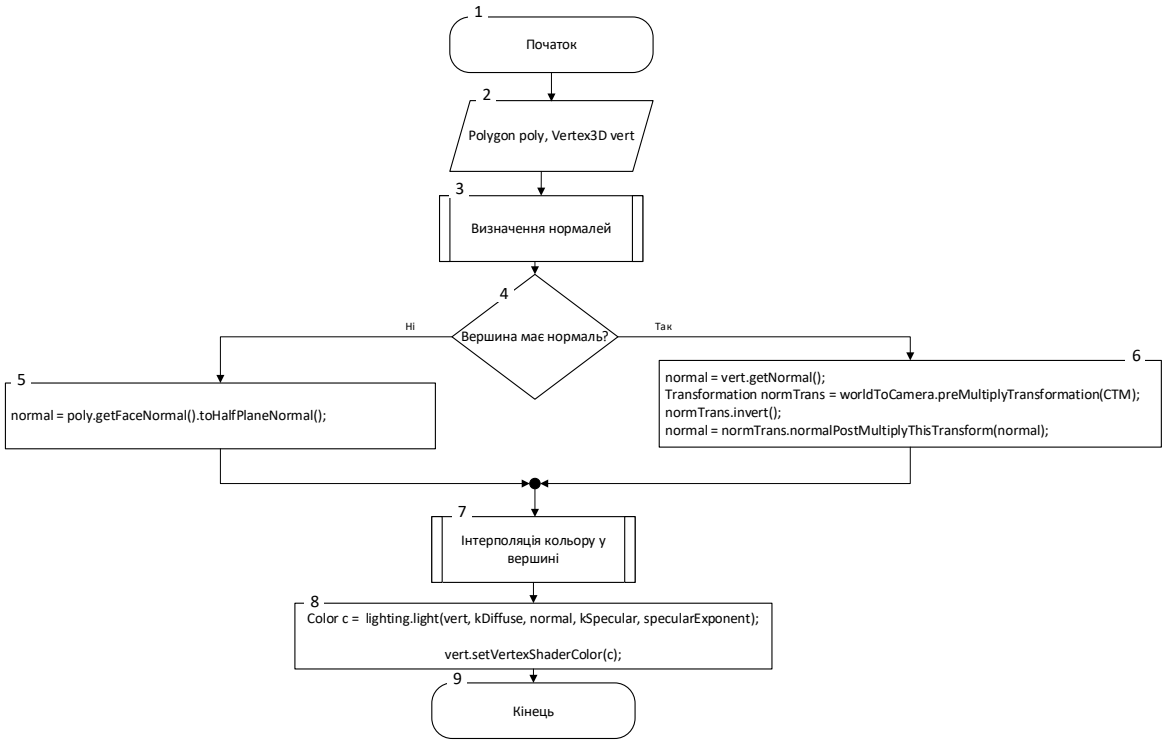


Рисунок 3.11 – Функція рендерингу пікселів за ГУРО

Це дозволяє врахувати позицію спостерігача та отримати реалістичні ефекти відбиття світла.

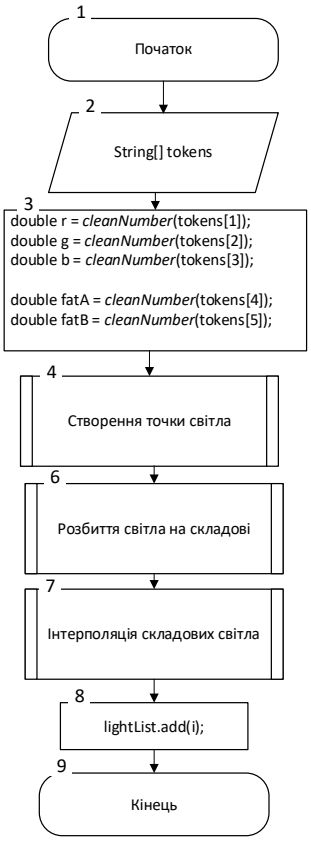


Рисунок 3.12 – Метод обрахування відбиття світла

Далі виконується рендеринг кожного полігону тривимірної моделі (рис. 3.13). Для цього застосовується інтерполяція камери, обрахування впливу точки освітлення та застосування функції зафарбовування до кожного пікселю 3д моделі.

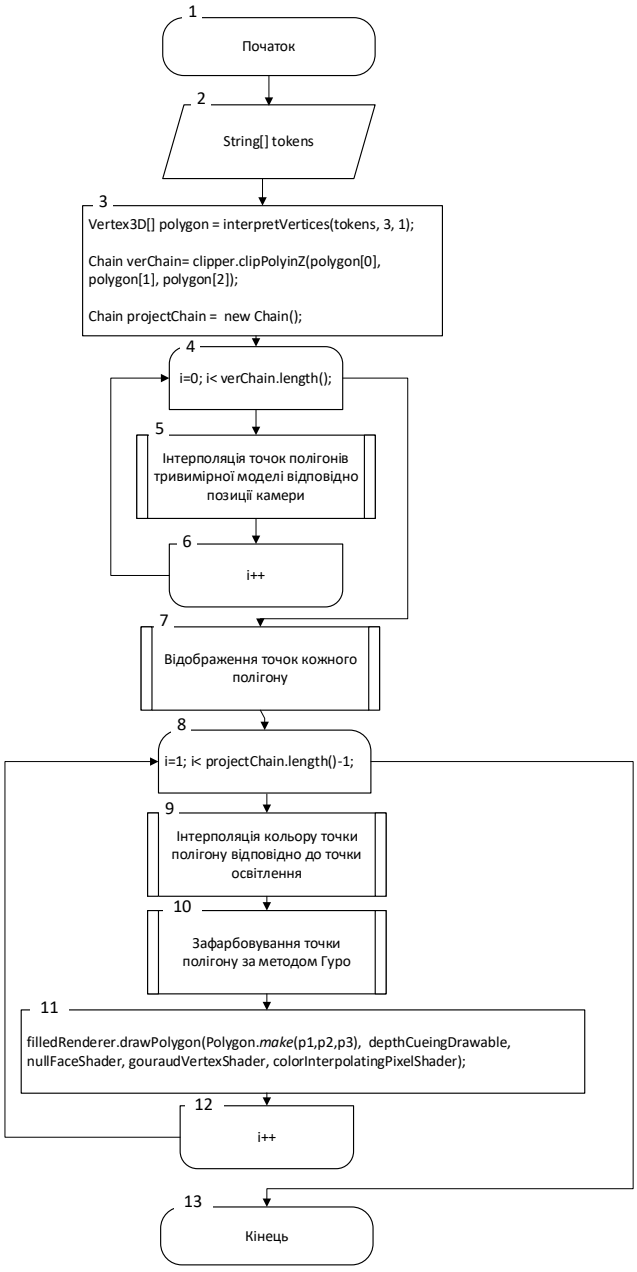


Рисунок 3.13 – Рендеринг полігонів тривимірної моделі

3.4 Висновки

У третьому розділі було розроблено алгоритми роботи програмного забезпечення та модель, що демонструє можливі стани розробленого

програмного додатку. У розділі також було описано розробку основних програмних модулів: розробку модулів для зафарбовування тривимірної моделі за методом ГУРО та високопродуктивного формування зображень.

4 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Тестування програмного забезпечення

Тестування програмного забезпечення є важливим етапом у життєвому циклі розробки програмних систем, оскільки воно дозволяє виявити та усунути помилки, забезпечуючи відповідність програми встановленим вимогам і специфікаціям [18]. Основною метою тестування є перевірка функціональних і нефункціональних характеристик програми, включаючи її надійність, ефективність, зручність у використанні та безпеку. Тестування також передбачає верифікацію та валідацію програмного забезпечення для підтвердження його правильності та відповідності потребам користувачів.

Верифікація програмного забезпечення включає перевірку відповідності розробленого коду встановленим специфікаціям та вимогам. Це забезпечується шляхом оцінки результатів роботи програми відносно попередньо визначених критеріїв. Верифікація охоплює як статичні методи, такі як аналіз коду та рев'ю, так і динамічні методи, включаючи виконання тестів для перевірки правильності окремих компонентів системи.

Валідація, з іншого боку, спрямована на перевірку того, чи задовольняє система потреби кінцевих користувачів і чи відповідає вона реальним умовам експлуатації. Це включає тестування функціональних можливостей програми, оцінку її зручності в користуванні та відповідності вимогам замовника. Валідація гарантує, що розроблена система є коректною та придатною для використання [19].

Існує кілька методів тестування програмного забезпечення, які відрізняються рівнем знань про внутрішню структуру програми [20]:

1. Тестування методом "чорної скриньки": Цей підхід базується на аналізі функціональних або нефункціональних специфікацій системи без знання її внутрішньої будови. Основна мета такого тестування полягає у виявленні помилок у реалізації функцій, структури даних та інтерфейсу користувача.

Тестувальник перевіряє програму з точки зору кінцевого користувача, зосереджуючись на зовнішніх проявах функціональності.

2. Тестування методом "білої скриньки": У цьому підході тестувальник має повне знання внутрішньої структури та реалізації системи. Тестування проводиться на основі аналізу коду, дозволяючи ретельно перевірити логіку програми та її шляхів виконання. Цей метод забезпечує високе покриття тестами та допомагає виявляти логічні помилки та уразливості.

3. Тестування методом "сірої скриньки": Цей комбінований підхід передбачає часткове знання тестувальником внутрішньої структури програми. Тестування проводиться з позиції користувача, але з використанням знань про внутрішню реалізацію для створення більш ефективних тест-кейсів. Метод "сірої скриньки" поєднує переваги підходів "чорної" та "білої скриньок", забезпечуючи збалансоване та гнучке тестування.

Для тестування програмного забезпечення було обрано метод білої скриньки. Для тестування було встановлено програмне забезпечення на операційну систему Windows 10.

На першому етапі тестування перевіряється реакція програми на помилкові данні. На рисунку 4.1 представлено реакція розробленого програмного забезпечення на імпортування тривимірної моделі невірного формату.

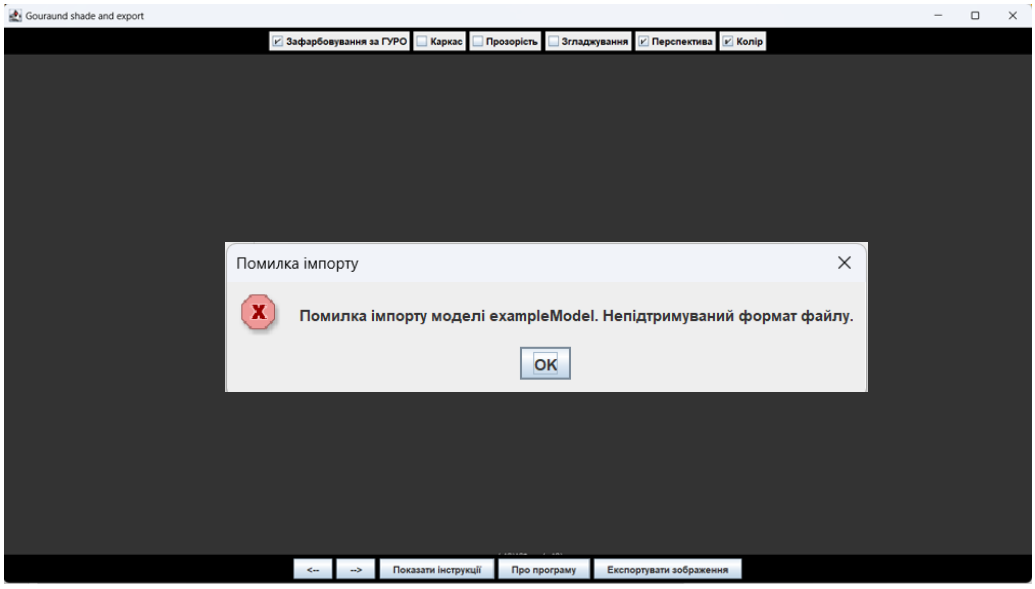


Рисунок 4.1 – Імпортування 3Д моделі іншого формату

При спробі імпортування файлу іншого формату система видає відповідне повідомлення про помилку, яке інформує користувача про некоректний формат файлу та необхідність імпортувати модель у підтримуваному форматі. Це забезпечує захист системи від можливих збоїв і допомагає користувачу зрозуміти, які дії необхідно виконати для успішного імпортування.

Після проведення тестування програмного забезпечення на можливі випадки відмови було прийнято рішення провести тестування імпортування тривимірної моделі правильного формату. На рисунку 4.2 відображено робоче середовище програмного забезпечення, у якому успішно імпортовано тривимірну модель. У цьому випадку програма коректно обробляє вхідні дані, завантажуючи та відображаючи тривимірну модель без помилок, що свідчить про правильну роботу функціоналу імпорту.

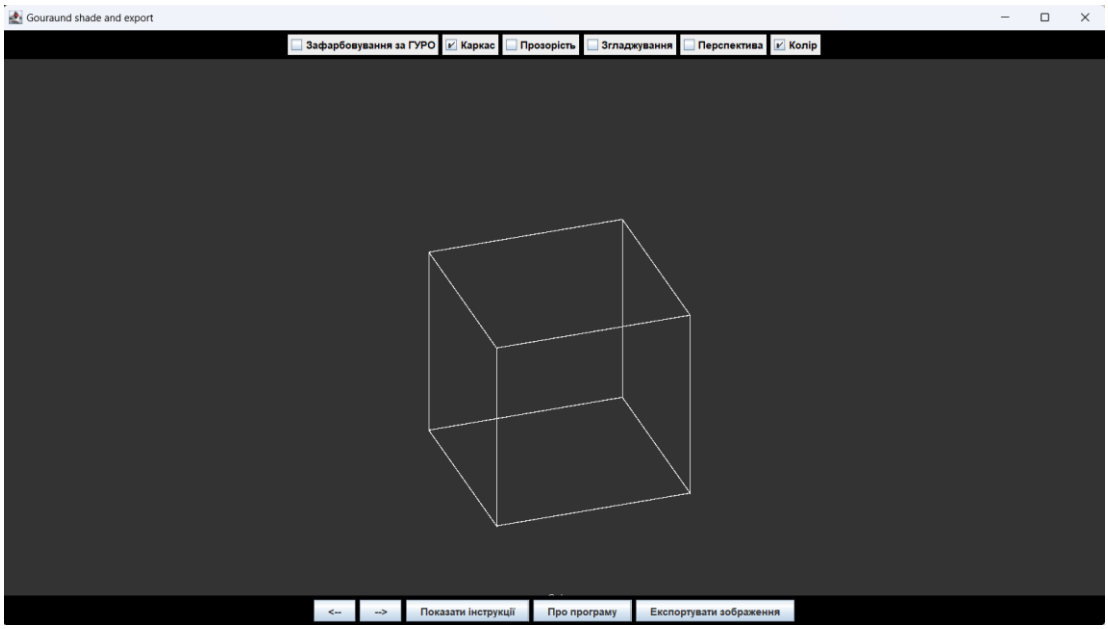


Рисунок 4.2 – Робоче середовище програмного забезпечення

Наступним етапом тестування є перевірка виконання розроблених функцій програмного забезпечення, а саме функціональне тестування розробленого програмного забезпечення. Це тестування включає в себе перевірку коректності виконання всіх основних функцій програми, таких як маніпуляція тривимірною моделлю, зміна масштабу, обертання, а також використання різних інструментів

і опцій, доступних у функціональній стрічці.

Після успішного імпортування тривимірної моделі користувач може використовувати функціонал програми через функціональну стрічку, а також змінювати відображення тривимірної моделі шляхом зміни масштабу та кута огляду. На рисунку 4.3 відображено можливість взаємодії з тривимірною моделлю, включаючи обертання, масштабування та інші маніпуляції, які дозволяють користувачу детально розглянути модель з різних ракурсів. Це забезпечує зручність у використанні програмного забезпечення та дає змогу користувачу ефективно виконувати необхідні завдання.

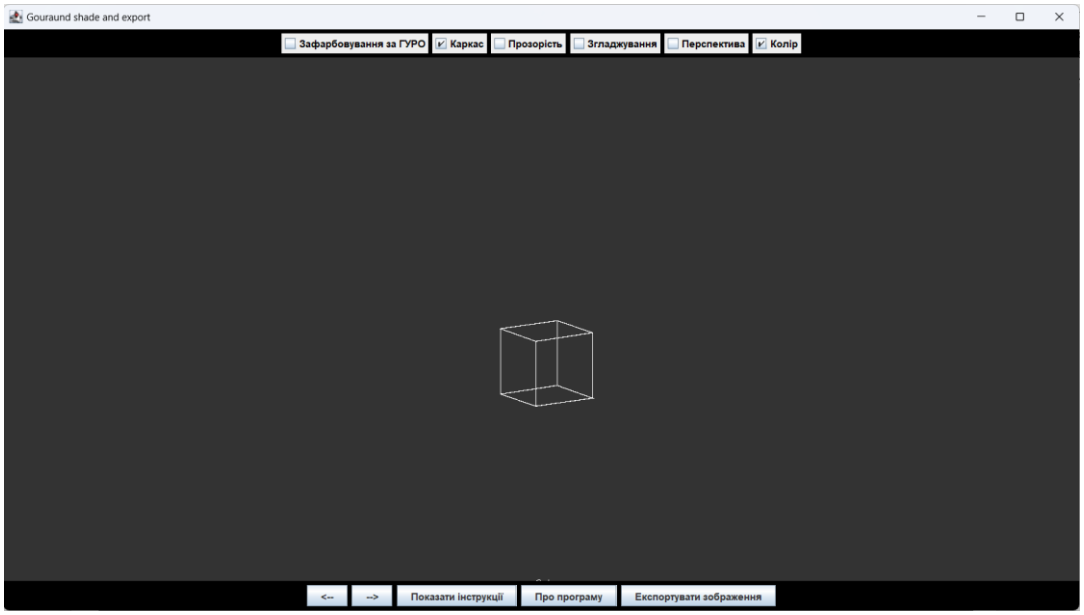


Рисунок 4.3 – Результат імпортування 3Д моделі з дефектами

Наступним етапом є перевірка правильності виконання розроблених функцій. На цьому етапі основна увага приділяється функціональному тестуванню, яке має на меті переконатися, що всі заявлені функції програмного забезпечення працюють коректно та відповідають вимогам специфікації. Однією з ключових функцій програмного забезпечення є зафарбовування тривимірних фігур за методом Гуро, що використовується для плавного градієнтного заповнення полігонів, що дозволяє отримувати більш реалістичні зображення. На рисунку 4.4 відображено результати тестування цієї функції. Тестування

показало, що програмне забезпечення здатне швидко та ефективно зафарбовувати тривимірні фігури, забезпечуючи високу якість візуалізації без видимих артефактів.

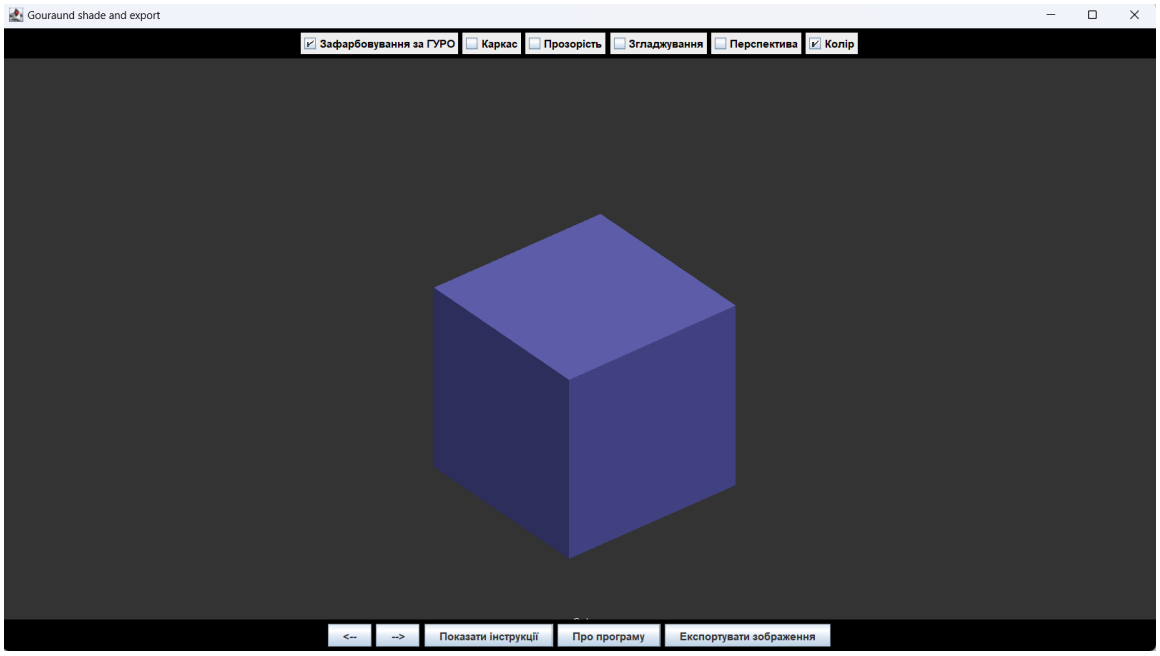


Рисунок 4.4 – Тестування зафарбовування за методом ГУРО

Також було проведено тестування зміни параметрів прозорості, що дозволяє користувачу бачити інші грані тривимірної фігури при зафарбовуванні. Це корисно для більш точного аналізу структури моделі. Крім того, було протестовано можливість покращення відображення моделі шляхом застосування згладжування, яке усуває нерівності та робить зображення більш приємним для ока. На рисунку 4.5 наведено результати цього тестування, які підтверджують коректну роботу цих функцій.

У рамках тестування програмного забезпечення також було проведено аналіз його сумісності з різними версіями операційних систем та мінімальною конфігурацією апаратного забезпечення. Зокрема, програмний продукт був перевірений на сумісність з операційними системами Windows 10, Windows 8.1 та MacOS. Тестування проводилося як на високопродуктивних комп'ютерах, так і на пристроях з обмеженими технічними можливостями. Результати тестування показали, що програмне забезпечення успішно працює на всіх перевірених

конфігураціях, що вказує на його високу оптимізацію та ефективність.

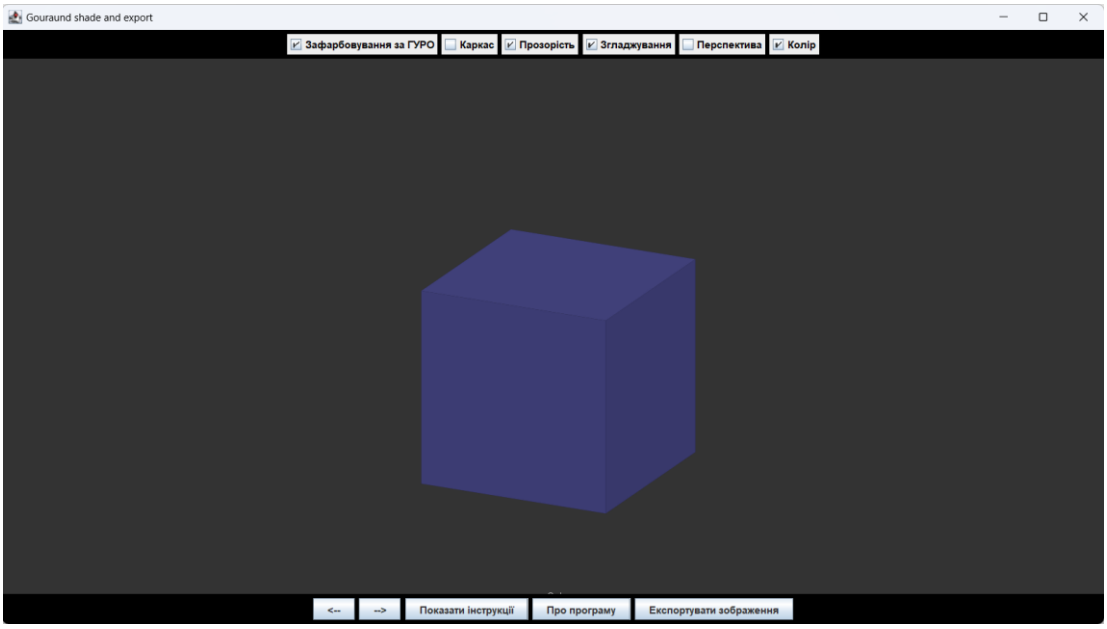


Рисунок 4.5 – Результат тестування зміни параметрів відображення

Також проведено тестування зміни кольору зафарбовування. Для цього користувач має натиснути кнопку «Колір», після чого колір зафарбовування змінюється на інший. На рисунку 4.6 відображено результат цього тестування, який підтверджує коректність роботи функції зміни кольору.

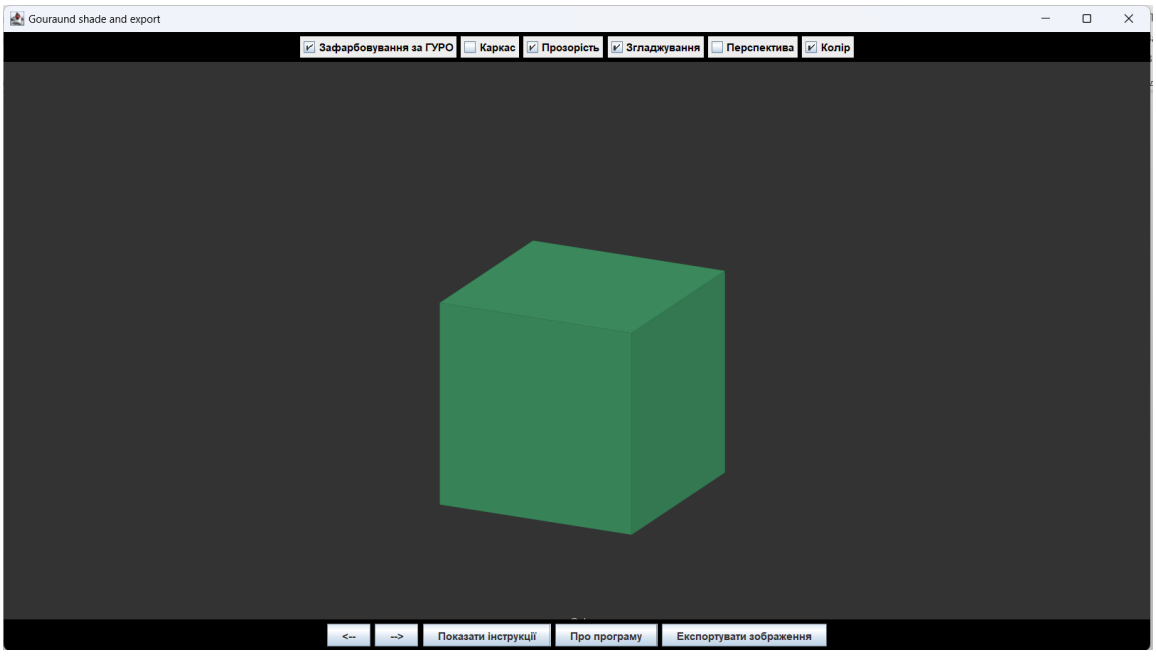


Рисунок 4.6 – Результат тестування зміни кольору

Для поліпшення відображення тривимірної моделі можна застосувати параметр перспективи. Ця функція додає до тривимірної моделі перспективу, що значно підвищує реалістичність її відображення. На рисунку 4.7 представлено результат тестування функції перспективи, яка була успішно реалізована та продемонструвала свою ефективність.

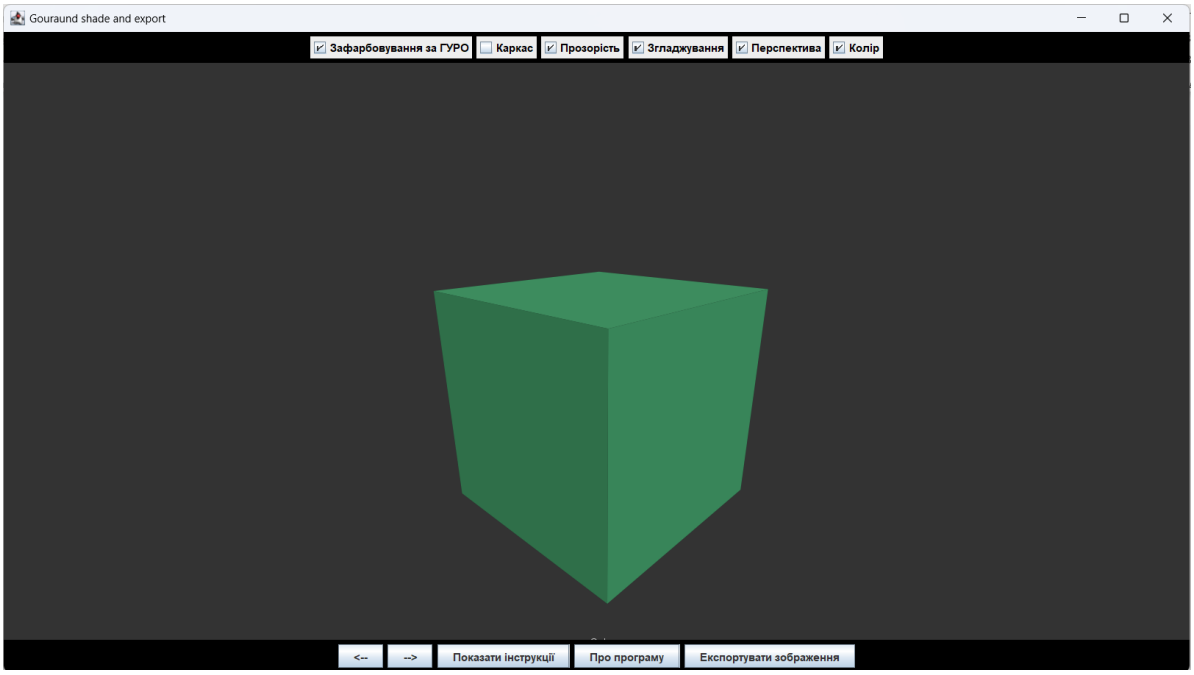


Рисунок 4.7 – Результат тестування перспективи

Програмне забезпечення також дозволяє використовувати базові тривимірні моделі. Для цього у нижньому меню користувач може перейти до першої базової тривимірної моделі, натиснувши стрілочку «Назад». На рисунку 4.8 наведено результат тестування цієї функції, яка показала коректне завантаження та відображення базової тривимірної моделі.

Додатково до тестування функціональності програмного забезпечення було проведено аналіз часу, необхідного для зафарбовування тривимірної моделі за методо ГУРО та її візуалізації у тривимірному просторі. Під час тестування виявлено, що система здатна зафарбовувати тривимірну модель за методом ГУРО за часовий проміжок 0.1-0.5 секунди, що є значно швидше ніж до оптимізації цього методу. Це означає, що користувачі отримують миттєвий

зворотний зв'язок під час використання функції зафарбовування.

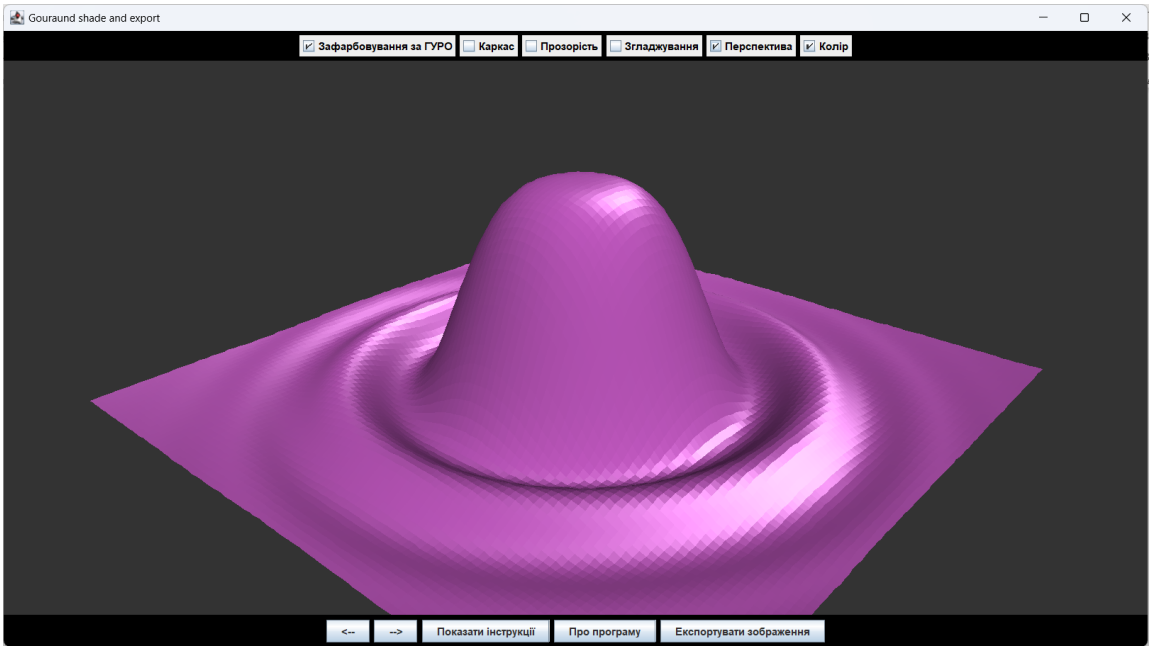


Рисунок 4.8 – Тестування базової тривимірної моделі

4.2 Розробка інструкції користувача

Розробка програмного забезпечення для високопродуктивного формування зображень з використанням методу Гуро передбачає створення інструкції користувача, що є ключовим елементом для розуміння та ефективного використання програмного продукту. Ця інструкція має на меті надати користувачам чітку інформацію про функціонал програми, рекомендації щодо оптимальних налаштувань та кроки для досягнення бажаних результатів.

Програмне забезпечення має інтуїтивний та зрозумілий інтерфейс користувача, який включає в себе наступні основні елементи:

- меню: меню програми містить основні команди для відкриття, збереження та експорту зображень, а також довідкову інформацію;
- панель інструментів: на панелі інструментів розташовані основні інструменти для роботи з тривимірними об'єктами, змінювання параметрів зафарбовування та освітлення;

- вікно перегляду: у вікні перегляду відображається тривимірна сцена, а також тривимірна модель.

Інструкція користувача передбачає визначення технічних вимог для запуску програмного продукту. Деталі щодо мінімальної конфігурації розміщено в таблиці 4.1. Деталі щодо рекомендованої конфігурації розміщено в таблиці 4.2.

Таблиця 4.1 – Мінімальна конфігурація:

Процесор	Intel Core i3
Оперативна пам'ять	2 ГБ RAM
Вільний простір на диску	200 МБ
Відеокарта	NVIDIA XEGraphics 720.1
Операційна система	Windows 8.1

Таблиця 4.2 – Рекомендована конфігурація:

Процесор	Intel Core i7
Оперативна пам'ять	4 ГБ RAM
Вільний простір на диску	500 МБ
Відеокарта	NVIDIA GeForce 1050
Операційна система	Windows 11

У таблицях 4.1 і 4.2 наведені відомості про мінімальні та рекомендовані системні вимоги, такі як тип операційної системи, кількість оперативної пам'яті, обсяг вільного простору на диску, тип і швидкість процесора, роздільна здатність екрану та інші важливі параметри. Ця інформація допомагає користувачам оцінити сумісність свого обладнання та визначити, чи вони можуть ефективно використовувати програму.

Перед початком використання програмного забезпечення рекомендується уважно ознайомитися з інструкцією користувача, щоб краще зрозуміти його функції та можливості. До основних функцій програмного забезпечення належать:

- імпортування тривимірної моделі;
- відображення каркасу моделі;
- оперування параметрами огляду тривимірної моделі;
- зміна параметрів відображення тривимірної моделі;
- зафарбовування тривимірної моделі за методом Гуро;
- експортування зображення із подальшим рендерингом.

Наступним етапом розробки інструкції користувача є опис процесу користування розробленим програмним забезпеченням.

Після інсталяції та запуску програмного забезпечення, користувач очікує на завантаження робочого середовища, після чого може використовувати реалізований функціонал програмного забезпечення. Першим кроком взаємодії із програмними засобами передбачається імпортування тривимірної моделі у форматі .obj або вибір базової моделі. Після візуалізації тривимірної моделі у робочому середовищі, користувач має доступ до панелі інструментів та меню програмного забезпечення. Панель інструментів включає в себе ряд функцій для операцій з тривимірною моделлю та її відображенням у робочому середовищі. Користувач може виконати зафарбовування моделі за методом Гуро, що дозволяє створювати плавні переходи між кольорами та тіні, надаючи тривимірній моделі більш реалістичний вигляд. Також, користувач може відобразити або приховати каркас (лінії формування) тривимірної моделі за допомогою відповідної функції на панелі інструментів. Панель інструментів також надає можливість користувачу впливати на відображення моделі, змінюючи параметри прозорості та перспективи. Зміна прозорості дозволяє встановити ступінь прозорості тривимірних об'єктів, що дозволяє створити ефект прозорості або полупрозорості. Встановлення перспективи змінює кут огляду на модель, що додає глибину та реалізм відображенню. Крім того, користувач має можливість змінювати колір заливки програмного забезпечення для моделі. Ця функція дозволяє користувачеві вибрати бажаний колір або палітру кольорів для зафарбовування тривимірних об'єктів з використанням методу Гуро.

У стрічці меню користувач має змогу відкривати допоміжні вікна програми, а саме «Про програму» та «Довідка». Також користувач може імпортувати тривимірну модель, натиснувши кнопку «Вправо», а також експортувати зображення, яке перед цим буде проходити процес додаткового рендерингу.

На рисунку 4.9 відображено результат рендерингу імпортованої тривимірної моделі.

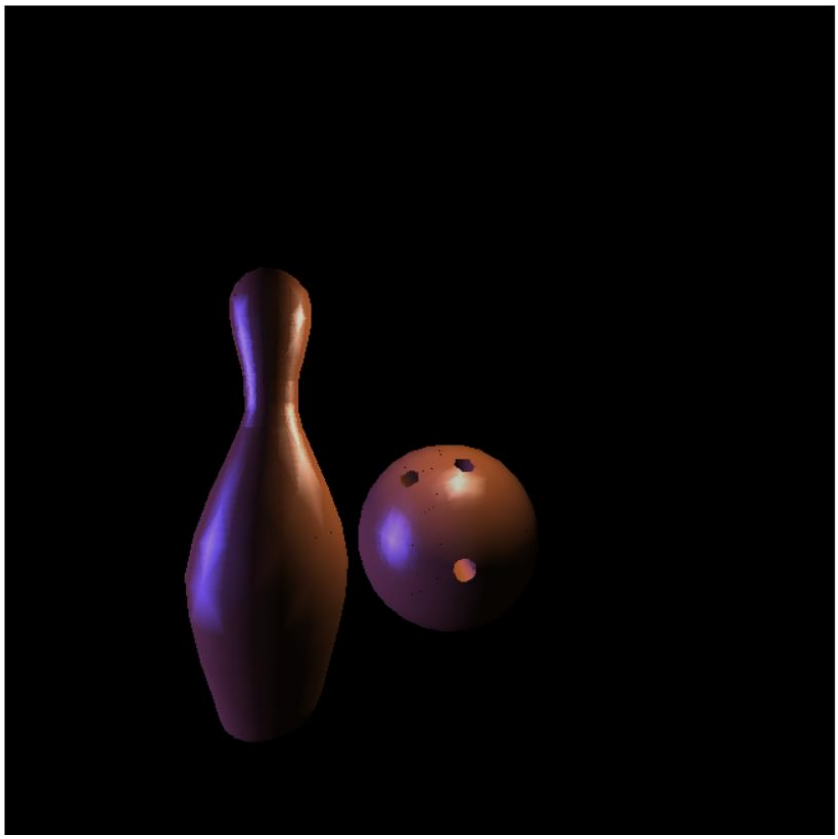


Рисунок 4.9 – Експортоване зображення

4.3 Висновки

У четвертому розділі було проведено тестування програмних модулів програмних засобів високопродуктивного формування зображень з використанням методу Гуро. Було перевірено реакцію програмних засобів на помилкові ситуації, вхідні дані різних форматів. Також було розроблено інструкцію користувача по початковій експлуатації програмного продукту.

ВИСНОВКИ

У бакалаврській дипломній роботі було розроблено методи та програмні засоби для високопродуктивного формування зображень за методом Гуро. Було виконано обґрунтування розробки власних методів формування зображень за методом Гуро з метою підвищення продуктивності. Розроблено методи високопродуктивного формування зображень за методом Гуро використовуючи засоби тривимірного моделювання та розроблені методи, що дозволяють ефективно використовувати обчислювальні ресурси та зменшити час обробки графічних зображень

На основі отриманих в бакалаврській дипломній роботі теоретичних положень запропоновано алгоритми та розроблено програмні засоби для високопродуктивного формування зображень шляхом зафарбовування тривимірних об’єктів за методом Гуро.

Розроблені засоби вимірювання антропометричних параметрів можна використати для побудови високопродуктивних систем рендерингу.

Ключові слова: тривимірне моделювання, метод Гуро, рендеринг, продуктивність, програмне забезпечення.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. A. Watt та S. Maddock, 3D Computer Graphics. Pearson Educ., Limited, 2021.

2. J. Cardoso, 3D Photorealistic Rendering. Boca Raton : CRC Press, 2016.: K Peters/CRC Press, 2017.

3. Затемнення за ГУРО [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki (дата звернення: 18.03.2024)

4. Gouraud Shading in Computer Graphics [Електронний ресурс] – Режим доступу до ресурсу: https://www.geeksforgeeks.org/gouraud-shading-in-computer-graphics/ (дата звернення: 18.03.2024)

5. Gouraud Shading [Електронний ресурс] – Режим доступу до ресурсу: https://garykeen27.wixsite.com/portfolio/gouraud-shading (дата звернення: 18.03.2024)

6. Романюк О. Н., Комісарик А.С. Аналіз методу ГУРО для формування тривимірних зображень // Матеріали міжнародної науково-практичної інтернет-конференції студентів аспірантів та молодих науковців «Молодь в науці: дослідження, проблеми, перспективи (МН-2024)» : збірник матеріалів. – Вінниця: ВНТУ, 2024.

7. Blender [Електронний ресурс] – Режим доступу до ресурсу: https://www.blender.org/ (дата звернення: 28.03.2024)

8. Autodesk Maya [Електронний ресурс] – Режим доступу до ресурсу: https://www.autodesk.com/products/maya/overview?term=1- YEAR&tab= subscription (дата звернення: 28.03.2024)

9. Cinema 4D [Електронний ресурс] – Режим доступу до ресурсу: https://www.maxon.net/en (дата звернення: 28.03.2024)

10. Unity [Електронний ресурс] – Режим доступу до ресурсу: https://unity.com/ru/releases/editor/whats-new/2022.2.10 (дата звернення: 28.03.2024)

11. 3D MAX [Електронний ресурс] – Режим доступу до ресурсу: https://apps.autodesk.com/3DSMAX/ (дата звернення: 28.03.2024)

12. Gouraud shading [Електронний ресурс] – Режим доступу до ресурсу: https://graphics.fandom.com/wiki/Gouraud_shading (дата звернення: 28.03.2024)

13. How to Gouraud shading with diffuse lighting in WebGl? [Електронний ресурс] – Режим доступу до ресурсу: https://stackoverflow.com/questions/55634273/how-to-gouraud-shading (дата звернення: 28.03.2024)

14. Hast A. Models that use logarithmic and power functons [Text] / A. Hast, O. N. Romanyuk, O. M. Lyashenko // The Sixth International Conference “INTERNET –EDUCATION - SCIENCE”, Vinnytsia, Ukraine, October 7 –11, 2008 . – 2008. – С. 538-542.

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

16. Романюк О. Н. Високопродуктивні методи та засоби зафарбовування тривимірних графічних об’єктів. Монографія. // О. Н. Романюк – Вінниця : УНІВЕСУМ-Вінниця, 2006

17. Visio [Електронний ресурс] – Режим доступу до ресурсу: https://www.microsoft.com/uk-ua/microsoft-365/visio/flowchart-software (дата звернення: 08.04.2024)

18. Що таке тестування програмного забезпечення? [Електронний ресурс] – Режим доступу до ресурсу: https://qalight.ua/baza-znaniy/shho-take-testuvannya-programnogo-zabezpechennya/ (дата звернення: 20.04.2024)

19. Функціональне тестування [Електронний ресурс] – Режим доступу до ресурсу: https://mate.academy/blog/qa/functional-non-functional-testing (дата звернення: 20.04.2024)

20. Тестування на відмову та поновлення [Електронний ресурс] – Режим доступу до ресурсу: https://qalight.ua/baza-znaniy/testuvannya-na-vidmovu-ta-ponovlennya (дата звернення: 20.04.2024)

ДОДАТКИ

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., професор
_____ О. Н. Романюк
«12» березня 2024 р.

Технічне завдання

на бакалаврську дипломну роботу «Розробка методів і програмних засобів для високопродуктивного формування зображень з використанням методу ГУРО» за спеціальністю 121 – Інженерія програмного забезпечення

Керівник бакалаврської дипломної роботи:
_____ д.т.н., професор О. Н. Романюк
«12» березня 2024 р.

Виконав:
_____ студент гр. 2ПІ-20б А. С. Комісарик
«12» березня 2024 р.

Вінниця – 2024

1. Найменування та галузь застосування

Бакалаврська дипломна робота: «Розробка методів і програмних засобів для високопродуктивного формування зображень з використанням методу ГУРО».

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 р. ректора ВНТУ про закріплення тем БДР.

3. Мета та призначення розробки

Метою дослідження є підвищення продуктивності та роздільної здатності графічних зображень за рахунок використанням методу ГУРО, що дозволить ефективно використовувати обчислювальні ресурси та зменшити час обробки графічних зображень.

4. Вихідні дані для проведення НДР

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

1. A. Watt та S. Maddock, 3D Computer Graphics. Pearson Educ., Limited, 2021.
2. J. Cardoso, 3D Photorealistic Rendering. Boca Raton : CRC Press, 2016.: K Peters/CRC Press, 2017.

5. Технічні вимоги

Вихідні дані до роботи: модель освітлення – модель Бліна; графічний режим – TrueColor; вихідні дані для формування зображень за методом Гуро – координати вершин трикутників, інтенсивності кольору у вершинах трикутника; вихідні дані – кінцеве зображення, зафарбоване згідно методу Гуро.

6. Конструктивні вимоги

Графічна та текстова документація повинна відповідати діючим стандартам України.

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

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

8. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ

9. Стадії та етапи розробки:

№ з\п	Назва етапів бакалаврської дипломної роботи	Строк виконання етапів роботи
1	Аналіз розвитку технологій формування зображень з використанням методу Гуро	14.03.24 –22.03.24
2	Розробка методів для формування зображень з використанням методу Гуро	22.03.24 – 19.04.24
3	Розробка алгоритмів роботи програмного забезпечення	19.04.24 –10.05.24
4	Розробка модулів програмного засобу	10.05.24 –24.05.24
5	Тестування програми	24.05.24 – 27.05.24
6	Оформлення матеріалів до захисту БДР	27.05.24 – 03.06.24

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.

Додаток Б – Протокол перевірки на плагіат

ПРОТОКОЛ
ПЕРЕВІРКИ БАКАЛАВРСЬКОЇ ДИПЛОМНОЇ РОБОТИ
НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: Розробка методів і програмних засобів для високопродуктивного формування зображень з використанням методу ГУРО

Тип роботи: БДР

Підрозділ: кафедра програмного забезпечення, ФІТКІ

Науковий керівник: Романюк О.Н.

Оригінальність	90,1%
Схожість	9,9%

Аналіз звіту подібності

▪ **Запозичення, виявлені у роботі, оформленні коректно і не містять ознак плагіату.**

☐ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цілісності 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 приховання недобросовісних запозичень.

Особа, відповідальна за перевірку  Черноволик Г.О.

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck

Автор роботи  Комісарик А.С.

Керівник роботи Романюк О.Н.

Додаток В – Лістинг програмного забезпечення

```
  public SimpInterpreter(String filename,
      Drawable drawable,
      RendererTrio renderers) {
    this.drawable = drawable;
    this.depthCueingDrawable = drawable;
    this.lineRenderer = renderers.getLineRenderer();
    this.filledRenderer = renderers.getFilledRenderer();
    this.wireframeRenderer = renderers.getWireframeRenderer();
    this.currentFaceShader = nullFaceShader;
    this.currentVertexShader = phongVertexShader;
    this.currentPixelShader = phongPixelShader;
    this.defaultColor = Color.WHITE;
    this.ambientLight = Color.BLACK;
    this.lightList =  new ArrayList<Light>();
    this.lighting = new Lighting (ambientLight, lightList);

    makeWorldToScreenTransform(drawable.getDimensions());
    reader = new LineBasedReader(filename);
    kSpecular = 0.3;
    specularExponent = 8;
    readerStack = new Stack<>();

    CTMStack = new Stack<Transformation>();
    shaderStyle = ShaderStyle.PHONG;
    renderStyle = RenderStyle.FILLED;
    CTM = Transformation.identity();
    worldToCamera = Transformation.identity();
  }
  private void makeWorldToScreenTransform(Dimensions dimensions ) {
    worldToScreen = Transformation.identity();
    double scaleX = dimensions.getWidth() / (WORLD_HIGH_X - WORLD_LOW_X);
    double scaleY = dimensions.getHeight() / (WORLD_HIGH_Y - WORLD_LOW_Y);
    worldToScreen.scale(scaleX, scaleY, 1);
    worldToScreen.translate(dimensions.getWidth()/2.0, dimensions.getHeight()/2.0, 0);
  }

  private void makeProjectedToScreenTransform(Dimensions dimensions,double CamLowX,
double CamLowY, double CamHighX, double CamHighY) {

    double scaleX = dimensions.getWidth() / (CamHighX - CamLowX);
    double scaleY = scaleX;

    double ratio = (CamHighY - CamLowY)/(CamHighX - CamLowX)>1? 1: (CamHighY -
CamLowY)/(CamHighX - CamLowX);

    double cameraCenterX = (CamHighX + CamLowX)/2;
    double cameraCenterY = (CamHighY + CamLowY)/2;

    double cameraShiftX = dimensions.getWidth()/2.0 - cameraCenterX*dimensions.getWidth();
    double cameraShiftY = dimensions.getHeight()/2.0 -
ratio*cameraCenterY*dimensions.getHeight();
```

```
    projectedToScreen = Transformation.identity();
    projectedToScreen.scale(scaleX, scaleY, 1);
    projectedToScreen.translate((int)Math.round(cameraShiftX), (int)Math.round(cameraShiftY),
0);
  }
  private Vertex3D gouraudShadeVetex(Polygon poly, Vertex3D vert) {
    Halfplane3DH normal;
    if (vert.hasNormal()) {
      normal = vert.getNormal();
      Transformation normTrans = worldToCamera.preMultiplyTransformation(CTM);
      normTrans.invert();
      normal = normTrans.normalPostMultiplyThisTransform(normal);

    }else {
      normal = poly.getFaceNormal().toHalfPlaneNormal();

    }

    Color kDiffuse = vert.getColor();
    Color c =  lighting.light(vert, kDiffuse, normal, kSpecular, specularExponent);

    vert.setVertexShaderColor(c);
    return vert;
  }

  private Color gouraundShadePixel(Polygon poly, Vertex3D vert) {
    return vert.getVertexShaderColor();
  }


    private void interpretLight(String[] tokens) {
    double r = cleanNumber(tokens[1]);
    double g = cleanNumber(tokens[2]);
    double b = cleanNumber(tokens[3]);
    double fatA = cleanNumber(tokens[4]);
    double fatB = cleanNumber(tokens[5]);

    Point3DH origin = new Point3DH(0,0,0);
    Point3DH lightInCS = worldToCamera.transformPoint3DH(CTM.transformPoint3DH(origin));

    Light i = new Light(r, g, b, fatA, fatB, lightInCS);

    lightList.add(i);

  }
  private void wire() {
    renderStyle = RenderStyle.WIREFRAME;
  }
  private void filled() {
    renderStyle = RenderStyle.FILLED;
  }
```

```
    private void interpretGouraud(String[] tokens) {
      shaderStyle = ShaderStyle.GOURAUD;
      lighting = new Lighting (ambientLight, lightList);
      currentFaceShader = nullFaceShader;
      currentVertexShader = gouraudVertexShader;
      currentPixelShader = colorInterpolatingPixelShader;
    }

    private void interpretObj(String[] tokens) {
      String tempfile = tokens[1];
      tempfile = tempfile.replace("\"","");
      String fileN = "D:\\Универ\\4 курс\\2 семестр\\Комісарик\\Програма
ЮДР\\komisaruk.bdr.gouraund\\src\\main\\java\\"+tempfile+".obj";
      ObjReader objfile = new ObjReader(fileN, defaultColor);
      objfile.render(this);
    }
    private void interpretAmbient(String[] tokens) {
      double r = cleanNumber(tokens[1]);
      double g = cleanNumber(tokens[2]);
      double b = cleanNumber(tokens[3]);
      ambientLight = new Color(r, g, b);
      lighting = new Lighting (ambientLight, lightList);
    }

    private void interpretSurface(String[] tokens) {
      double r = cleanNumber(tokens[1]);
      double g = cleanNumber(tokens[2]);
      double b = cleanNumber(tokens[3]);
      kSpecular = cleanNumber(tokens[4]);
      specularExponent  = cleanNumber(tokens[5]);
      defaultColor = new Color(r, g, b);
    }

    private void interpretCamera(String[] tokens) {

      double camLowX  = cleanNumber(tokens[1]);
      double camLowY  = cleanNumber(tokens[2]);
      double camHighX = cleanNumber(tokens[3]);
      double camHighY = cleanNumber(tokens[4]);
      double camNear = cleanNumber(tokens[5]);
      double camFar = cleanNumber(tokens[6]);

      makeProjectedToScreenTransform(drawable.getDimensions(), camLowX, camLowY,
camHighX, camHighY);

      clipper =  new Clipper(camLowX, camLowY , camHighX, camHighY, camNear, camFar,
VIEW_PLANE );
      worldToCamera.copyTransform(CTM);
      worldToCamera.invert();
    }
    public Vertex3D transformToCamera(Vertex3D vertex) {
      Vertex3D transformedVertex =
```

```
worldToCamera.transformVertex3D(CTM.transformVertex3D(vertex));

      Point3DH cameraPoint = new
Point3DH(transformedVertex.getX(),transformedVertex.getY(),transformedVertex.getZ());
      transformedVertex.setCameraPoint(cameraPoint);

      return  transformedVertex;
    }

    private void push() {
      Transformation x = new Transformation();
      x.copyTransform(CTM);
      CTMStack.push(x);
    }

    private void pop() {
      CTM= CTMStack.pop();
    }
    private void interpretFile(String[] tokens) {
      String quotedFilename = tokens[1];
      int length = quotedFilename.length();
      assert quotedFilename.charAt(0) == "" && quotedFilename.charAt(length-1) == "";
      String filename = quotedFilename.substring(1, length-1);
      file("D:\\Универ\\4 курс\\2 семестр\\Комісарик\\Програма
ЮДР\\komisaruk.bdr.gouraund\\src\\main\\java\\"+filename + ".simp");
    }

    private void file(String filename) {
      readerStack.push(reader);
      reader = new LineBasedReader(filename);
    }

    private void interpretScale(String[] tokens) {
      double sx = cleanNumber(tokens[1]);
      double sy = cleanNumber(tokens[2]);
      double sz = cleanNumber(tokens[3]);
      CTM.post_scale(sx, sy, sz);

    }
    private void interpretTranslate(String[] tokens) {
      double tx = cleanNumber(tokens[1]);
      double ty = cleanNumber(tokens[2]);
      double tz = cleanNumber(tokens[3]);
      CTM.post_translate(tx, ty, tz);

    }
    private void interpretRotate(String[] tokens) {
      String axisString = tokens[1];
      double angleInDegrees = cleanNumber(tokens[2]);
      double angleInRaidiance =  (angleInDegrees/180.0)*(Math.PI);

      switch(axisString) {
```

```
      case "X": CTM.post_rotateX(angleInRaidiance);break;
      case "Y": CTM.post_rotateY(angleInRaidiance);break;
      case "Z": CTM.post_rotateZ(angleInRaidiance);break;
      default: break;
      }
    }

    private static double cleanNumber(String string) {
      return Double.parseDouble(string);
    }

    private enum VertexColors {
      COLORED(NUM_TOKENS_FOR_COLORED_VERTEX),
      UNCOLORED(NUM_TOKENS_FOR_UNCOLORED_VERTEX);

      private int numTokensPerVertex;

      private VertexColors(int numTokensPerVertex) {
        this.numTokensPerVertex = numTokensPerVertex;
      }
      public int numTokensPerVertex() {
        return numTokensPerVertex;
      }
    }
    private Vertex3D interpretVertex(String[] tokens, int startingIndex, VertexColors colored) {
      Point3DH point = interpretPoint(tokens, startingIndex);

      Color color = defaultColor;
      if(colored == VertexColors.COLORED) {
        color = interpretColor(tokens, startingIndex + NUM_TOKENS_FOR_POINT);
      }

      Vertex3D newPoint = new Vertex3D(point, color);
      Vertex3D transformVetex = transformToCamera(newPoint);
      transformVetex.setCameraPoint(newPoint);
      return  transformVetex;
    }

    public Vertex3D[] interpretVertices(String[] tokens, int numVertices, int startingIndex) {
      VertexColors vertexColors = verticesAreColored(tokens, numVertices);
      Vertex3D vertices[] = new Vertex3D[numVertices];

      for(int index = 0; index < numVertices; index++) {
        vertices[index] = interpretVertex(tokens, startingIndex + index *
vertexColors.numTokensPerVertex(), vertexColors);
      }
      return vertices;
    }
    private void interpretPolygon(String[] tokens) {

      Vertex3D[] polygon = interpretVertices(tokens, 3, 1);
```

```
      Chain verChain= clipper.clipPolyinZ(polygon[0], polygon[1], polygon[2]);
      Chain projectChain =  new Chain();
      for(int i=0; i< verChain.length(); i++) {
        Vertex3D point = verChain.get(i);

        point = projectVertexToScreenSpace(point);
        point = moveProjectedVertexToScreenCenter(point);
        projectChain.add(point);
      }

      projectChain = clipper.clipRightAfterProj (projectChain, drawable.getWidth());
      projectChain = clipper.clipLeftAfterProj (projectChain, 0);
      projectChain = clipper.clipTopAfterProj (projectChain, getClipWindowTop());
      projectChain = clipper.clipBottomAfterProj (projectChain, getClipWindowBottom());

      for(int i=1; i< projectChain.length()-1; i++) {

        Vertex3D p1 = projectChain.get(0);
        Vertex3D p2 = projectChain.get(i);
        Vertex3D p3 = projectChain.get(i+1);
       filledRenderer.drawPolygon(Polygon.make(p1,p2,p3),  depthCueingDrawable,
nullFaceShader, gouraudVertexShader, colorInterpolatingPixelShader);
        }

        if(shaderStyle == ShaderStyle.LIGHT) {
          filledRenderer.drawPolygon(Polygon.make(p1,p2,p3),  depthCueingDrawable);
        }

      }
    }

    public int getClipWindowTop() {
      double clipWindowRatio = clipper.ratio;
      double clipWindowTop = 0.5* drawable.getHeight() + 0.5*
drawable.getHeight()*clipWindowRatio;
      return (int)Math.round(clipWindowTop);
    }

    public int getClipWindowBottom() {
      double clipWindowRatio = clipper.ratio;
      double clipWindowBottom = 0.5* drawable.getHeight() - 0.5*
drawable.getHeight()*clipWindowRatio;
      return (int)Math.round(clipWindowBottom);
    }

    public VertexColors verticesAreColored(String[] tokens, int numVertices) {
      return hasColoredVertices(tokens, numVertices) ? VertexColors.COLORED :
                                  VertexColors.UNCOLORED;
    }
    public boolean hasColoredVertices(String[] tokens, int numVertices) {
      return tokens.length == numTokensForCommandWithNVertices(numVertices);
    }
```

```
  public int numTokensForCommandWithNVertices(int numVertices) {
    return NUM_TOKENS_FOR_COMMAND +
numVertices*(NUM_TOKENS_FOR_COLORED_VERTEX);
  }

  public Vertex3D projectVertexToScreenSpace(Vertex3D point) {
    double factor = VIEW_PLANE/point.getZ();

    Vertex3D projectVertex = new Vertex3D(factor*point.getX(), factor*point.getY(),
point.getZ(), point.getColor());

    projectVertex.setCameraPoint(point);

    if (point.hasNormal()) {
      projectVertex.setNormal(point.getNormal());
    }

    if (point.isForPhongShading()) {
      projectVertex.setToPhongShading();
    }

    return projectVertex;

  }
  public Vertex3D moveProjectedVertexToScreenCenter(Vertex3D point) {
    return projectedToScreen.transformVertex3D(point);
  }
  public static Point3DH interpretPointWithW(String[] tokens, int startingIndex) {
    double x = cleanNumber(tokens[startingIndex]);
    double y = cleanNumber(tokens[startingIndex + 1]);
    double z = cleanNumber(tokens[startingIndex + 2]);
    double w = cleanNumber(tokens[startingIndex + 3]);
    Point3DH point = new Point3DH(x, y, z, w);
    return point;
  public void interpretDepth(String[] tokens) {

    fogNear = (int) cleanNumber(tokens[1]);
    fogFar = (int) cleanNumber(tokens[2]);

    double r = cleanNumber(tokens[3]);
    double g = cleanNumber(tokens[4]);
    double b = cleanNumber(tokens[5]);

    Color farColor = new Color(r,g,b);
    depthCueingDrawable =  new DepthCueingDrawable(drawable, fogNear, fogFar, farColor);

  };

  public static Point3DH interpretPoint(String[] tokens, int startingIndex) {
    double x = cleanNumber(tokens[startingIndex]);
    double y = cleanNumber(tokens[startingIndex + 1]);
    double z = cleanNumber(tokens[startingIndex + 2]);
```

```
        return new Point3DH(x, y, z);
    }


    public static Color interpretColor(String[] tokens, int startingIndex) {
        double r = cleanNumber(tokens[startingIndex]);
        double g = cleanNumber(tokens[startingIndex + 1]);
        double b = cleanNumber(tokens[startingIndex + 2]);
        return new Color(r, g, b);
    }

    private void line(Vertex3D p1, Vertex3D p2) {
        Vertex3D screenP1 = transformToCamera(p1);
        Vertex3D screenP2 = transformToCamera(p2);
    }

    private void polygon(Vertex3D p1, Vertex3D p2, Vertex3D p3) {
        Vertex3D screenP1 = transformToCamera(p1);
        Vertex3D screenP2 = transformToCamera(p2);
        Vertex3D screenP3 = transformToCamera(p3);
    }
}
    public ShapesGUI() {
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        this.draw = new mainDraw();

        addKeyListener(this);
        addMouseListener(this);
        addMouseMotionListener(this);
        addMouseWheelListener(this);
        setFocusable(true);
        setFocusTraversalKeysEnabled(false);

        JPanel bottom_panel = new JPanel();
        bottom_panel.setBackground(Color.BLACK);
        getContentPane().add(bottom_panel, BorderLayout.SOUTH);

        JButton btnPreviousObject = new JButton("Попередня модель");
        btnPreviousObject.setFocusable(false);
        bottom_panel.add(btnPreviousObject);
        btnPreviousObject.setAction(action_prev_object);

        JButton btnNextObject = new JButton("Наступна модель");
        btnNextObject.setFocusable(false);
        bottom_panel.add(btnNextObject);
        btnNextObject.setAction(action_next_object);

        JButton btnShowInstructions = new JButton("Показати інструкції");
        btnShowInstructions.setFocusable(false);
        bottom_panel.add(btnShowInstructions);
        btnShowInstructions.setAction(action_show_instructions);
```

```
JButton btnShowAbout = new JButton("Про програму");
btnShowAbout.setFocusable(false);
bottom_panel.add(btnShowAbout);
btnShowAbout.setAction(action_show_about);

JButton btnExportImage = new JButton("Експортувати зображення");
btnExportImage.setFocusable(false);
bottom_panel.add(btnExportImage);
btnExportImage.setAction(action_export_image);

JPanel top_panel = new JPanel();
top_panel.setBackground(Color.BLACK);
getContentPane().add(top_panel, BorderLayout.NORTH);

JCheckBox chckbxGouRaundShading = new JCheckBox("Зафарбовування за ГУРО");
chckbxGouRaundShading.setFocusable(false);
top_panel.add(chckbxGouRaundShading);
chckbxGouRaundShading.setSelected(mainDraw.enableShading);
chckbxGouRaundShading.setAction(action_GouRaund_shading);
chckbxGouRaundShading.setForeground(Color.BLACK);

JCheckBox chckbxWireframe = new JCheckBox("Каркас");
chckbxWireframe.setFocusable(false);
top_panel.add(chckbxWireframe);
chckbxWireframe.setSelected(mainDraw.enableWireframe);
chckbxWireframe.setForeground(Color.BLACK);
chckbxWireframe.setAction(action_wire_shading);
JCheckBox chckbxTransparency = new JCheckBox("Прозорість");
chckbxTransparency.setFocusable(false);
top_panel.add(chckbxTransparency);
chckbxTransparency.setSelected(mainDraw.enableTransparency);
chckbxTransparency.setForeground(Color.BLACK);
chckbxTransparency.setAction(action_transparency);

JCheckBox chckbxAntialias = new JCheckBox("Згладжування");
chckbxAntialias.setFocusable(mainDraw.enableAntialias);
top_panel.add(chckbxAntialias);
chckbxAntialias.setForeground(Color.BLACK);
chckbxAntialias.setAction(action_antialias);

JCheckBox chckbxPerspective = new JCheckBox("Перспектива");
chckbxPerspective.setFocusable(false);
top_panel.add(chckbxPerspective);
chckbxPerspective.setSelected(mainDraw.enablePerspective);
chckbxPerspective.setForeground(Color.BLACK);
chckbxPerspective.setAction(action_perspective);

JCheckBox chckbxColor = new JCheckBox("Колір");
chckbxColor.setFocusable(false);
top_panel.add(chckbxColor);
chckbxColor.setSelected(mainDraw.enableColor);
```

```
    chckbxColor.setForeground(Color.BLACK);
    chckbxColor.setAction(action_color);
  }

  public static void main(String[] args) {
    float windowToScreenScale = 0.85f;
    Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
    double screenHeight = screenSize.getHeight();
    double screenWidth = screenSize.getWidth();
    int width = (int)(screenSize.getWidth()*windowToScreenScale);
    int height = (int)(screenSize.getHeight()*windowToScreenScale);
    int x0 = (int) screenWidth/2 - width/2;
    int y0 = (int) screenHeight/2 - height/2;
    frame = new ShapesGUI();
    frame.setResizable(true);
    frame.setBounds(x0, y0, width, height);
    frame.setMinimumSize(new Dimension(640, 480));
    frame.setPreferredSize(new Dimension(width, height));
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.getContentPane().add(frame.draw);
    frame.setTitle("Gouraund shade and export");
    frame.pack();
    frame.setVisible(true);
  }
  private class SwingAction_Prev extends AbstractAction {
    public SwingAction_Prev() {
      putValue(NAME, "<--");
    }
    public void actionPerformed(ActionEvent e) {
      draw.changeShape(-1);
    }
  }
  private class SwingAction_Next extends AbstractAction {
    public SwingAction_Next() {
      putValue(NAME, "-->");
    }
    public void actionPerformed(ActionEvent e) {
      draw.changeShape(1);
    }
  }
  private class SwingAction_GouRaund extends AbstractAction {
    public SwingAction_GouRaund() {
      putValue(NAME, "Зафарбовування за ГУРО");
    }
    public void actionPerformed(ActionEvent e) {
      draw.setGouRaundShading();
    }
  }
  private class SwingAction_Wire extends AbstractAction {
    public SwingAction_Wire() {
      putValue(NAME, "Каркас");
    }
```

```
    public void actionPerformed(ActionEvent e) {
      draw.setWireframeShading();
    }
  }
  private class SwingAction_Trans extends AbstractAction {
    public SwingAction_Trans() {
      putValue(NAME, "Прозорість");
    }
    public void actionPerformed(ActionEvent e) {
      draw.setTransparency();
    }
  }
  private class SwingAction_Alias extends AbstractAction {
    public SwingAction_Alias() {
      putValue(NAME, "Згладжування");
    }
    public void actionPerformed(ActionEvent e) {
      draw.setAntialias();
    }
  private class SwingAction_Pers extends AbstractAction {
    public SwingAction_Pers() {
      putValue(NAME, "Перспектива");
    }
    public void actionPerformed(ActionEvent e) {
      draw.changePerspective();
    }
  }
  private class SwingAction_Color extends AbstractAction {
    public SwingAction_Color() {
      putValue(NAME, "Колір");
    }
    public void actionPerformed(ActionEvent e) {
      draw.changeColor();
    }
  }
  private class SwingAction_ShowInstructions extends AbstractAction {
    public SwingAction_ShowInstructions() {
      putValue(NAME, "Показати інструкції");
    }
    public void actionPerformed(ActionEvent e) {
      new InstructionsWindow();
    }
  }
  private class SwingAction_ShowAbout extends AbstractAction {
    public SwingAction_ShowAbout() {
      putValue(NAME, "Про програму");
    }
    public void actionPerformed(ActionEvent e) {
      new AboutWindow();
    }
  }
  private class SwingAction_ExportImage extends AbstractAction {
```

```
    public SwingAction_ExportImage() {
      putValue(NAME, "Експортувати зображення");
    }
    public void actionPerformed(ActionEvent e) {
      try {
        BufferedImage image = new BufferedImage(draw.getWidth(), draw.getHeight(),
BufferedImage.TYPE_INT_ARGB);
        Graphics2D g2 = image.createGraphics();
        draw.paint(g2);
        g2.dispose();

        File output = new File("image.png");
        ImageIO.write(image, "png", output);
      } catch (Exception ex) {
        ex.printStackTrace();
      }
    }
  }
}

@SuppressWarnings("serial")
class mainDraw extends JComponent {
  private static Graphics2D g2d; private static Shapes currentModel;   private static Random rand
= new Random();
  private static int window_width;   private static int window_height;
  private static int objScale0 = 200;
  private static int objScaleMin = 10;
  private static int objScaleMax = 1000;
    private static double[] objRot = {0,-60};
  private static double[] objRot0 = {0,-60};
  private static double[] objPos = {0,0,4};
  private static double[] objPos0 = {0,0,4};
  private static double objMinDistance = 2;
  private static double objMaxDistance = 10;
  private static int objType = 0;
  private static int objLast = 0;
  private static float objTransparency = 1.0f;
  private static Color objLineColor = Color.black;
  private static int[] objColor = {150,150,255};
  private static double[] lightPos = {-1,2,5};
  private static double[] lightPos0 = {-1,2,5};
  private static double lightAmbient = 0.2;
  private static double lightDiffuse = 0.6;
  private static double lightSpecular = 0.5;
  private static double lightShininess = 32;
  public static boolean enableWireframe = false;
  public static boolean enableShading = true;
  public static boolean enableTransparency = false;
  public static boolean enableAntialias = false;
  public static boolean enablePerspective = true;
  public static boolean enableColor = true;
  public void paintComponent(Graphics g) {
```

```
        super.paintComponent(g);
         Rectangle window = ShapesGUI.frame.getBounds();
         window_width = window.width;
         window_height = window.height;
        g2d = (Graphics2D) g;
         g2d.fillRect(0, 0, window_width, window_height);
         g2d.setComposite(AlphaComposite.getInstance(AlphaComposite.SRC_OVER,
objTransparency));

g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING,RenderingHints.VALUE_ANTIALIAS
_ON);
         drawShapes();
    }
    public void moveXY(double dx, double dy) {
      double speedScale = 255;
      objPos[0] = objPos0[0] + dx/speedScale;
      objPos[1] = objPos0[1] + dy/speedScale;
      repaint();
    }
    public void setXY() {
      objPos0[0] = objPos[0];
      objPos0[1] = objPos[1];
    }
    public void moveLight(double dx, double dy) {
      double speedScale = 255;
      lightPos[0] = lightPos0[0] - dx/speedScale;
      lightPos[1] = lightPos0[1] - dy/speedScale;
      repaint();
    }
    public void setLight() {
      lightPos0[0] = lightPos[0];
      lightPos0[1] = lightPos[1];
    }
    public void moveZ(double factor) {
      double initial = objPos[2];
      double moveScale = 10;
      objPos[2] += factor/moveScale;
      if (objPos[2] < objMinDistance || objPos[2] > objMaxDistance) {
        objPos[2] = initial;
      }
      repaint();
    }
    public void moveTheta(double tx, double ty) {
      double rotateScale = 5;
      objRot[0] = objRot0[0] + tx/rotateScale;
      objRot[1] = objRot0[1] + ty/rotateScale;
      objRot[1] = (objRot[1] > 0) ? 0 : objRot[1];
      objRot[1] = (objRot[1] < -180) ? -180 : objRot[1];
      repaint();
    }
    public void setTheta() {
      objRot0[0] = objRot[0];
```

```
    objRot0[1] = objRot[1];
  }
  public void zoom(double step) {
    int scalefactor0 = objScale0;
    double zoomScale = 20;
    objScale0 += step*zoomScale;
    if (objScale0 < objScaleMin || objScale0 > objScaleMax) {
      objScale0 = scalefactor0;
    }
    repaint();
  }
  public void changePerspective() {
    enablePerspective = !enablePerspective;
    repaint();
  }
  public void changeShape(int i) {
    int numShapes = 15;
    objType = (objType + i) % numShapes;
    objType = (objType < 0) ? numShapes+objType : objType;
    applyColor();
    repaint();
  }
  public void changeColor() {
    enableColor = !enableColor;
    applyColor();
    repaint();
  }
  public void applyColor() {
    int lightness = 100;

    if (enableColor) {
      objColor[0] = lightness+rand.nextInt(255-lightness);
      objColor[1] = lightness+rand.nextInt(255-lightness);
      objColor[2] = lightness+rand.nextInt(255-lightness);
    } else {
      objColor[0] = 255;
      objColor[1] = 255;
      objColor[2] = 255;
    }
  }
  public void setAntialias() {
    enableAntialias = !enableAntialias;
    repaint();
  }
  public void setGouRaundShading() {
    enableShading = !enableShading;
    changeLineColor();
  }
  public void setWireframeShading() {
    enableWireframe = !enableWireframe;
    changeLineColor();
  }
```

```
    private void changeLineColor() {
      if (enableShading & enableWireframe) {
        objLineColor = Color.black;
      } else {
        objLineColor = Color.white;
      }
      repaint();
    }
    public void setTransparency() {
      enableTransparency = !enableTransparency;
      if (enableTransparency) {
        objTransparency = 0.8f;
      } else {
        objTransparency = 1f;
      }
      repaint();
    }
    private static void drawShapes() {

      if (enableAntialias) {

g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING,RenderingHints.VALUE_ANTIALIAS
_ON);
      } else {

g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING,RenderingHints.VALUE_ANTIALIAS
_OFF);
      }
      g2d.setColor(Color.WHITE);
      int topLabelMargin = 100;
      FontMetrics fm = g2d.getFontMetrics();
      String titleText = "";
           switch (objType) {
        case 0:
          // Animate by recomputing when object is moving
          titleText = "cos(r^2)^2*exp(-r^2)";
          currentModel = drawFunction();
          break;
        case 1:
          titleText = "Sphere";
          if (objType != objLast)
            currentModel = drawSphere();
          break;
        case 2:
          titleText = "Cube";
          if (objType != objLast)
            currentModel = Shapes.cube;
          break;
        case 3:
          titleText = "Dodecahedron";
          if (objType != objLast)
            currentModel = Shapes.dodecahedron;
```

```
    break;
  case 4:
    titleText = "Icosahedron";
    if (objType != objLast)
      currentModel = Shapes.icosahedron;
    break;
  case 5:
    titleText = "Octahedron";
    if (objType != objLast)
      currentModel = Shapes.octahedron;
    break;
  case 6:
    titleText = "Rhombic Dodecahedron";
    if (objType != objLast)
      currentModel = Shapes.rhombicDodecahedron;
    break;
  case 7:
    titleText = "Soccer Ball";
    if (objType != objLast)
      currentModel = Shapes.socerBall;
    break;
  case 8:
    titleText = "Stellated Dodecahedron";
    if (objType != objLast)
      currentModel = Shapes.stellatedDodecahedron;
    break;
  case 9:
    titleText = "Stellated Icosahedron";
    if (objType != objLast)
      currentModel = Shapes.stellatedIcosahedron;
    break;
  case 10:
    titleText = "Stellated Octahedron";
    if (objType != objLast)
      currentModel = Shapes.stellatedOctahedron;
    break;
  case 11:
    titleText = "Tetrahedron";
    if (objType != objLast)
      currentModel = Shapes.tetrahedron;
    break;
  case 12:
    titleText = "Truncated Icosahedron";
    if (objType != objLast)
      currentModel = Shapes.truncatedIcosahedron;
    break;
  case 13:
    titleText = "Truncated Rhombic Dodecahedron";
    if (objType != objLast)
      currentModel = Shapes.truncatedRhombicDodecahedron;
    break;
  case 14:
```

```
            titleText = "Stellated Dodecahedron";
            if (objType != objLast)
              currentModel = Shapes.stellatedDodecahedron;
            break;
          default:
            break;
        }
        objLast = objType;
        g2d.drawString(titleText, (window_width-fm.stringWidth(titleText))/2, window_height-
topLabelMargin);
        drawFaces(currentModel.vertices, currentModel.faces, currentModel.normals);
      }
      private static double function(double x, double y) {
        double nWaves = 3;
        double decay = 2;
        double speed = 4;
        return -Math.pow(Math.cos(nWaves*(x*x +
y*y)+Math.toRadians(objRot[0])*speed),2)*Math.exp(-decay*(x*x + y*y));
      }
      private static Shapes drawFunction() {
        int nx = 100;
        int ny = 100;
        double[] xrange = {-1.5, 1.5};
        double[] yrange = {-1.5, 1.5};
        int[][] objFaces = new int[nx*ny][4];
        double[][] objVertices = new double[nx*ny*4][3];
        double[][] objNormals = new double[nx*ny][3];
        double dx = (xrange[1]-xrange[0])/nx;
        double dy = (yrange[1]-yrange[0])/ny;
        for (int j=0; j<ny; ++j) {
          for (int i=0; i<nx; ++i) {
            int f = j*nx + i;
            int v = 4*f;
            objVertices[v+0][0] = xrange[0]+i*dx;
            objVertices[v+0][1] = yrange[0]+j*dy;
            objVertices[v+0][2] = function(objVertices[v+0][0], objVertices[v+0][1]);
            objVertices[v+1][0] = xrange[0]+(i+1)*dx;
            objVertices[v+1][1] = yrange[0]+j*dy;
            objVertices[v+1][2] = function(objVertices[v+1][0], objVertices[v+1][1]);
            objVertices[v+2][0] = xrange[0]+(i+1)*dx;
            objVertices[v+2][1] = yrange[0]+(j+1)*dy;
            objVertices[v+2][2] = function(objVertices[v+2][0], objVertices[v+2][1]);
            objVertices[v+3][0] = xrange[0]+i*dx;
            objVertices[v+3][1] = yrange[0]+(j+1)*dy;
            objVertices[v+3][2] = function(objVertices[v+3][0], objVertices[v+3][1]);
            int face[] = {v+0, v+1, v+2, v+3};
            objFaces[f] = face;
            double edge1[] = subtract(objVertices[v+1], objVertices[v+0]);
            double edge2[] = subtract(objVertices[v+2], objVertices[v+1]);
            objNormals[f] = normalize(crossProduct(edge1,edge2));
          }
        }
```

```
    return new Shapes(objVertices, objFaces, objNormals);
  }

  // Method to draw a sphere
  private static Shapes drawSphere() {
    int nt = 48;            // theta resolution
    int np = 24;             // phi resolution
    double dt = 2*Math.PI/nt; // delta theta angle
    double dp = Math.PI/np;   // delta phi angle
     double radius = 1;       // radius
    // arrays to hold faces and vertices
    int[][] objFaces = new int[nt*np][4];
    double[][] objVertices = new double[nt*np*4][3];
    double[][] objNormals = new double[nt*np][3]; definitions
     for (int t=0; t<nt; ++t) {
      for(int p=0; p<np; ++p) {
        int f = t*np + p;
        int v = 4*f;
        objVertices[v+0][0] = radius * Math.sin(dp*p)*Math.cos(dt*t);
        objVertices[v+0][1] = radius * Math.sin(dp*p)*Math.sin(dt*t);
        objVertices[v+0][2] = radius * Math.cos(dp*p);
        // second quad vertex
        objVertices[v+1][0] = radius * Math.sin(dp*p)*Math.cos(dt*(t+1));
        objVertices[v+1][1] = radius * Math.sin(dp*p)*Math.sin(dt*(t+1));
        objVertices[v+1][2] = radius * Math.cos(dp*p);
        // third quad vertex
        objVertices[v+2][0] = radius * Math.sin(dp*(p+1))*Math.cos(dt*(t+1));
        objVertices[v+2][1] = radius * Math.sin(dp*(p+1))*Math.sin(dt*(t+1));
        objVertices[v+2][2] = radius * Math.cos(dp*(p+1));
        objVertices[v+3][0] = radius * Math.sin(dp*(p+1))*Math.cos(dt*t);
        objVertices[v+3][1] = radius * Math.sin(dp*(p+1))*Math.sin(dt*t);
        objVertices[v+3][2] = radius * Math.cos(dp*(p+1));
        if (p == 0) {
          objVertices[v+0][0] = radius * Math.sin(dp*(p+1))*Math.cos(dt*(t+1));
          objVertices[v+0][1] = radius * Math.sin(dp*(p+1))*Math.sin(dt*(t+1));
          objVertices[v+0][2] = radius * Math.cos(dp*(p+1));

          objVertices[v+2][0] = radius * Math.sin(dp*p)*Math.cos(dt*t);
          objVertices[v+2][1] = radius * Math.sin(dp*p)*Math.sin(dt*t);
          objVertices[v+2][2] = radius * Math.cos(dp*p);

          objVertices[v+3][0] = radius * Math.sin(dp*p)*Math.cos(dt*(t+1));
          objVertices[v+3][1] = radius * Math.sin(dp*p)*Math.sin(dt*(t+1));
          objVertices[v+3][2] = radius * Math.cos(dp*p);
          objVertices[v+1][0] = radius * Math.sin(dp*(p+1))*Math.cos(dt*t);
          objVertices[v+1][1] = radius * Math.sin(dp*(p+1))*Math.sin(dt*t);
          objVertices[v+1][2] = radius * Math.cos(dp*(p+1));
        }
        int face[] = {v+0, v+1, v+2, v+3};
        objFaces[f] = face;
        double edge1[] = subtract(objVertices[v+1], objVertices[v+0]);
        double edge2[] = subtract(objVertices[v+2], objVertices[v+1]);
```

```
        objNormals[f] = normalize(crossProduct(edge1,edge2));
      }
    }
    return new Shapes(objVertices, objFaces, objNormals);
}

private static void drawFaces(double[][] vertices, int[][] faces, double[][] normals) {
    double[][] face;
   int nfaces = faces.length;
   int nVertices = 0;
   double avgz = 0;
   BinaryTree bt = new BinaryTree();
    for (int f=0; f<nfaces; ++f) {
      nVertices = faces[f].length;
      face = new double[nVertices][3];
     for(int v=0; v<nVertices; ++v) {
        face[v][0] = vertices[(faces[f][v])][0];
        face[v][1] = vertices[(faces[f][v])][1];
        face[v][2] = vertices[(faces[f][v])][2];
     }
     face = transformFace(face);
      double[] norm = transformVertex(normals[f]);
     avgz = averageDistance(face);
     Node node = new Node(face, norm, -avgz);
     bt.add(node);
    }
    bt.InOrder();
}

public static void drawFace(double[][] face, double[] normal) {
    int[] xarr;
    int[] yarr;
   int nVertices;
   double[] projectedFace;
    nVertices = face.length;
    xarr = new int[nVertices+1];
    yarr = new int[nVertices+1];
   for(int i=0; i<nVertices; ++i) {
     projectedFace = face[i];
     if (enablePerspective) {
        projectedFace = perspectiveTransform(face[i]);
     }
     projectedFace = scale(projectedFace, objScale0);
     xarr[i] = (int) (window_width/2 + projectedFace[0]);
     yarr[i] = (int) (window_height/2 + projectedFace[1]);
   }
   xarr[nVertices] = xarr[0];
   yarr[nVertices] = yarr[0];
   if (enableShading) {
     shade(face, normal);
     g2d.fillPolygon(xarr, yarr, nVertices+1);
```

```
      }
      if (enableWireframe) {
        g2d.setColor(objLineColor);
        g2d.drawPolyline(xarr, yarr, nVertices+1);
      }
    }
    private static void shade(double[][] face, double[] normal) {
      double[] facePos = averagePosition(face);
      double[] viewDir = normalize(subtract(objPos, facePos));
      double[] lightDir = normalize(subtract(lightPos, facePos));
      double[] refDir = reflect(lightDir, normal);
      double amb = Math.max(lightAmbient, 0);
      double dif = lightDiffuse * Math.max(dotProduct(normal, lightDir), 0);
      double spec = lightSpecular * Math.pow(Math.max(dotProduct(viewDir, refDir), 0),
lightShininess);
      // compute color values based on lighting components
      int cr = (int) Math.min((amb + dif)*objColor[0] + spec*255, 255); // clamp values to 255
      int cg = (int) Math.min((amb + dif)*objColor[1] + spec*255, 255); // clamp values to 255
      int cb = (int) Math.min((amb + dif)*objColor[2] + spec*255, 255); // clamp values to 255
      // set color value
      g2d.setColor(new Color(cr, cg, cb));
    }

    private static double[][] transformFace(double[][] face){

      for (int v=0; v<face.length; ++v) {
        face[v] = transformVertex(face[v]);
      }

      return face;
    }

    private static double[] transformVertex(double[] vec){

      vec = rotateZX(vec, objRot[0], objRot[1]);

      vec = translate(vec, objPos[0], objPos[1], 0);

      return vec;
    }

    private static double averageDistance(double[][] face){
      double avgz = 0;

      for (int p=0; p<face.length; ++p) {
        avgz = avgz + face[p][2];
      }

      return avgz;
    }
    private static double[] averagePosition(double[][] face){
      double pos[] = {0,0,0};
```

```
    int nVertices = face.length;

    for (int p=0; p<nVertices; ++p) {
      pos[0] += face[p][0];
      pos[1] += face[p][1];
      pos[2] += face[p][2];
    }
    if (nVertices != 0) {
      return scale(pos, 1.0/face.length);
    } else {
      return pos;
    }
  }
  private static double[] reflect(double[] norm, double[] light){
    double ref[] = normalize( subtract( scale(norm, 2*dotProduct(norm, light)), light));
    return ref;
  }
  private static double[] normalize(double[] vec) {
    double mag = magnitude(vec);
    if (mag != 0) {
      return scale(vec, 1.0/mag);
    } else {
      return vec;
    }
  }
  private static double magnitude(double[] vec){
    double mag = Math.sqrt(vec[0]*vec[0] + vec[1]*vec[1] + vec[2]*vec[2]);;
    return mag;
  }
  private static double[] translate(double[] vec, double x, double y, double z) {
    double vec2[] = {vec[0]+x, vec[1]+y, vec[2]+z};
    return vec2;
  }
  private static double[] scale(double[] vec, double s) {
    double vec2[] = {vec[0]*s, vec[1]*s, vec[2]*s};
    return vec2;
  }
  private static double[] subtract(double[] vec1, double[] vec2) {
    double sub[] = {vec1[0]-vec2[0], vec1[1]-vec2[1], vec1[2]-vec2[2]};
    return sub;
  }
  private static double dotProduct(double[] vec1, double[] vec2){
    double dot = vec1[0]*vec2[0] + vec1[1]*vec2[1] + vec1[2]*vec2[2];
    return dot;
  }
  private static double[] crossProduct(double[] vec1, double[] vec2){
    double newVec[] = {vec1[1]*vec2[2] - vec1[2]*vec2[1],
                       vec1[2]*vec2[0] - vec1[0]*vec2[2],
                       vec1[0]*vec2[1] - vec1[1]*vec2[0]};
    return newVec;
```

Додаток Г – Графічна частина

ГРАФІЧНА ЧАСТИНА

РОЗРОБКА МЕТОДІВ І ПРОГРАМНИХ ЗАСОБІВ ДЛЯ ВИСОКОПРОДУКТИВНОГО ФОРМУВАННЯ ЗОБРАЖЕНЬ З ВИКОРИСТАННЯМ МЕТОДУ ГУРО

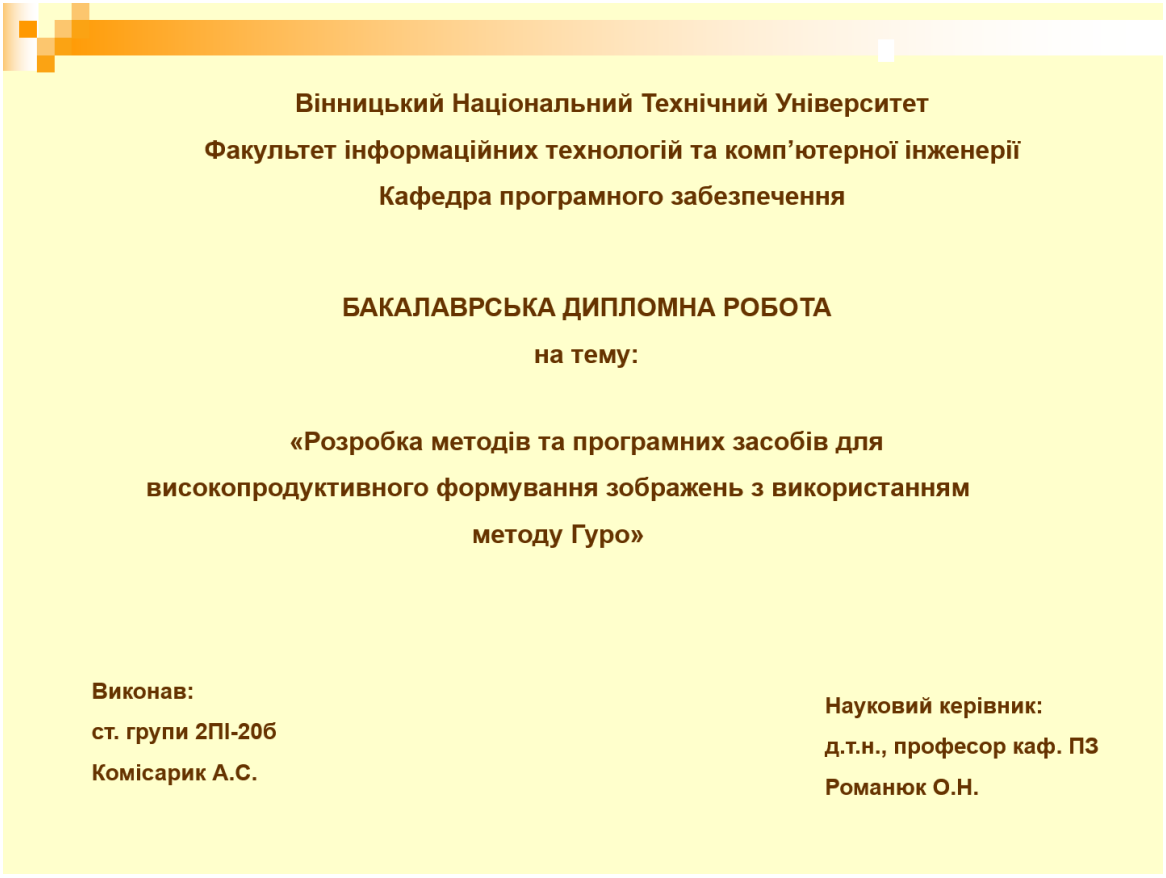
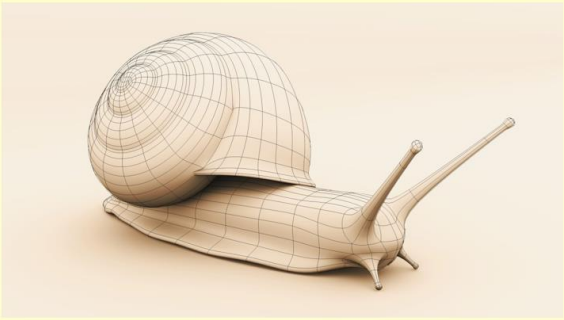


Рисунок Г.1 – Титульний аркуш

Рисунок Г.2 – Аналіз методів рендерингу

Галузі застосування за методом Гуро



1. Комп’ютерні ігри
2. Анімація та зафарбовування
3. Зафарбовування тривимірних моделей
4. Проектування та інженерія
5. Візуалізація

Рисунок Г.3 – Галузі застосування

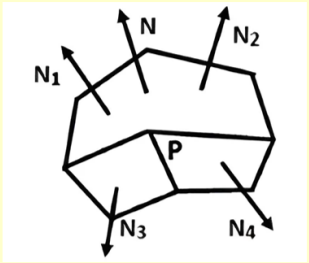
Мета дослідження

- **Мета та завдання дослідження.** Метою дослідження є підвищення точності вимірювання антропометричних параметрів людини за рахунок використанням тривимірних моделей, що дозволяють реєструвати усі деталі геометрії тіла людини.
- Відповідно до поставленої мети в бакалаврській дипломній роботі потрібно вирішити такі **задачі**:
  - виконати аналіз методу Гуро і засобів зафарбовування поверхонь тривимірних об’єктів;
  - розробити методи підвищення продуктивності рендерингу Гуро;
  - розробити алгоритм зафарбовування тривимірних моделей на основі методу Гуро;
  - розробити алгоритми рендерингу, які забезпечать високу продуктивність у процесі формування зображень за Гуро;
  - покращення процесу формування зображень за методом Гуро із забезпеченням високої продуктивності.
  - тестування розробленого програмного застосунку.
- **Об'єкт дослідження** – процес вимірювання антропометричних параметрів з використанням тривимірного моделювання.
- **Предмет дослідження** – методи та засоби вимірювання антропометричних параметрів з використанням тривимірного моделювання.
- **Методи дослідження.** У процесі дослідження застосовувалися: теорія антропометричних вимірювань; теорія алгоритмів для обробки та аналізу даних із тривимірних моделей та методи тривимірного моделювання для вимірювання антропометричних відстаней та визначення позицій точок антропометрії, комп’ютерне моделювання для аналізу та перевірки отриманих теоретичних положень.

Рисунок Г.4 – Мета та задачі дослідження

Розробка методу зафарбовування тривимірної моделі за Гуро

Значення інтенсивності для кожного полігону координуються зі значеннями інтенсивності сусідніх полігонів уздовж спільних ребер



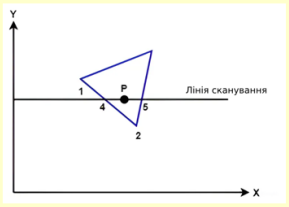
Таким чином, для довільного положення вершини V на тривимірній моделі, отримуємо одиничну нормаль до вершини, відповідно до формули:

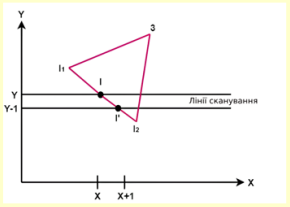
$$N_v = \frac{\sum_{k=1}^{n} N_k}{\left|\sum_{k=1}^{n} N_k\right|}$$

Отримавши вершинні нормалі певного полігону, визначимо інтенсивність у вершинах за допомогою моделі освітлення:

$$I_4 = \frac{y_4 - y_2}{y_1 - y_2} I_1 + \frac{y_1 - y_4}{y_1 - y_2} I_2$$

Отримавши формулу визначення інтенсивності у точці, необхідним кроком є визначення інтерполяції інтенсивностей кольору вздовж ребер полігону на тривимірній моделі





Після того, як ці граничні значення інтенсивності визначено для лінії сканування, внутрішня точка (наприклад, точка P) інтерполюється від граничних значень інтенсивності в точках 4 і 5 як:

$$I_P = \frac{x_5 - x_P}{x_5 - x_4} I_4 + \frac{x_P - x_4}{x_5 - x_4} I_5$$

Звідси, можна вивести формулу для обчислення інтенсивості кольору за методом Гуро на краю полігона(x, y):

$$I = \frac{y - y_2}{y_1 - y_2} I_1 + \frac{y_1 - y}{y_1 - y_2} I_2 \qquad I^1 = I + \frac{I_2 - I_1}{y_1 - y_2}$$

Рисунок Г.5 – Розробка методу зафарбовування тривимірної моделі за Гуро

Модифікації зафарбовування зображень за методом Гуро

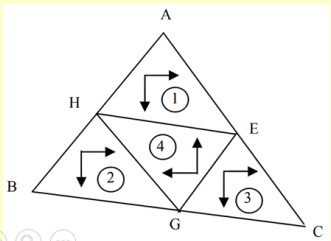
Розглянемо взаємозв'язок між змінами інтенсивності кольору вздовж країв полігонів та растрових ліній.

$$\Delta I_\Gamma = \frac{(I_A - I_C) * \Delta Y_{BC} - (I_B - I_C) * \Delta Y_{AC}}{\Delta X_{AC} * \Delta Y_{BC} - \Delta X_{BC} * \Delta Y_{AC}}$$

$$\Delta I_B = \frac{(I_B - I_C) * \Delta Y_{BC} - (I_A - I_C) * \Delta Y_{AC}}{\Delta X_{AC} * \Delta Y_{BC} - \Delta X_{BC} * \Delta Y_{AC}}$$

$$\Delta I_B = \Delta I_\Gamma + \Delta I_B$$

Отримана властивість дає змогу розпаралелити рендеринг за методом Гуро використовуючи тріангуляцію Серпінського першого порядку. Для цього початковий трикутник ділиться на чотири частини за серединами кожного ребра.



У циклі підготовки інтенсивності кольорів I_1, I_2 обчислюються для першої та наступної точок на лінії сканування. Інтенсивності I_{PB} для перших точок рядків визначаються шляхом інтерполяції інтенсивностей кольорів вздовж базових ребер, тоді як I_2 обчислюється за допомогою наступної формули:

$$I_2 = I_{PB} + \Delta I_\Gamma$$

Доведено, що в цьому випадку максимальні значення інтенсивності кольору знаходяться в точках на межах трикутника, які задовольняють рівнянню:

$$t_1 = \frac{\cos\gamma\cos\phi - \cos\beta}{(\cos\phi - 1)(\cos\beta + \cos\gamma)} \quad t_2 = \frac{\cos\gamma\cos\phi - \cos\lambda}{(\cos\phi - 1)(\cos\lambda + \cos\gamma)}$$

$$t_3 = \frac{\cos\beta\cos\vartheta - \cos\lambda}{(\cos\vartheta - 1)(\cos\beta + \cos\lambda)}$$

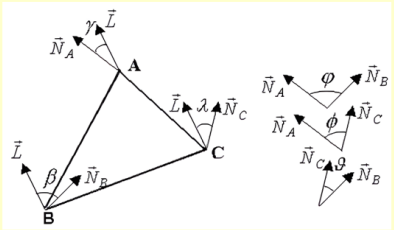


Рисунок Г.6 – Модифікація методу зафарбовування зображень за Гуро

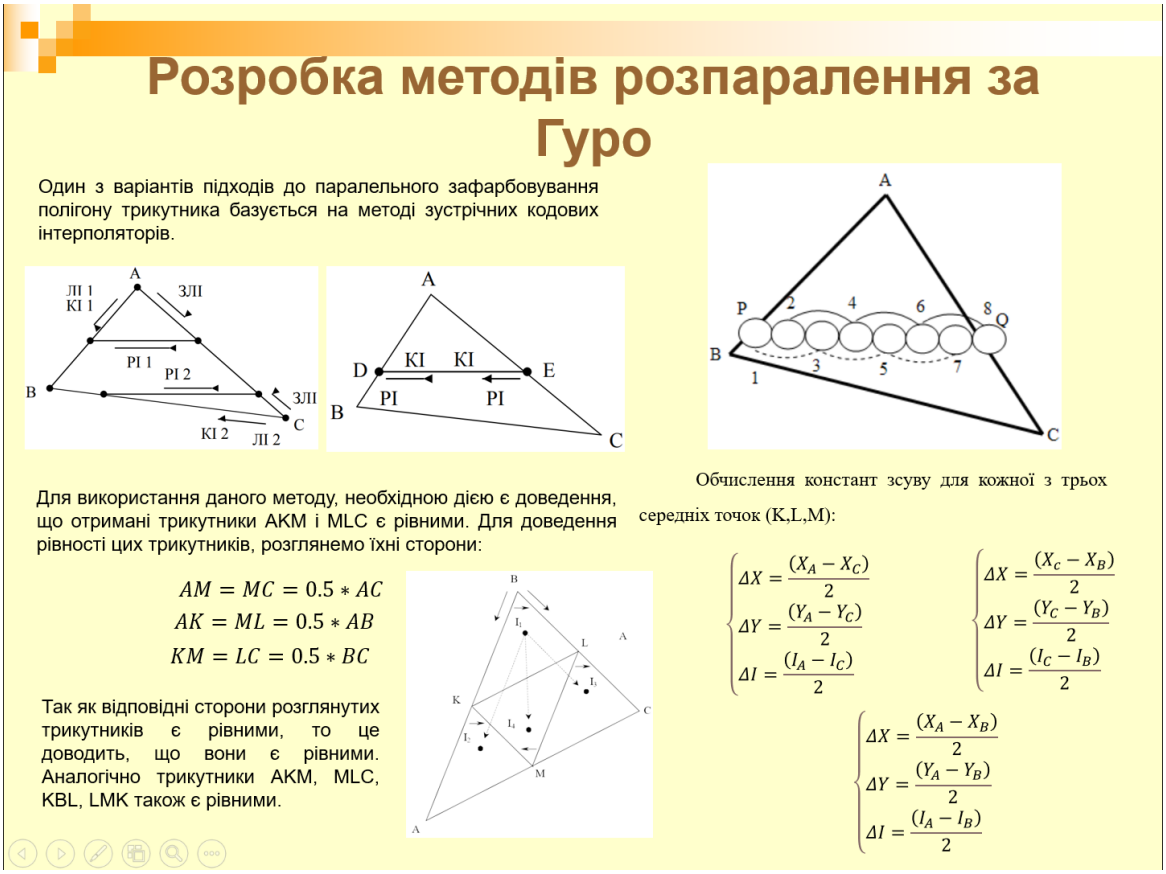


Рисунок Г.7 – Розробка методу розпаралелення рендерингу за Гуро

Рисунок Г.8– Граф-схеми алгоритмів програми

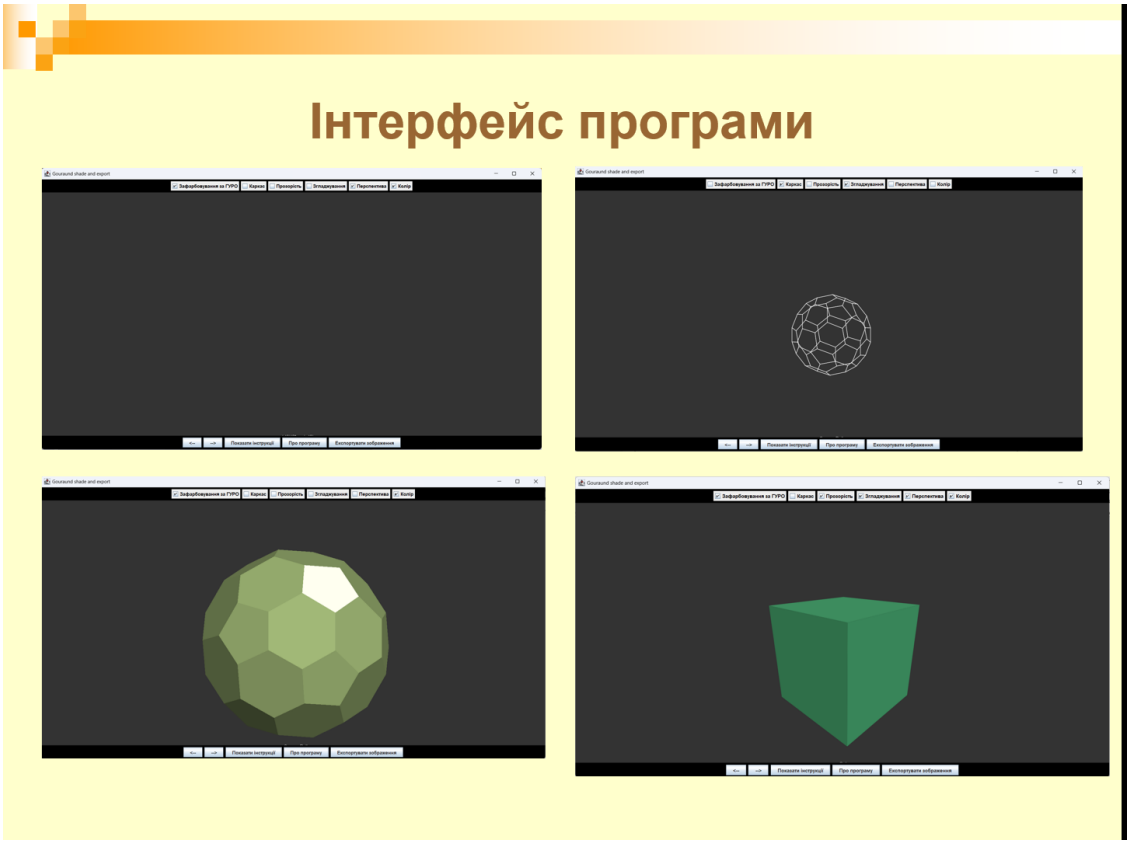


Рисунок Г.9 – Інтерфейс програмного забезпечення

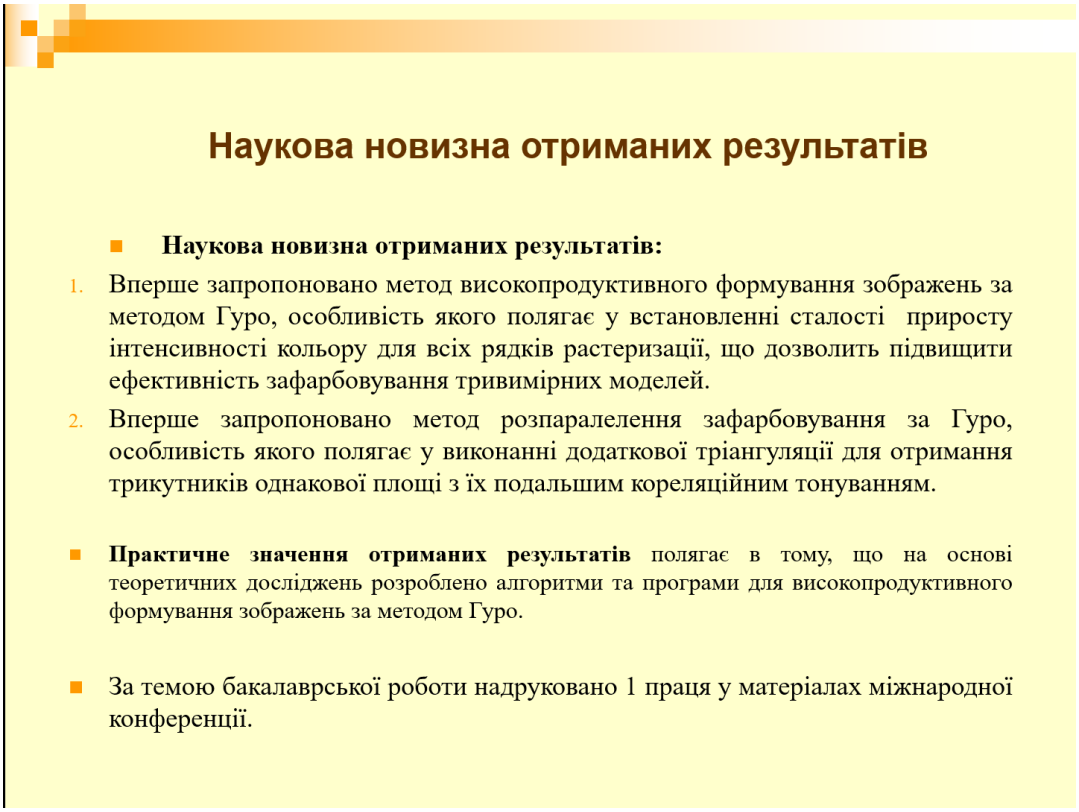


Рисунок Г.10 – Наукова новизна дослідження

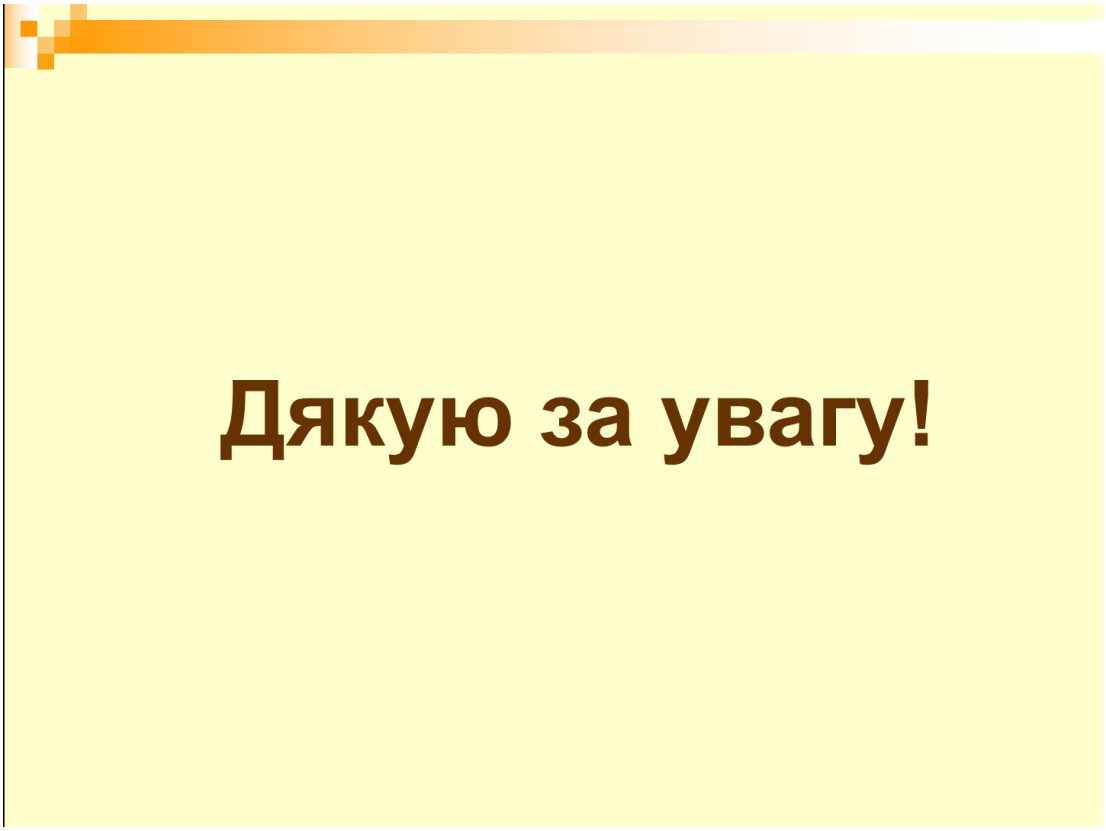


Рисунок Г.11 – Закінчення презентації