Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра програмного забезпечення

Бакалаврська дипломна робота

на тему: «Розробка програмного застосунку блочної візуалізації LIDAR даних»

Виконав: студент 3 курсу
групи 1ПІ-19мс2
спеціальності
121 – Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)
Верчинський М.О.
(прізвище та ініціали)
Керівник: к.т.н., доц. каф. ПЗ Рейда О.М.
(прізвище та ініціали)
Рецензент: к.т.н., доц. каф. КН Белзецький Р.С.
(прізвище та ініціали)

Допущено до захисту
Зав. кафедри _____
«____» _____ 2022 р.

ВНТУ – 2022

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

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
Романюк О. Н.
25 березня 2022 р.

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

Верчинському Максиму Олександровичу

1. Тема роботи – розробка програмного застосунку блочної візуалізації LIDAR даних.

Керівник роботи: Рейда Олександр Миколаєвич, к.т.н., доц. кафедри ПЗ, затверджений наказом вищого навчального закладу від 24 березня 2022 р. № 66.

2. Строк подання студентом роботи 13 червня 2022 р.

3. Вихідні дані до роботи: середовище розробки JetBrains Rider, мова розробки C#, операційна система – Windows 10, рушій для виведення інформації – Unity.

4. Зміст розрахунково-пояснювальної записки: вступ; аналіз та постановка задачі; порівняльний аналіз аналогів; аналіз методів розв'язання поставленої задачі; розробка алгоритмів системи; розробка методу управління камерою, зчитування та форматування даних; розробка інтерфейсу користувача; обґрунтування вибору засобів для реалізації програмного засобу; розробка модуля візуалізації даних; розробка шейдера для оптимізації рендеринга; тестування системи; висновки; додатки.

5. Перелік графічного матеріалу: мета, об’єкт та предмет дослідження; задачі дослідження; наукова новизна; практичне значення; метод і модель роботи системи; графічний інтерфейс; блок-схема роботи застосунку; діаграма функціональної залежності; тестування системи.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-4	Рейда О. М., к.т.н., доцент кафедри ПЗ		

7. Дата видачі завдання _____25 березня 2022 року _____

КАЛЕНДАРНИЙ ПЛАН

з/п	Назва етапів бакалаврської дипломної роботи	Строк виконання етапів роботи	Примітка
	Аналіз технології LIDAR та вибір цілей для поставленої задачі	26.03.2022 – 16.04.2022	Вик.
	Розробка алгоритмів візуалізації LIDAR даних	14.04.2022 – 01.05.2022	Вик.
	Розробка модуля блочного відображення LIDAR даних	2.05.2022 – 30.06.2022	Вик.
	Розробка модуля шейдерного рендеринга	1.06.2022 – 10.06.2022	Вик.

Студент _____ Верчинський М.О.
(підпис) (прізвище та ініціали)

Керівник бакалаврської дипломної роботи _____ Рейда О.М.
(підпис) (прізвище та ініціали)

Анотація

Бакалаврська дипломна робота складається з 80 сторінок формату А4, на яких є 20 рисунків, 2 таблиці, список використаних джерел містить 18 найменувань.

У бакалаврській дипломній роботі описано розробку програмного застосунку для блочної візуалізації LIDAR даних. Сформульовано мету досліджень – розробка програмного застосунку блочної візуалізаці LIDAR даних для підвищення швидкодії відображення на екрані.

Запропоновано метод підвищення швидкодії відображення LIDAR даних за допомогою кластерних блоків у відеопамяті. Запропоновано модель динамічного динамічного завантаження блоків LIDAR даних у динамічні кластери.

Отримані в бакалаврській дипломній роботі результати можна використовувати науковцями LIDAR технологій для досліджень в різних галузях.

Програмний продукт було створено з використанням мови програмування C#. Для розробки графічного інтерфейсу було використано рушій Unity. В якості середовища розробки програмного забезпечення було обрано JetBrains Rider.

Abstract

The bachelor's thesis consists of 80 A4 pages, which have 20 figures, 2 tables, a list of sources used contains 18 items.

The bachelor's thesis describes the development of a software application for block visualization of LIDAR data. The purpose of the research is formulated - to develop a software application for block visualization of LIDAR data to increase the speed of display on the screen.

A method for increasing the speed of LIDAR data display using cluster blocks in video memory is proposed. A model of dynamic dynamic loading of LIDAR data blocks into dynamic clusters is proposed.

The results obtained in the bachelor's thesis can be used by scientists of LIDAR technologies for research in various fields.

The software product was created using the C # programming language. The Unity engine was used to develop the graphical interface. JetBrains Rider was chosen as the software development environment.

ЗМІСТ

ВСТУП

Обґрунтування вибору теми дослідження. З початку 2010-х років технології автоматизованих транспортних засобів розвивалися швидкими темпами, а тому зросла їх популярність. Майже всі погоджуються, що майбутнє автомобілів – це безпілотні автомобілі. Вони можуть полегшити життя кожному та зменшити кількість дорожньо-транспортних пригод через людські помилки.

Один із кроків назустріч автоматизації – технологія LIDAR.

LIDAR – це метод дистанційного зондування, який використовується для вимірювання точної відстані до об’єкта на земній поверхні. Незважаючи на те, що вперше він був використаний у 1960-х роках, коли лазерні сканери були встановлені на літаках, LIDAR отримав заслужену популярність лише через двадцять років. Лише протягом 1980-х років після впровадження GPS він став популярним методом для розрахунку точних геопросторових вимірювань. LIDAR зазвичай використовується для створення карт високої роздільної здатності, які застосовуються в геодезії, геоматиці, археології, географії, геології, геоморфології, сейсмології, лісовому господарстві, атмосферній фізиці, лазерному наведенні, бортовому лазерному картографуванні смуги (ALSM) та лазерній альтиметрії. Він також використовується для керування та навігації для деяких автономних автомобілів та для гелікоптера Ingenuity під час його рекордних польотів над місцевістю Марса.

LIDAR використовує ультрафіолетове, видиме або ближнє інфрачервоне світло для зображення об’єктів. Він може націлюватися на широкий спектр матеріалів, включаючи неметалічні предмети, скелі, дощ, хімічні сполуки, аерозолі, хмари і навіть окремі молекули. Вузький лазерний промінь може картувати фізичні характеристики з дуже високою роздільною здатністю; наприклад, літак може картувати місцевість з роздільною здатністю 30 сантиметрів (12 дюймів) або краще.

Технології LIDAR поділяється на 3 основні типи:

– система на основі орієнтації – може бути орієнтований в надир, зеніт або збоку. Наприклад, лідарні висотоміри дивляться вниз, атмосферний лідар дивиться вгору, а системи запобігання зіткненням на основі лідарів дивляться збоку;
– система на основі механізму сканування – лазерними проекціями лідарів можна маніпулювати за допомогою різних методів і механізмів для отримання ефекту сканування: стандартного шпиндельного типу, який обертається для огляду на 360 градусів; твердотільний LIDAR, який має фіксоване поле зору, але не має рухомих частин, і може використовувати або MEMS, або оптичні фазовані решітки для керування променями; і спалах-лідар, який розповсюджує спалах світла на велике поле зору, перш ніж сигнал відбивається назад на детектор;
– система на основі платформи – застосування LIDAR можна розділити на повітряні та наземні. Для цих двох типів потрібні сканери з різними специфікаціями, що залежать від призначення даних, розміру області, яку потрібно захопити, бажаного діапазону вимірювань, вартості обладнання тощо.

Для забезпечення швидкодії і підвищення ефективності візуалізації LIDAR даних необхідно використати велику кількість ресурсів, що приводить до значного збільшення вартості обчислювальної системи і витрат на її обслуговування.

Існуючі методи і засоби системи візуалізації LIDAR даних характеризуються використанням спеціалізованих технічних засобів або значною обчислювальною складністю, що в значній мірі впливає на конкурентну спроможність системи візуалізації.

Тому актуальними є питання підвищення швидкості візуалізації LIDAR даних і використання малої кількості системних ресурсів для підвищення ефективності роботи і покращення програмної системи візуалізації.

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

Мета та завдання дослідження. Мета бакалаврської дипломної роботи полягає у підвищення швидкодії і ефективності відображення LIDAR даних шляхом використання блочної візуалізації.

Головні задачі дослідження:

- провести аналіз існуючих методів і засобів візуалізації LIDAR даних;
- запропонувати нові:
    - метод підвищення швидкодії відображення LIDAR даних особливість якого полягає в використанні кластерних блоків у відеопамяті для відображення даних;
    - метод динамічного завантаження блоків LIDAR даних у динамічні кластери для оптимізацію процесу відображення даних;
- розробити програмні компоненти та систему візуалізації на основі запропонованих методів:
    - модуль зчитування LIDAR файлів;
    - модуль форматування LIDAR даних;
    - модуль блокового відображення даних;
    - графічний шейдер для точок у просторі;
- провести експериментальні дослідження розроблених засобів текстурування.

Об’єкт дослідження – процес розробки програмного застосунку для блочної візуалізації LIDAR.

Предмет дослідження – методи та засоби розробки програмного застосунку для блочної візуалізації LIDAR.

Методи дослідження. У процесі досліджень використовувались методи дослідження: теорія чисел та чисельних методів, методи відображення інформації,

комп’ютерне моделювання для аналізу та перевірки отриманих теоретичних положень, методи аналізу та синтезу для побудови ключових моделей системи відображення даних, методи прикладної теорії інформації і теорії алгоритмів для розробки алгоритмів і програмного забезпечення, методи тестування для перевірки працездатності системи.

Наукова новизна отриманих результатів.

1. Уперше запропоновано метод відображення LIDAR даних, який, навідміну від існуючих, використовує кластерні блоки відеопамяті, що дозволяє підвищити швидкодію відображення даних.
2. Подальшого розвитку отримав метод динамічного завантаження блоків LIDAR даних у динамічні кластери, що дозволяє підвищити швидкодію відображення даних і зменшити ресурси ОЗП для формування кластерних блоків у памяті.

Практична цінність отриманих результатів. Практична цінність одержаних результатів полягає в тому, що на основі отриманих в бакалаврській дипломній роботі результатів, розроблено програмний застосунок для візуалізації LIDAR даних, що може використовуватися для аналізу даних дистанційного зондування земної поверхні.

Особистий внесок здобувача. Усі наукові результати, викладені у бакалаврській дипломній роботі, отримані автором особисто. У друкованих працях, опублікованих у співавторстві, автору належать такі результати: аналіз систем відображення LIDAR даних [1].

Апробація матеріалів бакалаврської дипломної роботи. Основні положення бакалаврської дипломної роботи доповідалися та обговорювалися на: Науково-технічна конференція підрозділів Вінницького національного технічного університету (Вінниця 2022).

Публікації. Основні результати досліджень опубліковано у матеріалах конференції.

1 ОБҐРУНТУВАННЯ ВИБОРУ МЕТОДУ РОЗРОБКИ ТА ПОСТАНОВКА ЗАДАЧ ДОСЛІДЖЕННЯ

1.1 Аналіз технології LIDAR

LIDAR – це метод дистанційного зондування, який використовує світло у вигляді імпульсного лазера для вимірювання діапазонів (змінних відстаней) до Землі. Ці світлові імпульси – у поєднанні з іншими даними, записаними бортовою системою – генерують точну тривимірну інформацію про форму Землі та характеристики її поверхні [2].

Технології зондування відіграють ключову роль в успіху вбудованих систем зору. Як наслідок, можна знайти новітніші технології, зокрема технології тривимірного зондування глибини, такі як виявлення світла та визначення дальності (LIDAR), стереобачення та час польоту (ToF). Вони набули значення в автономних транспортних засобах, автоматизації заводів, розумній роздрібній торгівлі тощо.

Найстарішим методом тривимірного зондування глибини є пасивні стереокамери, які працюють шляхом обчислення розбіжності пікселів від двох синхронізованих датчиків. Однак пасивна технологія мала той недолік, що ці камери не можна використовувати в темряві. Також якість глибини залежить від текстури об’єктів у сцені. Щоб подолати це, використовується технологія активного стереобачення. Ця технологія використовує ІЧ-проектор для освітлення сцени, що покращує продуктивність при слабкому освітленні та добре працює на сценах з об’єктами менш складної текстури.

Однак стереокамери не змогли запропонувати великий діапазон глибини (у діапазоні 10 метрів). Крім того, для обчислення глибини необхідно додатково обробляти дані зі стереокамер. Крім того, стереокамери часто не відповідають необхідному рівню точності, особливо коли текстура цільового об’єкта нерівна. Тут може допомогти LiDAR. У випуску Технологічного четверга цього тижня

давайте дізнаємося, що таке датчик LiDAR, його компоненти та програми, де він використовується.

Системи 3D LIDAR можна класифікувати, виходячи з їх функціональних можливостей, на бортовий LIDAR і наземний LIDAR.

Повітряний лідар – датчик 3D LIDAR, як правило, встановлюється на дроні або вертольоті. Він посилає імпульс світла до поверхні землі і чекає, поки імпульс повернеться відразу після удару об'єкта, щоб забезпечити точне вимірювання його відстані. Повітряний LIDAR можна далі класифікувати на топологічний LIDAR і батиметричний LIDAR. Батиметричний LIDAR використовує зелене світло, що проникає в воду, для вимірювання висоти морського дна та русла річки.

Наземний лідар – наземна система LIDAR встановлюється на транспортні засоби, що рухаються, або штативи на поверхні землі. Цей 3D-датчик LIDAR використовується для картування природних характеристик будівель і уважного спостереження за автострадами. Він також надзвичайно корисний для створення точних 3D-моделей об'єктів спадщини. Наземну систему LIDAR можна далі розділити на мобільний LIDAR і статичний LIDAR.

Лідарний прилад в основному складається з лазера, сканера та спеціалізованого GPS-приймача. Літаки та гелікоптери є найбільш часто використовуваними платформами для отримання даних з лідарів у великих районах. LIDAR буває двох типів — топографічні та батиметричні. Топографічний лідар зазвичай використовує лазер ближнього інфрачервоного випромінювання для картування суші, тоді як батиметричний лідар використовує зелене світло, що проникає в воду, щоб також вимірювати висоту морського дна та русла річки.

Системи LIDAR дозволяють науковцям і фахівцям з картографування досліджувати як природне, так і створене людиною середовище з точністю та гнучкістю. Вчені NOAA використовують лідар для створення більш точних карт берегової лінії, створення цифрових моделей висот для використання в географічних інформаційних системах, для допомоги в операціях з реагування на надзвичайні ситуації та в багатьох інших програмах [3].

Існує широкий спектр спеціалізацій де може застосовуватись системи LIDAR. Одними з них є:

– сільсько-господарська промисловість;
– археологія;
– автономні транспортні засоби;
– геологія та ґрунтознавство;
– воєна промисловість.

Принцип роботи технології LIDAR можна пояснити наступними загальними елементами:

– лазерне джерело;
– сканер;
– детектор;
– GPS приймач;
– інерціальна одиниця вимірювання (IMU).

На рис. 1.1 наведено архітектуру датчика LiDAR.

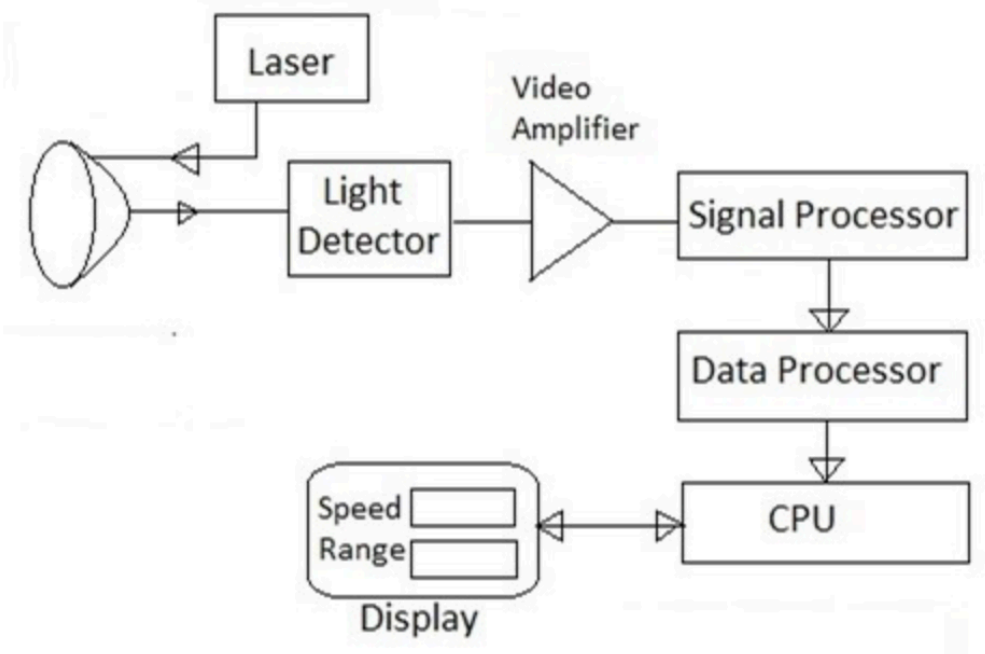

Рисунок 1.1 – Архітектура датчика LIDAR

Розглянемо кожен із компонентів камери LIDAR детально.

Лазерне джерело – випромінює лазерні імпульси різної довжини хвилі. Найпоширенішим джерелом лазера є легований неодимом ітрій-алюмінієвий гранат (Nd-YAG). Наприклад, топографічний датчик LIDAR використовує Nd-YAG лазери з діодною накачкою 1064 нм або безпечну для сітківки довжину хвилі 1550 нм. На відміну від цього, батиметричні системи LiDAR використовують Nd-YAG-лазери з подвійною діодною накачкою 532 нм для проникнення через водні поверхні з меншим ослабленням. Зазвичай для збільшення дальності дії систем використовуються дуже короткі лазерні імпульси в невидимому БІК діапазоні.

Сканер використовує відхиляючі дзеркала для відхилення лазерного променя і досягнення широкого поля зору (FoV). Сканер для вимірювання глибини розроблений з дальною дистанцією, широким кутом зору та швидкісним скануванням.

Детектор вловлює відбите світло від перешкод. Основними фотоприймачами є твердотільні фотоприймачі, такі як кремнієві лавинні фотодіоди та фотопомножувачі.

GPS приймач.У бортовій системі LIDAR GPS-приймач відстежує висоту та розташування літаків. Ця змінна важлива для отримання точних значень висот місцевості.

Інерціальні одиниці вимірювання (IMU) – відстежує швидкість та орієнтацію літака (або будь-якого іншого автономного транспортного засобу, де використовується LIDAR). Ці вимірювання точно визначають фактичне положення імпульсу на землі.

Тепер, коли ми розуміємо, як працюють датчики LIDAR, давайте розглянемо два найпопулярніші випадки їх використання.

Автономні транспортні засоби та пристрої.

Автономне обладнання, таке як дрони, автономні трактори, роботизована зброя тощо, залежить від 3D-камер із визначенням глибини для виявлення перешкод, локалізації, навігації та вибору чи розміщення об’єктів. Виявлення

перешкод вимагає низької затримки для швидкого реагування і, у багатьох випадках, більш широкого поля зору. Для транспортних засобів, які рухаються на великій швидкості, знадобиться датчик LIDAR. Він має функцію лазерного променя, що обертається на 360 градусів, що, у свою чергу, означає, що вони мають точний і точний огляд, щоб уникнути перешкод і зіткнень. Оскільки LIDAR генерує мільйони точок даних в режимі реального часу, він легко створює точну карту свого постійно мінливого оточення для безпечної навігації автономних транспортних засобів. Крім того, точність відстаней LIDAR дозволяє системі автомобіля ідентифікувати об’єкти в найрізноманітніших погодних умовах і умовах освітлення.

Автономні мобільні роботи (AMR).

AMR використовуються для вибору, транспортування та сортування товарів у виробничих приміщеннях, складах, роздрібних магазинах та установах розподілу без безпосереднього нагляду оператора. Незалежно від того, чи це робот для вибору та розміщення, патрульний робот або сільськогосподарський робот, майже всі AMR повинні вимірювати глибину. А з системою LIDAR немає необхідності в обширній обробці для виявлення об’єктів або створення карт, що робить її ідеальним рішенням для AMR.

1.2 Порівняльний аналіз інструментів для відображення LIDAR даних

Існує досить велике різноманіття інструментів для візуалізації LIDAR даних. До найбільш відомих відносять:

– Online LIDAR PCV;
– Plas.io;
– Displaz;
– LasTools.

Online LIDAR PCV (див. рис. 1.2) – веб-система візуалізації хмарних точок [4].

Особливістю інструмента є зручний інтерфейс та доступ до великої кількості LIDAR даних. Система дозволяє глибоко настроювати відображення хмарних точок та передивлятись дані у встроєному каталозі.

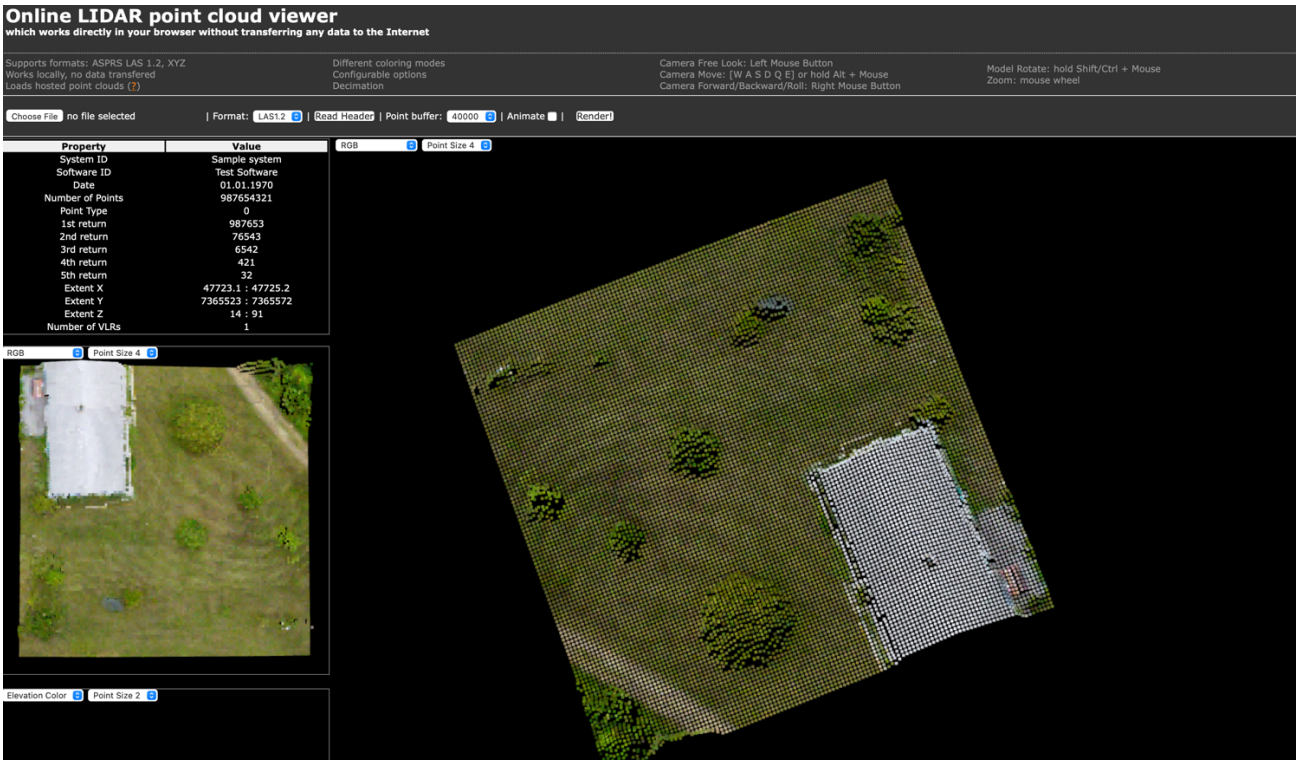

Рисунок 1.2 – Головна сторінка сервісу Online LIDAR PCV

Plas.io (див. рис. 1.3) – сучасна веб-платформа для швидкого відображення LIDAR даних [5]. Дана технологія є Open Source додатком, отже любий може відкрити вихідний код та вдосконалити або модернізувати систему. Також присутня велика кількість налаштувань. Plasio — це проект Удая Верми та Говарда Батлера, який реалізує можливість візуалізації хмари точок у браузері. Зокрема, він забезпечує функціональну реалізацію формату ASPRS LAS і може споживати стиснені LASzip дані за допомогою модуля LASzip NaCl. Цей інструмент корисний, тому що не потрібно нічого встановлювати, щоб використовувати його, необхідно просто перетянути файл в браузер і почати візуалізацію.

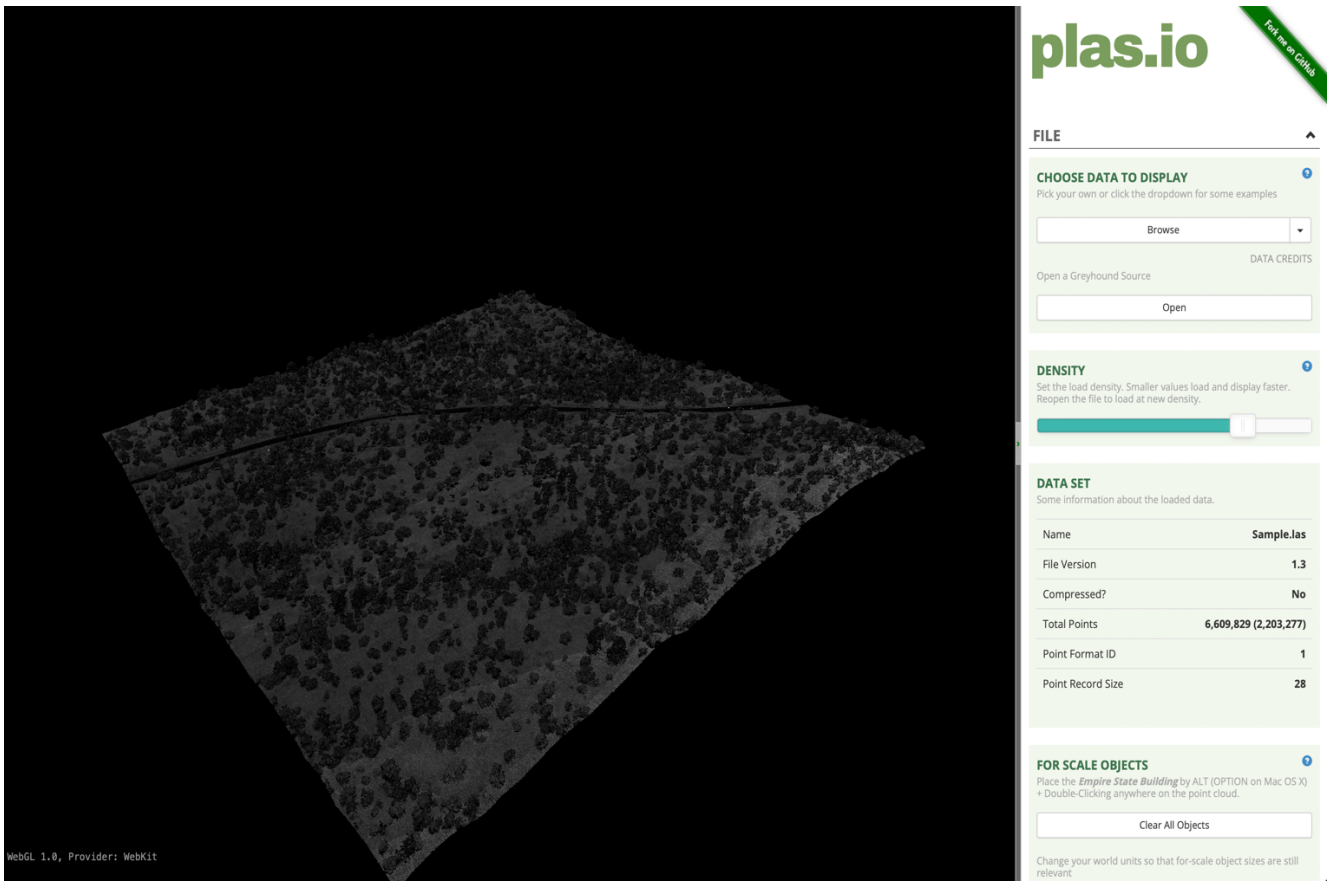

Рисунок 1.3 – Головна сторінка сервісу Plas.io

Displaz (див. рис. 1.4) – це кросплатформний засіб перегляду для відображення хмар точок лідару та отриманих артефактів, таких як встановлені сітки[6]. Інтерфейс спочатку був розроблений для перегляду великих бортових лазерних сканів, але також добре працює для хмарних точок, отриманих за допомогою наземного LIDAR та інших джерел, таких як батиметричний гідролокатор. Система дозволяє забезпечити гнучкий і програмований технічний інструмент для дослідження великих наборів даних точок лідару та отриманої геометрії. Мета інструмента полягає в тому, щоб забезпечити гнучкий і програмований технічний інструмент для дослідження великих наборів даних точок лідару та отриманої геометрії.

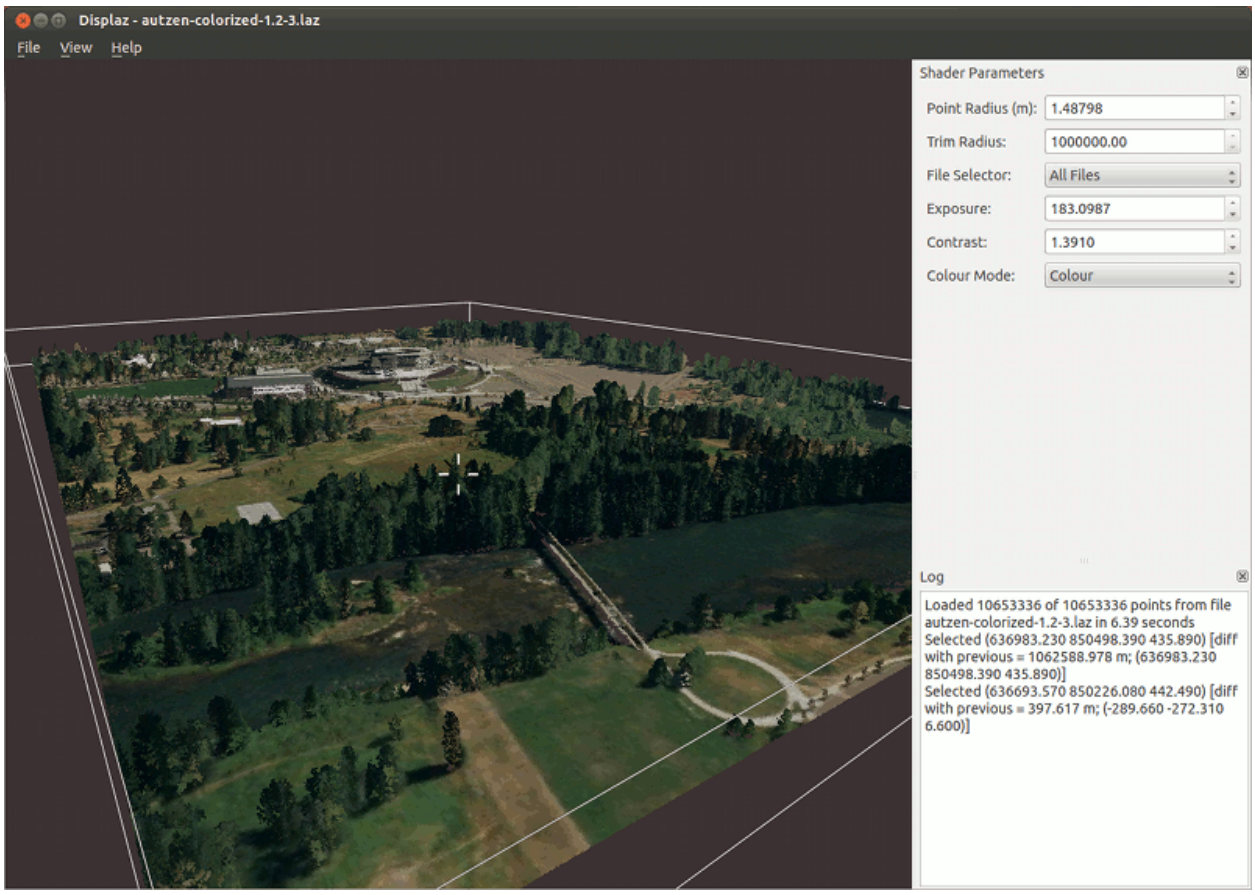

Рисунок 1.4 – Головне вікно інструменту Displaz

LasTools (див. рис. 1.5) – це в основному набір або поєднання інструментів командного рядка, які призначені для класифікації, тріангуляції та відсікання даних LIDAR на користь користувачів. Принадність LASTools полягає в тому, що їх також можна використовувати в графічному інтерфейсі користувача або як пакет для всіх цілей обробки LIDAR у різних версіях. Весь код написаний на легкому, дуже ефективному та надшвидкому C++. Основні модулі надаються як вихідний код (наприклад, для Linux) і як DLL Windows. Це набір високоефективних інструментів із сценарієм із багатоядерними пакетами, які обробляють LAS, стиснений LAZ, Terrasolid BIN, ESRI Shapefiles, ASCII та інші [7].

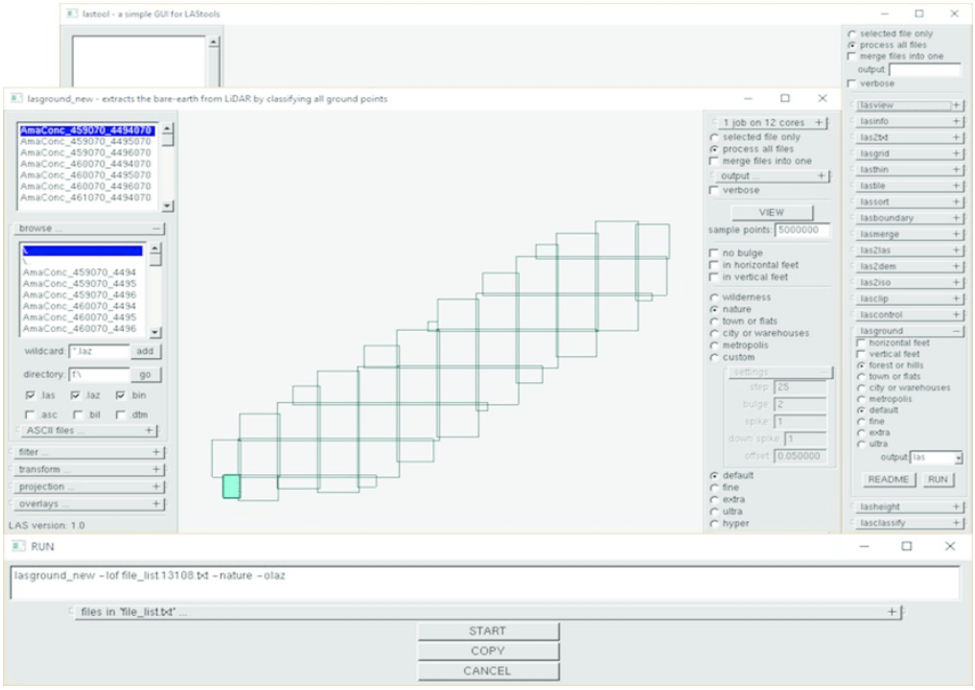

Рисунок 1.5 – Головна вікно інструменту LasTools

Результати порівняння аналогів приведено в табл. 1.1.

Таблиця 1.1 – Порівняльна характеристика інструментів

Критерії	Online LIDAR PCV	Plas.io	Displaz	LasTools	FastLasVisualize
Наявність веб версії	1	1	0	0	0
Підтримка відображення точок через шейдер	1	0	1	0	1
Підтримка LAS та LAZ форматів	0	0	0	1	1
Можливість запуску на мобільних девайсах	0	0	1	0	1
Підтримка блочної візуалізації	0	0	1	1	1
Сумарний коефіцієнт	2	1	3	2	4

Відповідно до таблиці порівняльних характеристик розробка власного програмного застосунку для візуалізації LIDAR даних є доцільною. Отриманий продукт зможе покрити недоліки існуючих рішень та забезпечити функціонал блочної візуалізації LIDAR даних.

1.3 Аналіз методів розв’язання поставленої задачі

Для розв'язання поставленої задачі існує кілька рішень. Найбільш швидким в розробці є варіант з використанням готової бібліотеки, яка дозволяє реалізувати систему зчитування та записування LIDAR файлів. Даний підхід можна вважати достатньо ефективним, однак він передбачає докладання додаткових зусиль зі сторони відображення хмарних точок. Основним недоліком є те, що необхідно буде створювати свою систему рендера та відображення даних. Тому даний підхід не варто реалізовувати.

Альтернативою попереднього методу є розробка власного модуля зчитування бінарних LIDAR даних та використання готової бібліотеки для рендера та візуалізації. Даний підхід дає можливість зосередитись на розробці саме необхідного модуля без повномасштабної розробки програмного застосунку. Користувачі даного інструменту матимуть можливість вибрати будь яку бібліотеку для відображення LIDAR даних. Це зменшить час на створення повного інструменту з нуля. Однак, для зчитування LAS або LAZ файлів потрібно досконало знати специфіку роботи LIDAR датчиків. Тому даний метод вважається ефективним за можливості виділення додаткового часу на розробку модуля зчитування бінарних LIDAR даних [8].

Найбільш досконалим є рішенням розробки власного програмного застосунку для візуалізації на базі рушія Unity з використаням бібліотек для зчитування бінарних LIDAR даних. У даному випадку зникає необхідність створення системи зчитування та записування LAS та LAZ форматів. Також результатом даного методу є повноціна система рендеринга яка дозволяє за допомогою шейдерів у будь якому вигляді відображати хмарні точки. Даний підхід

теж має свій недолік оскільки потребує знання специфіки створення програмованих мешей та розробки шейдерів [9].

Врахувавши переваги та недоліки кожного методу було вирішено використати метод розробки власного програмного застосунку візуалізації LIDAR даних на основі рушія Unity та бібліотек роботи з LAS та LAZ форматами файлів

1.4 Постановка задач для програмного застосунку візуалізації даних

Проаналізувавши переваги та недоліки існуючих інструментів візуалізації LIDAR даних, було сформульовано такі задачі бакалаврської дипломної роботи:

- розробити метод автоматизації процесу зчитування LAS та LAZ форматів файлів;
- розробити метод розділення всіх даних на блоки для подальшої оптимізації візуалізації;
- розробити шейдерну програму для зручного відображення хмарних точок;
- розробити зручний інтерфейс для огляду візуалізації;
- провести тестування програмного застосунку згідно поставлених задач.

1.5 Висновки

У першому розділі розглянуто стан технології LIDAR. Було проведено аналіз інструментів для візуалізації LIDAR даних, таких як: Online LIDAR PCV, Plas.io, Displaz та LasTools. У результаті виявлено, що досліджувані інструменти набули популярності серед користувачів та науковців завдяки своїй доступності та швидкодії. Проаналізувавши переваги та недоліки відомих систем візуалізації, було виявлено, що вони не надають можливості зручної візуалізації LIDAR даних. Таким чином доведено доцільність розробки власного програмного рішення. На основі отриманої інформації було сформовано перелік задач, які необхідно виконати для розробки власного програмного застосунку.

2 РОЗРОБКА МЕТОДУ БЛОЧНОЇ ВІЗУАЛІЗАЦІЇ ТА АЛГОРИТМІВ ВІДОБРАЖЕННЯ LIDAR ДАНИХ

2.1 Аналіз даних

Основним аспектом LIDAR даних є тривимірні хмари точок. Хмара точок – це сукупність величезної кількості вимірювань: набір точок даних або координат в трьох вимірах. Вимірювання зазвичай виконуються за допомогою лазерних 3D-сканерів і технології виявлення і визначення дальності. Лазер вимірює, де світло падає на поверхні в межах його прямої видимості. Щоб швидко захопити величезну кількість знятих точок деякі лазерні 3D-сканери мають швидкість вимірювання до 2 мільйонів точок в секунду [10].

Хмару точок необхідно обробити, щоб створити тривимірну модель реальності. Необхідно зареєструвати і зв’язати остаточні дані.

Реєстрація хмари точок – це коли вирівнюються хмари точок які перекриваються – якщо переміщували лазерний сканер в різні позиції на об’єкті, щоб захопити більшу або більш повну сцену – для формування однієї точної моделі області. Створення сітки – це процес коли програмне забезпечення перетворює дані хмари точок в трикутники або багатокутники, щоб представити поверхню відсканованих об’єктів; сітка зберігає дані вихідних точок, але вона менше і працює швидше.

Деякі з зібраних координат не знадобляться, і подальша обробка може відфільтрувати і уточнити дані. Наприклад, якщо просканувати торговий центр, можна помітите багато людей, які ходять навколо. Остаточна обробка реєстрації видалить ці небажані примарні об’єкти, в результаті буде чистіший, менший і більш точний набір даних. Можливість візуалізувати дані, щойно зібрані, перебуваючи на місці, означає, що є можливість вносити виправлення або виконувати додаткове сканування без повторного відвідування об’єкта. Надалі, для остаточної реєстрації точок в офісі використовуються вже більш потужні програми. Вони потрібні також для побудови сітки, подальшої обробки та створення набору

остаточних результатів 3D і звітів. Крім того, існують рішення для хмарного зберігання проектів, що забезпечує візуалізацію тривимірних даних і спільну роботу в Інтернеті всіх зацікавлених сторін.

LIDAR – це активний оптичний датчик, який передає лазерні промені на ціль під час руху певними маршрутами дослідження. Відображення лазера від цілі виявляється та аналізується приймачами в датчику лідару. Ці приймачі фіксують точний час від моменту, коли лазерний імпульс покинув систему, до моменту його повернення, щоб обчислити відстань між датчиком і метою. У поєднанні з позиційною інформацією (GPS та INS) ці вимірювання відстані перетворюються на вимірювання фактичних тривимірних точок відбиваючої цілі в просторі об’єкта.

Дані точок обробляються після збору даних лідаром у високоточні координати x,y,z з географічними прив’язками шляхом аналізу діапазону часу лазера, кута лазерного сканування, положення GPS та інформації INS.

Лазерні імпульси, що випромінюються системою лідарів, відбиваються від об’єктів як на поверхні землі, так і над її поверхнею: рослинність, будівлі, мости тощо. Один випромінюваний лазерний імпульс може повертатися до датчика лідару як один або декілька одиниць.

Перший повернений лазерний імпульс є найбільш значущим поверненням і буде пов’язаний з найвищою об’єктом ландшафту, як-от верхівка дерева або верхівка будівлі. Перше повернення також може представляти землю, і в цьому випадку системою лідарів буде виявлено лише одне повернення.

Кілька повернень здатні виявити висоту кількох об’єктів у межах лазерного відбитка вихідного лазерного імпульсу. Проміжні повернення, як правило, використовуються для структури рослинності, а остання повернення для моделей рельєфу голої землі.

Останнє повернення не завжди буде з наземного повернення. Наприклад, розглянемо випадок, коли імпульс потрапляє на товсту гілку на шляху до землі, а імпульс насправді не досягає землі. У цьому випадку останній відхід відбувається не від землі, а від гілки, яка відбивала весь лазерний імпульс.

Атрибути точки LIDAR.

Додаткова інформація зберігається разом із кожним позиційним значенням x, y та z. Для кожного зареєстрованого лазерного імпульсу зберігаються наступні атрибути точки лідару: інтенсивність, число повернення, кількість повернень, значення класифікації точок, точки, що знаходяться на краю лінії польоту, значення RGB (червоний, зелений і синій), час GPS , кут сканування та напрямок сканування. Нижче описані атрибути, які можна надати кожній точці LIDAR:

- Інтенсивність - зворотна сила лазерного імпульсу, який генерував точку лідару.
- Id повернення - випромінюваний лазерний імпульс може мати до п’яти повернень залежно від характеристик, від яких він відбивається, і можливостей лазерного сканера, який використовується для збору даних. Перше повернення буде позначено як повернення номер один, друге як повернення номер два і так далі.
- Кількість повернень – це загальна кількість повернень для даного імпульсу. Наприклад, лазерна точка даних може повертати другий (номер повернення) у загальній кількості п’яти повернень.
- Точкова класифікація - кожна точка лідару, що піддається подальшій обробці, може мати класифікацію, яка визначає тип об’єкта, який відбив лазерний імпульс. Лідарні точки можна розділити на кілька категорій, включаючи голу землю або землю, верх навісу та воду. Різні класи визначаються за допомогою числових цілих кодів у файлах LAS.
- RGB. Лідарні дані можуть бути віднесені до RGB (червоних, зелених і синіх) смуг. Ця атрибуція часто походить із зображень, зібраних одночасно з дослідженням лідаром.
- Час GPS.Відмітка часу GPS, за якою лазерна точка була випромінена з літака. Час вказано в GPS секундах тижня.
- Напрямок сканування — це напрямок, у якому рухалося дзеркало лазерного сканування під час вихідного лазерного імпульсу. Значення 1

є позитивним напрямком сканування, а значення 0 – негативним напрямком сканування. Позитивне значення вказує, що сканер рухається з лівого боку на правий бік напрямку польоту на колії, а від'ємне значення – протилежне.

2.2 Розробка методу управлінням камерою

Для зручності огляду візуалізації необхідно добавити спосіб управляння камерою. Було вирішено зробити його максимально простим для використання.

За основу було взято звичайне управління 3D простору в іграх. Схема управління камерою наступна:

– W – летіти вперід;
– A – летіти вліво;
– S – летіти назад;
– D – летіти вправо;
– Shift – збільшити швидкість руху камери в два рази;
– LMB + Mouse – змінити кут огляду камери.

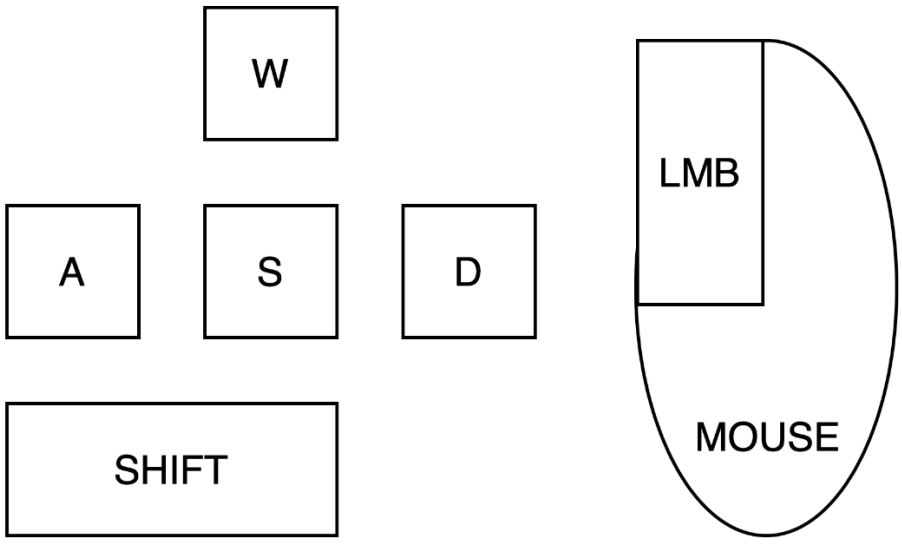

Рисунок 2.1 – Схема управління візуалізацією

2.3 Розробка методу зчитування та форматування LIDAR даних

Для візуалізації хмарних точок необхідно отримати дані цих точок з файлу. Формати файлу LIDAR даних бувають різними. Але основними вважаються два:

– LAS – формат який займає більше простору на диску ніж LAZ але дозволяє дуже швидко зчитувати та записувати дані [11];
– LAZ – протилежність до LAS, це зкомпресований формат який займає менше місця в порівняні з LAS але досить повільний для роботи з ним

Даний метод передбачає зчитування любого з вище перечислених форматів та форматування його в зручний для роботи з Unity вигляд (див. рис. 2.2). Далі ці дані необхідно передати в модуль блочної візуалізації яка вже буде відображати хмари точок.

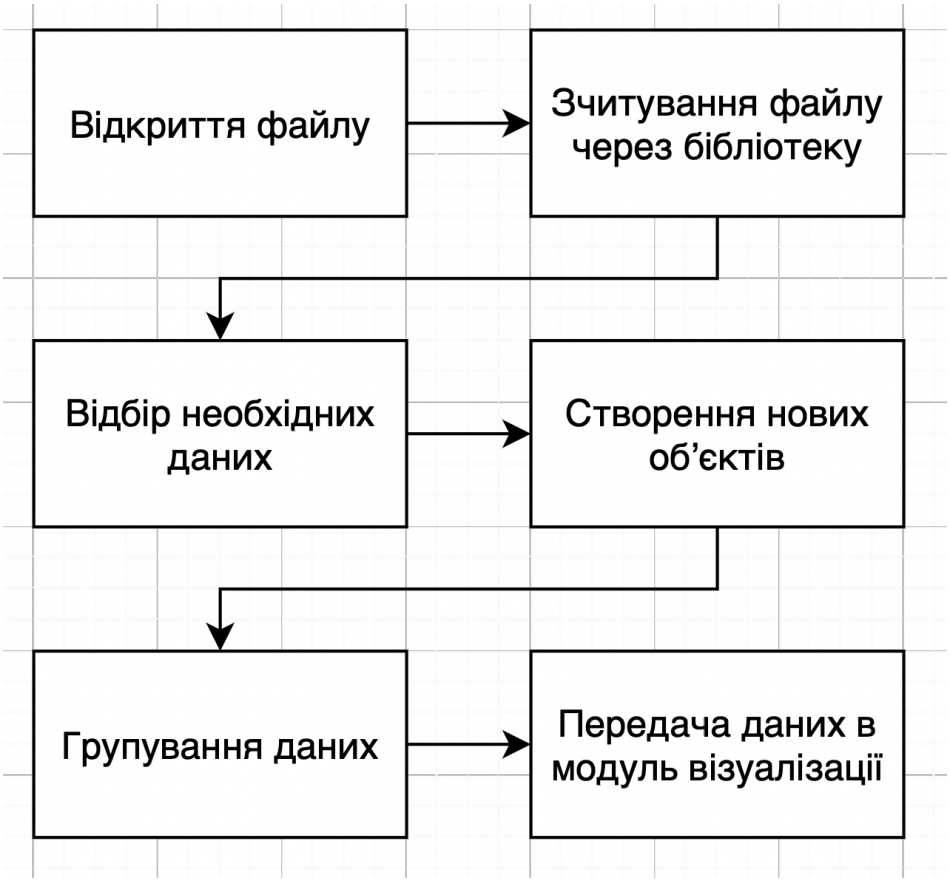

Рисунок 2.2 – Метод зчитування та форматування LIDAR файлів

2.4 Розробка моделі роботи візуалізації LIDAR даних

Для кращого розуміння роботи програмного застосунку для блочної візуалізації LIDAR даних вирішено розробити модель роботи системи з використанням UML діаграми класів (див. рис. 2.3). Дані діаграми найкращим чином дозволяють описати роботу програми та взаємодію різних модулів. Згідно вказаної діаграми спочатку клієнт форматує та отримує необхідні дані. Наступним кроком програма передає відформатовані дані в блок візуалізації. Перед початком візуалізація дані знову форматуються поблоково та передаються в модуль рендера. Далі на екрані відображаються хмари точок які формують фінальну візуалізацію LIDAR даних.

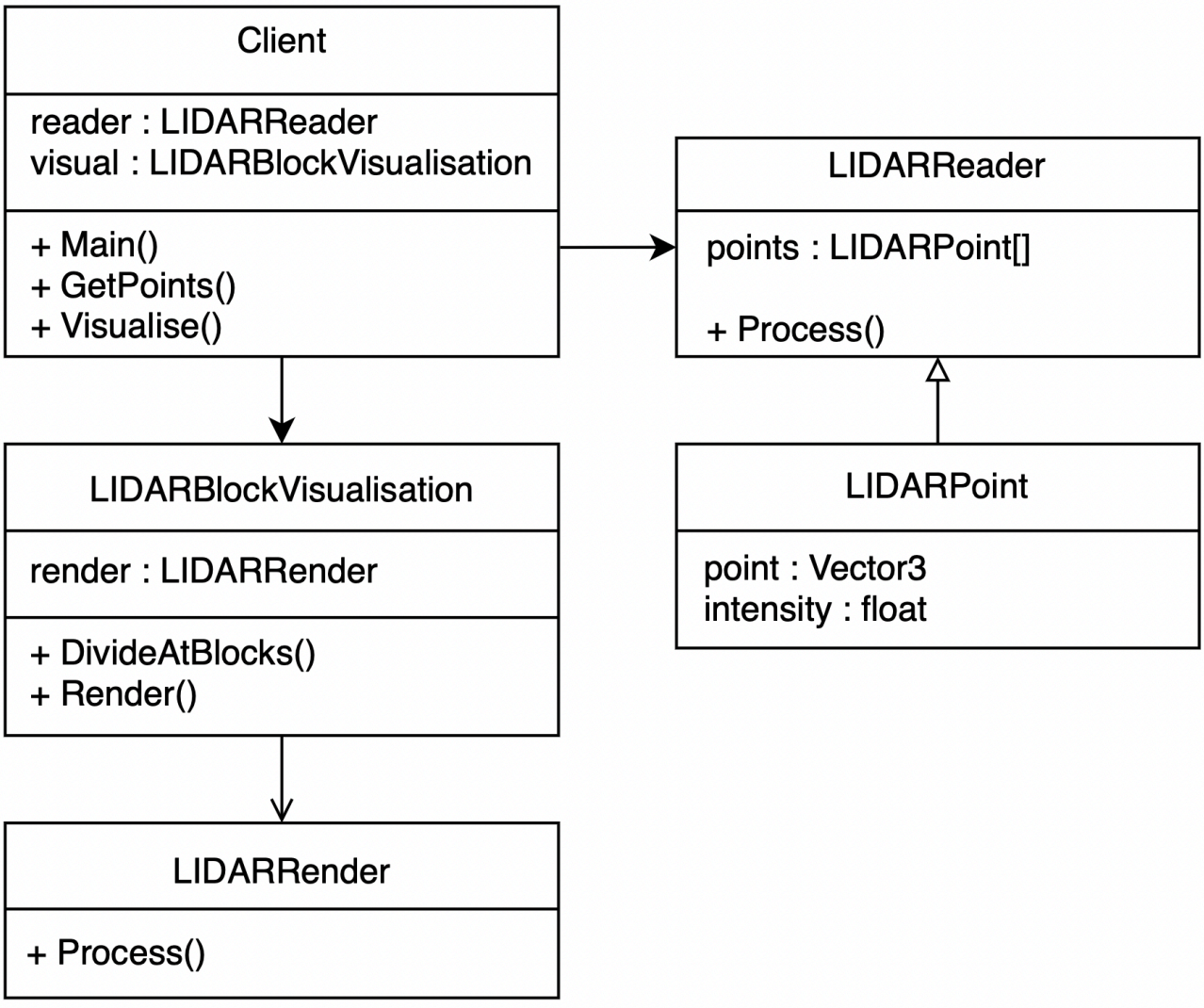

Рисунок 2.3 – UML діаграма класів візуалізації LIDAR даних

2.5 Розробка алгоритмів роботи візуалізації LIDAR даних

Для розробки програмного застосунку блочної візуалізації LIDAR даних необхідно розробити загальний алгоритм, згідно якого буде працювати система (див. рис. 2.4). Перший крок алгоритму - користувач відкриває файл який містить LIDAR дані. Далі файл проходить перевірку на недопустимий формат та на кількість хмарних точок. Якщо хоча б одна з цих перевірок не пройде, з’явиться помилка та вікно відкриття нового LIDAR файлу. Програма підтримує тільки два формати, LAS та LAZ. Тому що це найбільш популярні та розповсюджені формати файлів які легко знайти на спеціалізованих ресурсах [12]. Якщо всі перевірки будуть пройдені, програма відформатує вміст файлу, перетворить його на більш зручний для роботи вигляд та порахує їх кількість. Далі розрахується кількість блоків візуалізації на основі загальної кількості хмарни точок. Наступним кроком програма буде в циклі створювати блоки візуалізації та добавляти їх до загального списку. Цикл буде працювати доки кількість списку не буде дорівнювати кількості блоків візуалізації. Далі список з блоками візуалізації передається до модуля візуалізації в якому вже ці дані будуть рендеритись на екрані. Результатом буде тривимірна хмара точок яка є вмістом файлу LIDAR даних.

Особливість представленого алгоритму полягає у використанні динамічного числа кластері яке розраховується в залежності від розміру оперативної пам'яті, що вказується користувачем або частина оперативної пам'яті, що дозволено використовувати. Форматування даних відбувається за рахунок позиціювання точок LIDAR даних у певнені кластери, які обмежується блоками простору заданого розміру.

Розглянувши даний алгоритм, можна зрозуміти, що даний підхід до візуалізації даних дозволить ефективно відображати вміст любого LIDAR файлу.

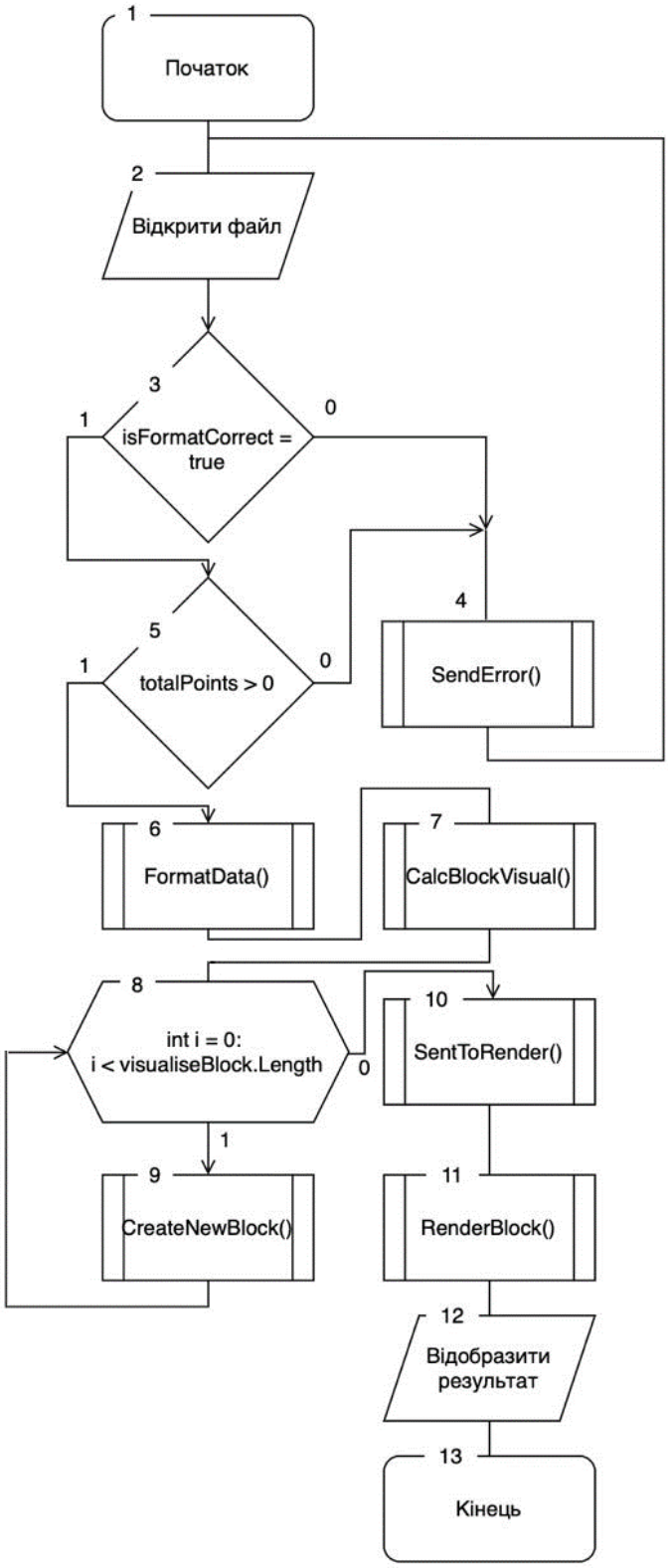

Рисунок 2.4 – Блок-схема загального алгоритму програми

2.6 Висновки

У другому розділі було проведено аналіз LIDAR даних. Розглянуто методи зчитування та форматування LAS та LAZ файлів з тривимірними хмарами точок. Було розроблено основний алгоритм роботи блочної візуалізації LIDAR даних. Також було розроблено метод блочного відображення для опимізації роботи програмного застосунку. Розроблено модель блочної візуалізації LIDAR даних за допомогою UML діаграми класів.

3 РОЗРОБКА ПРОГРАМНОГО ЗАСТОСУНКУ БЛОЧНОЇ ВІЗУАЛІЗАЦІЇ LIDAR ДАНИХ

3.1 Варіантний аналіз і обґрунтування вибору засобів для реалізації програмного засобу

Розробка програмного забезпечення завжди починається з вибору правильних технологій. Від вибору залежить час який буде затрачений на роботу та якість результату. Вибір технологій повинен залежати від цілей його використання та задач які стоять перед розробником. Необхідно знайти баланс між простотою використання інструмента та його ефективністю.

C# – це об'єктно-орієнтована мова програмування від Microsoft, яка призначена щоб поєднати обчислювальну потужність C++ із легкістю програмування Visual Basic. C# заснований на C++ і містить функції, подібні до Java. Мова розроблена для роботи з платформою Microsoft .NET. Метою Microsoft є полегшення обміну інформацією та послугами через Інтернет, а також надання розробникам можливості створювати дуже портативні програми. C# спрощує програмування завдяки використанню Extensible Markup Language (XML) і Simple Object Access Protocol (SOAP), які дозволяють отримати доступ до об'єкта або методу програмування, не вимагаючи від програміста писати додатковий код для кожного кроку. Оскільки програмісти можуть використовувати наявний код, а не повторювати його, очікується, що C# зробить його швидшим і менш дорогим для виведення нових продуктів і послуг на ринок [13].

Unity – це кросплатформний ігровий рушій, розроблений Unity Technologies, вперше анонсований і випущений у червні 2005 року на Apple Worldwide Developers Conference як ігровий движок Mac OS X. З того часу рушій поступово розширювався для підтримки різноманітних настільних, мобільних, консольних і віртуальних платформ. Він особливо популярний для розробки мобільних ігор для iOS і Android і вважається простим у використанні для початківців розробників і популярним для розробки інді-ігор. Рушій можна використовувати для створення

тривимірних і двовимірних ігор, а також інтерактивних симуляцій та інших можливостей. Рушій був прийнятий у галузях за межами відеоігор, таких як кіно, автомобільна промисловість, архітектура, інженерія та будівництво [14].

The High-Level Shader Language або High-Level Shading Language (HLSL) – це запатентована мова затінення, розроблена Microsoft для Direct3D 9 API, щоб розширити мову ассемблера шейдерів. HLSL є аналогом мови затінення GLSL, що використовується у стандарті OpenGL. Він дуже схожий на мову затінення Nvidia Cg, оскільки був розроблений разом з ним [15]. Шейдери HLSL можуть забезпечити глибоке збільшення швидкості та деталізації, а також багато спеціальних ефектів як у 2D, так і в 3D комп'ютерній графіці. Програми HLSL випускаються в шести формах: піксельні шейдери (фрагмент у GLSL), вершинні шейдери, геометричні шейдери, обчислювальні шейдери, шейдери теселяції (шейдери Hull і Domain) і шейдери трасування променів. Вершинний шейдер виконується для кожної вершини, поданої програмою, і в першу чергу відповідає за перетворення вершини з об'єктного простору в простір перегляду, генерування текстурних координат і обчислення коефіцієнтів освітлення, таких як нормаль, дотична та бітангенс вершини. Коли група вершин (зазвичай 3, щоб утворити трикутник) проходять через вершинний шейдер, їх вихідне положення інтерполюється, щоб утворити пікселі в межах його області; цей процес відомий як растеризація [16].

Шейдери — це прості програми, які описують ознаки вершини або пікселя. Вершинні шейдери описують атрибути (положення, координати текстури, кольори тощо) вершини, тоді як піксельні шейдери описують ознаки (колір, z-глибину та альфа-значення) пікселя. Вершинний шейдер викликається для кожної вершини в примітиві (можливо, після теселяції); таким чином одна вершина входить, одна (оновлена) вершина виходить. Кожна вершина потім відображається у вигляді серії пікселів на поверхні (блоку пам'яті), які в кінцевому підсумку будуть відправлені на екран.

Шейдери замінюють частину графічного обладнання, яку зазвичай називають конвеєром фіксованих функцій (FFP), так званим, оскільки він виконує освітлення та відображення текстур жорстко закодованим способом. Шейдери забезпечують програмовану альтернативу цьому жорстко запрограмованому підходу.

Основний графічний конвеєр виглядає наступним чином:

– ЦП відправляє інструкції (складені програми мови затінення) і дані геометрії в графічний процесор, розташований на відеокарті.
– Усередині вершинного шейдера геометрія трансформується.
– Якщо шейдер геометрії знаходиться в блоці графічної обробки і активний, виконуються деякі зміни геометрії в сцені.
– Якщо шейдер теселяції знаходиться в блоці графічної обробки і активний, геометрії в сцені можна розділити.
– Розрахована геометрія тріангулюється (розбивається на трикутники).
– Трикутники розбиваються на фрагменти чотирикутника (один фрагмент фрагмент є примітивом 2 × 2).
– Чотири фрагменти змінюються відповідно до фрагментного шейдера.
– Виконується перевірка глибини; фрагменти, які проходять, будуть записані на екран і можуть бути змішані з буфером кадру.

Графічний конвеєр використовує ці кроки, щоб перетворити тривимірні (або двовимірні) дані в корисні для відображення двовимірні дані. Загалом, це велика піксельна матриця або «буфер кадру».

3.2 Розробка управління камери під час візуалізації

Управління – це базовий функціонал який дає змогу керувати програмним застосунком. Для його реалізації необхідно створити об’єкт камери в сцені (див. рис. 3.1) та новий скрипт управління (див. рис. 3.2).

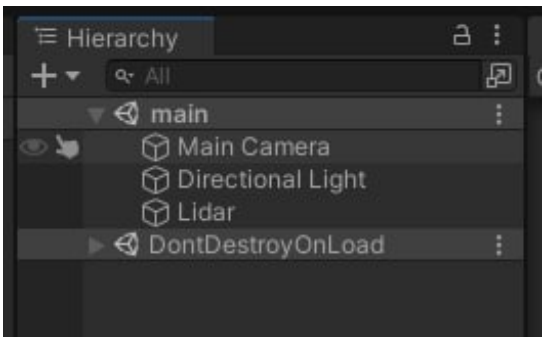

Рисунок 3.1 – Вікно ієрархії з створеною камерою

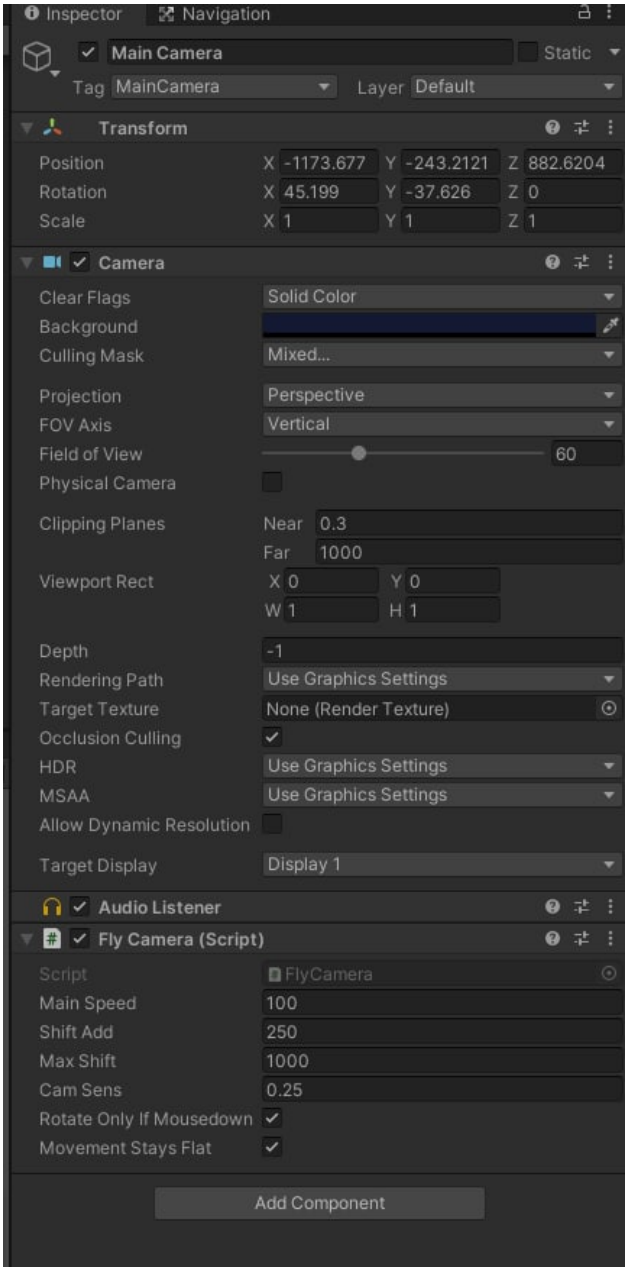

Рисунок 3.2 – Вікно інспектора з скриптом камери

Програмний код ініціалізації полів руху камери приведено на рис.3.3. Оголошено зміні для швидкості камери, чутливості та зручності керування.

```
public float mainSpeed = 100.0f;  Unchanged
public float shiftAdd = 250.0f;  Unchanged
public float maxShift = 1000.0f;  Unchanged
public float camSens = 0.25f;  "0.25"
public bool rotateOnlyIfMousedown = true;  Unchanged
public bool movementStaysFlat = true;  Unchanged
```

Рисунок 3.3 – Ініціалізації полів руху камери

Метод зчитування клавіатури який буде рухати камеру представлено на рис 3.4.

```
private Vector3 GetBaseInput() {
    Vector3 p_Velocity = new Vector3();
    if (Input.GetKey (KeyCode.W)){
        p_Velocity += new Vector3( x: 0, y: 0 , z: 1);
    }
    if (Input.GetKey (KeyCode.S)){
        p_Velocity += new Vector3( x: 0, y: 0, z: -1);
    }
    if (Input.GetKey (KeyCode.A)){
        p_Velocity += new Vector3( x: -1, y: 0, z: 0);
    }
    if (Input.GetKey (KeyCode.D)){
        p_Velocity += new Vector3( x: 1, y: 0, z: 0);
    }
    return p_Velocity;
}
```

Рисунок 3.4 – Метод зчитування клавіатури

Далі необхідно прописати в методі Update() логіку руху камери (рис 3.5). Напрям руху множиться на швидкість та на Time.deltaTime.

```
float f = 0.0f;
Vector3 p = GetBaseInput();
if (Input.GetKey (KeyCode.LeftShift)){
    totalRun += Time.deltaTime;
    p  = p * totalRun * shiftAdd;
    p.x = Mathf.Clamp( value: p.x, min: -maxShift, maxShift);
    p.y = Mathf.Clamp( value: p.y, min: -maxShift, maxShift);
    p.z = Mathf.Clamp( value: p.z, min: -maxShift, maxShift);
}
else{
    totalRun = Mathf.Clamp( value: totalRun * 0.5f, min: 1f, max: 1000f);
    p = p * mainSpeed;
}
```

Рисунок 3.5 – Логіка руху камери

Програмний код перевірки натискання клавіші миші і повороту камери в напрямку миші представлено на рис 3.6.

```
if (Input.GetMouseButtonDown(1))
{
    lastMouse = Input.mousePosition;
}

if (!rotateOnlyIfMousedown ||
    (rotateOnlyIfMousedown && Input.GetMouseButton(1)))
{
    lastMouse = Input.mousePosition - lastMouse ;
    lastMouse = new Vector3( x: -lastMouse.y * camSens, y: lastMouse.x * camSens, z: 0 );
    lastMouse = new Vector3( x: transform.eulerAngles.x + lastMouse.x , y: transform.eulerAngles.y
    transform.eulerAngles = lastMouse;
    lastMouse =  Input.mousePosition;
    //Mouse  camera angle done.
}
```

Рисунок 3.6 – Метод зчитування миші та повороту камери

3.3 Розробка модуля форматування та виводу LIDAR даних

Розроблено програмний модуль, що контролює відкриття, зчитування та форматування LIDAR даних. Для відкриття LAS або LAZ файлу використовується відкрита бібліотека laszip.net. Вона має багато користних

інструментів та великий функціонал по архівації, модифікації та різних маніпуляцій з LIDAR даними.

На рис. 3.7 предмтавлено програмний код створення полів для управління модулем.

```
private List<Chunk> chunks;
[SerializeField] private GameObject goCamera; Main Camera
[SerializeField] private uint totalCountPoints; Unchanged
[SerializeField] private uint countPointsDisplayed; Unchanged
[Space]
[SerializeField] private int viewDistance = 10; "0"
[SerializeField] private Vector3 offset = Vector3.zero; Serializable
[SerializeField] private int chunkCount = 10; "1"
[SerializeField] private int pointsInChunk = 128; "6609829"
[SerializeField] private float size = 1.0f; Unchanged
```

Рисунок 3.7 – Оголошені поля

Поля підписуються атрибутом [SerializeField] щоб його можна було серіалізувати в префабі та зберегти налаштування.

Далі необхідно перекинути посилання на камеру та зробити налаштування для блоків:

– viewDistance – відповідає за даліність рендера блока.
– offset – робить відступ для зручного перегляду.
– chunkCount – вказує на кількість блоків в сцені.
– pointInChunk – необхідне для вказування кількості точок в одному блоці.
– size – треба для задання розміру точки.

В методі Start()завантажується файл даних у пам’ять і розбивається на блоки та форматується (рис 3.8).

```
IEnumerator Start()
{
    countPointsDisplayed = (uint)( chunkCount * pointsInChunk );

    chunks = new List<Chunk>( chunkCount );

    laszip_dll lasReader = new laszip_dll();
    bool compressed = true;
    lasReader.laszip_open_reader(path, ref compressed);
    totalCountPoints = lasReader.header.number_of_point_records;

    double[] coordArray = new double[3];

    for ( int i = 0; i < chunkCount; i++ )
    {
        LoadChunk( lasReader, coordArray );
        yield return null;
    }

    lasReader.laszip_close_reader();

    //goCamera.transform.localPosition = chunks[0].chunkPos;

    SetMesh();
}
```

Рисунок 3.8 – Метод Start()

У методі Start() реалізовано підключення бібліотеки laszip_dll для зчитування файла.

Читання даних відбувається у циклі відповідно до загальної кількості блоків та створює нові блоки. Після завершення читання закривається потік для зчитування та відбувається перехід в наступний метод (рис 3.9).

```
1 usage
private void SetMesh()
{
    mesh = new Mesh();

    GetComponent<MeshFilter>().mesh = mesh;
    GetComponent<MeshRenderer> ().material = new Material (Shader.Find("Custom/VertexColor"));
    CreateMesh();
}
```

Рисунок 3.9 – Метод SetMesh()

У методі SetMesh() створюється новий меш та задається для нього шейдер для рендеринга (рис 3.10).

```
1 usage
private void CreateMesh()
{
    Chunk chunk       = chunks[0];
    int   countPoints = chunks[0].points.Length;

    int[]   indecies = new int[countPoints];
    Color[] colors   = new Color[countPoints];

    for ( int i = 0; i < chunk.points.Length; ++i )
    {
        indecies[i] = i;
        colors[i] = Color.white;
    }

    mesh.indexFormat = UnityEngine.Rendering.IndexFormat.UInt32;
    mesh.vertices = chunk.points;
    mesh.colors = colors;
    mesh.SetIndices( indecies, MeshTopology.Points, submesh: 0 );
}
```

Рисунок 3.10 – Метод CreateMesh()

Після створення мешей візуалізація готово до перегляду. Залишилось лише створити новий скрипт для керування розміру хмарної точки (рис 3.11).

Також є необхідним зробити можливість змінювати розмір хмарних точок. Оскільки відтворенням візуалізації займається шейдер необхідно написати скрипт який буде міняти розмір.

```
public UInt32 GL_VERTEX_PROGRAM_POINT_SIZE = 0x8642;  "59533"
public UInt32 GL_POINT_SMOOTH = 0x0B10;  "5800"

const string LibGLPath =
#if UNITY_STANDALONE_WIN
    "opengl32.dll";
#elif UNITY_STANDALONE_OSX
    "/System/Library/Frameworks/OpenGL.framework/OpenGL";
#elif UNITY_STANDALONE_LINUX
    "libGL";  // Untested on Linux, this may not be correct
#else
    null;  // OpenGL ES platforms don't require this feature
#endif
```

Рисунок 3.11 – Оголошення зміних

Залишилось в методі OnPreRender() вказати необхідні команди для збільшення точок в шейдері (рис 3.12).

```
void Start()
{
    mIsOpenGL = SystemInfo.graphicsDeviceVersion.Contains("OpenGL");

    Debug.Log (message: SystemInfo.graphicsDeviceName + " : " + SystemInfo.graphicsDeviceVersion);
}

Event function
void OnPreRender()
{
    if (mIsOpenGL) {
        glEnable (GL_VERTEX_PROGRAM_POINT_SIZE);
        glEnable (GL_POINT_SMOOTH);
    }
```

Рисунок 3.12 – Опис методу OnPreRender()

3.4 Розробка шейдера візуалізації LIDAR даних

Основною задачею шейдера візуалізації LIDAR даних - це коректне відображення хмар точок у просторі. Для цього необхідно створити файл фрагментного шейдера. Фрагментний шейдер працює з фрагментами зображення. Під фрагментом зображення в даному випадку розуміється піксель, якому поставлено у відповідність деякий набір атрибутів, таких як колір, глибина,

текстурні координати. Фрагментний шейдер використовується на останній стадії графічного конвеєра для формування фрагмента зображення. Для написання шейдера використовується мова GLSL. GLSL - мова високого рівня, синтаксис мови базується на мові програмування ANSI C, однак, через його специфічну спрямованість, з нього були вилучені багато можливостей, для спрощення мови та підвищення продуктивності. У мову долучені додаткові функції і типи даних, наприклад для роботи з векторами і матрицями.

Для початку необхідно оголосити зміні шейдера візуалізації LIDAR даних для відображення хмар точок (рис 3.14).

```
CGPROGRAM
#pragma vertex VSMain
#pragma fragment PSMain
#pragma target 5.0

StructuredBuffer<float> depthbuffer;
StructuredBuffer<float4> colorbuffer;

struct shaderdata
{
    float4 vertex : SV_POSITION;
    float4 color : TEXCOORD1;
};
```

Рисунок 3.13 – Оголошення зміних у шейдері

Наступним кроком є логіка відображення хмарних точок на екрані (рис 3.14).

```
shaderdata VSMain(uint id : SV_VertexID)
{
    shaderdata vs;
    float depth = depthbuffer[id];
    float factor = 512;
    float u = fmod (id, factor) / factor - 0.5;
    float v = floor (id / factor) / factor - 0.5;
    vs.vertex = UnityObjectToClipPos(float4(4.0*u, depth, 4.0*v, 1.0));
    vs.color = colorbuffer[id];
    return vs;
}

float4 PSMain(shaderdata ps) : SV_TARGET
{
    return ps.color;
}
```

Рисунок 3.14 – Опис логіки у шейдері

Далі створюються групи властивостей для зручного створення візуалізаціх в якій задається колір (рис 3.15).

```
Properties
{
    _MainTex ("Texture", 2D) = "white" {}
}
```

Рисунок 3.15 – Опис властивостей

Щоб налаштувати коректне відображення потрібно створити структуру для управління вертексами шейдера (рис 3.16).

```
CGPROGRAM
#pragma vertex vert
#pragma fragment frag

#include "UnityCG.cginc"

struct appdata
{
    float4 vertex : POSITION;
    float2 uv : TEXCOORD0;
};
```

Рисунок 3.16 – Структура шейдера

У структурі описсано позиція вертекса та координати точки на екрані. Далі в методі рендеринга виводиться інформація (рис 3.17).

```
struct v2f
{
    float2 uv : TEXCOORD0;
    float4 vertex : SV_POSITION;
};

v2f vert (appdata v)
{
    v2f o;
    o.vertex = UnityObjectToClipPos(v.vertex);
    o.uv = v.uv;
    return o;
}
```

Рисунок 3.17 – Логіка відображення шейдера

3.5 Налаштування сцени в Unity

Для того щоб закінчити розробку залишилось коректно налаштувати сцену сцену в Unity. Спочатку необхідно створити пустий GameObject (див. рис. 3.3). Далі перейменувати його та повішати скрипт модулю форматування та виводу LIDAR даних (див. рис. 3.4). Також необхідно добавити скрипт для збільшення хмарних точок (див. рис. 3.5). Залишилось лише провести фінальні налаштування (див. рис. 3.6).

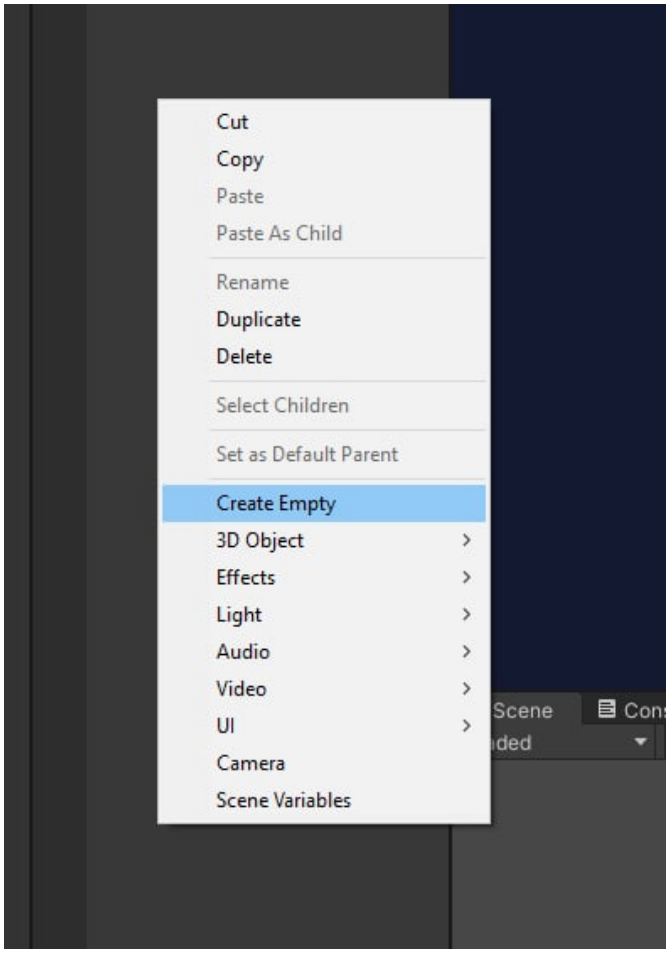

Рисунок 3.3 – Створення нового GameObject

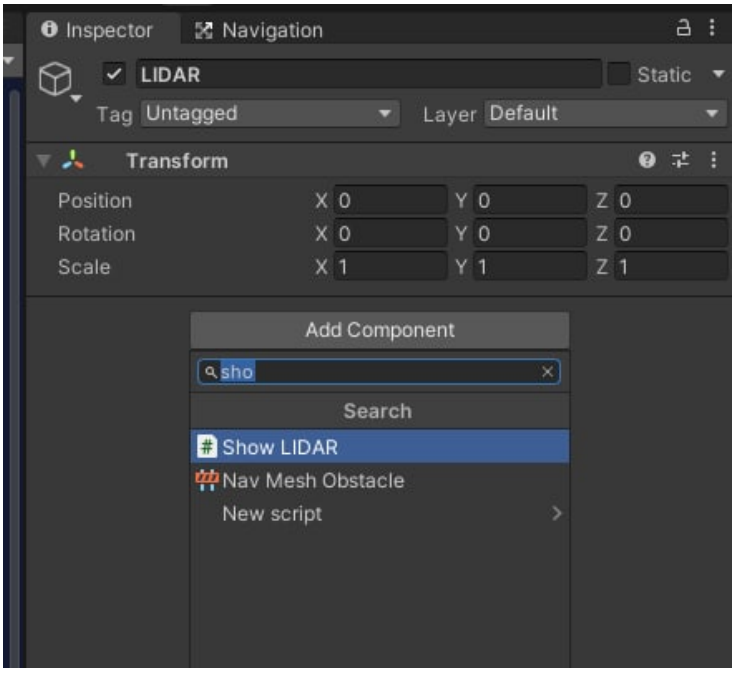

Рисунок 3.4 – Підключення модуля для виводу точок

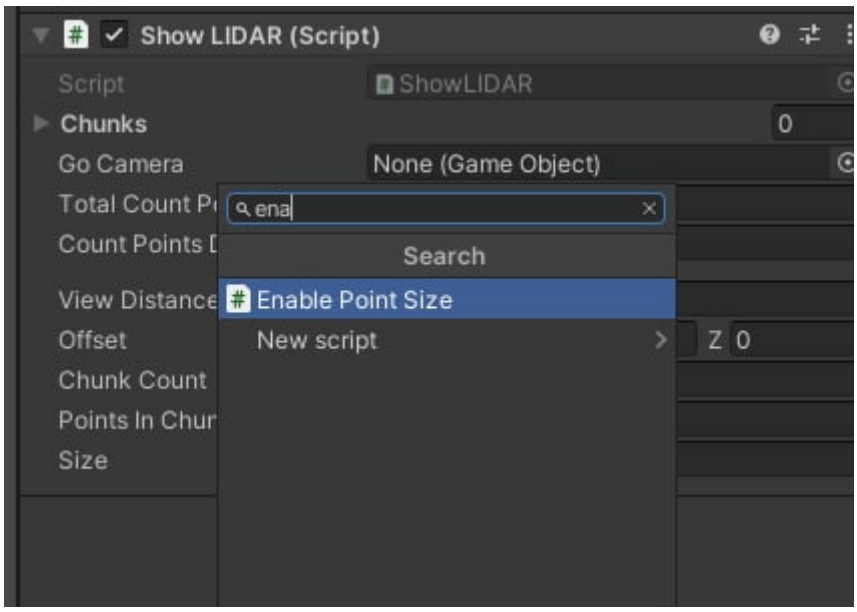

Рисунок 3.5 – Підключення скрипта для збільшення точок

Для налаштування модуля виводу хмар точок потрібно провести певні зміни в компоненті, як на рисунку 3.6.

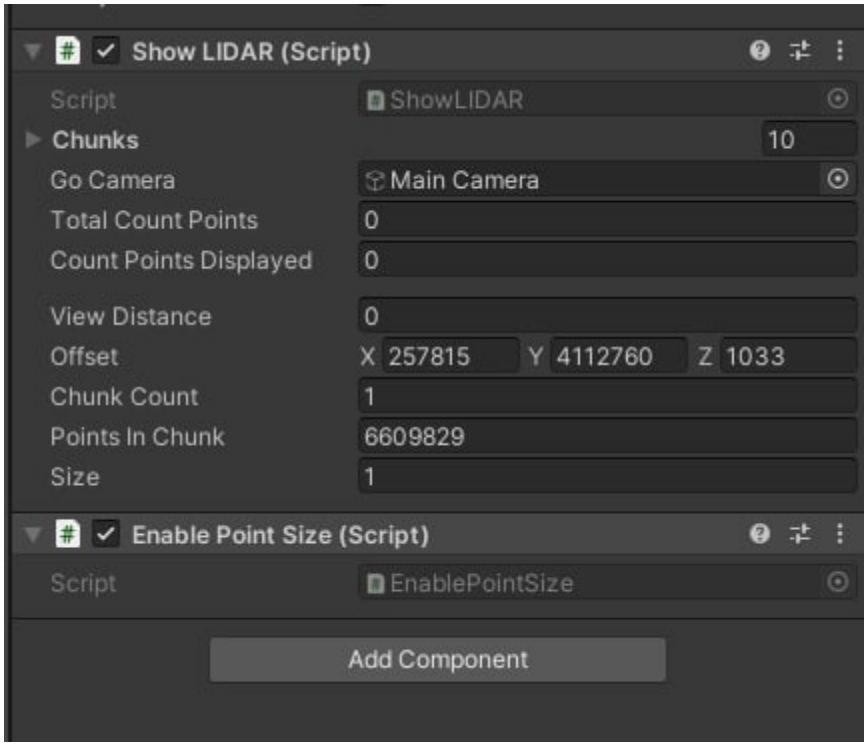

Рисунок 3.6 – Налаштування скрипта виводу хмар точок

3.6 Висновки

У третьому розділі було обґрунтовано вибір мови програмування та технологій для розробки програмного продукту. Проаналізувавши потреби програмного продукту було обрано мову програмування C# та мову GLSL для створення шейдерів. Для відображення візуалізації LIDAR даних використано рушій Unity через встроєний рендер та простоту роботи з графікою. Для зручності управління було створений модуль який дозволяє вільно переміщатись по візуалізації. Було розроблено модуль форматування та виводу LIDAR даних. Також був написаний шейдер для коректного та оптимізованого виводу хмарних точок у просторі.

4 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАСТОСУНКУ

4.1 Вибір методу тестування

Тестування програмного забезпечення – процес перевірки відповідності заявлених до продукту вимог і реально реалізованої функціональності, який здійснюють шляхом спостереження за його роботою в штучно створених ситуаціях і на обмеженому наборі тестів, обраних певним чином.

Тестування можна проводити, як тільки створено виконуваний код (навіть частково завершений). Процес розробки зазвичай передбачає, коли та як буде відбуватися тестування. Наприклад, при поетапному процесі більшість тестів відбувається після визначення системних вимог і тоді вони реалізуються в тестових програмах. На противагу цьому, відповідно до вимог гнучкої розробки ПЗ, програмування і тестування часто відбувається одночасно [17].

Зазвичай тестування поділяють на три категорії.

– функціональне тестування;
– не функціональне тестування або тестування продуктивності;
– технічне обслуговування (регресія та обслуговування).

Тестування програмного забезпечення важливе, оскільки якщо в програмному забезпеченні є помилки, їх можна виявити завчасно та вирішити до доставки програмного продукту. Правильно перевірений програмний продукт забезпечує надійність, безпеку та високу продуктивність, що в подальшому призводить до економії часу, ефективності та задоволення клієнтів.

Переваги використання тестування програмного забезпечення:

– Економічна ефективність: це одна з важливих переваг тестування програмного забезпечення. Своєчасне тестування будь-якого IT-проекту допоможе вам заощадити гроші в довгостроковій перспективі. Якщо помилки виявлені на ранньому етапі тестування програмного забезпечення, їх виправлення коштує дешевше.

- Безпека: це найбільш вразлива і чутлива перевага тестування програмного забезпечення. Люди шукають надійні продукти. Це допомагає усунути ризики та проблеми раніше.
- Якість продукту: це важлива вимога будь-якого програмного продукту. Тестування гарантує, що якісний продукт буде доставлений клієнтам.
- Задоволеність клієнта: головна мета будь-якого продукту - задовольнити своїх клієнтів. Тестування UI/UX забезпечує найкращий досвід користувача.

Тестування програмного забезпечення може визначити правильність програмного забезпечення за припущення деяких конкретних гіпотез, тестування не може визначити всі збої в програмному забезпеченні. Натомість він дає критику або порівняння, що порівнює стан і поведінку продукту з тестовими оракулами — принципами або механізмами, за допомогою яких можна розпізнати проблему. Ці оракули можуть включати специфікації, контракти, порівнянні продукти, попередні версії того самого продукту, висновки про передбачувану або передбачувану мету, очікування користувачів або клієнтів, відповідні стандарти, застосовні закони чи інші критерії.

Оскільки використовується шаблон проектування «Впровадження залежностей» – це дуже спрощує процес тестування. Користувач може в любий момент підставити в інтерфейс любий клас з різними даними та перевірити стійкість модуля до помилок. Для тестування програмного застосунку найбільше підходе модульне тестування.

Модульне тестування – це метод тестування програмного забезпечення, який полягає в окремому тестуванні кожного модуля коду програми. Модулем називають найменшу частину програми, яка може бути протестованою. У процедурному програмуванні модулем вважають окрему функцію або процедуру. В об'єктно-орієнтованому програмуванні – метод [18].

Модульні тести, або unit-тести, розробляють в процесі розробки програмісти та, іноді, тестувальники білої скриньки (white-box testers).

Зазвичай unit-тести застосовують для того, щоб упевнитися, що код відповідає вимогам архітектури та має очікувану поведінку.

Метою модульного тестування є ізоляція кожної частини програми та впевненість у тому, що кожна окрема частина є коректною. Модульний тест забезпечує жорсткий «контракт», за яким має працювати тестований код. Як результат, це надає деякі переваги. Модульне тестування допомагає знайти помилки раніше в циклі розробки ПЗ, що робить розробку дешевшою та швидшою.

Модульне тестування дозволяє програмісту, коли він буде змінювати код (проводити рефакторинг) бути впевненим, що модуль працює вірно (це — регресивне тестування). Оскільки модульне тестування вимагає написання тестів для всіх функцій та методів у програмі, помилки швидко локалізуються та виправляються.

Модульне тестування може бути застосоване в інтеграційному тестуванні: тестування окремих модулів та сукупності цих модулів робить інтеграційне тестування легшим. Однак модульне тестування знизу вгору не є інтеграційним тестуванням. Інтеграція з зовнішніми модулями має включатися до інтеграційних тестів, а не до модульних [19].

Модульні тести являють собою специфічний вид документації до системи. Розробники можуть подивитися на модульний тест, щоб дізнатися про функції, що виконує модуль, та як його застосовувати.

Unit-тест перевіряє критичні характеристики модулю. Відповідність чи невідповідність цим характеристикам демонструє коректність модуля. Модульний тест «документує» ці критичні характеристики, але не треба покладатися лише на код в документуванні ПЗ під час розробки.

Слід відзначити, що звичайна письмова документація дуже повільно реагує на зміни в коді, тоді як модульні тести завжди відображають поточний стан модуля.

4.2 Методика тестування програмного застосунку

Для початку тестування нам потрібно мати дані. LIDAR дані бувають різних форматів але найбільш розповсюджені: LAS та LAZ. Формат файлу LAS – це стандартний формат ASPRS для зберігання даних LiDAR у вигляді точок. Це векторний формат, а не растровий. Формат LAZ ідентичний формату LAS, за винятком стиснення.

Для візуалізації використано файл формату LAS. Його необхідно відкрити в програмі через клавішу Open (див. рис. 4.1).

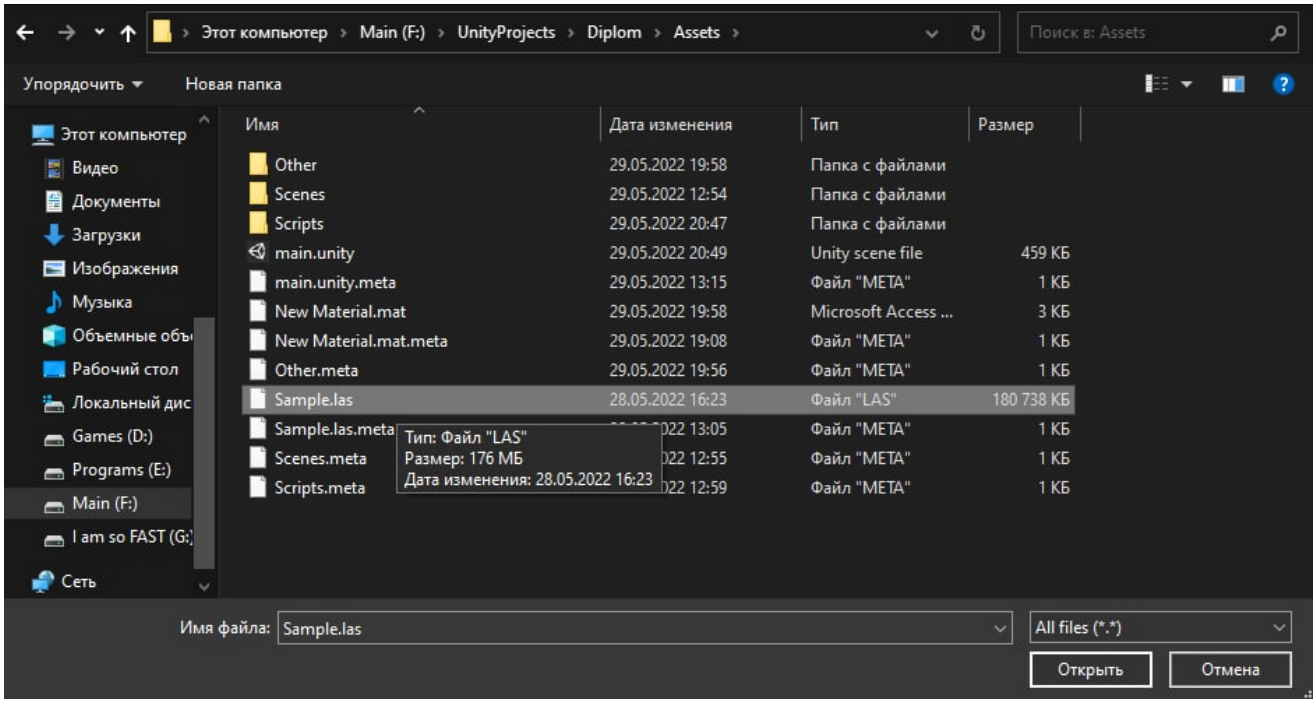

Рисунок 4.1 – Відкриття LIDAR даних

Далі необхідно зачекати доки файл відкриється і завантажиться. Оскільки LIDAR мають великий об’єм, це може зайняти декілька хвилин. Після завантаження, перед користувачем з’явиться візуалізації LIDAR даних (див. рис. 4.2).

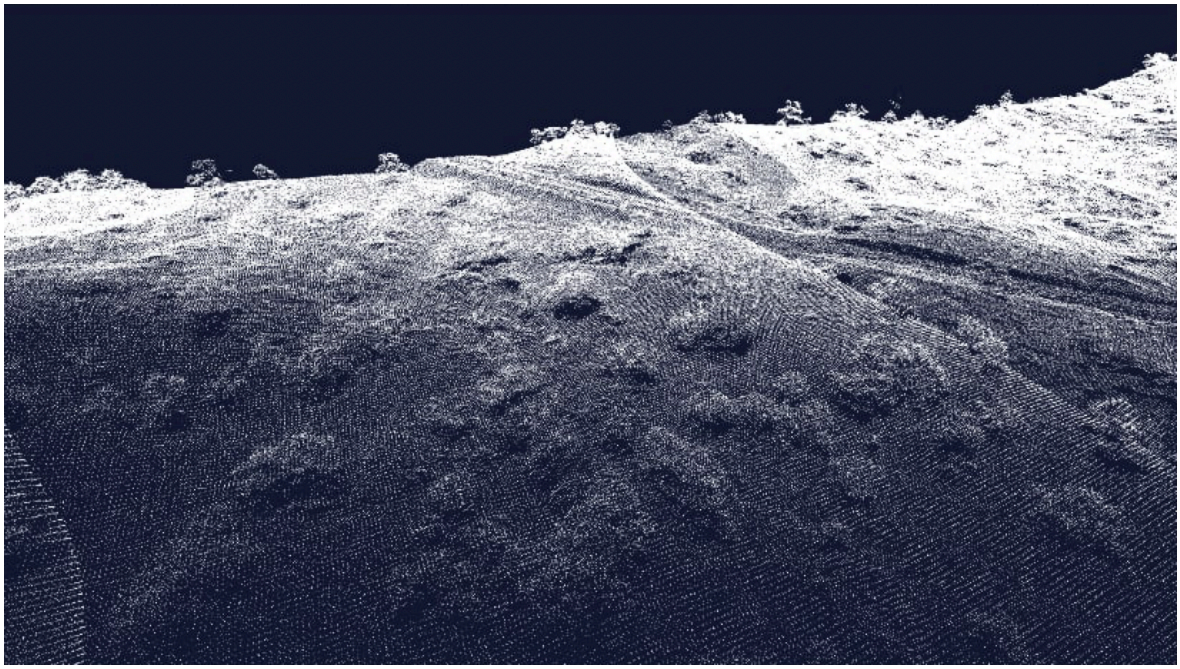

Рисунок 4.2 – Візуалізації LIDAR даних

Оскільки для візуалізації використовується шейдер та блочне відображення, швидкість відтворення та кількість кадрів в секунду на високому показнику. В кадрі може бути 6 мільйонів хмарних точок але рендер показує більше 100 кадрів в секунду (див. рис. 4.3).

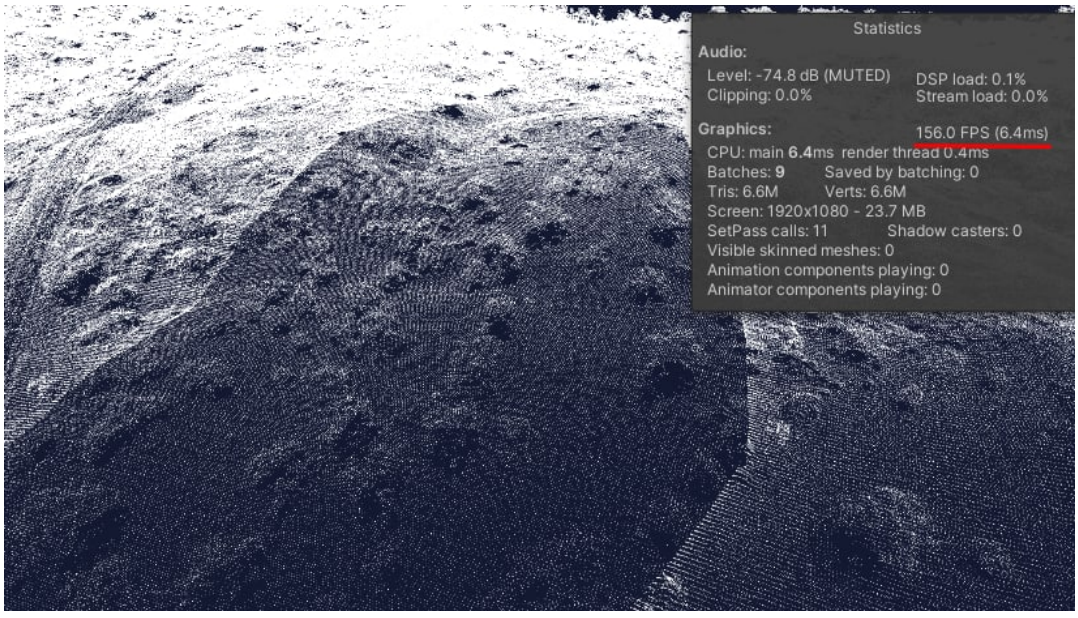

Рисунок 4.3 – Статистика відображення

Для управлінням камерою потрібно використовувати клавіатуру та мишку. Це допоможе змінити погляд на візуалізацію та роздивитись більше дельно хмару точок (див. рис. 4.4).

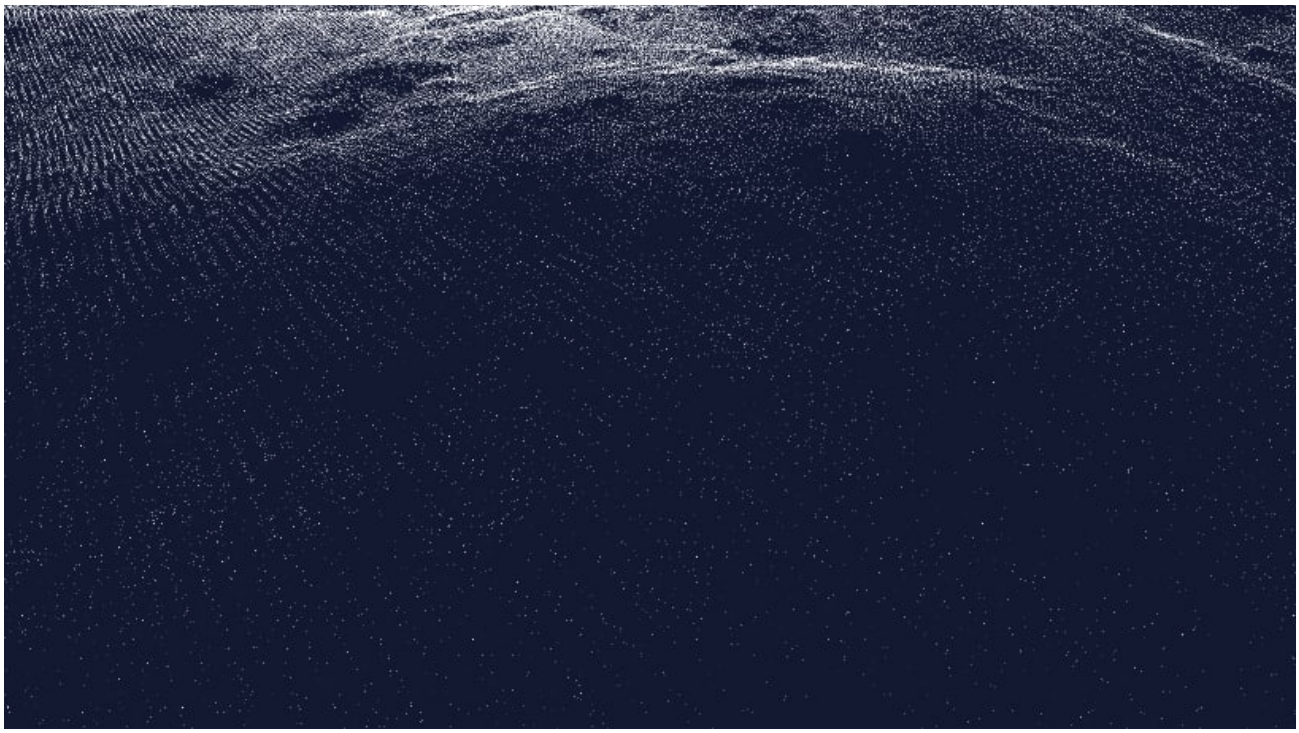

Рисунок 4.4 – Інший погляд на візуалізацію LIDAR даних

Отже, переглянувши отриманий результат можна зробити висновок, що візуалізація LIDAR даних працює коректно та немає проблем які б могли поломати базовий функціонал.

Користувач також має змогу відкрити любий інший файл LIDAR даних та розглянути зручним для себе методом візуалізацію. Оскільки є багато інтернет ресурсів з різними відкритими LIDAR даними, користувач має великий простір для експерементів та перегляду.

4.3 Розробка інструкції користувача

Інструкція користувача передбачає визначення технічних вимог для запуску програмного продукту. Деталі мінімальної та рекомендованої конфігурації серверного комп’ютера представлено в табл. 4.1 та табл. 4.2.

Таблиця 4.1 – Мінімальна конфігурація:

Тип процесора	32-розрядний (x86) або 64-розрядний (x64) процесор з тактовою частотою 2.5 ГГц
Об’єм оперативної пам’яті	3 ГБ для 32-розрядної системи і 16 ГБ для 64-разрядної системи
Місце на жорсткому диску	1 ГБ
Операційна система	Windows 10

Таблиця 4.2 – Рекомендована конфігурація:

Тип процесора	32-розрядний (x86) або 64-розрядний (x64) процесор з тактовою частотою 3.5 ГГц
Об’єм оперативної пам’яті	3 ГБ для 32-розрядної системи і 32 ГБ для 64-разрядної системи
Місце на жорсткому диску	1 ГБ
Операційна система	Windows 10

Для запуску програмного продукту необхідно установити LasTools. Завдяки даній технологій користувач може використовувати дані в графічному інтерфейсі користувача або як пакет для всіх цілей обробки LIDAR у різних версіях. Таким чином користувачу залишається тільки виконати EXE файл запуску програмного забезпечення.

Пошагова інструкція роботи з програмним застосунком:

1) інсталювати застосунок;
2) інсталювати бібліотеки LasTools;
3) відкрити EXE файл застосунку;
4) вибрати файл LIDAR даних;
5) нажати на клавішу «Завантажити».

В результаті він отримає повну візуалізацію LIDAR даних. Для зміни погляду на візуалізацію користувач може використовувати клавіатуру та мишку. Після завершення дослідження та перегляду хмарних точок, користувач може закрити програмний застосунок.

4.4 Висновки

У четвертому розділі було проведено тестування програмного застосунку для візуалізації LIDAR даних. Був проведений аналіз методу тестування та протестовано програмне забезпечення на швидкість виконання та коректність роботи. Також було розроблено інструкцію користувача по встановленню та початковій експлуатації програмного застосунку.

ВИСНОВКИ

У бакалаврській дипломній роботі було розроблено програмний застосунок блочної візуалізації LIDAR даних. Для розробки було використано рушій Unity та середовище програмної розробки JetBrains Rider.

Під час виконання бакалаврської дипломної роботи проведено аналіз стану проблеми на сьогоднішній день, аналіз аналогів програмного продукту. Визначено недоліки систем візуалізації LIDAR даних та порівняно з власним програмним продуктом. Базуючись на порівнянні було встановлено основні задачі бакалаврської дипломної роботи.

Під час аналізу технологій розробки обґрунтовано вибір мови програмування C# та GLSL для створення власного шейдера. Також було розглянуто переваги використання Unity для рендеринга візуалізації.

У першому розділі розглянуто стан технології LIDAR, проведено аналіз інструментів для візуалізації даних. Проведено аналіз переваги та недоліки відомих систем візуалізації. Доведено доцільність розробки власного програмного рішення. На основі отриманої інформації було сформовано перелік задач, які необхідно виконати для розробки власного програмного застосунку

У другому розділі проведено аналіз LIDAR даних. Розглянуто методи зчитування та форматування LAS та LAZ файлів з тривимірними хмарами точок. Розроблено алгоритм роботи блочної візуалізації LIDAR даних. Також було розроблено метод блочного відображення для опимізації роботи програмного застосунку. Розроблено модель блочної візуалізації LIDAR даних за допомогою UML діаграми класів.

У третьому розділі обґрунтовано вибір мови програмування та технологій для розробки програмного продукту. Проаналізувавши потреби програмного продукту було обрано мову програмування C# та мову GLSL для створення шейдерів.

У четвертому розділі проведено тестування програмного застосунку для візуалізації LIDAR даних. Був проведений аналіз методу тестування та протестовано програмне забезпечення на швидкість виконання та коректність роботи. Розроблено інструкцію користувача по встановленню та початковій експлуатації програмного застосунку

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

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Верчинський М.О. Аналіз засобів візуалізації LIDAR даних / Електронні інформаційні ресурси: створення, використання, доступ. Пам'яті Олексія Петровича Стахова. Збірник матеріалів Міжнародної науково-практичної Інтернет конференції 9-10 листопада 2021 р. – Суми/Вінниця: НІКО/ВНТУ, 2021.
2. What is lidar technology [Електронний ресурс] – Режим доступу до ресурсу: https://www.geospatialworld.net /what-is-lidar-technology.
3. LiDAR [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Lidar.
4. Online LIDAR point cloud viewer [Електронний ресурс] – Режим доступу до ресурсу: http://lidarview.com.
5. Войтко В.В. Розробка спеціалізованого веб-ресурсу для професійного відбору кандидатів / В.В. Войтко., С.В. Бевз, С.М. Бурбело, Д.П. Паламарчук // Матеріали міжнародної науково-технічної конференції «Інформаційно-комп'ютерні технології - 2021». – Житомир: Житомирська політехніка, 2021. – 205 с.
6. Displaz [Електронний ресурс] – Режим доступу до ресурсу: https://c42f.github.io/displaz/.
7. Plas.io [Електронний ресурс] – Режим доступу до ресурсу: https://plas.io.
8. LasTools [Електронний ресурс] – Режим доступу до ресурсу: http://lastools.org.
9. LAS, LAZ formats [Електронний ресурс] – Режим доступу до ресурсу: https://manifold.net/doc/mfd9/las,_laz_lidar.html.
10. Lidar сканери [Електронний ресурс] – Режим доступу до ресурсу: https://yablyk.com/940169-what-is-the-lidar-scanner-for/.

11. Lidar formats [Електронний ресурс] – Режим доступу до ресурсу: https://library.carleton.ca/guides/help/lidar-formats.
12. Lidar data [Електронний ресурс] – Режим доступу до ресурсу: https://library.carleton.ca/find/gis/lidar-data.
13. Andrew Troelsen Pro C# 8 with .NET Core 3 : навч. посіб. / Andrew Troelsen, Phil Japikse – Apress, 2020. – 1881 ст.
14. Краткий обзор языка C# [Електронний ресурс] – Режим доступу до ресурсу: https://docs.microsoft.com/ru-ru/dotnet/csharp/tour-of-csharp/.
15. Шейдер [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Шейдер.
16. GLSL shader [Електронний ресурс] – Режим доступу до ресурсу: https://developer.mozilla.org/ru/docs/Games/Techniques/3D_on_the_web/GLSL_Shaders.
17. Тестування програмного забезпечення [Електронний ресурс] – Режим доступу до ресурсу: https://www.quality-assurance-group.com/shho-take-testuvannya-programnogo-zabezpechennya-ta-yake-jogo-znachennya/.
18. Модульне тестування [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Модульне_тестування.
19. Test case [Електронний ресурс] – Режим доступу до ресурсу: https://www.guru99.com/test-case.html.

ДОДАТКИ

ДОДАТОК А

Міністерство освіти і науки України

Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

ЗАТВЕРДЖУЮ

д.т.н., проф. О. Н. Романюк

31 березня 2022 р.

Технічне завдання

на бакалаврську дипломну роботу

«Розробка програмного застосунку блочної візуалізації LIDAR даних»

за спеціальністю

121 – Інженерія програмного забезпечення

Керівник бакалаврської дипломної роботи:

_____ к.т.н., доц. О.М. Рейда

31 березня 2022 р.

Виконав:

_____студент гр. 1ПІ-19мс2 М.О. Верчинький

31 березня 2022 р.

Вінниця – 2022 року

1. Найменування та галузь застосування

Бакалаврська дипломна робота: «Розробка програмного застосунку блочної візуалізації LIDAR даних».

Галузь застосування – наукові дослідження.

2. Підстава для розробки.

Підставою для виконання бакалаврської дипломної роботи (БДР) є індивідуальне завдання на БДР та наказ №66 від 24 березня 2022 року ректора по ВНТУ про закріплення тем БДР.

3. Мета та призначення розробки.

Метою роботи є удосконалення процесу візуалізації LIDAR даних шляхом пришвидчення роботи та оптимізації працездатності за допомогою блочного відображення та шейдерного рендеринга.

Призначення роботи – пришвидчення та оптимізації працездатності процесу візуалізації LIDAR даних.

4. Технічні вимоги

Вхідні дані – LIDAR дані формату LAS або LAZ; вихідні дані – графічна візуалізації хмар точок.

5. Конструктивні вимоги.

Конструкція пристрою повинна відповідати естетичним та ергономічним вимогам, повинна бути зручною в обслуговуванні та керуванні.

Графічна та текстова документація повинна відповідати діючим стандартам України.

6. Перелік технічної документації, що пред’являється по закінченню робіт:

– пояснювальна записка до бакалаврської дипломної роботи;

– технічне завдання;

– лістинги програми.

7. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

8. Стадії та етапи розробки:

№ з/п	Назва етапів бакалаврської дипломної роботи	Строк виконання етапів роботи
1	Аналіз проблеми, обґрунтування актуальності розробки системи та постановка задачі	26.03.2022 – 10.04.2022
2	Розробка архітектури та алгоритмів роботи системи	11.04.2022 – 19.04.2022
3	Вибір середовища та мови розробки	20.04.2022 – 27.04.2022
4	Розробка програмного продукту	28.04.2022 – 10.05.2022
5	Тестування роботи системи	11.05.2022 – 19.05.2022
6	Оформлення матеріалів до захисту БДР	20.05.2022 – 10.06.2022

9. 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 КВАЛІФІКАЦІЙНОЇ РОБОТИ
НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: Розробка програмного застосунку блочної візуалізації LIDAR даних

Тип роботи: БДР

Підрозділ : кафедра програмного забезпечення, ФІТКІ

Науковий керівник: Рейда О. М.

Оригінальність	83,9%
Схожість	16,1%

Аналіз звіту подібності

■ **Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.**

□ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її автора. Роботу направити на доопрацювання.

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

Особа, відповідальна за перевірку _____ Черноволик Г. О.

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck

Автор роботи _____ Верчинький М.О.

Керівник роботи _____ Рейда О. М.

ДОДАТОК В

CameraControl.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

/// <summary>
/// Camera control.
/// Control Camera with different behaviour for Android and Unity Editor.
/// </summary>
public class CameraControl : MonoBehaviour {


  #if UNITY_ANDROID && !UNITY_EDITOR
  Vector2?[] oldTouchPositions = { null, null };
  Vector2 oldTouchVector;
  float oldTouchDistance;
  private Camera camera;

  void Start(){
    camera = Camera.main;
    if (camera.orthographic == false) {
      camera.orthographic = true;
    }
  }
  #endif

  #if UNITY_EDITOR
  public int speed = 50;
  #endif

  void Update() {
    #if UNITY_ANDROID && !UNITY_EDITOR
    if (Input.touchCount == 0) {
      oldTouchPositions[0] = null;
      oldTouchPositions[1] = null;
    }
    else if (Input.touchCount == 1) {
    if (oldTouchPositions[0] == null || oldTouchPositions[1] != null) {
      oldTouchPositions[0] = Input.GetTouch(0).position;
      oldTouchPositions[1] = null;
    }
    else {
      Vector2 newTouchPosition = Input.GetTouch(0).position;

      transform.position += transform.TransformDirection((Vector3)((oldTouchPositions[0] -
newTouchPosition) * camera.orthographicSize / camera.pixelHeight * 2f));

      oldTouchPositions[0] = newTouchPosition;
    }
    }
    else {
      if (oldTouchPositions[1] == null) {
        oldTouchPositions[0] = Input.GetTouch(0).position;
        oldTouchPositions[1] = Input.GetTouch(1).position;
        oldTouchVector = (Vector2)(oldTouchPositions[0] - oldTouchPositions[1]);
        oldTouchDistance = oldTouchVector.magnitude;
```

```
      }
      else {
        Vector2 screen = new Vector2(camera.pixelWidth, camera.pixelHeight);

        Vector2[] newTouchPositions = {
          Input.GetTouch(0).position,
          Input.GetTouch(1).position
        };
        Vector2 newTouchVector = newTouchPositions[0] - newTouchPositions[1];
        float newTouchDistance = newTouchVector.magnitude;

        transform.position += transform.TransformDirection((Vector3)((oldTouchPositions[0] +
oldTouchPositions[1] - screen) * camera.orthographicSize / screen.y));
        transform.localRotation *= Quaternion.Euler(new Vector3(0, 0,
Mathf.Asin(Mathf.Clamp((oldTouchVector.y * newTouchVector.x - oldTouchVector.x *
newTouchVector.y) / oldTouchDistance / newTouchDistance, -1f, 1f)) / 0.0174532924f));
        camera.orthographicSize *= oldTouchDistance / newTouchDistance;
        transform.position -= transform.TransformDirection((newTouchPositions[0] +
newTouchPositions[1] - screen) * camera.orthographicSize / screen.y);

        oldTouchPositions[0] = newTouchPositions[0];
        oldTouchPositions[1] = newTouchPositions[1];
        oldTouchVector = newTouchVector;
        oldTouchDistance = newTouchDistance;
      }
    }
    #endif

    #if UNITY_EDITOR
    if(Input.GetKey(KeyCode.D))
    {
      transform.Translate(new Vector3(speed * Time.deltaTime,0,0));
    }
    if(Input.GetKey(KeyCode.A))
    {
      transform.Translate(new Vector3(-speed * Time.deltaTime,0,0));
    }
    if(Input.GetKey(KeyCode.S))
    {
      transform.Translate(new Vector3(0,-speed * Time.deltaTime,0));
    }
    if(Input.GetKey(KeyCode.W))
    {
      transform.Translate(new Vector3(0,speed * Time.deltaTime,0));
    }
    if(Input.GetKey(KeyCode.LeftAlt))
    {
      transform.Translate(new Vector3(0,0,-speed * Time.deltaTime));
    }
    if(Input.GetKey(KeyCode.LeftControl))
    {
      transform.Translate(new Vector3(0,0,speed * Time.deltaTime));
    }
    #endif
  }
}
```

EnablePointSize.cs

```
#if UNITY_STANDALONE
```

```
#define IMPORT_GLENABLE
#endif

using UnityEngine;
using System;
using System.Collections;
using System.Runtime.InteropServices;

/// <summary>
/// Enable point size.
/// This script allows you to modify the size of the points within the shader.
/// </summary>
public class EnablePointSize : MonoBehaviour
{
    const UInt32 GL_VERTEX_PROGRAM_POINT_SIZE = 0x8642;
    const UInt32 GL_POINT_SMOOTH = 0x0B10;

    const string LibGLPath =
    #if UNITY_STANDALONE_WIN
        "opengl32.dll";
    #elif UNITY_STANDALONE_OSX
      "/System/Library/Frameworks/OpenGL.framework/OpenGL";
    #elif UNITY_STANDALONE_LINUX
      "libGL";  // Untested on Linux, this may not be correct
    #else
      null;   // OpenGL ES platforms don't require this feature
    #endif

    #if IMPORT_GLENABLE
    [DllImport(LibGLPath)]
    public static extern void glEnable(UInt32 cap);

    private bool mIsOpenGL;

    void Start()
    {
        mIsOpenGL = SystemInfo.graphicsDeviceVersion.Contains("OpenGL");

    Debug.Log (SystemInfo.graphicsDeviceName + " : " + SystemInfo.graphicsDeviceVersion);
    }

    void OnPreRender()
    {
    if (mIsOpenGL) {
      glEnable (GL_VERTEX_PROGRAM_POINT_SIZE);
      glEnable (GL_POINT_SMOOTH);
    }
    }
    #endif
}
```

FlyCamare.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class FlyCamera : MonoBehaviour
{
  public float mainSpeed = 100.0f; //regular speed
  public float shiftAdd = 250.0f; //multiplied by how long shift is held.  Basically running
```

```
  public float maxShift = 1000.0f; //Maximum speed when holdin gshift
  public float camSens = 0.25f; //How sensitive it with mouse
  public bool rotateOnlyIfMousedown = true;
  public bool movementStaysFlat = true;

  private Vector3 lastMouse = new Vector3(255, 255, 255); //kind of in the middle of the
screen, rather than at the top (play)
  private float totalRun= 1.0f;

  void Awake() {
    Debug.Log ("FlyCamera Awake() - RESETTING CAMERA POSITION"); // nop?
    // nop:
    //transform.position.Set(0,8,-32);
    //transform.rotation.Set(15,0,0,1);
    //transform.position = new Vector3(0,8,-32);
  }


  void Update () {

    if (Input.GetMouseButtonDown(1))
    {
      lastMouse = Input.mousePosition; // $CTK reset when we begin
    }

    if (!rotateOnlyIfMousedown
        (rotateOnlyIfMousedown && Input.GetMouseButton(1)))
    {
      lastMouse = Input.mousePosition - lastMouse ;
      lastMouse = new Vector3(-lastMouse.y * camSens, lastMouse.x * camSens, 0 );
      lastMouse = new Vector3(transform.eulerAngles.x + lastMouse.x , transform.eulerAngles.y
+ lastMouse.y, 0);
      transform.eulerAngles = lastMouse;
      lastMouse =  Input.mousePosition;
      //Mouse  camera angle done.
    }

    //Keyboard commands
    float f = 0.0f;
    Vector3 p = GetBaseInput();
    if (Input.GetKey (KeyCode.LeftShift)){
      totalRun += Time.deltaTime;
      p  = p * totalRun * shiftAdd;
      p.x = Mathf.Clamp(p.x, -maxShift, maxShift);
      p.y = Mathf.Clamp(p.y, -maxShift, maxShift);
      p.z = Mathf.Clamp(p.z, -maxShift, maxShift);
    }
    else{
      totalRun = Mathf.Clamp(totalRun * 0.5f, 1f, 1000f);
      p = p * mainSpeed;
    }

    p = p * Time.deltaTime;
    Vector3 newPosition = transform.position;
    if (Input.GetKey(KeyCode.Space)
        (movementStaysFlat && !(rotateOnlyIfMousedown && Input.GetMouseButton(1)))){ //If
player wants to move on X and Z axis only
      transform.Translate(p);
      newPosition.x = transform.position.x;
      newPosition.z = transform.position.z;
      transform.position = newPosition;
    }
    else{
      transform.Translate(p);
```

```
    }

  }

  private Vector3 GetBaseInput() { //returns the basic values, if it's 0 than it's not active.
    Vector3 p_Velocity = new Vector3();
    if (Input.GetKey (KeyCode.W)){
      p_Velocity += new Vector3(0, 0 , 1);
    }
    if (Input.GetKey (KeyCode.S)){
      p_Velocity += new Vector3(0, 0, -1);
    }
    if (Input.GetKey (KeyCode.A)){
      p_Velocity += new Vector3(-1, 0, 0);
    }
    if (Input.GetKey (KeyCode.D)){
      p_Velocity += new Vector3(1, 0, 0);
    }
    return p_Velocity;
  }
}
```

VertexColor.cs

```
// Upgrade NOTE: replaced 'mul(UNITY_MATRIX_MVP,*)' with 'UnityObjectToClipPos(*)'
Shader "Custom/VertexColor" {
  Properties {
    _PointSize("PointSize", Float) = 50
  }
    SubShader {
    Pass {
        LOD 500

        CGPROGRAM
        #pragma vertex vert
        #pragma fragment frag

        struct VertexInput {
            float4 v : POSITION;
            float4 color: COLOR;
        };

        struct VertexOutput {
            float4 pos : SV_POSITION;
            float size : PSIZE;
            float4 col : COLOR;
        };

        float _PointSize;

        VertexOutput vert(VertexInput v) {

            VertexOutput o;
            o.pos = UnityObjectToClipPos(v.v);
            o.size = _PointSize;
            o.col = v.color;

            return o;
        }

        float4 frag(VertexOutput o) : COLOR {
            return o.col;
```

```
        }

        ENDCG
        }
    }

}
```

ShowLidar.cs

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.Linq;
using laszip.net;
using UnityEngine;


[RequireComponent(typeof(MeshFilter), typeof(MeshRenderer))]
public class ShowLIDAR : MonoBehaviour
{
    [SerializeField] private List<Chunk> chunks;
    [SerializeField] private GameObject  goCamera;
    [SerializeField] private uint        totalCountPoints;
    [SerializeField] private uint        countPointsDisplayed;
    [Space]
    [SerializeField] private int viewDistance = 10;
    [SerializeField] private Vector3 offset        = Vector3.zero;
    [SerializeField] private int     chunkCount    = 10;
    [SerializeField] private int     pointsInChunk = 128;
    [SerializeField] private float   size          = 1.0f;


    private Mesh   mesh;
    private string path = $"F:/UnityProjects/Diplom/Assets/Sample.las";

    IEnumerator Start()
    {
        countPointsDisplayed = (uint)( chunkCount * pointsInChunk );

        chunks = new List<Chunk>( chunkCount );

        laszip_dll lasReader  = new laszip_dll();
        bool       compressed = true;
        lasReader.laszip_open_reader(path, ref compressed);
        totalCountPoints = lasReader.header.number_of_point_records;

        double[] coordArray = new double[3];

        for ( int i = 0; i < chunkCount; i++ )
        {
            LoadChunk( lasReader, coordArray );
            yield return null;
        }

        lasReader.laszip_close_reader();

        //goCamera.transform.localPosition = chunks[0].chunkPos;

        SetMesh();
    }
```

```
    private void LoadChunk( laszip_dll lasReader, double[] coordArray )
    {
        Chunk chunk = new Chunk();
        chunk.points = new Vector3[pointsInChunk];

        for (int pointIndex = 0; pointIndex < pointsInChunk; pointIndex++)
        {
            lasReader.laszip_read_point();
            lasReader.laszip_get_coordinates(coordArray);

            chunk.points[pointIndex] = new Vector3( (float)( coordArray[0] - offset.x ),
(float)( coordArray[1] - offset.y ), (float)( coordArray[2] - offset.z ) );
        }

        chunk.chunkPos = new Vector3(
            chunk.points.Average( point => point.x ),
            chunk.points.Average( point => point.y ),
            chunk.points.Average( point => point.z )
        );

        chunks.Add( chunk );
    }

    private void SetMesh()
    {
        mesh = new Mesh();

        GetComponent<MeshFilter>().mesh = mesh;
        GetComponent<MeshRenderer> ().material = new Material
(Shader.Find("Custom/VertexColor"));
        CreateMesh();
    }

    private void CreateMesh()
    {
        Chunk chunk       = chunks[0];
        int   countPoints = chunks[0].points.Length;

        int[]     indecies = new int[countPoints];
        Color[]   colors   = new Color[countPoints];

        for ( int i = 0; i < chunk.points.Length; ++i )
        {
            indecies[i] = i;
            colors[i] = Color.white;
        }

        mesh.indexFormat = UnityEngine.Rendering.IndexFormat.UInt32;
        mesh.vertices = chunk.points;
        mesh.colors = colors;
        mesh.SetIndices( indecies, MeshTopology.Points, 0 );
    }
}

[Serializable]
public struct Chunk
{
    public Vector3[] points;
    public Vector3   chunkPos;
}

[Serializable]
public struct Point3D
```

```
{
    public Vector3 pos;
    public ushort intensity;
}
```

ДОДАТОК Г

ГРАФІЧНА ЧАСТИНА

РОЗРОБКА ПРОГРАМНОГО ЗАСТОСУНКУ БЛОЧНОЇ ВІЗУАЛІЗАЦІЇ LIDAR ДАНИХ

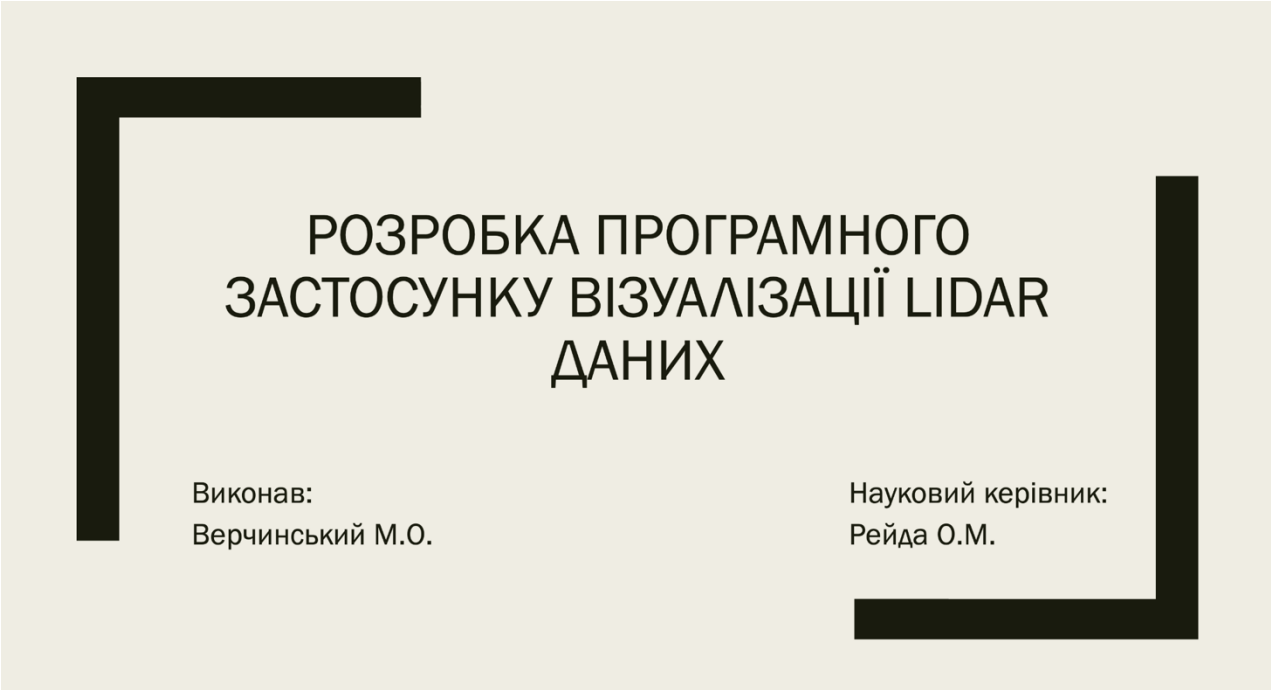

Рисунок В.1 – Назва роботи

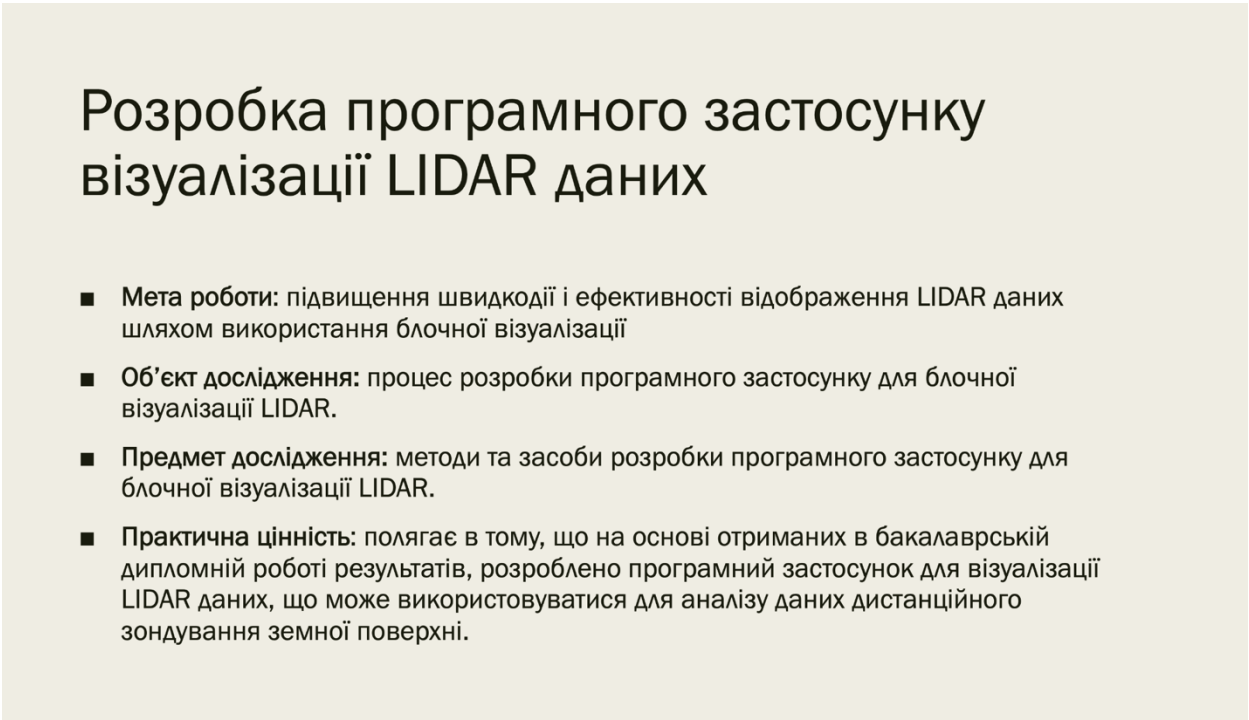

Рисунок В.2 – Мета, об'єкт і предмет дослідження

Рисунок В.3 – Наукова новизна та практична цінність

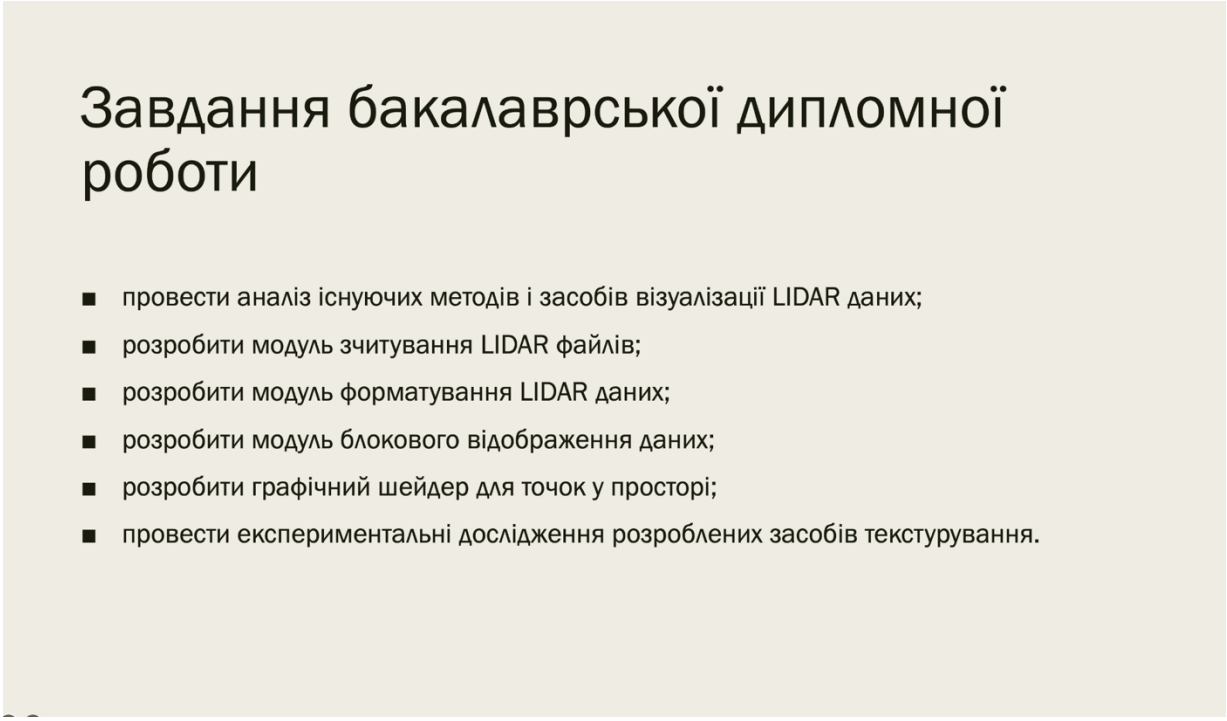

Рисунок В.4 – Структура бакалаврської дипломної роботи

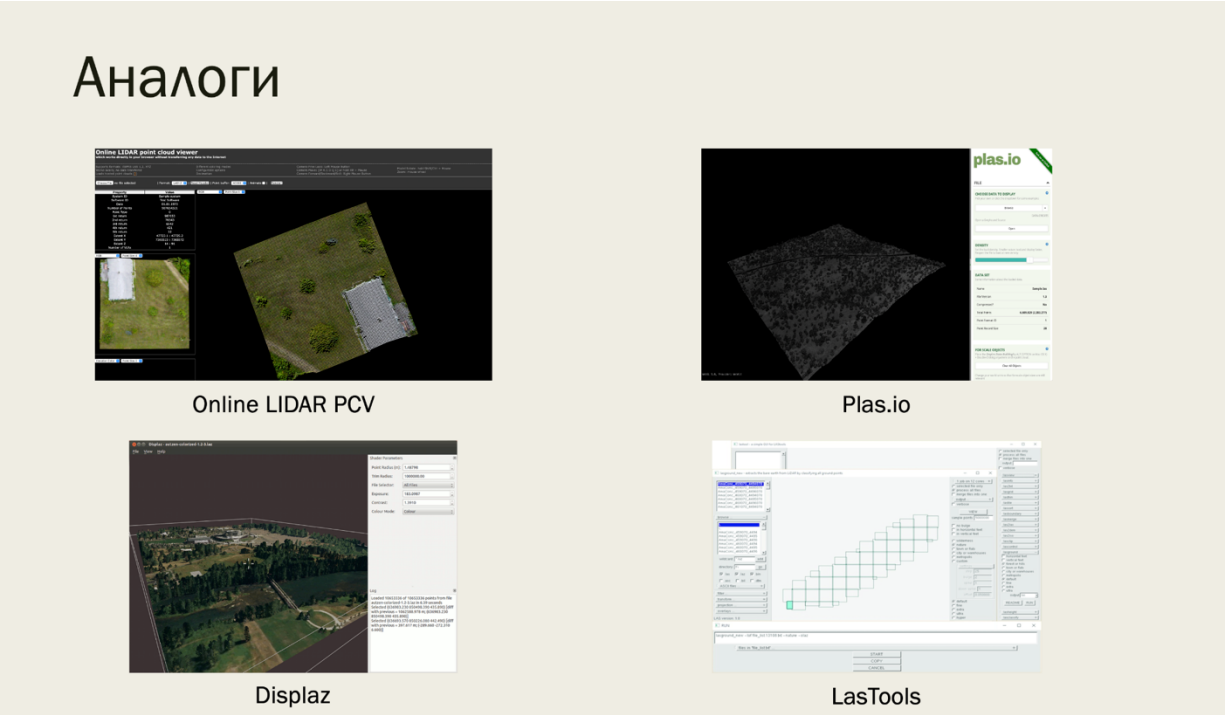

Рисунок В.5 – Аналоги

Критерії	Online LIDAR PCV	Plas.io	Displaz	LasTools	FastLasVisualize
Наявність веб версії	1	1	0	0	0
Підтримка відображення точок через шейдер	1	0	1	0	1
Підтримка LAS та LAZ форматів	0	0	0	1	1
Можливість запуску на мобільних девайсах	0	0	1	0	1
Підтримка блочної візуалізації	0	0	1	1	1
Сумарний коефіцієнт	2	1	3	2	4

Рисунок В.6 – Порівняльний аналіз

Блок-схема загального алгоритму програми

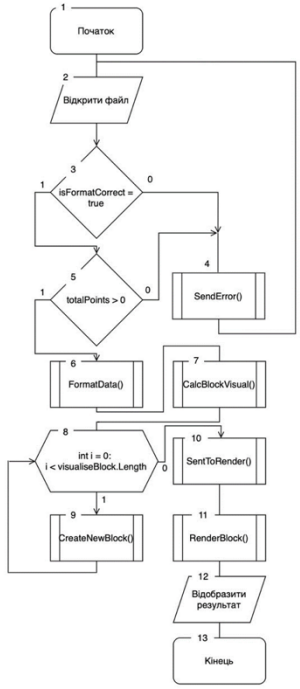

Рисунок В.7 – Блок-схема

Метод зчитування та форматування LIDAR файлів

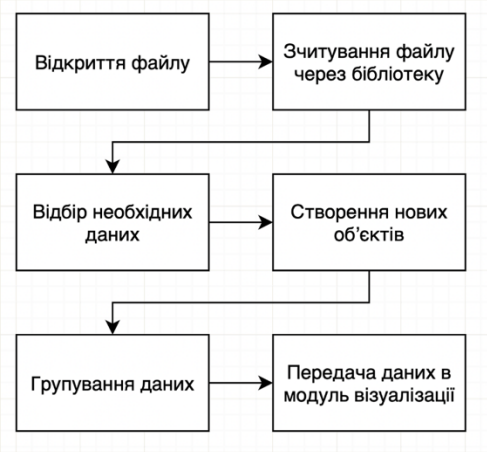

Рисунок В.8 – Методи зчитування

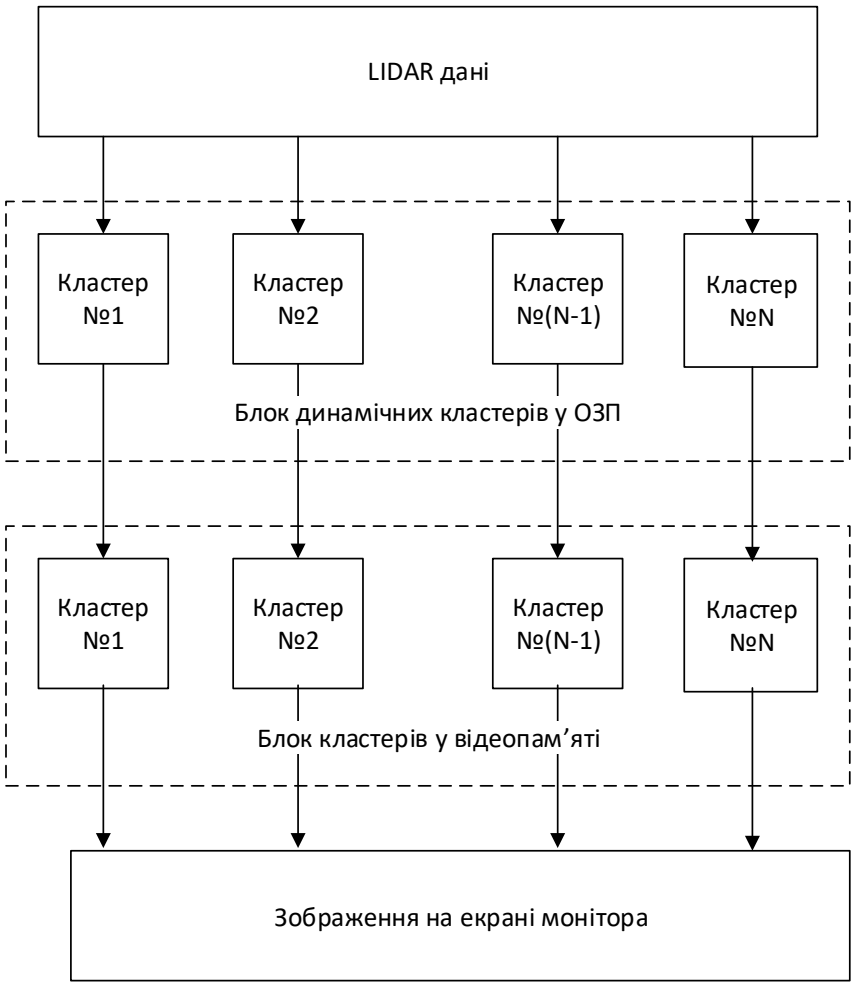

Рисунок В.9 – Схема відображення LIDAR даних

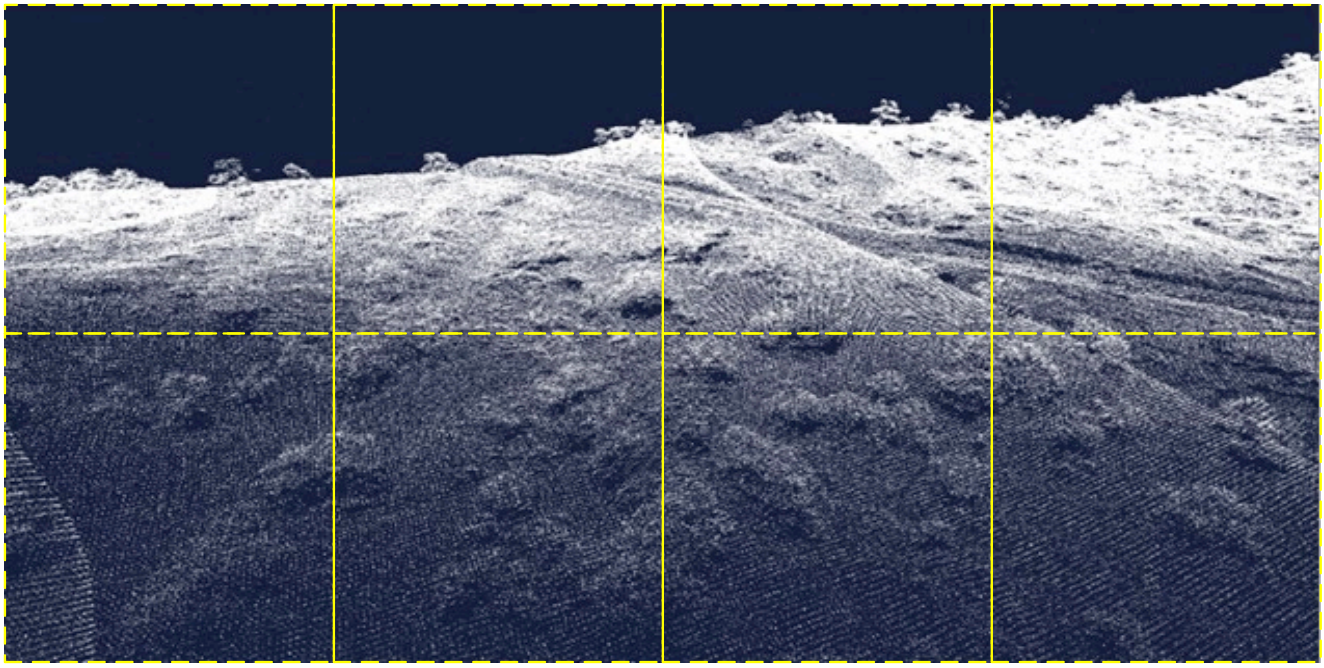

Рисунок В.10 – Схема розташування кластерів у відеопам’яті

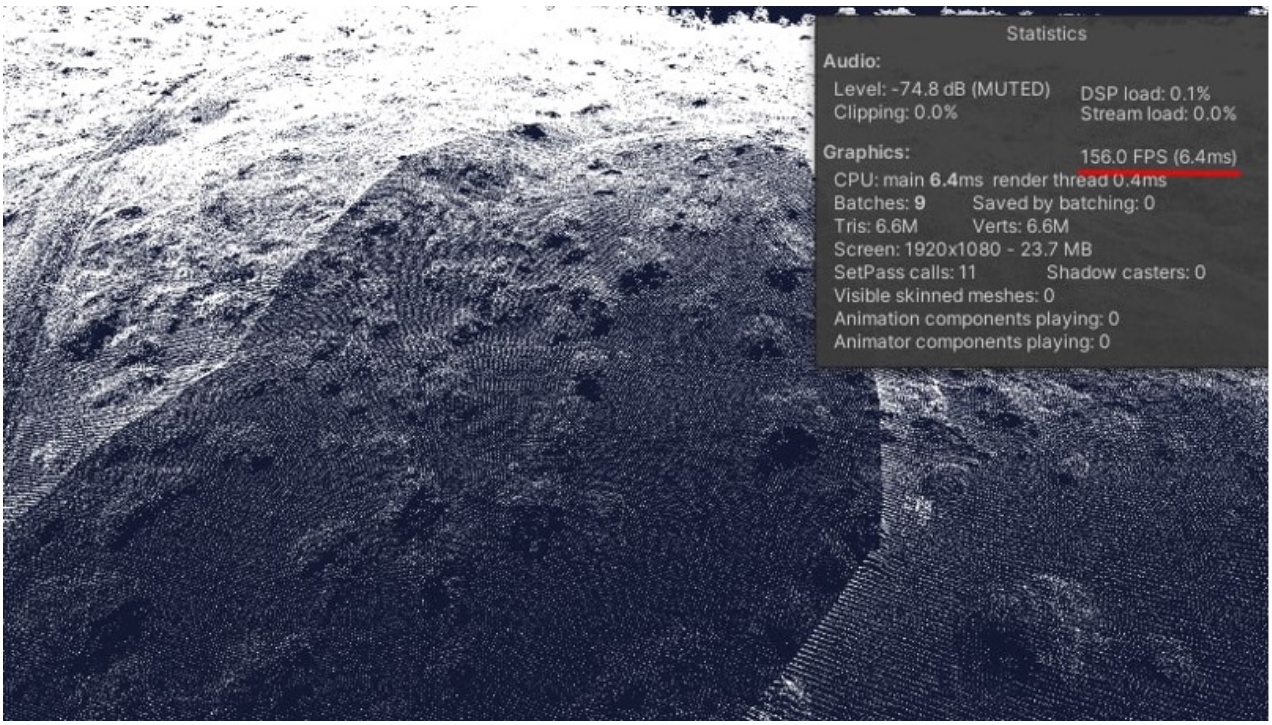

Рисунок В.11 – Тестування застосунку: Статистика

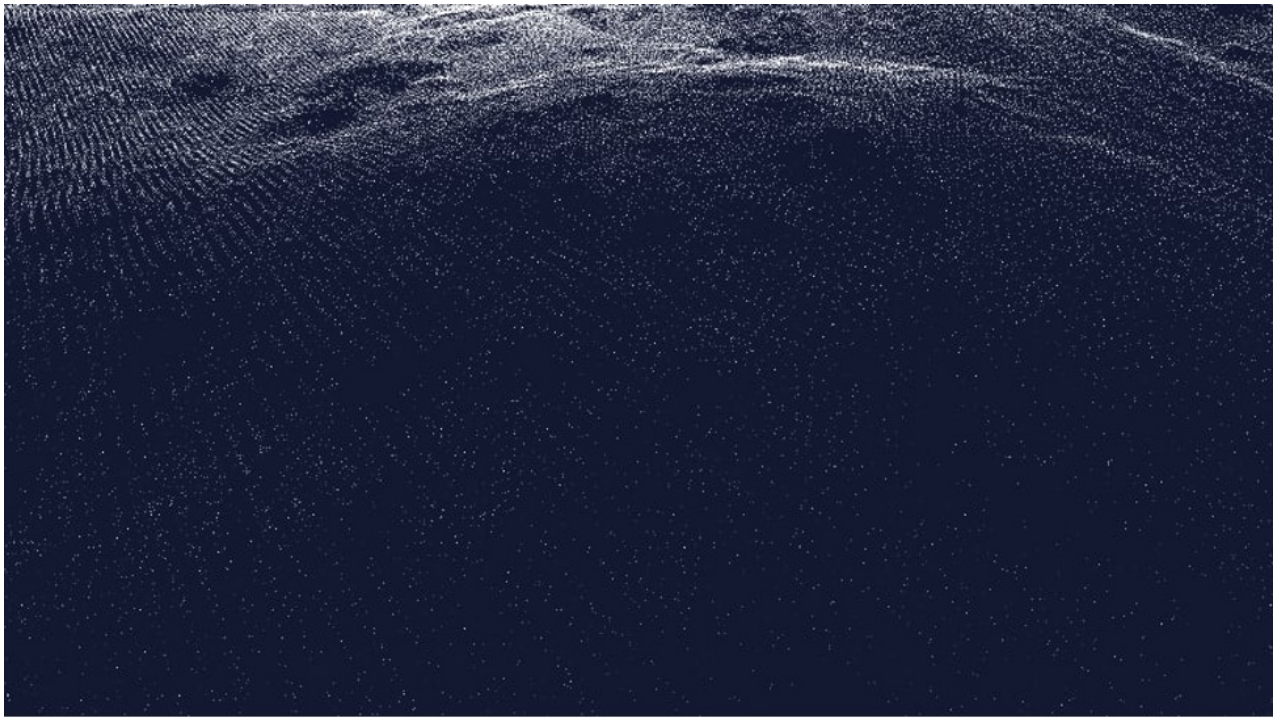

Рисунок В.12 – Тестування застосунку: Інший погляд на візуалізації

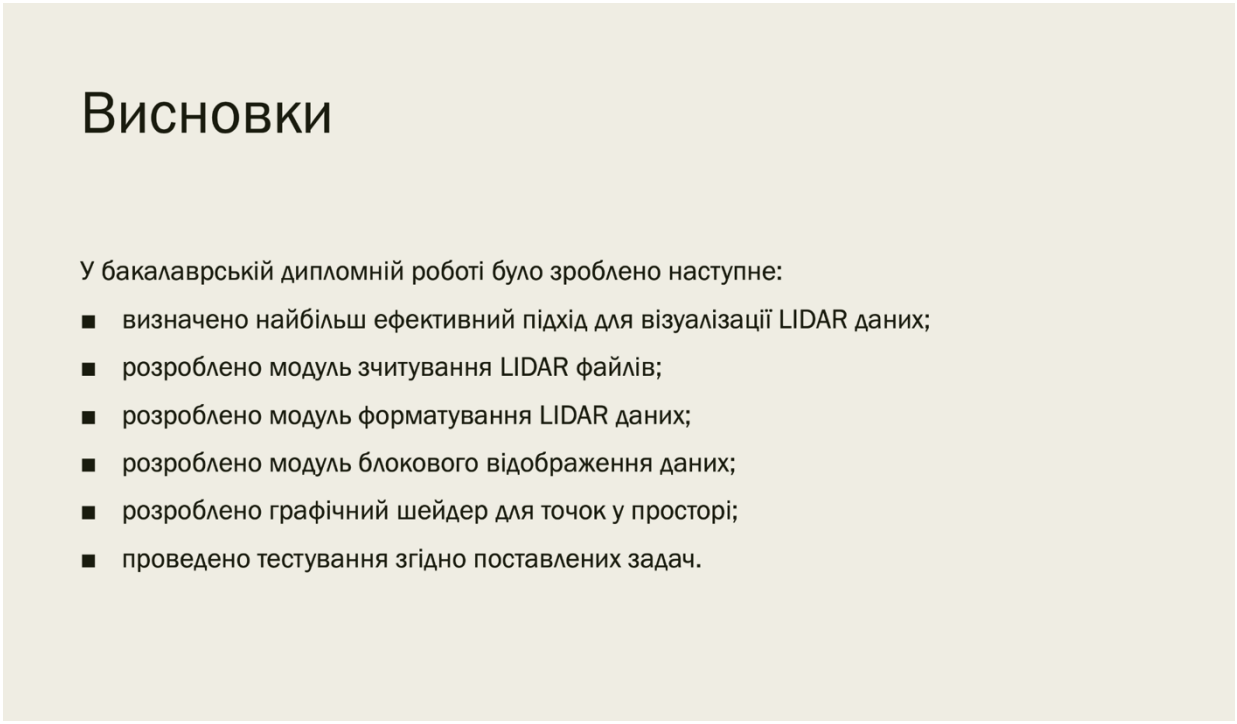

Рисунок В.13 – Висновки

Рисунок В.14 – Фінальний плакат