Науково-практичний звіт на тему

Тріангуляція Просто Многокутника

О. О. Хоменко, студент 3 курсу, групи Комп’ютерна Математика 2

**Анотація.** У роботі запропоновано метод тріангуляції простого многокутника.

**1 Вступ**

Постановка проблеми. В роботі розглядається один із підходів розв’язання задачі тріангуляції простого многокутника під назвою «Метод відтину вух» з певними удосконаленнями.   
Тріангуляція многокутника - це розбиття многокутника на трикутники, що мають спільні ребра з умовою, що множина вершин трикутників збігається з множиною вершин многокутника. Тріангуляція многокутників є основою багатьох важливих геометричних алгоритмів, наприклад просте рішення задачі галереї мистецтв.

Перевірку належності точки простому многокутнику можна розбити на задачі перевірки належності точки якомусь трикутнику із тріангуляції даного просто многокутника, що легше реалізувати.

Тріангуляція використовується в мережах, комп'ютерному баченні, різних аналітичних алгоритмах, а також в деяких задачах обчислювальної геометрії, як об'єднання і виключення багатокутників один з одного (що часто корисно в задачах, що виникають при розробці ігор).

Аналіз останніх досліджень.

Перше тривіальне рішення:

У випадку довільного n-кутнику всього n^2 можливих варіантів побудови діагоналей.   
За O(n) перевіримо кожен із них. Для цього з'ясуємо:

чи перетинає дана діагональ багатокутник - знаходиться за лінійний час перевіркою по всіх ребрах

чи належить діагональ внутрішньої області багатокутника.

Щоб побудувати тріангуляцію, потрібно знайти n−3

діагоналей.

Вихідна складність O(n^4).

Метод відтину вух:

Знайти вухо за O(n), кожне вухо відповідає за трикутник у тріангуляції: при загальній кількості трикутників у будь-якій тріангуляції (n-2), отримаємо вихідну складність O(n^2).  
[2]Теорема1  
У будь-якого простого n-вершинного багатокутника P завжди існує тріангуляція, причому кількість трикутників у ній n−2 незалежно від самої тріангуляції.

Вихідна складність O(n^2).

Монотоний метод:

Розбиття багатокутника на монотонні частини займає O(n \* log(n)) часу. Тріангуляція кожної з частин займає лінійний час. Враховуючи те, що сумарна кількість вершин у всіх частинах O(n), тріангуляція всіх частин займе O(n) за часом.

Вихідна складність O(n \* log(n)).

Метод розбиття многокутника на монотонні частини з подальшою тріангуляцією кожної частини:

Розбиття многокутника на монотонні частини чи регуляризація відбувається за   
O(n \* log(n)). В подальшому тріангуляція монотонного многокутника за O(n).

Вихідна складність O(n \* log(n)).

Тріангуляція Делоне:  
Вихідна складність O(n \* log(n)).

Метод тріангуляції трапецієподібне розкладання:

Вихідна складність O(n \* log(n)).

Також існують алгоритми тріангуляції за лінійний чи майже лінійний час:  
Raimund Seidel triangulation algorithm O(n \* c), c=const

**2. Основна частина.**

**Постановка задачі тріангуляції простого многокутника.** Нехай заданий просто многокутник. Необхідно розбити його на трикутники(провести тріангуляцію простого многокутника) без додавання нових точок.

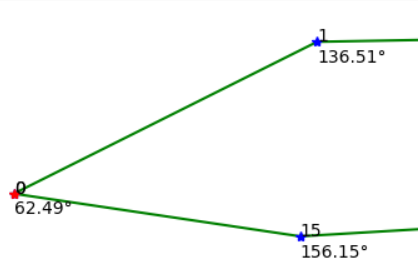
**2.1. Побудова розв’язку задачі тріангуляції простого многокутника.**

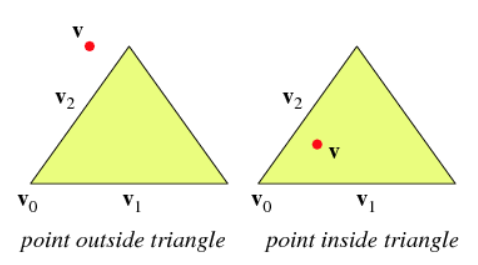
[6]У геометрії Теорема про два вуха стверджує, що кожен простий багатокутник з більше трьох вершин має не менше двох вух , вершини, які можуть бути видалені з багатокутника без введення будь-яких перетинів.

[2]Вухо це таких кут в многокутнику, який задовольняє певним умовам:  
1. Кут < 180 градусів

2. Побудований на цьому куті трикутник не містить жодної вершини многокутника.

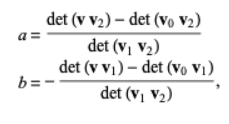
(або трикутник лежить всередині многокутника)





[5]Точка лежить в середины трикутника, якщо задовольняються наступні нерівності:

a > 0, b > 0, a + b < 1, де



, де



Для того щоб побудувати тріангуляцію будемо перебирати вершини многокутника, перевіряючи чи є кут при цій вершину вухом,

якщо так, видаляємо її разом з суміжними їй ребрами з многокутника.

Продовжуємо поки не залишиться три вершини.

Для перевірки точки на те, чи є вона вухом знадобляться певні структури даних:

1) масив відсортованих точок по x координаті,

2) масив відсортованих точок по y координаті.

Перевірка кута на те, чи є вона вухом буде проходити наступним чином:

1. Якщо кут >= 180 градусів, кут не є вухом,
2. Якщо ребро на якому стоїть кут, належить також іншому куту, який >= 180 градусів(ребро є зовнішнім ребром многокутника), кут не є вухом,
3. Якщо ж перша та друга умові не виконуються, побудуємо трикутник на даному куті,
4. Побудуємо запитний регіон чи іншими словами прямокутник, навколо нашого трикутника, такий що,

верхня ліва вершина прямокутника буде мати координати x та y,

які будуть мінімальною та максимальною координатами, точок трикутника, відповідно по x та y,(назвемо цю вершину ліва верхня чи ЛВ)

а права нижня вершина буде мати координати x та y,

які будуть максимальною та мінімальною координатами, точок трикутника, відповідно по x та y. (назвемо цю вершину права нижня чи ПН)

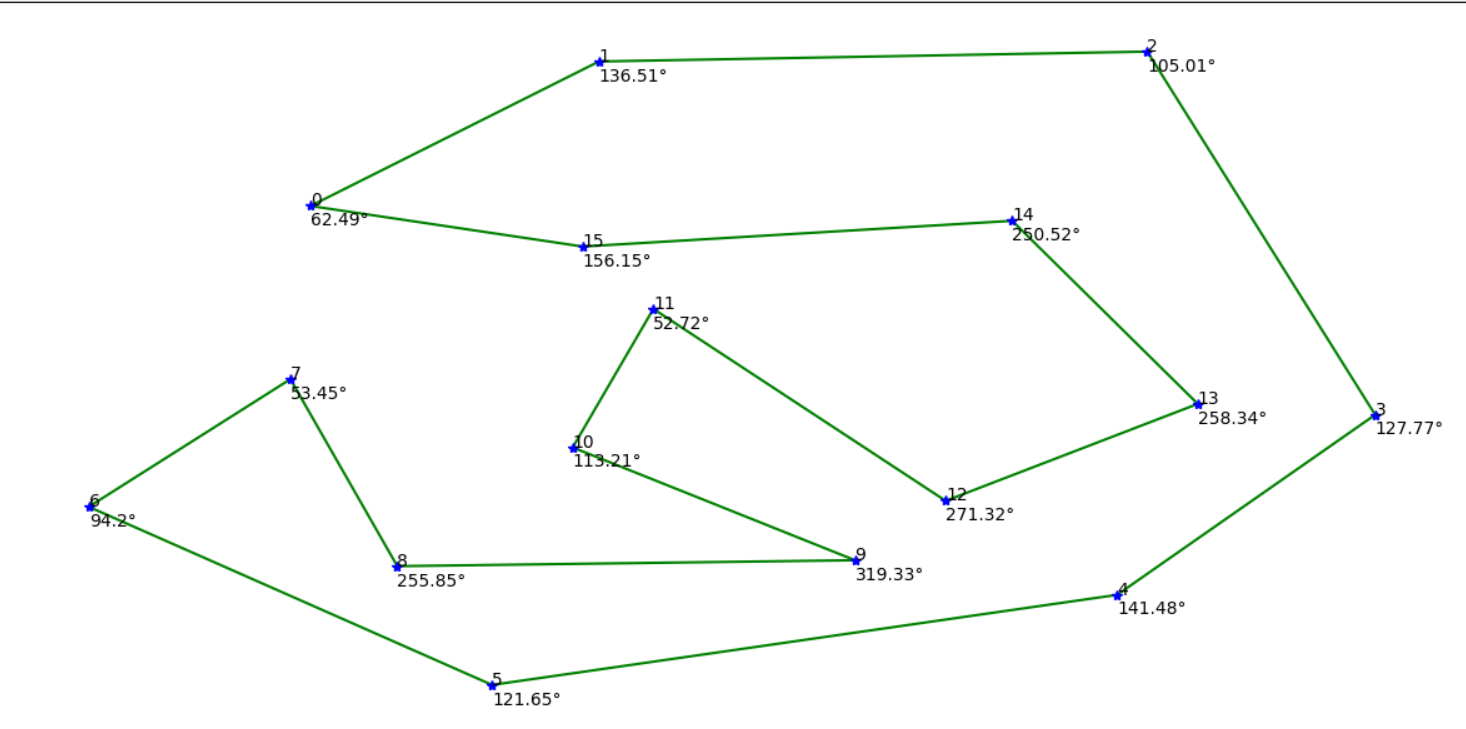
1. Використовуючи відсортовані списки точок по x та y, знайдемо, за майже логарифмічний час, точки многокутника, які належать запитному регіону,
2. Проходимося по точках запитного регіону доти доки не знайдемо точку, яка лежить в нашому трикутнику, якщо таких точок не знайшлося, кут є вухом.

**Алгоритм.**

Побудуємо масив всіх точок, записаних до масиву за годинниковою стрілку (Pts) та множину ребер многокутника (Edges).

Створимо поки порожню множину зовнішніх ребер (ExtEdges).

Пройдемося по цій множині точок та запишемо в кожну точку її внутрішній кут у многокутнику.

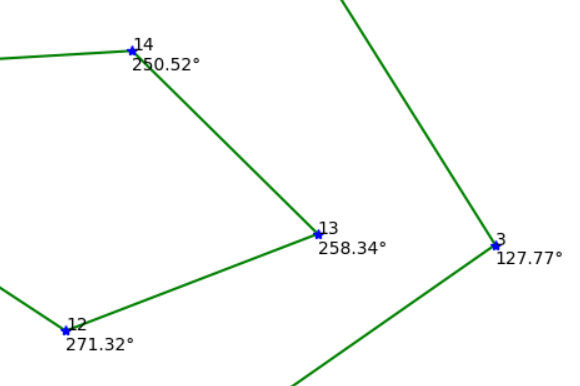


//під множиною ребер мається на увазі будь-яка похідна структура даних від хеш-таблиці, в якій вставка та пошук елемента відбуваються за O(1).//

Побудуємо масив відсортованих точок по x координаті (XPts) та масив відсортованих точок по y координаті (YPts).

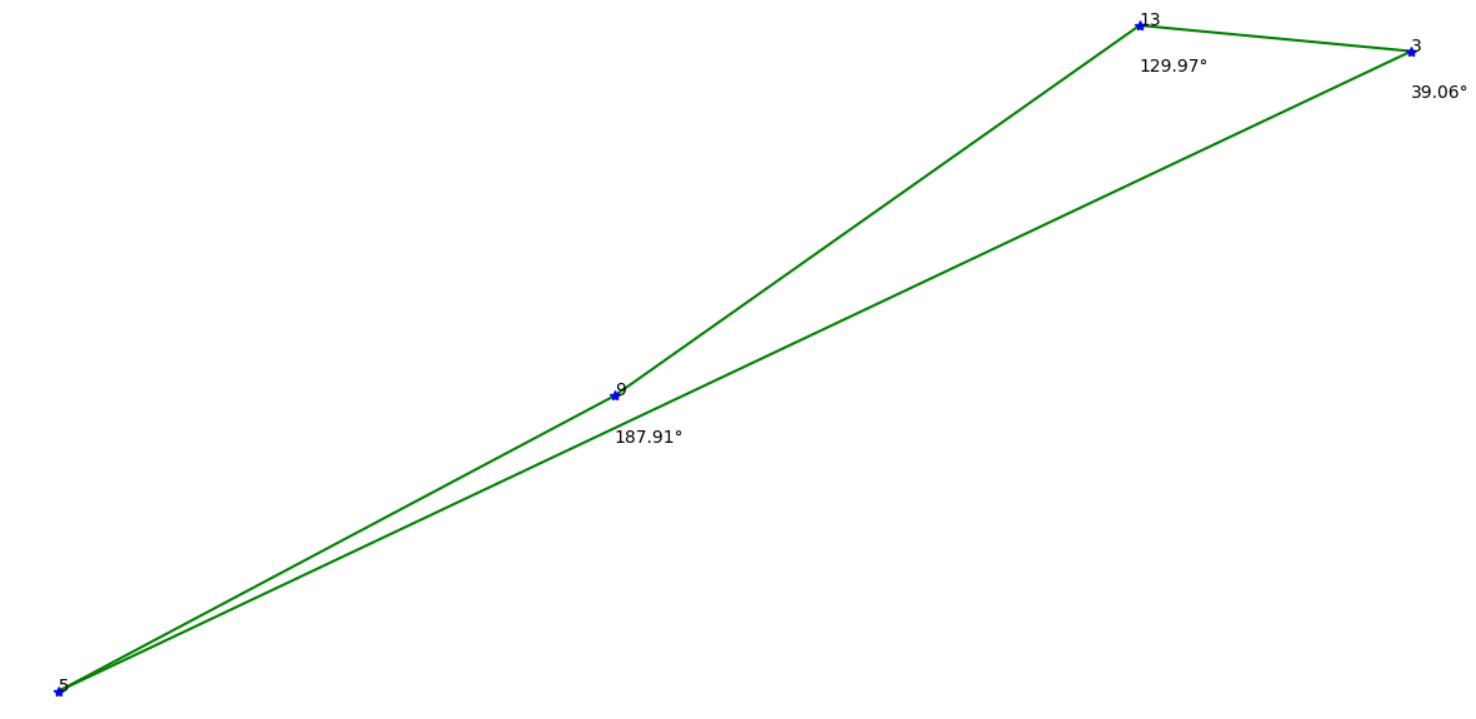
Почнемо перебирати точки по індексу в масиві Pts, класифікуючи їх за вже заданими правилами:

1. Якщо кут >= 180 градусів, кут не є вухом, добавляємо в ExtEdges ребро, на якому стоїть даних кут, беремо наступну вершину,



Вершина 13 на рисунку має внутрішній кут в многокутнику > 180 градусів.

1. Якщо ребро належить ExtEdges, кут не є вухом, беремо наступну вершину



Кут Вершини 3 лежить на ребрі 5-13, яке також є ребром кута при вершині 9, який > 180 градусів.

1. Якщо ж перша та друга умові не виконуються, побудуємо трикутник на даному куті,

Побудуємо запитний регіон, який буду задаватися вершинами ЛВ та ПН.

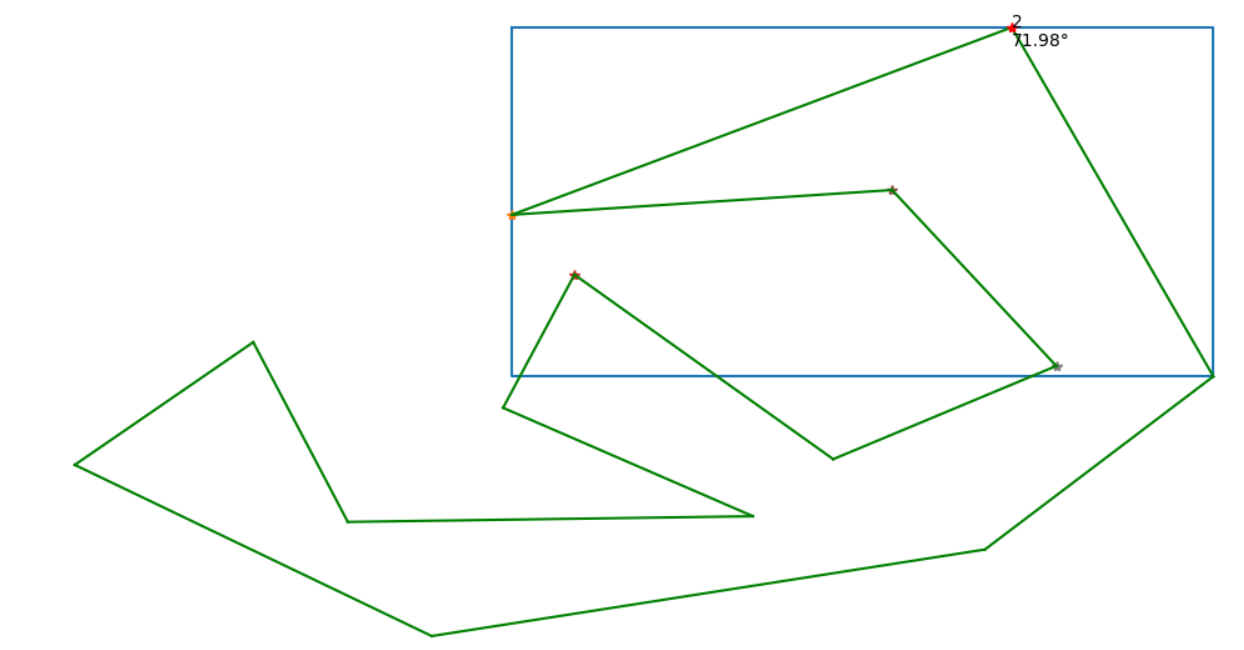
Використовуючи бінарний пошук знайдемо першу точку із XPts, яка більша ЛВ\_х та останню точку із XPts, яка менша ПН\_х. O(log(n))

Запишемо знайдені вершини у список. Проходимося по списку викидаючи вершини, які більші ЛВ\_у та точки, які менші ПН\_у.

Отримали вершини запитного регіону.

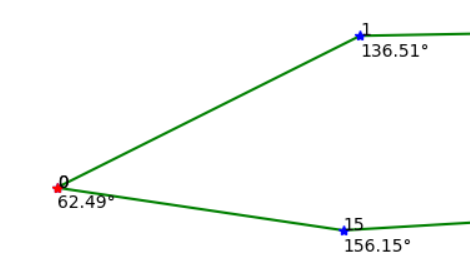
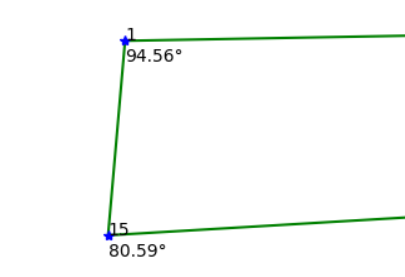
Проходимося по вершинах запитного регіону доти, доки не знайдемо вершину, яка лежить в нашому трикутнику,

Якщо така точка знайшлася, кут не є вухом: беремо наступну вершину,



Якщо таких точок не знайшлося, кут є вухом: видаляємо вершину з даним кутом з Pts та видаляємо суміжні вершині ребра з Edges, беремо наступну

Вершину

//оскільки ми перебирали індекс точки видалення відбувається за O(n).

Метод відтину вух проходить по кожній точці ( O(n) ) многокутника та будує третє відсутнє ребро для трикутника для кута в даній точці.

Потім перебирає всі ребра многокутника ( O(n) ) у пошуку перетину,

-якщо перетин знайдений - кут не є вухом;

-якщо перебравши всі ребра, так і не знайшлось перетину, - кут є вухом, тому ми видаляємо два суміжні цій точці ребра з масиву ребер, а саму точку видаляємо з масиву точок.

Переходимо до перевірки наступної точки.

Таким чином загальна складність виходить близькою до O(n \* log(n)).

**3. Обґрунтування складності**

1) Виміряти кут при кожній вершині -> O(n).

2) Створити та відсортувати масиви XPts, YPts ->

O(n) + O(n \* log(n)) = O(n \* log(n)), така сам складність, як і у алгоритма швидкого сортування, яким ці масиви ми і будемо сортувати.

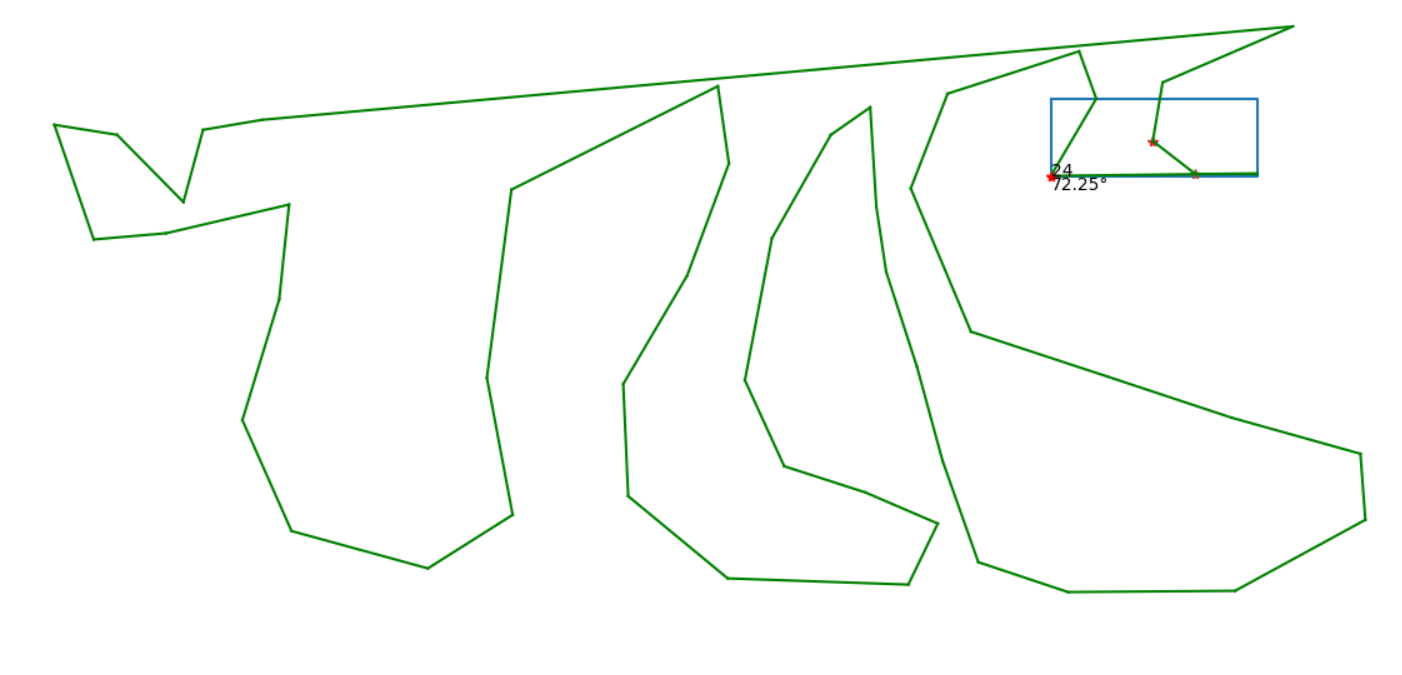
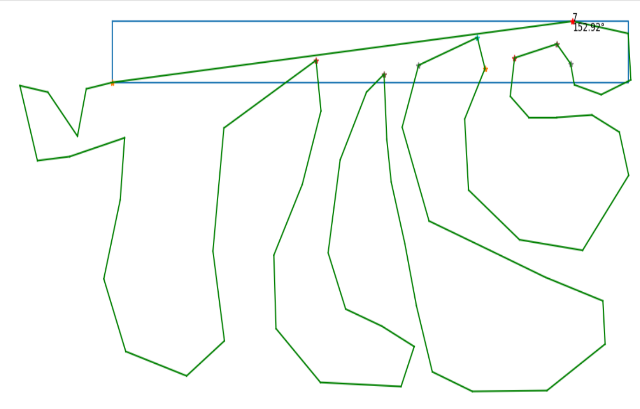
3) Перебір вершин многокутника -> O(n).

Для кожної вершини многокутника ми повинні визначити є кут при цій вершині вухом, чи ні.

//

Складність функції з класифікації(чи є вухом) вершини:

1. Якщо кут при вершині >= 180 -> O(1),
2. Якщо кут при вершині спирається на зовнішнє ребро, -> O(1),
3. Якщо не 1 і не 2, то:
4. Знайти ЛВ і ПН -> O(1)
5. Спроектувати запитний регіон на XPts –> O(log2(n)) (складність бінарного пошуку)
6. Виділити вершини з попередньої множини, які б належали YPts, буде мати зазвичай малу складність, бо ми проходимося по вершинам, які виділив запитний регіон по х координаті, яких буде сильно менше, ніж усіх вершин многокутника. -> O(log(n)) || O(1)



Але можливе виродження -> O(n).

1. Далі потрібно пройтись по вершинам запитного регіону та дізнатися чи належать трикутнику побудованому на куті при нашій вершині.
2. Як і в попередньому випадку зазвичай Операція буде мати малу складність -> O(log(n)) || O(1).

Але можливе виродження -> O(n).

//

Сумарно 1,2 -> O(n \* log(n)).

3 пункт -> O(n).

4 пункт зазвичай буде давати -> O(log(n)),

Але можливе виродження до O(n).

Тож 3 з 4 пунктом зазвичай будуть давати -> O(n \* log(n)),

Але можливе виродження до -> O(n^2).

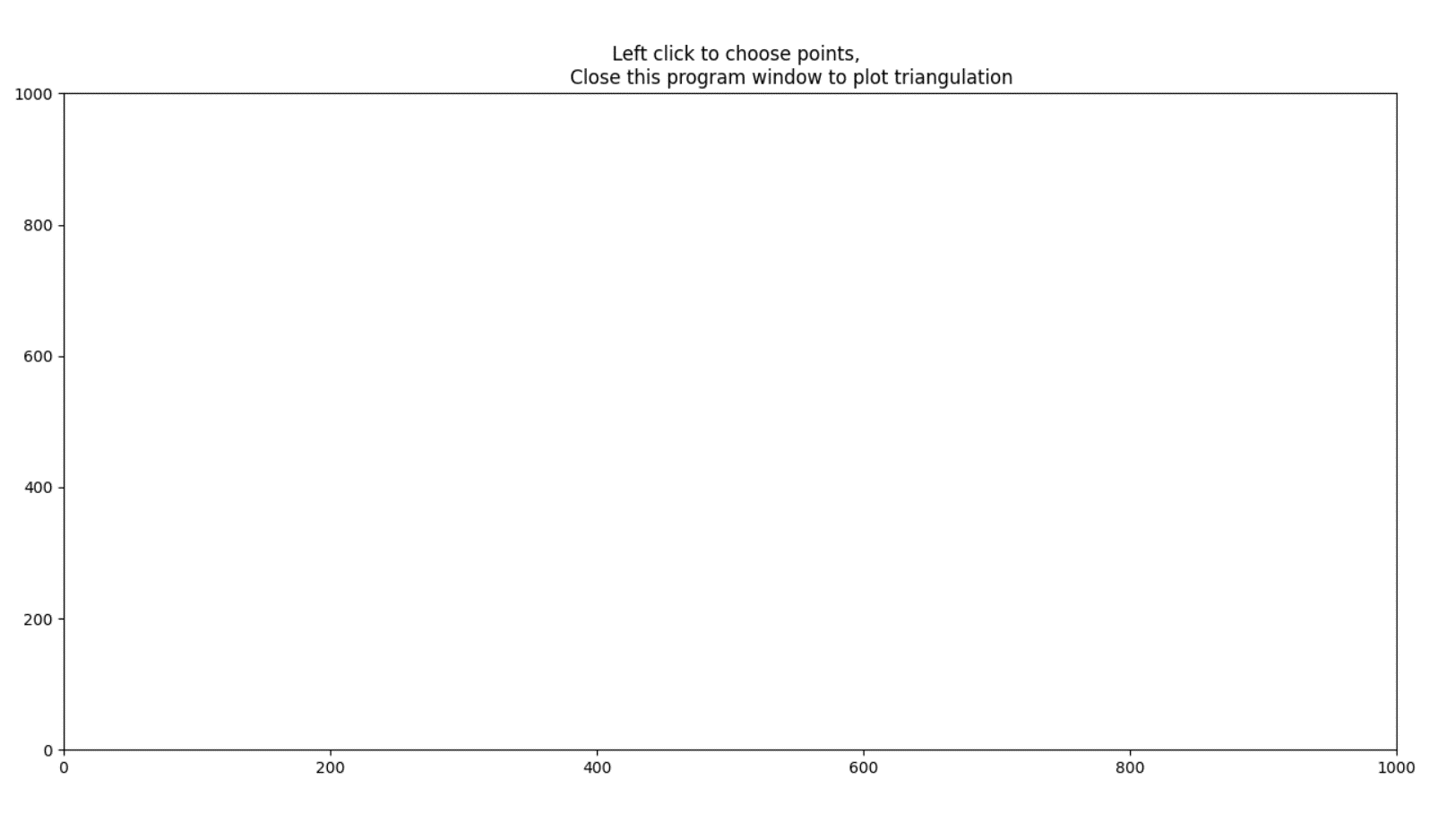
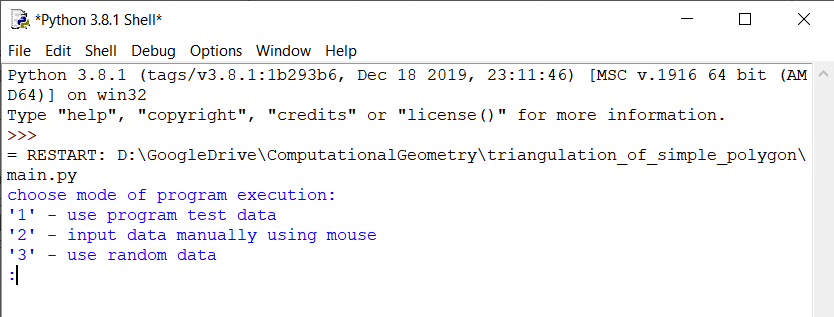
Загальна складність зазвичай буде -> O(n \* log(n)),

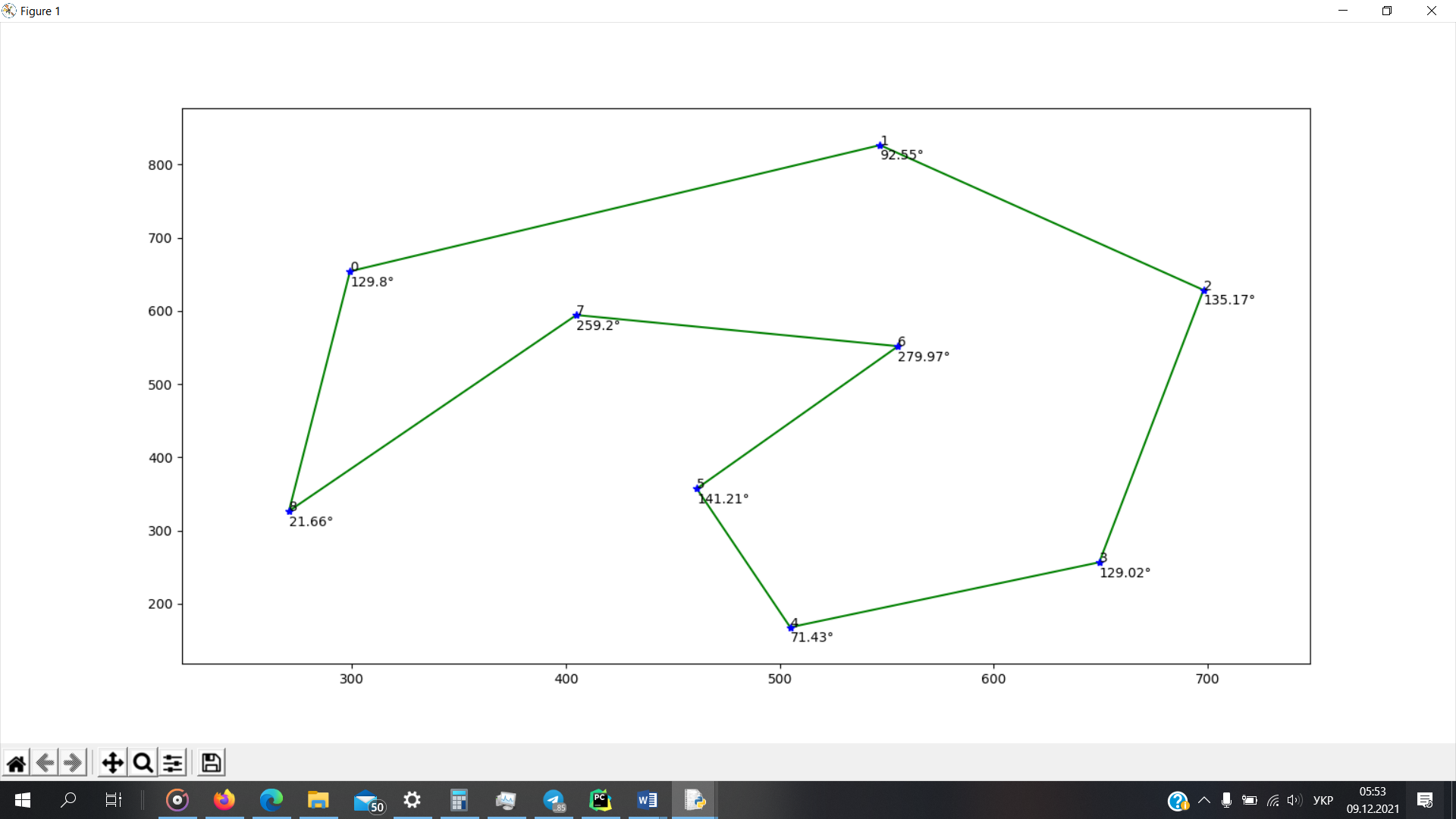
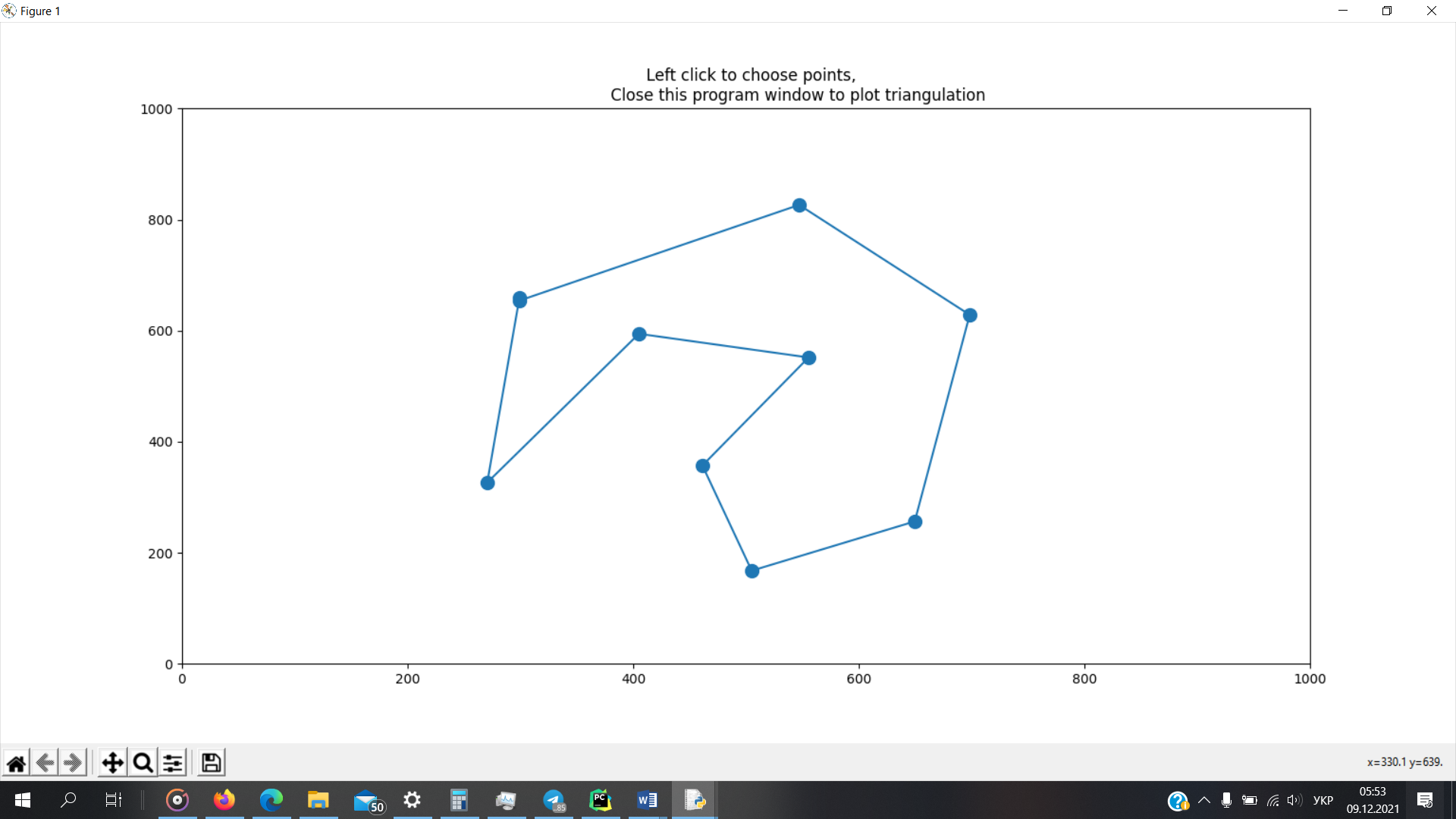
Але можливе виродження до -> O(n^2), що є складністю роботи звичайного алгоритму тріангуляції методом відтину вух.

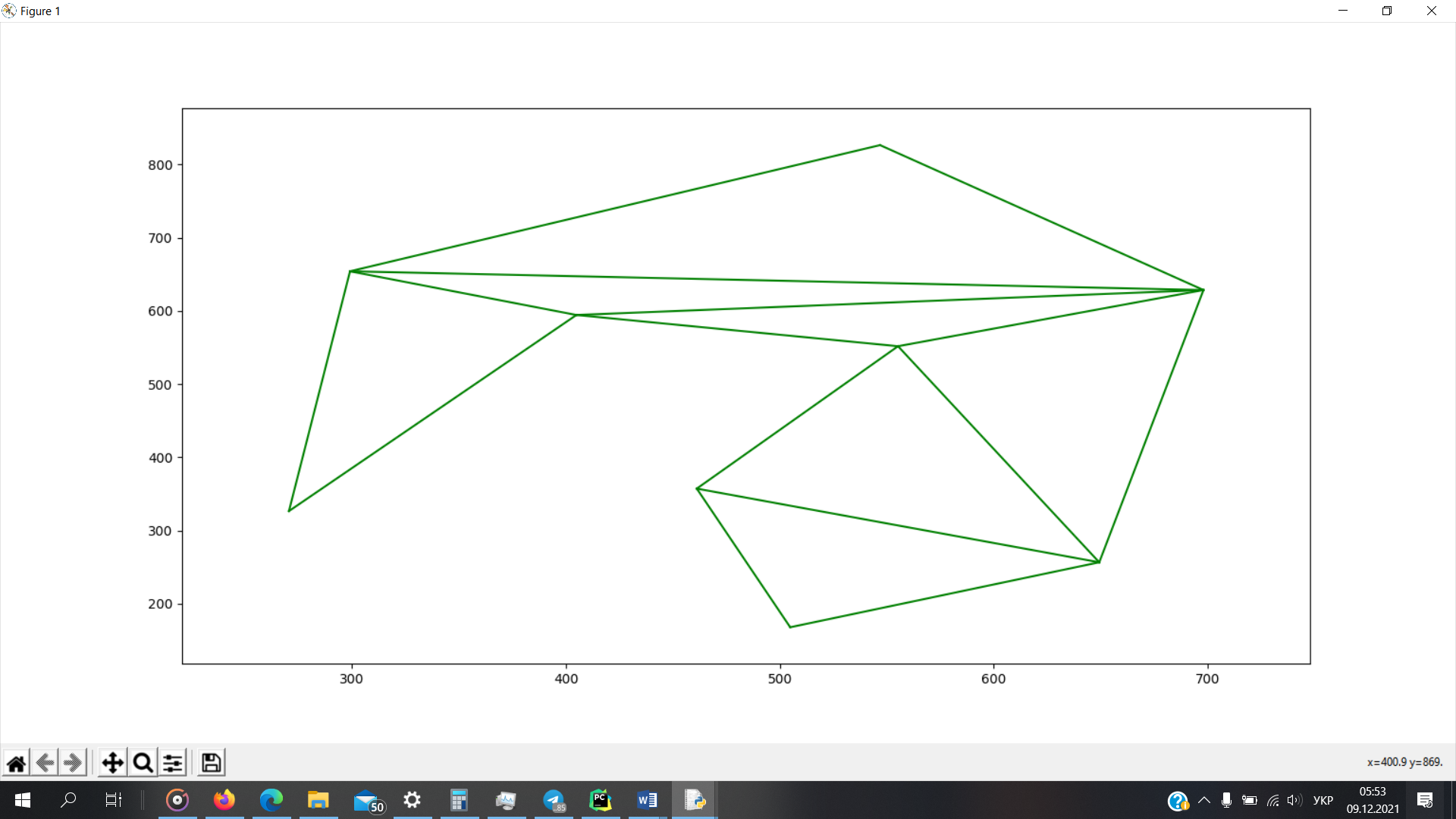
**4. Практична частина**

На рисунках знизу ілюструється інтерфейс програми.

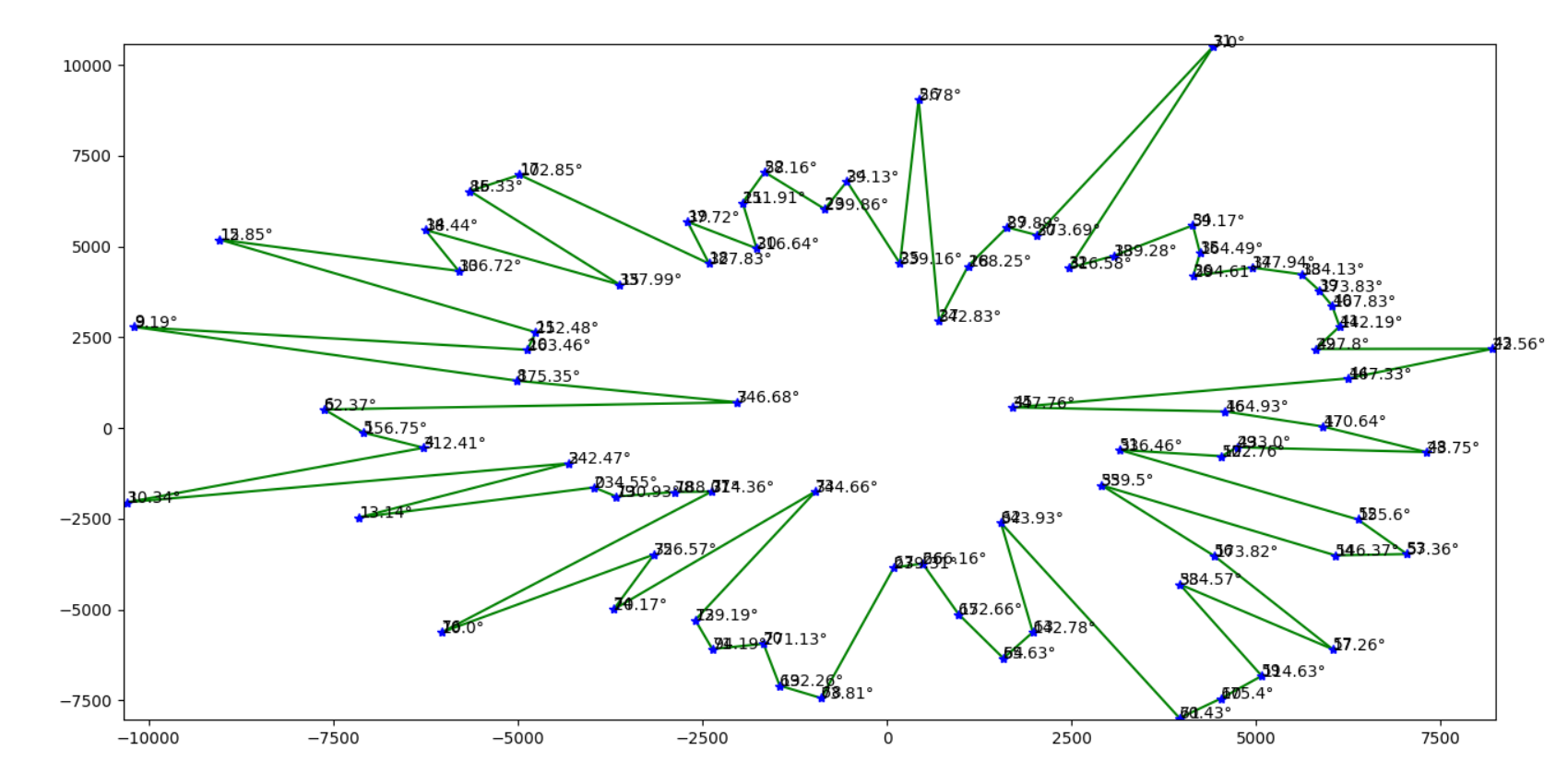
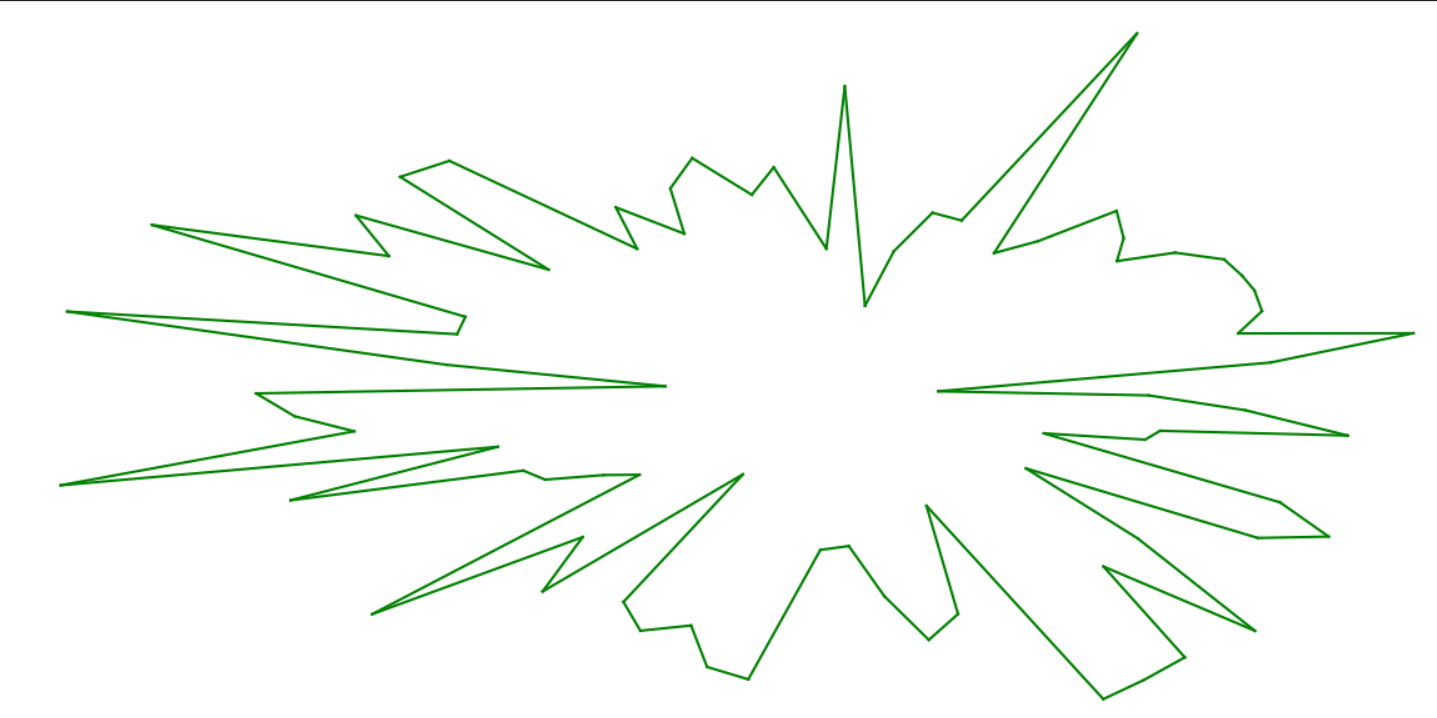
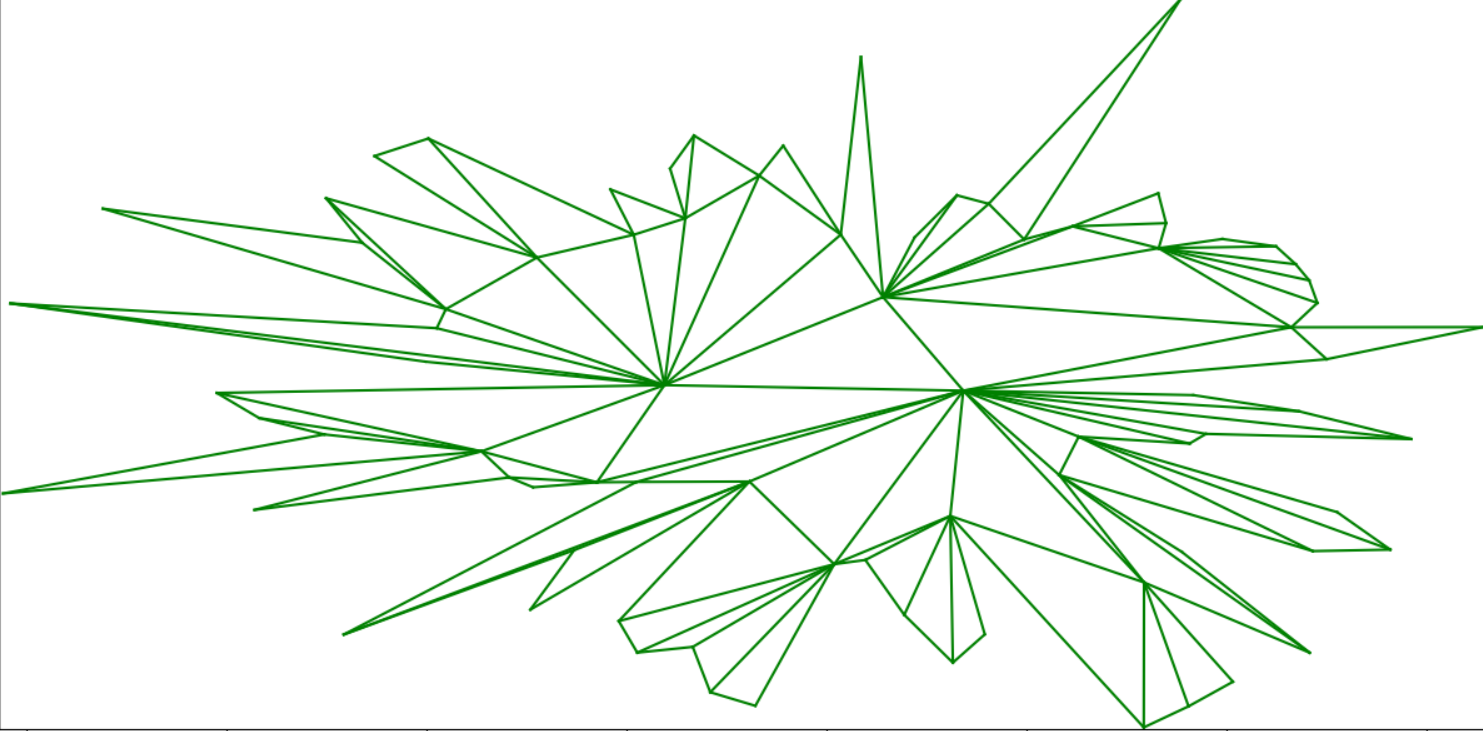
Ввід данних вручну.





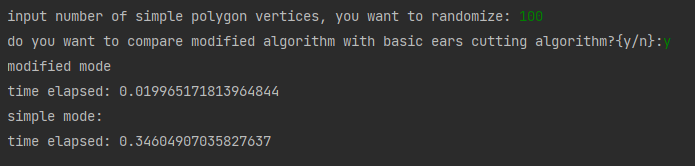
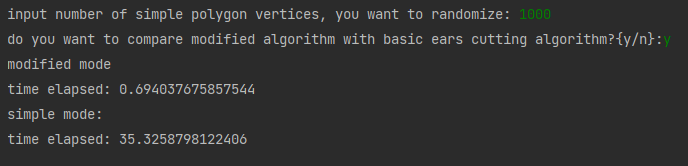


Тріангуляція випадкового просто многокутника.

Алгоритм тріангуляція був написаний на мові програмування Python.

Порівняння часу роботи з базовим алгоритмом тріангуляції методом відтину вух.

**5. Висновки**

У роботі запропонований покращений алгоритм для тріангуляції методом відтину вух, який має середню складність близьку до O(n \* log(n)),

але може виродитися до звичайної реалізації алгоритму вищезгаданим методом і тоді складність буде O(n^2).

**Список літератури**

1. <https://deparkes.co.uk/2015/02/05/trapezoidal-decomposition-polygons-python/>
2. <https://sites.cs.ucsb.edu/~suri/cs235/Triangulation.pdf>
3. <https://uk.погода-синоптик.pp.ua/1000328/1/triangulyatsiya-geometriya.html>
4. «Компьютерная Геометрия и Алгоритмы Компьютерной Графики» Е. А. Никулин
5. <https://en.wikipedia.org/wiki/Barycentric_coordinate_system>
6. <https://tftwiki.ru/wiki/Two_ears_theorem>