НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Теплоенергетичний факультет

Кафедра автоматизації проектування енергетичних процесів і систем

До захисту допущено:

Завідувач кафедри

_____ Наталія АУШЕВА

«___»_____2022 р.

Дипломна робота

на здобуття ступеня бакалавра

спеціальності 122 «Комп'ютерні науки»

освітня програма «Комп'ютерний моніторинг та геометричне моделювання процесів і систем»

на тему: « Редактор батиметрії для сцени гідроакустичного експерименту »

Виконав:
студент IV курсу, групи ТМ-81
Савчук Богдан Ігорович _____

Керівник:
доцент,
Варава Іван Андрійович _____

Рецензент:

_____

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2022

Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет теплоенергетичний
Кафедра автоматизації проектування енергетичних процесів і систем
Рівень вищої освіти перший
спеціальність 122 «Комп’ютерні науки»
освітня програма «Комп’ютерний еколого-економічний моніторинг процесів та систем»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Наталія АУШЕВА
(підпис)
“___”_____2022р.

ЗАВДАННЯ
на дипломну роботу студенту

Савчуку Богдану Ігоровичу
(прізвище, ім’я, по батькові)

1. Тема роботи Редактор батиметрії для сцени гідроакустичного експерименту.
керівник роботи Варава Іван Андрійович, доцент
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджена наказом вищого навчального закладу від “__” травня 2022р.
№ _____

2. Строк подання студентом роботи _____
3. Вихідні дані до роботи: мова програмування C#, мова програмування GLSL, середовище розробки Visual Studio Blend, мова розмітки XAML.
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) Дослідити та проаналізувати існуючі рішення моделювання та обробки даних батиметрії, обґрунтувати алгоритм розробки програмного додатку, спроєктувати об’єкти системи, розробити програмне забезпечення, розробити інтерфейс користувача, зробити висновки за результатами роботи.
5. Перелік ілюстративного матеріалу: Діаграма класів, опис функціоналу додатку, сутності об’єктів, засоби розробки, результати, приклади роботи, висновки.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання “10” вересня 2021 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітки
1.	Затвердження теми роботи	10.09.21	
2.	Вивчення та аналіз задачі	02.04.22-08.04.22	
3.	Розробка архітектури та загальної структури системи	09.04.22-15.04.22	
4.	Розробка структур окремих підсистем	16.04.22-30.04.22	
5.	Програмна реалізація системи	01.05.22-22.05.22	
6.	Оформлення пояснювальної записки	23.05.22-28.05.22	
7.	Захист програмного продукту	26.05.22-27.05.22	
8.	Передзахист	06.06.22-09.06.22	
9.	Захист	20.06.22-30.06.22	

Студент _____ Савчук Б. І.
(підпис) (прізвище та ініціали,)

Керівник роботи _____ Варава І. А.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Метою даної роботи є розробка додатку для моделювання, редагування та відображення об’єктів батиметрії, а також застосування до них методів інтерполяції та тріангуляції алгоритмами Делоне.

Об’єктом дослідження є батиметрія та моделювання її об’єктів у розробленому додатку. У роботі наведено результати аналізу предметної області, застосування додатку для моделювання об’єктів батиметрії, опис розробленого програмного забезпечення, його проектування та програмна реалізація.

Обсяг дипломної роботи складає 122 сторінка, робота містить в собі 49 рисунків, 1 таблиця, 2 додатки та 20 посилань.

Ключові слова: C#, WPF, OpenGL, OpenTK, GLSL, БАТИМЕТРІЯ, ІНТЕРПОЛЯЦІЯ, ТРІАНГУЛЯЦІЯ ДЕЛОНЕ, JSON

ABSTRACT

The aim of this work is to develop an application for modeling, editing and mapping of batimetry objects, as well as the application of methods of interpolation and triangulation by Delaunay algorithms.

The object of research is batimetry and modeling of its objects in the developed application. The work describes the results of the analysis of the subject area, the usage of the application for modeling batimetry objects, the developed software, its design and software implementation.

The volume of the thesis consists of 122 pages, the work contains 49 figures, 1 table, 2 appendices and 20 references.

Keywords: C#, WPF, OpenGL, OpenTK, GLSL, BATIMETRY, INTERPOLATION, DELONE TRIANGULATION, JSON

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

C# – мова програмування.

ОС – операційна система.

.NET – платформа для розробки програмного забезпечення.

XAML – декларативна мова розмітки.

ООП – об’єктно орієнтовне програмування.

WPF – графічна підсистема.

Шейдер – програма для одного із ступенів графічного конвеєра.

OpenGL – специфікація програмного інтерфейсу для використання двовимірної та тривимірної графіки.

ПЗ – програмне забезпечення.

ВСТУП

На сьогоднішній час існує в велика кількість програмного забезпечення для моделювання об’єктів, але більшість з них або складна в освоєнні, або(та) надає управління даними незадовільне для користувача. Такі програми потребують великої кількості обчислювальних ресурсів, що при моделюванні в розробці ігор, створенні візуальних об’єктів кіно або при розробці, наприклад, деталей на підприємстві, можливо і є рентабельним в сенсі придбання оснащення для пришвидшення складних обчислень для модельованих об’єктів, але у випадку моделювання такого об’єкту як рельєф це не завжди так.

Для потенційного користувача є актуальним використання програмного продукту, функціонал та призначення якого представленні у вигляді простого в освоєнні інтерфейсу управління та виконання задач, які стосуються керування моделлю батиметрії.

Оскільки тематика об’єкту не є популярною на ринку програмного забезпечення, а саме існує лише невелика кількість спеціалізованих програм для редагування об’єкту батиметрії.

На жаль, такі програмні рішення, зазвичай можна отримати лише за вартість, величина якої розрахована на використання деякої кількості користувачів та не включають в себе рішень для придбання для особистого використання, що зменшує кількість задоволених користувачів та користувачів, які мають потребу в моделювання та керуванні даними батиметрії.

1 ПОСТАНОВКА ЗАДАЧІ РОЗРОБКИ РЕДАКТОРУ БАТИМЕТРІЇ

Для розробки програмного забезпечення необхідно чітко визначити його функціонал, діяльність користувача в системі та результат, який надаватиме програма в результаті роботи, а також систематизація даних всередині системи.

Була поставлена задача розробки програмного забезпечення, в якому користувач матиме можливість імпортувати та експортувати дані до та з системи, здійснювати перетворення, керування та візуальне зображення даних батиметрії.

1.1 Створення системи імпорту та експорту даних

Для зручного керування даними користувачу необхідно мати можливість отримати дані з системі у зручному для перегляду користувачем форматі. Для внесення та виведення даних до системи обрано використання формату JSON, що є зручним для перегляду людині та не є складним для подій серіалізації та десеріалізації з та до даних в системі.

Для реалізації таких події обрано використання фреймворк для мови C# під назвою Json.NET від Newtonesoft[9]. Він спеціально розроблений для перетворення даних з та в JSON форматі, постійно оновлюється, оптимізується та є легкий у використанні. Його реалізовані механізми задовольняють потреби для форматування даних.

1.2 Створення системи обробки даних

Для обробки даних в системі є необхідним врахування моделей поведінки даних різного формату, а також їх обробки та збереження в універсальному форматі для візуального їх представлення в інтерфейсі користувачеві, гнучкого їх використання на різних етап існування та в просте застосування різних модулях. Вхідними даними є файл у форматі JSON – характеристика об'єкту. Вихідними даними є візуальне зображення об’єкту та файл у форматі JSON – характеристика обробленого об’єкту.

1.3 Створення частини перетворення даних

Необхідно розробити систему, в якій буде об’єднане керування об’єктами системами та їх перетворення в залежності від типу керованого об’єкту. Система керування має бути написана на мові C#[4] із використанням фреймворків від ALGLIB[10] та Triangle.Net[11], які навіть у випадку некомерційного використання надають необхідний функціонал для перетворення даних об’єкту в бажаний користувачеві.

1.4 Створення інтерфейсу користувача

Інтерфейс користувача повинен надавати візуальне та інформаційне зображення даних об’єкту, над яким користувач здійснює перетворення, а також керування такими об’єктами та системою.

Конфігурація системи керування системи має динамічно відображати дані, а також панель його керування з набором налаштувань об’єкту та функцій для його обробки.

1.5 Висновок до розділу

У даному розділі було описано мету дипломної роботи, вимоги до розроблюваного додатка, представлено необхідний функціонал, що має бути реалізований в програмному забезпеченні для досягнення поставленої мети.

2 ОПИС ПРЕДМЕТНОЇ ОБЛАСТІ

Аналіз та прогнозування даних такого об’єкта як морське дно є не менш важливим аніж аналіз наземного рельєфу[13]. Вивчення об’єкту батиметрії пов’язане з геологією, географією та гідролокацію. Дані батиметрії подаються за допомогою системи координат, значення висоти яких рахується за допомогою середнього рівня приливів та відливів. Отримані в результаті дані через це можуть мати деяку похибку в свої значеннях, що погіршує якість аналізу та обробки таких даних.

Аналіз даних батиметрії використовується в багатьох сферах, серед них[12]:

1. Моделювання поведінки вод в океанах в залежності від погоди.
2. Моделювання генерації та розповсюдження цунамі.
3. Зсув ґрунту на берегах озер, річок, морів та океанів.
4. Геолокація морських суден, барж, катерів тощо.
5. Моделювання рельєфу морського дна для оцінки впливу землетрусів.

Для того, щоб зменшити похибку отриманих даних після дослідження рельєфу, користуються різними методами інтерполяції, апроксимації та тріангуляції Делоне[2], яку застосовую до множини точок, що описуються об’єкти нерегулярної сітки, зокрема останньої, оскільки складність в отриманні даних морського рельєфу вища ніж в отримання даних для наземного, оскільки водне середовище може впливати на якість отриманих даних, а також змінювати стан рельєфу в залежності від багатьох факторів, що може різко змінювати актуальність даних та моделей які базуються на їхньому аналізі. Якщо дані рівномірно розподілені та умовно формують регулярну сітку: з однаковим інтервалами між

сусідніми точками, то доцільно застосовувати інтерполяцію многочленом, наприклад кубічним, або квадратичним.

3 ЗАСОБИ РОЗРОБКИ РЕДАКТОРА БАТИМЕТРІЇ

Для розробки даного проекту було розглянуто багато мов програмування та додатків, в яких реалізовано принципи моделювання об’єктів. Для оцінки та вибору мови програмування та допоміжних ресурсів були обрані наступні критерії оцінювання:

1. Кроссплатформенність.
2. Спрямованість.
3. Системні потреби.
4. Складність встановлення та використання.
5. Степінь підтримки розробниками.

Мовою програмування для розробки усіх модулів програми була обрана мова C# із використанням платформи .Net 5.0. Ця мова програмування є гарним вибором для розробки додатків для операційної системи Windows, використовується для розробки корпоративних рішень та навіть для розробки ігор.

C#[4] це мова для об’єктно-орієнтовного програмування (ООП). Концепт дотримування принципів ООП дає змогу створювати типи та структури даних та використовувати набір стандартних функцій для них. Принципи ООП дозволяють групувати дані в об’єкти, що полегшує розділяти програму на менші частини, що в результаті дає змогу швидше будувати, керувати та об’єднувати різні елементи програми.

Користуючись ООП, є можливим керування внутрішніми атрибутами об’єктів без необхідності напряму звертатися до них, а також описувати поведінку об’єкта за допомогою декларації класів. ООП мови програмування дають змогу розробляти додатки, які легше тестувати, читати, давати оцінку та визначати

місцезнаходження помилок, що значить легший підхід до написання додатку на такій мові програмування.

Мова C# є високорівневою мовою програмування, бо її синтакс схожий на людську мову. Іншими словами, вона має високий рівень абстракції відносно машинного коду, що значить, що код написаний на даній мові має бути скомпільований для того, щоб головний пристрій керування системою міг розуміти її команди.

Високорівневі мови є корисними для розробників, оскільки такі мови мають значно простіший синтакс, ніж низькорівневі мови як C.

З іншого боку, одна із важливих можливостей таким мов як C є можливість доступу до пам'яті пристрою напряму, використовуючи спеціальні типи команд, які називаються вказівниками. В той час як C# – це здебільшого високорівнева мова програмування й тільки невелика кількість способів управління вказівниками є доступною для розробників.

Мова C#[14] сама по собі більше підходить для розробки додатків для операційної системи Windows. Завдяки широким можливостям .NET вона стає гнучкою мовою, яку можна використовувати на різних платформах.

.NET підтримує розробників C# з різними типами середовищ виконання такими як власний CLI від Microsoft. Середовищ виконання застосовуються в програмах із використанням .NET для перекладу коду написаний на мові C# у машинні інструкції на будь-якій із підтримуваних операційних систем.

Сумісність мов — це ще одна особливість .NET, яка робить програмування на C# зручним. Сумісність означає, що код C# може взаємодіяти з програмами, написаними сумісними мовами, такими як C++, F#, Visual Basic і Windows PowerShell. Це дозволяє розробникам використовувати різні мови під час розробки одного додатку.

Із використанням платформи .NET розробникам на мові C# також доступний потужний фреймворк .NET Core, розроблений для створення веб-сервісів і програм. Фреймворк поставляється з набором компонентів, які можна

налаштовувати, уніфікованих бібліотек класів та інших функцій, які значно прискорюють процес розробки.

Управління розподілом та використання пам’яті пристрою під час роботи додатку є одним із найважливіших завдань для продуктивності роботи програми. Для цього в мові C# є вбудований збірник сміття.

Збірник сміття – це система керування пам’яті, яка відстежує невикористані ініціалізовані екземпляри об’єкти та автоматично звільняє пам’ять від них. Під час розробки додатків зазвичай потрібно написати додаткову систему, щоб уникнути накопичування об’єктів, які вже не використовуються в пам’яті. Автоматичне керування пам’яттю пришвидшує роботу розробників та звільняє їх від написання команд для відновлення невикористаних об’єктів, очищення пам’яті та розподілу її для нових.

У мові C# змінна не може змінити свій тип у коді. Наприклад, якщо оголосили змінну IamJustAVariable як ціле число, то є неможливим надати їй будь-яке значення, крім недробового числа.

Безпека використання типу гарантує, що усі змінні будуть поводити себе передбачуваним чином, та дії з ними будуть можливі допоки вони відповідають типу. Код, що містить змінну IamJustAVariable, матиме доступ лише до тих слотів пам’яті, які призначені для цілих чисел. Використання такого підходу зменшує вірогідність помилками з’явитися.

Мова C# виконує перевірку типу під час компіляції, що носить назву статична типізація. Такий підхід допомагає виявляти помилки використання неправильного типу до повноцінного запуску програми. Однак, на момент написання поточна версія підтримує динамічне призначення типу. Це дозволяє створювати об’єкти як динамічні, але виявлення помилки буде лише можливим під час виконання програми.

Динамічний параметр надає розробникам більшу гнучкість на етапі розробки та спрощує взаємодії з фрагментами коду, що надходять з інших середовищ виконання.

Для розробки інтерфейсу був обраний ресурс Windows Form Presentations[15]. Він має низку переваг. По-перше, WPF має підтримку яскравих візуальних зображень та анімації. Також він є чудовою платформою для використання в програмах, де застосовуються різні типи медіа. Ще WPF підходить, коли існує потреба в розробці комплексного інтерфейсу для користувача або якщо потрібно динамічно завантажувати дані до інтерфейсу, або якщо потрібно динамічно отримувати дані з веб-служб, або при створенні настільного додатку з можливістю додання веб-навігації. WPF має широкий функціонал, а також велику кількість підтримуваних компонентів та засобів роботи з ним. Крім того можна виділити також наступні його переваги:

1. Розроблені програми легко масштабувати.
2. Розроблені програми можуть бути запущені навіть на таких старих операційних системах як Windows Vista.
3. Платформа використовую декларативний спосіб кодування елементів;
4. Низькі системні потреби:
    1) .NET Framework 3.0;
    2) DirectX9 сумісна відеокарта.
5. Можливість повторного використання коду.

XAML – це мова, яка базується на мові XML, де реалізовано можливість використання елементів текстових блоків, випадаючих списків, кнопок та інших елементів базового інтерфейсу. Також в ньому можна описувати стилі усіх компонентів: змінювати їх розмір, колір тощо. Інтерфейс на платформі WPF може бути написаний за допомогою мови XAML.

OpenGL – це найбільш поширений на адаптований для використання двовимірної та тривимірної графіки API[7], який використовується в великій кількості додатків. Цей API не залежить від окремих додатків та операційних систем так само й від наявності з’єднання з Інтернетом. OpenGL дає змогу розробникам програмного забезпечення створювати високопродуктивні графічні

додатки для моделювання, створення візуального контенту, розваг, розробки ігор, використовується під час розробки додатків для сфери мануфактури, медицини та віртуальної реальності. OpenGL надає можливість використання усіх особливостей останніх версій графічних пристроїв.

OpenTK – це бібліотека[8], що забезпечий високошвидкісний доступ до OpenGL в додатках на базі платформи .NET. OpenTK розроблений для полегшення роботи із API OpenGL на мові C# не впливаючи на продуктивність та є безпечним для використання, хоч і написаний на мові C та має прямий доступ до пам’яті пристрою під час роботи програми.

OpenTK не є бібліотекою з таким великим функціоналом, як ігровий двигун, фреймворк або двигун для моделювання графіки, або повноцінною аудіосистемою сам по собі. Замість цього його низькорівневий рівень реалізації дозволяє розробляти усі ці речі.

OpenTK також включає в собі математичну бібліотеку для стандартний типів даних у графіці такі як вектори та матриці, що дозволяє розробнику уникнути потреби розробки власних. Ці ти розроблені таким чином, щоб ними було зручно користуватись та в той же час їх робота була продуктивною. Весь функціонал OpenTK не потребує додаткових налаштувать на доступний одразу після встановлення бібліотеки.

Для роботи з WPF[20] інтерфейсами, саме використання графічного вікна OpenGL в ньому OpenTK має рішення у вигляді пакету OpenTK.GLWpfControl[19], встановлення якого дозволяє використовувати графічне вікно OpenGL як один із стандартних елементів інтерфейсу та керувати ним, а також у такого вікна є доступ до всього функціоналу основної бібліотеки OpenTK.

Для серіалізації да десеріалізації даних був обраний пакет Json.NET від Newtonsoft[9]. Цей пакет дозволяє зручно та швидко перетворювати об’єкти між платформою .NET і JSON та навпаки[18].

Крім цього він дозволяє створювати власні типи об’єктів прямо під час заповнення JSON файлу. Його реалізація за продуктивністю є швидшою[16] ніж

вбудований пакет для серіалізації даних в JSON форматі від .NET. А також його підтримують з версії .NET 2.0.

ALGLIB – це кроссплатформена бібліотека[10] чисельного аналізу та обробки даних. Вона забезпечую аналіз даних, оптимізацію нелінійні операції, інтерполяцію та лінійна й нелінійна підгонка методом найменших квадратів, алгоритми лінійної алгебри, прямі та ітеративні лінійні операції, метод FFT швидкого перетворення рівняння Фурьє та багато інших алгоритмів. ALGLIB доступний на мовах C++, C#, Delphi, а також здатний працювати на багатьох операційних системах включаючи Windows, POSIX, Linux тощо.

Triangle.NET – бібліотека[11], засоби якої дозволяють створювати двовимірні тріангуляцію методами Делоне та високоякісні сітчасті поверхні на основі даних точок або плоских прямолінійних графів.

Тріангуляція методами даної бібліотеки може бути виконана з наступними особливостями:

1. Обмежена тріангуляція Делоне плоских прямолінійних графів.
2. Алгоритми тріангуляції Делоне: інкрементний, розгорнутий та розділяй і володарюй.
3. Високоякісні трикутні сітки з обмеженням мінімального та максимального кута.
4. Згладжування сітки за допомогою центроїдальної теселяції Вороного.
5. Перенумерація вузлів графів.

4 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

Структура програмного додатку складається з графічного інтерфейсу користувача, який реалізований на мові XAML, та системи об’єктів для

управлінням графічного вікна OpenGL та об’єктів управління обміну даних між ними.

4.1 Опис компонентів програми

Зв’язки компонентів системи представлені за допомогою діаграми класів (рисунок 4.1).

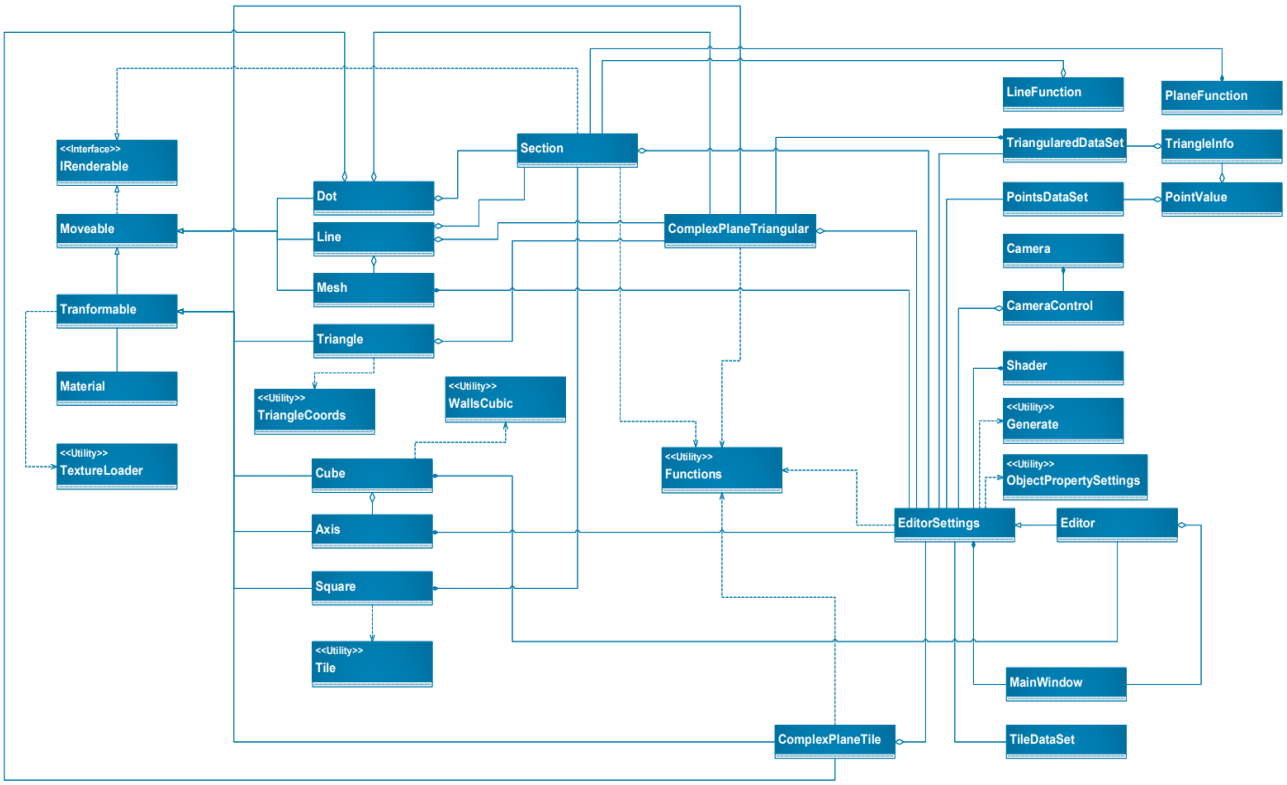

Рисунок 4.1 – Діаграма властивостей об’єкта Editor

Система втілює в собі можливість керування поточним об’єктом перегляду та перехід між ними. Також в системі реалізовано можливість переходу між декількома відкритими додатками, незалежних один від одного. Відображення елементі у графічному елементі сцени відбувається за допомогою інтерфейсу IRenderable. Кожен об’єкт, що може бути зображений на сцені є реалізацією даного інтерфейсу. Детальна діаграма IRenderable (Рисунок 4.2).

Рисунок 4.2 – Діаграма інтерфейсу IRenderable

У системі є наступні об’єкти, що є реалізацією даного інтерфейсу: Moveable, Section. Вони реалізують метод інтерфейсу під назвою Render(Shader, PrimitiveType?), за яким до цих об’єктів звертається об’єкт керування сцени при графічному відображенні цих елементів на сцені. Об’єкт Moveable є базовим для об’єктів, що можуть змінювати свою позицію на сцені. В той час як об’єкт Section не змінює свою позицію, але змінюється його структура побудови. Детальний опис об’єкта Moveable представлено на діаграмі (рисунок 4.3).

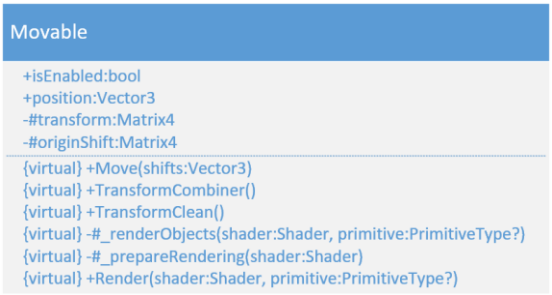

Рисунок 4.3 – Діаграма об’єкта Moveable

Кожен об’єкт, що є нащадком Moveable матиме наступні властивості та методи від нього:

- isEnabled – властивість, що зберігає значення чи буде відображений об’єкт на сцені;
- position – позиція об’єкта на сцені;
- transform – матриця перетворень об’єкта;
- originShift – позиція об’єкта від початку координат;

- Move(Vector3) – перемістити об’єкт у напрямку деякого вектора на величину цього вектора;
- TransformCombiner() – сформувати матрицю перетворень;
- TransformClean() – очистити будь-які перетворення об’єкту;
- _renderObjects(Shader, PrimitiveType?) – відобразити об’єкт;
- _prepareRenreding(Shader) – застосувати значення перетворень об’єкту для програми шейдеру;
- Render(Shader, PrimitiveType?) – підготувати та відобразити елементи об’єкту на сцені.

Крім того, що об’єкти можуть змінювати свою позицію, також існують елементи на сцені до яких застосовується додаткові перетворення аніж тільки переміщення, зокрема зміна кута нахилу по осях та зміна масштабування розміру елемента, що відображається. Базову структура такого об’єкта реалізовано в об’єкті Transformable, що є нащадком від об’єкта Moveable. Діаграму об’єкта Transformable зображено на (рисунок 4.4).

Рисунок 4.4 – Діаграма об’єкта Transformable

Об’єкт Transformable є базовим для об’єктів, що можуть змінювати своє положення відносно сцени, розмір та величину, що додатково втілюється в матриці перетворень transform, що наслідується цим об’єктом. Кожен об’єкт, що є нащадком від цього отримує наступні властивості та методи:

- textureHandlers – індикація текстур, що використовуються даним об’єктом;
- rotation – матриця, що зберігає в собі усі кути нахилу об’єкта по осях;
- scale – матриця, що зберігає в собі усі зміни розміру об’єкта;
- _range – вектор найменшого та найбільшого значень об’єкта по осі Oz (використовується для коректної побудови градієнтного змішування текстура для візуального представлення екстремумів об’єкту на сцені);
- parent – об’єкт, характеристики матеріалу та діапазону висоти якого використовуються для даного об’єкта;
- material – характеристика матеріалу об’єкту;
- Rotate(Vector3) – застосувати кути нахилу до об’єкта;
- Scale(Vector3) – застосувати зміну розмірів до об’єкта;
- TransformCombiner() – перевизначення методу базового об’єкта із врахуванням матриць зміни кутів та розмірів.
- TransformClean() – перевизначення методу базового об’єкта із врахуванням матриць зміни кутів та розмірів.
- _applyTextures(Shader) – застосування текстур для даного об’єкта;
- _applyMaterial(Shader) – застосування матеріалу для даного об’єкта;
- _prepareRenreding(Shader) – застосувати значення перетворень об’єкту для програми шейдеру та його діапазону висоти чи цього значення від властивості parent;
- Render(Shader, PrimitiveType?) – підготувати та відобразити елементи об’єкту на сцені.

Об’єкт Transformable надає об’єктам-нащадкам методи управління власними геометричними та візуальними характеристиками на сцені. Також цей об’єкт має асоціацію з об’єктом Material типу один до одному, в якому зберігаються

характеристики матеріалу об’єкту. Діаграма об’єкту Material зображена на (рисунок 4.5).

Рисунок 4.5 – Діаграма об’єкта Material

В цьому об’єкті представлені наступні властивості, що впливають на відображення елементу на сцені:

- ambient – степінь значень відбивання навколишнього світла;
- diffuse – степінь значень розсіювання навколишнього світла;
- opacity – степінь непрозорості об’єкта.

Ці значення впливають на характеристику зображення об’єкту, що користувач зможе бачити на сцені об’єктів.

Також об’єкт Transformable використовує додаткові функції допоміжного об’єкта TextureLoader. Його функціонал полягає в завантаженні та застосуванні текстур об’єктів. Діаграму об’єкта TextureLoader зображено на (рисунок 4.6).

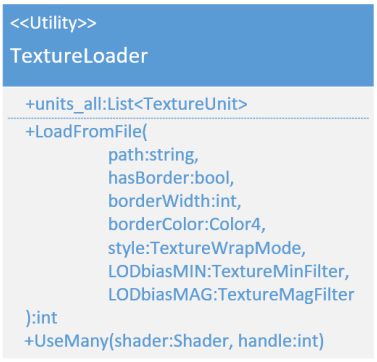

Рисунок 4.6 – Діаграма об’єкта TextureLoader

За допомогою цього об’єкта є можливим завантаження текстур із файлу у стандартних форматах растрових зображень, а також застосування багатьох текстур на програмі шейдеру водночас. Об’єкт TextureLoader має наступні властивості та методи:

- units_all – Список усіх текстурних елементів, що можуть бути використані водночас для одного об’єкта;
- LoadFromFile(string, bool, int, Color4, TextureWrapMode, TextureMinFilter, TextureMagFilter):int – метод завантаження файлу текстури, надання та прив’язання його індексного номеру як цільової текстури для подальшого використання;
- UseMany(Shader, int) – застосування деякої кількості текстур об’єкта на програмі шейдеру.

Для відображення переліку точок на сцені був створений об’єкт Dot, який є нащадком об’єкту Moveable. В ньому реалізовано елементи створення та зв’язування об’єкту точки для подальшого звернення до них на програмі шейдеру. Діаграму об’єкту Dot представлено на (рисунок 4.7).

Рисунок 4.7 – Діаграма об’єкта Dot

Об’єкт Dot має наступні властивості та методи:

- vertices – характеристика позиції точки в її локальних координатах та її колір;
- VBO – індекс цього елемента в контексті інших об’єктів;

- VAO – індекс характеристики цього елемента для вертексного шейдеру;
- size – розмір точки;
- Dot(Vector3, Vector3, float) – конструктор даного об’єкту;
- BindObject() – виділення та зв’язування індексів VBO та VAO;
- _renderObjects(Shader, PrimitiveType?) – відобразити об’єкт.

Крім точок, існують елементи на сцені, що мають бути відображеними за допомогою тільки ліній. Для таких потреб потрібно реалізувати клас Line, що є нащадком об’єкту Moveable. Діаграма об’єкту Line зображена на (рисунок 4.8).

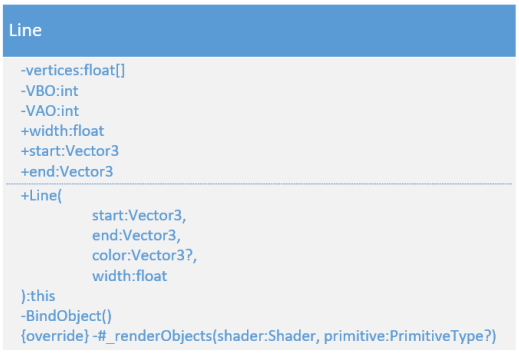

Рисунок 4.8 – Діаграма об’єкта Line

Об’єкт Line має наступні властивості та методи:

- vertices – характеристика позиції точки в її локальних координатах та її колір;
- VBO – індекс цього елемента в контексті інших об’єктів;
- VAO – індекс характеристики цього елемента для вертексного шейдеру;
- width – ширина лінії;
- start – позиція початку відрізку;
- end – позиція кінця відрізку;

- Line(Vector3, Vector3, Vector3?, float):this – конструктор даного об’єкту;
- BindObject() – виділення та зв’язування індексів VBO та VAO;
- _renderObjects(Shader, PrimitiveType?) – відобразити об’єкт.

Для полегшеної просторової візуалізації об’єктів на сцені необхідно створити об’єкт сітки Mesh. З його допомогою користувачу є легшим орієнтуватися в тривимірному просторі сцени та візуально сприймати масштаб різних об’єктів на сцені відносно один одного. Діаграма об’єкту Mesh зображена на (рисунок 4.9).

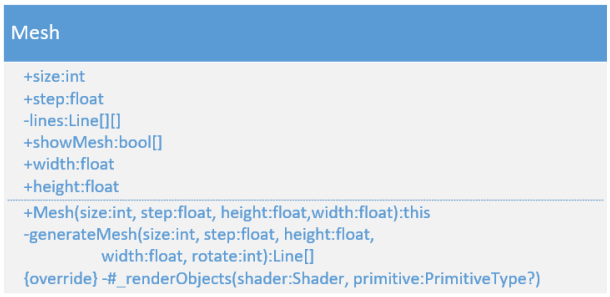

Рисунок 4.9 – Діаграма об’єкта Mesh

Об’єкт Mesh є нащадком об’єкту Moveable має наступні властивості та методи:

- size – довжина сторони області сітки;
- step – довжина кроку: відстань між сусідніми клітинками сітки;
- lines – масив сіток для трьох осей координат: Oxy, Oxz, Oyz;
- showMesh – значення чи потрібно відображати конкретні сітки;
- width – ширина лінії сітки;
- height – висота розміщення сітки;
- Mesh(int, float, height, width):this - конструктор даного об’єкту;
- generateMesh(int, float, float, float, int):Line[] – створення сітки для обраної координатної осі;

- _renderObjects(Shader, PrimitiveType?) – відобразити об’єкт.

Існує потреба у відображенні трикутних об’єктів на сцені. Для цього розроблено об’єкт Triangle, який є нащадком Transformable. Діаграма об'єкта Triangle зображена на (рисунок 4.10).

Рисунок 4.10 – Діаграма об’єкта Triangle

Об’єкт Triangle має наступні властивості та методи:

- vertices – характеристика позиції точки в її локальних координатах та її колір;
- VBO – індекс цього елемента в контексті інших об’єктів;
- VAO – індекс характеристики цього елемента для вертексного шейдеру;
- Triangle(Vector3[], Vector3?, string[]?, bool):this – конструктор даного об’єкту;
- BindObject() – виділення та зв’язування індексів VBO та VAO;
- _renderObjects(Shader, PrimitiveType?) – відобразити об’єкт.

Допоміжний об’єкт TriangleCoords був розроблений для полегшеного створення екземпляру об’єкту Triangle. Діаграма об’єкту TriangleCoords зображено на (рисунок 4.11).

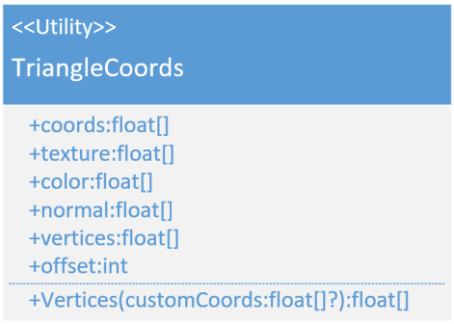

Рисунок 4.11 – Діаграма допоміжного об’єкта TriangleCoords

Цей об’єкт має наступні властивості та методи:

- coords – стандартне розміщення вершин в просторі локальної системи координат;
- texture – координати накладання текстури на елемент;
- color – колір об’єкт від його вершин;
- normal – вектори нормалей в точках вершин;
- vertices – масив групованих характеристик елементу;
- offset – зміщення в груповаому масиві між кінцем та початком характеристики одного типу;
- Vertices(float[]?):float[] – метод об’єднання характеристик групований масив.

Для відображення елементів на сцені, які мають форму куба, було розроблено об’єкт Cube, який є нащадком об'єкту Transformable. Діаграму об’єкту Cube зображено на (рисунок 4.12).

Рисунок 4.12 – Діаграма об’єкта Cube

Об’єкт Cube має наступні властивості та методи:

- vertices – характеристика позиції точки в її локальних координатах та її колір;
- VBO – індекс цього елемента в контексті інших об’єктів;
- VAO – індекс характеристики цього елемента для вертексного шейдеру;
- Cube(string[]?, Vector3, Vector3?):this – конструктор даного об’єкту;
- concatSet(float[]?, Vector3?) – метод об’єднання характеристик групований масив;
- BindObject() – виділення та зв’язування індексів VBO та VAO;
- _renderObjects(Shader, PrimitiveType?) – відобразити об’єкт.

Для спрощеної ініціалізації екземпляру об’єкта Cube був розроблений допоміжний об’єкт WallsCubic. У ньому реалізовані стандартні значення для генерації властивостей об’єкту Cube. Діаграма об’єкту WallsCubic зображена на (рисунок 4.13).

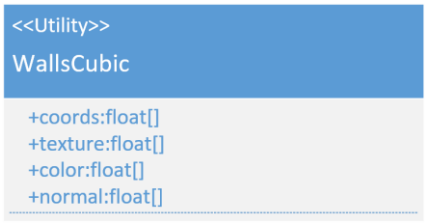

Рисунок 4.13 – Діаграма допоміжного об’єкта WallsCubic

Даний об'єкт має наступні властивості:

- coords – стандартне розміщення вершин в просторі локальної системи координат;
- texture – координати накладання текстури на елемент;
- color – колір об’єкт від його вершин;

- normal – вектори нормалей в точках вершин.

Для відображення напрямків Ox, Oy, Oz було розроблено об’єкт Axis, який є нащадком об’єкту Transformable та дає користувачу візуально зображення координатних осей по напрямкам, де їх значення додатні. Діаграму об’єкта Axis зображено на (рисунок 4.14).

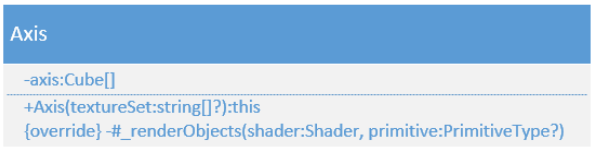

Рисунок 4.14 – Діаграма об’єкта Axis

Даний об’єкт має наступні властивості та методи:

- axis – об’єкти зображення яких знаходяться по координатним осям;
- Axis(string[]?):this - конструктор даного об’єкту;
- _renderObjects(Shader, PrimitiveType?) – відобразити об’єкт.

Для відображення об’єктів, що мають квадратну форму або форму, що відповідає двом прямокутним трикутникам, зі спільним ребром та рівними протилежними катетами й при цьому між площинами, що формують ці трикунтники існує деякий кут; було розроблено об’єкт Square, який є нащадком об’єкта Transformable. Діаграму об’єкта Square зображено на (рисунок 4.15).

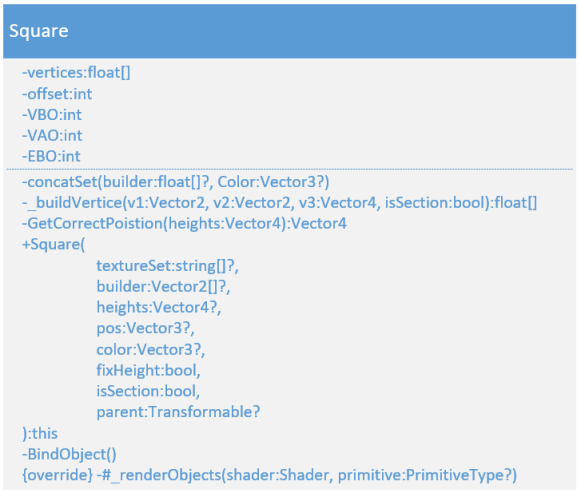

Рисунок 4.15 – Діаграма об’єкта Square

Об’єкт Square має наступні властивості та методи:

- vertices – характеристика позиції точки в її локальних координатах та її колір;
- offset – зміщення в групованому масиві між кінцем та початком характеристики одного типу;
- VBO – індекс цього елемента в контексті інших об’єктів;
- VAO – індекс характеристики цього елемента для вертексного шейдеру;
- EBO – індекси використання порядку вершин при побудові цього комплексного елементу для зображення;
- concatSet(float[]?, Vector3) – метод об’єднання характеристик групований масив;
- _buildVertice(Vector2, Vector2, Vector4, bool):float[] – побудова геометричної характеристики об’єкту в локальних координатах;
- GetCorrectPosition(Vector4):Vector4 – корегування висоти об’єкту відносно її мінімального для нього значення;
- Square(string[]?, Vector2[]?, Vector4?, Vector3?, Vector3?, bool, bool, Transformable?):this – конструктор даного об’єкту;

- BindObject() – виділення та зв’язування індексів VBO, VAO, EBO;
- _renderObjects(Shader, PrimitiveType?) – відобразити об’єкт.

Допоміжний об’єкт Tile спрощує ініціалізацію екземпляру об’єкту Square. Діаграм об’єкту Tile зображено на (рисунок 4.16).

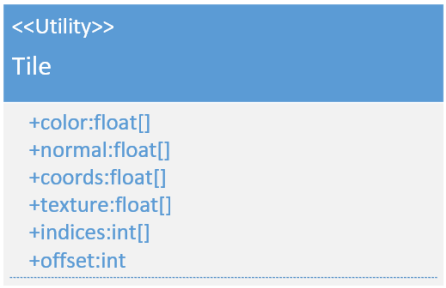

Рисунок 4.16 – Діаграма допоміжного об’єкта Tile

Даний об’єкт має наступні властивості:

- color – колір об’єкт від його вершин;
- normal – вектори нормалей в точках вершин;
- coords – стандартне розміщення вершин в просторі локальної системи координат;
- texture – координати накладання текстури на елемент;
- indices – масив порядку використання вершин при формуванні геометрії об'єкту;
- offset – зміщення в групованому масиві між кінцем та початком характеристики одного типу.

Для побудови комплексного сітчастого об’єкту на сцені було розроблено об’єкт ComplexPlaneTile, який є нащадком об’єкту Transformable. Діаграма об’єкту ComplexPlaneTile зображена на (рисунок 4.17).

Рисунок 4.17 – Діаграма об’єкта ComplexPlaneTile

Об’єкт ComplexPlaneTile має наступні властивості та методи:

- tiles – масив складових елементів сітчастого об’єкту;
- showGeometry – значення чи відображати сітчастий об’єкт;
- data – масив даних точок для об’єкту;
- Xmesh – масив значень по осі Ox;
- Ymesh – масив значень по осі Oy;
- DataStock – двовимірний масив незмінюваних даних;
- DataBuffer – двовимірний масив змінюваних даних;
- Range – масив в якому зберігаються діапазони індексів елементів сітчастого об’єкту;
- dots – масив точок, розташованих в точках вершин об’єктів;
- showDots – значення чи відображати масив точок;
- primitiveCurrent – індекс типу відображення сітчастого об’єкту;
- lineWidth – ширина лінії в режимі відображення лініями;

- stylesSwitcher – масив підтримуваних типів відображення сітчастого об’єкту;
- drawStyle – поточний тип відображення об’єкту;
- ComplexPlaneTile(Vector3[]?, float[]?, float[]?, float[][]?, string[]?, int[]?, float, Material?):this – конструктор даного об’єкту;
- LoadData(float[], float[], float[][]) – внесення/оновлення даних до об’єкту;
- Interp(float, float) – інтерполяція даних;
- MeshCompatibleRange() – встановлення границь відображення об’єкту для оптимізації зображення;
- Move(int, int, int, int, int, int) – переміщення границі відображення об’єкту;
- ResetMeshVisibility() – скинути налаштування відображення границі об’єкту;
- RebuildRelief() – побудова сітчастого об’єкту на основі поточних даних;
- SwitchDrawStyle() – змінити поточний режим відображення об’єкту;
- _calcHeightRange() – обчислити крайні екстремуми по осі Oz;
- Move(Vector3) – перемістити об’єкт у напрямку деякого вектора на величину цього вектора;
- _renderObjects(Shader, PrimitiveType?) – відобразити об’єкт.

При імпорті та експорті даних з та до системи об’єкту ComplexPlaneTile використовується об’єкт TileDataSet, який базується на поточних значеннях Xmesh, Ymesh, DataBuffer з об’єкту ComplexPlaneTile. Діаграма об’єкта TileDataSet зображена на (рисунок 4.18).

Рисунок 4.18 – Діаграма об’єкта TileDataSet

В об’єкті TileDataSet присутні наступні властивості та методи:

- X – масив значень абсцис точок;
- Y – масив значень ординат точок;
- Z – двовимірний масив значень висоти в позиціях точок;
- TileDataSet(float[], float[], float[]):this – конструктор даного об’єкту;
- _writeStream(JsonWriter, string, float[]) – серіалізація масиву;
- _writeStream(JsonWriter, string, float[][]) – серіалізація двовимірного масиву;
- WriteStreamPointsDataSet() – серіалізація як об’єкту PointsDataSet.

Для відображення масиву точок та їх подальшої тріангуляції або/та інтерполяції використовується об’єкт ComplexPlaneTriangular, який є нащадком об’єкта Transform. Діаграма об’єкта ComplexPlaneTriangular зображена на (рисунок 4.19).

Рисунок 4.19 – Діаграма об’єкта ComplexPlaneTriangular

Властивості та методи об’єкту ComplexPlaneTriangular описано нижче:

- connections – масив трикутників, що створюється після тріангуляції;
- dataTriangularedDataSet – дані тріангуляції в екземплярі об’єкта типу TriangularedDataSet;
- showGeometry – значення чи відображати тріангульований об’єкт;
- data – вхідний масив точок;
- dataSetAdjustable – список точок;
- dots – масив точок для візуального відображення їх на сцені;
- dotsize – розмір точки;
- dotColor – колір точки;
- dotColorHighlighted – колір точки, яку виділив користувач;
- showDots – значення чи відображати вхідні точки на сцені;
- interp – об’єкт інтерполянту;
- primitiveCurrent – індекс типу відображення тріангульованого об’єкту;

- lineWidth – ширина лінії в режимі відображення лініями;
- stylesSwitcher – масив підтримуваних типів відображення тріангульованого об’єкту;
- drawStyle – поточний тип відображення об’єкту;
- ComplexPlaneTriangular(Vector3[]?, float[]?, float[]?, float[]?, string[]?, bool):this – конструктор даного об’єкту;
- ComplexPlaneTriangular(TriangularedDataSet, string[]?):this – конструктор даного об’єкту;
- UpdateVertices() – оновити дані масиву точок на основі списку точок;
- _createPolyFromPoints() – створити об’єкт Polygon на основі масиву точок;
- generateDelaunayTriangulationFromData(bool, bool, bool):bool – виконати тріангуляцію на основі масиву точок;
- HighlightDot(int) – замінити колір обраної точки;
- _readConnectionsFromMesh(IMesh) – створити масив трикунтників на основі даних отриманих після тріангуляції;
- ConvertToTiledByInterpolation(int, bool):ComplexPlaneTile – інтерполювати точки та записати їх в новий об’єкт ComplexPlaneTile;
- SwitchDrawStyle() – змінити поточний режим відображення об’єкту;
- ExportPointDataSet():TriangularedDataSet? – експортувати дані до об’єкту TriangularedDataSet;
- ExportPoints():Vector3[]? – експортувати масив точок, якщо не тріангуляція не відбувалася;
- _renderObjects(Shader, PrimitiveType?) – відобразити об’єкт.

Для відображення та опрацювання перерізу сітчастого об’єкту ComplexPlaneTile був створений об’єкт Section, який реалізує інтерфейс IRenderable. Діаграму об’єкта Section зображено на (рисунок 4.20).

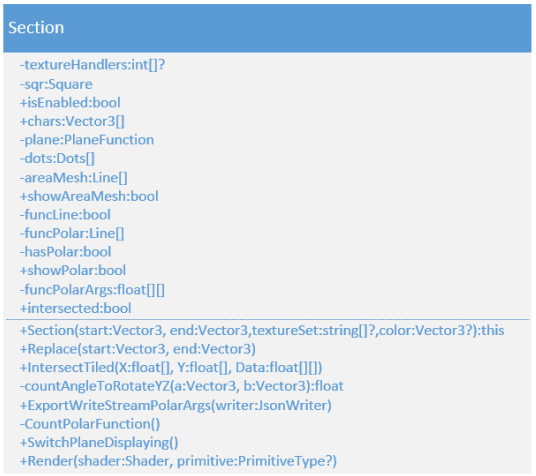

Рисунок 4.20 – Діаграма об’єкта Section

Об’єкт Section має наступні властивості та методи:

- textureHandlers – індикація текстур, що використовуються даним об’єктом;
- sqr – прямокутник границі площини перерізу;
- isEnabled – значення чи відображати площину перерізу;
- chars – характеристика геометрії границь перерізу;
- plane – об’єкт типу PlaneFunction, що характеризує рівняння площини;
- dots – масив точок перерізу;
- areaMesh – сітчастий об’єкт області перерізу;
- showAreaMesh – значення чи відображати область перерізу;
- funcLine – масив об’єкту Line, що відображає відрізки між точками перерізу;
- funcPolar – масив об’єкту Line, що відображає ламану лінію перерізу нахилену на площину Oyz;
- hasPolar – значення чи існують обчислені значення функції у полярній системі координат;
- showPolar – значення чи відображати геометричне пояснення перерерізу;

- funcPolarArgs – масив коефіцієнтів полярної функції в точках перерізу;
- intersected – значення чи відбулося обчислення перерізу перерізу;
- Section(Vector3, Vector3, string[]?, Vector3?):this – конструктор даного об’єкту;
- Replace(Vector3, Vector3) – побудувати переріз за іншими координатами;
- IntersectTiled(float[], float[], float[][]) – обчислити переріз за даними сітчастого об’єкта;
- countAngleToRotateYZ(Vector3, Vector3) – обчислити кут на який потрібно повернути ламану лінію перерізу до площини Oyz;
- ExportWriteStreamPolarArgs(JsonWriter) – експорт даних масиву коефіцієнтів полярної функції;
- CountPolarFunction() – обчислити коефіцієнти полярної функції;
- SwitchPlaneDisplaying() – змінити стан відображення площини перерізу;
- Render(Shader, PrimitiveType?) – підготувати та відобразити елементи об’єкту на сцені.

Додаткові допоміжні функції, що не відносяться до окремого об’єкту було згруповано в об’єкт Functions. Діаграма цього об’єкту зображена на (рисунок 4.21).

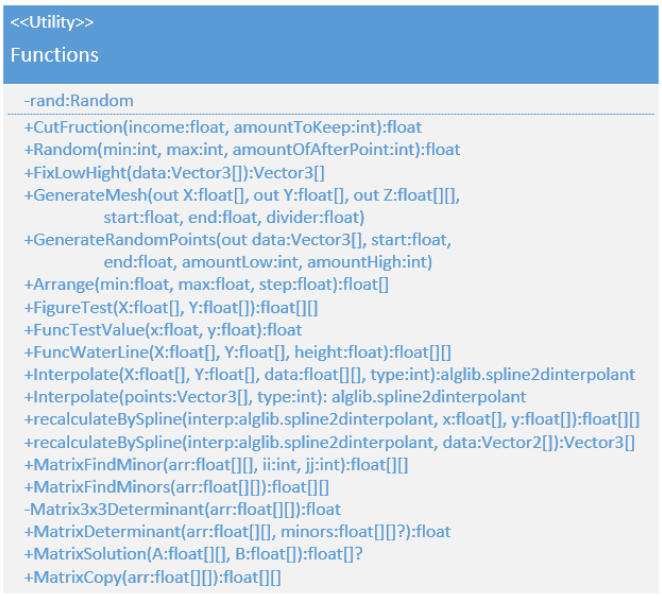

Рисунок 4.21 – Діаграма допоміжного об’єкта Functions

Дані про властивості та методи об’єкту Functions наведено в наступному переліку:

- rand – екземпляр об’єкту Random для генерації псевдовипадкових чисел;
- CutFunction(income:float, amountToKeep:int):float – позбутися деякої кількості знаків після коми для числа;
- Random(min:int, max:int, amountOfAfterPoint:int):float – створити значення числа в межах інтервалу на основі генерації псевдовипадкових чисел;
- FixLowHight(data:Vector3[]):Vector3[] – зменшити/збільшити значення висоти точок, щоб найменше було на площині Oxy;
- GenerateMesh(out float[], out float[], out float[][], float, float, float) – створити масиви для характеристики сітчастого об’єкта;
- GenerateRandomPoints(out Vector3[], float, float, int, int) – створити масив псевдовипадкових точок;
- Arrange(float, float, float):float[] – створити масив точок на відрізку з інтервалом;

- FigureTest(float[], float[]):float[][] – обчислити значення точок для тестової комплексної фігури;
- FuncTestValue(float, float):float – обчислити тестове значення функції в точці;
- FuncWaterLine(float[], float[], float):float[][] – обчислити значення висоти для екземпляру об’єкта ComplexPlaneTile;
- Interpolate(float[], float[], float[][], int): alglib.spline2dinterpolant – обчислити об’єкт інтерполянту на основі точок із сітчастим розміщенням;
- Interpolate(Vector3[], int): alglib.spline2dinterpolant – обчислити об’єкт інтерполянту на основі масиву точок;
- recalculateBySpline(alglib.spline2dinterpolant, float[], float[]):float[][] – порахувати значення в точках методом інтерполяції;
- recalculateBySpline(alglib.spline2dinterpolant, Vector2[]):Vector3[] – порахувати значення в точках методом інтерполяції;
- MatrixFindMinor(float[][], int, int):float[][] – обчислити мінор;
- MatrixFindMinors(float[][]):float[][] – обчислити матрицю мінорів;
- Matrix3x3Determinant(float[][]):float – обчислити детермінант матриці 3-го розряду;
- MatrixDeterminant(float[][], float[][]?):float – обчислити детермінант n-розрядної матриці;
- MatrixSolution(float[][], float[]):float[]? – розв’язок системи лінійно-алгебраїчних рівнянь методом Крамера;
- MatrixCopy(float[][]):float[][] – створення нової копії матриці.

Для обчислення методів функції прямої лінії було створено об’єкт LineFunction. Діаграма цього об’єкта зображена на (рисунок 4.22).

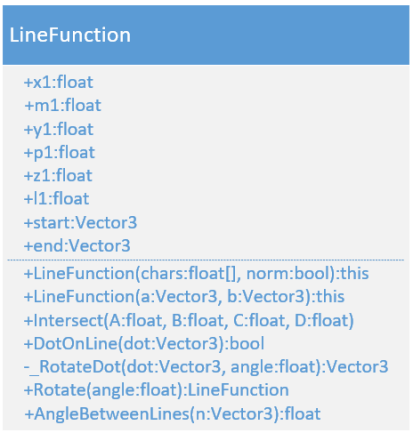

Рисунок 4.22 – Діаграма об’єкта LineFunction

Властивості та методи об’єкту LineFunction описано в наступному переліку:

- x1 – значення абсциси початку відрізку;
- m1 – значення абсциси напрямного вектору;
- y1 – значення ординати початку відрізку;
- p1 – значення ординати напрямного вектору;
- z1 – значення висоти розташування початку відрізку;
- l1 – значення направлення зміни висоти напрямного вектору;
- start – точка початку відрізку;
- end – точка кінця відрізку;
- LineFunction(float[], bool):this – конструктор об’єкта;
- LineFunction(Vector3, Vector3):this – конструктор об’єкта;
- Intersect(float, float, float, float) – метод обчислення точки перетину прямої та площини;
- DotOnLine(Vector3):bool – метод перевірки точки на її знаходження на прямій;
- RotateDot(Vector3, float):Vector3 – змінити кут вектору відносно осі на деяку величину;
- Rotate(float):LineFunction – обчислити зміну положення прямої після повороту відносно центру координатної осі на деякий кут;

- AngleBetweenLine(Vector3):float – обчислити кут між двома прямими.

Для полегшеного представлення та обчислення параметрів площини, що створює переріз, розроблено об’єкт PlaneFunction. Діаграма цього об’єкта зображена на (рисунок 4.23).

Рисунок 4.23 – Діаграма об’єкта PlaneFunction

Об’єкт PlaneFunction має наступні властивості та методи:

- A – параметр в загальному рівнянні площини;
- B – параметр в загальному рівнянні площини;
- C – параметр в загальному рівнянні площини;
- D – параметр в загальному рівнянні площини;
- PlaneFunction(Vector3, Vector3):this.

Для експорту та імпорту даних, в яких зберігаються дані про тріангулюваний об’єкт розроблено проміжний об’єкт перетворення TriangularedDataSet. Діаграму даного об’єкта зображено на (рисунок 4.24).

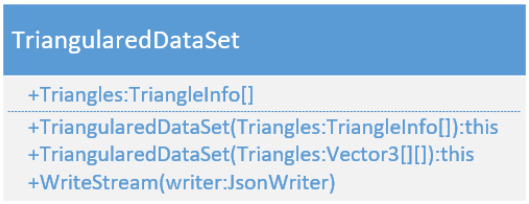

Рисунок 4.24 – Діаграма об’єкта TriangularedDataSet

Властивості та методи об’єкта представлено у наступному переліку:

- Triangles – масив об'єкту TriangleInfo в якого зберігається інформація про геометричне розташування трикутників;
- TriangularedDataSet(TriangleInfo[]):this – конструктор даного об’єкта;
- TriangularedDataSet(Vector3[][]):this – конструктор даного об’єкта;
- WriteStream(JsonWriter) – сериалізація даних об’єкту до накопичувача.

Для полегшеного збереження інформації про геометричне положення трикутника розроблено об’єкт TriangleInfo. Діаграма об’єкта зображена на (рисунок 4.25).

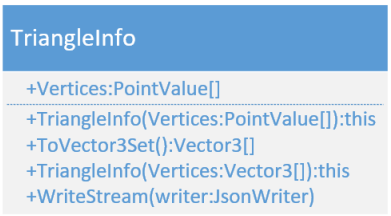

Рисунок 4.25 – Діаграма об’єкта TriangleInfo

Даний об’єкт має наступні властивості та методи:

- Vertices – характеристика вершин трикутника;
- TriangleInfo(PointValue[]):this – конструктор даного об’єкта;
- TriangleInfo(Vector3[]):this – конструктор даного об'єкта;
- WriteStream(JsonWriter) – сериалізація даних об’єкту до накопичувача.
- ToVector3Set():Vector3[] – отримати дані з даного об’єкта у вигляді масиву векторів.

Для полегшеного збереження інформації про координати точок був розроблений об’єкт PointValue. Діаграму даного об’єкта зображено на (рисунок 4.26).

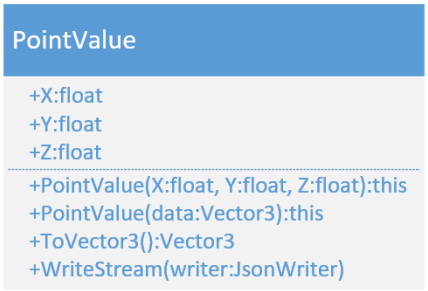

Рисунок 4.26 – Діаграма об’єкта PointValue

Даний об’єкт має наступні властивості та методи:

- X – значення абсциси точки;
- Y – значення ординати точки;
- Z – значення висоти точки;
- PointValue(float, float, float):this – конструктор даного об’єкта;
- PointValue(Vector3):this – конструктор даного об’єкта;
- ToVector3():Vector3 – отримати дані точки у вигляді вектора;
- WriteStream(JsonWriter) – сериалізація даних об’єкту до накопичувача.

Для збереження інформації у вигляді масиву даних про точки був розроблений об’єкт PointsDataSet. Діаграму цього об’єкту зображено на (рисунок 4.27).

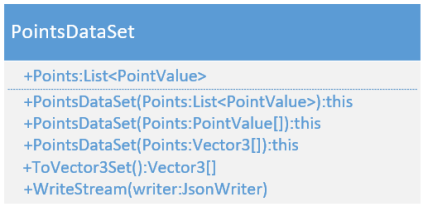

Рисунок 4.27 – Діаграма об’єкта PointsDataSet

Даний об’єкт має наступні властивості та методи:

- Points – список координат точок;
- PointsDataSet(List<PointValue>):this – конструктор даного об’єкта;

- PointsDataSet(PointValue[]):this – конструктор даного об’єкта;
- PointsDataSet(Vector3[]):this – конструктор даного об’єкта;
- ToVector3():Vector3 – отримати дані точки у вигляді вектора;
- WriteStream(JsonWriter) – серіалізація даних об’єкту до накопичувача.

Для ініціалізації та керування програмою шейдера був розроблений об’єкт Shader[1,5,6]. Діаграму даного об’єкта зображено на (рисунок 4.28).

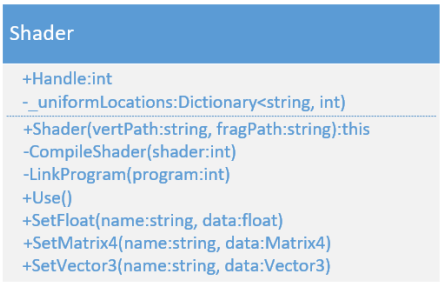

Рисунок 4.28 – Діаграма об’єкта Shader

Дані про властивості та методи даного об’єкта представлено в наступному списку:

- Handle – індекс програми шейдеру;
- _uniformLocations – перелік змінних типу uniform, які існують у скомпільованому шейдері;
- Shader(string, string):this – конструктор даного об’єкта;
- CompileShader(int) – скомпілювати програму шейдеру;
- LinkProgram(int) – прив’язати програму шейдеру для використання;
- Use() – використати поточний шейдер;
- SetFloat(string, float) – встановити значення змінної за ім’ям типу float;
- SetMatrix4(string, Matrix4) – встановити значення змінної за ім’ям типу Matrix4;

- SetVector3(string, Vector3) – встановити значення змінної за ім’ям типу Vector3;

Для полегшеного створення елементів інтерфейсу був розроблений допоміжний об'єкт Generate. Діаграма даного об’єкта зображена на (рисунок 4.29).

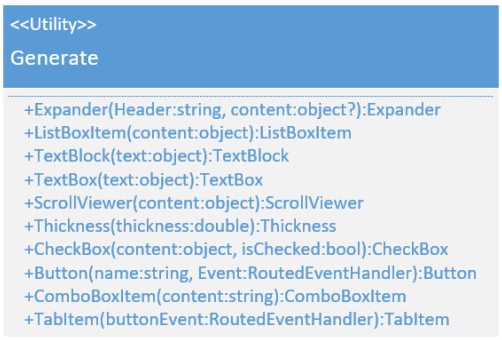

Рисунок 4.29 – Діаграма допоміжного об’єкта Generate

Даний об’єкт має наступні методи:

- Expander(string, object?):Expander – створити об’єкт типу Expander;
- ListBoxItem(object):ListBoxItem – створити об’єкт типу ListBoxItem;
- TextBlock(object):TextBlock – створити об’єкт типу TextBlock;
- TextBox(object):TextBox – створити об’єкт типу TextBox;
- ScrollViewer(object):ScrollViewer – створити об’єкт типу ScrollViewer;
- Thickness(double):Thickness – створити об’єкт типу Thickness;
- CheckBox(object, bool):CheckBox – створити об’єкт типу CheckBox;
- Button(string, RoutedEventHandler):Button – створити об’єкт типу Button;
- ComboBoxItem(string):ComboBoxItem – створити об’єкт типу ComboBoxItem;
- TabItem(RoutedEventHandler):TabItem – створити об’єкт типу TabItem.

Для збереження імен відображуваних імен об’єктів ComplexPlaneTile, Section та ComplexPlaneTriangular був розроблений допоміжний об’єкт ObjectNames. Діаграму даного об’єкта зображено на (рисунок 4.30).

Рисунок 4.30 – Діаграма допоміжного об’єкта ObjectNames

Даний об’єкт має властивість objectNames, в якому представлено стандартні імена для об’єктів, що можуть бути додані або виключені зі сцени.

Для керування камерою на сцені був розроблений об’єкт CameraControl. Діаграму об’єкта CameraControl зображено на (рисунок 4.31).

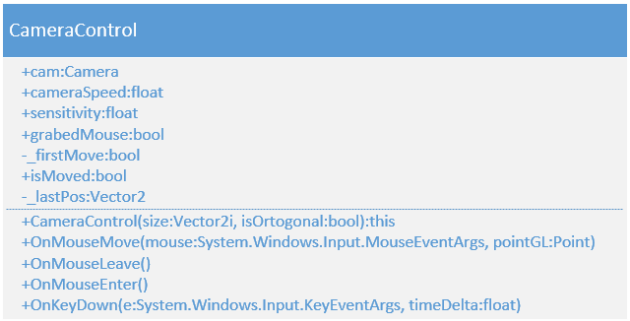

Рисунок 4.31 – Діаграма об’єкта CameraControl

За допомогою даного об’єкта є можливим отримання управлінням точки формування перспективи, керування, переміщення та зміна налаштувань об’єкту камери Camera. Даний об’єкт має наступні методи та властивості:

- cam – екземпляр об’єкта Camera;
- cameraSpeed – швидкість руху камери;
- sensitivity – чутливість реакції камери на рух курсора миші;
- grabedMouse – значення чи зчитувати рух курсора миші;

- _firstMove – значення чи є рух курсора миші у вікні першим після того як позиція миші опинилася в області існування графічного вікна;
- isMoved – значення чи здійснилося переміщення камери після натискання клавіші;
- _lastPos – остання позиція камери на сцені;
- CameraControl(Vector2i, bool):this – конструктор даного об’єкта;
- OnMouseMove(MouseEventArgs, Point) – подія руху курсора миші;
- OnMouseLeave() – подія курсор миші покидає вікно графічного вікна;
- OnMouseEnter() – подія курсор миші входить в границю існування графічного вікна;
- OnKeyDown(KeyEventArgs, float) – подія натискання клавіші користувачем.

Характеристика об’єкту камери описана в об’єкті Camera. В ньому зберігається інформація про позицію, направлення, кут нахилу та поле зору камери. Об’єкт Camera описаний у діаграмі на (рисунок 4.32).

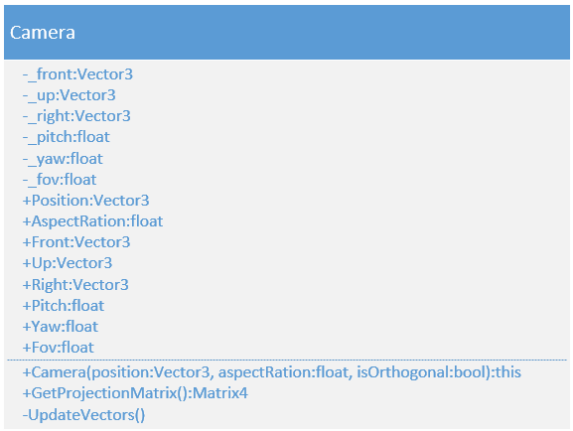

Рисунок 4.32 – Діаграма об’єкта Camera

Даний об’єкт має наступні властивості та методи:

- _front – вектор нахилу камери вперед;

- _up – вектор нахилу камери в сторону;
- _right – вектор нахилу камери набік;
- _pitch – нахил камери відносно вертикалі;
- _yaw – нахил камери відносно горизонталі;
- _fov – поле зору камери;
- Position – отримати, встановити позицію камери;
- AspectRation – співвідношення сторін графічного вікна;
- Front – отримати, встановити вектор нахилу камери вперед;
- Up – отримати, встановити вектор нахилу камери в сторону;
- Right – отримати, встановити вектор нахилу камери набік;
- Pitch – отримати, встановити нахил камери відносно вертикалі;
- Yaw – отримати, встановити нахил камери відносно горизонталі;
- Fov – отримати, встановити поле зору камери;
- Camera(Vector3, float, bool):this – конструктор даного об’єкта;
- GetProjectionMatrix():Matrix4 – отримати матрицю проекції камери;
- UpdateVectors() – оновити дані векторів характеристики камери.

Для відображення інтерфейсу користувачеві та контролю налаштувань основних елементів інтерфейсу був розроблений об'єкт MainWindow. Діаграму даного об’єкта зображено на (рисунок 4.33).

Рисунок 4.33 – Діаграма об’єкта MainWindow

Даний об’єкт є частковим. На діаграмі представлено розроблену частину. Інша частина автоматично генерується за допомогою фреймворку .NET 5.0 відповідно до розроблених елементів інтерфейсу на мові XAML. Об'єкт MainWindow має наступні реалізовані властивості та методи:

- editors – список існуючих редакторів с системі;
- currentEditor – поточний редактор;
- currentEditorNum – поточний індекс редактора;
- ticker – об’єкт накопичення часу;
- fps_max – максимальна кількість кадрів в секунду;
- fps_min – мінімальна кількість кадрів в секунду;
- fps_lim – поточний ліміт кадрів в секунду;
- MainWindow() – конструктор даного об’єкта;
- initItems() – ініціалізація даного об’єкта;
- OnPreviewMouseWheel(object, MouseWheelEventArgs) – подія прокрутки;

- OnMouseLeave(object, MouseEventArgs) – подія покидання курсора вікна;
- OnMouseEnter(object, MouseEventArgs) – подія входу курсора в границю вікна;
- OnKeyDown(object, KeyEventArgs) – подія натискання клавіші;
- OnMouseMove(object, MouseEventArgs) – подія переміщення курора;
- OnPreviewMouseKeyPressed(object, MouseButtonEventArgs) – подія натискання клавіші миші;
- switchVisibleGLStatus() – змініти статус графічного вікна;
- OnReady() – подія об’єкт графічного вікна завнтажений;
- EditorStructureEditorChanged(object, SelectionChangedEventArgs) – подія обраний об’єкт з дерева структура редактора;
- EditorSceneSettingsLoaded(object, RoutedEventArgs) – подія об’єкт структури готовий до роботи;
- OnRender(TimeSpan) – подія графічне зображення поточного кадру графічного об’єкта;
- structureOnReady(object, RoutedEventArgs) – подія об’єкт структури завантажений;
- structureUpdate() – оновити дані об’єкта структури;
- OnResize(object, SizeChangedEventArgs) – подія зміни розміру вікна;
- editorAddNew() – додати новий редактор;
- ButtonEventDeleteCurrentEditor(object, RoutedEventArgs) – видалити поточний редактор;
- editorChangeCurrent(int) – змінити поточний редактор.

Для відображення поточних об’єктів в редакторі розроблений об’єкт Editor. Його діаграма зображена на (рисунок 4.34).

Рисунок 4.34 – Діаграма об’єкта Editor

Опис даних про властивості та методи даного об’єкту представлено в наступному списку:

- stopwatch – об’єкт годиника для визначення часу, який проходить між етапами зображення одного кадра та початком побудови іншого;
- elapsedTime – значення часу, яке пройшло з моменту запуску редактора;
- timeDelta – значення часу між етапами побудови сусідніх кадрів;
- model – матриця моделі відображення;
- modelCramble – матриця корекції відображення світла для векторів нормалі об’єктів;
- textureHandlers – перелік індексів стандартних текстур, які використовуються для об’єкта без власних;
- isLoaded – значення стану чи завантажений редактор;
- isRendered – значення стану чи закінчився рендер поточного кадру;
- Editor(Size) – конструктор даного об’єкта;
- Initialize() – ініціалізація елементів редактора;
- _setupObjects() – ініціалізація основних об’єктів редактора;

- _setupTextures() – ініціалізація основних текстур редактора;
- _setupCam() – ініціалізація камери редактора;
- _setupShader() – ініціалізація програми шейдеру;
- OnResize(Size) – подія розмір вікна змінився;
- _applyTextureUnits() – застосувати текстури;
- _applyShaderSettings() – застосувати налаштуванн шейдеру;
- _applyRenderQueue() – сформувати зображення об’єктів;
- Render() – сформувати кадр;
- RenderObject(IRenderable) – сформувати зображення об’єкта;
- OnKeyDown(KeyEventArgs) – подія клавіша натиснута;
- OnMouseMove(MouseEventArgs, Point) – подія курсор змінив своє розташування.

Для керування налаштуваннями редактора та відображення даних його елементів в інтерфейсі, зокрема оброблювати дії користувача, був розроблений об’єкт EditorSettings. Діаграма даного об’єкта зображена на (рисунок 4.35 – 4.37).

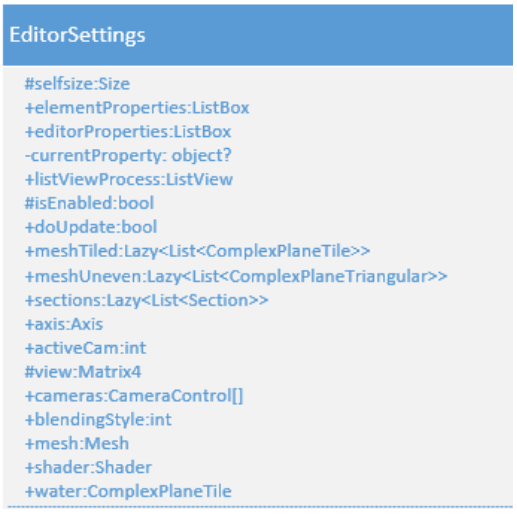

Рисунок 4.35 – Діаграма властивостей об’єкта EditorSettings

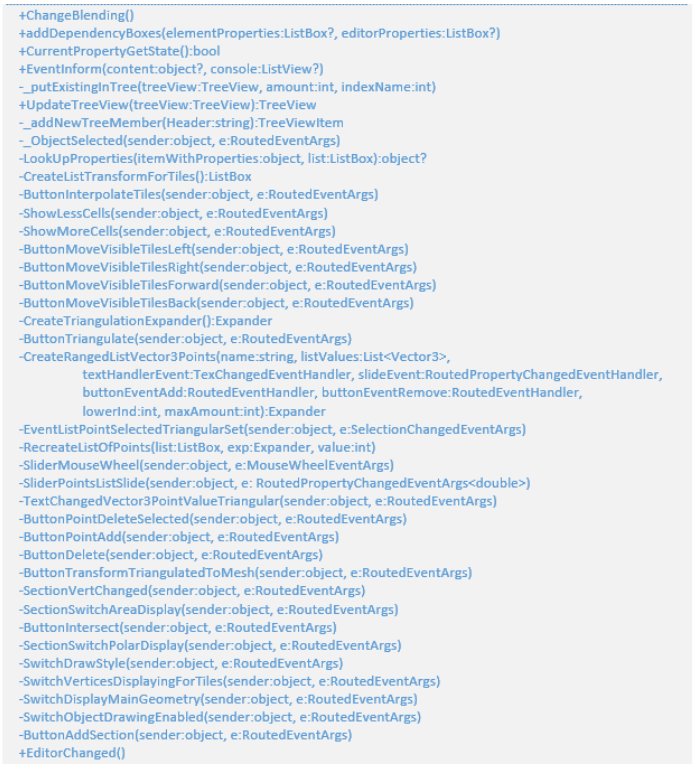

Рисунок 4.36 – Діаграма методів об’єкта EditorSettings

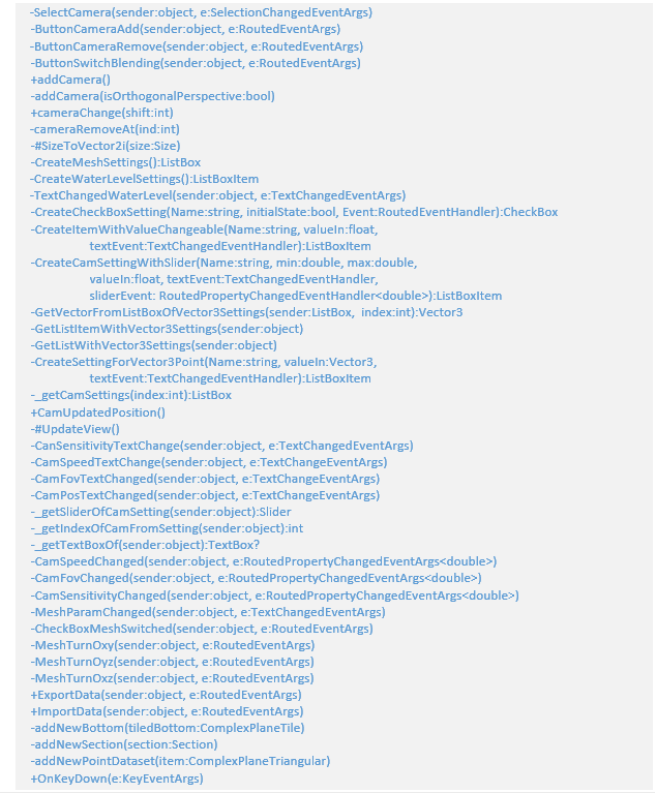

Рисунок 4.37 – Діаграма методів об’єкта EditorSettings(продовження)

Детальніше про властивості та методи даного об’єкта в наступному переліку:

- selfsize – розмір вікна редоктора;
- elementProperties – меню налаштувань поточного об’єкта редагування;
- editorProperties – меню налаштувань редактора;
- currentProperty – поточний обраний об’єкт;
- listViewProcess – список повідомлень;
- isEnabled – значення чи увікнений редактор;
- doUpdate – значення чи оновити структуру об’єктів;
- meshTiled – список об’єктів типу ComplexPlaneTile;
- meshUneven – список об’єктів типу ComplexPlaneTriangulared;
- sections – список об’єктів типу Sections;
- axis – екземпляр об’єкту Axis;
- activeCam – індекс поточної камери;
- view – матриця виду на зображення;
- cameras – список камер;
- blendingStyle:int – поточний режим відображення об’єктів
- mesh – сітка;
- shader – екземпляр об’єкта Shader для керування програмою шейдером;
- water – об’єкт для відображення рівня води;
- ChangeBlending() – метод зміни режиму відображення;
- addDependencyBoxes(ListBox?, ListBox?) – метод підключення меню налаштувань до поточного редактора
- CurrentPropertyGetState() – значення чи є обраний об’єкт зі структури;
- EventInform(object?, ListView?) – інформувати користувача за допомогою повідомлення;
- _putExistingInTree(TreeView, int, int) – додати існуючі об’єкти редактора до дерева структури;
- UpdateTreeView(tTreeView):TreeView – оновити дерево структури;

- _addNewTreeMember(string):TreeViewItem – додати об’єкт до дерева структури;
- _ObjectSelected(object, RoutedEventArgs) – подія об’єкт структури виділений;
- LookUpProperties(object, ListBox):object? – отримати налаштування поточного об’єкта;
- CreateListTransformForTiles():ListBox – створити список налаштувань для трансформації сітчастого об’єкта;
- ButtonInterpolateTiles(object, RoutedEventArgs) – подія розпочати інтерполяцію об’єкта;
- ShowLessCells(object, RoutedEventArgs) – подія відобразити менше елементів комплексного об’єкта;
- ShowMoreCells(object, RoutedEventArgs) – подія відобразити більше елементів комплексного об’єкта
- ButtonMoveVisibleTilesLeft(object, RoutedEventArgs) – подія змістити границю елементів комплексного об’єкта в бік зменшення значення ординати;
- ButtonMoveVisibleTilesRight(object, RoutedEventArgs) – подія змістити границю елементів комплексного об’єкта в бік збільшення значення ординати;
- ButtonMoveVisibleTilesForward(object, RoutedEventArgs) – подія змістити границю елементів комплексного об’єкта в бік збільшення значення абсциси;
- ButtonMoveVisibleTilesBack(object, RoutedEventArgs) – подія змістити границю елементів комплексного об’єкта в бік зменшення значення абсциси;
- CreateTriangulationExpander():Expander – створти налаштування тріангуляції;

- ButtonTriangulate(object, RoutedEventArgs) – подія виконати тріангуляцію об’єкта;
- CreateRangedListVector3Points(string, List<Vector3>, TexChangedEventHandler, RoutedPropertyChangedEventHandler, RoutedEventHandler, RoutedEventHandler, int, int):Expander – створити список точок об’єкта;
- EventListPointSelectedTriangularSet(sender:object, e:SelectionChangedEventArgs) – подія точка зі списку виділена;
- RecreateListOfPoints(ListBox, Expander, int) – оновити список точок;
- SliderMouseWheel(object, MouseWheelEventArgs) – подія прокручування колеса мищі, коли курсор знаходиться поверх слайдеру;
- SliderPointsListSlide(object, RoutedPropertyChangedEventArgs<double>) – подія значення слайдеру змінено;
- TextChangedVector3PointValueTriangular(object, RoutedEventArgs) – подія значення точки змімено;
- ButtonPointDeleteSelected(object, RoutedEventArgs) – подія видалити виділену точку;
- ButtonPointAdd(object, RoutedEventArgs) – подія додати точку до списку;
- ButtonDelete(object, RoutedEventArgs) – подія видалити об’єкт;
- ButtonTransformTriangulatedToMesh(object, RoutedEventArgs) – подія перетворити набір точок у об’єкт сітчастого виду за допомогою інтерполяції;
- SectionVertChanged(object, RoutedEventArgs) – подія змінено положення перерізу;
- SectionSwitchAreaDisplay(object, RoutedEventArgs) – подія змінений статус відображення області перерізу;
- ButtonIntersect(sobject, RoutedEventArgs) – подія обчислення значень перерізу;

- SectionSwitchPolarDisplay(object, RoutedEventArgs) – подія змінений статус відображення функції лінії перерізу на площині Oyz;
- SwitchDrawStyle(object, RoutedEventArgs) – подія змінений режим відображення;
- SwitchVerticesDisplayingForTiles(object, RoutedEventArgs) – подія змінений режим відображення точок вершин об’єкта;
- SwitchDisplayMainGeometry(object, RoutedEventArgs) – подія змінений статус відображення геометрії поточного об’єкта;
- SwitchObjectDrawingEnabled(object, RoutedEventArgs) – подія змінений статус відображення об’єкта;
- ButtonAddSection(object, RoutedEventArgs) – подія новий переріз було створено;
- EditorChanged() – оновити дані про налаштування редактора;
- SelectCamera(object, SelectionChangedEventArgs) – подія камера обрана;
- ButtonCameraAdd(object, RoutedEventArgs) – подія камера додана;
- ButtonCameraRemove(object, RoutedEventArgs) – подія камера видалена;
- ButtonSwitchBlending(object, RoutedEventArgs) – подія змінений режим відображення об’єктів;
- addCamera() – додання камери до списку камер;
- addCamera(bool) – додання камери до списки камер, обробка первинних даних камери;
- cameraChange(int) – змінити поточну камеру;
- cameraRemoveAt(int) – видалити камеру за індексом;
- SizeToVector2i(Size) – конвертувати розмір об’єкта в типі Size до Vector2i;

- CreateMeshSettings():ListBox – створити список налаштувань для об’єкта Mesh;
- CreateWaterLevelSettings():ListBoxItem – створити список налаштування для властивості water даного об’єкта;
- TextChangedWaterLevel(object, TextChangedEventArgs) – подія висота рівня води змінена;
- CreateCheckBoxSetting(string, bool, Event:RoutedEventHandler):CheckBox – створити налаштування в виді CheckBox-у;
- CreateItemWithValueChangeable(string, float, TextChangedEventHandler):ListBoxItem – створити налаштування зі змінюванним чисельним значенням
- CreateCamSettingWithSlider(string, double, double, float, TextChangedEventHandler, RoutedPropertyChangedEventHandler<double>):ListBoxItem – створити налаштування для камери у вигляді слайдеру;
- GetVectorFromListBoxOfVector3Settings(ListBox, int):Vector3 – отримати векторне значення зі списку значень;
- GetListItemWithVector3Settings(object) – отримати елемент списку із налаштуванням вектора;
- GetListWithVector3Settings(object) – отримати список з налаштуванням значень векторів;
- CreateSettingForVector3Point(string, Vector3, TextChangedEventHandler):ListBoxItem – створити налаштування для точки в просторі;
- _getCamSettings(int):ListBox – отримати список налаштувань камери за індексом;
- CamUpdatedPosition() – оновити позицію камери;

- UpdateView() – оновити матрицю виду;
- CanSensitivityTextChange(object, TextChangedEventArgs) – подія змінене налаштування чутливості камери;
- CamSpeedTextChange(object, TextChangeEventArgs) – подія змінене налаштування швидкості камери;
- CamFovTextChanged(object, TextChangeEventArgs) – подія змінене налаштування поля зору камери;
- CamPosTextChanged(object, TextChangeEventArgs) – подія змінене налаштування позиції камери;
- _getSliderOfCamSetting(object):Slider – отримати об’єкт слайдеру з налаштувань камери;
- _getIndexOfCamFromSetting(object):int – отримати індекс камери із налаштувань камери;
- _getTextBoxOf(object):TextBox? – отримати об’єкт із чисельним значенням налаштування;
- CamSpeedChanged(object, RoutedPropertyChangedEventArgs<double>) – подія змінена швидкість камери;
- CamFovChanged(object, RoutedPropertyChangedEventArgs<double>) – подія змінене поле зору камери;
- CamSensitivityChanged(object, RoutedPropertyChangedEventArgs<double>) – подія змінена чутливість камери;
- MeshParamChanged(object, TextChangedEventArgs) – подія змінене налаштування сітки;
- CheckBoxMeshSwitched(object, RoutedEventArgs) – подія увімкнуте або викмкнуте одне з налаштувань сітки;
- MeshTurnOxy(object, RoutedEventArgs) – відобразити або сховати сітку на площині Oxy;

- MeshTurnOyz(object, RoutedEventArgs) – відобразити або сховати сітку на площині Oyz;
- MeshTurnOxz(object, RoutedEventArgs) – відобразити або сховати сітку на площині Oxz;
- ExportData(object, RoutedEventArgs) – подія розпочато експорт даних поточного об’єкта;
- ImportData(object, RoutedEventArgs) – подія розпочато імпорт даних до редактора;
- addNewBottom(ComplexPlaneTile) – додати новий об’єкт ComplexPlaneTile;
- addNewSection(Section) – додати новий об’єкт перерізу;
- addNewPointDataset(ComplexPlaneTriangular) – додати новий об’єкт ComplexPlaneTriangular;
- OnKeyDown(e:KeyEventArgs) – подія клавіша натиснута.

4.2 Опис режимів візуалізації об’єктів

Для відображення накладання об’єктів а також візуалізації об’єктів із за допомогою різних функцій накладання та змішування їх зображення було використано наступну функцію GL.BlendFunc[7,17]. Ця вбудована функція специфікації OpenGL має два параметри: s – фактор: визначає як кольори та прозорість мають бути обчислені основуючись на вхідні дані об’єкта; та d – фактор: визначає як само будуть обчислюватись колір та прозорість об’єкта в результаті s – фактор обчислень. Кожен метод обчислення визначає спосіб використання чотирьох каналів кольору: червоний: R, зелений: G, синій: B, прозорість: A. Надалі вхідні дані по чотирьом каналам позначатимуться як (R_{s0}, G_{s0}, B_{s0}, A_{s0}), а кольори, що отримуються в результаті обчислень як (R_d, G_d, B_d, A_d). Вхідні значення також

конвертуються до діапазону [0,1] та позначаються відповідно як (k_R, k_G, k_B, k_A), де значення коефіцієнтів k_c, де C – колір відповідного каналу; розраховується за формулою:

$$k_c = 2^{m_c} - 1$$

Та (m_R, m_G, m_B, m_A) – значення відповідних за індексом кольорів. Вхідний та результуючий фактори зміни кольорів визначаються як (s_R, s_G, s_B, s_A) та (d_R, d_G, d_B, d_A) відповідно.

Було розроблено декілька режимів відображення, базуючись на поєднанні даних факторів. Було використано наступні факторні параметри (f_R, f_G, f_B, f_A), що зображені в (таблиця 4.2):

Таблиця 4.2 – Задіяні параметри графічних даних об’єкту

Параметр	(f_R, f_G, f_B, f_A)
BlendingFactor.One	$(1,1,1,1)$
BlendingFactor.SrcAlpha	$(\frac{A_{s0}}{k_A}, \frac{A_{s0}}{k_A}, \frac{A_{s0}}{k_A}, \frac{A_{s0}}{k_A})$
BlendingFactor.OneMinusSrcColor	$(1,1,1,1) - (\frac{R_{s0}}{k_R}, \frac{G_{s0}}{k_G}, \frac{B_{s0}}{k_B}, \frac{A_{s0}}{k_A})$
BlendingFactor.SrcColor	$(\frac{R_{s0}}{k_R}, \frac{G_{s0}}{k_G}, \frac{B_{s0}}{k_B}, \frac{A_{s0}}{k_A})$
BlendingFactor.OneMinusSrcAlpha	$(1,1,1,1) - (\frac{A_{s0}}{k_A}, \frac{A_{s0}}{k_A}, \frac{A_{s0}}{k_A}, \frac{A_{s0}}{k_A})$
BlendingFactor.OneMinusDstColor	$(1,1,1,1) - (\frac{R_d}{k_A}, \frac{G_d}{k_A}, \frac{B_d}{k_A}, \frac{A_d}{k_A})$

Щоб обчислити остаточний колір, функція змішування використовую наступні формули:

$$R_d = \min(k_R, R_s s_R + R_d d_{R)}$$

$$G_d = \min(k_G, G_s s_G + G_d d_{G)}$$

$$B_d = \min(k_B, B_s s_B + B_d d_{B)}$$

$$A_d = \min(k_A, R_s s_A + A_d d_{A)}$$

В роботі були використані наступні комбінацію параметрів для виокремлення режимів відображення об’єктів, де першим вказується s – фактор, другим – d – фактор відповідно:

1. BlendingFactor.One та BlendingFactor.OneMinusSrcAlpha.
2. BlendingFactor.OneMinusDstColor та BlendingFactor.SrcAlpha.
3. BlendingFactor.One та BlendingFactor.SrcColor.
4. BlendingFactor. SrcColor та BlendingFactor.OneMinusSrcAlpha.

5 РОБОТА КОРИСТУВАЧА З ПРОГРАМНОЮ СИСТЕМОЮ РЕДАКТОРА БАТИМЕТРІЇ

5.1 Робота користувача в інтерфейсі редактора

При відкритті програми користувач може бачити наступні об’єкти: головне графічне вікно сцени, де знаходяться сітка, напрямок координатних осей у бік додатних значень позначених різним кольором: червоним – Ox, зеленим – Oy, синім – Oz; та площина, що ілюструє рівень води; під графічним вікном знаходиться перелік повідомлень, які інформують користувача про певні виконані дії в системі та рекомендації; з правої сторони знаходяться два вікна: у верхньому зображено перелік існуючих редакторів та основні елементи, які на них розташовані; у нижньому знаходяться налаштування редактора та доданих користувачем об’єктів[2]. Демонстрацію стану програми після її запуску зображено на (рисунок 5.1).

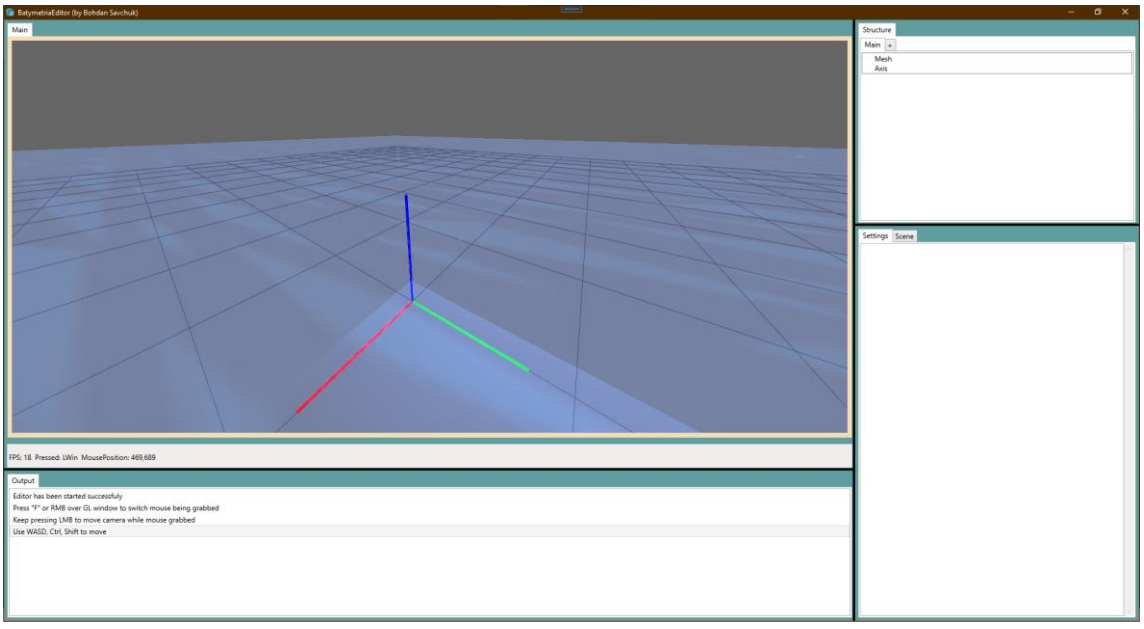

Рисунок 5.1 – Демонстрація програми після її відкриття

Якщо користувач бажає налаштувати редактор або імпортувати дані з файлу, він може натиснути на назву вікна налаштувань “Scene”. Демонстрація налаштувань редактора зображена на (рисунок 5.2).

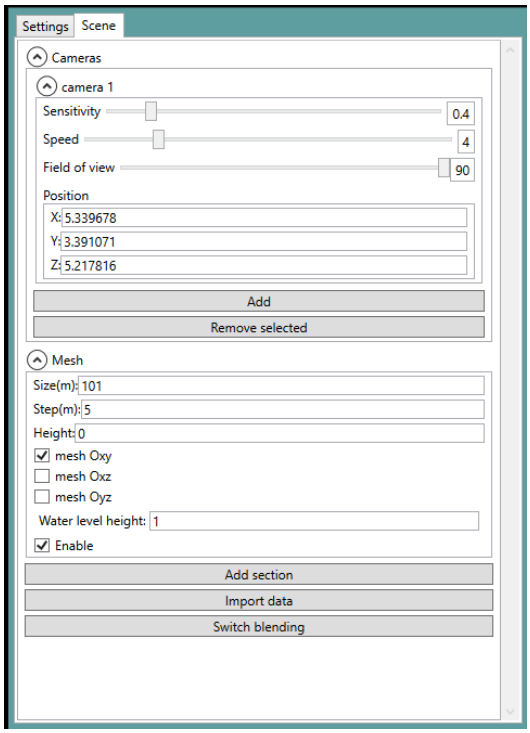

Рисунок 5.2 – Демонстрація налаштувань поточного редактора

В цьому меню налаштувань користувач може виконувати наступні дії: змінювати налаштування камери, додавати, видаляти: кнопки “Add” та “Remove selected” для кожної; та змінювати поточну камери; змінювати налаштування сітки та рівня води, вмикати та вимикати їх відображення, імпортувати дані до поточного редактора – кнопка “Import data”, створювати переріз – кнопка “Add section”, змінювати режим відображення об’єктів у редакторі сцени – кнопка “Switch blending”.

Об’єкти, що додаються на сцену до редактора внаслідок імпорту файлу, відображаються в переліку об’єктів в дереві об’єктів редактора. За бажанням користувача може бути створено декілька редакторів, натиснувши на назву вікна

“+”, які матимуть повний функціонал що й основний “Main”, а також редактори можуть бути видалені з системи за бажанням користувача, натиснувши кнопку “Delete this editor”. Демонстрація вікна редакторів зображена на (рисунок 5.3).

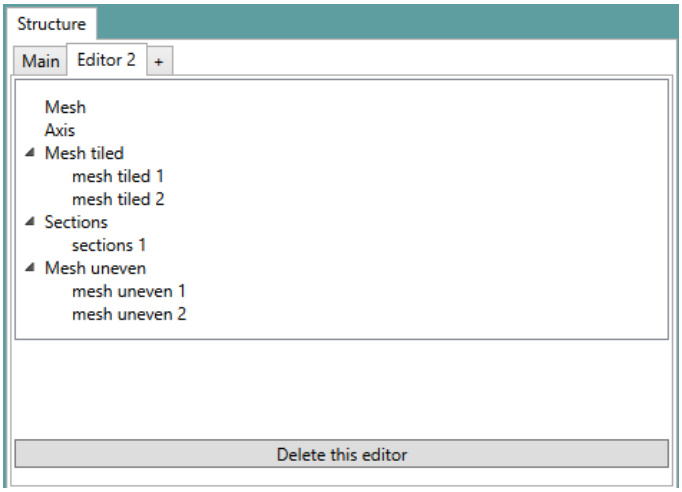

Рисунок 5.3 – Стандартний режим відображення об’єктів

В цьому меню користувач може бачити існуючі об’єкти в редакторі, а також перейти до редагування їх налаштувань в меню “Settings”, крім об’єктів “Mesh”, налаштування якого знаходяться в меню “Scene” та “Axis”.

Демонстрація роботи різних режимів відображення об’єктів представлена на (рисунок 5.4 – 5.7).

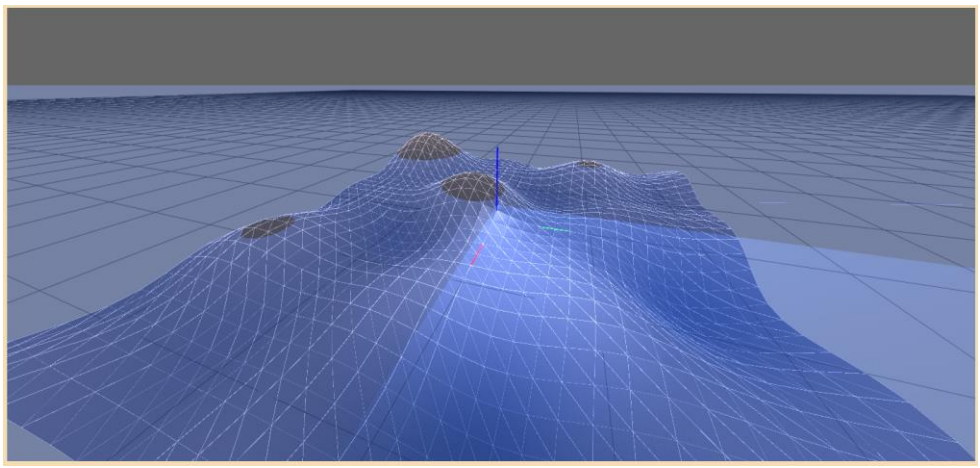

Рисунок 5.4 – Стандартний режим відображення об’єктів

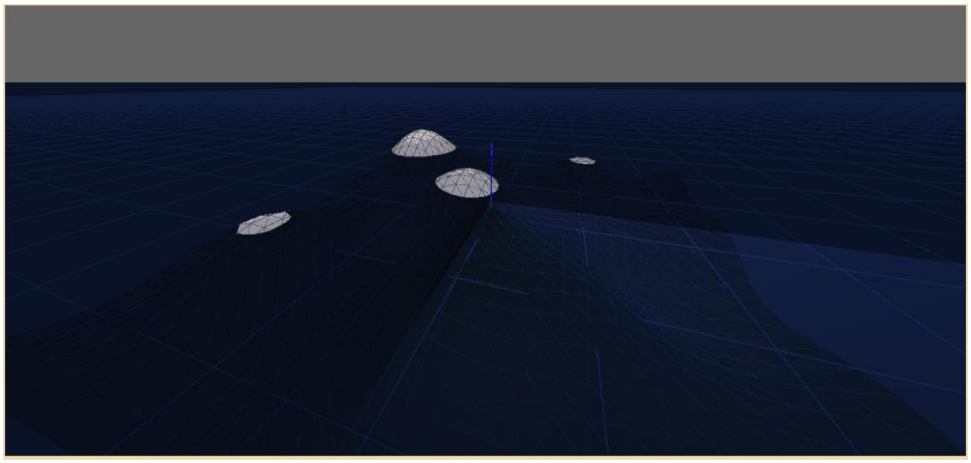

Рисунок 5.5 – Режим відображення об’єктів: підсвічення області над рівнем води

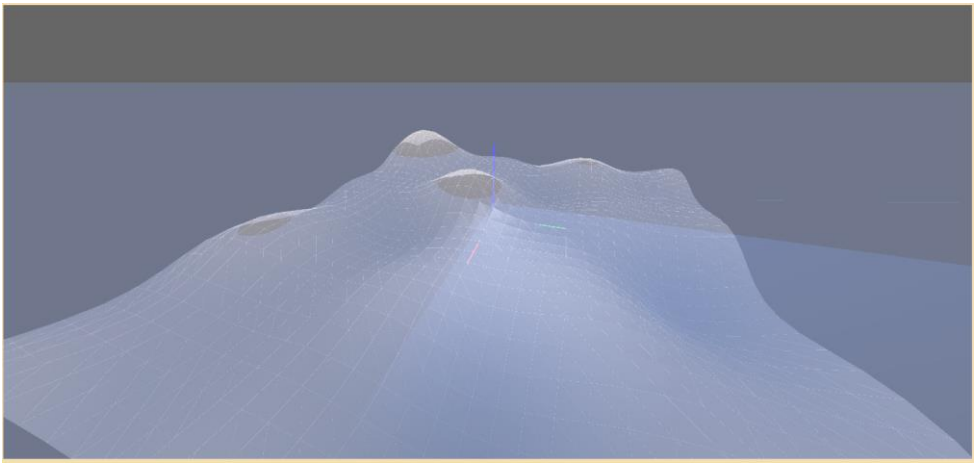

Рисунок 5.6 – Режим відображення об’єктів: підвищений рівень прозорості

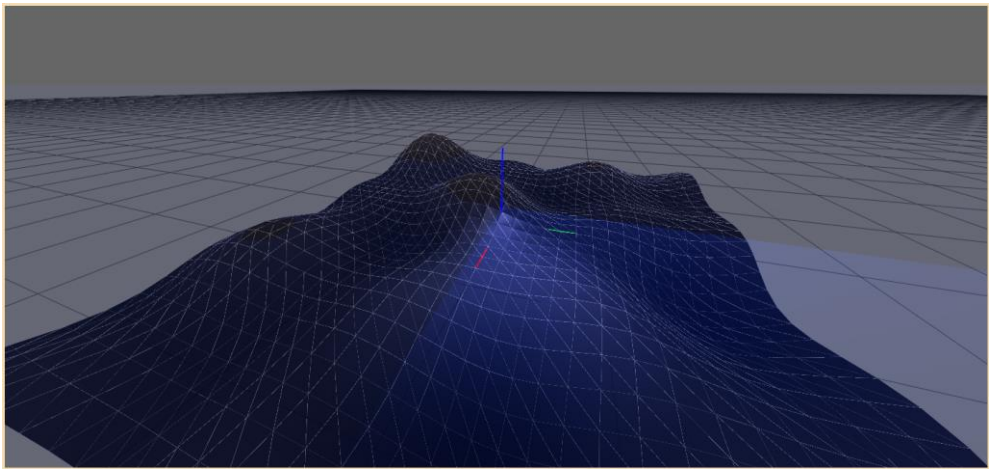

Рисунок 5.7 – Режим відображення об’єктів: акцент на геометрію

Обравши для редагування об’єкт із вітки “Mesh tiled” в меню налаштувань з’являються налаштування даного об’єкта. Серед них: вмикання та вимикання відображення вершин у вигляді точок – поле “Show vertices”, вмикання та вимикання відображення геометрії об’єкта – поле “Show geometry”, зміна типу його відображення – кнопка “Switch primitive type”, трансформація об’єкта – меню “Transform”: зміна границь відображення: кнопка “Expand” – розширити границю відображення, “Shift to the left” – змістити границю в бік зменшення значення ординати, “Shift to the right” – змістити границю в бік збільшення значення ординати, “Shift forward” – змістити границю в бік збільшення значення абсциси, “Shift back” – змістити границю в бік зменшення значення абсциси та “Shrink” – звузити границю; інтерполяція натиснувши кнопку “Interpolate” та застосування її результату до даних об’єкта із вказанням масштабування його геометрії в полі “Scale”; експортувати дані об’єкта до нового або існуючого файлу в форматі JSON, натиснувши кнопку “Export data”; увімкнути або вимкнути відображення об’єкта та його частин на в графічному редакторі – поле “Enable”; та видалення об’єкта з переліку поточного редактора за допомогою кнопки “Remove me”. Демонстрація вікна налаштувань для даного об’єкта зображена на (рисунок 5.8).

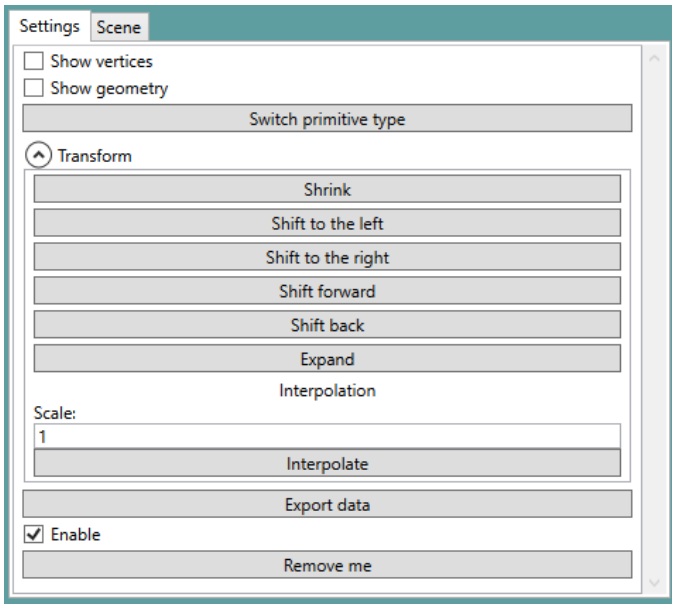

Рисунок 5.8 – Демонстрація переліку налаштувань для об'єктів із вітки “Mesh tiled”

Обравши для редагування об’єкт із вітки “Sections”, користувачу будуть доступні наступні налаштування в меню “Settings”: зміна точок, за якими будується площина перерізу перпендикулярна площині Oxy – меню “Vertices”; обчислення точок перерізу з об’єктом із вітки “Mesh tiled”, натиснувши кнопку “Intersect” та обравши об’єкт для перерізу із списку над цією кнопкою; увімкнути або вимкнути відображення об’єкта та його частин на в графічному редакторі – поле “Enable” та видалення об’єкта з переліку поточного редактора за допомогою кнопки “Remove me”. Якщо переріз був обчислений, то до меню налаштувань додаються наступні опції: увімкнення та вимкнення відображень для області перерізу та лінії перерізу по об’єкту – поле “Show intersected area”; увімкнення та вимкнення відображення лінії перерізу, яка нахилена на площину Oyz – поле “Show polar function”; та експорт даних коефіцієнтів для функцій точок у полярній системі координат – кнопка “Export data”. Демонстрація меню для об’єкта із вітки “Sections” зображена на (рисунок 5.9).

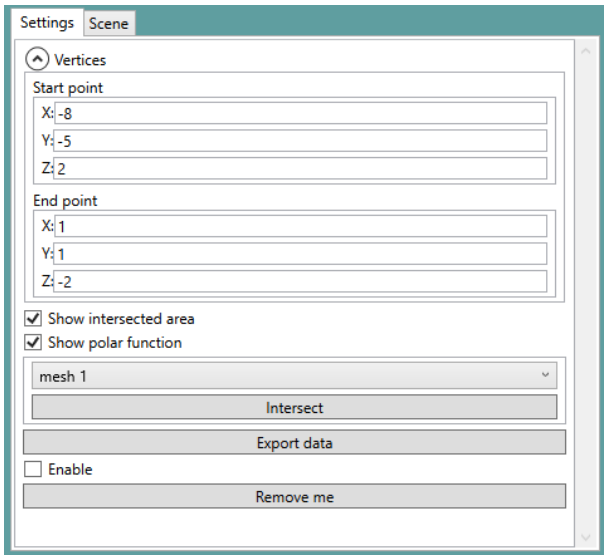

Рисунок 5.9 – Демонстрація переліку налаштувань для об'єктів із вітки “Sections”

Обравши об’єкт із вітки “Mesh uneven”, в переліку налаштувань об’єктів користувачу подаються наступний перелік налаштувань для обраного об’єкта: вмикання та вимикання відображення вершин у вигляді точок – поле “Show vertices”, вмикання та вимикання відображення геометрії об’єкта – поле “Show geometry”, зміна типу його відображення – кнопка “Switch primitive type”, створення нового об’єкта вітки “Mesh tiled” на основі результатів інтерполяції по точкам з даних поточного об’єкта – кнопка “Transform to mesh”; перелік точок даного об’єкта, кожна з яких доступна для редагування та підсвічується зеленим кольором при виділенні – список “Points”; застосування тріангуляції по до точок даного об’єкту із варіацією налаштувань – “Triangulation options”; експорт даних об’єкта до нового або існуючого файлу в форматі JSON, натиснувши кнопку “Export data”; увімкнення або вимкнення відображення об’єкта та його частин на в графічному редакторі – поле “Enable”; та видалення об’єкта з переліку поточного редактора за допомогою кнопки “Remove me”. Демонстрація меню для об’єкта із вітки “Mesh uneven” зображена на (рисунок 5.10).

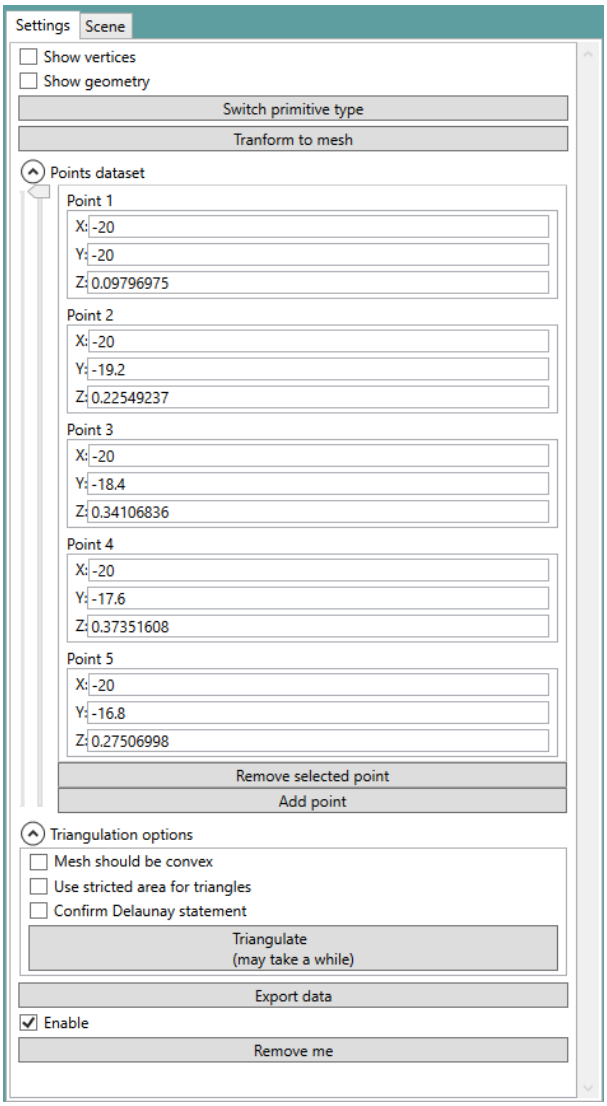

Рисунок 5.10 – Демонстрація переліку налаштувань для об'єктів із вітки “Mesh uneven”

При виконанні більшості дій в редакторі, користувачеві буде надходити повідомлення про успішне виконання дії або виявлену помилку. Повідомлення відображатимуться в списку повідомлень “Output”. Демонстрація списку повідомлень зображена на (рисунок 5.11).

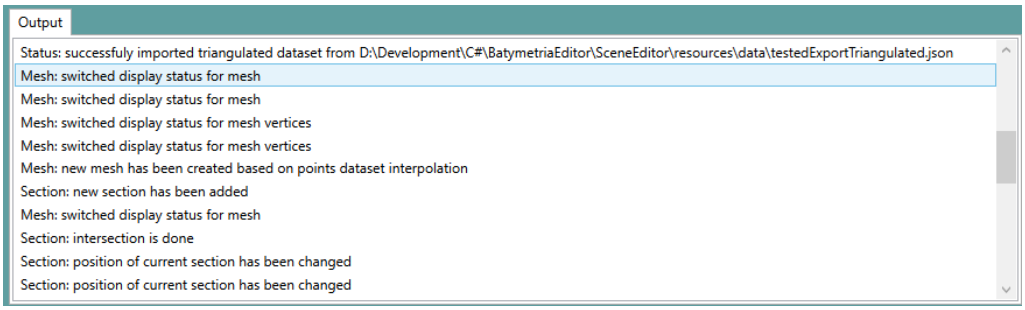

Рисунок 5.11 – Демонстрація списку повідомлень

Змінюючи тип відображення об’єктів користувач може порівнювати результати роботи редактора після виконаних користувачем дій. Демонстрація відображення декількох об’єктів водночас зі зміненим типом відображення зображена на (рисунок 5.12).

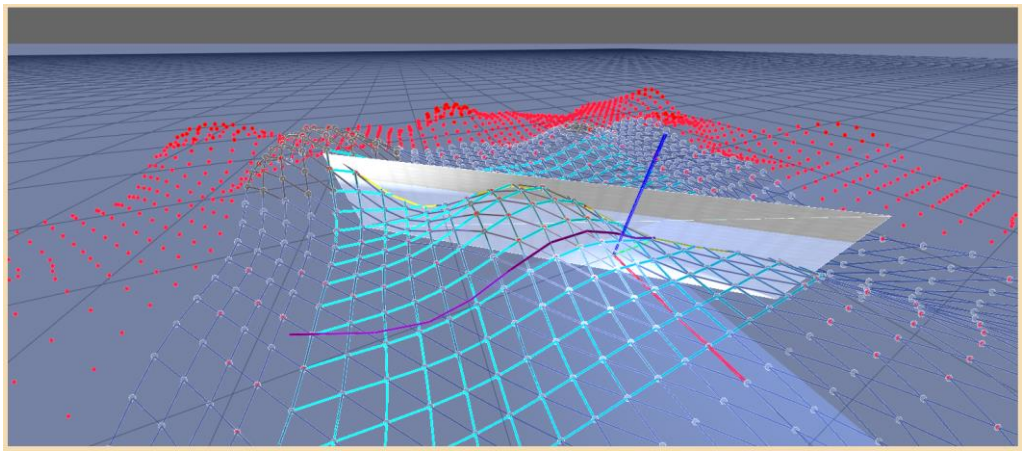

Рисунок 5.12 – Демонстрація відображання декількох об’єктів водночас із зміною режиму відображення для них

5.2 Системні вимоги для коректної роботи додатка

Для того, щоб користувач мав змогу користуватися розробленим програмним забезпеченням є за необхідним дотримання наступних системних вимог до програмного та апаратного забезпечення системи:

- Відеокарта із підтримкою версії OpenGL v4.5.0 або новіших;
- Встановлений пакет .NET Framework v4.7.2 або новіших;
- Встановлений пакет підтримки .NET v5.0 або новіших;
- Встановлена кількість оперативної пам’яті системи: 6 гігабайтів або більше;
- Вільного місця на носії: 20 мегабайт або більше;
- Операційна система Windows XP або новіша;
- Процесор потужністю від 1.2 Ггц на ядро та кількість ядер від 4 та більше для оптимальної роботи.

ВИСНОВКИ

У ході виконання роботи предметна область була проаналізована та поставлені задачі були виконані у вигляді розробленого програмного рішення.

Для ефективного проектування було проведено аналіз подібних програмних забезпечень у сфері моделювання та геології.

Результатом проведеної роботи є розробка програмного забезпечення, в якому втілено дружній та інтуїтивний для користувача інтерфейс. Також розроблена програма має тривимірний редактор сцени змодельованої батиметрії із можливістю створення декількох сцен водночас. Керування сценою відбувається за допомогою елементів клавіатури та миші. Відображення сцени існує за рахунок об’єкту камери, кількість яких та керування якими користувач може регулювати за власним бажанням.

Розроблене програмне забезпечення має більший функціонал для роботи з даними батиметрії в різному форматі вхідних даних та методами обробки ніж конкурентні програмні забезпечення.

Актуальність розробленого додатка обумовлена максимально ефективними рішеннями, які втілені в розробку його функціоналу в сфері обробки та збереження геометричних даних та моделюванні об’єктів.

Програмне забезпечення було протестоване на працездатність його функціоналу.

Даний додаток може мати наступні елементи в дорожній карті розробки:

- Додання нових алгоритмів обробки наборів точок та сітчастої поверхні;
- Покращена реалізація програми шейдеру;
- Підтримка інших форматів для вхідних та вихідних даних;
- Можливість зберігати виконані користувачем дії.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Вольф Д. Open GL 4. Язык шейдеров. Книга рецептов. ДМК Пресс, 2015. 368 с.
2. Хромых В. В., Хромых О. В. Цифровые модели рельефа : навч. посіб. Томск : ТМЛ-Пресс, 2007. 174 с.
3. Blender 3.1 reference manual. Blender 3.1 Reference Manual. URL: https://docs.blender.org/manual/en/latest (дата звернення: 23.05.2022).
4. C# docs - get started, tutorials, reference. Developer tools, technical documentation and coding examples | Microsoft Docs. URL: https://docs.microsoft.com/en-us/dotnet/csharp/ (дата звернення: 23.05.2022).
5. de Vries J. LearnOpenGL. LearnOpenGL. URL: http://learnopengl.com (дата звернення: 23.05.2022).
6. Gonzalez Vivo P., Lowe J. The book of shaders. The Book of Shaders. URL: https://thebookofshaders.com/ (дата звернення: 23.05.2022).
7. OpenGL® 4.5 reference pages. The Khronos Group Inc. URL: http://www.khronos.org/registry/OpenGL-Refpages/gl4/html/start.html (дата звернення: 23.05.2022).
8. Basics & Requirements - OpenTK. OpenTK. URL: https://opentk.net/faq.html (дата звернення: 23.05.2022).
9. Json.NET - newtonsoft. Json.NET - Newtonsoft. URL: https://www.newtonsoft.com/json/help/html/Introduction.htm (дата звернення: 23.05.2022).
10. ALGLIB - C++/C# numerical analysis library. ALGLIB - C++/C# numerical analysis library. URL: https://www.alglib.net/ (дата звернення: 23.05.2022).

11. Triangle.NET: C# / .NET version of jonathan shewchuk's triangle mesh generator. GitHub. URL: https://github.com/wo80/Triangle.NET (дата звернення: 23.05.2022).

12. Численное моделирование цунами и рельеф дна / Е. А. Куликов та ін. Журналы Физического факультета МГУ имени М. В. Ломоносова. URL: http://vmu.phys.msu.ru/html/2016/6/16-6-03/ (дата звернення: 23.05.2022).

13. Документация | documentation. ArcGis Desktop. URL: https://desktop.arcgis.com/ru/documentation/ (дата звернення: 23.05.2022).

14. Язык программирования C# и платформа .NET. METANIT.COM - Сайт о программировании. URL: https://metanit.com/sharp/ (дата звернення: 23.05.2022).

15. Welcome - The complete WPF tutorial. Welcome - The complete WPF tutorial. URL: https://wpf-tutorial.com/ (дата звернення: 23.05.2022).

16. How to serialize and deserialize (marshal and unmarshal) JSON in .NET. Developer tools, technical documentation and coding examples | Microsoft Docs. URL: https://docs.microsoft.com/en-us/dotnet/standard/serialization/system-text-json-how-to?pivots=dotnet-6-0 (дата звернення: 23.05.2022).

17. GLSL Programming/GLUT/Transparent Textures - Wikibooks, open books for an open world. Wikibooks. URL: https://en.wikibooks.org/wiki/GLSL_Programming/GLUT/Transparent_Textures (дата звернення: 23.05.2022).

18. C# FileStream, StreamWriter, StreamReader, TextWriter, TextReader Class. Software Testing Help. URL: https://www.softwaretestinghelp.com/c-sharp/csharp-filestream-streamwriter-streamreader/ (дата звернення: 23.05.2022).

19. ZerooBat. Качество и масштабирование цифрового изображения. Сравнение, мифы, способы и экономическая целесообразность. Хабр. URL: https://habr.com/ru/post/408915/ (дата звернення: 23.05.2022).

20. Stop slider raising events while sliding. MSDN. URL: https://social.msdn.microsoft.com/forums/vstudio/en-US/3f89658b-9651-4f64-93c7-e1a9bbf7a587/stop-slider-raising-events-while-sliding?forum=wpf (дата звернення: 23.05.2022).

ДОДАТОК А

Редактор батиметрії для сцени гідроакустичного експерименту

Лістинг програми

УКР.НТУУ”КПІ імені Ігоря Сікорського”_ТЕФ_АПЕПС_ТМ-81269_22Б 12-1

Аркушів 38

Київ 2022

ComplexPlane.cs:

```
using LearnOpenTK.Common;
using OpenTK.Graphics.OpenGL4;
using OpenTK.Mathematics;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace SceneEditor.editor
{
   public class ComplexPlaneTile : Transformable
   {
      public Square[]? tiles;
      public bool showGeometry = false;

      public Vector3[] data = new Vector3[0];

      public float[] Xmesh;
      public float[] Ymesh;

      public float[][] DataStock;

      public float[][] DataBuffer;

      public int[] Range = new int[4];

      private Dot[] dots;
      public bool showDots = false;

      private int primitiveCurrent = 0;
      private float lineWidth;
      private PrimitiveType[] stylesSwitcher = new PrimitiveType[]
         {
            PrimitiveType.Triangles,
            PrimitiveType.Lines,
            PrimitiveType.LineStrip,
            PrimitiveType.Points
         };
      public PrimitiveType drawStyle;
```

```
    public ComplexPlaneTile(Vector3[]? inputData = null,
                            float[]? X = null,
                            float[]? Y = null,
                            float[][]? Z = null,
                            string[]? textureSet = null,
                            int[]? textureHandlersCopy = null,
                            float lineWidth = 2f,
                            Material? material = null
                            )
    {
        if (inputData == null)
        {
            if (X == null || Y == null || Z == null)
            {
                Functions.GenerateMesh(out X, out Y, out Z, start: -20, end: 20, divider:
50);
            }
            LoadData(X, Y, Z);
            this.lineWidth = lineWidth;
        }
        if (textureSet != null)
        {
            textureHandlers = new int[textureSet.Length];
            for (int i = 0; i < textureSet.Length; i++)
            {
                textureHandlers[i] = TextureLoader.LoadFromFile(textureSet[i]);
            }
        }
        else
        {
            if (textureHandlersCopy != null)
            {
                textureHandlers = textureHandlersCopy;
            }
        }
        if(material != null)
        {
            this.material = material;
        }
```

```
            isEnabled = true;
            showGeometry = true;
            drawStyle = stylesSwitcher[primitiveCurrent];
        }

        public void LoadData(float[] X, float[] Y, float[][] Data)
        {
            Xmesh = X;
            Ymesh = Y;
            DataStock = Data;
            DataBuffer = Functions.MatrixCopy(DataStock);

            RebuildRelief();
        }

        public void Interp(float scale = 1, float shift = 0)
        {
            alglib.spline2dinterpolant interp = Functions.Interpolate(Xmesh, Ymesh,
DataBuffer);


            float dx = (Xmesh[1] - Xmesh[0]) * scale;
            float dy = (Ymesh[1] - Ymesh[0]) * scale;

            float[] X = Functions.Arrange(Xmesh[0] - shift, Xmesh[Xmesh.Length - 1] +
shift, dx);
            float[] Y = Functions.Arrange(Ymesh[0] - shift, Ymesh[Ymesh.Length - 1] +
shift, dy);

            Xmesh = X;
            Ymesh = Y;

            DataBuffer = Functions.recalculateBySpline(interp, X, Y);

            RebuildRelief();
        }

        public void MeshCompatibleRange()
        {
            int limit = 30;
            int xfactor = 0;
            int yfactor = 0;
            if (Xmesh.Length > limit)
```

```
        {
            xfactor = Xmesh.Length - limit;
        }
        if (Ymesh.Length > limit)
        {
            yfactor = Ymesh.Length - limit;
        }
        MoveVisibleMesh(0, 0, 0, 0, xfactor / 2, yfactor / 2);
    }

    public void MoveVisibleMesh(int x = 0, int y = 0, int scalex = 1, int scaley = 1, int
shrinkx = 0, int shrinky = 0)
    {
        int rows = Xmesh.Length - 1;
        int cols = Ymesh.Length - 1;

        if (shrinkx != 0 && Range[1] + shrinkx * 2 < Range[3])
        {
            if (Range[1] + shrinkx >= 0)
            {
                Range[1] += shrinkx;
            }
            if (Range[3] - shrinkx < rows)
            {
                Range[3] -= shrinkx;
            }
        }
        if (shrinky != 0 && Range[0] + shrinky * 2 < Range[2])
        {
            if (Range[0] + shrinky >= 0)
            {
                Range[0] += shrinky;
            }
            if (Range[2] - shrinky < cols)
            {
                Range[2] -= shrinky;
            }
        }

        int temp = Range[1] + x;
        Range[1] = temp >= 0 && temp < Range[3] ? temp : Range[1];

        temp = Range[3] + x;
```

```
    Range[3] = temp > Range[1] && temp < rows ? temp : Range[3];

    temp = Range[0] + y;
    Range[0] = temp >= 0 && temp < Range[2] ? temp : Range[0];

    temp = Range[2] + y;
    Range[2] = temp > Range[0] && temp < cols ? temp : Range[2];

    int dx = Range[3] - Range[1];
    int dy = Range[2] - Range[0];

    int cx = Range[0] + dx / 2;
    int cy = Range[1] + dy / 2;

    Range[0] = Range[0] < 0 ? 0 : Range[0];
    Range[1] = Range[1] < 0 ? 0 : Range[1];
    Range[2] = Range[2] >= cols ? cols - 1 : Range[2];
    Range[3] = Range[3] >= rows ? rows - 1 : Range[3];
}

public void ResetMeshVisibility()
{
    var rows = Xmesh.Length - 1;
    var cols = Ymesh.Length - 1;
    Range[0] = Range[1] = 0;
    Range[2] = rows - 1;
    Range[3] = cols - 1;
}

public void RebuildRelief()
{
    var rows = Xmesh.Length - 1;
    var cols = Ymesh.Length - 1;
    tiles = new Square[rows * cols];

    for (int i = 0; i < rows; i++)
    {
        for (int j = 0; j < cols; j++)
        {
            Vector2 top_left = new Vector2(Xmesh[i], Ymesh[j]);
            Vector2 bottom_right = new Vector2(Xmesh[i + 1], Ymesh[j + 1]);
            Vector4 heights = new Vector4(
                DataBuffer[i + 1][j + 1],
```

```
                DataBuffer[i][j + 1],
                DataBuffer[i][j],
                DataBuffer[i + 1][j]);
            tiles[i * cols + j] = new Square(
                builder: new Vector2[] { top_left, bottom_right },
                heights: heights,
                fixHeight: false,
                parent: this
                );
        }
    }

    dots = new Dot[Xmesh.Length * Ymesh.Length];

    for (int i = 0; i <= rows; i++)
    {
        for(int j = 0; j <= cols; j++)
        {
            dots[i * (cols + 1) + j] = new Dot(pos: new Vector3(Xmesh[i], Ymesh[j],
DataBuffer[i][j]), size: 3);
        }
    }

    _calcHeightRange();
    ResetMeshVisibility();
    MeshCompatibleRange();
}

public void SwitchDrawStyle()
{
    primitiveCurrent++;
    if(stylesSwitcher.Length == primitiveCurrent)
    {
        primitiveCurrent = 0;
    }
    drawStyle = stylesSwitcher[primitiveCurrent];
}

private void _calcHeightRange()
{
    float min = DataBuffer[0][0];
    float max = DataBuffer[0][0];
    for (int i = 0; i < DataBuffer.Length; i++)
```

```
    {
        for(int j = 0; j < DataBuffer[i].Length; j++)
        {
            float v = DataBuffer[i][j];
            if (v < min)
            {
                min = v;
            }
            else
            {
                if(v > max)
                {
                    max = v;
                }
            }
        }
    }

    position = new Vector3(0, 0, min);
    _range = new Vector2(min, max);
}

public override void Move(Vector3 shifts)
{
    position += shifts;
    for(int i = 0; i < Xmesh.Length; i++)
    {
        Xmesh[i] += shifts.X;
        for(int j = 0; j < Ymesh.Length; j++)
        {
            DataBuffer[i][j] += shifts.Z;
        }
    }

    for (int i = 0; i < Ymesh.Length; i++)
    {
        Ymesh[i] += shifts.Y;
    }

    RebuildRelief();
}
```

```
        private protected override void _renderObjects(Shader shader, PrimitiveType?
primitive)
        {
            if (showDots)
            {
                for (int i = 0; i < dots.Length; i++)
                {
                    dots[i].Render(shader);
                }
            }

            if (tiles != null && showGeometry)
            {
                var rows = Xmesh.Length - 1;
                var cols = Ymesh.Length - 1;
                GL.LineWidth(lineWidth);
                GL.PointSize(lineWidth * 5);

                for (int i = Range[0]; i <= Range[2]; i++)
                {
                    int ii = i * rows;
                    for (int j = Range[1]; j <= Range[3]; j++)
                    {
                        tiles[ii + j].Render(shader, drawStyle);
                    }
                }
            }
        }
    }
}
```

Section.cs:

```
using LearnOpenTK.Common;
using Newtonsoft.Json;
using OpenTK.Graphics.OpenGL4;
using OpenTK.Mathematics;
using System;
using System.Collections.Generic;
```

```
namespace SceneEditor.editor
{

    public class LineFunction
    {
        // x - x1   y - y1   z - z1
        // ------ = ------ = ------
        //   m1       p1       l1
        public float x1, m1, y1, p1, z1, l1;

        public Vector3 start;
        public Vector3 end;

        public LineFunction(float[] chars, bool norm = false)
        {
            this.x1 = chars[0];
            this.y1 = chars[2];
            this.z1 = chars[4];

            this.m1 = chars[1];
            this.p1 = chars[3];
            this.l1 = chars[5];

            if (!norm)
            {
                m1 -= x1;
                p1 -= y1;
                l1 -= z1;
            }

            start = new Vector3(x1, y1, z1);
            end = new Vector3(m1 + x1, p1 + y1, l1 + z1);
        }

        public LineFunction(Vector3 a, Vector3 b)
        {
            start = a;
            end = b;

            x1 = a.X;
            y1 = a.Y;
            z1 = a.Z;
```

```
    m1 = b.X - x1;
    p1 = b.Y - y1;
    l1 = b.Z - z1;
}

public Vector3? Intersect(float A, float B, float C, float D)
{
    Vector3 n1 = new Vector3(A, B, C);
    Vector3 n2 = new Vector3(m1, p1, l1);

    if((A*m1 + B*p1 + C*l1) / (n1.Length * n2.Length) == 0)
    {
        return new Vector3(x1, y1, z1);
    }

    float[][] arr = new float[3][];
    float[] right = new float[3];
    for (int i = 0; i < 3; i++)
    {
        arr[i] = new float[3];
    }
    arr[0][0] = p1;
    arr[0][1] = -m1;
    arr[0][2] = 0;

    arr[1][0] = 0;
    arr[1][1] = l1;
    arr[1][2] = -p1;

    arr[2][0] = A;
    arr[2][1] = B;
    arr[2][2] = C;

    right[0] = p1 * x1 - m1 * y1;
    right[1] = l1 * y1 - p1 * z1;
    right[2] = -D;

    float[]? X = Functions.MatrixSolution(arr, right);
    if (X != null)
    {
        return new Vector3(X[0], X[1], X[2]);
    }
    return null;
```

```
    }

    public bool DotOnLine(Vector3 dot)
    {
        float eps = MathF.Pow(10, -4);
        float x = (dot.X - x1);
        float y = (dot.Y - y1);
        float z = (dot.Z - z1);

        float dz = MathF.Abs(start.Z - end.Z);
        float dz1 = MathF.Abs(dot.Z - MathF.Min(start.Z, end.Z));
        if(dz1 > dz || dot.Z < MathF.Min(start.Z, end.Z))
        {
            return false;
        }

        List<float> c = new List<float>();
        if(m1 != 0)
        {
            c.Add(x/m1);
        }
        if(p1 != 0)
        {
            c.Add(y / p1);
        }
        if(l1 != 0)
        {
            c.Add(z / l1);
        }
        float[] arr = c.ToArray();
        switch (arr.Length)
        {
            case 1: return true;
            case 2: {
                    return Math.Abs(arr[0] - arr[1]) < eps;
                };
            default: return Math.Abs(arr[0] - arr[1]) < eps && Math.Abs(arr[1] - arr[2]) <
eps;
        }
    }

    private Vector3 _RotateDot(Vector3 dot, float angle)
    {
```

```
        float x = dot.X;
        float y = dot.Y;
        dot.X = -MathF.Sin(angle) * y + MathF.Cos(angle) * x;
        dot.Y = MathF.Cos(angle) * y + MathF.Sin(angle) * x;
        return dot;
    }

    public LineFunction Rotate(float angle)
    {
        return new LineFunction(_RotateDot(start, angle), _RotateDot(end, angle));
    }

    public float AngleBetweenLines(Vector3 n)
    {
        Vector3 v = new Vector3(m1, p1, l1);

        float cos = Vector3.Dot(v, n) / v.Length / n.Length;
        return MathF.Acos(cos);
    }
}

public class PlaneFunction
{
    public float A;
    public float B;
    public float C;
    public float D;


    public PlaneFunction(Vector3 a, Vector3 b)
    {
        Vector3 c = new Vector3(b.X, b.Y, a.Z);

        float[] arr = new float[6];
        arr[0] = b.X - a.X; arr[1] = b.Y - a.Y; arr[2] = b.Z - a.Z;

        arr[3] = c.X - a.X; arr[4] = c.Y - a.Y; arr[5] = c.Z - a.Z;

        A = arr[1] * arr[5] - arr[2] * arr[4];
        B = -(arr[0] * arr[5] - arr[2] * arr[3]);
        C = arr[0] * arr[4] - arr[1] * arr[3];
        D = A * (-1f) * a.X + B * (-1f) * a.Y + C * (-1f) * a.Z;
    }
```

```
    public PlaneFunction(float A, float B, float C, float D)
    {
        this.A = A;
        this.B = B;
        this.C = C;
        this.D = D;
    }
}

public class Section : IRenderable
{
    private int[]? textureHandlers = null;

    private Square sqr;
    public bool isEnabled = false;


    public Vector3[] chars = new Vector3[2];

    private PlaneFunction plane;

    private Dot[] dots;

    private Line[] areaMesh;
    public bool showAreaMesh = false;

    private Line[] funcLine;

    private Line[] funcPolar;
    private bool hasPolar = false;
    public bool showPolar = false;

    private float[][] funcPolarArgs;

    public bool intersected = false;

    public Section(
        Vector3 start,
        Vector3 end,
        string[]? textureSet = null,
        Vector3? color = default
        )
```

```
        {
            Replace(start, end);

            if (textureSet != null)
            {
                textureHandlers = new int[textureSet.Length];
                for (int i = 0; i < textureSet.Length; i++)
                {
                    textureHandlers[i] = TextureLoader.LoadFromFile(textureSet[i]);
                }
            }
        }

        public void Replace(Vector3 start, Vector3 end)
        {
            chars[0] = start;
            chars[1] = end;

            plane = new PlaneFunction(start, end);

            Vector2 v1 = new Vector2(start.X, start.Y);
            Vector2 v2 = new Vector2(end.X, end.Y);

            Vector4 heights = new Vector4(end.Z, start.Z, start.Z, end.Z);

            sqr = new Square(builder: new Vector2[] { v1, v2 }, heights: heights, isSection:
true);

            showAreaMesh = false;
            intersected = false;
            showPolar = false;
        }

        public Vector3[]? IntersectTiled(float[] X, float[] Y, float[][] Data)
        {
            int cols = X.Length - 1;
            int rows = Y.Length - 1;

            float xmax = MathF.Max(chars[0].X, chars[1].X);
            xmax = X[cols] < xmax ? X[cols] : xmax;

            float xmin = MathF.Min(chars[0].X, chars[1].X);
            xmin = X[0] > xmin ? X[0] : xmin;
```

```
float ymax = MathF.Max(chars[0].Y, chars[1].Y);
ymax = Y[rows] < ymax ? Y[rows] : ymax;

float ymin = MathF.Min(chars[0].Y, chars[1].Y);
ymin = Y[0] > ymin ? Y[0] : ymin;

if (xmax < xmin)
{
   float temp = xmin;
   xmin = xmax;
   xmax = temp;
}
if (ymax < ymin)
{
   float temp = ymin;
   ymin = ymax;
   ymax = temp;
}

int iis = 0;
int iie = 0;
int jjs = 0;
int jje = 0;

for (iis = 0; iis < cols && xmin > X[iis]; iis++) ;
for (iie = jjs; iie < cols && xmax > X[iie]; iie++) ;
for (jjs = 0; jjs < rows && ymin > Y[jjs]; jjs++) ;
for (jje = jjs; jje < rows && ymax > Y[jje]; jje++) ;

float[] x = new float[iie - iis + 1];
float[] y = new float[jje - jjs + 1];
float[][] data = new float[x.Length][];
for (int i = 0; i < x.Length; i++)
{
   data[i] = new float[y.Length];
}

for (int j = jjs; j <= jje; j++)
{
   y[j - jjs] = Y[j];
}
```

```
        for (int i = iis; i <= iie; i++)
        {
            x[i - iis] = X[i];
            for (int j = jjs; j <= jje; j++)
            {
                data[i - iis][j - jjs] = Data[i][j];
            }
        }
        int amount = (x.Length - 1) * y.Length + (y.Length - 1) * x.Length;
        LineFunction[] lines = new LineFunction[amount];

        for (int i = 0; i < x.Length - 1; i++)
        {
            for (int j = 0; j < y.Length; j++)
            {
                lines[i * y.Length + j] = new LineFunction(new float[] { x[i], x[i + 1], y[j],
y[j], data[i][j], data[i + 1][j] });
            }
        }

        for (int i = 0; i < x.Length; i++)
        {
            for (int j = 0; j < y.Length - 1; j++)
            {
                lines[(x.Length - 1) * y.Length + i * (y.Length - 1) + j] = new
LineFunction(new float[] { x[i], x[i], y[j], y[j + 1], data[i][j], data[i][j + 1] });
            }
        }
        _assignAreaMesh(lines);

        List<Vector3> res = new List<Vector3>();
        for (int i = 0; i < amount; i++)
        {
            LineFunction line = lines[i];
            Vector3? pos = line.Intersect(plane.A, plane.B, plane.C, plane.D);

            if (pos != null && line.DotOnLine(pos.Value))
            {
                res.Add(new Vector3(pos.Value));
            }
        }

        amount = res.Count;
```

```
            if (amount != 0)
            {
                intersected = true;
                dots = new Dot[amount];
                funcLine = new Line[amount - 1];
                Vector3[] intersects = res.ToArray();

                for (int i = 0; i < amount; i++)
                {
                    dots[i] = new Dot(intersects[i], color: ((Vector4)Color4.Red).Xyz, size: 6);

                    if(i != 0)
                    {
                        funcLine[i - 1] = new Line(dots[i - 1].position, dots[i].position, color:
((Vector4)Color4.Yellow).Xyz, width: 5);
                    }
                }

                CountPolarFunction();
                return intersects;
            }
            return null;
        }

        private void _assignAreaMesh(LineFunction[] lines)
        {
            areaMesh = new Line[lines.Length];
            for (int i = 0; i < lines.Length; i++)
            {
                areaMesh[i] = new Line(lines[i].start, lines[i].end,
((Vector4)Color4.Cyan).Xyz, width: 3);
            }
            showAreaMesh = true;
        }

        private float _countAngleToRotateYZ(Vector3 a, Vector3 b)
        {
            LineFunction line = new LineFunction(a, b);
            LineFunction line1 = new LineFunction(new Vector3(0, 1, a.Z), new Vector3(0,
0, b.Z));
            return (line1.AngleBetweenLines(new Vector3(line.m1, line.p1, line.l1)) -
MathF.PI / 2) / 2;
        }
```

```
private void CountPolarFunction()
{
   if (intersected)
   {
      if (dots.Length > 1)
      {
         Vector3 a = dots[0].position;
         Vector3 b = dots[1].position;

         float angle = _countAngleToRotateYZ(a, b);

         LineFunction[] rotated = new LineFunction[funcLine.Length];

         funcPolar = new Line[rotated.Length];
         for(int i = 0; i < rotated.Length; i++)
         {

            rotated[i] = new LineFunction(dots[i].position, dots[i+1].position);

            rotated[i] = rotated[i].Rotate(angle);

            funcPolar[i] = new Line(
               new Vector3(0, rotated[i].start.Y, rotated[i].start.Z),
               new Vector3(0, rotated[i].end.Y, rotated[i].end.Z),
               color: ((Vector4)Color4.Purple).Xyz,
               width: 3
               );
         }

         funcPolarArgs = new float[dots.Length][];
         for(int i = 0; i < funcPolarArgs.Length; i++)
         {
            funcPolarArgs[i] = new float[2];

            float x, y;
            if (i < funcPolar.Length)
            {
               x = funcPolar[i].position.Y;
               y = funcPolar[i].position.Z;
            }
            else
            {
```

```
                x = rotated[i - 1].end.Y;
                y = rotated[i - 1].end.Z;
            }
            funcPolarArgs[i][0] = MathF.Sqrt(x * x + y * y);
            funcPolarArgs[i][1] = MathF.Atan(y / x);
        }
        hasPolar = true;
        showPolar = true;
    }
  }
}

public void ExportWriteStreamPolarArgs(JsonWriter writer)
{
  writer.WriteStartObject();
  writer.WritePropertyName("Args");
  writer.WriteStartArray();
  foreach (var f in funcPolarArgs)
  {
    writer.WriteStartObject();
    writer.WritePropertyName("r");
    writer.WriteValue(f[0]);
    writer.WritePropertyName("phi");
    writer.WriteValue(f[1]);
    writer.WriteEndObject();
  }
  writer.WriteEndArray();
  writer.WriteEndObject();
}

public void SwitchPlaneDisplaying()
{
  isEnabled = !isEnabled;
}

public void Render(Shader shader, PrimitiveType? primitive = null)
{
  if (intersected)
  {
    for (int i = 0; i < dots.Length && showAreaMesh; i++)
    {
      dots[i].Render(shader, primitive);
    }
```

```
            for (int i = 0; i < funcLine.Length && showAreaMesh; i++)
            {
               funcLine[i].Render(shader, primitive);
            }
            for (int i = 0; i < areaMesh.Length && showAreaMesh; i++)
            {
               areaMesh[i].Render(shader, primitive);
            }
            if (hasPolar && showPolar)
            {
               for(int i = 0; i < funcPolar.Length; i++)
               {
                  funcPolar[i].Render(shader, primitive);
               }
            }
         }

         if (isEnabled)
         {
            TextureLoader.UseMany(shader, textureHandlers);

            sqr.Render(shader);
         }

      }
   }
}
```

ComplexPlaneTriangular.cs:

```
using LearnOpenTK.Common;
using OpenTK.Graphics.OpenGL4;
using OpenTK.Mathematics;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Windows.Forms;
using TriangleNet.Geometry;
using TriangleNet.Meshing;
using TriangleNet.Smoothing;

namespace SceneEditor.editor
```

```
{
   public class ComplexPlaneTriangular : Transformable
   {
      Triangle[]? connections = null;
      private TriangularedDataSet? dataTriangulared = null;
      public bool showGeometry = false;

      public Vector3[] data;
      public List<Vector3> dataSetAdjustable;

      Dot[] dots;
      private int dotsize = 6;
      private Vector3 dotColor = new Vector3(255, 0, 0);
      private Vector3 dotColorHighlighted = new Vector3(0, 255, 0);
      public bool showDots = true;


      private alglib.spline2dinterpolant? interp = null;

      int primitiveCurrent = 1;
      float lineWidth;
      PrimitiveType[] stylesSwitcher = new PrimitiveType[]
         {
            PrimitiveType.Triangles,
            PrimitiveType.LineStrip,
            PrimitiveType.Points
         };
      public PrimitiveType drawStyle;


      public ComplexPlaneTriangular(Vector3[]? inputData = null,
                                    float[]? X = null,
                                    float[]? Y = null,
                                    float[]? Z = null,
                                    string[]? textureSet = null,
                                    bool shouldTriangulate = true)
      {
         if (inputData == null)
         {
            if (X == null || Y == null || Z == null)
            {
               Functions.GenerateRandomPoints(out data, start: -20f, end: 20f,
amountLow: 100, amountHigh: 200);
```

```
            data = Functions.FixLowHight(data);
        }
    }
    else
    {
        data = inputData;
    }

    dataSetAdjustable = data.ToList();
    lineWidth = 1;

    UpdateVertices();

    if (shouldTriangulate)
    {
        generateDelaunayTriangulationFromData();
    }

    if (textureSet != null)
    {
        textureHandlers = new int[textureSet.Length];
        for (int i = 0; i < textureSet.Length; i++)
        {
            textureHandlers[i] = TextureLoader.LoadFromFile(textureSet[i]);
        }
    }

    isEnabled = true;
    drawStyle = stylesSwitcher[primitiveCurrent];
}

public ComplexPlaneTriangular(TriangularedDataSet dataSet,
                              string[]? textureSet = null)
{
    connections = new Triangle[dataSet.Triangles.Length];
    dataSetAdjustable = new();
    for (int i = 0; i < connections.Length; i++)
    {
        Vector3[] points = dataSet.Triangles[i].ToVector3Set();
        connections[i] = new Triangle(points);
        for (int j = 0; j < points.Length; j++)
        {
            dataSetAdjustable.Add(points[j]);
```

```
        }
    }

    lineWidth = 1;
    UpdateVertices();

    if (textureSet != null)
    {
        textureHandlers = new int[textureSet.Length];
        for (int i = 0; i < textureSet.Length; i++)
        {
            textureHandlers[i] = TextureLoader.LoadFromFile(textureSet[i]);
        }
    }

    isEnabled = true;
    drawStyle = stylesSwitcher[primitiveCurrent];
}

public void UpdateVertices()
{
    data = dataSetAdjustable.ToArray();
    dots = new Dot[data.Length];
    for (int i = 0; i < dots.Length; i++)
    {
        dots[i] = new Dot(data[i], dotColor, dotsize);
    }
}

private Polygon _createPolyFromPoints()
{
    Polygon poly = new Polygon();

    for (int i = 0; i < data.Length - 1; i++)
    {
        poly.Add(new Vertex(data[i].X, data[i].Y));
        poly.Add(new Segment(
                new Vertex(data[i].X, data[i].Y),
                new Vertex(data[i + 1].X, data[i + 1].Y)
                ));
    }
    poly.Add(new Vertex(data[data.Length - 1].X, data[data.Length - 1].Y));
    poly.Add(new Segment(
```

```
                new Vertex(data[data.Length - 1].X, data[data.Length - 1].Y),
                new Vertex(data[0].X, data[0].Y)
                ));
            return poly;
        }

        public bool generateDelaunayTriangulationFromData(bool isConvex = false, bool
fixArea = false, bool accurateTriangulation = false)
        {
            try
            {
                interp = Functions.Interpolate(data);
                var Coptions = new ConstraintOptions();

                Coptions.Convex = isConvex;
                if (accurateTriangulation)
                {
                    Coptions.ConformingDelaunay = true;
                }
                var Qoptions = new QualityOptions();
                if (fixArea)
                {
                    Qoptions.MaximumArea = (Math.Sqrt(3) / 4 * 0.2 * 0.2) * 1.45;
                }

                var mesh = _createPolyFromPoints().Triangulate(Coptions, Qoptions);

                if (accurateTriangulation && fixArea)
                {
                    try
                    {
                        var smoother = new SimpleSmoother();
                        smoother.Smooth(mesh, 25);
                        _readConnectionsFromMesh(mesh);
                    }
                    catch (Exception ex)
                    {
                        _readConnectionsFromMesh(mesh);
                        MessageBox.Show("Failed to make mesh smoother");
                    }
                }
                else
                {
```

```
            _readConnectionsFromMesh(mesh);
        }

    } catch (Exception ex) { return false; }
    return true;
}

public void HighlightDot(int index)
{
    for (int i = 0; i < index; i++)
    {
        dots[i] = new Dot(dots[i].position, dotColor, dotsize);
    }
    dots[index] = new Dot(dots[index].position, dotColorHighlighted, dotsize * 3);
    for (int i = index + 1; i < dots.Length; i++)
    {
        dots[i] = new Dot(dots[i].position, dotColor, dotsize);
    }
}

private void _readConnectionsFromMesh(IMesh mesh)
{
    List<Vector2> trgs = new();
    List<int> indices = new();

    foreach (var t in mesh.Triangles)
    {
        for (int j = 2; j >= 0; j--)
        {
            int amount = trgs.Count;
            bool found = false;
            var vx = t.GetVertex(j);
            for (int i = 0; i < amount && !found; i++)
            {
                if ((trgs[i].X == vx.X) && (trgs[i].Y == vx.Y))
                {
                    indices.Add(i);
                    found = true;
                }

            }
            if (!found)
            {
```

```
                trgs.Add(new Vector2((float)vx.X, (float)vx.Y));
                indices.Add(amount);
            }
        }
    }

    int[] iRes = indices.ToArray();

    Vector3[] vector3s = Functions.recalculateBySpline(interp, trgs.ToArray());

    connections = new Triangle[iRes.Length / 3];
    TriangleInfo[] triangles = new TriangleInfo[connections.Length];
    for (int i = 0; i < connections.Length; i++)
    {
        Vector3[] points = new Vector3[3];
        for (int j = 0; j < 3; j++)
        {
            points[j] = new(vector3s[iRes[i * 3 + j]]);
        }
        triangles[i] = new(points);
        connections[i] = new(points);
    }
    dataTriangulared = new(triangles);
    showGeometry = true;
}

public ComplexPlaneTile ConvertToTiledByInterpolation(
    int limitSide = 50,
    bool copyCurrentTexturePack = false
    )
{
    if (interp == null)
    {
        interp = Functions.Interpolate(data);
    }
    float xmin = data[0].X;
    float xmax = xmin;
    float ymin = data[0].Y;
    float ymax = ymin;

    float mindifx = float.MaxValue;
    float mindify = float.MaxValue;
```

```
for (int i = 1; i < data.Length; i++)
{
   float vx = data[i].X;
   float vy = data[i].Y;
   float lengthx = MathF.Abs(data[i - 1].X - data[i].X);
   float lengthy = MathF.Abs(data[i - 1].Y - data[i].Y);

   if (mindifx > lengthx)
   {
      mindifx = lengthx;
   }
   if (mindify > lengthy)
   {
      mindify = lengthy;
   }

   if (xmin > vx)
   {
      xmin = vx;
   }
   else
   {
      if (xmax < vx)
      {
         xmax = vx;
      }
   }
   if (ymin > vy)
   {
      ymin = vy;
   }
   else
   {
      if (ymax < vy)
      {
         ymax = vy;
      }
   }
}

float dividerx = (xmax - xmin) / limitSide;
float dividery = (ymax - ymin) / limitSide;
```

```
            float[] X = Functions.Arrange(xmin, xmax, (xmax - xmin) / mindifx > limitSide
* 2 ? dividerx : mindifx);
            float[] Y = Functions.Arrange(ymin, ymax, (ymax - ymin) / mindify > limitSide
* 2 ? dividery : mindify);
            float[][] Z = Functions.recalculateBySpline(interp, X, Y);

            return new ComplexPlaneTile(X: X, Y: Y, Z: Z,
                textureHandlersCopy: copyCurrentTexturePack ? (textureHandlers != null ?
textureHandlers : null) : null
                );
        }

        public void SwitchDrawStyle()
        {
            primitiveCurrent++;
            if (stylesSwitcher.Length == primitiveCurrent)
            {
                primitiveCurrent = 0;
            }
            drawStyle = stylesSwitcher[primitiveCurrent];
        }

        public TriangularedDataSet? ExportPointDataSet()
        {
            return dataTriangulared;
        }

        public Vector3[]? ExportPoints()
        {
            return data;
        }

        private protected override void _renderObjects(Shader shader, PrimitiveType?
primitive)
        {
            if (showDots)
            {
                for (int i = 0; i < dots.Length; i++)
                {
                    dots[i].Render(shader);
                }
            }
```

```
            if (connections != null && showGeometry)
            {
                GL.LineWidth(lineWidth);
                GL.PointSize(lineWidth * 5);
                for (int i = 0; i < connections.Length; i++)
                {
                    connections[i].Render(shader, drawStyle);
                }
            }
        }
    }
}
```

lightShader.vert:

```
#version 450 core

layout (location = 0) in vec3 aPos;
layout (location = 1) in vec3 aColor;
layout (location = 2) in vec2 aTexCoord;
layout (location = 3) in vec3 aNormal;

out vec3 vertexColor;
out vec3 Normal;
out vec3 FragPos;
out vec3 lightPos;
out vec2 TexCoord;
out float height;
out vec2 TextureGradRange;

uniform mat4 transform;
uniform mat4 model;
uniform mat4 view;
uniform mat4 projection;

uniform vec2 textureGradRange;

uniform mat4 model_cramble;

uniform vec3 LightPos = vec3(0, 0, 100);
```

```
void main()
{
	vec4 transformed = transform * vec4(aPos, 1.0);
	gl_Position = transformed * model * view * projection;
	FragPos = vec3(model * transformed);

	Normal = aNormal;

	vertexColor = aColor;

	lightPos = LightPos;

	TexCoord = aTexCoord;

	height = aPos.z / aPos.length + transform[3][2];

	TextureGradRange = textureGradRange;
}
```

shader.frag:

```
#version 450 core
out vec4 FragColor;

in vec3 vertexColor;
in vec3 Normal;
in vec3 FragPos;
in vec3 lightPos;
in vec2 TexCoord;


in float height;
in vec2 TextureGradRange;
const int amountOfTextures = 3;

uniform sampler2D textures[amountOfTextures];

uniform vec3 lightColor = vec3(1);

uniform vec3 viewPos;

uniform float lightPosHeight;
```

```
struct Material {
   vec3 ambient;
   vec3 diffuse;
   float opacity;
};

uniform Material material;

float textureGradCalc(vec2 range, float value){
   if(range.x == 0 && range.y == 0)
   {
     return 0.5;
   }

   float valueN = value * 0.2 + 0.2;

   return valueN;
}

vec4 textureMixCalc(float grad){

   vec4 textureColor = vec4(0);

   vec4 oTexture12 = mix(texture(textures[0], TexCoord), texture(textures[1],
TexCoord), grad);
   vec4 oTexture23 = mix(texture(textures[1], TexCoord), texture(textures[2],
TexCoord), grad);
   vec4 oTexture13 = mix(texture(textures[0], TexCoord), texture(textures[2],
TexCoord), grad);

   vec4 oTexture1 = mix(oTexture12, oTexture13, grad);
   vec4 oTexture2 = mix(oTexture12, oTexture23, grad);
   vec4 oTexture = mix(oTexture1, oTexture2, grad);

   textureColor = oTexture;

   return textureColor;
}

void main()
{
   vec3 ambient = material.ambient * lightColor;
```

```
    vec3 norm = normalize(Normal);
    vec3 lightDir = normalize(lightPos - FragPos);

    vec3 viewDir = normalize(viewPos - FragPos);
    vec3 reflectDir = reflect(-lightDir, norm);

    float spec = pow(max(dot(viewDir, reflectDir), 0.0), material.shininess);
    vec3 specular = material.specular * spec * lightColor;

    float diff = max(dot(norm, lightDir), 0.0);
    vec3 diffuse = diff * lightColor * material.diffuse;

    vec4 oTexture = textureMixCalc(textureGradCalc(TextureGradRange, height));

    vec3 colorBrightness = (ambient + diffuse) * vertexColor;

    FragColor = oTexture * vec4(colorBrightness, material.opacity);
}
```

Moveable.cs:

```
using LearnOpenTK.Common;
using OpenTK.Graphics.OpenGL4;
using OpenTK.Mathematics;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace SceneEditor.editor
{
    public class Moveable : IRenderable
    {
        public bool isEnabled = false;

        public Vector3 position = Vector3.Zero;

        private protected Matrix4 transform = Matrix4.Identity;
        private protected Matrix4 originShift = Matrix4.Identity;
```

```
        public virtual void Move(Vector3 shifts)
        {
            position += shifts;
            originShift = originShift * Matrix4.CreateTranslation(shifts);

            TransformCombiner();
        }

        public virtual void TransformCombiner()
        {
            transform *= originShift;
        }

        public virtual void TransformClean()
        {
            transform = originShift = Matrix4.Identity;
        }

        private protected virtual void _renderObjects(Shader shader, PrimitiveType?
primitive)
        {
            return;
        }

        private protected virtual void _prepareRendering(Shader shader)
        {
            var id = GL.GetUniformLocation(shader.Handle, "transform");
            GL.UniformMatrix4(id, false, ref transform);
        }

        public virtual void Render(Shader shader, PrimitiveType? primitive = null)
        {
            if (isEnabled)
            {
                _prepareRendering(shader);
                _renderObjects(shader, primitive);
            }
        }


    }
}
```

Transformable.cs:

```
using LearnOpenTK.Common;
using OpenTK.Graphics.OpenGL4;
using OpenTK.Mathematics;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace SceneEditor.editor
{
    public class Material
    {
        public Vector3 ambient = new Vector3(1);
        public Vector3 diffuse = new Vector3(1f);
        public float opacity = 1;

        public Material(
            Vector3? ambient = null,
            Vector3? diffuse = null,
            float? opacity = null
            )
        {
            if (ambient != null)
                this.ambient = ambient.Value;
            if(diffuse != null)
                this.diffuse = diffuse.Value;
            if(opacity != null)
                this.opacity = opacity.Value;
        }
    }


    public class Transformable : Moveable
    {
        protected int[] textureHandlers;

        private protected Matrix4 rotation = Matrix4.Identity;
        private protected Matrix4 scale = Matrix4.Identity;
```

```
private protected Vector2 _range = new Vector2(0,0);

private protected Transformable? parent = null;
private protected Material material = new();

public virtual void Rotate(Vector3 angles)
{
   Matrix4 X = Matrix4.CreateRotationX(angles.X);
   Matrix4 Y = Matrix4.CreateRotationY(angles.Y);
   Matrix4 Z = Matrix4.CreateRotationZ(angles.Z);
   rotation = rotation * X * Y * Z;
   TransformCombiner();
}

public virtual void Scale(Vector3 scalar)
{
   scale = scale * Matrix4.CreateScale(scalar);
   TransformCombiner();
}

public virtual new void TransformCombiner()
{
   transform = scale * rotation * originShift;
}

public virtual new void TransformClean()
{
   transform = scale = originShift = rotation = Matrix4.Identity;
}

private protected void _applyTextures(Shader shader)
{
   TextureLoader.UseMany(shader, textureHandlers);
}

private protected void _applyMaterial(Shader shader)
{
   Material mat = (parent != null) ? parent.material : new();

   shader.SetVector3("material.ambient", mat.ambient);
   shader.SetVector3("material.diffuse", mat.diffuse);
   shader.SetFloat("material.opacity", mat.opacity);
}
```

```
        private protected override void _prepareRendering(Shader shader)
        {
            var id = GL.GetUniformLocation(shader.Handle, "transform");
            GL.UniformMatrix4(id, false, ref transform);

            int attr = GL.GetUniformLocation(shader.Handle, "textureGradRange");
            if(parent == null)
            {
                GL.Uniform2(attr, ref _range);
            }
            else
            {
                GL.Uniform2(attr, parent._range);
            }

        }

        public override void Render(Shader shader, PrimitiveType? primitive = null)
        {
            if (isEnabled)
            {
                _applyTextures(shader);
                _prepareRendering(shader);
                _applyMaterial(shader);
                _renderObjects(shader, primitive);
            }
        }
    }
}
```

MainWindow.xaml

```
<Window x:Class="SceneEditor.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:SceneEditor"
        xmlns:glWpfControl="clr-namespace:OpenTK.Wpf;assembly=GLWpfControl"
        mc:Ignorable="d"
```

```
      Title="BatymetriaEditor (by Bohdan Savchuk)" Width="1600" Height="900"
WindowStyle="ThreeDBorderWindow">
   <Grid x:Name="panelRoot" Background="Silver" Focusable="True"
PreviewMouseDown="OnPreviewMouseKeyPressed">

      <Grid.ColumnDefinitions>
         <ColumnDefinition Width="3*"></ColumnDefinition>
         <ColumnDefinition Width="1*"></ColumnDefinition>
      </Grid.ColumnDefinitions>
      <Rectangle Fill="DimGray" Grid.Column="0"></Rectangle>
      <Rectangle Fill="SlateGray" Grid.Column="1"></Rectangle>

      <Grid x:Name="panelBench" Background="FloralWhite" Grid.Column="0"
Grid.Row="0" Focusable="True">
         <Grid.RowDefinitions>
            <RowDefinition Height="3*"/>
            <RowDefinition Height="1*"/>
         </Grid.RowDefinitions>
         <Rectangle Fill="CadetBlue" Grid.Row="0"></Rectangle>
         <Rectangle Fill="CadetBlue" Grid.Row="1"></Rectangle>
         <GridSplitter Grid.Row="0" BorderThickness="2" BorderBrush="Black"
VerticalAlignment="Bottom" HorizontalAlignment="Stretch"/>

         <Grid Grid.Row="0" x:Name="panelTabs">
            <Grid.RowDefinitions>
               <RowDefinition Height="*"/>
               <RowDefinition Height="50"/>
            </Grid.RowDefinitions>

            <glWpfControl:NonReloadingTabControl Grid.Row="0" Margin="5"
x:Name="tabWindows" Background="Wheat">
               <TabItem Header="Main" x:Name="glControl1">
                  <glWpfControl:GLWpfControl x:Name="glMain" Margin="5"
Focusable="True" Render="OnRender"
PreviewMouseWheel="OnPreviewMouseWheel"/>
               </TabItem>
            </glWpfControl:NonReloadingTabControl>

            <StatusBar Grid.Row="1" Height="45" Margin="5">
               <StatusBarItem>
                  <TextBlock x:Name="textFps"></TextBlock>
               </StatusBarItem>
               <StatusBarItem>
```

```
                <TextBlock x:Name="textKey"></TextBlock>
            </StatusBarItem>
            <StatusBarItem>
                <TextBlock x:Name="textMouse"></TextBlock>
            </StatusBarItem>
        </StatusBar>
    </Grid>

    <TabControl Grid.Row="1" Margin="5">
        <TabItem Header="Output">
            <ListView x:Name="listViewProcess"/>
        </TabItem>
    </TabControl>

</Grid>

<Grid x:Name="panelInfo" Background="CadetBlue" Grid.Column="1"
Focusable="True">
    <Grid.RowDefinitions>
        <RowDefinition Height="1.2*"></RowDefinition>
        <RowDefinition Height="2.3*"></RowDefinition>
    </Grid.RowDefinitions>
    <GridSplitter Grid.Row="0" BorderThickness="2" BorderBrush="Black"
VerticalAlignment="Bottom" HorizontalAlignment="Stretch"/>

    <Grid Grid.Row="0">
        <TabControl Margin="5">
            <TabItem Header="Structure">
                <TabControl x:Name="editorStructure">
                    <TabItem Header="Main">
                        <DockPanel VerticalAlignment="Stretch">
                            <TreeView DockPanel.Dock="Top" VerticalAlignment="Top">

                            </TreeView>
                        </DockPanel>
                    </TabItem>
                </TabControl>
            </TabItem>
        </TabControl>
    </Grid>

    <Grid Grid.Row="1">
        <TabControl Margin="5">
```

```
                <TabItem Header="Settings">
                    <ScrollViewer>
                        <ListBox Width="Auto" x:Name="editorElementSettings">

                        </ListBox>
                    </ScrollViewer>
                </TabItem>
                <TabItem Header="Scene">
                    <ScrollViewer>
                        <ListBox Width="Auto" x:Name="editorSceneSettings">

                        </ListBox>
                    </ScrollViewer>
                </TabItem>
            </TabControl>
        </Grid>

    </Grid>

    <GridSplitter Grid.Column="0" BorderThickness="2"
BorderBrush="Black"></GridSplitter>


  </Grid>
</Window>
```

ДОДАТОК Б

Редактор батиметрії для сцени гідроакустичного експерименту

Тези

УКР.НТУУ”КПІ імені Ігоря Сікорського”_ТЕФ_АПЕПС_ТМ-81269_22Б 12-1

Аркушів 3

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
УКРАЇНСЬКА АСОЦІАЦІЯ З ПРИКЛАДНОЇ ГЕОМЕТРІЇ
МЕЛІТОПОЛЬСЬКИЙ ДЕРЖАВНИЙ ПЕДАГОГІЧНИЙ УНІВЕРСИТЕТ
ІМЕНІ БОГДАНА ХМЕЛЬНИЦЬКОГО
МЕЛІТОПОЛЬСЬКА ШКОЛА ПРИКЛАДНОЇ ГЕОМЕТРІЇ

ТЕЗИ ДОПОВІДЕЙ

23 МІЖНАРОДНОЇ НАУКОВО – ПРАКТИЧНОЇ КОНФЕРЕНЦІЇ

СУЧАСНІ ПРОБЛЕМИ ГЕОМЕТРИЧНОГО МОДЕЛЮВАННЯ

УКРАЇНА, МЕЛІТОПОЛЬ
01-04 ЧЕРВНЯ 2021 р.

ОРГАНІЗАТОРИ КОНФЕРЕНЦІЇ

Міністерство освіти і науки України
Українська асоціація з прикладної геометрії
Мелітопольський державний педагогічний університет
імені Богдана Хмельницького
Мелітопольська школа прикладної геометрії

ПРИЙМАЮЧА ОРГАНІЗАЦІЯ: Мелітопольський державний педагогічний університет імені Богдана Хмельницького

НАУКОВО-ПРОГРАМНИЙ КОМІТЕТ:
Голова: Солоненко А.М. – ректор Мелітопольського державного педагогічного університету імені Богдана Хмельницького
Заступник голови: Найдиш А.В. – Мелітополь, Україна
Співголови:
Ванін В.В. – НТУУ «КПІ», Київ, Україна
Підгорний О.Л. – КНУБА, Київ, Україна
Плоский В.О. – КНУБА, Київ, Україна
Члени науково-програмного комітету:
Балюба І.Г. – Мелітополь, Україна;
Белицький Г. – Беер Шева, Ізраїль;
Боуди В. – Ель-Айн, Оае;
Верещага В.М. – Мелітополь, Україна;
Гнатушенко В.В. - Дніпропетровськ, Україна;
Єремєєв В.С. – Мелітополь, Україна;
Ковальов С.М. – Київ, Україна;
Ковальов Ю.М. – Київ, Україна;
Корчинський В.М. – Дніпропетровськ, Україна;
Куценко Л.М. – Харків, Україна;
Мартин Є.В. – Львів, Україна;
Мартинов В.Л. – Київ, Україна;
Панченко А.І. – Мелітополь, Україна;
Пилипака С.Ф. – Київ, Україна;
Протасов Р.В. - Братислава, Словакія;
Репелевич О. – Ченстохово, Польща;
Сергейчук О.В. – Київ, Україна;
Сердюкова Н.В. – Ла-Хойя, Каліфорнія, США;
Тулученко Г.Я. – Херсон, Україна;
Уяма А. – Ченстохово, Польща;
Хомченко А.Н. - Миколаїв, Україна;
Черніков О.В. – Харків, Україна;
Шоман О.В. - Харків, Україна.

Анпілогова В.О. к. т. н.,
Ботвіновська С.І., д. т. н.,
Золотова А.В., к. т. н.,
Суліменко А.Г., к. т. н.,
Київський національний університет будівництва і архітектури (Україна)

Залевська О.В., к.т.н.,
Яблонський П.М., к.т.н.,
Ладогубець Т. С., к.т.н.,
Савчук Б.І.,
Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського»

УДОСКОНАЛЕННЯ РЕАЛІЗАЦІЇ МЕТОДУ АПРОКСИМАЦІЇ ПРИ РОБОТІ З ГЕОІНФОРМАЦІЙНИМИ ДАНИМИ

З розвитком комп'ютерної графіки новий виток розвитку отримав напрям побудови ландшафтного середовища. Більшість існуючих програмних застосунків працюють з неперервними даними. Такі дані отримані за допомогою точкових знімків землі та передаються у вигляді зображень. Для обробки методу побудови неперервного ландшафту використовують методи апроксимації, які можливо поділили на 4 групи: з використанням регулярної решітки, з використання іррегулярної решітки графу місцевості (триангуляція) та за допомогою клітинних автоматів. Серед наведених методів істотно вирізняється метод апроксимації за допомогою клітинних автоматів, оскільки позбавлений недоліків трьох попередніх методів. До таких недоліків відносять накладання структур, можливість побудови дискретних елементів. А до переваг методу відносять динамічну поведінку в процесі генерації та можливість використання в реальному часі.

Велика кількість програмних застосунків, що реалізують алгоритми апроксимації не надають весь необхідний математичний апарат для побудови та генерації ландшафтів, як реальних так і віртуальних. Пропонується використовувати програмний застосунок на основі json файлу відповідного зображення. Це призводить до зменшення кількості витраченого об'єму оперативної пам'яті на зберігання та відновлення графічної складової геоінформаційних файлів, що в свою чергу збільшує швидкодію комп'ютеру та зменшує вимоги до техніки під час його використання. Залевська О.В., к.т.н., Яблонський П.Н., к.т.н., Фіногенов О.Д., к.т.н., Сидоренко Ю.В, к.т.н., Ситник А.Ю. Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського»