МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра фундаментальної та прикладної математики

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

на тему: **«АЛГОРИТМИ ТРІАНГУЛЯЦІЇ ДЕЛОНЕ ТА ЇХ ПРОГРАМНА РЕАЛІЗАЦІЯ»**

Виконав: студент 2 курсу, групи 8.1133

спеціальності 113 Прикладна математика
(шифр і назва спеціальності)

освітньої програми Прикладна математика
(назва освітньої програми)

О.В. Старовойтов
(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 завідувач кафедри програмної інженерії, доцент, к.ф.-м.н. Лісняк А.О.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2024

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)

« ____ » _____ 2024 р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Старовойтову Олександру Вікторовичу
(прізвище, ім'я та по-батькові)

1. Тема роботи (проекту) Алгоритми тріангуляції Делоне та їх програмна реалізація

керівник роботи (проекту) Гребенюк Сергій Миколайович, професор, д.т.н.
(прізвище, ім'я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 01 » березня 2024 року № 385-с

2. Строк подання студентом роботи 20.11.2024

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

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Огляд основних алгоритмів дискретизації
2. Алгоритм 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 7.03.2024

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	01.05.2024	
2.	Збір вихідних даних.	03.06.2024	
3.	Обробка методичних та теоретичних	05.07.2024	
	джерел.		
4.	Розробка першого розділу.	02.09.2024	
5.	Розробка другого розділу.	07.10.2024	
6.	Оформлення та нормоконтроль	20.11.2024	
	кваліфікаційної роботи.		
7.	Захист кваліфікаційної роботи.	10.12.2024	

Студент _____ О.В. Старовойтов
(підпис) (ініціали та прізвище)

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

Нормоконтроль пройдено

Нормоконтролер _____ О.Г. Спиця

РЕФЕРАТ

Кваліфікаційна робота магістра «Алгоритми тріангуляції Делоне та їх програмна реалізація»: 58 с., 22 рис., 1 табл., 9 джерел, 1 додаток.

АВТОМАТИЗАЦІЯ, АЛГОРИТМ, ГЕОМЕТРИЧНЕ МОДЕЛЮВАННЯ, ОПТИМІЗАЦІЯ, ТРІАНГУЛЯЦІЯ ДЕЛОНЕ.

Об’єкт дослідження: тріангуляція Делоне для плоских областей.

Мета роботи: розробка методики побудови тріангуляції Делоне та створення програмного забезпечення для автоматизації цього процесу, включно з перевіркою та оптимізацією отриманих структур.

Метод дослідження – аналітичний.

У роботі розглядається задача побудови тріангуляції Делоне для плоских областей як ефективного методу геометричного моделювання. Описано основні вимоги до тріангуляції, зокрема виконання умови порожнього описаного кола, та алгоритми, які використовуються для її побудови. Розглядаються обмеження й особливості процесу, зокрема корекція трикутників, частково розташованих у межах області, та оптимізація геометричної сітки.

На основі проведеного аналізу в роботі запропоновано методику побудови тріангуляції Делоне, яка враховує створення початкової сітки, проєкцію точок трикутників на межу області, перевірку відповідності умовам Делоне та оптимізацію структури.

На основі запропонованої методики у роботі розроблено програмне забезпечення, яке дозволяє автоматизувати процес виконання побудови тріангуляції, здійснювати перевірку її властивостей, оптимізувати структуру та виводити результати на екрані у вигляді графічних візуалізацій для подальшого аналізу.

SUMMARY

Master’s Qualifying Thesis "Delaunay Triangulation Algorithms and Their Software Implementation”: 58 pages, 22 figures, 1 table, 9 references, 1 supplement.

AUTOMATION, ALGORITHM, GEOMETRIC MODELING, OPTIMIZATION, DELAUNAY TRIANGULATION.

The object of the study: Delaunay triangulation for planar regions.

The aim of the study is a development of a methodology for constructing Delaunay triangulation and creating software to automate this process, including verification and optimization of the resulting structures.

The method of research is analytical.

The study addresses the problem of constructing Delaunay triangulation for planar regions as an efficient method of geometric modeling. It describes the main requirements for triangulation, including adherence to the empty circumcircle condition, and the algorithms used for its construction. The limitations and specific features of the process are examined, including the correction of triangles partially located within the region and the optimization of the geometric grid.

Based on the conducted analysis, the work proposes a methodology for constructing Delaunay triangulation that accounts for the creation of the initial mesh, projection of triangle vertices onto the region boundary, verification of compliance with Delaunay conditions, and structural optimization.

Using the proposed methodology, the work develops software that automates the triangulation process, verifies its properties, optimizes the structure, and displays results as graphical visualizations for further analysis.

ЗМІСТ

Завдання на кваліфікаційну роботу ……. 2

Реферат…… 4

Summary……..… 5

Вступ…… 8

1 Огляд та класифікація методів геометричного моделювання та дискретизації тіл складної форми …… 9

1.1 Поняття суцільного тіла……..…….. 9

1.2 Методи геометричного моделювання складних тіл…… 10

1.3 Інженерні креслення ……..……... 11

1.4 Каркасне представлення…….. 11

1.5 Плоскогранне представлення…….. 12

1.6 Макетування параметризованих об’єктів…….. 13

1.7 Межове представлення…… 13

1.8 Конструктивна блочна геометрія……... 14

1.9 Витягування, рух, обертання…….. 16

1.10 Параметричне представлення…… 17

1.11 Схеми на основі декомпозиції……... 17

1.11.1 Декомпозиція на основі об’єкта…….. 18

1.11.2 Декомпозиція на основі простору……..… 18

1.12 Представлення на основі серединних осей……... 19

1.13 Функціональне представлення……... 19

1.14 Методи побудови дискретного представлення складних об’єктів чотирикутними та шестигранними елементами……. 20

1.15 Структуровані та блочно-структуровані сітки……. 22

1.16 Фронтальні методи…….. 22

1.17 Просторовий вигін континуума……. 23

1.18 Методи суперпозиції……... 24

2 Тріангуляція плоских областей та алгоритми делоне як об’єкт дослідження ……... 26
2.1 Характеристика тріангуляції плоских областей та використовуваних алгоритмів Делоне……..…. 27
2.2 Особливості тріангуляції Делоне……... 28
2.2.1 Єдиність тріангуляції Делоне……...…. 28
2.2.2 Мінімізація вузьких кутів……..…… 28
2.2.3 Оптимальність для моделювання……..…... 29
2.2.4 Зв’язок із опуклою оболонкою……..…… 29
2.3 Алгоритми побудови тріангуляції Делоне…… 30
2.3.1 Алгоритм Brute Force ("Груба сила")…… 30
2.3.2 Алгоритм Flip ("Переворот ребер")……... 30
2.3.3 Інкрементальний алгоритм……. 31
2.4 Порівняння алгоритмів…… 32
3 Автоматизація процесу побудови тріангуляції Делоне…….. 34
3.1 Огляд існуючих програмних засобів та інструментів для автоматизації побудови тріангуляції……. 34
3.2 Опис розробленої програмної системи та її можливостей для автоматизації побудови тріангуляції Делоне…… 35
3.3 Приклад практичного використання програмного забезпечення….. 38
Висновки……. 46
Перелік посилань……..……..…… 48
Додаток А Лістинг програмного коду…….…….. 49

ВСТУП

Сучасний науковий світ невпинно розширює свої горизонти, що ставить перед дослідниками нові виклики та завдання, зокрема в галузі обчислювальної геометрії та геометричного моделювання. Однією з важливих проблем у цій галузі є ефективна дискретизація та побудова сіткових структур, зокрема тріангуляцій, для складних геометричних форм. Тріангуляція Делоне є одним із ключових методів для розбиття плоских областей на трикутники з максимальним мінімальним кутом, що забезпечує високу якість сітки та її стійкість до деформацій.

Алгоритми тріангуляції Делоне знайшли широке застосування у різних наукових та інженерних галузях, зокрема в комп'ютерній графіці, обробці зображень, геометричному моделюванні та в чисельних методах при розв'язанні задач для диференціальних рівнянь. Важливість тріангуляції Делоне обумовлена її здатністю створювати оптимальні сіткові структури, що значно підвищує точність обчислень та ефективність алгоритмів, які використовуються для моделювання складних геометричних об'єктів та областей.

Метою даної роботи є розгляд алгоритмів тріангуляції Делоне та створення їх програмної реалізації для практичного використання у задачах дискретизації. Основними завданнями роботи є аналіз основних методів побудови тріангуляції, розробка ефективного алгоритму з урахуванням обмежень і особливостей геометричної моделі, а також тестування створеної програмної реалізації на різних наборах вхідних даних.

Структурно робота складається з трьох розділів. У першому розділі розглядаються та класифікуються методи геометричного моделювання та дискретизації тіл складної форми. У другому розділі розглянуто тріангуляцію плоских областей та алгоритми Делоне як об'єкт дослідження. Третій розділ присвячено програмній реалізації обраного алгоритму та його тестуванню на прикладах для перевірки ефективності та точності роботи.

1 ОГЛЯД ТА КЛАСИФІКАЦІЯ МЕТОДІВ ГЕОМЕТРИЧНОГО МОДЕЛЮВАННЯ ТА ДИСКРЕТИЗАЦІЇ ТІЛ СКЛАДНОЇ ФОРМИ

З метою розкриття основних характеристик алгоритмів тріангуляції Делоне та їх застосування у задачах геометричного моделювання, у даному розділі надаються основні поняття та визначення теорії обчислювальної геометрії, які в подальшому використовуватимуться при розробці програмної реалізації обраних алгоритмів. Наводиться огляд основних методів геометричного моделювання та дискретизації тіл складної форми

1.1 Поняття суцільного тіла

Для моделювання об’єктів, які мають ненульовий об’єм, використовуються так звані суцільні тіла (solid) або просто тіла.

Суцільними називають ті об’єкти, що можуть бути представлені у вигляді підмножини тривимірного Евклідового простору E^3, яка відповідає наступним критеріям:

- їхня структура та форма залишаються незмінними незалежно від розташування у просторі (рис. 1.1а);
- межа об’єкта не має ізольованих або "висячих" елементів, тобто є однорідною (рис. 1.1, б);
- об’єкт займає обмежену частину простору, що характеризується скінченними площею чи об’ємом;
- при виконанні булевих операцій результатом є нові суцільні тіла, що забезпечує їх замкненість у цьому контексті;
- опис таких тіл має бути скінченним;
- межі об’єкта чітко визначені та не допускають неоднозначностей.

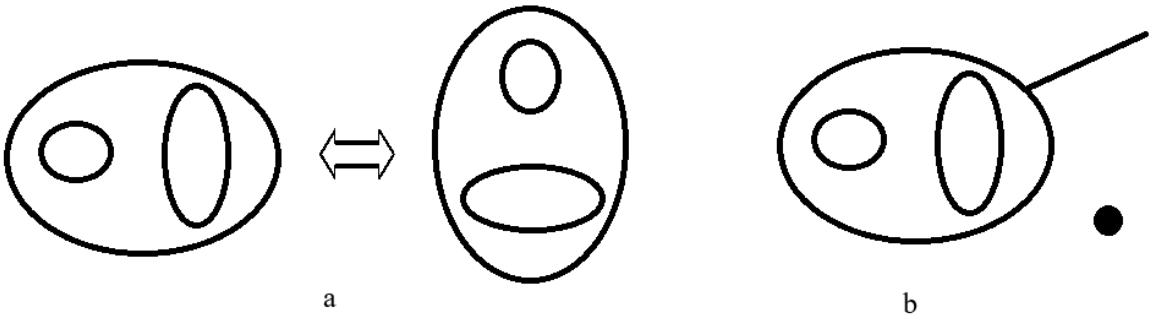


Рисунок 1.1 – Суцільна та несуцільна множина

1.2 Методи геометричного моделювання складних тіл

Для методів геометричного моделювання можна виділити такі ключові характеристики:

- область застосування – множина тіл, які можуть бути описані за допомогою даного методу моделювання;
- однозначність – метод вважається однозначним, якщо кожній геометричній моделі відповідає лише один конкретний об’єкт;
- унікальність – можливість створення геометричної моделі заданого об’єкта лише одним способом;
- точність – геометрична модель є точною, якщо вона не базується на апроксимації геометричних властивостей об’єкта;
- валідність (уявність) – гарантія того, що метод не дозволяє створювати моделі вироджених об’єктів;
- замкненість щодо логічних операцій – результати операцій, таких як об’єднання чи перетин моделей, також належать до множини моделей суцільних тіл;
- компактність і ефективність – можливість стислого збереження геометричної структури об’єкта та забезпечення ефективних алгоритмів для розрахунку необхідних фізичних параметрів.

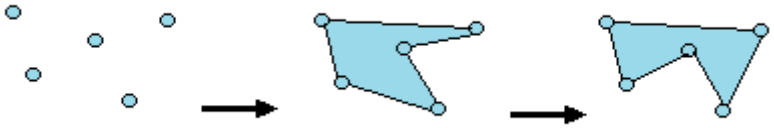


Рисунок 1.2 – Неоднозначність представлення многокутника неупорядкованою множиною точок

1.3 Інженерні креслення

У контексті складних геометричних форм креслення служать засобом передачі інформації про структуру, розміри, розташування деталей і особливості виробництва.

Для роботи з тілами складної форми креслення відображають як прості геометричні елементи, так і більш складні криві, поверхні чи об’єми. Головними елементами таких креслень є:

– геометричні види: фронтальний, горизонтальний, профільний. Вони показують основні контури об’єкта;
– перетини і розрізи: застосовуються для деталізації внутрішньої структури або прихованих елементів;
– розгортки поверхонь: використовуються для плоского представлення складних об’ємних форм.

1.4 Каркасне представлення

Каркасне представлення базується на принципі зберігання даних про структуру модельованих многогранників у двох таблицях: таблиці вершин і таблиці ребер.

У цьому випадку таке представлення може бути неоднозначним (рис. 1.3).

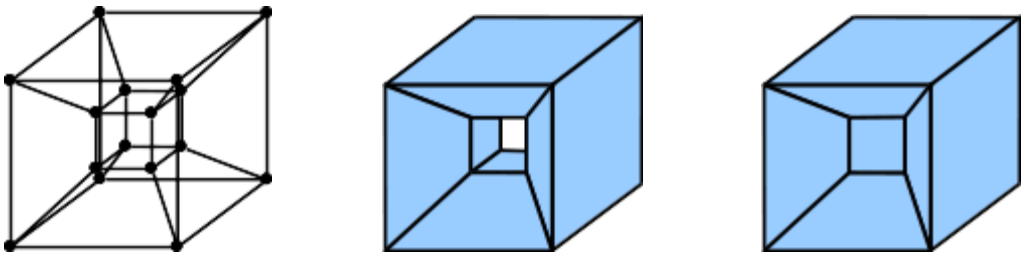


Рисунок 1.3 – Приклад неоднозначності каркасного представлення

1.5 Плоскогранне представлення

Плоскогранне представлення є способом опису складних геометричних форм через багатогранники, поверхня яких складається з плоских граней. У цьому підході будь-яка складна форма апроксимується набором плоских поверхонь, зазвичай у вигляді трикутників або чотирикутників, що дозволяє значно спростити її візуалізацію, аналіз і обробку. В інженерних кресленнях цей метод використовується для полегшення побудови виглядів, перетинів і проєкцій, оскільки плоскі грані простіше відобразити, ніж складні криволінійні поверхні. Таке представлення є основою для багатьох технологічних процесів, включаючи лазерне різання, фрезерування та 3D-друк, оскільки саме плоскогранна структура зручна для програмування виробничого обладнання. Крім того, плоскогранне моделювання дозволяє виконувати точні розрахунки площ, об’ємів, а також аналіз фізичних властивостей об’єкта. У кресленнях об’єкт, представлений плоскими гранями, зазвичай зображується як набір ліній або контурів, що формують його сітчасту структуру, а розгортки таких багатогранників дозволяють отримати плоске представлення для виготовлення

1.6. Макетування параметризованих об’єктів

Процес створення моделей або креслень, у яких геометрія описується за допомогою змінних параметрів. Цей підхід дозволяє гнучко змінювати форму та розміри об’єктів, адаптуючи їх до різних умов чи вимог проєкту. В основі параметризації лежить використання математичних залежностей, де кожен елемент об’єкта (довжина, ширина, кут нахилу, радіус тощо) може бути пов’язаний із конкретним набором вхідних параметрів.

1.7 Межове представлення

Спосіб опису геометричних об’єктів через їхні зовнішні межі (рис. 1.4). У цьому підході тривимірний об’єкт визначається набором поверхонь, які утворюють його замкнуту оболонку. Головна ідея полягає в тому, що внутрішня структура об’єкта не враховується, а вся інформація концентрується на його гранях, ребрах і вершинах.

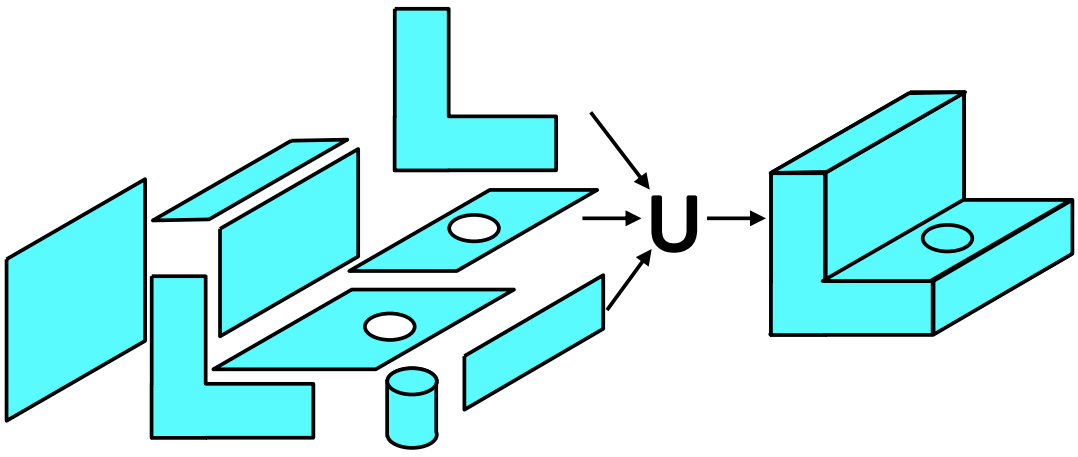


Рисунок 1.4 – Межове представлення тіла

Граничне представлення містить два основних типи інформації: топологічну та геометричну.

Топологічна інформація описує взаємозв’язки між вершинами, ребрами та гранями, включаючи орієнтацію цих елементів. Одним із найпоширеніших способів зберігання такої інформації є структура даних, відома як «крилате представлення» (winged edge).

У межах цього підходу ребра розглядаються як відрізки, а грані – як многокутники. Для кожного ребра важливими є такі дані (рис. 1.5):

– вершини, які утворюють це ребро;
– грані, розташовані ліворуч і праворуч від ребра;
– попередні та наступні ребра в послідовності обходу лівої грані;
– попередні та наступні ребра в послідовності обходу правої грані.

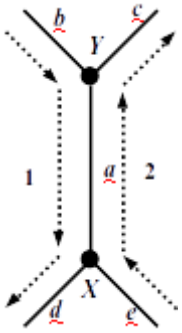


Рисунок 1.5 – Структура топологічної інформації про ребро при крилатому представленні

1.8 Конструктивна блочна геометрія

Конструктивна блочна геометрія (Constructive Solid Geometry, CSG) – це метод геометричного моделювання, який дозволяє створювати складні об’єкти шляхом застосування булевих операцій до набору простих геометричних форм.

Основними елементами цього підходу є так звані примітиви – базові геометричні фігури, такі як куби, сфери, циліндри, піраміди або конуси.

Примітиви задаються через аналітичні рівняння чи параметри, наприклад, розмір, положення або орієнтацію у просторі.

Для створення більш складних форм використовується послідовне застосування булевих операцій:

– об’єднання (Union): створює нове тіло, яке містить об’єми всіх вхідних примітивів;
– перетин (Intersection): зберігає лише ту частину тіла, яка є спільною для кількох примітивів;
– різниця (Difference): віднімає один об’єкт з іншого, формуючи нову форму.

Дане представлення схематично можна відобразити у вигляді бінарного дерева (CSG-дерево) (рис. 1.6). У такому дереві листові вузли представляють примітиви, а внутрішні вузли – бульові операції, які виконуються над цими примітивами.

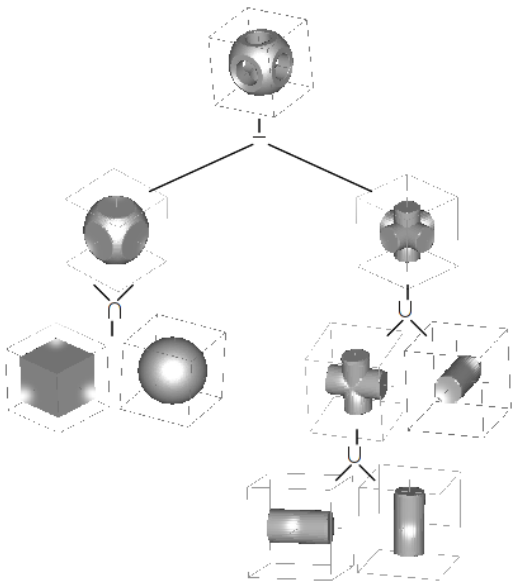


Рисунок 1.6 – CSG-дерево моделі складного тіла

CSG-дерево дозволяє компактно і зрозуміло описати послідовність операцій, використаних для створення об’єкта. Завдяки цьому легко вносити зміни до моделі: наприклад, замінити один із примітивів, змінити його розміри або переструктурувати операції.

1.9 Витягування, рух, обертання

Витягування, рух і обертання є фундаментальними методами у створенні тривимірних геометричних моделей. Ці підходи базуються на концепції, що об’ємні моделі можуть бути побудовані шляхом переміщення плоскої фігури у просторі, створюючи таким чином об’ємне тіло.

Витягування (Extrusion).

Базова ідея витягування полягає у переміщенні плоскої множини (наприклад, контуру або профілю) вздовж певного напрямку, створюючи тривимірну геометрію. Параметри витягування включають довжину переміщення, напрямок і можливе обертання профілю навколо осі. Цей метод використовується для створення таких об’єктів, як балки, стрижні або стінки.

Рух (Sweep).

Метод руху дозволяє створити об’ємне тіло шляхом переміщення плоскої фігури уздовж довільної траєкторії (наприклад, лінії, кривої або спіралі). Цей підхід дає змогу створювати складніші форми, зокрема труби, кабелі чи елементи складної кривизни. Важливим аспектом є контроль за орієнтацією профілю під час руху, що дозволяє уникнути перекосів або деформацій.

Обертання (Revolution).

Обертання застосовується для створення тіл обертання, коли плоский контур обертається навколо осі, формуючи тривимірний об’єкт. Прикладами таких об’єктів є циліндри, конуси, сфери або складні деталі, як-от фланці та елементи машин. Параметри обертання включають кут і напрямок обертання.

Ці методи також можуть поєднуватися для створення складніших моделей. Наприклад, профіль може бути спочатку витягнутий, а потім обернений або переміщений по кривій (рис. 1.7).

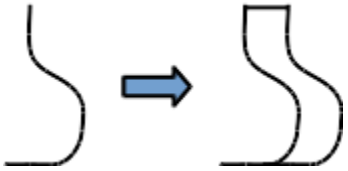


Рисунок 1.7 – Область з частиною «висячої» межі, отримане рухом кривої

1.10 Параметричне представлення

Параметричне представлення – це підхід у геометричному моделюванні, що дозволяє описувати об’єкти за допомогою математичних параметрів, забезпечуючи високу точність і гнучкість у створенні тривимірних моделей. Його основна ідея полягає не лише у параметризації двовимірних елементів, таких як криві чи поверхні, але й у забезпеченні явного доступу до внутрішньої структури об’єктів за допомогою тривимірної параметризації.

У цьому підході кожна точка тіла визначається як функція від одного чи кількох параметрів. Наприклад, тривимірна точка може бути описана координатами (x,y,z), які залежать від параметрів (u,v,w), де u,v,w – параметри, що належать заданому діапазону. Завдяки цьому метод дозволяє описувати як об’ємні тіла, так і їхні внутрішні властивості, включаючи структуру, щільність чи матеріальні характеристики.

1.11 Схеми на основі декомпозиції

Схеми на основі декомпозиції – це підхід у геометричному моделюванні, де тіло розбивається на сукупність елементів, які не перетинаються. Ці елементи можуть бути простими геометричними примітивами або частинами, які разом формують цілісну структуру об’єкта.

Розбиття може бути виконане різними способами, що визначає два основні типи представлення: на основі об’єкта та на основі простору.

1.11.1 Декомпозиція на основі об’єкта

Цей тип декомпозиції фокусується на поділі самого тіла на простіші частини. Об’єкт розбивається на складові елементи, які повністю описують його геометрію. Наприклад:

- **прямолінійні сегменти, площини або грані** – використовуються для розбиття багатогранних тіл;
- **складні криволінійні поверхні** – застосовуються для опису об’єктів зі складною геометрією, таких як криві об’єкти чи об’єкти обертання. Така декомпозиція зручна для точного опису складних форм, де кожна частина моделі відповідає реальному структурному чи функціональному компоненту.

1.11.2 Декомпозиція на основі простору

У цьому підході розбивається простір, у якому знаходиться тіло, а не саме тіло. Простір ділиться на елементи (наприклад, куби, піраміди, тетраедри), які повністю покривають тіло. Основні методи включають:

- **розбиття на регулярну сітку:** простір ділиться на рівномірні елементи. Це зручно для аналізу об’єктів, які можна описати у дискретному вигляді.
- **адаптивне розбиття:** простір ділиться залежно від складності форми тіла. У місцях зі складною геометрією сітка стає густішою, а у простих ділянках залишається грубою.

1.12 Представлення на основі серединних осей

Представлення на основі серединних осей (або медіальних осей) є важливим підходом у геометричному моделюванні, який використовується для аналізу, обробки та спрощення складних геометричних форм. Основна ідея полягає в тому, щоб описати тіло через його скелет – сукупність точок, кривих або поверхонь, які є геометричними центрами певного типу фігур (кіл, сфер), що дотикаються до меж об’єкта з максимальним можливим радіусом (рис. 1.8).

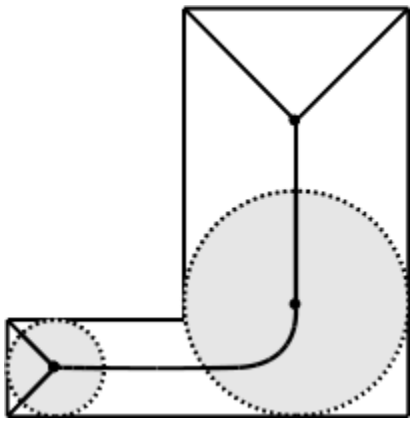


Рисунок 1.8 – Серединні осі многокутника

1.13 Функціональне представлення

Функціональне представлення є методом геометричного моделювання, в якому об’єкт описується математичними функціями або співвідношеннями, що визначають його форму, структуру та внутрішні властивості. Цей підхід базується на використанні рівнянь, які задають залежності між координатами точок і параметрами, що характеризують тіло.

Головна ідея полягає в тому, що будь-який об’єкт можна описати як множину точок, які задовольняють певному математичному рівнянню. Наприклад, для об’єктів у тривимірному просторі їхня геометрія може бути

задана у вигляді рівняння F(x,y,z)=0, F(x,y,z) – функція, що визначає форму тіла. Точки, які відповідають цьому рівнянню, утворюють поверхню або об’єм об’єкта.

Методи функціонального представлення:

- **імпліцитні функції**: описують поверхні об’єктів рівняннями вигляду F(x,y,z)=0. Наприклад: сфера: $x^2 + y^2 + z^2 - r^2 = 0$;
- **експліцитні функції:** задають залежність координат однієї змінної від інших, наприклад $z = f(x, y)$;
- **процедурні методи:** функції створюються як алгоритми, що генерують геометричні форми. Це дозволяє описувати більш складні або випадкові структури, як-от ландшафти чи органічні форми.

1.14 Методи побудови дискретного представлення складних об’єктів чотирикутними та шестигранними елементами

Методи дискретизації на чотирикутні елементи поділяються на дві основні категорії:

- методи створення структурованих (або блочно-структурованих) сіток;
- методи формування неструктурованих сіток.

Ключова різниця між цими типами сіток полягає у способі представлення даних, що описують сітку. Структурована сітка чотирикутників формується як множина координат, які організовано у вигляді матриці (рис. 1.9). У такій структурі сусідні точки сітки відповідають сусіднім елементам матриці.

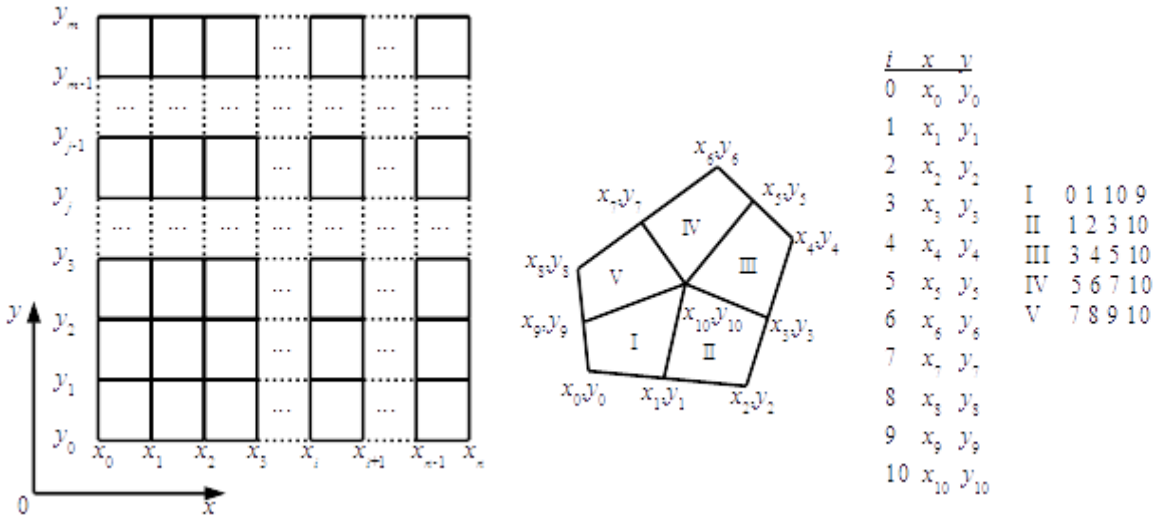


Рисунок 1.9 – Структурована та неструктурована сітки

Неструктурована сітка не може бути описана у вигляді матриці та потребує додаткових даних про зв’язки між вузлами. Зазвичай для цього використовують дві таблиці: одну для зберігання координат вузлів і другу для опису зв’язків між ними (рис1.9).

Головною перевагою неструктурованої сітки є її гнучкість: можна додавати або видаляти вузли й елементи відповідно до геометричних особливостей об’єкта або фізичних вимог задачі без необхідності повної перебудови сітки.

Методи створення неструктурованих сіток з чотирикутних та шестигранних елементів поділяються на два основні типи:

– прямі методи, які передбачають безпосереднє формування чотирикутних елементів у межах заданої області.
– непрямі методи, що ґрунтуються на перетворенні попередньо побудованої тріангуляції області.

1.15 Структуровані та блочно-структуровані сітки

Один із підходів до представлення тіл базується на трансформуванні сітки, спочатку визначеної на одиничному кубі, у необхідну геометрію за допомогою відображення $F: [0,1]^3 \to \Omega$ (рис. 1.10).

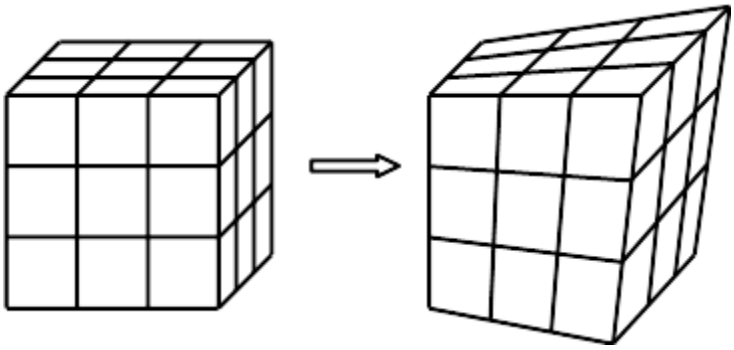


Рисунок 1.10 – Відображення сітки, визначеній на одиничному кубі на потрібну геометрію

Для розв’язання задачі дискретизації тіл складнішої форми в цьому підході застосовують методи декомпозиції та блочної декомпозиції. Ці методи передбачають (напівавтоматичний або автоматичний) поділ складного тіла на блоки, які за формою наближені до кубів.

1.16 Фронтальні методи

Основна ідея прямих фронтальних методів генерації полягає в тому, щоб почати з дискретизації межі та поступово заповнювати внутрішню область дискретними елементами по шарах (рис. 1.11).

Основними недоліками цього підходу є складнощі з виявленням вироджених елементів при перетині двох фронтів, а також труднощі в уникненні утворення пустот у процесі генерації.

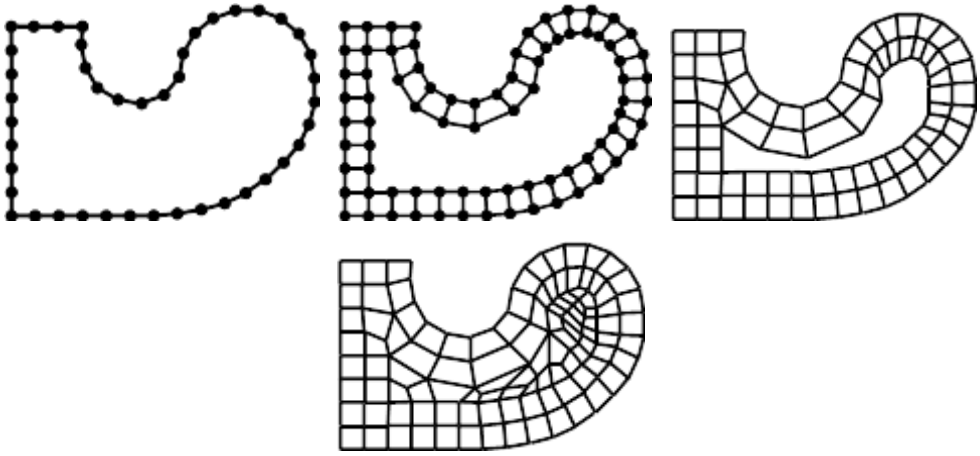


Рисунок 1.11 –Прямий фронтальний метод

Непрямі фронтальні методи передбачають використання попередньо створеної трикутної сітки, після чого трикутники трансформуються в чотирикутники. Цей підхід позбавлений проблеми перетину елементів і забезпечує захист від утворення пустот всередині контуру.

1.17 Просторовий вигін континуума

Основу просторового вигону континуума (the Spatial Twist Continuum) складає STC представлення сітки: якщо надана неструктурована сітка чотирикутних елементів, $M = (V, E, F)$ тоді STC представлення $M = (V', E', F')$ визначається наступним чином:

– середня точка v'кожної грани $f \in F$ є вузлом V';

– кожному ребру$e \in E$ відповідає ребро $e' = (v_1', v_2') \in E'$ де v_1' та v_2' – середні точки суміжних в ребрі e чотирикутників.

Кожному вузлу $v \in V$ відповідає грань $f' \in F'$, що визначається середніми точками суміжних в v чотирикутників (рис. 1.17).

Щоб створити сітку чотирикутних елементів, слід виконати наступні кроки:

1 додати хорди, які з’єднують межові сегменти;

2 додати хорди, які усувають вироджені елементи;

3 створити амбівалентне представлення.

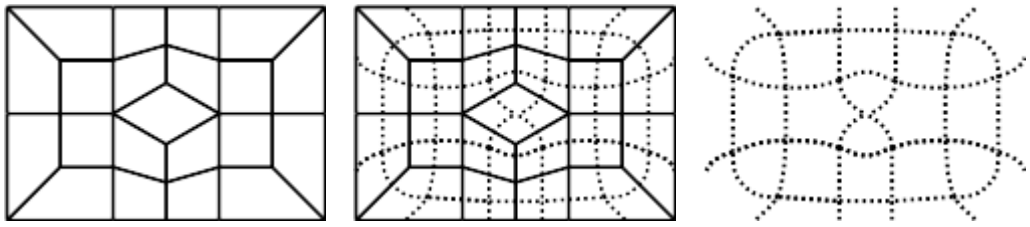


Рисунок 1.12 – Сітка чотирикутних елементів та STC-представлення

Цей метод застосовується для дискретизації множини шестигранників (рис. 1.13)

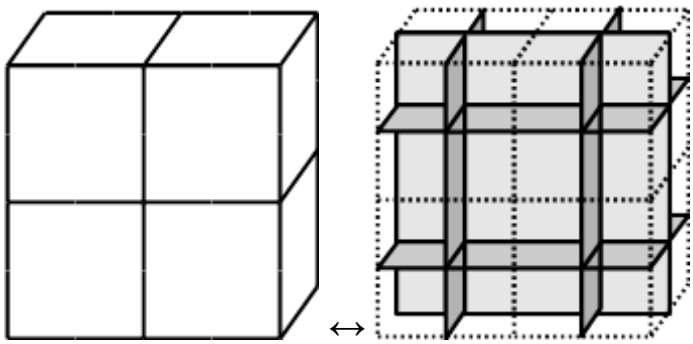


Рисунок 1.13 – STC-представлення шестигранних елементів

1.18 Методи суперпозиції

У цьому класі алгоритмів, як вихідні дані, використовується сітка, яка може бути легко побудована та покриває область навколо об’єкта **Ω** .Після цього, вихідна сітка адаптується до межі об’єкта. Для цього є два основних підходи: сітковий і проекційний методи.

У сітковому методі вилучають всі комірки, розташовані поза межами об’єкта або занадто близько до нього. Залишені комірки формують початкову сітку (позначені сірим на рис. 1.14). Потім проміжок між межею об’єкта та початковою сіткою заповнюється додатковими елементами (рис. 1.14).

Важливо зазначити, що структура остаточної дискретизації визначається структурою початкової сітки.

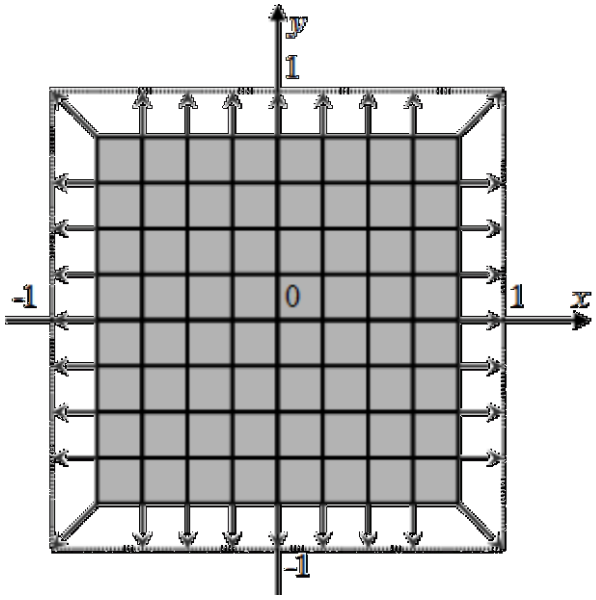


Рисунок 1.14 – Дискретизація квадрата

2 ТРІАНГУЛЯЦІЯ ПЛОСКИХ ОБЛАСТЕЙ ТА АЛГОРИТМИ ДЕЛОНЕ ЯК ОБ'ЄКТ ДОСЛІДЖЕННЯ

У цьому розділі наведено характеристику об'єкта дослідження, яким є тріангуляція плоских областей із застосуванням алгоритмів Делоне. Розглянуто основні відомості щодо геометричних принципів, які лежать в основі тріангуляції, а також вимоги та обмеження, необхідні для забезпечення її коректності та оптимальності.

Особливу увагу приділено тріангуляції Делоне як спеціальному виду розбиття, що відповідає умовам порожнього описаного кола, забезпечує мінімізацію вузьких кутів у трикутниках та дозволяє побудову високоякісних сіток. Представлено основні властивості тріангуляції Делоне, зокрема її зв'язок із діаграмою Вороного та унікальність у межах заданої множини точок.

Крім того, у цьому розділі розглядаються алгоритми побудови тріангуляції Делоне:

- Brute Force як базовий підхід із перевіркою всіх можливих трикутників;
- Flip, що виконує поступове вдосконалення тріангуляції шляхом перевороту ребер;
- інкрементальний алгоритм, який додає точки по одній, поступово оновлюючи тріангуляцію.

За результатами аналізу алгоритмів надається методика їх використання для побудови тріангуляції Делоне, оцінюються обчислювальні витрати кожного з підходів, а також виділяються основні переваги й недоліки методів у контексті їхнього застосування до різних задач обчислювальної геометрії.

2.1 Характеристика тріангуляції плоских областей та використовуваних алгоритмів Делоне

Як об’єкт дослідження в роботі розглядається тріангуляція плоских областей, яка є геометричною структурою, що забезпечує розбиття множини точок або області на трикутники, які не перетинаються між собою. Особливий інтерес викликає тріангуляція Делоне, яка задовольняє умову порожнього описаного кола, згідно з якою жодна точка множини не потрапляє всередину описаного кола будь-якого трикутника.

Тріангуляція Делоне знаходить застосування у багатьох складних системах, таких як інженерні розрахунки, моделювання рельєфу, аналіз даних та комп’ютерна графіка. Головною її особливістю є оптимізація геометричної структури розбиття, що дозволяє мінімізувати гострі кути трикутників, підвищуючи точність і стабільність подальших обчислень.

Основною математичною моделлю, що використовується для опису тріангуляції, є набір геометричних точок на площині, зв’язків між ними та умов, які гарантують коректність структури. Для побудови тріангуляції Делоне застосовуються різні алгоритми, серед яких особливу увагу приділено алгоритмам Brute Force, Flip та інкрементальному алгоритму.

Алгоритми побудови забезпечують правильність та ефективність розбиття навіть для складних і неоднорідних множин точок. Наприклад:

- Brute Force підходить для невеликих множин точок, забезпечуючи повну перевірку можливих трикутників;
- Flip поступово вдосконалює розбиття шляхом локального аналізу сусідніх трикутників;
- Інкрементальний алгоритм дозволяє покроково додавати точки та адаптувати структуру.

Прикладами систем, у яких тріангуляція використовується як модель, є цифрові моделі рельєфу, мережі зв’язку, симуляції фізичних процесів, а також задачі візуалізації та графічного дизайну. В кожному із цих випадків

правильність і оптимальність тріангуляції мають ключове значення для отримання якісних результатів.

2.2 Особливості тріангуляції Делоне

Тріангуляція Делоне є однією з ключових структур у обчислювальній геометрії завдяки її математичним властивостям, які забезпечують широкий спектр застосувань у моделюванні, аналізі даних і чисельних розрахунках.

2.2.1 Єдиність тріангуляції Делоне

Тріангуляція Делоне для заданої множини точок є унікальною, за умови, що:

- точки не лежать на одній прямій. Це гарантує, що множина є двовимірною;
- жодні чотири точки не належать одному колу. Інакше виникає невизначеність при виборі ребер.

Якщо ці умови виконуються, тріангуляція Делоне визначається однозначно. Ця властивість забезпечує стабільність моделі і дає змогу отримувати однакові результати незалежно від обраного алгоритму.

2.2.2 Мінімізація вузьких кутів

Однією з головних переваг тріангуляції Делоне є те, що вона максимізує мінімальний кут у трикутниках. Це досягається завдяки умові порожнього описаного кола.

- уникнення плоских трикутників: Наявність "вузьких" трикутників може призвести до чисельної нестабільності, особливо в задачах інтерполяції та симуляцій;

– збалансована сітка: Тріангуляція Делоне створює сітки, у яких трикутники мають більш рівномірний розподіл кутів.

Ця властивість є особливо важливою в задачах моделювання поверхонь або рельєфів, де якість сітки суттєво впливає на точність розрахунків.

2.2.3 Оптимальність для моделювання

Тріангуляція Делоне мінімізує кілька ключових метрик, які забезпечують її придатність для чисельних розрахунків:

– мінімізація максимальної довжини ребра: Це сприяє покращенню стабільності сітки;
– зменшення похибок: Завдяки рівномірному розподілу трикутників, зменшуються похибки при інтерполяції та інтегруванні;
– ефективність у симуляціях: Тріангуляція Делоне використовується як основа для розв’язання диференціальних рівнянь методом скінченних елементів.

2.2.4 Зв’язок із опуклою оболонкою

Тріангуляція Делоне безпосередньо пов’язана з геометричною концепцією опуклої оболонки:

– проєкція в просторі: тріангуляція Делоне в d-вимірному просторі відповідає "проєкції" опуклої оболонки множини точок у просторі d+1. Наприклад, у 2D площині це можна уявити як "зріз" тривимірної оболонки;
– глобальна оптимальність: тріангуляція Делоне відповідає структурі, яка мінімізує енергетичні витрати в багатьох фізичних системах;
– цей зв’язок дозволяє використовувати ефективні алгоритми побудови тріангуляції, такі як методи опуклої оболонки, для розв’язання задач у вищих вимірах.

2.3 Алгоритми побудови тріангуляції Делоне

Розглянемо базові алгоритми: Brute Force, Flip та інкрементальний алгоритм.

2.3.1 Алгоритм Brute Force ("Груба сила")

Цей алгоритм є найпростішим підходом, який перевіряє всі можливі трикутники:

– початкова перевірка. Вибираються всі можливі комбінації трьох точок множини;
– обчислення описаного кола. Для кожного трикутника обчислюється описане коло;
– перевірка умови Делоне. Якщо всередині кола немає інших точок, трикутник додається до тріангуляції.

Переваги:

– проста реалізація;
– легко зрозуміти принцип роботи.

Недоліки:

– висока обчислювальна складність:$O(n^4)$, де n – кількість точок;
– малоефективний для великих наборів даних.

2.3.2 Алгоритм Flip ("Переворот ребер")

Цей алгоритм починається з довільної тріангуляції й поступово вдосконалює її.

– побудова початкової тріангуляції. Може бути будь-якою (наприклад, за допомогою алгоритму Sweep Line);
– перевірка умови Делоне. Для кожної пари суміжних трикутників перевіряється, чи виконується умова порожнього описаного кола;

– переворот ребра. Якщо умова не виконується, діагональ між трикутниками замінюється на іншу;
– повторення. Процес триває до тих пір, поки всі трикутники не відповідатимуть умові Делоне.

Переваги:

– порівняно проста реалізація;
– типова складність: $O(n^2)$

Недоліки:

– ускладнюється для високих розмірностей простору;
– необхідно виконувати багато перевірок для великих даних.

2.3.3 Інкрементальний алгоритм

Цей алгоритм додає точки до тріангуляції по одній, поступово оновлюючи сітку, щоб вона залишалась тріангуляцією Делоне.

Кроки алгоритму.

Початковий трикутник:

– створюється початковий великий трикутник, який охоплює всі точки множини;
– його вершини є штучними й за потреби пізніше видаляються.

Покрокове додавання точок.

– точка додається до існуючої тріангуляції;
– визначається трикутник, всередині якого лежить ця точка.

Розбиття трикутника:

– трикутник, в який потрапила нова точка, розбивається на три менші трикутники, утворені новою точкою та вершинами початкового трикутника.

Перевірка ребер:

– для кожного нового трикутника перевіряється, чи відповідають його ребра умові Делоне;

– якщо ні, виконується "переворот" ребер, щоб забезпечити правильність.

Повторення.

– процес триває до тих пір, поки всі точки не будуть додані, і всі трикутники відповідатимуть умові Делоне.

Приклад.

Уявімо, що ми додаємо точку P:

– знаходимо трикутник ABC, всередині якого лежить точка P;
– розбиваємо трикутник ABC на три нових трикутники: ABP, BCP, CAP;
– перевіряємо умову Делоне для кожного нового ребра (наприклад, AP). Якщо умова не виконується, виконується "переворот";
– продовжуємо додавати інші точки.

Переваги інкрементального алгоритму:

– простота реалізації. Кроки інтуїтивно зрозумілі;
– інкрементальність. Підходить для задач, де точки додаються поступово.

Недоліки:

– часова складність: у найгіршому випадку $O(n^2)$;
– чутливість до порядку додавання точок: різний порядок може призвести до різних проміжних тріангуляцій, що впливає на продуктивність.

2.4 Порівняння алгоритмів

Для оцінки ефективності різних підходів до розв’язання задачі побудови тріангуляції Делоне виконано порівняння трьох алгоритмів: Brute Force, Flip та Інкрементальний. У таблиці 1 наведено ключові характеристики та результати роботи кожного з них, включно з часовою складністю, ефективністю на великих даних та основними перевагами.

Таблиця 1 – Порівняння алгоритмів для триангуляції Делоне

Алгоритм	**Принцип роботи**	**Часова склад-ність**	**Переваги**	**Недоліки**
Brute Force	Перевіряє всі можливі трикутники	$O(n^4)$	Проста реалізація	Дуже повільний для великих наборів
Flip	Перевіряє суміжні трикутники, виконує "перевороти"	$O(n^2)$	Ефективний для 2D	Ускладнений для високих вимірів
Інкрементальний	Додає точки по одній, оновлює тріангуляцію	$O(n^2)$	Інкрементальність, зручність	Чутливість до порядку точок

3 АВТОМАТИЗАЦІЯ ПРОЦЕСУ ПОБУДОВИ ТРІАНГУЛЯЦІЇ ДЕЛОНЕ

У цьому розділі представлено огляд існуючих програмних засобів і алгоритмічних рішень для автоматизації побудови тріангуляції Делоне. Наведено характеристику розробленого в межах роботи програмного забезпечення, яке забезпечує автоматизацію процесу побудови тріангуляції плоских областей та перевірки її відповідності умові Делоне.

3.1 Огляд існуючих програмних засобів та інструментів для автоматизації побудови тріангуляції

Огляд існуючих програмних засобів та інструментів для автоматизації процесу побудови тріангуляції є важливим етапом дослідження. Сьогодні доступний широкий вибір програмного забезпечення, яке забезпечує реалізацію тріангуляції Делоне, її аналіз, а також візуалізацію отриманих результатів.

Одним із найпоширеніших інструментів є MATLAB із вбудованими бібліотеками, такими як Triangulation та DelaunayTriangulation, які забезпечують високопродуктивні засоби для обчислення та візуалізації тріангуляції Делоне. MATLAB також дозволяє користувачам здійснювати оптимізацію сітки, проводити аналіз кута трикутників та реалізовувати різноманітні алгоритми.

Ще одним популярним інструментом є Python із бібліотеками SciPy, NumPy та Matplotlib. Бібліотека SciPy має функцію scipy.spatial.Delaunay, яка забезпечує швидке виконання тріангуляції Делоне для двовимірних і тривимірних задач.

Недоліки стандартних рішень:

Оскільки тріангуляція Делоне часто має специфічні вимоги, такі як адаптація до нестандартних геометрій або додаткові критерії оптимізації (наприклад, зменшення числа вузьких кутів), існуюче програмне забезпечення не завжди дозволяє гнучко налаштовувати процес.

Необхідність у розробці нового програмного забезпечення

Зважаючи на вказані недоліки, у межах роботи розроблено спеціалізоване програмне забезпечення, яке враховує методики побудови тріангуляції Делоне та дозволяє:

- реалізовувати різні алгоритми тріангуляції (наприклад, Brute Force, Flip, інкрементальний);
- автоматично перевіряти виконання умови порожнього описаного кола;
- проводити оптимізацію сітки відповідно до специфіки задачі (наприклад, мінімізація максимальної довжини ребер);
- візуалізувати отримані результати та забезпечувати інтерактивний аналіз властивостей тріангуляції.

Запропоноване програмне забезпечення розроблене з урахуванням сучасних обчислювальних потреб та забезпечує можливість інтеграції з існуючими інструментами для вирішення задач геометричного моделювання та аналізу.

3.2 Опис розробленої програмної системи та її можливостей для автоматизації побудови тріангуляції Делоне

Розроблена програмна система для автоматизації процесу побудови тріангуляції Делоне складається з кількох функціональних частин, кожна з яких виконує специфічну задачу. Програмне забезпечення реалізує весь процес створення тріангуляції, перевірки її властивостей та оптимізації отриманих результатів.

Перша частина: побудова початкової тріангуляції.

На першому етапі навколо області, що підлягає тріангуляції, створюється великий початковий трикутник.

Побудова інцентра:

– інцентр трикутника – це точка перетину бісектрис його кутів, рівновіддалена від усіх сторін. Після визначення інцентра початковий трикутник розбивається на три нових, які мають спільну вершину в цій точці.

Ітеративний процес:

– кожен із нових трикутників знову ділиться на трикутники через їхні інцентри. Процес повторюється визначену кількість разів, утворюючи сітку з великої кількості менших трикутників, вкладених у великий початковий трикутник.

Результат цього етапу – базова структура, яка складається з безлічі трикутників, що слугують основою для подальшої роботи.

Друга частина:

Відсікання трикутників, що лежать поза областю тріангуляції.

На другому етапі програма працює з отриманою базовою структурою, щоб залишити тільки ті трикутники, які входять у задану область тріангуляції.

Перевірка вершин:

– для кожного трикутника перевіряється, чи входить хоча б одна його вершина в межі області тріангуляції;

– "Підтягування" вершин: Якщо хоча б одна вершина належить області, дві інші вершини цього трикутника "підтягуються" на межу області. Це забезпечує точне узгодження меж тріангуляції з заданою областю.

На цьому етапі утворюється структура, яка відображає тріангуляцію всередині заданої області.

Третя частина:

– перевірка на відповідність умовам Делоне.

У фінальній частині програма перевіряє кожен трикутник отриманої тріангуляції на відповідність умові порожнього описаного кола (умові Делоне):

– послідовна перевірка: Кожен трикутник аналізується на предмет того, чи не входять інші точки множини в описане навколо нього коло;
– корекція: Якщо умова не виконується, структура тріангуляції автоматично виправляється шляхом локального "перевороту" ребер.

Виведення результатів.

На кожному етапі програма виводить результати на екран у вигляді графічного зображення:

– початковий великий трикутник із вкладеними інцентрами;
– результати відсікання трикутників поза заданою областю;
– фінальна тріангуляція Делоне, яка задовольняє всі умови.

Особливості програмної системи:

– інтерактивність: Можливість візуалізації проміжних етапів дає змогу користувачам оцінювати кожний крок роботи програми;
– гнучкість: Кількість ітерацій, параметри початкового трикутника та область тріангуляції можна налаштовувати відповідно до задачі;
– автоматизація: Програма виконує всі обчислення без втручання користувача, надаючи кінцевий результат у вигляді тріангуляції Делоне.

Розроблена система забезпечує зручний і ефективний інструмент для автоматизації побудови тріангуляції Делоне, що робить її універсальним рішенням для задач геометричного моделювання та обчислювальної геометрії.

3.3 Приклад практичного використання програмного забезпечення

Застосування програмного забезпечення є ключовим етапом у дослідженні методів побудови тріангуляції плоских областей. Використання спеціалізованих програм дозволяє автоматизувати процес створення тріангуляції, проводити детальний аналіз отриманих структур, а також досліджувати вплив різних початкових геометричних параметрів на якість побудованої сітки.

У цьому розділі наведено результати обчислювального експерименту, проведеного за допомогою розробленої програмної системи. Вона забезпечує реалізацію тріангуляції Делоне, перевіряє кожен трикутник на відповідність умові порожнього описаного кола та дозволяє оптимізувати геометричну структуру побудованої сітки для конкретних задач.

Обчислювальний експеримент у межах роботи проводився для області, яка є колом із центром у точці (0;41). Під час експерименту були виконані наступні кроки:

- побудова початкового «великого» трикутника – супертрикутника – із розбиттям його інцентрами на дрібніші трикутники (рис. 3.1);
- відсікання трикутників, вершини яких не належать заданій області – колу (рис. 3.2-3.3);
- проекція точок трикутників на межу області для частково розташованих у ній трикутників (рис. 3.4);
- перевірка отриманої тріангуляції на виконання умови Делоне та локальна оптимізація трикутників(рис. 3.4).

Результати кожного етапу були виведені у графічному вигляді для оцінки точності й якості виконаних обчислень. Проведений експеримент продемонстрував ефективність розробленої програмної системи для задач тріангуляції плоских областей, а також її здатність автоматизувати весь процес, починаючи з побудови базової сітки і закінчуючи аналізом отриманої тріангуляції.

```
Отримані трикутники:
[[-100, 0], [0, 100], [-30.001897770165304, 63.442581792723 52]]
[[0, 100], [-26.529672086325395, 60.295539873979656], [-30.001897770165304, 63.4
42581792723 52]]
[[-26.529672086325395, 60.295539873979656], [-100, 0], [-30.001897770165304, 63.
442581792723 52]]
[[0, 100], [-19.00427977891125, 54.11961001461971], [-20.76544656834497, 61.1505
8195858303]]
[[-19.00427977891125, 54.11961001461971], [-26.529672086325395, 60.2955398739796
56], [-20.76544656834497, 61.150581958583 03]]
[[-26.529672086325395, 60.295539873979656], [0, 100], [-20.76544656834497, 61.15
0581958583 03]]
[[-19.00427977891125, 54.11961001461971], [-100, 0], [-26.529672086325398, 54.48
93088319623 54]]
[[-100, 0], [-26.529672086325395, 60.295539873979656], [-26.529672086325398, 54.
489308831962354]]
[[-26.529672086325395, 60.295539873979656], [-19.00427977891125, 54.119610014619
71], [-26.529672086325398, 54.489308831962354]]
[[0, 100], [0.0, 41.4213562373095], [-4.141181437079744, 57.95387926332645]]
[[0.0, 41.4213562373095], [-8.491864057829112, 57.308516460311665], [-4.14118143
7079744, 57.953879263326 45]]
[[-8.491864057829112, 57.308516460311665], [0, 100], [-4.141181437079744, 57.953
879263326 45]]
[[0.0, 41.4213562373095], [-19.00427977891125, 54.11961001461971], [-10.34481398
9375382, 52.835099605174 36]]
[[-19.00427977891125, 54.11961001461971], [-8.491864057829112, 57.308516460311665
5], [-10.344813989375382, 52.835099605174 36]]
[[-8.491864057829112, 57.308516460311665], [0.0, 41.4213562373095], [-10.3448139
89375382, 52.835099605174 36]]
[[-19.00427977891125, 54.11961001461971], [0, 100], [-11.988863882699365, 60.478
01235164327]]
[[0, 100], [-8.491864057829112, 57.308516460311665], [-11.988863882699365, 60.47
801235164327]]
[[-8.491864057829112, 57.308516460311665], [-19.00427977891125, 54.1196100146197
1], [-11.988863882699365, 60.478012351643 27]]
[[0.0, 41.4213562373095], [-100, 0], [-18.097967779229748, 38.736776314340 27]]
[[-100, 0], [-19.004279778911254, 43.293114422526656], [-18.097967779229748, 38.
736776314340 27]]
[[-19.004279778911254, 43.293114422526656], [0.0, 41.4213562373095], [-18.097967
779229748, 38.736776314340 27]]
```

Рисунок 3.1 – Скріншот роботи програми – виведення побудови початкового супертрикутника із розбиттям його інцентрами на дрібніші трикутники

На першому етапі роботи програми було побудовано початкову структуру, яка складається з супертрикутника та вкладених у нього дрібніших трикутників, утворених через інцентри.

Однак, не всі утворені трикутники є релевантними для задачі, оскільки наша цільова область обмежена заданою геометричною формою (колом). Тому наступним етапом є відбір тих трикутників, які хоча б однією своєю вершиною знаходяться всередині цієї області.

Цей підхід дозволяє звузити розрахунки лише до релевантної частини структури та гарантує, що кінцева тріангуляція буде відповідати вимогам до заданої області.

```
трикутники які входять:
[[0, 100], [0.0, 41.4213562373095], [-4.141181437079744, 57.95387926332645]]
[[0.0, 41.4213562373095], [-8.491864057829112, 57.308516460311665], [-4.14118143
7079744, 57.95387926332645]]
[[-8.491864057829112, 57.308516460311665], [0, 100], [-4.141181437079744, 57.953
87926332645]]
[[0.0, 41.4213562373095], [-19.00427977891125, 54.11961001461971], [-10.34481398
9375382, 52.83509960517436]]
[[-19.00427977891125, 54.11961001461971], [-8.491864057829112, 57.30851646031166
5], [-10.344813989375382, 52.83509960517436]]
[[-8.491864057829112, 57.308516460311665], [0.0, 41.4213562373095], [-10.3448139
89375382, 52.83509960517436]]
[[0, 100], [-8.491864057829112, 57.308516460311665], [-11.988863882699365, 60.47
801235164327]]
[[-8.491864057829112, 57.308516460311665], [-19.00427977891125, 54.1196100146197
1], [-11.988863882699365, 60.47801235164327]]
[[0.0, 41.4213562373095], [-100, 0], [-18.097967779229748, 38.73677631434027]]
[[-100, 0], [-19.004279778911254, 43.293114422526656], [-18.097967779229748, 38.
73677631434027]]
[[-19.004279778911254, 43.293114422526656], [0.0, 41.4213562373095], [-18.097967
779229748, 38.73677631434027]]
[[-19.00427977891125, 54.11961001461971], [-19.004279778911254, 43.2931144225266
56], [-23.38704103456596, 45.92004042500092]]
[[-19.004279778911254, 43.293114422526656], [-100, 0], [-23.38704103456596, 45.9
20040425000921]]
[[-19.00427977891125, 54.11961001461971], [0.0, 41.4213562373095], [-15.10595237
973214, 46.826352424682421]]
[[0.0, 41.4213562373095], [-19.004279778911254, 43.293114422526656], [-15.105952
37973214, 46.826352424682421]]
[[-19.004279778911254, 43.293114422526656], [-19.00427977891125, 54.119610014619
71], [-15.10595237973214, 46.826352424682421]]
[[100, 0], [0.0, 41.4213562373095], [18.097967779229748, 38.73677631434027]]
[[0.0, 41.4213562373095], [19.004279778911254, 43.293114422526656], [18.09796777
9229748, 38.73677631434027]]
[[19.004279778911254, 43.293114422526656], [100, 0], [18.097967779229748, 38.736
77631434027]]
[[0.0, 41.4213562373095], [19.00427977891125, 54.11961001461971], [15.1059523797
3214, 46.826352424682421]]
[[19.00427977891125, 54.11961001461971], [19.004279778911254, 43.293114422526656
```

Рисунок 3.2 – Скріншот роботи програми 2 – виведення відсікання трикутників, вершини яких не належать заданій області (колу)

Проекція точок трикутників на межу області для частково розташованих у ній трикутників. На цьому етапі програма аналізує отримані трикутники базової структури, які частково знаходяться в межах заданої

області (кола). Якщо хоча б одна вершина трикутника лежить всередині області, решта вершин коригується таким чином, щоб належати її межі.

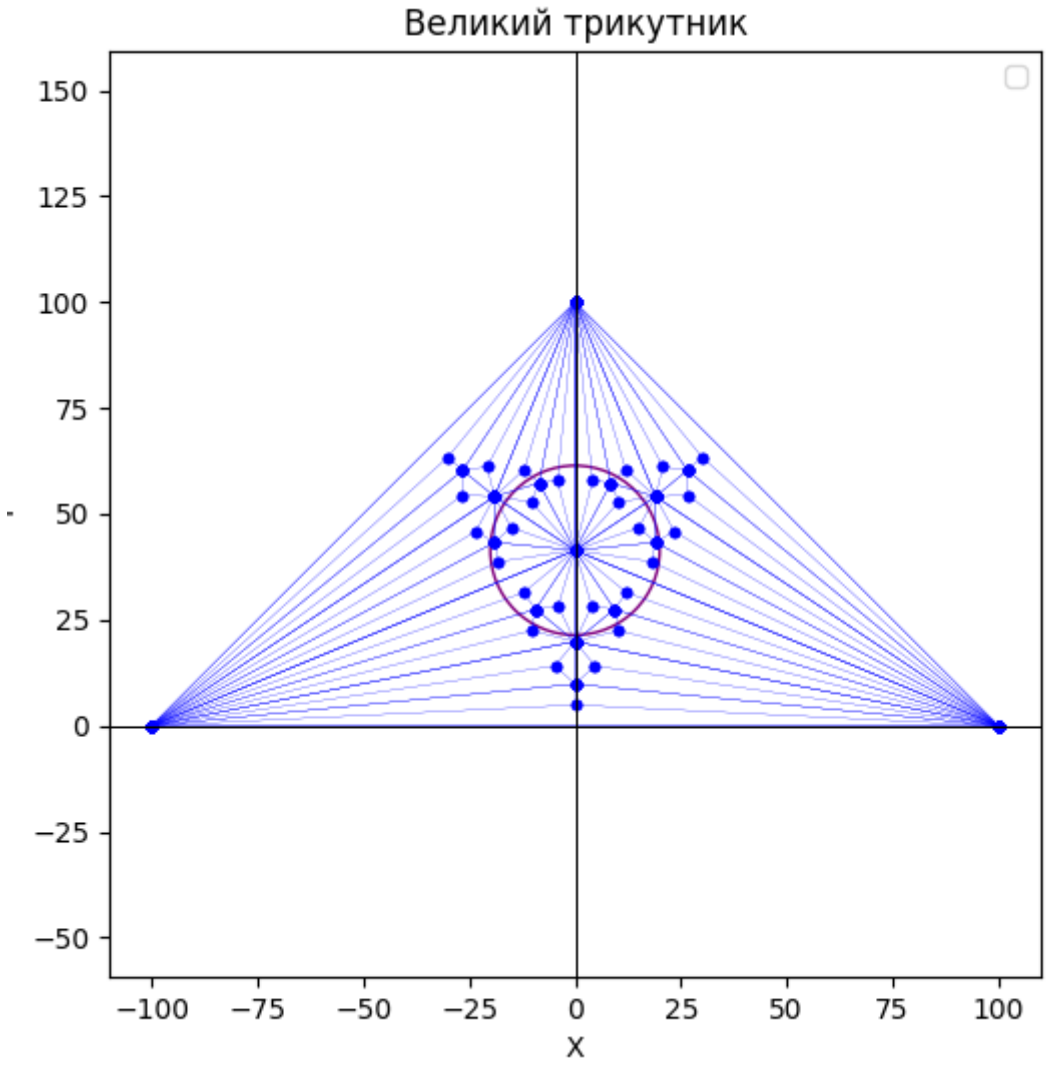


Рисунок 3.3 – Скріншот роботи програми 3 – графічне відображення отриманих трикутників та шуканої області

Алгоритм.

Перевірка вершин:

– кожен трикутник перевіряється на належність його вершин області.

Визначення точки входження:

– якщо одна або дві вершини знаходяться в області, визначається перетин сторін трикутника з межею області.

Зміщення вершин:

– вершини, які знаходяться за межами області, зміщуються на лінію межі;
– корекція виконується так, щоб геометрія трикутника залишалась коректною.

Актуалізація сітки:

– після коригування трикутники перезаписуються в структурі даних програми.

Усі трикутники, хоча б одна вершина яких лежала всередині області, адаптуються до межі. Це забезпечує правильне представлення тріангуляції в межах заданої геометрії та дозволяє уникнути виходу точок за межі цільової області.

Підсумком цього етапу є сітка, повністю узгоджена з заданою областю (рис. 3.4), що готує структуру для подальшої перевірки на відповідність умовам Делоне.

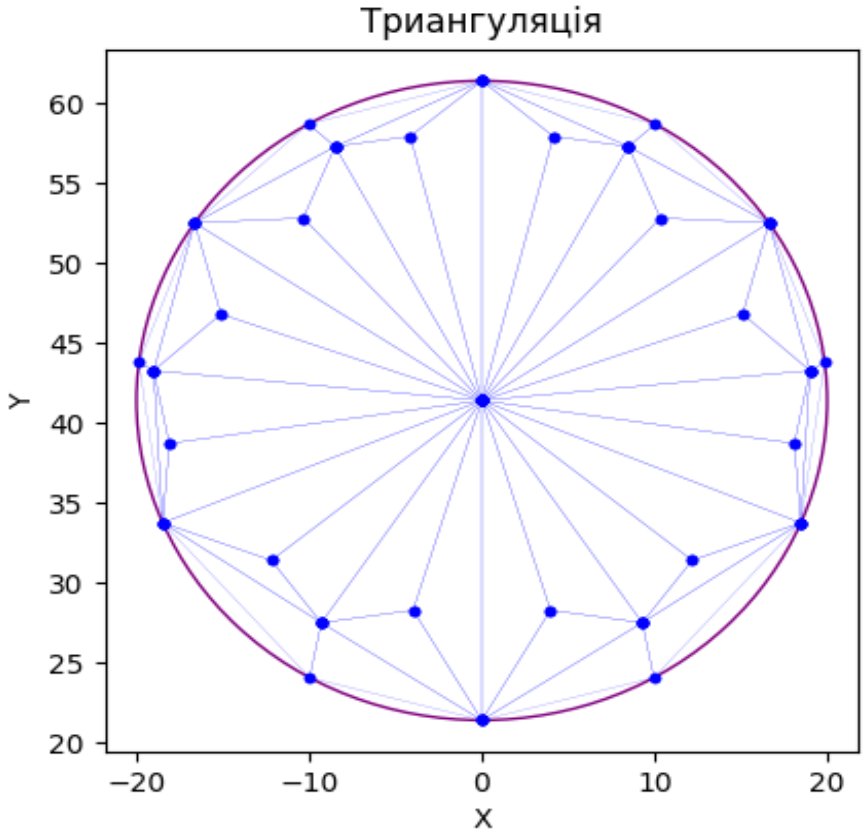


Рисунок 3.4 – Скріншот роботи програми 4 – проекція точок трикутників на межу області

Перевірка трикутників на відповідність умовам Делоне.

На цьому етапі програма аналізує отриману тріангуляцію, перевіряючи кожен трикутник на виконання умови Делоне. Основний критерій цієї умови – порожнє описане коло, тобто для будь-якого трикутника описане навколо нього коло не повинно містити жодної іншої точки множини.

Програма виводить оновлену тріангуляцію (рис. 3.5), яка задовольняє умові Делоне. Для кожного трикутника може бути візуалізовано його описане коло, що дозволяє оцінити правильність отриманої структури.

Результат перевірки:

У підсумку формується тріангуляція Делоне, яка є оптимальною з точки зору геометричних властивостей, таких як мінімізація вузьких кутів. Ця структура готова до подальшого використання в задачах моделювання та аналізу.

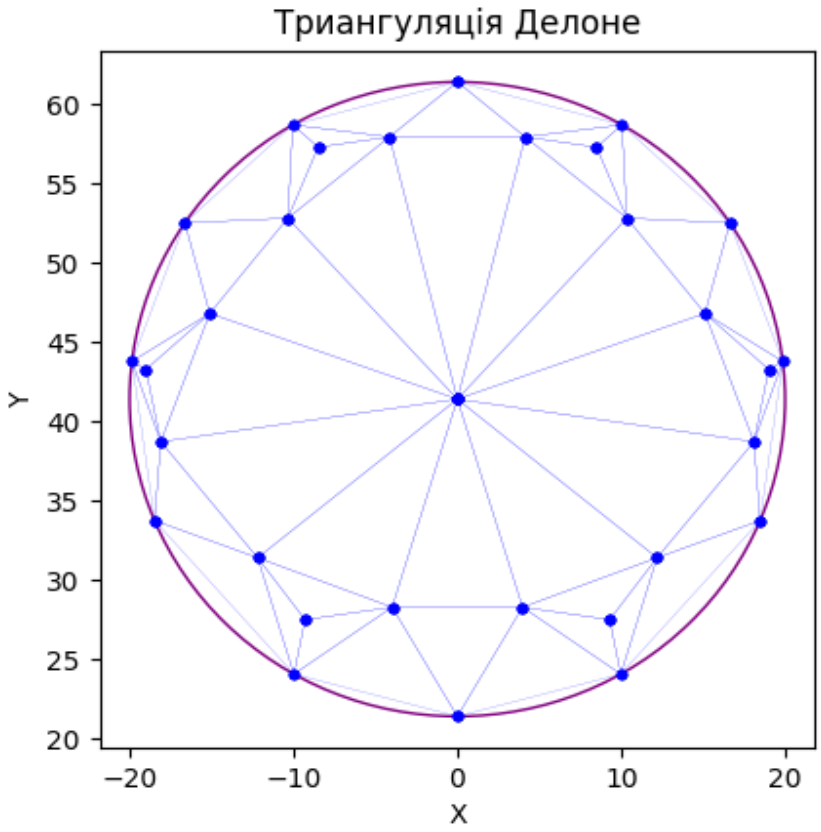


Рисунок 3.5 – Скріншот роботи програми 5 – графічне виведення тріангуляції Делоне

Другий обчислювальний експеримент у межах роботи проводився для області, яка є прямокутником(рис.3.6) із вершинами в точках, заданих координатами:

(−20;21.4213562373095),

(20;21.4213562373095),

(20;61.4213562373095),

(−20;61.4213562373095)

Результат роботи програми представлений на рисунках 3.6, 3.7 і 3.8. На рисунку 3.7 зображено проекцію точок трикутників на межу області прямокутника. На рисунку 3.8 продемонстровано графічне виведення тріангуляції Делоне для заданого прямокутника.

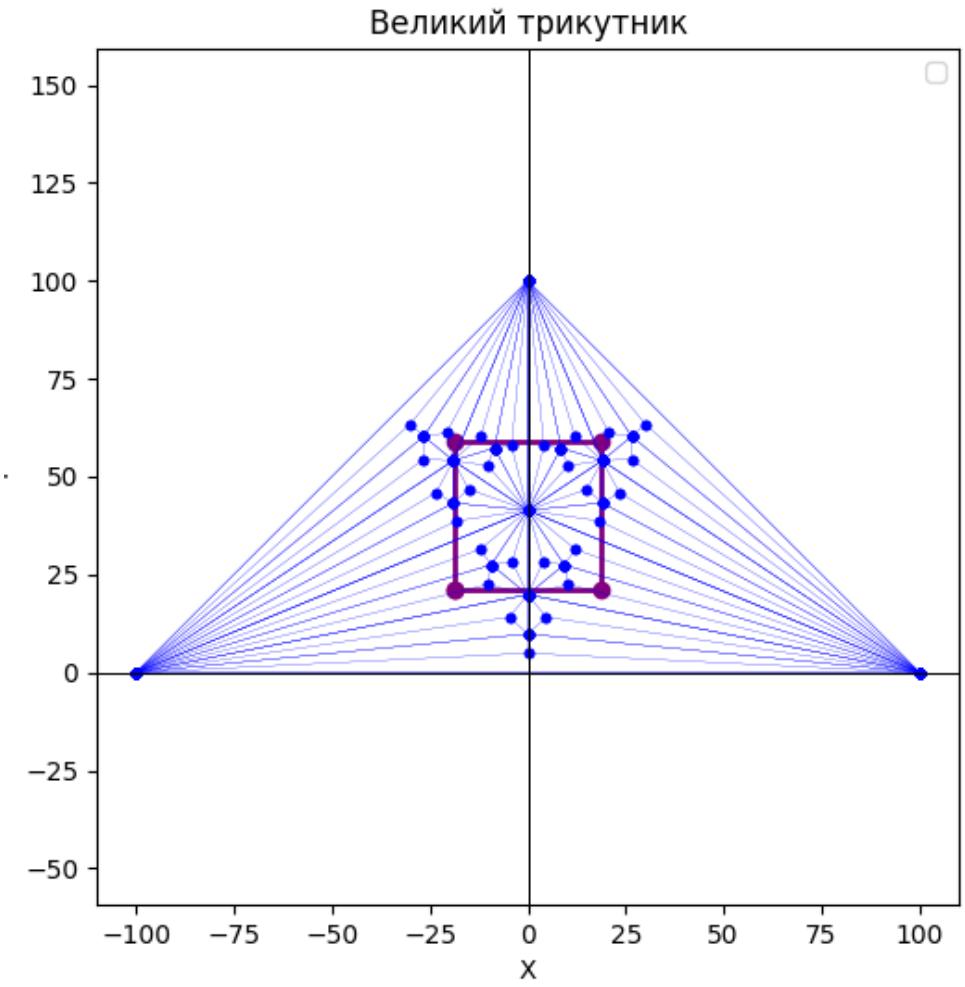


Рисунок 3.6 – Скріншот роботи програми 6 – графічне відображення отриманих трикутників та шуканої області прямокутника

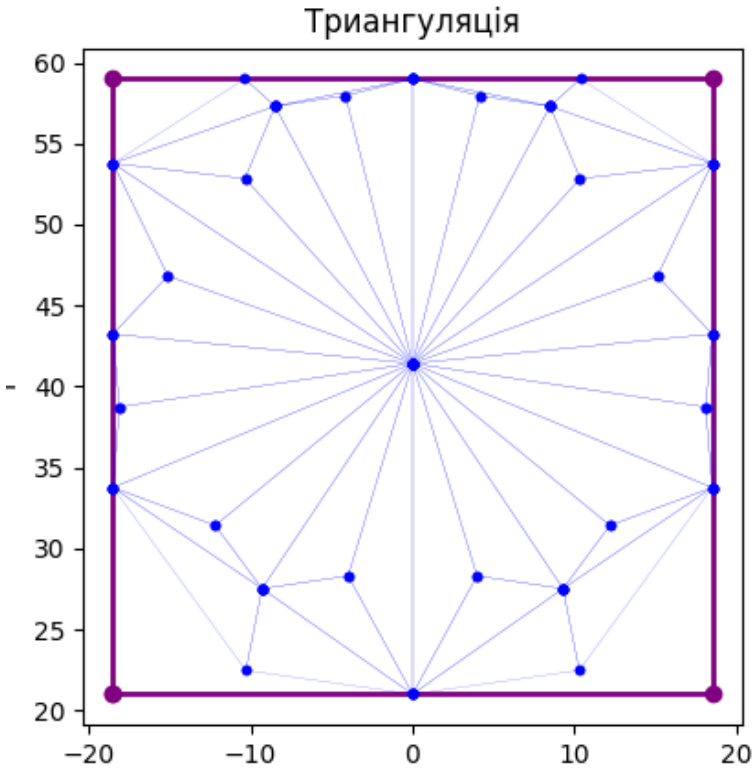


Рисунок 3.7 – Скріншот роботи програми 7 – проекція точок трикутників на межу області прямокутника

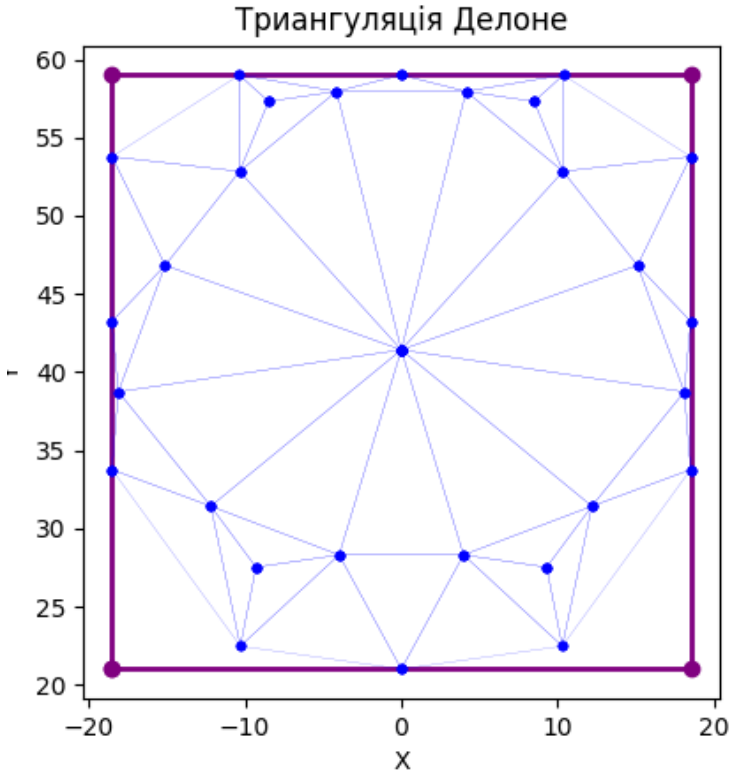


Рисунок 3.8 – Скріншот роботи програми 8 – графічне виведення тріангуляції Делоне для прямокутника

ВИСНОВКИ

Кваліфікаційна робота присвячена розробці методики побудови тріангуляції Делоне та реалізації програмної системи для автоматизації цього процесу, що дозволяє проводити аналіз отриманої тріангуляції, перевіряти її відповідність умовам Делоне та оптимізувати структуру для використання у задачах обчислювальної геометрії. Дослідження проводилося з метою зменшення трудомісткості обчислень, підвищення точності та якості отриманих результатів, а також забезпечення можливості автоматизованого виконання завдань моделювання.

Структурно кваліфікаційна робота складається з трьох розділів.

У першому розділі здійснено огляд та класифікацію методів геометричного моделювання та дискретизації тіл складної форми. Розглянуто загальні принципи розбиття областей на елементи, наведено основні типи геометричних сіток.

У другому розділі наведено характеристики тріангуляції як об’єкта дослідження. Описано основні етапи роботи алгоритмів побудови тріангуляції Делоне, зокрема формування початкового великого трикутника, ітеративний поділ через інцентри, обмеження трикутників областю роботи, а також перевірка на відповідність умовам Делоне. Викладено основні вимоги до структури тріангуляції, необхідні для забезпечення її коректності та оптимальності, які стали основою для реалізації програмного забезпечення, що дозволяє автоматизувати процес побудови й аналізу тріангуляції.

Третій розділ присвячено опису розробленого програмного продукту – системи автоматизації побудови тріангуляції Делоне. Наведено особливості алгоритмічної реалізації: створення базової тріангуляції, корекція положення вершин трикутників, що частково входять у задану область, та перевірка відповідності умовам Делоне. Розроблена система інтерактивно відображає результати кожного етапу у вигляді графіків, що дозволяє користувачу

спостерігати за процесом і аналізувати отримані результати. У розділі також представлено результати обчислювальних експериментів, проведених для кругової та прямокутної областей із заданими параметрами, що підтвердили ефективність і надійність програмного забезпечення.

Отримані в роботі результати демонструють високу ефективність запропонованої методики та розробленої програми. Система може бути використана для побудови тріангуляцій у задачах моделювання рельєфу, обробки географічних даних, комп’ютерної графіки, чисельного моделювання фізичних процесів і розв’язання інженерних задач. Це робить розробку актуальною як у наукових дослідженнях, так і у практичному застосуванні.

ПЕРЕЛІК ПОСИЛАНЬ

1. Копей В. Б. Мова програмування Python для інженерів і науковців : навч. посіб. Івано-Франківськ : ІФНТУНГ, 2019. 274 с.

2. Кундрат А. М., Кундрат М. М. Науково-технічні обчислення засобами MathCAD та MS Excel : навч. посіб. Рівне : НУВГП, 2014. 252 с.

3. Нікітенко О. М. Maple. Розв’язання інженерних та наукових задач : навч. посіб. Харків : ХНУРЕ, 2014. 289 с.

4. Яковенко А. В. Основи програмування. Python. Ч.1: підручн. для студ. спеціальності 122 «Комп’ютерні науки», спеціалізації «Інформаційні технології в біології та медицині». Київ : КПІ ім. Ігоря Сікорського, 2018. 195 с.

5. Joshi Prateek. Artificial Intelligence with Python. Birmingham : Packt, 2017. 437 p.

6. Y. S. Elshakhs, K. M. Deliparaschos, T. Charalambous, G. Oliva, and A. Zolotas, "A Comprehensive Survey on Delaunay Triangulation: Applications, Algorithms, and Implementations Over CPUs, GPUs, and FPGAs," IEEE Access, vol. 12, pp. 12562-12585, 2024. Publisher: IEEE.

7. Shewchuk J. R. Delaunay refinement algorithms for triangular mesh generation Computational Geometry. 2002. Vol. 22, No. 1–3. P. 21–74.

8. De Berg M., Van Kreveld M., Overmars M., Schwarzkopf O. Computational Geometry : Algorithms and Applications. 3rd ed. Berlin: Springer, 2008. 386 p.

9. Choporov S. V., Homeniuk S. I., Kudin O. V., Lisnyak A. O. "Design Patterns for Object-Oriented Scientific Software. CEUR Workshop Proceedings. 2018. Т. 2105. С. 441–444.

ДОДАТОК А

Лістинг програмного коду

```
import numpy as np
import math
import matplotlib.pyplot as plt

def calculate_incenter(vertices):
    """
    Обчислює інцентр трикутника.
    :param vertices: Список з трьох вершин трикутника [[x1, y1], [x2, y2], [x3, y3]].
    :return: Координати інцентра [ix, iy].
    """
    A, B, C =map(np.array, vertices)
    # Обчислюємо довжини сторін
    AB = math.sqrt((C[0] - B[0])**2 + (C[1] - B[1])**2)
    AC = math.sqrt((C[0] - A[0])**2 + (C[1] - A[1])**2)
    BC = math.sqrt((B[0] - A[0])**2 + (B[1] - A[1])**2)
   # Інцентр
    ix = ((AB * A[0] + AC * B[0] + BC * C[0]) / (AB + AC + BC))
    iy = ((AB * A[1] + AC * B[1] + BC * C[1]) / (AB + AC + BC))
    #print('incenter: ',ix,iy)
    return [float(ix), float(iy)]

def generate_new_triangles(vertices):
    """
    Створює три нових трикутника з початкового та його інцентра.
    :param vertices: Список з трьох вершин трикутника [[x1, y1], [x2, y2], [x3, y3]].
    :return: Список з трьох нових трикутників.
    """
    data = calculate_incenter(vertices)
    #incenter=[float(value) for value in data]
    incenter=[value for value in data]
    #print('incenter: ',incenter)
    A, B, C = vertices
    # Створюємо три нових трикутника
    triangles = [
        [A, B, incenter],
        [B, C, incenter],
        [C, A, incenter]
    ]

    #print('\n\n')
    return triangles
```

```
triangles=[]

A=[-100,0]
B=[0,100]
C=[100,0]
triangles.append([A,B,C])
get_all_incenters=[]


get_all_incenters.append(A)
get_all_incenters.append(B)
get_all_incenters.append(C)
print('get_all_incenters',get_all_incenters)

#depth=40# глубина трианглуяц.   #<112
depth=40
k=0
L=0
plt.ion()

plt.figure(figsize=(6, 6))
plt.legend()
plt.title('Великий трикутник')
plt.xlabel('X')
plt.ylabel('Y')
plt.axis('equal')



circle_center = (0.0, 41.4213562373095)
radius = 20

circle = plt.Circle((circle_center), radius, color='purple', fill=False, linewidth=1, label='Круг')  # Определяем круг
plt.gca().add_artist(circle)
#a=input()
while depth>0:
    appending=generate_new_triangles(triangles[k])
    get_all_incenters.append(calculate_incenter(triangles[k]))
    for i in range(3):
        triangles.append(appending[i])
    for L in range(len(appending)):

        triangle = appending[L]
        x_coords = [point[0] for point in triangle] + [triangle[0][0]]
        y_coords = [point[1] for point in triangle] + [triangle[0][1]]
        #plt.pause(0.5)
        plt.plot(x_coords, y_coords, marker='.', linestyle='-', color='b',linewidth=0.1, label='Треугольник')  # Рисуем контур
        plt.axhline(0, color='black',linewidth=0.5)  # Ось X
```

```
        plt.axvline(0, color='black',linewidth=0.5)  # Ось Y
        #plt.pause(0.005)
    depth-=1

    triangles.pop(k)

print('Отримані трикутники:')
for i in range(len(triangles))  :
    print(triangles[i])

# Функція для перевірки попадання точки в коло
def is_inside_circle(point, center, radius):
    x, y = point
    cx, cy = center
    if (x - cx) ** 2 + (y - cy) ** 2 <= radius ** 2:
        return(True)

 # Додаємо круг на графік
trianlges_in=[]
for i in range(len(triangles)):
    if is_inside_circle(triangles[i][0],circle_center, radius)or
is_inside_circle(triangles[i][1],circle_center, radius) or
is_inside_circle(triangles[i][2],circle_center, radius):
        trianlges_in.append(triangles[i])

print('\n\n\n\nтрикутники які входять: ')
for i in range(len(trianlges_in)):
    print(trianlges_in[i])



plt.show()

triangles_in=trianlges_in

#tianlges_made_by_incenters:
tr_md_by_inc=[[]for _ in range(len(triangles_in))]

# Для кожного з елементів в triangles_in шукаємо індекс в get_all_incenters
for i, triangle in enumerate(triangles_in):
    for vertex in triangle:
        try:
            # Спробуємо знайти індекс вершини в get_all_incenters
            index = get_all_incenters.index(vertex)
            #print(f"Вершина {vertex} трикутника {i} знайдена на позиції {index} в
get_all_incenters")
            #tr_md_by_inc[i][k]=index
            tr_md_by_inc[i].append(index)
            #координати вершини в vertex
            #номер трикутника в i
            #номер координат вершини в index
            #
```

```
        except ValueError:
            # Якщо вершина не знайдена
            print(f"Вершина {vertex} треугольника {i} не найдена в get_all_incenters")

temp=[‘a’,’b’,’c’]
check_tr_in=[[‘a’,’b’,’c’]for _ in range(len(triangles_in))]

def find_intersection(circle_center, radius, point1, point2):
    # Декомпозиція вхідних даних
    x_c, y_c = circle_center
    x1, y1 = point1
    x2, y2 = point2
    intersections = []

      # Перевіряємо, чи обидві точки знаходяться всередині окружності
    d1 = (x1 - x_c)**2 + (y1 - y_c)**2
    d2 = (x2 - x_c)**2 + (y2 - y_c)**2
    if d1 < radius**2 and d2 < radius**2:
        return point2  # Обидві точки всередині окружності, перетинів немає



    # Коєфіцієнти параметричного рівняння
    dx, dy = x2 - x1, y2 - y1

    # Коєфіцієнти квадратного рівняння at^2 + bt + c = 0
    a = dx**2 + dy**2
    b = 2 * (dx * (x1 - x_c) + dy * (y1 - y_c))
    c = (x1 - x_c)**2 + (y1 - y_c)**2 - radius**2


    if a == 0 and b == 0:
        # Перевіряємо, чи лежить точка на окружності
        if c == 0:
            intersections.append(x1)
            intersections.append(y1)
            return intersections  # Точка лежить на окружності
        return []  # Точка поза окружністю

    # Якщо a == 0 (але b != 0), це лінійне рівняння bt + c = 0
    if a == 0 and b != 0:
        t = -c / b
        if 0 <= t <= 1:
            x = x1 + t * dx
            y = y1 + t * dy
            intersections.append(x)
            intersections.append(y)
            return intersections  # Перетин з окружністю
        return []  # Немає перетину на відрізку    # Дискримінант
    discriminant = b**2 - 4 * a * c
    # Якщо дискримінант < 0, перетинів немає
    if discriminant < 0:
```

```
        return []
    if discriminant == 0:
        t = -b / (2 * a)
        if 0 <= t <= 1:
            x = x1 + t * dx
            y = y1 + t * dy
            #print('x,y:',x,y)
            intersections.append(x)
            intersections.append(y)
            return intersections  # Точка дотику
        return []
   # Рішення квадратного рівняння
    t1 = (-b + math.sqrt(discriminant)) / (2 * a)
    t2 = (-b - math.sqrt(discriminant)) / (2 * a)

    # Перевіряємо, чи попадають t в діапазон [0, 1]

    for t in (t1, t2):
        if 0 <= t <= 1:
            x = x1 + t * dx
            y = y1 + t * dy
            intersections.append(x)
            intersections.append(y)
            return intersections

    return intersections


#print('\n\n\n\n\nТУТ ПОЧИНАЄТЬСЯ ПЕРЕВІРКА',end='\n\n\n\n')
for i in range(len(triangles_in)):
    if is_inside_circle(get_all_incenters[tr_md_by_inc[i][0]],circle_center,radius):
        check_tr_in[i][0]=1
    else:
        check_tr_in[i][0]=0

    if is_inside_circle(get_all_incenters[tr_md_by_inc[i][1]],circle_center,radius):
        check_tr_in[i][1]=1
    else:
        check_tr_in[i][1]=0
    if is_inside_circle(get_all_incenters[tr_md_by_inc[i][2]],circle_center,radius):
        check_tr_in[i][2]=1
    else:
        check_tr_in[i][2]=0
    #[1,0,0]
    if ((check_tr_in[i][0]==1)and(check_tr_in[i][1]==0)and(check_tr_in[i][2]==0)):
        #функцію яка отримує дві точки,виводить перетин другої з окр.
        get_all_incenters[tr_md_by_inc[i][1]]=find_intersection(circle_center,
radius,get_all_incenters[tr_md_by_inc[i][0]],get_all_incenters[tr_md_by_inc[i][1]])

        get_all_incenters[tr_md_by_inc[i][2]]=find_intersection(circle_center,
radius,get_all_incenters[tr_md_by_inc[i][0]],get_all_incenters[tr_md_by_inc[i][2]])
```

```
    #[0,1,0]
    if ((check_tr_in[i][0]==0)and(check_tr_in[i][1]==1)and(check_tr_in[i][2]==0)):
        get_all_incenters[tr_md_by_inc[i][0]]=find_intersection(circle_center,
radius,get_all_incenters[tr_md_by_inc[i][1]],get_all_incenters[tr_md_by_inc[i][0]])
        get_all_incenters[tr_md_by_inc[i][2]]=find_intersection(circle_center,
radius,get_all_incenters[tr_md_by_inc[i][1]],get_all_incenters[tr_md_by_inc[i][2]])
    #[0,0,1]
    if ((check_tr_in[i][0]==0)and(check_tr_in[i][1]==0)and(check_tr_in[i][2]==1)):
        get_all_incenters[tr_md_by_inc[i][0]]=find_intersection(circle_center,
radius,get_all_incenters[tr_md_by_inc[i][2]],get_all_incenters[tr_md_by_inc[i][0]])
        get_all_incenters[tr_md_by_inc[i][1]]=find_intersection(circle_center,
radius,get_all_incenters[tr_md_by_inc[i][2]],get_all_incenters[tr_md_by_inc[i][1]])
    #[1,1,0]
    if ((check_tr_in[i][0]==1)and(check_tr_in[i][1]==1)and(check_tr_in[i][2]==0)):
        get_all_incenters[tr_md_by_inc[i][2]]=find_intersection(circle_center,
radius,get_all_incenters[tr_md_by_inc[i][1]],get_all_incenters[tr_md_by_inc[i][2]])
    #[0,1,1]
    if ((check_tr_in[i][0]==0)and(check_tr_in[i][1]==1)and(check_tr_in[i][2]==1)):
       get_all_incenters[tr_md_by_inc[i][0]]=find_intersection(circle_center,
radius,get_all_incenters[tr_md_by_inc[i][1]],get_all_incenters[tr_md_by_inc[i][0]])

    #[1,0,1]
    if ((check_tr_in[i][0]==1)and(check_tr_in[i][1]==0)and(check_tr_in[i][2]==1)):
        get_all_incenters[tr_md_by_inc[i][1]]=find_intersection(circle_center,
radius,get_all_incenters[tr_md_by_inc[i][0]],get_all_incenters[tr_md_by_inc[i][1]])


triangles=[['a','b','c']for _ in range(len(tr_md_by_inc))]
#print('triangles',triangles)
for i in range(len(tr_md_by_inc)):
    triangles[i][0]=get_all_incenters[tr_md_by_inc[i][0]]
    triangles[i][1]=get_all_incenters[tr_md_by_inc[i][1]]
    triangles[i][2]=get_all_incenters[tr_md_by_inc[i][2]]

cleaned_triangles = []
for triangle in triangles:
    cleaned_triangle = []
    for vertex in triangle:
        if isinstance(vertex, list) or isinstance(vertex, tuple):
            cleaned_triangle.append(tuple(vertex[0]) if isinstance(vertex[0], tuple) else tuple(vertex))
    cleaned_triangles.append(cleaned_triangle)


fig, ax = plt.subplots()
circle = plt.Circle((circle_center), radius, color='purple', fill=False, linewidth=1, label='Круг')  #
Визначаємо круг
plt.gca().add_artist(circle)
for triangle in cleaned_triangles:
    x_coords = [v[0] for v in triangle] + [triangle[0][0]]  # Додаємо першу точку для замикання
трикутника
    y_coords = [v[1] for v in triangle] + [triangle[0][1]]
```

```
    ax.plot(x_coords, y_coords, marker='.',linestyle='-', color='b',linewidth=0.1,)  # Лінії з точками на вершинах
    #plt.pause(2)

ax.set_aspect('equal')
plt.xlabel('X')
plt.ylabel('Y')
plt.title('Триангуляція')

plt.show()


#частина з Делоне:


# Функція для знаходження сусідніх трикутників
def find_adjacent_triangles(triangles, target):
    adjacent = []
    target_edges = [
        {target[0], target[1]},
        {target[1], target[2]},
        {target[2], target[0]}
    ]
    for triangle in triangles:
        if triangle == target:
            continue
        triangle_edges = [
            {triangle[0], triangle[1]},
            {triangle[1], triangle[2]},
            {triangle[2], triangle[0]}
        ]
        common_edges = [list(edge) for edge in target_edges if edge in triangle_edges]
        if common_edges:
          adjacent.append((triangle, common_edges))
    return adjacent

# Функція для обчислення центру та радіусу описаної окружності
def circumcircle(triangle, coordinates):
    x1, y1 = coordinates[triangle[0]]
    x2, y2 = coordinates[triangle[1]]
    x3, y3 = coordinates[triangle[2]]

    # Обчислюємо окружність через систему рівнянь
    D = 2 * (x1 * (y2 - y3) + x2 * (y3 - y1) + x3 * (y1 - y2))
    if D == 0:
        raise ValueError("Точки трикутника вироджуються в лінію.")

    Ux = ((x1**2 + y1**2) * (y2 - y3) + (x2**2 + y2**2) * (y3 - y1) + (x3**2 + y3**2) * (y1 - y2)) / D
    Uy = ((x1**2 + y1**2) * (x3 - x2) + (x2**2 + y2**2) * (x1 - x3) + (x3**2 + y3**2) * (x2 - x1)) / D
```

```
    radius = math.sqrt((x1 - Ux)**2 + (y1 - Uy)**2)
    return (Ux, Uy), radius

def is_point_strictly_in_circle(point, circle_center, radius):
    px, py = point
    #print('point',point)
    cx, cy = circle_center

    radius = round(radius,4)
    distance = round(math.sqrt((px - cx)**2 + (py - cy)**2),4)

    return distance < radius  # Умова строго для внутрішньої частини

# Функція для перевірки другого трикутника
def check_second_triangle_in_circle(first_triangle, second_triangle, coordinates):
    # Знаходимо окружність першого трикутник
    circle_center, radius = circumcircle(first_triangle, coordinates)
    ####print(f"Центр окружності: {circle_center}, Радіус: {radius}")
    ans=0
    # Перевіряємо кожну вершину другого трикутника
    for vertex in second_triangle:
        point = coordinates[vertex]
        if is_point_strictly_in_circle(point, circle_center, radius):
          ####print(f"Вершина {vertex} з координатами {point} знаходиться ВНУТРІ
окружності.")
            ans=1
        else:
            asdas=0
            ####print(f"Вершина {vertex} з координатами {point} знаходиться НА або ЗА
окружністю.")


    if ans==1:
        return(True)
    else:
        return(False)
def split_triangles_by_common_edge(triangle1, triangle2, common_edge):
    # Находим координаты, которые не входят в общую сторону
    unique_points_triangle1 = [point for point in triangle1 if point not in common_edge]
    unique_points_triangle2 = [point for point in triangle2 if point not in common_edge]

    if len(unique_points_triangle1) != 1 or len(unique_points_triangle2) != 1:
        raise ValueError("Трикутники задані некоректно або мають більше однієї унікальної
координати.")
    new_triangles_array=[]
    # Формуємо два нових трикутника
    new_triangle1 = [common_edge[0], unique_points_triangle1[0], unique_points_triangle2[0]]
    new_triangle2 = [common_edge[1], unique_points_triangle1[0], unique_points_triangle2[0]]
    new_triangles_array.append(new_triangle1)
    new_triangles_array.append(new_triangle2)
    return new_triangles_array
```

```
def triangle_exists(triangle, triangles_list):
    # Перетворюємо вхідний трикутник в відсортований вигляд
    sorted_triangle = sorted(triangle)
        # Перетворюємо кожен трикутник зі списку в відсортований вигляд і перевіряємо
    return any(sorted_triangle == sorted(t) for t in triangles_list)


target_triangle = tr_md_by_inc[0]  # [1, 3, 19]
k=0
count=0
n=0 # позиція в трикутниках змінюється тільки якщо всередині описаної окр. немає інших точок
while len(tr_md_by_inc)-k !=0:
    target_triangle = tr_md_by_inc[n]

    adjacent_triangles = find_adjacent_triangles(tr_md_by_inc, target_triangle)
    k+=1
    for triangle,edges in adjacent_triangles:
        first_triangle = target_triangle  # [1, 3, 19]
        second_triangle = triangle  # [3, 8, 19]
        if(check_second_triangle_in_circle(first_triangle, second_triangle, get_all_incenters))==True:
            new_triangles=split_triangles_by_common_edge(first_triangle, second_triangle, edges[0])
            tr_md_by_inc.pop(0)
            tr_md_by_inc.remove(second_triangle)
            if triangle_exists(new_triangles[0], tr_md_by_inc):
                print('трикутник',new_triangles[0],'вже входить в tr_md_by_inc')

            else:

                tr_md_by_inc.append(new_triangles[0])

            if triangle_exists(new_triangles[1], tr_md_by_inc):
                print('трикутник',new_triangles[1],'вже входить в tr_md_by_inc')
            else:

                tr_md_by_inc.append(new_triangles[1])


            break
        else:
            count+=1
        if count==3:
            n+=1

        count=0
    a=1
    if a=='stop':
        break
    if k==int(len(tr_md_by_inc)/2):
        break
```

```
    #k+=1

triangles=[[‘a’,’b’,’c’]for _ in range(len(tr_md_by_inc))]
#print(‘triangles’,triangles)
for i in range(len(tr_md_by_inc)):
    triangles[i][0]=get_all_incenters[tr_md_by_inc[i][0]]
    triangles[i][1]=get_all_incenters[tr_md_by_inc[i][1]]
    triangles[i][2]=get_all_incenters[tr_md_by_inc[i][2]]


cleaned_triangles = []
for triangle in triangles:
    cleaned_triangle = []
    for vertex in triangle:
        if isinstance(vertex, list) or isinstance(vertex, tuple):
            cleaned_triangle.append(tuple(vertex[0]) if isinstance(vertex[0], tuple) else tuple(vertex))
    cleaned_triangles.append(cleaned_triangle)

fig, ax = plt.subplots()

circle = plt.Circle((circle_center), radius, color=‘purple’, fill=False, linewidth=1, label=‘Круг’)  # Определяем круг
plt.gca().add_artist(circle)

#circle2 = plt.Circle((circle_center2), radius2, color=‘purple’, fill=False, linewidth=0.2, label=‘Круг2’)  # Определяем круг
#plt.gca().add_artist(circle2)


for triangle in cleaned_triangles:
    x_coords = [v[0] for v in triangle] + [triangle[0][0]]  # Добавляем первую точку для замыкания треугольника
    y_coords = [v[1] for v in triangle] + [triangle[0][1]]
    ax.plot(x_coords, y_coords, marker=‘.’,linestyle=‘-’, color=‘b’,linewidth=0.1,)  # Линии с точками на вершинах
    #plt.pause(2)


ax.set_aspect(‘equal’)
plt.xlabel(‘X’)
plt.ylabel(‘Y’)
plt.title(‘Триангуляція Делоне’)


plt.ioff()
plt.show()
```