Міністерство освіти і науки України
Луцький національний технічний університет
Факультет комп'ютерних та інформаційних технологій
Кафедра інженерії програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА ЗА СТУПЕНЕМ ВИЩОЇ ОСВІТИ «БАКАЛАВР»

СТВОРЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ КОМП'ЮТЕРНОЇ 3D ГРАФІКИ

CREATION OF SOFTWARE FOR COMPUTER 3D GRAPHICS

спеціальність 121 Інженерія програмного забезпечення
(шифр і назва спеціальності)

освітня програма «Інженерія програмного забезпечення»
(назва освітньої програми)

Виконав: здобувач вищої освіти
Групи ІПЗ-42
Лопух Владислав Володимирович
_____
(підпис)

Керівник:
к.т.н., доцент
Сичук Віктор Анатолійович
_____
(підпис)

Кваліфікаційну роботу
допущено до захисту
«8» червня 2024 р.
к.т.н., доцент
Гарант освітньої програми:
Ліщина Наталія Миколаївна
_____
(підпис)

Луцьк – 2024 рок

ЛУЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Факультет *комп'ютерних та інформаційних технологій*
Кафедра *інженерії програмного забезпечення*
Ступінь вищої освіти: *бакалавр*
Галузь знань: *12 Інформаційні технології*
Спеціальність: *121 Інженерія програмного забезпечення*
Освітня програма: *«Інженерія програмного забезпечення»*

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Н. Лінкіна
«2» 02 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧУ ВИЩОЇ ОСВІТИ

Лопуху Владиславу Володимировичу
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи: *Створення програмного забезпечення для комп'ютерної 3D графіки*

Керівник роботи: *к.т.н., доцент, Сичук Віктор Анатолійович*
затверджені наказом закладу вищої освіти від «*30*» *грудня 2023 р.* № *477/01-02*
2. Строк подання здобувачем вищої освіти кваліфікаційної роботи *«5» червня 2024 р.*
3. Вихідні дані до роботи: *технічне та програмне забезпечення ЕОМ, вимоги до розробки програмного забезпечення, ергономічні вимоги до функціонування програмного засобу.*

4. Зміст розрахунково-пояснювальної записки (перелік питань, що потрібно розробити):
Аналіз сучасного стану проблеми, існуючих методів і засобів її розв'язання, аналіз, визначення вимог до розроблюваного програмного забезпечення та проектування програмного забезпечення, опис функціонального наповнення об'єкта проектування, практична реалізація об'єкта проектування.

5. Перелік графічного матеріалу: *9 рисунків, 2 таблиці, 1 формула.*

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис	
		завдання видав	завдання прийняв
Аналіз предметної області	*Сичук В. А.*		
Специфікації вимог до розробленої системи	*Сичук В. А.*		
Розробка об'єкта проєктування	*Сичук В. А.*		
Нормоконтроль	*Повстяна Ю. С.*		
Гарант ОП	*Ліщина Н. М.*		
Показник запозичень тексту		27 %	
Академічна доброчесність	*Ящук А. А.*		

7. Дата видачі завдання *«2» лютого 2024 р.*

КАЛЕНДАРНИЙ ПЛАН

№ З/п	Назва етапів кваліфікаційної роботи бакалавра	Строк виконання етапів роботи	Примітка
1	*Огляд літературних джерел по темі кваліфікаційної роботи бакалавра*	*07.02.2024 р.*	
2	*Аналіз проблеми розробки та впровадження об'єкту проектування*	*27.02.2024 р.*	
3	*Обґрунтування вибору шляхів, технологій (алгоритмів) і засобів вирішення поставленого завдання*	*12.03.2024 р.*	
4	*Розробка функціонально-структурної схеми роботи об'єкта проектування та проектування бази даних*	*17.04.2024 р.*	
5	*Практична реалізація об'єкта проектування та розробка бази даних*	*18.05.2024 р.*	
6	*Тестування та налагодження об'єкта проектування*	*01.06.2024 р.*	
7	*Здача чистового варіанту кваліфікаційної роботи бакалавра на кафедру*	*05.06.2024 р.*	

Здобувач вищої освіти _____ (підпис) *Лопух В. В.* (прізвище, ініціали)

Керівник кваліфікаційної роботи _____ (підпис) *Сичук В. А.* (прізвище, ініціали)

Міністерство освіти і науки України

Луцький національний технічний університет

Рецензія
на кваліфікаційну роботу

Здобувач вищої освіти: Лопух Владислав Володимирович

Тема роботи: Створення програмного забезпечення для комп'ютерної 3D графіки

Коротка характеристика кваліфікаційної роботи: у роботі проведено аналіз предметної області технології програмування 3D графіки та здійснено огляд інструментів і засобів розробки. Розроблено програмне забезпечення для роботи з 3D графікою та візуалізації хмари точок.

Самостійні розробки і пропозиції автора: в роботі представлено програмне забезпечення, яке працює з 3D графікою та конвертує масив вихідних координат у тривимірну хмару точок. Реалізовано алгоритм конвертації координат однієї осі у окремі тривимірні вершини

Практичне значення роботи: спрямоване на тривимірну візуалізацію вихідних даних приладів, які сканують об'єкт і видають масив з координатами

Недоліки: недоліків не виявлено

Загальний висновок: вважаю, що кваліфікаційна робота відповідає вимогам до випускних робіт і заслуговує позитивної оцінки

Рецензент Поліщук Микола Миколайович, к.т.н., доц. кафедри
(прізвище, ім'я, по-батькові, посада)

комп'ютерної інженерії Луцького національного технічного університету

Рецензент кваліфікаційної роботи: _____
(підпис)

«11» червня 2024 р.

Міністерство освіти і науки України
Луцький національний технічний університет
Факультет комп'ютерних та інформаційних технологій
Кафедра інженерії програмного забезпечення

ВІДГУК
КЕРІВНИКА НА КВАЛІФІКАЦІЙНУ РОБОТУ

Здобувач вищої освіти *Лопух Владислав Володимирович*
(прізвище, ім'я, по батькові)
група *ІПЗ-42*

Тема кваліфікаційної роботи:
Створення програмного забезпечення для комп'ютерної 3D графіки

Керівник *Сичук Віктор Анатолійович*

Актуальність теми
Актуальність зумовлена невеликим вибором програмних засобів 3D візуалізації для задачі візуалізації хмари точок.

Об'єкт дослідження
Програмне забезпечення для роботи з тривимірним середовищем.

Характеристика теоретичного рівня, наявності самостійних розробок і практичної значимості роботи:
Автор розробив програмний інструмент для обробки тривимірної графіки, який здійснює перетворення вхідного масиву координат у тривимірну хмару точок.

Зауваження та недоліки: *Суттєвих недоліків в роботі не виявлено.*

Загальний висновок:
Кваліфікаційна робота бакалавра виконана на високому рівні. Робота виконана відповідно до вимог, логічно й послідовно описує хід виконання поставленого завдання. Пояснювальна записка відповідає стандартам до її оформлення.

Керівник _____ (***Сичук В. А.***)
(підпис) (прізвище, ім'я, по батькові)

«11» червня 2024 р.

АНОТАЦІЯ

Лопух В. В. Створення програмного забезпечення для комп’ютерної 3D графіки. Рукопис.

Кваліфікаційна робота бакалавра ОП «Інженерія програмного забезпечення». Луцький національний технічний університет. Луцьк, 2024.

Кваліфікаційна робота бакалавра складається зі вступу, трьох розділів, висновків і пропозицій, списку використаних джерел та додатків.

У першому розділі здійснено аналіз сучасного стану проблеми розробки комп’ютерних ігор і сформульовано завдання. В другому розділі визначено вимоги до розроблюваної системи, а також засоби, методи і алгоритми розробки. У третьому розділі розроблено та протестовано програмне забезпечення для комп’ютерної 3D графіки. У висновках узагальнено інформацію, відображену у попередніх частинах. Результати розробки демонструють можливості розробленого програмного забезпечення для комп’ютерної 3D графіки.

Ключові слова: тривимірне середовище, хмара точок, Vulkan API, OpenGL.

ABSTRACT

Lopukh V. V. Creation of Software for Computer 3D Graphics. Manuscript.

Bachelor's qualifying thesis of the educational program "Software Engineering". Lutsk National Technical University. Lutsk, 2024.

The bachelor's qualifying thesis consists of an introduction, three sections, conclusions and proposals, a list of used sources and annexes.

In the first chapter, an analysis of the current state of the problem of developing computer games was carried out and the task was formulated. The second section defines the requirements for the developed system, as well as means, methods and development algorithms. In the third chapter, software for computer 3D graphics was developed and tested. The conclusions summarize the information presented in the previous parts. The development results demonstrate the possibilities of the developed software for computer 3D graphics.

Keywords: 3D environment, point cloud, Vulkan API, OpenGL.

ЗМІСТ

ВСТУП 7
РОЗДІЛ 1 АНАЛІЗ І ПОСТАНОВКА ЗАВДАНЬ НА КВАЛІФІКАЦІЙНУ РОБОТУ 9
1.1 Аналіз сучасного стану проблеми 9
1.2 Постановка завдання на кваліфікаційну роботу бакалавра 14
Висновки до розділу 1 15
РОЗДІЛ 2 СПЕЦИФІКАЦІЯ ВИМОГ ДО РОЗРОБЛЮВАНОЇ СИСТЕМИ 16
2.1 Аналіз, визначення вимог до розроблюваного програмного забезпечення та проектування програмного забезпечення 16
2.2 Вибір засобів, методів і алгоритмів вирішення поставленого завдання . 20
Висновки до розділу 2 27
РОЗДІЛ 3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ КОМП'ЮТЕРНОЇ 3D ГРАФІКИ 28
3.1 Практична реалізація об'єкта проектування 28
3.2 Тестування та налагодження інформаційно-комп'ютерної системи 37
3.3 Розробка інсталяційного пакета 39
3.4 Розробка документації для програмного продукту 39
3.5 Економічне обґрунтування розробки 40
Висновки до розділу 3 43
ВИСНОВКИ 44
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 46

ВСТУП

Актуальність теми. Програмне забезпечення для роботи з комп'ютерною 3D графікою застосовується у багатьох галузях науки, економічного та суспільного рівнях задач. Такі програми користуються широкою популярністю у розважальній сфері, де вони використовуються, у переважній більшості, різного виду проектах. Проте, використання такої технології розвивається і в науково-дослідницькій галузі, де з-за допомогою візуалізації пакету даних, отриманих через різного роду датчики, отримуються наглядні результати у тривимірному просторі. Головною проблемою програмних забезпечень, які використовуються у науковій галузі є те, що вони мають невелику підтримку аудиторії, що впливає на швидкість розробки таких програм або навіть відсутність оновлень. Через відсутність перспектив для великої капіталізації, команди розробників програмного забезпечення та інвестори не вбачають сенсу в реалізації такого роду застосунків. Тому, інженерам, науковцям та дослідникам, для реалізації необхідної для них, специфічної, задачі, слід шукати окремих програмістів, які могли б розробити потрібну програму, провести пошук існуючих рішень або пробувати самому реалізовувати існуючу проблему.

Більшість наявних рішень, в якості елемента розробки програмного забезпечення, використовують OpenGL, який, фактично, є застарілим. На заміну OpenGL прийшов Vulkan API, який вимагає від розробника більших знань у сфері графічного програмування, проте він надає ширші можливості налаштувань різних процесів, за рахунок чого, підвищується продуктивність та ефективність програми написаної з використанням Vulkan. Окрім цього, новий прикладний програмний інтерфейс підтримує кросплатформність, що розширює потенційне коло користувачів та дає можливість деяким інженерам працювати у нативній операційній системі. Можливості кросплатформної роботи описували К. Йоннідіс та А. М. Бутсі у своїй статті [1].

Метою роботи є розробка програмного забезпечення для роботи з комп’ютерною 3D графікою, яке здатне візуалізувати хмару точок та мати конкурентні параметри продуктивності.

Об’єкт дослідження – програмне забезпечення для роботи з тривимірним середовищем.

Предмет дослідження – можливості використання прикладного програмного інтерфейсу Vulkan.

Завдання, які необхідно виконати:

– дослідити предметну область використання;

– визначити можливі варіанти реалізації програмного забезпечення;

– розробити функціональне програмне забезпечення для роботи з 3D графікою;

– проаналізувати можливості та перспективи розробленого програмного забезпечення;

– розробити інсталяційний пакет для програмного забезпечення;

– розробити документацію до програмного забезпечення;

– навести економічне обґрунтування розробки.

Розробку було представлено на науковому івенті «Іноваційні технології ЛНТУ».

РОЗДІЛ 1
АНАЛІЗ І ПОСТАНОВКА ЗАВДАНЬ НА КВАЛІФІКАЦІЙНУ РОБОТУ

1.1 Аналіз сучасного стану проблеми

Сучасні методи зображення реальних об’єктів використовують, зазвичай, відомі системи автоматизованого проектування і розрахунку. Проте, всі вони представляють із себе набір інструментів, з-за допомогою якого, вручну, можна втілювати в життя різні технічні новації. Для створення автоматизації створення 3Д візуалізації, інженери різних галузей стикаються з наступними варіантами розвитку подій, щодо вирішення їхньої проблеми: найм фахівців у сфері програмування комп’ютерної 3Д графіки, вивчення програмування для відповідного завдання або спроба налаштування автоматизації, ресурсами використовуваного САПР та інакших, незалежних від САПР, програм, які можуть автоматизувати різні дії у вашому, комп’ютерному робочому середовищі.

Проблема створення 3Д об’єкту із примітивного набору даних, залежить від можливості профінансувати таке програмне забезпечення, тим чи іншим науковцем, інженером, дослідником або будь-якою юридичною особою. Чим більше фінансування – тим більше можна імплементувати функціоналу у свій фізичний, технологічний виріб.

Основним типом даних, які можуть повертати фізичні датчики, без попереднього оброблення цих даних – це текстові файли, з ними зручно працювати та вони можуть наглядно показати результат роботи пристрою. Саме текстовий файл із координатами має опрацьовувати розроблювана програма, створюючи на основі зчитаних координат 3Д модель.

Набір даних, які являють собою двовимірну або тривимірну матрицю із координат і які містяться у файлах текстового формату можна назвати хмарою точок. Одиницею хмари точок, можна вважати вершину (Vertex) із координатами у тривимірному або двовимірному просторі.

Із доступних програмних застосунків, які є на ринку, такий функціонал можуть надати наступні: SolidWorks, CloudCompare, AutoCAD, MeshLab. Кожен з цих варіантів, має свої унікальні аспекти використання, проте об’єднує їх – фактор ручного налаштування імпортованого файлу і виставлення середовища для візуалізації. Проте окрім, ринкового рішення, існують і програми, написані ентузіастами чи науковцями, які пов’язані із візуалізацією, проте, функцію зображення моделі за рахунок текстових координат, вони не організовують, що не дає потрібної можливості вирішення існуючої проблеми. Зазвичай, такі програмні рішення фокусуються на вирішенні однієї чи декількох, індивідуальних, проблем, з якими стикнувся інженер-розробник [2], [3].

Що б отримати повноцінний висновок, слід провести аналіз існуючих варіантів, визначити їх переваги та недоліки, а також врахувати специфіку існуючої проблеми та її можливі нюанси.

SolidWorks – оскільки цей програмний застосунок відноситься до систем автоматизованого проектування і розрахунку, то він більш пристосований до задач параметричного моделювання, хоча SolidWorks не спеціалізується на обробці текстових файлів із координатами, як інші програми, проте він має достатньо функцій для інженерів, які працюють з такими даними. Він підтримує наступні формати, з яких він може зчитувати координати: .asc, .txt, .pts. Після створення 3Д моделі, можна використовувати базовий функціонал усього застосунку. Попри його можливість підвантажувати файли з інформацією, він не може напряму взаємодіяти із фізичними пристроями, які такого роду файли, генерують, також ця програма може використовуватись тільки після купівлі ліцензії, яка вітчизняним інженерам не завжди доступна.

CloudCompare – безкоштовний інструмент для створення 3Д-моделі із хмари точок, містить у собі велику кількість різноманітних фільтрів та алгоритмів, з-за допомогою яких, можна підготувати чи оптимізувати, створену модель до подальшої обробки. Застосунок надає можливість імпортувати дані з різних форматів файлів, серед яких: .txt, .asc, .neu, .xyz, .pts, .csv, .obj, .stl і багато інших;

також після завершення операцій над вашим об’єктом, можна експортувати модель у формати, які зчитуються іншими програмами. Результат створення візуалізації хмари точок, із згенерованого фізичним пристроєм набором даних, у тривимірному просторі, зображено на рисунку 1.1.

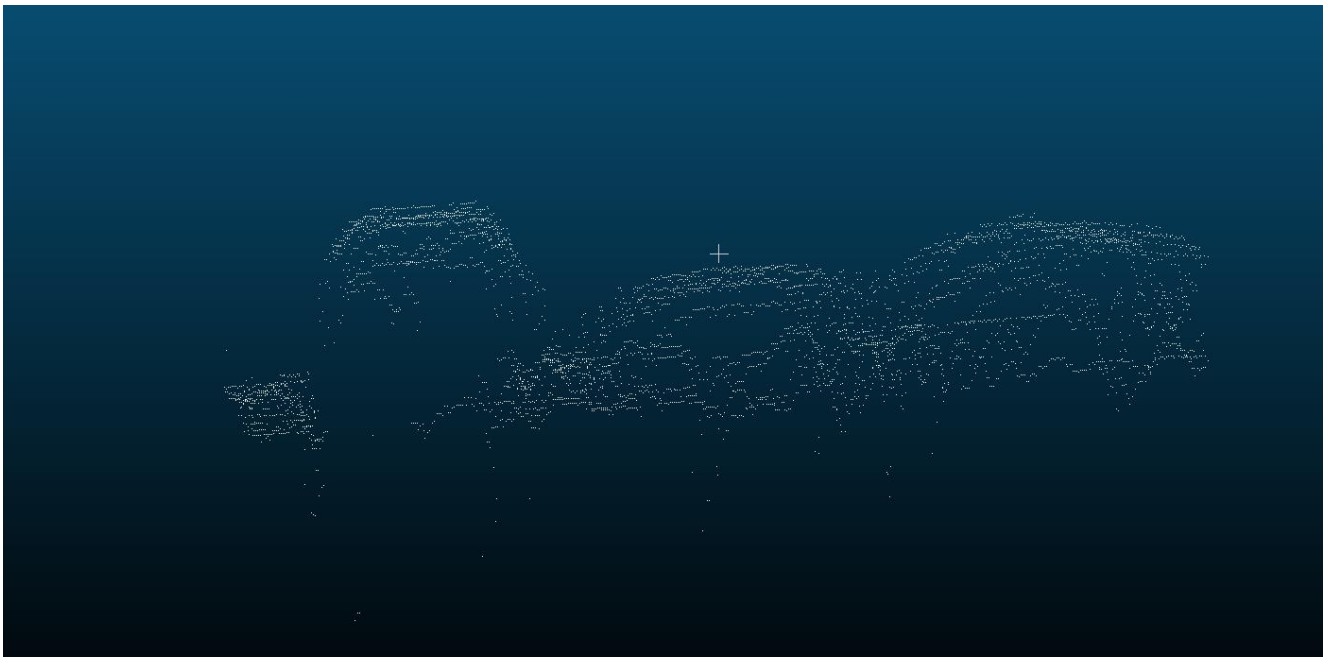

Рисунок 1.1 – Візуалізація хмари точок у програмі CloudCompare

Крім того, CloudCompare надає інструменти для ручного редагування моделей, що дає можливість користувача вносити корективи. За рахунок можливості вільного використання програми як у навчальних, так і в комерційних цілях, його можна визнати хорошим варіантом для дослідників та інженерів, які не мають достатнього фінансування на свої розробки.

AutoCAD – так як і SolidWorks, AutoCAD є системою автоматизованого проектування та розрахунку, проте по функціональних можливостях, для роботи із хмарою точок, він програє своєму аналогу. Для імпорту даних хмари використовується тільки 2 формати: PCG та ISD, які є рідними форматами програми. Також програма надає доступ до базових маніпуляцій із імпортованими даними, а саме: масштабування та обертання, окрім базових маніпуляцій, є функціонал перетворення хмари точок у повноцінну 3Д модель, використовуючи

поєднання точок методом тріангуляції або створення NURBS кривих. Висока ціна та менший функціонал при роботі з хмарою точок роблять AutoCAD менш привабливим вибором для інженерів, які фокусуються на роботі із набором точок.

MeshLab – є спеціалізованим програмним забезпеченням для роботи з хмарою точок, по своїм можливостям та характеристикам дуже схожий до CloudCompare. Обидві програми підтримують різноманітні формати файлів хмар точок та мають схожий інструментарій для взаємодії та обробки даних. Проте MeshLab має переваги у сфері створення 3Д моделей, оскільки пропонує ширший спектр алгоритмів для перерахування точок у повноцінний меш, із подальшим його редагуванням, приклад створеного мешу із хмари точок можна побачити на рисунку 1.2.

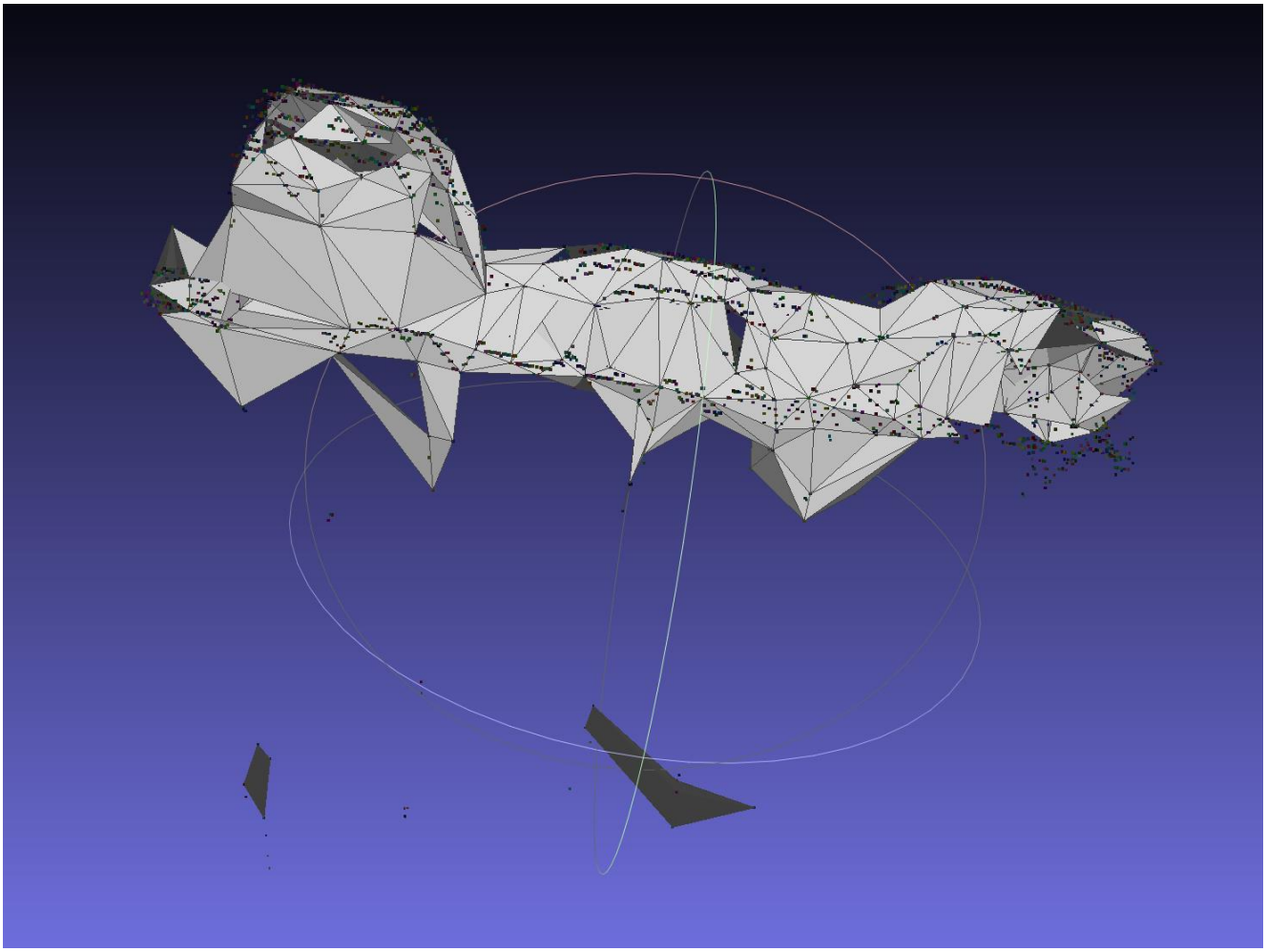

Рисунок 1.2 – Створена 3Д модель із хмари точок у застосунку MeshLab

Попри перевагу у створенні 3Д моделей, MashLab не має повноцінного функціоналу для аналізу та порівняння хмари точок, який є у CloudCompare. Обидві програми є безкоштовними, що дає значну перевагу перед традиційними САПР AutoCAD та SolidWorks, тому MeshLab є хорошим рішенням для індивідуальних науково-дослідницьких проектів.

Загалом задачі з якими стикаються науковці, в яких можна використати технологію візуалізації – це представлення статистичних даних у тривимірному середовищі, візуалізація даних різноманітних датчиків та створення симуляцій і моделей тих чи інших структур. Часто технологію використовують у таких наукових галузях як:

– астрономія. Зазвичай отриманий масив даних, які отримують науковці із телескопів, датчиків та приймачів – це необроблений масив із хмари точок, який слід фільтрувати, оскільки, отримані дані часто мають шум та артефакти, які можуть бути спричинені навколишнім забрудненням або неполадками роботи самого пристрою. Використовують візуалізацію і для різного роду симуляцій, серед них: еволюції галактик, зіткнення космічних об’єктів, представлення орбіт об’єктів;

– медицина. Одним із явних прикладом використання цієї технології є візуалізація даних магнітно-резонансної томографії (МРТ) та комп’ютерної томографії (КТ), отриманий масив даних із цих приладів обробляється в комп’ютері лікаря, після чого створюється картина чи модель структури організму людини;

– геологія. З-за допомогою акустичних приладів виявляють потенційні родовища корисних копалин, які можна зобразити на тривимірній карті, після чого розробити план видобутку ресурсів.

Проаналізувавши сферу розробки програмного середовища для комп’ютерної 3Д графіки та наукового використання технології візуалізації хмари точок, можна констатувати, що технологія використовується у специфічних задачах і програмне забезпечення, зазвичай, напряму пов’язане з пристроєм, який

вихідні дані генерує, а із існуючих засобів трансформації, чудовими рішеннями будуть тільки 2 програми: CloudCompare та MeshLab, вибір окремої з них, залежить від науковця, який має індивідуальну задачу для виконання.

Нестача швидкодії, в наявних програмах, може стате критичною проблемою при спробі імпортування великого масиву даних, особливо на апаратних засобах із обмеженими ресурсами, тому актуальність створення оптимізованого та ефективного програмного забезпечення для комп'ютерної 3Д графіки з високою продуктивністю зберігається [4]. Окрім цього, сучасною вимогою до програмного забезпечення є кросплатформність, оскільки це розширює аудиторію потенційних користувачів та збільшує практичну цінність програми.

1.2 Постановка завдання на кваліфікаційну роботу бакалавра

Актуальність дослідження зумовлена невеликим вибором програмних засобів 3Д візуалізації, для такої задачі як: візуалізація хмари точок. А також вивчення швидкодії роботи програми, порівнюючи її з існуючими засобами. Тому основними завданнями на кваліфікаційну роботу бакалавра є:

– проведення аналізу існуючих засобів та методів візуалізації хмари точок в 3Д-графіці, їх переваги та недоліки;

– розробка та реалізація програмного забезпечення для візуалізації хмари точок у тривимірному середовищі;

– створення алгоритмів обробки вихідних даних для перетворення у хмару точок;

– проведення тестування розробленого програмного забезпечення;

– аналіз отриманих результатів та формулювання висновків щодо ефективності та перспектив використання розробленого програмного забезпечення.

Висновки до розділу 1

Виконавши аналіз предметної області, було розглянуто проблематику використання існуючих програмних рішень та їх функціональність, методи створення візуалізації у тривимірному середовищі хмар точок, а також обґрунтовано доцільність створення нової системи.

Було встановлено, що важливим аспектом для конкретної задачі є вибір оптимального програмного рішення, опираючись на специфіку даних та наявні ресурси. Також були проаналізовані такі основні рішення в цій галузі як: AutoCAD, MeshLab, CloudCompare та SolidWorks.

РОЗДІЛ 2
СПЕЦИФІКАЦІЯ ВИМОГ ДО РОЗРОБЛЮВАНОЇ СИСТЕМИ

2.1 Аналіз, визначення вимог до розроблюваного програмного забезпечення та проектування програмного забезпечення

Програма, яка має розроблятися – це програмне забезпечення, яке працює із 3Д графікою, основним завданням якого є перенесення вихідних даних хмари точок у тривимірне середовище, для наглядної ілюстрації того, що цей масив даних із себе представляє.

У контексті розробки спеціалізованого програмного забезпечення для візуалізації хмари точок, слід використати архітектуру, яка базується на графічному інтерфейсі користувача для відображення тривимірного середовища, основним елементом інтерфейсу якого, є вікно, яке має слугувати контейнером для візуалізації 3Д сцени та рендерингу хмари точок.

Вихідними даними для хмари точок – є текстові файли з масивом координат, тому програма має передбачати завантаження файлу і подальшу обробку цього файлу. Це вимагає реалізації механізмів парсингу, алгоритмів обробки вихідних даних та подальше їх конвертування.

Оскільки базового макету інтерфейсу не планується розроблятися, то для інтерактивної взаємодії з екраном інтерфейсу, слід додати базові маніпуляції з камерою, які будуть взаємодіяти із натисканнями клавіш користувача, серед них: обертання, віддалення та приближення, збільшення кута обзору.

Завдання, яке має вирішувати програмне забезпечення, відноситься, скоріше, до науково-дослідних цілей, чим до комерційних, тому цільовою аудиторією такого застосунку можуть бути: інженери, геологи, науковці в медичній, фізичній та астрономічній сферах, статисти. Відповідно областю застосування будуть такі галузі як: медицина, інженерія та наукові дослідження.

Для забезпечення повноцінної роботи розроблюваної програми, слід визначити ряд функціональних та нефункціональних вимог, завдяки яким можна

визначити потрібні вимоги та широкий спектр характеристик до майбутнього програмного забезпечення. До функціональних вимог відносять такі характеристики, які описують що система повинна роботи, тобто функції та поведінку. Такі вимоги фокусуються на взаємодії користувача із системою. До опису функціональних вимог відносять: вхідні та вихідні дані, інтерфейси, опис функцій та завдань, обробка помилок та нештатних ситуацій.

Функціональні вимоги до розроблюваного програмного забезпечення для роботи з комп'ютерною 3Д графікою та візуалізацією хмари точок:

– початковим етапом для програми є завантаження потрібних даних, та подальша взаємодія цих даних із програмою. Тому, до функціональних вимог слід віднести підтримку завантаження файлу із вихідними даними у програму та подальшу конвертацію отриманих даних у потрібний формат;

– інтерактивна взаємодія з камерою, для більш наглядного перегляду отриманого результату. До взаємодії з камерою відносяться такі базові операції керування камерою як: обертання, переміщення та масштабування. З-за допомогою цих речей можна зручно досліджувати 3Д сцену, забезпечуючи краще вивчення отриманого результату.

Нефункціональні вимоги визначають те, як система повинна працювати, її якості, характеристики та обмеження системи, а не її конкретні функції чи методи. Сутність таких вимог полягає у якісній взаємодії користувача з програмою, можливості до масштабування та обслуговування. До нефункціональних вимог розроблюваного програмного забезпечення слід віднести:

– висока продуктивність. Вимога визначає наскільки швидко та ефективно програма повинна працювати;

– надійність. Стабільна працездатність програми без збоїв та багів;

– сумісність. Можливість програми працювати на різних девайсах та різних операційних системах;

– масштабованість. Програма має мати можливість до покращень та простого процесу імплементації нововведень.

Для ефективної реалізації функціональних вимог, визначених раніше, було розроблено UML моделі для кожної з вимог, завдяки створеним моделям, можна простіше зрозуміти майбутні задачі та ефективніше розробляти програмне забезпечення.

Перша UML модель складається із 4 класів, які описують процес завантаження файлу із вихідними даними до програмного застосунку, модель зображено на рисунку 2.1.

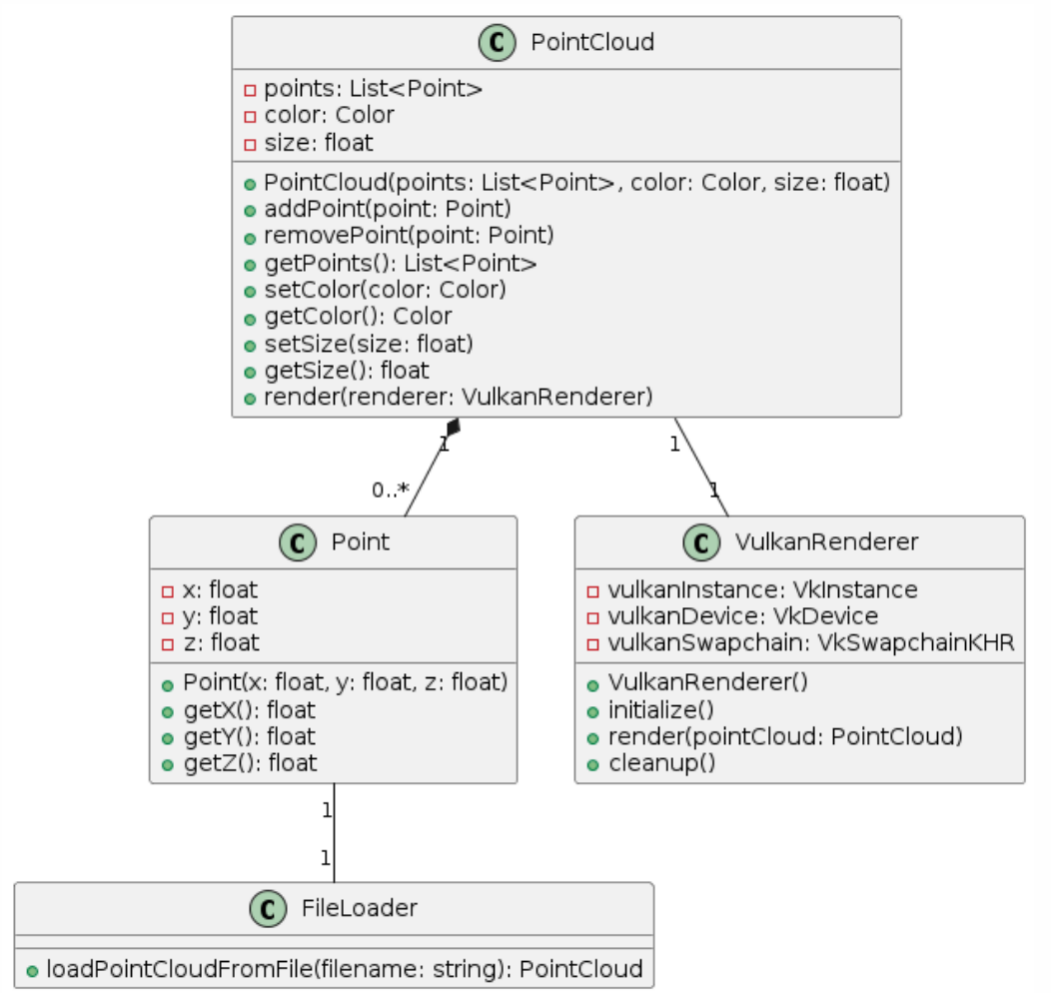

Рисунок 2.1 – UML модель класу завантаження файлу із вихідними даними

Головним елементом моделі є клас PointCloud, який являє собою хмару точок, цей клас містить список об’єктів класу Point, який описує окремо взяту вершину. Для завантаження файлу, використовується клас FileLoader, який зчитує дані із файлу та конвертує їх у окрему вершину. Оскільки програма буде

використовувати низькорівневий API Vulkan, тому слід реалізовувати такі класи ініціалізації як VulkanRenderer, окрім початкової ініціалізації цей клас відповідає за рендер сцени і отриманих вершин.

Наступна UML модель класів відповідає за організацію роботи камери у програмі та взаємодію користувача із базовими маніпуляціями з камерою, для цього було створено 4 класи, які зображені на рисунку 2.2.

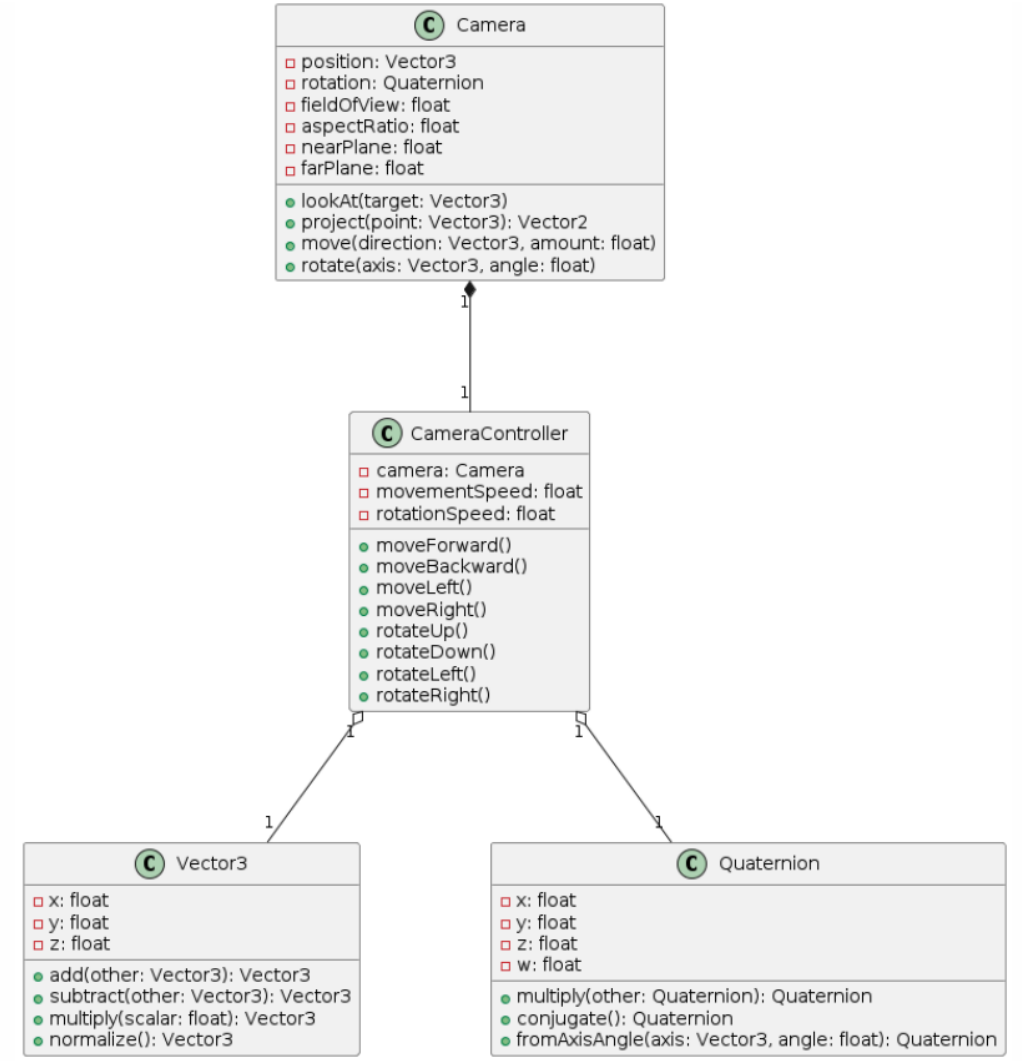

Рисунок 2.2 – UML діаграма класу, яка описує роботу з камерою

Центральним класом моделі є Camera, який інкапсулює поведінку стан та поведінку віртуальної камери, він містить атрибути, які описують різні параметри камери. Для забезпечення роботи з камерою, розроблений клас CameraController,

який надає можливість взаємодіяти з користувачем, цей клас залежить від двох інших класів: Vector3 та Quaternion. Vector3 відповідає за опис тривимірного вектора, який використовується для збереження позиції камери та визначення напряму переміщення. Клас Quaternion є кватерніоном, який відповідає за орієнтацію камери у просторі.

Верифікація вимог програми є невід'ємним етапом розробки будь-якої програми, оскільки вона визначає відповідність фінального результату згідно з початковим плануванням системи. У контексті розроблюваного проекту для тестування та оптимізації програми, слід виділити два етапи, до першого етапу відноситься тестування функціональних характеристик, до другого – тестування нефункціональних можливостей програмного забезпечення. Ще одним важливим аспектом тестування, є перевірка коректності відображення сцени та вершин, тобто тестування рендерингу та візуалізації тривимірного середовища. Не менш важливим етапом є тестування нефункціональних вимог. Слід перевірити функціонал на зручність роботи та переглянути оптимізацію програми.

2.2 Вибір засобів, методів і алгоритмів вирішення поставленого завдання

Графічне програмування охоплює широкий спектр технологій та засобів для його розробки. Для реалізації програми, яка має мати хоч якийсь функціонал в 3Д сфері, слід мати знання не тільки в програмній сфері, а і у математичній, оскільки графіка тісно пов'язана із лінійною алгеброю та геометрією. Графічне програмування використовується у багатьох сферах життя і охоплює як двовимірне, так і тривимірне середовище. Окрім базового програмування об'єктів графіки, в цю сферу почав входити штучний інтелект, з-за допомогою якого можна автоматизувати купу задач, які пов'язані з аналізом середовища у реальному житті.

Для програмування графіки, використовують різні мови програмування, кожна з яких має свої, певні, переваги та недоліки. Те яку мову програмування слід

обрати для графічного програмування, залежить напряму від ваших потреб чи задач. Проте найпоширенішими варіантами мов програмування є:

– C та C++. Наразі є стандартом у графічному програмуванні, з-за рахунок високооптимізованого коду, який забезпечує високу продуктивність роботи програми, його часто використовують для цієї задачі. Окрім цього, вони забезпечують низькорівневий контроль над пам’яттю та апаратним забезпеченням, що також є частиною оптимізації;

– C#. Головною перевагою цієї мови є вищий рівень абстракції та інтеграція з .NET, яка розширює функціонал мови. Окрім цього мова підтримує інтероперабельність з C++ та DirectX, тобто код на мові C# буде працювати з кодом C++. Проте, зазвичай мову використовують для розробки різних ігор у рушії для розробки ігор Unity, а як окремий варіант для розробки графічного програмування його не використовують;

– Python. Мова має широкий спектр вибору бібліотек на всі потреби у програмуванні, попри свою простоту, мова не виділяється швидкодією та оптимізацією, що обмежує програмістів у цій сфері, оскільки для великих проектів для роботи із 3Д графікою, потрібна хороша оптимізація проекту;

– JavaScript/WebGL. Мова використовуються для створення тривимірної графіки у браузерах, оскільки стандартні мови програмування не підтримують такого функціоналу, то для такої задачі була створена бібліотека WebGL мови програмування JavaScript.

Для розробки програмного забезпечення була обрана мова C++, оскільки вона оптимізована та має добре зарекомендовані прикладні програмні інтерфейси для роботи і з тривимірною графікою, а саме: OpenGL,Vulkan, DirectX та Metal.

Слід проаналізувати кожен із варіантів, для того щоб вибрати найбільш підходящий, враховуючи особливості розроблюваного проекту, вимоги до його продуктивності та доступні, на розробку, ресурси.

– OpenGL. Відкритий графічний програмний інтерфейс для роботи з 2Д та 3Д графікою, зарекомендоване рішення яке, попри завершення підтримки,

використовується і досі. Одним із головних переваг інтерфейсу є його кросплатформність, завдяки цьому його можна використовувати у різних операційних системах, проте тільки комп’ютерних. Він надає низькорівневий доступ до графічного обладнання, що позитивно відображається на продуктивності роботи програми, проте це потребує більше часу та знань від програміста. OpenGL використовує модель графічного конвеєра рендерингу, ця модель являє собою ступінчатий конвеєр, де, на кожному етапі, графічні дані обробляються і повертають графічну картину даних;

– Vulkan. Є еволюцією OpenGL під сучасні реалії та потреби, проте фактично вони відрізняються підходами до рендерингу та мають певні відмінності у архітектурі. Vulkan також є низькорівневим API і він надає ще більше можливостей у контролі над графічним ядром, що вимагає більше знань та зусиль від програміста, проте, за рахунок такого контролю Vulkan є дуже оптимізованим та ефективним, тому більшість застосунків, додатків та програм, які працюють чи містять у собі 3Д графіку, переходять на цей інтерфейс прикладного програмування [5]. Також він підтримує кросплатформність і може взаємодіяти з різними апаратними операційними системами: Linux, Windows, Android, MacOS;

– DirectX. Являє собою перелік із прикладних програмних інтерфейсів від Microsoft, всі вони відрізняються один від одного і виконують свої, спеціалізовані задачі. За робою з 3Д графікою відповідає Direct3D, довгий час він був схожий по функціоналу на OpenGL, проте з потребою у новому поколінні API, для роботи з 3Д графікою, вийшла нова версія, а саме DirectX 12, яка по можливостям була схожа на Vulkan. Головною проблемою інтерфейсу є те, що вона працює тільки на операційних системах Windows, що значно обмежує її використання у сучасних, мультиплатформних реаліях;

– Metal. Низькорівнева API від Apple, яка працює на пристроях екосистеми Apple, підтримується тільки їхніми графічними ядрами, що як і з DirectX, значно скорочує можливості для його використання. Для уніфікації та полегшення розробки, для інтерфейсу Vulkan, була розроблена бібліотека, яка дозволяє

інтерпретувати код від Vulkan у код Metal. Попри це, завдяки цьому інтерфейсу можна повністю розкрити потенціал графічних ядр компанії Apple, наприклад використання архітектури рендерингу Apple, яка називається Tile-Based Deferred Rendering. Вона полягає у тому, що сцена чи екран розбивається на прямокутники, де обробка кожного, окремої області відбувається незалежно від інших.

Для розробки програмного забезпечення було обрано Vulkan API, оскільки головними пріоритетними задачами були: можливість роботи на різних апаратних частинах та висока ефективність роботи програми. Оскільки сам Vulkan не надає можливості передачі картинки на екран, то слід додати бібліотеку, яка буде відображувати графічні елементи на екрані. Для цього було обрано фреймворк GLFW (Graphics Library Framework), він надає функціонал створення вікон та обробки взаємодії програми з користувачем. Він може обробляти такі аспекти введення як: клавіші клавіатури, миші та геймпадів, що дає чудовий спектр можливостей для інтерактивної взаємодії з користувачем. Бібліотека має хорошу підтримку розробників та активно оновлюється. Для забезпечення базового функціоналу програми, можливості, які надає ця бібліотека, забезпечують потрібний рівень інтерактивності. Для забезпечення виконання математичних функцій, використовується бібліотека GLM (OpenGL Mathematics), оскільки Vulkan не містить у собі підтримку математичних функцій. Слід зазначити, що більшість бібліотек, які були на початку спеціалізовані для роботи з OpenGL з часом отримали підтримку і для Vulkan.

Основним елементом розробки програми на Vulkan є його пайплайн, який можна поділити на 8 пунктів [6]:

– створення та налаштування екземпляра Vulkan. Початковий крок ініціалізації, який встановлює зв’язок між застосунком та драйвером Vulkan, також на цьому кроці визначаються глобальні параметри. Для ініціалізації цього кроку використовуються функції vkCreateInstance та vkInstanceCreateInfo;

– створення фізичного та логічного пристрою, створення фізичного пристрою це вибір графічного обладнання, яке є на тому апаратному забезпеченні,

з яким працює програма. Логічний пристрій є абстракцією над фізичним, він використовується для визначення того, з яким функціоналом буде працювати Vulkan, тобто на цьому етапі визначаються підтримувані можливості графічного ядра з якими буде працювати інтерфейс;

– створення свопчейну. Свопчейн – це набір зображень, які закидаються у буфер і пізніше використовуються для відображення під час рендерингу, завдяки йому можна отримати ефективний та синхронізований рендер фінального результату;

– створення фреймбуферу використовується для запису команд рендеру у буфері команд, на цьому етапі визначається те, в якій послідовності, які дії будуть виконуватися і в подальшому ці інструкції будуть використовуватись у інших етапах;

– створення проходів рендеру відповідає за визначення того, як дані будуть оброблятися під час рендерингу, це дозволяє встановити порядок обробки одного або більше фреймбуферів. Один прохід може містити декілька підпроходів, які дозволяють розділити процес рендерингу на етапи, які можуть працювати послідовно або паралельно. Кожен підпрохід може мати власні налаштування, такі як стани рендеру, шейдери та ресурси, що дозволяє гнучко налаштовувати процес рендерингу для різних ефектів та методів оптимізації;

– створення графічного конвеєру містить кілька етапів, які описують обробку графічних даних, він аналізує сцену, яка складається із тривимірних об’єктів, текстур, матеріалів, джерел світла, пост-обробки і перетворює отриманий опис сцени у двовимірне зображення, яке буде відображено на екрані [7]. На цьому етапі відбувається компіляція шейдерів з-за допомогою спеціального байт-коду від Vulkan, а саме SPIR-V (Standart Portable Intermediate Representation), завдяки якому можна використовувати один і той самий код шейдеру на різних пристроях без необхідності перекомпіляції. Сім базових етапів, які являють собою графічний конвеєр зображені на рисунку 2.3 [8].

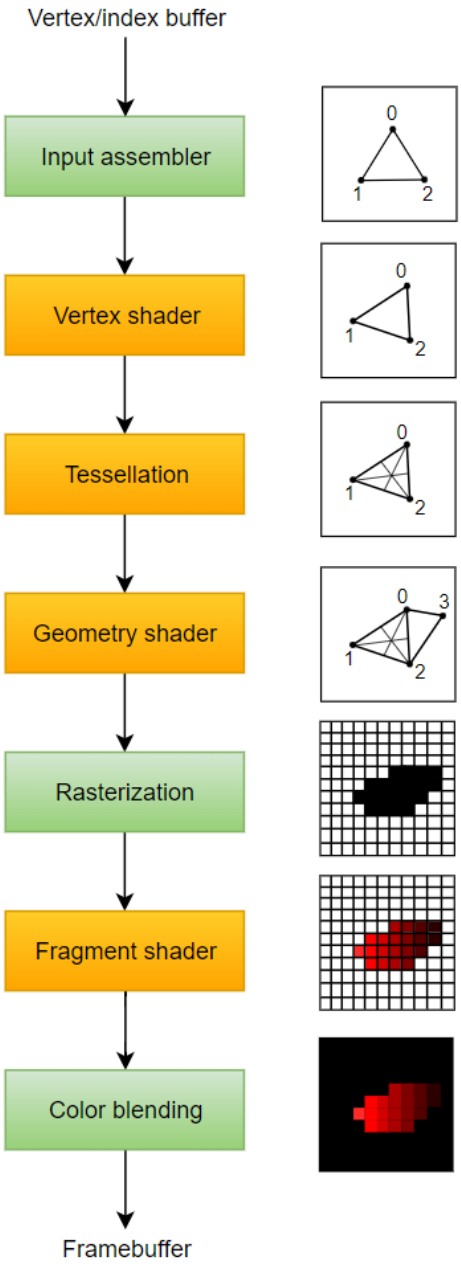

Рисунок 2.3 – Етапи графічного конвеєру [8]

– створення буферів команд та їх запис є механізмом для передачі інструкцій графічному процесору, етап являє собою чергу із команд, які повинен буде виконати графічне ядро, ці команди зазвичай відносяться до рендерингу кадру або виконання обчислень. Ці команди завантажуються у область пам’яті після чого зчитуються відеокартою;

– основний цикл обробляє усі дані, які були записані та попередньо оброблені у минулих етапах, та синхронізує їх у єдиний цикл, який послідовно виконує

функції обробки буферованих даних. Він забезпечує синхронізацію роботи між процесором та графічним процесором, управляє ресурсами даних та їх представлення на екрані.

Процес обробки файлів з хмарами точок включає в себе декілька послідовних кроків, спрямованих на вилучення, перетворення та підготовку даних для подальшої візуалізації. На початковому етапі слід визначити об’єкт, з якого буде отримуватися хмара точок. Приклад сканованого об’єкту зображено на рисунку 2.4.

Рисунок 2.4 – Приклад сканованого об’єкту

Наступним етапом є завантаження вхідних даних до функції алгоритму обробки хмари точок. Алгоритм зчитування має послідовно зчитувати файл, виділяючи такі роздільні елементи як коми чи двокрапки, обираючи тільки цифри. Після чого числа мають записуватись у окремий, виділений масив. Окрім цього, для зчитаних чисел слід обрати координату, до якої вони будуть прикріплені, а також додати ітераційні дані для інших координат, а саме відстань між вершинами по 2 координатам. Також, для того, що б працював текстурний шейдер, слід, з-за

допомогою технології випадкових чисел, додати випадкове число із діапазону від 0 до 256, яке буде позначати колір у адаптивній колірній моделі RGB. Після чого, отриманий масив із координатами вершин, будуть передані у двовимірний масив із вершин у програму, для його відображення.

Висновки до розділу 2

Розробивши специфікацію вимог до розроблюваної системи, було визначено, які необхідні аспекти потрібно реалізувати для програмного проекту, потрібні класи для реалізації необхідного функціоналу та визначені функціональні та нефункціональні вимоги до програмного забезпечення.

Було розглянуто можливі варіанти засобів для реалізації завдання, визначено їх переваги та недоліки. Було обрано та обґрунтовано єдину конфігурацію засобів, для реалізації програмного продукту, крім того, було проаналізовано потрібні методи та алгоритми розробки.

В результаті чого, було здійснено висновок, що для розробки необхідної системи слід використати C++ та Vulkan API, з-за рахунок його кросплатформності, високого рівня оптимізації та широкого контролю процесів. Був визначений алгоритм обробки вхідних даних із текстового файлу.

РОЗДІЛ 3
РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ КОМП'ЮТЕРНОЇ 3D ГРАФІКИ

3.1 Практична реалізація об'єкта проектування

Для початкової реалізації можливостей програми, необхідно розробити початкову ініціалізацію прикладного програмного інтерфейсу Vulkan, бібліотеки для інтерактивної роботи з екраном GLFW та підключити бібліотеку для роботи з математичними функціями GLM.

Реалізація вікна вимагає виклику функцій із бібліотеки GLFW, де необхідно вказати певні специфікації проекту та самого вікна. Проте, при базовому налаштуванні вікна, його не можна збільшувати чи зменшувати, це пов'язано зі специфікою роботи Vulkan, оскільки вікно для відображення, представляє собою відрендерену сцену, то для будь яких маніпуляцій з вікном відображення слід і обробити зміни налаштувань рендеру у Vulkan. Код для реалізації відображення сцени представлений у лістингу 3.1.

Лістинг 3.1 – Реалізація вікна для відображення сцени

```
void initWindow() {
    glfwInit();

    glfwWindowHint(GLFW_CLIENT_API, GLFW_NO_API);

    window = glfwCreateWindow(width, height, "Vulkan", nullptr, nullptr);
    glfwSetWindowUserPointer(window, this);
    glfwSetFramebufferSizeCallback(window, framebufferResizeCallback);
}

static void framebufferResizeCallback(GLFWwindow* window, int width, int
height) {
    auto app =
reinterpret_cast<CloudVulkan*>(glfwGetWindowUserPointer(window));
    app->framebufferResized = true; }
```

Кінець лістингу 3.1

На початку функції initWindow() відбувається ініціалізація бібліотеки GLFW, після чого необхідно вказати цій бібліотеці, що створюване вікно не буде використовувати OpenGL, оскільки бібліотека має базове налаштування на використання саме з інтерфейсом OpenGL. Після чого створюється змінна вікна, яка проводить ініціалізацію об’єкта вікна від бібліотеки GLFW, надалі саме ця змінна буде використовуватись у коді, для взаємодії об’єкту екрана та Vulkan. Для реалізації обробки зміни розміру вікна використана функція framebufferResizeCallback(), яка відповідає за повідомлення зміни стану вікна та передає отриманий сигнал Vulkan.

Для ініціалізації Vulkan та підготовки його до подальшої роботи необхідно визвати та описати багато функцій, тому краще завернути усі функції ініціалізації у єдину функцію, яка зображена у лістингу 3.2.

Лістинг 3.2 – Функції ініціалізації Vulkan

```
void createVulkan() {
    crInstance();
    setupDebugMessenger();
    crSurface();
    pickPhysicalDevice();
    crLogicalDevice();
    crSwapChain();
    crImageViews();
    crRenderPass();
    crDescriptorSetLayout();
    crGraphicsPipeline();
    crFramebuffers();
    crCommandPool();
    crVertexBuffer();
    crIndexBuffer();
    crUniformBuffers();
    crDescriptorPool();
    crDescriptorSets();
    crCommandBuffers();
    crSyncObjects();
}
```

Кінець лістингу 3.2

Функція createVulkan() відповідає за ініціалізацію усіх компонентів Vulkan. Для кращого розуміння призначення кожної складової, слід розглянути їх детальніше:

– crInstance(). Створює екземпляр Vulkan, який представляє підключення програми до бібліотеки та слугує початковим ініціалізатором інших об’єктів;

– setupDebugMessenger(). Функція, яка дозволяє перехоплювати і виводити повідомлення та попередження про помилки, слугує об’єктом налагодження, оскільки сам Vulkan і інструменти для розробки не підтримують обробку помилок цієї бібліотеки;

– crSurface(). Створює вікно операційної системи, в якому відбувається візуалізація. У функції викликаються функції бібліотеки GLFW для реалізації вікна;

– pickPhysicalDevice(). Відповідає за вибір фізичного пристрою, з яким буде працювати Vulkan у подальшому, зазвичай це дискретні відеокарти, проте можуть бути використані будь, які апаратні частини, які підтримують роботу з Vulkan;

– crLogicalDevice(). Створює абстракцію фізичного пристрою, бере участь у створенні таких об’єктів Vulkan як черги команд, конвеєри рендерингу та буфери;

– crSwapChain(). Створює ланцюжок обміну, який представляє собою набір зображень для рендерингу. У Vulkan процес рендерингу полягає у тому, що програма обробляє рендер на одному зображенні, після чого міняє зображення зі старим кадром на нове, з новим відображенням;

– crImageViews(). Реалізовує представлення зображень, для кожного іншого зображення в ланцюжку обміну, функція описує як інтерпретувати дані зображення;

– crRenderPass(). Створює об’єкт рендерингу, який описує формат зображень, які будуть використовуватися під час рендерингу, а також операції над цими зображеннями;

– crDescriptorSetLayout(). Створює макет дескриптора, який використовується для зв’язування ресурсів з шейдерами;

– crGraphicsPipeline(). Функція описує конвеєр рендерингу в якому відбуваються всі етапи пов’язані з рендером зображення;

– crFramebuffers(). Створює кадровий буфер, який прикріплює представлення зображення до об'єкту рендерингу, в створений буфер записуються результати рендерингу;

– crCommandPool(). Створює командний пул, завдяки якому відбувається виділення буферів команд, командний пул слугує контейнером для буферних команд, такий вид організації дозволяє ефективно управляти буферами, що забезпечує високу продуктивність;

– crVertexBuffer(). Створює буфер вершин, який відповідає за збереження вершин 3Д об’єкта, які будуть відображатись на екрані;

– crIndexBuffer(). Створює буфер індексів, який відповідає за порядок з’єднування вершин і організації з них базових площин;

– crUniformBuffers(). Створює буфер відповідальний за передачу даних з головного процесора на графічний процесор, уніфіковані буфери зберігають дані, до яких часто звертаються шейдери;

– crDescriptorPool(). Створює пул дескрипторів, який використовується для виділення наборів дескрипторів, що дозволяє ефективно використовувати групу дескрипторів для єдиних типів завдань;

– crDescriptorSets(). Створює набори дескрипторів, які використовуються для зв’язування ресурсів;

– crCommandBuffers(). Створює командний буфер, який містить набір команд для виконання графічним процесором, це дозволяє виконувати команди у асинхронному режимі на центральному та графічному процесорах;

– crSyncObjects(). Створює об’єкти синхронізації, які використовуються для координації роботи з головним та графічним процесорами.

Після ініціалізації Vulkan, необхідно розробити функціонал програми, а саме: завантаження файлів, відображення хмари точок та взаємодія з камерою для навігації та маніпуляції.

Оскільки вхідні дані являють собою двовимірний масив із 9 координат однієї осі, приклад вхідних даних зображено на рисунку 3.1, то слід додати функцію перетворення та додавання існуючих координат до потрібного рівня, який зчитує шейдер обробки вершин.

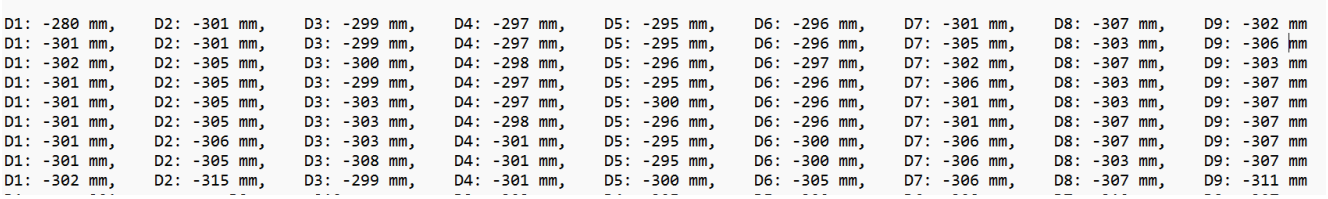

Рисунок 3.1 – Приклад вхідних даних

Для завантаження файлів, було розроблену функцію, яка конвертує вхідні, необроблені дані у масив вершин, окрім цього, для того що б, програма розуміла у якій послідовності з’єднувати вершини для створення ребер чи полігонів, слід організувати алгоритм, який з’єднував би послідовні точки. Для конвертації даних було розроблену функцію transformFile(), яку зображено у лістингу 3.3.

Лістинг 3.3 – Функція для конвертації даних у вершини

```
void transformFile() {
    std::ifstream inputFile(file_BasicData);
    std::ofstream outputFile(file_Vertices);
    std::string line;

    int iteration = 0;
    srand(time(NULL));

    while (getline(inputFile, line))
{
        std::istringstream iss(line);
        std::string token;
        double x = 0.0;

        while (getline(iss, token, ','))
 {
            size_t pos = token.find(":");
            std::string dValue = token.substr(1, pos - 1);
```

```
            std::string zValue = token.substr(pos + 1);
            zValue.erase(zValue.find("mm"), 2);
            Point point;
            point.x = x;
            point.y = iteration;
            point.z = stod(zValue);
            point.r = (double)rand() / RAND_MAX;
            point.g = (double)rand() / RAND_MAX;
            point.b = (double)rand() / RAND_MAX;
            outputFile << point.x << " " << point.y << " " << point.z << " ";
            outputFile << point.r << " " << point.g << " " << point.b << std::endl;
            x += 5;
        }
        iteration ++;
    }
    inputFile.close();
    outputFile.close();
}
```

Кінець лістингу 3.3

Функція зчитує файл, парсить дані та записує їх у окремий файл, окрім цього до тих даних, які були відпарсені, додаються нові, а саме координати по вертикалі X та Y, вони додаються через ітерації циклів обробки зчитування, також до цих даних додаються координати кольорів у форматі RGB, які генерується з-за допомогою функції рандому.

Для відображення хмари точок на екрані, слід реалізувати багато пунктів, які виходять один від одного, кінцева функція реалізації цього процесу є drawFrame(), який показано у лістингу 3.4.

Лістинг 3.4 – Функція візуалізації кадру

```
void drawFrame()
{
    vkWaitForFences(device, 1, &inFlightFences[currentFrame], VK_TRUE,
UINT64_MAX);

    uint32_t imageIndex;
```

```
    VkResult reslt = vkAcquireNextImageKHR(device, swapChain,
UINT64_MAX, imageAvailableSemaphores[currentFr], VK_NULL_HANDLE,
&imageIndex);

    if (res == VK_ERROR_OUT_OF_DATE_KHR) {
        recreateSwapChain();
        return;
    }
    else if (reslt!= VK_SUCCESS && reslt!= VK_SUBOPTIMAL_KHR) {
        throw std::runtime_error("Failed!");
    }

    updateUniformBuffer(currentFr);

    vkResetFences(device, 1, &inFlightFences[currentFr]);

    vkResetCommandBuffer(commandBuffers[currentFr], 0);
recordCommandBuffer(commandBuffers[currentFr], imageIndex);
}
```

Кінець лістингу 3.4

На початку функції відбувається синхронізація з графічним процесором, функція vkWaitForFences(), відповідає за те, що виклик гарантує, що відеокарта завершила обробку попереднього кадру. Після чого функція отримує зображення із ланцюжка обміну, та запитує у Vulkan чи доступне наступне зображення для використання. Далі у функції відбувається логічний запит, на основі результату виклику функції vkAcquireNextImageKHR(), якщо результат повертає помилку того, що ланцюжок обміну застарів, то викликається функція recreateSwapChain(), яка відповідає за перетворення нового ланцюжка обміну. У випадку помилки при отриманні зображення, повертається помилка. Далі у функції оновлюється буфер, який містить дані для теперішнього кадру, після чого очищується буфер команд, для можливості записати нові команди поточного кадру і відповідно, викликається функція для запису команд нового кадру [9].

Для створення функціоналу інтерактивності камери, було написано функцію processInput(), яка головним чином працює з вікном GLFW, та оброблює сигнали у

вигляді натискання клавіш користувачем, та на основі натиснутих клавіш міняється змінна куту чи позиції камери. Код функції зображено у лістингу 3.5.

Лістинг 3.5 – Функція обробки положення камери

```
void processInput(GLFWwindow* window) {
    static auto startTime = std::chrono::high_resolution_clock::now();

    auto currentTime = std::chrono::high_resolution_clock::now();
    float time = std::chrono::duration<float,
std::chrono::seconds::period>(currentTime - startTime).count();

    const float rotationSpeed = glm::radians(1.0f);
    const glm::vec3 axisChange(1.0f, 0.0f, 0.0f);
    const float camS = 1.5f;

    switch (glfwGetKey(window, GLFW_KEY_LEFT)) {
    case GLFW_PRESS: rotationAngle += rotationSpeed; break;
    case GLFW_RELEASE: break;
    }
    switch (glfwGetKey(window, GLFW_KEY_RIGHT)) {
    case GLFW_PRESS: rotationAngle -= rotationSpeed; break;
    case GLFW_RELEASE: break;
    }

    glm::vec3 camM(0.0f);
    if (glfwGetKey(window, GLFW_KEY_W) == GLFW_PRESS) camM.z +=
camS;
    if (glfwGetKey(window, GLFW_KEY_S) == GLFW_PRESS) camM.z -=
camS;
    if (glfwGetKey(window, GLFW_KEY_A) == GLFW_PRESS) camM.x +=
camS;
    if (glfwGetKey(window, GLFW_KEY_D) == GLFW_PRESS) camM.x -=
camS;
    cameraPos += camM;
}
```

Кінець лістингу 3.5

На початку функції визначаються змінні, які визначають час що сплинув після того, як програма запустилась, ці змінні можуть бути використані для поліпшення анімацій чи плавного руху камери. Потім визначаються константи, які

визначають швидкість руху чи обертання камери. Для обробки натискання клавіш використовується функція glfwGetKey() від бібліотеки GLFW, яка обробляє натиск відповідної клавіші у обраному вікні.

Для обробки кожної вершини, відповідають вершинні шейдери, мова програмування, яка використовується для написання шейдерів є GLSL, вона є стандартом для багатьох графічних прикладних програмних інтерфейсів. Для програми був написаний власний вершинний шейдер, який зображено у лістингу 3.6.

Лістинг 3.6 – Вершинний шейдер мовою GLSL

```
#version 450
layout(location = 0) in vec3 inPos;
layout(location = 1) in vec3 inCol;
layout(location = 0) out vec3 fragCol;
layout(set = 0, binding = 0) uniform UBO
{
   mat4 model; mat4 view; mat4 proj;
} ubo;

void main()
{
   gl_Pos = ubo.proj * ubo.view * ubo.model * vec4(inPos, 1.0);
   fragCol = inCol;
}
```

Кінець лістингу 3.6

На початку вказується директива, яка вказує версію GLSL, яка використовується в коді, далі проходить ініціалізація вхідних змін, які відповідають за позицію та колір вершини, та одну вихідну, яка відповідає за передачу кольору фрагментному шейдеру, в кінці ініціалізується уніформований блок, для передачі матриці трансформації з головного процесору на графічний. У функції main описується математична функція, яка буде обробляти кожну вершину.

В результаті проведеного дослідження, було створене програмне забезпечення, яке призначене для зчитування та конвертування вихідних даних

датчика виміру дистанції у тривимірні вершини, після чого створюється візуалізація хмари точок у тривимірному середовищі, приклад роботи програми зображено на рисунку 3.2.

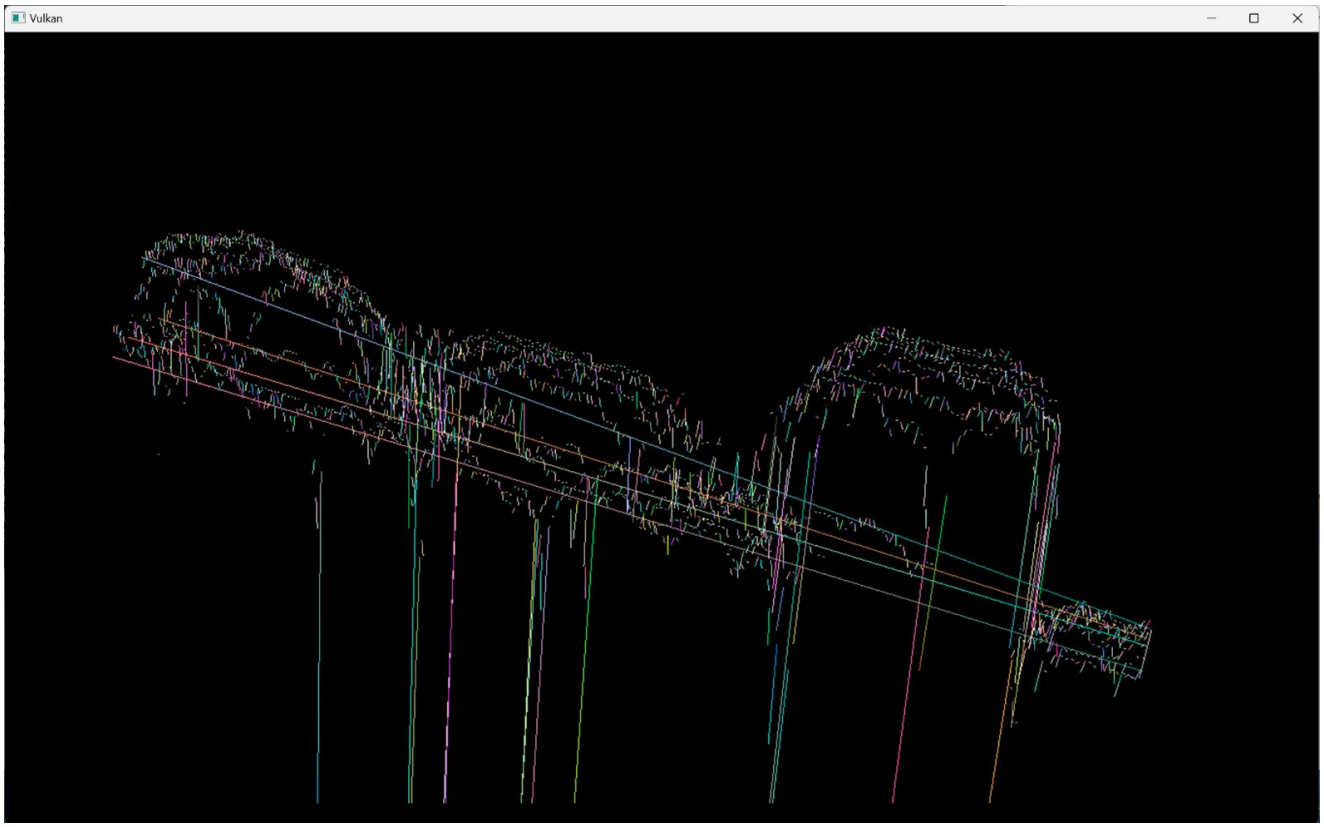

Рисунок 3.2 – Результат роботи розроблюваної програми

3.2 Тестування та налагодження інформаційно-комп’ютерної системи

Після розробки програмного забезпечення для візуалізації тривимірних хмар точок було проведено порівняльний аналіз його продуктивності з існуючими рішеннями. Для оцінки швидкодії було використано метрику FPS, тобто кількість кадрів за секунду, наразі є стандартом показником плавності у програмах та іграх, які використовують чи взаємодіють з тривимірним середовищем.

У ході експерименту було протестовано три системи: MeshLab, CloudCompare та розроблене програмне забезпечення. Середнє значення FPS для MeshLab складає 200, що виявилось найвищою продуктивністю серед 3 програм,

для розроблюваного програмного забезпечення цей показник становив 160 кадрів за секунду, в той час як CloudCompare видавав у середньому 140 FPS. Результати тестування зображенні на рисунку 3.3.

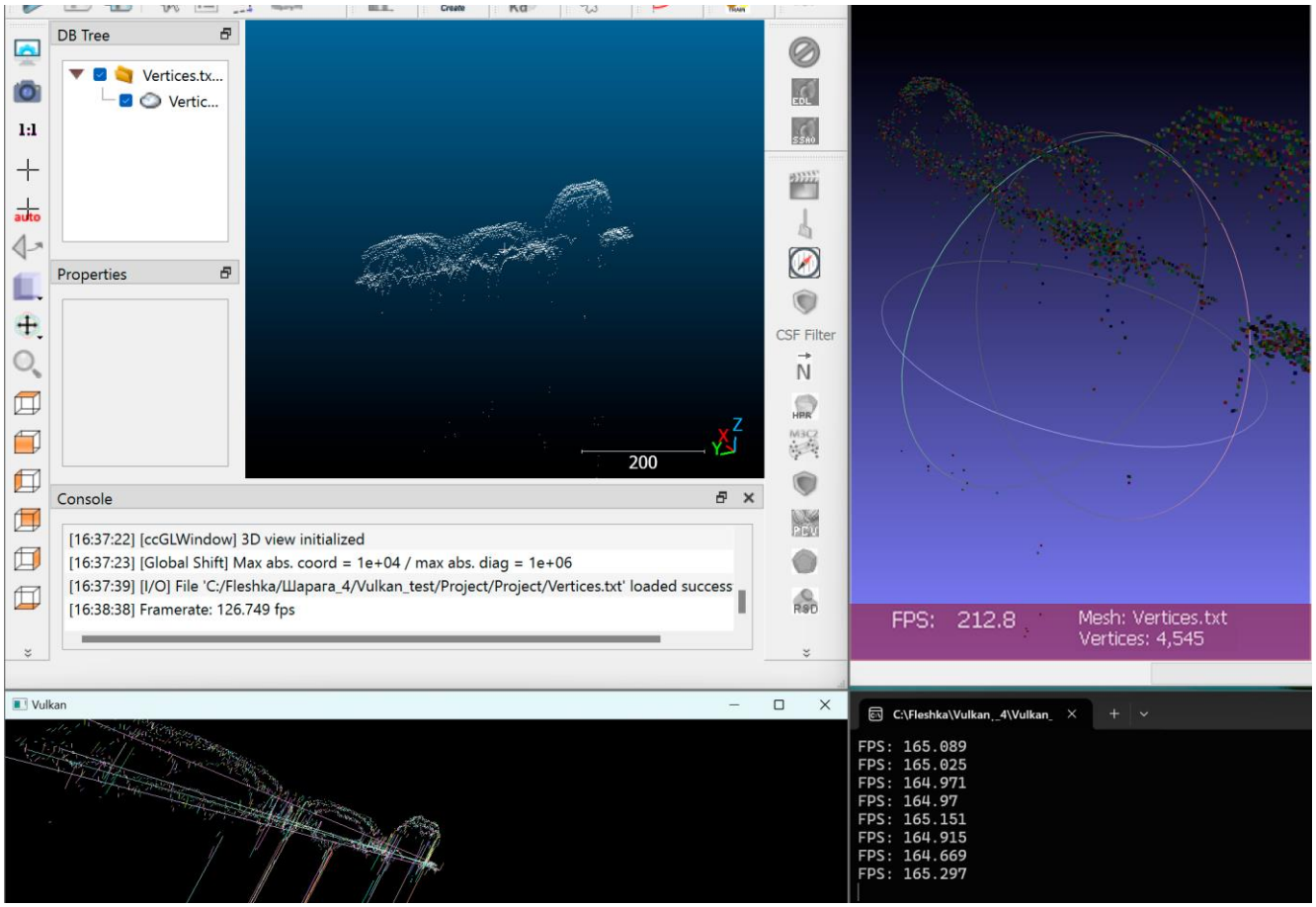

Рисунок 3.3 – Тестування продуктивності програм

Отримані результати дозволяють зробити висновок, що розроблене програмне забезпечення має конкурентну продуктивність порівняно з існуючими рішеннями, хоча MeshLab демонструє деяку перевагу у швидкодії, тому подальша оптимізація розробленої системи може бути спрямована на підвищення показнику FPS, для перевершення існуючого рівня MeshLab.

Мінімальні системні вимоги програмного забезпечення, визначаються наявністю підтримки Vulkan API графічними процесорами та відповідними драйверами. Для коректної роботи програми необхідний обсяг пам’яті на диску має складати щонайменше 150 мегабайт, окрім цього слід розраховувати та додавати

до цього файли з вхідними даними, якими володіє користувач. Мінімальний обсяг оперативної пам’яті складає 2 гігабайти, такий обсяг має забезпечувати базову функціональність програми, проте показник зростає пропорціонально до необхідних для візуалізації кількості вершин. Згідно підтримки драйверів для Vulkan, мінімальною відеокартою для роботи з програмним забезпеченням є GT630.

3.3 Розробка інсталяційного пакета

Розробка інсталяційного пакету для програмного забезпечення була виконана з-за допомогою програми Advanced Installer. Інструмент дозволяє автоматизувати процес створення пакету інсталяції.

Для створення пакету, у застосунку слід налаштувати конфігурацію проекту, а саме визначити назву програми, яка була розроблена, її версію та автора програмного забезпечення. Далі потрібно обрати набір файлів та папок, які будуть завантажені у інсталятор, враховуючи специфіку програмного забезпечення, який використовує Vulkan, до інсталяційного пакету були додані необхідні для роботи бібліотеки та шейдери.

Структура кінцевого продукту складається з 2 основних папок та виконуваного файлу. Перша папка містить у собі бібліотеки, необхідні для роботи програми, друга папка містить шейдери, відповідальні за обробку графічних даних.

3.4 Розробка документації для програмного продукту

Документація до програми надається у вигляді текстового файлу у якій описаний функціонал, можливості використання та можливості взаємодії. Для встановлення програмного продукту необхідно завантажити інсталятор, який розгорне програму у вашій системі. Для того, що б програма відобразила б хмару точок, необхідно завантажити файлі з вхідними даними, після чого на екрані

з’явиться візуалізації хмари. Для взаємодії з камерою необхідно використовувати клавіші W, A, S, D, які відповідають за переміщення камери по осям та клавіші стрілок для повороту камери. Мінімальні системні вимоги для коректної роботи програми включають:

– операційна система: 64-бітна система Windows, дистрибутиви Linux або MacOS;

– графічний процесор: залежить від наявності підтримки відеокарти користувача інтерфейсу Vulkan, мінімальна можлива відеокарта для запуску NVDIA GeForce GT 630 або його аналог від AMD;

– оперативна пам’ять: мінімальний обсяг оперативної пам’яті повинен складати не менше 2 гігабайт, для громіздких обчислень необхідно мати більше оперативної пам’яті;

– місце на диску: для встановлення програми необхідно мати мінімум 150 мегабайт вільного простору, файли зі вхідними даними необхідно враховувати окремо.

3.5 Економічне обґрунтування розробки

Економічна вигода від програмного забезпечення для роботи з тривимірною графікою залежить від конкурентного середовища та позиціонування продукту на ринку. Серед основних конкурентів можна виділити MeshLab та CloudCompare, які є безкоштовними, що робить їх привабливим рішенням для широкої аудиторії користувачів. Однак, по функціональним можливостям вони поступаються таким комерційним продуктам, як AutoCAD та SolidWorks, які стали стандартом у своїй галузі, проте висока вартість їх підписки робить їх придбання обдуманим рішенням, що дає можливість розвиватись можливим конкурентним альтернативам.

Розроблене програмне забезпечення, незважаючи на обмежений функціонал на поточному етапі, має потенціал для успішного розвитку завдяки використанню

Vulkan API, який забезпечує високу продуктивність та оптимізацію для роботи з тривимірною графікою.

Перспективи комерційного успіху розробленого продукту залежать від подальшого розвитку його функціональних можливостей та визначення оптимальної ціни. Ціннник повинен враховувати як конкурентне середовище, так і співвідношення між можливостями розробленого продукту та аналогічними рішеннями на ринку. Подальші дослідження ринку та потреб користувачів дозволяють визначити оптимальну стратегію монетизації та забезпечити успішне просування розробленого продукту.

Для визначення економічної ефективності слід визначити наступні параметри: визначення трудомісткості, розрахунок витрат на розробку, визначення можливої ціни розробленого застосунку, порівняння з існуючими варіантами [10].

Для визначення трудомісткості слід скласти перелік всіх виконаних робіт і визначити витрачений на це час. Після чого, слід визначити розрахунок витрат на розробку, це можна зробити через визначення добутку витраченого часу на середній рівень почасової оплати в сфері.

Розробка програми велась у три етапи: розробка плану та архітектури, написання програмного коду, виконання тестування. На перший етап було витрачено близько 20 годин, другий етап зайняв 49 годин, та на третій було витрачено 6 годин. Середня заробітна платня у сфері розробки програмного забезпечення для 3D графіки, з використанням API Vulkan в Україні становить близько 60 000 гривень на місяць для позиції junior, в погодинній оплаті це близько 300 гривень за годину. Розрахунок витрат на розробку зображено в таблиці 3.1.

Таблиця 3.1 – Визначення витрат на розробку

Етап розробки	Витрачені години	Кінцева ціна етапу
Розробка плану	20	6000
Написання коду	49	14700
Тестування	6	1800

Для визначення можливої ціни на розроблене програмне забезпечення слід скористатися формулою 3.1, яка описує можливу ціну на створену програму.

Відомо, що

$$\text{Ціна } = З_{\text{ндр}} * \left(1 + \frac{\text{Р}}{100}\right), \quad (3.1)$$

де $З_{\text{ндр}}$ – витрати на розробку;

Р – середній рівень рентабельності, % (в середньому 20-30%) [10].

З розрахунку формули, очікувана вартість продукту при рентабельності 25%, виходить 28 125 гривень. Спосіб отримання прибутку слід визначити від маркетингової компанії власника та від функціональних можливостей програми.

Для порівняння економічної ефективності було створено таблицю 3.2.

Таблиця 3.2 – Порівняння економічної ефективності програм

Назва програми	Вартість ліцензії. Доларів у рік	Функціональні можливості	Можливості розширення
AutoCAD	2030	Високі	Відсутні через закритий код
SolidWorks	50+, залежить від сфери використання	Високі	Відсутні через закритий код
MeshLab	0	Середні	Можливо, за рахунок відкритого коду
CloudCompare	0	Низькі	Можливо, за рахунок відкритого коду
Розроблена програма	200	Низькі	Власна розробка дозволяє реалізовувати будь-які доповнення

Аналіз таблиці показує, що вибір оптимального програмного рішення для роботи з 3D графікою залежить від специфічних вимог користувача. Для пересічного покупця, розроблений варіант буде не найкращим вибором, оскільки розробка спеціалізується, в першу чергу, на окремій, специфічній задачі, які інші варіанти не можуть надати або надають з певними обмеженнями.

Власна розробка вимагає значних інвестицій часу, проте тільки завдяки розробці власного рішення можна досягти високої гнучкості у розширенні, що дозволяє імплементувати будь-які потреби до розроблюваного проекту.

Висновки до розділу 3

У цьому розділі було розглянуто процес написання коду програмного забезпечення для тривимірної графіки, використовуючи прикладний програмний інтерфейс Vulkan. Були розроблені та описані основні елементи ініціалізації бібліотеки та самого програмного забезпечення, окрім цього було створено шейдер для обробки вершин та їх зовнішнього вигляду, також була розроблена базова взаємодія з камерою, для більш наглядного відображення результату програми.

Також були розглянуті такі важливі аспекти розробки програмного забезпечення як: визначення системних вимог, створення інсталяційного пакету, написання базової документації та аналіз економічної вигоди та перспективи комерційного використання продукту.

ВИСНОВКИ

В ході реалізації кваліфікаційної роботи було розроблене програмне забезпечення для роботи з комп’ютерною, тривимірною графікою. Були розглянуті існуючі аналоги розроблюваної системи, визначені їх переваги та недоліки, в результаті проробленого аналізу сучасного стану проблеми, було обґрунтовано причини необхідності реалізації нової системи з використанням прикладного програмного інтерфейсу Vulkan.

Під час розробки проектування програмного забезпечення, були проаналізовані варіанти реалізації майбутньої системи та інструменти їх реалізації. Згідно необхідних, визначених вимог було обрано варіант використання мови програмування C++ та прикладного програмного інтерфейсу Vulkan. Визначений комплект для розробки програмного забезпечення повинен максимізувати ефективність використання ресурсів апаратного забезпечення користувача, а також розширити коло потенційних споживачів розробленого програмного забезпечення, за рахунок кросплатформності.

Реалізоване програмне забезпечення являє собою вікно з тривимірною сценою, у якому відбувається візуалізація хмари точок, програма може зчитувати дані із файлу та конвертувати їх у вершини, для подальшої візуалізації. Інтерактивність програми полягає у взаємодії з камерою через клавіші клавіатури, завдяки цьому можна переміщатись по сцені для огляду реалізованої хмари точок.

В ході аналізу ефективності роботи розробленої програми та існуючих рішень, було визначено, що програма має середню продуктивність, поступаючись застосунку MeshLab та переважаючи CloudCompare, що показує конкурентоспроможність на початковому етапі реалізації системи.

Для розробленого програмного забезпечення було розроблено інсталяційний пакет, що забезпечує спрощений та автоматизований процес встановлення на цільові системи. Також було написано документацію, яка поліпшить впровадження та роботу системи з початковими користувачами.

Економічну доцільність розробки було обґрунтовано шляхом аналізу витрачених годин. В ході аналізу було визначено, що орієнтовна ціна за розробку становить 28 125 гривень.

Розроблене програмне забезпечення може бути корисним інженерам та дослідникам, які працюють з приладами, що збирають дані на основі випромінювання та відбиття сигналів, які потім конвертуються у хмару точок.

Існуючий функціонал програми не забезпечує широкого спектру задач для виконання, тому можливими варіантами розвитку застосунку можуть бути такі варіанти покращення: розширення функціоналу у вигляді добавлення можливості редагування хмари точок, вимірювання відстані між окремими точками, фільтрація та сегментація скупчень у єдині об’єкти, конвертування вершин у тривимірні моделі, оптимізація алгоритмів для підвищення ефективності роботи програми, імпорт та експорт різних форматів файлів та реалізація повноцінного інтерфейсу для кращої інтерактивної взаємодії з користувачем.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Ioannidis C., Boutsi A. M. Multithreaded rendering for cross-platform 3D visualization based on Vulkan API. The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences, Volume XLIV-4/W1-2020, 2020 3rd BIM/GIS Integration Workshop and 15th 3D GeoInfo Conference, 7–11 September 2020, London, UK. URL: https://isprs-archives.copernicus.org/articles/XLIV-4-W1-2020/57/2020/isprs-archives-XLIV-4-W1-2020-57-2020.pdf (дата звернення: 10.03.2024).

2. Zhang A., Chen K., Johan H., Erdt M. High-performance adaptive texture streaming and rendering of large 3D cities. The Visual Computer (2022) 38:1245–1262. URL: https://www.researchgate.net/publication/352030795_High-performance_adaptive_texture_streaming_and_rendering_of_large_3D_cities (дата звернення: 28.03.2024).

3. Bakken M. S., George S., Sole A., Yngve J. Standardization of digitized heritage: a review of implementations of 3D in cultural heritage. Storeide et al. Heritage Science (2023) 11:249. URL: https://doi.org/10.1186/s40494-023-01079-z (дата звернення: 28.03.2024).

4. Huang Z., Wen Y., Wang Z., Ren J., Jia K. Surface Reconstruction from Point Clouds: A Survey and a Benchmark. URL: https://arxiv.org/pdf/2205.02413 (дата звернення: 04.04.2024).

5. Панфілов Я. Система моделювання динамічного освітлення в реальному часі. URL: https://ela.kpi.ua/server/api/core/bitstreams/ea792b77-71de-4fd3-a6c0-67471428d645/content (дата звернення: 10.04.2024).

6. Новіков О. Системне програмування та спеціалізовані комп’ютерні системи. URL: https://ela.kpi.ua/server/api/core/bitstreams/99477f82-e305-43c9-b724-8d8a717392f1/content (дата звернення: 13.04.2024).

7. Vulkan Tutorial. URL: https://vulkan-tutorial.com/Drawing_a_triangle/Graphics_pipeline_basics/Introduction (дата звернення: 23.04.2024).

8. Vulkan graphics pipeline image. URL: https://vulkan-tutorial.com/images/vulkan_simplified_pipeline.svg (дата звернення: 23.04.2024).

9. Shirley P., Black T. D., Hollasch S. Ray Tracing in One Weekend. URL:https://github.com/RayTracing/raytracing.github.io/blob/release/books/RayTracingInOneWeekend.html (дата звернення: 15.05.2024).

10. Паздрій І. Р. Опорний конспект лекцій з дисципліни «Техніко-економічне обґрунтування розробки комп'ютерних систем». URL: http://dspace.wunu.edu.ua/retrieve/19273/fkit_kki_dteo_ksm_LEK.pdf. (дата звернення 20.05.2024).