

**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ НАЦІОНАЛЬНИЙ АГРАРНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ ЕКОНОМІКИ І МЕНЕДЖМЕНТУ**

Кафедра кібернетики та інформатики

КВАЛІФІКАЦІЙНА РОБОТА

освітній ступінь - «Бакалавр»

на тему: **“РОЗРОБКА ГЕНЕРАТОРІВ 3D КОНТЕНТУ
НА ПРИКЛАДІ ГЕНЕРАТОРУ МЕША РЕЛЬЄФУ
МІСЦЕВОСТІ”**

Виконав:

студент спеціальності

126 «Інформаційні системи та технології»

Ковальов Роман Станіславович

Керівник:

В'юненко Олександр Борисович

к.е.н., доцент кафедри кібернетики та інформатики

Рецензент:

Ободяк Віктор Корнелійович

к.т.н., доцент кафедри кібербезпеки СумДУ

Суми – 2024

СУМСЬКИЙ НАЦІОНАЛЬНИЙ АГРАРНИЙ УНІВЕРСИТЕТ

Факультет	Економіки і менеджменту
Кафедра	Кібернетики та інформатики
Освітній ступінь	«Бакалавр»
Спеціальність	126 «Інформаційні системи та технології» ОП «Інформаційні системи та технології»

Затверджую:
Завідувач кафедри _____ Світлана АГАДЖАНОВА

« _____ » _____ 202_ р.

ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ *Ковальову Роману Станіславовичу*

1. Тема роботи: Розробка генераторів 3d контенту на прикладі генератору меша рельєфу місцевості

2. База практичного дослідження: Сумський НАУ

керівник роботи В'юненко Олександр Борисович к.т.н., доцент

затверджена наказом по університету від « _____ » _____ 202_ р. № _____

3. Строк подання студентом закінченого проекту (роботи)

«28» червня 2024 року

4. Вихідні дані до проекту (роботи):

Результати аналітичних досліджень зарубіжних та вітчизняних вчених, інтернет-джерела, нормативні документи за чинним законодавством.

5. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити):

- виконати постановку задачі;

- вибрати середовище для реалізації;

- розробити технічне завдання;

- розробити програмний продукт відповідно до поставлених задач;

- розробити інструкцію для користувача.

6. Дата видачі завдання:

« _____ » _____ 202_ р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів проекту	Примітка
1	Визначення об'єкту, предмету дослідження, формулювання мети та задач кваліфікаційної роботи, складання плану	січень 2024 р.	
2	Підбір та вивчення літературних джерел, технічної документації	лютий 2024 р.	
3	Розробка технічного завдання	березень 2024 р..	
4	Розробка додатку	лютий-квітень 2024 р.	
6	Оформлення пояснювальної записки	до 30 травня 2024 р..	
7	Оформлення презентації	до 1 червня 2024 р.	
8	Опрацювання зауважень керівника, перевірка на автентичність	5 червня 2024 р.	
9	Підготовка доповіді та її попередній захист	10 червня 2024 р.	
10	Перевірка з академічної доброчесності кваліфікаційної роботи	10 червня 2024 р.	
11	Представлення кваліфікаційної роботи на кафедрі, деканат	14 червня 2024 р.	
12	Захист кваліфікаційної роботи	28 червня 2024р.	

Студент

(підпис)

Роман КОВАЛЬОВ

Керівник роботи

(підпис)

Олександр В'ЮНЕНКО

Перевірку на автентичність проведено

(підпис)

Надія БАРАНІК

Кваліфікаційна робота допущена до захисту

(підпис)

Маргарита ЛИШЕНКО

АНОТАЦІЯ

Ковальов Р.С. Розробка генераторів 3D контенту на прикладі генератору меша рельєфу місцевості.ч

Кваліфікаційна робота зі спеціальності 126 «Інформаційні системи та технології» ОП Інформаційні системи та технології, СНАУ, Суми, 2024р. – Рукопис.

Ця робота присвячена розробці та програмуванню генератору мешу рельєфу місцевості. Головною метою роботи є створення прототипу інструменту, що допоможе створювати 3D контент на основі двовимірних вхідних даних, а також аналіз проблем зв'язаних з реалізацією такого продукту.

У роботі проведений аналіз сучасного стану генераторів 3D контенту та виявлені проблеми, з якими зіштовхуються розробники при реалізації таких інструментів. Розроблено прототип такого типу програмного забезпечення, виокремлені основні проблеми та задачі при розробці, вироблені і запропоновані оптимальні підходи до них.

Для розробки використані сучасні технології візуалізації трьох вимірної графіки. Реалізована генерація трьох вимірної сітки мешу на основі вхідної карти висот, інтерфейс користувача для керування генерацією.

Проведено аналіз труднощів та завдань, які з'явилися в процесі розробки, для кожної з них представлено теоретичне рішення та практична реалізація.

Отримані результати є масивом практичних рекомендацій і рішень типових проблем реалізації такого проекту, які дозволять прискорити і полегшити розробку генераторів контенту фахівцям, які зіткнуться з таким завданням у майбутньому.

Ключові слова: 3D, csharp, dotnet, OpenGL, shaders.

SUMMARY

Kovalov R.S. Development of 3D content generators on the example of terrain relief mesh generator.

Qualification work on specialty 126 "Information systems and technologies" OP Information systems and technologies, SNAU, Sumy-2024. - Manuscript.

This work is devoted to the development and programming of the terrain relief mesh generator. The main goal of the work is the creation of a prototype of a tool that will help create 3D content based on two-dimensional input data, as well as the analysis of problems related to the implementation of such a product.

The paper analyzes the current state of 3D content generators and identifies the problems faced by developers when implementing such tools. A prototype of this type of software was developed, the main problems and tasks during the development were identified, optimal approaches to them were developed and proposed.

Modern technologies of visualization of three-dimensional graphics were used for development. Realized generation of three-dimensional mesh grid based on the input height map, user interface for generation control.

An analysis of the difficulties and tasks that appeared during the development process was carried out, and a theoretical solution and practical implementation were presented for each of them.

The obtained results are an array of practical recommendations and solutions to typical problems of implementing such a project, which will speed up and facilitate the development of content generators for specialists who will face such a task in the future.

Keywords: 3D, csharp, dotnet, OpenGL, shaders.

ЗМІСТ

ВСТУП	7
РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ	
1.1. Огляд останніх досліджень і публікацій	9
1.2. Аналіз існуючих систем	10
РОЗДІЛ 2. ПОСТАНОВКА ЗАДАЧІ ТА ВИБІР ЗАСОБІВ РЕАЛІЗАЦІЇ	
2.1. Постановка задачі та визначення мети	13
2.2. Вибір засобів реалізації	14
РОЗДІЛ 3. ПРОЄКТУВАННЯ ТА РЕАЛІЗАЦІЯ ГЕНЕРАТОРУ МЕШУ РЕЛЬЄФА МІСЦЕВОСТІ	
3.1. Розробка алгоритмів функціонування генератору	20
3.2. Розробка генератору	25
3.3. Імплементация інтерфейсу користувача	40
ВИСНОВКИ	45
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	54
ДОДАТКИ	
Додаток А ТЕХНІЧНЕ ЗАВДАННЯ	50
Додаток Б КОПІЇ ПУБЛІКАЦІЙ	56
Додаток В ЛІСТИНГИ ПРОГРАМ	64

ВСТУП

Актуальність теми. У сучасному світі, де технології швидко розвиваються, розробка програмного забезпечення для генерації тривимірних об'єктів, зокрема, рельєфних мереж, є важливою задачею. Це програмне забезпечення має широке застосування, включаючи створення терейнів для візуалізації географічних даних, в ігровій індустрії, а також для моделювання природних ландшафтів у наукових дослідженнях. Ця робота спрямована на розробку інструментів, які допоможуть ефективно створювати тривимірні рельєфи.

Мета і завдання дослідження. Метою даної роботи є розробка прототипу програми для генерації рельєфної мережі, ідентифікація викликів та проблем розробки такого програмного забезпечення, їх опис та запропоновані рішення.

Відповідно до поставленої мети сформульовані такі задачі:

- провести огляд останніх досліджень та публікацій у галузі генерації тривимірних об'єктів, зокрема, рельєфних мереж;
- здійснити аналіз існуючих програмних рішень для автоматизованої генерації терейнів;
- сформулювати постановку задачі та вибрати необхідні інструменти для її реалізації;
- розробити концепцію та структуру програмного продукту для генерації рельєфної мережі;
- здійснити фізичну реалізацію програмного продукту та розробити інтерфейс користувача у вибраному програмному середовищі;

Об'єкт і предмет дослідження. Об'єктом дослідження є процес генерації рельєфної мережі. Предметом дослідження є практична реалізація програмного забезпечення для автоматизованої генерації терейнів.

Апробація результатів кваліфікаційної роботи. Modern science: innovations and prospects. Proceedings of the 5th International scientific and practical conference. SSPG Publish. Stockholm, Sweden. 2022. - . 175с.

Матеріали науково-практичної конференції викладачів, аспірантів та

студентів Сумського НАУ (14-16 травня 2024 р.). – Суми, 2024. – 429с.; 430с.

Публікації. За матеріалами кваліфікаційної роботи опубліковано 3 тези наукових конференцій.

Структура і обсяг роботи. Кваліфікаційна робота складається із вступу, трьох розділів, висновків та пропозицій, списку використаних джерел з 16 найменуваннями, додатків. Основний текст викладений на 50 сторінках комп'ютерного тексту.

РОЗДІЛ 1

АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Огляд останніх досліджень і публікацій

Останні роки характеризуються значним прогресом у галузі генерації тривимірних об'єктів, зокрема, рельєфних мереж. Дослідження та публікації у цій області активно вивчають методи та технології для автоматизованої генерації терейнів з високим ступенем реалізму та деталізації.

Одним з перспективних напрямків у цій області є застосування алгоритмів генетичного програмування та штучного інтелекту для створення оптимальних терейнів. Дослідники виявили, що за допомогою генетичних алгоритмів можна ефективно оптимізувати параметри рельєфної мережі з урахуванням заданих критеріїв, таких як реалізм, гладкість та геометрична складність.

Крім того, велика увага приділяється розробці алгоритмів для генерації терейнів з використанням шумових функцій та фрактальних аналізів. Такі підходи дозволяють отримати рельєфні моделі з натуральними деталями та варіаціями, що підсилюють реалізм отриманих об'єктів.

Більшість досліджень також активно досліджує можливості паралельних обчислень та оптимізацію алгоритмів для підвищення швидкодії та ефективності генерації терейнів. Використання сучасних технологій, таких як графічні процесори та обчислювальні хмари, дозволяє значно прискорити процес створення складних тривимірних об'єктів.

Загальний тренд у дослідженнях полягає у прагненні до створення програмного забезпечення, яке б забезпечувало широкий спектр можливостей для генерації різноманітних терейнів з врахуванням потреб користувачів у візуалізації та аналізі географічних даних.

1.2 Аналіз існуючих систем

На сьогоднішній день існує ряд програмних рішень для генерації тривимірних об'єктів, включаючи рельєфні мережі. Деякі з цих рішень надають широкий функціонал, в той час як інші обмежені застосуванням або функціоналом.

Одним з найпоширеніших варіантів є програми для редагування та моделювання тривимірних об'єктів, такі як Blender, Autodesk Maya, або 3ds Max. Ці програми мають ряд інструментів для створення рельєфних мереж, але їх основна спрямованість - моделювання об'єктів загалом, тому їм може бракувати специфічних інструментів для ефективної генерації терейнів.

Blender, Autodesk Maya, та 3ds Max - це потужні програмні засоби для редагування та моделювання тривимірних об'єктів, які використовуються в різних галузях, таких як ігрова індустрія, анімація, архітектура, та візуалізація.

Blender (рисунок 1.1) є безкоштовним, відкритим програмним забезпеченням з широким функціоналом для моделювання, анімації, текстурної роботи, візуалізації, компонування та інших задач. Це потужний інструмент, який широко використовується як новачками, так і професіоналами.

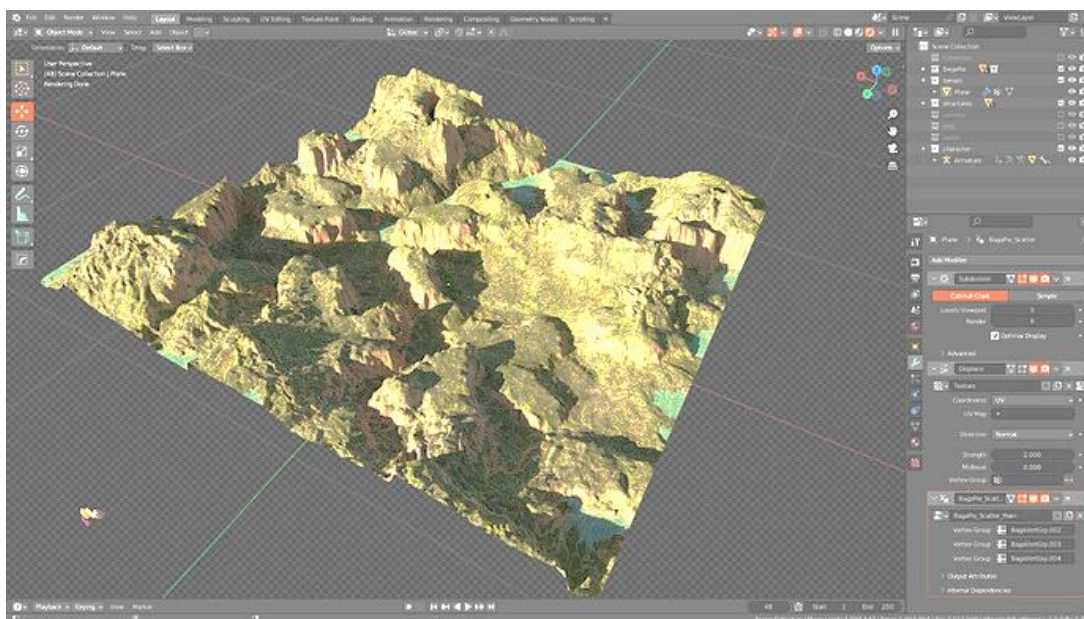


Рисунок 1.1 – Інтерфейс програмного забезпечення Blender

Джерело: [1]

Autodesk Maya (рисунок 1.2) є комерційним програмним забезпеченням, яке володіє величезним набором інструментів для створення складних анімацій, ефектів, а також моделювання, текстурування та візуалізації. Він використовується у великих студіях та компаніях для створення фільмів, ігор та реклами.

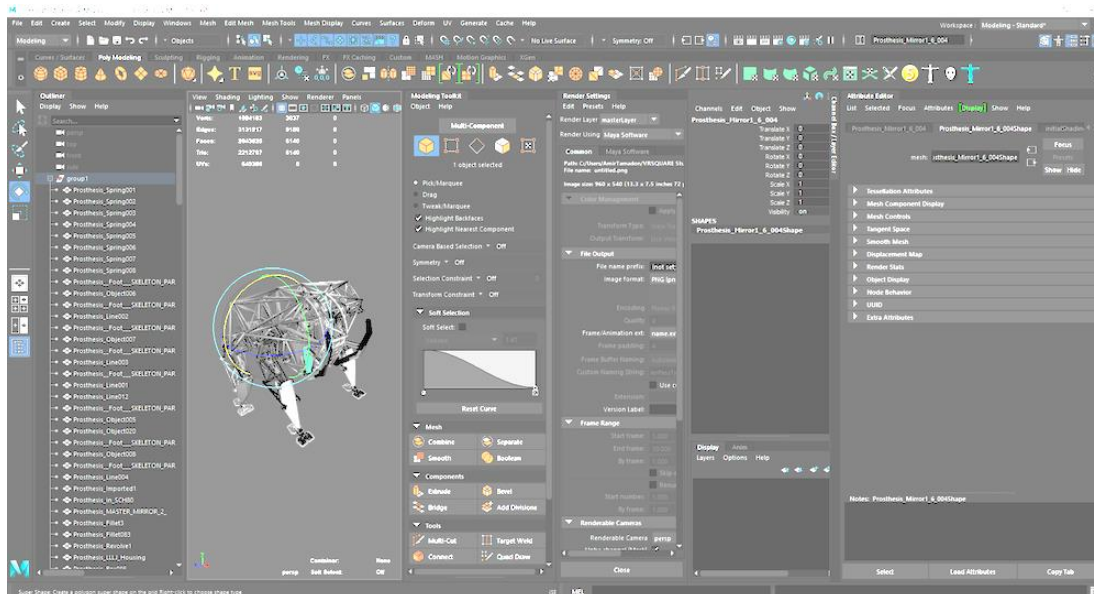


Рисунок 1.2 – Інтерфейс програмного забезпечення Autodesk Maya

Джерело: [2]

3ds Max (рисунок 1.3), також розроблений компанією Autodesk, є іншим комерційним програмним засобом, який спеціалізується на моделюванні, візуалізації та анімації. Він має ряд інструментів для створення реалістичних тривимірних об'єктів та сцен, що робить його популярним серед візуалізаторів, архітекторів та дизайнерів.

Ці програмні засоби володіють різними функціями та можливостями, але вони всі дозволяють створювати тривимірні об'єкти високої якості та здійснювати редагування у тривимірному просторі.

Іншими варіантами є спеціалізовані програми, такі як World Machine або Vue, які спеціалізуються на створенні природних ландшафтів. Ці програми часто мають більші можливості для генерації деталізованих рельєфів та різноманітних природних елементів. Проте, вони можуть бути обмежені у можливостях для створення специфічних типів терейнів або інтеграції з іншими програмами.

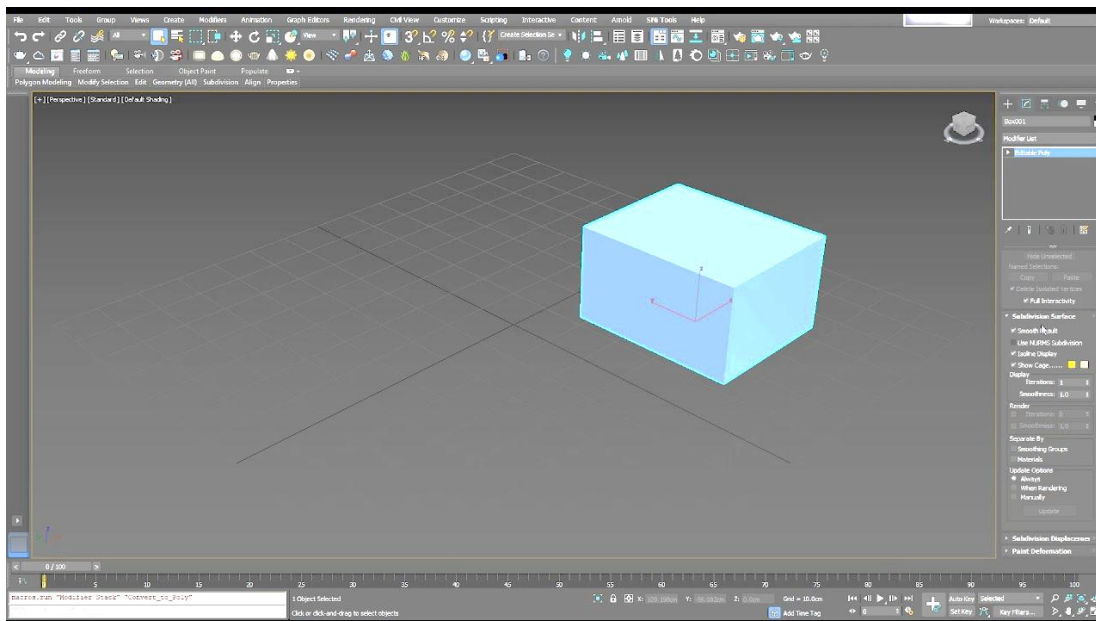


Рисунок 1.3 – Інтерфейс програмного забезпечення 3ds max

Джерело: [3]

Деякі розробники також надають програми з відкритим вихідним кодом, такі як L3DT або Terragen, які можуть бути корисними для експериментів або розробки власних алгоритмів. Однак, ці програми можуть мати обмежений функціонал порівняно з комерційними аналогами, а також можуть вимагати додаткового часу для оволодіння.

В цілому, незважаючи на наявність різноманітних програмних рішень, існують певні недоліки щодо їхньої функціональності, ефективності або спеціалізації. Це створює підґрунтя для подальшого дослідження та розробки нових методів та інструментів для генерації тривимірних терейнів.

РОЗДІЛ 2

ПОСТАНОВКА ЗАДАЧІ ТА ВИБІР ЗАСОБІВ РЕАЛІЗАЦІЇ

2.1 Постановка задачі та визначення мети

Метою даної роботи є розробка прототипу програми для генерації рельєфної мережі, ідентифікація викликів та проблем розробки такого програмного забезпечення, їх опис та запропоновані рішення.

Метою програми є створення мешу рельєфу місцевості на основі вхідних даних, а саме карти рельєфу місцевості, так званої heightmap.

На вхід програма має приймати:

- зображення карти висот основних поширених форматів таких як PNG, JPG;
- параметри генерації що задаються через інтерфейс користувача: розмір площини згенерованого мешу, компонент кольору що є фактором висоти рельєфу, множник висоти рельєфу, LOD(Level of detail) – кількість вершин на одиницю площини;

На виході маємо отримувати:

- згенерована 3D сітка мешу;
- візуалізація згенерованого мешу, з картою висот накладеною на меш у вигляді текстури;

Для забезпечення максимальної доступності такого рішення цільовою платформою буде найпоширеніша середньостатистична конфігурація персонального комп'ютера 2024 року:

- операційні системи Windows 11/10/7, Linux(Ubuntu), Mac OS X -64/32bit;
- 2 та більше ядерний процесор;
- об'єм оперативної пам'яті від 2 GB;
- графічний прискорювач що підтримує Opengl 3.3;

Зображення карти висот має бути розміру не більше 8192x8192. Максимальний розмір текстури для OpenGL 3.3, вибраної специфікації, залежить

від апаратної частини та драйверів конкретної графічної карти. Однак, загально прийнятий максимальний розмір текстури для апаратного забезпечення, сумісного з OpenGL 3.3, становить 8192x8192 пікселів як для ширини, так і для висоти текстури. Цей розмір зазвичай підтримується сучасними графічними картами та драйверами.

Проте важливо зауважити, що фактичний максимальний розмір текстури може відрізнятися в залежності від виробника GPU та конкретних можливостей апаратного забезпечення та драйверів. Рекомендується запитувати максимальний розмір текстури під час роботи програми за допомогою функцій OpenGL, таких як “glGetIntegerv(GL_MAX_TEXTURE_SIZE, &maxSize)”.

Знання максимального розміру текстури важливо для ефективного управління пам'яттю та оптимізації використання текстур у OpenGL програмі. Використання текстур з розмірами, що перевищують максимально підтримуваний розмір, може призвести до непередбачуваного поведінки програми або зниження продуктивності.

2.2 Вибір засобів реалізації

Visual Studio Code (VSCode) обрано для використання в цьому проекті з наступних причин, які варто розглянути. По-перше, VSCode є потужним та розширюваним інструментом розробки, який надає зручне середовище для програмістів [4]. Він має широкий спектр функцій, включаючи підтримку розширень, синтаксичне підсвічування, автодоповнення, вбудовану систему керування версіями та інші.

У контексті даного проекту, в якому розроблено програму для створення рельєфних мешів на основі карт висот, VSCode надавав зручне середовище для редагування коду на мові програмування C#, що використана в цьому проекті. Крім того, завдяки розширенням, розробник мав змогу легко інтегрувати різноманітні інструменти та додаткові функції, що сприяло покращенню продуктивності та зручності розробки.

Додатково, VSCode є крос-платформовим інструментом, що означає, що він підтримується на різних операційних системах, таких як Windows, macOS та Linux.

Отже, використання Visual Studio Code в даному проекті обґрунтованим з точки зору його функціональності, зручності використання та підтримки крос-платформеності, що сприяло успішному розробленню програмного забезпечення для генерації рельєфних мешів.

Технологія .NET обрана для використання у проекті з наступних причин:

- мова програмування C#: Однією з ключових складових .NET є мова програмування C#, яка володіє широким спектром функціональності та є ефективною для розробки програмного забезпечення. Використання C# дозволило розробникам швидко і ефективно реалізувати алгоритми для генерації рельєфних мешів;

- широкий функціонал .NET Framework: .NET Framework надає багато вбудованих бібліотек і інструментів для розробки програмного забезпечення, що дозволило ефективно використовувати готові рішення та скоротити час розробки;

- інтеграція з іншими сервісами Microsoft: Розробка в середовищі .NET дозволила ефективно інтегрувати програму з іншими сервісами та інструментами від Microsoft, що може бути корисним у випадках, коли необхідно взаємодіяти з іншими продуктами цієї компанії;

- підтримка та спільнота: .NET має велику активну спільноту розробників, яка надає підтримку та здійснює постійний розвиток технології. Це забезпечує наявність ресурсів та інформації для вирішення проблем та підтримки проекту;

Використання технології .NET має наступні переваги у контексті даного проекту:

- продуктивність: .NET володіє високою продуктивністю, що дозволило швидко реалізувати алгоритми генерації рельєфних мешів та забезпечити плавну та ефективну роботу програми;

- крос-платформеність: Завдяки .NET Core та .NET 5, програма може бути розгорнута на різних операційних системах, включаючи Windows, macOS та Linux, що розширює коло потенційних користувачів та сприяє зручності розгортання[5];

- безпека: Технологія .NET надає високий рівень безпеки, включаючи механізми контролю доступу та вбудовані інструменти для захисту даних, що особливо важливо у випадку обробки конфіденційної інформації [6];

Мова програмування C# обрана для використання у проекті з наступних причин:

- широкі можливості для розробки програмного забезпечення: C# є потужною мовою програмування, яка надає розробникам широкий спектр можливостей для реалізації різноманітних проектів. Використання C# дозволило ефективно втілити концепції та алгоритми, необхідні для створення генератора рельєфних мешів;

- інтеграція з платформою .NET: C# є ключовою мовою програмування для платформи .NET, що забезпечує йому велику кількість готових бібліотек та інструментів для розробки програмного забезпечення. Це спростило процес розробки та дозволило ефективно використовувати різноманітні функціональності .NET Framework;

- інтеграція з іншими середовищами розробки: Мова C# підтримується широким спектром середовищ розробки, включаючи Visual Studio [7], Visual Studio Code [8], інші IDE та редактори коду. Це дозволило розробникам вибрати оптимальне середовище для роботи з проектом;

- простота вивчення та використання: C# має простий та зрозумілий синтаксис, що робить його доступним для розробників з різним рівнем досвіду;

Використання мови програмування C# мало наступні переваги у контексті даного проекту:

- продуктивність: C# дозволила розробнику швидко реалізувати алгоритми та функціональність генератора рельєфних мешів, що забезпечило високу продуктивність розробки;

- інтеграція з платформою .NET: Використання C# у поєднанні з .NET Framework надало доступ до розширених можливостей та бібліотек, що спростило розробку та забезпечило більшу гнучкість у виборі рішень;

- підтримка та спільнота: C# має велику та активну спільноту розробників, яка надає підтримку, ресурси та інформацію для вирішення проблем та підвищення рівня навичок [9];

У контексті розробки програмного забезпечення для створення рельєфних мешів на основі карт висот, вибір бібліотеки OpenGL та фреймворку OpenTK для реалізації графічної складової був обґрунтований рядом ключових переваг.

OpenGL (Open Graphics Library) є крос-платформеним програмним інтерфейсом для розробки графічних програм. Це стандарт, який надає функції для взаємодії з графічним апаратним забезпеченням комп'ютера з метою відображення 2D та 3D графіки. OpenGL дозволяє розробникам створювати візуально складні та динамічні графічні додатки для різних платформ, включаючи комп'ютери, мобільні пристрої та ігрові консолі. Однією з основних переваг OpenGL є його широке поширення та підтримка різними виробниками апаратного забезпечення, що гарантує стабільну роботу на різноманітних пристроях. Це особливо важливо для проєктів, які мають на меті охопити широку аудиторію користувачів.

Основні функції OpenGL включають в себе низькорівневі операції, такі як відображення примітивів (точок, ліній, трикутників), роботу з текстурами, налаштування освітлення та матеріалів, а також здійснення перетворень координат. Крім того, OpenGL має відкритий стандарт, що означає, що його специфікація та реалізації доступні для використання різними розробниками та виробниками графічного обладнання. Це забезпечує велику різноманітність у підтримці та використанні OpenGL на різних пристроях та платформах.

Перш за все, OpenGL, як крос-платформене графічне API, забезпечує універсальність в роботі на різних операційних системах, що важливо для проєкту, орієнтованого на різних користувачів. Це дозволяє нашій програмі бути доступною і функціональною як на Windows, так і на macOS та Linux. Другим важливим аспектом є продуктивність та ефективність OpenGL. Він надає прямий доступ до апаратного забезпечення, що дозволяє оптимізувати графічні операції та забезпечувати високу швидкість програми, що особливо важливо у випадку обробки великих обсягів даних.

OpenTK, у свою чергу, забезпечує зручний інтерфейс для роботи з OpenGL у середовищі мови програмування C#. Це робить процес розробки більш простим та зручним, дозволяючи розробникам швидко впроваджувати нові функції та ефекти у графічну частину програми. Крім того, OpenTK включає в себе додаткові функції для роботи з віконною системою та введенням/виведенням, що полегшує інтеграцію OpenGL у проекти на C#.

Загалом, використання OpenGL та OpenTK в проекті дало змогу створити графічну частину програми, яка поєднує в собі високу продуктивність, крос-платформенність та зручний інтерфейс розробки. Це зробило генератор рельєфних мешів потужним інструментом для візуалізації географічних даних та створення реалістичних ландшафтів. Завдяки OpenGL та OpenTK вдалося досягти високої якості зображень та ефективної обробки графічних даних.

В проекті використано ImGui для реалізації користувацького інтерфейсу. ImGui є легкою, гнучкою та простою у використанні бібліотекою для створення графічного інтерфейсу користувача в програмах. Вона дозволяє швидко і зручно створювати різноманітні елементи інтерфейсу, такі як кнопки, поля введення, списки та інші, що дозволяє ефективно взаємодіяти з користувачем і керувати програмою. ImGui також має підтримку для різних стилів оформлення, що дозволяє налаштовувати вигляд інтерфейсу з урахуванням потреб проекту. Його легко інтегрувати з OpenGL та іншими графічними бібліотеками, що робить його ідеальним вибором для використання в проекті [12].

У проекті використана бібліотека SixLabors.ImageSharp для роботи з зображеннями. SixLabors.ImageSharp є потужним інструментом для обробки та маніпуляції зображеннями у програмах на мові програмування C#. Ця бібліотека надає розробникам широкий спектр функціональності для роботи з растровими зображеннями, включаючи завантаження, збереження, зміну розміру, розмите та обрізання зображень, а також взаємодію з пікселями та каналами кольору. Використання SixLabors.ImageSharp забезпечує високоякісну обробку зображень, що є важливим для створення точних та реалістичних текстур [13].

Однією з ключових переваг SixLabors.ImageSharp є його простота використання та широкий функціональний спектр. Вона надає зручний та інтуїтивно зрозумілий API, що дозволяє швидко і ефективно виконувати різноманітні завдання з обробки зображень без значного зусилля. Крім того, SixLabors.ImageSharp є крос-платформеною бібліотекою, що дозволяє використовувати її на різних операційних системах, включаючи Windows, macOS та Linux. Це забезпечує універсальність та портативність розроблених за допомогою неї програм.

У контексті розроблюваного проекту SixLabors.ImageSharp використовується для завантаження та обробки вхідних зображень, а також для створення текстур та різноманітних візуальних ефектів у вигляді рельєфних мешів. Це дозволяє ефективно працювати з графічними даними та створювати якісні візуальні ефекти у програмному забезпеченні. Завдяки своїй гнучкості та потужності, SixLabors.ImageSharp забезпечує високий рівень контролю над обробкою зображень, що сприяє досягненню найкращих результатів у розробці графічного інтерфейсу та візуальних ефектів.

РОЗДІЛ 3

ПРОЄКТУВАННЯ ТА РЕАЛІЗАЦІЯ ГЕНЕРАТОРУ МЕШУ РЕЛЬЄФА МІСЦЕВОСТІ

3.1 Розробка алгоритмів функціонування генератора

Розділ "Розробка алгоритмів функціонування генератора" вимагає докладного розгляду блок-схем, які визначають процеси та процедури, що використовуються у програмному забезпеченні для створення рельєфних мешів. Нижче наведено опис ключових блок-схем, які використовуються в процесі розробки алгоритмів функціонування генератора.

Блок-схеми — це графічні зображення алгоритмів або процесів, що складаються з різних геометричних фігур, таких як прямокутники, ромби та овали, які з'єднані стрілками. Кожна фігура репрезентує конкретний етап або дію, а стрілки вказують на послідовність виконання цих дій. Блок-схеми є потужним інструментом, який дозволяє візуально відобразити логіку та послідовність процесів, роблячи їх зрозумілими навіть для тих, хто не є експертами у відповідній галузі. Блок-схеми сприяють розумінню складних процесів, розділяючи їх на окремі кроки. Вони слугують інструментом для документування алгоритмів, що спрощує їх подальший аналіз і підтримку [14].

Блок-схема обробки вхідних даних (рисунок 2.1):

- визначає процес обробки вхідних даних, зокрема висотної карти, яка передається до генератора;
- включає кроки зчитування файлу вхідних даних, перевірку коректності формату, валідацію даних та підготовку до подальшої обробки;

Блок-схема алгоритму генерації мешів (рисунок 2.2):

- описує основний алгоритм, що використовується для створення рельєфного мешу на основі вхідної висотної карти;

- вона включає кроки визначення топології мешу, обчислення координат вершин, нормалей поверхні, а також створення індексів для тріангуляції;

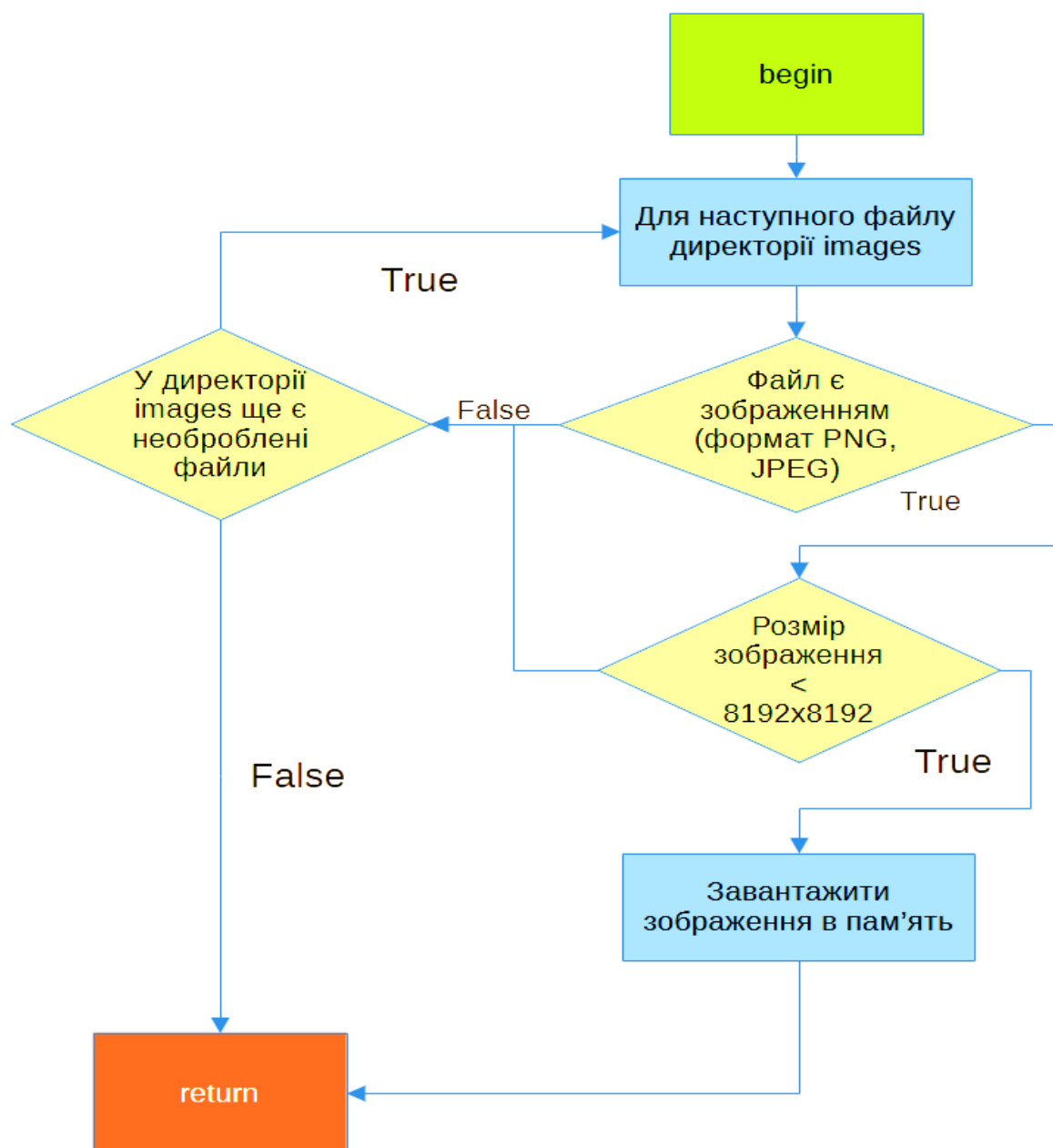


Рисунок 2.1 – Блок-схема обробки вхідних даних

Джерело: запропоновано автором



Рисунок 2.2 – Блок-схема алгоритму генерації мешів

Джерело: запропоновано автором

Блок-схема візуалізації результатів (рисунок 2.3):

- описує процес візуалізації рельєфного мешу для подальшого аналізу та використання;

- включає кроки створення вікна відображення, налаштування параметрів відображення, а також рендерингу мешу з використанням OpenGL;

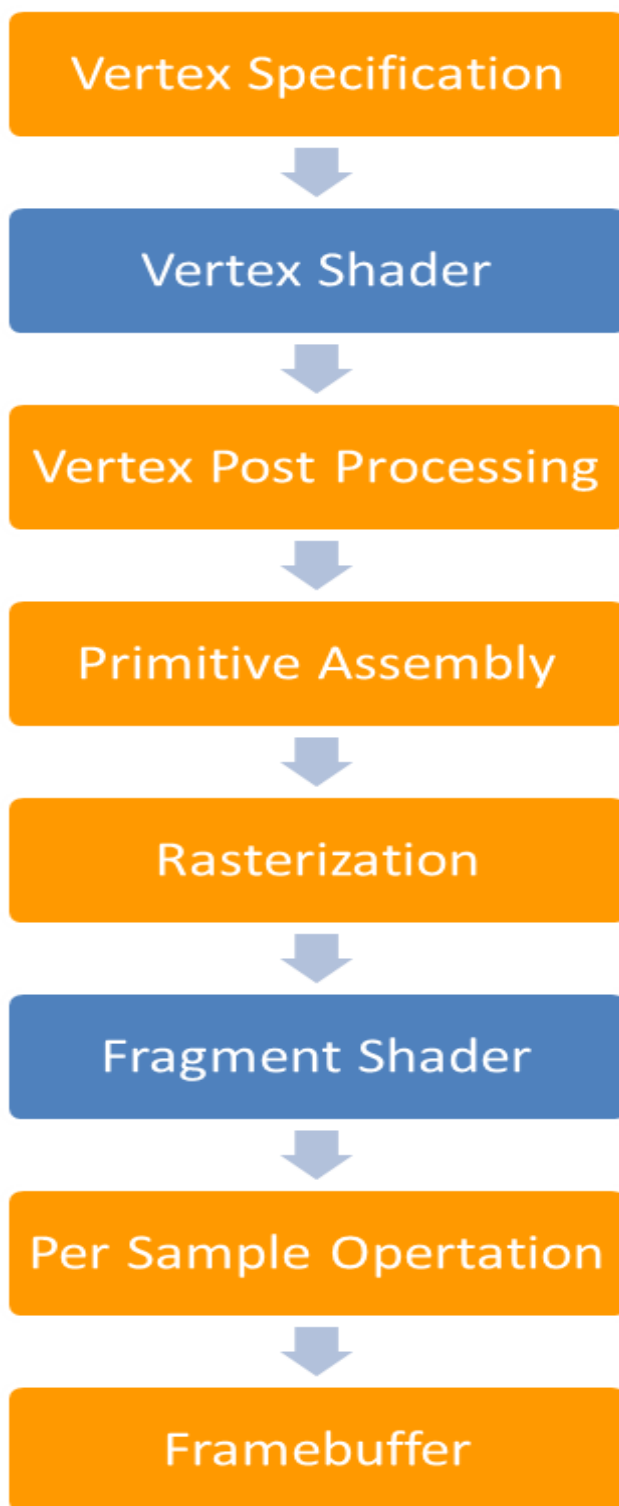


Рисунок 2.3 – Послідовність алгоритму візуалізації OpenGL

Джерело: [15]

OpenGL пайплайн - це послідовність кроків, які програма пройде, щоб обробити графічні дані та відобразити їх на екрані. Пайплайн складається з наступних етапів:

- Vertex Specification (Специфікація вершин): На цьому етапі програма приймає вхідні дані - вершини, кожна з яких характеризує точку у просторі. Ці вершини можуть містити різноманітні атрибути, такі як позиція, колір, текстурні координати тощо;

- Vertex Shader (Вершинний шейдер): Вершинний шейдер обробляє кожну вершину окремо. Він може змінювати їх позиції, кольори, виконувати трансформації та інші операції. Результатом цього етапу є оброблені вершини;

- Primitive Assembly (Збірка примітивів): На цьому етапі OpenGL об'єднує вершини в примітиви, такі як точки, лінії або трикутники. Ці примітиви будуть використані для подальшого відображення на екрані;

- Geometry Shader (Геометричний шейдер): Геометричний шейдер може обробляти цілі примітиви, а не просто окремі вершини. Він може створювати нові примітиви, видаляти або модифікувати існуючі. Це дозволяє реалізувати такі ефекти як генерація тіней, видалення вершин та інші;

- Rasterization (Растеризація): На цьому етапі OpenGL перетворює геометричні примітиви в пікселі, які будуть відображені на екрані. Для кожного примітиву визначається його положення та колір пікселів, які відповідають йому;

- Fragment Shader (Фрагментний шейдер): Фрагментний шейдер обробляє кожен піксель, що відображається на екрані. Він може застосовувати різні ефекти, такі як текстуркування, освітлення, тіні тощо;

- Per-Sample Processing (Обробка на рівні зразка): На цьому етапі виконується обробка кожного зразка (або пікселя) окремо, включаючи анти-аліасинг та інші післяобробні ефекти;

- Framebuffer (Буфер кадру): Завершальний результат роботи пайплайна зберігається у буфері кадру, який потім може бути відображений на екрані;

Після завершення пайплайна OpenGL, відбувається процес візуалізації та рендерингу, який включає наступні етапи:

- налаштування відображення: Програма встановлює параметри відображення, такі як розмір вікна, камеру, освітлення тощо;
- налаштування OpenGL контексту: Ініціалізація OpenGL контексту, налаштування параметрів OpenGL;
- відображення сцени: Програма рендерить сцену, використовуючи пайплайн OpenGL. Це включає передачу графічних об'єктів у відеопам'ять, запуск пайплайна OpenGL та відображення результуючого зображення на екрані;
- оновлення відображення: У разі зміни сцени або параметрів відображення, програма оновлює відображення шляхом повторного рендерингу сцени та виведення оновленого зображення на екран;
- завершення роботи програми: Після завершення відображення сцени та взаємодії з користувачем, програма завершує роботу OpenGL контексту та виводить результат;

Ці блок-схеми становлять ключову складову процесу розробки алгоритмів функціонування генератору та дозволяють чітко визначити послідовність операцій та логіку роботи програми. Розгляд цих блок-схем сприятиме розумінню та аналізу роботи генератору рельєфних мешів.

3.2. Розробка генератору

Для розробки генератора було створено комплексний набір класів, які служать основою для формування базового фреймворку, забезпечуючи інтеграцію та узгоджену взаємодію всіх компонентів системи для ефективної генерації тривимірних об'єктів і рельєфних мереж.

Клас “Texture”. Цей клас (рисунк 3.1) призначений для завантаження, налаштування та використання текстури з файлу зображення в контексті OpenGL.

Визначення класу. Клас Texture визначається в просторі імен Fwork і включає необхідні директиви для роботи з колекціями, операціями OpenGL та обробкою зображень, їх завантаженням.

```

using System.Collections.Generic;
using OpenTK.Graphics.OpenGL;
using SixLabors.ImageSharp;
using SixLabors.ImageSharp.PixelFormats;

namespace Fwork
{
    public class Texture
    {
        private int id;
        public string Name { get; set; }
        public Texture(string path)
        {
            // Реалізація конструктора
        }

        public void Use(TextureUnit unit)
        {
            // Реалізація методу
        }
    }
}

```

Рисунок 3.1 – Сигнатура класу Texture

Джерело: запропоновано автором

Приватне поле та публічна властивість:

- id: Це приватне ціле число зберігає унікальний ідентифікатор текстури, згенерований OpenGL;
- Name: Ця публічна властивість дозволяє задавати та отримувати ім'я текстури, що полегшує її ідентифікацію;

Конструктор класу Texture приймає один параметр типу string, який представляє шлях до файлу. Цей шлях вказує на місце розташування зображення, яке буде завантажене як текстура. Під час створення об'єкта цього класу, конструктор виконує кілька важливих кроків для завантаження та налаштування текстури.:

Генерація та прив'язка текстури (рисунок 3.2):

- GL.GenTexture() генерує унікальний ідентифікатор текстури;
- GL.ActiveTexture(TextureUnit.Texture0) активує текстурний блок 0;
- GL.BindTexture(TextureTarget.Texture2D, id) прив'язує згенеровану текстуру до цільового об'єкта 2D-текстури;

Завантаження даних зображення – рисунок 3.3:

- зображення завантажується за допомогою бібліотеки ImageSharp, яка читає файл зображення і зберігає піксельні дані в об'єкті Image<Rgba32>;
- вкладений цикл здійснює ітерацію по пікселях зображення у зворотному порядку, починаючи з нижнього правого кута і рухаючись поступово вгору та вліво. В процесі цієї ітерації він витягує компоненти RGB кожного окремого пікселя, що складаються з червоного, зеленого, синього. Ці компоненти додаються до спеціалізованого списку байтів, відомого як colorData. Завдяки цьому ретельному процесу забезпечується правильне збереження всієї кольорової інформації зображення в структурованому форматі. Кожен піксель, пройшовши цей процес, забезпечує точне представлення кольору, що критично важливо для подальшої обробки та відображення зображення.;

Створення текстури (рисунок 3.4):

- GL.TexImage2D створює 2D-текстуру з піксельних даних із заданими параметрами ширини, висоти та формату пікселів;

Налаштування параметрів текстури(рисунок 3.5):

- виклики налаштовують фільтри мінімізації та збільшення текстури для лінійної фільтрації, забезпечуючи плавні переходи;
- режими обгортання для координат S і T встановлені на повторення, що дозволяє текстурі безшовно плиткуватися;

Генерація міпмапів:

- GL.GenerateMipmap(GenerateMipmapTarget.Texture2D);
- генеруються міпмапи для покращення рендерингу текстури на різних відстанях та рівнях деталізації;

Метод Use(TextureUnit unit) (рисунок 3.6) прив'язує текстуру до заданого текстурного блоку, дозволяючи використовувати її в подальших операціях рендерингу:

- GL.ActiveTexture(unit) активує вказаний текстурний блок;
- GL.BindTexture(TextureTarget.Texture2D, id) прив'язує текстуру до цільового об'єкта 2D-текстури для використання в рендерингу;

```
id = GL.GenTexture();
GL.ActiveTexture(TextureUnit.Texture0);
GL.BindTexture(TextureTarget.Texture2D, id);
```

Рисунок 3.2 – Генерація та прив'язка текстури

Джерело: запропоновано автором

```
List<byte> colorData = new List<byte>();
Image<Rgba32> image = Image.Load<Rgba32>(path);
for (int y = image.Height - 1; y >= 0; y--)
{
    for (int x = image.Width - 1; x >= 0; x--)
    {
        colorData.Add(image[x, y].R);
        colorData.Add(image[x, y].G);
        colorData.Add(image[x, y].B);
        colorData.Add(image[x, y].A);
    }
}
```

Рисунок 3.3 – Завантаження даних зображення

Джерело: запропоновано автором

```
GL.TextureImage2D(TextureTarget.Texture2D, 0, PixelInternalFormat.Rgba, image.Width, image.Height, 0,
PixelFormat.Rgba, PixelType.UnsignedByte, colorData.ToArray());
```

Рисунок 3.4 – Завантаження даних зображення

Джерело: запропоновано автором

```
GLTexParameteri(TextureTarget.Texture2D, TextureParameterName.TextureMinFilter,
(int)TextureMinFilter.Linear);
GLTexParameteri(TextureTarget.Texture2D, TextureParameterName.TextureMagFilter,
(int)TextureMagFilter.Linear);
GLTexParameteri(TextureTarget.Texture2D, TextureParameterName.TextureWrapS,
(int)TextureWrapMode.Repeat);
GLTexParameteri(TextureTarget.Texture2D, TextureParameterName.TextureWrapT,
(int)TextureWrapMode.Repeat);
```

Рисунок 3.5 – Налаштування параметрів текстури

Джерело: запропоновано автором

```
public void Use(TextureUnit unit)
{
    GL.ActiveTexture(unit);
    GL.BindTexture(TextureTarget.Texture2D, id);
}
```

Рисунок 3.6 – Налаштування параметрів текстури

Джерело: запропоновано автором

Клас “Shader”. Клас Shader, що є частиною простору імен Fwork,

призначений для обробки операцій шейдерів в OpenGL за допомогою бібліотеки OpenTK. Цей клас інкапсулює функціональність, необхідну для завантаження, компіляції та керування шейдерами в контексті OpenGL. Нижче наведено детальний опис класу Shader та його компонентів (рисунок 3.7):

```
using System;
using System.IO;

using OpenTK.Graphics.OpenGL;

namespace Fwork
{
    class Shader
    {
        public int Id { get; set; }

        public void Load(string path, ShaderType shaderType)
        {
            Id = GL.CreateShader(shaderType);
            GL.ShaderSource(Id, File.ReadAllText(path));
            GL.CompileShader(Id);

            string infoLog = GL.GetShaderInfoLog(Id);
            if (!string.IsNullOrEmpty(infoLog))
                throw new Exception(infoLog);
        }
    }
}
```

Рисунок 3.7 – Клас Shader

Джерело: запропоновано автором

Клас Shader включає в себе такі компоненти і виконує таку функціональність:

Простір імен та залежності:

- клас знаходиться в просторі імен Fwork;
- використовуються простори імен System, System.IO та OpenTK.Graphics.OpenGL. System та System.IO є стандартними бібліотеками .NET, тоді як OpenTK.Graphics.OpenGL є частиною бібліотеки OpenTK, яка забезпечує прив'язки до API OpenGL;

Члени класу:

- властивість Id: Ця властивість зберігає унікальний ідентифікатор для об'єкта шейдера, створеного OpenGL. Це ціле число, і доступ до нього є публічним;
- метод Load: Цей метод відповідає за завантаження, компіляцію та перевірку помилок шейдера з файлу;

Метод Load виконує кілька критично важливих операцій для підготовки шейдера до використання в програмі OpenGL:

1) Створення шейдера: `Id = GL.CreateShader(shaderType);` Цей рядок створює новий об'єкт шейдера зазначеного типу `shaderType` (наприклад, `ShaderType.VertexShader`, `ShaderType.FragmentShader`) і присвоює його ідентифікатор властивості `Id`;

2) Завантаження коду шейдера: `GL.ShaderSource(Id, File.ReadAllText(path));` Зчитує код шейдера з файлу, вказаного параметром `path`, використовуючи `File.ReadAllText`. Код шейдера передається в OpenGL за допомогою `GL.ShaderSource`;

3) Компіляція шейдера: `GL.CompileShader(Id);` Ця команда інструктує OpenGL скомпілювати код шейдера, який був завантажений раніше;

4) Перевірка помилок – рисунок 3.8: Після компіляції метод отримує журнал інформації про шейдер за допомогою `GL.GetShaderInfoLog`, щоб перевірити наявність помилок або попереджень під час компіляції. Якщо журнал містить повідомлення (вказуючи на помилки або попередження), викликається виключення з повідомленням журналу як текстом.

```
string infoLog = GL.GetShaderInfoLog(Id);
if (!string.IsNullOrEmpty(infoLog))
    throw new Exception(infoLog);
```

Рисунок 3.8 – Перевірка помилок компіляції шейдеру

Джерело: запропоновано автором

Клас `Shader` є необхідним для будь-якої програми OpenGL, де потрібні користувацькі шейдери. Шейдери - це невеликі програми, які працюють на GPU і використовуються для широкого спектру ефектів, від простих кольорових

трансформацій до складних моделей освітлення та затінення. Метод Load спрощує процес інтеграції коду шейдера в програму, обробляючи як завантаження з файлу, так і компіляцію, а також надаючи простий механізм перевірки помилок.

У прикладі, наведеному на рисунку 3.9, демонструється процес створення нового об'єкта класу Shader. Після створення цього об'єкта використовується метод Load для завантаження та компіляції вершинного шейдера. Метод Load приймає шлях до файлу, в якому зберігається код вершинного шейдера, і завантажує цей файл.

Після завантаження файлу метод Load переходить до етапу компіляції. Під час компіляції шейдерний код перевіряється на наявність синтаксичних та логічних помилок. Якщо компіляція проходить успішно, вершинний шейдер готовий до використання в графічному конвеєрі. Однак, якщо під час компіляції виникають помилки, вони перехоплюються спеціалізованими механізмами обробки помилок.

```
// Створення нового екземпляра Shader
Shader vertexShader = new Shader();

// Завантаження та компіляція вершинного шейдера з файлу
try
{
    vertexShader.Load("path/to/vertex_shader.glsl", ShaderType.VertexShader);
    Console.WriteLine("Вершинний шейдер успішно скомпільовано.");
}
catch (Exception ex)
{
    Console.WriteLine($"Не вдалося скомпілювати вершинний шейдер: {ex.Message}");
}
```

Рисунок 3.9 – Приклад використання класу Shader

Джерело: запропоновано автором

Клас ShaderProgram (рисунок 3.10), що є частиною простору імен Fwork, призначений для керування шейдерними програмами в OpenGL за допомогою бібліотеки OpenTK. Цей клас інкапсулює функціональність, необхідну для завантаження, компіляції та зв'язування вершинних і фрагментних шейдерів у шейдерну програму. Нижче наведено детальний опис класу ShaderProgram та його компонентів.

```

using System;
using OpenTK.Graphics.OpenGL;

namespace Fwork
{
    class ShaderProgram
    {
        public int Id { get; set; }

        public void Load(string vertShaderPath, string fragShaderPath)
        {
            Id = GL.CreateProgram();

            Shader vertShader = new Shader(),
            fragShader = new Shader();

            vertShader.Load(vertShaderPath, ShaderType.VertexShader);
            fragShader.Load(fragShaderPath, ShaderType.FragmentShader);

            GL.AttachShader(Id, vertShader.Id);
            GL.AttachShader(Id, fragShader.Id);
            GL.LinkProgram(Id);
            GL.DetachShader(Id, vertShader.Id);
            GL.DetachShader(Id, fragShader.Id);
            GL.DeleteShader(vertShader.Id);
            GL.DeleteShader(fragShader.Id);

            string infoLog = GL.GetProgramInfoLog(Id);
            if (!string.IsNullOrEmpty(infoLog))
                throw new Exception(infoLog);
        }
    }
}

```

Рисунок 3.10 – Визначення класу ShaderProgram

Джерело: запропоновано автором

Компоненти та функціональність включають:

- простір імен та залежності: клас знаходиться в просторі імен Fwork, використовуються простори імен System та OpenTK.Graphics.OpenGL. System є стандартною бібліотекою .NET, тоді як OpenTK.Graphics.OpenGL є частиною бібліотеки OpenTK, яка забезпечує прив'язки до API OpenGL;
- члени класу: властивість Id. Ця властивість зберігає унікальний ідентифікатор для об'єкта шейдерної програми, створеного OpenGL. Це ціле число, і доступ до нього є публічним.

Метод Load виконує кілька важливих операцій для підготовки шейдерної програми до використання в програмі OpenGL:

- створення програми: `Id = GL.CreateProgram();` Цей рядок створює новий об'єкт шейдерної програми і присвоює його ідентифікатор властивості `Id`;
- завантаження шейдерів (рисунки 3.11): Створюються два об'єкти `Shader` для вершинного та фрагментного шейдерів. Кожен шейдер завантажується з відповідного файлу за допомогою методу `Load` класу `Shader`;
- прикріплення шейдерів до шейдерної програми (рисунки 3.12): Вершинний та фрагментний шейдери прикріплюються до шейдерної програми за допомогою `GL.AttachShader`. Програма зв'язується за допомогою `GL.LinkProgram`, що означає, що всі прикріплені шейдери компілюються та зв'язуються в єдину програму;
- від'єднання та видалення шейдерів (рисунки 3.13): Після зв'язування шейдери від'єднуються від програми і видаляються, оскільки вони більше не потрібні окремо;
- перевірка помилок (рисунки 3.14): Після зв'язування методу отримується журнал інформації про програму за допомогою `GL.GetProgramInfoLog`, щоб перевірити наявність помилок або попереджень. Якщо журнал містить повідомлення (вказуючи на помилки або попередження), викликається виключення з повідомленням журналу як текстом;

```
Shader vertShader = new Shader(),
    fragShader = new Shader();
vertShader.Load(vertShaderPath, ShaderType.VertexShader);
fragShader.Load(fragShaderPath, ShaderType.FragmentShader);
```

Рисунки 3.11 – Створення і завантаження шейдерів

Джерело: запропоновано автором

```
GL.AttachShader(Id, vertShader.Id);
GL.AttachShader(Id, fragShader.Id);
GL.LinkProgram(Id);
```

Рисунки 3.12 – Прикріплення шейдерів до шейдерної програми

Джерело: запропоновано автором

```
GL.DetachShader(Id, vertShader.Id);
GL.DetachShader(Id, fragShader.Id);
GL.DeleteShader(vertShader.Id);
GL.DeleteShader(fragShader.Id);
```

Рисунок 3.13 – Від'єднання та видалення шейдерів

Джерело: запропоновано автором

```
string infoLog = GL.GetProgramInfoLog(Id);
if (!string.IsNullOrEmpty(infoLog))
    throw new Exception(infoLog);
```

Рисунок 3.14 – Перевірка помилок

Джерело: запропоновано автором

Клас ShaderProgram є необхідним елементом для будь-якої програми OpenGL, в якій використовуються користувацькі шейдерні програми. Ці шейдерні програми представляють собою набір шейдерів, об'єднаних і використовуваних разом для виконання різноманітних графічних завдань на GPU, таких як обчислення освітлення, тіней, текстуровання та інших ефектів. Метод Load суттєво спрощує процес інтеграції шейдерів у програму, автоматизуючи всі етапи, починаючи від завантаження файлів шейдерів, їх компіляції та закінчуючи створенням та зв'язуванням програми.

Приклад використання класу ShaderProgram наведено на рисунку 3.15. У цьому прикладі створюється новий об'єкт ShaderProgram, а метод Load застосовується для завантаження та компіляції вершинного та фрагментного шейдерів з вказаних шляхів до файлів. У разі виникнення помилок під час компіляції або зв'язування шейдерів, вони перехоплюються та відображаються, забезпечуючи зручний механізм налагодження і усунення помилок. Цей механізм дозволяє розробникам швидко і ефективно інтегрувати шейдерні програми в їхні проекти, значно підвищуючи продуктивність і зручність роботи з графічними додатками.

```
// Створення нового екземпляра ShaderProgram
ShaderProgram shaderProgram = new ShaderProgram();

// Завантаження та компіляція вершинного та фрагментного шейдерів з файлів
try
{
    shaderProgram.Load("path/to/vertex_shader.glsl",
        "path/to/fragment_shader.glsl");
    Console.WriteLine("Шейдерна програма успішно скомпільована та зв'язана.");
}
catch (Exception ex)
{
    Console.WriteLine($"Не вдалося скомпілювати шейдерну програму: {ex.Message}");
}
```

Рисунок 3.15 – Приклад використання класу ShaderProgram

Джерело: запропоновано автором

Клас Camera, що є частиною простору імен Fwork, призначений для керування камерою в OpenGL за допомогою бібліотеки OpenTK. Цей клас дозволяє реалізувати різні режими роботи камери та забезпечує зручні методи для переміщення і орієнтації камери в тривимірному просторі.

Клас Camera включає в себе:

- простір імен та залежності. Клас знаходиться в просторі імен Fwork. Використовуються простори імен OpenTK.Graphics.OpenGL, OpenTK.Mathematics та OpenTK.Windowing.GraphicsLibraryFramework;

- перерахування CameraMode - визначає два режими камери: Orbiting (орбітальний) та Free (вільний);

- поля та властивості. projectionMat - матриця проєкції, яка визначає перспективу камери. viewMat - матриця виду, яка визначає положення і орієнтацію камери. position - вектор, що зберігає положення камери. Mode - властивість, що зберігає поточний режим камери. Right - вектор, що визначає напрямок праворуч від камери. Front - вектор, що визначає напрямок вперед від камери. Position - властивість, що встановлює та повертає положення камери, з автоматичним перерахунком матриці виду. Rotation - кут обертання камери навколо цілі. Rotation - властивість, що встановлює та повертає кут обертання, з автоматичним перерахунком матриці виду. toTargetDistance - відстань від камери до цілі.

ToTargetDistance - властивість, що встановлює та повертає відстань до цілі, з автоматичним перерахунком матриці виду. Target - вектор, що зберігає положення цілі камери;

- конструктор (рисунок 3.16): Ініціалізує матрицю виду з початковими значеннями положення та орієнтації камери;
- метод Update - Оновлює положення та орієнтацію камери залежно від режиму та натиснутих клавіш;
- методи переміщення камери:
 - 1) MoveForward - переміщує камеру вперед.
 - 2) MoveBackward - переміщує камеру назад.
 - 3) MoveLeft - переміщує камеру вліво.
 - 4) MoveRight: Переміщує камеру вправо.
 - 5) MoveUp: Переміщує камеру вгору.
 - 6) MoveDown: Переміщує камеру вниз;
- метод Apply (рисунок 3.17): Застосовує матриці проекції та виду до поточної шейдерної програми;
- метод RecalculateViewMatrix (рисунок 3.18): Перераховує матрицю виду залежно від режиму камери, положення та орієнтації;

```
public Camera()
{
    viewMat = Matrix4.LookAt(
        new Vector3((float)MathHelper.Sin(Position.X), Position.Y,
        Position.Z), new Vector3(0, 0, 0), new Vector3(0, 1, 0));
}
```

Рисунок 3.16 – Ініціалізація матриці виду з початковими значеннями положення та орієнтації камери.

Джерело: запропоновано автором

```

public void Apply()
{
    GL.UniformMatrix4(
        GL.GetUniformLocation(GL.GetInteger(GetPName.CurrentProgram),
            "projection_mat4"), false, ref projectionMat);
    GL.UniformMatrix4(
        GL.GetUniformLocation(GL.GetInteger(GetPName.CurrentProgram),
            "view_mat4"), false, ref viewMat);
}

```

Рисунок 3.17 – Метод Apply

Джерело: запропоновано автором

```

public void RecalculateViewMatrix()
{
    switch (Mode)
    {
        case CameraMode.Orbiting:
            viewMat = Matrix4.LookAt(
                new Vector3(MathHelper.Sin(Rotation) * ToTargetDistance,
                    Position.Y,
                    (float)MathHelper.Cos(Rotation) * ToTargetDistance),
                Target, new Vector3(0, 1, 0));
            break;
        case CameraMode.Free:
            viewMat = Matrix4.LookAt(Position, Target, new Vector3(0, 1,
2            0));
            break;
        default: break;
    }
}

```

Рисунок 3.18 – Метод RecalculateViewMatrix

Джерело: запропоновано автором

Клас Camera забезпечує зручне управління камерою в тривимірному просторі, дозволяючи розробникам легко змінювати положення та орієнтацію камери залежно від режиму роботи (орбітальний або вільний). Цей клас також включає методи для оновлення стану камери та застосування матриць до шейдерної програми.

Приклад того, як можна використовувати клас Camera в програмі OpenGL наведений на рисунку 3.19.

```
// Створення нового екземпляра
Camera camera = new Camera();

// Оновлення стану камери в головному циклі програми
while (gameRunning)
{
    camera.Update(keyboardState); // Інші операції рендерингу...
    camera.Apply();
}
```

Рисунок 3.19 – Приклад використання класу Camera

Джерело: запропоновано автором

Клас Camera в просторі імен Fwork є інструментом для керування камерою в додатках OpenGL. Завдяки підтримці різних режимів роботи та зручним методам для переміщення та орієнтації камери, цей клас дозволяє розробникам легко інтегрувати управління камерою у свої проекти, забезпечуючи більш реалістичний та динамічний рендеринг сцен.

Клас HeightMap, що є частиною простору імен DataVisualisation3D_OpenTK, призначений для роботи з висотними картами в OpenGL за допомогою бібліотеки OpenTK і ImGui.NET. Цей клас дозволяє генерувати різні рівні деталізації (LOD), завантажувати текстури та відображати висотну карту з можливістю керування налаштуваннями через графічний інтерфейс.

Основні компоненти класу:

- простір імен та залежності: Простір імен: DataVisualisation3D_OpenTK. Залежності: System, System.IO, System.Collections.Generic, OpenTK.Graphics.OpenGL, ImGuiNET, Fwork, System.Numerics;

- поля та властивості: vao, vbo, ebo: Ідентифікатори для Vertex Array Object, Vertex Buffer Object та Element Buffer Object. Lods, Indices: Списки рівнів деталізації та індексів. Textures: Список текстур. Інші: vertexFrequency, wireframeMode, currentTexture, lod, rgb;

- конструктор: Ініціалізує об'єкт HeightMap, генерує LODs, завантажує текстури та налаштовує VBO, EBO та VAO;

Основні методи:

- UpdateLod: Оновлює буфери VBO та EBO новими даними для поточного рівня деталізації (LOD);
- GeneratePlane: Генерує вершини та індекси для сітки висотної карти залежно від частоти вершин;
- Draw: Малює висотну карту з урахуванням поточного рівня деталізації та текстури, також викликає метод DrawUI;
- DrawUI: Відображає графічний інтерфейс для керування налаштуваннями висотної карти за допомогою ImGui;
- Terminate: Видаляє VBO, EBO та VAO при завершенні роботи;
- LoadTextures: Завантажує текстури з заданої папки та зберігає їх у списку Textures;
- GenLods: Генерує рівні деталізації (LOD) для висотної карти залежно від мінімальної та максимальної частоти вершин та кількості кроків. Рівень деталізації (LOD, Level of Detail) — це метод, який використовується в комп'ютерній графіці для оптимізації рендерингу тривимірних об'єктів. Ідея полягає в тому, що об'єкти, які знаходяться ближче до камери, відображаються з високою деталізацією, тоді як об'єкти, які знаходяться далі, рендеряться з меншою кількістю полігонів. Це дозволяє зменшити навантаження на графічний процесор, зберігаючи при цьому високий рівень візуальної якості;

Приклад використання наведено на рисунку 3.20.

```
// Створення нового екземпляра HeightMap
HeightMap heightMap = new HeightMap();

// В головному циклі програми
while (gameRunning)
{
    // Інші операції рендерингу...
    heightMap.Draw();
}
heightMap.Terminate(); // кінець роботи, видалення використаних
```

Рисунок 3.20 – Приклад використання класу Camera

Джерело: запропоновано автором

Клас HeightMap в просторі імен DataVisualisation3D_OpenTK_ є потужним інструментом для роботи з висотними картами в OpenGL. Завдяки підтримці різних рівнів деталізації, завантаженню текстур та інтеграції з ImGui для графічного інтерфейсу, цей клас дозволяє розробникам легко інтегрувати висотні карти у свої проекти, забезпечуючи більш реалістичний та динамічний рендеринг сцен.

3.3 Імплементація інтерфейсу користувача

Розробка програмного забезпечення для генерації сітки рельєфу з карт висот є складним завданням, що потребує як глибоких технічних знань, так і розробки інтуїтивно зрозумілого користувацького інтерфейсу. Бібліотека ImGui (Immediate Mode Graphical User Interface) є потужним інструментом, що дозволяє розробникам швидко створювати гнучкі та інтерактивні елементи інтерфейсу користувача (рисунк 3.21), полегшуючи процес взаємодії з програмою.

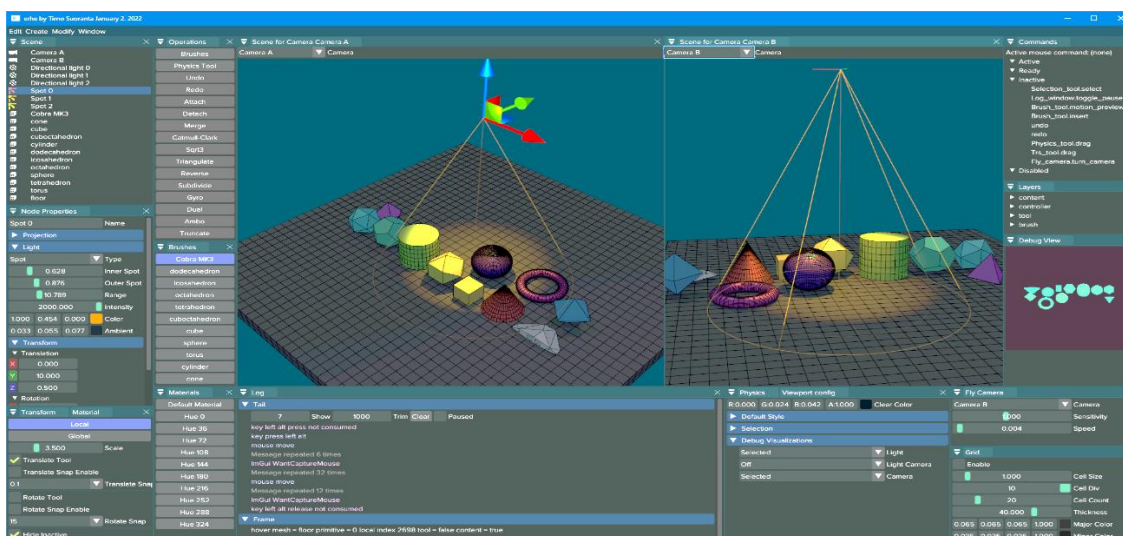


Рисунок 3.21 – Приклад інтерактивного інтерфейсу ImGui

Джерело: [16]

Основна перевага ImGui полягає в її простоті інтеграції та використання. Ця бібліотека надає можливість створення динамічних інтерфейсних елементів, які

реагують на зміни в режимі реального часу, що є критично важливим для додатків, які працюють з тривимірними моделями та великими обсягами даних.

Використання ImGui у програмі генерації сітки рельєфу з карт висот сприяє створенню високоякісного інтерфейсу, який надає користувачам такі переваги:

- інтуїтивна взаємодія: Бібліотека дозволяє створювати прості у використанні елементи інтерфейсу (рисунок 3.22), що полегшують процес налаштування параметрів та взаємодії з тривимірною моделлю рельєфу;
- гнучкість: Завдяки гнучкості ImGui, розробники можуть швидко додавати нові функції та адаптувати інтерфейс під потреби користувачів без значних витрат часу та ресурсів;
- продуктивність: Використання цієї бібліотеки мінімізує затрати на рендеринг інтерфейсних елементів, що дозволяє зосередитися на основних функціях програми та забезпечує високу продуктивність навіть при обробці великих обсягів даних;

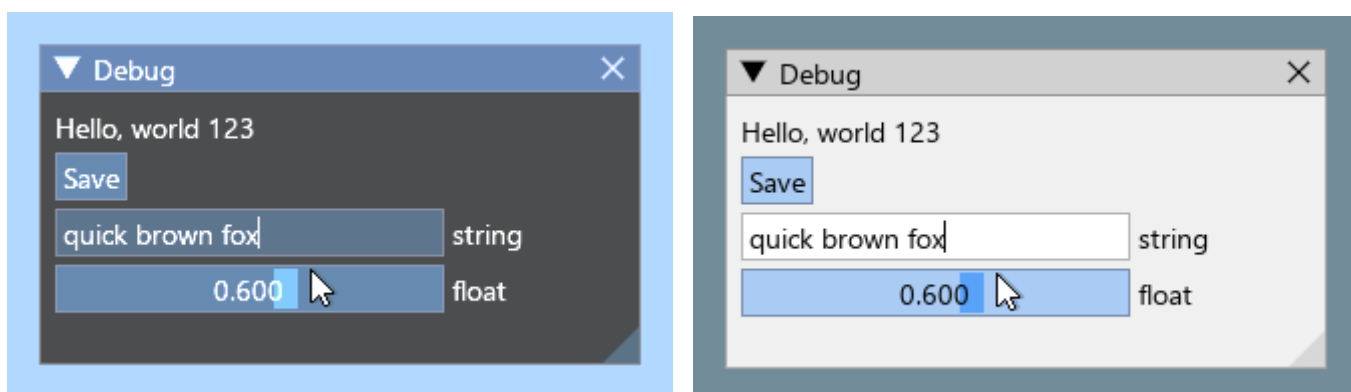


Рисунок 3.22 – Приклад інтерфейсу ImGui

Джерело: [16]

Впровадження ImGui в процес розробки програми для генерації сітки рельєфу з карт висот дозволяє створити професійний, функціональний та ефективний інтерфейс користувача. Це, в свою чергу, підвищує задоволеність користувачів і покращує загальний досвід взаємодії з програмою. Завдяки можливостям, що надає ImGui, розробники можуть забезпечити високу якість продукту, зосередившись на його основному функціоналі та інноваційних рішеннях.

Розроблено декілька важливих блоків інтерфейсу, кожен з яких виконує специфічні функції та забезпечує різні аспекти керування і налаштування:

- Camera Control (рисунок 3.23) - цей модуль відповідає за керування камерою, дозволяючи користувачеві детально роздивитися згенерований меш рельєфу. Завдяки цьому блоку, користувач має змогу змінювати положення та кут огляду камери, що забезпечує повну інтерактивність та детальний аналіз створеного рельєфу. Такий підхід значно полегшує процес вивчення та візуалізації тривимірних моделей;

- Heightmap Settings (рисунок 3.24) - цей модуль включає в себе ряд параметрів, які можуть бути змінені користувачем для досягнення оптимального результату. Основні налаштування цього блоку включають: Вибір карти висот - користувач може обирати з кількох доступних карт висот, з яких буде згенеровано меш. Це дозволяє гнучко налаштовувати вихідні дані для рельєфу. Перемикач "Wireframe mode" - цей параметр дозволяє користувачеві обрати режим рендерингу мешу як сітки. Режим "Wireframe" (рисунок 3.25) дає можливість побачити структуру мешу, що є корисним для технічного аналізу та виправлення можливих недоліків у моделі. "Level of Detail" (Рівень деталізації) - цей параметр надає можливість регулювати кількість вершин, що формують меш рельєфу. Збільшення рівня деталізації покращує якість зображення, проте може впливати на продуктивність системи. Тому користувач може обрати баланс між якістю та продуктивністю залежно від своїх потреб. "Height Factor Color Component" - цей параметр дозволяє вибрати компонент кольору (R - червоний, G - зелений, B - синій), який буде використовуватись як фактор висоти на карті висот. Такий підхід забезпечує додаткову гнучкість у налаштуванні візуальних характеристик рельєфу та дозволяє більш точно відображати необхідні деталі. Крім того, в модулі передбачені додаткові параметри для більш тонкого налаштування, такі як масштабування висоти та зміщення по осі Z, що розширюють можливості користувача у створенні детальних і реалістичних рельєфів;



Рисунок 3.23 – Camera control модуль

Джерело: запропоновано автором

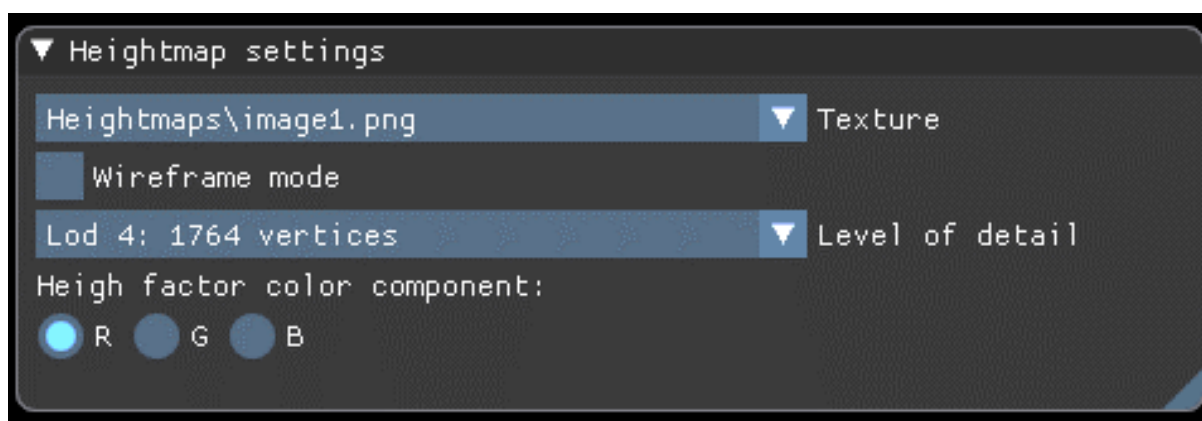


Рисунок 3.24 – Модуль Heightmap Settings

Джерело: запропоновано автором

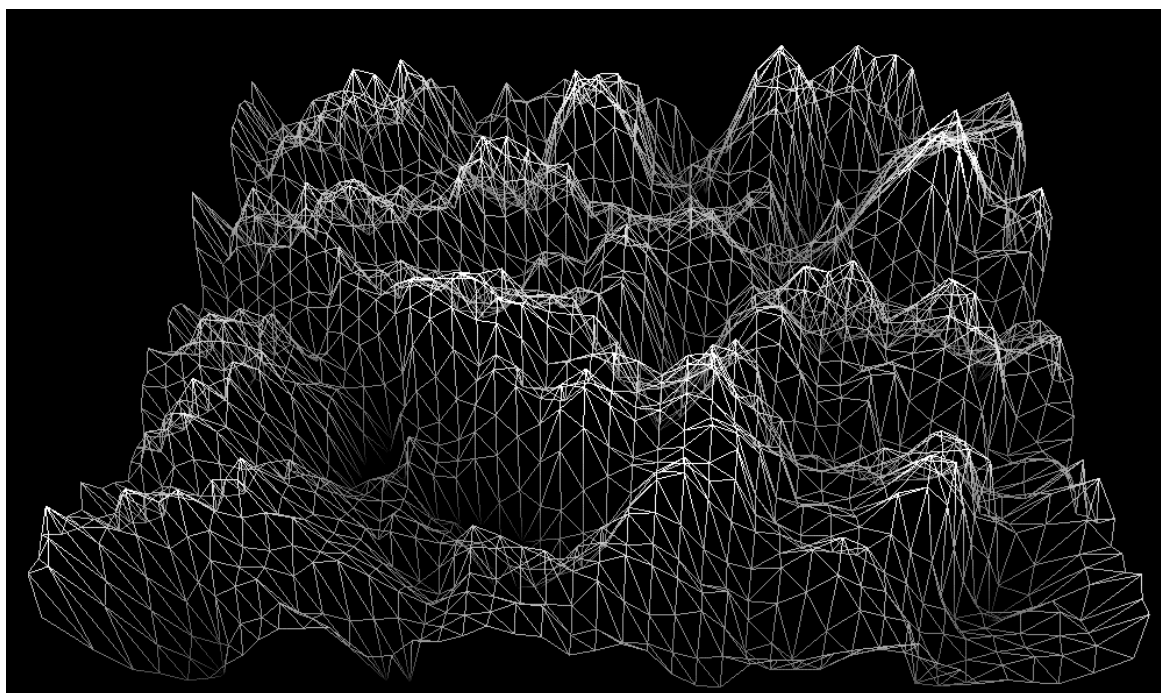


Рисунок 3.25 – Режим "Wireframe"

Джерело: запропоновано автором

Ці блоки інтерфейсу разом утворюють комплексний інструмент (рисунок 3.26) для створення, аналізу та налаштування тривимірних моделей рельєфу, забезпечуючи користувачеві повний контроль над процесом та результатом.

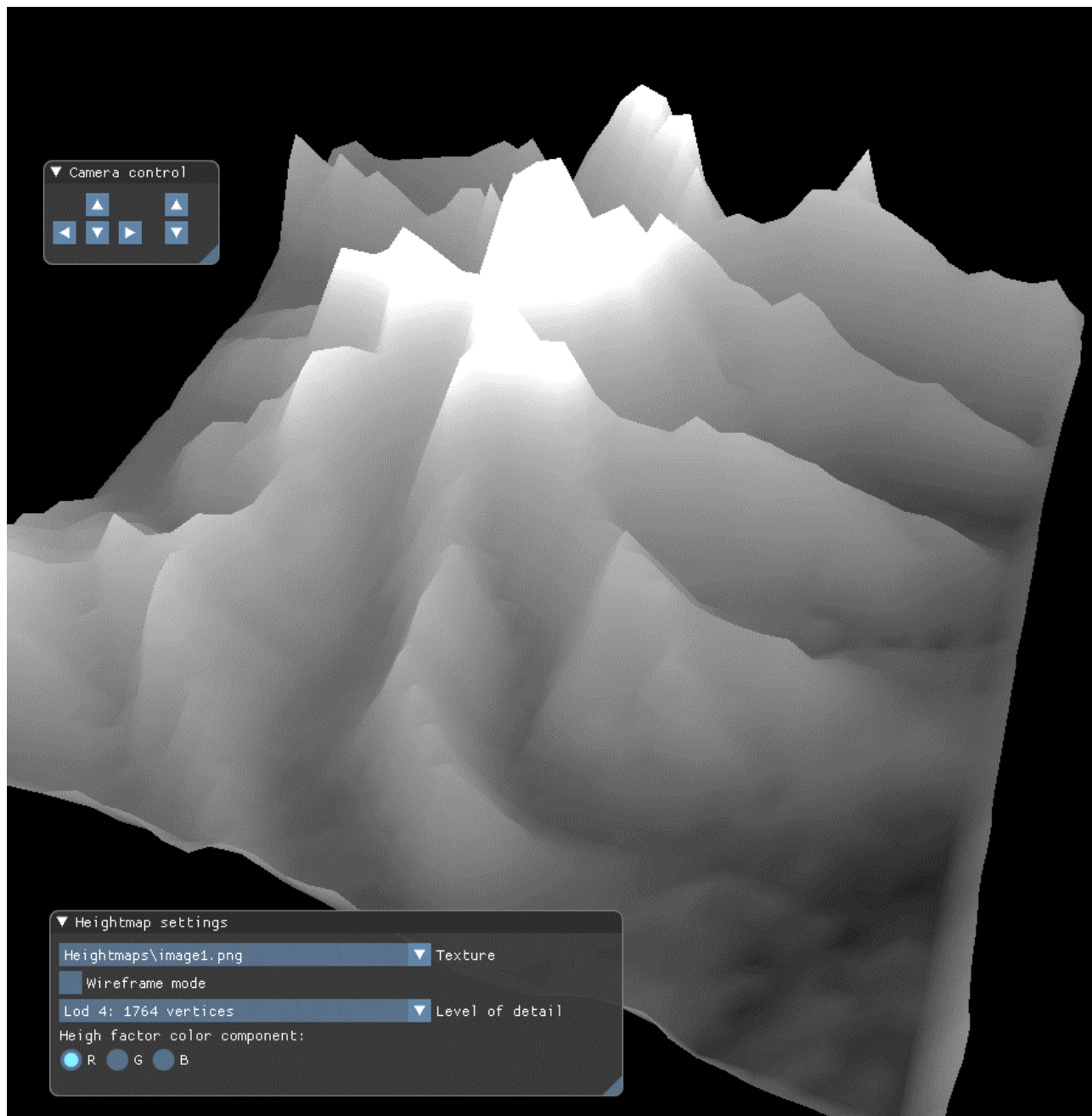


Рисунок 3.26 – Вікно генератору мешу рельєфу

Джерело: запропоновано автором

Інтерфейс забезпечує гнучкість та зручність у роботі, дозволяючи легко змінювати параметри та налаштування моделі.

ВИСНОВКИ

У процесі роботи були встановлені вимоги до системи та розглянуто її функціональні можливості. Також був проведений аналіз схожих програмних рішень, оцінені їх переваги та недоліки. Генератор розроблений з використанням сучасних технологій, таких як OpenTK, C#, .NET та ImGui, що дозволило створити ефективний та продуктивний додаток. Інтерфейс користувача був розроблений з метою забезпечити його інтуїтивну зрозумілість та зручність використання при виконанні різних операцій з генерації 3D контенту.

Розроблений генератор меша рельєфу місцевості включає різні модулі, які виконують такі функції, як завантаження висотних карт, генерація тривимірних мешів та їх візуалізація. Крім того, система надає можливість збереження згенерованих мешів у різних форматах, що сприяє зручному використанню та інтеграції з іншими системами. В рамках розробки були створені прототипи інтерфейсу користувача для сторінок завантаження висотних карт та візуалізації згенерованих мешів, що покращує загальний досвід користувача та полегшує його взаємодію з системою. Також була проведена ретельна перевірка та тестування всіх функціональних модулів, щоб гарантувати стабільність та надійність роботи генератора. Це дозволило виявити та усунути потенційні проблеми, покращуючи таким чином загальну якість додатка.

В результаті роботи було досягнуто поставленої мети і розроблений функціональний генератор меша рельєфу місцевості, який відповідає вимогам та потребам користувачів. Цей генератор може бути використаний в різних галузях і допомагати ефективно створювати та аналізувати тривимірний рельєф місцевості.

Подальший розвиток генератора меша рельєфу місцевості може включати:

- оптимізація продуктивності - впровадження механізму кешування для зменшення навантаження на систему, покращення алгоритмів генерації для швидшого виконання запитів;
- підвищення безпеки - реалізація шифрування даних як при передачі (SSL / TLS), так і при зберіганні на локальних пристроях;

- покращення взаємодії з користувачем - розробка адаптивного дизайну для правильного відображення на мобільних пристроях, додавання інтерактивних елементів, таких як діаграми, для візуального представлення даних;

- розширені можливості - впровадження модуля аналізу для створення аналітичних звітів на основі зібраних даних;

Запропоновані покращення підвищують продуктивність, безпеку та зручність використання генератора, якість створення 3D контенту та ефективність управління даними. В результаті роботи було досягнуто поставлених цілей і розроблений функціональний генератор меша рельєфу місцевості, який відповідає вимогам та потребам користувачів у різних галузях, сприяючи ефективному створенню та аналізу тривимірного рельєфу місцевості.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Terrain Creation Gaea/Blender. Blender Artists Community. URL: <https://blenderartists.org/t/terrain-creation-gaea-blender/1413092> (дата звернення: 11.05.2024).
2. Finding an automated workflow in 3D design and optimization - Meshmatic. Meshmatic. URL: <https://meshmatic3d.com/technical/complex-ui-and-automated-optimization/> (дата звернення: 11.05.2024).
3. Jake Denham. The 3ds Max User Interface, 2021. YouTube. URL: <https://www.youtube.com/watch?v=xga1OYsL0QM> (дата звернення: 11.05.2024).
4. Microsoft. Visual Studio Code - Code Editing. Redefined. Visual Studio Code - Code Editing. Redefined. URL: <https://code.visualstudio.com/> (дата звернення: 12.05.2024).
5. .NET | Build. Test. Deploy. Microsoft. URL: <https://dotnet.microsoft.com/en-us/> (дата звернення: 12.05.2024).
6. Security in .NET - .NET. Microsoft Learn: Build skills that open doors in your career. URL: <https://learn.microsoft.com/en-us/dotnet/standard/security/> (дата звернення: 12.05.2024).
7. Visual Studio IDE with .NET - Develop Any App Using C#, F#, VB. Visual Studio. URL: <https://visualstudio.microsoft.com/vs/features/net-development/> (дата звернення: 13.05.2024).
8. Microsoft. .NET and Visual Studio Code. Visual Studio Code - Code Editing. Redefined. URL: <https://code.visualstudio.com/docs/languages/dotnet> (дата звернення: 12.06.2024).
9. .NET developer community | Meet with like-minded developers. Microsoft. URL: <https://dotnet.microsoft.com/en-us/platform/community> (дата звернення: 12.06.2024).

- 10.OpenGL - Wikipedia. Wikipedia, the free encyclopedia. URL: <https://en.wikipedia.org/wiki/OpenGL> (дата звернення: 12.06.2024).
- 11.OpenTK - OpenTK. OpenTK. URL: <https://opentk.net/> (дата звернення: 12.06.2024).
- 12.An introduction to the Dear ImGui library. Conan C/C++ Package Manager Official Blog. URL: <https://blog.conan.io/2019/06/26/An-introduction-to-the-Dear-ImGui-library.html> (дата звернення: 12.06.2024).
- 13.Six Labors : ImageSharp. Six Labors : Six Labors. URL: <https://sixlabors.com/products/imagesharp/> (дата звернення: 12.06.2024).
- 14.Учасники проєктів Вікімедіа. Блок-схема – Вікіпедія. Вікіпедія. URL: <https://uk.wikipedia.org/wiki/Блок-схема> (дата звернення: 12.06.2024).
- 15.C++ Game Development By Example. Packt Subscription | Advance your knowledge in tech. URL: <https://subscription.packtpub.com/book/game-development/9781789535303/3/ch03lvl1sec12/render-pipeline> (дата звернення: 12.06.2024).
- 16.GitHub - ocornut/imgui: Dear ImGui: Bloat-free Graphical User interface for C++ with minimal dependencies. GitHub. URL: <https://github.com/ocornut/imgui?tab=readme-ov-file> (дата звернення: 12.06.2024).

ДОДАТКИ

ДОДАТОК А

ТЕХНІЧНЕ ЗАВДАННЯ

**На розробку генератора 3D контенту на прикладі генератору меша рельєфу
місцевості**

Погоджено:

Кандидат технічних наук, доцент кафедри
кібернетики та інформатики

Студент групи ІСТ 2001 с.т.

Ковальов Роман Станіславович

СУМИ 2024

1. Призначення й мета створення програми

1.1 Призначення програми

Програма призначена для генерації 3D контенту, зокрема, створення мешів рельєфу місцевості на основі висотних карт.

1.2 Мета створення програми

Розробка інструменту для автоматичного створення 3D моделей рельєфу місцевості, що дозволить користувачам швидко генерувати та візуалізувати меші рельєфу для різних застосувань, включаючи ігри, симуляції та картографію.

1.3 Цільова аудиторія

Цільова аудиторія програми включає:

1. Інженерів-геодезистів
2. Розробників генераторів 3D контенту
3. Архітекторів та дизайнерів
4. Розробників ігор

2 Вимоги до програми

2.1 Вимоги до програми в цілому

2.1.1 Вимоги до структури й функціонування програми

Програма повинна бути реалізована у вигляді настільного застосунку з інтуїтивно зрозумілим інтерфейсом користувача. Інтерфейс має складатися з двох модулів: для управління камерою та перегляду моделі, а також для налаштування параметрів генерації мешів.

2.1.2 Вимоги до персоналу

Від персоналу не має вимагатися особливих технічних навичок для експлуатації програми, окрім загальних навичок роботи з персональним комп'ютером і базових знань про 3D графіку.

2.1.3 Вимоги до збереження даних

Уся інформація, створена у програмі, буде зберігатися у локальних файлах або базах даних за вибором користувача.

2.1.4 Вимоги до розмежування доступу

Програма не вимагає обов'язкового розмежування доступу, але має підтримувати можливість роботи з кількома користувачами через локальні облікові записи.

2.2 Структура програми

2.2.1 Навігація

Інтерфейс програми повинен мати зручну систему меню для швидкого переміщення між модулями та налаштуваннями.

2.2.3 Наповнення програми (контент)

Усі сторінки та модулі програми повинні бути створені на основі сучасних шаблонів дизайну, що забезпечують зручність використання та налаштування параметрів без необхідності програмування.

2.3 Вимоги до функціонування системи

2.3.1 Потреби користувача

Адміністративна частина програми повинна надавати можливість додавання, редагування й видалення налаштувань генерації та візуалізації 3D моделей.

2.3.2 Функціональні вимоги

На головній сторінці інтерфейсу будуть представлені такі елементи:

- Меню навігації
- Вікно перегляду 3D моделі
- Панель налаштувань параметрів генерації мешів

2.3.3 Системні вимоги

Системні вимоги відповідно до потреб користувачів:

- Наявність функцій для управління камерою та перегляду 3D моделі
- Налаштування параметрів генерації мешів (висотні карти, опції LOD, режим каркасу тощо)
- Збереження та завантаження конфігурацій

2.4 Вимоги до видів забезпечення

2.4.1 Вимоги до інформаційного забезпечення

Реалізація програми відбувається з використанням:

- C#
- .NET Framework
- OpenTK
- GLSL

2.4.2 Вимоги до лінгвістичного забезпечення

Програма має бути виконана англійською мовою.

2.4.3 Вимоги до програмного забезпечення

Програмне забезпечення клієнтської частини повинне задовольняти наступним вимогам:

- Сумісність з операційними системами Windows 7 і вище
- Графічний прискорувач з підтримкою OpenGL 3.3

3 Склад і зміст робіт зі створення програми

Докладний опис етапів роботи зі створення програми наведено в таблиці А.1.

Таблиця А.1 – Етапи створення програми

№	Склад і зміст робіт	Строк розробки
1	Аналіз та планування	4 дні
2	Вивчення аналогів	2 дні
3	Вибір технологічного стеку	3 дні
4	Розробка функціональності	10 днів
5	Розробка інфраструктури	5 днів
6	Розробка бази даних	3 дні
7	Розробка UI/UX	7 днів
8	Комплексне тестування	5 днів
9	Виправлення помилок та вдосконалення	2 дні
10	Впровадження та підтримка	3 дні
	Загальна тривалість робіт	44 дні

4. Вимоги до складу й змісту робіт із введення програми в експлуатацію

Для того, щоб програмою могли користуватися різні категорії користувачів, необхідно забезпечити її розповсюдження через офіційні канали, такі як власний вебсайт або платформи дистрибуції програмного забезпечення.

ДОДАТОК Б
КОПІЇ ПУБЛІКАЦІЙ

UDC 001.1

The 5th International scientific and practical conference “Modern science: innovations and prospects” (February 6-8, 2022) SSPG Publish, Stockholm, Sweden. 2022. 771 p.

ISBN 978-91-87224-02-7

The recommended citation for this publication is:

Ivanov I. Analysis of the phaunistic composition of Ukraine // Modern science: innovations and prospects. Proceedings of the 5th International scientific and practical conference. SSPG Publish. Stockholm, Sweden. 2022. Pp. 21-27. URL: <https://sci-conf.com.ua/v-mezhdunarodnaya-nauchno-prakticheskaya-konferentsiya-modern-science-innovations-and-prospects-6-8-fevralya-2022-goda-stokgolm-shvetsiya-arhiv/>.

Editor

Komarytsky M.L.

Ph.D. in Economics, Associate Professor

Collection of scientific articles published is the scientific and practical publication, which contains scientific articles of students, graduate students, Candidates and Doctors of Sciences, research workers and practitioners from Europe, Ukraine, Russia and from neighbouring countries and beyond. The articles contain the study, reflecting the processes and changes in the structure of modern science. The collection of scientific articles is for students, postgraduate students, doctoral candidates, teachers, researchers, practitioners and people interested in the trends of modern science development.

e-mail: sweden@sci-conf.com.ua

homepage: <https://sci-conf.com.ua>

©2022 Scientific Publishing Center “Sci-conf.com.ua” ®

©2022 SSPG Publish ®

©2022 Authors of the articles

РОЗРОБКА ГЕНЕРАТОРУ 3D КОНТЕНТУ НА ПРИКЛАДІ ГЕНЕРАТОРУ МЕША РЕЛЬЄФУ МІСЦЕВОСТІ

В'юненко Олександр Борисович

к.е.н., доцент

Ковальов Роман Станіславович

Студент

Сумський національний аграрний університет

м. Суми, Україна

Вступ. Розвиток комп'ютерної графіки, а саме в сферах суміжних з комп'ютерною графікою, таких як кінематограф, аугментована реальність, симуляційні системи, відео ігри, тощо — відкриває широке поле для поліпшення процесів створення або генерації контенту чи наповнення відповідного продукту візуальним змістом.

Мета роботи. На створення подібних ассетів вручну, будь то зображення місцевості в симуляції, кінематографі або відео грі, зазвичай витрачається велика кількість часу та людських ресурсів, що породжує природний потенціал для розробок, покращень та нових ідей. Такі ідеї та розробки у свою чергу можуть бути практично корисні для різноманітних проєктів, зменшуючи витрати часу та ресурсів на створення віртуальних оточень.

Мета роботи - створення прототипу рішення генерації 3d контенту на прикладі генератору меша рельєфу місцевості.

Матеріали та методи. Генерація сітки мешу - це практика створення сітки, підрозділи безперервного геометричного простору на дискретні геометричні та топологічні осередки. Часто ці клітини утворюють симпліційний комплекс. Зазвичай осередки розбивають геометричну вхідну область. Комірки сітки використовуються як дискретні локальні апроксимації більшої області.

Сітки створюються за допомогою комп'ютерних алгоритмів, часто під керівництвом людини через графічний інтерфейс, залежно від складності

предметної області та бажаного типу сітки.

Результати та обговорення. Одним із результатів дослідження є додаток - генератор меша рельєфу місцевості. Додаток - конвеєр приймає на вхід rgb біт-карту(Рис. 1), простіше кажучи - зображення форматів png, jpeg, bmp, gif, tga чи jpg.

Зображення має являти собою графічне визначення висоти, де біт з вищим числом суми компонентів кольору (Red, Green, Blue) символізує вертикальну відстань точки рельєфу від певної "нульової" точки. Сума компонентів кольору становлять білий колір - тому проглядається проста закономірність: яскравіший фрагмент біт-карти - вище за точку рельєфу.

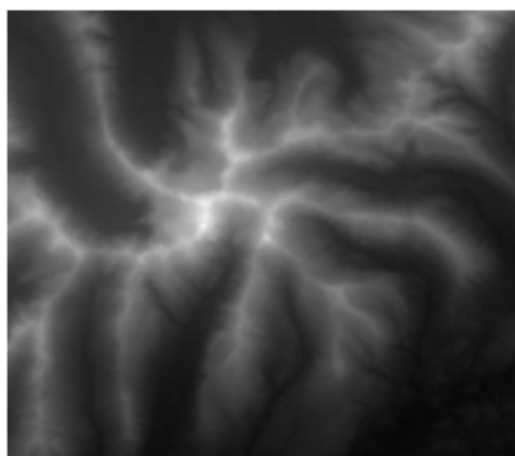


Рис.1 Біт карта рельєфу

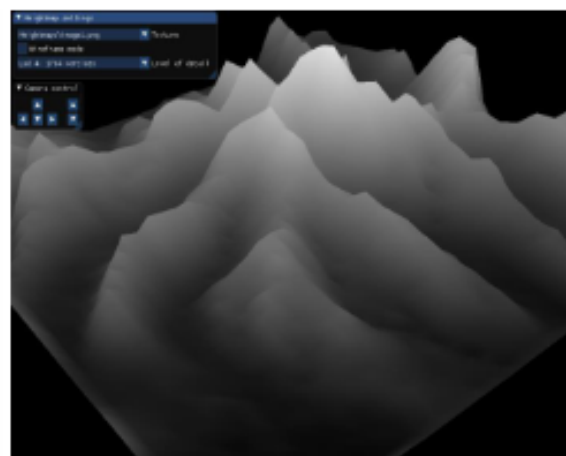


Рис. 2 Готовий 3Д меш рельєфу

Далі формується площина з рівновіддалених між собою точок у тривимірному просторі. Кількість згенерованих точок, або вершин, визначатимуть так званий LOD - Level Of Detail - рівень деталізації об'єкта. Вищевказаний масив вершин відправляється на зберігання в VRAM - Video Random Access Memory - пам'ять графічного прискорювача випадкового доступу, звідки потім вони будуть відмальовані в простір(Рис. 2) за допомогою OpenGL (Open Graphics Library) - специфікації, що визначає платформонезалежний (незалежний від мови програмування) програмний інтерфейс для написання додатків, що використовують двовимірну та тривимірну комп'ютерну графіку.

Висновки. Як і для будь-якої графіки для створення 3д сцен та оточень необхідні ресурси, так звані ассети, на створення яких вручну зазвичай йде дуже багато людських ресурсів та часу. Генерація таких ассетів абсолютно змінює правила гри. Створення штучних ресурсів з допомогою генераторів контенту драматично зменшить час що витрачається на створення оточень.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

СУМСЬКИЙ НАЦІОНАЛЬНИЙ
АГРАРНИЙ УНІВЕРСИТЕТ

МАТЕРІАЛИ

науково-практичної конференції
викладачів, аспірантів та студентів
Сумського НАУ

(14-16 травня 2024 р.)

ПРАКТИЧНЕ ПРОГРАМУВАННЯ ГЕНЕРАТОРІВ 3D КОНТЕНТУ НА ПРИКЛАДІ ГЕНЕРАТОРУ МЕША РЕЛЬЄФУ МІСЦЕВОСТІ

Ковальов Р. С., студ. 4 курсу ФЕІМ, спец. «Інформаційні системи та технології»
 Науковий керівник: к.е.н., доц. О.Б. В'юненко
 Сумський НАУ

Вступ. У світі віртуальної реальності, візуалізації даних та комп'ютерних ігор, генерування 3D контенту стає все більш важливою задачею. Подолати складнощі, пов'язані з реалістичним відображенням об'єктів та ландшафтів, можливо завдяки застосуванню принципів практичного програмування генераторів.

Мета, – розглянути процес розробки та впровадження генератора мешів рельєфу місцевості як прикладу практичного програмування генераторів 3D контенту на основі вхідних даних. Генератори мешів рельєфу місцевості використовуються для створення реалістичних ландшафтів у віртуальних середовищах або інформаційних візуалізаціях, які можуть бути використані в симуляціях, візуалізаційних проектах або ігровій індустрії. Основним завданням є дослідження методів практичного програмування, що дозволяють створювати генератори мешів рельєфу місцевості з високою ефективністю та якістю. Це включає в себе вивчення алгоритмів генерації топографічних карт, методів генерації текстур та апроксимації геологічних формацій. Крім того, теза розглядає практичні аспекти розробки, такі як оптимізація алгоритмів, використання паралельних обчислень та інтеграція з іншими системами. Досліджуються також підходи до візуалізації та інтерактивного взаємодії зі згенерованим контентом.

Одним з ключових аспектів проекту є алгоритм генерації мешів, який базується на аналізі вхідної карти висот та використовує математичні методи для створення реалістичних топографічних форм. Карта висот (рисунк 1) використовується як основне джерело інформації для визначення висоти кожної точки мешу. Це дозволяє точно відтворити рельєф місцевості з урахуванням всіх варіацій висот (рисунк 2).

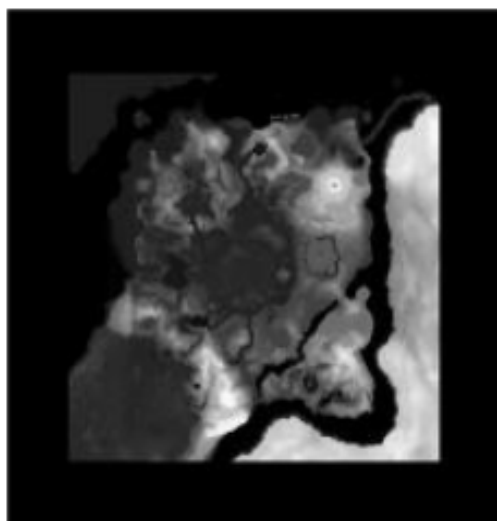


Рисунок 1 – Карта висот

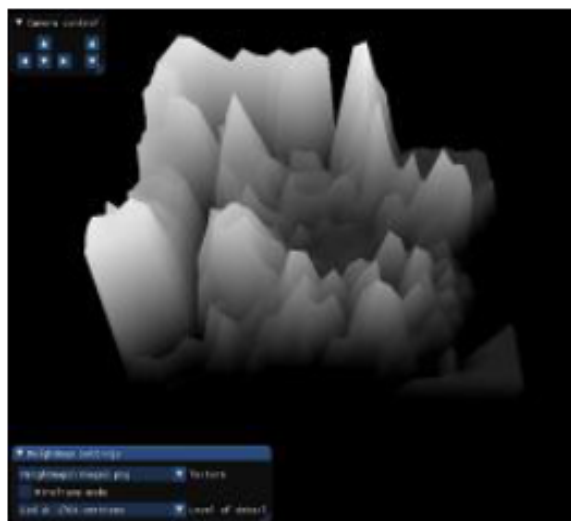


Рисунок 2 – Згенерований 3D меш

В результаті розробки рішення створено додаток, що виконує просту, базову генерацію мешу рельєфу місцевості. Додаток представляє інтерактивне візуальне зображення згенерованого контенту, з можливістю розглянути його в режимах "solid" - повний, цільний, чи "grid" або "frame" – режим сітки або каркасу. Додаток також надає можливість регенерації мешу в реальному часі з відмінними налаштуванням частоти вершин мешу. Дане налаштування також доступне в інтерфейсі користувача.

Висновок. На сьогоднішній день важко уявити світ без комп'ютерної графіки. У будь-якій галузі життя суспільства комп'ютерна графіка знаходить своє застосування. Архітектори використовують графіку під час проектування будівель, мультиплікатори - під час створення нових мультфільмів. Особливо важливу роль займає саме 3d графіка, особливо у світі, де навіть примітивні обчислювальні машини володіють графічним прискорювачем. Як і для будь-якої графіки для створення 3d сцен та оточень необхідні ресурси, так звані ассети, на створення яких вручну зазвичай йде дуже багато людських ресурсів та часу. Генерація таких ассетів абсолютно змінює правила гри. Створення штучних ресурсів драматично зменшить час що витрачається на створення оточень. Розроблене рішення є основою, мінімальним прикладом/прототипом інструменту створення таких штучних ресурсів.

ПРАКТИЧНЕ ПРОГРАМУВАННЯ ГЕНЕРАТОРІВ 3D КОНТЕНТУ НА ПРИКЛАДІ ГЕНЕРАТОРУ МЕША РЕЛЬЄФУ МІСЦЕВОСТІ

Ковальов Р. С., студ. 4 курсу ФЕІМ, спец. «Інформаційні системи та технології»
 Науковий керівник: к.е.н., доц. О.Б. В'юненко
 Сумський НАУ

Вступ. У світі віртуальної реальності, обробки даних та комп'ютерних ігор, генерація 3D контенту набуває все більшої вагомості. Подолання труднощів, пов'язаних із реалістичним відтворенням об'єктів та ландшафтів, можливе завдяки застосуванню принципів практичного програмування генераторів.

Матеріали та методи. Метою даної роботи є дослідження процесу розробки та впровадження генератора мешів рельєфу місцевості як прикладу практичного програмування 3D контенту на основі вхідних даних. Генератори мешів рельєфу місцевості використовуються для створення реалістичних ландшафтів у віртуальних середовищах або в інформаційних візуалізаціях, які використовуються у симуляціях, візуалізаційних проектах або в ігровій індустрії.

Основним завданням роботи є дослідження методів практичного програмування для створення генераторів мешів рельєфу місцевості з високою ефективністю та якістю. Це включає вивчення алгоритмів генерації топографічних карт, методів генерації текстур та апроксимації геологічних формацій.

Додатково, робота розглядає практичні аспекти розробки, такі як оптимізація алгоритмів, використання паралельних обчислень та інтеграція з іншими системами. Досліджуються також підходи до візуалізації та інтерактивної взаємодії зі згенерованим контентом.

Одним із ключових аспектів роботи є алгоритм генерації мешів, що базується на аналізі вхідної карти висот та використанні математичних методів для створення реалістичних топографічних форм. Результати дослідження показують, що розроблений додаток здатний проводити просту генерацію мешів рельєфу місцевості та надає можливість інтерактивної взаємодії зі згенерованим контентом.

• Огляд Проекту:

Створений для автоматизації процесу генерації 3D рельєфних мешів з використанням вхідної карти висот.

• Технологія Генерації Мешів:

Одним з ключових аспектів проекту є алгоритм генерації мешів, який базується на аналізі вхідної карти висот та використовує математичні методи для створення реалістичних топографічних форм.

• Використання Карт Висот:

Карта висот використовується як основне джерело інформації для визначення висоти кожної точки нашої меші. Це дозволяє точно відтворити рельєф місцевості з урахуванням всіх варіацій висот.

• Графічна Візуалізація:

Проект включає в себе графічну візуалізацію, що дозволяє побачити реалістичний вигляд згенерованого рельєфу в реальному часі. Це допомагає користувачам оцінити результати генерації перед їхнім використанням.

• Оптимізація та Швидкість:

Одним із пріоритетів є оптимізація алгоритмів для забезпечення швидкої та ефективної роботи генератора мешів. Використано ряд технік, таких як паралельні обчислення, для забезпечення оптимальної продуктивності.

• Застосування:

Проект може бути використаний у різних сферах, включаючи розробку комп'ютерних ігор, симуляції реального середовища та архітектурного візуалізації. Це робить його універсальним інструментом для будь-якого, хто працює з 3D контентом.

Висновок. У сучасному світі, де комп'ютерна графіка займає важливе місце, розроблене рішення є важливим кроком у створенні штучних ресурсів для 3D сцен та оточень. Генерація таких ассетів дозволяє ефективно використовувати ресурси та скорочує час, необхідний для створення оточень.

ДОДАТОК В

ЛІСТИНГИ ПРОГРАМ

```
using System;
```

```
using OpenTK.Graphics.OpenGL;
```

```
using OpenTK.Mathematics;
```

```
using OpenTK.Windowing.Desktop;
```

```
using OpenTK.Windowing.Common;
```

```
using Fwork;
```

```
using ImGuiNET;
```

```
namespace DataVisualisation3D_OpenTK_
```

```
{
```

```
    static class Program
```

```
    {
```

```
        static private GameWindow window;
```

```
        static private Dear_ImGui_Sample.ImGuiController imguiController;
```

```
        static void Main(string[] args)
```

```
        {
```

```
            GameWindowSettings gameWindowSettings = new  
GameWindowSettings();
```

```
            gameWindowSettings.RenderFrequency = 0;
```

```
            gameWindowSettings.UpdateFrequency = 60d;
```

```
            NativeWindowSettings nativeWindowSettings = new  
NativeWindowSettings();
```

```
            nativeWindowSettings.API =  
OpenTK.Windowing.Common.ContextAPI.OpenGL;
```

```

nativeWindowSettings.APIVersion = Version.Parse("3.3");
nativeWindowSettings.Size = new Vector2i(800, 600);
nativeWindowSettings.StartFocused = true;
nativeWindowSettings.Title = "Data Visualisation 3D";

window = new GameWindow(gameWindowSettings,
nativeWindowSettings);

window.VSync = VSyncMode.Off;
//-----

HeightMap hmap = new HeightMap();
//-----

Camera camera = new Camera();
camera.Mode = Camera.CameraMode.Orbiting;

ShaderProgram shaderProgram = new ShaderProgram();

window.Load += () =>
{
    imguiController = new
Dear_ImGui_Sample.ImGuiController(window.Bounds.Size.X,
window.Bounds.Size.Y);

    GL.Enable(EnableCap.DepthTest);

```

```

        shaderProgram.Load("3.3.heightmap.vert",
"3.3.heightmap.frag");

        GL.Uniform1(GL.GetUniformLocation(shaderProgram.Id,
"texture0"), 0);

    };

    window.Resize += (ResizeEventArgs args) =>
    {
        imguiController.WindowResized(window.Bounds.Size.X,
window.Bounds.Size.Y);
        GL.Viewport(0, 0, window.Size.X, window.Size.Y);
    };

    window.UpdateFrame += (FrameEventArgs args) =>
    {
        camera.Update(window.KeyboardState);
    };

    window.RenderFrame += (FrameEventArgs args) =>
    {
        GL.Clear(ClearBufferMask.ColorBufferBit |
ClearBufferMask.DepthBufferBit);
        imguiController.Update(window, (float)args.Time);

        GL.UseProgram(shaderProgram.Id);

```

```

camera.Apply();

hmap.Draw();


// ImGui.ShowDemoWindow();

ImGui.Begin("Camera control");
//////////////////// Camera controll buttons
///
ImGui.PushButtonRepeat(true);
ImGui.SameLine(35f);
if (ImGui.ArrowButton("forward", ImGuiDir.Up))
camera.MoveForward();
ImGui.SameLine(100f);
if (ImGui.ArrowButton("up", ImGuiDir.Up))
camera.MoveUp();
if (ImGui.ArrowButton("left", ImGuiDir.Left))
camera.MoveLeft();
ImGui.SameLine();
if (ImGui.ArrowButton("backward", ImGuiDir.Down))
camera.MoveBackward();
ImGui.SameLine();
if (ImGui.ArrowButton("right", ImGuiDir.Right))
camera.MoveRight();
ImGui.SameLine(100f);
if (ImGui.ArrowButton("down", ImGuiDir.Down))
camera.MoveDown();

////////////////////
ImGui.End();

```

```

        imguiController.Render();
        GL.Enable(EnableCap.DepthTest);

        window.SwapBuffers();
    };

    window.KeyDown += (KeyboardKeyEventArgs args) =>
    {
        imguiController.PressChar((char)args.Key);
    };

    window.Unload += () =>
    {
        GL.BindBuffer(BufferTarget.ArrayBuffer, 0);
        GL.BindVertexArray(0);
        GL.UseProgram(0);

        // Delete all the resources.
        hmap.Terminate();

        GL.DeleteProgram(shaderProgram.Id);
    };

    window.Run();
}
}

```

```

}
using System.Collections.Generic;

using OpenTK.Graphics.OpenGL;
using SixLabors.ImageSharp;
using SixLabors.ImageSharp.PixelFormats;

namespace Fwork
{
    public class Texture
    {
        private int id;
        public string Name { get; set; }
        public Texture(string path)
        {
            id = GL.GenTexture();

            GL.ActiveTexture(TextureUnit.Texture0);
            GL.BindTexture(TextureTarget.Texture2D, id);

            List<byte> colorData = new List<byte>();
            Image<Rgba32> image = Image.Load<Rgba32>(path);
            for (int y = image.Height - 1; y >= 0; y--)
            {
                for (int x = image.Width - 1; x >= 0; x--)
                {
                    colorData.Add(image[x, y].R);
                    colorData.Add(image[x, y].G);
                    colorData.Add(image[x, y].B);
                }
            }
        }
    }
}

```

```

        colorData.Add(image[x, y].A);
    }
}

GL.Texture2D(TextureTarget.Texture2D, 0,
PixelInternalFormat.Rgba, image.Width, image.Height,
0, PixelFormat.Rgba, PixelType.UnsignedByte,
colorData.ToArray());

GL.TextureParameter(TextureTarget.Texture2D,
TextureParameterName.TextureMinFilter, (int)TextureMinFilter.Linear);
GL.TextureParameter(TextureTarget.Texture2D,
TextureParameterName.TextureMagFilter, (int)TextureMagFilter.Linear);

GL.TextureParameter(TextureTarget.Texture2D,
TextureParameterName.TextureWrapS, (int)TextureWrapMode.Repeat);
GL.TextureParameter(TextureTarget.Texture2D,
TextureParameterName.TextureWrapT, (int)TextureWrapMode.Repeat);

GL.GenerateMipmap(GenerateMipmapTarget.Texture2D);
}

public void Use(TextureUnit unit)
{
    GL.ActiveTexture(unit);
    GL.BindTexture(TextureTarget.Texture2D, id);
}
}
}

using System;

```

```

using OpenTK.Graphics.OpenGL;

namespace Fwork
{
    class ShaderProgram
    {
        public int Id { get; set; }

        public void Load(string vertShaderPath, string fragShaderPath)
        {
            Id = GL.CreateProgram();

            Shader vertShader = new Shader(),
                fragShader = new Shader();

            vertShader.Load(vertShaderPath, ShaderType.VertexShader);
            fragShader.Load(fragShaderPath, ShaderType.FragmentShader);

            GL.AttachShader(Id, vertShader.Id);
            GL.AttachShader(Id, fragShader.Id);
            GL.LinkProgram(Id);
            GL.DetachShader(Id, vertShader.Id);
            GL.DetachShader(Id, fragShader.Id);
            GL.DeleteShader(vertShader.Id);
            GL.DeleteShader(fragShader.Id);

            string infoLog = GL.GetProgramInfoLog(Id);
            if (!string.IsNullOrEmpty(infoLog))

```

```
        throw new Exception(infoLog);
    }
}
}
```