МІНІСТЕРСТВО ОСВІТИ І НАУКИ КРАЇНИ
Західноукраїнський національний університет
Фаультет комп'ютерних інформаційних технологій
Кафедра комп'ютерної інженерії

Бучинський Тарас Богданович

АВТОМАТИЧНА СЕГМЕНТАЦІЯ ЗОБРАЖЕНЬ НА ОСНОВІ МЕТРИКИ ХАУСДОРФА / AUTOMATIC IMAGE SEGMENTATION BASED ON HAUSDORF METRICS

спеціальність; 123 – Комп'ютерна інженерія
освітньо-професійна програма - Комп'ютерна інженерія

Кваліфікаційна робота

Виконав студент групи Кім-21
Т.Б. **Бучинський**
_____

Науковий керівник:
д.т.н., професор, О.М. Березький
_____

Кваліфікаційну роботу
допущено до захисту
«___» _____2021 р.

Завідувач кафедри КІ
О.М.Березький
_____

Тернопіль – 2021

ЗМІСТ

ВСТУП

Актуальність теми. Одним з основних завдань в галузі комп'ютерного зору є задача добування інформації з зображень. Для того щоб обробляти зображення, необхідно розбити його на елементи.

Сегментацією зображення називають розбиття зображення на області або сектори, що відрізняються один від одного за будь-якими ознаками. При вирішенні завдань обробки зображень та комп'ютерного зору, сегментація відіграє важливу роль.

Сегментація активно використовується при обробці медичних зображень, наприклад, для виявлення пухлин та інших патологій, визначення об’ємів тканин, різних діагностик, планування лікування; при виділенні об'єктів на супутникових знімках, розпізнавання осіб, розпізнавання відбитків пальців, в системах управління дорожнім рухом і т.д. [1-7].

Виділення границь різними алгоритмами мали б призводити до однакового результату, але оскільки на сегментацію впливає велика кількість різних факторів, жоден з алгоритмів не дає достатньо точної інформації про границі і області. Сегментація зазвичай використовується не самостійно, а як частина деякої системи (наприклад, системи машинного зору), тому якість роботи алгоритму оцінюється виходячи з роботи системи в цілому, один і той же алгоритм сегментації може виявитися хорошим для однієї задачі і поганим для іншої [1].

Особливо актуальна проблема оцінки якості зображень при розробці різних алгоритмів обробки зображень, так як часто перед фінальним тестуванням і порівнянням з аналогічними рішеннями потрібно провести ряд досліджень і експериментів по налаштуванню і оптимізації розроблювального алгоритму. Наприклад, при розробці алгоритмів сегментації зображень потрібно визначити наскільки відкинута інформація є суттєвою при сприйнятті [5, 6, 8].

Відомо, що найнадійнішим способом отримання оцінки якості зображень є використання експертного підходу, але його застосування вимагає великої кількості людей і часу, особливо у випадку великої кількості оцінюваних зображень. Ефективним рішенням зменшення трудомісткості і часу отримання оцінки якості є застосування об'єктивних (математичних) метрик, що дають максимально наближену до експертного підходу оцінку якості зображень [8]. Звідси, виникає задача розробки алгоритмів кількісної оцінки якості алгоритмів сегментації.

Для кількісної оцінки якості сегментації зараз існує велика кількість критеріїв. Ці критерії діляться на дві групи:

1) несупервізорні критерії. Ця група базується на обчисленні різного виду статистик і використовується при бракові попередньої інформації про сегменти зображень;

2) супервізорні критерії. Вони базуються на обчисленні міри відмінності результатів сегментації від справжньої форми об’єктів зображень. Форма об’єктів може задаватись експертами або вважатися відомою [1]. При кількісній оцінці якості сегментації за супервізорними критеріями застосовуються метричні простори.

Метрика Хаусдорфа дозволяє знайти відстань (подібність) між двома областями. Метрика Хаусдорфа широко використовуються для кількісної оцінки якості сегментації зображень.

Мета роботи. Метою роботи є розроблення алгоритму та програмного засобу для кількісної оцінки подібності зображень на основні метрики Хаусдорфа.

Для досягнення поставленої мети необхідно виконати такі завдання:

– проаналізувати методи оцінки якості зображень;

– проаналізувати програмні засоби кількісної оцінки алгоритмів сегментації;

– розробити алгоритми порівняння областей зображень в метриці Хаусдорфа;

– здійснити програмну реалізацію системи;

– провести тестування функціональних вимог системи;

– здійснити комп’ютерні експерименти.

Об'єкт дослідження – аналіз подібності зображень.

Предмет дослідження – алгоритми порівняння областей зображень в метричних просторах.

Методи досліджень базуються на використанні алгоритмів комп’ютерного зору (алгоритмів сегментації зображень), теорії метричних просторів (алгоритмів кількісної оцінки якості сегментації зображень), методів контурного аналізу (алгоритми обходу контурів областей зображень), теорія алгоритмів (для оцінки обчислювальної складності), методів об’єктно-орієнтованого програмування (для реалізації програмного засобу).

Наукова новизна полягає у розробці алгоритмів порівняння областей зображень у метриці Хаусдорфа.

Практичне значення. Розроблено програмний засіб для порівняння областей зображень у метриці Хаусдорфа.

Публікації та апробація КР. Основні результати дослідження опубліковано на V-ій науково-практичної конференції молодих вчених і студентів «Інтелектуальні комп’ютерні системи та мережі», 2 грудня 2021 р., м. Тернопіль, Західноукраїнський національний університет:

1. Полагнюк І. В., Клімовський Д. Б., Вдодович О. В., Бучинський Т. Б. Сегментація зображень з використанням метричних мір. *Інтелектуальні комп’ютерні системи та мережі* : тези доп. V Наук.-практ. конф. молодих вчених і студентів (2 груд. 2021 р.). Тернопіль : ЗУНУ, 2021. С. 14.
2. Вдодович О. В., Клімовський Д. Б., Полагнюк І. В., Бучинський Т. Б. Алгоритми порівняння зображень в метричних просторах. *Інтелектуальні комп’ютерні системи та мережі* : тези доп. V Наук.-практ. конф. молодих вчених і студентів (2 груд. 2021 р.). Тернопіль : ЗУНУ, 2021. С. 11.

У вступі обґрунтовано актуальність і доцільність теми роботи, визначено мету та завдання, об’єкт, предмет і методи дослідження. Крім цього, наведено наукову новизну роботи та практичне значення одержаних результатів.

У першому розділі проаналізовано особливості біомедичних зображень, методи оцінки якості сегментації зображень, проведено аналіз методів кількісної оцінки сегментації зображень на основі метричних просторів, проаналізовано основні метричи, досліджено програмні засоби для обробки біомедичних зображень.

У другому розділі проаналізовано основні маски (оператори) для визначення контурів зображень, досліджено алгоритми обчислення відстані Хаусдорфа для багатокутників що перетинаються і не перетинаються, дсліджен алгоритми спрощення контурів, проведено аналіз алгоритмів визначення серпів і тріангуляції, розроблено алгоритми кількісної оцінки якості алгоритмів сегментації в метриці Хаусдорфа.

У третьму розділі сформовано вимоги до програмного засобу, розроблено архітектуру системи, проведено тестування функціональних вимог і комп’ютерні експерименти, а також порівняно конкуруючі алгоритми спрощення контурів і обчислення відстані у метриці Хаусдорфа.

У додатках наведено діаграму класів розробленого програмного засобу, код програми, додано світлокопії публікацій з конференції молодих вчених і студентів «Інтелектуальні комп’ютерні системи та мережі», а також довідку про використання результатів кваліфікаційної роботи.

1 АНАЛІЗ МЕТОДІВ, МЕТРИК І АЛГОРИТМІВ ДЛЯ КІЛЬКІСНОЇ ОЦІНКИ РЕЗУЛЬТАТІВ СЕГМЕНТАЦІЇ БІОМЕДИЧНИХ ЗОБРАЖЕНЬ

1.1 Аналіз біомедичних зображень

Цифровим зображенням називають масив даних, що отримується шляхом дискретизації (аналого-цифрового перетворення) оригіналу. Кодування цифрових зображень проходить за допомогою особливих алгоритмів, записується на носії інформації і цей масив даних стає файлом [9].

Зображення визначають як двовимірну функцію $f(x,y)$, де x і y – координати в просторі (конкретно, на площині), і значення $f(x,y)$ якої в будь-якій точці, з координатами (x,y), називається інтенсивністю або рівнем сірого зображення в цій точці. Деколи цю властивість називають ще й яскравістю точки. Якщо величини x, y, і $f(x,y)$ приймають скінчене число дискретних значень, то йде мова про цифрове зображення. Цифровою обробкою зображень називається обробка цифрових зображень за допомогою цифрових обчислювальних машин (комп’ютерів). Цифрове зображення складається із скінченої кількості елементів, кожен з яких розташований в конкретному місці і приймає визначене значення. Ці елементи називаються елементами зображення або пікселями [10].

Біомедичні технології застосовуються для отримання зображень, які використовуються у діагностичних і терапевтичних цілях. В природніх умовах фізіологічні знімки та фізіологічні процеси можуть бути отримані завдяки сучасним давачам та комп’ютерним технологіям. Біомедичні технології використовують або Х-промені (комп’ютерна томографія (КТ)), звукові (ультра звукова діагностика (УЗД)), магнетизм (магнітно-резонансна томографія(МРТ)), радіоактивні фармацевтичні препарати (ядерна медицина, одно фотонна емісійна комп’ютерна томографія(ОФЕКТ), позитрон-емісійна томографія(ПЕТ)) або світло (ендоскопія, оптична когерентна томографія (ОКТ)) для оцінки поточного

стану органу або тканини і може контролювати пацієнта протягом певного часу для діагностики і оцінки лікування [11].

Обробка біомедичних зображень аналогічна концепції обробки біомедичних сигналів в кількох вимірах. Вона включає в себе аналіз, покращення та відображення зображень, утворених за допомогою рентгенівських променів, ультразвуку, МРТ, ядерної медицини та технологій оптичної візуалізації [11].

Комп'ютерні алгоритми можуть надати тимчасову і просторовий аналіз для виявлення закономірностей і характеристик, що вказують пухлин та інших захворювань. В залежності від технології візуалізації і діагнозу який в даний момент визначається, обробка та аналіз зображень можуть використовуватись для визначення діаметру, об’єму і судинної мережі пухлини чи органу; поточні параметри крові чи інших рідин і мікроскопічних змін, які ще не виникли, але мають передумови для цього [11].

Сучасна медицина розрізняє такі біомедичні зображення: зображення цифрової мікроскопії (цитологічні, гістологічні зразки), рентгенограми, ультразвукові зображення, зображення магнітно-резонансної томографії та ін.

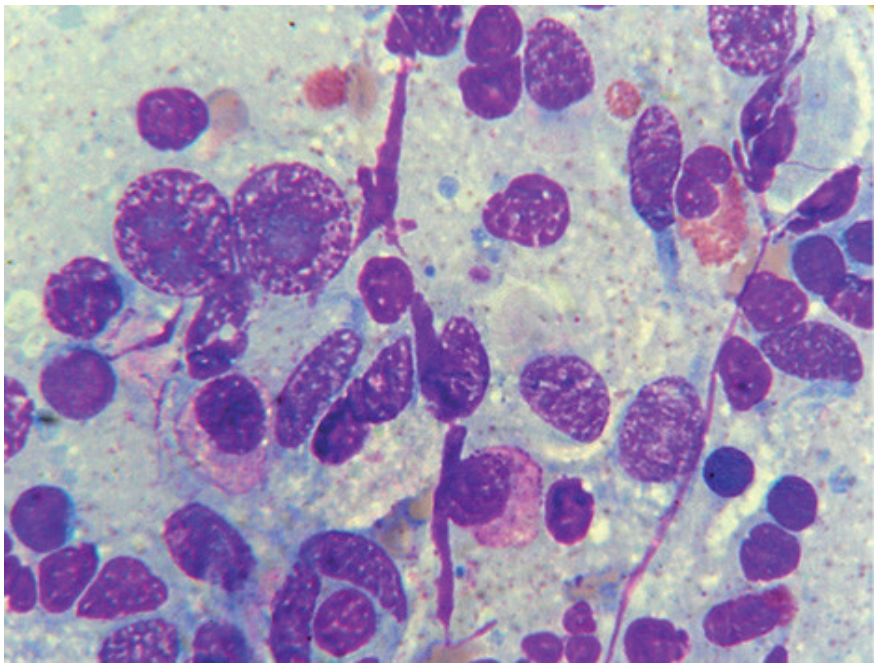


Рисунок 1.1 – Цитологічне зображення

Сучасна гістологія і цитологія досліджує нормальні і патологічні клітини і тканини, та їх зображення, отримані в світлових та електронних мікроскопах [3] (рисунок 1.1).

Зображення цитологічних препаратів мають наступні особливості, які ускладнюють вирішення задач систем автоматизованої мікроскопії [12]:

– границя між цитоплазмою і ядром клітини слабко розрізняється через низьку якість барвників;

– клітини можуть бути розташовані близько один до одного, навіть зливатись;

– ядро, що виділяється може мати менш чіткі границі ніж границі сусідніх ядер;

– наявність всередині ядер сильних перепадів яскравості, які відбивають будову хроматину.

1.2 Методи оцінки якості зображень

Сегментація є одним із найважливіших етапів аналізу зображень [12, 13]. Підвищений інтерес наукової спільноти до цієї тематики виявляється у великій кількості публікацій, що присвячені розробці, модернізації і застосуванню різноманітних алгоритмів сегментації зображень. В результаті перед розробниками системи комп’ютерного зору постає непроста проблема вибору найбільш адекватних алгоритмів.

Якість методу сегментації, хоча б в грубій оцінці, фіксується за кількома властивостями. І в кожній ця оцінка повинна ними володіти. Якість роботи методу сегментації оцінюється залежно від того, наскільки отримана сегментація вміщує ці властивості. Найбільш часто використовуються наступні властивості [14-16]:

– однорідність областей (однорідність кольору або текстури);

– неподібність сусідніх областей;

– гладкість границі області;

– маленька кількість дрібних «дірок» всередині області.

Цілком природно, що існування такої проблеми визвало появу досліджень, що спрямовані на порівняльне тестування різноманітних алгоритмів сегментації зображень. Основні відмінності методик, що застосовуються в таких дослідженнях є:

– різні набори тестових зображень, що відрізняються як по типу зображення (реальні чи синтезовані), так і по розміру, кількості, джерелам (оригінальні зображення чи і загальнодоступних баз даних) і т.д.;

– різні процедури вибору оптимальних параметрів алгоритмів;

– різні критерії оцінки якості сегментації (кількісні чи якісні, що використовуються еталонне сегментоване зображення або ні).

Було розроблено декілька класифікацій методик оцінки алгоритмів сегментації [17-21]. Наведемо тут запропоновану в [21] класифікацію, у відповідності з якою методики оцінки діляться на:

1. Суб’єктивні.

2. Об’єктивні (системні, прямі, аналітичні, емпіричні, контрольовані, неконтрольовані (автоматичні)).

Суб’єктивні (або візуальні) – найбільш широко використовувані методики оцінки. Їх основний недолік – оцінка якості виконується людиною, тому у різних експертів ця оцінка може кардинально відрізнятись. До того ж, така методика потребує великих людських і часових ресурсів.

Системні методики дають оцінку якості сегментації на основі кінцевих результатів роботи всієї системи розпізнавання зображень. Для прикладу можна навести оцінку роботи різних детекторів границь на основі результатів розпізнавання об’єктів, виділених на зображені. Така оцінка не обов’язково визначає якість сегментації, а може просто вказувати на найбільш підходящий результат для подальшої обробки. Y.J. Zhang і J.J. Gerbrands в статтях [22, 23] пропонували використовувати так звані «вимірювання кінцевого результату».

Якщо ціль сегментації – отримання характеристик об’єкта, то точність в визначення таких характеристик може бути оцінкою точності сегментації. Схожий підхід пропонувався і в роботах [24, 25].

Прямі методики оцінюють безпосередньо сам алгоритм сегментації, або результати його роботи. Прямі методики поділяються на аналітичні і емпіричні.

Аналітичні методики розглядають алгоритм незалежно від його виходу [17, 27]. Вивчаються такі властивості алгоритму, як стратегія сегментування, складність, можливість розпаралелювання, ресурсоємність. Ці властивості не мають прямого відношення до якості сегментації. Аналітичні методи, що розглядаються в літературі, мають справу в основному з задачами спеціального виду [28-30].

Контрольовані (supervised) методики часто називають ще в англомовній літературі як discrepancy methods [17]. Вони використовують кількісні міри відмінності результату роботи алгоритму з деяким еталонним зображенням (також його називають ground truth, gold standard в англомовній літературі). Такі методи дають дуже хороші результати. Але сегментація експертом деякого еталонного зображення вносить в цей метод чинники суб’єктивності.

В автоматичних (unsupervised) методиках (інша назва – goodness methods [17, 31]) виконується кількісна оцінка деяких бажаних властивостей сегментованого зображення, на основі чого робиться висновок про якість сегментації. Вони не потребують наявності зразка сегментації, що, мабуть і є їх основною перевагою. Ця властивість дозволяє здійснювати контроль і самонавчання в системах реального часу.

Критерії якості сегментації, що використовуються в контрольованих методиках оцінки алгоритмів сегментації даються кількісну оцінку міру відмінності результату роботи алгоритму з еталонною сегментацією, що була створена експертом вручну або отримана автоматично при генерації синтетичного зображення (якщо методика оцінки припускає використання такого типу зображень). Більша частина таких критеріїв може бути віднесена до

одної з п’яти основних груп, в залежності від того, які з нижче перечислених характеристик використовуються для обчислення міри відмінності:

– кількість пікселів, віднесених при сегментації не до свого сегмента (процес сегментації можна розглядати як класифікацію пік селів вхідного зображення, тому будемо далі говорити про «неправильно класифікованих пікселях»);

– положення таких пік селів на еталонному зображенні відносно сегмента, до якого вони було помилково віднесені при сегментації;

– степінь фрагментації еталонного і сегментованого зображення;

– значення використовуваних для подальшого аналізу зображення характеристик сегментів, отримані на еталонному і результуючому зображеннях;

– відмінності значень характеристик вихідного зображення в даному пікселі від репрезентативних значень сегмента, до якого цей пік сель був віднесений.

1.3 Метричні методи оцінки якості зображень

Для оцінки за допомогою обчислювальних систем подібності або відмінності об'єктів, необхідно ввести формальну міру подібності (відмінності), в термінах якої ЕОМ і буде порівнювати об'єкти між собою. Подібність або відмінність між об'єктами класифікації встановлюється залежно від обраного метричної відстані між ними [32-35].

Метрики – важливий інструмент вирішення багатьох завдань розпізнавання образів та інтелектуального аналізу даних. Наявність метрики в просторі дозволяє приймати рішення про належність до множини або про подібність множин на основі кількісного показника. Величина метрики (відстані) часто безпосередньо пов'язана з імовірнісними характеристиками віднесення елемента до класу. Завдання пошуку та синтезу нових метрик залишається

актуальною. Основною метою створення метрик є підвищення ефективності при вирішенні нових практичних завдань [35].

Метрика – функція, яка кожної впорядкованої пари точок x і y простору ставить у відповідність, дійсне число $d(x,y)$. При цьому функція $d(x,y)$ має такі властивості [14]:

1) $d(x,y) \geq 0$, $d(x,y) = 0$ тоді і тільки тоді, коли $x = y$;

2) $d(x,y) = d(y,x)$;

3) $d(x,y) \leq d(x,z) + d(z,y)$.

Введення метрики $d(x,y)$ в просторі зображень дозволяє говорити про близькість або віддаленість точок в цьому просторі або про міру схожості або відмінності аналізованих зображень.

В сучасній літературі описано достатньо велику кількість об’єктивних метрик, які можна розділити на три класи [35-37]:

- Еталонні (full-reference, FR) – припускають наявність вхідного зображення, яке розглядається як опорне чи еталонне зображення при порівнянні, так як воно не зачумлене і має ідеальну якість.
- Нееталонні (no-reference, NR) – припускають, що в процесі отримання оцінки якості зображення опорне чи еталонне зображення відсутнє. Такі метрики найскладніші у реалізації і часто орієнтовані на конкретний вид спотворення.
- Квазіеталонні (reduced-reference, RR) – припускають, що деяка частина інформації про еталонне зображення присутня разом із зачумленим зображенням, причому кількість цієї інформації менше об’єму інформації, потрібної для еталонного зображення.

Метрику Евкліда можна застосовувати при обчисленні відстані між об'єктами, якщо вони описуються кількісними, якісними і дихотомічними ознаками. Її використовувати варто, коли ознаки є однорідними за смисловим навантаженням і однаково важливими для задачі, що розв'язується.

Евклідова метрика – найбільш часто використовувана міра відстані. Вона є геометричною відстанню в багатовимірному просторі і обчислюється таким чином:

$$L(S_k, X_i) = \sqrt{\sum_{k=1}^{n}(S_{ik} - X_{jk})^2} \quad (1.1)$$

Застосування евклідової відстані виправдано в таких випадках:

- властивості (ознаки) об'єкта однорідні за фізичним змістом і однаково важливі для класифікації;
- просторі ознак збігається з геометричним простором.

Попереднє задання вагових коефіцієнтів у формулах, що визначають відстані, вимагає наявності певної апріорної інформації і не завжди може бути зроблено оптимальним чином. Тому особливий інтерес представляють відстані, в яких закладена ідея вирівнювання ваг доданків від різних компонент, якщо вони істотно відрізняються за своїми абсолютними значеннями. Прикладом такої відстані є відстань по Камберру:

$$L(S_k, X_i) = \sum_{k=1}^{n} \frac{|S_{ik} - X_{jk}|}{|S_{ik} + X_{jk}|} \quad (1.2)$$

Відстань Камберра частково забезпечує інваріантність до масштабних змін зображень. Але, все ж, цього не достатньо для забезпечення високої достовірності розпізнавання зображення.

На практиці також широко використовується лінійна метрика або, як її ще іноді називають, метрика міських кварталів, якою задається манхетенська відстань. Ця відстань є різницею по координатах і визначається наступною формулою:

$$L(S_k, X_i) = \sum_{k=1}^{n}(|S_{ik}| - |X_{jk}|) \quad (1.3)$$

У більшості випадків ця міра відстані призводить до таких же результатів, як і для звичайного відстані Евкліда. Проте відзначимо, що для цієї міри вплив окремих великих різниць (викидів) зменшується (так як вони не зводяться в квадрат).

Відстань Хеммніга найбільш часто використовується для визначення відмінностей між об'єктами, що задаються дихотомічними ознаками і інтерпретується як число розбіжностей значень ознак у розглянутих об'єктів S_{ik} і X_{jk}. Для дихотомічних ознак вона відповідає квадрату евклідової відстані. Так само як і для евклідової відстані, може застосовуватися зважена відстань Хеммінга:

$$L(S_k, X_i) = \frac{1}{n}\sum_{k=1}^{n}(S_{ik} - X_{jk})^2 \qquad (1.4)$$

Дана відстань показує кількість позицій, в який об’єкти S_{ik} і X_{jk} відрізняються. Відстань Хеммінга має властивості метрики, задовольняючи усі властивості.

Відстань Хаусдорфа — відстань, визначена на всіх замкнених обмежених підмножинах метричного простору. Таким чином, відстань Хаусдорфа перетворює множину всіх непорожніх компактних підмножин метричного простору в метричний простір.

Для тощо, щоб знайти відстань у метриці Хаусдорфа потрібно володіти наступними поняттям відхилення. Відхиленням множини (рисунок 1.2), що складається із однієї точки $\{x_0\}$ від множини G називається:

$$p(x_0, G) = \min_{y' \epsilon G} \|x_0 - y'\|, \qquad (1.5)$$

де $\|x_0 - y'\|$ – евклідова відстань між точками x_0 та y' в даній метриці.

Відхиленням множини G1 від множини G2 (рисунок 1.3) називається:

$$p(G_1,\ G_2) = \max_{x' \in G_1} p(x',\ G_2), \qquad (1.6)$$

де $p(x',\ G_2)$ - відхилення множини, що складається з одної точки $\{x_0\}$ від множини G2 (1.5).

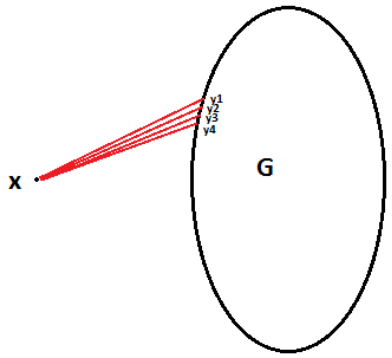


Рисунок 1.2 – Пошук відхилення $\{x_0\}$ від G

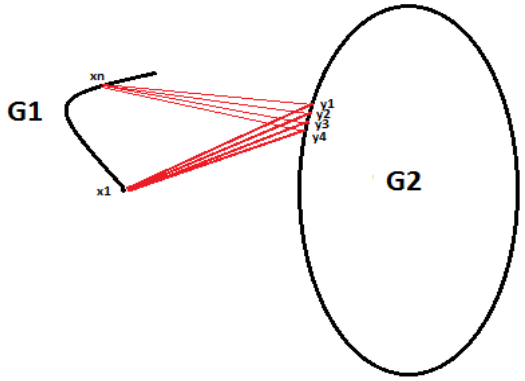


Рисунок 1.3 – Пошук відхилення G1 від G2

Відстанню Хаусдорфа (рисунок 1.4) називається:

$$d(G_1,\ G_2) = max\{p(G_1, G_2), p(G_2, G_1)\}, \qquad (1.7)$$

де $p\left(G_i,\ G_j\right)$ - відхилення множини G_i від множини G_j (1.6).

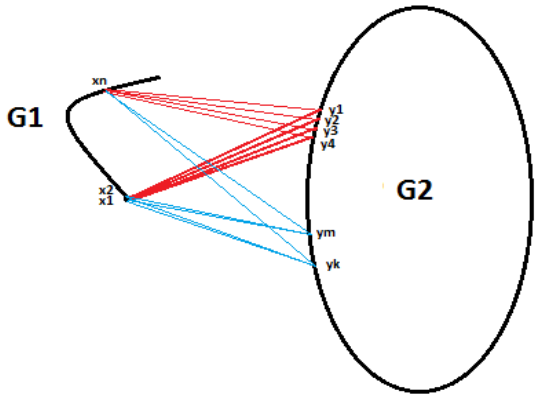


Рисунок 1.4 – Пошук відстані Хаусдорфа

1.4 Програмні засоби кількісної оцінки алгоритмів сегментації

На сьогоднішній день відомо багато алгоритмів сегментації зображень, які використовують різноманітні ознаки і характеристики зображень. Необхідно відмітити, що при дослідженні алгоритмів сегментації зображень виникають проблеми кількісної оцінки їх якості.

Відмітимо, що оцінка результатів сегментації може бути проведена візуально, однак при цьому кінцеві висновки є дуже суб'єктивними. Відомий альтернативний підхід, в якому оцінка якості алгоритмів сегментації проводиться по кінцевому результаті роботи технічної системи, наприклад в системах технічного зору. На сьогоднішній день існує велика кількість програмного забезпечення для аналізу цифрових зображень, наприклад: ImageJ, Image-Pro Plus, ImageTool, PhotoLib, ScopoTek та ін..

ImageJ – це програма з відкритим вихідним кодом для аналізу і обробки зображень (рисунок 1.5). Написана мовою Java співробітниками National Institutes of Health і поширюється без ліцензійних обмежень як суспільне надбання. Відкритий API дозволяє гнучко нарощувати функціонал за рахунок додаткових плагінів, а вбудована макромова – автоматизувати складні повторювані дії. ImageJ широко застосовується в біомедичних дослідженнях,

астрономії, географії та інших дисциплінах, пов'язаних з аналізом зображень [38].

Плагіни сторонніх розробників охоплюють широке коло завдань аналізу і обробки зображень: дозволяють проводити тривимірну візуалізацію в діапазоні від клітин до рентгенологічних зображень, автоматичні порівняння аж до створення автоматизованих систем вивчення, наприклад, в гематології. Архітектура плагінів ImageJ і вбудована в програму система розробки робить цю платформу вельми популярною для роботи і викладання аналізу та обробки зображень [38].

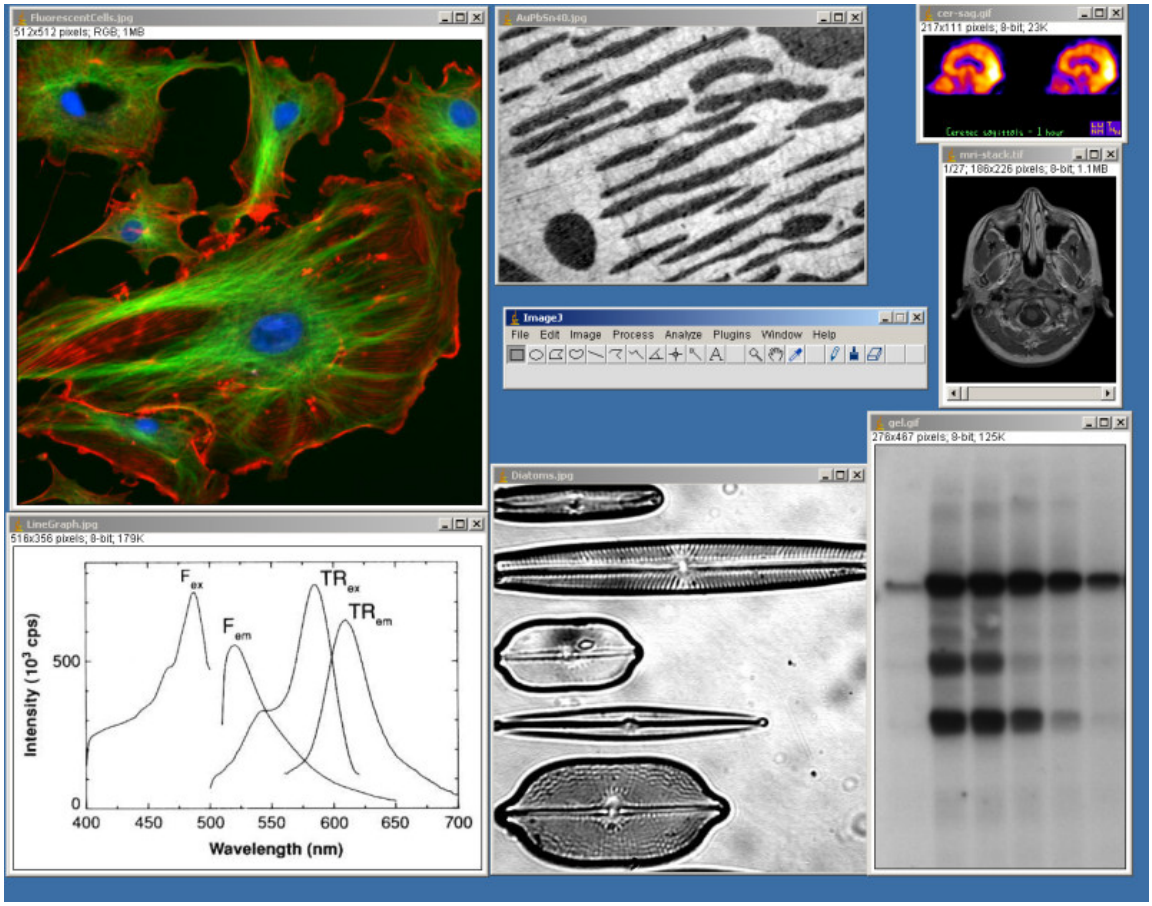


Рисунок 1.5 – Скріншот програми ImageJ

Altami Studio – програмне забезпечення для управління цифровими камерами, проведення вимірювань і автоматичного аналізу зображень (рисунок 1.6).

Програма аналізу зображень Altami Studio дозволяє [39]:

– управляти процесом захвату зображень і здійснювати аналіз і обробку зображення;

– управляти налаштуваннями камери (яскравість, гама, насиченість, експозиція, підсилення, кадрове накопичення), а також встановлювати доступні розширення;

– виконувати калібрування масштабу по об’єкт-мікрометру і зберігати/завантажувати вироблені калібрування;

– оперативно оцінювати лінійні розміри елементів зображення за допомогою спеціального інструмента – лінійки;

– відображати результати поточних вимірювань в окремому списку, з можливістю його редагування;

– застосовувати до зображення різноманітні растрові операції: геометричні, морфологічні, порогові і багато інших.

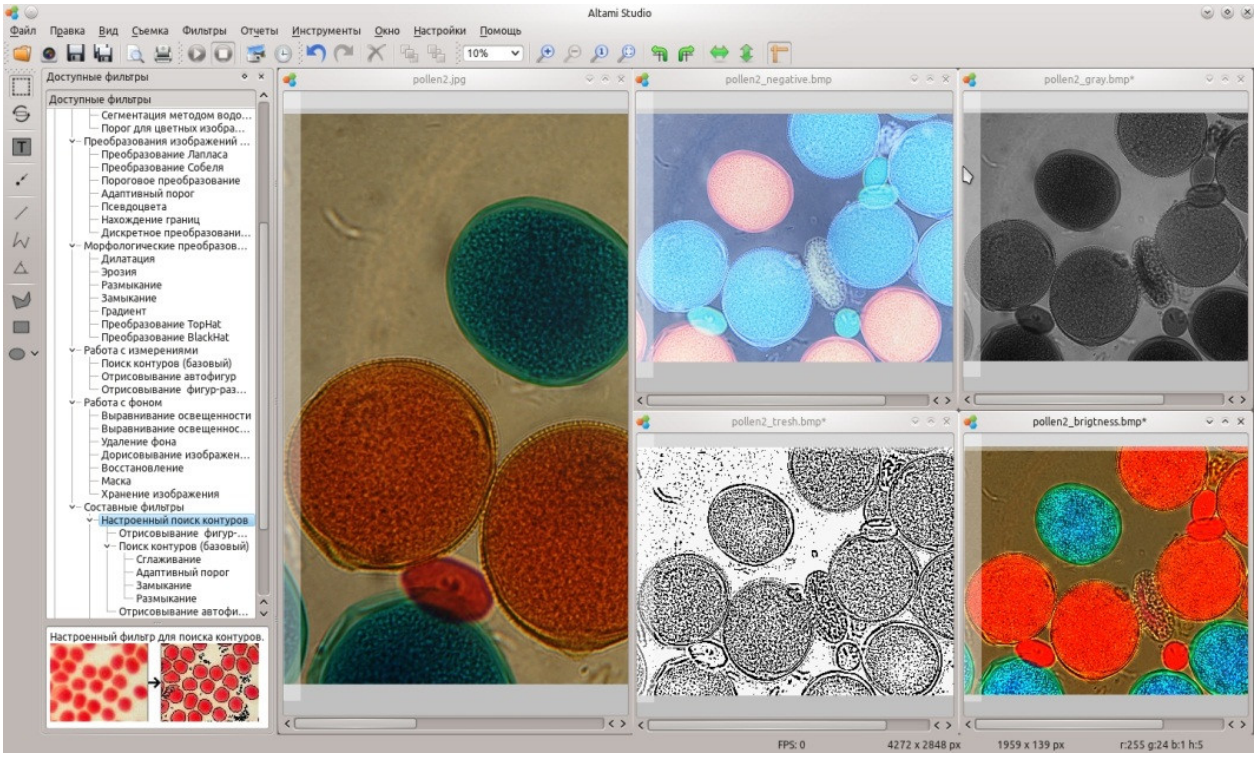


Рисунок 1.6 – Скріншот програми Altami Studio

Програмне забезпечення Image-Pro Premier (рисунок 1.7) пропонує інтуїтивно зрозумілі інструменти, які дозволяють легко захоплювати, обробляти,

вимірювати, аналізувати та обмінюватись зображеннями і цінними даними. Програма пропонує 64-бітну підтримку, дружній інтерфейс, інтуїтивно зрозумілі макроси і додаток будівельних інструментів, нові та вдосконалені способи автоматичної сегментації, класифікації і вимірювання предметів, і більше інструментів для налаштування робочого процесу [40].

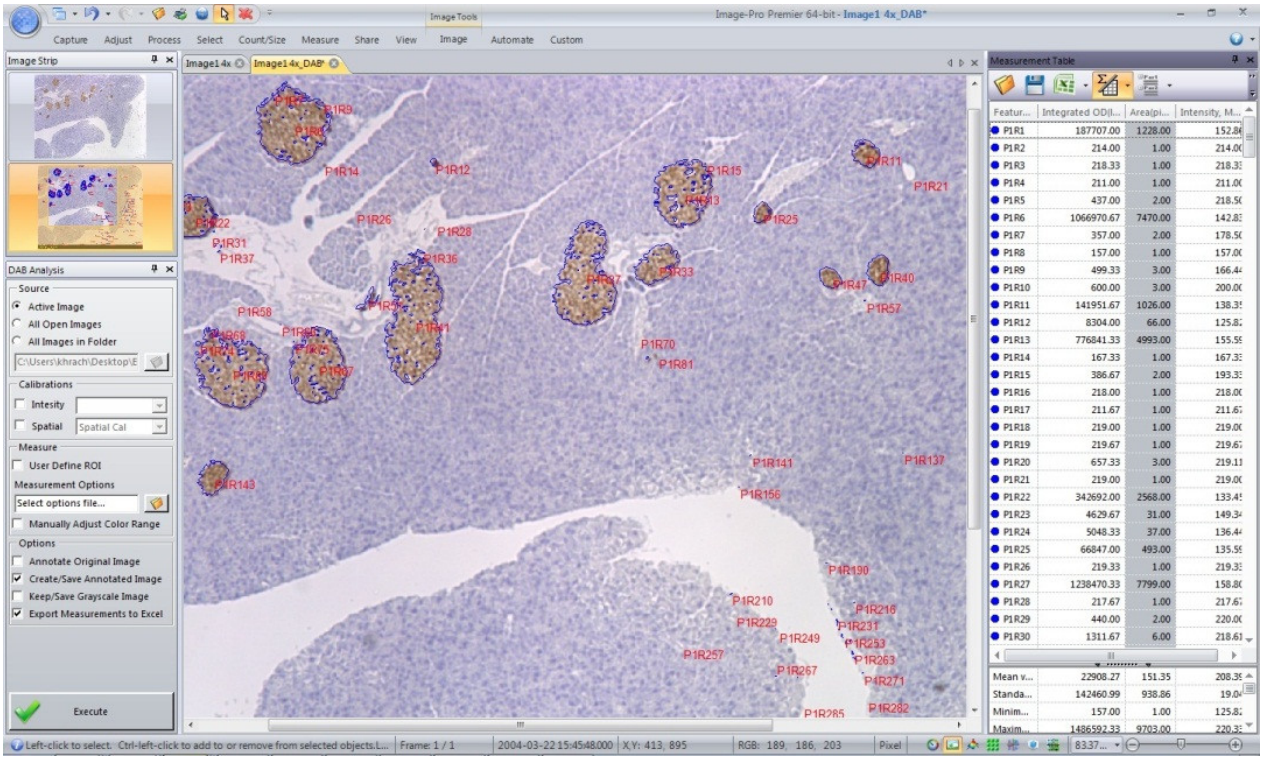


Рисунок 1.7 – Скріншот програми Image-Pro Premiere

1.5 Постановка завдання

Сегментація активно використовується при обробці медичних зображень, наприклад, для виявлення пухлин та інших патологій, визначення об’ємів тканин, різних діагностик, планування лікування.

Виділення границь різними алгоритмами мали б призводити до однакового результату, але оскільки на сегментацію впливає велика кількість різних факторів. Один і той же алгоритм сегментації може виявитися хорошим для

однієї задачі і поганим для іншої. Тому, виникає необхідність оцінки якості сегментації.

Найбільш надійним способом отримання оцінки якості зображень є використання експертного підходу, але його застосування вимагає великої кількості людських ресурсів і часу, особливо у випадку великої кількості оцінюваних зображень. Ефективним рішенням зменшення трудомісткості і часу отримання оцінки якості є застосування об'єктивних (математичних) метрик, що дають максимально наближену до експертного підходу оцінку якості зображень.

Звідси випливає актуальність розробки алгоритмів оцінки якості сегментації. Серед відомих методів оцінки якості сегментації, на мою думку, потрібно використовувати об’єктивні методи на основі метричних просторів. Найбільш популярна метрика для порівняння областей є метрика Хаусдорфа.

Наявні програмні засоби не містять модуля оцінки якості алгоритмів сегментації, тому розробка програмного модуля є актуальним завданням.

Об’єктом дослідження є алгоритми сегментації зображень та метрики оцінки результатів сегментації.

Метою роботи є розробка програмного засобу для проведення кількісної оцінки результатів роботи алгоритмів сегментації зображень в метриці Хаусдорфа.

Для досягнення поставленої мети необхідно вирішити наступні задачі:

- Проаналізувати відомі алгоритми сегментації;
- Проаналізувати методи кількісної оцінки алгоритмів сегментації;
- Проаналізувати відомі метрики для оцінки відстаней між зображеннями;
- Розробити алгоритми знаходження контурів областей зображень;
- Розробити алгоритми знаходження відстані між двома областями в метриці Хаусдорфа;
- Розробити архітектуру програмної системи;
- Програмно реалізувати розроблені алгоритми;

- Провести комп’ютерні експерименти на прикладі цитологічних зображень.

2 АЛГОРИТМИ ПОРІВНЯННЯ ОБЛАСТЕЙ ЗОБРАЖЕНЬ В МЕТРИЦІ ХАУСДОРФА

2.1 Аналіз алгоритмів виділення границь

Існування границі об'єктів на зображенні в вагомій мірі зменшило кількість даних, які необхідно опрацювати. Границя зберігає важливу інформацію про об'єкти на зображенні, їх власивості: форму, розмір, кількість тощо. Головною властивістю техніки виділення границь є перспектива витягти точну лінію, яка має хорошу орієнтацію. У літературі існує багато алгоритмів, які дозволяють знаходити кордони об'єктів, але нема опису того, як оцінювати результати обробки. В кожному випадку результати оцінюються індивідуально і залежать від області їх застосування.

Фундаментальним інструментом для сегментації зображення є інструмент виділення границь. Такі алгоритми перетворять вхідне зображення в зображення з контурами об'єктів, переважно в сірих тонах. За допомогою виділення контуру в системах комп'ютерного зору (чи задачах обробки зображень), розглядають великі зміни ступеня яскравості, фізичні та геометричні параметри об'єкта на зображенні. Виділення границь описує в загальних рисах об'єкти і за рахунок цього отримуються деякі знання про зображення. Виявлення границь дозволяє виявити значні неоднорідності [41].

Границя – це локальна зміна яскравості на зображенні. Вони, як правило, проходять по краю між двома областями. За допомогою границь можна отримати базові знання про зображення. Функції їх отримання використовуються передовими алгоритмами комп'ютерного зору і таких областях, як медична обробка зображень, біометрія і тому подібні. Виявлення границь - активна область досліджень, оскільки вона полегшує високорівневий аналіз зображень. На напівтонових зображеннях існує три види розривів: точка, лінія і кордон. Для виявлення всіх трьох видів неоднорідностей можуть бути використані просторові маски [42].

У технічній літературі наведено і описано велику кількість алгоритмів виділення контурів та границь. До найбільш популярних відносяться: оператор Робертса, Собеля, Превітта, Кірша, Робінсона, алгоритм Кенні і LoG-алгоритм [41].

Оператор Робертса для виділення границь (рисунок 2.1) введений Лоуренсом Робертсом в 1964 році. Він виконує прості і швидкі обчислення двовимірного просторового виміру на зображенні. Цей метод підкреслює області високої просторової частоти, які часто відповідають краям. На вхід оператора подається напівтонове зображення. Значення пікселів вихідного зображення в кожній точці передбачає якусь величину просторового градієнта вхідного зображення в цій же точці.

Gx

+1	0
0	-1

Gy

0	+1
-1	0

Рисунок 2.1 – Оператор Робертса

Оператор Собеля був винайдений Собелем в 1970 році. Даний метод виявлення границь використовує наближення до похідної. Це дозволяє виявляти край в тих місцях, де градієнт найвищий. Даний спосіб виявляє кількість градієнтів на зображенні, тим самим виділяючи області з високою просторовою частотою, які відповідають границям. В цілому це призвело до знаходження передбачуваної абсолютній величині градієнта в кожній точці вхідного зображення. Даний оператор складається з двох матриць, розміром 3 × 3 (рисунок 2.2). Друга матриця відрізняється від першої тільки тим, що повернута на 90 градусів. Оператор Собеля дуже похожий до оператора Робертса. Виявити границі оператором Робертса обчислювально простіше, але призводить до більшої зашумленості результуючого зображення [43].

Gx			Gy		
-1	0	+1	+1	+2	+1
-2	0	+2	0	0	0
-1	0	+1	-1	-2	-1

Рисунок 2.2 – Оператор Собеля

Оператор Превітта для виявлення границь на зображенні було запропоновано у 1970 році. Правильним напрямком в даному алгоритмі була оцінка величини і орієнтація границі. Навіть при тому, що виділення границь є вельми трудомістким завданням, такий підхід дає досить непогані результати. Даний алгоритм базується на використанні масок розміром 3 на 3, які враховують 8 можливих напрямків, але прямі напрямки дають найкращі результати (рисунок 2.3). Всі маски згортки розраховані.

Gx			Gy		
-1	-1	-1	-1	0	1
0	0	0	-1	0	1
1	1	1	-1	-2	1

Рисунок 2.3 – Маски оператора Превітта

Алгоритм Кірша базується на використанні всього однієї маски, яку обертають по восьми головних напрямках: північ, північний захід, захід, південний захід, південь, південний схід, схід і північний схід (рисунок 2.4). Величина границі визначена як максимальне значення, знайдене за допомогою маски. Визначений маскою напрямок видає максимальну величину. Наприклад, маска k0 відповідає вертикальній границі, а маска k5 - діагональній. Можна також зауважити, що останні чотири маски фактично такі ж, як і перші, вони є дзеркальним відображенням відносно центральної осі матриці [44].

Схід

5	-3	-3
5	0	-3
5	-3	-3

Захід

-3	-3	5
-3	0	5
-3	-3	5

Північний схід

-3	-3	-3
5	0	-3
5	5	-3

Південний захід

-3	5	5
-3	0	5
-3	-3	5

Північ

-3	-3	-3
-3	0	-3
5	5	5

Південь

5	5	5
-3	0	-3
-3	-3	-3

Північний захід

-3	-3	-3
-3	0	5
-3	5	5

Південний схід

5	5	-3
5	0	-3
-3	-3	-3

Рисунок 2.4 – Маски оператора Кірша

Метод Робінсона подібний до метода Кірша, але є більш простим в реалізації в силу використання коефіцієнтів 0, 1 і 2. Маски даного оператора симетричні щодо центральної осі, заповненої нулями (рисунок 2.5). Достатньо отримати результат від обробки перших чотирьох масок, інші ж можна отримати інвертуючи перші. Максимальне значення, отримане після застосування всіх чотирьох масок до пікселя і його оточенню вважається величиною градієнта, а

кут градієнта можна апроксимувати як кут ліній нулів в масці, які дають максимальний відгук [45].

Схід

-1	0	1
-2	0	2
-1	0	1

Захід

1	0	-1
2	0	-2
1	0	-1

Північний схід

0	1	2
-1	0	1
-2	-1	0

Південний захід

0	-1	-2
1	0	-1
2	1	0

Північ

1	2	1
0	0	0
-1	-2	-1

Південь

-1	-2	-1
0	0	0
1	2	1

Північний захід

2	1	0
1	0	-1
0	-1	-2

Південний схід

-2	-1	0
-1	0	1
0	1	2

Рисунок 2.5 – Маски Робінсона

Метод Marr-Hildreth використовується для виявлення границь в цифрових зображеннях. Він виявляє неперервні криві всюди, де помічено швидкі і разючі зміни яскравості групи пікселів. Цей метод є доволі простим, працює він за допомогою згортки зображення з LoG-функцією або як швидка апроксимація з

DoG. В обробленому результаті контурам відповідають нулі. Алгоритм граничного детектора відбувається за кроками [41]:

– розмиття зображення методом Гаусса;

– примінення оператора Лапласа до розмитого зображення (досить часто перші два кроки об'єднуються в один);

– виробляємо цикл обчислень і в отриманому результаті дивимося чи змінювався знак. Якщо він змінився з негативного на позитивний і значення зміни значення більше за деякий заданий поріг, то визначаємо цю точку, як границю;

– для одержання кращих результатів крок 2, що використовує оператор Лапласа, можна здійснтти через гістерезис так, як це зроблено в алгоритмі Кенні.

Алгоритм виділення контурів Лаплассіан Гауссіана (LoG) проходить в два кроки. На першому кроці він вигладжує зображення. Потім за алгоритмом обчислюється функція Лапласа, що призводить до утворення подвійних контурів. Визначення контурів зводиться до знаходження нулів на перетині подвійних кордонів. Комп'ютерна реалізація функції Лапласса зазвичай здійснюється за допомогою масок наведених на рисунку 2.6. Лаплассіан зазвичай використовує знаходження пікселя на темній або світлій стороні границі [48].

Gx

0	-1	0
-1	4	-1
0	-1	0

Gy

-1	-1	-1
-1	8	-1
-1	-1	-1

Рисунок 2.6 – Маски LoG

Детектор границь Кенні є одним з найпопулярніших алгоритмів виявлення контурів. Вперше він був запропонований Джоном Кенні в магістерській дисертації в 1983 році, і до цих пір є кращим за багатьох алгоритмів, розроблених

пізніше. Важливим кроком у даному алгоритмі є усунення шуму на контурах, який значною мірою може вплинути на результат, при цьому необхідно максимально зберегти границі. Для цього необхідний ретельний підбір порогового значення при обробці даним методом.

Алгоритм:

– розмиття вихідного зображення f(r, c) за допомогою функції Гаусса;

– виконати пошук градієнта. Границі намічаються там, де градієнт приймає максимальне значення;

– придушення не-максимумів. Тільки локальні максимуми відзначаються як границі;

– підсумкові межі визначаються шляхом придушення всіх країв, не пов'язаних з визначеними границями.

На відміну від операторів Робертса і Собеля, алгоритм Кенні не надто чутливий до шуму на зображенні.

2.2 Алгоритми кусково-лійнійної апроксимації

Попередньо виділений контур об’єкта задається як набір координаті його пікселів. У великих об’єктів кількість точок контуру є досить великою, що призводить до збільшення виконання будь-яких операцій над даним контуром (в тому числі і обчислення відстані між двома контурами).

Для спрощення (проріджения) контура певного багатокутника використовують алгоритми кусково-лінійної апроксимації. На даний момент існує багато таких алгоритмів, та найпоширенішим є алгоритм Дугласа-Рамера-Пекера, який ще називають алгоритмом ітеративної найближчої точки чи алгоритмом розбиття і злиття [49-50].

Суть алгоритму полягає в тому, щоб по даній ломаній, побудувати ломану з меншою кількістю точок. Алгоритм визначає розбіжність, яка обчислюється по

максимальній відстані між вихідною і спрощеною кривими. Спрощена крива складається з підмножини точок, які визначаються з вихідної кривої.

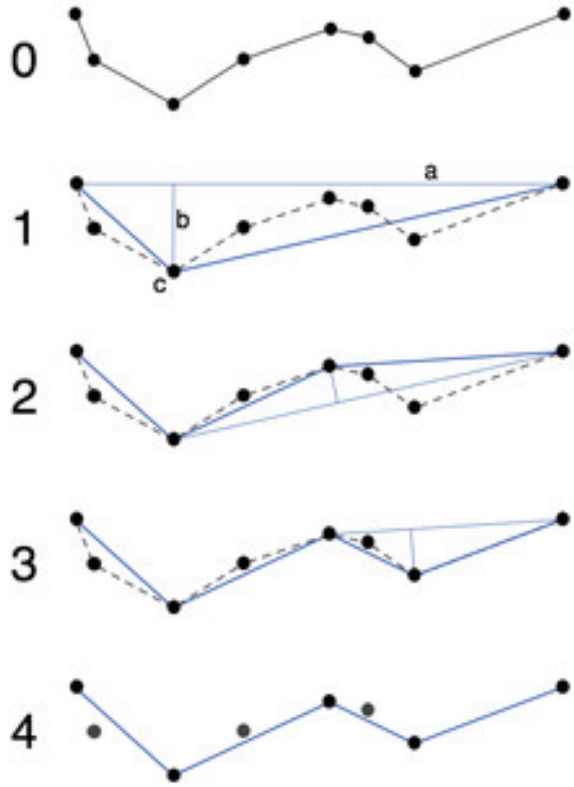


Рисунок 2.7 – Згладжування кривої алгоритмом Дугласа-Рамера-Пекера

Алгоритм Дугласа-Рамера-Пекера:

1. Крок 1. Початкова крива складається з впорядкованого набору точок або ліній на відстані ε > 0. На рисунку 2.7 початкову криву позначено цифрою 0, спрощену – цифрою 4.

2. Крок 2. Алгоритм рекурсивно ділить лінію. На вхід алгоритму поступають координати всіх точок між першою і останньою – ці дві точки не змінюються. За алгоритмом далі знаходиться точка, яка найбільш віддалена від відрізка, що з'єднує першу і останню точки. На рисунку 2.7 (1) найбільш віддалена позначена *c*, найбільша відстань від вірізка до точки – *b*. Якщо точка знаходиться на відстані меншій ніж *ε*, то всі точки, які ще не були відзначені до збереження, можуть бути викинуті з набору і отримана пряма згладжує криву з точністю не нижче ε.

3. Крок 3. Якщо ж відстань є більшою ε, то алгоритм рекурсивно викликає себе з набору від початкової до даної і від даної до кінцевої точки (що означає, що дана точка буде відзначена до збереження).

4. По закінченню всіх рекурсивних викликів вихідна ламана будується тільки з тих точок, що були відзначені до збереження.

Псевдокод алгоритму:

```
function DouglasPeucker(PointList[], epsilon)
//Знаходимо точку з максимальною відстанню від прямої між першою і
останньою точками набору
 dmax = 0
 index = 0
 for i = 2 to (length(PointList) - 1)
  d = PerpendicularDistance(PointList[i], Line(PointList[1],
PointList[end]))
  if d > dmax
   index = i
   dmax = d
  end
 end

// Якщо максимальна дистанція більша ніж епсілон, то рекурсивно
викликаємо її на ділянках
 if dmax >= epsilon
  //Рекурсивний виклик
  recResults1[] = DouglasPeucker(PointList[1...index], epsilon)
  recResults2[] = DouglasPeucker(PointList[index...end], epsilon)

// Будуємо кінцевий набір точок
  ResultList[] = {recResults1[1...end-1] recResults2[1...end]}
 else
  ResultList[] = {PointList[1], PointList[end]}
 end

 // Повертаємо результат
 return ResultList[]
end
```

На даний час алгоритм Дугласа-Рамера-Пекера є найбільш відомим, але алгоритм Вісвалінгема є більш ефективним і має більш інтуїтивно зрозуміле пояснення: він поступово видаляє точки з найменш відчутними змінами. Спрощення часто дозволяє усунути близько 95% або й більше точок, в той же час зберігаючи попередню форму об’єкта.

Для того, щоб визначити, видалення якої точки призведе до найменших візуальних змін алгоритм Вісвалінгема обчислює площі трикутників, вершинами яких є тріплети точок, що лежать вдовж кожної лінії; точка, що асоціюється з трикутником в якого площа найменша , видаляється. Після кожного видалення, площа кожного сусіднього трикутника пере обчислюється і процес повторюється знову [51].

Для наочності розглянемо приклад. Нехай ламана складається з 6 точок (рисунок 2.8). Ігноруючи кінцеві точки, ефективна площа кожної точки визначається площею трикутника асоційованого з нею. Фіолетовий трикутник є найменшим, отже п’ята точка видаляється першою. Це видалення вимагає пере обчислення площі суміжного червоного трикутника. Зелений трикутник тепер є найменшим, отже, третя точка видаляється, і пере обчислюються суміжні трикутники – помаранчевий і червоний. Цей процес продовжується доти, поки залишиться тільки 1 трикутник.

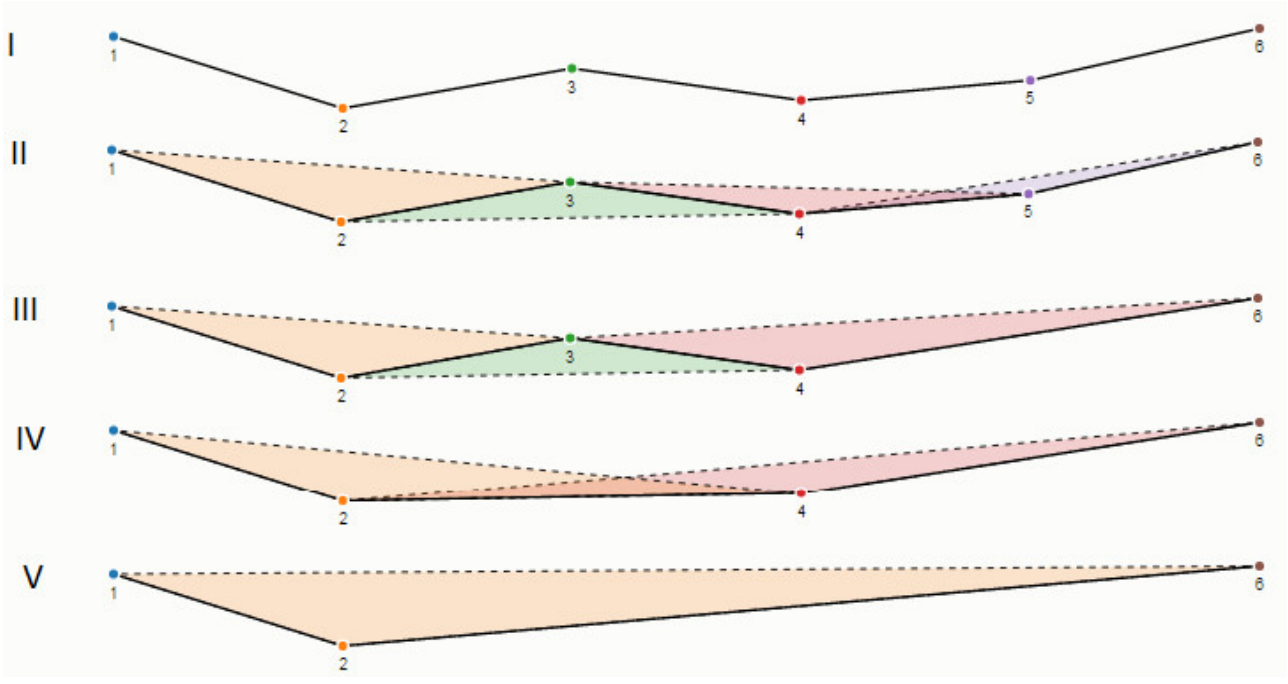


Рисунок 2.8 – Ілюстрація алгоритму Вісвалінгема

Псевдокод алгоритму:

```
function VisvalingamWhyatt(PointList[], number_to_keep)
```

```
// Створюємо копію PointList[]
newPointList= PointList.clone()
// Поки розмір більше 2
while newPointList.length > 2
     // Створюємо масив для зберігання площ
     areas[PointList.length-2]
     // Обчислюємо площі для кожної точки
     for i=2 to i=last-1
          areas[i]= effectiveArea(newPointList[i])
     // Знаходимо точку з найменшою площою
     leastArea= areas[2]
     for i=3 to i=last-1
          if areas[3] < leastArea
               leastArea= areas[3]
     // Створюємо масив для зберігання точок у порядку спадання їх
площі
     stack[]
     // Видаляємо точку з найменшою площею і поміщаємо в стек
     stack[stack.length]= newPointList[
newPointList.find(leastArea) ]
     newPointList.remove( newPointList.find(leastArea) )
     // Створюємо нову пусту колецію точок такого ж розміру що і
PointList
     returnPointList[PointList]
     // Додаємо оригінальну першу і останню точку до масиву
returnPointList
     returnPointList[0]=PointList[0]
     returnPointList[last]=PointList[last]
     // Поки розмір returnPointList менший ніж number_to_keep
     while returnPointList.length < number_to_keep
          returnPointList[ PointList.find(stack[last] ) ] =
stack[last]
     // Повертаємо returnPointList
     return returnPointList
                              end
```

Немає гарантії, що видалення точки збільшить площу суміжних трикутників, тому у алгоритмі пропонується вважати найбільшим той трикутник, який включає в себе трикутник видалений на попередньому кроці.

Одною з найкращих особливостей алгоритму Вісвалінгема є те, що ефективна площа може бути потім збережена у геометрії. Наприклад, точка може мати z-координату, що вказує на її ефективну площу. Це дозволить ефективно фільтрувати для динамічного спрощення навіть якщо алгоритм працює на сервері.

2.3 Алгоритми обчислення відстані Хаусдорфа

Після знаходження контурів об'єктів зображень та їх проріджения можна приступити до обчислення відстані Хаусдорфа між двома об'єктами. Зазвичай дані об'єкти мають форму багатокутника. Оскільки пошук відстані Хаусдорфа стандартним алгоритмом є дуже обчислювально складним, то є потреба у використанні модифікованих алгоритмів, які дозволять зменшити обчислювальну складність.

У роботі [52] представлені алгоритми лінійного часу для обчислення відстані Хаусдорфа між опуклими багатокутниками. А саме, розглядаються 2 алгоритми: для багатокутників що перетинаються та для багатокутників що не перетинаються.

Вхідними даними для обох алгоритмів є:

- два багатокутники P_1 та P_2, що задаються у вигляді списку вершин їх контура;

- значеннями n_1 та n_2 – кількість вершин багатокутників P_1 та P_2 відповідно;

- координати (a_j^1, b_j^1) (відповідно(a_j^2, b_j^2)) j-ої вершини v_j^1 (відповідно v_j^2);

- така нумерація вершин P_1 (відповідно P_2), що для будь-якої $j \in \{0,1, \dots, n_1 - 1\}$ (відповідно $j \in \{0,1, \dots, n_2 - 1\}$), відрізок що з'єднує вершини v_j^1 та $v_{j+1\ mod\ n1}^1$ (відповідно v_j^2 та $v_{j+1\ mod\ n2}^2$) є j-тим краєм багатокутника P_1 (відповідно P_2).

У випадку, коли багатокутники P_1 та P_2 є розділеними (вони не перетинаються і один не міститься всередині іншого) для знаходження відстанні Хаусдорфа використовується допоміжна лінія r. Допоміжна лінія r опуклого багатокутника P є прямою лінією, що проходить через одну з його вершин таким чином, що внутрішня частина P лежить по одну сторону від r.

Перша частина алгоритму полягає у обчисленні відстаней з кожної вершини P_1 до її проекції на P_2. Його друга частина полягає в обчисленні відстаней з кожної вершини P_2 до її проекції на P_1. Алгоритм починається з обчислення $y_0 = Proj_{P_2}(a_0^1, b_0^1)$, що може бути легко виконано за O(n^2) час використовуючи звичайний алгоритм.

Алгоритм знаходження відстані Хаусдорфа для розділених багатокутників:

1. Визначаємо найменшу відстань d_0^2 для точки v_0^1 багатокутника P_1 стандартним алгоритмом. Точку на багатокутнику P_2 позначаємо y_0. Зберігаємо відстань у масив.

2. Знаходимо точку y_{k+1}:

Випадок 1: якщо v_{k+1}^1 знаходиться по ліву сторону від відрізка $y_k v_k^1$, то шукаємо y_{k+1} проходячи багатокутник P_2 проти годинникової стрілки починаючи з y_k, до тих пір поки не виконається одна з умов:

а) Ребро багатокутника P_2, що визначене точками $v_q^2 v_{q+1}^2$, є таким що основа перпендикуляра (позначимо її z) з v_{k+1}^1 до $v_q^2 v_{q+1}^2$ лежить між v_q^2 та v_{q+1}^2. Тоді $y_{k+1} = z$.

б) Вершина v_q^2 багатокутника P_2 є такою, що перпендикуляр до $v_{k+1}^1 v_q^2$ у точці v_q^2 є допоміжною лінією r багатокутника P_2. У такому випадку $y_{k+1} = v_q^2$.

Випадок 2: v_{k+1}^1 знаходиться по праву сторону відносно $y_k v_k^1$. У цьому випадку виконуємо ті ж операції що і у випадку 1, окрім того, що обхід багатокутника P_2 виконується за годинниковою стрілкою, починаючи з точки y_k.

Випадок 3: v_{k+1}^1 лежить на відрізку $y_k v_k^1$. Тоді $y_{k+1} = y_k$.

3. Обчислюємо відстань d_k^2 і зберігаємо у масив.

4. Виконуємо операції 2 і 3 для усіх вершин багатокутника P_1 та P_2.

5. Відстанню Хаусдорфа буде максимальне значення з масиву відстаней.

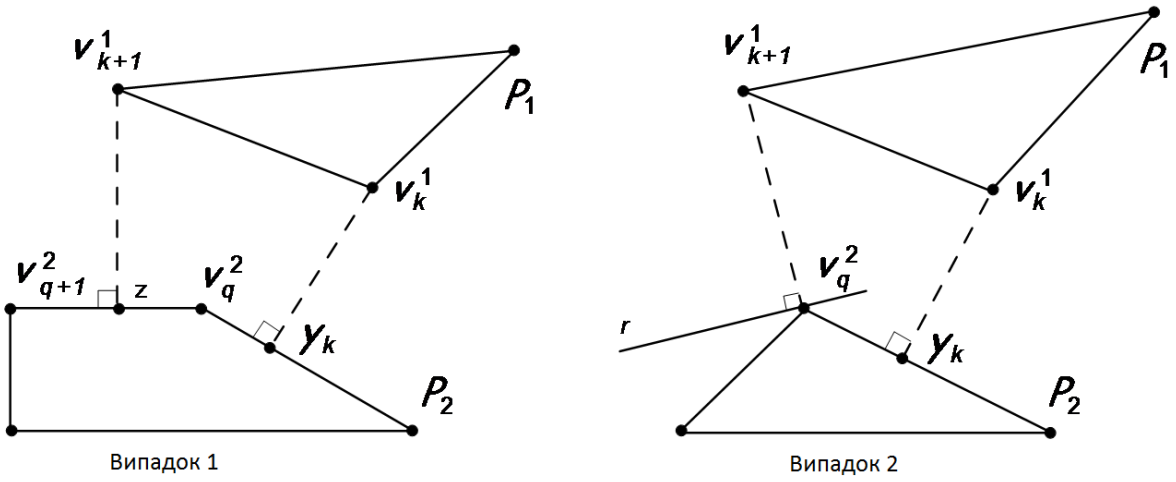


Рисунок 2.9 – Ілюстрація алгоритму для розділених багатокутників

У випадку коли P_1 та P_2 є не обов'язково розділеними і жоден багатокутник не міститься цілком у іншому використовуємо алгоритм знаходження відстані Хаусдорфа для багатокутників що перетинаються.

Він грунтується на знаходженні фігури, яку позначимо P, і визначимо таким чином:

$$P = (P_1 \cup P_2) - Interior(P_1 \cap P_2). \qquad (2.1)$$

За визначенням область P – це область P_1 або P_2, без внутрішньої частини перетину (слово *Interior* в перекладі означає внутрішня частина). По-іншому P – це множина частин P_1 або P_2, які в них не є спільними. Область P складена з m не опуклих багатокутників (рисунок 2.10).

Границя кожної такої частини складається із двох полігональних ланцюгів: один пов'язаний з багатокутником P_1 (позначимо його як P_1-ланцюг), а інший з P_2 (позначимо його P_2-ланцюг). Припустимо, що частини які появляються в результаті частинно-декомпозиції P нумеруються у порядку проти годинникової стрілки, тим же, як і вершини багатокутників. Позначимо P_1-$chain_k$ (відповідно P_2-$chain_k$) P_1-ланцюг (відповідно P_2-ланцюг) у k-тій частині

декомпозиції P. «Попередні» і «наступні» частини k-тої частини є відповідно $(\mathrm{k}-1) \bmod \mathrm{m}$ та $(\mathrm{k}+1) \bmod \mathrm{m}$ частини декомпозиції P.

На прикладі k-тої частини, припустимо, що $P_1\text{-}chain_k$ є зовнішнім ланцюгом, а $P_1\text{-}chain_k$ внутрішнім. Тобто усі вершини $P_2\text{-}chain_k$ знаходяться на границі P_1. Отже, відстані з усіх вершин $P_2\text{-}chain_k$ (тобто внутрішнього ланцюга) до P_1 рівні нулю і не повинні враховувати при обчисленні відстані.

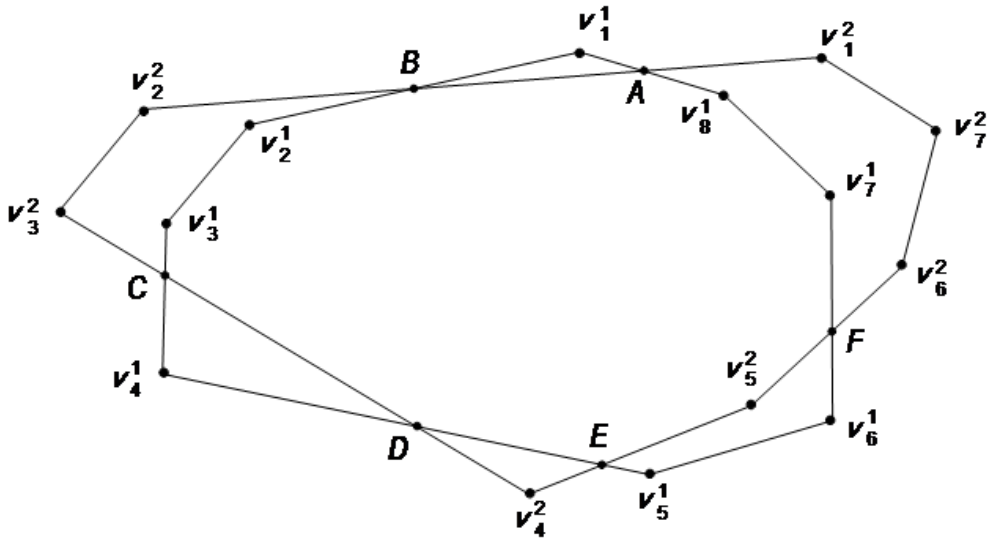


Рисунок 2.10 – Ілюстрація алгоритму для багатокутників що перетинаються

Алгоритм знаходження відстані Хаусдорфа для багатокутників, що перетинаються:

1. Виконуємо частинно-декомпозицію багатокутників, отримуємо область P.
2. Для кожної вершини v_k^j зовнішнього ланцюга частини $\mathrm{m_k}$:
   2.1. Знаходимо відстань Хаусдорфа $d^{k-1\,mod\,m}$ до зовнішнього ланцюга попередньої частини $\mathrm{m_{k-1}}$ mod m.
   2.2. Знаходимо відстань Хаусдорфа $d^{k+1\,mod\,m}$ до зовнішнього ланцюга наступної частини $\mathrm{m_{k+1}}$ mod m.

2.3. Знаходимо відстань Хаусдорфа d^k до внутрішнього ланцюга частини m_k .

2.4. Відстань Хаусдорфа для цієї вершини d_h = min ($d^{k-1 \, mod \, m}$, d^k, $d^{k+1 \, mod \, m}$). Зберігаємо значення d_h у масив.

3. Відстанню Хаусдорфа між двома багатокутниками буде найбільше значення із масиву відстаней.

Алгоритм пошуку відстані Хаусдорфа для багатокутників що перетинаються на псевдокоді:

Begin

Обчислюємо об'єднання і перетин P_1 та P_2 використовуючи алгоритм лінійного часу.

Обчислити і зберегти результат декомпозиції *P.*

Нехай *m* – кількість частин отриманих від декомпозиції *P* і n_1^k (відповідно n_2^k*)* – кількість вершин P_1 (відповідно P_2) у ланцюзі *P1-chain*$_k$ (відповідно *P2-chain*$_k$*), для k*=1,2,..,*m.*

Нехай *external*$_k$ (відповідно *internai*$_k$*)* буде зовнішнім(внутрішнім) ланцюгом частини *k,* для *k*= 1,2,..,*m.*

Встановлюємо $n_{ext}^k = n_1^k (n_{int}^k = n_2^k)$ якщо *P1-chaink* є зовнішнім ланцюгом; і $n_{ext}^k = n_2^k$ $(n_{int}^k = n_2^k)$ в іншому випадку

Встановлюємо *Dh* = 0 and *k* = 1.

While k <= m do

For each node of *externalk* do

Обчислюємо відстані $d^{k-1 \bmod m}$, $d^{k \bmod m}$, $d^{k+1 \bmod m}$, до ланцюгів $external_{k-1 \bmod m}$, $internak_k$, $external_{k+1 \bmod m}$

distances.add(min($d^{k-1 \bmod m}$, $d^{k \bmod m}$, $d^{k+1 \bmod m}$))

end foreach

k = k + 1

end while

Dh = max(distances)

end begin

2.4 Розбиття неопуклого багатокутника

Алгоритми пошуку відстані Хаусдорфа, що описані в двох попередніх підрозділах підходять тільки для опуклих фігур. Тому перш ніж використовувати їх, потрібно перевірити вхідний багатокутник на опуклість. Якщо він не є опуклий, то тоді розбити його на опуклі фігури.

Розглянемо метод триангуляції, який проводить розбиття будь-якої фігури на трикутники. Будь-який трикутник є опуклою фігурою. Тоді застосування триангуляції до неопуклої фігури приведе до її розбиття на опуклі фігури. Існують декілька алгоритмів триангуляції, але найпоширенішими є алгоритм відсікання вух та монотонна триангуляція [53].

Алгоритм відсікання вух. Ідея цього алгоритму полягає в послідовному відсіканні трикутників (вух). Вершину v_i називають вухом, якщо діагональ проведена з v_{i-1} до v_{i+1} повністю розміщена у внутрішній області багатокутника P. На рисунку 2.11 (а) вершина позначена червоним кольором є вухом, а на рисунку 2.11 (б, в) – ні. Розглянемо вершини багатокутника в порядку обходу. Кожній вершині присвоємо індекс. Якщо вершина v_i є вухом, тоді будуємо діагональ $v_{i-1}\, v_{i+1}$ і відрізаємо трикутник $\Delta v_{i-1} v_i v_{i+1}$ від P. Далі переходимо в порядку обходу до наступної вершини v_{i+1}.

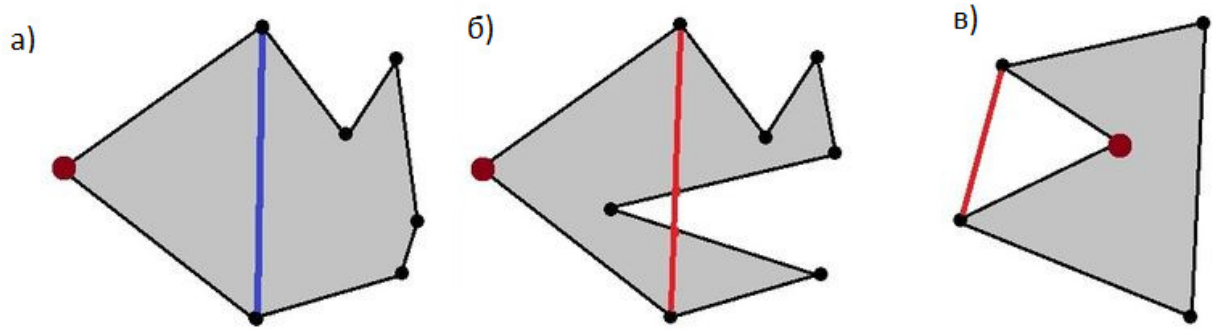


Рисунок 2.11 – Визначення вершини – вуха (випадок а – вухо, випадок б, в – ні)

Перед тим як розглянути вершину як вухо, треба перевірити її на опуклість. Інакше відпадає потреба в розгляді. Це можна зробити використавши лівий поворот. Цю перевірку вершини можна здійснювати за алгоритмом належності точки багатокутнику (у нашому випадку трикутнику). На рисунку 2.12 приведено приклад роботи алгоритму.

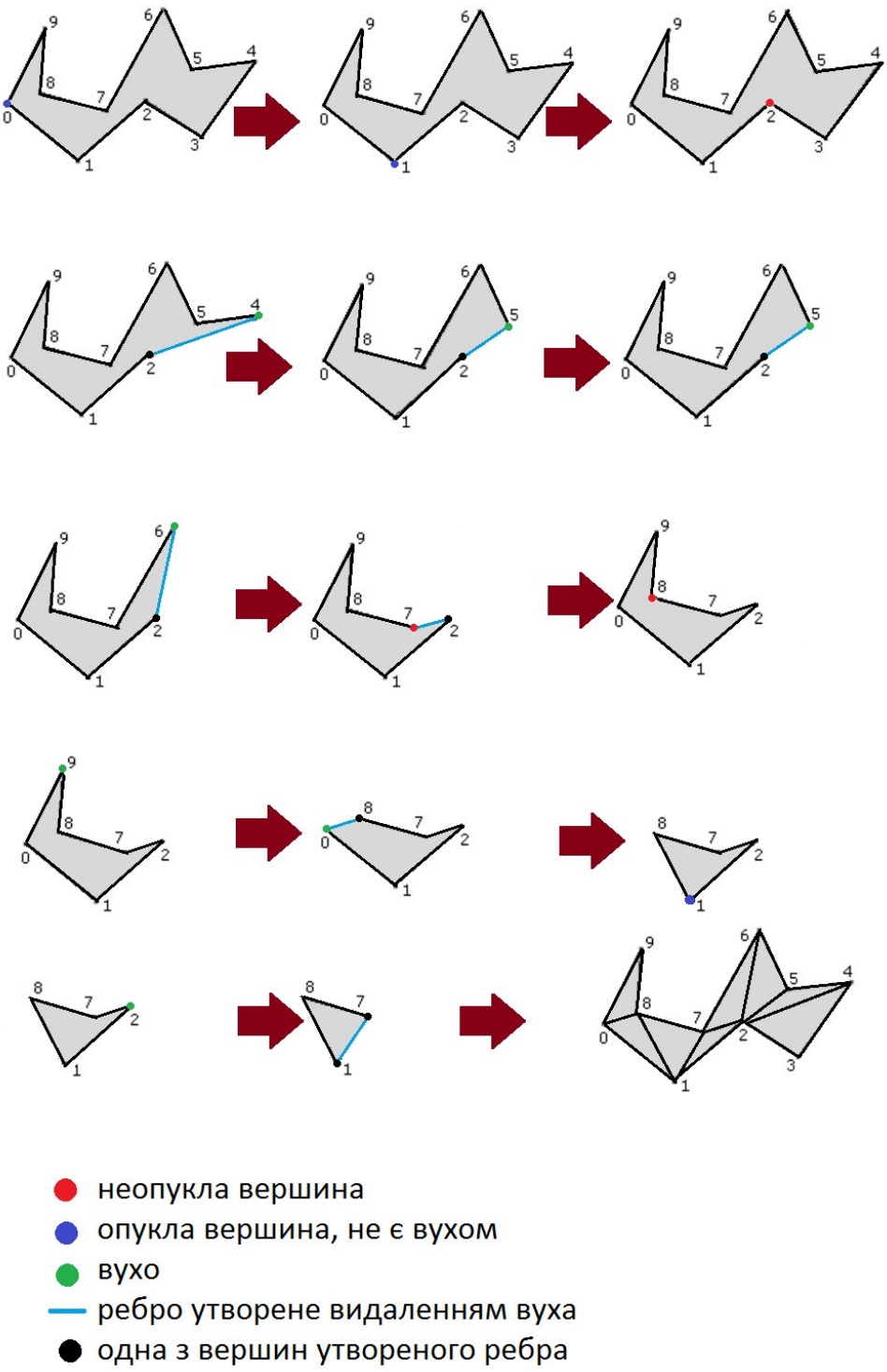


Рисунок 2.12 – Приклад алгоритму відсікання вуха

Псевдокод:

```
List<Vertex> D1 //Колекція вершин вхідної фігури
List<Triangle> T1 //Колекція трикутників (результат)
Int curIndex = 0; // індекс поточної вершини

// Поки у D1 не залишиться 3 точки
```

```
while size_of(D1 > 3){

     Vertex v = D1.get(curIndex);
     isEar = true;
     // Якщо v опукла вершина
     if(v.isConvex()){
          for each Vertex v_i in D1{
               //перевіряємо чи можна провести внутрішню діагональ
               if(v_i != v && v_i != v.prev() && v_i != v.netxt()
               && v_i ∈ Triangle(v.prev(), v, v.next()) ){
                    // діагональ провести не можна
                    // поточна вершина не є вухом
                    isEar = false;
                    break;
               }
          }
          if(isEar){
               // якщо поточна вершина є вухом
               // створюємо трикутник і додаємо до колекції
результатів
               Triangle t = new Triangle(v.prev(), v, v.next());
               T1.add(t);
               //видаляємо вухо
               D1.remove(v);
               //переходимо до наступної вершини
               curIndex++;
          }
     }
}
```

Коли знаходиться вухо, то від багатокутника P відсікається трикутник, вершини якого – саме вухо і його дві суміжні вершини. По завершенні алгоритму, коли всі вуха від P вже відрізані, залишиться тільки один Δ-ник. Алгоритм триангуляції є коректним.

Нехай в багатокутнику є $O(n)$ вух. Коли проходить процес відрізання вух, то суміжні точки теж можуть ставати вухами. При триангуляції утворюються $n-3$ діагоналі, звідси максимальна кількість вершин, які в результаті триангуляції можуть бути вухами $2n-6$. Визначення належності вершини до вух проходить за $O(n)$, адже для однієї точки визначається приналежність за $O(1)$. Отже, загальний процес відрізання вух триватиме $O(n^2)$. Будуємо списки ребер і вершин. Їх є дваОскільки додавання ребра і видалення вершини в працює за однаковий час, тому загальний час обробки $O(n^2)$.

2.5 Алгоритми кусково-частинної декомпозиції

Розглянемо алгоритм M. J. Atallah, який використовується при знаходженні відстані Хаусдорфа для багатокутників що перетинаються. Він грунтується на знаходженні фігури, яку позначимо P, яку визначено (2.1).

Завдання визначення області P можна вирішити алгоритмом для знаходження точок перетину двох опуклих багатокутників запропонованим авторами [53-54].

Нехай маємо два опуклі полігони P і Q. Необхідно обчислити область їх перетину $P \cap Q$. Припустимо, що два полігони перетинаються невироджено, окрім за винятком особливо обумовлених випадків. Перетинатися невироджено – це означає, що перетин двох ребер здійснюється в одній єдиній точці, і ця точка не є вершиною якого-небудь полігону. Згідно такого припущення, постійно маємо, що полігон складається з ланцюжків з P і Q, що чергуються. Усяка пара послідовних ланцюжків буде з'єднуватися в точці перетину кордонів полігонів P і Q (рисунок 2.13).

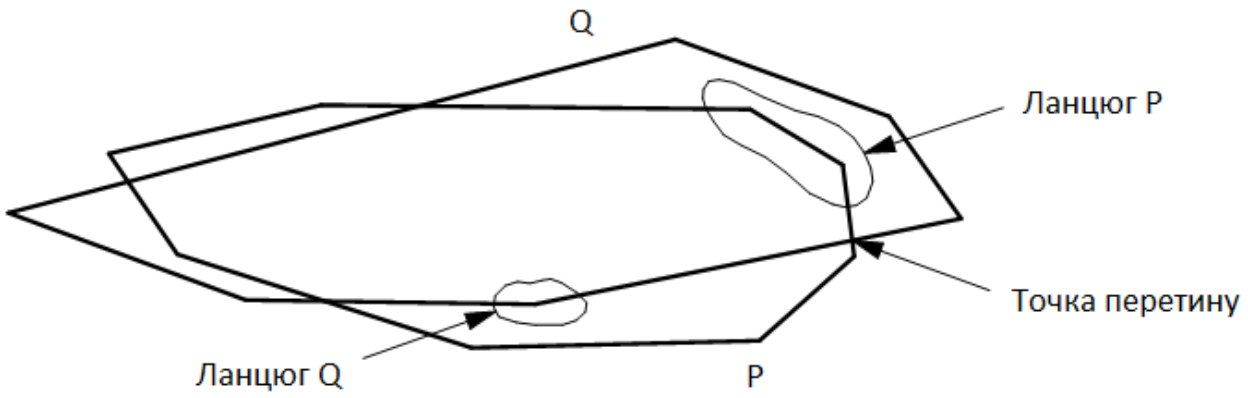


Рисунок 2.13 – Структура багатокутника перетину $P \cap Q$

Нехай є на вході два задані опуклі полігони P і Q. Якщо їх накласти, то за алгоритмом маємо вікно на ребрі полігону P, позначимо його p і друге вікно на q ребрі полігону Q. Ці вікна просуваємо вздовж меж полігону і формуємо полігон перетину $P \cap Q$. Вікна мовби штовхають один одного вздовж границі своїх відповідних полігонів. Іншими словами одне вікно закріплене, а друге крутиться в напрямку за годинниковою стрілкою для пошуку точок перетину ребер. Полігон перетину буде сформованим тоді, коли деяка точка перетину буде виявлена заново. В протилежному випадку, якщо після достатнього числа ітерацій, не буде виявлено жодної точки перетину, то означає, що границі полігонів не перетинаються. Це може статися у тому випадку, якщо один полігон знаходиться всередині іншого або вони зовсім не перетинаються [55].

Для пояснення алгоритму введемо поняття серпа. Серпом буде та частина полігону, яка виступає над полігоном перетину. Наприклад, на рисунку 2.14 показані вісім серпів (вісім затінених полігонів). Кожен з серпів обмежений ланцюжком, взятим від полігону P і ланцюжком від полігону Q, та двома послідовними точками перетину. Внутрішній ланцюжок серпа буде належати полігону перетину. Перетин полігонів завжди буде оточений парним числом серпів. Внутрішні ланцюжки серпів будуть почергово частинами границь полігонів P і Q.

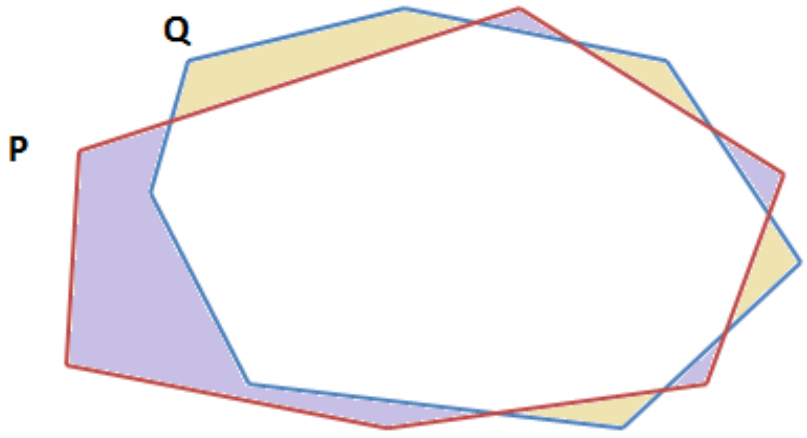


Рисунок 2.14 – Серпи, що оточують полігон перетину

Приведемо алгоритм пошуку полігону перетину у термінах серпів. Він проходить два етапи. На першому етапі вікно p полігону P і вікно q полігону Q пересуваються в напрямку за годинниковою стрілкою до того часу, поки вони не будуть встановлені на ребрах, що належать нараз одному і тому ж серпу. Рух кожного вікна починається з довільної позиції. Позначимо через p ребро у вікні p, а через q ребро у вікні q. Тобто позначимо однаковими символами вікна і ребра. Якщо говоримо "початок p", то це означає точку початку ребра у вікні полігону P, а якщо "перемістити p" то – переміщення вікна полігону P на наступне ребро. Аналогічно це стосується q і Q [55].

На другому етапі вікна p і q також переміщаються за годинниковою стрілкою, але на цього разу вони рухаються від одного серпа до єдиного серпа. Переходячи будь-яке вікно з поточного серпа до наступного, ребра p і q перетинаються в точці перетину, що зв'язує обидва серпи. В цей момент будується полігон перетину. Перед кожним пересуванням p кінцева точка ребра p вписується в полігон перетину, у випадку якщо ребро p припадає на внутрішній ланцюжок поточного серпа. За аналогією перед пересуванням вікна q фіксується кінцева точка ребра q, у випадку якщо ребро p належить внутрішній частині ланцюжка поточного серпа.

Для того, щоб прийняти рішення, яке з вікон потрібно переміщати, в алгоритмі застосовується правило переміщення. Правило використовує наступне означення. Будемо вважати, що ребро *a* націлене на ребро *b*, якщо безмежна пряма лінія, на якій лежить ребро *b*, розташована перед ребром *a*. На рисунку 2.15 жирними лініями показані ребра, які націлені на ребро *q*.

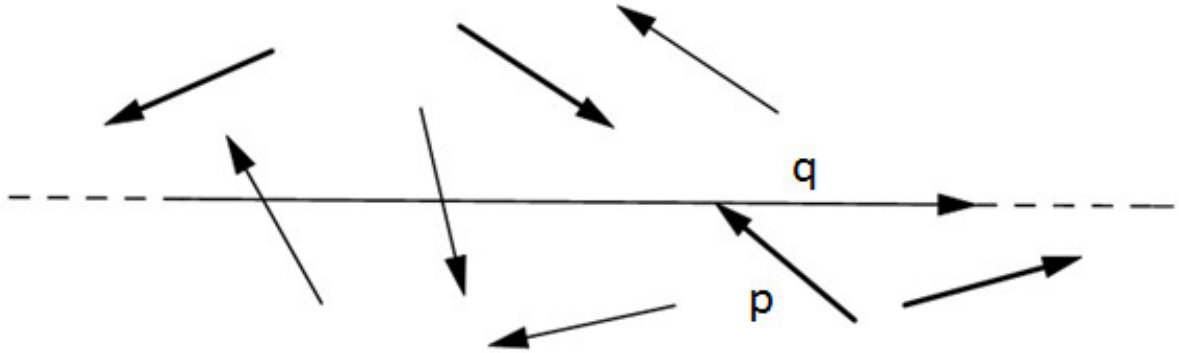


Рисунок 2.15 – Націлені і ненацілені ребра на ребро *q*

Ребро *a* націлене на ребро *b*, якщо скалярний добуток: $a \times b > 0$ і кінцева точка a.dest не лежить праворуч від *b* або $a \times b < 0$ і точка a.dest не лежить зліва від *b*. Скалярний добуток $a \times b \geq 0$ відповідає випадку коли кут між векторами *a* і *b* гострий, $a \times b < 0$ тоді, коли кут між векторами *a* і *b* тупий.

Якщо ребра *a* і *b* колінеарні, то ребро *a* націлене на *b*, якщо кінцева точка a.dest не лежить після *b*. Цей випадок використовується для того, щоб просунути *a* замість *b*, у випадку виродження (коли два ребра перетинаються більш, ніж в одній точці). Якщо *a* буде наздоганяти *b*, тоді жодна точка перетину не буде пропущена.

Сформулюємо правила переміщення таким чином, щоб не пропустити наступну точку перетину. В правилах будемо відрізняти поточне ребро, яке може містити наступну точку перетину, від поточного ребра, яке можливо не може містити наступної точки перетину. В останньому випадку вікно переноситься цілком безпечно. Виділимо чотири ситуації (рисунок 2.16). В даному випадку ребро *a* буде поза ребром *b*, якщо кінцева точка a.dest розташована ліворуч від *b*.

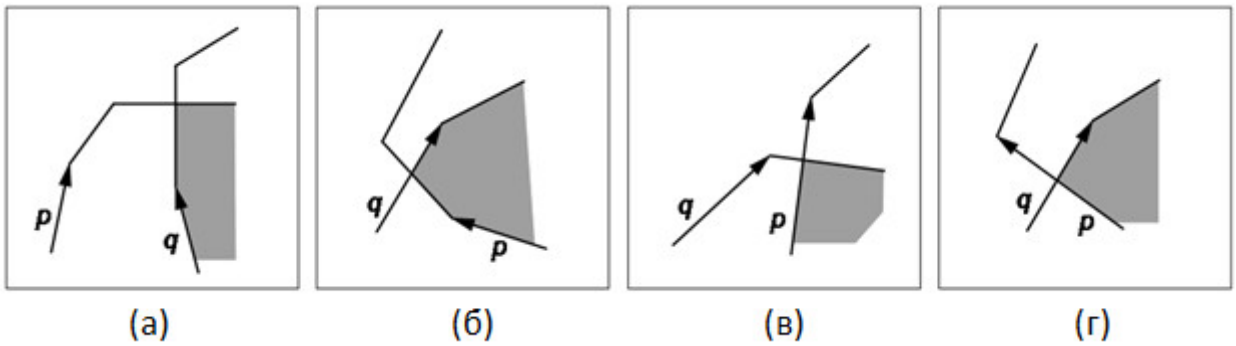


Рисунок 2.16 – Чотири правила переміщення: а) просунути p; б) просунути p, в) просунути q; г) просунути p

Правила переміщення [55]:

Ситуація 1 (рисунок 2.16 (а)). p і q націлені один на другого. Треба перемістити вікно, яке відповідає тому ребру (p або q), яке знаходиться зовні іншого. Для нарисованого прикладу має бути перенесено вікно на ребрі p. Наступна точка перетину не буде лежати на p, оскільки ребро p знаходиться поза полігоном перетину.

Ситуація 2 (рисунок 2.16 (б)). p націлене на q, а q не є націлене на p. Кінцеву точку ребра p треба перенести в полігон перетину, якщо p не знаходиться зовні q і потім перенести вікно p. Наступна точка перетину не буде лежати на q (хоча q може містити деяку точку перетину, якщо q не є зовні від p).

Ситуація 3 (рисунок 2.16 (в)). q націлене на p, але p не є націлене на q. Кінцеву точку ребра q треба перенести в полігон перетину, якщо q не знаходиться зовні p і потім перенести вікно q. Наступна точка перетину не буде лежати на p. Цей випадок є симетричним ситуації 2.

Ситуація 4 (рисунок 2.16 (г)). p і q не є націлені один на одного. Треба перенести те вікно, яке відноситься до ребра, розташованого зовні від іншого. На

рисунку 2.16 (г) потрібно перенести вікно p, тому, що воно знаходиться зовні від ребра q.

Робота описаного алгоритму продемонстрована на рисунку 2.17. Кожне ребро має відмітку i, яка відповідає кроку обробки i. Деякі ребра, що обробляються двічі, мають подвійну позначку. Два початкових ребра мають позначку 0. Фаза 2, коли два поточних ребра належать одному і тому ж серпу, для цього рисунку починається після трьох ітерацій.

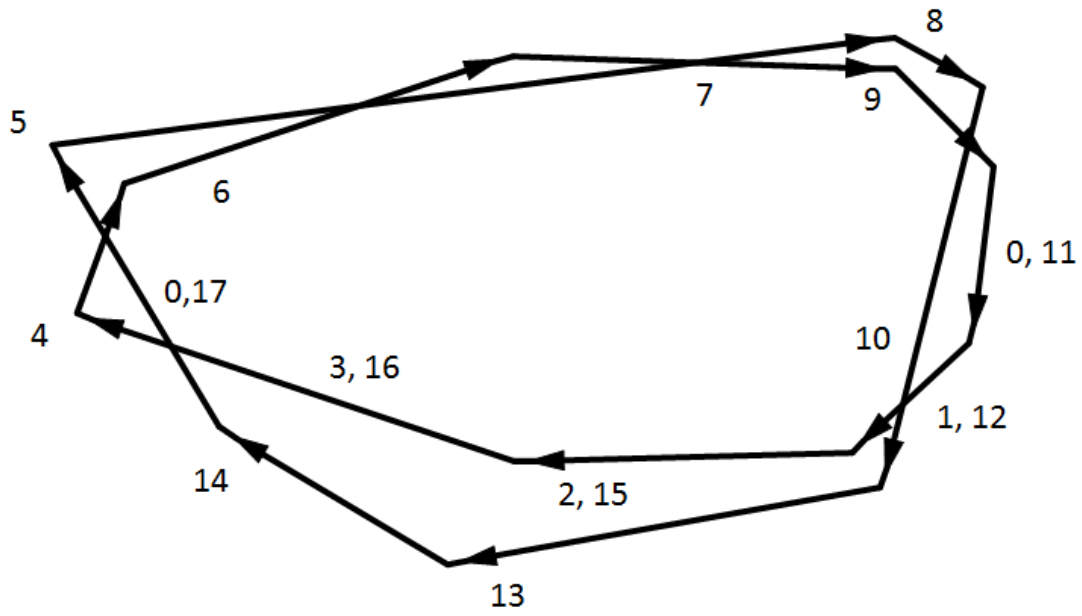


Рисунок 2.17 – Ілюстрація кроків алгоритму

З алгоритму випливає, що перша фаза (пошук першої точки перетину) виконується звичайним перебором і виконується максимум за $M * N$ ітерацій, де M – кількість вершин першого полігону, а N – другого. Фаза друга завжди виконується за $M + N$ ітерацій. Звідси, $M * N + M + N$ є загальною максимальною кількістю ітерацій.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ

3.1 Вимоги до програмного продукту

При реалізації програмного продукту повинні бути враховані вимоги до функціональних характеристик, надійності проекту, параметрів технічних засобів, інформаційної та програмної сумісності.

Функціональні вимоги:

– підтримка зображень форматів png, jpg, tif, будь якого розміру;

– визначення границь областей зображень;

– спрощення (проріджения) границь областей зображень, без значного спотворення;

– розбиття не опуклих областей на опуклі частини;

– визначення метричних відстаней між областями зображень;

Нефункціональні вимоги:

– можливість використання програмного продукту як модуль до інших систем, зокрема графічного редактора ImageJ.

– кросплатформеність;

Вимоги до надійності:

– програма не повинна завершуватись аварійно;

– програма не повинна призводити до зависання операційної системи;

– повне і безпомилкове виконання заявлених функцій;

Мінімальні системні вимоги до ПК:

– ОС: Windows(Vista, 7, 8.x (Desktop), 10, Server 2008, Server 2012), Mac OS (Mac OS X 10.7.3 (Lion) і пізніші версії), Linux (Linux 5.5+, Red Hat 5.5+, Suse Server 10.x+, Ubuntu 10.04 і вище);

– RAM: 512MB

– Дисковий простір: 800 MB

– Процесор: 800 MHz Intel Pentium III або 800 MHz AMD Athlon або 1.2GHz Intel Celeron або 1.2 GHz AMD Duron.

Рекомендовані системні вимоги до ПК:

– ОС: Windows(Vista, 7, 8.x (Desktop), 10, Server 2008, Server 2012), Mac OS (Mac OS X 10.7.3 (Lion) і пізніші версії), Linux (Linux 5.5+, Red Hat 5.5+, Suse Server 10.x+, Ubuntu 10.04 і вище);

– RAM: 1 GB

– Дисковий простір: 1 GB

– Процесор: Intel Core 2 Duo 1,8 GHz (AMD Athlon II X2 240 чи аналогічний).

3.2 Розробка архітектури системи

Реалізація прикладної програмної системи, спроектованої за допомогою UML, починається з реалізації класів, визначених на етапі проектування. При цьому важливо зберегти значення імен класів, властивостей та методів.

Основним компонентом програми є бібліотеки OpenCV [56]. OpenCV являє собою бібліотеку функцій та алгоритмів комп'ютерного зору, обробки зображень і чисельних алгоритмів загального призначення. Код бібліотеки є відкритим. OpenCV вміщує засоби для оброблення і аналізу вмістимого зображень. Основними задачами є: розпізнавання об'єктів на фотографіях (наприклад, особи і фігури людей, текст тощо); відстежування руху об'єктів, перетворення зображень, застосування методів машинного навчання і знаходження загальних елементів на різних зображеннях. Основне завдання, яке покладається на Open CV, в даній програмі – це визначення контурів областей зображень.

Клас PolySimplifier призначений для прорідження контурів областей, містить у собі тільки статичні методи у яких реалізовані різні алгоритми спрощення (таблиця 3.1). А саме алгоритм Дугласа-Рамера-Пекера, Вісвалінгема-Ваєтта, і простий алгоритм який відкидає точки з певним кроком.

Таблиця 3.1 – Клас PolySimplifier

Методи	
Модифікатор і тип	Метод і опис
`public static MatOfPoint`	`reduceSimple(MatOfPoint mop, int step)` Повертає проріджений контур, залишаючи точки з певним кроком. `mop` – вхідний контур; `step` – крок, через який буде додаватись точка до вихідного контуру.
`public static List<Point>`	`reduceRDP(List<Point> contour, double epsilon)` Повертає набір точок прорідженого контуру. Прорідження реалізоване на алгоритмі Рамера-Дугласа-Пекера. `contour` – набір точок вхідного контуру; `epsilon` – максимальна відстань відхилення від перпендикуляра.
`public static List<Point>`	`reduceRDP(List<Point> contour, double numToKeep)` Повертає набір точок прорідженого контуру. Прорідження реалізоване на алгоритмі Вісвалінгема-Ваєтта `contour` – набір точок вхідного контуру; `numToKeep` – кількість точок, яка має залишитись після прорідження.

Клас Distance призначений для визначення метричної відстані між границями областей. Тут реалізовані алгоритми знаходження відстані Хаусдорфа [57-58] методом звичайного перебору кожної точки, а також модифіковані алгоритми для опуклих границь, які перетинаються або не перетинаються (таблиця 3.2). Алгоритми описані у роботі [26]. Також у класі

містяться приватні допоміжні методи, наприклад знаходження відхилення точки від множини, і множини від множини.

Таблиця 3.2 – Клас Distance

Методи	
Модифікатор і тип	Метод і опис
`public static double`	`getHausdorffDistance(List<Point> c1, List<Point c2>)` Знаходить відстань Хаусдорфа між двома багатокутниками. Реалізований на стандартному алгоритмі. `c1,c2` – набори точок границь областей;
`public static double`	`getHausdorffDistanceModJoined(List<Point> c1, List<Point c2>)` Знаходить відстань Хаусдорфа між двома багатокутниками, які обов'язково опуклі і перетинаються. Реалізований на алгоритмі М. Дж. Аталаха. `c1,c2` – набори точок границь областей;
`public static double`	`getHausdorffDistanceModDisjoined(List<Point> c1, List<Point c2>)` Знаходить відстань Хаусдорфа між двома багатокутниками, які обов'язково опуклі і не перетинаються. Реалізований на алгоритмі М. Дж. Аталаха. `c1,c2` – набори точок границь областей;
`private static double`	`getDeviation(Point p, List<Point> contour)` Знаходить відхилення точки від множини точок в метриці Хаусдорфа. `p` – точка; `contour` – множина точок контура.

Продовження таблиці 3.2 – Клас Distance

`private static double`	`getDeviation(List<Point> c1, List<Point> c2)` Знаходить відхилення множини точок від множини точок в метриці Хаусдорфа. `c1,c2` – множини точок границь областей;
`private static double`	`getAttalahDisjointDeviation(List<Point> c1, List<Point> c2)` Знаходить відхилення множини точок від множини точок в метриці Хаусдорфа. Алгоритм М. Дж. Аталаха для опуклих багатокутників, що не перетинаються. `c1,c2` – множини точок границь областей;
`private static int`	`getNextVertexIndex(int curPosition, int polySize, boolean clockwise)` Знаходить індекс наступної вершини множини. `curPosition` – індекс поточної вершини; `polySize` – кількість точок множини; `clockwise` – прапорець напрямку обходу контура.
`private static int`	`increment(int index, int size)` Знаходить індекс наступної вершини у множині. `index` – індекс поточної вершини; `size` - кількість точок множини.
`private static int`	`decrement(int index, int size)` Знаходить індекс попередньої вершини у множині. `index` – індекс поточної вершини; `size` - кількість точок множини.

Клас GeometryUtils містить у собі набір статичних методів для різноманітних геометричних обчислень (таблиця 3.3). Наприклад знаходження координат основи перпендикуляра, обчислення евклідової відстані між двома точками, визначення площі трикутника за координатами трьох точок, методи для різноманітних векторних обчислень (скалярний і векторний добутки, знаходження взаєморозташування точки і напрямку вектора, чи

взаєморозташування напрямків векторів). Ці методи використовуються іншими класами, наприклад Distance чи PolySimplifier.

Таблиця 3.3 – Клас GeometryUtils

Методи	
Модифікатор і тип	Метод і опис
`public static Point`	`getPerpendicularBasePoint(Point fromPoint, Point startSeg, Point endSeg)` Знаходить точку основи перпендикуляра. І `null` якщо такої точки не існує. `fromPoint` – точка з якої опускається перпендикуляр; `startSeg`, `endSeg` – точки відрізку на який опускається перпендикуляр.
`public static double`	`getEuclideanDistance(Point p1, Point p2)` Знаходить евклідову відстань між двома точками. `p1, p2` – точки, між якими шукається відстнь;
`public static double`	`getTriangleArea(Point a, Point b, Point c)` Знаходить площу трикутника за координатами вершин (формула Герона). `a, b, c` – вершини трикутника.
`public static boolean`	`isPolyIntersect(List<Point> poly1, List<Point> poly2)` Перевіряє чи перетинаються два багатокутники. `poly1, poly2` - набір точок вершин багатокутників.
`public static boolean`	`isPolyConvex(List<Point> poly)` Перевіряє чи багатокутник є опуклим. `poly` - набір точок вершин багатокутника.
`static public boolean`	`isAimsAt(Point startA, Point endA, Point startB, Point endB)` Перевіряє чи спрямований вектор А на вектор В. `startA, endA` - точки початку і кінця вектора А; `startB, endB` - точки початку і кінця вектора В;

Продовження таблиці 3.3 – Клас GeometryUtils

static public Position	getPointPosition(Point point, Point vectorStart, Point vectorEnd) Знаходить положення точки відносно вектора. point – точка; vectorStart,vectorEnd – точки початку і кінця вектора;
static public Point	getCrossPoint(Point startA, Point endA, Point startB, Point endB) Знаходить точку перетину двох відрізків. І null якщо такої точки не існує. startA, endA – точки початку і кінця вектора А; startB, endB – точки початку і кінця вектора В.
static public boolean	isCollinear(Point startA, Point endA, Point startB, Point endB) Визначає чи вектори колінеарні. startA, endA – точки початку і кінця вектора А; startB, endB – точки початку і кінця вектора В.
static public double	getScalarProduct(Point startA, Point endA, Point startB, Point endB) Знаходить скалярний добуток векторів. startA, endA – точки початку і кінця вектора А; startB, endB – точки початку і кінця вектора В.
static public double	getVectorProduct(Point startA, Point endA, Point startB, Point endB) Знаходить векторний добуток. startA, endA – точки початку і кінця вектора А; startB, endB – точки початку і кінця вектора В.
static public double	getVectorProduct(Point vectorA, Point vectorB) Знаходить векторний добуток. vectorA, vectorB – вектори.
static public boolean	isVectorsIntersect(Point startA, Point endA, Point startB, Point endB) Визначає чи перетинаються напрямки векторів. startA, endA – точки початку і кінця вектора А; startB, endB – точки початку і кінця вектора В.

Продовження таблиці 3.3 – Клас GeometryUtils

`static public boolean`	`isPointLiesOnSegment(Point point, Point segStart, Point segEnd)` Визначає чи лежить точка на відрізку. `point` – точка; `segStart`, `segEnd` – точки відрізка;

Перерахування (enum) Position визначає всі можливі положення точки відносно напрямку вектора заданого двома точками (таблиця 3.4).

Таблиця 3.4 – Перерахування (enum) Position

Значення	Опис
`LEFT`	Зліва від напрямку вектора
`RIGHT`	Справа від напрямку вектора
`AHEAD`	Спереду вектора
`BEHIND`	Позаду вектора
`LIES_ON`	На векторі

Класи Decompositor (таблиця 3.5) призначений для виконання шматково-частинною декомпозиції, яка потрібна для знаходження відстані між опуклими багатокутниками, що перетинаються, у алгоритмі запропонований М. Дж. Аталахом у роботі [26].

Список об’єктів Chain (таблиця 3.6) є результатом шматково-частинної декопозиції. Він містить у собі списки внутрішніх і зовнішніх точок серпа.

Таблиця 3.5 – Клас Decompositor

Поля	
Тип і назва	Опис
`List<Point> A, B`	Набір точок багатокутника A і B
`int sizeA, sizeB`	Кількість точок багатокутника A і B
`int curIndexPointA, curIndexPointB`	Індекс поточної точки багатокутника A і B
`Point startA, endA`	Точки початку і кінця поточного ребра A
`Point startB, endB`	Точки початку і кінця поточного ребра B
`Chain currentChain`	Поточний ланцюг
`ChainType chainType`	Тип поточного ланцюга
Конструктори	
`Decompositor(List<Point> pointsA, List<Point> pointsB)`	`pointsA,pointsB` – набори точок границь областей.
Методи	
Модифікатор і тип	Опис
`private void`	`findFirstCrossPoint()` Знаходить першу точку перетинку ребер багатокутників
`public List<Chain>`	`decompose()` Виконує шматково-частинну декомпозицію. Алгоритм О'Рурка.
`private void`	`initSegments()` Ініціалізує поточні ребра багатокутників
`private boolean`	`isAExternal()` Визначає чи ребро багатокутника A в поточному вікні є зовнішнім
`private boolean`	`isBExternal()` Визначає чи ребро багатокутника B в поточному вікні є зовнішнім
`private void`	`advanceA()` Просуває поточне ребро багатокутника A у вікні
`private void`	`advanceB()` Просуває поточне ребро багатокутника B у вікні

Таблиця 3.6 – Клас Chain

Поля	
Тип і назва	Опис
`List<Point> internal`	Набір точок внутрішнього ланцюга серпа
`List<Point> external`	Набір точок зовнішнього ланцюга серпа
Методи	
Модифікатор і тип	Опис
`private void`	`addInternalPoint(Point point)` Додає точку до внутрішнього ланцюга
`public void`	`addExternalPoint(Point point)` Додає точку до зовнішнього ланцюга
`public List<Point>`	`getInternalChain()` Повертає множину точок внутрішнього ланцюга
`public List<Point>`	`getExternalChain()` Повертає множину точок зовнішнього ланцюга
`public MatOfPoint`	`getContour()` Повертає контур серпа

Перерахування (enum) ChainType визначає можливі типи серпів утворених внаслідок частинно-шматкової декомпозиції (таблиця 3.7).

Таблиця 3.7 – Перерахування (enum) ChainType

Значення	Опис
`UNKNOWN`	Невідомий, невизначений
`A_EXTERNAL`	Зовнішній ланцюг це ребра багатокутника A
`B_EXTERNAL`	Зовнішній ланцюг це ребра багатокутника B

Можливість перетворення не опуклої фігури в опуклу, а точніше тріангуляція фігури, забезпечується класом TrianglePartition (таблиця 3.8). Варто

відзначити, що вхідна фігура має бути не обов’язково опукла. На вихід отримується колекція трикутників, де кожен трикутник є об’єктом класу MatOfPoint і містить у собі три точки. Кількість трикутників рівна N-2, де N – кількість вершин вхідної фігури.

Таблиця 3.8 – Клас TrianglePartition

Методи	
Модифікатор і тип	Метод і опис
`public static List<MatOfPoint2f>`	`splitPoly(List<Point> list)` Повертає колецію трикутників (множин із трьох точок). `list` – множина точок фігури.
`private static boolean`	`isConvex(Point point, Point prevPoint, Point nextPoint)` Визначає чи поточна точка є випуклою.
`static private int`	`nextPoint(int curPointIndex, int pointsNum)` Повертає індекс наступної точки.
`static private int`	`prevPoint(int curPointIndex, int pointsNum)` Повертає індекс попередньої точки.

Також для відлагоджування і відображення результатів, передбачені класи GUI та ImageConverter (таблиця 3.9 і 3.10). В класі GUI зосереджені статичні методи які дозволяють виводити на дисплей різні елементи: контури, точки, відрізки, вектори, результати тріангуляції і т.д. А клас ImageConverter призначений для перетворення зображення типу Mat, що використовує OpenCV в об'єкт типу BufferedImage, що використовується в java.

Головним класом, з якого починається запуск програми є QualityEstimator. В якому виконується завантаження зображення, знаходження контурів, спрощення контурів, знаходження відстані між об’єктами та видача результатів. Діаграма класів розроблюваної системи знаходиться у додатку А.

Таблиця 3.9 – Клас GUI

Поля	
Тип і назва	Опис
`static final int PADDING`	Відступ у фреймі в пікселях від границь зображення
Методи	
Модифікатор і тип	Опис
`public static void`	`displayImage(Image img)` Виводить зображення на екран.
`public static void`	`displayImage(Image img, String title, boolean scaleToScreenSize)` Виводить зображення на екран. `title` – заголовок вікна; `scaleToScreenSize` – визначає, чи масштабувати зображення до розмірів дисплею.
`private static Image`	`scaleImage2ScreenSize(Image img)` Масштабує зображення до розмірів дисплею.
`public static void`	`displayContours(int height, int width, List<MatOfPoint> mop, String name)` Виводить на дисплей множину контурів. `int screenHeight, int screenWidth` – розміри зображення в пік селях. `List<MatOfPoint> mop` – колекція контурів. `name` – заголовок вікна в якому буде виведене зображення.
`public static void`	`displayContour(int height, int width , MatOfPoint mop, String name)` Виводить на екран контур.
`public static void`	`displayContour(int screenHeight, int screenWidth, List<Point> points, String name)` Виводить на екран контур.
`public static void`	`displayPoly(int screenHeight, int screenWidth, List<Point> points, String name)` Виводить на екран контур багатокутника.

Продовження таблиці 3.9

public static void	displayPolyPartition(int height, int width, List<MatOfPoint2f> parts, String name) Виводить на екран тріангульований багатокутник.
public static void	displayVectors(int height, int width, Point[] points, String name) Виводить на екран points.length/2 векторів. points – масив точок, де кожних 2 точки належать 1 вектору. Обов'язкова парна кількість елементів.

Таблиця 3.10 – Клас ImageConverter

Методи	
Модифікатор і тип	Метод і опис
public static BufferedImage	Mat2BufferedImage(Mat m) Повертає зображення типу BufferedImage, конвертуючи вхідне типу Mat.

3.3 Тестування функціональних вимог системи

Для початку перевіримо завантаження зображень найпоширеніших на сьогодні форматів png, jpg, tif. Уривок коду для завантаження зображення засобами open CV та виведення його на екран наведено в лістингу 3.1, а результат виконання на рисунку 3.1.

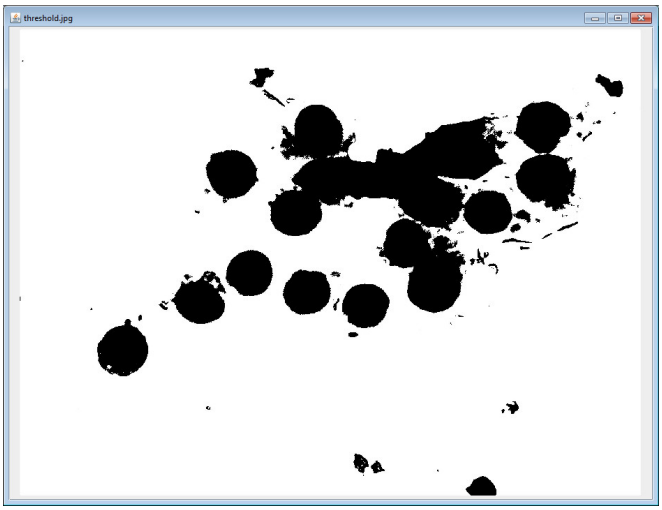


Рисунок 3.1 – Завантаження зображення і вивід його на екран

Код програми для завантаження зображення і виведення його на екран:

```
Mat img = Imgcodecs.imread("images\\TS_07_04_10_47_24.jpg");
GUI.displayImage(ImageConverter.Mat2BufferedImage(img), "TS_07_04_10_47_24", true);
```

Тепер коли в нас є завантажене зображення потрібно знайти контури його областей. Цю дію також виконаємо за допомогою OpenCV (рисунок 3.2).

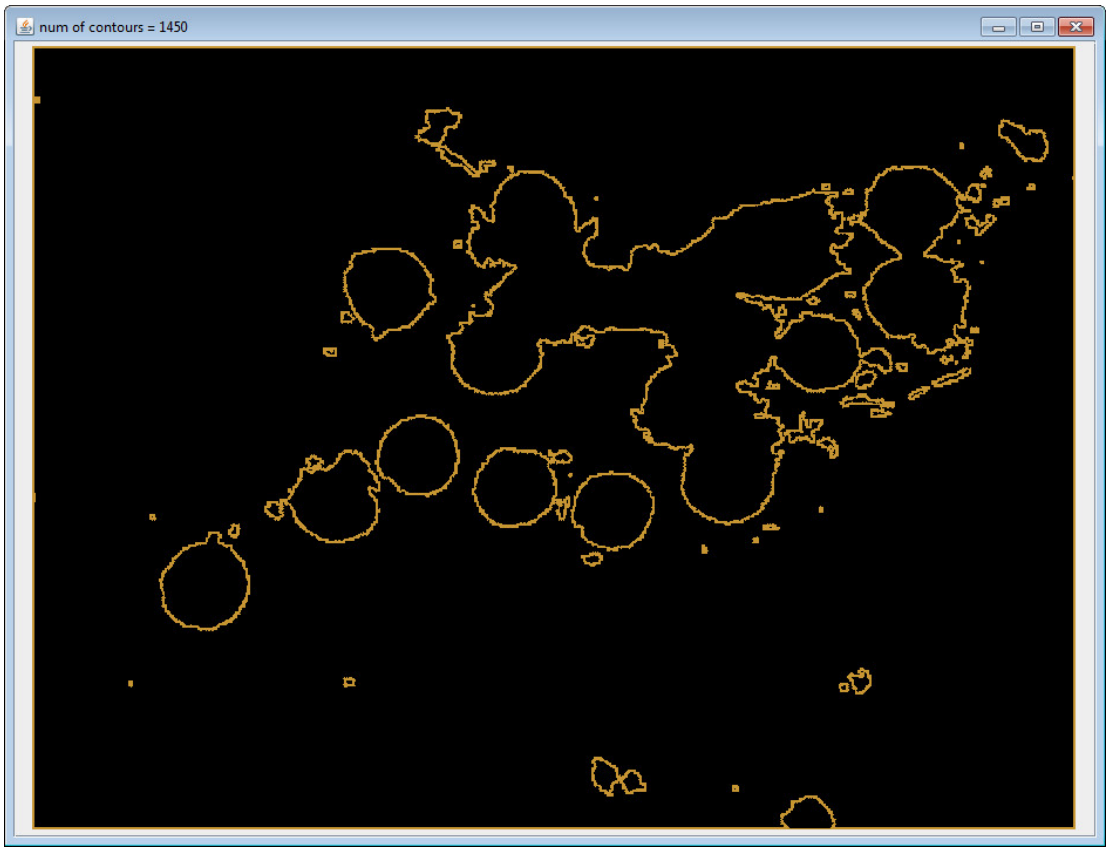


Рисунок 3.2 – Знайдені контури на зображенні

Код для знаходження контурів і виведення їх на екран:

```
List<MatOfPoint> contours1 = getContoursList(img);
GUI.displayContours(img.height(), img.width(), contours1, "num of contours = " +
contours1.size());
```

Зображення отримане після знаходження границь областей (див. рисунок 3.2) містить у собі багато маленьких не потрібних для подальших обчислень областей. Тому відфільтруємо зображення і залишимо тільки великі області (рисунок 3.3).

Код для відсіювання малих областей:

```
List<MatOfPoint> largeContours = new ArrayList<>();

for(int i=0; i< contours1.size(); i++){
    if(contours1.get(i).toList().size() > 1000){
        MatOfPoint mop = new MatOfPoint();
        largeContours.add(contours1.get(i));
    }
}

GUI.displayContours(img.height(), img.width(), largeContours, "num of
contours = " + largeContours.size());
```

Таким способом, було зменшено кількість границь областей з 1450 до 8, що значно пришвидшить обчислення, без втрати важливих об’єктів на зображенні. Тепер детально проаналізуємо одну з областей, а точніше визначимо кількість вершин, що задають її границю (рисунок 3.4). Дана область складається із 1568 точок і має площу 47438,5 пікселів квадратних .

У програмній системі реалізовано 3 методи спрощення контуру. Проаналізуємо їх роботу на час та на спотворення (візуально). Час роботи алгоритмів наведено в таблиці 3.11, зміна площі відносно оригіналу у таблиці 3.12, а результат роботи на рисунку 3.5.

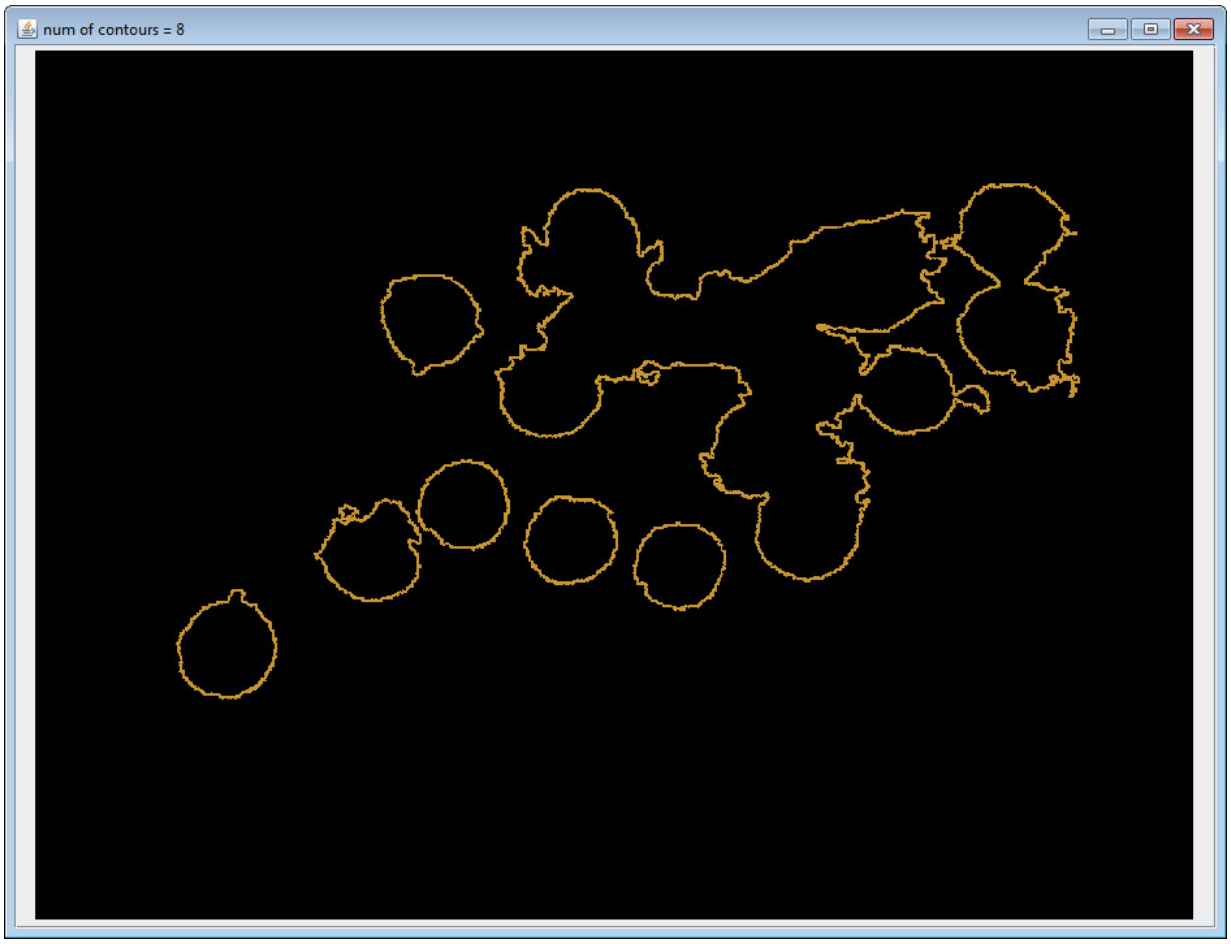


Рисунок 3.3 – Результат відсіювання малих областей

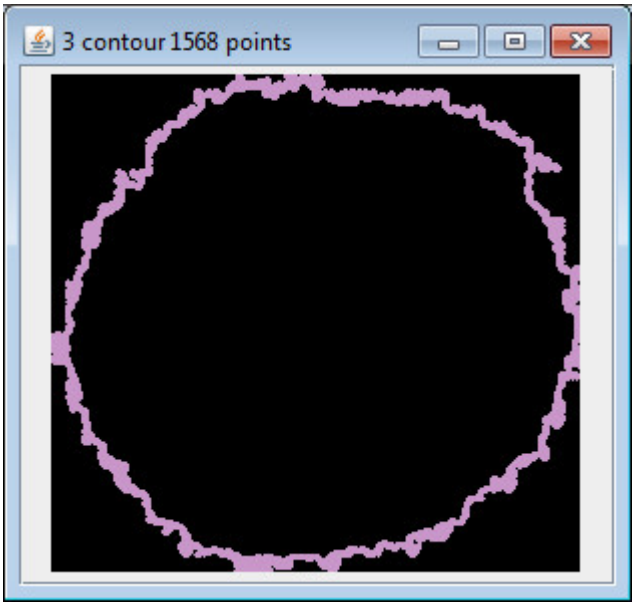


Рисунок 3.4 – Обрана область для спрощення

Таблиця 3.11 – Час роботи алгоритмів спрощення контуру

Алгоритм	500 точок, мс	300 точок, мс	100 точок, мс	25 точок, мс
RDP	18	14	12	10
VW	60	62	70	82

Таблиця 3.12 – Зміна площі

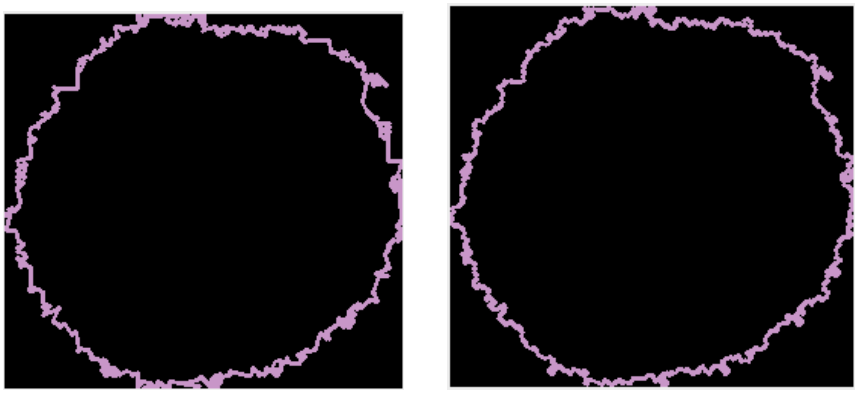
Алгоритм	500 точок	300 точок	100 точок	25 точок
RDP	47865.0 (+0.8%)	48064.0 (+1.3%)	48408.0 (+2.0%)	49313.5 (+3.9%)
VW	47440.5 (+0,0004)	47414.0 (-0,006%)	47333.0 (-0,23%)	47564.0 (+0,26%)

Отже, експериментально досліджено, що швидшим є алгоритм Дугласа-Рамера-Пекера. Але він у свою чергу дає гірші результати, бо він не враховує важливості точки. Також цей алгоритм не дозволяє задати кількість точок до якої буде спрощуватись вихідна фігура. Алгоритм Вісвалінгема-Ваєтта є повільнішим, але він дає кращу якість спрощення, а також дозволяє визначити кількість точок вихідної фігури.

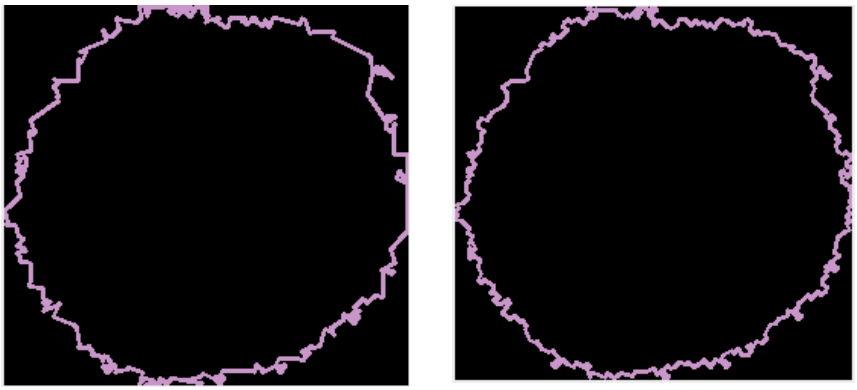
Алгоритм М. Дж. Аталлаха для знаходження відстані у метриці Хаусдорфа працює тільки для опуклих фігур. Неопуклу фігуру можна розбити на декілька опуклих фігур, наприклад на трикутники (тріангуляція). Результат тріангуляції фігури зображено на рисунку 3.6. Недоліком тріангуляції є те, що з фігури розміром N буде отримано N-2 трикутників. Також збільшується обсяг обчислень, бо одна вершина фігури може бути вершиною одразу багатьох трикутників. Перевагою такого способу є швидкодія алгоритму і просто реалізації.

RDP VW

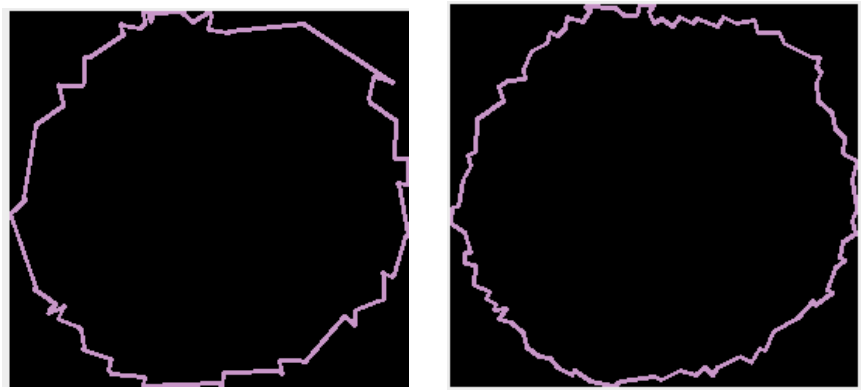
500 точок



300 точок



100 точок



25 точок

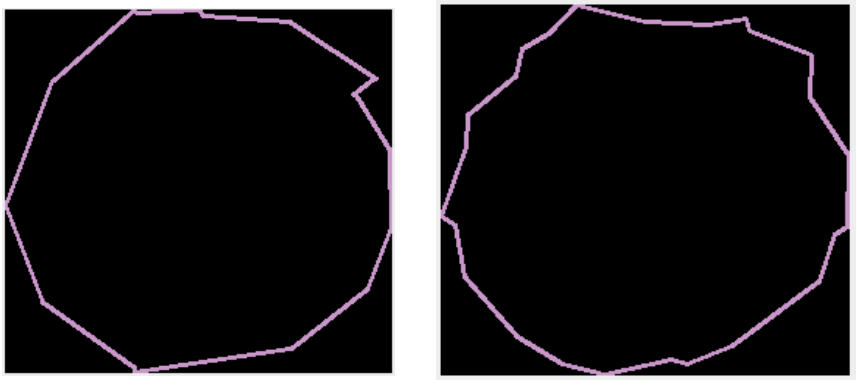


Рисунок 3.5 – Візуальне порівняння алгоритмів спрощення контуру

Вхідне зображення Вихідне зображення

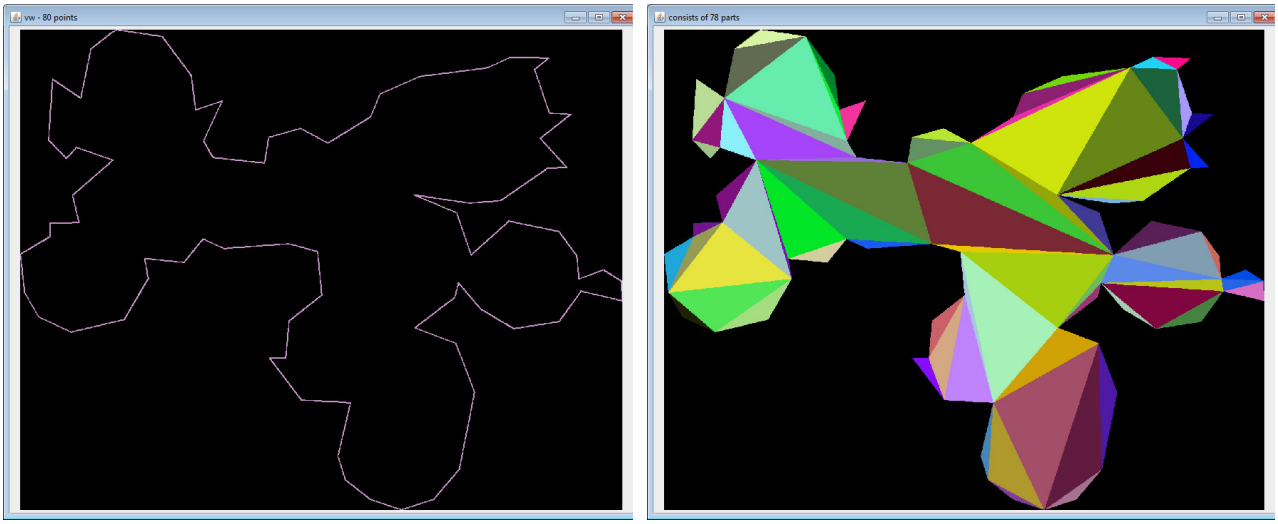


Рисунок 3.6 – Результат застосування тріангуляції

В даній програмній системі реалізовані 3 алгоритми обчислення відстані між областями у метриці Хаусдорфа. Два багатокутники можуть перетинатись, не перетинатись, або повністю співпадати. Результат обчислення відстані між багатокутниками що не перетинаються зображені на рисунку 3.7 та 3.8.

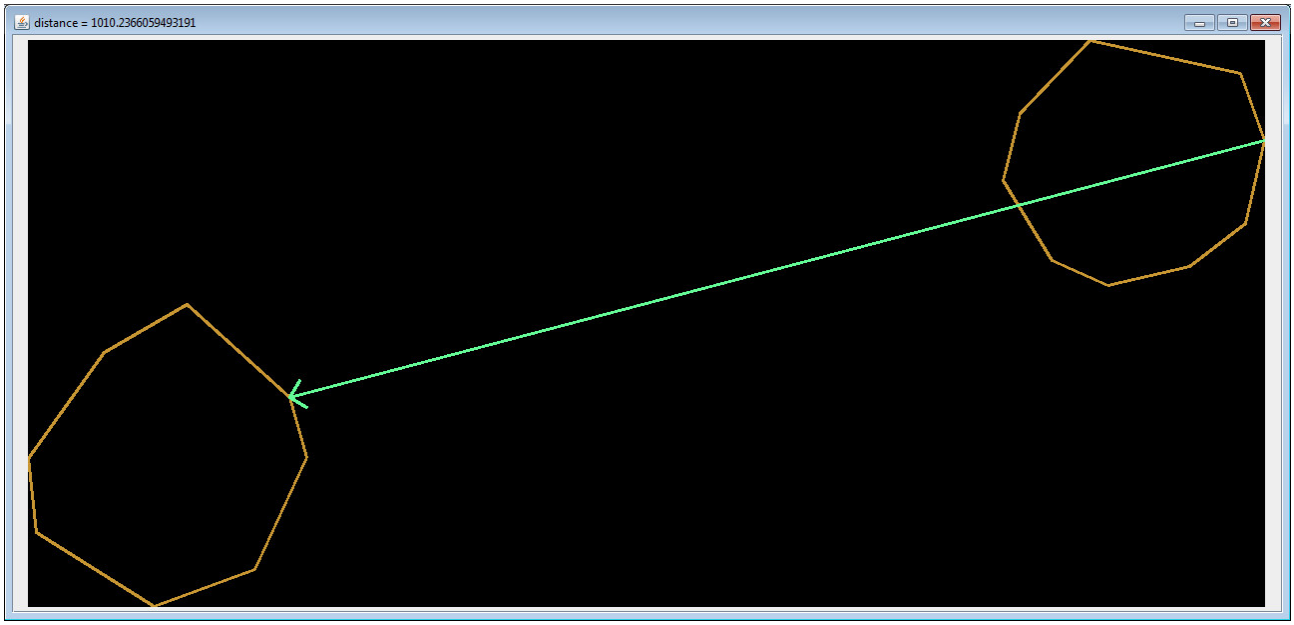


Рисунок 3.7 – Відстань Хаусдорфа між областями що не перетинаються

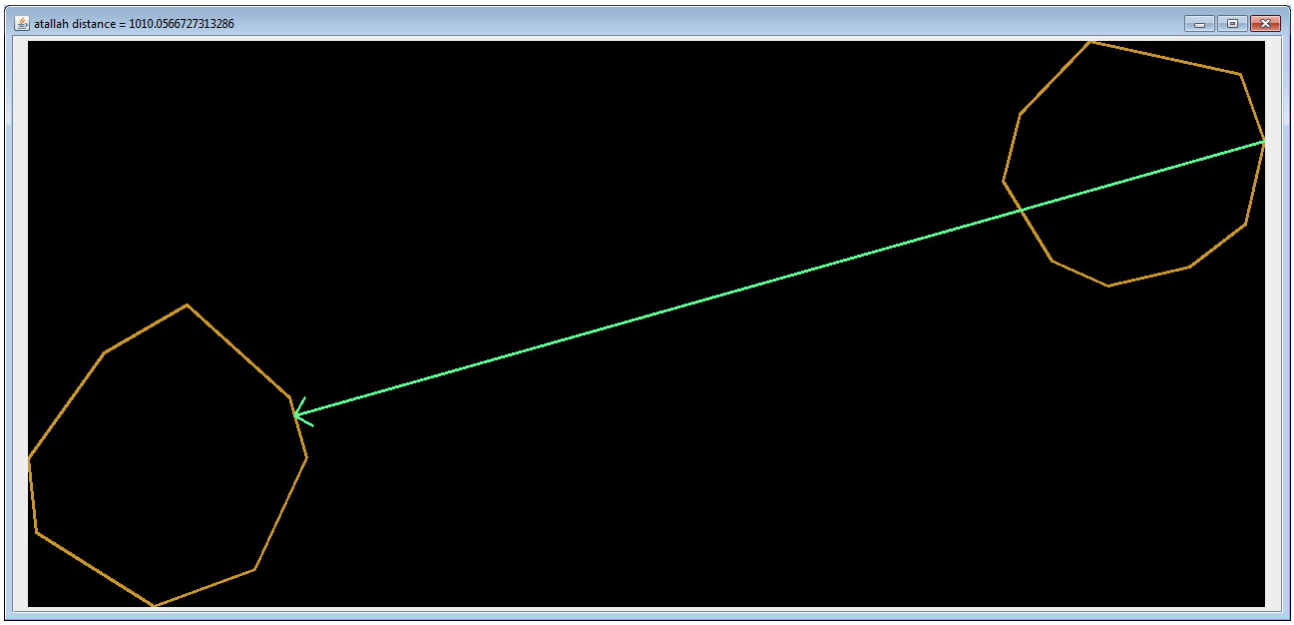


Рисунок 3.8 – Відстань Хаусдорфа (алгоритм М.Дж.Аталлаха)

Повним перебором відстань знаходиться швидше ніж модифікованим. Але модифікований алгоритм дозволяє знайти точнішу відстань, оскільки він враховує не тільки відстань до вершин, але й до ребер, опускаючи на них перпендикуляр.

У випадку коли багатокутники перетинаються, за модифікованим алгоритмом, спочатку виконується шматково-частинна декомпозиція (рисунок 3.9), а потім вже розраховується відстань.

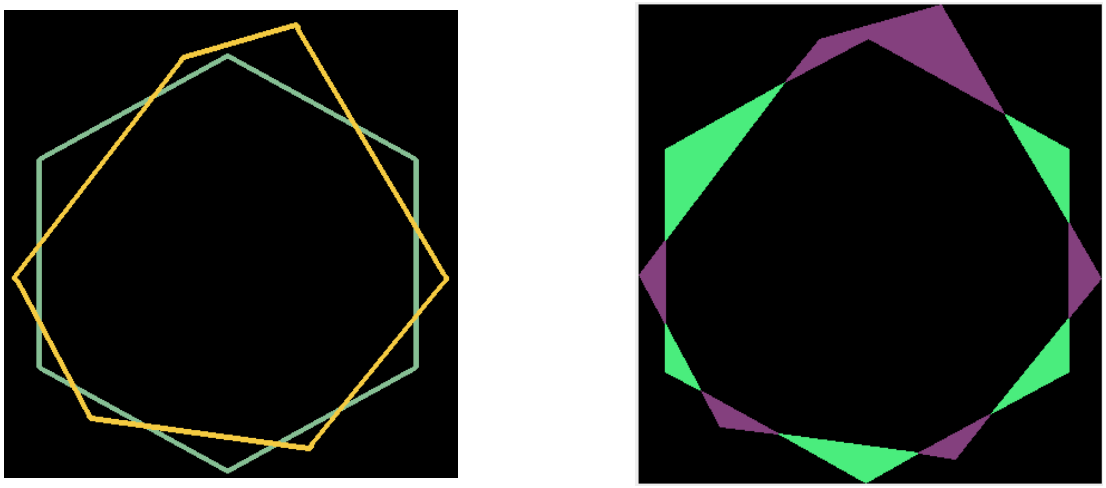


Рисунок 3.9 – Кусково-частинна декомпозиція

Результат обчислення відстані Хаусдорфа стандартним алгоритмом і алгоритмом М.Дж.Аталлаха зображено на рисунку 3.10.

Звичайний перебір Модифікований алгоритм

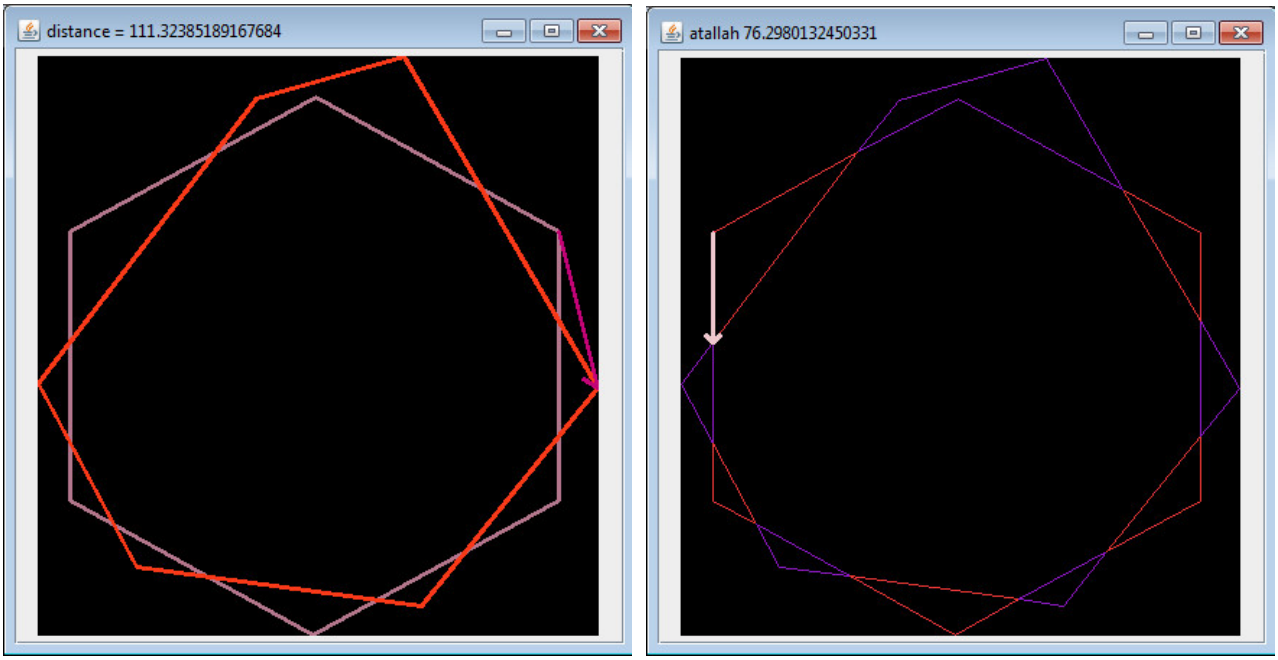


Рисунок 3.10 – Відстань Хаусдорфа між областями що перетинаються

В таблиці 3.13 приведена порівняльна характеристика алгоритмів: алгоритму повного перебору, модифікованого алгоритму для контуру 500 точок і 200 точок.

Отже, враховуючи дані таблиці 3.13, можна зробити висновок, що модифікований алгоритм працює значно швидше алгоритму повного перебору. Але модифікований алгоритм повільно працює з не опуклими фігурами, а також при обчисленні відстані між областями що перетинаються.

Таблиця 3.13 – Порівняльна характеристика алгоритмів

Номер досліду	Алгоритм повного перебору		Модифікований алгоритм	
	Відстань, пікселі	Час, мс	Відстань, пікселі	Час, мс
Контур 500 точок				
1	1030,92	28	1032,32	7
2	318,3	27	337,25	7
3	589,57	25	598,57	5
4	318,79	25	340,15	6
5	755,22	24	755,22	7
6	1223,73	25	1282,99	8
Контур 200 точок				
1	1030,92	16	1033,01	2
2	319,88	15	368	3
3	589,66	17	589,66	2
4	318,79	16	333,51	3
5	754,21	15	754,21	3
6	1199,58	16	1257,91	3

3.4 Комп’ютерні експерименти

Використаємо розроблений програмний засіб для кількісної оцінки якості алгоритмів сегментації зображень. Дослідження проведемо з цитологічними зображеннями. Для цього пропонується визначити відстані в метричному просторі для зображення сегментованого експертом та зображенням сегментованим алгоритмом. Для дослідження було обрано наступні алгоритми: пороговий, вододілу та k-середніх.

Експеримент для кожного зображення проходить наступним чином: на вхідному зображенні (рисунок 3.11) експерт виділяє потрібні області і зображення перетворюється в чорно-біле (рисунок 3.12 б). Далі вхідне зображення сегментується алгоритмом порогової сегментації (рисунок 3.13), алгоритмом вододілу (рисунок 3.14), алгоритмом k-середніх (рисунок 3.15).

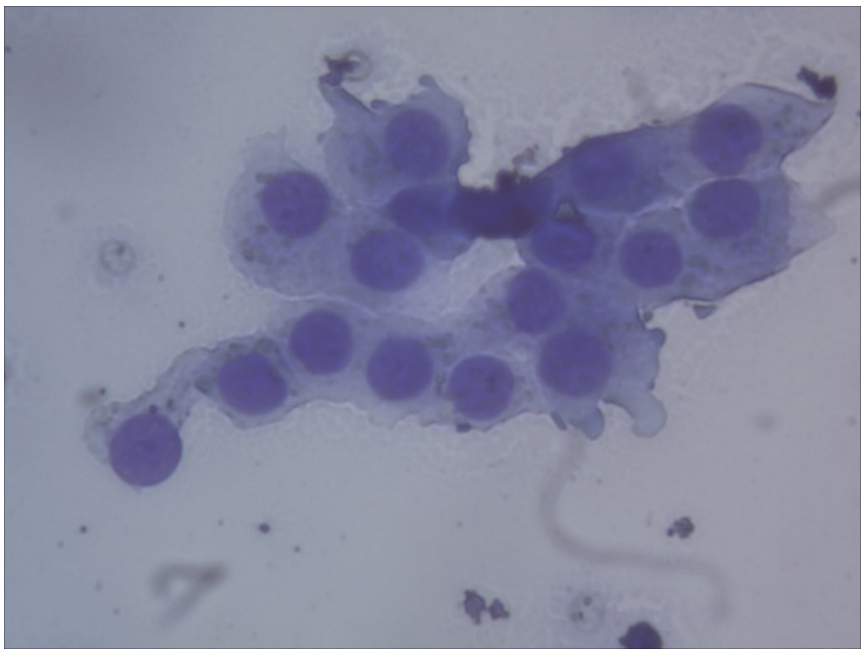


Рисунок 3.11 – Вхідне зображення

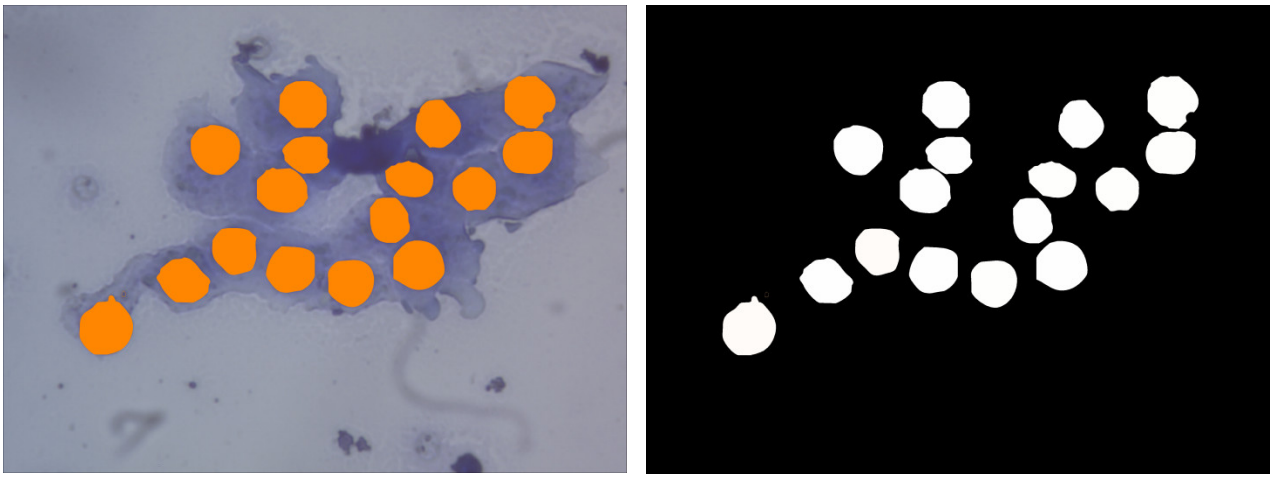


Рисунок 3.12 – Сегментація, що виконана експертом

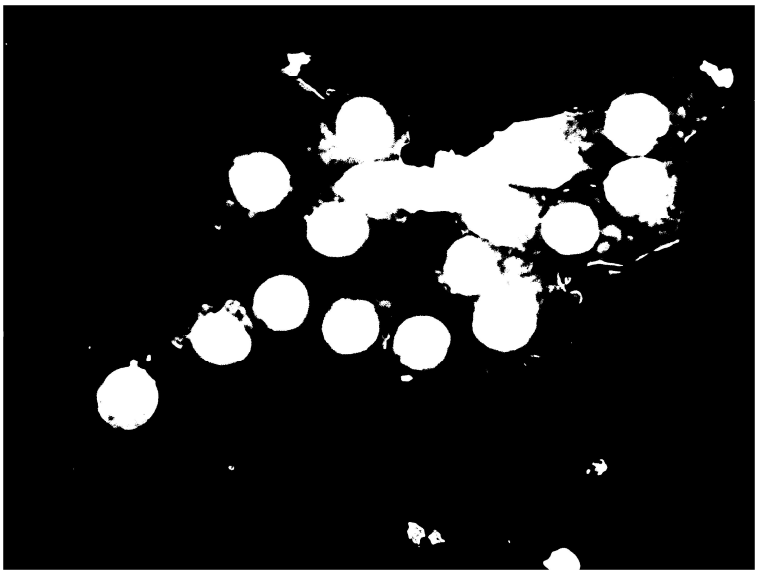


Рисунок 3.13 – Порогова сегментація

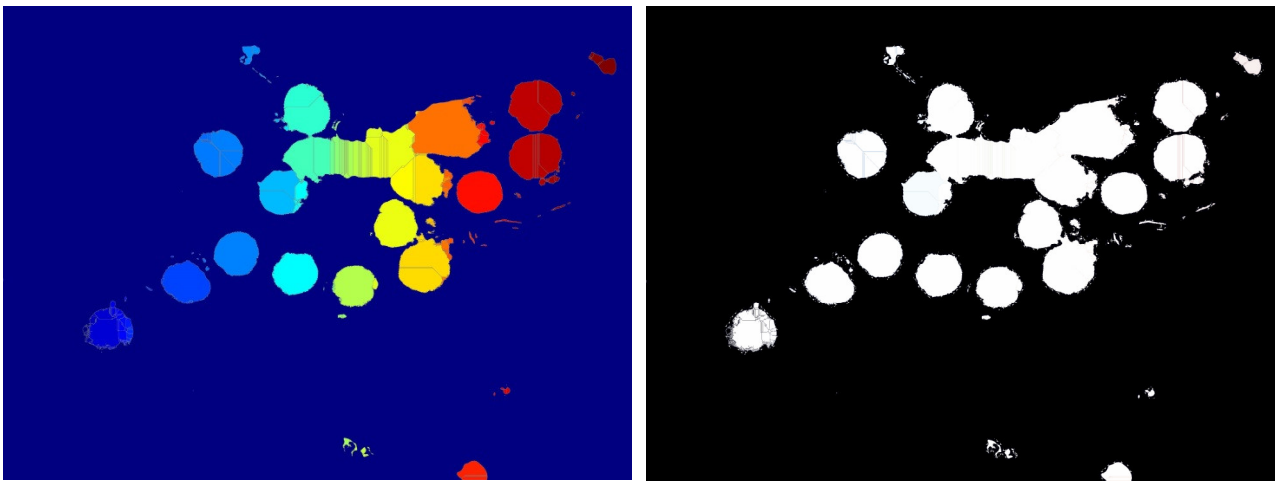


Рисунок 3.14 – Сегментація алгоритмом вододілу

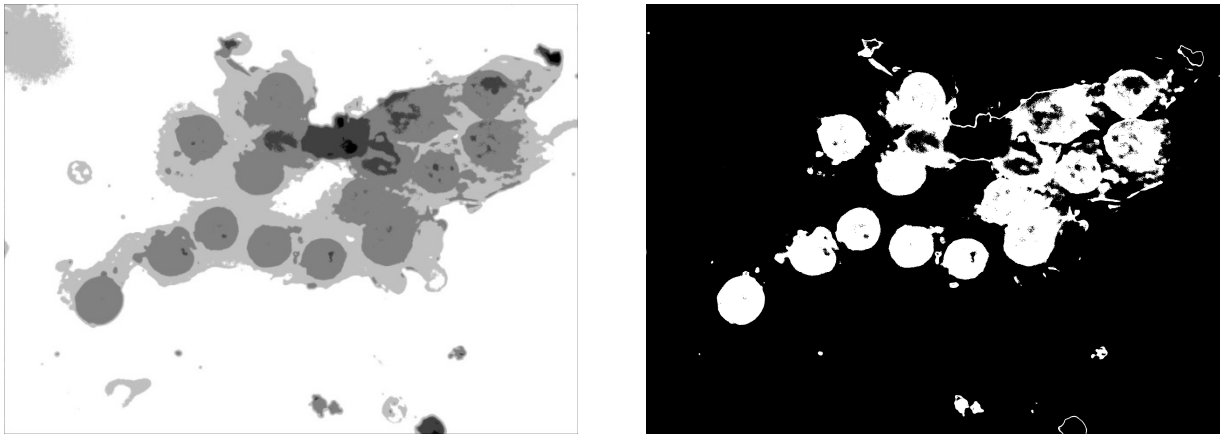


Рисунок 3.15 – Сегментація алгоритмом k-середніх

Результати досліджень наведені в таблиці 3.14. Згідно з результатами досліджень, можна зробити висновок, що для такого типу зображень найкраще працює алгоритм порогової сегментації. Трішки гірші результати показав алгоритм вододілу. А алгоритм k-середніх сильно поступається двом попереднім [59].

Таблиця 3.14 – Порівняння алгоритмів сегментації

Номер досліду	Відстань, пікселі		
	Порогова	Вододілу	k-середніх
1	649,82	515,24	1333,00
2	315,69	442.4	750,2
3	489,5	470,8	509,6
4	789,3	736,3	945,5
5	545,98	652,32	682,32
6	579,3	621,6	694,9
7	177,3	256,36	271,11
8	710,12	675,97	840,6
9	328,47	420,32	507,3
10	645,36	510,79	557,34
11	286,34	320,1	369.45
12	379.67	385,35	430.54
13	532.7	509,4	560,3
14	467,6	632,0	823,3
15	655.4	830,54	736,64

Отже, згідно з результатами експериментів, найкращі результати показав алгоритм порогової сегментації, дещо гірші результати дає алгоритм вододілу. А найгіршим для цього типу зображень став алгоритм k-середніх.

ВИСНОВКИ

1. Проаналізовано методи оцінки якості сегментації зображень. Експертний підхід вимагає багато часових ресурсів і є суб’єктивним. Тому актуальним є розробка алгоритмів кількісної оцінки яскості сегментації зображень.

2. Проаналізовано метричні простори. На основі чого було встановлено, що метрика Хаусдорфа може бути використана для встановлення міри подібності областей зображень.

3. Проаналізовано алгоритми спрощення контурів: алгоритми Дугласа-Рамера-Пекера та Вісвалінгема-Вайтта. Алгоритм Вісвалінгема-Вайтта дає незначну похибку (близько 1% зміни площі відносно оригіналу) при спрощенні у 60 разів. Алгоритм Рамера-Дугласа-Пекера дає похибку близько 4%, але є швидшим у 4-5 раз за конкурента.

4. Розроблено алгоритми для обчислення відстані у метриці Хаусдорфа. Модифікований алгоритм дозволяє знайти більш точну відстань, оскільки він враховує ребра та точки перетину фігур. Недоліком такого алгоритму є те, що він працює тільки з опуклими фігурами.

5. Проаналізовано і реалізовано алгоритми тріангуляції неопуклих фігур, що дало змогу використати модифікований алгоритм обчислення відстані у метриці Хаусдорфа для не опуклих фігур.

6. На основі поставлених вимог до програмного забезпечення було розроблено архітектуру системи, на основі якої розроблено програмне забезпечення для порівняння областей зображень у метриці Хаусдорфа.

7. В результаті тестування, було встановлено, що програмна система відповідає всім поставленим вимогам. Експериментальні дослідження встановили коректність вихідних результатів, що дає можливість кількісно оцінювати якісь сегментації зображень.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Абламейко С.В. Обработка оптических изображений клеточных структур в медицине. / С.В. Абламейко, Недзьведь А.М. – Мн.: ОИПИ НАН Беларуси, 2005. – 156 с.

2. Березький О.М. Segmentation algorithms of biomedical images: development and quantitative evaluation / О.М.Березький, Ю.М.Батько, Г.М. Мельник, С.О.Вербовий // Штучний інтелект, Київ, 2016. - №3 (73). - С. 104-116.

3. Березький О. М. Методи сегментації біомедичних зображень / О. М. Березький, Ю.М. Батько, Г.М.Мельник // Вісник Хмельницького національного університету. Технічні науки. – 2010. – №1. – С.189- 197.

4. Березький О. М. Текстурна сегментація біомедичних зображень на основі просторових моментів / О. М. Березький, Г.М.Мельник, Ю.М. Батько // Матеріали 4-ї Міжнародної науково-технічної конференції "Комп’ютерні науки та інформаційні технології 2009", 15-17 жовтня, 2009, м. Львів. – Львів: ПП «Вежа і Ко», 2009 – С.42-45.

5. Методи, алгоритми та програмні засоби опрацювання біомедичних зображень / Березький О. М., Батько Ю.М., Березька К.М., Вербовий С.О., Дацко Т.В., Дубчак Л.О., Ігнатєв І.В., Мельник Г.М., Николюк В.Д., Піцун О.Й. – Тернопіль: Економічна думка, ТНЕУ, 2017. – 330 с.

6. Березский О. Н. Количественная оценка качества сегментации изображений на основе метрик / О. Н. Березский, Е. Н. Березская // Управляющие системы и машины. – 2015. – №6. – С.59-65.

7. Березький О.М. Адаптивний метод сегментації зображень на основі метрик / О.М. Березький, О.Й. Піцун // Науковий вісник НЛТУ України: збірник науково-технічних праць. Львів: РВВ НЛТУ України. – 2018. – №. 28(3). – С.122-126.

8. Wang Z. Modern image quality assessment / Wang Z., Bovik A.C – N.Y.:Morgan & Claypool, 2006. – 157 p.

9. Цифрове зображення. URL: http://uk.wikipedia.org/wiki/Цифрове_зображення.

10. Гонсалес Р. Цифровая обработка зображений. / Р. Гонсалес, Р. Вудс / Москва: Техносфера, 2005. – 1072 с.

11. IEEE Engineering in Medicine & Biology Society. Biomedical Imaging & Image Processing. URL: http://www.embs.org/about-biomedical-engineering/our-areas-of-research/biomedical-imaging-a-image-processing.

12. Haralick R.M. Computer and Robot Vision, Vol.1, Addison / R.M.Haralick, L.G.Shapiro // Wesley, Reading, 1992.

13. Cheng H.D. Color image segmentation: advances and prospects, Pattern Recognition, / H.D.Cheng, X.H.Jiang, Y.Sun, and J.Wang // Vol.34, No.12, 2001, pp.2259-2281.

14. Мурашов Д.М. Метод автоматизированной сегментации изображений цитологических препаратов на основе модели активного контура // Труды МФТИ, 2009, Том 1, №1.

15. Березький О. М. Синтез альтернативних рішень при структурному проектуванні систем автоматизованої мікроскопії / О. М. Березький, Ю. М. Батько, Г. М. Мельник // Науковий вісник НЛТУ України: зб. наук.-техн. праць. – Львів: РВВ НЛТУ України. – 2009. – Вип. 19.5. – С. 258-268.

16. Berezsky O. Modern automated microscopy systems in oncology / O. Berezsky, O.Pitsun, N. Batryn, T. Datsko, K. Berezska, L. Dubchak // Proceedings of the 1st International Workshop on Informatics & Data-Driven Medicine, Lviv, Ukraine, 28-30 november 2018. – P. 311-325.

17. Zhang Y.J. A survey on evaluation methods for image segmentation, Pattern Recognition, / Y.J.Zhang / Vol.29, No.8, 1996, pp.1335-1346.

18. Zhang Y.J. A review of recent evaluation methods for image segmentation, Proc. of Sixth International Symposium on Signal Processing and its Applications(ISSPA 2001) / Y.J.Zhang, Vol.1, 2001, pp.148-151.

19. Zhang Y.J. Image segmentation evaluation in this century / Y.J.Zhang / Encyclopedia of Information Science and Technology, ed. M.Khosrow-Pour, 2nd ed., IGI Global, 2009, pp.1812-817.

20. Jiang X. Performance evaluation of image segmentation algorithms, in Handbook of Pattern Recognition and Computer Vision / C. H. Chen and P. S. P.Wang, Eds. / World Scientific, Singapore, 3rd edition, 2005, pp.525-542.

21. Zhang H. Image segmentation evaluation: A survey of unsupervised methods / H.Zhang, J.E.Fritts, S.A.Goldman // Computer Vision and Image Understanding, Vol.110, Issue 2, 2008, pp.260-280.

22. Zhang Y.J. Segmentation evaluation using ultimate measurement accuracy / Y.J.Zhang and J.J.Gerbrands // Proceedings CVPR, Vol.1657, 1992, pp.449-460.

23. Zhang Y.J. Objective and quantitative segmentation evaluation and comparison / Y.J.Zhang and J.J.Gerbrands // Signal Processing, Vol.39, No.1-2, 1994, pp.43-54.

24. Mattana M.F. Evaluation by recognition of thresholding based segmentation techniques on brazilian bankchecks / J.Facon, and A.S.Britto // Proceedings SPIE, Vol.3572, 1999, pp.344-348.

25. Huo Z.M. Evaluation of a computer segmentation method based on performances of an automated classification method / Z.M.Huo and M.L.Giger // Proceedings SPIE,Vol.3981, 2000, pp.16-21.

26. Jiang X. Distance Measures for Image Segmentation Evaluation, EURASIP Journal on Applied Signal Processing / X.Jiang, C.Marti, C.Irniger, and H.Bunke // - Vol. 2006, Article ID 35909, 2006, 10 pages.

27. Cardoso J.S. Toward a Generic Evaluation of Image Segmentation / J.S.Cardoso and L.Corte-Real // IEEE Transactions on Image Processing, Vol.14, No.11, 2005, pp.1773-1782.

28. Grau O. Applications of depth metadata / O.Grau, S.Minelly, and G.A.Thomas // in Proceedings of International Broadcasting Convention (IBC 2001), 2001, pp.62-70.

29. Thomas G.A. 3d image sequence acquisition for tv and film production / G.A.Thomas and O.Grau // - Proceedings of 1st International Symposium on 3D Data Processing, Visualisation and Transmission, 2002, pp.320-326.

30. Graaf C.N. Validation of the interleaved pyramid for the segmentation of 3D vector images // C.N.Graaf, A.S.E.Koster, K.L.Vincken and M.A.Viergever // - Pattern Recognition Letters, 15, 1994, pp.467-475.

31. Березький О. М. Статистичне оброблення цитологічних зображень / О. М. Березький, К. М. Березька, С. Ю. Попіна // Вісник Хмельницького національного університету: зб. наук.-техн. праць. – Сер.: Технічні науки. – 2012. – № 5. – С. 161–164.

32. Berezsky. O. Development of a metric and the methods for quantitative estimation of the segmentation of biomedical images / Berezsky O., Zarichnyi M., Pitsun O. // Eastern-European Journal of Enterprise Technologies. – 2017. – V. 6, № 4. – P. 4–11.

33. Berezsky O.M. Evaluation methods of image segmentation quality / O.M. Berezsky, O.Y. Pitsun // Радіоелектроніка, інформатика, управління. – 2018. – №1. – С. 41-61.

34. Березький О.М. Аналіз алгоритмів співставлення областей зображень для кількісної оцінки результатів сегментації / О.М. Березький, Г.М. Мельник, Ю.М. Батько, О.Й. Піцун // Матеріали Міжн. наук. конф. «Інтелектуальні системи прийняття рішень та проблеми обчислювального інтелекту» (ISDMCI’2016), м. Залізний Порт, 24–28 травня 2016 р. – Херсон: ПП Вишемирський В.С., 2016. – С. 252-253.

35. Oleh Berezsky, Mykhailo Zarichnyi, Metric Methods In Computer Vision And Pattern Recognition In book: Advances in Intelligent Systems and Computing, Publisher: Springer, 2020. Pp. 188-209.

36. Березский О. Н. Топологические методы и алгоритмы преобразования контуров и областей плоских изображений / О. Н. Березский // Проблемы информатики и управления. – 2010. – № 5. – С.123-131.

37. Березький О. М. Методи та алгоритми перетворення контурів зображень в афінному просторі / О. М. Березький // Вісник Національного університету «Львівська політехніка». Комп'ютерні науки та інформаційні технології. – 2009. – № 638. – С. 185-189.

38. ImageJ. URL: https://imagej.nih.gov/ij/.

39. Altami Studio. URL: http://altami.ru/soft/altami_studio/.

40. MediaCybernetics official site. Image-Pro Premier. URL: http://www.mediacy.com/index.aspx?page=IP_Premier.

41. Алгоритмы выделения контуров для сегментации изображений. URL: http://masters.donntu.org/2014/fknt/metelytsia/library/article11.htm

42. Hanzi Wang. Robust Statistics for Computer Vision: Model Fitting, Image Segmentation and Visual Motion Analysis / Hanzi Wang / Ph.D thesis, Monash University, Australia.

43. Lakshmi S. A Study of edge detection techniques for segmentation computing approaches / Lakshmi S, Sankaranarayanan V. // Computer Aided Soft Computing Techniques for Imaging and Biomedical Applications. – P. 35–41.

44. Ramadevi Y. Segmentation and object recognition using edge detection techniques / Ramadevi Y. // International Journal of Computer Science and Information Technology, Vol 2, No.6. – P. 153–161.

45. Senthilkumaran N. Edge Detection Techniques for Image Segmentation / Senthilkumaran N., Rajesh R. / A Survey of Soft Computing Approaches, International Journal of Recent Trends in Engineering, Vol. 1, No. 2. – P. 250–254.

46. Sowmya B. Colour Image Segmentation Using Soft Computing Techniques / Sowmya B., Sheelarani B. // International Journal of Soft Computing Applications, Issue 4. – P. 69–80.

47. Umesh Sehgal. Edge detection techniques in digital image processing using Fuzzy Logic, International Journal of Research in IT and Management / Umesh Sehgal. / Vol.1, Issue 3. – P. 61–66.

48. Rafael C. Digital Image Processing Using MATLAB / Rafael C., Gonzalez, Richard E. Woods, Steven L. Eddins // Pearson Education Ptd. Ltd, Singapore.

49. Упрощение полигональной цепи. URL: http://neerc.ifmo.ru/wiki/index.php?title=Упрощение_полигональной_цепи.

50. Алгоритм Рамера — Дугласа — Пекера. URL: https://ru.wikipedia.org/wiki/Алгоритм_Рамера_—_Дугласа_—_Пекера.

51. Mike Bostock's Blog. Line Simplification. URL: http://bost.ocks.org/mike/simplify/

52. Atallah M. J. Alinear time algorithm for the computation of some distance functions between convex polygons/ M. J. Atallah, C. C. Ribeiro, S. Lifschitz // Revue française d'automatique, d'informatique et de recherche opérationnelle. Recherche opérationnelle, tome 25, no 4 (1991), p. 413-424.

53. Тріангуляція монотонних полігонів. URL: https://kerchtt.ru/uk/triangulyaciya-monotonnyh-poligonov/.

54. A New Linear Algorithm for Intersecting Convex Polygons, O'Rourke, Chien, Olson, and Naddor (1982). URL: https://www.cs.jhu.edu/~misha/Spring20/ORourke82.pdf

55. Ласло М. Вычислительная геометрия и компьютерная графика на C++: Пер. с англ. / Ласло М. — М.: БИНОМ, 1997. — 304 с.

56. OpenCV. URL: https://uk.wikipedia.org/wiki/OpenCV.

57. Daniel P. Huttenlocher. Comparing Images Using the Hausdorff Distance / Daniel P. Huttenlocher, Gregory A. Klanderman, William J.Rucklidge // IEEE Transactions on pattern analysis and machine intelligence, vol. 15, No. 9, Sept. 1993.

58. Pishchulin L. Matching algorithms for image recognition./ L. Pishchulin - RWTH Aachen University Aachen. 2010.

59. Березький О. М. Комп'ютерна система аналізу біомедичних зображень / О. М. Березький, Ю.М. Батько, Г.М.Мельник // Вісник Національного університету «Львівська політехніка». Комп'ютерні науки та інформаційні технології. – 2009. – № 570. – С. 84-89.

60. Полагнюк І. В., Клімовський Д. Б., Вдодович О. В., Бучинський Т. Б. Сегментація зображень з використанням метричних мір. *Інтелектуальні комп'ютерні системи та мережі* : тези доп. V Наук.-практ. конф. молодих вчених і студентів (2 груд. 2021 р.). Тернопіль : ЗУНУ, 2021. С. 14.

61. Вдодович О. В., Клімовський Д. Б., Полагнюк І. В., Бучинський Т. Б. Алгоритми порівняння зображень в метричних просторах. *Інтелектуальні комп'ютерні системи та мережі* : тези доп. V Наук.-практ. конф. молодих вчених і студентів (2 груд. 2021 р.). Тернопіль : ЗУНУ, 2021. С. 11.

62. Березький О. М., Дубчак Л. О., Мельник Г. М. Методичні рекомендації до виконання кваліфікаційної роботи з освітнього ступеня "Магістр". Спеціальність: 123 - Комп'ютерна інженерія. Магістерська програма – «Комп'ютерна інженерія». Тернопіль : ЗУНУ, 2021. 32 с.