МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра фундаментальної та прикладної математики

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

на тему: **«ПОБУДОВА ДИСКРЕТНОГО ПРЕДСТАВЛЕННЯ ПЛОСКИХ ОБЛАСТЕЙ ЗА ДОПОМОГОЮ ТРІАНГУЛЯЦІЇ ДЕЛОНЕ»**

Виконав: студент 2 курсу, групи 8.1132

спеціальності 113 Прикладна математика
(шифр і назва спеціальності)

освітньої програми Прикладна математика
(назва освітньої програми)

М. С. Барабаш
(ініціали та прізвище)

Керівник завідувач кафедри фундаментальної та прикладної математики, професор, д.т.н. Гребенюк С.М.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Рецензент декан математичного факультету, професор, д.т.н. Гоменюк С.І.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя
2023

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра фундаментальної та прикладної математики

Рівень вищої освіти магістр

Спеціальність 113 Прикладна математика

(шифр і назва)

Освітня програма Прикладна математика

ЗАТВЕРДЖУЮ

Завідувач кафедри
фундаментальної та прикладної
математики, професор, д.т.н.

_____ Гребенюк С.М.

(підпис)

« ____ » _____ 2023 р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Барабашу Микиті Сергійовичу

(прізвище, ім'я та по-батькові)

1. Тема роботи (проекту) Побудова дискретного представлення плоских областей за допомогою тріангуляції Делоне

керівник роботи (проекту) Гребенюк Сергій Миколайович, професор, д.т.н.

(прізвище, ім'я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від «____» _____ 2023 року № _____

2. Строк подання студентом роботи _____

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Основні поняття та властивості тріангуляції Делоне
2. Реалізація алгоритму тріангуляції Делоне

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) _____

Презентація

6. Консультанти розділів роботи

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

7. Дата видачі завдання _____

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	22.05.2023	
2.	Збір вихідних даних.	28.05.2023	
3.	Обробка методичних та теоретичних	14.06.2023	
	джерел.		
4.	Розробка програмного забезпечення	25.09.2023	
5.	Опис алгоритмів та реалізації програмного забезбечення	06.10.2023	
6.	Оформлення та нормоконтроль	11.11.2023	
	кваліфікаційної роботи.		
7.	Захист кваліфікаційної роботи.		

Студент _____ М. С. Барабаш
(підпис) (ініціали та прізвище)

Керівник роботи _____ С. М. Гребенюк
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ О. Г. Спиця

РЕФЕРАТ

Кваліфікаційна робота магістра «Побудова дискретного представлення плоских областей за допомогою тріангуляції Делоне»: 62 с., 15 рис., 10 джерел.

АЛГОРИТМИ ТРИАНГУЛЯЦІЇ, АПРОКСИМАЦІЯ ПОВЕРХНІ, ВЗАЄМОДІЯ ІЗ ВВЕДЕННЯМ КОРИСТУВАЧА, ВЛАСТИВОСТІ ТРИАНГУЛЯЦІЇ ДЕЛОНЕ, ВІДОБРАЖЕННЯ ДАНИХ ДЛЯ ОБЛАСТЕ, ВІЗУАЛІЗАЦІЯ ГЕОМЕТРИЧНИХ ОБ'ЄКТІВ, ГРАФІЧНЕ ПРЕДСТАВЛЕННЯ ТРИАНГУЛЯЦІЇ, МЕТОДИ ОПТИМІЗАЦІЇ ТРИАНГУЛЯЦІЇ, ОБ'ЄКТНО-ОРІЄНТОВАНЕ ПРОГРАМУВАННЯ, ОПТИМАЛЬНІСТЬ ТРИАНГУЛЯЦІЇ, ПЛОСКІ ОБЛАСТІ, ПОБУДОВА ТРИАНГУЛЯЦІЇ, РОЗДІЛЬНІСТЬ ТА ТОЧНІСТЬ ТРИАНГУЛЯЦІЇ, ТРІАНГУЛЯЦІЯ ДЕЛОНЕ.

Об’єкт дослідження – методи побудови дискретного представлення плоских областей, зокрема тріангуляція Делоне.

Мета роботи: Розгляд та розробка підходів та програмного забезпечення для ефективної побудови дискретного представлення плоских областей за допомогою алгоритмів тріангуляції Делоне.

Метод дослідження – аналітичний, чисельний, порівняльний.

У даній роботі проведено комплексне дослідження підходів та реалізацію програмного додатка, спрямованого на побудову дискретного представлення плоских областей за допомогою тріангуляції Делоне. Ретельно розглянуті основні поняття тріангуляції плоских областей, що становлять основу для подальших досліджень. Проведено вивчення особливостей даного методу тріангуляції, що допомогло у зрозумінні технічних відмінностей та переваг цього підходу. Розглянуті та порівняні різноманітні алгоритми, включаючи інкрементальну тріангуляцію, "Bowyer-Watson", алгоритм замітання прямою та "поділи та пануй", для визначення оптимальних методів у конкретних ситуаціях. Розглянуті ключові етапи роботи з алгоритмами та

визначено критерії ефективності вхідних та вихідних даних. Розроблений програмний додаток на мові програмування C# з використанням бібліотек OpenTK та OpenGL, які є важливим інструментом для візуалізації та перевірки результатів застосування тріангуляції Делоне в практичних задачах.

SUMMARY

Master’s Qualification Thesis "Construction of a discrete representation of flat regions using Delaunay triangulation": 62 pages, 15 figures, 10 sources.

TRIANGULATION ALGORITHMS, SURFACE APPROXIMATION, INTERACTIONS FROM INTRODUCED KORISTUVACH, AUTHORITY OF DELAUNAY TRIANGULATION, ADDING DATA TO THE AREA, VISUALIZATION OF GEOMETRIC OBJECTS, TRIANGULATION GRAPHICS, OP METHODS OPTIMALITY OF TRIANGULATION, OBJECT-ORIENTED PROGRAMMING, OPTIMALITY OF TRIANGULATION, FLAT AREAS, FUTURE TRIANGULATION, DISTINCTION AND ACCURACY OF TRIANGULATION, DELONAY TRIANGULATION.

The object of research is methods of constructing a discrete representation of flat areas, in particular, Delaunay triangulation.

The purpose of the work: Review and development of software for efficient construction of a discrete representation of flat areas using Delaunay triangulation algorithms.

The research method is analytical, numerical, comparative.

In this thesis, a comprehensive study and implementation of a software application aimed at constructing a discrete representation of flat regions using Delaunay triangulation was carried out. The work covers the following key aspects: the main concepts of triangulation and planar areas, which form the basis for further research, are carefully considered. The peculiarities of this triangulation method were studied, which helped in understanding the technical differences and advantages of this approach. A variety of algorithms, including incremental triangulation, "Bowyer-Watson," straight-line sweep, and divide-and-conquer algorithms, are reviewed and compared to determine optimal methods in specific situations. The key stages of working with algorithms are considered, and the criteria for the effectiveness of input and output data are defined. The developed software

application in the C# programming language using the OpenTK and OpenGL libraries is an important tool for visualizing and verifying the results of applying Delaunay triangulation in practical problems.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

UI – графічний інтерфейс користувача.

ООП – Об’єктно-орієнтоване програмування

API – Прикладний програмний інтерфейс додатку

OpenTK (Open Toolkit) – це набір бібліотек для розвитку графічних додатків

OpenGL (Open Graphics Library) – кросплатформенна специфікація програмного інтерфейсу для рендерингу 2D та 3D графіки

Фреймворк – зовнішній програмний продукт

ВСТУП

У сучасному інформаційному віці, коли обробка геометричних даних є важливою складовою численних технологічних застосувань, питання побудови ефективних та точних представлень плоских областей стають актуальним завданням. Одним із потужних інструментів у цьому контексті є тріангуляція Делоне – метод, у якому трикутники характеризуються тим, що вписані кола кожного трикутника не містять жодної іншої точки з множини, що забезпечує оптимальні геометричні властивості. Для кожної множини точок існує тільки одна тріангуляція Делоне. Це дозволяє отримати стійкі та добре визначені результати навіть для різних реалізацій алгоритмів. Тріангуляція Делоне гарантує глобальну оптимальність у тому сенсі, що вона максимізує найменший кут трикутника, що важливо для численних задач геометричного моделювання та обчислювальної геометрії.

Області застосування тріангуляції Делоне різноманітні: від комп’ютерної графіки та геоінформаційних систем до обчислювальної геометрії та наукових досліджень. У даній кваліфікаційній роботі проводиться глибокий аналіз та дослідження процесу побудови дискретного представлення плоских областей з використанням триангуляції Делоне. Вивчаються особливості цього методу, розглядаються різноманітні алгоритми, а також розробляється програмний додаток для візуалізації та аналізу отриманих результатів.

Ця робота має на меті висвітлення та вивчення ключових аспектів побудови та використання тріангуляції Делоне для ефективного представлення плоских областей. Розглядаються теоретичні основи методу, різні алгоритми його реалізації, а також створення програмного додатка для наочного аналізу отриманих результатів.

Вибір мови програмування C# був зумовлений його високим рівнем абстракції, ефективною роботою з об’єктами та вбудованими засобами

обробки подій. Ця мова відмінно підходить для реалізації графічних додатків та має широкий спектр функцій для реалізації високопродуктивних програм.

Використання бібліотеки OpenTK дозволило зручно взаємодіяти з графічним API OpenGL у середовищі C#. OpenTK забезпечує високорівневий доступ до функцій OpenGL, що робить процес візуалізації дискретних представлень плоских областей більш ефективним та гнучким. Використання OpenGL як графічного API стало ключовим елементом у візуалізації результатів тріангуляції Делоне. Це потужне інструментарій для взаємодії з графічним обладнанням, що забезпечує високу швидкість відображення графічних об’єктів та можливості реалізації складних ефектів.

1 ПОСТАНОВКА ЗАДАЧІ ПОБУДОВИ ДИСКРЕТНОГО ПРЕДСТАВЛЕННЯ ПЛОСКИХ ОБЛАСТЕЙ ЗА ДОПОМОГОЮ ТРІАНГУЛЯЦІЇ ДЕЛОНЕ

Метою даної кваліфікаційної роботи є розробка та реалізація алгоритму побудови дискретного представлення плоских областей з використанням тріангуляції Делоне.

У рамках дослідження планується провести ретельний аналіз існуючих методів тріангуляції Делоне та їхніх застосувань. Після цього буде розроблена математична модель для ефективного представлення плоских областей та їх тріангуляційного відображення. Наступним кроком є реалізація алгоритму тріангуляції Делоне з метою побудови дискретного представлення цих областей. На завершальному етапі дослідження планується порівняння розробленого методу з існуючими алгоритмами з урахуванням якості репрезентації та обчислювальної ефективності.

Програма отримує координати точок і використовує алгоритм тріангуляції Делоне для створення мережі трикутників, яка репрезентує геометричну область. Це дозволяє ефективно визначати взаєморозташування точок і будувати структуровану мережу для подальших обчислень чи візуалізації області.

1.1 Визначення тріангуляції та плоских областей

Тріангуляція – це процес розбиття геометричної фігури на трикутники. У своїй основі тріангуляція представляє собою поділ геометричного об’єкта на трикутники, що не перетинаються. Цей метод широко використовується в обчислювальній геометрії, комп’ютерній графіці, географічних інформаційних системах та інших сферах.

Де використовується тріангуляція:

- візуалізація та графіка: Тріангуляція дозволяє представляти складні форми або об’єкти у формі простих трикутників, що полегшує їх візуальне відображення та обробку графіки;
- обчислення площі та об’єму: Тріангуляція допомагає вирішити задачі обчислення площі фігур або об’єму об’єктів, замінивши їх на набір трикутників;
- моделювання та аналіз: У галузях, таких як комп’ютерне моделювання та аналіз структури об’єктів, тріангуляція є ефективним інструментом для подання форм та їх властивостей;
- географічні інформаційні системи (ГІС): Тріангуляція використовується для роботи з геодезичними даними, побудови та аналізу цифрових моделей рельєфу, визначення контурів;
- обчислення величин на поверхні: У біології, медицині та інших науках тріангуляція застосовується для визначення величин та характеристик на поверхні об’єктів;
- сіткові методи: Тріангуляція використовується у чисельних методах для наближеного розв’язання диференціальних рівнянь, які виникають у фізиці, інженерії та інших галузях;
- топологічні аналізи: В математиці та комп’ютерній науці тріангуляція може використовуватися для вирішення топологічних задач, таких як пошук вершин, ребер чи граней;
- використання тріангуляції різноманітне і залежить від конкретної області застосування. У загальному розумінні, тріангуляція є важливим інструментом для обробки та аналізу геометричних об’єктів у різних галузях науки та техніки.

Теорія визначення плоских областей є частиною геометрії, яка вивчає структуру та властивості плоских фігур та їх взаємні відношення. Плоскі області включають в себе різноманітні геометричні фігури, такі як трикутники, чотирикутники, круги та їх комбінації.

Основні поняття та терміни, пов’язані з визначенням плоских областей, включають:

- точка: найменша одномірна геометрична фігура без розміру;
- пряма: найкоротша відстань між двома точками. Пряма має нульову ширину та визначається двома точками;
- відрізок: частина прямої лінії між двома точками, яка має визначену довжину;
- площина: нескінченна плоска поверхня, яка розташована безпосередньо перед або за областю дослідження. Площина розглядається як безмежно велика та безтовщинна;
- трикутник: геометрична фігура, обмежена трьома відрізками;
- чотирикутник: геометрична фігура, обмежена чотирма відрізками;
- коло: множина точок у площині, що рівновіддалені від заданої точки, називаної центром кола.

Визначення плоских областей включає в себе розглядання таких понять, як периметр (сума довжин сторін) та площа (площа, закрита геометричною фігурою). Для кожного типу фігури існують відомі формули для обчислення цих характеристик. Наприклад, для трикутника площа обчислюється за такою формулою 1/2 * основа * висота, а периметр – сумою довжин усіх його сторін.

Узагальнення та загальні теореми в геометрії також дозволяють визначати властивості складних плоских областей, отримані шляхом комбінування базових геометричних елементів. Такі вивчення допомагають розуміти і прогнозувати взаємодії та властивості складних структур у плоскій геометрії.

1.2 Особливості тріангуляції Делоне

Тріангуляція Делоне – це метод поділу площини на найменші трикутники таким чином, щоб описані кола для кожного трикутника не містили жодної іншої точки з множини.

Тріангуляцію Делоне запропонував російський математик та інженер Борис Делоне в 1934 році. Також цей метод тріангуляції іноді називають триангуляцією Делоне-Вороного, оскільки його робота також включала утворення діаграм Вороного. Ці математичні концепції широко використовуються в області обчислювальної геометрії та графіки.

Делоне розробив свій алгоритм тріангуляції, щоб ефективно подавати точкові множини у вигляді трикутників, забезпечуючи оптимальність розташування цих трикутників відносно точок. Така тріангуляція широко використовується в галузі комп’ютерної графіки, обробки геодезичних та географічних даних, чисельних методів, та інших областях, де важливо використовувати прості та ефективні тріангуляційні структури.

Особливості тріангуляції Делоне включають в себе:

- оптимальність відносно вписаних кіл: Трикутники тріангуляції Делоне характеризуються тим, що вписані кола кожного трикутника не містять жодної іншої точки з множини, що забезпечує оптимальні геометричні властивості;
- унікальність: Важливо відзначити, що існує можливість існування кількох різних тріангуляцій Делоне для одного і того ж набору точок. Різні алгоритми та їх реалізації можуть давати різні результати, і враховування числових апроксимацій та погрішностей є важливим аспектом при обчисленнях. Отже, унікальність тріангуляції Делоне для конкретного набору точок залежить від конкретного алгоритму та його реалізації, а також від обробки випадків, де точки можуть лежати на одному колі;

– глобальна оптимальність: Тріангуляція Делоне гарантує глобальну оптимальність у тому сенсі, що вона максимізує найменший кут трикутника, що важливо для численних задач геометричного моделювання та обчислювальної геометрії.

Тріангуляція Делоне застосовується в побудові сіток для методу скінченних елементів завдяки гарним кутам та швидким алгоритмам побудови. Ця властивість робить її ефективним інструментом для численних обчислювальних завдань, де точність та швидкість грають ключову роль.

Далі наведено рисунок 1.1 поділеної області за допомогою тріангуляції, ілюструє ефективність цього методу у візуальному представленні геометричних структур. На згаданому рисунку видно, як площина розбита на трикутники так, що кожен з них визначений оптимально та ефективно та не містить зайвих точок в своїх вписаних кругах. Це наочно демонструє, як тріангуляція Делоне створює структуровану мережу, що може бути використана для аналізу та обробки географічних або геометричних даних.

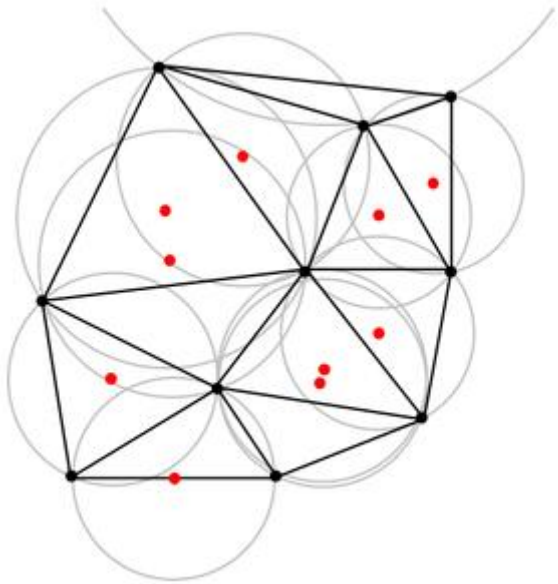

Рисунок 1.1 – Тріангуляція Делоне і описані навколо трикутників кола

1.3 Алгоритми тріангуляції Делоне

Існує багато алгоритмів тріангуляції Делоне, і їхні модифікації розвиваються в науковому та індустріальному середовищі. Деякі з них є класичними та добре вивченими, інші є об’єктом активних досліджень.

Деякі з відомих алгоритмів тріангуляції Делоне:

- інкрементна тріангуляція Делоне: включає алгоритми, такі як Bowyer-Watson та інші інкрементальні методи;
- локалізація та Фліппінг: алгоритми, що використовують локалізацію та операції фліппінга для побудови тріангуляції;
- алгоритм замітання прямою: методи, що використовують прямі для поетапного побудови тріангуляції.
- алгоритми «поділи та пануй»: алгоритми, що використовують стратегію "поділи та пануй" для побудови тріангуляції;
- використання структур даних, таких як DCEL (Doubly-Connected Edge List): алгоритми, що використовують спеціальні структури даних для зручного представлення та модифікації тріангуляції;
- алгоритми, що використовують призначені GPU: Розроблені для прискорення обчислень на графічних процесорах;
- рандомізовані алгоритми: алгоритми, які використовують випадковий елемент для покращення ефективності;
- апроксимаційні алгоритми: методи, які намагаються надати приблизний розв’язок з меншими обчислювальними витратами.

Далі описані деякі алгоритми тріангуляції Делоне.

1.3.1 Алгоритм інкрементальної тріангуляції Делоне

Одним з популярних алгоритмів для вирішення цього завдання є алгоритм інкрементальної тріангуляції Делоне "Incremental Delaunay Triangulation":

Для початку тріангуляції Делоне інкрементальним методом, алгоритм вибирає початковий трикутник, який включає три точки, не лежать на одній прямій. Цей початковий трикутник додається до списку трикутників.

Далі, при додаванні нової точки до масиву точок, алгоритм шукає трикутник, який містить цю точку. Цей пошук виконується інкрементально та швидко за O(log n), де n – кількість точок.

Після знаходження трикутника, що містить нову точку, відбувається фаза піддерева фліппінга (рис. 1.2). Алгоритм видаляє обраний трикутник та додає три нових трикутники, утворені новою точкою та ребрами видаленого трикутника. Забезпечується перевірка та виправлення властивостей Делоне за допомогою операції "фліпання".

Процес повторюється для кожної нової точки, що дозволяє алгоритму інкрементально та ефективно розширювати тріангуляцію. Зазначено, що цей алгоритм має часову складність O(n log n), де n – кількість точок.

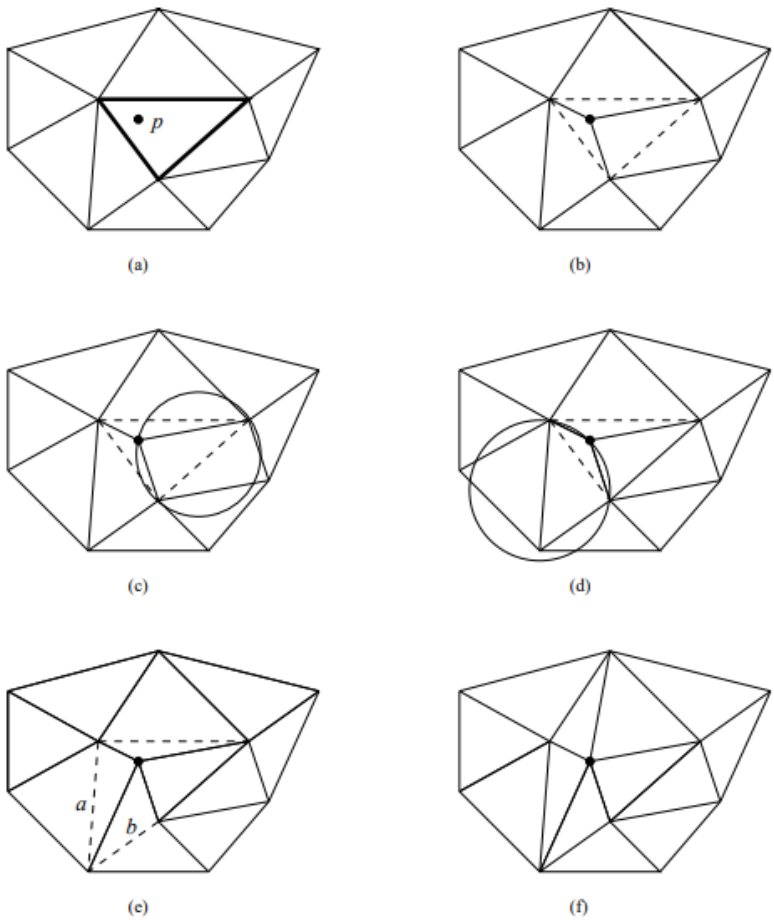

Рисунок 1.2 – Вставка точки в триангуляцию та перевірка умов

1.3.2 Алгоритм "Bowyer-Watson"

Для початку алгоритму "Bowyer-Watson" створюється великий трикутник, який обгортає всі точки, і цей трикутник додається до списку трикутників. Це дозволяє включити всі точки в область тріангуляції.

При додаванні нової точки алгоритм вставляє цю точку в масив точок і розпочинає фазу локалізації. Локалізація визначає трикутник, який містить нову точку. Це може відбуватися шляхом ітеративного переходу від поточного трикутника до сусіднього, який містить точку, або застосування швидших методів, таких як дерева та індексація.

Після визначення трикутника відбувається фаза фліппінга. Видаляються трикутники, які містять нову точку, та видаляються всі трикутники, які мають їхні ребра. Після цього додаються нові трикутники, утворені новою точкою та ребрами видалених трикутників.

Цей процес забезпечує, що властивості Делоне залишаються збереженими для всіх точок. Фліппінг використовується для корекції трикутників та підтримання умов тріангуляції Делоне.

Алгоритм "Bowyer-Watson" повторює цей процес для кожної нової точки, що дозволяє інкрементально побудувати тріангуляцію Делоне (рис. 1.3). Цей метод також має часову складність O(n log n), де n – кількість точок, що робить його ефективним для великої кількості точок.

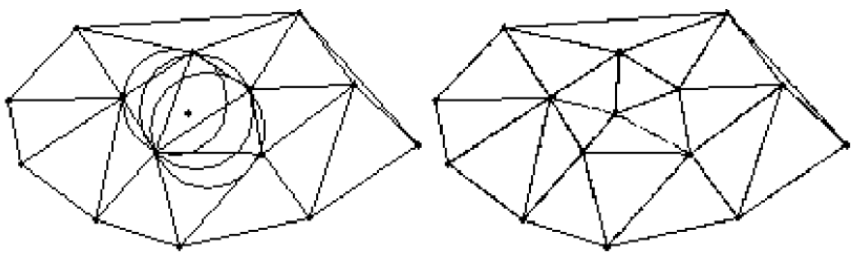

Рисунок 1.3 – Алгоритм "Bowyer-Watson"

1.3.3 Алгоритм замітання прямою

Алгоритм замітання прямої, застосований до тріангуляції Делоне, використовує уявну вертикальну пряму, що рухається площиною і зупиняється у точках, де проводяться геометричні обчислення (рис 1.4). Головна ідея полягає в тому, щоб обмежити геометричні операції об’єктами, які перетинаються або прилягають до цієї уявної прямої. Повний розв’язок досягається, коли пряма пройде через усі об’єкти.

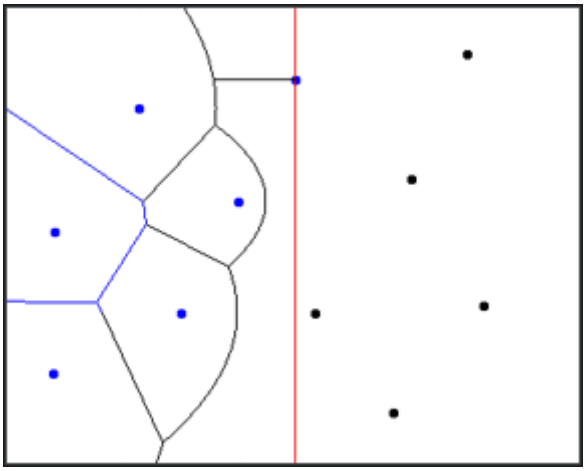

Рисунок 1.4 – Алгоритм замітання прямою

Цей підхід важливий для тріангуляції Делоне, де використання замітальної прямої дозволяє ефективно вирішувати проблеми перетину відрізків та обмежувати обчислення точок визначення трикутників. Зокрема, комбінування замітання прямої з ефективними структурами даних, такими як збалансовані двійкові дерева, дозволяє швидко виявляти перетини та будувати тріангуляцію Делоне зі складністю O(N log N), де N – кількість точок.

Такий підхід знаходить застосування в розв’язанні геометричних задач і грає ключову роль у побудові високоефективних алгоритмів для тріангуляції та інших операцій над геометричними об’єктами.

1.3.4 Алгоритм "поділи та пануй" для тріангуляції Делоне

Алгоритм "поділи та пануй" для тріангуляції Делоне (рис. 1.5) може бути реалізований наступним чином.

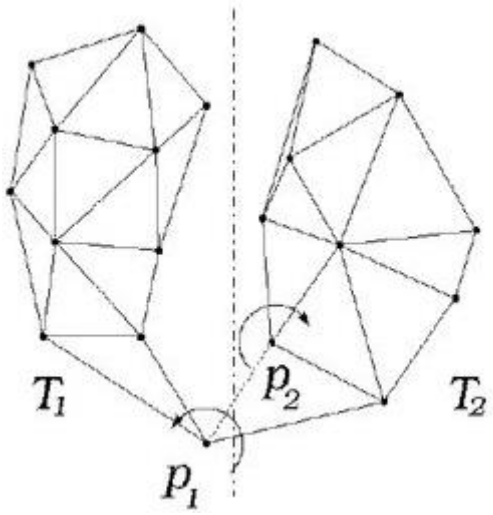

Рисунок 1.5 – Алгоритм "поділи та пануй"

Для початку потрібно розділіть вихідну множину точок на дві менші підмножини, наприклад, верхню і нижню половини. Це можна зробити, наприклад, розташовуючи точки вздовж горизонтальної лінії та вибираючи ті, що знаходяться вище і нижче цієї лінії.

Далі застосовується алгоритм тріангуляції Делоне рекурсивно до кожної з отриманих підмножин. Це може бути виконано, наприклад, викликом того ж алгоритму для кожної половини.

Після того, як обидві підмножини були тріангульовані, розглядається межа між ними. Будуються трикутники, які з’єднують точки з обох половин. Для цього визначаються межові точки, які є часткою тріангуляції для обох підмножин. Якщо в результаті об’єднання виникли трикутники, які не задовольняють умовам тріангуляції Делоне (наприклад, точки в околі їх описаного кола), вони видаляються із загального списку трикутників.

Алгоритм продовжує виконуватись рекурсивно, доки кожна підмножина не буде складена з декількох точок, що дозволить побудувати тріангуляцію всієї вихідної множини.

Цей підхід "поділи та пануй" дозволяє розбити задачу тріангуляції Делоне на менші частини, що полегшує реалізацію та зберігає властивість тріангуляції Делоне для кожного окремого підмножини.

1.4 Вхідні та вихідні дані тріангуляції Делоне

Вхідні дані для алгоритму тріангуляції Делоне можуть бути задані різними способами, що робить його універсальним для застосування в різних сценаріях. Програма, яка використовує цей алгоритм, може приймати координати точок від користувача через консоль або зчитувати їх з файлу.

Користувач може вводити координати точок вручну через консоль. Кожна точка представлена парою координат (x, y), де x – горизонтальна координата, y – вертикальна координата.

Координати точок також можуть бути зчитані з файлу. Формат файлу може бути текстовим або у вигляді таблиці, де кожний рядок містить координати однієї точки.

Після обробки вхідних даних програма буде виводити графічне представлення області, побудованої тріангуляцією Делоне, разом із координатами вершин трикутників та координатами крайових точок цієї області.

Такий вихідний формат надає користувачеві повний огляд результатів тріангуляції Делоне для заданих точок у плоскій області.

2 МЕТОДИ ТА ЗАСОБИ РОЗРОБКИ

2.1 Вибір мови програмування

Вибір мови програмування для реалізації завдань кваліфікаційної роботи – це важливе рішення, і C# може бути відмінним вибором залежно від конкретних вимог та завдань проекту.

C# є однією з основних мов програмування для розробки на платформі Microsoft. Visual Studio, офіційна інтегрована середовище розробки (IDE) для C#, надає широкий спектр інструментів та ресурсів, що полегшують розробку.

Мова програмування C# (вимовляється "C-Sharp") була розроблена компанією Microsoft і представлена в 2000 році як частина їхньої ініціативи під назвою ".NET Framework". Офіційно вона вперше була випущена разом з Visual Studio .NET у 2002 році.

Основний та головний розробник мови C# – це Андерс Хейлсберг (Anders Hejlsberg), який раніше був відомий своєю роботою над Turbo Pascal і Delphi. Історія C# пов’язана з розвитком платформи .NET, яка була розроблена для полегшення розробки програм на різних мовах та для вирішення проблеми "захоплення" (vendor lock-in), коли розробник був обмежений вибором платформи.

Основною метою створення C# та .NET було надання зручного середовища для розробки програм, яке б підтримувало об’єктно-орієнтоване програмування, було масштабованою та забезпечувало можливість легкої інтеграції з іншими технологіями Microsoft.

Важливим моментом була здатність C# працювати на різних платформах, що розширило його застосування. З часом мова отримала підтримку аспектів паралельного програмування, вбудовану безпеку та інші функції, що роблять її сучасною та потужною мовою програмування.

C# став не лише ключовою мовою для розробки програм на платформі .NET, але і знайшов широке застосування в різних сферах розробки програмного забезпечення, включаючи веб-розробку, мобільний розвиток, ігри та інші області програмування.

2.1.1 Основи мови програмування C# та об'єктно-орієнтованого програмування

C# (C-Sharp) – це об'єктно-орієнтована мова програмування, розроблена компанією Microsoft. Об'єктно-орієнтоване програмування (ООП) – це парадигма програмування, яка базується на концепціях об'єктів та класів, де об'єкти представляють екземпляри класів, а взаємодія між ними відбувається через обмін повідомленнями. Ось деякі основні риси C# як об'єктно-орієнтованої мови:

- класи та об'єкти: В C# основною одиницею програмування є клас, який визначає структуру та поведінку об'єкта. Використовуючи класи, можна групувати дані (поля) та функції (методи) в один об'єктно-орієнтований модуль. Об'єкти є екземплярами класів і представляють конкретні елементи або екземпляри в програмі.
- інкапсуляція: C# підтримує концепцію інкапсуляції, що дозволяє обмежити доступ до деяких компонентів класу та приховати деталі реалізації від користувача.
- спадкування: Використання спадкування дозволяє створювати нові класи на основі існуючих, успадковуючи їхні властивості та методи. Спадкування допомагає в реорганізації та розширенні коду.
- поліморфізм: Поліморфізм в C# може бути досягнутий через перевизначення методів у підкласах та використання віртуальних методів та абстрактних класів. Здатність об'єктів різних класів

використовувати однаковий інтерфейс спрощує код та робить його більш гнучким.

- поліформізм з використанням Інтерфейсів: Інтерфейси в C# визначають контракти, які клас повинен реалізувати. Це дозволяє використовувати поліморфізм через різні класи, які реалізують той самий інтерфейс.
- абстракція: Абстракція в C# дозволяє виділити основні аспекти об’єктів та ігнорувати деталі, які не є суттєвими для вирішення конкретного завдання.
- події та делегати: В C# введені події та делегати, що дозволяють реалізувати принципи спостереження та інших шаблонів проєктування.

Завдяки властивостям об’єктно-орієнтованого програмування, вбудованим в C#, ця мова надає програмістам потужний інструментарій для розв’язання різноманітних складних задач. Можливість створювати структуровані програми за допомогою класів та об’єктів дозволяє ефективно групувати функціонал та дані. Інкапсуляція допомагає управляти доступом до компонентів класу, забезпечуючи контроль конфіденційності та безпеки даних.

Спадкування дозволяє створювати нові класи на основі існуючих, спрощуючи розширення та модифікацію функціоналу. Поліморфізм робить код більш гнучким, забезпечуючи можливість використання об’єктів різних класів через їхній загальний інтерфейс.

Використання інтерфейсів дозволяє реалізовувати поліморфізм через різні класи, що реалізують спільні контракти. Абстракція дозволяє виокремлювати суттєві аспекти об’єктів, спрощуючи розробку та підтримку коду.

З використанням подій та делегатів, C# дозволяє реалізувати шаблони проєктування, такі як принципи спостереження, роблячи мову ефективним інструментом для розробки розширюваних та модульних програм.

2.2 Огляд інтегрованого середовища розробки (integrated development environment)

Для цього проєкту обрано Visual Studio як інтегроване середовище розробки (IDE). Visual Studio надає потужні інструменти для написання, відлагодження та управління кодом. За допомогою його редактора коду, візуальних дизайнерів і вбудованих функцій відлагодження можна ефективно розробляти програми, зосереджуючись на функціональності та якості коду. Також, Visual Studio інтегрується з іншими сервісами Microsoft, що полегшує роботу зі сховищами коду, системами керування версіями та іншими компонентами розробки.

Visual Studio – це інтегроване середовище розробки (IDE) від компанії Microsoft. Це програмне забезпечення призначене для розробки різноманітних типів програм, включаючи програми для операційних систем Windows, веб-додатки, мобільні додатки, ігри, а також інші застосунки.

Основні особливості Visual Studio включають:

- редактор коду: Має потужний редактор коду з різними функціями підсвічування синтаксису, автоматичного завершення коду та іншими інструментами для полегшення написання коду.
- візуальний дизайнер: Дозволяє вам візуально створювати інтерфейси користувача для Windows Forms, WPF (Windows Presentation Foundation), ASP.NET, та інших технологій.
- відлагодження: Надає інструменти для ефективного відлагодження коду, включаючи точки зупинки, відстеження змінних, перегляд стеку викликів та інші функції.
- підтримка мов програмування: Підтримує різні мови програмування, такі як C#, Visual Basic, C++, F#, Python, та інші.
- Інтеграція з платформами Microsoft: Взаємодіє з різними технологіями та сервісами Microsoft, такими як Azure, Git, Team Foundation Server (TFS), та інші.

- можливості для роботи в команді: Забезпечує інструменти для спільної роботи команд розробників, включаючи системи керування версіями.

Visual Studio використовується розробниками для розробки різноманітних програм і є однією з найпопулярніших IDE у світі програмування.

Також було би можливо використовувати інші інтегровані середовища розробки (IDE) або текстові редактори, такі як:

- Visual Studio Code: Це легкий та потужний текстовий редактор від Microsoft, який відмінно підходить для розробки на різних мовах програмування.
- JetBrains Rider: Це інтегроване середовище розробки, яке підтримує різні мови програмування, включаючи C#. Відоме своєю продуктивністю та інтелектуальними інструментами.
- Eclipse: Відкрите інтегроване середовище розробки, яке може використовуватися для роботи з різними мовами програмування за допомогою відповідних розширень.

2.3 Використання бібліотек у проєкті

Бібліотеки в програмуванні представляють собою набір підготовлених та передбачуваних функцій або об’єктів, які можуть бути використані в програмному коді для вирішення конкретних завдань. Ці функції і об’єкти розроблені заздалегідь та зберігаються у вигляді викликабельних модулів або пакетів, які можна використовувати в різних програмах.

Основні характеристики бібліотек включають:

- повторне використання коду: Бібліотеки мають на меті забезпечити механізм для збереження та повторного використання функціональності в різних частинах програми чи навіть в різних програмах.

- організація коду: Бібліотеки допомагають організовувати код, розділяючи його на логічні блоки, які можна використовувати незалежно.
- зменшення дублювання коду: Використання бібліотек дозволяє уникати повторення однакового коду в різних місцях програми чи між різними проектами.
- підтримка розробки: Багато бібліотек надають готові реалізації загальних завдань, таких як операції з обробкою зображень, взаємодія з базами даних, мережеві операції та інші, спрощуючи розробку.

Для розробки мого програмного додатку я використав низку потужних бібліотек, які надали необхідний функціонал для досягнення моїх цілей. Серед цих бібліотек використані такі, як System для загальних операцій, System.Collections.Generic для роботи з колекціями, System.Drawing для роботи з графікою, а також бібліотеки OpenTK та OpenTK.Graphics.OpenGL для графічного програмування та візуалізації. Використання цих бібліотек дозволило ефективно реалізувати необхідний функціонал, забезпечуючи високу якість та продуктивність програмного додатку.

2.3.1 Візуалізація з використанням бібліотеки OpenTK та OpenTK.Graphics.OpenGL

Для реалізації візуалізації фінальних областей після процесу тріангуляції Делоне у моєму проекті було використано бібліотеки OpenTK та OpenTK.Graphics.OpenGL. Ці бібліотеки забезпечили необхідний інструментарій для взаємодії з графічним контекстом та викликом функцій OpenGL, дозволяючи ефективно візуалізувати фінальну картинку областей після використання алгоритму тріангуляції Делоне.

OpenGL (Open Graphics Library) – це відкритий стандарт для програмного інтерфейсу графічного програмування, який надає доступ до функцій низькорівневого програмування графіки. Він розроблений для

створення високоефективних та переносимих програм графічної обробки, особливо у контексті 2D та 3D графіки.

OpenGL відзначається своєю переносимістю, що дозволяє розробникам створювати код, який працює на різних операційних системах, таких як Windows, Linux та macOS. Це забезпечує гнучкість у розробці та розповсюдженні графічних програм.

Стандарт також надає низькорівневий доступ до графічного обладнання, дозволяючи розробникам безпосередньо взаємодіяти з графічними можливостями системи.

OpenGL включає підтримку як 2D, так і 3D графіки, що дозволяє створювати різноманітні візуальні ефекти та анімації. Введення шейдерів дозволяє програмістам настроювати обробку графіки, що веде до створення складних візуальних сцен та ефектів.

Гнучкість конфігурацій OpenGL дозволяє налаштовувати параметри відображення, такі як кольорові простори та розрядність кольору, для досягнення відповідного візуального результату.

Загальна активність та спільнота розробників, а також наявність ресурсів, таких як документація та уроки, роблять OpenGL потужним інструментом для графічної розробки. Узагальнюючи, цей стандарт надає засоби для створення високоефективних та інноваційних графічних застосунків у різних середовищах.

2.3.2 Імпорт параметрів для алгоритму триангуляції Делоне зовнішнім файлом

Для імпорту параметрів з зовнішнього файла в мові програмування C# була використана бібліотека System.IO.File. Ця бібліотека входить до складу .NET Framework і надає засоби для взаємодії з файловою системою. Використання System.IO.File дозволяє ефективно здійснювати операції

зчитування та запису даних у файл, а також здійснювати перевірку наявності файлів та інші операції.

На початку програма визначає шлях до файлу, де збережені координати стартових точок. Вона використовує клас System.IO.File, який надає метод ReadAllText для зчитування всього вмісту файлу у вигляді рядка.

Далі рядок інтерпретується, і кожна точка розділяється за пробілами. Для кожної точки виконується подальший розбір координат. Отримані координати (X та Y) перетворюються в числовий формат, щоб бути використаними у програмі.

Кожна зчитана точка виводиться на екран для перевірки правильності зчитування, а програма також враховує можливість винятків та надає повідомлення про помилку у випадку невдалого зчитування файлу. Такий підхід забезпечує надійність операцій та полегшує відлагодження.

Отже, в даному коді використовується бібліотека System.IO.File для ефективного зчитування та обробки даних з зовнішнього файлу. Цей функціонал має особливе значення у реалізації триангуляції Делоне, де точність та коректність вхідних параметрів є важливою складовою успішності алгоритму.

3 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

3.1 Архітектура програми

Архітектура програми для тріангуляції Делоне включає в себе два основних класи: OpenGLWindow та DelaunayTriangulation, які спільно взаємодіють для відображення та обчислення тріангуляції. Програма використовує OpenTK та OpenGL для графічного представлення результатів.

Клас OpenGLWindow служить графічним інтерфейсом для відображення тріангуляції. У конструкторі він отримує розміри вікна, назву та масив точок для тріангуляції. Метод OnLoad викликається при завантаженні вікна та викликає функцію відображення. Графічна ініціалізація виконується за допомогою OpenTK та OpenGL.

Клас DelaunayTriangulation виконує алгоритм тріангуляції Делоне. Його конструктор приймає масив точок та ініціалізує необхідні параметри та структури даних для алгоритму. Основні методи цього класу включають в себе логіку тріангуляції, оновлення границі (hull), а також виведення інформації про трикутники, напівребра та границю через метод PrintTriangulationInfo.

Метод Main використовується як точка входу в програму. У цьому методі генеруються або отримуються точки для тріангуляції в залежності від вибраного джерела даних: файлу, консолі чи генерації випадковим чином. Наприклад, метод може викликати спеціальний внутрішній метод для читання точок з файлу або з консолі, або ж генерувати їх випадковим чином. Після отримання точок створюється екземпляр класу DelaunayTriangulation, викликається метод для виведення інформації про тріангуляцію, та ініціалізується вікно OpenTK (OpenGLWindow) для відображення результатів тріангуляції.

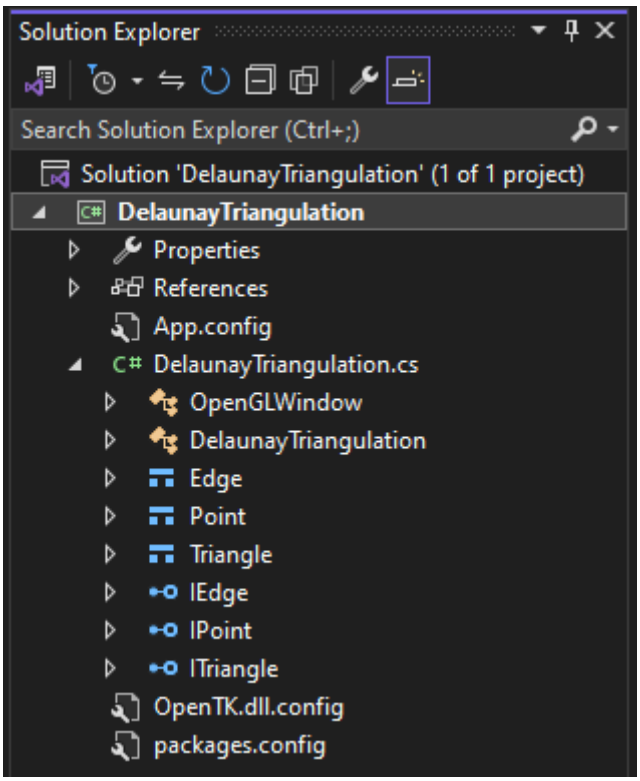

Рисунок 3.1 – Класи, структури і інтерфейси

Загальна архітектура програми використовує об’єктно-орієнтований підхід, розділення відповідальностей та використання сторонніх бібліотек для графічного відображення. Додатково, є можливість покращення коду через розподіл функціоналу на більші невеликі та самодостатні частини для полегшення розуміння та обслуговування.

3.2 Методи та функції, використані для реалізації тріангуляції Делоне

У програмному додатку реалізовано обчислення тріангуляції Делоне за допомогою алгоритму інкрементальної тріангуляції.

Алгоритм інкрементальної тріангуляції визначається високою ефективністю у часі, що робить його особливо відмінним вибором для реалізації в програмах на C#. Алгоритм може бути зручно реалізований на мові

програмування C# через його об'єктно-орієнтовану природу та високу рівень абстракції, що полегшує розробку та обслуговування коду.

Розглянемо ключові методи та функції, які використовуються для логіки створення триангуляції методом Делоне в програмному коді. Кожен з елементів цієї логіки відіграє важливу роль у визначенні та корекції трикутників, щоб вони відповідали умовам Делоне. Нижче подано короткий опис основних методів та функцій цього розділу:

– Legalize(int a)

Цей метод використовується для перевірки та виправлення трикутників, які не відповідають умовам Делоне. Він використовує стек замість рекурсії для оптимізації виконання.

– InCircle(...)

Метод, який визначає, чи знаходиться точка (px, py) всередині описаного кола, утвореного трикутником (ax, ay), (bx, by), (cx, cy).

– AddTriangle(...)

Метод для додавання нового трикутника до масиву трикутників та встановлення зв’язків між ним та суміжніми трикутниками.

– Link(...)

Метод для встановлення зв’язків між півребрами трикутника.

– HashKey(...)

Функція для визначення ключа для хешування, що використовується для швидкого доступу до зовнішнього трикутника за допомогою хеш-таблиці.

– PseudoAngle(...)

Функція для обчислення псевдокута, який використовується для порівняння кутів.

– Quicksort(...)

Реалізація алгоритму швидкого сортування для сортування точок за відстанню.

– Swap(...)

Метод для обміну елементів в масиві.

– Orient(...)

Функція для визначення орієнтації точок.

– Circumradius(...)

Обчислення радіусу описаного кола для трикутника.

– Circumcenter(...)

Обчислення центру описаного кола для трикутника.

– Dist(...)

Обчислення квадрату відстані між двома точками.

Ці методи та функції взаємодіють для ефективного створення тріангуляції методом Делоне, використовуючи оптимізовані алгоритми та структури даних.

3.3 Інтерфейси та структури: організація даних для моделювання тріангуляції методом Делоне

Розглянемо інтерфейси та структури, які використовуються для моделювання тріангуляції методом. Вони визначають основні об’єкти, які використовуються у логіці алгоритму та забезпечують зручний інтерфейс для роботи з отриманими результатами.

– структура Edge

Edge представляє собою ребро тріангуляції та має такі характеристики:
P та Q: Кінці ребра, представлені об’єктами типу IPoint.
Index: Індекс ребра.

– структура Point

Point визначає точку в двовимірному просторі з координатами X та Y.

– структура Triangle

Triangle представляє трикутник у тріангуляції та має такі характеристики:
Points: Колекція точок, що утворюють трикутник.
Index: Індекс трикутника.

– інтерфейс IEdge

IEdge визначає загальний інтерфейс для роботи з ребрами. Містить властивості P та Q, представлені об’єктами типу IPoint, а також індекс ребра.

– інтерфейс IPoint

IPoint визначає загальний інтерфейс для роботи з точками в двовимірному просторі за допомогою властивостей X та Y.

– інтерфейс ITriangle

ITriangle визначає загальний інтерфейс для трикутників. Містить властивість Points, яка є колекцією точок, та індекс трикутника.

Використання інтерфейсів та структур в даному контексті дозволяє забезпечити чітку структуру даних та полегшити подальшу роботу з результатами алгоритму. Застосування інтерфейсів спрощує роботу з об’єктами, надаючи загальний інтерфейс для їхнього використання. Структури ж забезпечують ефективне представлення об’єктів з точки зору пам’яті та швидкодії.

3.4 Візуалізація триангуляції в графічному інтерфейсі з використанням OpenTK OpenGL

Робота з точками і вершинами є ключовим етапом для створення та візуалізації триангуляції Делоне. Основні концепції, що використовуються для цієї мети, включають в себе роботу з вершинами, координатами точок, та визначення графічного контексту для подальшого відображення.

У бібліотеці OpenTK OpenGL точки та вершини представлені об’єктами, які зберігають інформацію про координати точок у тривимірному просторі. Координати цих точок використовуються для побудови графічних об’єктів, таких як трикутники, які утворюють триангуляцію. Важливою є можливість визначити порядок точок для кожного трикутника та їх зв’язок, що визначає

топологію тріангуляції. На рисунку 3.2 показана як по точкам будуються деякі види фігур.

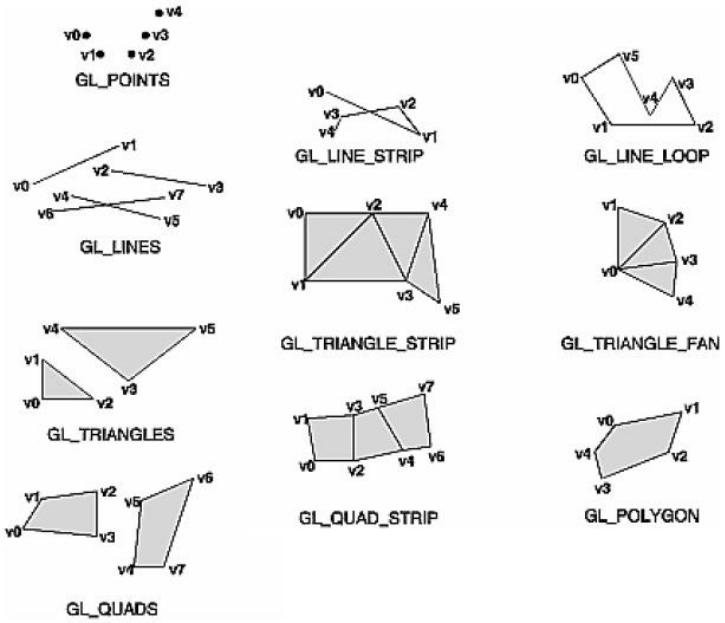

Рисунок 3.2 – Приклади фігур побудованих за точками

Після визначення точок і вершин, важливим етапом є побудова графічного контексту за допомогою бібліотеки OpenTK OpenGL. Графічний контекст визначає параметри відображення, освітлення, та інші властивості, які впливають на зовнішній вигляд тріангуляції на екрані.

Сам процес візуалізації включає в себе подання точок і вершин у формі графічних об’єктів, таких як трикутники, і їх рендеринг на екрані. OpenTK OpenGL надає можливості для визначення розмірів та положення об’єктів, налаштування кольорів та текстур, щоб створити візуально зручний зображення тріангуляції Делоне.

В бібліотеці OpenTK для реалізації OpenGL-додатків використовуються різні методи та події, які визначають основні моменти життєвого циклу програми та взаємодію з графічним контекстом. Декілька ключових методів та подій цього циклу включають:

OnLoad метод:

- цей метод викликається при завантаженні графічного контексту та ініціалізації OpenGL.
- використовується для встановлення початкових параметрів, таких як кольори, матриці проекції, робота з текстурами тощо.

OnUpdateFrame метод:

- цей метод викликається перед кожним кадром відображення.
- використовується для оновлення параметрів та обчислень, які впливають на стан сцени.

OnRenderFrame метод:

- цей метод викликається при кожному кадрі для малювання зображення.
- використовується для визначення логіки малювання та відображення об’єктів на екрані.

OnResize метод:

- цей метод викликається при зміні розмірів вікна.
- використовується для оновлення параметрів відображення при зміні розмірів вікна.

Ці методи і події є ключовими для реалізації графічних додатків з використанням OpenTK та OpenGL. Вони надають можливість взаємодії з графічним контекстом у різних моментах життєвого циклу програми.

3.4.1 Реалізація візуалізації тріангуляції методом Делоне

В програмі клас OpenGLWindow використовує бібліотеку OpenTK для створення вікна та обробки подій OpenGL. Деякі основні функції, такі як OnLoad та OnRenderFrame, є ключовими для ініціалізації та відображення графіки у вікні.

Клас OpenGLWindow в програмі відповідає за візуалізацію області тріангуляції методом Делоне. Після завершення тріангуляції та обчислення координат точок вершин трикутників він створює OpenGL-вікно для відображення результатів.

При завантаженні вікна викликається метод OnLoad. У цьому методі встановлюються параметри OpenGL, такі як колір фону, матриці проекції та модельного виду. Далі викликається метод GL.Clear, який очищує буфери кольорів та глибини. Потім викликається метод GL.Begin для початку малювання трикутників.

Далі йде код для малювання трикутників у методі OnLoad. Використовується цикл для ітерації по всім трикутникам, а потім викликається метод GL.Vertex2 для кожної точки трикутника, вказуючи координати та кольори. Для відображення точок у вікні використовується метод ConvertToPoint, який конвертує внутрішні координати та центрує їх на екрані.

4 РОБОТА КОРИСТУВАЧА З ПРОГРАМОЮ

4.1 Системні вимоги та інсталяція

Для успішної інсталяції та запуску розробленого шахового додатка необхідно дотриматися наступних системних вимог та інструкцій:

- наявність встановленого C++ компілятора, наприклад, Visual Studio (версії 2017 або вище) з підтримкою C++ 17
- бібліотека OpenTK. Її можна завантажити у проект кількома способами:
  - використання NuGet пакету. Найпростіший спосіб – використовувати менеджер пакетів NuGet у вашому середовищі розробки (наприклад, Visual Studio). Ви можете встановити OpenTK, шляхом виклику команди: «Install-Package OpenTK»
  - вибрати "Manage NuGet Packages" у вашому проекті та знайти OpenTK (рис 4.1).
  - ручне завантаження бібліотек. Ви можете завантажити бібліотеку з офіційного сайту OpenTK, а потім додати їх у ваш проект вручну. https://opentk.net/index.html (рис 4.2).
- бібліотека OpenTK.GLControls. Для її завантаження потрібно перейти до меню "Tools" > "NuGet Package Manager" > "Manage NuGet Packages for Solution". Знайти пакет OpenTK.GLControl у списку та вибрати проект, в який потрібно встановити пакет.

Також можно завантажити через косоль NuGet: Відкрийте консоль NuGet за допомогою команди View > Other Windows > Package Manager Console. Введіть команду: Install-Package OpenTK.GLControl та натисніть "Enter" (рис 4.3).

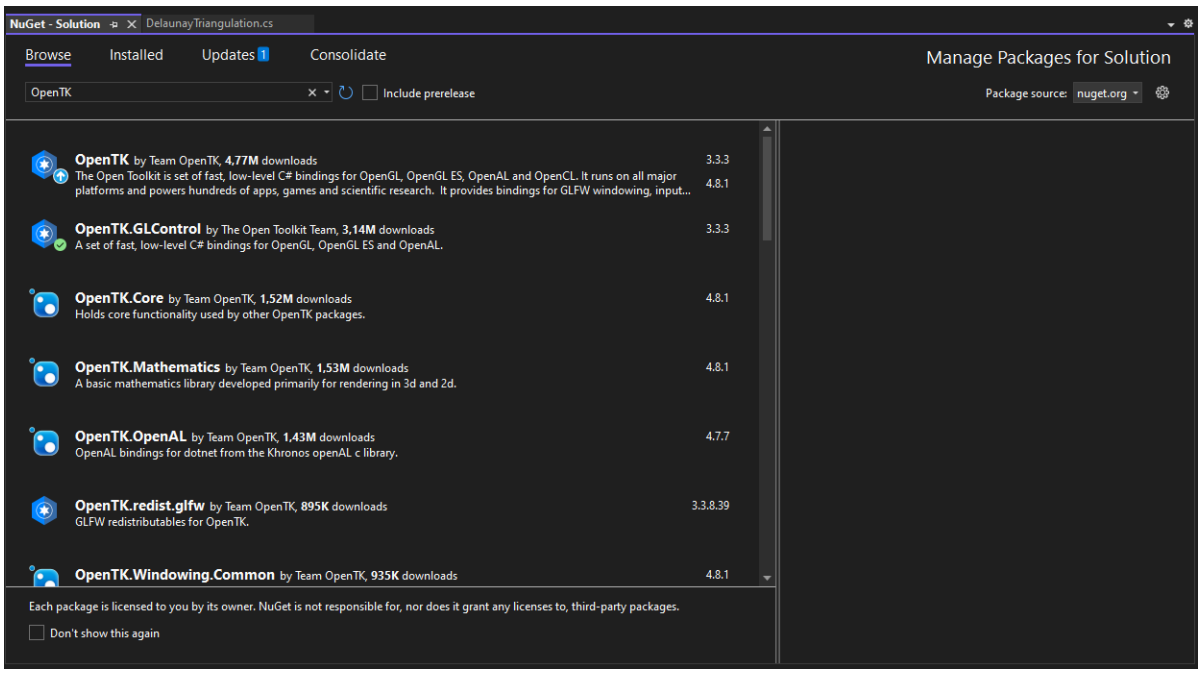

Рисунок 4.1 – Менеджер пакетів NuGet

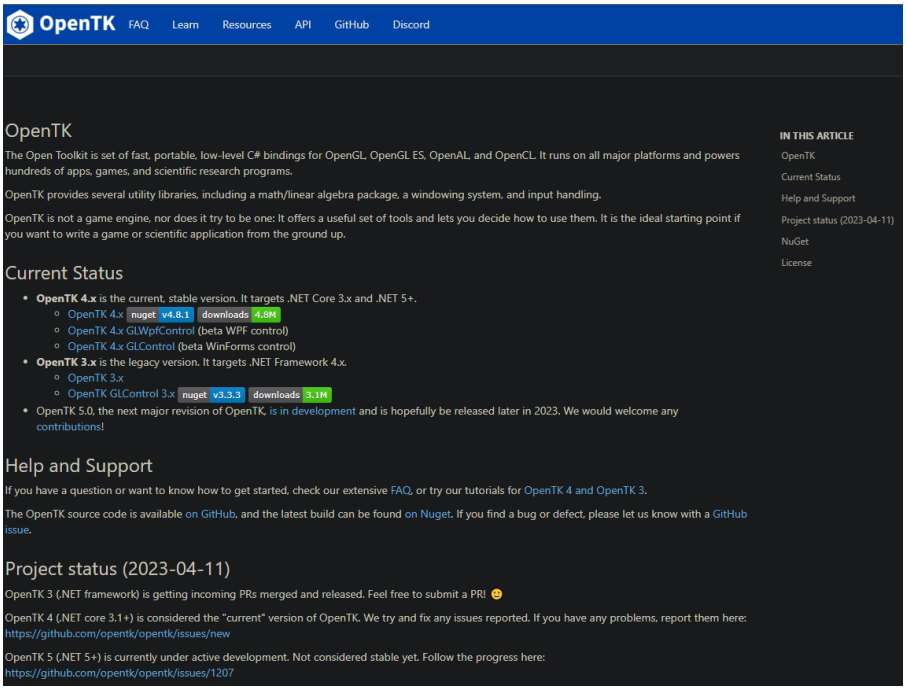

Рисунок 4.2 – Вид офіційного сайту OpenTK

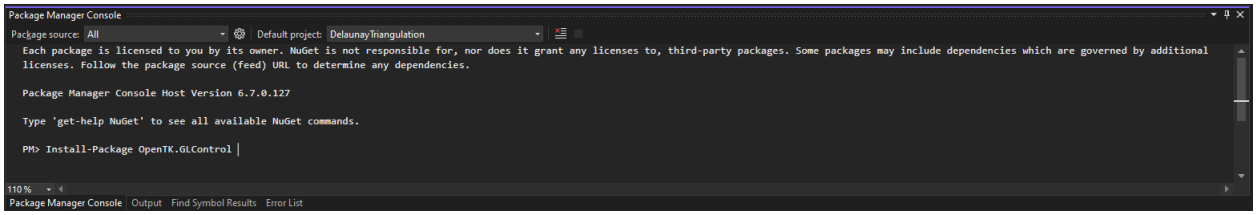

Рисунок 4.3 – Консоль менеджера пакетів

4.2 Опис можливостей програми для користувача

Для того, щоб запустити консольний додаток на C#, виконайте наступні кроки:

- відкрийте проект в Visual Studio: Відкрийте у Visual Studio файл із кодом.
- впевніться в налаштуваннях проекту: Переконайтеся, що у вас встановлено конфігурацію "Debug" (Відлагодження) або "Release" (Реліз), залежно від вашого наміру.
- компіляція проекту: Натискайте "Ctrl + Shift + B" або виберіть "Build" (Збірка) > "Build Solution" (Зібрати рішення), щоб скомпілювати ваш проект.
- запуск програми: Після компіляції перейдіть до каталогу вашого проекту у файловому провіднику. У папці bin\Debug (або bin\Release залежно від конфігурації) знаходитиметься виконуючий файл з розширенням .exe. Це і є ваш виконуючий файл. Подвійний клік на цьому файлі або виклик його з командного рядка запустить ваш консольний додаток.

  Після запуску програми відкриється консольне вікно (рис 4.4).

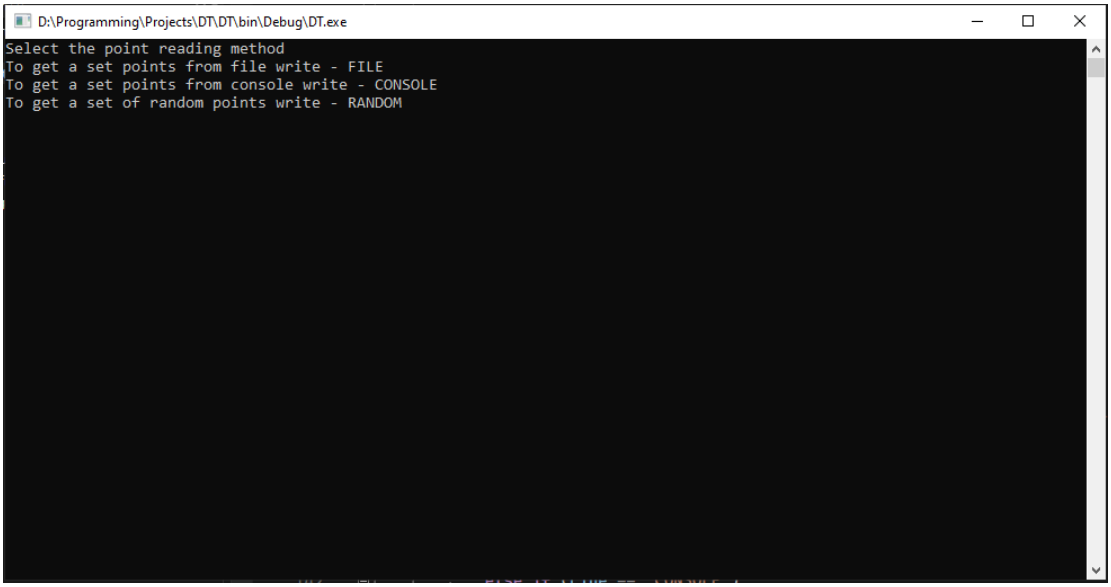

Рисунок 4.4 – Консоль програмного додатка

Після запуску, додаток запропонує вибрати один з трьох способів зчитування точок:

- вибрати точки з файлу: FILE
- вести точки через консоль: CONSOLE
- генерувати випадкові точки: RANDOM

Якщо користувач вибере “FILE”, програма прочитає точки з файлу "points.txt" у поточному каталозі. При записі точок у файл їх координати можна представити у вигляді пар (X, Y), розділених пробілом (рис 4.5). Кожна пара координат записується у новому рядку. Такий формат можна розглядати як таблицю, де перший стовпець містить координати X, а другий – координати Y.

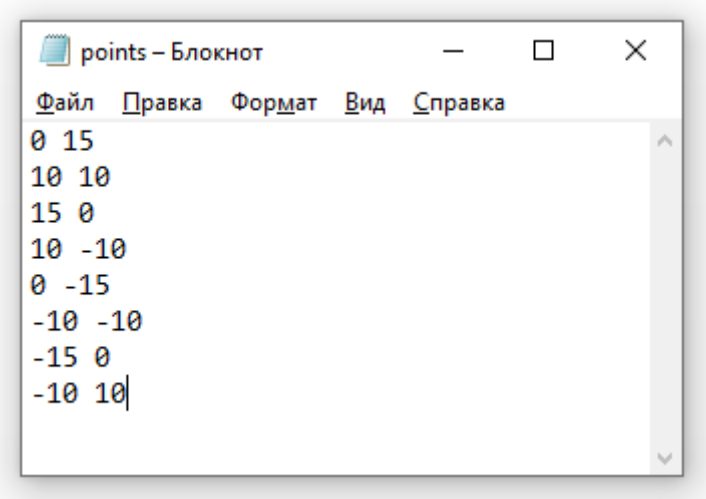

Рисунок 4.5 – Формат запису точок у текстовий файлі

Використання простого розділення пробілом полегшує читання та обробку цих даних у програмі. Такий підхід дозволяє ефективно використовувати ці дані для подальших обчислень та візуалізації.

Якщо обрано CONSOLE, користувачу потрібно спочатку ввести кількість точок і потім їх координати через консоль. Вводити координати через консоль потрібно у форматі "X Y", де X та Y – це числові значення координат точки і після кожної пари координат потрібно натискати ENTER

(рис 4.6). Якщо користувач запише координати неправильно, програма попросить ввести їх повторно.

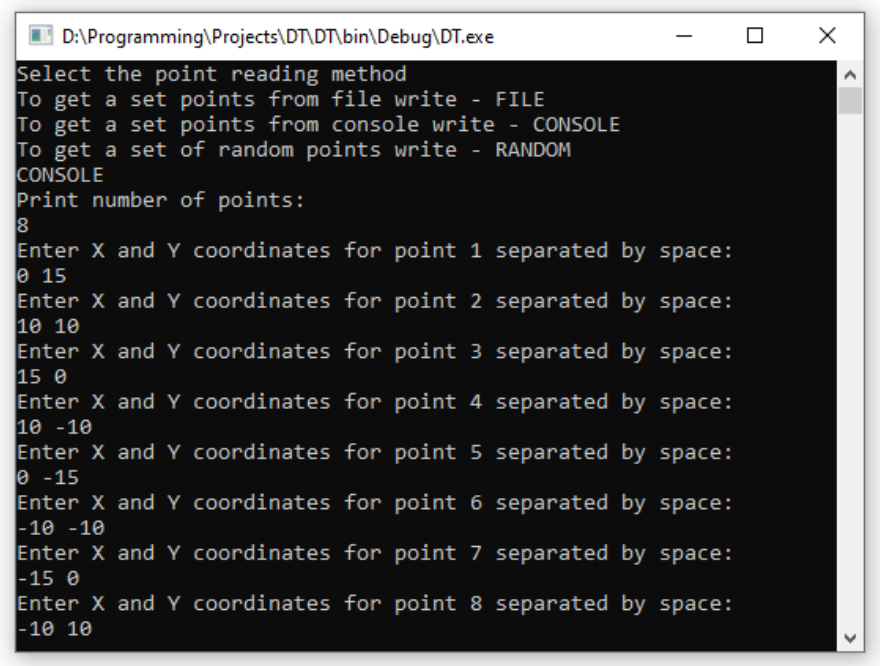

Рисунок 4.6 – Запис координат точок через консоль

Якщо користувач обирає метод введення випадкових точок RANDOM, то він повинен вказати кількість точок, які програма має випадковим чином створити.

4.3 Приклади роботи програмного додатку

Далі наведено кілька прикладів роботи програмного додатку:

Приклад 1

Початкові дані для триангуляції складаються з координат восьми точок, які утворюють восьмикутник на одній плоскості – (0;15),(10;10),(15;0),(10;−10),(0;−15),(−10;−10),(−15;0),(−10;10).

Користувач вибирає метод введення даних через консоль. Програма повинна обробити введені дані, виконати триангуляцію методом Делоне та

відобразити результат у вікні з рисунком, де плоскість розбита на трикутники (рис 4.7).

На малюнку ви можно побачити область, яка була розбита на трикутники за допомогою триангуляції Делоне. Кожен трикутник виділено окремим кольором, що дозволяє вам легко визначити межі кожного з них. Програма також виводить інформацію у консоль, де для кожного трикутника вказані три номери точок, які формують цей трикутник. Програмний додаток також виводить у консоль інформацію, які точки лежать на межах області, проте в даному випадку варто відзначити, що всі надані точки розташовані на межах області.

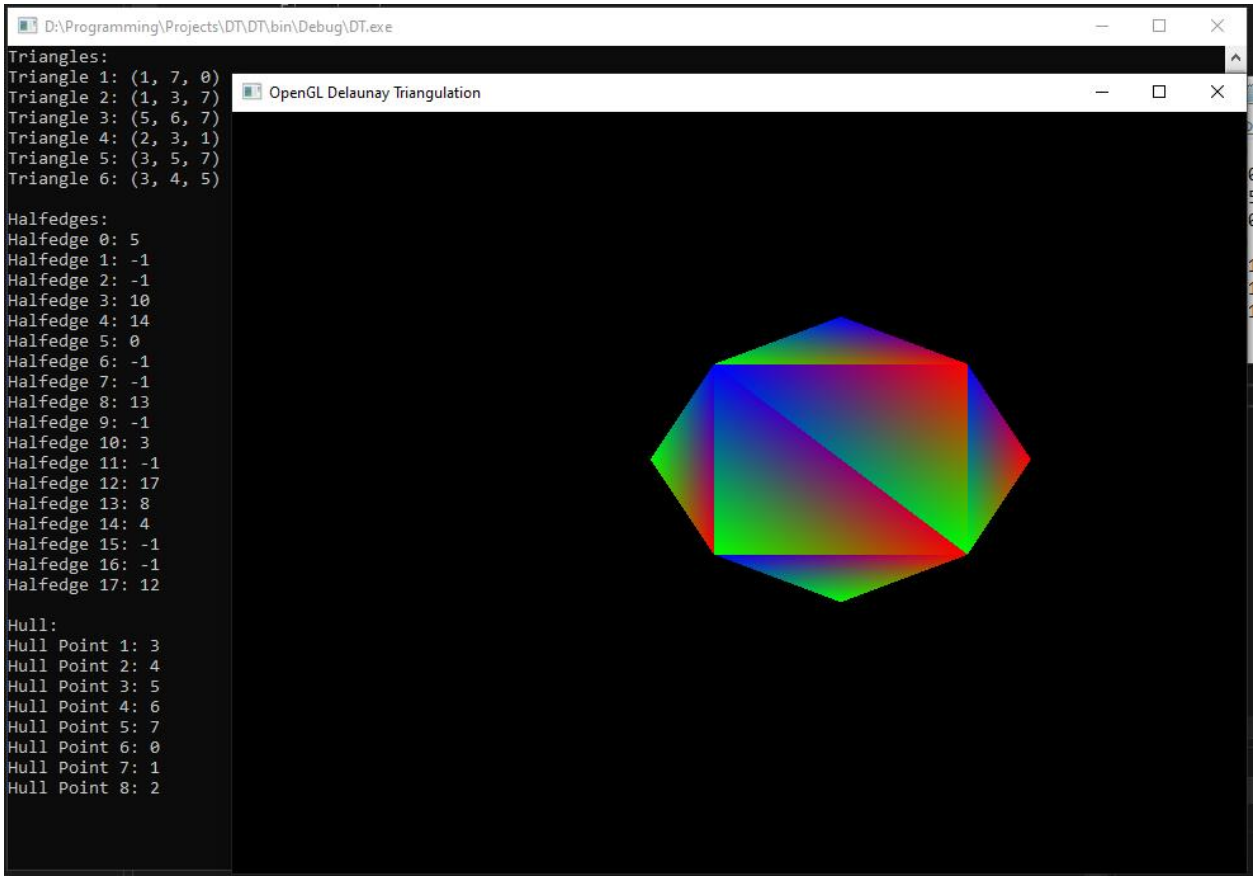

Рисунок 4.7 – Приклад роботи програмного додатку 1

Приклад 2

Якщо користувачу необхідно побудувати область, використовуючи велику кількість точок, оптимальним рішенням буде скористатися введенням

координат точок через текстовий файл. У цьому випадку всі дані будуть записані у файл у форматі пар координат, які розділені пробілом. Після цього користувач може запустити програму та обрати метод отримання точок з файлу, вказавши "FILE".

У файл записано координати п’ятдесяти точок:
(-16;21), (0;-2), (-23;-10), (-8;10), (-3;-11), (4;1), (24;-5), (-8;23), (15;3), (-19;-21), (-10;-18), (-6;-5), (-24;-19), (9;9), (-2;19), (17;15), (-25;8), (1;-14), (-3;25), (20;-20), (-6;10), (7;23), (-8;-21), (-17;-13), (-5;12), (-3;-6), (18;3), (-20;-14), (21;-12), (22;-11), (-8;18), (-21;8), (12;2), (4;-24), (7;-2), (-24;-24), (11;15), (2;7), (-12;-7), (25;-17), (15;16), (-19;-25), (1;-20), (6;4), (0;12), (22;-7), (9;18), (10;4), (-3;6).

Після обчислень із заданими точками програма сформувала область, яку можна побачити на рисунку 4.8, розбивши її на трикутники за допомогою триангуляції Делоне.

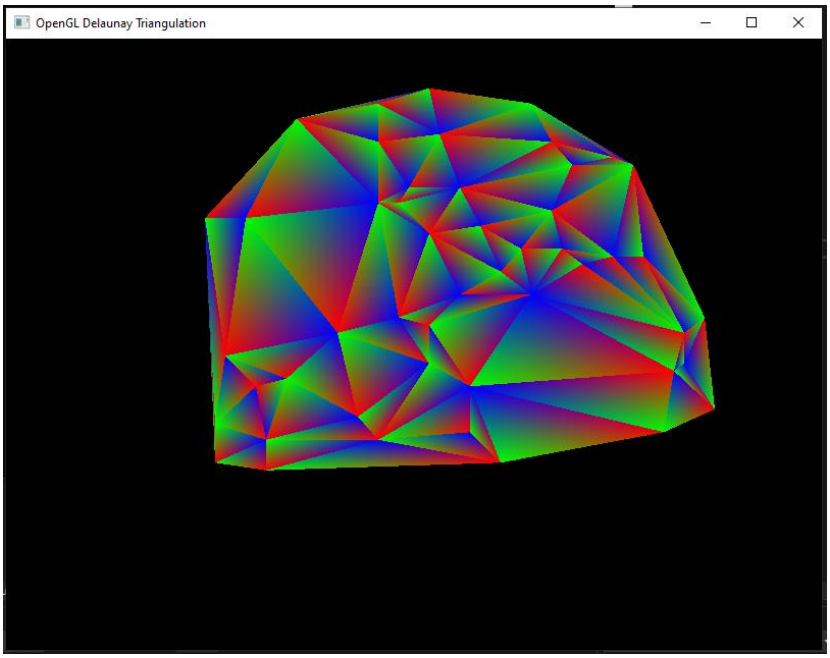

Рисунок 4.8 – Приклад роботи програмного додатку 2

Після обчислень програма у консолі вивела номери точок, які складають 85 трикутників, утворюючих тріангуляцію Делоне у заданій області. Загальна кількість точок, які були задані на початку, становила 50. З цих 50 точок 11

розташовані на межі області, їх номери та координати були також виведені у консоль.

ВИСНОВКИ

У ході дослідження було звернуто увагу на ключові аспекти, пов’язані з побудовою дискретного представлення плоских областей за допомогою триангуляції Делоне.

Було ретельно розглянуто та визначено основні поняття триангуляції та плоских областей, що лежать в основі подальших досліджень.

Вивчення особливостей триангуляції Делоне сприяло кращому розумінню технічних відмінностей та переваг цього методу. Розглянуті різноманітні алгоритми, такі як інкрементальна тріангуляція, "Bowyer-Watson", алгоритм замітання прямою, а також "поділи та пануй", дозволили визначити оптимальні методи для конкретних ситуацій.

Розроблений програмний додаток на мові програмування C# з використанням бібліотек OpenTK та OpenGL є важливим доповненням до теоретичного дослідження. Він дозволяє візуалізувати та перевіряти результати застосування триангуляції Делоне в практичних задачах.

У своїй роботі також було надано основи мови програмування C# та об’єктно-орієнтованого програмування, визначаючи структуру та організацію розробленого програмного додатка. Зокрема, було висвітлено переваги інтегрованого середовища розробки, яке сприяло зручній навігації, відладці та взаємодії з ресурсами мови програмування C#. Окремо було визначено імпорт параметрів для алгоритму триангуляції Делоне зовнішнім файлом. Це розширило можливості програми, надаючи більш зручну можливість для введення вхідних даних.

Загалом, отримані результати свідчать про важливість та актуальність вивчення триангуляції Делоне для побудови дискретних представлень плоских областей. Розглянуті алгоритми та програмний додаток створюють основу для подальших досліджень та застосувань у галузі комп’ютерної графіки та суміжних областях.

ПЕРЕЛІК ПОСИЛАНЬ

1. Cignoni P., Montani C., Scopigno R. DeWall: A fast divide and conquer Delaunay triangulation algorithm in E d. Elsevier, 1998. P. 333-341.

2. Guibas, L.J., Knuth, D.E. & Sharir, M. Randomized incremental construction of Delaunay and Voronoi diagrams. Algorithmica 7, (1992). P. 381-413.

3. Lischinski D. Incremental delaunay triangulation. Cornell University, 1993. 13 p.

4. Börgers C. Generalized delaunay triangulations of non-convex domains. Elsevier, 1990. P. 45-49.

5. Troelsen A., Japikse P. Pro C# 7 With .NET and .NET Core (Eight Edition). Apress, 2017. 1410 c.

6. Schildt H. C# 4.0 The Complete Reference. McGraw Hill, 2010. 976 p.

7. Microsoft Visual Studio. URL : https://visualstudio.microsoft.com/ (дата звернення : 12.10.2023).

8. OpenTK – open toolkit. URL : https://opentk.net/ (дата звернення : 16.10.2023).

9. OpenGL official site. URL : https://www.opengl.org/ (дата звернення : 16.10.2023).

10. C# documentation. URL : https://learn.microsoft.com/en-us/dotnet /csharp/ (дата звернення : 25.09.2023).

ДОДАТОК А

Текст програми

Class OpenGLWindow:

```
public class OpenGLWindow : GameWindow
{
    private DelaunayTriangulation delaunayTriangulator;

    public OpenGLWindow(int width, int height, IPoint[] points)
        : base(width, height, GraphicsMode.Default, "OpenGL Delaunay Triangulation")
    {
        delaunayTriangulator = new DelaunayTriangulation(points);
        VSync = VSyncMode.On;
    }

    protected override void OnLoad(EventArgs e)
    {
        base.OnLoad(e);

        GL.Color3(0.0f, 0.0f, 0.0f);

        GL.MatrixMode(MatrixMode.Projection);
        GL.LoadIdentity();
        GL.Ortho(-2000, 2000, -2000, 2000, -1, 1);

        GL.MatrixMode(MatrixMode.Modelview);
        GL.LoadIdentity();



        GL.Clear(ClearBufferMask.ColorBufferBit | ClearBufferMask.DepthBufferBit);

        GL.Begin(PrimitiveType.Triangles);

        for (int i = 0; i < delaunayTriangulator.Triangles.Length; i += 3)
        {
            int p0 = delaunayTriangulator.Triangles[i];
            int p1 = delaunayTriangulator.Triangles[i + 1];
            int p2 = delaunayTriangulator.Triangles[i + 2];

            Point point0 = ConvertToPoint(delaunayTriangulator.Points[p0]);
            Point point1 = ConvertToPoint(delaunayTriangulator.Points[p1]);
            Point point2 = ConvertToPoint(delaunayTriangulator.Points[p2]);

            Random rand = new Random();

            GL.Color3(1.0f, 0.0f, 0.0f);
            GL.Vertex2(point0.X, point0.Y);
```

```
            GL.Color3(0.0f, 1.0f, 0.0f);
            GL.Vertex2(point1.X, point1.Y);

            GL.Color3(0.0f, 0.0f, 1.0f);
            GL.Vertex2(point2.X, point2.Y);
        }

        GL.End();

        SwapBuffers();
    }

    protected override void OnRenderFrame(FrameEventArgs e)
    {
        base.OnRenderFrame(e);
    }

    private Point ConvertToPoint(IPoint point)
    {
        float x = (float)((point.X – delaunayTriangulator.cx) * 50 + Width / 2);
        float y = (float)((point.Y – delaunayTriangulator.cy) * 50 + Height / 2);
        return new Point(x, y);
    }
}
```

Class DelaunayTriangulation:

```
public class DelaunayTriangulation
{
    private readonly double EPSILON = Math.Pow(2, -52);
    private readonly int[] EDGE_STACK = new int[512];
    public int[] Triangles { get; private set; }
    public int[] Halfedges { get; private set; }
    public IPoint[] Points { get; private set; }
    public int[] Hull { get; private set; }

    private readonly int hashSize;
    private readonly int[] hullPrev;
    private readonly int[] hullNext;
    private readonly int[] hullTri;
    private readonly int[] hullHash;

    public double cx;
    public double cy;

    private int trianglesLen;
    private readonly double[] coords;
    private readonly int hullStart;
    private readonly int hullSize;

    static void Main()
    {
        string currentDirectory = AppDomain.CurrentDomain.BaseDirectory;
```

```
        string filePath = Path.Combine(currentDirectory, "points.txt");

        int numberOfPoints = 25;
        IPoint[] pointsArray = new IPoint[numberOfPoints];
        float xR = 50.0f;
        float yR = 50.0f;


        Console.WriteLine("Select the point reading method");
        Console.WriteLine("To get a set points from file write – FILE");
        Console.WriteLine("To get a set points from console write – CONSOLE");
        Console.WriteLine("To get a set of random points write – RANDOM");

        String Line = Console.ReadLine();
        if (Line == "FILE")
        {
            pointsArray = ReadPointsFromFile(filePath);
        }
        else if (Line == "CONSOLE")
        {
            Console.WriteLine("Print number of points: ");
            numberOfPoints = int.Parse(Console.ReadLine());
            pointsArray = new IPoint[numberOfPoints];

            for (int i = 0; i < numberOfPoints; i++)
            {
                Console.WriteLine($"Enter X and Y coordinates for point {i + 1} separated by space: ");
                string[] input = Console.ReadLine().Split();

                if (input.Length == 2 && double.TryParse(input[0], out double x) &&
double.TryParse(input[1], out double y))
                {
                    pointsArray[i] = new Point { X = x, Y = y };
                }
                else
                {
                    Console.WriteLine("Invalid input. Please enter two valid coordinates.");
                    i--;
                }
            }
        }
        else if (Line == "RANDOM")
        {
            Console.WriteLine("Print number of points: ");
            numberOfPoints = int.Parse(Console.ReadLine());
            pointsArray = GetRandomPointsInRange(numberOfPoints, xR, yR);
        }

        DelaunayTriangulation delaunator = new DelaunayTriangulation(pointsArray);
        delaunator.PrintTriangulationInfo();

        using (var window = new OpenGLWindow(800, 600, pointsArray))
        {
            window.Run(60.0);
        }

    }
```

```
    private static IPoint[] ReadPointsFromFile(string filePath)
    {
        List<IPoint> pointsList = new List<IPoint>();

        try
        {
            using (StreamReader reader = new StreamReader(filePath))
            {
                string line;
                while ((line = reader.ReadLine()) != null)
                {
                    string[] coordinates = line.Split(' ');
                    if (coordinates.Length == 2 && double.TryParse(coordinates[0], out double x) &&
double.TryParse(coordinates[1], out double y))
                    {
                        pointsList.Add(new Point { X = x, Y = y });
                    }
                    else
                    {
                        Console.WriteLine($"Skipping invalid line: {line}");
                    }
                }
            }
        }
        catch (IOException ex)
        {
            Console.WriteLine($"Error reading file: {ex.Message}");
        }

        return pointsList.ToArray();
    }
    private static IPoint[] GetRandomPointsInRange(int numOfPoints, float xRange, float yRange)
    {
        Random random = new Random();
        IPoint[] pointsArray = new IPoint[numOfPoints];
        for (int i = 0; i < numOfPoints; i++)
        {
            double x = random.NextDouble() * xRange – xRange / 2;
            double y = random.NextDouble() * yRange – yRange / 2;

            pointsArray[i] = new Point { X = x, Y = y };
        }

        return pointsArray;
    }
    public void PrintTriangulationInfo()
    {
        Console.WriteLine("Triangles:");
        for (int i = 0; i < trianglesLen; i += 3)
        {
            int p0 = Triangles[i];
            int p1 = Triangles[i + 1];
            int p2 = Triangles[i + 2];
            Console.WriteLine($"Triangle {i / 3 + 1}: ({p0}, {p1}, {p2})");
        }

        Console.WriteLine("\nHalfedges:");
```

```
    for (int i = 0; i < trianglesLen; i++)
    {
        int edge = Halfedges[i];
        Console.WriteLine($"Halfedge {i}: {edge}");
    }

    Console.WriteLine("\nHull:");
    for (int i = 0; i < hullSize; i++)
    {
        int hullPoint = Hull[i];
        Console.WriteLine($"Hull Point {i + 1}: {hullPoint}");
    }
}
public DelaunayTriangulation(IPoint[] points)
{
    if (points.Length < 3)
    {
        throw new ArgumentOutOfRangeException("Need at least 3 points");
    }

    Points = points;
    coords = new double[Points.Length * 2];

    for (var i = 0; i < Points.Length; i++)
    {
        var p = Points[i];
        coords[2 * i] = p.X;
        coords[2 * i + 1] = p.Y;
    }

    var n = points.Length;
    var maxTriangles = 2 * n – 5;

    Triangles = new int[maxTriangles * 3];

    Halfedges = new int[maxTriangles * 3];
    hashSize = (int)Math.Ceiling(Math.Sqrt(n));

    hullPrev = new int[n];
    hullNext = new int[n];
    hullTri = new int[n];
    hullHash = new int[hashSize];

    var ids = new int[n];

    var minX = double.PositiveInfinity;
    var minY = double.PositiveInfinity;
    var maxX = double.NegativeInfinity;
    var maxY = double.NegativeInfinity;

    for (var i = 0; i < n; i++)
    {
        var x = coords[2 * i];
        var y = coords[2 * i + 1];
        if (x < minX) minX = x;
        if (y < minY) minY = y;
        if (x > maxX) maxX = x;
```

```
    if (y > maxY) maxY = y;
    ids[i] = i;
}

var cx = (minX + maxX) / 2;
var cy = (minY + maxY) / 2;

var minDist = double.PositiveInfinity;
var i0 = 0;
var i1 = 0;
var i2 = 0;

// pick a seed point close to the center
for (int i = 0; i < n; i++)
{
    var d = Dist(cx, cy, coords[2 * i], coords[2 * i + 1]);
    if (d < minDist)
    {
        i0 = i;
        minDist = d;
    }
}
var i0x = coords[2 * i0];
var i0y = coords[2 * i0 + 1];

minDist = double.PositiveInfinity;

// find the point closest to the seed
for (int i = 0; i < n; i++)
{
    if (i == i0) continue;
    var d = Dist(i0x, i0y, coords[2 * i], coords[2 * i + 1]);
    if (d < minDist && d > 0)
    {
        i1 = i;
        minDist = d;
    }
}

var i1x = coords[2 * i1];
var i1y = coords[2 * i1 + 1];

var minRadius = double.PositiveInfinity;

// find the third point which forms the smallest circumcircle with the first two
for (int i = 0; i < n; i++)
{
    if (i == i0 || i == i1) continue;
    var r = Circumradius(i0x, i0y, i1x, i1y, coords[2 * i], coords[2 * i + 1]);
    if (r < minRadius)
    {
        i2 = i;
        minRadius = r;
    }
}
var i2x = coords[2 * i2];
var i2y = coords[2 * i2 + 1];
```

```
if (minRadius == double.PositiveInfinity)
{
   throw new Exception("No Delaunay triangulation exists for this input.");
}

if (Orient(i0x, i0y, i1x, i1y, i2x, i2y))
{
   var i = i1;
   var x = i1x;
   var y = i1y;
   i1 = i2;
   i1x = i2x;
   i1y = i2y;
   i2 = i;
   i2x = x;
   i2y = y;
}

var center = Circumcenter(i0x, i0y, i1x, i1y, i2x, i2y);
this.cx = center.X;
this.cy = center.Y;

var dists = new double[n];
for (var i = 0; i < n; i++)
{
   dists[i] = Dist(coords[2 * i], coords[2 * i + 1], center.X, center.Y);
}

// sort the points by distance from the seed triangle circumcenter
Quicksort(ids, dists, 0, n – 1);

// set up the seed triangle as the starting hull
hullStart = i0;
hullSize = 3;

hullNext[i0] = hullPrev[i2] = i1;
hullNext[i1] = hullPrev[i0] = i2;
hullNext[i2] = hullPrev[i1] = i0;

hullTri[i0] = 0;
hullTri[i1] = 1;
hullTri[i2] = 2;

hullHash[HashKey(i0x, i0y)] = i0;
hullHash[HashKey(i1x, i1y)] = i1;
hullHash[HashKey(i2x, i2y)] = i2;

trianglesLen = 0;
AddTriangle(i0, i1, i2, -1, -1, -1);

double xp = 0;
double yp = 0;

for (var k = 0; k < ids.Length; k++)
{
   var i = ids[k];
```

```
var x = coords[2 * i];
var y = coords[2 * i + 1];

// skip near-duplicate points
if (k > 0 && Math.Abs(x – xp) <= EPSILON && Math.Abs(y – yp) <= EPSILON) continue;
xp = x;
yp = y;

// skip seed triangle points
if (i == i0 || i == i1 || i == i2) continue;

// find a visible edge on the convex hull using edge hash
var start = 0;
for (var j = 0; j < hashSize; j++)
{
   var key = HashKey(x, y);
   start = hullHash[(key + j) % hashSize];
   if (start != -1 && start != hullNext[start]) break;
}


start = hullPrev[start];
var e = start;
var q = hullNext[e];

while (!Orient(x, y, coords[2 * e], coords[2 * e + 1], coords[2 * q], coords[2 * q + 1]))
{
   e = q;
   if (e == start)
   {
      e = int.MaxValue;
      break;
   }

   q = hullNext[e];
}

if (e == int.MaxValue) continue;

// add the first triangle from the point
var t = AddTriangle(e, i, hullNext[e], -1, -1, hullTri[e]);

// recursively flip triangles from the point until they satisfy the Delaunay condition
hullTri[i] = Legalize(t + 2);
hullTri[e] = t;
hullSize++;

// walk forward through the hull, adding more triangles and flipping recursively
var next = hullNext[e];
q = hullNext[next];

while (Orient(x, y, coords[2 * next], coords[2 * next + 1], coords[2 * q], coords[2 * q + 1]))
{
   t = AddTriangle(next, i, q, hullTri[i], -1, hullTri[next]);
   hullTri[i] = Legalize(t + 2);
   hullNext[next] = next;
   hullSize--;
```

```
            next = q;

            q = hullNext[next];
        }

        // walk backward from the other side, adding more triangles and flipping
        if (e == start)
        {
            q = hullPrev[e];

            while (Orient(x, y, coords[2 * q], coords[2 * q + 1], coords[2 * e], coords[2 * e + 1]))
            {
                t = AddTriangle(q, i, e, -1, hullTri[e], hullTri[q]);
                Legalize(t + 2);
                hullTri[q] = t;
                hullNext[e] = e;
                hullSize--;
                e = q;

                q = hullPrev[e];
            }
        }

        // update the hull indices
        hullStart = hullPrev[i] = e;
        hullNext[e] = hullPrev[next] = i;
        hullNext[i] = next;

        // save the two new edges in the hash table
        hullHash[HashKey(x, y)] = i;
        hullHash[HashKey(coords[2 * e], coords[2 * e + 1])] = e;
    }

    Hull = new int[hullSize];
    var s = hullStart;
    for (var i = 0; i < hullSize; i++)
    {
        Hull[i] = s;
        s = hullNext[s];
    }

    hullPrev = hullNext = hullTri = null;

    Triangles = Triangles.Take(trianglesLen).ToArray();
    Halfedges = Halfedges.Take(trianglesLen).ToArray();
}

#region CreationLogic
private int Legalize(int a)
{
    var i = 0;
    int ar;

    while (true)
    {
        var b = Halfedges[a];
```

```
int a0 = a – a % 3;
ar = a0 + (a + 2) % 3;

if (b == -1)
{
   if (i == 0) break;
   a = EDGE_STACK[--i];
   continue;
}

var b0 = b – b % 3;
var al = a0 + (a + 1) % 3;
var bl = b0 + (b + 2) % 3;

var p0 = Triangles[ar];
var pr = Triangles[a];
var pl = Triangles[al];
var p1 = Triangles[bl];

var illegal = InCircle(
   coords[2 * p0], coords[2 * p0 + 1],
   coords[2 * pr], coords[2 * pr + 1],
   coords[2 * pl], coords[2 * pl + 1],
   coords[2 * p1], coords[2 * p1 + 1]);

if (illegal)
{
   Triangles[a] = p1;
   Triangles[b] = p0;

   var hbl = Halfedges[bl];

   if (hbl == -1)
   {
      var e = hullStart;
      do
      {
         if (hullTri[e] == bl)
         {
            hullTri[e] = a;
            break;
         }
         e = hullPrev[e];
      } while (e != hullStart);
   }
   Link(a, hbl);
   Link(b, Halfedges[ar]);
   Link(ar, bl);

   var br = b0 + (b + 1) % 3;

   if (i < EDGE_STACK.Length)
   {
      EDGE_STACK[i++] = br;
   }
}
else
```

```
            {
                if (i == 0) break;
                a = EDGE_STACK[--i];
            }
        }

        return ar;
    }
    private static bool InCircle(double ax, double ay, double bx, double by, double cx, double cy, double
px, double py)
    {
        var dx = ax – px;
        var dy = ay – py;
        var ex = bx – px;
        var ey = by – py;
        var fx = cx – px;
        var fy = cy – py;

        var ap = dx * dx + dy * dy;
        var bp = ex * ex + ey * ey;
        var cp = fx * fx + fy * fy;

        return dx * (ey * cp – bp * fy) -
               dy * (ex * cp – bp * fx) +
               ap * (ex * fy – ey * fx) < 0;
    }
    private int AddTriangle(int i0, int i1, int i2, int a, int b, int c)
    {
        var t = trianglesLen;

        Triangles[t] = i0;
        Triangles[t + 1] = i1;
        Triangles[t + 2] = i2;

        Link(t, a);
        Link(t + 1, b);
        Link(t + 2, c);

        trianglesLen += 3;
        return t;
    }
    private void Link(int a, int b)
    {
        Halfedges[a] = b;
        if (b != -1) Halfedges[b] = a;
    }
    private int HashKey(double x, double y) => (int)(Math.Floor(PseudoAngle(x – cx, y – cy) *
hashSize) % hashSize);
    private static double PseudoAngle(double dx, double dy)
    {
        var p = dx / (Math.Abs(dx) + Math.Abs(dy));
        return (dy > 0 ? 3 – p : 1 + p) / 4;
    }
    private static void Quicksort(int[] ids, double[] dists, int left, int right)
    {
        if (right – left <= 20)
        {
```

```
            for (var i = left + 1; i <= right; i++)
            {
                var temp = ids[i];
                var tempDist = dists[temp];
                var j = i – 1;
                while (j >= left && dists[ids[j]] > tempDist) ids[j + 1] = ids[j--];
                ids[j + 1] = temp;
            }
        }
        else
        {
            var median = (left + right) >> 1;
            var i = left + 1;
            var j = right;
            Swap(ids, median, i);
            if (dists[ids[left]] > dists[ids[right]]) Swap(ids, left, right);
            if (dists[ids[i]] > dists[ids[right]]) Swap(ids, i, right);
            if (dists[ids[left]] > dists[ids[i]]) Swap(ids, left, i);

            var temp = ids[i];
            var tempDist = dists[temp];
            while (true)
            {
                do i++; while (dists[ids[i]] < tempDist);
                do j--; while (dists[ids[j]] > tempDist);
                if (j < i) break;
                Swap(ids, i, j);
            }
            ids[left + 1] = ids[j];
            ids[j] = temp;

            if (right – i + 1 >= j – left)
            {
                Quicksort(ids, dists, i, right);
                Quicksort(ids, dists, left, j – 1);
            }
            else
            {
                Quicksort(ids, dists, left, j – 1);
                Quicksort(ids, dists, i, right);
            }
        }
    }
    private static void Swap(int[] arr, int i, int j)
    {
        var tmp = arr[i];
        arr[i] = arr[j];
        arr[j] = tmp;
    }
    private static bool Orient(double px, double py, double qx, double qy, double rx, double ry) => (qy –
py) * (rx – qx) – (qx – px) * (ry – qy) < 0;
    private static double Circumradius(double ax, double ay, double bx, double by, double cx, double
cy)
    {
        var dx = bx – ax;
        var dy = by – ay;
        var ex = cx – ax;
```

```
    var ey = cy – ay;
    var bl = dx * dx + dy * dy;
    var cl = ex * ex + ey * ey;
    var d = 0.5 / (dx * ey – dy * ex);
    var x = (ey * bl – dy * cl) * d;
    var y = (dx * cl – ex * bl) * d;
    return x * x + y * y;
}
private static Point Circumcenter(double ax, double ay, double bx, double by, double cx, double cy)
{
    var dx = bx – ax;
    var dy = by – ay;
    var ex = cx – ax;
    var ey = cy – ay;
    var bl = dx * dx + dy * dy;
    var cl = ex * ex + ey * ey;
    var d = 0.5 / (dx * ey – dy * ex);
    var x = ax + (ey * bl – dy * cl) * d;
    var y = ay + (dx * cl – ex * bl) * d;

    return new Point(x, y);
}
private static double Dist(double ax, double ay, double bx, double by)
{
    var dx = ax – bx;
    var dy = ay – by;
    return dx * dx + dy * dy;
}
#endregion CreationLogic
```