

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ  
Тернопільський національний економічний університет  
Навчально-науковий інститут інноваційних освітніх технологій  
Кафедра комп'ютерної інженерії

**ФІРКОВСЬКИЙ Богдан Олегович**

**Розпаралелення алгоритмів оцінки сегментації  
зображень засобами графічних процесорів /  
Parallelization for image evaluation segmentation  
algorithms based on GPUs**

спеціальність: 123 - Комп'ютерна інженерія  
магістерська програма - Комп'ютерна інженерія

Магістерська робота

Виконав студент групи КІм-22

Б. О. Фірко́вський

Науковий керівник: д.т.н., професор

О.М. Березький

Магістерську роботу допущено до захисту:

ТЕРНОПІЛЬ -2018

## РЕЗЮМЕ

Магістерська робота на тему «Розпаралелення алгоритмів оцінки сегментації зображень засобами графічних процесорів» зі спеціальності 123 «Комп'ютерна інженерія» написана обсягом 129 сторінок і містить 33 ілюстрації, 9 таблиць, 3 додатки та 48 джерел за переліком посилань.

Метою роботи є дослідження алгоритмів оцінки сегментації зображень та їх розпаралелення.

Методи досліджень. Для досягнення поставлених цілей в магістерській роботі використано: алгоритми комп'ютерного зору, теорію метрик, методи об'єктно-орієнтованого програмування.

Результати дослідження: алгоритми оцінки якості сегментації та їх розпаралелення, програмний модуль оцінки сегментації зображень і їх розпаралелення.

Результати роботи можуть бути використані у системах комп'ютерного зору.

Орієнтовні напрямки розвитку досліджень: розроблення додатків до бібліотек комп'ютерного зору та навчальному процесі.

**КЛЮЧОВІ СЛОВА:** АЛГОРИТМ, СЕГМЕНТАЦІЯ, ОЦІНКА СЕГМЕНТАЦІЇ ЗОБРАЖЕНЬ, РОЗПАРАЛЕЛЕННЯ АЛГОРИТМІВ.

## RESUME

Master's work on «Parallelization for image evaluation segmentation algorithms based on GPUs» from the specialty 123 «Computer engineering» written 129 page volume and contains 33 illustrations, 9 tables, 3 applications and 48 sources for references.

The purpose of the work is to study the algorithms for estimating image segmentation and their parallelism.

Research methods. To achieve the set goals in the master's work used: algorithms of computer vision, theory of metrics, methods of object-oriented programming.

Results of the study: algorithms for assessing the quality of segmentation and their parallelism, software module for estimating the segmentation of images and their parallelism.

The results of the work can be used in computer vision systems.

The approximate directions of research development: the development of applications to computer vision libraries and the educational process.

KEY WORDS: ALGORITHM, SEGMENTATION, EVALUATION OF IMAGES SEGMENTATION , ALGORITHMS PARALLELIZATION.

## ЗМІСТ

Вступ.....	7
1 Аналіз алгоритмів і засобів сегментації та оцінки сегментації зображень	9
1.1 Аналіз алгоритмів сегментації зображень .....	9
1.2 Технології розпаралелення інформації .....	26
1.3 Аналіз графічних процесорів .....	30
1.4 Аналіз завдання на магістерську роботу та постановка задач .....	34
2 Алгоритми оцінки якості сегментації зображень .....	35
2.1 Аналіз алгоритмів оцінки якості сегментації .....	35
2.2 Алгоритми кількісної оцінки якості сегментації .....	38
2.3 Алгоритми формування опуклих полігонів із неопуклих .....	43
2.4 Алгоритм знаходження дискретної відстані Фреше .....	48
2.5 Алгоритм співставлення областей на основі зважених хорд .....	50
2.6 Комп'ютерні експерименти .....	51
2.7 Алгоритм автоматичного вибору алгоритмів сегментації на основі метрик .....	53
3 Програмна реалізація алгоритмів оцінки сегментації зображень .....	61
3.1 Узагальнена структура програмного модуля .....	61
3.2 Алгоритми функціонування програмної системи .....	64
3.3 Комп'ютерні експерименти .....	79
Висновки.....	84
Список використаних джерел.....	85
Додаток А Лістинг основних модулів програми .....	90
Додаток Б Світлокопія виданої публікації .....	121
Додаток В Довідка про використання результатів дипломної роботи .....	129

## ВСТУП

Актуальність теми. Одним із актуальних напрямків розвитку сучасної прикладної науки є широке впровадження комп'ютерних засобів у медицину. Зокрема комп'ютерні технології з набором спеціалізованих програмних засобів широко використовуються для діагностики пацієнтів, швидкої обробки інформації та обміну даними, навчання майбутніх спеціалістів тощо [1]. На сучасному етапі поєднання прогресивних інформаційних технологій, нових методів та алгоритмів обробки, аналізу та синтезу зображень і медицини привели до народження нової області – телемедицини, що передбачає постановку діагнозу на відстані на основі аналізу та обробки зображень клітин органів людини. Як відомо [2-4], кожен тип клітин має свої ознаки: відповідну геометричну форму та характерне комбіноване забарвлення. Виділення цих ознак в автоматичному режимі є однією з основних задач, які стоять перед розробниками алгоритмічного та програмного забезпечення в даній галузі.

Метою роботи є дослідження алгоритмів оцінки сегментації зображень та їх розпаралелення.

Для досягнення поставленої мети необхідно розв'язати такі задачі:

- проаналізувати алгоритми сегментації зображень;
- проаналізувати технології розпаралелення інформації;
- проаналізувати графічні процесори;
- проаналізувати алгоритми оцінки якості сегментації зображень;
- розробити алгоритм автоматичного вибору алгоритмів сегментації на основі метрик;
- здійснити програмну реалізацію алгоритмів сегментації зображень;
- розпаралелити алгоритми оцінки сегментації.
- Методи досліджень: комп'ютерний зір, теорія метрик, методи об'єктно-орієнтованого програмування.

Наукова новизна.

1. Розроблено алгоритм автоматичного вибору алгоритмів сегментації на основі метрик.
2. Розроблено алгоритм розпаралелення алгоритмів оцінки сегментації зображень.

Практичне значення. Розроблено програмний модуль оцінки якості сегментації зображень.

Публікації та апробація ДР. Основні результати досліджень опубліковано в тезах доповідей на всеукраїнській науково-практичній конференції «Прикладна геометрія та інформаційні технології в моделюванні об'єктів, явищ і процесів» (AGIT-2018), (м. Миколаїв) [1]:

Березький О.М. Алгоритми опрацювання біомедичних зображень на основі графічних процесорів / О.М. Березький, Б.О. Фірко́вський, М.І. Хомин // Матеріали III всеукраїнської науково-практичної конференції «Прикладна геометрія та інформаційні технології в моделюванні об'єктів, явищ і процесів» (AGIT-2018), м. Миколаїв, 17–19 жовтня 2018 р. – Миколаїв: МНУ імені В.О. Сухомлинського, 2018. – С. 118-119.

# 1 АНАЛІЗ АЛГОРИТМІВ І ЗАСОБІВ СЕГМЕНТАЦІЇ ТА ОЦІНКИ СЕГМЕНТАЦІЇ ЗОБРАЖЕНЬ

## 1.1 Аналіз алгоритмів сегментації зображень

Нехай задане  $D$  розміром  $n \times m$ , де  $n$  і  $m$  – число рядків і стовпців - дискретне поле зору. Область  $D$  можна є растр, матриця  $B$  – функція на растрі: значення  $B(i, j)$  рівне яскравості в точці  $(i, j) \in D$ ,  $i = 1, \dots, n, j = 1, \dots, m$ . Тоді функцію  $B(i, j)$  будемо називати цифровим зображенням (або просто зображенням) на растрі  $D$ .

Зображення  $B(i, j)$  реальної сцени є сукупність зображень окремих об'єктів і фону. Представимо його у вигляді

$$B(i, j) = H_1(i, j) + \dots + H_s(i, j) + H_\phi(i, j) \quad (1.1)$$

де  $s$  – число об'єктів сцени;  $H_k(i, j)$  – зображення  $k$ -го об'єкту або видимої його частини ( $k = 1 \dots, s$ );  $H_\phi(i, j)$  – зображення фону. При цьому

$$\left. \begin{aligned} H_k(i, j) &= 0 \text{ при } (i, j) \in \bar{D}_k; \\ H_\phi(i, j) &= 0 \text{ при } (i, j) \in \bar{D}_\phi; \end{aligned} \right\} \quad (1.2)$$

де  $D_k \subset D$  – область  $k$ -го об'єкту;  $D_\phi \subset D$  – область фону;

$$D_1 \cup \dots \cup D_s \cup D_\phi = D, D_i \cap D_j = \emptyset \text{ при } i \neq j. \quad (1.3)$$

Формування із зображення  $B(i, j)$  зображень  $H_1(i, j), \dots, H_s(i, j)$  і  $H_\phi(i, j)$ , для яких виконана умова (1.2) є задача знаходження областей об'єктів  $D_1, \dots, D_s$  і області фону  $D_\phi$ . Перехід від зображення сукупності об'єктів і фону до зображень окремих об'єктів включає і визначення числа об'єктів  $s$ . Здійснення

цих задач є кінцевою метою сегментації зображень [5-10]. Таким чином, сегментація включає дві задачі: задачу знаходження фону і об'єктів і задачу ідентифікації об'єктів.

Алгоритми порогового обмеження за яскравістю. Дані алгоритми є поширеними методами сегментації, оскільки зображення об'єктів часто мають достатньо однорідну яскравість і різко виділяються з фону [13]. Найбільш просто порогова обробка здійснюється у разі, коли заздалегідь відомо, що зображення складається з одного об'єкту ( $s = 1$ ) і фону, причому яскравість точок об'єкту знаходиться в межах  $[T_1, T_2]$ , а яскравість точок фону або менше  $T_1$  або більше  $T_2$ . В цьому випадку кожній точці  $(i, j) \in D$  співставляється мітка 1, якщо  $B(i, j) \in [T_1, T_2]$ , і мітка 0 інакше. Проведена таким чином груба сегментація є остаточною внаслідок умови  $s = 1$ . У випадку коли  $s \geq 2$  вказана інформація дозволяє також провести грубу сегментацію, проте перехід до остаточної сегментації вимагає ще реалізувати алгоритм розфарбовування бінарних зображень. Таким чином, в цьому випадку остаточно сегментація вимагає комбінації глобального методу (порогового відсікання) і локального методу (розфарбовування по критерію зв'язності). Іноді відомо, що яскравості об'єктів різні і, більше того, відомі пороги: яскравість  $j$ -го об'єкту знаходиться в межах  $[T_1^j, T_2^j]$ . В цьому випадку можна сформулювати наступне правило розмітки точок:

$$\pi(m, n) = \begin{cases} j, & \text{якщо } T_1^j \leq B(m, n) \leq T_2^j, \quad j = 1, \dots, s, \\ 0, & \text{в іншому випадку.} \end{cases}$$

Проте така розмітка буде правильною тільки при виконанні умов:

$$[T_1^j, T_2^j] \cap [T_1^i, T_2^i] = \emptyset \quad (i \neq j);$$



у точці  $(m, n) \in D_\Phi$  виконується умова  $B(m, n) \in \bigcup_{j=1, \dots, s} [T_1^j, T_2^j]$ .

Іншими словами, інтервали яскравості об'єктів не повинні перетинатися, яскравість фону повинна мінятися поза інтервалами яскравості об'єктів.

Проте вказані умови в більшості випадків не виконуються, і отримати остаточну сегментацію за допомогою інформації про пороги  $[T_1^j, T_2^j]$  не вдається. Проте можна отримати часткову або багатозначну сегментацію. Допустимо, що окрім інтервалів  $Q_j = [T_1^j, T_2^j]$  відома множина  $Q_\Phi$  значень яскравості в точках фону. Якщо щодо фону нічого апріорі невідомо, можна вважати, що  $Q_\Phi$  складається із рівнів квантування яскравості. Аналогічно у відсутність інформації про яскравість  $j$ -го об'єкту можна вказати  $T_1^j = \alpha$ ,  $T_2^j = \beta$ , де  $\alpha$  і  $\beta$  – відповідно нижній і верхній рівні квантування.

Множини  $Q_j, j=1, \dots, s$  і  $Q_\Phi$  можуть бути такими, що за деякими значеннями яскравості можна однозначно встановити належність відповідної точки тієї або іншої області. З кожною точкою  $(m, n) \in D$  зв'яжемо число  $\gamma_{m,n}$ , що обчислюється як число множин з набору  $Q_\Phi, Q_1, \dots, Q_s$ , яким належить значення яскравості  $B(m, n)$ . Число  $\gamma(m, n) = \gamma_{m,n} - 1$  є показником неоднорідності в точці  $(m, n)$ . Точки з показником, рівним 0, розмічаються однозначно. Дійсно, якщо  $\gamma(m, n) = 0$ , то необхідно покласти  $\pi(m, n) = k$ , де  $k$  однозначно визначається з умови  $B(m, n) \in Q_k$ .

У випадку, коли пороги яскравості об'єктів і фону невідомі, тому алгоритм порогового обмеження слід доповнити способом визначення порогів. Визначення порогів зв'язане з аналізом гістограм. Гістограма - це відображення з множини  $\{\alpha, \dots, \beta\}$  значень яскравості в множину натуральних чисел, кожному  $b \in \{\alpha, \dots, \beta\}$  співставляється число точок  $(m, n) \in D$ , для яких  $B(m, n) = b$  (рисунок 1.1).

Глобальний максимум гістограми відповідає значенню яскравості  $b_{\max}^0$ , що найбільш часто зустрічається. В більшості задач домінує фон, так що

значення  $b_{\max}^0$  відповідає фону (у приведених вище позначеннях  $b_{\max}^0 \in Q_\phi$ ). Слід чекати, що і близькі до  $b_{\max}^0$  значення яскравості також відповідають фону. Для визначення порогу, що відокремлює яскравість об'єктів від яскравості фону, досить мати в своєму розпорядженні додаткову інформацію. Приведемо найбільш поширені приклади такої інформації.

Допустимо, що відоме деяке співвідношення (типу нерівності), що зв'язує яскравість будь-якої точки фону і об'єктів, наприклад:

$$B(m, n) - B(u, v) > T \quad (1.4)$$

для будь-яких точок  $(m, n) \in D_\phi$ ,  $(u, v) \in D_1 \cup \dots \cup D_s$

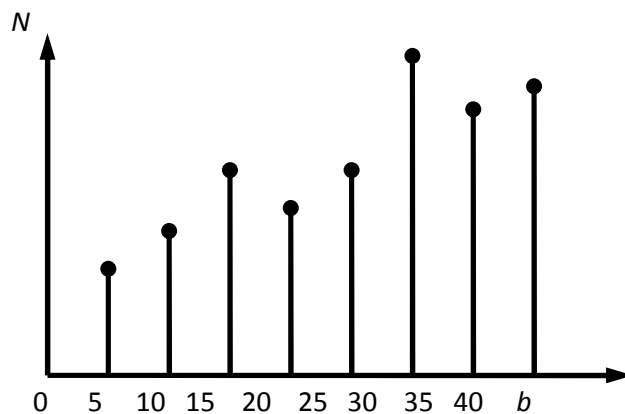


Рисунок 1.1 – Приклад гістограмного розподілу яскравості

В цьому випадку можна заключити, що для будь-якої точки  $(u, v)$  будь-якого об'єкту виконана умова

$$B(u, v) < b_{\max}^0 - T. \quad (1.5)$$

Знайдемо тепер глобальний максимум частини гістограми – в області  $[\alpha, b_{\max}^0 - T]$ . Нехай він досягається в точці  $b_{\max}^1$ . Слід чекати, що  $b_{\max}^1$

$\in Q \cup \dots \cup Q_s$ , тобто  $b_{\max}^1$  – яскравість, що найчастіше зустрічається в точках об'єктів. Із співвідношення (1.5) можна заключити, що для будь-якої точки  $(m, n)$  області фону виконана умова

$$B(m, n) > b_{\max}^1 + T. \quad (1.6)$$

Співвідношення (1.5) і (1.6) дають пороги яскравості для точок об'єктів і фону, тобто вони дозволяють побудувати часткову або багатозначну розмітку.

Алгоритми центроїдного зв'язування. Розглянуті вище алгоритми були глобальними, оскільки використовували гістограмні характеристики. Тепер перейдемо до локальних методів сегментації, що отримали назву нарощування областей. Алгоритми нарощування областей використовують інформацію про зв'язність об'єктів і засновані на рекурентному способі розмітки точок. На кроці з номером  $k$  розмічаються ті і лише ті точки, які мають сусідів з числа розмічених на попередньому кроці  $(k-1)$ . Розмітка точок здійснюється згідно деякому критерію однорідності. Конкретні алгоритми розрізняються вибором критерію однорідності, способом перегляду точок і вибором початкових "стартових" точок, що розмічаються на нульовому кроці.

Відомі два основні підходи до стратегії вибору стартових точок і порядку перегляду останніх: центроїдне зв'язування і злиття – розщеплювання областей [13]. Вибір стартових точок і їх міток в алгоритмах центроїдного зв'язування повинен бути здійснений так, щоб ніякі дві точки з різними мітками не були сусідніми.

Крім того, якщо апріорі відома деяка інформація про розміщення об'єктів в полі зору, то бажано прагнути до виконання наступних вимог:

- точки з різними мітками повинні відповідати областям різних об'єктів;
- точки з одною міткою повинні відповідати одному і тому ж об'єкту.

Останні вимоги будуть виконані в тому випадку, якщо кожній стартовій точці зіставляється своя мітка, а самі точки вибираються на достатньо великій відстані одна від одної, перевищуючим максимальний розмір об'єкту. При

виборі декількох точок з однією міткою вони повинні утворювати достатньо однорідну по яскравості множину; зазвичай воно є зв'язною.

Таким чином у алгоритмах центроїдного зв'язування апріорна інформація про об'єкти враховується в основному на етапі вибору стартових точок.

Вибір стартових точок – нульовий крок алгоритмів центроїдного зв'язування. Перехід від кроку з номером  $k$  до кроку з номером  $(k+1)$  здійснюється таким чином. Нехай  $S_k \subset D$  – множина точок, розмічених після реалізації кроку з номером  $k$ . На черговому,  $(k+1)$ -му, кроці розмітці підлягають точки, що примикають до  $S_k$ , тобто ті, що не належать  $S_k$ , але ті, що мають хоч би одного сусіда з  $S_k$ . Нехай  $A \in D$  – одна з таких точок. Через  $\lambda_1, \dots, \lambda_p$  позначимо мітки, використовувані для розмітки точок з  $S_k$ . Розглянемо два можливі випадки.

Всі розмічені сусіди  $A$  мають одну і ту ж мітку  $\lambda_j$ . У цьому випадку точка  $A$  розмічається або міткою  $\lambda_j$  або міткою  $\lambda_{p+1}$  тобто ще не використаною. Перший варіант здійснюється при виконанні деякої умови однорідності. Зазвичай вона має вигляд  $|B(A) - B_{\lambda_j}| < T$ , де  $B(A)$  – яскравість в точці  $A$ ,  $B_{\lambda_j}$  – середня яскравість точок з міткою  $\lambda_j$ ,  $T$  – вибраний поріг.

Сусідні до  $A$  точки з  $S_k$  мають різні мітки  $\lambda_1, \dots, \lambda_p$  (при 4-зв'язності  $l \leq 4$ ). В цьому випадку перевіряється умова

$$|B(A) - B_{\lambda_i}| < T \quad (1.7)$$

при кожному  $i = 1, \dots, l$ . Якщо вона виконується для єдиного  $i = i^0$ , то точка  $A$  отримує мітку  $\lambda_{i^0}$ . Якщо умова (1.3) не виконана ні при якому  $i = 1, \dots, l$ , то точка  $A$  отримує нову мітку  $\lambda_{p+1}$ . У найбільш важкому випадку умови (1.10) справедливі для декількох значень, наприклад для  $i = 1$  і  $i = 2$ . В цьому випадку

зливаються дві області - області точок з мітками  $\lambda_1$  і  $\lambda_2$  відповідно. Під злиттям розуміється перехід від міток  $\lambda_1$  і  $\lambda_2$  до єдиної мітки  $\min(\lambda_1 \text{ і } \lambda_2)$ .

У разі злиття областей міняється часткова розмітка, отримана на кроці  $k$ . Тому аналіз більшості точок зображення слід провести наново. Звичайно, необхідність повторних проходів по полю зображення різко збільшує час реалізації алгоритму.

Одночасне виконання умов (1.7) при  $i = 1, 2$  є вельми грубим критерієм злиття відповідних областей. Тонший критерій злиття областей базується на усуненні так званих "слабких границь" [10].

Алгоритми злиття-розщеплювання. Алгоритми злиття-розщеплювання відрізняються від алгоритмів центроїдного зв'язування головним чином порядком перегляду точок. Вони направлені на виділення у полі зору  $D$  однорідних областей як можна великих розмірів. Заздалегідь виділені однорідні області інтерпретуються як один елемент зображення.

Вважатимемо, що поле зору  $D$  є растром розміром  $m \times m$ , причому  $m = 2^k$  (рисунок 1.2,  $k = 4$ ). Окремі точки растру (піксели) назвемо областями нульового рівня і позначимо їх  $S_{ij}^0, i, j = 1, \dots, 2^k$ . Визначимо тепер області першого рівня як об'єднання чотирьох суміжних областей нульового рівня, а саме:

$$S_{i,j}^1 = S_{2i-1,2j-1}^0 \cup S_{2i-1,2j}^0 \cup S_{2i,2j-1}^0 \cup S_{2i,2j}^0,$$

причому  $i, j = 1, 2, \dots, 2^{k-1}$ .

При довільному  $p \leq k$  визначимо тепер області  $p$ -го рівня у вигляді об'єднання областей  $(p-1)$ -го рівня:

$$S_{i,j}^p = S_{2i-1,2j-1}^{p-1} \cup S_{2i-1,2j}^{p-1} \cup S_{2i,2j-1}^{p-1} \cup S_{2i,2j}^{p-1},$$

$i, j = 1, \dots, 2^{k-p}$ . Останній рівень має номер  $k$ . Для  $k$ -го рівня є тільки одна область, співпадаюча з растром  $D$ .

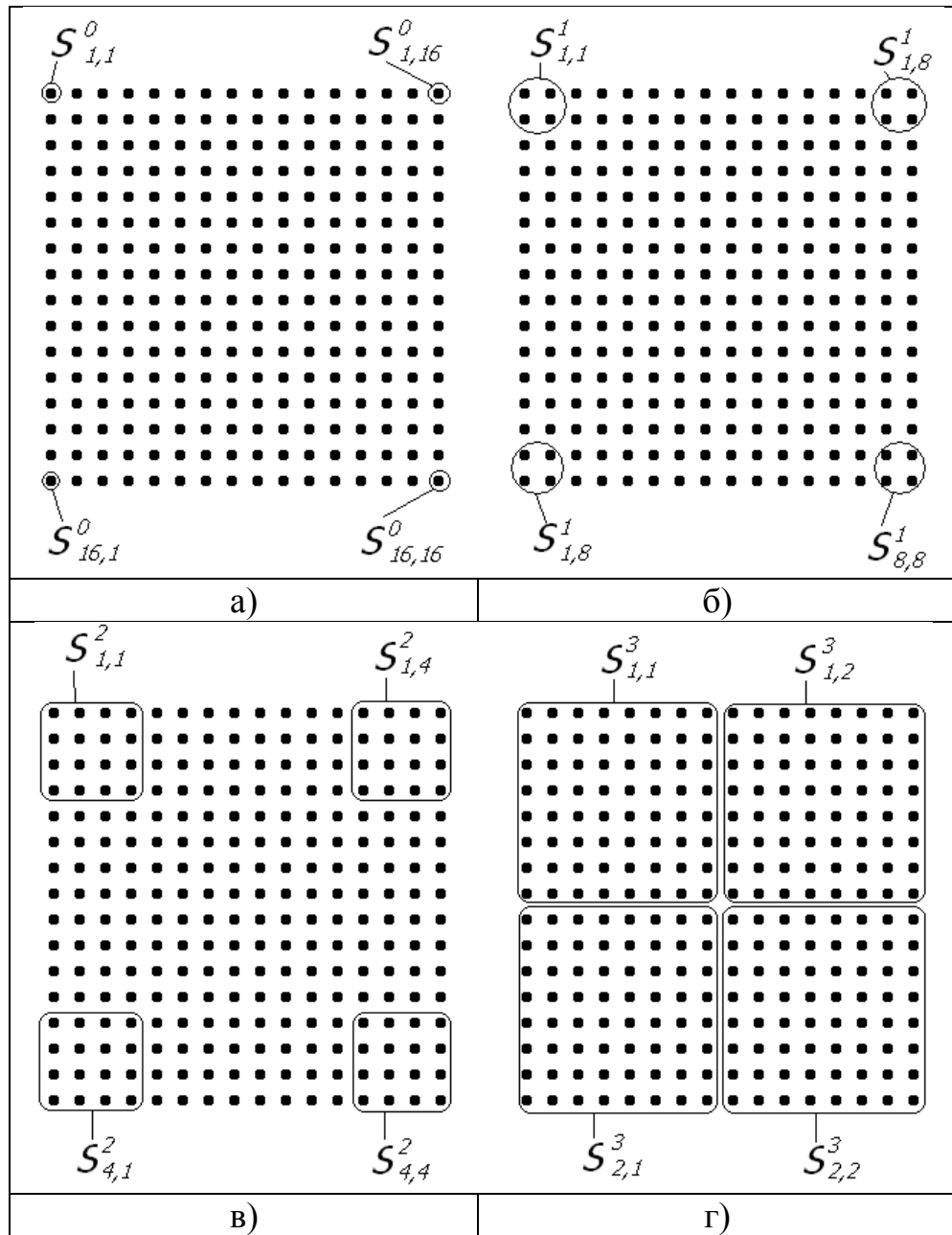


Рисунок 1.2 – Розбивання растру розміром  $16 \times 16$  на області різних рівнів:

$a$  – 0-го,  $b$  – 1-го,  $v$  – 2-го;  $z$  – 3-го

Відмітимо основні властивості вказаного розбиття:

- число областей рівня  $p$ ,  $0 \leq p \leq k$ , рівне  $2^{2(k-p)}$ ;
- об'єднання областей будь-якого рівня є растр  $D$ ;

- кожна область рівня  $p$  складається з  $2^{2p}$  точок растру;
- кожна область рівня  $(p + 1)$  є об'єднанням чотирьох суміжних областей рівня  $p$ .

Перший крок алгоритму полягає в аналізі всіх областей 1-го рівня і виділенні серед них однорідних областей, розкид яскравості точок яких не перевершує фіксованого порогу. Однорідні області при подальшому аналізі наступних рівнів інтерпретуються як один елемент зображення з яскравістю, рівній середній яскравості точок відповідної області. Таким чином, при виконанні критерію однорідності для області  $S_{ij}^1$  області  $S_{2i-1,2j-1}^0, \dots, S_{2i,2j}^0$  окремо не розглядаються – відбувається їх злиття. На наступному кроці розглядаються області 2-го рівня, але не все, а тільки такі області  $S_{i,j}^2$ , для яких виконується умова: для кожної з областей  $S_{2i-1,2j-1}^1, \dots, S_{2i,2j}^1$  виконано критерій однорідності (це перевіряється на попередньому кроці), тобто вони вже є одним елементом зображення. Тепер критерій однорідності перевіряється для набору  $S_{2i-1,2j-1}^1, \dots, S_{2i,2j}^1$ , тобто обчислюється розкид чотирьох значень яскравості, відповідних цим областям (тепер уже - елементам зображення). При виконанні умови однорідності вказані області зливаються в єдиний елемент зображення про усередненою яскравістю. Така процедура здійснюється послідовно на 1-, 2-, ...  $l$ -му рівнях. Закінчується вона на деякому рівні  $l$ , коли виконується наступна умова: для будь-якої області  $S_{i,j}^l$  або жодна з областей  $S_{2i-1,2j-1}^{l-1}, \dots, S_{2i,2j}^{l-1}$  не є єдиним елементом зображення (інформація про це є після аналізу  $(l - 1)$ -го рівня), або ті з цих областей, які є єдиними елементами зображення, не задовольняють умові однорідності, тобто не можуть злитися.

В результаті виходить розбиття растру  $D$  на певне число однорідних областей, що належать різним рівням і точок, що складаються з різного числа. Таким чином, тим областям поля зору, в якому яскравість коливається в незначних межах, відповідають великі області в розбитті растру, неоднорідні по яскравості області "дробляться" на дрібні частини.

Алгоритми розфарбовування зображень. Методи порогового відсікання і нарощування областей дозволяють у ряді випадків тільки виділити область об'єктів  $D_{об}$  і фону, тобто отримати бінарне зображення (яскравість 1 відповідає точкам з області об'єктів, яскравість 0 – точкам фону). Для отримання остаточної сегментації залишається виконати розфарбовування отриманого бінарного зображення або, що те ж саме, розфарбовування множини  $D_{об}$ . При цьому поняття "об'єкт" формалізується в термінах бінарного зображення, так що для розфарбовування існують алгоритми, що не містять елементів евристики. Оскільки при розфарбовуванні бінарного зображення вирішальну роль грає зв'язність області окремого об'єкту, всі алгоритми розфарбовування відносяться, строго кажучи, до методу нарощування областей. Точне розфарбовування можна отримати як методом злиття-розщеплювання, так і методом центроїдного зв'язування.

Алгоритми центроїдного зв'язування визначаються в першу чергу критерієм однорідності області, а також правилом перегляду точок. Для бінарних зображень критерій однорідності очевидний: однорідна область вся складається з точок з яскравістю 1 або з точок з яскравістю 0. Тому стосовно бінарних зображень визначальним для алгоритмів центроїдного зв'язування буде порядок перегляду точок. У зв'язку з цим отримали поширення алгоритми, що базуються на природному порядку перегляду точок растру, - по рядках. Найпростіший спосіб полягає в наступному: точці  $(m, n)$  зіставляється мітка (колір) з номером, мінімальним з номерів міток вже розмічених сусідніх точок. Якщо ж таких сусідів немає, точці  $(m, n)$  зіставляється нова, ще не використана мітка.

Простота реалізації такого алгоритму компенсується обмеженою областю його застосування - лише для простих зображень, наприклад тих, краї яких паралельні рядкам растру або для дещо більш складних зображень (рисунок 1.3 а). Для ряду простих зображень (рисунок 1.3б) вказаний спосіб непридатний - замість одного кольору отримуємо п'ять кольорів (колір з номером 1 з'являється при розфарбовуванні верхнього рядка, колір з номером 2 - при розфарбовуванні



наступного рядка і т. д.; при розфарбовуванні чергового рядка з'являється новий колір). Розподіл кольорів, який відповідатиме бінарному зображенню на рисунку 1.2 б при вказаному "безпосередньому" розфарбовуванні, приведений на рисунку 1.3 в.

1 1 1 1 1	0 0 0 0 1	0 0 0 0 1	0 0 0 0 1	0 0 0 0 1	0 0 0 0 1
0 1 1 1 1	0 0 0 1 1	0 0 0 2 1	0 0 0 1 1	0 0 0 1 1	0 0 0 1 1
0 0 1 1 1	0 0 1 1 1	0 0 3 2 1	0 0 2 1 1	0 0 1 1 1	0 0 1 1 1
0 0 0 1 1	0 1 1 1 1	0 4 3 2 1	0 3 2 1 1	0 2 1 1 1	0 1 1 1 1
0 0 0 0 1	1 1 1 1 1	5 4 3 2 1	4 3 2 1 1	3 2 1 1 1	2 1 1 1 1
а)	б)	в)	г)	д)	е)

Рисунок 1.3 – Приклад безпосереднього розфарбовування:

а, б - вхідні зображення; в - результат оброблення зображення б; г, д, е - результат неоднократних проходів (двох, трьох, чотирьох відповідно)

Результати послідовного застосування процедур розфарбовування представлені на рисунку 1.3 г, д, е.

Введемо декількох додаткових позначень і визначень. Нехай вибраний порядок точок растру:  $D = \{A_1, \dots, A_{\alpha\beta}\}$ . Через  $U(A_k)$  позначимо підмножину множини  $\{A_1, \dots, A_{k-1}\}$  – що складається з сусідніх з  $A_k$  точок (воно складається не більше ніж з чотирьох елементів, а при рядковому скануванні – не більше ніж з двох).

Алгоритм розфарбовування має наступний вигляд:

1. Якщо  $B(A_1) = 0$ , то  $\pi_1(A_1) = 0$ , інакше  $\pi_1(A_1) = 1$ . Відображення  $v_1$  визначимо як тотожне.

Нехай знайдені значення  $\pi_1$  для точок  $A_1, \dots, A_{k-1}$  і визначено відображення  $v_{k-1}$

2. Якщо  $B(A_k) \neq 0$ , то  $\pi_1(A_k) = 0$ .

3. Якщо  $B(A_k) \neq 0$ , але  $U(A_k) = \emptyset$  (або  $\max_{A \in U(A_k)} \pi_1(A) = 0$ ), то

$$\pi_1(A_k) = 1 + \max_{A \in \{A_1, \dots, A_{k-1}\}} \pi_1(A).$$

4. Якщо  $B(A_k \neq 0)$ ,  $U(A_k) = \emptyset$ ,  $\max_{A \in U(A_k)} \pi_1(A) \neq 0$ , то

$$\pi_1(A_k) = \min_{A \in U(A_k)} \pi_1(A).$$

Якщо при цьому серед точок  $U(A_k)$  є дві точки  $A'$ ,  $A''$  такі, що  $B(A') \neq 0$ ,  $B(A'') \neq 0$ , то змінюється відображення  $v_{k-1}$  (тобто  $v_k \neq \mathfrak{E}_{k-1}$ ), а саме:

- якщо  $p \in \overline{\{\mathfrak{E}_{k-1}(\pi_1(A)), A \in U(A_k)\}}$ , то  $v_k(p) = v_{k-1}(p)$ ;
- якщо ж  $p \in \{\mathfrak{E}_{k-1}(\pi_1(A)), A \in U(A_k)\}$ , то

$$v_k(p) = \min_{A \in U(A_k), \pi_1(A) \neq 0} \mathfrak{E}_{k-1}(\pi_1(A)).$$

5. Після того, як побудовано відображення  $\pi_1$  для всіх точок  $A_1, \dots, A_{\alpha\beta}$ , будується відображення  $\pi : D \rightarrow N$  за правилом

$$\pi(A_k) = \mathfrak{E}_{\alpha\beta}(\pi_1(A_k))$$

Доказ того, що відображення  $\pi$  дає точне розфарбовування будь-якого бінарного зображення при будь-якому порядку перегляду точок (сам по собі цей факт зовсім не очевидний), міститься в роботі [11].

Алгоритми сегментації шляхом виділення границь. Розглянемо принципово інший (в порівнянні з методом розмітки точок) спосіб сегментації - сегментацію шляхом виділення границь об'єктів. При такому способі сегментації об'єкти представляються їх границями. Виділення границь об'єктів можна розглядати і як самостійне практичне задача, не пов'язане безпосередньо

з сегментацією. Границі - основа формування різних ознак і граматик при розпізнаванні зображень. Обмежувачим критерієм є обсяг обчислень.

Поняття границі об'єкту так само, як і поняття самого об'єкту, неможливо точно формалізувати в термінах цифрового зображення  $B(i, j)$ . З евристичних міркувань граничні точки шукають як точки різкого перепаду функції яскравості.

Розглянемо неперервну модель зображення. Для цього замінимо растр  $D$  на прямокутну область в площині  $(x, y)$  і визначимо зображення як функцію  $B(x, y)$ . Для цього (абстрактного) зображення вважатимемо, що  $B(x, y)$  - функція своїх аргументів, що диференціюється. Перепад яскравості в точці  $(x_0, y_0)$  визначається в цьому випадку як норма градієнта функції  $B(x, y)$  в точці  $(x_0, y_0)$ , тобто норма вектора

$$\|\nabla B(x_0, y_0)\| = \left\| \left. \frac{\partial B(x, y)}{\partial x} \right|_{x_0, y_0}, \left. \frac{\partial B(x, y)}{\partial y} \right|_{x_0, y_0} \right\|.$$

Норму вектора можна визначити різними способами [12], наприклад:

$$\|(a_1, a_2)\| = \sqrt{a_1^2 + a_2^2}; \quad (1.8)$$

$$\|(a_1, a_2)\| = \max\{|a_1|, |a_2|\}; \quad (1.9)$$

$$\|(a_1, a_2)\| = |a_1| + |a_2|. \quad (1.10)$$

Проте через дискретність поля зору  $D$  безпосереднє обчислення  $\nabla B(x, y)$  неможливо. Щоб скористатися диференціюванням для визначення перепаду яскравості в точці, необхідно вибрати один з двох шляхів:

- за допомогою інтерполяції перейти від функції  $B(i, j)$  до функції  $B(x, y)$ , заданій в прямокутній області площині  $(x, y)$ , що містить всі точки растру  $D$ ;
- скористатися методами чисельного диференціювання, тобто диференціювання таблично заданої функції.

Перший шлях отримав назву методу функціональної апроксимації. Він пов'язаний з великими обчислювальними витратами і в задачах робототехніки не набув великого поширення. Найбільш поширеним є другий шлях, що називається градієнтним методом, методом контрастування, або методом просторового диференціювання. Існує ще один метод виділення границь, не пов'язаний з диференціюванням, - метод високочастотної фільтрації [9].

Алгоритми просторового диференціювання засновані на обчисленні градієнта функції яскравості  $B(i, j)$  за допомогою однієї з чисельних формул. Вибір формули дискретного диференціювання є головною відмінністю одного алгоритму від іншого.

Обчисливши в кожній точці  $(i, j) \in D$  норму градієнта  $G(i, j)$ , ми переходимо від початкового зображення  $B(i, j)$  до "зображення градієнтів". Зображення  $G(i, j)$  відрізняється від  $B(i, j)$  підкресленими перепадами яскравості. Перехід  $B(i, j) \rightarrow G(i, j)$  називають контрастуванням зображення.

Контрастування є тільки частиною задачі виділення границь. Для виділення границь слід сформулювати критерій приналежності точки  $(i, j)$  границі області об'єктів. Для формування такого критерію вибирають деякий поріг  $T$  і точку  $(i, j) \in D$  вважають граничними, якщо  $G(i, j) > T$ . При відсутності апріорної інформації для вибору порогу  $T$  його знаходять шляхом аналізу гістограми зображення  $G(i, j)$  приблизно тим же шляхом, що і поріг, що розділяє яскравість об'єктів і фону. Формули чисельного диференціювання для переходу до зображення  $G(i, j)$  можуть бути самими різними. Приведемо їх основні характеристики:

- число точок в околиці точки  $(i_0, j_0)$ , значення яскравості в яких впливають на значення  $G(i_0, j_0)$ ;
- того або іншого виду норми;
- вид диференціювання (одновимірне або двовимірне).

Зупинимося докладніше на цих характеристиках. Визначимо "вікно" з центром в точці  $(i, j)$  як множина точок  $(i, j) + (m, n)$ , де  $-q \leq m, n \leq q$ ,  $q$  -

параметр вікна, від якого залежить його розмір (вікно містить  $(2q + 1)^2$  точок). Вікно з параметром  $q = 1$  приведено на рисунку 1.4 а, з параметром  $q = 2$  - на рисунку 1.4 б. Вікно з центром в точці  $(i, j) = 1$  і параметром  $q$  позначатимемо  $O_{i,j}(q)$ .

Допустимо, що для обчислення  $G(i, j)$  використовується інформація про яскравість в деяких точках вікна  $O_{i,j}(q)$ . Нехай  $R \subset O_{i,j}(q)$  - множина цих точок. Якщо всі точки з  $R$  належать одній горизонталі, вертикалі або діагоналі растру, то контрастування називається одновимірним, інакше – двовимірним. Наприклад, одновимірне диференціювання уздовж прямої з орієнтацією  $45^\circ$  отримуємо у разі  $R = \{(i, j), (i+1, j+1), \dots, (i+q, j+q)\}$ . Диференціювання уздовж прямих з орієнтацією, відмінною від  $0^\circ, 45^\circ, 90^\circ, 135^\circ$ , складно, оскільки ці прямі неприродні для растру [13].

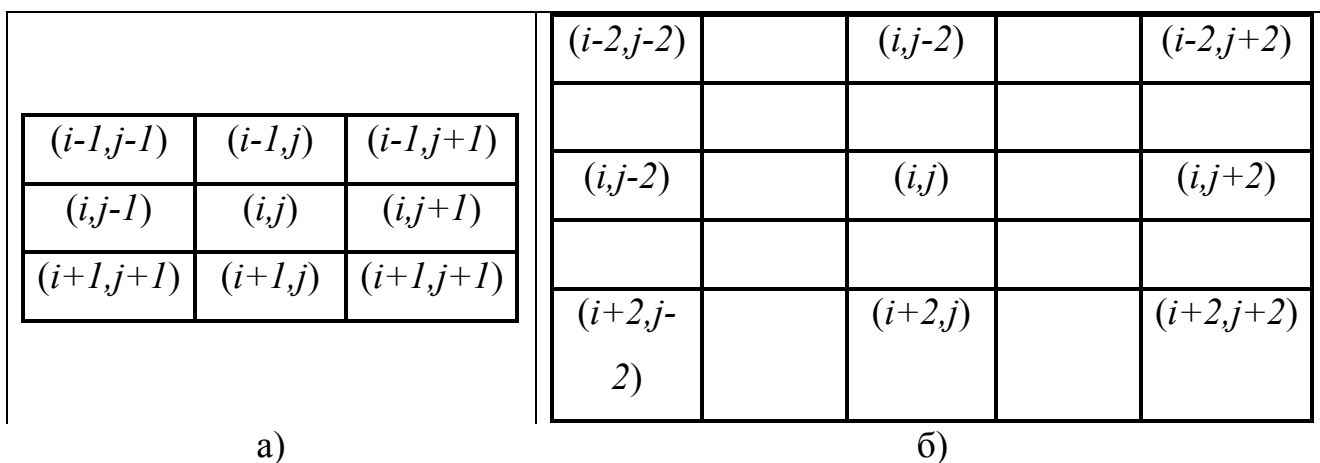


Рисунок 1.4 – Вікна розмірами  $3 \times 3$  (а) і  $5 \times 5$  (б)

У разі вибору норми (1.10) має місце лінійне контрастування, при нормі (1.8) або (1.9) – нелінійне.

Значення  $G(i, j)$  може відрізнитися від наближеного значення  $\nabla B(i, j)$  на постійний (не залежний від точки  $(i, j)$ ) множник. Для обчислення  $G(i, j)$  зручно ввести так звані маски, кожна з яких пов'язана з вікном певного розміру. Маска  $H_q$ , відповідна вікну з параметром  $q$ , має вид матриці розміром  $(2q+1) \times (2q+1)$ . Для кожного зображення  $B(i, j)$  і маски  $H_q$  визначимо згортку

$$(B * H_q)_{i,j} = \sum_{m,n=-(q-1)}^{q+1} H_q(q+m, q+n)B(i+m-1, j+n-1).$$

Наприклад, для вікна  $3 \times 3$  і маски  $H_1 = \begin{pmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{pmatrix}$  згортка

$$(B * H_1)_{i,j} = B(i-1, j-1) + B(i, j-1) + B(i+1, j-1) - \\ - B(i-1, j+1) - B(i, j+1) - B(i+1, j+1).$$

Більшість використовуваних обчислювальних формул засновані на застосуванні масок розміром  $3 \times 3$ . Основними є оператори Робертса, Собела, Превітта, Кирша, різницевий оператор, що мають загальний вигляд:

$$G(i, j) = \|d_1(i, j), d_2(i, j)\|; \\ d_1(i, j) = B * H_1^{(1)}; d_2(i, j) = B * H_1^{(2)}.$$

Для оператора Робертса

$$H_1^{(1)} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{pmatrix}, H_1^{(2)} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{pmatrix};$$

для оператора Собела

$$H_1^{(1)} = \begin{pmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{pmatrix}, H_1^{(2)} = \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix};$$

для оператора Превігга

$$H_1^{(1)} = \begin{pmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{pmatrix}, H_1^{(2)} = \begin{pmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{pmatrix};$$

для оператора Кірша

$$H_1^{(1)} = \begin{pmatrix} 5 & 5 & 5 \\ -3 & 0 & -3 \\ -3 & -3 & -3 \end{pmatrix}, H_1^{(2)} = \begin{pmatrix} 5 & -3 & -3 \\ 5 & 0 & -3 \\ 5 & -3 & -3 \end{pmatrix};$$

для різницевого оператора

$$H_1^{(1)} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & -1 \\ 0 & 1 & -1 \end{pmatrix}, H_1^{(2)} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & -1 & -1 \\ 0 & 1 & 1 \end{pmatrix}.$$

Всі ці маски відповідають двовимірному диференціюванню. Для обчислення норми  $\|d_1(i, j), d_2(i, j)\|$  можна застосувати будь-яку з формул (1.8)-(1.10). Відзначимо, що маска, використовувана для диференціювання, повинна володіти тією властивістю, що сума всіх елементів відповідної матриці рівна 0. Проаналізовані алгоритми сегментації зображень показали неможливість однозначного точного визначення “фону” і “об’єкта”, а також відсутність універсального алгоритму сегментування, що приводить до пошуку алгоритмів в залежності від класу зображень.

## 1.2 Технології розпаралелення інформації

Складність обчислювальних завдань вимагає різкого збільшення ресурсів і швидкодії комп'ютерів. Найбільш перспективним напрямком збільшення швидкості розв'язку прикладних завдань є широке впровадження ідей паралелізму в роботу обчислювальних систем. Сьогодні спроектовано і випробувано сотні різних комп'ютерів, що використовують у своїй архітектурі той чи інший вид паралельної обробки даних. Основна складність при проектуванні паралельних програм – забезпечення правильної послідовності взаємодій між різними обчислювальними процесами, а також координація ресурсів, що розділяються між ними [11].

Більшість сучасних мікропроцесорних архітектур використовують різні методи підвищення продуктивності виконуваних програм за рахунок їх розпаралелення, а саме:

- конвеєризація виконання операцій - розбиття процесу виконання на стадії і одночасне виконання операцій, які перебувають на різних стадіях конвеєра;

- одночасне виконання декількох операцій, що знаходяться на однаковій стадії конвеєрного виконання;

- підтримка багато потокового виконання команд всередині одного процесорного ядра, на декількох процесорних ядрах або в багатопроцесорній системі, що працює на загальній пам'яті;

- застосування однієї операції до декількох даних одночасно;

- підтримка такої багатопроцесорної системи, в якій якій процесори можуть працювати одночасно, використовуючи локальну чи загальну пам'ять.

Існують різні способи реалізації паралельних обчислень. Наприклад, кожен обчислювальний процес може бути реалізований у вигляді окремого процесу операційної системи, або обчислювальні процеси можуть являти собою набір потоків виконання всередині одного процесу ОС. Паралельні програми



можуть фізично виконуватися або послідовно на єдиному процесорі - по черзі виконувати операції кожного обчислювального процесу, або паралельно - виділяючи кожному обчислювальному процесу один або кілька процесорів (що знаходяться в цій системі або розподілених в комп'ютерну мережу) [12].

Зараз набувають популярності обчислення на графічних процесорах. Наприклад, у [13] описано застосування графічних процесорів для аналізу медичних зображень. Архітектура GPU побудована трохи інакше, ніж CPU. Оскільки графічні процесори спочатку використовувалися тільки для графічних розрахунків, що припускають незалежну паралельну обробку даних, то GPU призначений саме для паралельних обчислень. Він спроектований так, щоб виконувати велику кількість потоків (елементарних паралельних процесів) [1].

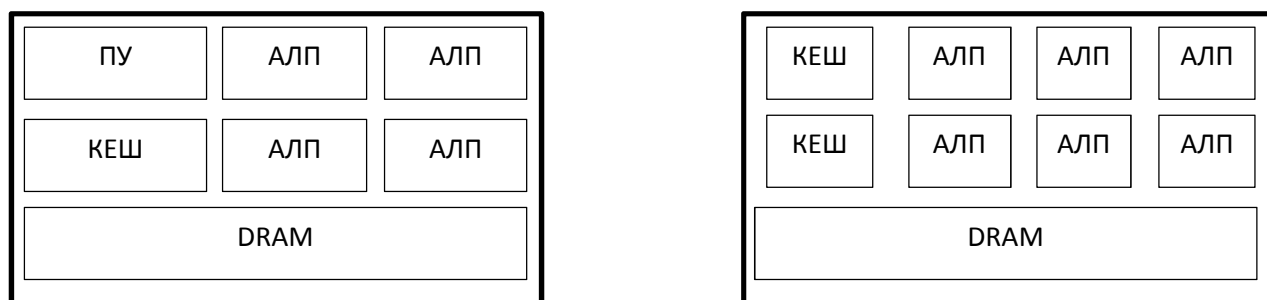


Рисунок 1.5 – Архітектура CPU (зліва) та GPU (справа)

Як видно з рисунка 1.6 – GPU містить дуже багато простих арифметико-логічних пристроїв (АЛП), які об'єднані в декілька груп і мають спільну пам'ять. Це допомагає підвищити продуктивність в обчислювальних завданнях, але трохи ускладнює програмування.

GPU орієнтований на виконання програм з великим обсягом даних та розрахунків і являє собою масив поточкових процесорів (Streaming Processor Array), що складається з кластерів текстурних процесорів (Texture Processor Clusters, TPC). TPC складається з набору мультипроцесорів (SM – Streaming Multi-processor), кожен з яких містить кілька поточкових процесорів (SP – Streaming Processors) або ядер (у сучасних графічних процесорах к-сть ядер перевищує 1024). Набір ядер кожного мультипроцесора працює за

принципом SIMD\_ (проте, з деякою відмінністю) – реалізація, що дозволяє групі процесорів, які працюють паралельно, працювати з різними даними, але при цьому всі вони в будь-який момент часу повинні виконувати однакову команду.

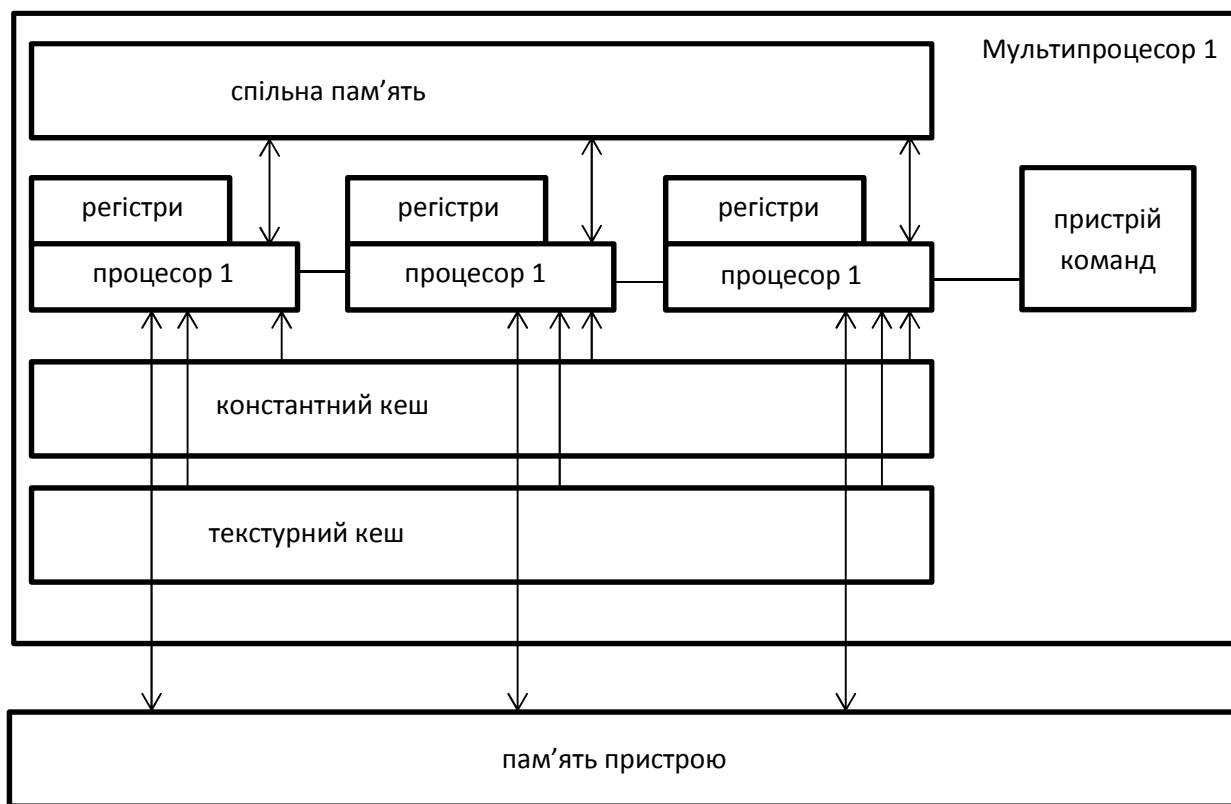


Рисунок 1.6 – Мультипроцесори в GPU

В результаті GPU фактично став пристроєм, що реалізує потокову обчислювальну модель (stream computing model) – є потоки вхідних і вихідних даних, що складаються з однакових елементів, які можуть бути оброблені незалежно один від одного. На рисунку 1.7 зображено потокову модель GPU.

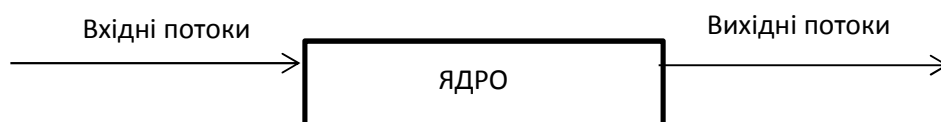


Рисунок 1.7 – Потокова модель виконання команд

На даний час існують дві потужні компанії з виготовлення графічних процесорів – AMD та Nvidia. Відповідно, існує і дві технології для розпаралелення інформації використовуючи графічні процесори – AMD FireStream (на OpenCL) та CUDA.

CUDA і OpenCL пропонують два різних інтерфейси для програмування графічних процесорів. OpenCL – це відкритий стандарт, який можна використовувати для програмування CPU, GPU та інших пристроїв різних постачальників, тоді як CUDA – специфічна для NVIDIA GPU. При використанні інструментів компілятора NVIDIA, перетворення ядра CUDA в ядро OpenCL передбачає мінімальні модифікації. Створення такого ядра з інструментами компіляції AMD передбачає більшу кількість модифікацій [14]. Незважаючи на те, що OpenCL підтримує графічні процесори різних виробників – у роботі «A Performance Comparison of CUDA and OpenCL» було продемонстровано, що OpenCL програє CUDA у швидкодії та продуктивності в межах від 13% до 63% [13].

Технологія CUDA використовує дуже багато окремих потоків для розрахунків, що групуються в ієрархію сітка/блок/потік (рисунок 1.8) [1].

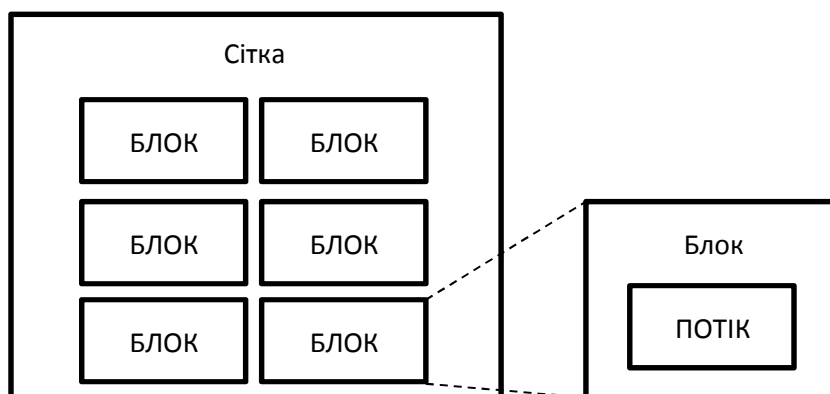


Рисунок 1.8 – Ієрархія потоків

Верхній рівень – grid – відповідає рівню ядра та об'єднує всі потоки, що виконують дане ядро. Grid – одновимірний або двовимірний масив блоків (block). Кожен блок (block) являє собою 1 / 2 / 3 - мірний масив потоків (threads). При

цьому кожен блок являє собою повністю незалежний набір скоординованих між собою потоків, але потоки з різних блоків не можуть між собою взаємодіяти.

Вище було згадано про відмінність від SIMD архітектури. Існує таке поняття, як warp – група з 32 потоків (залежно від архітектури GPU, але переважно 32). Тільки потоки в межах однієї групи (варпу) можуть фізично виконуватися одночасно. Потоки з різних варпів можуть знаходитися на різних стадіях виконання програми. Такий метод обробки даних називають терміном SIMT (Single Instruction – Multiple Theads). Управління роботою варпів виконується на апаратному рівні [1].

Перевагами технології CUDA на OpenCL є:

- Інтерфейс програмування додатків CUDA (CUDA API) заснований на стандартній мові програмування C з деякими обмеженнями. На думку розробників, це повинно спростити і згладити процес вивчення архітектури CUDA;
- Колективна між потоками пам'ять (shared memory) розміром в 16 Кб може бути використана під організований користувачем кеш з більш широкою смугою пропускання, ніж при вибірці зі звичайних текстур;
- Більш ефективні транзакції між пам'яттю центрального процесора і відеопам'яттю;
- Повна апаратна підтримка цілочисельних і побітових операцій

### 1.3 Аналіз графічних процесорів

Кожен графічний процесор має так звані обчислювальні можливості – кількісна характеристика швидкості виконання певних операцій на графічному процесорі. Це число показує те, наскільки швидко відеокарта буде виконувати свою роботу.

В таблиці 1.1 наведені деякі відеокарти й відповідні їм версії обчислювальних можливостей.

Таблиця 1.1 – Відеокарти Nvidia

Обчислювальні можливості (версія)	Графічний процесор	Відеокарта
1.0	G80, G92, G92b, G94, G94b	GeForce 8800GTX/Ultra, Tesla C/D/S870
1.1	G86, G84, G98, G96, G96b, G94, G94b, G92, G92b	GeForce 8400GS/GT, 8600GT/GTS, 8800GT/GTS, 9400GT, 9600 GSO, 9600GT, 9800GTX/GX2, 9800GT, GTS 250, GT 120/30/40, FX 4/570, 3/580, 17/18/3700, 4700x2, 1xxM, 32/370M, 3/5/770M, 16/17/27/28/36/37/3800M, NVS420/50
1.2	GT218, GT216, GT215	GeForce 210, GT 220/40, FX380 LP, 1800M, 370/380M, NVS 2/3100M
1.3	GT200, GT200b	GeForce GTX 260, GTX 275, GTX 280, GTX 285, GTX 295, Tesla C/M1060, S1070, Quadro CX, FX 3/4/5800
2.0	GF100, GF110	GeForce (GF100) GTX 465, GTX 470, GTX 480, Tesla C2050, C2070, S/M2050/70, Quadro Plex 7000, Quadro 4000, 5000, 6000, GeForce (GF110) GTX 560 TI 448, GTX570, GTX580, GTX590
5.0	GM107, GM108	GeForce GTX 750 Ti, GeForce GTX 750, GeForce GTX 860M,

Наведені графічні процесори відрізняються між собою, перш за все, обчислювальними можливостями. Загалом, на обчислювальні можливості кожного графічного процесора впливає кількість ядер CUDA, пам'ять, її тип та частота і багато інших факторів. Компанія NVIDIA також випускає графічні процесори, які спеціально розробляються для складних розрахунків – сімейство Quadro та Tesla.

У таблиці 1.2 наведено деякі відео карти компанії AMD (ATI).

Таблиця 1.2 – Відеокарти AMD

Найменування	Серія
R7000-R7200	R100
R8500,R9000-R9250	R200
R9500-R9800, X300-X600, X1050	R300
R5 210-230, R7 240-260, R9 270-290	Volcanic Islands
R5 3xx, R7 3xx, R9 3xx, R9 Nano / Fury / Fury X	Pirate Islands, Caribbean Islands
AMD Radeon RX 400 series	Arctic Islands, Polaris
AMD Radeon RX 500 series	Polaris

Для прикладу, порівняємо дві відеокарти середнього цінового сегменту. Розглянемо Radeon R9 270x від компанії AMD, та GeForce GTX 950 – від компанії NVIDIA. Результати порівняння наведені у таблиці 1.3.

Таблиця 1.3 – Порівняння відеокарт

Найменування	Radeon R9 270x	GeForce GTX 950
Ядро	Curacao	GM206
Техпроцес (мкм)	0,028	0,028
Транзисторів (млн)	2800	2940
Частота ядер (МГц)	1050	1024-1188

Продовження таблиці 1.3

Частота пам'яті (МГц)	1400	1650
Шина і тип пам'яті	256-bit GDDR5	128-bit GDDR5
Пропускна здатність (Гб/с)	179,2	105,6
Кількість ядер	1280	768
Обсяг пам'яті (Мб)	2048/4096	2048
DirectX	11.2	12
Інтерфейс	PCI 3.0	PCI 3.0

Як видно з таблиці 1.3, по певних параметрах виграє AMD, а по інших – NVIDIA. Майже в два рази GTX 950 програє конкуренту по кількості ядер. Але це не означає, що відеокарта в якій більше ядер працюватиме швидше. Тут велику роль відіграють частота пам'яті, частота ядра, затримка, архітектура, пропускна здатність та багато інших факторів.

Пропускна здатність – це головне поняття в продуктивності пам'яті відеокарти. Вимірюється в гігабайтах за секунду. Зараз активно використовується відеопам'ять GDDR5, яка має значно більший частотний потенціал, ніж GDDR3, а значить і більшу пропускну здатність. Проте, частота – не найголовніший показник. Другим важливим фактором є розрядність шини пам'яті. Чим вона більша, тим швидше працює пам'ять. Для прикладу, пам'ять з частотою 1000МГц і шиною 256bit в два рази швидша за пам'ять з частотою 1000МГц і шиною 128bit.

У проведеному порівнянні, майже по всіх параметрах виграє відеокарта від компанії AMD. Проте, для складних обчислень в більшості випадків використовують відеокарти NVIDIA, бо технологія CUDA, для програмування цих відеокарт, розроблена самою компанією NVIDIA, а значить добре оптимізована та дає можливість повністю розкрити потенціал графічного процесора.

## 1.4 Аналіз завдання на магістерську роботу та постановка задач

Виходячи із завдання на магістерську роботу метою є дослідження алгоритмів оцінки сегментації зображень та їх розпаралелення.

Відповідно об'єктом дослідження є процес аналізу сегментації.

Тому задачами магістерської роботи, які впливають з мети, об'єкту та предмету необхідно:

- проаналізувати алгоритми сегментації зображень;
- проаналізувати технології розпаралелення інформації;
- проаналізувати графічні процесори;
- проаналізувати алгоритми оцінки якості сегментації зображень;
- розробити алгоритм автоматичного вибору алгоритмів сегментації на основі метрик;
- здійснити програмну реалізацію алгоритмів сегментації зображень;
- розпаралелити алгоритми оцінки сегментації.



## 2 АЛГОРИТМИ ОЦІНКИ ЯКОСТІ СЕГМЕНТАЦІЇ ЗОБРАЖЕНЬ

### 2.1 Аналіз алгоритмів оцінки якості сегментації

Підбір алгоритму сегментації для опрацювання певного типу зображень – досить тривалий та трудомісткий процес. Виділяють два способи оцінки якості сегментації: об'єктивний та суб'єктивний. Ключовий недолік суб'єктивних способів оцінки якості сегментації полягає у тому, що оцінки робляться людиною. На рисунку 2.1 наведено класифікацію критеріїв оцінки якості сегментації.

Об'єктивні критерії оцінки сегментації використовують кількісні показники, а не візуальні. Zhang та Mattana [15] вважають, що точність виділення окремих об'єктів може слугувати оцінкою якості сегментації.

Для оцінки методів порогової сегментації Lee, Chung та Park [16] запровадили критерій вірогідності помилки сегментації на основі підрахунку неправильно класифікованих пікселів. Yasnoff та Mui ввели критерій «pixel distance error» (PDE), щоб отримати відстані між пікселями у вихідному та сегментованому зображеннях. Чим більшою буде відстань, тим вищою буде помилка сегментації.

У роботі [17] автори представили критерій FOM для визначення відстані між сегментованим пікселем та "правильним" розташуванням пікселя.

Критерій RUMA розроблений Zhang [18], використовує параметри геометричного об'єкта для оцінки якості сегментації.

Проаналізовані критерії в основному оцінюють відстані між окремими об'єктами в еталонному та сегментованому зображеннях. На практиці необхідно оцінити якість сегментації для групи об'єктів.

Алгоритми для порівняння результатів сегментації за допомогою метрик базуються на відомих метриках Фреше та Хасдорфа.

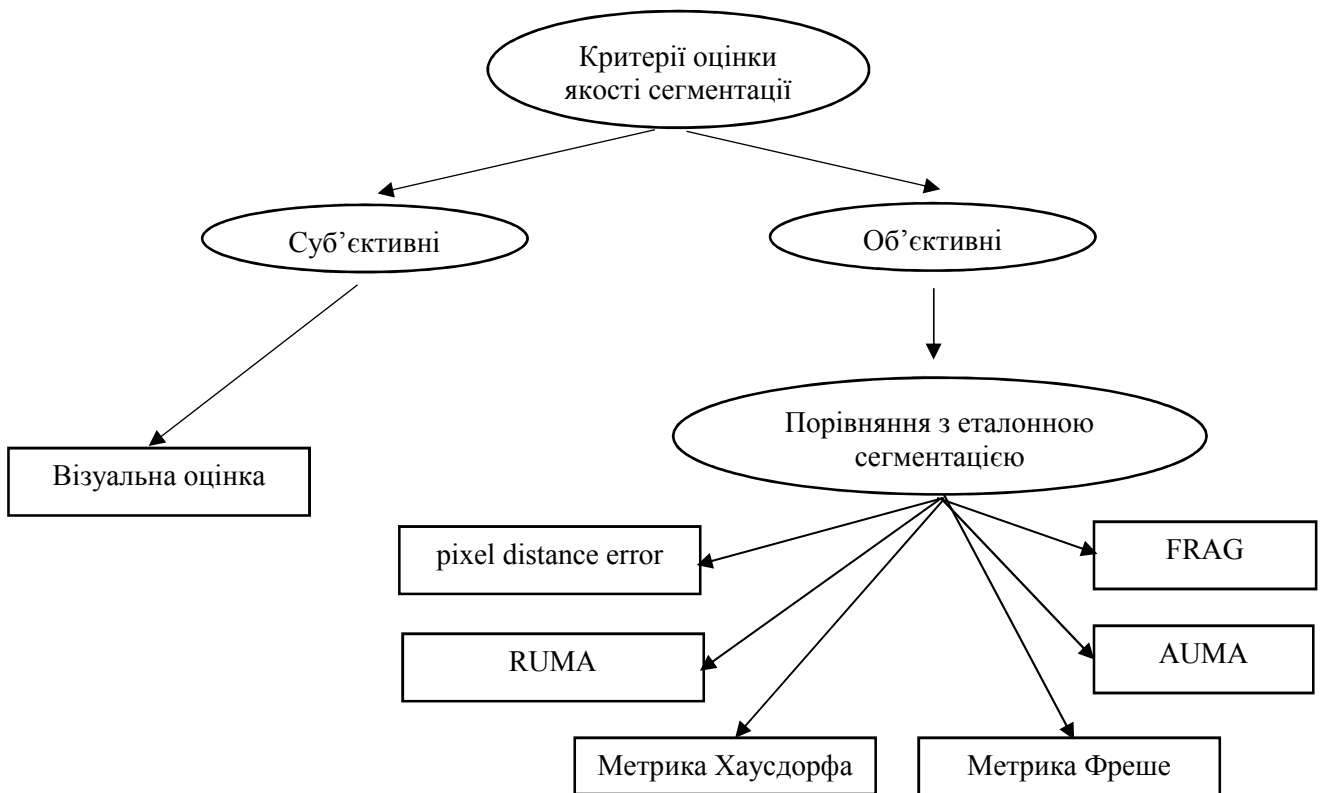


Рисунок 2.1 – Класифікація критеріїв оцінки якості сегментації

Mario A. Lopez та Shlomo Reisner [20] розробили алгоритм для зменшення кількості вершин опуклого полігону для заданої похибки  $\xi$  в метриці Хаусдорфа. Алгоритм використовується тільки для випуклих полігонів. Н. Alt та L. Scharfz [21] обчислюють відстань Хаусдорфа між алгебраїчними плоскими кривими з використанням діаграм Вороного. Алгоритм використовується для часткових випадків алгебраїчних кривих і має високу обчислювальну складність. L.P. Chew, K. Kedem [22] розробили алгоритм для знаходження мінімальної відстані Хаусдорфа в метриках  $L_i$  та  $L_\infty$ . Отримана обчислювальна складність становить  $O(n^2 \log^2 n)$ . С. Knauer, M. Scherfenberg [23] розробили метод пошуку по заданому патерну зображення, який має найменшу відстань в метриці Хаусдорфа. При цьому, використовують трансляцію заданого патерну до шуканого зображення. Алгоритм має високу обчислювальну складність. V. Alvarez [24] розробив метод пошуку на основі метрики Хаусдорфа для  $d$  – вимірному простору. Задача апроксимації такого дерева розв'язана за

поліноміальний час. Atallah [25] розробив алгоритм для пошуку відстані Хаусдорфа між випуклими полігонами. Обчислювальна складність алгоритму складає  $O(m*n)$ , де  $m$  та  $n$  – вершинами першого та другого полігонів відповідно.

Ряд публікацій присв'ячені розробці алгоритмів для знаходження відстані Фреше між кривими. Перша відома праця Н. Alt, М. Godau [26] присвячена розробці алгоритмів для знаходження відстані між параметрично заданими кривими. Обчислювальна складність становить  $O(mn \log mn)$ . У роботі [27] розроблено алгоритм обчислення дискретної відстані Фреше для полігональних кривих. Обчислювальна складність становить  $O(m^2n^2)$ , де  $m$  та  $n$  – кількість відрізків на першій та другій криві. К. Vuchin [28] розробив алгоритм для обчислення відстані Фреше для поверхонь, які представлені простими полігонами. Алгоритм має поліноміальну складність. Rote G. [29] розробив алгоритм обчислення відстані Фреше між двома кривими, які задані множиною  $m+n$  лінійно апроксимованих відрізків. Обчислювальна складність рівна  $O(m*n)$ . В роботі [30] для замкнутих полігональних кривих автори отримали обчислювальну складність  $O(m*n)$  для метрики Фреше. Н.-К. Ahn [31] і інші розробили алгоритм для обчислення дискретної відстані Фреше з неточно заданими вершинами. Для  $d$  – вимірного простору отримана обчислювальна складність дорівнює  $O(d*m*n)$ . У роботі [32] автори розробили алгоритм для обчислення відстані Фреше між неплоскими поверхнями. Автори досягли поліноміального часу в  $L_\infty$  метриці. J. Gudmundsson, M. Smid [33] розробили швидкий алгоритм для знаходження подібності полігональних дерев в метриці Фреше. Алгоритм має поліноміальну складність.

Одним із найпростіших методів кількісної оцінки сегментації є відношення правильно виділених пікселів на досліджуваному зображенні до еталонного зображення. Однак, даний підхід є простим і не завжди надійним. Критерій Pixel Distance Error враховує не лише належність неправильно ідентифікованих пікселів до певного об'єкту (сегменту), але й враховує розміщення пікселів до даного сегменту [34].

Критерій AUMA (absolute ultimate measurement accuracy) обчислюється за формулою:

$$AUMA_f = |R_f - S_f|,$$

де  $R_f$  – значення ознаки  $f$ , отримане на еталонному зображенні,  $S_f$  – значення ознаки  $f$ , отримане на досліджуваному зображенні .

Коефіцієнт RUMA обчислюється за допомогою такого перетворення:

$$RUMA_f = \frac{|R_f - S_f|}{R_f} * 100 ,$$

де  $R_f$  – значення ознаки  $f$ , отримане на еталонному зображенні,  $S_f$  – значення ознаки  $f$ , отримане на досліджуваному зображенні. В якості ознак для обчислення критеріїв AUMA та RUMA використовують площу, периметр, окружність, відношення сторін та ін. В результаті чим ближче отримане значення до нуля, тим вища оцінка сегментації .

## 2.2 Алгоритми кількісної оцінки якості сегментації

Для знаходження відстаней між зображеннями використаємо метрики Хаусдорфа, Фреше. Для знаходження найменших відстаней між зображеннями використаємо метрики Громова-Хаусдорфа та Громова-Фреше [35].

Дамо основні визначення метрик.

Метрика Фреше. Для двох кривих  $f : [a, b] \rightarrow X$  і  $g : [a', b'] \rightarrow X$  відстань Фреше між ними рівна:

$$d_F = \inf_{\alpha, \beta} \sup_{t \in [0,1]} d(f(\alpha(t)), g(\beta(t))), \quad (2.1)$$

де  $d(x, y)$  – евклідова відстань між точками  $x$  і  $y$ ,  $\alpha$  та  $\beta$  – довільні неперервні неспадні функції з проміжку  $[0,1]$  на проміжки  $[a, b]$  та  $[a', b']$  відповідно. Значення функції  $\alpha(0) = 0$  і  $\alpha(1) = 1$ , і аналогічно для функції  $\beta$  [36,37].

Метрика Хаусдорфа. Для метричного простору  $(X, d)$  хаусдорфовою метрикою  $d_H$  називається метрика на сукупності  $\mathfrak{Z}$  всіх непорожніх компактних підмножин  $X$ , яка задається так:

$$d_H^X(A, B) := \max \left\{ \max_{x \in A} \min_{y \in B} d(x, y), \max_{y \in B} \min_{x \in A} d(x, y) \right\}. \quad (2.2)$$

Метрики Громова–Хаусдорфа. Ця відстань між двома компактними множинами  $A$  і  $B$  рівна:

$$d_{GH}(A, B) := \inf_{X, f, g} d_H^X(f(A), g(B)), \quad (2.3)$$

де  $f: A \rightarrow X$ ,  $g: B \rightarrow X$  – ізометричні вкладення у деякий метричний простір  $(X, d)$  [38].

Метрика Громова-Фреше.

Є дві параметризовані криві у метричних просторах  $\gamma_i: [0,1] \rightarrow X_i$ ,  $i = 1, 2$ .  $(X_i, d_i)$  – метрика,  $t \rightarrow \gamma_i(t) \in X_i$ .

Вкладаємо ізометрично  $j_i: X_i \rightarrow Z$ ,  $i = 1, 2$ ,  $(Z, d)$  (рисунок 2.2).

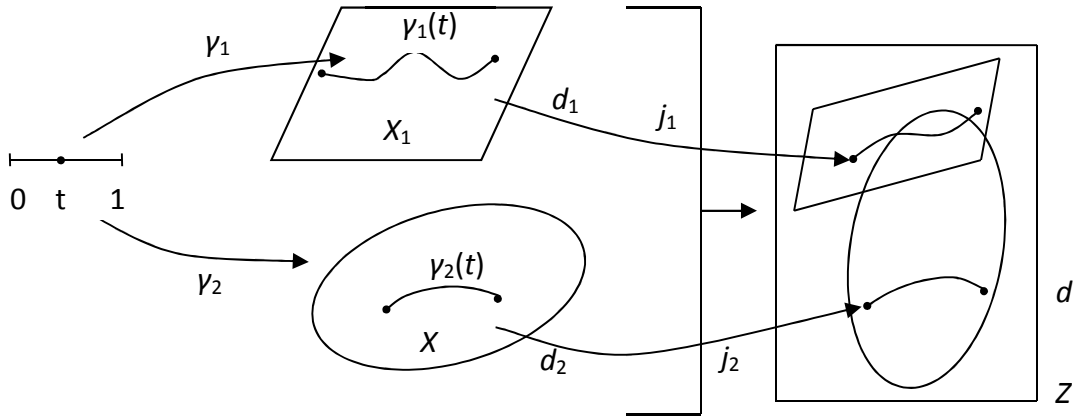


Рисунок 2.2 – Графічне представлення метрики Громова –Фреше

Тоді беремо відстань Фреше між  $j_1\gamma_1$  та  $j_2\gamma_2$ ,

$$d_F(j_1\gamma_1, j_2\gamma_2) \quad (2.4)$$

Тоді інфімум таких (2.4) по всіх ізометричних вкладеннях  $j_1$ ,  $j_2$  і є відстанню Громова-Фреше

$$d_{GF}(\gamma_1, \gamma_2) = \inf_{j_1, j_2, Z} d_F(j_1\gamma_1, j_2\gamma_2) \quad (2.5)$$

Версія для площини  $R^2$ .

$$\gamma_1, \gamma_2 : [0, 1] \rightarrow R^2.$$

Тоді:

$$\tilde{d}_{GF}(\gamma_1, \gamma_2) = \inf_{j_1, j_2, R^2} d_F(j_1\gamma_1, j_2\gamma_2) = \inf_{j_1, j_2, R^2} d_F(j_2^{-1}j_1\gamma_1, \gamma_2) = \inf_{j_1, j_2, R^2} d_F(j\gamma_1, \gamma_2) \quad (2.6)$$

$$\tilde{d}_{GF}(\gamma_1, \gamma_2) = \inf_{j: R^2 \rightarrow R^2} d_F(j\gamma_1, \gamma_2).$$

Метричний метод кількісної оцінки якості сегментації.

Після процесу сегментації отримуємо множину сегментів, які не перетинаються. Одержані сегменти лінійно апроксимуємо та отримуємо множину полігонів. Вони в загальному випадку є не опуклими. Отже, задача зводиться до порівняння двох не опуклих полігонів після алгоритму сегментації та експерта. Розглянемо два не опуклі полігони  $P$  і  $Q$  (рисунок 2.3).



Рисунок 2.3 – Полігони  $P$  і  $Q$  після сегментації

Розіб'ємо полігони  $P$  і  $Q$  на множини опуклих полігонів, тобто  $P = P_1 \cup \dots \cup P_i \cup \dots \cup P_n$ ,  $Q = Q_1 \cup \dots \cup Q_j \cup \dots \cup Q_m$ . Кожний із опуклих полігонів  $P_i$  і  $Q_j$  представимо у вигляді  $P_i = C_{P_i} \cup O_{P_i}$ , де  $C_{P_i}$  і  $O_{P_i}$  – контур (зовнішня границя) та внутрішня область  $P_i$ -го опуклого полігону відповідно. Аналогічно і для полігону

$$Q: Q_j = C_{Q_j} \cup O_{Q_j}.$$

Тоді отримаємо

$$P = (C_{P_1} \cup O_{P_1}) \cup \dots \cup (C_{P_i} \cup O_{P_i}) \cup \dots \cup (C_{P_n} \cup O_{P_n}) | V, \quad (2.7)$$

$$Q = (C_{Q_1} \cup O_{Q_1}) \cup \dots \cup (C_{Q_j} \cup O_{Q_j}) \cup \dots \cup (C_{Q_m} \cup O_{Q_m}) | W.$$

Представимо останні вирази у вигляді

$$P = (C_{P_1} \cup \dots \cup C_{P_i} \cup \dots \cup C_{P_n}) \cup (O_{P_1} \cup \dots \cup O_{P_i} \cup \dots \cup O_{P_n}), \quad (2.8)$$

$$Q = (C_{Q_1} \cup \dots \cup C_{Q_j} \cup \dots \cup C_{Q_m}) \cup (O_{Q_1} \cup \dots \cup O_{Q_j} \cup \dots \cup O_{Q_m}).$$

Позначимо

$$C_{P_1} \cup \dots \cup C_{P_i} \cup \dots \cup C_{P_n} = V_1, \quad (2.9)$$

$$O_{P_1} \cup \dots \cup O_{P_i} \cup \dots \cup O_{P_n} = V_2, \quad (2.10)$$

$$C_{Q_1} \cup \dots \cup C_{Q_j} \cup \dots \cup C_{Q_m} = W_1, \quad (2.11)$$

$$O_{Q_1} \cup \dots \cup O_{Q_j} \cup \dots \cup O_{Q_m} = W_2. \quad (2.12)$$

Тоді відстань між полігонами  $P$  і  $Q$  рівна сумі відстаней між контурами та внутрішніми областями опуклих полігонів  $P_i$  і  $Q_j$ . Відстань між областями рівна:

$$d_1(V_1, W_1) = \inf\{\varepsilon_1 > 0 \mid \forall i = \overline{1, n}, \exists j = \overline{1, m}, \text{ таке, що } d_H(O_i, O_j) \leq \varepsilon_1 \text{ і навпаки } \forall j = \overline{1, m}, \exists i = \overline{1, n}, \text{ таке, що } d_H(O_i, O_j) \leq \varepsilon_1\},$$

$d_H$  – відстань Хаусдорфа.

Аналогічно для обчислення відстаней між контурами:

$$d_2(V_2, W_2) = \inf\{\varepsilon_2 > 0 \mid \forall i = \overline{1, n}, \exists j = \overline{1, m}, \text{ таке, що } d_F(C_i, C_j) \leq \varepsilon_2 \text{ і навпаки } \forall j = \overline{1, m}, \exists i = \overline{1, n}, \text{ таке, що } d_F(C_i, C_j) \leq \varepsilon_2\},$$

$d_F$  – відстань Фреше.

Метричний метод кількісної оцінки якості сегментації (ММКОЯС) ґрунтується на сукупності алгоритмів, які забезпечують знаходження найменших відстаней між зображеннями. Ця сукупність включає множину таких алгоритмів: алгоритм формування опуклих полігонів із не опуклих, алгоритм зважених хорд, алгоритм знаходження відстані Хаусдорфа, алгоритм визначення дискретної відстані Фреше.

Покроково ММКОЯС представимо так [39]:

1. Формування множини опуклих полігонів із заданих.



2. Проведення ізометричних перетворень для накладання опуклих полігонів з максимальним перетином.
3. Знаходження відстані Фреше для опуклих полігонів.
4. Знаходження відстані Хаудорфа для опуклих полігонів.
5. Знаходження найменшої відстані між полігонами на основі зваженої метрики (метрик Фреше та Хаудорфа) між полігонами  $P$  і  $Q$  згідно виразу:  $D = \alpha d_H + \beta d_F$ , де  $\alpha, \beta$  – вагові коефіцієнти. Необхідно знайти  $D \rightarrow D_{\min}$ .

### 2.3 Алгоритми формування опуклих полігонів із неопуклих

Знаходження відстані Хаудорфа між двома довільними полігонами дуже обчислювально затратна задача. У дисертації M. Goda доведена теорема про те, що для двох опуклих множин відстань Хаудорфа рівна відстані між границями цих множин [40].

Теорема. Нехай задано  $A, B$  – дві непорожні випуклі множини, тоді відстань Фреше  $d_F(dA, dB) = d_H(dA, dB) = d_H(A, B)$  для двох границь рівна відстані Хаудорфа для границь і дорівнює відстані Хаудорфа між двома множинами.

Існують ряд ефективних (обчислювально простих) алгоритмів для знаходження відстані між опуклими областями. Тому першою задачею є формування опуклих полігонів із не опуклих. Розглянемо алгоритми перетворення не опуклих полігонів у опуклі.

Алгоритм триангуляції [40]. Триангуляція – метод, який дозволяє розбити будь-яку фігуру на трикутники. Оскільки трикутник є опуклою фігурою, то триангуляція неопуклої фігури приведе до її розбиття на опуклі. Найпоширенішими алгоритмами триангуляції є алгоритм відсікання «вух» та монотонна триангуляція.

Ідея алгоритму відсікання «вух» полягає в послідовному відсіканні трикутників («вуха»). Вершина  $v_i$  називається «вухом», якщо діагональ

проведена з  $v_{i-1}$  до  $v_{i+1}$  лежить строго у внутрішній області багатокутника  $P$  (рисунок 2.4).

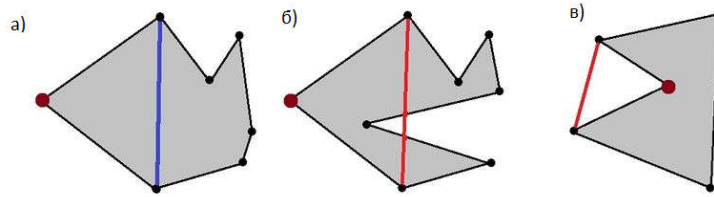


Рисунок 2.4 – Визначення вершини-вуха  
(випадок а – «вуха», випадок б, в – ні)

Будемо розглядати вершини багатокутника в порядку обходу. Індексуння вершин для зручності будемо вести по модулю  $n$ , тобто  $v_{-1} = v_{n-1}$  і  $v_0 = v_n$ . Якщо вершина  $v_i$  є вухом, побудуємо діагональ  $v_{i-1} v_{i+1}$  і відріжемо трикутник від  $\Delta v_{i-1} v_i v_{i+1}$  від  $P$ . В іншому випадку переходимо до наступної вершині  $v_{i+1}$  в порядку обходу.

При знаходженні кожного «вуха» від багатокутника  $P$  відсікається трикутник, що складається з самого «вуха» і його двох суміжних вершин. Наприкінці алгоритму, коли всі «вуха» від  $P$  відрізані, залишається тільки один трикутник. Як нескладно бачити, триангуляція будується коректно.

Спочатку в багатокутнику міститься  $O(n)$  «вух». Неважко зрозуміти, що в процесі відрізання вух, суміжні точки можуть теж ставати «вухами». У результаті триангуляції утворюється  $n-3$  діагоналі, відповідно максимальна кількість вершин, які в процесі можуть ставати «вухами»  $2n-6$ . Разом загальна кількість «вух» буде  $O(n)$ . Визначити, чи є вершина «вухом» можна за  $O(n)$ , оскільки використовується алгоритм визначення належності точки трикутнику - це  $O(1)$ . Таким чином загальний процес відрізання «вух» займе  $O(n^2)$ . Неопуклих вершин всього  $O(1)$ , кожна з них обробляється за константу, тому загальний час для їх оброблення  $O(n)$ . Списки ребер і вершин будуються за лінійний час, додавання ребра і видалення вершини в кожному з них працює за

константу. Загальний час рівний  $O(n^2)$ . Оскільки зберігаємо тільки два списки то пам'ять лінійна.

Алгоритм розбиття не опуклого полігону на опуклі області.

Нехай задано полігон  $P$ , який представлено вершинами  $V = \{v_0, v_1, \dots, v_{n-1}\}$ , де  $n$  – кількість вершин у полігоні.

Алгоритм формування опуклих полігонів складається з таких кроків [42]:

1. Починаючи з крайньої верхньої вершини шукаємо ті вершини, внутрішні кути  $\alpha_i$  яких більші за  $180^\circ$ . Тоді отримуємо масив  $B = \{b_0, b_1, \dots, b_m\}$ , де  $m$  – кількість вершин, що відповідають умові  $\alpha_i > 180^\circ$ .

2. З'єднуємо послідовно отримані вершини  $b_i$ , починаючи з крайньої верхньої та отримаємо полігон  $P_1$ .

3. Для отриманого полігона  $P_1$  повторюємо крок 1.

4. Повторюємо кроки 1,2 до полігонів  $P_i$  доти, поки внутрішній кут для кожної вершини не буде відповідати умові  $\alpha_i < 180^\circ$ . Таким чином утворюється набір випуклих полігонів.

Графічно роботу алгоритму наведено на рисунку 2.5.

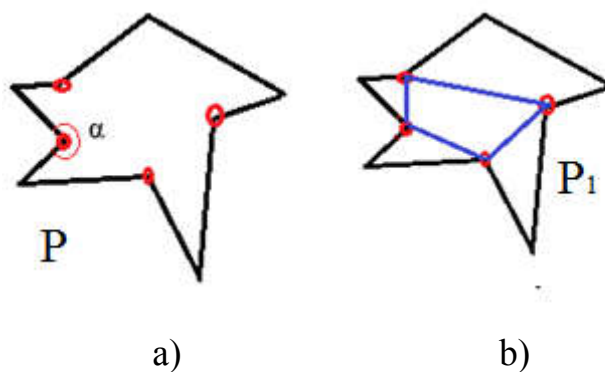


Рисунок 2.5 – Розбиття неопуклого полігону на опуклі (а – полігон  $P$ , внутрішні кути яких більше 180 градусів, б – полігон  $P_1$  опуклих областей)

Обчислювальна складність алгоритму становить  $O(n + m)$ , де  $n$  – кількість вершин полігону.

Алгоритм знаходження відстані Хаусдорфа між випуклими полігонами.

В основі алгоритму М. J. Atallah для знаходження відстані Хаусдорфа для багатокутників що перетинаються, є знаходження деякої фігури  $P$ , що визначається так:

$$P = (P_1 \cup P_2) \setminus (P_1 \cap P_2) \quad (2.13)$$

Тобто,  $P$  – це область  $P_1$  або  $P_2$ , яка не є результатом перетину їх внутрішньої частини. Іншими словами, це сукупність частинок  $P_1$  або  $P_2$ , які в них не спільні. Область  $P$  складається з  $m$  не опуклих частин.

Розглянемо задачу обчислення області перетину  $P \cap Q$  двох опуклих полігонів  $P$  і  $Q$ . За винятком особливо обумовлених випадків будемо припускати, що два полігони перетинаються не вироджено: перетин двох ребер відбувається в одній єдиній точці, яка не є вершиною якого-небудь полігону. Враховуючи таке припущення про невиродженість, завжди отримаємо, що полігон  $P \cap Q$  складається з поперемінних ланцюжків з  $P$  і  $Q$ . Кожна пара послідовних ланцюжків з'єднується в точці перетину, в якій перетинаються кордони полігонів  $P$  і  $Q$  (рисунок 2.6).

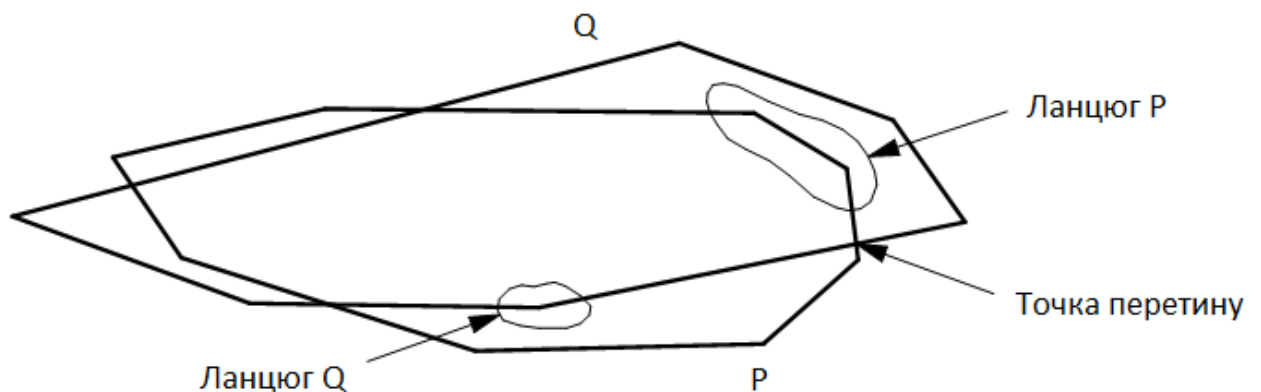


Рисунок 2.6 – Структура багатокутника перетину  $P \cap Q$

Для двох заданих на вході опуклих полігонів  $P$  і  $Q$  алгоритм визначає вікно на ребрі полігону  $P$  і ще одне вікно на ребрі полігону  $Q$ . Ідея полягає у просуванні цих вікон уздовж меж полігону у міру формування полігону перетину  $P \cap Q$ : вікна ніби проштовхують один одного вздовж границі своїх відповідних полігонів в напрямку за годинниковою стрілкою для пошуку точок перетину ребер. Оскільки точки перетину виявляються в тому порядку, в якому вони розташовуються навколо полігону  $P \cap Q$ , полігон перетину виявляється сформованим, коли деяка точка перетину буде виявлена вразі. В іншому випадку, якщо не буде виявлено жодної точки перетину після достатнього числа ітерацій, то значить границі полігонів не перетинаються. У цьому випадку потрібно додатковий простий тест, чи не міститься одна полігон всередині іншого або вони зовсім не перетинаються.

Для пояснення роботи виявляється досить корисним ввести поняття серпа. Кожен з них обмежений ланцюжком, взятої від полігону  $P$ , і ланцюжком від полігону  $Q$ , обмежених двома послідовними точками перетину. Внутрішнім ланцюжком серпа буде та частина, яка належить полігону перетину. Відзначимо, що полігон перетину оточений парним числом серпів, внутрішні ланцюжка яких поперемінно є частинами границь полігонів  $P$  і  $Q$ .

У термінах серпів алгоритм пошуку полігону перетину проходить дві фази. У процесі першої фази вікно  $p$  полігону  $P$  і вікно  $q$  полігону  $Q$  переміщуються в напрямку по руху годинникової стрілки до тих пір, поки вони не будуть встановлені на ребрах, що належать одночасно одному того ж серпу. Кожне вікно починає свій рух з довільною позиції. Тут для стислості будемо використовувати один і той же символ  $p$  для позначення як вікна полігону  $P$ , так і ребра в цьому вікні. Тоді термін "початок  $p$ " буде ставитися до точки початку ребра у вікні полігону  $P$ , команда "просунути  $p$ " означатиме, що необхідно перемістити вікно полігону  $P$  на наступне ребро. Аналогічним чином буквою  $q$  будемо позначати як вікно полігону  $Q$ , так і ребро у вікні. Іноді ребра  $p$  і  $q$  будемо вважати поточними ребрами.

Під час фази 2  $p$  і  $q$  продовжують переміщатися в напрямку за часовою стрілкою, але на цей раз вони рухаються в унісон від одного серпа до єдиному серпу. Перед тим як будь-яке вікно перейде з поточного серпа до наступного, ребра  $p$  і  $q$  перетнуться в точці перетину, що з'єднує обидва «серпи». Полігон перетину будується саме під час другої фази. Перед кожним переміщенням  $p$  кінцева точка ребра  $p$  заноситься в полігон перетину, якщо ребро  $p$  належить внутрішньому ланцюжку поточного серпа аналогічним чином перед переміщенням  $q$  фіксується кінцева точка ребра  $q$ , якщо  $q$  належить внутрішньої ланцюжку поточного серпа. Обчислювальна складність алгоритму рівна  $O(l)$ ,  $l = \max(n, m)$ , де  $n, m$  – кількість вершин полігону  $P$  і  $Q$  відповідно.

#### 2.4 Алгоритм знаходження дискретної відстані Фреше

Дискретну відстань Фреше представляють за допомогою двох жаб, які можуть перебувати на  $m$ -ому та  $n$ -ому каменях відповідно [42]. Каміння – це послідовність точок відповідних полігональних кривих  $f$  та  $g$ . Жаби перестрибують з одного каменя на інший, при цьому, повернення не дозволяється. Мінімальна довжина мотузки, яка їх з'єднує але дозволяє стрибати за маршрутом, називатиметься дискретною відстанню Фреше. Математично, дискретну відстань Фреше можна розрахувати згідно формули (2.33) [42]:

$$d_{dF}(f, g) = \max \left\{ \begin{array}{l} d_E(f_n, g_m) \\ d_{dF}(\langle f_1 \dots f_{n-1} \rangle, \langle g_1 \dots g_m \rangle), \forall n \neq 1 \\ d_{dF}(\langle f_1 \dots f_n \rangle, \langle g_1 \dots g_{m-1} \rangle), \forall m \neq 1 \\ d_{dF}(\langle f_1 \dots f_{n-1} \rangle, \langle g_1 \dots g_{m-1} \rangle), \forall n \neq 1, m \neq 1 \end{array} \right. , \quad 2.14$$

де  $d_{dF}(f, g)$  – дискретна відстань Фреше,

$d_E(f_n, g_m)$  – евклідова відстань між точками.

Дискретна відстань Фреше відрізняється від класичної відстані Фреше. Результат обчислення лежатиме в діапазоні:

$$d_F(f, g) \leq d_{dF}(f, g) \leq d_F(f, g) + L_{max}, \quad (2.15)$$

де  $L_{max}$  – довжина найдовшого сегмента полігональних кривих.

Для обчислення дискретної відстані Фреше існує декілька алгоритмів. Перший алгоритм був запропонований Альтом та Годом [43]. Його обчислювальна складність дорівнює  $O(pq \log^2 pq)$ . Набагато простіший алгоритм запропонували Томас Ейтер та Хейкі Маніла [44]. Його можна легко запрограмувати, а складність виконання становить  $O(pq)$ . Даний алгоритм можна описати так: нехай  $P [0, n]$ ,  $Q [0, m]$  – полігональні криві.  $\sigma(P) = (u_1, u_2, \dots, u_p)$ ,  $\sigma(Q) = (v_1, v_2, \dots, v_q)$  – сегменти відповідних кривих. Сукупність  $L$  між  $P$  та  $Q$  є послідовністю:

$$L = (u_{a_1}, v_{b_1}), (u_{a_2}, v_{b_2}), \dots, (u_{a_m}, v_{b_m}), \quad (2.16)$$

де  $a_1 = 1, b_1 = 1, a_m = p, b_m = q$ .

Сукупність довжин послідовності  $L$  позначається  $\|L\|$  та знаходиться за формулою:

$$\|L\| = \max_{i,j=1,\dots,m} d(u_{a_i}, v_{b_j}), \quad (2.17)$$

де  $d(u_{a_i}, v_{b_j})$  – відстань між точками  $u_{a_i}$  та  $v_{b_j}$ .

$d(u_{a_i}, v_{b_j})$  знаходиться із наступних умов:

1) якщо  $i=1$  та  $j=1$ , то дана відстань знаходиться, як евклідова відстань

$$\text{між точками: } dE = \sqrt{(v_{b_j} - u_{a_i})^2};$$

2) якщо  $i > 1$  та  $j = 1$ , тоді відстань знаходиться за формулою:

$$\max\{d(u_{a_{i-1}}, v_1), dE(u_{a_i}, v_1)\};$$

3) якщо  $i = 1$  та  $j > 1$ , тоді відстань знаходиться за формулою:

$$\max\{d(u_1, v_{b_{j-1}}), dE(u_1, v_{b_j})\};$$

4) якщо  $i > 1$  та  $j > 1$ , тоді відстань знаходиться за формулою:

$$\max\{\min(d(u_{a_{i-1}}, v_{b_j}), d(u_{a_{i-1}}, v_{b_{j-1}}), d(u_{a_i}, v_{b_{j-1}})), dE(u_1, v_{b_j})\};$$

Таким чином, шукається максимальна із відстаней між першою точкою із кривої  $P$  та усіма точками кривої  $Q$ . Відстань Фреше у даному випадку знаходиться так:

$$\delta_{dF}(P, Q) = \min_{i=1, \dots, m} \|L\|_i \quad (2.18)$$

В результаті, отримана мінімальна зі всіх максимальних відстаней і буде відстанню між кривими у метриці Фреше.

## 2.5 Алгоритм співставлення областей на основі зважених хорд

Для полігонів  $O_1 = (p_1, p_2, \dots, p_m)$  і  $O_2 = (p_1, p_2, \dots, p_m)$ , які задані вершинами, проведемо хорди. Для полігону  $O_1$  заданого  $m$  вершинами, отримаємо множину хорд  $\{h_1, h_2, \dots, h_k\}$ , де  $k = \frac{m(m-3)}{2}$ . Для полігону  $O_2$ , який заданий  $n$  вершинами – множину хорд  $\{l_1, l_2, \dots, l_p\}$ , де  $p = \frac{n(n-3)}{2}$ . Отже, для випадку повного перебору, отримаємо таку обчислювальну складність  $O(n^2 \cdot m^2)$ .

Для зменшення обчислювальної складності відсортуємо хорди шляхом зважування, використовуючи такі коефіцієнти:



а) коефіцієнт відносної довжини хорди:

$$\delta_{l_i} = \frac{l_i}{l_{\max}}, \quad (2.19)$$

де  $l_i$  – довжина  $i$ -ої хорди полігону,

$l_{\max}$  – довжина максимальної хорди полігону;

б) коефіцієнт перекриття полігону хордою:

$$\delta_{O_i} = \frac{l_{O_i}}{l_i},$$

де  $l_{O_i}$  – довжина  $i$ -ої хорди полігону, що належить внутрішній області полігону,

$l_i$  – довжина  $i$ -ої хорди полігону.

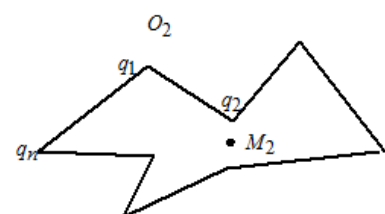
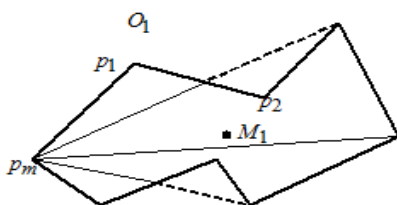
На основі заданих коефіцієнтів  $\delta_{l_i}$  і  $\delta_{O_i}$  формуємо коефіцієнти (2.40):

$$W_i = \alpha \delta_{l_i} + \beta \delta_{O_i}, \quad (2.20)$$

де  $\alpha$  та  $\beta$  – вагові коефіцієнти, що обираються з множини значень  $[0..1]$  і  $\alpha + \beta = 1$ .

Тоді масив зважених хорд для полігону  $O_1$  рівний  $\{h_{W_1}, h_{W_2}, \dots, h_{W_k}\}$ , а для  $O_2$  –  $\{l_{W_1}, l_{W_2}, \dots, l_{W_p}\}$ .

Графічне представлення алгоритму зважених хорд наведено на рисунку 2.7.



## 2.6 Комп'ютерні експерименти

Для оцінки якості сегментації групи мікрооб'єктів використаємо цитологічні зображення. На рисунку 2.8 наведено еталонне та просегментовані алгоритмами thresholding, k-means, watershed зображення. Для цього використаємо метрики Хаусдорфа, Фреше, Громова – Хаусдорфа, Громова – Фреше.

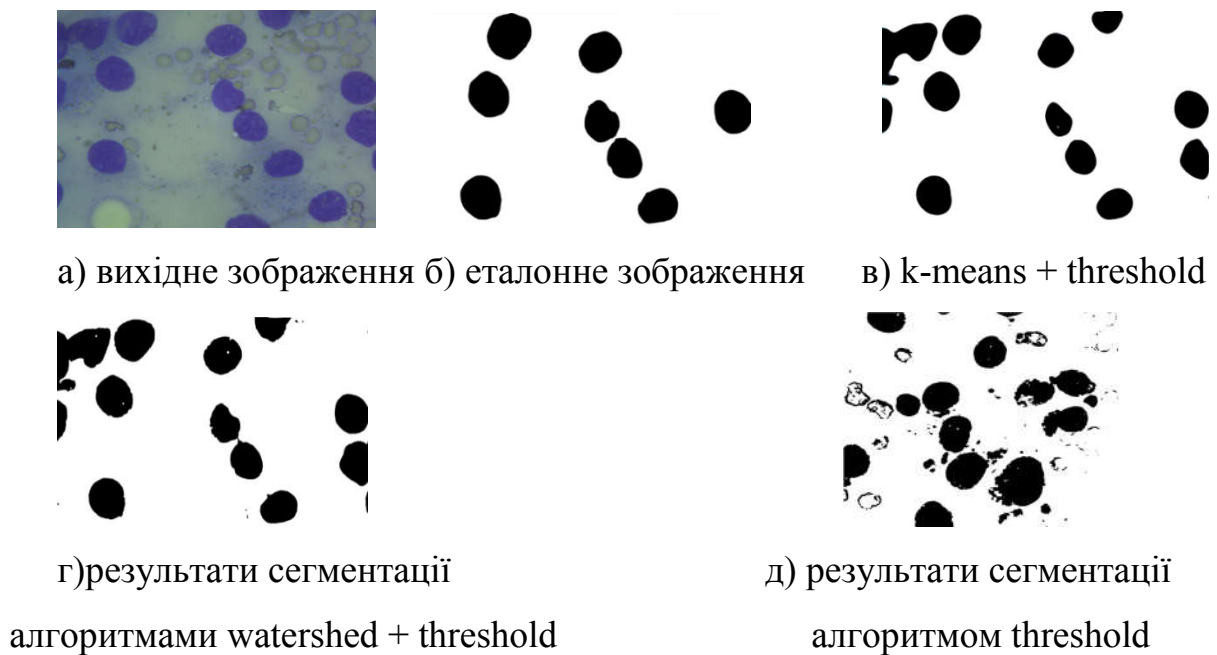


Рисунок 2.8 – Сегментація зображень

У таблиці 2.1 наведено результати оцінки якості сегментації зображень, наведених на рисунку 2.8.

Таблиця 2.1 – Порівняльний аналіз сегментації зображень на основі метрик

Метрика	б) -> б)	б) -> в)	б) -> г)	б) -> д)
---------	----------	----------	----------	----------

Хаусдорфа	0	19.20	21.37	46.57
Громов- Хаусдорфа	0	19.20	21.37	46.57
Фреше	0	36.76	26.4	52.77
Громова – Фреше	0	36.76	24.18	52.77
Зважена метрика	0	27.98	22.78	49.67

В результаті аналізу даних в таблиці 2.1 можна зробити висновок, що метрики Громова – Фреше та Громова – Хаусдорфа показали найкращі результати порівняно з іншими.

## 2.7 Алгоритм автоматичного вибору алгоритмів сегментації на основі метрик

Процес сегментації зображень є трудомістким і не завжди вдається виконати його в автоматичному режимі. В результаті комп'ютерних експериментів було протестовано сучасні алгоритми сегментації та їх комбінації та підібрано межі параметрів алгоритмів [45].

Метод вибору алгоритмів сегментації та їх параметрів представимо у вигляді послідовності кроків [46]:

1. Визначення вхідних параметрів зображення (рівень яскравості, середні значення червоного, зеленого та синього каналів).

2. Сегментація зображення. На даному етапі застосовуються такі алгоритми: порогова сегментація, метод водорозподілу, метод k – середніх. Для порогової сегментації застосовується набір значень нижнього порогу (35 - 175) з кроком 5. Метод k – середніх для тестування використовує набір різних значень прапорців.

3. Оцінка сегментації. Кожне зображення порівнюється із еталонною сегментацією, проведеною експертом. Для оцінки подібності між зображеннями застосовується метрики Громова – Хаусдорфа та Громова- Фреше і параметр FRAG. Додатково використовується експертна оцінка.

4. Найкращий результат записується до бази даних знань для подальшого формування правил.

Графічне представлення послідовності етапів тестування алгоритмів сегментації гістологічних і цитологічних зображень наведено на рисунку 2.9.

Тестування роботи розробленої системи проводилось на базі гістологічних і цитологічних зображень.

Етапи автоматичної сегментації наведено на рисунку 2.10.

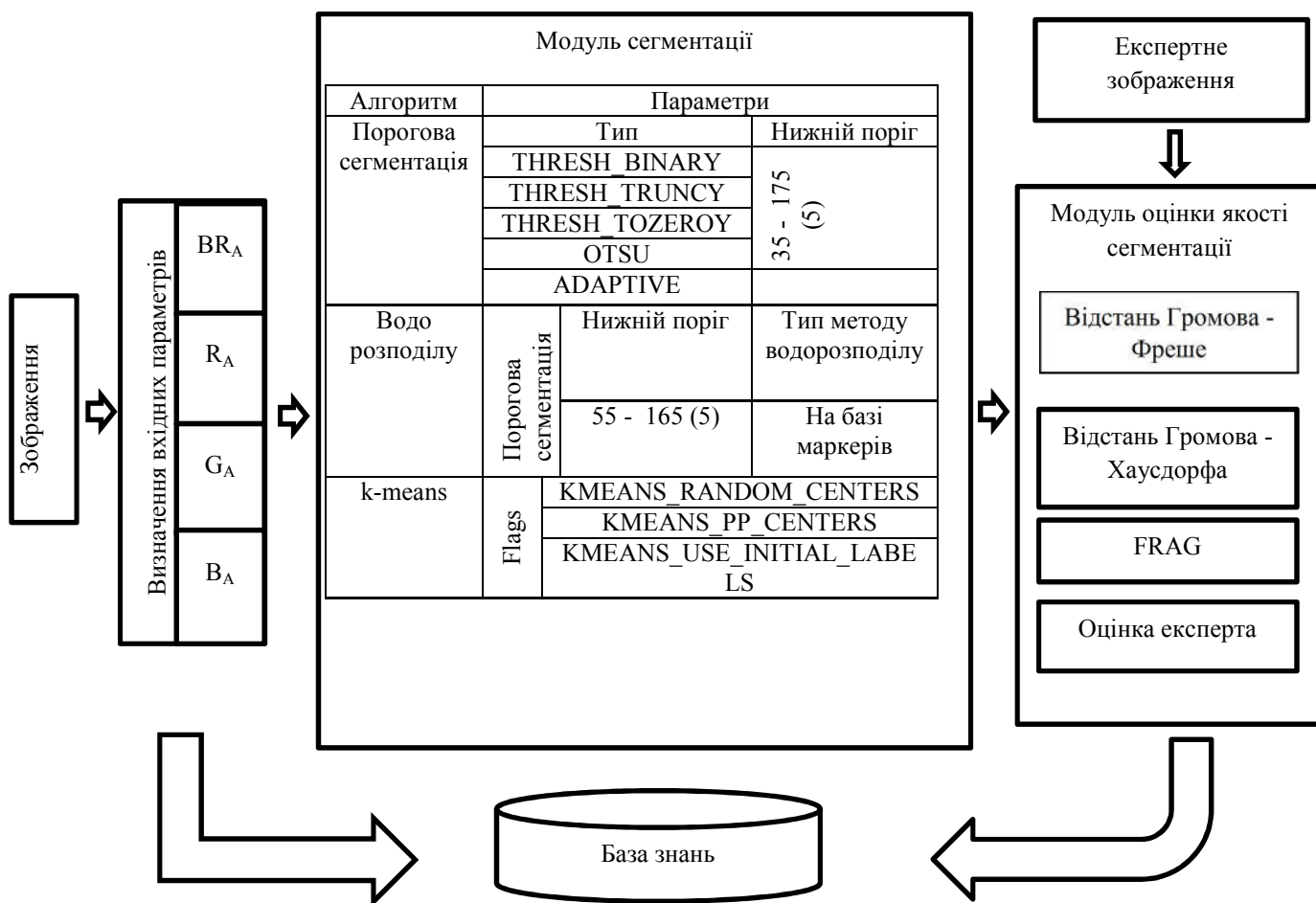


Рисунок 2.9 – Етапи тестування алгоритмів сегментації зображень

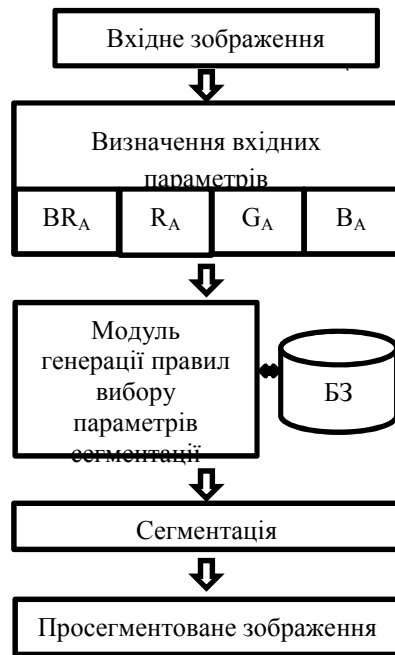


Рисунок 2.10 – Етапи автоматичної сегментації зображень

Послідовність кроків етапів автоматичної сегментації такий:

1. Завантаження зображення;
2. Виділення вхідних параметрів зображення (рівень яскравості, середні значення червоного, зеленого та синього каналів);
3. Пошук алгоритму сегментації та його параметрів у базі даних;
4. Сегментація вибраним алгоритмом;
5. Збереження результату.

Структура бази даних. База даних для зберігання результатів навчання системи складається з двох таблиці. Таблиця «InputParameters» призначена для зберігання інформації про вхідні параметри зображення. Наприклад, рівень яскравості, середні значення каналів RGB зображення. Таблиця «AlgorithmParameters» призначена для зберігання результатів навчання. Таблиця складається з полів, де зберігається інформація про найкращий алгоритм та його параметри в залежності від вхідних параметрів зображення.

Структуру таблиць наведено на рисунку 2.11.

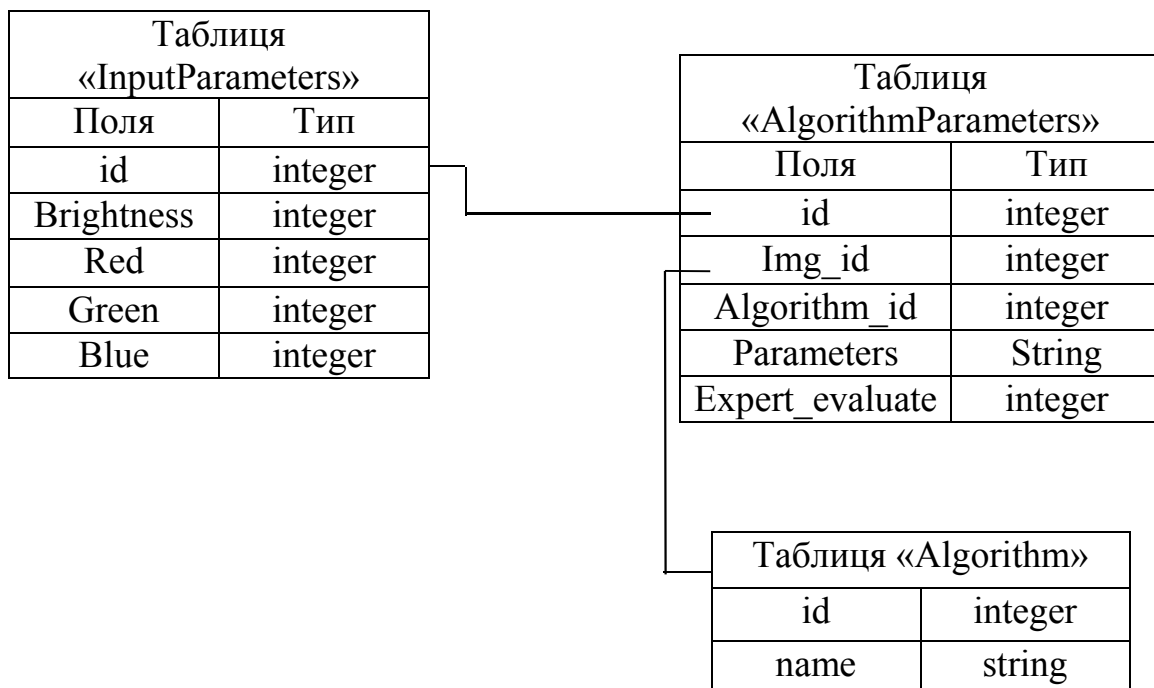


Рисунок 2.11 – Структура таблиць бази даних

Для кожної групи вхідних параметрів на основі експертних оцінок було виділено лінгвістичні оцінки розподілу алгоритмів сегментації та їх параметрів в залежності від вхідних параметрів. У таблиці 2.2 наведені лінгвістичні оцінки значень сегментації зображень, де ДН – дуже низька якість, Н- низька якість, С – середня, В – висока якість сегментації.

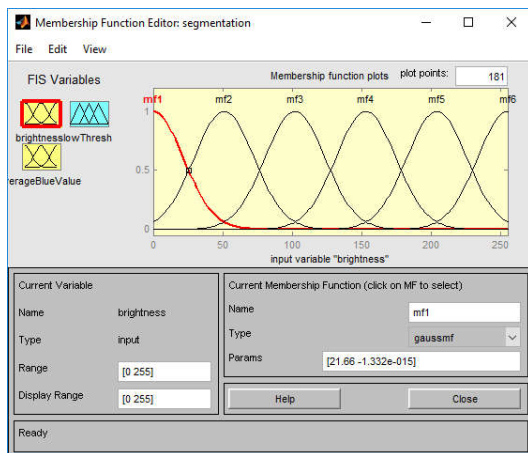
Таблиця 2.2 – Лінгвістичні оцінки методів сегментації зображень

Нижнє порогове значення Алгоритм	65	85	95	105	120	140	160
THRESHOLD thresh_binary	ДН	ДН	Н	С	С	Н	Н
THRESHOLD thresh_otsu	Н	Н	Н	В	С	С	Н
THRESHOLD thresh_binary + thresh_otsu	С	С	С	В	В	С	Н
THRESHOLD	Н	С	С	С	С	С	Н

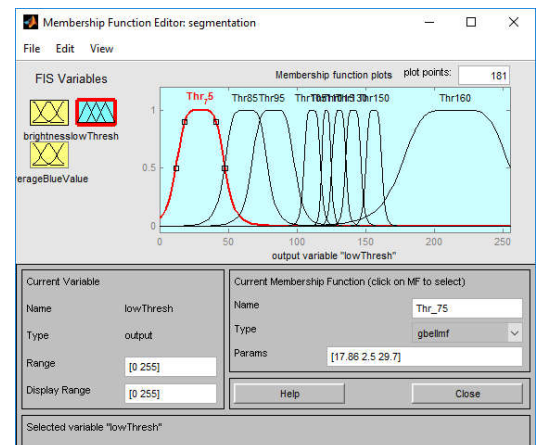
adaptive_thresh _gaussian_c							
WATERSHED thresh_binary + thresh_otsu	C	C	B	B	B	C	C
K-MEANS thresh_binary + thresh_otsu	H	C	C	B	C	C	H

Модель бази нечітких знань побудовано у редакторі Fuzzy Logic Toolbox. Характеристиками терму «Brightness» є такі значення: «Дуже низький», «Низький», «Середній», «Високий», «Дуже високий» та означають рівень яскравості вхідного зображення. Функція належності – гаусівська. Функцію належності змінної «Brightness» наведено на рисунку 2.12.

Терм «averageBlueValue» характеризує рівень синього каналу RGB зображення у проміжку від 0 до 255 та складається із 6 значень. Для даної змінної було обрано узагальнену дзвоноподібну функцію належності.



а) brightness



б) lowThresh

Рисунок 2.12 – Функція належності змінних

Вікно дії правил зображено на рисунку 2.13.

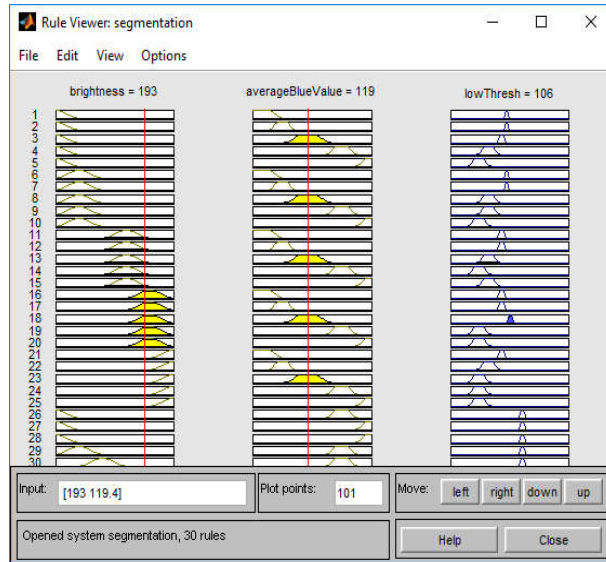


Рисунок 2.13 – Приклад реалізації правил за допомогою Fuzzy Logic Toolbox

Правила представлені так :

- IF** Brightness = 32 **AND** BlueValue = 201 **THEN** Algorithm = Watershed + Threshold, lowThreshold = 150
- IF** Brightness = 119 **AND** BlueValue = 229 **THEN** Algorithm = Watershed + Threshold, lowThreshold = 95
- IF** Brightness = 194 **AND** BlueValue = 114 **THEN** Algorithm = Watershed + Threshold, lowThreshold = 100
- IF** Brightness = 230 **AND** BlueValue = 142 **THEN** Algorithm = Watershed + Threshold, lowThreshold = 100
- IF** Brightness = 41 **AND** BlueValue = 139 **THEN** Algorithm = Watershed + Threshold, lowThreshold = 90
- IF** Brightness = 120 **AND** BlueValue = 138 **THEN** Algorithm = Threshold, lowThreshold = 85, type = THRESH\_BINARY
- IF** Brightness = 50 **AND** BlueValue = 155 **THEN** Algorithm = Threshold, lowThreshold = 95, type = THRESH\_BINARY
- IF** Brightness = 254 **AND** BlueValue = 128 **THEN** Algorithm = Watershed + Threshold, lowThreshold = 90
- IF** Brightness = 158 **AND** BlueValue = 133 **THEN** Algorithm = Watershed + Threshold, lowThreshold = 90
- IF** Brightness = 58 **AND** BlueValue = 151 **THEN** Algorithm = Watershed + Threshold, lowThreshold = 85
- IF** Brightness = 170 **AND** BlueValue = 174 **THEN** Algorithm = Watershed + Threshold, lowThreshold = 65
- IF** Brightness = 180 **AND** BlueValue = 141 **THEN** Algorithm = Watershed + Threshold, lowThreshold = 75
- IF** Brightness = 45 **AND** BlueValue = 117 **THEN** Algorithm = Watershed + Threshold, lowThreshold = 80
- IF** Brightness = 185 **AND** BlueValue = 120 **THEN** Algorithm = Watershed + Threshold, lowThreshold = 90
- IF** Brightness = 168 **AND** BlueValue = 114 **THEN** Algorithm = Watershed + Threshold, lowThreshold = 85
- IF** Brightness = 179 **AND** BlueValue = 124 **THEN** Algorithm = Watershed + Threshold, lowThreshold = 75
- IF** Brightness = 100 **AND** BlueValue = 106 **THEN** Algorithm = Watershed + Threshold, lowThreshold = 85
- IF** Brightness = 65 **AND** BlueValue = 110 **THEN** Algorithm = Watershed + Threshold, lowThreshold = 95
- IF** Brightness = 3 **AND** BlueValue = 144 **THEN** Algorithm = Watershed + Threshold, lowThreshold = 105
- IF** Brightness = 95 **AND** BlueValue = 196 **THEN** Algorithm = Watershed + Threshold, lowThreshold = 105
- IF** Brightness = 48 **AND** BlueValue = 162 **THEN** Algorithm = Threshold, lowThreshold = 80,



type = ADAPTIVE\_THRESH\_MEAN\_C

**IF** Brightness = 100 AND BlueValue = 92 **THEN** Algorithm = K-means, Core.KMEANS\_PP\_CENTERS

**IF** Brightness = 157 AND BlueValue = 121 **THEN** Algorithm = Watershed + Threshold, lowThreshold = 85

**IF** Brightness = 117 AND BlueValue = 123 Algorithm = K-means, Core.KMEANS\_PP\_CENTERS

Комп'ютерні експерименти. Для прикладу розглянемо результати порогової сегментації з різними значеннями нижнього порогу  $l$ , наведеними на рисунку 2.14.

Результати роботи системи автоматичного підбору параметрів сегментації гістологічних та цитологічних зображень наведено у таблиці 2.3. Вхідними даними у таблиці є початкове зображення, маска оброблена експертом, результат автоматичної порогової сегментації програмним засобом ImageJ та результат роботи розробленого модуля.

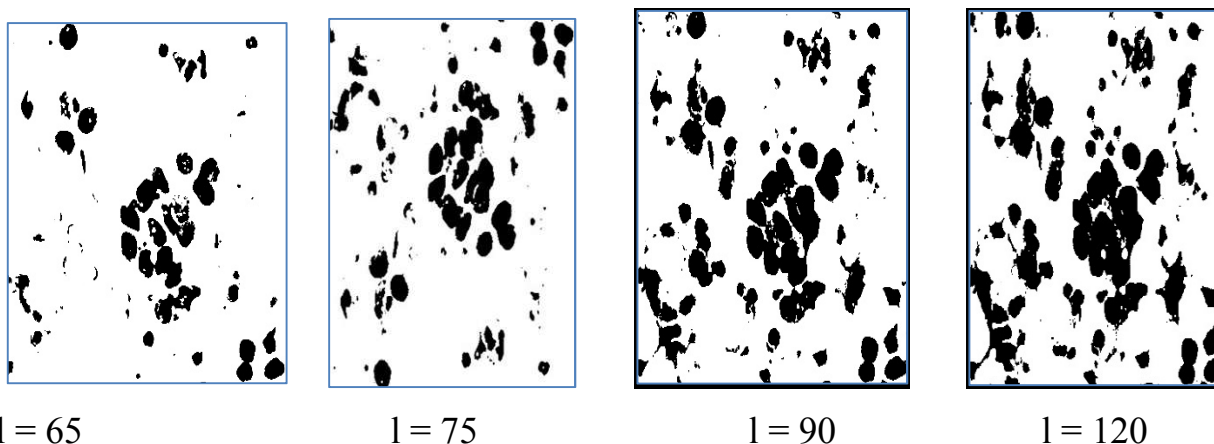
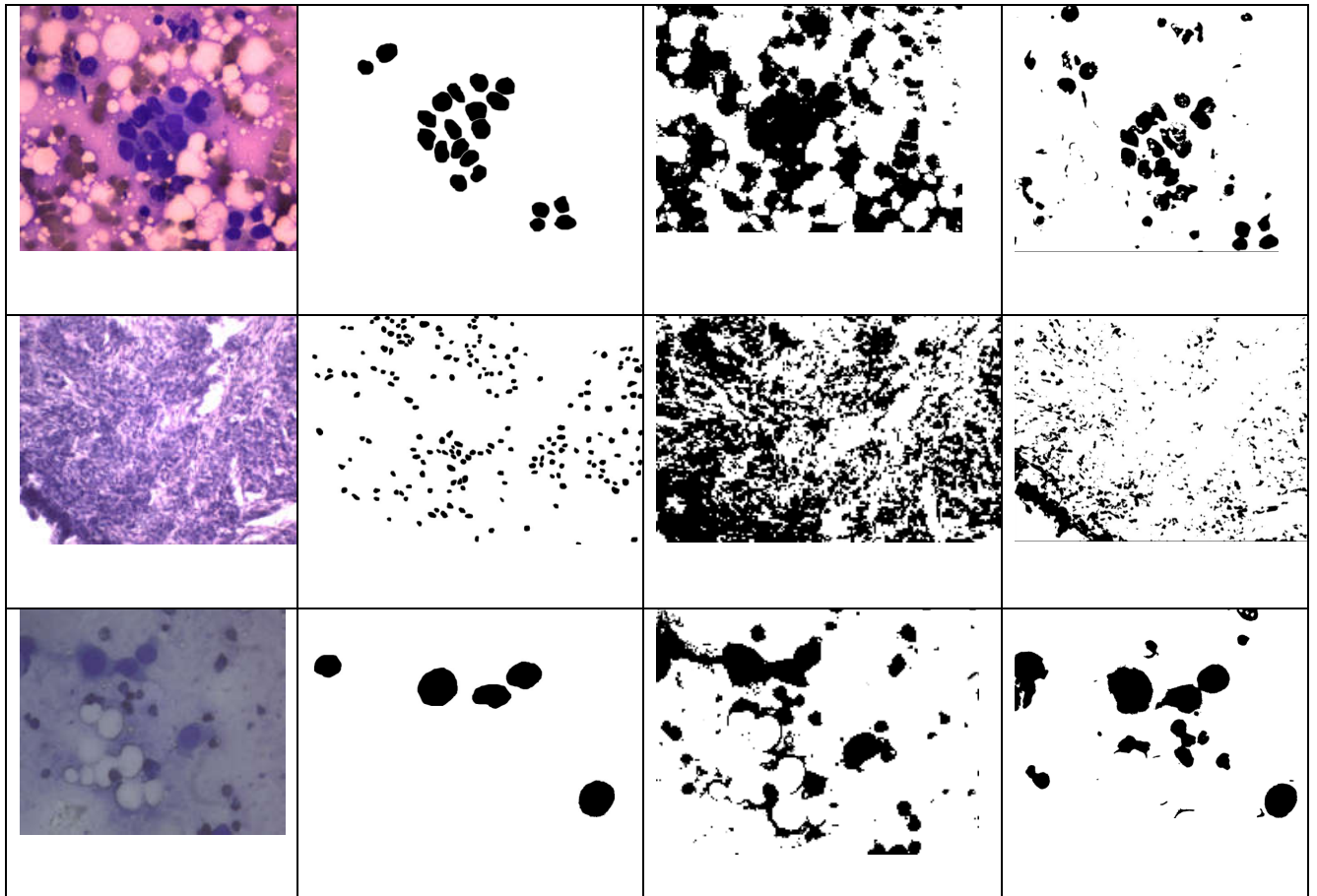


Рисунок 2.14 – Порогова сегментація зображень

Таблиця 2.3 – Результати сегментації

Вхідне зображення	Експертне оброблення	Автоматична сегментація (ImageJ)	Розроблений модуль
-------------------	----------------------	----------------------------------	--------------------



Виходячи із вищенаведених результатів, можна зробити висновок, що розроблена система автоматичного підбору параметрів сегментації дає кращі результати у порівнянні із методами автоматичної сегментації програмного комплексу ImageJ.

### 3 ПРОГРАМНА РЕАЛІЗАЦІЯ АЛГОРИТМІВ ОЦІНКИ СЕГМЕНТАЦІЇ ЗОБРАЖЕНЬ

#### 3.1 Узагальнена структура програмного модуля

Узагальнена структура програмного модуля приведена на рисунку 3.1.

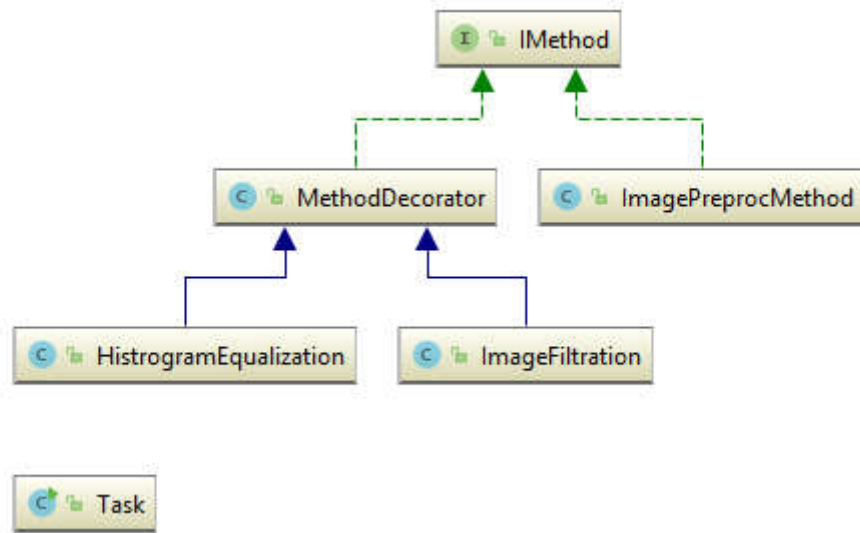


Рисунок 3.1 – Структура програмного модулю

Опишемо компоненти модуля.

Декоратор, Decorator — структурний шаблон проектування, призначений для динамічного підключення додаткових можливостей до об'єкта. Шаблон Decorator надає гнучку альтернативу методу визначення підкласів з метою розширення функціональності.

Клас ConcreteComponent — клас, в який за допомогою шаблону Декоратор додається нова функціональність. В деяких випадках базова функціональність надається класами, похідними від класу ConcreteComponent. У подібних випадках клас ConcreteComponent є вже не абстрактним, а конкретним. Абстрактний клас Component визначає інтерфейс для використання всіх цих класів.

Переваги структурного шаблону проектування:

- декоратори забезпечують гнучку альтернативу підкласу для розширення функціональності;
- декоратори дозволяють модифікувати поведінку під час виконання, а не повертатися до існуючого коду та вносити зміни;
- декоратори – це хороше рішення для перестановки завдань, тому що ви можете загорнути компонент з будь-якою кількістю декораторів;

Шаблон декоратора підтримує принцип, що класи повинні бути відкриті для розширення, але закриті для модифікації.

Недоліки:

- декоратори можуть призвести до багатьох невеликих об'єктів у нашому дизайні, і надмірне використання може бути складним;
- декоратори можуть викликати проблеми, якщо клієнт сильно залежить від компонентів конкретного типу;
- декоратори можуть ускладнити процес аналізу компонента, оскільки вам потрібно не лише інвентувати компонент, але і обернути його кількома декораторами.

Може бути складно, щоб декоратори відслідковували інших декораторів, тому що повертатися назад до декількох шарів ланцюга декораторів починає натискати шаблон декоратора поза його справжнім наміром. Структура проекту приведена на рисунку 3.2.

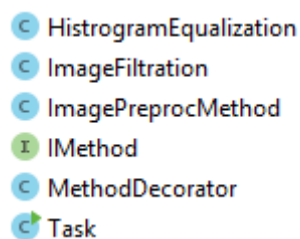


Рисунок 3.2– Структура проекту

Опишемо файли проекту.

Файл «HistogramEqualization» відповідає за вирівнювання гістограми.

Файл «ImageFiltration» відповідає за фільтрацію зображень.

Файл «ImagePreProcMethod» відповідає за попередню обробку зображення.

Графічний інтерфейс системи для знаходження відстані між об'єктами наведено на рисунку 3.3.

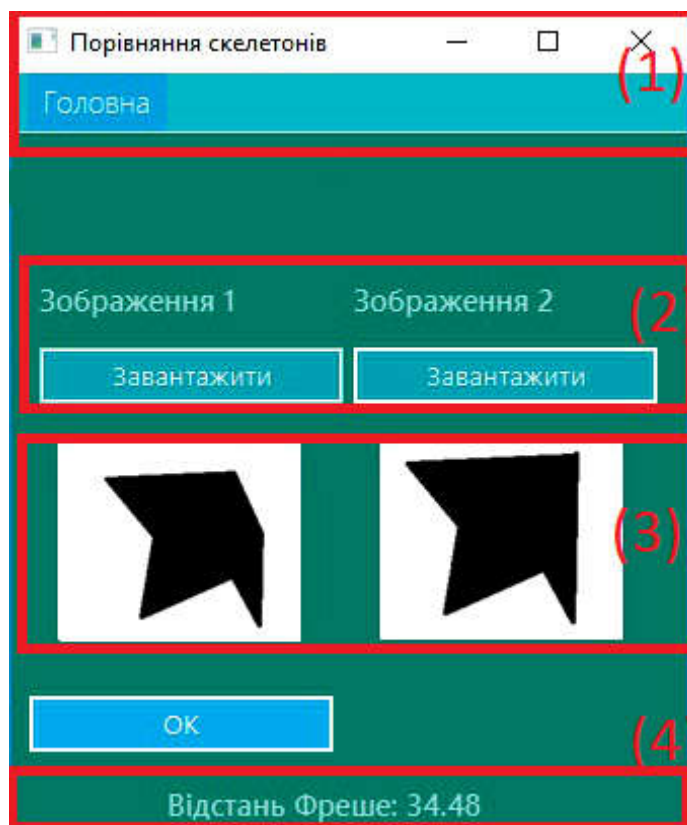


Рисунок 3.3 – Графічний інтерфейс розробленої системи

Опишемо підменю графічного інтерфейсу.

- (1) – меню;
- (2) – блок завантаження зображень для порівняння;
- (3) – блоку виводу на екран завантажених зображень після порогової сегментації;
- (4) – вивід результату порівняння.

Характеристика системи:

Мова програмування: java.

Шаблон проектування програмної системи: прототип.

Бібліотеки:

javafx, jfoenix, controllfx – для графічного інтерфейсу.

opencv – для опрацювання зображень.

- завантаження в пам'ять,
- порогова сегментація,
- апроксимація полігональних областей,
- пошук центру мас, накладання областей.

Операційна система: Windows 10, Linux.

Середовище розробки програмного забезпечення: IntelliJIDEA.

Лістинг файлу Main.java

```
public class Task {  
    public static void main(String[] args) {  
        IMethod method = new HistrogramEqualization(new ImageFiltration(new  
ImagePreprocMethod()));  
        System.out.println(method.applyMethod());  
    }  
}
```

Апаратні характеристики системи на якій проводилось тестування наведено у таблиці 3.1.

Таблиця 3.1 – Апаратні характеристики системи.

Параметри	Характеристика
Процесор	Intel(R) Pentium (R ) CPU N350 2.16GHz
Оперативна пам'ять	4 Гб
Тип системи	64 - розрядна

### 3.2 Алгоритми функціонування програмної системи

На сьогоднішній день є багато підходів та базисів (L\*a\*b\*, YIQ, HLS, HSB), що дозволяють розглядати зображення як набір яскравостей кожної точки. На основі аналізу результатів проведених експериментів для отримання яскравісного представлення зображення було використано базис HLS (тон, яскравість, насиченість). Перевід з базису RGB в HLS здійснюється за наступними формулами [2]:

$$\begin{aligned} H &= \arcsin\left(\sqrt{\frac{3}{2}}(G - R) / S\right) \\ L &= (R + G + B) / 3 \\ S &= \sqrt{R^2 + G^2 + B^2 - RG - GB - RB} \end{aligned}$$

Оскільки для проведення сегментації необхідні тільки значення яскравості, то обрахунок інших параметрів можна упустити.

Процес сегментації відбувається в процедурі PixelFon(). Виділення інформативних пікселів на вхідному зображенні відбувається шляхом порівняння яскравості в досліджуваній точці з значенням порогу.

$$M(x, y) = \begin{cases} 1, & f(A(x, y)) \geq \delta \\ 0, & f(A(x, y)) < \delta \end{cases}$$

де  $M(x, y)$  – значення точки в масці зображення,

$f(A(x, y))$  – значення функції яскравості в точці  $A(x, y)$ ,

$\delta$  – поріг.

Визначення порогу є однією з важливих задач теорії цифрової обробки зображень. Оскільки завелике значення порогу яскравості може призвести до

страсти інформативних точок, що в свою чергу може призвести до погіршення якості обробки зображення. Проте недостатнє значення порогу може призвести до віднесення до фонових точок до інформативних, що в свою чергу спричиняє деяку кількість надлишкової інформації (зашумленості зображення), що призводить до зменшення швидкодії роботи програми у зв'язку із збільшенням кількості вхідної інформації та погіршенням якості сегментації.

В програмі використано гістограмний підхід до визначення порогу. Даний підхід полягає у побудові гістограми яскравості та встановлення порогу в точці у якій на гістограмі спостерігається різке збільшення рівня яскравості. Оскільки наперед відомо, що на зображеннях фон має набагато меншу яскравість (фон наближається до чорного кольору), то використання даного підходу дає хороші результати.

Для збільшення якості виділення не фонових точок система проводить додаткову перевірку на наявність необхідного числа інформативних пікселів. Втрата цінної інформації під час даної перевірки, як показали експериментальні дані, незначна.

В результатів обробки вхідного зображення утворюється бінарне зображення, в якому інформативні пікселі кодуються 1, а фонові – 0.

Для виділення контурів визначених областей в системі використовується покращений (адаптований) алгоритм „Radial Sweep” з поєднаними критеріями зупинки Джакобса („Jacob's stopping criterion”) та зупинкою при заходженні на точку, що належить контуру. Основним результатом покращення є можливість повертатися на  $n$  кроків назад, що дозволяє уникати приєднання до контуру ліній товщиною 1 піксел. Експериментальним шляхом було доведено, що ліній товщиною 1 піксел, як правило є дефектом зображення (шуми, погана якість фотоприймача, тощо) і є малоінформативними.

Критерій зупинки завершує роботу алгоритму при попаданні контуру на точку, яка вже належить до контуру, що дозволяє уникнути зациклення системи.

Після виділення об'єктів необхідно провести їхній опис. Опис зображення – це процес виділення інформативних ознак об'єкта. Зокрема



однією з важливих ознак об'єкта є велика вісь. Для визначення великої осі необхідно визначити на контурі дві найвіддаленіші точки.

Для знаходження великої осі реалізовано процедуру `CentralLine(integer)`. На контурі клітини випадковим чином вибирається стартова точка (піксель) з координатами  $(x, y)$  (рисунок 3.4).

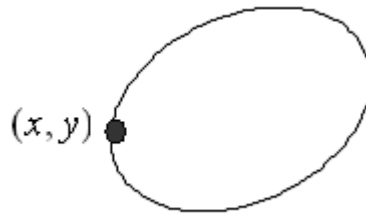


Рисунок 3.4 – Вибір стартової точки для визначення великої осі об'єкта

Визначається наступна, за стартовою, сусідня точка контуру з координатами  $(x_1, y_1)$ . Наступна точка визначається шляхом вибірки з масиву точок, що належать до контуру об'єкта. Після визначення найближчої точки, стартова точка  $(x, y)$  виключається з подальшої обробки.

Обчислюється довжина відрізка  $(x, y) (x_1, y_1)$  за формулою:

$$L = \sqrt{((x_1 - x)^2 + (y_1 - y)^2)}.$$

Після виконання цих дій відшукується наступна точка контуру  $(x_2, y_2)$ , що є сусідньою для точки  $(x_1, y_1)$ . Точка  $(x_1, y_1)$  виключається з подальшої обробки і проходить обчислення довжини відрізка  $(x, y) (x_2, y_2)$ .

Описані кроки повторюються доти, доки не буде знайдено таку точку  $(x_n, y_n)$ , що відрізок, утворений точками  $(x, y)$  і  $(x_n, y_n)$  буде мати максимальну довжину  $L_m$  (рисунок 3.5). Критерієм максимальної довжини

відрізку є зменшення (відсутність зростання) довжин відрізків на наступних

$\frac{P}{2} + 1$  кроках, де  $P$  – периметр об'єкта.

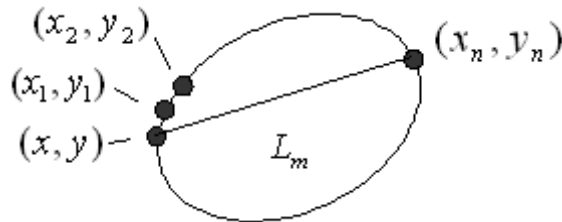


Рисунок 3.5 – Визначення першої ключової точки великої осі

Точка (піксель) з координатами  $(x_n, y_n)$  фіксується як перша ключова точка контуру об'єкта. Визначення другої ключової точки, точки найбільш віддаленої від першої, проходить аналогічно до визначення першої ключової точки. В якості стартової точки вибирається попередньо визначена перша ключова точка з координатами  $(x_n, y_n)$ . Обробка точок проходить в тому ж напрямку за яким проходила визначення першої точки. Після  $m$  кроків отримується друга ключова точка з координатами  $(x_{n+m}, y_{n+m})$  (рисунок 3.6). Точки з координатами  $(x_n, y_n)$  та  $(x_{n+m}, y_{n+m})$  утворюють відрізок, що сполучає дві найбільш віддалені точки контуру об'єкта, тобто утворюють велику вісь. Велика вісь є частиною прямої  $y = kx + b$ , що перетинає клітину, і має найбільшу довжину  $L_{\max}$ .

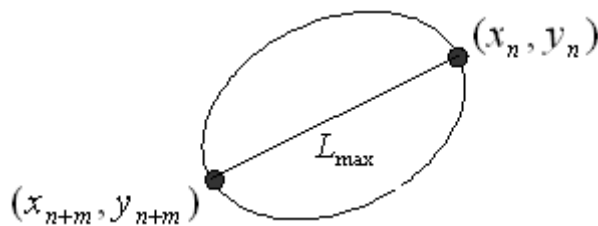


Рисунок 3.6 – Визначення великої осі об'єкта

Велика вісь є однією з основних метричних ознак об'єкта. Даний підхід до визначення великої осі забезпечує високу точність результату при мінімальній трудоемності процесу. Недоліком даного підходу є необхідність обрахунку кожної точки контуру. Звичайно проходження контуром можна робити з деяким кроком  $k$ , проте даний варіант подолання недоліку може негативно впливати на точність роботи. Також до недоліків можна віднести кількість кроків в критерії зупинки перебору при знаходженні ключових точок., проте зменшення цієї кількості в деяких випадках, як показали результати тестування, може мати негативні результати для якості визначення великої осі.

Для спрощення процесу апроксимації контурів об'єктів виконується поворот клітини на кут  $\alpha$ . Для повороту клітини виконується процедура `Poworotklitunu(integer)`. Для виконання повороту клітини проводиться координатна сітка з початком відрізка в точці  $M$ , середині відрізка  $AB$ . Відрізок  $AB$  з'єднує 2 найбільш віддалені точки контуру і найбільшою хордою клітини. Кут між віссю абсцис та найбільшою хордою вибирається як кут повороту.

Кут  $\alpha$  визначається як  $\arctan()$  кутового коефіцієнта прямої  $AB$ :

$$\alpha = \arctan(k),$$

$k$  – кутовий коефіцієнт рівняння прямої  $y = kx + b$ ;

$\alpha$  – кут повороту клітини.

Для повороту клітини в цілому здійснюється поворот кожної точки, що належить даній клітині. Обрахунок нових значень координат точок, для здійснення повороту, проводиться за наступними формулами:

$$X_i = x_i \cos(\alpha) - y_i \sin(\alpha)$$

$$Y_i = x_i \sin(\alpha) + y_i \cos(\alpha)$$

де  $i$  – загальна кількість точок, що входять до контуру клітини,  $i = 1..n$ ;

$x_i$  – початкове значення  $i$ -ої точки контуру зображення відносно вісі абсцис до повороту;

$y_i$  – початкове значення  $i$ -ої точки контуру зображення відносно вісі ординат до повороту;

$X_i$  – значення  $i$ -ої точки контуру зображення відносно вісі абсцис після повороту точки відносно точки  $M$  на кут  $\alpha$ ;

$Y_i$  – значення  $i$ -ої точки контуру зображення відносно вісі ординат після повороту точки відносно точки  $M$  на кут  $\alpha$ ;

$\alpha$  – кут повороту клітини.

Для зменшення кількості обчислювальних операцій під час проведення апроксимації з точок контурної лінії проводиться вибірка точок в яких з контурна функція проходять найбільші зміни (процедури `TochkuAproxVerx(integer)` та `TochkuAproxNuz(integer)` для верхньої та нижньої дуги відповідно). Для покращення точності апроксимації вузлові точки вибираються за трьома критеріями:

– Зміна знаку приросту кривої. Даний критерій вказує на точки дуги АВ, в яких приріст змінює свій знак. Для визначення даного критерію необхідно між двома сусідніми точками, що належать прямій та вибрані з певним кроком  $n$ , провести пряму, що описується рівнянням  $y = kx + b$ , та обрахувати коефіцієнт  $k$ . Коефіцієнт  $k$  визначається з формул. Приклад визначення даного критерію показано нижче (рисунок 3.7). В прикладі між сусідніми точками, що належать дузі АВ та взяті з відповідним кроком  $n$ ,  $K(x_1, y_1)$  та  $M(x_2, y_2)$  коефіцієнт  $k$  має додатне значення, а між точками  $M(x_2, y_2)$  та  $C(x_3, y_3)$  від'ємне, отже точка  $M(x_2, y_2)$  визнається як вузлова.

– Визначення рівня кривизни функції. Визначення рівня кривизни функції. Даний критерій вказує на точки дуги АВ, в яких кривизна дуги більше ніж граничний поріг  $d$ . На основі експериментальних даних, оптимальне значення граничного порога  $d$  було визнано 2,5. Приклад визначення даного критерію показано нижче (рисунок 3.8). Для визначення кривизни дуги необхідно провести ряд додаткових обрахунків. Визначити кутові коефіцієнти

відрізків, що з'єднують дві сусідні пари точок, що належать прямій та вибрані з певним кроком  $n$ , провести прямі, що описується рівнянням  $y_1 = k_1x_1 + b_1$  та  $y_2 = k_2x_2 + b_2$  і обрахувати коефіцієнти  $k_1$  та  $k_2$ . Також необхідно визначити довжину відрізків, що з'єднують сусідні точки:

$$l = \sqrt{((x_2 - x_1)^2 + (y_2 - y_1)^2)}.$$

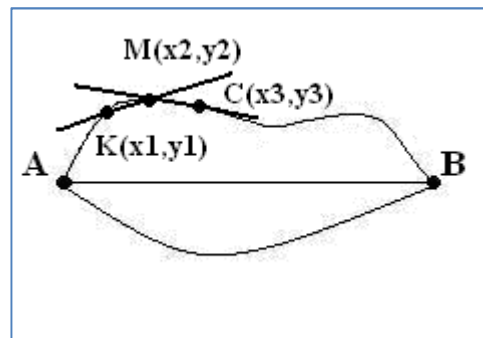


Рисунок 3.7 – Визначення точки в якій проходить зміна знаку контурної функції

Визначається коефіцієнт  $\alpha$ :

$$\alpha = \arctan \left| \frac{k_1 - k_2}{1 + k_2} \right|.$$

Визначається кривизна дуги K:

$$K = \frac{\alpha}{l}.$$

В прикладі між сусідніми точками, що належать дузі АВ та взяті з відповідним кроком  $n$ ,  $K(x_1, y_1)$ ,  $M(x_2, y_2)$  та  $C(x_3, y_3)$  кривизна дуги більша за значення граничного порога  $d$ , отже точка  $M(x_2, y_2)$  визнається вузловою.

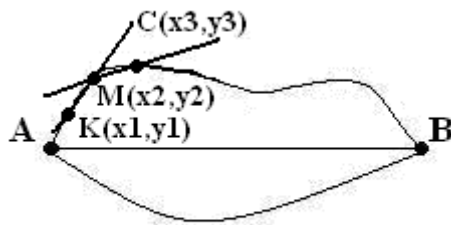


Рисунок 3.8 – Визначення точки в якій кривизна контурної функції перевищує деякий поріг  $\delta_1$

– Відношення приростів функції в сусідніх точках. Даний критерій вказує на точки дуги АВ, в яких відношення приростів відрізняється в більше ніж  $m$  раз. Для визначення даного критерію необхідно між двома сусідніми парами точок, що належать прямій та вибрані з певним кроком  $n$ , провести прямі, що описується рівнянням  $y_1 = k_1x_1 + b_1$  та  $y_2 = k_2x_2 + b_2$  і обчислити коефіцієнти  $k_1$  та  $k_2$ . Приклад визначення даного критерію показано нижче (рисунок 3.9). В прикладі між сусідніми точками, що належать дузі АВ та взяті з відповідним кроком  $n$ ,  $K(x_1,y_1)$  та  $M(x_2,y_2)$  коефіцієнт  $k_1$  більший за коефіцієнт  $k_2$  взятий між точками  $M(x_2,y_2)$  та  $C(x_3,y_3)$  в  $m$  раз, що дозволяє визнати точку  $M(x_2,y_2)$  як вузлову. Коефіцієнт  $m=1.5$ , дане значення було отримане експериментальним шляхом і визнане як оптимальне.

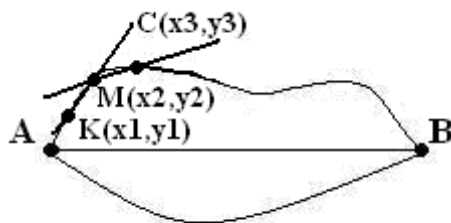


Рисунок 3.9 – Визначення точки в якій приріст контурної функції перевищує деякий поріг  $\delta_2$

Для перевірки якості апроксимації, а також для більш точного передання конфігурації кривої використовується четвертий тип ключових точок – додаткові точки. Дані точки обраховуються, якщо довжина апроксимованої кривої більша за коефіцієнт  $l$ . Коефіцієнт  $l$  вираховується в точках (пікселях) між двома сусідніми ключовими точками, визначеними за допомогою трьох критеріїв визначення ключових точок. Експериментальним шляхом було встановлено, що для якісної роботи алгоритму коефіцієнт  $l \geq 20$ . Приклад визначення даного критерію показано нижче. Точка  $L(x_3, y_3)$  визначається як середина апроксимованої кривої  $M(x_1, y_1)$  та  $K(x_2, y_2)$ . Особливістю даного типу ключових точок є те, що вони необов'язково будуть внесені в пам'ять, вони використовуються лише для додаткової перевірки якості апроксимації контуру.

В результаті об'єднання вузлових точок, відібраних за критеріями, отримується масив вузлових точок, який зберігається в табличному вигляді (таблиця 3.2).

Таблиця 3.2 – Представлення критичних точок контуру

$x_i$	$x_0$	$x_1$	$x_2$	...	$x_n$	...	$x_m$
$f(x_i)$	$y_0$	$y_1$	$y_2$	...	$y_n$	...	$y_m$

Використання критичних точок дозволяє зменшити кількість операцій та збільшити швидкодію програми в 4 рази, оскільки апроксимація здійснюється на основі вибірки, а не цілому масиві значень контурної функції.

Визначення типу кривої  $n$ -го порядку для апроксимації (пряма, парабола, гіпербола) здійснюється за наступним алгоритмом (процедури `TochkuAproxVerx(integer)` та `TochkuAproxNuz(integer)` для верхньої та нижньої дуги відповідно):

1. Ключові точки збираються зліва направо (рисунок 3.10).

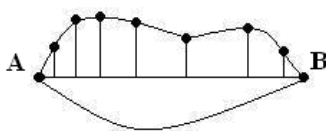


Рисунок 3.10 – Приклад дуги АВ з ключовими точками

2. Визначаються 4 сусідні ключові точки визначені за першим критерієм (зміна знаку приросту функції). Якщо ключових даного типу недостатньо, то переходимо до пункту 4.

3. Проводиться спроба апроксимації дуги гіперболою. Якщо результат апроксимації негативний, то відкидаємо одну з ключових точок (з права). Якщо результат апроксимації позитивний, то переходимо до пункту 2, причому відлік починається з кінця попередньої кривої.

4. Проводиться спроба апроксимації дуги параболою. Якщо результат апроксимації негативний, то відкидаємо одну з ключових точок (справа).

5. Перевіряється кількість ключових точок визначені за першим критерієм (зміна знаку приросту функції). Якщо ключових даного типу більше 2, то переходимо до пункту.4. Якщо результат апроксимації позитивний, то переходимо до пункту 2, причому відлік починається з кінця попередньої кривої.

6. Проводиться спроба апроксимації дуги прямою. Якщо результат апроксимації негативний, то відкидаємо одну з ключових точок (з права) та проводимо ще одну спробу апроксимувати прямою. Якщо результат апроксимації позитивний, то переходимо до пункту 2, причому відлік починається з кінця попередньої кривої.

7. Проводиться перевірка на наявність ключових точок. Якщо ще не всі ключові точки використані, то переходимо до пункту 2. Якщо всі ключові точки використанні, то робота завершується.



Для знаходження коефіцієнтів  $a_2$ ,  $a_1$ ,  $a_0$  (процедури `Аprox()` та `АproxLine()`) використовується наступний алгоритм (даний алгоритм ілюструє знаходження коефіцієнтів параболі):

1. Сформуванати матрицю  $B$ :

$$B = \begin{bmatrix} b_{11} & b_{12} & \dots & b_{1m} \\ b_{21} & b_{22} & \dots & b_{2m} \\ b_{31} & b_{32} & \dots & b_{3m} \end{bmatrix} = \begin{bmatrix} (x_1)^0 & (x_2)^0 & \dots & (x_m)^0 \\ (x_1)^1 & (x_2)^1 & \dots & (x_m)^1 \\ (x_1)^2 & (x_2)^2 & \dots & (x_m)^2 \end{bmatrix},$$

де  $m$  – кількість ключових точок для визначення параболі;

$b_{1i}$  – це  $(x_i)^0=1$ ,  $i=1..m$ ;

$b_{2i}$  – це  $(x_i)^1$ ,  $i=1..m$ ;

$b_{3i}$  – це  $(x_i)^2$ ,  $i=1..m$ .

2. Для формування матриці  $C$  необхідно провести наступне:

$$C = [c_1 \quad c_2 \quad \dots \quad c_m] = [y_1 \quad y_2 \quad \dots \quad y_m],$$

де  $m$  – кількість точок для визначення параболі;

$c_i$  – це  $y_i$ ,  $i=1..m$ .

3. Визначається добуток матриць  $B$  та  $C$ , де

$$D = B \times C,$$

де  $D$  – матриця добутку матриці  $B$  та матриці  $C$ ;

$B$  – матриця  $(x_i)$  в відповідному степені;

$C$  – матриця  $f(x_i)=y_i$ .

4. Для знаходження коефіцієнтів необхідно перемножити матрицю  $D$  на матрицю  $B^{-1}$ . Обернену матрицю  $B^{-1}$  визначити за формулами:

$$X = B^{-1} \times D = [a_2 \quad a_1 \quad a_0],$$

де  $X$  – матриця коефіцієнтів;

$B^{-1}$  – обернена матриця ( $x_i$ ) в відповідному степені;

$D$  – матриця добутку матриці  $B$  та матриці  $C$ .

Для визначення оберненої матриці  $B^{-1}$  необхідно виконати наступні кроки:

1) сформуувати матрицю  $B^T$ , яка є транспонованою матрицею  $B$ :

$$B^T = \begin{bmatrix} b_{11} & b_{12} & \dots & b_{1m} \\ b_{21} & b_{22} & \dots & b_{2m} \\ b_{31} & b_{32} & \dots & b_{3m} \end{bmatrix} = \begin{bmatrix} b_{11} & b_{21} & b_{31} \\ b_{12} & b_{22} & b_{32} \\ \dots & \dots & \dots \\ b_{1m} & b_{2m} & b_{3m} \end{bmatrix},$$

де  $m$  – кількість вузлових точок для визначення параболи;

$b_{1i}$  – це  $(x_i)^0=1$ ,  $i=1..m$ ;

$b_{2i}$  – це  $(x_i)^1$ ,  $i=1..m$ ;

$b_{3i}$  – це  $(x_i)^2$ ,  $i=1..m$ .

2) Обчислити матрицю  $H$ , яка дорівнює добутку матриць  $B$  та  $B^T$

$$H = B \times B^T.$$

3) Обчислити мінори матриці  $H$ .

4) Визначити алгебраїчні доповнення матриці  $H$ :

$$A_{ij} = (-1)^{i+j} M_{ij},$$

де  $A_{ij}$  – алгебраїчне доповнення елемента  $i$ -го рядка,  $j$ -го стовпця;

$i$  – кількість рядочків в матриці  $H$ ,  $i=1..m$ ;

$j$  – кількість стовпців в матриці  $H$ ,  $i=1..3$ ;

$M^{ij}$  – мінор елемента  $i$ -го рядка,  $j$ -го стовпця.

5) Транспонувати матрицю алгебраїчних доповнень  $A^T$

6) Визначити детермінанти матриці  $A^T$

$$d = (a_{11} \times a_{22} - a_{12} \times a_{21}).$$

7) Отримати обернену матрицю  $B^{-1}$ :

$$B^{-1} = A^T / d .$$

Знаходження коефіцієнтів гіперболи відбувається аналогічно, але матриця  $B$  матиме наступний вигляд:

$$B = \begin{bmatrix} b_{11} & b_{12} & \dots & b_{1m} \\ b_{21} & b_{22} & \dots & b_{2m} \\ b_{31} & b_{32} & \dots & b_{3m} \\ b_{41} & b_{42} & \dots & b_{4m} \end{bmatrix} = \begin{bmatrix} (x_1)^0 & (x_2)^0 & \dots & (x_m)^0 \\ (x_1)^1 & (x_2)^1 & \dots & (x_m)^1 \\ (x_1)^2 & (x_2)^2 & \dots & (x_m)^2 \\ (x_1)^3 & (x_2)^3 & \dots & (x_m)^3 \end{bmatrix},$$

де  $m$  – кількість ключових точок для визначення гіперболи;

$b_{1i}$  – це  $(x_i)^0 = 1, i=1..m$ ;

$b_{2i}$  – це  $(x_i)^1, i=1..m$ ;

$b_{3i}$  – це  $(x_i)^2, i=1..m$ ;

$b_{4i}$  – це  $(x_i)^3, i=1..m$ .

Детермінант визначається за правилом трикутника.

Апроксимація прямих відбувається за наступними формулами:

$$b = \frac{(x_1 \cdot y_2 - x_2 \cdot y_1)}{x_1 - x_2},$$

де  $M(x_1, y_1)$  – координати першої сусідньої точки;

$K(x_2, y_2)$  – координати другої сусідньої точки;

$b$  – коефіцієнт прямої, що з'єднує дві сусідні точки.

$$k = \frac{(y_1 - b)}{x_1},$$

де  $M(x_1, y_1)$  – координати першої сусідньої точки;

$b$  – коефіцієнт прямої, що з'єднує дві сусідні точки;

$k$  – коефіцієнт прямої, що з'єднує дві сусідні точки.

Для перевірки похибки апроксимації кривої, використовуються наступні критерії (процедури `Апрох()` та `АпрохLine()`):

а) проводиться перевірка сумарної абсолютної похибки апроксимації:

$$E = \sum_{i=1}^m |f(x_i) - f(x_i^*)|,$$

де  $E$  – сумарна похибка апроксимації;

$m$  – кількість ключових точок для визначення апроксимуючої кривої  $n$ -го порядку;

$f(x_i)$  – значення функції  $f(x)$  в  $i$ -ій точці;

$f(x_i^*)$  – значення функції  $f(x)$  в  $i$ -ій точці обчислене за допомогою апроксимуючих кривих  $n$ -го порядку.

Якщо  $E$  більша за задану похибку, то обчислений апроксимуюча крива вважається неефективною.

б) перевірка точності в певній точці:

$$e_i = |f(x_i) - f(x_i^*)|,$$

де  $e_i$  – похибка значення в  $i$ -ій точці,  $i = 1..m$ ;

$m$  – кількість ключових точок для визначення апроксимуючої кривої  $n$ -го порядку;

$f(x_i)$  – значення функції  $f(x)$  в  $i$ -ій точці;

$f(x_i^*)$  – значення функції  $f(x)$  в  $i$ -ій точці обчислене за допомогою апроксимуючих кривих.

Якщо похибка апроксимації в деякій точці  $D(x_n, y_n)$  більша за допустиму, то спроба апроксимації визнається неефективною.

Перевірка точності апроксимації здійснюється для кожної апроксимуючої кривої  $n$ -го. Якщо апроксимуюча крива була визнана неефективною, то вводяться додаткові ключові точки та перераховується тільки дана крива, що дозволяє працювати з окремими кусками частинами контурної функції. Даний підхід забезпечує мінімальну кількість обрахунків, якщо необхідно повторити (збільшити точність) процес апроксимації, що дозволяє збільшити точність апроксимації при невеликих втратах швидкодії.

### 3.3 Комп'ютерні експерименти

Основною метою розпаралелення процесу оцінки якості сегментації зображень є зменшення часу опрацювання зображень та підвищення продуктивності роботи алгоритмів.

Важливим етапом розпаралелення алгоритмів є забезпечення синхронізації.

Узагальнену структуру системи оцінки якості сегментації на основі метрик наведено на рисунку 3.11.

Ярусно-паралельна форма представлення розпаралелення процесу оцінки якості сегментації наведено на рисунку 3.12.

Приклади полігональних об'єктів наведено у таблиці 3.3.

На рисунку 3.13 зображено порівняльний аналіз часу процесу оцінки якості сегментації при послідовному виконанні та з використанням розпаралелення.

Середній час процесу оцінки якості сегментації приведений на рисунку 3.14.

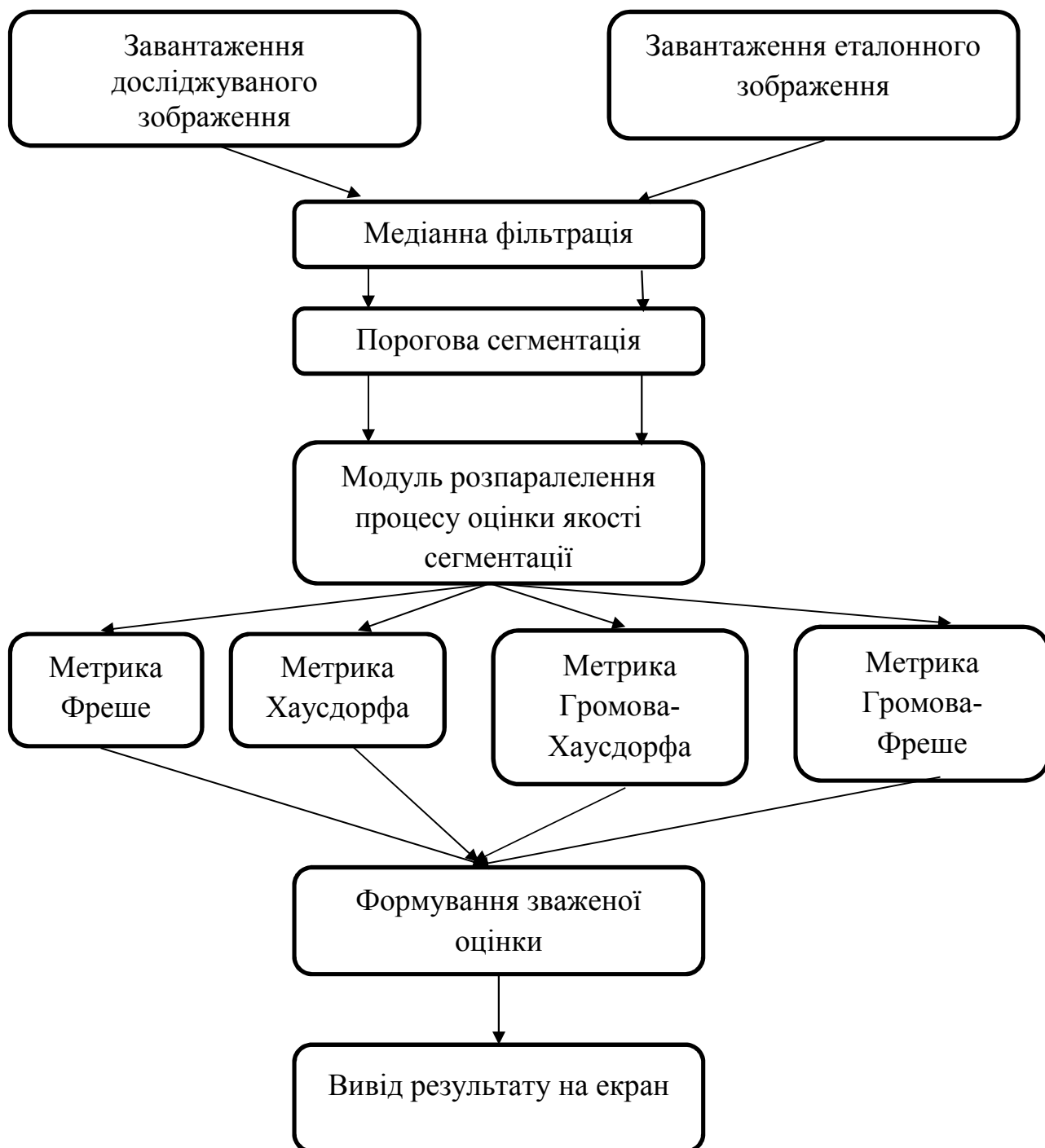


Рисунок 3.11 – Загальна структура системи оцінки якості сегментації на основі метрик



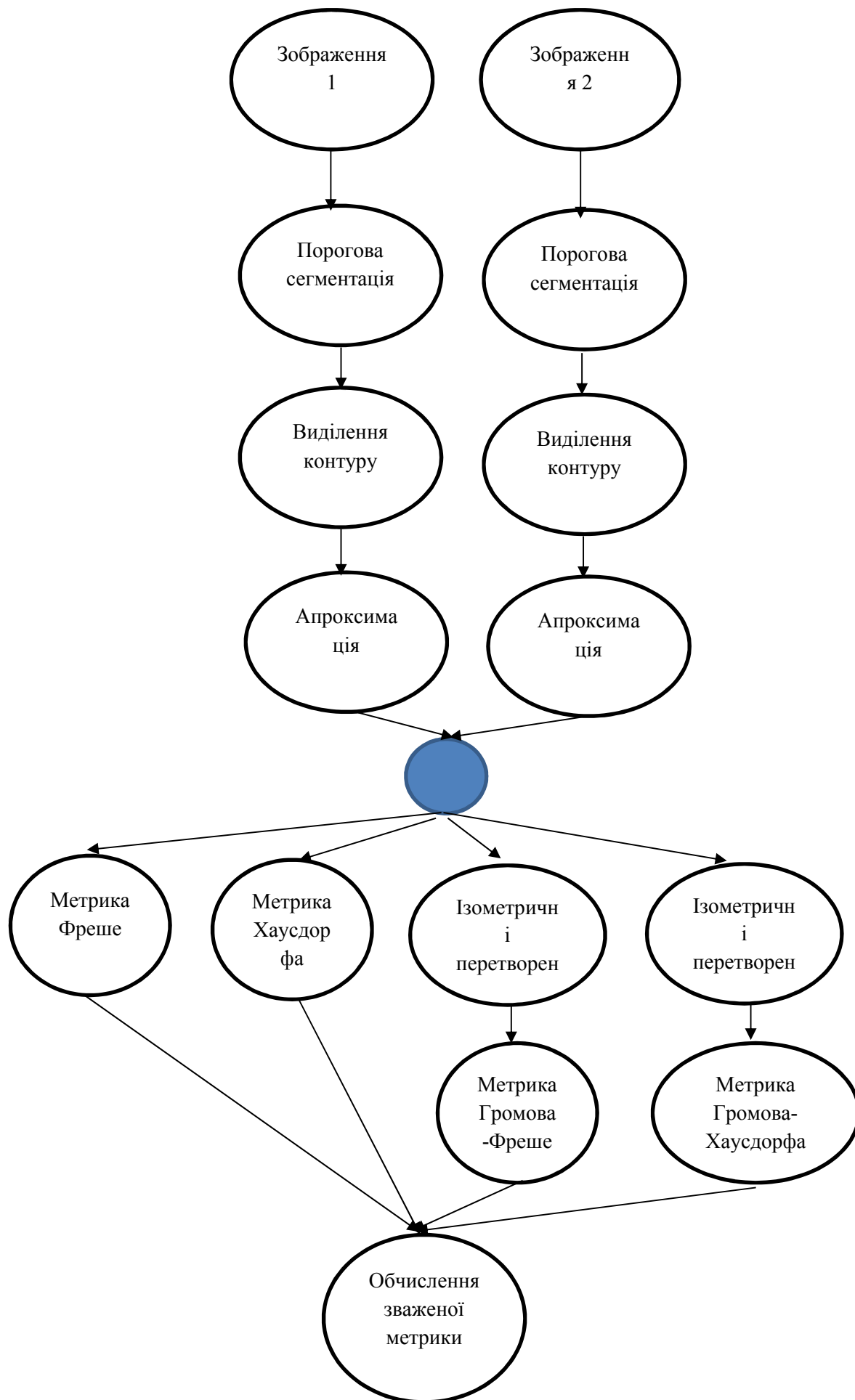











Рисунок 3.12 – Ярусно-паралельна форма представлення процесу розпаралелення



Таблиця 3.3 – Приклади полігональних об'єктів

№	Об'єкт 1	Об'єкт 2	Об'єкт 3
1			
2			
3			

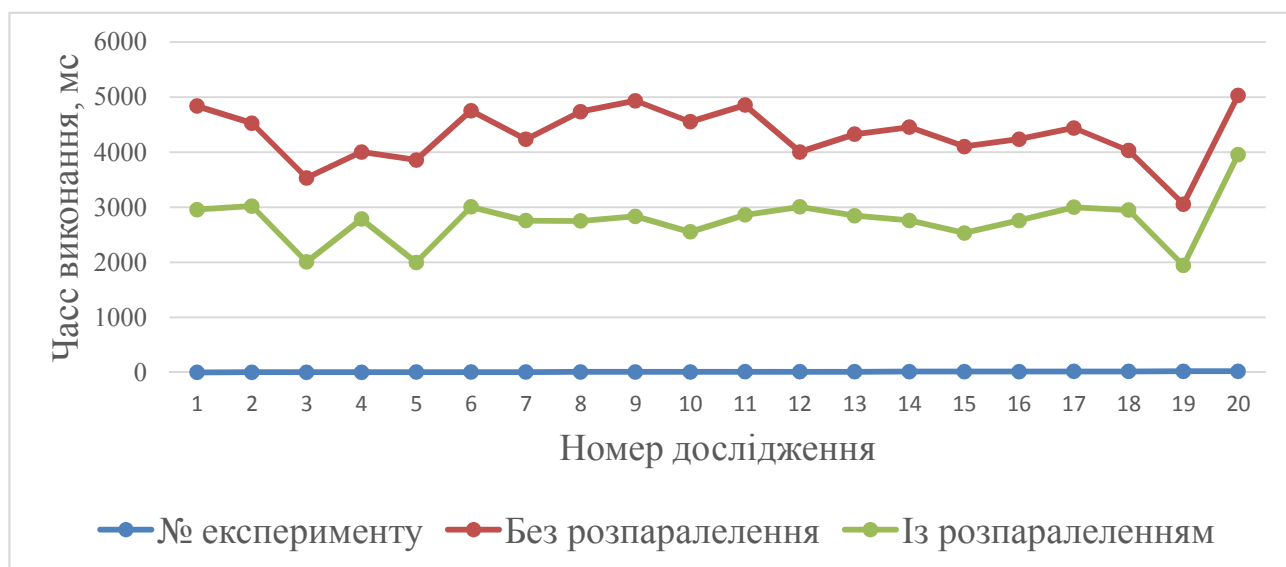


Рисунок 3.13 – Порівняльний аналіз часу процесу оцінки якості сегментації при послідовному виконанні та з використанням розпаралелення

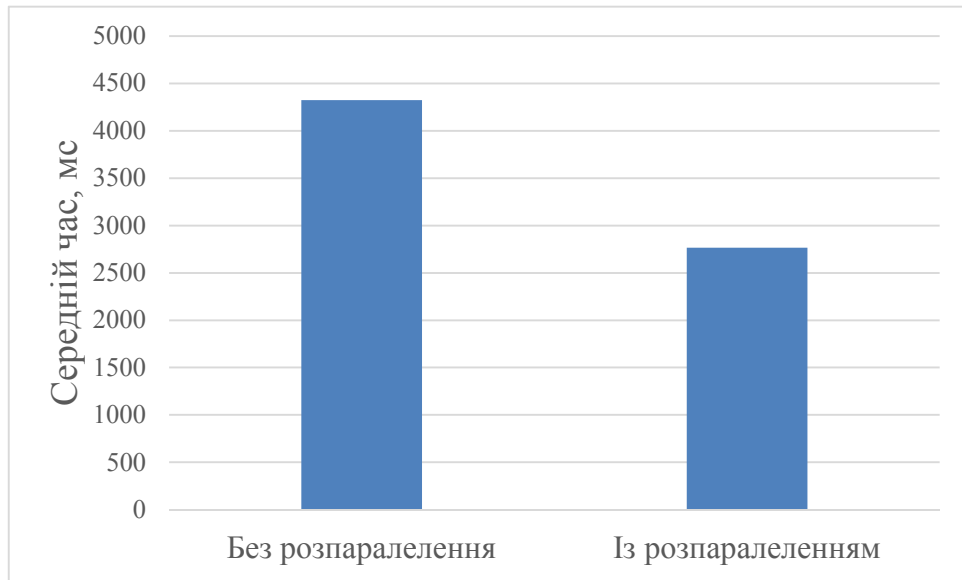


Рисунок 3.14 – Середній час процесу оцінки якості сегментації

## ВИСНОВКИ

В результаті виконання магістерської роботи одержані такі результати:

3. Проаналізовано відомі алгоритми сегментації: порогового обмеження по яскравості, центроїдного зв'язування, злиття-розщеплювання, розфарбовування зображень, проведено аналітичний огляд існуючих систем аналізу зображень, що дало змогу показати актуальність задач сегментації і поставити магістерського дослідження.
4. Досліджено алгоритми оцінки якості сегментації зображень.
5. Досліджено технології розпаралелення інформації та проаналізовано графічні процесори.
6. Розроблено алгоритм автоматичного вибору алгоритмів сегментації на основі метрик
7. Розроблено алгоритм розпаралелення алгоритмів оцінки сегментації зображень.
8. Програмно реалізовано модуль оцінки сегментації зображень

## СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. А.О Дарченко, И.П. Романов, А.П. Креницкий и др. Автоматизация и компьютеризация цитологических исследований в гематологии. Биомедицинские технологии и радиоэлектроника, 2003, №2. – С. 55-61.
2. Мельник А.Н. Цитоморфологическая диагностика опухолей. К.: Здоров'я, 1983.-240 с.
3. Фильченков А.А., Стойка Р.С. Апоптоз и рак. Киев. Морион, 1999, 189 с.
4. Г.М., Попова, В.Н. Степанов. Анализ и обработка изображений медико-биологических объектов. Автоматика и телемеханика, 2004, №1, С.131-142.
5. Дуда Р. Распознавание образов и анализ сцен / Дуда Р., Харт П. – М.: Мир, 1976. – 512 с.
6. Путятин Е.П., Аверин С.И. Обработка изображений в робототехнике. – М.: Машиностроение, 1990. – 320 с.
7. Павлидис Т. Алгоритмы машинной графики и обработки изображений. М. Радио и связь, 1986.
8. Гонсалес Р. Цифровая обработка изображений / Р. Гонсалес, Р.Вудс // М.: Техносфера. – 2012. – 1104 с.
9. Гонсалес Р. Цифровая обработка изображений / Р. Гонсалес, Р.Вудс // М.: Техносфера. – 2012. – 1072 с.
10. Форсайт Дж. Компьютерное зрение. Современный подход. – М.: Вильямс. – 2004.
11. Karimi K. A Performance Comparison of CUDA and OpenCL [Электронный ресурс] / K. Karimi, N. G. Dickson, F. Hamze // Computing Research Repository - CORR. – 2010. – Режим доступа до ресурсу: <https://arxiv.org/ftp/arxiv/papers/1005/1005.2581.pdf>.
12. Параллельные вычисления [Электронный ресурс] – Режим доступа до ресурсу: [https://ru.wikipedia.org/wiki/Параллельные\\_вычисления](https://ru.wikipedia.org/wiki/Параллельные_вычисления).

13. Kalaiselvi T. Survey of using GPU CUDA programming model in medical image analysis / T. Kalaiselvi, P. Sriramakrishnan, K. Somasundaram // Informatics in Medicine Unlocked. – 2017. – Vol. 9. – P. 133-144.
14. Karimi K. A Performance Comparison of CUDA and OpenCL [Электронный ресурс] / K. Karimi, N. G. Dickson, F. Hamze // Computing Research Repository - CORR. – 2010. – Режим доступа до ресурсу: <https://arxiv.org/ftp/arxiv/papers/1005/1005.2581.pdf>.
15. Zhang Y.J. A review of recent evaluation methods for image segmentation / Y.J. Zhang // Proc. of Sixth International Symposium on Signal Processing and its Applications( ISSPA 2001). – Vol.1. – 2001. – P.148-151.
16. Lee S.U. A comparative performance study of several global thresholding techniques for segmentation / S.U. Lee, S.Y. Chung, R.H.Park // Computer Vision, Graphics, and Image Processing, Vol.52(2). – 1990. – P. 171-190.
17. Zhang Y.J. Segmentation evaluation using ultimate measurement accuracy/ Y.J. Zhang, J.J.Gerbrands // Proceedings CVPR. – Vol. 1657. – 1992. – P.449-460
18. Zhang Y.J. Objective and quantitative segmentation evaluation and comparison / Y.J. Zhang, J.J.Gerbrands // Signal Processing, Vol.39, No.1-2, 1994, pp.43-54.
19. Lopez M. Hausdorff approximation of convex polygons / M. A. Lopez, S. Reisner // Computational Geometry. – 2005. – Vol. 32(2). – P.139–158. DOI: 10.1016/j.comgeo.2005.02.002.
20. Alt H. Computing the Fréchet distance between two polygonal curves / Alt H., M. Godau // Int. J. of Computational Geometry and Applications. – 1995. – Vol 5. – P.75-91.
21. Chew L. P. Getting around a lower bound for the minimum Hausdorff distance / L. P. Chew, K. Kedem // Computational Geometry. – 1998. – Vol. 10 (3). – P. 197-202. DOI: S0925-7721(97)00032-1.
22. Knauer C. Approximate nearest neighbor search under translation invariant hausdorff distance / C. Knauer, M. Scherfenberg // International Journal of

Computational Geometry. – 2011. – Vol. 21(3). – P. 369–381. DOI: S0218195911003706.

23. Alvarez V. Approximating the minimum weight spanning tree of a set of points in the Hausdorff metric / V. Alvarez, R. Seidel // Computational Geometry. – 2010. – Vol. 43. – P. 94-98.

24. Atallah M.J. Computing Some Distance Functions Between Polygons / M. J. Atallah; C. Celso // Computer Science Technical Reports. – 1990. - Vol . 9. - pp.11-20.

25. Mosig A. Approximately matching polygonal curves with respect to the Fréchet distance / A. Mosig, M. Clausen // Computational Geometry. – 2005. – Vol. 30(2) – P. 113-127. DOI: 10.1016/j.comgeo.2004.05.004.

26. Buchin K. Computing the Fréchet distance between simple polygons / K. Buchin, M. Buchin, C. Wenk // Computational Geometry. – 2008. – Vol. 44(1-2). – P. 2–20. DOI: 10.1145/1137856.1137870.

27. Rote G. Computing the Fréchet distance between piecewise smooth curves / G. Rote // Computational Geometry. – 2007. – Vol. 37. – P. 162–174. DOI: 10.1016/j.comgeo.2005.01.004.

28. Schlesinger M. I. Fréchet Similarity of Closed Polygonal Curves / M. I. Schlesinger, E. V. Vodolazskiy, V. M. Yakovenko // International Journal of Computational Geometry. – 2016. – Vol. 26. – P. 53–66. DOI: 10.1142/S0218195916500035.

29. Computing the discrete Fréchet distance with imprecise impute / [H.-K. Ahn, C. Knauer, M. Scherfenberg et al.] // International Journal of Computational Geometry. – 2016. – Vol. 22. – P. 27–44. DOI: 10.1142/S0218195912600023.

30. Computing the Fréchet distance between folded polygons / [A. F. Cook, Anne Driemel, Jessica Sherette et al.] // Computational Geometry. – 2015. – Vol. 50. – P. 1-16.

31. Gudmundsson J. Fast algorithms for approximate Fréchet matching queries in geometric trees / J. Gudmundsson, M. Smid // Computational Geometry. – 2015. – Vol. 48. – P. 479–494. DOI:10.1016/j.comgeo.2015.02.003.

32. Betanzos A.A. Analysis and evaluation of hard and fuzzy clustering segmentation techniques in burned patient images / A.A. Betanzos // IVC. – Vol. 18(13). – 2000. – P. 1045-1054.

33. Zhang Y.J. Objective and quantitative segmentation evaluation and comparison / Y.J. Zhang, J.J.Gerbrands // Signal Processing. – Vol. 39. – 1994. – P. 43-54.

34. Zhang Y.J. Segmentation evaluation using ultimate measurement accuracy / Y.J. Zhang, J.J.Gerbrands // SPIE. – Vol. 1657. – 1992. – P. 449-460.

35. Березький О.М. Аналіз метрик знаходження відстані між областями зображень для кількісної оцінки результатів сегментації / О.М. Березький, Г.М. Мельник, Ю.М. Батько, О.Й. Піцун // ISDMCI'2016. XII міжнародна наукова конференція. Інтелектуальні системи прийняття рішень та проблеми обчислювального інтелекту 24-28 травня 2016р. – Херсон: Видавництво ПП Вишемирський В.С., 2016. – С. 252-253.

36. Berezsky O. M. Regions Matching Algorithms Analysis to Quantify the Image Segmentation Results / O.M. Berezsky, G. M. Melnyk, Y. M. Batko, O.Y. Pitsun // Sensors & Transducers. – 2017. – Vol. 208(1). – pp. 44-49.

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

38. Berezsky O. Regions Matching Algorithms Analysis to Quantify the Image Segmentation Results / O. Berezsky, G. Melnyk, Y. Batko, O. Pitsun // Proceedings of the IEEE International Conference «Computer Science and Information Technologies» CSIT'2016, Lviv. Ukraine - 6-10 September, 2016. – P. 33-36.

39. Berezsky O.M. Evaluation methods of image segmentation quality / O.M. Berezsky, O.Y. Pitsun // *Радіоелектроніка, інформатика, управління.* - 2018.-№5. – С. 41-61.

40. Университет ИТМО. Триангуляция полигонов (ушная + монотонная). [Электронный ресурс] - Режим доступа: [http://neerc.ifmo.ru/wiki/index.php?title=Триангуляция\\_полигонов](http://neerc.ifmo.ru/wiki/index.php?title=Триангуляция_полигонов).

41. Berezsky O. Computation of the minimum distance between non-convex polygons for segmentation quality evaluation / O. Berezsky, O. Pitsun // XIIth International Scientific and Technical Conference Computer Sciences and Information Technologies. CSIT 2017. 5-8 Sept. 2017 – P. 183-186.

42. Eiter Th. Computing Discrete Fréchet Distance [Text] / Thomas Eiter, Mannila Heikki // International Journal of Computational Geometry & Applications. – 1994: pp. 1-7.

43. Alt H. Computing the Fréchet distance between two polygonal curves / Alt H., M. Godau // Int. J. of Computational Geometry and Applications. – 1995. – Vol 5. – P.75-91.

44. Eiter Th. Computing Discrete Fréchet Distance [Text] / Thomas Eiter, Mannila Heikki // International Journal of Computational Geometry & Applications. – 1994: pp. 1-7.

45. Berezsky O. Automated Processing of Cytological and Histological Images / O. Berezsky, O. Pitsun // Proceedings of XII International Conference Perspective Technologies and methods in mems design (MEMSTECH 2016), 20-24 April, 2016, Lviv-Polyana, Ukraine, P. 51-53.

46. 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.110-123.

47. Березький О.М. Алгоритми опрацювання біомедичних зображень на основі графічних процесорів / О.М. Березький, Б.О. Фірко́вський, М.І. Хомин // Матеріали III всеукраїнської науково-практичної конференції «Прикладна геометрія та інформаційні технології в моделюванні об'єктів, явищ і процесів» (AGIT-2018), м. Миколаїв, 17–19 жовтня 2018 р. – Миколаїв: МНУ імені В.О. Сухомлинського, 2018. – С. 118-119.



48. Методичні рекомендації до виконання дипломної роботи з освітньо-кваліфікаційного рівня «Магістр». Спеціальність «Комп'ютерні системи та мережі» / О.М. Березький, Л.О. Дубчак, Г.М. Мельник / Під ред. О.М. Березького – Тернопіль: ТНЕУ, 2016.– 47 с.