Міністерство освіти і науки України

Харківський національний університет імені В.Н. Каразіна

Факультет комп'ютерних наук

Спеціальність 125 «Кібербезпека»

Освітня програма «Безпека інформаційних та комунікаційних систем»

«Допущено до захисту»

Зав. кафедрою БІСТ

Сергій РАССОМАХІН

_____

« » 2022р.

Пояснювальна записка

до кваліфікаційної роботи магістра

на тему: «Стеганографічна система приховування даних в векторних зображеннях: моделювання та програмна реалізація»

оцінка « »

Голова ЕК

Доценко С.І. _____

Керівник проф. Кузнецов О. О.

Рецензент проф. Толстолузька О. Г

Виконавець : студентка групи КБ-61

Кононченко А. В.

Харків – 2022

РЕФЕРАТ

Кваліфікаційна робота магістра, 77 с., 54 рис., 20 табл., 46 джерел.

Робота містить вступ, 4 розділи, висновки, список джерел інформації та 5 додатків.

Об’єкт дослідження: процеси стеганографічних перетворень над зображеннями векторної графіки.

Предмет дослідження: методи стеганографічних перетворень у точково-задані криві, які здатні забезпечувати стійкість до різного роду афінних перетворень.

Завдання дослідження:

- аналіз та порівняльне дослідження відомих методів приховування інформації у векторні зображення;
- обґрунтування та викладення сутності обраного перспективного методу, в основі якого лежить експлуатація алгоритму побудови кривої Без’є третього порядку;
- програмна реалізація системи, що дозволяє виконувати стегоперетворення;
- експериментальне дослідження ефективності такого методу.

Методи дослідження: математичне та програмне моделювання стеганографічної системи приховування даних в векторних зображеннях мовою програмування Java задля оцінки ефективності можливості приховування секретних даних, аналіз та обґрунтування отриманих результатів.

Під час проведення роботи було розглянуто та імплементовано мовою програмування Java методи стеганографічного вбудовування та вилучення даних у зображення векторної графіки, використовуючи математичний алгоритм побудови кривих Без’є третього ступеня. Виконано дослідження можливих показників та

критеріїв до стеганографічних систем та висунуто вимоги до системи, що реалізовується.

Експериментальні дослідження мають на меті продемонструвати швидкодію імплементованого методу при бітовому та патерновому режимах, а також відсотки збільшення стегоконтейнерів у порівнянні з оригінальним зображенням. Виявлено, що найбільшою мірою впливу на розмір є параметр точності, що застосовується при проведенні розрахунків нових точок при розриві кривої.

Ключові слова: ПРИХОВУВАННЯ ІНФОРМАЦІЇ, СТЕГАНОГРАФІЯ, ВЕКТОРНА ГРАФІКА, КРИВІ БЕЗ’Є, JAVA, БЕКЕНД

ABSTRACT

Qualifying work of a master, 77 pages, 54 images, 20 tables, 46 references.

The work consists of the introduction, 4 sections, conclusions, a list of references and 5 applications.

Object of study: processes of steganographic transformations over images of vector graphics.

Subject of study: methods of steganographic transformations into point-given curves, which are able to provide resistance to various kinds of affine transformations.

The objectives of the study are:

- analysis and comparative study of known methods of hiding information in vector images;
- justification and presentation of the essence of the chosen perspective method, which is based on the operation of the algorithm for constructing the Bezier curve of the third order;
- software implementation of the system, which allows you to perform hip transformation;
- experimental study of the effectiveness of this method.

Research methods: mathematical and software modeling of the steganographic system of data hiding in vector images in the Java programming language to assess the effectiveness of the possibility of hiding secret data, analysis and justification of the obtained results.

During the work, methods of steganographic embedding and data extraction into vector graphics were considered and implemented in the Java programming language using the mathematical algorithm for constructing Bezier curves of the third degree. The study of possible indicators and criteria for steganographic systems was carried out and requirements for the implemented system were put forward.

Experimental studies aim to demonstrate the speed of the implemented method in bit and pattern modes, as well as the percentage of increase in stegocontainers compared to the original image. It has been found that the greatest effect on size is the accuracy parameter used by calculations of new points when breaking a curve.

Key words: INFORMATION HIDING, STEGANOGRAPHY, VECTOR GRAPHICS, BEZIER CURVES, JAVA, BACKEND

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАЧЕНЬ 8
ВСТУП 9
1 АНАЛІЗ ТА ДОСЛІДЖЕННЯ ВІДОМИХ МЕТОДІВ ПРИХОВУВАННЯ ДАНИХ У ВЕКТОРНИХ ЗОБРАЖЕННЯХ 11
1.1 Математичні засади побітового та патернового методів 11
1.2 Афінні атаки на векторні зображення 17
1.3 Дослідження можливих шляхів удосконалення та обґрунтування вибору подальших досліджень 18
1.4 Висновки до першого розділу 21
2 МОДЕЛЬ ТА МЕТОД ПРИХОВУВАННЯ ДАНИХ У ВЕКТОРНІ ЗОБРАЖЕННЯ 22
2.1 Розробка та дослідження критеріїв та показників ефективності стеганосистем 22
2.2 Особливості контейнерів векторної графіки 31
2.2 Моделювання приховування даних у векторні зображення 35
2.3 Моделювання вилучення даних з векторних зображень 40
2.4 Висновки до другого розділу 44
3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ЕКСПЕРИМЕНТАЛЬНІ ДОСЛІДЖЕННЯ ПЛАТФОРМИ ДЛЯ ВИКОНАННЯ СТЕГАНОГРАФІЧНИХ ПЕРЕТВОРЕНЬ НАД ОБ’ЄКТАМИ ВЕКТОРНОЇ ГРАФІКИ 46
3.1 Обґрунтування вибору мови програмування, додаткових бібліотек та інструментів, середовища розробки 46
3.2 Розробка інтерфейсу та структури стеганографічної платформи 48
3.3 Розробка програмної моделі обробки контейнера 56
3.4 Розробка програмної реалізації алгоритмів приховування інформації з векторних зображень 60

3.5 Розробка програмної реалізації алгоритмів вилучення інформації з векторних зображень 68
3.6 Висновки до третього розділу 73
4 ЕКСПЕРИМЕНТАЛЬНІ ДОСЛІДЖЕННЯ РОЗРОБЛЕНИХ АЛГОРИТМІВ, ОБҐРУНТУВАННЯ ПРАКТИЧНОЇ РЕКОМЕНДАЦІЇ ЩОДО ЇХ ВПРОВАДЖЕННЯ 74
4.1 Дослідження показників втрати біт при атаках афінних перетворень 74
4.2 Дослідження швидкісних характеристик, коефіцієнтів спотворення та збільшення розміру контейнерів 75
4.2 Практичні рекомендації щодо подальшого впровадження 77
4.3 Висновки до четвертого розділу 78
ВИСНОВКИ 79
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ 82
ДОДАТОК А 87
ДОДАТОК Б 90
ДОДАТОК В 95
ДОДАТОК Г 112
ДОДАТОК Д 121

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАЧЕНЬ

JPEG – Joint Photographic Experts Group

SVG – Scalable Vector Graphic

XML – Extensible Markup Language

JSON – JavaScript Object Notation

HTML – Hypertext Markup Language

DOM – Document Object Model

UML – Unified Modeling Language

IDE – Integrated Development Environment

ГКІ – Графічний Користувацький Інтерфейс

CSS – Cascading Style Sheets

FXML – FX Markup Language

UI – User Interface

API – Application Programming Interface

ТП – Таблиця Патернів

ВСТУП

Стеганографія дозволяє впроваджувати такі способи комунікації, з використанням яких ускладнюється несанкціоноване отримання даних з певного каналу зв’язку. Поширеними вважаються методи кодування інформації у цифрових зображеннях растрової графіки, де одиницею аналізу є піксель. Однак більшість методів вразливі до великої кількості атак, що приводять до руйнування секретного повідомлення.

Вирішенням такої проблеми можуть стати методи вбудовування інформації у векторні зображення, які будуть стійкі до різного роду афінних перетворень. У зв’язку з тим, що одиницею векторної графіки є геометричні елементи, можливим є проведення стеганографічних перетворень із вбудовування та вилучення секретної інформації.

Використання зображень векторної графіки набуває широкої популярності, особливо в інтернет індустрії. Широка вживаність дозволяє використовувати такі контейнери для прихованої передачі інформації або збереження ключових даних, за якими можна проводити співвідношення щодо авторства, міток часу тощо.

Значна кількість методів таємного збереження інформації у зображеннях растрової графіки, так само як і методів їх вилучення або зруйнування, на сьогоднішній день відома. Однак область векторної графіки вважається не повністю дослідженою у сенсі стеганографічного втручання, а тому може стати новим етапом питання секретного збереження інформації з можливою подальшою передачею отримувачу повідомлення.

Найбільш гнучким контейнером для векторної графіки є SVG-файли, які представляють основану на XML мову розмітки, а також набір пов’язаних інтерфейсів графічних сценаріїв. Над даним форматом найбільш зручно виконувати стеганографічні перетворення, що пов’язано з можливістю отримання доступу до внутрішніх елементів через DOM-структуру.

Важливим елементом будь-якої стеганографічної системи є наявність ключа для отримання секретної інформації та забезпечення таємності такого ключа. У даній роботі досліджуються можливості підвищення таємності стегосистеми шляхом приховування стегоключів у стегоконтейнері.

Метою роботи є дослідження моделей та методів стеганографічних перетворень елементів векторних зображень, їх програмна реалізація та експериментальні дослідження. Мета досягається шляхом аналізу існуючих методів вбудовування секретних даних у елементи векторної графіки. Створення програмної реалізація стеганографічної системи базується на алгоритмі використання кривих Без'є третього порядку О. Кінзерявого. У результаті виконаної роботи виконуються експериментальні дослідження, аналізуються результати та розглядаються рекомендації щодо методу для формування найбільш ефективних параметрів вбудовування та обрання контейнеру.

1 АНАЛІЗ ТА ДОСЛІДЖЕННЯ ВІДОМИХ МЕТОДІВ ПРИХОВУВАННЯ ДАНИХ У ВЕКТОРНИХ ЗОБРАЖЕННЯХ

Пошуком та розробкою шляхів приховування інформації у зображення векторної графіки займаються вже давно [1] [2] [3] [4] [5] [6] [7] [8] [1-8]. Зокрема, у даній роботі досліджуються праці [9-10] [9]- [10], у яких запропоновані методи базуються на використанні кривих Без'є третього ступеня.

1.1 Математичні засади побітового та патернового методів

У термінах векторної графіки криві Без'є призвані для представлення геометричних об'єктів, що визначаються чотирьома контрольними точками P_0, P_1, P_2, P_3 [11]. При цьому крива точно проходить через першу та останню контрольні точки, а середні точки розраховуються через поліноміали [12]:

$$P_n(t) = \sum_{i=0}^{n} B_n^i(t) P_i,\ t = t + \Delta t,\ \ t \in [0,1], \tag{1.1}$$

де n – ступінь кривої,

$B_n^i(t)$ – поліном Бернштейна,

t – параметр побудови кривої, який змінюється згідно з кроком Δt,

P_i – це опорні точки кривої ($i \in \overline{0,n}$).

Поліном Берштейна виконує роль обчислення ваги опорної точки і є базисною функцією кривої Без'є та розраховується через біноміальний коефіцієнт [13]. Поліноми Бернштейна, обмежені інтервалом [0, 1], і є тим елементом, на основі якого створюються криві Без'є. Розрахунок точок кривої методом прямого знаходження значень поліномів хоч і є задачею швидкою, але чисельно стабільним шляхом обрахунку є використання рекурсивного алгоритму де Кастельжо [14]. Чисельна стабільність часто розглядається бажаною властивістю алгоритмів, що мають справу з обрахунками, при яких важливим є збереження деякої заданої точності [15].

Говорячи про чисельну стабільність, необхідно зауважити, що у контексті даного дослідження така властивість відіграє дуже важливу роль. У зв'язку з тим, що алгоритм де Кастельжо запроваджує можливість виконання розподілу кривої у точці бажаного параметру, то збереження заданої точності контрольних точок результуючих сегментів є пріоритетом при виконанні стеганографічних перетворень вбудовування та вилучення інформації.

На рис. 1.1 відображено приклад реальної кривої Без'є (зображення отримано з графічного редактора Inkscape [16]) із визначеними опорними точками. Як видно, траєкторія самої прямої дійсно проходить лише через першу P_0 та останню P_3 точки, середні ж точки P_1 та P_2 слугують деякими важелями для визначення ступеня ввігнутості кривої.

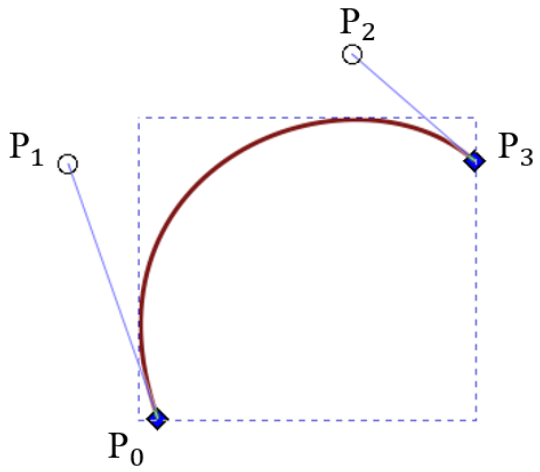


Рисунок 1.1 – Приклад простої кривої Без'є

Так, алгоритм де Кастельжо і відповідає за те, аби побудувати криву за опорними точками, що можна виконати, використовуючи:

$$P_i^0 = P_i,\ i = 0,\ldots,n,$$
$$P_i^j = P_i^{j-1}(1-t_0) + P_{i+1}^{j-1}t_0,\ i = 0,\ldots,n-j,\ j = 1,\ldots,n. \quad (1.2)$$

де n – ступінь кривої,

t – параметр побудови кривої,

P_i – опорні точки кривої ($i \in \overline{0,n}$).

Вираз (1.2) можна представити у вигляді таблиці розрахунків [17]:

Таблиця 1.1 – Алгоритм де Кастельжо для розрахунку кривих Без'є

$j \setminus i$	0	1		$n-2$	$n-1$	n
0	$P_0^0 = P_0$	$P_1^0 = P_1$	…	$P_{n-2}^0 = P_{n-2}$	$P_{n-1}^0 = P_{n-1}$	$P_n^0 = P_n$
1	$P_0^1 = = P_0^0(1-t) + P_1^0 t = = P_0 B_1^0(t) + P_1 B_1^0(t)$	$P_1^1 = = P_1^0(1-t) + P_2^0 t = = P_1 B_1^0(t) + P_2 B_1^0(t)$	…	$P_{n-2}^1 = = P_{n-2}^0(1-t) + P_{n-1}^0 t = = P_{n-2} B_1^0(t) + P_{n-1} B_1^0(t)$	$P_{n-1}^1 = = P_{n-1}^0(1-t) + P_n^0 t = = P_{n-1} B_1^0(t) + P_n B_1^0(t)$	
2	$P_0^2 = = P_0^1(1-t) + P_1^1 t = = P_0 B_2^0(t) + P_1 B_2^1(t) + + P_2 B_2^2(t)$	$P_1^2 = = P_1^1(1-t) + P_2^1 t = = P_1 B_2^0(t) + P_2 B_2^1(t) + + P_3 B_2^2(t)$	…	$P_{n-2}^2 = = P_{n-2}^1(1-t) + P_{n-1}^1 t = = P_{n-2} B_2^0(t) + P_{n-1} B_2^1(t) + + P_n B_2^2(t)$		
…	…	…	…	…	…	…
$n-1$	$P_0^{n-1} = = P_0^{n-2}(1-t) + P_1^{n-2} t = = \sum_{i=0}^{n-1} B_{n-1}^i(t) P_i$	$P_1^{n-1} = = P_1^{n-2}(1-t) + P_2^{n-2} t = = \sum_{i=0}^{n-1} B_{n-1}^i(t) P_{i+1}$	…			
n	$P_0^n = = P_0^{n-1}(1-t) + P_1^{n-1} t = = \sum_{i=0}^{n} B_n^i(t) P_i = P_n(t)$		…			

Отримані після розрахунку точки утворюють два сегменти: $\{P_0^0, P_0^1, \ldots, P_0^n\}\{P_0^n, P_1^{n-1}, \ldots, P_n^0\}$. Якщо розглянути криву на рис. 1.2, то можна помітити, що вона складається з двох сегментів, але при цьому остання точка першого сегменту є першою точкою для наступного, що забезпечує гладкий перехід і візуальне відображення в якості єдиної кривої.

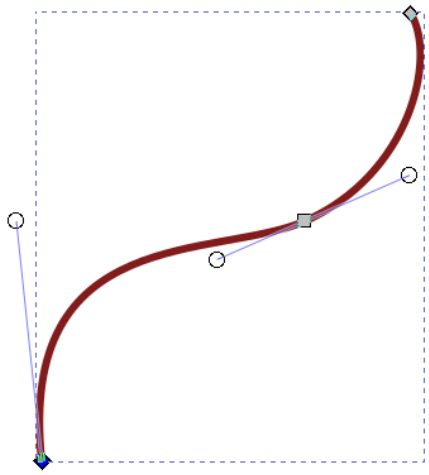


Рисунок 1.2 – Приклад кривої з декількома сегментами

Крива на рис. 1.2 наведена для того, аби продемонструвати її розрив на два сегменти, використовуючи можливості графічного редактора. Для такого розриву необхідно мати вузол, в якому виконується розподіл, який можна спостерігати на рис. 1.3.

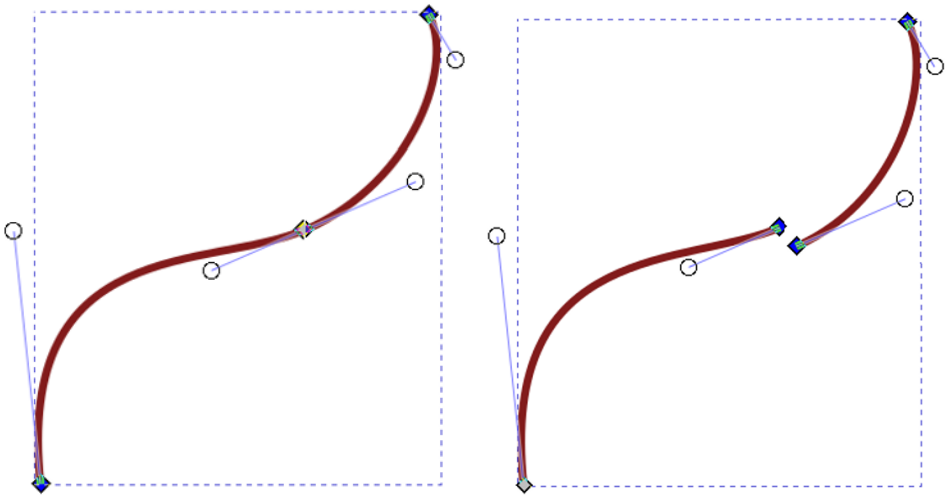


Рисунок 1.3 – Виконаний розрив над кривою з двома сегментами

Особливість наведено вище розподілу в тому, що крива до розриву і так мала неявні два сегменти, а для виконання стеганографічних перетворень вбудовування необхідно розподіляти криву у довільних точках. Для цього виконується створення штучного вузла, після чого з’являється можливість розривати будь-які сегменти:

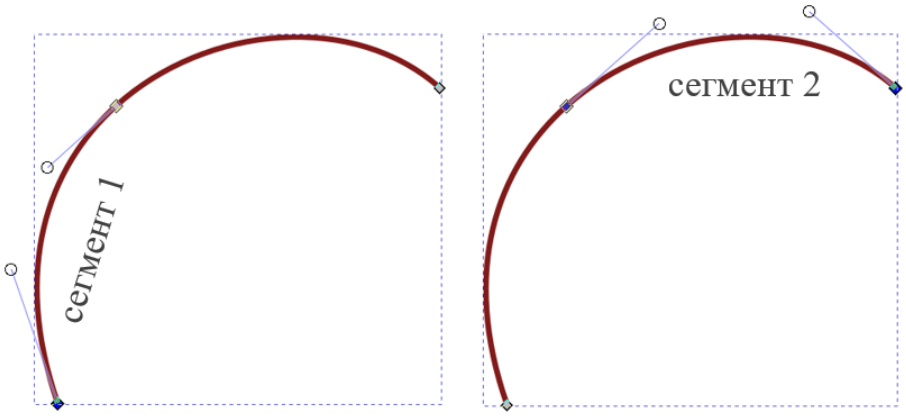


Рисунок 1.4 – Отримання двох сегментів з односегментної кривої

Саме така можливість розриву кривих без візуальної зміни їх відображення лежить в основі бітового та патернового методу, які відрізняються один від одного числом поділу кривих та кількістю інформації, що приховується.

Так, наприклад, маємо опорні точки $P_0, P_1, P_2, P_3 = (2,2),(3,6),(5,8),(8,7)$:

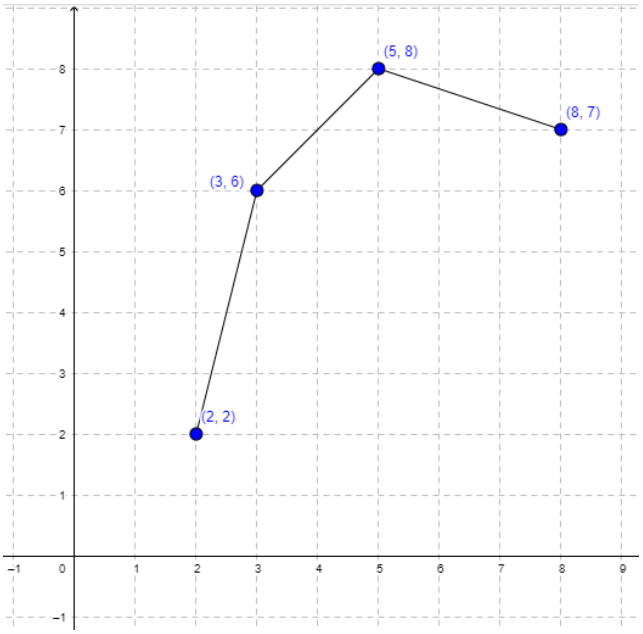


Рисунок 1.5 – Опорні точки простої кривої Без'є

Нехай криву з рис. 1.5 необхідно розбити при значенні $t = 0.3$. Тоді матриця де Кастельжо згідно з (1.2) (порядок кривої $n = 4$) виглядатиме таким чином:

P_0	P_1	P_2	P_3
P_0^1	P_0^2	P_0^3	
P_1^1	P_1^2		
P_2^1			

→

2,2	3,6	5,8	8,7
2.3,3.2	3.6,6.6	5.9,7.7	
2.69,4.22	4.29,6.93		
3.17,5.033			

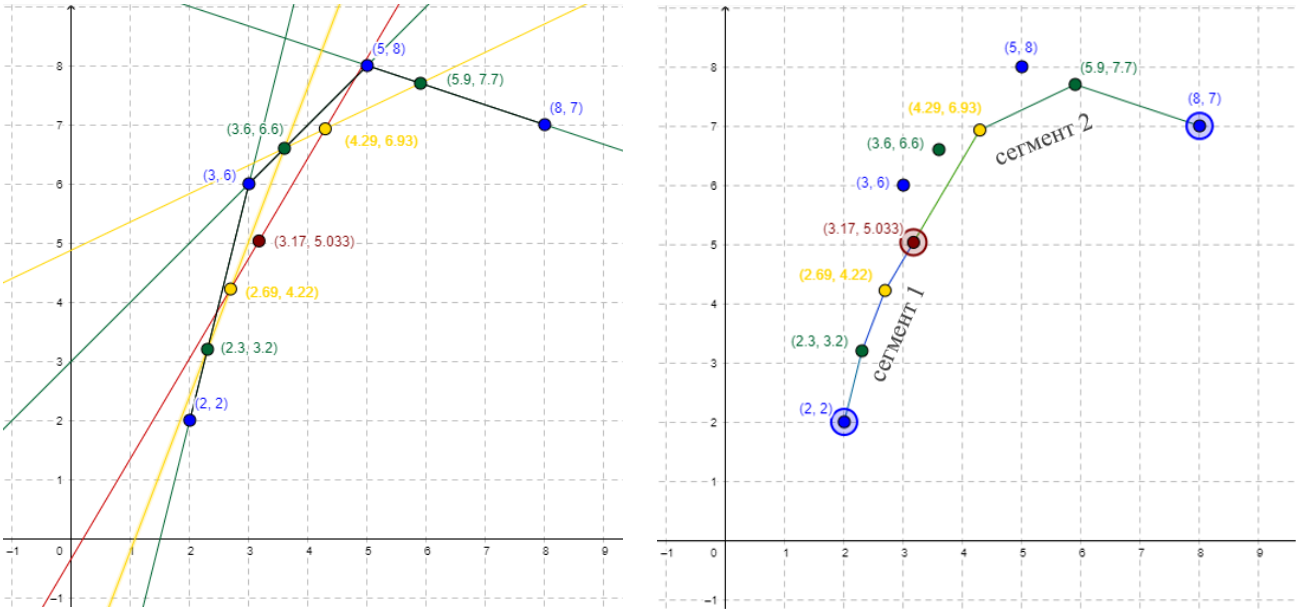
Рисунок 1.6 – Приклад таблиці де Кастельжо

Під час розрахунку матриці виконується операція упорядкування значень, отриманих при обрахунку проміжних точок на додаткових прямих. Якщо звернути увагу на рис. 1.7 (а), то можна відмітити, що новоутворені точки розташовуються на умовних прямих, і при цьому відкладаються від попередньої точки на відстань, рівну $t = 0.3$ або $t \cdot 100\% = 30\%$.

На рис. 1.7 (б) відображено, які саме точки використовуються для опису двох новоутворених сегментів:

$$\left\{P_0^0, P_0^1, \ldots, P_0^n\right\}\left\{P_0^n, P_1^{n-1}, \ldots, P_n^0\right\} =$$

$$= \left\{(2,2),(2.3,3.2),(2.69,4.22),(3.17,5.033)\right\}\left\{(3.17,5.033),(4.29,6.93),(5.9,7.7),(8,7)\right\}$$



а) б)

Рисунок 1.7 – Графічне відображення точок матриці де Кастельжо

Для того, аби об'єднати сегменти, необхідно відновити таблицю де Кастельжо. Так, маючи на вході два сегменти $\left\{P_0, \mathrm{P}_0^1, \mathrm{P}_1^1, \mathrm{P}_2^1\right\}$ $\left\{\mathrm{P}_2^1, \mathrm{P}_1^2, \mathrm{P}_0^3, P_3\right\}$ необхідно відновити точки P_1 та P_2 з початкового P_0, P_1, P_2, P_3:

$$P_2 = \frac{P_0^3 - tP_3}{1-t}$$
$$P_1 = \frac{P_{np} - tP_2}{1-t},\ P_{np} = \frac{P_1^2 - tP_0^3}{1-t} \qquad (1.3)$$

Повертаючись до прикладу на рис. 1.7, можна відновити початкові контрольні точки з рис. 1.5:

1) $P_2 = \frac{(5.9, 7.7) - 0.3(8,7)}{1-0.3} = (5,8)$

2) $P_{np} = \frac{(4.29, 6.93) - 0.3(5.9, 7.7)}{1-0.3} = (3.6, 6.6)$

3) $P_1 = \frac{(3.6, 6.6) - 0.3(5,8)}{1-0.3} = (3,6)$

Так, отримується початкова крива $P_0, P_1, P_2, P_3 = (2,2),(3,6),(5,8),(8,7)$.

Так само, як при приховуванні сегмент розривається в певних точках, що відповідають значенню параметра t, при вилученні секретних даних відбувається

відтворення початкового вигляду кривої шляхом об'єднання сегментів при узгоджених раніше точках. Для прийняття рішення про вилучення даних використовуються такі умови [18]:

$$\left|P_1^t - P_3^t\right| \le error, \left|P_1^1 - P_1^t\right| \le error, \left|P_1^1 - P_3^t\right| \le error, \quad (1.4)$$

де $error$ – визначене значення похибки.

При цьому для розрахунку точок з (1.4) так само використовується матриця де Кастельжо, а точніше ті точки, за допомогою яких було утворено два сегменти $\left\{P_0, \mathrm{P}_0^1, \mathrm{P}_1^1, \mathrm{P}_2^1\right\}\left\{\mathrm{P}_2^1, \mathrm{P}_1^2, \mathrm{P}_0^3, P_3\right\}$ з початкового P_0, P_1, P_2, P_3. Тобто, отримання точок P_1^t, P_2^t та P_3^t виглядає наступним чином:

$$P_1^t = \frac{\left(P_2^1 - tP_1^2\right)}{\left(1-t\right)}, P_2^t = \frac{\left(P_1^2 - tP_0^3\right)}{\left(1-t\right)}, P_3^t = \left(1-t\right)P_0^1 + tP_2^t \quad (1.5)$$

Якщо одна з умов (1.4) при бітовому відтворенні задовольняється, то вилучається двійкова «1», якщо ні – двійковий «0». При методі патернів шукається такий запис із таблиці патернів, при якому одна з умов (1.4) виконається, і тоді вилучається відповідний блок двійкових даних.

1.2 Афінні атаки на векторні зображення

Загроза застосування геометричних атак над стегоконтейнерами існує для стеганографічних методів, що використовують для приховання зображення як растрової, так і векторної графіки. Основну проблему для векторних стегоконтейнерів представляють афінні перетворення, які здатні призвести до повного руйнування прихованого повідомлення, адже впливають на координати точок. Модифікація точки виконується шляхом наступного перетворення [19] [20]:

$$\begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} ax+by+c \\ dx+ey+f \\ 1 \end{bmatrix}. \quad (1.6)$$

Вираз (1.6) представляє собою загальний випадок афінного перетворення, який при різних типах набуває наступного вигляду [21]:

- перетворення перенесення:

$$\begin{bmatrix} 1 & 0 & c \\ 0 & 1 & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x+c \\ y+f \\ 1 \end{bmatrix}; \quad (1.7)$$

- перетворення повороту:

$$\begin{bmatrix} \cos(\alpha) & -\sin(\alpha) & 0 \\ \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x\cos(\alpha) - y\sin(\alpha) \\ x\sin(\alpha) + y\cos(\alpha) \\ 1 \end{bmatrix}; \quad (1.8)$$

- перетворення X-зсуву:

$$\begin{bmatrix} 1 & b & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x+by \\ y \\ 1 \end{bmatrix}; \quad (1.9)$$

- перетворення Y-зсуву:

$$\begin{bmatrix} 1 & 0 & 0 \\ d & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ dx+y \\ 1 \end{bmatrix}; \quad (1.10)$$

- перетворення масштабування:

$$\begin{bmatrix} a & 0 & 0 \\ 0 & e & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} ax \\ ey \\ 1 \end{bmatrix}; \quad (1.11)$$

- майже афінне перетворення з додаванням шуму:

$$\begin{bmatrix} a+n_1 & b+n_2 & 0 \\ d+n_3 & e+n_4 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x+n_5 \\ y+n_6 \\ 1 \end{bmatrix} = \begin{bmatrix} (a+n_1)(x+n_5)+(b+n_2)(y+n_6) \\ (d+n_3)(x+n_5)+(e+n_4)(y+n_6) \\ 1 \end{bmatrix}. \quad (1.12)$$

Дослідження Кінзяревого [9-10] демонструють можливість застосування методів, здатних бути стійкими до афінних атак та математичне підґрунтя яких було розглянуто вище.

1.3 Дослідження можливих шляхів удосконалення та обґрунтування вибору подальших досліджень

Математичні засади використання бітового та патернового методів дозволяють реалізувати гнучкий механізм для проведення стеганографічних

перетворень. Застосування матриці де Кастельжо попри більші часові затрати виправдовується можливістю виконання чисельно стабільного розриву кривої, що забезпечує подальшу реалізацію операцій вилучення даних на основі виконання умови (1.4) між раніше отриманими точками штучно створених сегментів.

Можливість збереження повідомлення при атаках афінних перетворень є актуальним питанням, яке у рамках розглянутих методів тісно пов’язане з параметром точності проведення розрахунків при виконанні стеганографічних перетворень. Виконання досліджень щодо виявлення параметрів закодування, при яких стійкість до атак зберігається, по суті, направлене на пошук значення найбільш сприятливого для кодування мінімального значення точності.

Розглянута вище математична модель виконання стеганографічних операцій має на увазі збереження певної інформації в якості стегоключів, які дозволять отримувачеві вилучити секретне повідомлення. У даному контексті стегоключами виступають параметри закодування: точність виконання обрахунків; кількість біт, що була прихована в одній кривій; а також крок зміни параметру – для бітового методу, та початковий крок і довжина патерну – для методу патернів. Ключами вилучення також виступають фінальні значення кроку зміни параметра побудови для кожної закодованої кривої.

У роботі [17] розглянуто збереження таких параметрів в окремий файл формату JSON, структура якого дозволяє не тільки просто записати необхідну інформацію, але і легко програмний шляхом отримати необхідні для декодування значення. І хоча практика збереження ключів в окремі файли є досить поширеною, у рамках даного дослідження розглядається можливість приховування стегоключів у самому файлі зображення. Такі методи приховування ключів розглядаються і при застосуванні стеганографічних перетворень растрової графіки (наприклад, LSB методах) [22].

При дослідженні бітового методу можна відмітити, що збереження фінальних параметрів не є обов’язковим, адже при його застосуванні у криві вбудовуються послідовності однакової довжини. При цьому крок зміни параметру

побудови є константою, а звідси – останнє значення буде однаковим для усіх кривих зображення та може бути програмно обраховано, базуючись на переданих ключах закодування.

Сенс такого збереження зберігається у контексті запровадження можливості визначення того, в які сегменти було виконано вбудовування, аби уникати помилок, зумовлених програмною реалізацією алгоритму. Так, наприклад, якщо у переданому ключі-послідовності фінальних параметрів поточний елемент ітерації дорівнює 0, то дана крива стовідсотково не була модифікована. Тоді дану ітерацію можна пропустити й перейти до виконання наступної, не марнуючи час на додаткові перевірки або обробку помилок.

Необхідність такого збереження також пояснюється тим, що у програмному сенсі заповнений стегосегменти фактично не володіють особливими відзнаками. Наприклад, людське око буде здатне помітити велику кількість близьких за значенням координат точок, проте програма таким самим чином, як і пустий контейнер, виконає аналіз заповненого стегозображення із встановленням валідності сегментів та представленні у канонічному вигляді.

У випадку блокового методу, на відміну від бітового, необхідними є останні значення параметру зміни побудови для кожної кривої, що була закодована, при виконанні вилучення повідомлення. Крок зміни параметру при застосуванні патернового методу не є постійним, а змінюється у залежності від двійкового блоку, що потрапляє на вхід, на величину, що відповідає закріпленому за патерном кроком у таблиці патернів. Звідси – кожна стегокрива матиме різні фінальні параметри і без їх знання відтворення закодованого повідомлення стане неможливим.

Таким чином, генерація ключів вилучення на основі параметрів вбудовування є необхідним етапом процесу стегоперетворення, а збереження усіх значень фінальних параметрів для методу патернів є виправданим. Зважаючи на те, що безпека стеганосистеми в повній мірі визначається таємністю ключа [23], то при

реалізації розглянутих методів, необхідно досліджувати шляхи підвищення скритності його передачі та збереження.

У даній роботі для досягнення підвищення таємності ключа використовуються можливості експлуатації структури файлу зображення векторної графіки, вбудовуючи у його тіло елемент посилання, що ніяким чином не впливає на візуальне відображення заповненого контейнеру.

1.4 Висновки до першого розділу

Математичним підґрунтям методів, що досліджуються, є використання алгоритму де Кастельжо для розриву кривих Без’є третього ступеня. Двійкове повідомлення для приховання інтерпретується розподілом кривої на множину сегментів у контрольованих точках параметра побудови кривої. У бітовому вбудовуванні приховується один біт інформації за раз, а у патерновому – двійковий блок визначеної довжини.

Основною загрозою векторних стегоконтейнерів є афінні перетворення, які здатні призвести до руйнування прихованого повідомлення, шляхом зміни координат точок. Стійкість до різного роду афінних перетворень є важливою характеристикою стеганографічного методу над векторними зображеннями.

Для підвищення таємності стеганографічної системи необхідно підвищувати секретність стегоключа. Бітовий та патерновий методи вимагають ключі для вилучення даних, при цьому бітовому достатньо знати параметри закодування, а патерновому – параметри закодування та фінальні параметри закодування кожної кривої. У даній роботі досліджується та реалізовується можливість приховування стегоключа у тілі самого стегоконтейнера.

2 МОДЕЛЬ ТА МЕТОД ПРИХОВУВАННЯ ДАНИХ У ВЕКТОРНІ ЗОБРАЖЕННЯ

2.1 Розробка та дослідження критеріїв та показників ефективності стеганосистем

Область теперішньої стеганографії поширюються в основному на цифрові файли мультимедіа, що пояснюється широким застосуванням цифрових каналів передачі інформації. Для визначення того, чи є система стеганографічно ефективною, необхідно використовувати ряд критеріїв та показників. Автори роботи [24] визначають три основні вимоги, які замовник очікує від системи, що виконує стеганографічні перетворення:

Рисунок 1.1 – Трикутник вимог до цифрової стеганографії

Основна ідея представленої вище концепції в тому, що всі три вимоги не можуть бути імплементовані в одній системі і степінь досягнення кожної з них має вирішуватися замовником шляхом визначення пріоритетів. Проте у роботі [25] описується більш комплексний підхід критерій та показників.

Такі показники не є обов’язковими для повного покриття, але вони описуються можливі обмеження та характеристики, на які замовник має звернути особливу увагу у залежності від ПрО, що реалізовується. Так, у табл. 1.1 відображено вимоги, що висуваються до повідомлення для вбудовування.

Таблиця 1.1 – Вимоги до повідомлення

Критерій	Опис	Метрики	Приклад
Максимальний розмір повідомлення	Описується максимальний розмір повідомлення, який система може бути здатна вбудувати у контейнер. Може визначатися у бітах або інших релевантних ПрО метриках.	Біт	Система мусить вміти вбудовувати розміром 1024 біт.
Модифікація повідомлення	Описується можливість модифікації повідомлення у вже заповненому контейнері без використання оригінального контейнера.	Так / Ні	Система має володіти можливістю модифікації повідомлення без оригінального контейнера.
Множина повідомлень	Описується необхідність можливості вбудови декількох повідомлень у єдиний контейнер.	Так / Ні	Система має вміти вбудовувати декілька повідомлень в один контейнер.

У табл. 1.2 описуються можливі вимоги до контейнера. Говорячи про тип контейнера, необхідно визначати, що, по-перше, він має володіти деякою надмірністю, яка може бути експлуатована. По-друге, необхідно визначити природу файлу – чи його довжина відома (фіксована довжина), чи файл є потоком даних (фінальний розмір невідомий). Також важливим фактором є описання вмісту контейнеру, його ймовірний контент (зображення природні або технічні тощо).

Визначаючи формати файлі, що підтримуються системою, можливим є варіант більш детального уточнення джерела контейнеру, його специфікації (наприклад, файли діаграм .drawio, згенеровані конструктором DrawIO ПЗ Diagrams.net). Критерій типу вибору контейнера описує, чи запроваджує система контейнер для вбудовування повідомлення [22]. Якщо файл надано системі і повідомлення має бути в нього приховане, то тип є не вибірковим. Вибірковий тип

визначає система, здатну надати лист з переліком контейнерів, запроваджуючи можливість вибору найкращого файлу для модифікації. Побудований тип генерує контейнер на основі того, яке повідомлення буде приховане.

Таблиця 1.2 – Вимоги до контейнера

Критерій	Опис	Метрики	Приклад
Тип контейнера	Описуються можливі типи цифрових контейнерів.	Текст, Зображення, Звук тощо	Система мусить вміти вбудовувати повідомлення у зображення растрової графіки (пейзажні мотиви), які мають фіксовану довжину.
Формати файлів, що підтримуються	Описуються формати файлів, які система має підтримувати для виконання вбудовування та вилучення інформації.	Формати файлів та специфікації	Система має виконувати операції вбудовування та вилучення над BMP-файлами.
Тип вибору контейнера	Визначається, чи запроваджує система вибір специфічних контейнерів для вбудовування.	Не вибірковий/ Вибірковий / Побудований	Система запроваджує список контейнерів.

У табл. 1.3 визначаються вимоги до стегоконтейнера. Критерій непомітності має визначатися, адже якщо зміни є надто помітними, то стегоконтейнер може здатися підозрілим, що призведе до виявлення секретного повідомлення. Зміни проявляються у надто помітному шумі, наявності артефактів вбудовування, помилок або надто різких перепадах кольору тощо. Можливими метриками можуть бути пікове співвідношення сигналу до шуму PSNR або співвідношення сигнал/шум SNR.

Таблиця 1.3 – Вимоги до стегоконтейнера

Критерій	Опис	Метрики	Приклад
Непомітність	Описує зміни стегоконтейнера по відношенню до оригінального контейнера.	PSNR для відео та зображень, SNR для звуку тощо	PSNR стегоконтейнеру по відношенню до оригінального контейнеру не має перевищувати 30 дБ.
Якість	Описує різницю між стегоконтейнером та оригінальним контейнером, що сприймається людиною.	Суб'єктивна оцінка, з використанням визначеної методології	Стегоконтейнер може володіти деяким шумом, який не має бути надто сприйнятним людським оком.
Розмір стегоконтейнеру	Визначає мінімальний та максимальний розміри стегоконтейнеру.	Біти, відсоток збільшення	Розміри контейнеру та стегоконтейнеру мають належати діапазону [1МБ, 5МБ].
Ефективність вбудовування	Визначає ймовірність вилучення вбудованих даних із стегоконтейнера.	Ймовірність [0,1]	Ефективність вбудування має бути 0.9 чи більше.
Помилковість виявлення повідомлення	Визначає ймовірність визначення того, що пустий контейнер має повідомлення, а заповнений контейнер – ні.	Відсоток / Ймовірність	Помилковість виявлення повідомлення має бути меншою, ніж 10^{-7}.
Тип вилучення	Визначає, як повідомлення має вилучатися.	Сліпе / Інформоване	Система має вилучати повідомлення без оригінального контейнера.

Якість є у деякій мірі суб’єктивною оцінкою, значення якої може бути визначено деякою методикою, наприклад, рекомендацією ITU-R BT.500-13 [26]. Суб’єктивність оцінки полягає у тому, якою мірою сприйняття володіє переглядач цифрового файлу – наскільки відчутною буде зміна зображення після виконання JPEG-стиснення.

Розмір стегоконтейнеру визначається для встановлення ліміту його збільшення, адже надто велика вага також викликає підозри щодо його вмісту, а також ускладнює процес передачі. Ефективність вбудовування описує можливість вилучення повідомлення, так як не всі методи запроваджують гарантоване правильне вбудовування і подальше вилучення повідомлення, яке можна було б зрозуміти.

Помилковість виявлення повідомлення описує ймовірність того, що контейнер може бути помилково визначений таким, що має вбудовані дані, а стегоконтейнер – таким, що не має. Тип вилучення описує можливі варіанти декодування стегоконтейнера: сліпим чином, тобто без наявності оригінального контейнера, чи інформованим, що припускає наявність і пустого, і заповненого контейнерів.

Безпека будь-якого стеганографічного алгоритму найчастіше представляє найбільший інтерес, тому до неї можуть висуватися такі вимоги:

Таблиця 1.4 – Вимоги до безпеки

Критерій	Опис	Метрики	Приклад
Стійкість до стегоаналізу	Описує можливість виявлення того, що контейнер містить секретні дані.	Значення, отримані за визначеною методикою	Стеганографічний алгоритм не має демонструвати статичні зміни контейнера.

Продовження таблиці 1.4 – Вимоги до безпеки

Надійність	Описує можливість протистояння трансформування стегоконтейнера.	Визначаються замовником системи	Вбудоване у зображення повідомлення не повинно руйнуватися при стандартному JPEG-стисненні.
Стійкість до атак	Описує можливість протистояння до різних атак.	Залежать від визначених атак для протистояння	Алгоритм має бути стійким до атак проти вбудованого повідомлення чи водяного знаку.
Тип ключів	Визначає типи ключів у системі.	Без ключів / Приватні / Публічні / Приватні та публічні	Система має використовувати приватні стеганографічні ключі. Повідомлення може бути закодоване, використовуючи публічні криптографічні ключі.

Стійкість до стегоаналізу може бути оцінена за допомогою опису факту існування алгоритму стегоаналізу, здатного зламати стеганографічний алгоритм, що використовується; статичних змін стегоконтейнеру; інформації, що може бути отримана шляхом використання некоректних ключів декодування [27]. Надійність характеризується можливістю протистояння таким трансформаціям стегоконтейнера як стиснення, зміна формату тощо, що можуть бути спричинені як зловмисником, так і каналом передачі.

Критерій стійкості до атак має описувати, при застосуванні яких форм зловмисної модифікації алгоритм має дотримуватися визначеного значення

цілісності секретних даних. Тип ключів описує існування ключів, їх модифікатори доступу, збереження тощо.

У табл. 1.5 відображено вимоги, що висуваються до стеганографічного алгоритму. Критерій місткості у значній мірі залежить від алгоритму, що застосовується. Область вбудовування визначає, яка сама надмірність експлуатується у цільовому контейнері.

Таблиця 1.5 – Вимоги до алгоритму

Критерій	Опис	Метрики	Приклад
Місткість	Описує кількість біт, що можуть бути вбудовані в одиницю даних.	Біт на одиницю даних (МБ, піксель, тощо)	Система має вбудовувати мінімум 1 біт в 1 піксель.
Швидкість	Описує швидкість виконання стеганографічних операцій.	Нотація Ландау, секунди тощо	Система має вбудовувати повідомлення максимального розміру у зображення 200×200 пікселів протягом 2 секунд.
Область вбудовування	Описує область зображення, яка експлуатується для приховання повідомлення.	Назва області (просторова область, часова область тощо)	Система вбудовує інформацію у просторову область контейнера.

Таким чином, за розглянутою вище методикою висунення критеріїв до стегосистеми, виконується визначення показників, яких має досягати система для проведення стеганографічних перетворень у зображення векторної графіки. У першу чергу необхідно визначити вимоги до повідомлення:

Таблиця 1.6 – Вимоги до повідомлення системи, що розглядається

Критерій	Значення
Максимальний розмір повідомлення	$M_{\max\_len}$
Модифікація повідомлення	Ні
Множина повідомлень	Ні

Максимальний розмір секретних даних можна обрахувати наступним чином:

$$M_{\max\_len} = bits \cdot I_{len}, \tag{2.1}$$

де $bits$ – кількість біт, що вбудовується в одну криву;

I_{len} – кількість доступних для вбудовування кривих.

Тобто, при встановленому значенні біт у 40 біт/крива та зображенні з 5 кривими можна виконати приховання 200 бітного повідомлення. Модифікація уже вбудованого повідомлення не передбачається, так само як і можливість вбудовування одразу декількох повідомлень.

Вибір контейнера має ряд наступних критеріїв:

Таблиця 1.7 – Вимоги до контейнера системи, що розглядається

Критерій	Значення
Тип контейнера	Векторне зображення фіксованої довжини з нетехнічними мотивами
Формати файлів, що підтримуються	Формат файлу SVG, специфікація W3C
Тип вибору контейнера	Не вибірковий та Вибірковий

Під нетехнічними мотивами маються на увазі зображення, які не є результатом автоматизованого створення, результатом конвертації до певного формату тощо. Детальніше особливості контейнера розглядаються далі. Система працює виключно із зображеннями формату .svg, забезпечує вибір із переліку зображень, а також не обмежує завантаження довільного контейнера (при цьому відповідальність за обрання контейнера, що підходить для вбудовування, лежить на користувачеві).

Показники, якими має володіти заповнений контейнер виглядає наступним чином:

Таблиця 1.8 – Вимоги до стегоконтейнера системи, що розглядається

Критерій	Значення
Непомітність	Коефіцієнт візуального спотворення за ПЗ AntiDupl.NET-2.3.9 не має перевищувати 0.1
Якість	Зображення не має суттєво відрізнятися від оригінального
Розмір стегоконтейнеру	Розмір не має збільшуватися більше, ніж на 40%
Ефективність вбудовування	0.95 та більше
Помилковість виявлення повідомлення	Не має перевищувати 10^{-6}
Тип вилучення	Сліпе

Помилковість вбудовування обмежується значенням, прийнятим для більшості систем відповідно до [28].

У табл. 1.9 відображено показники безпеки системи, а у табл. 1.10 – критерії алгоритму.

Таблиця 1.9 – Вимоги до безпеки системи, що розглядається

Критерій	Значення
Стійкість до стегоаналізу	Алгоритм не має демонструвати статичні характеристики стегоконтейнера.
Надійність	Передача файлу через мережу, стиснення зображення, геометричні трансформації графічними редакторами не мають руйнувати повідомлення.
Стійкість до атак	Стійкість до афінних перетворень
Тип ключів	Приватні

Таблиця 1.10 – Вимоги до алгоритму системи, що розглядається

Критерій	Значення
Місткість	Від 8 біт/крива до 80 біт/крива
Швидкість	Залежить від кількості кривих I_{len}, довжини повідомлення M_{len} та значення $bits$
Область вбудовування	Об’єкти <path>, що містять криві Без’є

2.2 Особливості контейнерів векторної графіки

У сьогоднішньому мультимедійному середовищі представлено широкий вибір форматів векторної графіки: .eps, .ai, .svg, .fcm, .pdf, .cdx тощо [29]. Деякі з них є специфічними файлами, що генеруються виключно для роботи в оригінальному графічному редакторі, деякі володіють сумісністю з іншими середовищами.

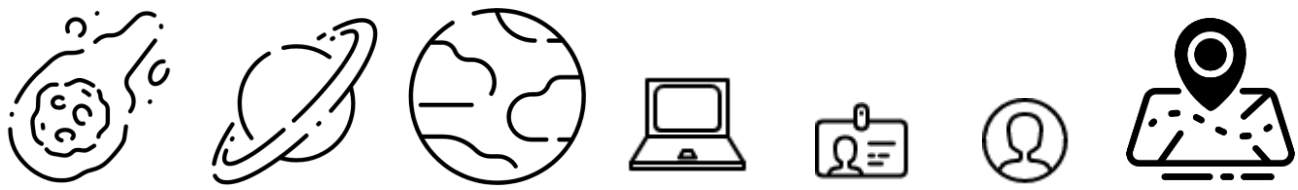
Одним із найпопулярніших форматів є масштабована векторна графіка SVG, що представляє собою мову для опису двовимірної графіки [30]. SVG-файли можуть бути відкриті за допомогою великого переліку редакторів, включаючи Adobe Illustrator, Corel PaintShop Pro (Windows), Inkscape (мультиплатформний), та GIMP (мультиплатформний). Такі популярні браузери як Google Chrome, Microsoft Edge, Mozilla Firefox та Apple Safari також підтримують SVG-формат [31]. По суті, .svg файли представляють собою текстовий формат на основі XML, тому їх можна створювати та модифікувати й у текстових або XML редакторах. У даній роботі для демонстрації фрагментів зображень використовується графічний редактор Inkscape.

SVG використовує мову розмітки XML, що запроваджує можливість його використання як окремого самостійного формату, так і змішування з іншими елементами XML та HTML. Він дозволяє використовувати три типи графічних об’єктів: векторні графічні фігури, вбудовані зображення та текст.

Застосування складних перетворень застосунками до SVG можливе завдяки існуванню Document Object Model (DOM), до якого можна звернутися програмним шляхом з метою отримання повного доступу до усіх елементів, атрибутів та властивостей. Графічний елемент SVG визначається як один із типів елементів, що може спричинити відображення візуального контенту на цільовому полотні, зокрема: «audio», «canvas», «circle», «ellipse», «foreignObject», «iframe», «image», «line», «path», «polygon», «polyline», «rect», «text», «textPath», «tspan» та «video» [32].

У даній роботі особлива увага надається елементу path, концепція якого лежить у використанні поняття поточної точки. Проводячи аналогію з реальним життям, можна сказати, що поточна точка – це положення грифелю олівця на папері, яке можна змінити, відображуючи бажаний контур або прямими лініями, або кривими. Контури представляють геометрію обрису об'єкта, визначену в термінах moveto (встановити нову поточну точку), lineto (намалювати пряму лінію), curveto (намалювати криву за допомогою кубічної команди Без’є), arc (еліптична або кругова дуга) і closepath (закрити поточну форму шляхом з'єднання з останньою moveto) [33]. Крім того, можливі ситуації створення складних path, які складаються з внутрішніх subpath.

Розглянемо векторні зображення представлені на рис. 2.2, які були взяті зі звичайного безкоштовного сервісу для пошуку іконок при веб-дизайні або веб-розробці. Такі зображення є ідеальними претендентами для вбудовування секретної інформації, адже не викликають підозр – іконки з рис. 2.2 (г-е) є типовими для веб-застосунків онлайн-магазинів, наприклад, а з рис. 2.2 (а-в) можуть бути так само застосовані для більш тематично направлених сайтів.



а) б) в) г) г) д) е)

Рисунок 2.2 – Приклади SVG-зображень у вигляді іконок

Складністю використання приведених вище зображень полягає у тому, що модерні SVG-іконки не мають достатньої кількості необхідних для виконання стеганографічного перетворення кривих, які до того ж можуть чергуватися з переходами до інших графічних елементів, наприклад до L/l (лінія від поточної точки до заданої), H/h (горизонтальна лінія до заданої точки), V/v (вертикальна лінія до заданої точки), A/a (еліптична крива) тощо. Таким чином, стандартне зображення може мати досить малий розмір, а з невеликої кількості елементів, що в ньому містяться, у граничному випадку може знаходитися лише одна крива.

Комплексність шляху іконок пояснюється тим, що завдяки комбінаціям з іншими елементами забезпечується малий розмір зображення і можливість представлення декількох візуальних елементів в одному шляху. Наприклад, рис. 2.2 (г) виглядає наступним чином:

```
<svg width="46px" height="46px" viewBox="0 -4.5 46 46"
xmlns="http://www.w3.org/2000/svg">
    <path id="_31.ID-Horizontal" data-name="31.ID-Horizontal"
          d="M43,42H5a4,4,0,0,1-4-4V17a4,4,0,0,1,4-
4H20V9a4,4,0,0,1,8,0v4H43a4,4,0,0,1,4,4V38A4,4,0,0,1,43,42ZM17.014,34.488v0S17,3
3.711,17,33.711a11.22,11.22,0,0,0,2.031-1.378,3.029,3.029,0,0,0,.741-
1.362,33.627,33.627,0,0,0,.238-3.7c0-2.062-1.033-4.28-4.007-4.28v0c-2.974,0-
4.007,2.219-
4.007,4.28a33.874,33.874,0,0,0,.238,3.7,3.029,3.029,0,0,0,.741,1.362A11.219,11.2
19,0,0,0,15,33.71s-.014.781-.018.781v0s.029,1.146.029,1.487a2.186,2.186,0,0,1-
2.223,2.018h0c-2.593.113-3.2.976-
3.21.984A4.877,4.877,0,0,0,9,40H23a4.657,4.657,0,0,0-.582-1.022c0-.009-.617-
.871-3.21-.984h0a2.186,2.186,0,0,1-2.223-
2.018C16.984,35.634,17.014,34.488,17.014,34.488ZM26,9a2,2,0,0,0-
4,0v6a2,2,0,0,0,4,0Zm19,8a2,2,0,0,0-2-2H28a4,4,0,0,1-8,0H5a2,2,0,0,0-
2,2V38a2,2,0,0,0,2,2H6.8a10.841,10.841,0,0,1,1.16-2.24c.025-.014.848-1.739,5-
1.79.006-.021.01-1.042.022-1.027a7.286,7.286,0,0,1-1.051-.816,4.187,4.187,0,0,1-
1.452-2.583,39.458,39.458,0,0,1-.488-4.166c0-3.171,1.265-6.381,5.953-
6.381h.121c4.688,0,5.953,3.21,5.953,6.381a39.458,39.458,0,0,1-
.488,4.166,4.187,4.187,0,0,1-1.452,2.583,7.111,7.111,0,0,1-1.051.816c.013-
.015.017,1.007.022,1.027,4.151.051,4.974,1.776,5,1.79A10.841,10.841,0,0,1,25.2,4
0H43a2,2,0,0,0,2-2Zm-5,9H28a1,1,0,0,1,0-
2H40a1,1,0,0,1,0,2ZM28,30h2a1,1,0,0,1,0,2H28a1,1,0,0,1,0-
2Zm0,4h6a1,1,0,0,1,0,2H28a1,1,0,0,1,0-2Zm4-3a1,1,0,0,1,1-
1h4a1,1,0,0,1,0,2H33A1,1,0,0,1,32,31ZM23,9h2v2H23Z"
          transform="translate(-1 -5)" fill-rule="evenodd"/>
</svg>
```

Рисунок 2.3 – Внутрішній контент SVG-зображення

Вирішенням описаного обмеження з вибору стегоконтейнера є підвищення комплексності обробника векторних зображень для того, аби мати змогу інтерпретувати більшу кількість випадків поєднань різних графічних елементів.

Також можливе перетворення вручну зображення-іконки у такий вигляд, аби можливим було вбудовування інформації. Так, на рис. 2.4 відображено створений за рис. 2.2 (ґ) контейнер, що містить (за можливістю) чисті криві Без'є.

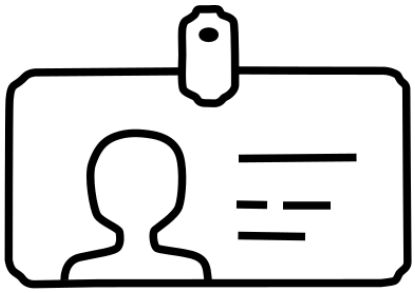


Рисунок 2.4 – Перетворений контейнер з більшою кількістю кривих Без'є

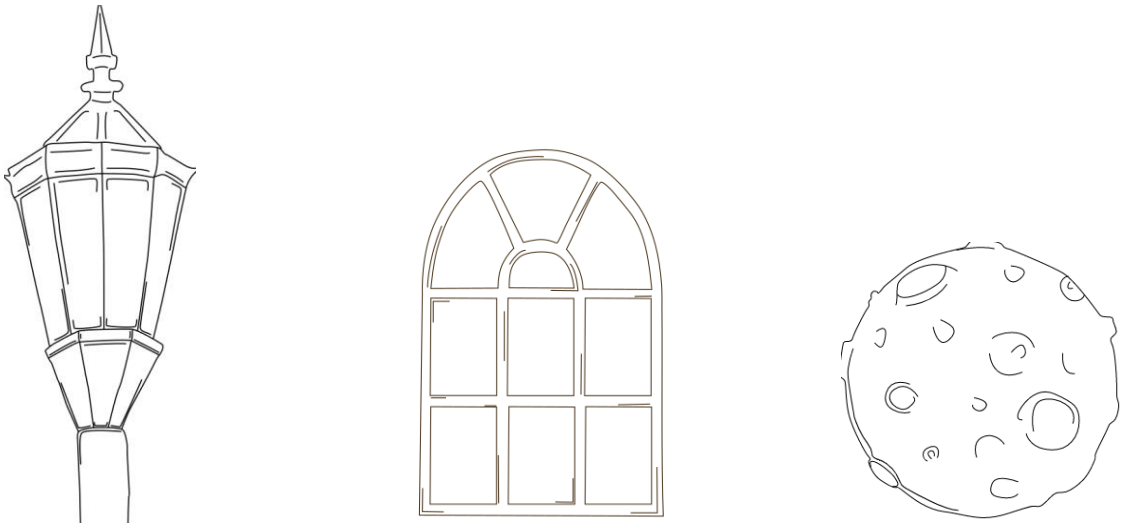
Вміст такого перетвореного зображення виглядає наступним чином:

```
<svg
   width="210mm" height="297mm" viewBox="0 0 210 297" version="1.1"id="svg28611"
  <g><path style="opacity:1;stroke:#000000;stroke-width:0.665;stroke-opacity:1"
       d="m 88.225827,179.38415 c 0.0017,-0.0799 0.03596,-0.70631 0.365273,-
0.86313 0.329319,-0.15682 0.561452,-0.35232 0.648499,-0.42452 0.173617,-0.144
0.136702,-0.50008 0.428954,-0.50008 H 90.4986 c 0,0 0.549204,-0.0384 0.706769,0
0.2114,0.0515 0.219553,0.36068 0.345,0.45477 0.125455,0.0941 0.851574,0.19849
0.992708,0.6219 0.141137,0.42341 0.130881,5.12685 0.12608,5.32515 -
0.0093,0.38607 -0.474883,0.83111 -0.81086,0.97624 -0.286079,0.12463 -
0.216364,0.31806 -0.647547,0.48883 -0.21527,0.0853 -0.432703,0.0248 -
1.013773,0.0266 -0.591315,0.002 -0.658977,-0.28378 -0.771099,-0.43813 -
0.136676,-0.18815 -0.849732,-0.34686 -1.084899,-0.68194 -0.235495,-0.33554 -
0.09336,-2.67641 -0.09947,-2.94706 -0.0084,-0.37193 -0.02678,-1.51192 -0.02678,-
1.51192 z"/>
    <path style="opacity:1;stroke:#000000;stroke-width:0.665;stroke-opacity:1"
       d="m 87.974918,182.81846 -12.074988,0.0941 c 0,0 -0.572157,0.0276 -
0.885794,0.37257 -0.313635,0.34499 -0.586721,0.51547 -0.837631,0.67227 -
0.250907,0.15682 -0.377938,0.46062 -0.440666,1.24471 -0.06273,0.78409 -
0.07354,13.45213 -0.07354,13.45213 0,0 0.07762,0.74294 0.476814,1.03125
0.279103,0.20157 0.517974,0.33083 0.654235,0.53226 0.255042,0.37702
0.643145,0.499 1.02016,0.499 0.377015,0 5.256048,-0.0887 5.256048,-0.0887 l
23.847784,-0.0111 c 0,0 0.9528,-0.10472 1.0995,-0.17212 0.2901,-0.1333 0.3199,-
0.47956 0.4783,-0.67432 0.19168,-0.23568 0.67102,-0.43111 0.65079,-0.93306 -
0.0209,-0.51708 -0.10194,-14.56056 -0.10194,-14.56056 0,0 0.0356,-0.2857 -
0.31364,-0.53318 -0.28766,-0.20386 -0.3404,-0.57643 -0.48613,-0.65864 -0.13727,-
0.0774 -0.61383,-0.33026 -1.2411,-0.31458 -0.62727,0.0157 -12.05708,-0.0305 -
12.05708,-0.0305"/>
    <path … />
    <path … />
    <path … />
    <path style="opacity:1;stroke:#000000;stroke-width:0.665;stroke-opacity:1"
       d="m 96.693547,194.09677 h 3.991933"/>
    <path style="opacity:1;stroke:#000000;stroke-width:0.665;stroke-opacity:1"
       d="m 92.923385,196.625 5.633067,0.0887"/></g>
</svg>
```

Рисунок 2.5 – Внутрішній контент перетвореного SVG-зображення (фрагменти)

З рисунку вище видно, що загалом концепція іконки зберіглась, але разом з цим спостерігається збільшення кількості елементів, яким воно описується. При

виконанні програмної імплементації використовувалися зображення, створені за природніми джерелами, схожі на скетчі, що можуть передаватися між відправником та отримувачем не заради зберігання у веб-застосунках, а для простого, на перший погляд, обміну. Створення таких зображень також виправдовується тим, що такі контейнери здатні зберігати значно більшу кількість інформації. Прикладами зображень, що застосовуються у даній роботі є наступні:



а) б) в)

Рисунок 2.6 – Приклади нетехнічних SVG-зображень

2.2 Моделювання приховування даних у векторні зображення

Модель виконання операцій приховування даних у контейнер виглядає наступним чином:

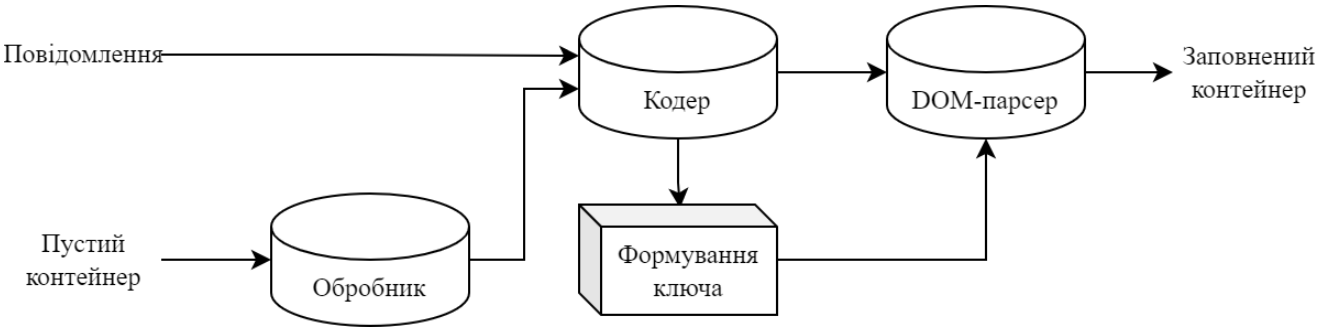


Рисунок 2.7 – Модель приховування даних

Як видно з рис. 2.7, на вхід подаються повідомлення та контейнер-зображення. Під кодером розуміється модуль, відповідний за виконання стеганографічних операцій вбудовування даних у контейнер. Попередньо оброблений в обробнику контейнер та повідомлення передаються у кодер, який

вбудовує дані у зображення та формує ключі. Сформовані ключі разом з модифікованим зображенням передаються в DOM-парсер, що відповідає за представлення заповненого контейнеру у належному вигляді.

Процес приховування даних найкраще відобразити у вигляді UML-діаграми послідовності:

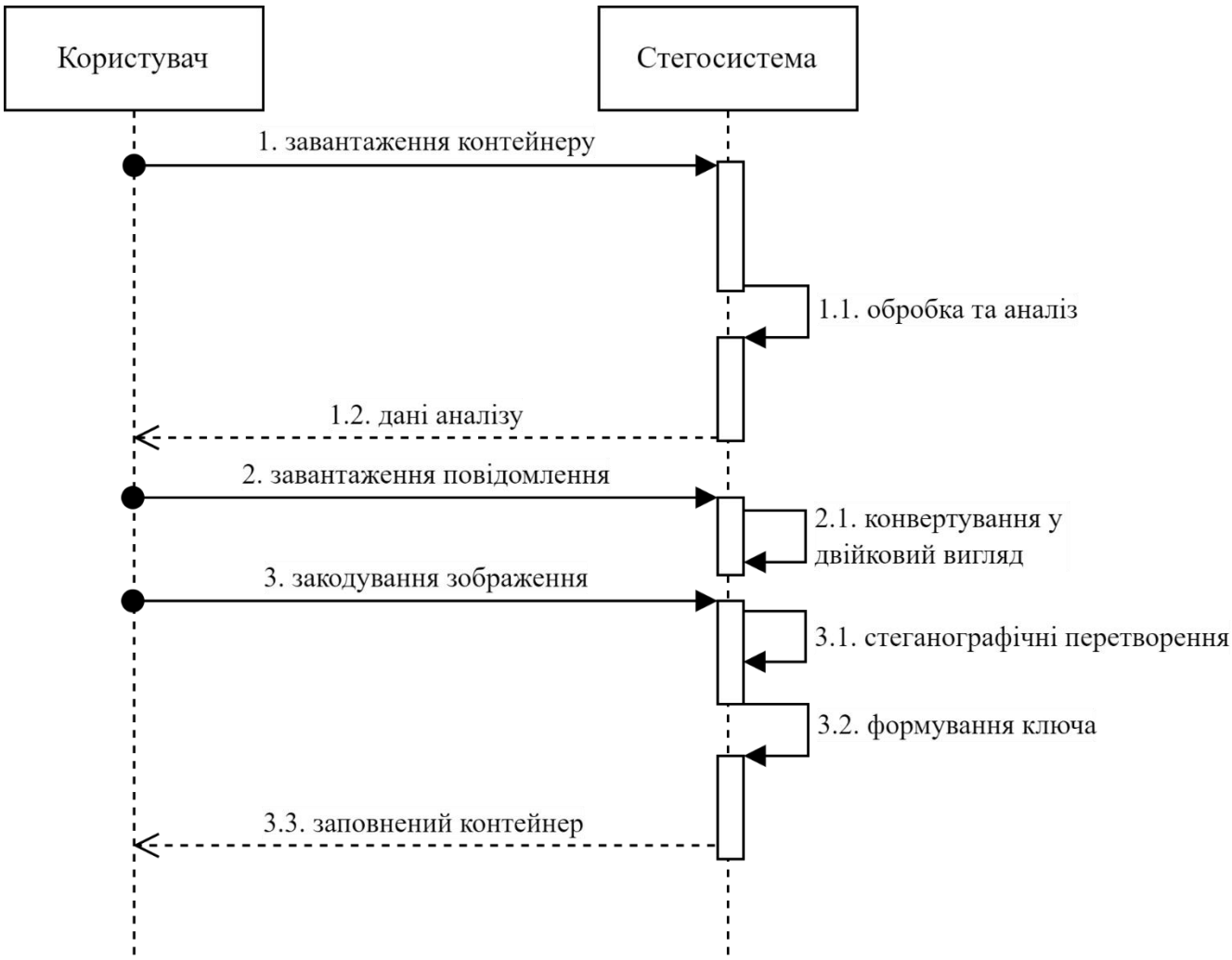


Рисунок 2.8 – UML-діаграма послідовності вбудовування даних

При цьому під стеганографічними перетвореннями на кроці 3.1 передбачається виконання алгоритму вбудовування двійкових послідовностей у контейнер векторної графіки у бітовому чи патерновому режимі. Для бітового режиму псевдокод виконання операції bitEncode виглядає наступним чином:

```
List<EncodedCurve> encodedCurves = new ArrayList<>();
for (Curve curve : this.image.getCurves()) {
        List<List<Segment>> encodedSegments = new ArrayList<>();
        for (Segment segment : curve.getSegments()) {
                if (сегмент валідний) {
                        int indexOfSegment = 0;
                        String[] currBlock = input.get(indexOfBlock).split("");
                        double k = this.parameter;
                        for (int i = 0; i < currBlock.length; i++) {
                                if (Objects.equals(currBlock[i], "1")) {
                                        розірвати сегмент згідно з (1.2) та табл. 1.1 за
                                        параметром k та додати отримані сегменти до
                                        encodedSegments;
                                        indexOfSegment++;
                                }
                                k += this.parameter;
                        }
                        indexOfBlock++;
                }
        }
        encodedCurves.add(new EncodedCurve(curve.getLiteral(), curve.getStartPoint(),
encodedSegments);
}
makeStegokey(parameters);
```

Рисунок 2.9 – Псевдокод операції bitEncode

Для формування ключа виконується процес складання рядка параметрів наступним чином:

```
private String parseParametersToString(){
    return "k" + this.parameter + ":b" + this.maxBitAmount + ":p" + this.precision + "/";
}
```

Рисунок 2.10 – Функція формування рядка бітових параметрів

Отримана послідовність ключа виглядає наступним чином:

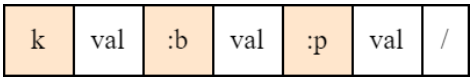


Рисунок 2.11 – Схематичне представлення бітового стегоключа

Для патернового режиму псевдокод виконання операції patternEncode виглядає наступним чином:

```
Сформувати таблицю патернів paternTable;
int indexOfBlock = 0;
List<EncodedCurve> encodedCurves = new ArrayList<>();
List<Double> finalParameters = new ArrayList<>();
for (Curve curve : this.image.getCurves())) {
        List<List<Segment>> encodedSegments = new ArrayList<>();
        for (Segment segment : curve.getSegments()) {
                List<String> inputValue = trim(input.get(indexOfBlock), patternLength);
                int indexOfSegment = 0;
                double k = startStep;
                for (int i = 0; i < inputValue.size(); i++) {
                        currentRecord = patternTable.find(inputValue.get(i));
                        розірвати сегмент згідно з (1.2) та табл. 1.1 за параметром (k +
                        currentRecord.step) та додати отримані сегменти до
                        encodedSegments;
                        indexOfSegment++;
                        k += currentRecord.step;
                }
                indexOfBlock++;
                finalParameters.add(k);
        }
        encodedCurves.add(new EncodedCurve(curve.getLiteral(), curve.getStartPoint(),
encodedSegments);
}
makeStegokey(parameters, finalParameters);
```

Рисунок 2.12 – Псевдокод операції patternEncode

У даному випадку для формування стегоключа makeStegokey(parameters, finalParameters) необхідно зберігати також значення останнього кроку вбудовування, тому виконується процес, як представлено на рис. 2.10 для кожного отриманого фінального параметру.

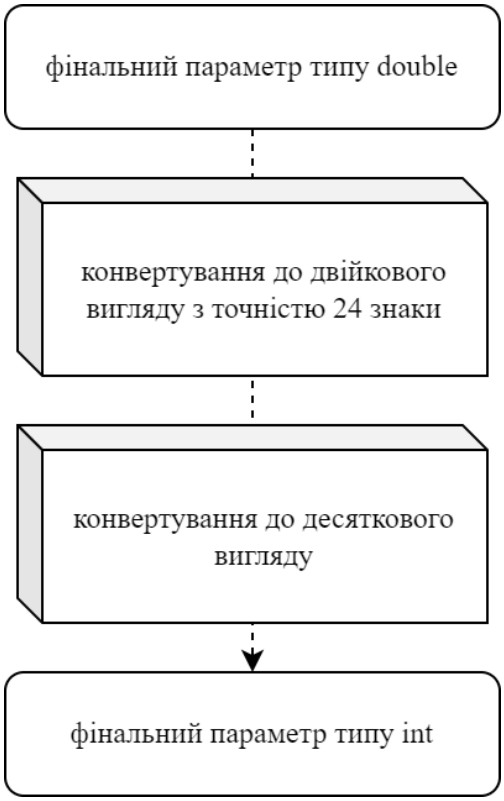


Рисунок 2.13 – Перетворення фінального параметра

Далі формується рядок параметрів вбудовування за допомогою parseParametersToString():

```
private String parseParametersToString(){
    return "l" + this.patternLength + ":s" + this.startStep + ":b" + this.maxBitAmount
    + ":p" + this.precision + "/";
}
```

Рисунок 2.14 – Функція формування рядка патернових параметрів

Утворений рядок конкатенується з перетвореними у цілі числа фінальними параметрами для генерації такої послідовності:

l	val	:s	val	:b	val	:p	val	/	v_0	int_0	$:int_1$	...	$:int_m$	v_1	int_0	$:int_1$	...	$:int_m$	...	v_n

Рисунок 2.15 – Схематичне представлення патернового стегоключа

На рис. 2.15 $n = I_{len}$ (кількість кривих для вбудовування), а $m = \frac{bits}{pattern_{len}}$, де $pattern_{len}$ – довжина патерну.

Послідовності стегоключей в обох випадках додаються у створюваний елемент посилання, якому можна додати деякий ідентифікатор, аби зробити його схожим на запис, споріднений зображенню. У даному випадку встановлюється id= "w3orgSnaps", посилаючись на специфікацію контейнера, в який вбудовується інформація. Основною ідеєю ідентифікатора посилання та елементів послідовностей «v» (рис. 2.11 та рис. 2.15) є концепція імітування версій зображення або редуктивних даних, залишених, наприклад, графічним редактором.

Звичайна людина у загальному випадку при передачі зображення не заглядатиме у його XML-вміст, а якщо ж таке відбулося – посилання у вигляді збережених значень версій не викличе підозр. Такий підхід до збереження ключа може бути модифікований у майбутньому, наприклад, шляхом додаткового інтерпретування всього рядка, його зашифрування тощо.

2.3 Моделювання вилучення даних з векторних зображень

Модель виконання операцій вилучення даних зі стегоконтейнера виглядає наступним чином:

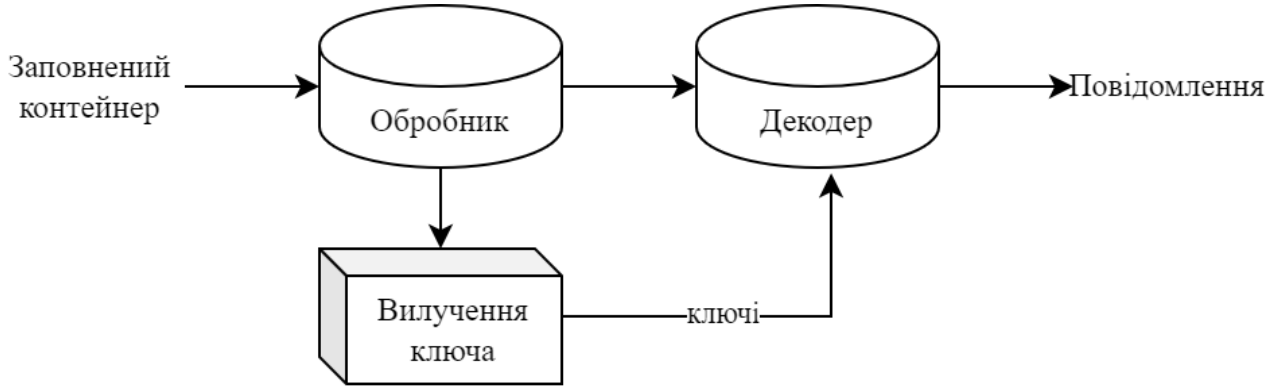


Рисунок 2.16 – Модель вилучення даних

Як видно з рис. 2.16, на вхід подається заповнений контейнер, який відразу обробляється обробником, під час чого відбувається вилучення стегоключа. Декодер виконує операцію вилучення секретний даних над обробленим

контейнером завдяки вилученим ключам закодування. На виході отримується текстове повідомлення.

Процес вилучення даних можна відобразити у вигляді наступної UML-діаграми послідовності:

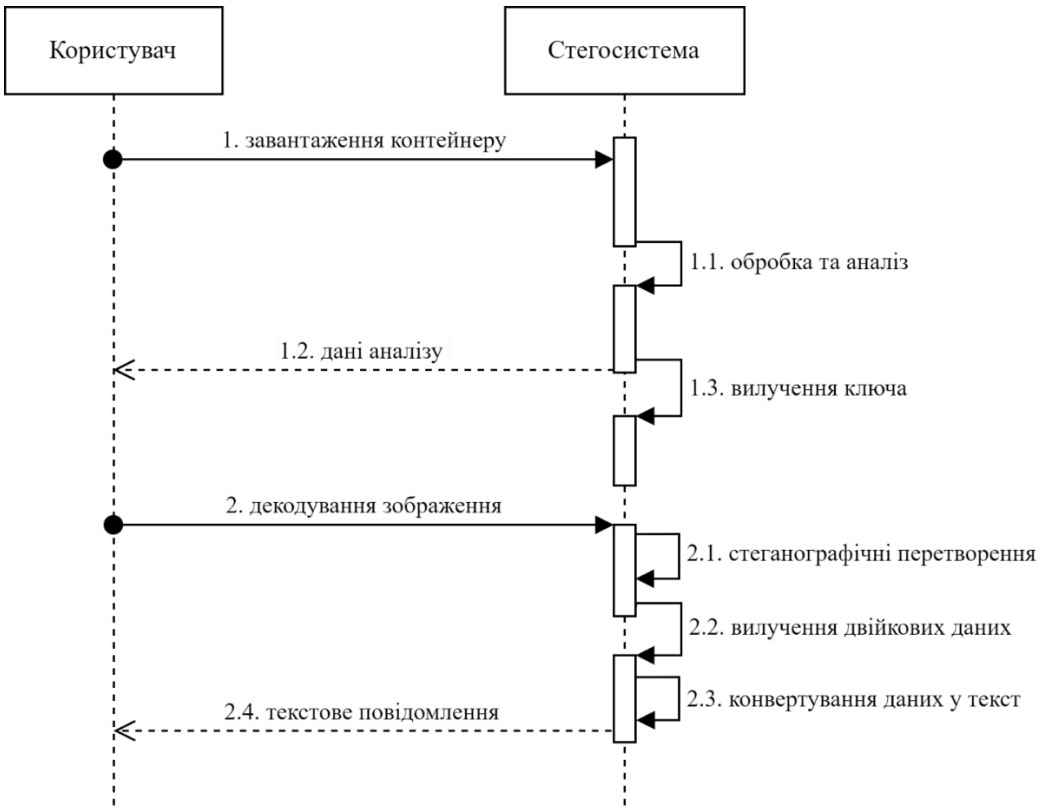


Рисунок 2.17 – UML-діаграма послідовності вилучення даних

Під стеганографічним перетворенням на кроці 2.1 розуміється виконання операцій вилучення даних у відповідності з обраним режимом. Перед виконанням вилучення повідомлення необхідно розпарсити ключ кодування, який на етапі обробки зображення був отриманий із стегоконтейнеру. Контент вилученого бітового ключа представляє собою рядок даних, який інтерпретується таким чином, як відображено на рис. 2.18.

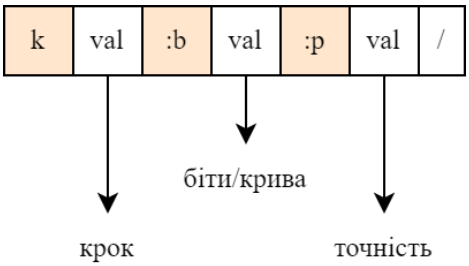


Рисунок 2.18 – Інтерпретація контенту бітового стегоключа

Бітовий режим відрізняється відсутністю необхідності зберігання фінальний параметрів, на відміну від патернового, адже його крок зміни параметра є сталою величиною. Через цей факт і існує необхідність передачі параметрів закодування, адже завдяки їм можливим стає відновлення бітового фінального параметру шляхом обрахунку наступного виразу:

$$k_{fin} = (bits + 1) \cdot k, \tag{2.2}$$

де k – значення сталого кроку, отриманого зі стегоключа.

Тепер починається декодування, яке для бітового режиму представляється наступним псевдокодом операції bitDecode:

```
List<String> decodedBinaryBlocks = new ArrayList<>();
for (int i = 0; i < this.image.getCurves().size(); i++) {
  for (Curve curve : this.image.getCurves().get(i)) {
        if (curve.getSegments().size() > 1) {
          StringBuilder output = new StringBuilder();
          обраховується lastEncodedValue згідно з (2.2);
          for (double t = lastEncodedValue - parameter; t >= 0; t -= parameter) {
            int indexOfSegment = curve.getSegments().size() - 1;
            обрахунок точок згідно з (1.5);
            if (умова (1.4) виконується){
              output.append("1");
              виконується об'єднання останніх двох сегментів згідно з (1.3);
            } else {
              output.append("0");
            }
          }
        output.reverse();
        decodedBinaryBlocks.add(output.toString());
        }
  }
}
```

Рисунок 2.19 – Псевдокод операції bitDecode

Після завершення декодування вилученні двійкові дані decodedBinaryBlocks конвертуються в текстове представлення і, по суті, отримується первинний вигляд зображення, адже всі розбиті криві при вилученні даних об’єднуються назад.

Отримавши на вхід стегоконтейнер, що був закодований патернами, спочатку виконується розпарсування стегоключа з вигляду рядка до параметрів закодування:

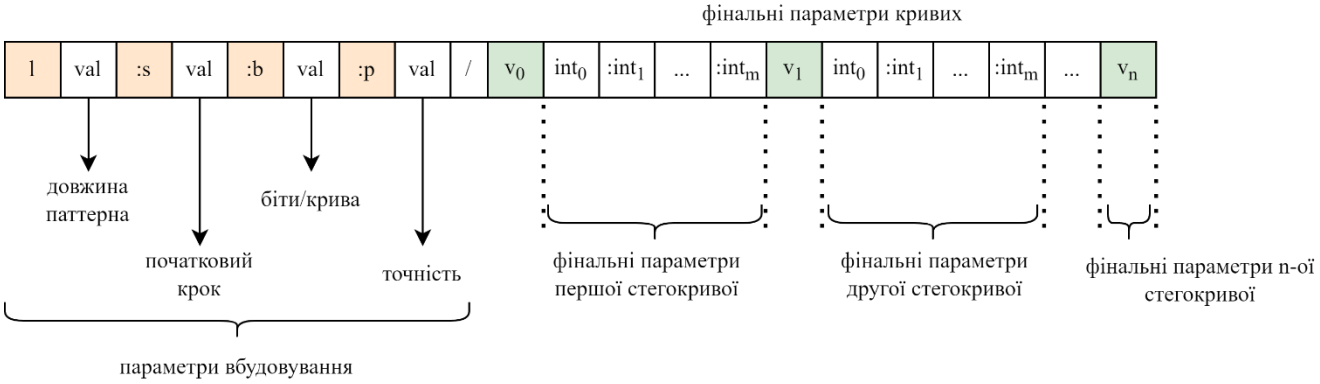


Рисунок 2.20 – Інтерпретація контенту патернового стегоключа

Після цього починається декодування, яке для патернового режиму представляється наступним псевдокодом операції patternDecode (рис. 2.21-2.22):

```
for (int i = 0; i < this.image.getCurves().size(); i++) {
 List<Curve> listOfCurves = this.image.getCurves().get(i);
 for (int j = 0; j < listOfCurves.size(); j++) {
  if (listOfCurves.get(j).getSegments().size() > bits/pattern_len ) {
      StringBuilder output = new StringBuilder();
      double t = finalParameters.get(i).get(j), parameter = 0;
      boolean flag = true;
      int indexOfSegment = listOfCurves.get(j).getSegments().size() - 1;
      do {
        if (indexOfSegment == 1) {
            for (int k = 0; k < patternTable.size(); k++) {
              if (parameter - patternTable.get(k).step == startStep){
                output.insert(0, patternTable.get(k).pattern);
                flag = false;
              }
            }
```

Рисунок 2.21 – Псевдокод операції patternDecode

```
            } else {
                if (indexOfSegment == listOfCurves.get(j).getSegments().size() - 1) {
                  об'єднати останні два сегменти згідно з (1.3);
                  indexOfSegment--;
                }
                for (int k = 0; k < patternTable.size(); k++) {
                 parameter = t - patternTable.get(k).step;
                 обраховуються точки згідно з (1.5) при значенні parameter;
                  if (умова (1.4) виконується){
                    output.insert(0, patternTable.get(k).pattern);
                    виконується об'єднання останніх двох сегментів згідно з (1.3);
                    t = parameter;
                   indexOfSegment--;
                   break;
                  }
                }
              }
            } while (flag);
            decodedBinaryBlocks.add(output.toString());
          }
        }
      }
```

Рисунок 2.22 – Псевдокод продовження операції patternDecode

Таким чином відбувається вилучення даних, що були вбудовані у зображення блоками.

2.4 Висновки до другого розділу

Будь-яка стеганосистема має володіти набором показників та критеріїв, за якими її можна оцінити та до яких має прагнути розробник. Для системи, що реалізовується, висуваються вимоги до повідомлення, контейнера, стегоконтейнера, безпеки та алгоритму для формалізації ПрО.

Стегоносистема працює з файлами векторної графіки SVG, як найбільш перспективного у сенсі застосування та програмної взаємодії формату. Складність застосування технічних файлів-іконок векторної графіки пов'язано з програмним

джерелом їх генерування, що призводить до малої наявності кривих Без’є у чистому вигляді. Це означає, що криві можуть існувати у складі інших геометричних елементів (або навпаки), що у рази ускладнює процес обробки контейнера.

Рекомендованими контейнерами для приховування секретних повідомлень є ті, що містять велику кількість однорідних кривих. Такі контейнери можуть мати нетехнічну спрямованість, природа яких не припускає використання прямих ліній та гострих кутів.

Під час моделювання приховування даних у векторні зображення було виявлено такі основні елементи: Обробник, Кодер та DOM-парсер. У кодері виконуються стеганографічні операції вбудовування повідомлення в оброблений Обробником контейнер та формування ключів, які передаються в DOM-парсер, що генерує стегоконтейнер.

Під час моделювання вилучення даних з векторних зображень було виявлено такі основні елементи: Обробник та Декодер. Обробник парсує заповнений контейнер та вилучає з нього стегоключ. Декодер виконує операцію вилучення секретний даних над обробленим контейнером завдяки вилученим ключам закодування.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ЕКСПЕРИМЕНТАЛЬНІ ДОСЛІДЖЕННЯ ПЛАТФОРМИ ДЛЯ ВИКОНАННЯ СТЕГАНОГРАФІЧНИХ ПЕРЕТВОРЕНЬ НАД ОБ'ЄКТАМИ ВЕКТОРНОЇ ГРАФІКИ

3.1 Обґрунтування вибору мови програмування, додаткових бібліотек та інструментів, середовища розробки

У даній роботі для проектування платформи для виконання стеганографічних перетворень було обрано серверну об'єктно орієнтовану мову програмування Java. З кінця 1995 року і по сьогоднішній день Java є абсолютним лідером в якості універсальної зв'язувальної ланки, яка може надати користувачу інформацію з будь-якого джерела (наприклад, з веб-сервер або баз даних). За всесвітнім індексом PYPL у 2022 році Java займає друге місце після Python [34].

PYPL індекс формується автоматично шляхом аналізу частоти пошукових запитів Google щодо певної мови програмування, отримуючи сирі дані з Google Trends. Такий індекс в сьогоденному інформаційному середовищі можна вважати найбільш ваговим, адже чим частіше виконується пошук вирішення задач стосовно деякої мови, тим більш актуальною вона вважається у сенсі попиту від розробників.

Так, у всесвітніх рамках популярність Python зросла за останні 5 років на 9.2%, при цьому Java втратила 5.0% [35]. Однак, саме стабільність застосування мови Java скрізь роки виступає гарантом її ще довгої присутності на ринку розробки програмного забезпечення. Роки плідного використання Java для розробки широкої різноманітності платформ та застосунків призвели до того, що майже на кожному ПК є встановлена версія мови Java, яка є безкоштовною та легкою у встановленні [36]. У рамках об'єктно-орієнтованості мови Java перед розробником ставиться задача створення об'єктів, які охоплюватимуть функції та дані для них, які і будуть складати структуру комплексної системи.

Зараз прикладний програмний інтерфейс API покриває багато різноманітних програмних областей, включаючи конструювання користувацьких інтерфейсів,

управління базами даних, інтернаціоналізацію, безпеку та обробку даних у форматі XML [37]. Останній факт є особливо привабливим для ПрО, що розглядається у даній роботі, адже для виконання стеганографічних перетворень обрано зображення векторної графіки формату SVG, який по суті являє собою структуру геометричних елементів, описаних у форматі XML.

Можливість створення графічних користувацьких інтерфейсів у Java також розвивалася крізь роки. Так, першою бібліотекою для розробки ГКІ була AWT, яка досить швидко була витіснена своїм нащадком Swing у силу великої кількості обмежень. Swing же у свій час запропонував більш досконалий підхід до конструювання ГКІ, однак його цільовими користувачами були корпоративні платформи без особливих вимог й до зовнішньої привабливості інтерфейсу [38].

Новим поколінням платформи та бібліотекою для конструювання ГКІ клієнтських застосунків Java стала JavaFX [39]. Вона є ефективним, раціональним, зручним каркасом для створення модерних візуально привабливих ГКІ, яка є перспективною і потужною заміною Swing. У даній роботі окрім самої бібліотеки JavaFX використовується інструмент візуального макету Scene Builder, розроблений Gluon [40].

Даний інструмент дозволяє проектувати ГКІ JavaFX застосунку, виключаючи необхідність створення макету програмним шляхом – користувачі можуть застосувати для свого інтерфейсу UI компоненти із вичерпного списку, що пропонується платформою, модифікувати їх властивості, додавати стилі CSS. Структура макету буде автоматично у реальному часі формуватися без безпосередньої участі користувача, а усі подальші зміни макету також будуть оновлювати файл з розміткою. Так, кінцевий файл формату FXML може комбінуватися з проєктом Java шляхом зв’язування UI компонентів з логікою застосунку.

Вибір середовища розробки виконувався з міркувань особистих уподобань, а також загальної зручності використання під час розробки комплексного проєкту. У даній роботі вибір прийшовся на IDE IntelliJ IDEA від Jet Brains зі студентською

ліцензією. З таблиці IDE індексів для різних СР, яка була сформована шляхом аналізування даних з Google Trends [41].

Кожен аспект обраного СР направлений на досягнення максимальної продуктивності розробників, включаючи індексування коду, інструменти для рефакторингу, ассестування для підключення об'єктів з інших мов програмування або модулів, запропонування покращення деяких блоків коду (наприклад, заміна на null-безпечне порівняння змінних), ергономічність інтерфейсу, вбудований маркет для встановлення плагінів та інше [42]. Внутрішня система контролю версій не дозволяє в аварійних ситуаціях втратити вже створений проєкт. Ергономічність інтерфейсу та широка підтримка файлів для перегляду та модифікації забезпечує зручний паралельний доступ до SVG-файлів, а гнучкість управління плагінами дозволяє викликати Scene Builder для створення та модифікації макету безпосередньо із дерева проєкту.

3.2 Розробка інтерфейсу та структури стеганографічної платформи

Для реалізації заданої платформи для проведення стеганографічних перетворень було застосовано значну кількість модулів, які відображено на рис. 3.1.

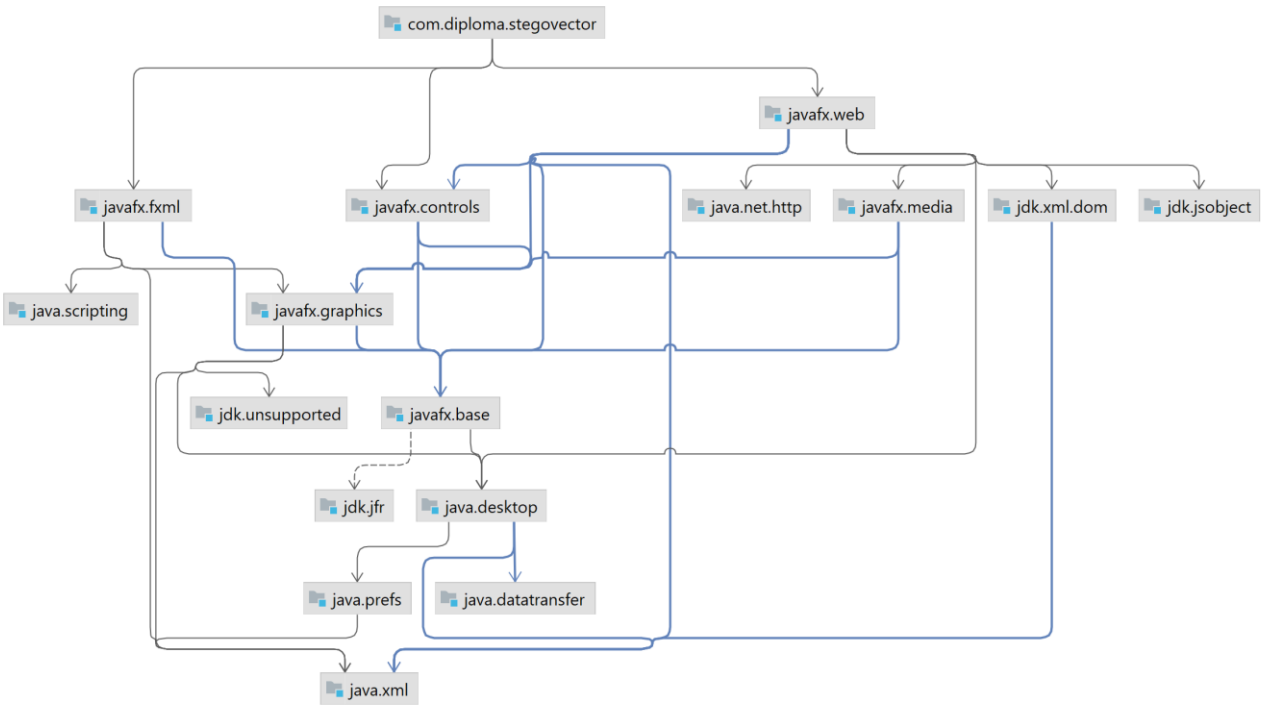


Рисунок 3.1 – UML-діаграма Java модулів

З рис. 3.1 видно, що при імплементації використовується функціонал таких модулів як [43]:

- javafx.web – визначає API для функціоналу WebView, впроваджуючи можливості для завантаження та відображення веб-контенту;
- javafx.fxml – визначає API для файлів макетів формату FXML та містить класи для завантаження ієрархії об’єктів з розмітки;
- javafx.controls – визначає UI контролери, присутні у наборі інструментів JavaFx UI;
- javafx.base – визначає основу API для набору інструментів JavaFX UI, включаючи API для зв’язування, властивостей, колекцій та подій;
- java.xml – визначає Java API для обробки XML (JAXP), потокового API для XML (StAX), та W3C Document Object Model (DOM) API.

Необхідно звернути увагу на перший зазначений у вище наведеному переліку модуль для відображення веб-контенту: даний модуль є невід’ємною частиною структури проєкту, адже саме він дозволяє виконувати відображення SVG-зображення. Така векторна графіка була створена для того, аби зображувати деякі графічні елементи у веб-застосунках, які містять вбудовану можливість зчитування та інтерпретування елементів XML-розмітки.

У даній роботі така властивість веб-платформи була експлуатована для того, аби мати змогу відобразити контент завантаженого зображення, шляхом використання UI елементу WebView та спорідненого не візуального об’єкту WebEngine [44]. WebEngine виконує всю роботу з завантаження, створення моделей документів, присвоєння стилів тощо [45].

Очевидно, що модуль XML призначений для виконання функцій отримання доступу до внутрішнього контенту зображення та його подальшого парсингу у необхідні структури даних. Base-модуль призначений для реалізації взаємодії з інтерфейсом користувача (доступ забезпечує java.desktop) під час обміну даними між застосунком та системою, що проваджується java.datatransfer.

Отриману структуру платформи для виконання стеганографічних перетворень над зображеннями векторної графіки можна представити у вигляді наступної діаграми класів:

Рисунок 3.2 – UML-діаграма класів проєкту

У наведеній вище діаграмі класів відображено власне назви класів та їхні властивості. Єдиний інтерфейс проєкту представляє собою загальний вигляд стеганографічного перетворення, яке буде застосоване до об’єкта зображення. З

рис. 3.3 видно, що інтерфейс передбачає обов'язкову реалізацію функцій кодування та декодування. Клас повідомлення служить для представлення повідомлення у звичайному текстовому форматі та у двійковому вигляді (рис. 3.4).

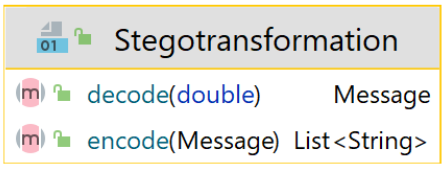


Рисунок 3.3 – Інтерфейс стегоперетворення

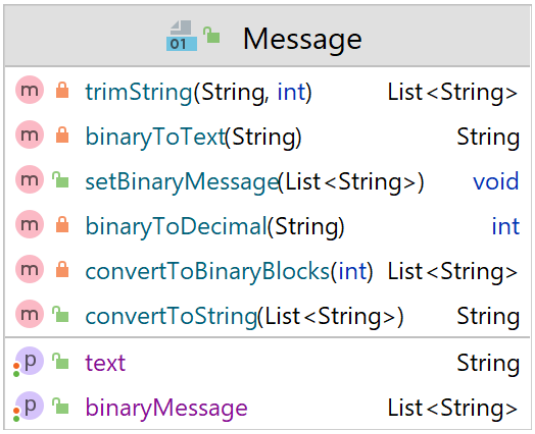


Рисунок 3.4 – Об'єкт повідомлення

Інтерфейс стегоперетворення імплементують два класи, призначення яких полягає у виконанні кодування та декодування зображення бітовим та патерновим методами. Метод патернів містить у собі два приватних класи (рис. 3.5): PatternTableRecord служить для представлення одного запису таблиці (індекс, крок зміни параметру та відповідний патерн), а PatternTable містить список об'єктів PatternTableRecord, та реалізує методи пошуку елементів у такому списку.

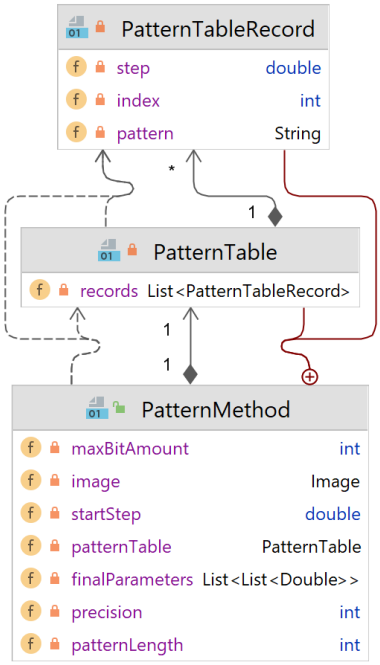


Рисунок 3.5 – Внутрішні класи об'єкту Pattern Method

Класи точки (Point), сегменту (Segment) та кривої (Curve) створюють єдиний об’єкт зображення (Image), який дозволяє проводити приховування та вилучення даних у векторних зображеннях. Змодельована структура може здатися надто громіздкою, але у рамках даної роботи таке представлення приймається найбільш оптимальним та таким, що дозволяє врахувати можливий контент зображення, над яким буде проводитися перетворення, а також зробити методи обробки XML-вмісту універсальними для пустого та заповненого контейнерів.

Клас закодованої кривої (EncodedCurve) є допоміжним класом, який використовується на етапі кодування зображення (рис. 3.6). Його основна мета – представити закодовані елементи об’єкту зображення у вигляді, готовому для прямого парсування у XML-файл заповненого контейнеру.

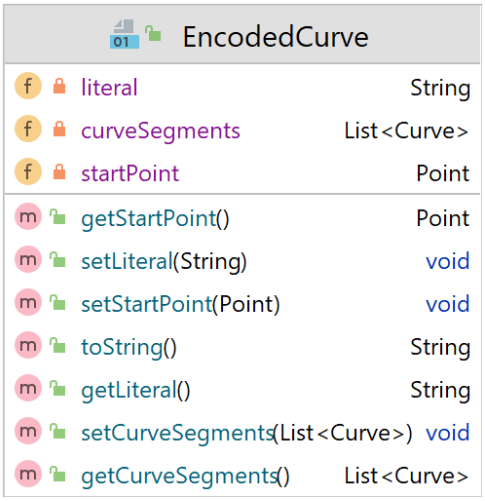


Рисунок 3.6 – Допоміжний клас закодованої кривої

Класи параметричного набору (Поточний параметрsSet), афінного перетворення (AffineTransformation) та експерименту (Experiment) реалізують можливості для проведення експериментальних досліджень щодо швидкості вбудовування та вилучення, а також відсотків втрачених біт під час повторюваного застосування атаки афінного перетворення до одного й того самого зображення при різних комбінаціях параметрів.

Класи контролеру застосунку (ApplicationController) та контролеру експериментів (ExperimentsContoller) виконують функції, що стосуються обробки подій UI елементів розмітки (кнопок завантаження зображення, кодування та

декодування тощо). Дані класи виконують роль зв'язувальної ланки між макетом FXML та Java класами. Клас застосунку (Application) є головним класом, з якого виконується старт проєкту.

На рис. 3.7 відображено головне вікно програми, яка впроваджує функціонал з приховування та вилучення даних з SVG-файлу.

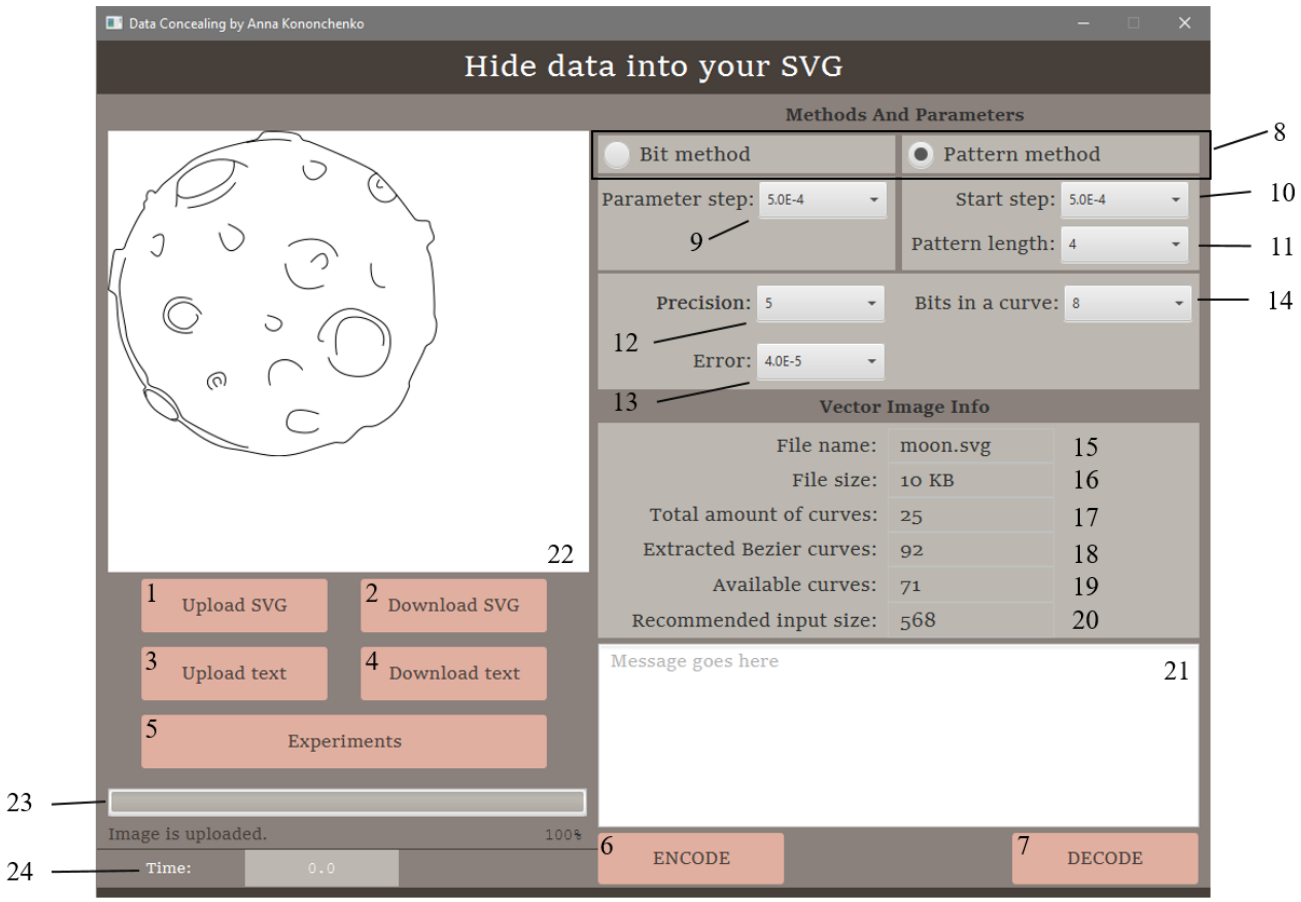


Рисунок 3.7 – Головне вікно програми та його елементи

У головному вікні програми присутні такі елементи:

1) – кнопка для завантаження зображення у форматі SVG у застосунок;

2) – кнопка для завантаження закодованого зображення у форматі SVG у локальне сховище;

3) – кнопка для завантаження зображення текстового файлу у застосунок;

4) – кнопка для завантаження зображення текстового файлу у локальне сховище;

5) – кнопка для переходу у дочірнє вікно для проведення експериментів;

6), 7) – кнопки для запуску процесу кодування та декодування зображення відповідно;

8) – панель з радіо-кнопками для обрання бажаного методу кодування та декодування;

9) – поле з випадним меню для вибору значення зміни параметру стеганографічного перетворення для побітового методу;

10) – поле з випадним меню для вибору значення початкового кроку виконання стеганографічного перетворення для методу патернів;

11) – поле з випадним меню для вибору довжини патерну при виконанні стеганографічного перетворення для методу патернів;

12) – поле з випадним меню для вибору значення точності виконання стеганографічного перетворення;

13) – поле з випадним меню для вибору значення максимальної кількості біт на одну криву при виконанні стеганографічного перетворення;

14) – поле з випадним меню для вибору значення похибки при виконанні вилучення інформації;

15), 16) – ім’я завантаженого файлу та його розмір;

17) – значення кількості кривих після проведення аналізу зображення;

18), 19) – значення кількості вилучених кривих із загальної кількості та число тих кривих, в які допускається вбудовування даних;

20) – рекомендована довжина повідомлення для вбудовування, отримана на основі кількості валідних кривих та максимальної кількості біт на одну криву;

21) – текстове поле для ручного вводу інформації при кодуванні та виводу повідомлення при декодуванні;

22) – полотно WebView для відображення завантаженого зображення;

23) – індикатор прогресу виконання операція завантаження з відображення статусу процесу;

24) – панель для відображення часу виконання кодування та декодування у мілісекундах.

При натисненні на кнопку експериментів (рис. 3.7, 5) виконується створення дочірнього вікна програми для проведення експериментальних досліджень. Перша вкладка такого вікна виглядає наступним чином:

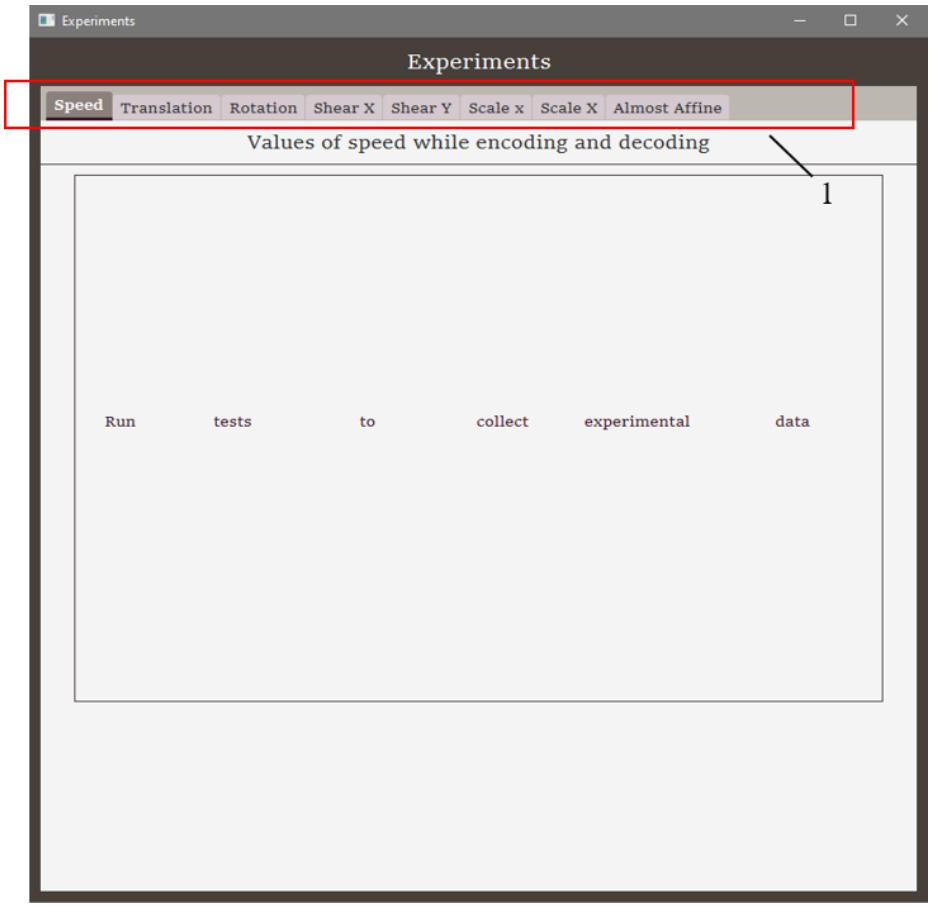


Рисунок 3.8 – Дочірнє вікно програми для проведення експериментів з вкладкою для вимірювання показників швидкості

З рис. 3.8 видно, що дочірнє вікно складається з восьми вкладок (елемент 1), які представлять собою різні типи експериментів. Перший тип є вимірюванням швидкості виконання операцій, значення яких оновлюються кожного разу, коли запускається інший експеримент з вимірювання відсотку втрачених біт при афінних перетвореннях. У даній реалізації передбачено 7 типів афінних перетворень: перенесення, поворот, зсув за віссю X, зсув за віссю У, пропорційне масштабування для зменшення, пропорційне масштабування для збільшення та майже афінне перетворення.

На рис. 3.9 відображено довільну вкладку з функціоналом для виконання афінного перетворення.

Рисунок 3.9 – Дочірнє вікно програми для проведення експериментів з вкладкою для вимірювання показників втрати біт

З рис. 3.9 можна помітити, що елемент під номером 1 (панель з радіо-кнопками) надає можливість змінювати методи при проведенні перетворень, 2 елемент запускає процес проведення експерименту, після завершення якого у вікні з'являються результати у табличній формі.

3.3 Розробка програмної моделі обробки контейнера

Обробка контейнера (тобто SVG-зображення) є окремим етапом процесу виконання задач, поставлених у даній роботі. Парсинг зображення з вигляду звичайного рядка символів у осмислену структуру даних є задачею нетривіальною, адже звичайне зображення векторної графіки може містити у собі не тільки криві Без'є (які є предметом виконання стеганографічних перетворень), а й лінії, кола, еліпси, полігони, прямокутники тощо.

Основну складність представляють собою елементи, які містять у собі різноманітні елементи, наприклад, у складі кривої Без’є може знаходитися перехід до команди створення прямої лінії до заданої точки. Дослідження шляхів інтерпретування таких випадків побудування шляху знаходять поза даною роботою, але у випадку успішної обробки й таких зображень методи, що розглядаються, матимуть більші показники кількості інформації, що може вбудовуватися в контейнер.

Так, у даній роботі для вдалого представлення необхідних траєкторій геометричних елементів SVG-зображення виконується декілька етапів. Наприклад, на вхід подається крива як зображено на рис. 3.10.

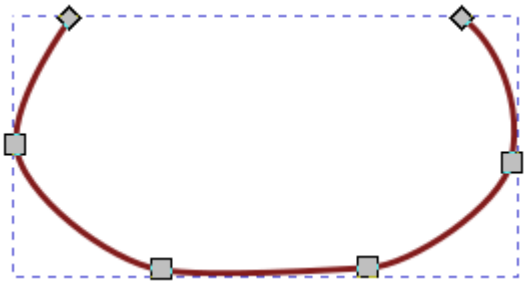


Рисунок 3.10 – Приклад кривої

Крива вище описується таким набором точок:

```
m 80.370966,132.70967 c 0,0 -2.572581,3.45968 -2.39516,5.58871 0.177115,2.12536 4.109612,5.1356 6.387097,5.5 C 86.580643,144.15322 91.72581,143.79837 93.58871,143.70968 95.45161,143.62098 99.46581,140.95655 99.88709,139.00807 100.59677,135.72582 98.55645,133.15323 97.75806,132.70968
String
```

Рисунок 3.11 – Представлення кривої точками

Спочатку крива на рис. 3.11 розбивається на повністю окремі елементи:

```
m, 80.370966, 132.70967, c, 0, 0, -2.572581, 3.45968, -2.39516, 5.58871, 0.177115, 2.12536, 4.109612, 5.1356, 6.387097, 5.5, C, 86.580643, 144.15322, 91.72581, 143.79837, 93.58871, 143.70968, 95.45161, 143.62098, 99.46581, 140.95655, 99.88709, 139.00807, 100.59677, 135.72582, 98.55645, 133.15323, 97.75806, 132.70968
List<String>
```

Рисунок 3.12 – Розбита крива на першому етапі

Далі крива на рис. 3.12 розбивається за окремими літералами кривої. Даний етап передбачений для того, аби мати змогу обробляти криві, які містять в собі внутрішні криві. Під внутрішніми кривими мається на увазі існування всередині опису літералів M або m (у залежності від типу координат), які у сенсі мови розмітки SVG мають на увазі початок нової траєкторії. Саме ця властивість використовується для представлення закодованої кривої, і тому така функція обробки необхідна – вона однаково інтерпретує оригінальні та розбиті криві, виключаючи необхідність написання окремого функціоналу для парсингу контейнеру при вбудовуванні чи вилученні даних.

У даному випадку крива не містить внутрішніх кривих, тому набуває такого вигляду:

```
[m 80.370966,132.70967 c 0.0,0.0 -2.572581,3.45968 -2.39516,5.58871 0.177115,2.12536 4.109612,5.1356 6.387097,5.5 C 86.580643,144.15322 91.72581,143.79837 93.58871,143.70968 95.45161,143.62098 99.46581,140.95655 99.88709,139.00807 100.59677,135.72582 98.55645,133.15323 97.75806,132.70968 ]
List<Curve>
```

Рисунок 3.13 – Список вилучених кривих на другому етапі

Тепер необхідно розбити криву на сегменти, що вже знаходяться в ній, але не очевидні для користувача – кількість пар точок завжди кратна трьом, тобто кожні три елементи по суті є окремою складовою шляху, що неявно пов’язані один з одним початковими і кінцевими точками (контрольними точками кривої Без’є). На рис. 3.14 відображено таке розбиття, яке полягає у створенні об’єктів сегментів з трьома точками.

```
[m 80.370966,132.70967 c 0.0,0.0 -2.572581,3.45968 -2.39516,5.58871 c 0.177115,2.12536 4.109612,5.1356 6.387097,5.5 C 86.580643,144.15322 91.72581,143.79837 93.58871,143.70968 C 95.45161,143.62098 99.46581,140.95655 99.88709,139.00807 C 100.59677,135.72582 98.55645,133.15323 97.75806,132.70968 ]
List<Curve>
```

Рисунок 3.14 – Список кривих з виокремленням сегментів на третьому етапі

Далі виконується представлення сегментів у в канонічному вигляді, при якому остання точка поточного сегменту є початковою точкою наступного:

```
[m 80.370966,132.70967
c 80.370966,132.70967 0.0,0.0 -2.572581,3.45968 -2.39516,5.58871
c -2.39516,5.58871 0.177115,2.12536 4.109612,5.1356 6.387097,5.5
C 6.387097,5.5 86.580643,144.15322 91.72581,143.79837 93.58871,143.70968
C 93.58871,143.70968 95.45161,143.62098 99.46581,140.95655 99.88709,139.00807
C 99.88709,139.00807 100.59677,135.72582 98.55645,133.15323 97.75806,132.70968]
List<Curve>
```

Рисунок 3.15 – Список кривих у канонічному вигляді на четвертому етапі

У зображеннях векторної графіки часто відбувається зміна координат від абсолютних до відносних та навпаки, і часто можливим є наявність обох типів координат в одному шляху. Цей факт не дозволяє модифікувати криві без руйнування оригінального зображення, тому на останньому етапі відбувається перехід до абсолютних координат:

```
[m 80.370966,132.70967
C 80.370966,132.70967 80.370966,132.70967 77.798385,136.16935 77.975806,138.29838
C 77.975806,138.29838 78.152921,140.42374 82.085418,143.43398 84.362903,143.79838
C 84.362903,143.79838 86.580643,144.15322 91.72581,143.79837 93.58871,143.70968
C 93.58871,143.70968 95.45161,143.62098 99.46581,140.95655 99.88709,139.00807
C 99.88709,139.00807 100.59677,135.72582 98.55645,133.15323 97.75806,132.70968]
List<Curve>
```

Рисунок 3.16 – Список кривих після переходу до абсолютних координат на п’ятому етапі

Саме таким чином відбувається парсинг рядку даних у структуру, над якою можна проводити стеганографічні перетворення. У контексті програмного представлення об’єкту зображення воно складається з двовимірного списку, елементами якого є списки кривих, в яких містяться сегменти з відповідними точками (рис. 3.17).

Рисунок 3.17 – Представлення зображення у вигляді об’єктів

3.4 Розробка програмної реалізації алгоритмів приховування інформації з векторних зображень

Згідно з наведеною у 2 розділі моделлю виконання вбудовування даних та описаною вище структурою проєкту виконується програмна реалізація. Лістинг коду наведений у Додатку А. Для демонстрації прикладу виконання реалізованих функцій приховування даних використовується зображення на рис. 3.10. Після того, як зображення було проаналізовано та був виконаний його парсинг зі строкового вигляду до об’єкту (рис. 3.16), можна починати процес приховування даних.

Бітовий режим

Для початку визначаються параметри кодування:

Таблиця 3.1 – Параметри приховування для бітового методу

Параметр	Значення
Крок зміни параметра	0.0005
Точність	5
Кількість біт у кривій	16

З рис. 3.10 та рис. 3.16 видно, що зображення містить 5 сегментів, враховуючи прийняту кількість біт, що може бути вбудована у криву, рекомендована довжина повідомлення складає 80 біт, згідно з (2.1). Таким чином, можна приховати повідомлення «secret», для чого спочатку воно представляється у двійковому вигляді:

[0111001101100101, 0110001101110010, 0110010101110100]

Рисунок 3.18 – Повідомлення для приховування у двійковому вигляді

Так як об’єкт повідомлення у даній програмній реалізації приймає в себе параметр довжини двійкових блоків (який співпадає з параметром кодування кількості біт в кривій), то двійкове повідомлення одразу представляється окремими елементами, кожен з яких буде вбудовуватися в окремий сегмент: $M = [m_0, m_1, ..., m_N]$, де $N = \frac{M_{len}}{bits}$.

Так, починається процес вбудовування:

1) Поточний сегмент:

C 80.370966,132.70967 80.370966,132.70967 77.798385,136.16935 77.975806,138.29838

Валідність: false – вбудовування не відбувається.

2) Поточний сегмент:

C 77.975806,138.29838 78.152921,140.42374 82.085418,143.43398 84.362903,143.79838

Валідність: true.

Далі над валідним сегментом відбувається перетворення в залежності від біту, що отримується на вхід:

Таблиця 3.2 – Процес закодування валідного сегменту побітовим методом

Поточний блок двійкових даних: 0111001101100101				
Ітерація	Кількість сегментів	Сегменти	Біт	Крок
0	1	(c_1)	0	0.0005
1	2	$(c_1 \cup c_2)$	1	0.001
2	3	$(c_1 \cup c_2 \cup c_3)$	1	0.0015
3	4	$(c_1 \cup c_2 \cup c_3 \cup c_4)$	1	0.002

Продовження таблиці 3.2 – Процес закодування валідного сегменту побітовим методом

4	4	$(c_1 \cup c_2 \cup c_3 \cup c_4)$	0	0.0025
5	4	$(c_1 \cup c_2 \cup c_3 \cup c_4)$	0	0.003
6	5	$(c_1 \cup c_2 \cup c_3 \cup c_4 \cup c_5)$	1	0.0035
7	6	$(c_1 \cup c_2 \cup c_3 \cup c_4 \cup c_5 \cup c_6)$	1	0.004
8	6	$(c_1 \cup c_2 \cup c_3 \cup c_4 \cup c_5 \cup c_6)$	0	0.0045
9	7	$(c_1 \cup c_2 \cup c_3 \cup c_4 \cup c_5 \cup c_6 \cup c_7)$	1	0.005
10	8	$(c_1 \cup c_2 \cup c_3 \cup c_4 \cup c_5 \cup c_6 \cup c_7 \cup c_8)$	1	0.0055
11	8	$(c_1 \cup c_2 \cup c_3 \cup c_4 \cup c_5 \cup c_6 \cup c_7 \cup c_8)$	0	0.006
12	8	$(c_1 \cup c_2 \cup c_3 \cup c_4 \cup c_5 \cup c_6 \cup c_7 \cup c_8)$	0	0.0065
13	9	$(c_1 \cup c_2 \cup c_3 \cup c_4 \cup c_5 \cup c_6 \cup c_7 \cup c_8 \cup c_9)$	1	0.007
14	9	$(c_1 \cup c_2 \cup c_3 \cup c_4 \cup c_5 \cup c_6 \cup c_7 \cup c_8 \cup c_9)$	0	0.0075
15	10	$(c_1 \cup c_2 \cup c_3 \cup c_4 \cup c_5 \cup c_6 \cup c_7 \cup c_8 \cup c_9 \cup c_{10})$	1	0.008

3) Поточний елемент:

C 84.362903,143.79838 86.580643,144.15322 91.72581,143.79837 93.58871,143.70968

Валідність: true.

Поточний блок двійкових даних: 0110001101110010

Далі закодування відбувається за аналогією з першим сегментом (табл. 3.2).

4) Поточний елемент:

C 93.58871,143.70968 95.45161,143.62098 99.46581,140.95655 99.88709,139.00807

Валідність: true.

Поточний блок двійкових даних: 0110010101110100

Далі закодування відбувається за аналогією з першим сегментом (табл. 3.2).

5) Поточний елемент:

C 99.88709,139.00807 100.59677,135.72582 98.55645,133.15323 97.75806,132.70968

Валідність: false.

Отримана крива має наступний вигляд:

m 80.370966,132.70967 C 80.370966,132.70967 77.798385,136.16935 77.975806,138.29838 m 0.0,0.0 C 77.97598,138.30051 77.97616,138.30264 77.97634,138.30477 77.97662,138.30796 77.97691,138.31115 77.9772,138.31434 77.97759,138.31859 77.978,138.32284 77.97842,138.3271 77.97915,138.33453 77.97993,138.34197 77.98075,138.34942 77.98169,138.35791 77.98269,138.36641 77.98375,138.37493 77.98506,138.38553 77.98646,138.39615 77.98795,138.40679 77.98959,138.41844 77.99133,138.43011 77.99318,138.44181 77.99552,138.45661 77.99803,138.47145 78.00071,138.48633 78.00375,138.50321 78.00701,138.52014 78.01048,138.53712 78.44092,140.64232 82.16947,143.44743 84.362903,143.79838 m 0.0,0.0 C 84.36512,143.79873 84.36734,143.79908 84.36956,143.79943 84.37289,143.79996 84.37623,143.80049 84.37958,143.80102 84.38737,143.80225 84.3952,143.80347 84.40306,143.80468 84.41202,143.80606 84.42102,143.80743 84.43007,143.80879 84.44133,143.81048 84.45266,143.81215 84.46406,143.8138 84.47654,143.81561 84.4891,143.8174 84.50175,143.81917 84.51547,143.82109 84.52929,143.82298 84.5432,143.82485 84.56049,143.82717 84.57792,143.82946 84.5955,143.83171 86.92184,144.12976 91.78824,143.7954 93.58871,143.70968 m 0.0,0.0 C 93.59057,143.70959 93.59244,143.7095 93.59431,143.70941 93.59711,143.70927 93.59991,143.70912 93.60272,143.70897 93.60833,143.70867 93.61396,143.70834 93.6196,143.70799 93.6271,143.70753 93.63464,143.70702 93.64221,143.70647 93.65164,143.70579 93.66112,143.70505 93.67065,143.70425 93.68108,143.70337 93.69157,143.70242 93.70212,143.70139 93.71356,143.70027 93.72507,143.69907 93.73665,143.69778 93.75008,143.69629 93.76361,143.69468 93.77723,143.69296 95.70906,143.44833 99.47952,140.89314 99.88709,139.00807 m 0.0,0.0 C 100.59677,135.72582 98.55645,133.15323 97.75806,132.70968

Рисунок 3.19 – Отримана крива закодована бітовим методом

Враховуючи, що параметр зміни кроку має маленьке значення, то відповідно створені сегменти будуть дуже маленькими, а їх контрольні точки знаходитимуться поруч одна з одною. Так, на рис. 3.23 відображено фрагмент кривої з рис. 3.10, при цьому довелося збільшувати зображення у 5600% раз, аби мати можливість

розглянути отримані точки (перегляд відбувається у графічному редакторі у режимі перегляду вузлів).

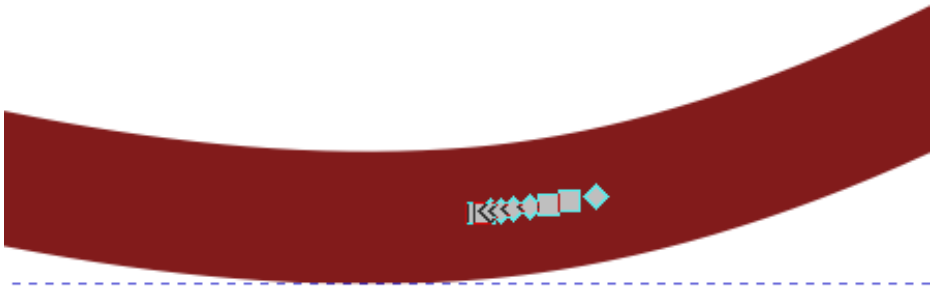


Рисунок 3.20 – Фрагмент закодованого сегменту з параметром кроку 0.0005

Для порівняння виконується закодування з параметром кроку 0.01, при якому штучно створені сегменти та їх точки більш розподілені по довжині сегменту:

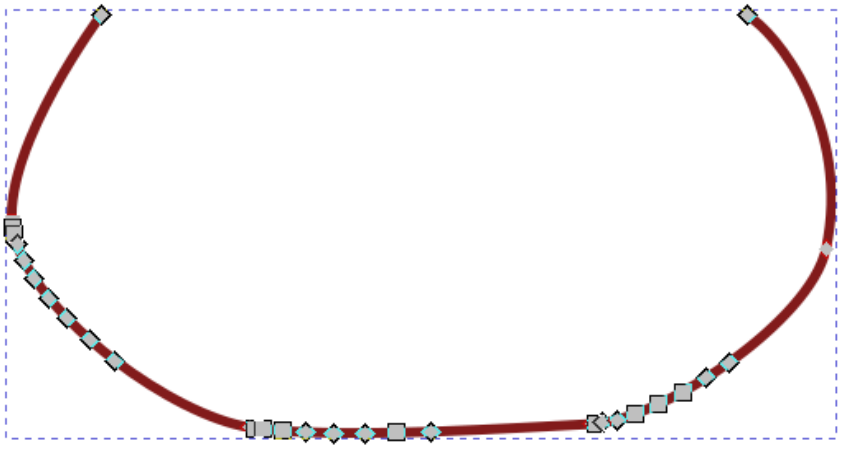


Рисунок 3.21 – Закодована крива з параметром кроку 0.01

Після закінчення приховування даних формується стегоключ за схемою, приведеною на рис. 2.9. Для цього використовуються параметри закодування з табл. 3.1 і отримується наступний ключ:

```
<a id="w3orgSnaps" href="k5.0E-4:b16:p5/"/>
```

Рисунок 3.22 – Бітовий стегоключ

Патерновий режим

Для початку визначаються параметри кодування:

Таблиця 3.3 – Параметри приховування для методу патернів

Параметр	Значення
Початковий крок	0.0005
Довжина патерну	4
Точність	5
Кількість біт у кривій	16

Повідомлення приховується те саме, що й у прикладі бітового закодування, при цьому рядок біт окрім поділу на рівні частини, довжина яких дорівнює кількості біт у кривій, ділиться також на блоки довжиною, що дорівнює визначеній довжині патерну. Враховуючи, що довжина патернів у даному випадку встановлена значенню 4, то можна побудувати таку просту таблицю патернів (ТП), як зображено у табл. 3.4.

Таблиця 3.4 – Приклад таблиці патернів

Індекс	Патерн	Крок
1	0000	0.0095
2	0001	0.009
3	0010	0.0085
4	0011	0.008
5	0100	0.0075
6	0101	0.007
7	0110	0.0065
8	0111	0.006
9	1000	0.0055
10	1001	0.005
11	1010	0.0045
12	1011	0.004
13	1100	0.0035
14	1101	0.003
15	1110	0.0025
16	1111	0.002

Так, маючи ініціалізовану ТП, починається процес вбудовування:

1) Поточний сегмент:

C 80.370966,132.70967 80.370966,132.70967 77.798385,136.16935 77.975806,138.29838

Валідність: false – вбудовування не відбувається.

Фінальний параметр: 0.

2) Поточний сегмент:

C 77.975806,138.29838 78.152921,140.42374 82.085418,143.43398 84.362903,143.79838

Валідність: true.

Далі над валідним сегментом відбувається перетворення в залежності від патерну, що отримується на вхід (при цьому при першій ітерації виконується додавання до параметру значення початкового кроку):

Таблиця 3.5 – Процес закодування валідного сегменту методом патернів

Поточний блок двійкових даних: 0111 0011 0110 0101					
Ітерація	Кількість сегментів	Сегменти	Патерн	Індекс у ТП	Крок
0	2	$(c_1 \cup c_2)$	0111	8	0.0065
1	3	$(c_1 \cup c_2 \cup c_3)$	0011	4	0.0145
2	4	$(c_1 \cup c_2 \cup c_3 \cup c_4)$	0110	7	0.021
3	5	$(c_1 \cup c_2 \cup c_3 \cup c_4 \cup c_5)$	0101	6	0.028

Фінальний параметр: 0.028.

3) Поточний елемент:

C 84.362903,143.79838 86.580643,144.15322 91.72581,143.79837 93.58871,143.70968

Валідність: true.

Поточний блок двійкових даних: 0110001101110010

Далі закодування відбувається за аналогією з першим сегментом (табл. 3.4).

Фінальний параметр: 0.0295.

4) Поточний елемент:

C 93.58871,143.70968 95.45161,143.62098 99.46581,140.95655 99.88709,139.00807

Валідність: true.

Поточний блок двійкових даних: 0110010101110100

Далі закодування відбувається за аналогією з першим сегментом (табл. 3.4).

Фінальний параметр: 0.0275.

5) Поточний елемент:

C 99.88709,139.00807 100.59677,135.72582 98.55645,133.15323 97.75806,132.70968

Валідність: false.

Фінальний параметр: 0.

Отримана крива має наступний вигляд:

```
m 80.370966,132.70967 C 80.370966,132.70967 77.798385,136.16935 77.975806,138.29838 m 0.0,0.0 C
77.976957,138.312195 77.978267,138.326047 77.979734,138.339936 77.982985,138.370717
77.987009,138.401677 77.991788,138.432805 77.998609,138.477233 78.00697,138.522004
78.016824,138.567088 78.029687,138.625937 78.045093,138.685318 78.062938,138.745163
78.682423,140.822629 82.240983,143.45887 84.362903,143.79838  m 0.0,0.0 C 84.378427,143.800864
84.394095,143.803313 84.409904,143.805728 84.443543,143.810866 84.477822,143.815848
84.51272,143.820677 84.560845,143.827337 84.610147,143.833706 84.660573,143.839793
84.729923,143.848165 84.801398,143.856004 84.874859,143.863331 87.2916,144.104385
91.857486,143.792101 93.58871,143.70968  m 0.0,0.0 C 93.60175,143.709059 93.614896,143.708312
93.628145,143.70744 93.654458,143.705708 93.681179,143.703482 93.708292,143.700772
93.746483,143.696955 93.785453,143.692176 93.825159,143.686461 93.878663,143.67876
93.933502,143.669358 93.989572,143.658316 95.972407,143.267837 99.493981,140.826254
99.88709,139.00807  m 0.0,0.0 C 100.59677,135.72582 98.55645,133.15323 97.75806,132.70968
```

Рисунок 3.23 – Отримана крива закодована методом патернів

Звернувши увагу на кількість ітерацій у табл. 3.5, очевидним стає основна перевага методу патернів над бітовим методом – у рази зменшене число розривів кривої. Візуально меншу сегментацію можна спостерігати на рис. 3.24.

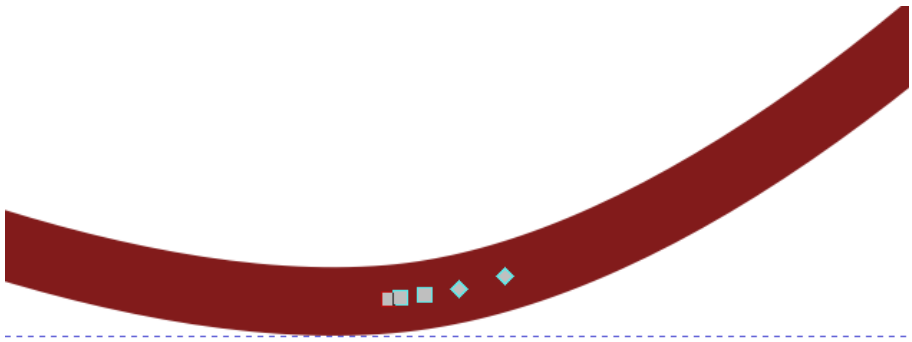


Рисунок 3.24 – Фрагмент закодованого сегменту при кодуванні методом патернів

Тепер виконується формування патернового стегоключа згідно зі схемою, представленою на рис. 2.15. Для цього беруться значення параметрів закодування із табл. 3.3 та фінальні параметри для кожної кривої, отримані під час вбудовування даних, та формується ключ наступного вигляду:

Фінальні параметри у двійковому вигляді:
000000000000000000000000, 000001110010101100000010, 000001111000110101001111, 000001110000101000111101, 000000000000000000000000

↓

Фінальні параметри у десятковому вигляді:
0, 469762, 494927, 461373, 0

↓

Стегоключ:

```
<a id="w3orgSnaps" href="l4:s5.0E-4:b16:p5/
v0:469762:494927:461373:0"/>
```

Рисунок 3.25 – Патерновий стегоключ

3.5 Розробка програмної реалізації алгоритмів вилучення інформації з векторних зображень

Згідно з наведеною у 2 розділі моделлю виконання вилучення даних та описаною вище структурою проєкту виконується програмна реалізація. Лістинг коду наведений у Додатку Б. Для демонстрації прикладу виконання імплементації вилучення даних використовуються зображення, що були отримані закодуванням обидва методами вище. Кожний заповнений контейнер аналізується, над ним проводиться парсинг таким самим чином, як і для пустого контейнеру, після чого зображення приводиться до виду об’єкту, що схематично відображений на рис. 3.17.

Бітовий режим

Для наведення прикладу вилучення інформації бітовим методом використовується крива на рис. 3.19. Спочатку обраховується значення кінцевого параметру за допомогою формули (2.2). Так як це значення отримується внаслідок

того, що отримувач знає параметри закодування, то фактично такі параметри виступають стегоключами і тільки з їх знанням стає можливим вилучення інформації.

Стегоключ отримується за схемою, представленою на рис. 2.18, після чого відновлюються параметри закодування: крок $k = 0.0005$, кількість біт у кривій $bits = 16$, точність виконання операцій 5. Таким чином, маємо, що останнє значення, при якому приймалось рішення про розрив кривої дорівнює: $(16+1)\cdot 0.0005 = 0.0085 - 0.0005 = 0.008$. Починається процес вилучення:

1) Поточна крива для вилучення:

m 80.370966,132.70967 C 80.370966,132.70967 77.798385,136.16935 77.975806,138.29838

Кількість сегментів дорівнює одному, тому вилучення не відбувається.

2) Поточна крива для вилучення:

m 0.0,0.0 C 77.97598,138.30051 77.97616,138.30264 77.97634,138.30477 C 77.97662,138.30796 77.97691,138.31115 77.9772,138.31434 C 77.97759,138.31859 77.978,138.32284 77.97842,138.3271 C 77.97915,138.33453 77.97993,138.34197 77.98075,138.34942 C 77.98169,138.35791 77.98269,138.36641 77.98375,138.37493 C 77.98506,138.38553 77.98646,138.39615 77.98795,138.40679 C 77.98959,138.41844 77.99133,138.43011 77.99318,138.44181 C 77.99552,138.45661 77.99803,138.47145 78.00071,138.48633 C 78.00375,138.50321 78.00701,138.52014 78.01048,138.53712 C 78.44092,140.64232 82.16947,143.44743 84.362903,143.79838

З даної кривої можна вилучати дані:

Таблиця 3.6 – Процес вилучення даних з кривої побітовим методом

Ітерація	Кількість сегментів	Сегменти	Біт	Крок
0	9	$(c_1 \cup c_2 \cup c_3 \cup c_4 \cup c_5 \cup c_6 \cup c_7 \cup c_8 \cup c_9)$	1	0.008
1	9	$(c_1 \cup c_2 \cup c_3 \cup c_4 \cup c_5 \cup c_6 \cup c_7 \cup c_8 \cup c_9)$	0	0.0075
2	8	$(c_1 \cup c_2 \cup c_3 \cup c_4 \cup c_5 \cup c_6 \cup c_7 \cup c_8)$	1	0.007

Продовження таблиці 3.6 – Процес вилучення даних з кривої побітовим методом

3	8	$(c_1 \cup c_2 \cup c_3 \cup c_4 \cup c_5 \cup c_6 \cup c_7 \cup c_8)$	0	0.0065
4	8	$(c_1 \cup c_2 \cup c_3 \cup c_4 \cup c_5 \cup c_6 \cup c_7 \cup c_8)$	0	0.006
5	7	$(c_1 \cup c_2 \cup c_3 \cup c_4 \cup c_5 \cup c_6 \cup c_7)$	1	0.0055
6	6	$(c_1 \cup c_2 \cup c_3 \cup c_4 \cup c_5 \cup c_6)$	1	0.005
7	6	$(c_1 \cup c_2 \cup c_3 \cup c_4 \cup c_5 \cup c_6)$	0	0.0045
8	5	$(c_1 \cup c_2 \cup c_3 \cup c_4 \cup c_5)$	1	0.004
9	4	$(c_1 \cup c_2 \cup c_3 \cup c_4)$	1	0.0035
10	4	$(c_1 \cup c_2 \cup c_3 \cup c_4)$	0	0.003
11	4	$(c_1 \cup c_2 \cup c_3 \cup c_4)$	0	0.0025
12	3	$(c_1 \cup c_2 \cup c_3)$	1	0.002
13	2	$(c_1 \cup c_2)$	1	0.0015
14	1	(c_1)	1	0.001
15	1	(c_1)	0	0.0005

Так як повідомлення вилучається з кінця, то відповідно для правильного інтерпретування інформації необхідно розвернути отриману послідовність біт. Тобто, отримується блок 0111001101100101, що співпадає з блоком, що був вбудований. Наступні внутрішні криві об’єднуються таким самим чином, після чого відбувається відновлення цілого повідомлення.

Метод патернів

Повідомлення вилучається із зображення, вміст якого представлений на рис. 3.22 та фрагмент якого відображений на рис. 3.23. Для початку виконується вилучення стегоключа із файлу зображення згідно зі схемою, що представлена на рис. 2.20. Так, вилучений ключ дає отримувачу інформацію про:

- точність закодування – 5;
- кількість біт у кривій – 16;
- початковий крок – 0.0005;
- довжину патерну – 4;
- фінальні параметри закодування – {0.0, 0.028, 0.0295, 0.0275, 0.0}.

На основі вилучених параметрів будується таблиця патернів та починається процес відновлення кривих:

1) Поточна крива для вилучення:

m 80.370966,132.70967 C 80.370966,132.70967 77.798385,136.16935 77.975806,138.29838

Поточний фінальний параметр: 0.

Кількість сегментів дорівнює 1, а значення фінального параметру дорівнює 0, тому вилучення не відбувається.

2) Поточна крива для вилучення:

m 0.0,0.0 C 0.0,0.0 77.976957,138.312195 77.978267,138.326047 77.979734,138.339936 C 77.979734,138.339936 77.982985,138.370717 77.987009,138.401677 77.991788,138.432805 C 77.991788,138.432805 77.998609,138.477233 78.00697,138.522004 78.016824,138.567088 C 78.016824,138.567088 78.029687,138.625937 78.045093,138.685318 78.062938,138.745163 C 78.062938,138.745163 78.682423,140.822629 82.240983,143.45887 84.362903,143.79838

Поточний фінальний параметр: 0.028.

Дана крива була закодована, тому з неї вилучаються дані:

Таблиця 3.7 – Процес вилучення даних із кривої методом патернів

Поточний блок двійкових даних: 0111 0011 0110 0101					
Ітерація	Кількість сегментів	Сегменти	Патерн	Індекс у ТП	Крок
0	5	$(c_1 \cup c_2 \cup c_3 \cup c_4 \cup c_5)$			0.028
1	4	$(c_1 \cup c_2 \cup c_3 \cup c_4)$	0101	6	0.028
2	3	$(c_1 \cup c_2 \cup c_3)$	0110	7	0.021
3	2	$(c_1 \cup c_2)$	0011	4	0.0145
4	1	(c_1)	0111	8	0.0065

У методі патернів на першій ітерації відбувається об’єднання останніх двох сегментів кривої, а потім реалізується пошук необхідного запису у таблиці патернів (табл. 3.4), при відповідному значенні зміни кроку якого виконується умова вилучення.

Так як вилучення відбувається з кінця кривої, то вилучене повідомлення треба читати також з кінця. Таким чином отримується:

$$M = \{0101, 0110, 0011, 0111\} \rightarrow \{0111, 0011, 0110, 0101\},$$

що і є закодованим першим блоком повідомлення.

3) Поточна крива:

m 0.0,0.0 C 0.0,0.0 84.378427,143.800864 84.394095,143.803313 84.409904,143.805728 C 84.409904,143.805728 84.443543,143.810866 84.477822,143.815848 84.51272,143.820677 C 84.51272,143.820677 84.560845,143.827337 84.610147,143.833706 84.660573,143.839793 C 84.660573,143.839793 84.729923,143.848165 84.801398,143.856004 84.874859,143.863331 C 84.874859,143.863331 87.2916,144.104385 91.857486,143.792101 93.58871,143.70968

Поточний фінальний параметр: 0.0295.

Дана крива була закодована, тому з неї вилучаються дані аналогічним способом як наведено у табл. 3.7.

4) Поточна крива:

m 0.0,0.0 C 0.0,0.0 93.60175,143.709059 93.614896,143.708312 93.628145,143.70744 C 93.628145,143.70744 93.654458,143.705708 93.681179,143.703482 93.708292,143.700772 C 93.708292,143.700772 93.746483,143.696955 93.785453,143.692176 93.825159,143.686461 C 93.825159,143.686461 93.878663,143.67876 93.933502,143.669358 93.989572,143.658316 C 93.989572,143.658316 95.972407,143.267837 99.493981,140.826254 99.88709,139.00807

Фінальний параметр: 0.0275.

Дана крива була закодована, тому з неї вилучаються дані аналогічним способом як наведено у табл. 3.7.

5) Поточна крива:

m 0.0,0.0 C 0.0,0.0 100.59677,135.72582 98.55645,133.15323 97.75806,132.70968

Фінальний параметр: 0.

Кількість сегментів дорівнює 1, а значення фінального параметру дорівнює 0, тому вилучення не відбувається.

Таким чином, відбулось успішне вилучення блокових даних із зображення.

3.6 Висновки до третього розділу

Обрана мова програмування Java дозволяє реалізувати складну структуру даних зображення, що вимагається для виконання стеганографічних перетворень вбудовування та вилучення даних. Для імплементації використовуються такі модулі як javafx.web, javafx.fxml, javafx.controls, java.xml. Розроблений ГКІ дозволяє користувачеві виконувати завантаження довільних зображень SVG, повідомлень, виставляти власні параметри стегоперетворень та проводити експериментальні дослідження.

Наведені приклади виконання вбудовування демонструють фрагментацію кривих у залежності від вхідної двійкової послідовності та формування ключів. Приклади ж вилучення даних показують процес відновлення початкових кривих та отриманими з контейнеру ключами закодування.

4 ЕКСПЕРИМЕНТАЛЬНІ ДОСЛІДЖЕННЯ РОЗРОБЛЕНИХ АЛГОРИТМІВ, ОБҐРУНТУВАННЯ ПРАКТИЧНОЇ РЕКОМЕНДАЦІЇ ЩОДО ЇХ ВПРОВАДЖЕННЯ

Для проведення експериментальних досліджень використовується зображення векторної графіки розміром 90392 байт або 88.27 кілобайт (рис. 2.6 (а)), у яке приховується повідомлення довжиною 2400 біт.

4.1 Дослідження показників втрати біт при атаках афінних перетворень

Для експерименту спочатку виконується закодування зображення, а після – його модифікація (імітація атаки) відповідно до обраного типу афінного перетворення визначеною кількістю ітерацій, вилучаючи при цьому кожного разу приховане повідомлення. Для проведення експериментів використовуються набори параметрів, що наведені у табл. В.1 (Додаток В).

Виявлення відсотків втрачених біт відбувається шляхом застосування до контейнера різних типів афінних перетворень: перенесення, поворот, X-зсув (за віссю абцис), Y-зсув (за віссю ординат), масштабування для збільшення, масштабування для збільшення, майже афінне перетворення – згідно з виразами (1.7)-(1.12) та коефіцієнтами, наведеними у табл. В.2.

На рис. В.1 наведено візуальне відображення застосування виразу (1.8) при афінному перетворенні повороту. Рис. В.1 (а) є звичайним відображенням, а рис. В.1 (б) з виділенням контурів для більшої наочності. Рис. В.2 демонструє варіанти стегоконтейнерів, над якими було виконано перетворення масштабування для збільшення (рис. В.2 (а), оригінал зліва) та зменшення (рис. В.2 (б), оригінал зліва).

На рис. В.3 можна побачити, що відбувається із зображенням при виконанні майже афінного перетворення із додаванням шуму. Справа знаходиться оригінал зображення, а зліва – його модифікована афінною атакою версія. Як видно, таке перетворення найбільш сильно впливає на відображення стегоконтейнеру (при цьому рис. В.3 є першою ітерацією експерименту).

Для експериментів використовувалось розроблена система, описана у розділі 3.2, зокрема функціонал, представлений на рис. 3.11-3.12. У Додатку В наведено отримані результати для обох бітового та патернового режимів у табличному та графічному виглядах.

Судячи з даних, представлених на рис. В.4-В.31, можна побачити, що деякі набори параметрів запроваджують більшу стійкість до збереження повідомлення при модифікації стегоконтейнера. Набори set1, set5, set9 відзначаються такими, за яких хибно вилучаються біти при усіх проведених експериментах. Найменші показники неправильно вилучених біт спостерігаються при перетворенні перенесення, а найбільші – при X- та Y-зсувах.

4.2 Дослідження швидкісних характеристик, коефіцієнтів спотворення та збільшення розміру контейнерів

Показники збільшення розміру контейнера у залежності від кількості біт, що приховуються в одній кривій, та точності закодування наведені у табл. 4.1.

Таблиця 4.1 – Показники збільшення стегоконтейнерів

Біт у кривій	Точність	Розмір, КБ			Збільшення, %	
		Оригінал	Бітовий	Патерновий	Бітовий	Патерновий
40	5	88	155.06	125.79	76,2	42,9
40	6		161.71	129.54	83,7	47,2
64	5		154.93	126.11	76	43,3
64	6		161.61	129.83	83,6	47,5
80	5		155.01	125.62	76,1	42,7
80	6		161.59	129.23	83,6	46,8

Як видно з таблиці вище, кількість біт, що приховуються в одній кривій, у даному випадку не впливає на розміри стегоконтейнеру, що пояснюється тим, що приховується повідомлення однією довжини (2400 біт) і контейнер має достатню кількість кривих Без'є. Тобто, навіть при різних значеннях кількості біт загальне число розривів усього зображення буде приблизно однаковим, а відрізнятися буде лише кількість кривих, використаних для приховування. У даному випадку,

параметром, що значним чином впливає на розмір контейнеру, є точність. Очевидно, що патерновий режим спричиняє менше збільшення стегоконтейнеру.

У Додатку Г наведено показники часу закодування, що були отримані при проведенні експериментів щодо стійкості до афінних перетворень. Значення часу приховування інформації визначалося для кожного набору параметрів, а вилучення – обраховувався середній час декодування за всі ітерації кожного набору. У табл. Г.1 наведено середні значення, що були отримані під час усіх проведених афінних перетворень. Узагальнюючи табл. Г.1, отримуються наступні показники швидкодії розробленого методу:

Таблиця 4.3 – Середні показники часу та швидкості закодування і декодування

Режим	Закодування		Декодування	
	Час, мс	Швидкість, біт/мс	Час, мс	Швидкість, біт/мс
Бітовий	107,57	22,31	134,71	17,81
Патерновий	34,14	70,29	27,28	87,97

Як видно з наведеної вище таблиці, бітовий режим виконує стеганографічні операції довше, ніж патерновий, що пов’язано з тим, що у другому випадку виконується менша кількість розривів кривої. На рис. Г.14 - Г.15 наведено візуальне відображення отриманих середніх значень часу.

У Додатку Д наведено результати дослідження візуального спотворення стегоконтейнера. Так як методи порівнянь зображень у випадку векторної графіки зводяться до порівняння координат геометричних елементів, то для алгоритму, що був реалізований у даній роботі, необхідно проводити конвертування стегоконтейнерів у растрову графіку. ПЗ AntiDupl.NET (версії 2.3.9) дозволяє виконати порівняння колекції зображень для пошуку однакових екземплярів [46]. За отриманими результатами (рис. Д.1-Д.2) видно, що різниці між контейнером і стегоконтейнером знайдено не було.

4.2 Практичні рекомендації щодо подальшого впровадження

Проведене дослідження щодо розмірів стегоконтейнерів демонструє, що бітовий метод призводить до збільшення контейнеру на близько 80%, а патерновий – на майже 50%. При цьому показники значно відрізняються при використанні більшого значення точності 6. Так, дана точність у той самий час забезпечує більшу стійкість до збереження повідомлення при атаках афінного перетворення, що також підтверджується у роботі [21]. У загальному випадку, результати підтверджують дані, наведені в оригінальній роботі Кінзерявого [9].

Таким чином, можна сказати, що для практичного використання реалізованого методу необхідно визначати пріоритети при виборі набору параметрів у залежності від того, яка мета має досягатися – більша скритність стегоконтейнера за рахунок малого розміру або точне збереження оригінального повідомлення.

У ході описаного дослідження раціональним приймається збільшення точності заради збереження повідомлення за умови використання контейнерів, що містять нетехнічні мотиви, скетчеві відображення, адже вони не визиватимуть підозр щодо їх розмірів і завжди матимуть достатню кількість кривих для поміщення секретного повідомлення.

Важливо також відмітити, що модифікації повороту або збільшення, або будь-якого іншого перетворення, проведені у графічному редакторі, призведуть до повного руйнування повідомлення. Це пов’язано з тим, що графічні середовища прагнуть оптимізувати зображення шляхом, наприклад, видалення надлишкових елементів. Геометричні перетворення у графічному середовищі також надто сильно змінюють структуру траєкторій, а звідси – й модифікованих точок.

У загальному випадку бітовий метод є надійнішим при виконанні вилучення при застосованих атаках, адже він у будь-якому випадку вилучить двійкові 0 або 1 (навіть якщо це неправильний біт) та продовжить перевірку умови вилучення на наступному кроці параметра побудови кривої. Патерновий метод у свою чергу має залежність від попередньо вилучених блоків двійкових даних – якщо на деякому

кроці не виконується умова вилучення, то подальше вилучення є гарантовано хибним, бо крок зміни параметра не змінюється відповідно до патерну, що шукається. У випадку, коли така ситуація стається при першому вилученні, втрачається уся послідовність даних, вбудованих у поточну криву, як наведено у [21].

4.3 Висновки до четвертого розділу

Застосування до контейнера різних типів афінних перетворень: перенесення, поворот, X-зсув (за віссю абцис), Y-зсув (за віссю ординат), масштабування для збільшення, масштабування для збільшення, майже афінне перетворення – приводить до виявлення відсотків хибно вилучених біт. При такому дослідженні було виявлено несприятливі набори параметрів, за яких досягаються найбільші відсотки втрат.

Вимірювання швидкості виконання стеганографічних перетворень показало, що показники патернового режиму є вищими, що пояснюється меншою кількістю виконуваних розривів кривої (при вбудовуванні) та об’єднання (при вилученні), хоча й в останньому випадку виконується пошук необхідного патерну у таблиці паттернів при кожній зміні кроку параметра побудови.

Розміри стегоконтейнерів напряму залежать від параметру точності, що застосовується при закодуванні, а також від кількості біт, що ховаються в одну криву. Проведені дослідження спотворення контейнерів за допомогою спеціалізованого ПЗ не показало відмінностей між стегоконтейнерами та оригінальним зображенням, які для цього були перетворені у растрову графіку.

ВИСНОВКИ

Під час проведення роботи було розглянуто та імплементовано мовою програмування Java методи стеганографічного вбудовування та вилучення даних у зображення векторної графіки, використовуючи математичний алгоритм побудови кривих Без’є третього ступеня. Виконано дослідження можливих показників та критеріїв до стеганографічних систем та висунуто вимоги до системи, що реалізовується.

Контейнером для проведення стеганографічних перетворень є зображення векторної графіки формату SVG. Даний формат використовує мову розмітки XML, що запроваджує можливість його використання як окремого самостійного формату, так і змішування з іншими елементами XML та HTML. Document Object Model запроваджує доступ до елементів зображення програмним чином.

Головним елементом у даній ПрО виступає path, який містить у собі необхідні траєкторії для відображення графіки. Такий елемент може бути надто комплексним, якщо складається із різнотипних підшляхів, що характеризують переходи від одного положення поточної точки до іншого. Такі траєкторії характерні технічним зображенням-іконкам, що найчастіше використовуються у веб-розробці. Цей факт висуває додаткові вимоги до моделювання попередньої обробки контейнера та не реалізується у межах даної роботи. Крім цього, варто відмітити, що такі SVG-іконки не мають достатньої кількості необхідних для виконання стеганографічного перетворення кривих, тобто у граничному випадку можуть містити лише одну криву.

При виконанні програмної імплементації використовувалися зображення, створені за природніми джерелами, схожі на скетчі, що можуть передаватися між відправником та отримувачем не заради зберігання у веб-застосунках, а для простого, на перший погляд, обміну. Створення таких зображень також

виправдовується тим, що такі контейнери здатні зберігати значно більшу кількість інформації.

При дослідженні було виявлено можливість збереження стегоключа у файлі самого стегоконтейнера під видом деякого посилання, що жодним чином не впливає на візуальне відображення зображення. Для бітового вбудовування необхідним є тільки збереження параметрів закодування, а для патернового – додатково зберігаються фінальні значення параметра кроку побудови кривої, попередньо приведені до десяткового вигляду.

Під час моделювання процесу вбудовування даних у зображення векторної графіки було виділено такі основні елементи як Обробник (для аналізу зображення та його перетворення у необхідну для вбудовування структуру даних), Кодер (що виконує модифікацію зображення) та DOM-парсер (виконує приведення об’єкту зображення у XML-структуру). При цьому для початку роботи користувач виконує завантаження контейнера, а система його обробляє та повідомляє дані аналізу (такі як: кількість кривих, в які можна виконувати приховання даних, та рекомендовану довжину текстового повідомлення). Після цього користувач завантажує повідомлення, яке конвертується стегосистемою до двійкового вигляду, та починає процес закодування. Під час даного процесу модифікуються початкові криві зображення відповідно до вхідної двійкової послідовності, а наприкінці формуються ключі закодування у залежності від обраного режиму (бітовий чи блоковий). На виході користувач отримує стегоконтейнер того ж самого формату, який візуально нічим не відрізняється від оригінального.

Під час моделювання процесу вилучення даних у зображення векторної графіки було виділено такі основні елементи як Обробник (для аналізу зображення та його перетворення у необхідну для вбудовування структуру даних) та Декодер (що виконує вилучення даних із модифікованого зображення). При цьому для початку роботи користувач виконує завантаження контейнера, а система його обробляє та вилучає стегоключі (якщо вони є). Після цього користувач ініціює процес декодування, під час якого об’єднуються криві, що були розбиті на етапі

кодування, та вилучаються двійкові дані. Наприкінці бінарна послідовність конвертується у текст та отримується секретне повідомлення.

Експериментальні дослідження демонструють швидкодію імплементованого методу при бітовому та патерновому режимах, а також відсотки збільшення стегоконтейнерів у порівнянні з оригінальним зображенням. Виявлено, що найбільшою мірою впливу на розмір є параметр точності, що застосовується при проведенні розрахунків нових точок при розриві кривої.

Дослідження щодо стійкості до афінних перетворень показують, що запропоновані Кінзерявим методи дійсно спроможні зберігати секретне повідомлення при застосуванні афінних атак. У даній роботі при 10 ітераціях застосування різних типів афінних перетворень стегоконтейнер зберігає близько 70% двійкових даних. При цьому деякі параметри закодування та декодування показали себе більш сприятливими для запровадження заявленої стійкості.

Практичне використання методу, що досліджується, тісно пов’язане з визначенням пріоритетності мети, що має досягатися (малі розміри стегоконтейнера або точність збереження оригінального повідомлення). У ході описаного дослідження раціональним приймається збільшення точності заради збереження повідомлення за умови використання контейнерів, що містять нетехнічні мотиви, скетчеві відображення, адже вони не визиватимуть підозр щодо їх розмірів і завжди матимуть достатню кількість кривих для поміщення секретного повідомлення.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Doncel V. R., Nikolaidis N., Pitas I. An Optimal Detector Structure for the Fourier Descriptors Domain Watermarking of 2D Vector Graphics in IEEE Transactions on Visualization and Computer Graphics. Sept.-Oct. 2007. vol. 13. № 5. P. 851-863. DOI: 10.1109/TVCG.2007.1050.
2. Reversible Data-Hiding Scheme for 2-D Vector Maps Based on Difference Expansion / Wang X., Shao C., Xu X., Niu X. in IEEE Transactions on Information Forensics and Security. Sept. 2007. vol. 2, № 3. P. 311-320. DOI: 10.1109/TIFS.2007.902677.
3. Wu D., Wang G., Gao X. Reversible Watermarking of SVG Graphics, 2009 WRI International Conference on Communications and Mobile Computing, 2009. P. 385-390, DOI: 10.1109/CMC.2009.86.
4. Reversible Data Hiding in Encrypted 2D Vector Graphics Based on Reversible Mapping Model for Real Numbers / Peng F., Lin Z. -X., Zhang X., Long M. in IEEE Transactions on Information Forensics and Security. Sept. 2019. vol. 14. № 9. P. 2400-2411. DOI: 10.1109/TIFS.2019.2899520.
5. Digital Geographical Map Watermarking Using Polyline Interpolation / Park K.T., Kim K.I., Kang H.I., Han S.S. in Advances in Multimedia Information Processing - PCM 2002. PCM 2002. Lecture Notes in Computer Science, vol 2532. Springer, Berlin, Heidelberg, DOI: 10.1007/3-540-36228-2_8
6. Illustration watermarks for vector graphics / Sonnet H., Isenberg T., Dittmann J., Strothotte T. in 11th Pacific Conference on Computer Graphics and Applications. 2003. P. 73-82. DOI: 10.1109/PCCGA.2003.1238249

7. Карпінець В.В., Яремчук Ю.Є. Аналіз стійкості до зловмисних атак методу вбудовування цифрових водяних знаків у векторні зображення. Вісн. Вінниц. політехн. Інституту № 4. 2011. С. 154-159.
8. Карпінець В.В., Яремчук Ю.Є. Зменшення відхилень координат точок внаслідок вбудовування цифрових водяних знаків у векторні зображення. Прав., нормат. та метрол. забезп. системи захисту інформації в Україні. Вип. 2, 2010. С. 69-78.
9. Steganographic Method of Bitwise Information Hiding in Point-Defined Curves of Vector Images / Kinzeryavyy O., Kinzeriava I., Olenyuk A., Sulkowsky K. in Advances in Computer Science for Engineering and Education. ICCSEEA 2018. Advances in Intelligent Systems and Computing, vol 754. Springer, Cham. DOI: 10.1007/978-3-319-91008-6_48
10. Kinzeryavyy, O. Steganographic methods for hiding data into vector images that are resistant to active attacks based on affine transformations. Thesis. 2015.
11. Pharr M., Wenzel J., Humphreys G. Physically Based Rendering: From Theory To Implementation. 2021. URL: https://pbr-book.org/3ed-2018/Shapes/Curves# (дата звернення: 09.09.2022).
12. Kuznetsov A., Kononchenko A. Software Implementation of Data Hiding in Vector Images, 2021 IEEE 3rd Ukraine Conference on Electrical and Computer Engineering (UKRCON), 2021. P. 572-577. DOI: 10.1109/UKRCON53503.2021.9575240.
13. Akhiezer N. I. Theory of approximation / пер. на англ. Hyman J. C., Ungar F. Mineola, NY : Dover Publications. 2003. P. 30–31.
14. De Casteljau's Algorithm and Bézier Curves : веб-сайт. URL: http://www.malinc.se/m/DeCasteljauAndBezier.php (дата звернення: 02.09.2022).
15. Higham J. N. Accuracy and stability of numerical algorithms. Philadelphia: Society for Industrial and Applied Mathematics, 1996. 690 P.

16. Inkscape : веб-сайт. URL: https://inkscape.org (дата звернення: 18.04.2022).
17. Kuznetsov A., Kononchenko A., Kryvinska N. Hiding data in vector images: software implementation and experimental research. Multimed Tools Appl. 2022. DOI: 10.1007/s11042-022-13829-5
18. Kuznetsov A., Kononchenko A. Data Hiding in Vector Images, 2021 IEEE 16th International Conference on Computer Sciences and Information Technologies (CSIT). 2021. P. 171-176, DOI: 10.1109/CSIT52700.2021.9648652.
19. Foundations of Physically Based Modeling and Animation : веб-сайт. URL: https://people.cs.clemson.edu/~dhouse/courses/401/notes/affines-matrices.pdf (дата звернення: 22.09.2022).
20. Coste A. Image Processing: Affine Transformation, Landmarks registration, Non linear Warping. DOI: 10.13140/RG.2.2.13653.27360.
21. Kuznetsov A., Kononchenko A. Message Concealing in Vector Images, 2021 11th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS), 2021. P. 465-470, DOI: 10.1109/IDAACS53288.2021.9660899.
22. Компьютерная стеганография, теория и практика (російською). Київ: МК-Прес, 2006. 288 с.
23. Кузнецов О., Євсеєв С., Король О. Стеганографія: навч. Посіб., Харків: ХНЕУ, 2011. 265 с.
24. Lashkari A. H. Magic Hexagon Image Steganography Evaluator. 2011. URL: https://www.academia.edu/1014807/Magic_Hexagon_Image_Steganography_Evaluator (дата звернення: 15.09.2022).
25. Grībermans D., Jeršovs A., Rusakovs P. Development of Requirements Specification. December 2016. vol. 20. P. 40–48, ISSN 2255-8691. DOI: 10.1515/acss-2016-0014.
26. ITU-R. Methodology for the subjective assessment of the quality of television pictures, Radiocommunication Sector of ITU. 2012.

27. An Evaluation Scheme for Steganalysis-proof Ability of Steganographic Algorithms / Luo G., Sun X. -M., Xiang L. -Y., Huang J. -W. in Third International Conference on Intelligent Information Hiding and Multimedia Signal Processing (IIH-MSP 2007). 2007. PP. 126-129. DOI: 10.1109/IIH-MSP.2007.85.
28. Digital Watermarking and Steganography, 2nd ed. / Cox I. J. та ін. Burlington: Morgan Kaufmann, 2008. 624 P.
29. Vector Image Files : веб-сайт. URL: https://fileinfo.com/filetypes/vector_image (дата звернення: 08.08.2022).
30. Scalable Vector Graphics (SVG) 2. W3C Candidate Recommendation. W3C: веб-сайт. URL: https://www.w3.org/TR/SVG2/intro.html#AboutSVG (дата звернення: 08.08.2022).
31. SVG (basic support) : веб-сайт. URL: https://caniuse.com/?search=svg (дата звернення: 15.07.2022).
32. Scalable Vector Graphics (SVG) 2. Document Structure. W3C : веб-сайт. URL: https://www.w3.org/TR/SVG2/struct.html (дата звернення: 08.08.2022).
33. Scalable Vector Graphics (SVG) 2. Chapter 9: Paths. W3C: веб-сайт. URL: https://www.w3.org/TR/SVG2/paths.html (дата звернення: 08.08.2022).
34. PYPL PopularitY of Programming Language : веб-сайт. URL: https://pypl.github.io/PYPL.html (дата звернення: 01.11.2022).
35. PYPL PopularitY of Programming Language : веб-сайт. URL: https://pypl.github.io/PYPL.html (дата звернення: 01.11.2022).
36. What is Java technology and why do I need it? Oracle : веб-сайт. URL: https://www.java.com/en/download/help/whatis_java.html (дата звернення: 23.06.2022).
37. Horstmann C. S. Core Java. Volume I – Fundamentals. Boston: Oracle Press. 2021. 909 P.

38. Schildt H. Java 8. Ninth Edition. The complete Reference. New York : Oracle Press. 2015. 1312 P.
39. JavaFX, Gluon OpenJFX : веб-сайт. URL: https://openjfx.io (дата звернення: 02.05.2022).
40. Scene Builder. Gluon : веб-сайт. URL: https://gluonhq.com/products/scene-builder. (дата звернення: 02.05.2022).
41. Top IDE index : веб-сайт. URL: https://pypl.github.io/IDE.html (дата звернення: 01.11.2022).
42. IntelliJ IDEA. JetBrains : веб-сайт. URL: https://www.jetbrains.com/idea (дата звернення: 01.11.2022).
43. Java Platform, Standard Edition & Java Development Kit : веб-сайт. URL: https://docs.oracle.com/javase/9/docs/api/overview-summary.html (дата звернення: 01.06.2022).
44. JavaFX WebView and WebEngine. Java Documentation : веб-сайт. URL: https://o7planning.org/11151/javafx-webview (дата звернення: 01.09.2022).
45. Class WebEngine. Java Documentation : веб-сайт. URL: https://openjfx.io/javadoc/11/javafx.web/javafx/scene/web/WebEngine.html (дата звернення: 01.09.2022).
46. AntiDupl.NET Description : веб-сайт. URL: https://ermig1979.github.io/AntiDupl/data/help/english/index.html?page=intro.html (дата звернення: 25.10.2022).

ДОДАТОК А

Лістинг функцій приховування даних

1) Бітове приховування

```
public List<String> encode(Message message) {
   message.setBinaryMessage(this.maxBitAmount);
   List<String> input = message.getBinaryMessage();
   int indexOfBlock = 0;
   List<List<Boolean>> resultMarks = new ArrayList<>();
   List<EncodedCurve> encodedCurves = new ArrayList<>();
   for (List<Curve> listOfCurves : this.image.getCurves()) {
       for (Curve curve : listOfCurves) {
           List<List<Segment>> result = new ArrayList<>();
           for (Segment segment : curve.getSegments()) {
               List<Segment> newSegments = new ArrayList<>();
               List<Segment> encodedSegments = new ArrayList<>();
               if (segment.getValid() && indexOfBlock < input.size()) {
                  int indexOfSegment = 0;
                  String[] currBlock = input.get(indexOfBlock).split("");
                  List<List<Segment>> splittedSegments = new ArrayList<>();
double k = this.parameter;
                  for (int i = 0; i < currBlock.length; i++) {
                     if (Objects.equals(currBlock[i], "1")) {
                        if (indexOfSegment == 0) {
                            splittedSegments.add(segment.splitSegment(k));
                        } else {
                           splittedSegments.add(splittedSegments.get(
indexOfSegment - 1).get(1).splitSegment(k));
                        }
              splittedSegments.get(indexOfSegment).get(0).getPoints().remove(0);
encodedSegments.add(splittedSegments.get(indexOfSegment).get(0));
indexOfSegment++;
           }
```

```
                            k = BigDecimal.valueOf(k + this.parameter).setScale(6,
RoundingMode.HALF_UP).doubleValue();

                        }

                        splittedSegments.get(indexOfSegment -
1).get(1).getPoints().remove(0);

                        encodedSegments.add(splittedSegments.get(indexOfSegment -
1).get(1));

                        newSegments.addAll(encodedSegments);

                        indexOfBlock++;

                        flag = true;

                    } else {

                        if (segment.getLiteral().matches("[Cc]")) {

                            segment.getPoints().remove(0);

                        }

                        newSegments.add(segment);

                    }

                    result.add(newSegments);

                }

                encodedCurves.add(new EncodedCurve(curve.getLiteral(),
curve.getStartPoint(), transformSetSegmentToEncodedCurve(result)));

            }

        }

        List<String> parsed = getReadyForParsingIntoSvg(encodedCurves);

        return parsed;

    }
```

2) Патернове приховування

```
public List<String> encode(Message message) {

        message.setBinaryMessage(this.maxBitAmount);
        List<String> input = message.getBinaryMessage();

        int indexOfBlock = 0;
        List<EncodedCurve> encodedCurves = new ArrayList<>();
        List<List<Double>> finalParameters = new ArrayList<>();
        for (List<Curve> listOfCurves : this.image.getCurves()) {
            for (Curve curve : listOfCurves)
                List<Double> finalParametersOfSegment = new ArrayList<Double>();
                List<List<Segment>> resultSegments = new ArrayList<>();

                for (Segment segment : curve.getSegments()) {
                    double finalParameter = 0;
                    List<Segment> encodedSegments = new ArrayList<>();
```

```
                    if (segment.getValid() && indexOfBlock < input.size()) {
                        System.out.println("Input block is " +
input.get(indexOfBlock));
                        List<String> inputValue =
trimString(input.get(indexOfBlock), patternLength);
                        int indexOfSegment = 0;
                        List<List<Segment>> splitSegments = new ArrayList<>();
                        double k = startStep;
                        for (int i = 0; i < inputValue.size(); i++) {
                            PatternTableRecord currentRecord =
patternTable.find(inputValue.get(i));
                            if (currentRecord != null) { // if exist
                                if (indexOfSegment == 0) {
splitSegments.add(segment.splitSegment(BigDecimal.valueOf(k +
currentRecord.step).setScale(6, RoundingMode.HALF_UP).doubleValue()));
                                } else
{splitSegments.add(splitSegments.get(indexOfSegment -
1).get(1).splitSegment(BigDecimal.valueOf(k + currentRecord.step).setScale(6,
RoundingMode.HALF_UP).doubleValue()));
                                }

splitSegments.get(indexOfSegment).get(0).getPoints().remove(0);

encodedSegments.add(splitSegments.get(indexOfSegment).get(0));
                                indexOfSegment++;
                                k = BigDecimal.valueOf(k +
currentRecord.step).setScale(6, RoundingMode.HALF_UP).doubleValue();
                            }
                        }
                        finalParameter = k;
                        splitSegments.get(indexOfSegment -
1).get(1).getPoints().remove(0);
                        encodedSegments.add(splitSegments.get(indexOfSegment -
1).get(1));
                        indexOfBlock++;
                    } else {
                        if (segment.getLiteral().matches("[Cc]")) {
                            segment.getPoints().remove(0);
                        }
                        encodedSegments.add(segment);
                    }
                    finalParametersOfSegment.add(finalParameter);
                    resultSegments.add(encodedSegments);

                }
                finalParameters.add(finalParametersOfSegment);
                encodedCurves.add(new EncodedCurve(curve.getLiteral(),
curve.getStartPoint(), transformSetSegmentToEncodedCurve(resultSegments)));

            }

        }

        this.finalParameters = finalParameters;

        List<String> parsed = getReadyForParsingIntoSvg(encodedCurves);
        return getReadyForParsingIntoSvg(encodedCurves);
    }
```

ДОДАТОК Б

Лістинг функцій вилучення даних

1) Бітове вилучення

```
public Message decode(double error) {

        List<String> decodedBinaryBlocks = new ArrayList<>();

        for (int i = 0; i < this.image.getCurves().size(); i++) {

            int indexOfFlag = 0;

            for (Curve curve : this.image.getCurves().get(i)) {

                if (curve.getSegments().size() > 1) {

                    StringBuilder output = new StringBuilder();

                    double lastEncodedValue =
BigDecimal.valueOf((this.maxBitAmount + 1) *
this.parameter).setScale(this.precision, RoundingMode.HALF_UP).doubleValue();

                    List<Segment> copy = curve.getSegments().subList(0,
curve.getSegments().size());

                    for (double t = lastEncodedValue - parameter; t >= 0; t -=
parameter) {



                        int indexOfSegment = copy.size() - 1;

                        if (copy.get(indexOfSegment).getLiteral().matches("[Cc]"))
{

                            t = BigDecimal.valueOf(t).setScale(this.precision,
RoundingMode.HALF_UP).doubleValue();

                            if (indexOfSegment != 0 && t != 0) {

                                List<Segment> lastTwoSegments =
List.of(copy.get(indexOfSegment - 1), copy.get(indexOfSegment));

                                Point p02_2 =
lastTwoSegments.get(1).getPoints().get(0).getAdditionalPoint(t,
lastTwoSegments.get(1).getPoints().get(1));

                                Point p11 =
lastTwoSegments.get(1).getPoints().get(1).getAdditionalPoint(t,
lastTwoSegments.get(1).getPoints().get(2));

                                Point p02_3 =
lastTwoSegments.get(0).getPoints().get(1).getCurvePoint(t, p11);

                                if (

                                        p02_2.isDifferenceSmaller(p02_3, error) ||
```

```
lastTwoSegments.get(1).getPoints().get(2).isDifferenceSmaller(p02_2, error) ||

lastTwoSegments.get(1).getPoints().get(2).isDifferenceSmaller(p02_3, error)
                                    ) {
                                        System.out.println("Got '1'");
                                        output.append("1");
                                        Segment merged =
lastTwoSegments.get(0).mergeSegments(t, lastTwoSegments.get(1));
                                        copy.remove(indexOfSegment);
                                        copy.set(indexOfSegment - 1, merged);
                                        indexOfSegment--;
                                    } else {
                                        output.append("0");
                                    }
                                } else {
                                    if (output.length() < this.maxBitAmount) {
                                        output.append("0");
                                    }
                                }
                            } else {
                                copy.remove(copy.size() - 1);
                            }
                        }

                        if (output.length() < this.maxBitAmount) {
                            while (output.length() < this.maxBitAmount) {
                                output.append("0");
                            }
                        }
                        output.reverse();
                        decodedBinaryBlocks.add(output.toString());
                    }
                    indexOfFlag++;
                }
```

```
        }
        Message decodedMessage = new Message();
        decodedMessage.setText(decodedBinaryBlocks);
        decodedMessage.setBinaryMessage(decodedBinaryBlocks);
        return decodedMessage;
    }
```

2) Патернове вилучення

```
public Message decode(double error) {
        List<String> decodedBinaryBlocks = new ArrayList<>();
        System.out.println("Stego keys list: " + this.finalParameters);
        for (int i = 0; i < this.image.getCurves().size(); i++) {
            List<Curve> listOfCurves = this.image.getCurves().get(i);
            for (int j = 0; j < listOfCurves.size(); j++) {
                if (listOfCurves.get(j).getSegments().size() > maxBitAmount /
patternLength) {
                    StringBuilder output = new StringBuilder();
                    List<Segment> merged =
listOfCurves.get(j).getSegments().subList(0,
listOfCurves.get(j).getSegments().size());
                    double t = finalParameters.get(i).get(j), parameter = 0;
                    boolean flag = true;
                    int indexOfSegment = listOfCurves.get(j).getSegments().size()
- 1;
                    if (t != 0.0) {
                        do {
                            t = BigDecimal.valueOf(t).setScale(this.precision,
RoundingMode.HALF_UP).doubleValue();
                            List<Segment> lastTwoSegments = new ArrayList<>();
                            if (indexOfSegment == 1) {
                                for (int k = 0; k <
this.patternTable.getRecords().size(); k++) {
                                    double nextParameter =
BigDecimal.valueOf(parameter -
this.patternTable.getRecords().get(k).step).setScale(this.precision,
RoundingMode.HALF_UP).doubleValue();
                                    if (nextParameter == startStep) {
                                        output.insert(0,
this.patternTable.findStep(this.patternTable.getRecords().get(k).step).pattern);
```

```
                                    flag = false;
                                }
                            }
                        } else {
                            if (indexOfSegment ==
listOfCurves.get(j).getSegments().size() - 1) {
                                Segment mergedLastTwo =
merged.get(indexOfSegment - 1).mergeSegments(t, merged.get(indexOfSegment));
                                merged.remove(indexOfSegment);
                                merged.set(indexOfSegment - 1, mergedLastTwo);
                                indexOfSegment--;
                            }
lastTwoSegments.addAll(List.of(merged.get(merged.size() - 2),
merged.get(merged.size() - 1)));
                            for (int k = 0; k <
this.patternTable.getRecords().size(); k++) {
                                parameter = BigDecimal.valueOf(t -
this.patternTable.getRecords().get(k).step).setScale(this.precision,
RoundingMode.HALF_UP).doubleValue();
                                Point p02_2 =
lastTwoSegments.get(1).getPoints().get(0).getAdditionalPoint(parameter,
lastTwoSegments.get(1).getPoints().get(1));
                                Point p11 =
lastTwoSegments.get(1).getPoints().get(1).getAdditionalPoint(parameter,
lastTwoSegments.get(1).getPoints().get(2));
                                Point p02_3 =
lastTwoSegments.get(0).getPoints().get(1).getCurvePoint(parameter, p11);
                                p02_2.roundPoint(this.precision);
                                p11.roundPoint(this.precision);
                                p02_3.roundPoint(this.precision);

                                if (
                                        p02_2.isDifferenceSmaller(p02_3,
error) ||
lastTwoSegments.get(1).getPoints().get(2).isDifferenceSmaller(p02_2, error) ||
lastTwoSegments.get(1).getPoints().get(2).isDifferenceSmaller(p02_3, error)
                                ) {
                                    output.insert(0,
this.patternTable.findStep(this.patternTable.getRecords().get(k).step).pattern);
```

```
                                        lastTwoSegments.clear();
                                        Segment mergedSegments =
merged.get(merged.size() - 2).mergeSegments(parameter, merged.get(merged.size() -
1));
                                        merged.remove(merged.size() - 2);
                                        merged.set(merged.size() - 1,
mergedSegments);
                                        t = parameter;
                                        indexOfSegment = merged.size();
                                        break;
                                    }
                                    if (k == this.patternTable.getRecords().size()
- 1) {
                                        flag = false;
                                    }
                                }
                            }
                        } while (flag);
                        if(output.length() < this.maxBitAmount){
                            while (output.length() < this.maxBitAmount) {
                                output.append("0");
                            }
                        }
                        decodedBinaryBlocks.add(output.toString());
                    }
                } else {
                    System.out.println("This curve was not encoded");
                }
            }
        }
        Message decodedMessage = new Message();
        decodedMessage.setText(decodedBinaryBlocks);
        decodedMessage.setBinaryMessage(decodedBinaryBlocks);
        return decodedMessage;
    }
```

ДОДАТОК В

Експериментальні дослідження щодо стійкості до афінних перетворень

Таблиця В.1 – Набір експериментальних параметрів

Назва	Похибка	Точність	Кількість біт у кривій
set1	0.0002	5	40
set2	0.0004	5	40
set3	0.0002	6	40
set4	0.0004	6	40
set5	0.0002	5	64
set6	0.0004	5	64
set7	0.0002	6	64
set8	0.0004	6	64
set9	0.0002	5	80
set10	0.0004	5	80
set11	0.0002	6	80
set12	0.0004	6	80

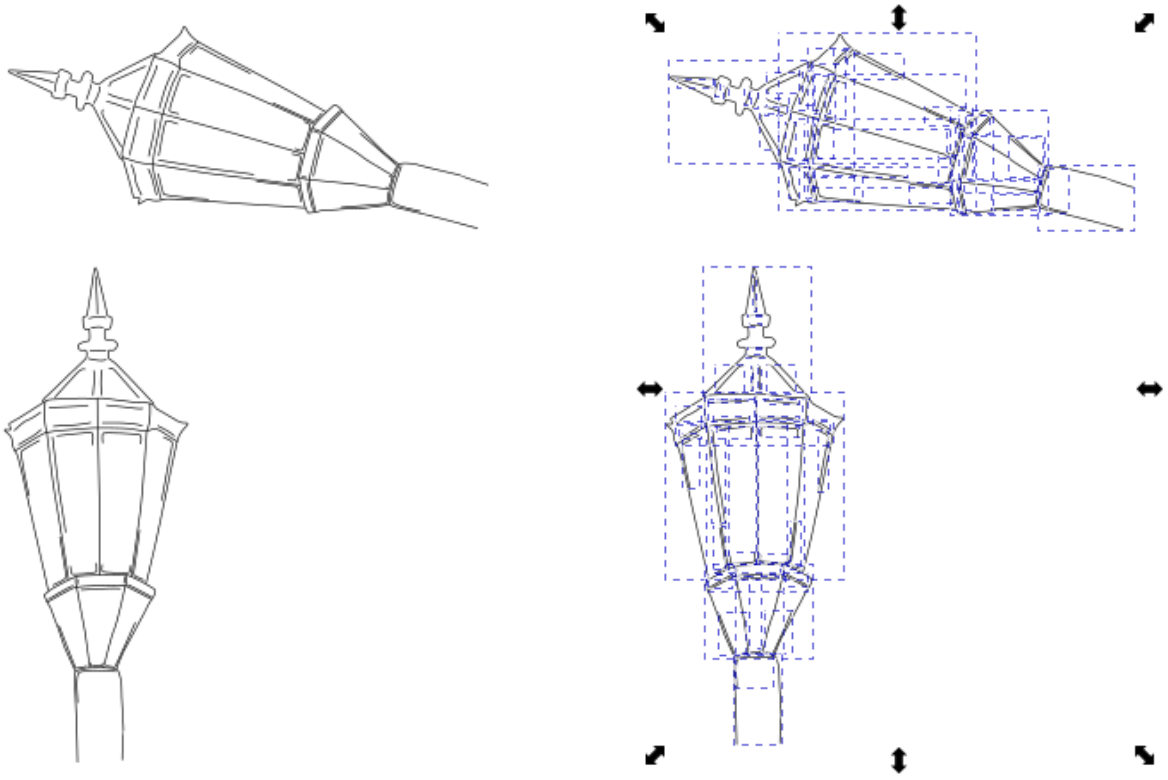
Таблиця В.2 – Типи атак для експериментальних досліджень

Афінне перетворення	Коефіцієнти	Кількість ітерацій
Перенесення	$(c, f) \in [-250, 0) \cup (0, 250]$	10
Поворот	$\alpha = 1$	
X-зсув	$b = 0,01$	
Y-зсув	$d = 0,01$	
Зменшення	$a = e = 0,99$	
Збільшення	$a = e = 1,01$	

Продовження таблиці В.2 – Типи атак для експериментальних досліджень

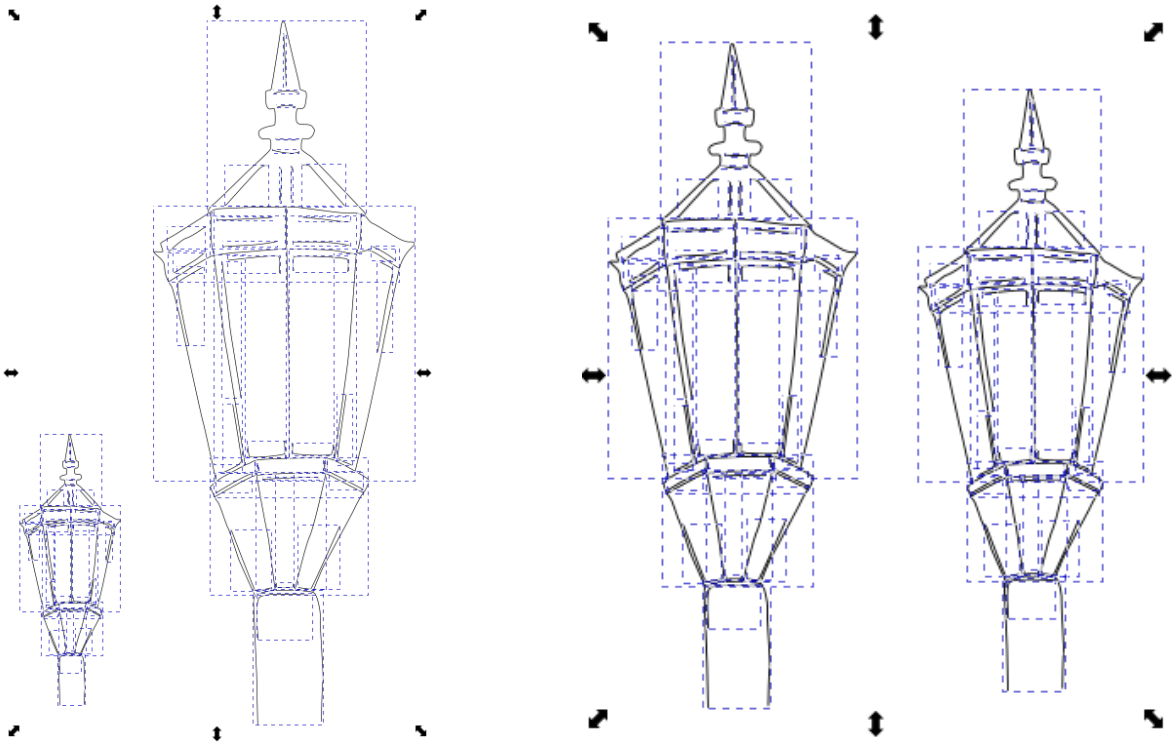
Майже афінне	$\alpha = 1$ $b = -\sin(\alpha)$ $d = \sin(\alpha)$ $a = \cos(\alpha)$	
	$e = -\cos(\alpha)$	
	$n_1 = n_4 = n_6 = -0,0001$	
	$n_2 = n_3 = n_5 = 0,0001$	

1) Вигляд проміжних контейнерів



а) б)

Рисунок В.1 – Проміжний вигляд стегоконтейнера при перетворенні повороту



а) б)

Рисунок В.2 – Проміжний вигляд стегоконтейнера при перетворенні масштабування для а) збільшення і б) стиснення

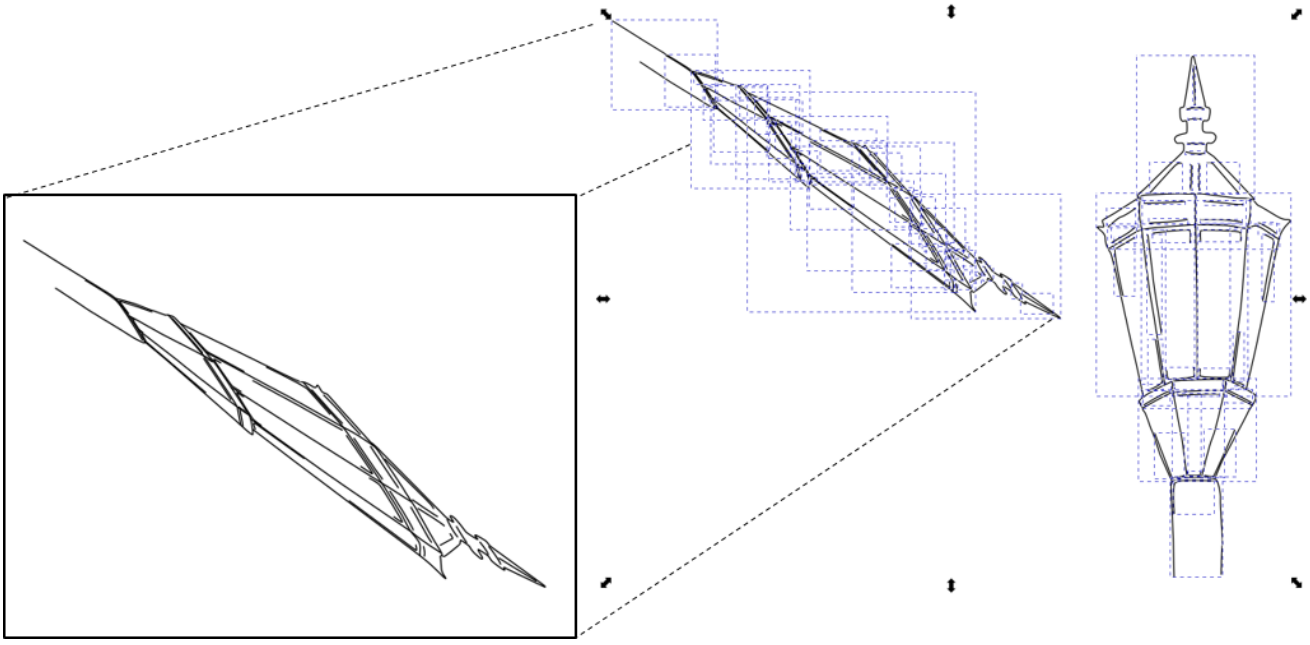
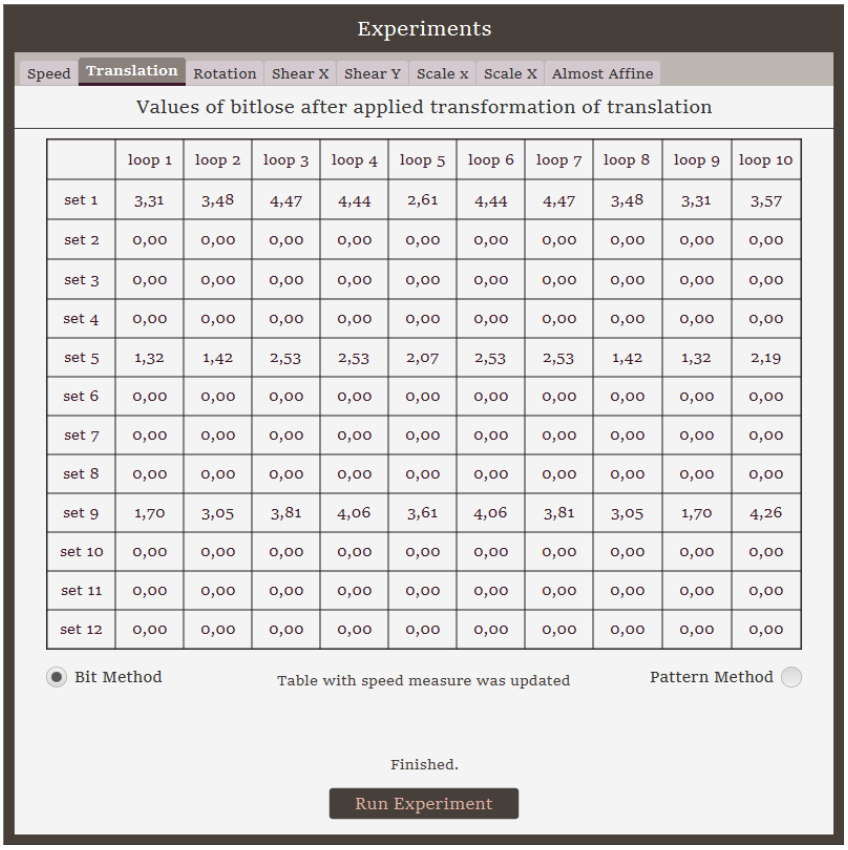


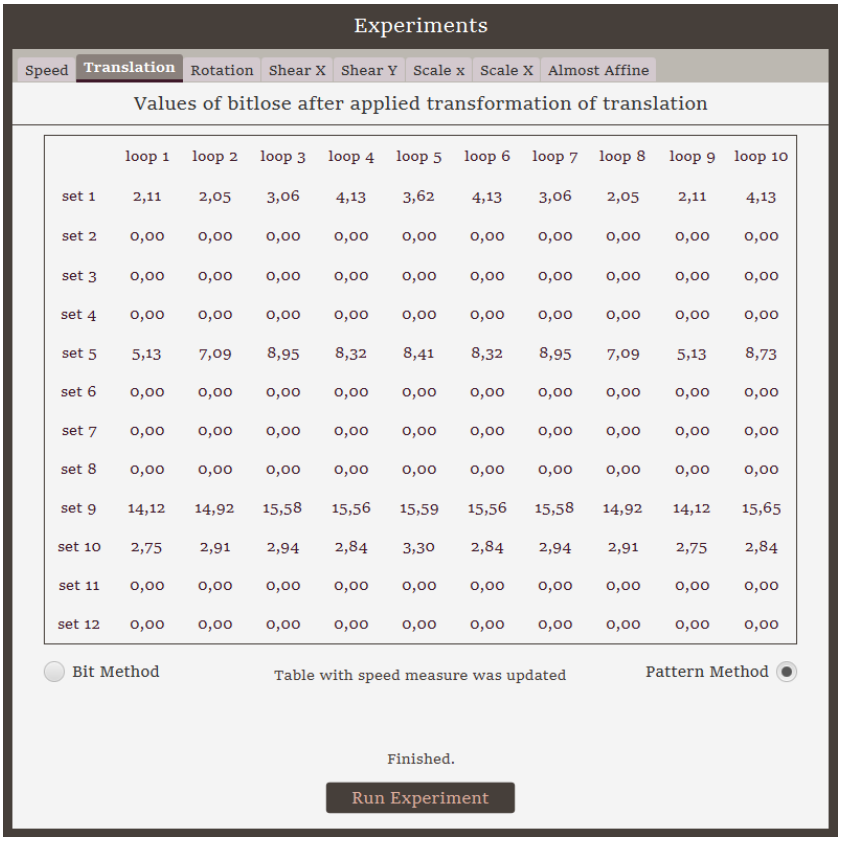
Рисунок В.3 – Проміжний вигляд стегоконтейнера при майже афінному перетворенні

2) Табличне відображення відсотку хибно вилучених біт



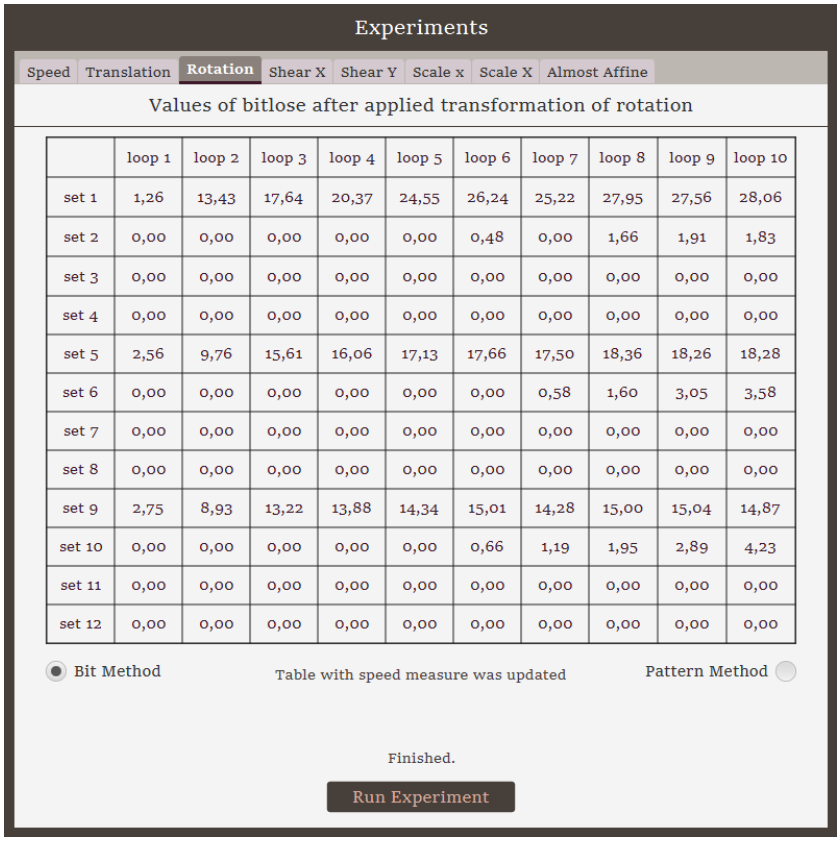
	loop 1	loop 2	loop 3	loop 4	loop 5	loop 6	loop 7	loop 8	loop 9	loop 10
set 1	3,31	3,48	4,47	4,44	2,61	4,44	4,47	3,48	3,31	3,57
set 2	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
set 3	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
set 4	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
set 5	1,32	1,42	2,53	2,53	2,07	2,53	2,53	1,42	1,32	2,19
set 6	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
set 7	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
set 8	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
set 9	1,70	3,05	3,81	4,06	3,61	4,06	3,81	3,05	1,70	4,26
set 10	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
set 11	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
set 12	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00

Рисунок В.4 – Показники втрати біт при переносі бітовим методом



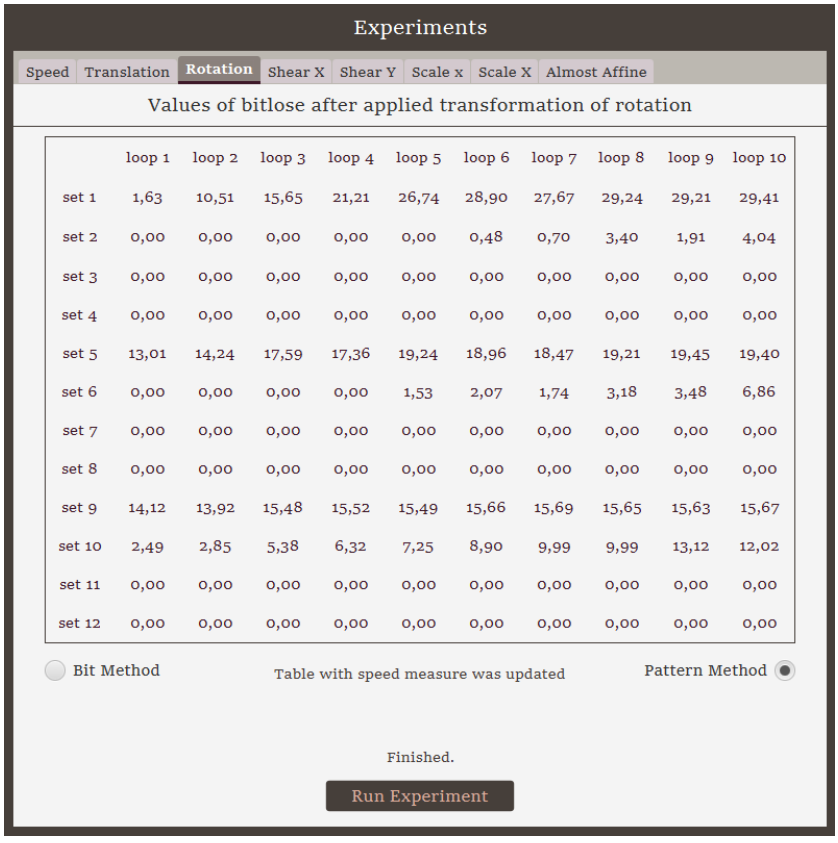
	loop 1	loop 2	loop 3	loop 4	loop 5	loop 6	loop 7	loop 8	loop 9	loop 10
set 1	2,11	2,05	3,06	4,13	3,62	4,13	3,06	2,05	2,11	4,13
set 2	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
set 3	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
set 4	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
set 5	5,13	7,09	8,95	8,32	8,41	8,32	8,95	7,09	5,13	8,73
set 6	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
set 7	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
set 8	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
set 9	14,12	14,92	15,58	15,56	15,59	15,56	15,58	14,92	14,12	15,65
set 10	2,75	2,91	2,94	2,84	3,30	2,84	2,94	2,91	2,75	2,84
set 11	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
set 12	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00

Рисунок В.5 – Показники втрати біт при переносі патерновим методом



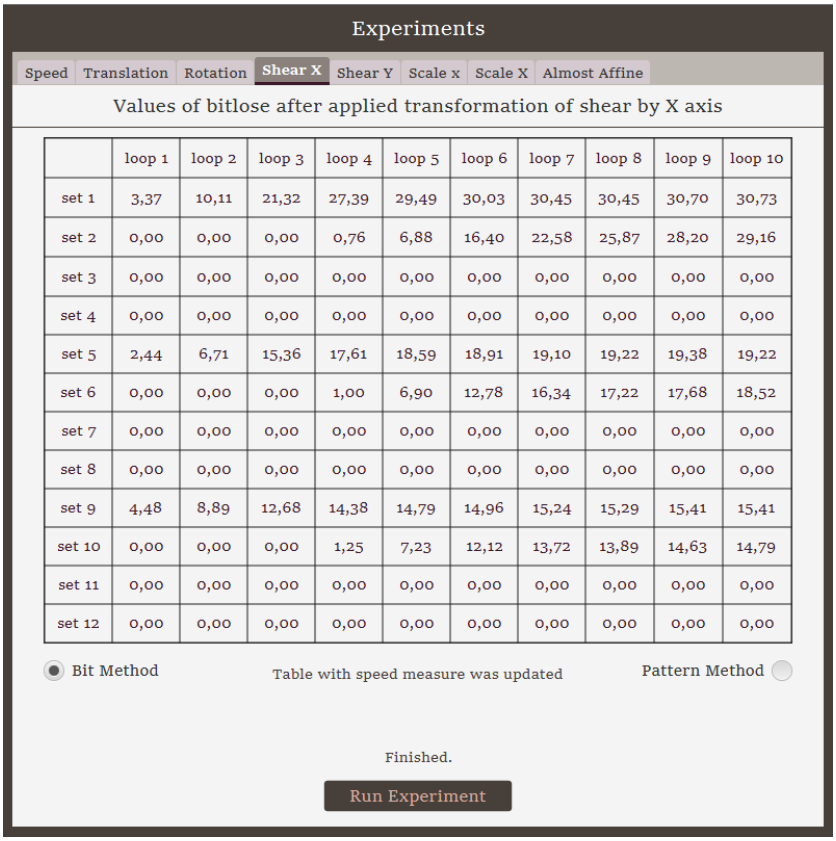
	loop 1	loop 2	loop 3	loop 4	loop 5	loop 6	loop 7	loop 8	loop 9	loop 10
set 1	1,26	13,43	17,64	20,37	24,55	26,24	25,22	27,95	27,56	28,06
set 2	0,00	0,00	0,00	0,00	0,00	0,48	0,00	1,66	1,91	1,83
set 3	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
set 4	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
set 5	2,56	9,76	15,61	16,06	17,13	17,66	17,50	18,36	18,26	18,28
set 6	0,00	0,00	0,00	0,00	0,00	0,00	0,58	1,60	3,05	3,58
set 7	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
set 8	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
set 9	2,75	8,93	13,22	13,88	14,34	15,01	14,28	15,00	15,04	14,87
set 10	0,00	0,00	0,00	0,00	0,00	0,66	1,19	1,95	2,89	4,23
set 11	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
set 12	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00

Рисунок В.6 – Показники втрати біт при повороті бітовим методом



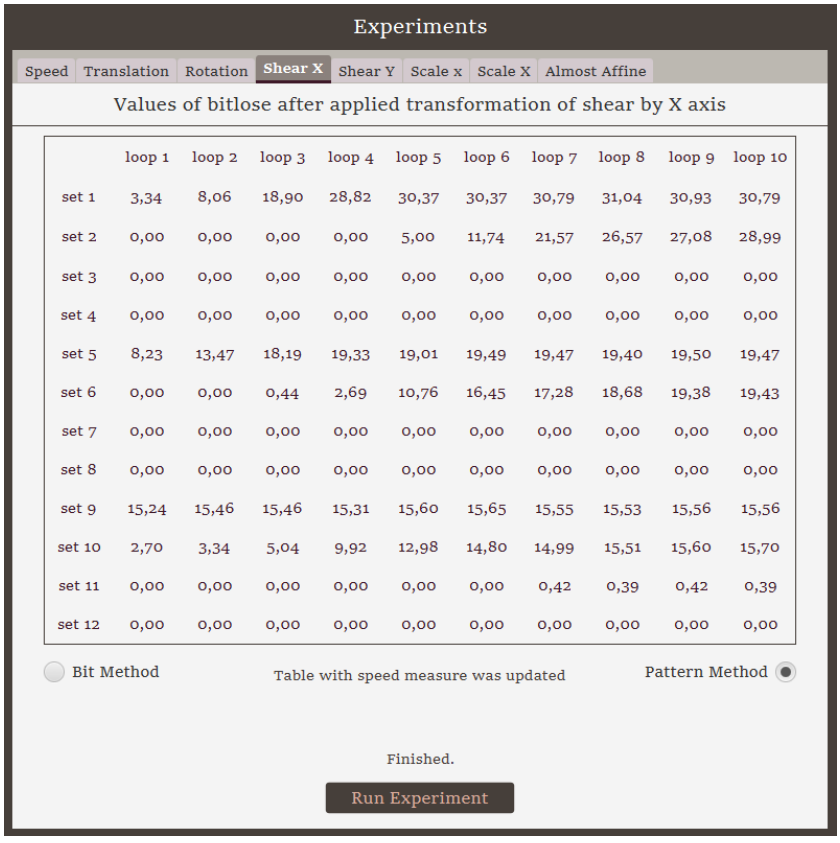
	loop 1	loop 2	loop 3	loop 4	loop 5	loop 6	loop 7	loop 8	loop 9	loop 10
set 1	1,63	10,51	15,65	21,21	26,74	28,90	27,67	29,24	29,21	29,41
set 2	0,00	0,00	0,00	0,00	0,00	0,48	0,70	3,40	1,91	4,04
set 3	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
set 4	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
set 5	13,01	14,24	17,59	17,36	19,24	18,96	18,47	19,21	19,45	19,40
set 6	0,00	0,00	0,00	0,00	1,53	2,07	1,74	3,18	3,48	6,86
set 7	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
set 8	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
set 9	14,12	13,92	15,48	15,52	15,49	15,66	15,69	15,65	15,63	15,67
set 10	2,49	2,85	5,38	6,32	7,25	8,90	9,99	9,99	13,12	12,02
set 11	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
set 12	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00

Рисунок В.7 – Показники втрати біт при повороті патерновим методом



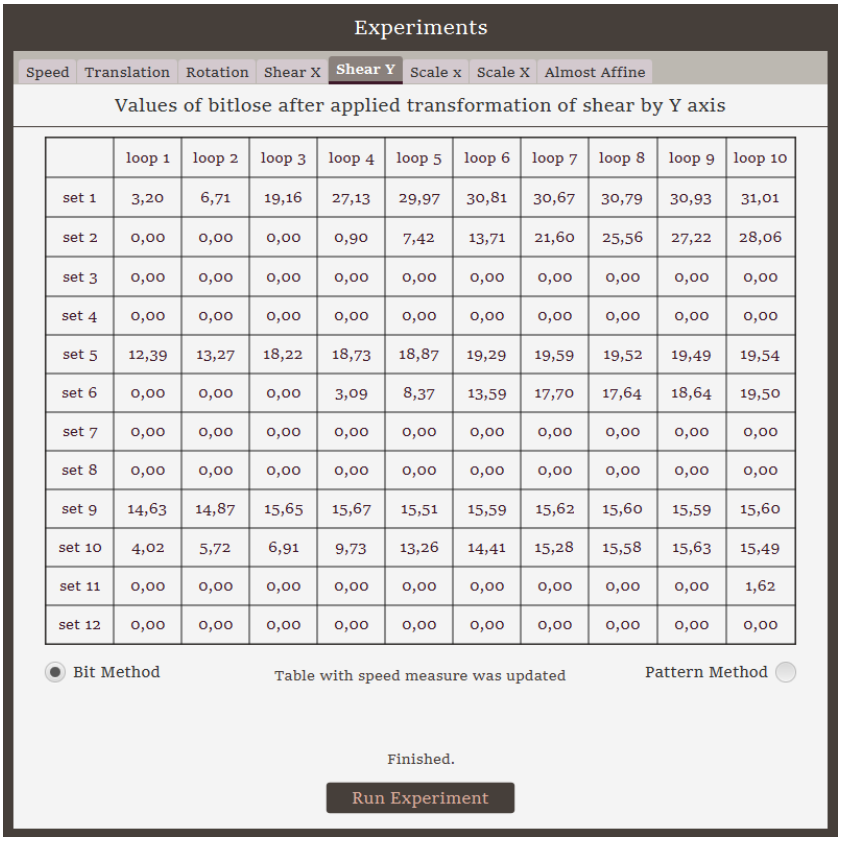
	loop 1	loop 2	loop 3	loop 4	loop 5	loop 6	loop 7	loop 8	loop 9	loop 10
set 1	3,37	10,11	21,32	27,39	29,49	30,03	30,45	30,45	30,70	30,73
set 2	0,00	0,00	0,00	0,76	6,88	16,40	22,58	25,87	28,20	29,16
set 3	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
set 4	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
set 5	2,44	6,71	15,36	17,61	18,59	18,91	19,10	19,22	19,38	19,22
set 6	0,00	0,00	0,00	1,00	6,90	12,78	16,34	17,22	17,68	18,52
set 7	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
set 8	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
set 9	4,48	8,89	12,68	14,38	14,79	14,96	15,24	15,29	15,41	15,41
set 10	0,00	0,00	0,00	1,25	7,23	12,12	13,72	13,89	14,63	14,79
set 11	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
set 12	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00

Рисунок В.8 – Показники втрати біт при X-зсуві бітовим методом



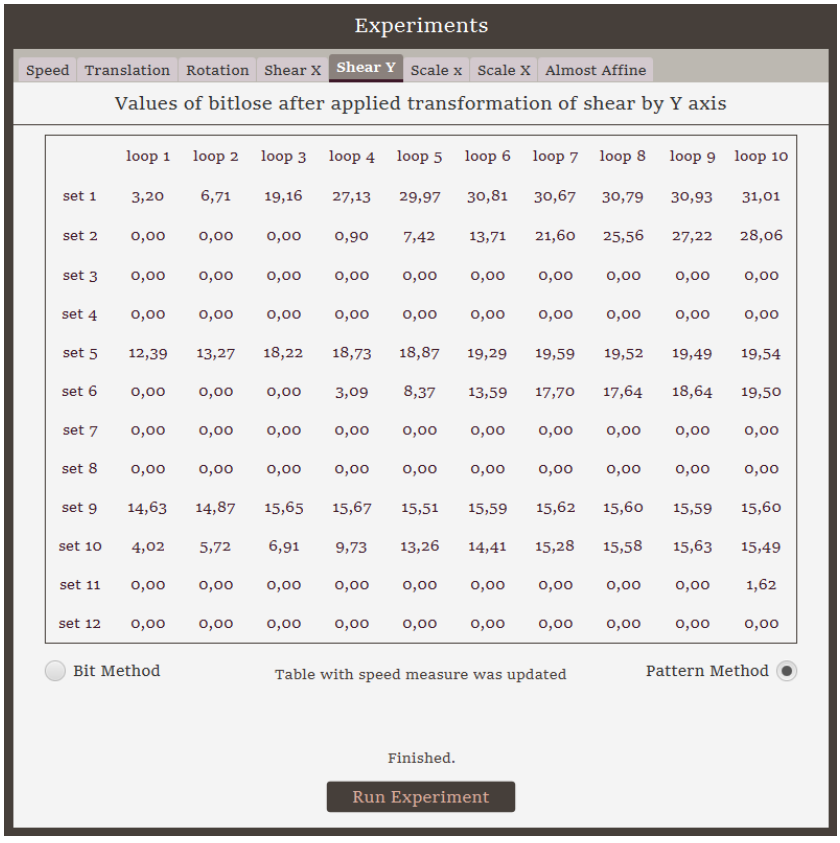
	loop 1	loop 2	loop 3	loop 4	loop 5	loop 6	loop 7	loop 8	loop 9	loop 10
set 1	3,34	8,06	18,90	28,82	30,37	30,37	30,79	31,04	30,93	30,79
set 2	0,00	0,00	0,00	0,00	5,00	11,74	21,57	26,57	27,08	28,99
set 3	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
set 4	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
set 5	8,23	13,47	18,19	19,33	19,01	19,49	19,47	19,40	19,50	19,47
set 6	0,00	0,00	0,44	2,69	10,76	16,45	17,28	18,68	19,38	19,43
set 7	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
set 8	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
set 9	15,24	15,46	15,46	15,31	15,60	15,65	15,55	15,53	15,56	15,56
set 10	2,70	3,34	5,04	9,92	12,98	14,80	14,99	15,51	15,60	15,70
set 11	0,00	0,00	0,00	0,00	0,00	0,00	0,42	0,39	0,42	0,39
set 12	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00

Рисунок В.9 – Показники втрати біт при X-зсуві патерновим методом



	loop 1	loop 2	loop 3	loop 4	loop 5	loop 6	loop 7	loop 8	loop 9	loop 10
set 1	3,20	6,71	19,16	27,13	29,97	30,81	30,67	30,79	30,93	31,01
set 2	0,00	0,00	0,00	0,90	7,42	13,71	21,60	25,56	27,22	28,06
set 3	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
set 4	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
set 5	12,39	13,27	18,22	18,73	18,87	19,29	19,59	19,52	19,49	19,54
set 6	0,00	0,00	0,00	3,09	8,37	13,59	17,70	17,64	18,64	19,50
set 7	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
set 8	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
set 9	14,63	14,87	15,65	15,67	15,51	15,59	15,62	15,60	15,59	15,60
set 10	4,02	5,72	6,91	9,73	13,26	14,41	15,28	15,58	15,63	15,49
set 11	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	1,62
set 12	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00

Рисунок В.10 – Показники втрати біт при Y-зсуві бітовим методом



	loop 1	loop 2	loop 3	loop 4	loop 5	loop 6	loop 7	loop 8	loop 9	loop 10
set 1	3,20	6,71	19,16	27,13	29,97	30,81	30,67	30,79	30,93	31,01
set 2	0,00	0,00	0,00	0,90	7,42	13,71	21,60	25,56	27,22	28,06
set 3	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
set 4	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
set 5	12,39	13,27	18,22	18,73	18,87	19,29	19,59	19,52	19,49	19,54
set 6	0,00	0,00	0,00	3,09	8,37	13,59	17,70	17,64	18,64	19,50
set 7	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
set 8	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
set 9	14,63	14,87	15,65	15,67	15,51	15,59	15,62	15,60	15,59	15,60
set 10	4,02	5,72	6,91	9,73	13,26	14,41	15,28	15,58	15,63	15,49
set 11	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	1,62
set 12	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00

Рисунок В.11 – Показники втрати біт при Y-зсуві патерновим методом

Experiments

Speed | Translation | Rotation | Shear X | Shear Y | Scale x | Scale X | Almost Affine

Values of bitlose after applied transformation of proportional scale for compression

	loop 1	loop 2	loop 3	loop 4	loop 5	loop 6	loop 7	loop 8	loop 9	loop 10
set 1	3,17	10,98	19,21	24,21	25,37	25,28	26,01	27,58	27,95	28,76
set 2	0,00	0,00	0,00	0,00	0,00	0,53	0,84	1,69	1,88	4,58
set 3	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
set 4	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
set 5	3,41	10,29	15,10	16,50	16,03	17,50	18,29	18,38	17,63	18,42
set 6	0,00	0,00	0,00	0,00	0,00	0,16	0,72	0,51	1,56	2,72
set 7	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
set 8	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
set 9	2,74	10,88	11,87	12,94	14,33	14,51	14,63	14,87	14,82	15,21
set 10	0,00	0,00	0,00	0,24	0,35	0,00	0,93	0,08	1,11	3,05
set 11	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
set 12	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00

Bit Method — Table with speed measure was updated — Pattern Method

Finished.

Run Experiment

Рисунок В.12 – Показники втрати біт при масштабуванні для зменшення бітовим методом

Experiments

Speed | Translation | Rotation | Shear X | Shear Y | Scale x | Scale X | Almost Affine

Values of bitlose after applied transformation of proportional scale for compression

	loop 1	loop 2	loop 3	loop 4	loop 5	loop 6	loop 7	loop 8	loop 9	loop 10
set 1	2,67	11,77	19,97	25,51	26,38	29,38	29,02	29,78	29,55	29,80
set 2	0,00	0,00	0,00	0,00	0,00	0,96	1,01	0,73	1,60	4,72
set 3	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
set 4	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
set 5	7,90	15,50	18,24	18,26	19,38	19,15	19,19	19,35	19,22	19,42
set 6	0,00	0,00	0,00	0,00	0,84	2,42	2,16	2,72	5,09	5,25
set 7	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
set 8	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
set 9	14,04	14,79	15,56	15,56	15,41	15,62	15,44	15,55	15,28	15,58
set 10	4,16	4,23	4,54	7,18	6,14	8,48	10,00	9,54	9,93	12,09
set 11	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
set 12	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00

Bit Method — Table with speed measure was updated — Pattern Method

Finished.

Run Experiment

Рисунок В.13 – Показники втрати біт при масштабуванні для зменшення патерновим методом

Experiments

Speed | Translation | Rotation | Shear X | Shear Y | Scale x | **Scale X** | Almost Affine

Values of bitlose after applied transformation of proportional scale for extension

	loop 1	loop 2	loop 3	loop 4	loop 5	loop 6	loop 7	loop 8	loop 9	loop 10
set 1	4,44	13,31	23,65	25,81	27,98	29,27	29,21	29,97	29,94	30,25
set 2	0,00	0,00	0,00	0,39	1,43	5,20	8,85	16,01	20,42	25,20
set 3	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
set 4	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
set 5	4,58	12,01	15,33	17,42	18,26	18,91	18,75	18,35	19,17	19,22
set 6	0,00	0,00	0,00	0,00	1,76	3,60	7,43	12,34	14,71	16,91
set 7	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
set 8	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
set 9	3,82	10,83	12,95	14,54	14,94	15,20	15,21	15,37	15,31	15,45
set 10	0,00	0,00	0,00	0,00	0,94	3,16	7,36	11,12	11,97	13,60
set 11	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
set 12	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00

(●) Bit Method Table with speed measure was updated ( ) Pattern Method

Finished.

Run Experiment

Рисунок В.14 – Показники втрати біт при масштабуванні для збільшення бітовим методом

Experiments

Speed | Translation | Rotation | Shear X | Shear Y | Scale x | **Scale X** | Almost Affine

Values of bitlose after applied transformation of proportional scale for extension

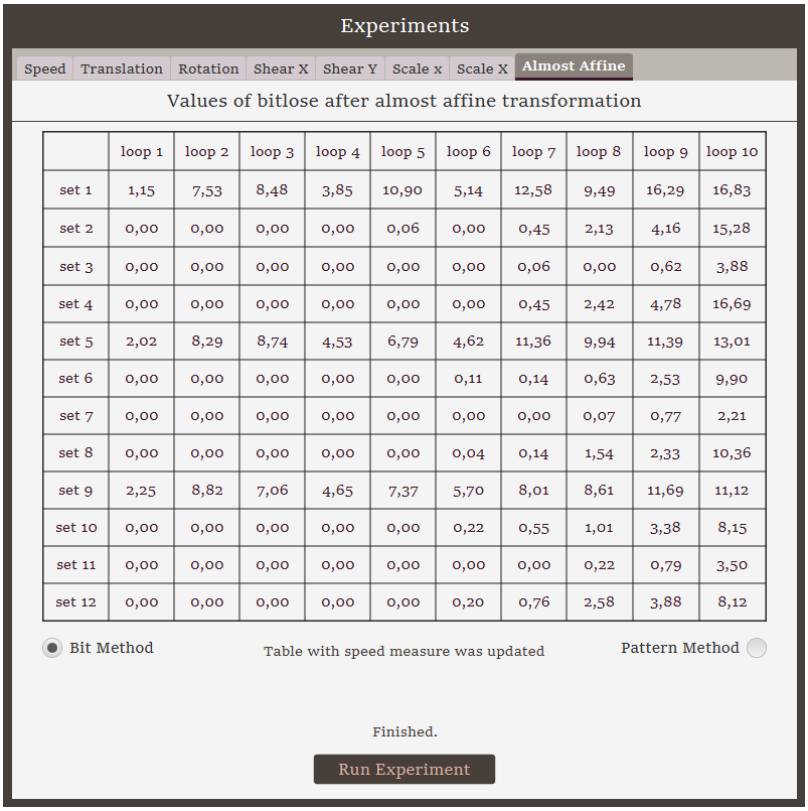
	loop 1	loop 2	loop 3	loop 4	loop 5	loop 6	loop 7	loop 8	loop 9	loop 10
set 1	2,33	13,68	22,44	26,71	29,41	29,58	29,27	30,67	31,04	30,87
set 2	0,00	0,00	0,00	0,00	0,56	3,90	6,52	11,80	18,01	22,95
set 3	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
set 4	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
set 5	9,59	15,22	17,68	19,40	18,86	19,40	19,35	19,63	19,50	19,49
set 6	0,00	0,00	0,00	0,00	1,58	10,34	12,25	15,57	19,00	18,29
set 7	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
set 8	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
set 9	15,11	14,87	15,31	15,67	15,41	15,55	15,53	15,53	15,58	15,56
set 10	4,14	3,88	6,15	7,01	10,27	11,54	14,45	15,44	15,66	15,69
set 11	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
set 12	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00

( ) Bit Method Table with speed measure was updated (●) Pattern Method

Finished.

Run Experiment

Рисунок В.15 – Показники втрати біт при масштабуванні для збільшення патерновим методом



	loop 1	loop 2	loop 3	loop 4	loop 5	loop 6	loop 7	loop 8	loop 9	loop 10
set 1	1,15	7,53	8,48	3,85	10,90	5,14	12,58	9,49	16,29	16,83
set 2	0,00	0,00	0,00	0,00	0,06	0,00	0,45	2,13	4,16	15,28
set 3	0,00	0,00	0,00	0,00	0,00	0,00	0,06	0,00	0,62	3,88
set 4	0,00	0,00	0,00	0,00	0,00	0,00	0,45	2,42	4,78	16,69
set 5	2,02	8,29	8,74	4,53	6,79	4,62	11,36	9,94	11,39	13,01
set 6	0,00	0,00	0,00	0,00	0,00	0,11	0,14	0,63	2,53	9,90
set 7	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,07	0,77	2,21
set 8	0,00	0,00	0,00	0,00	0,00	0,04	0,14	1,54	2,33	10,36
set 9	2,25	8,82	7,06	4,65	7,37	5,70	8,01	8,61	11,69	11,12
set 10	0,00	0,00	0,00	0,00	0,00	0,22	0,55	1,01	3,38	8,15
set 11	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,22	0,79	3,50
set 12	0,00	0,00	0,00	0,00	0,00	0,20	0,76	2,58	3,88	8,12

Рисунок В.16 – Показники втрати біт при майже афінному перетворенні бітовим методом

	loop 1	loop 2	loop 3	loop 4	loop 5	loop 6	loop 7	loop 8	loop 9	loop 10
set 1	2,56	11,18	5,34	3,51	6,80	6,91	11,99	14,61	16,01	16,40
set 2	0,00	0,00	0,00	0,00	0,00	0,00	1,01	1,69	3,51	8,79
set 3	0,00	0,00	0,00	0,00	0,00	0,00	0,17	0,00	0,84	1,26
set 4	0,00	0,00	0,00	0,00	0,00	0,00	0,53	0,79	2,44	5,93
set 5	10,09	14,31	11,85	9,04	14,47	11,17	14,71	14,71	16,99	17,15
set 6	0,00	0,00	0,00	0,00	0,00	0,51	1,19	2,79	2,21	5,44
set 7	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,56	1,23	1,00
set 8	0,00	0,00	0,00	0,00	0,00	0,14	1,04	2,00	2,04	4,04
set 9	10,96	14,73	12,39	13,48	15,52	13,41	14,90	13,88	15,10	14,92
set 10	1,38	1,83	2,39	2,47	4,94	6,14	6,56	6,18	7,51	9,76
set 11	0,00	0,00	0,00	0,00	0,00	0,00	0,44	1,32	2,40	4,49
set 12	0,00	0,00	0,00	0,00	0,44	0,91	1,81	4,06	4,16	8,02

Рисунок В.17 – Показники втрати біт при майже афінному перетворенні патерновим методом

3) Графічне відображення відсотку хибно вилучених біт

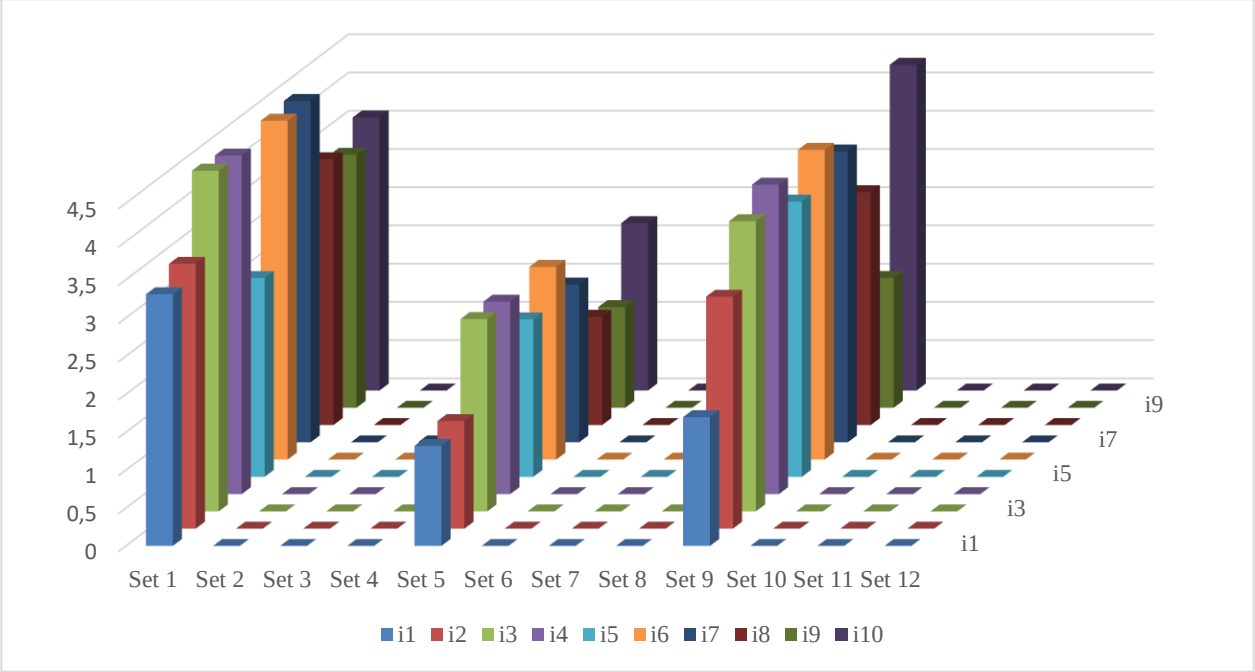


Рисунок В.18 – Графічне відображення показників втрачених біт бітовим методом при перенесенні

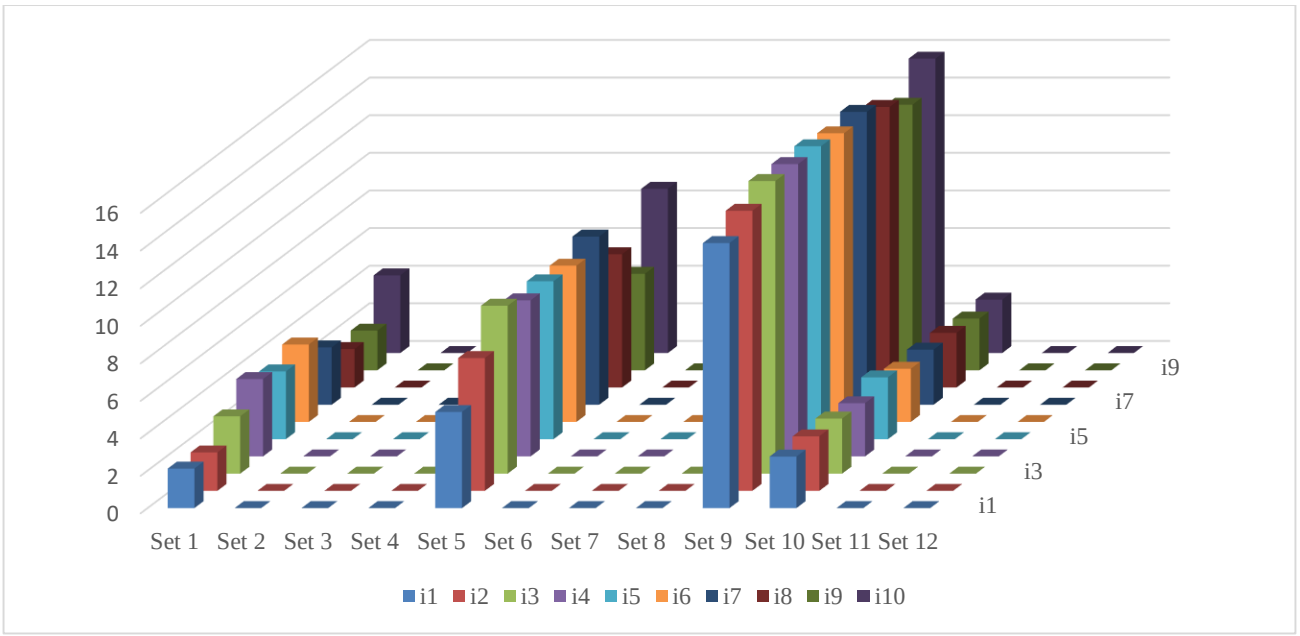


Рисунок В.19 – Графічне відображення показників втрачених біт патерновим методом при перенесенні

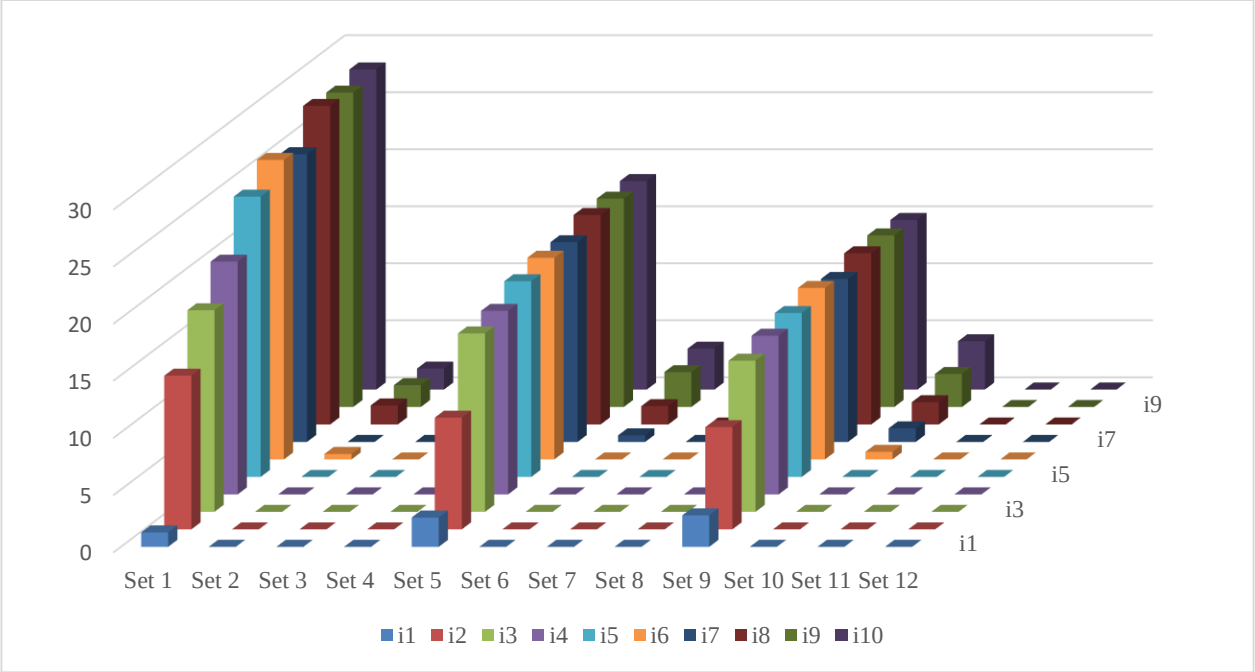


Рисунок В.20 – Графічне відображення показників втрачених біт бітовим методом при повороті

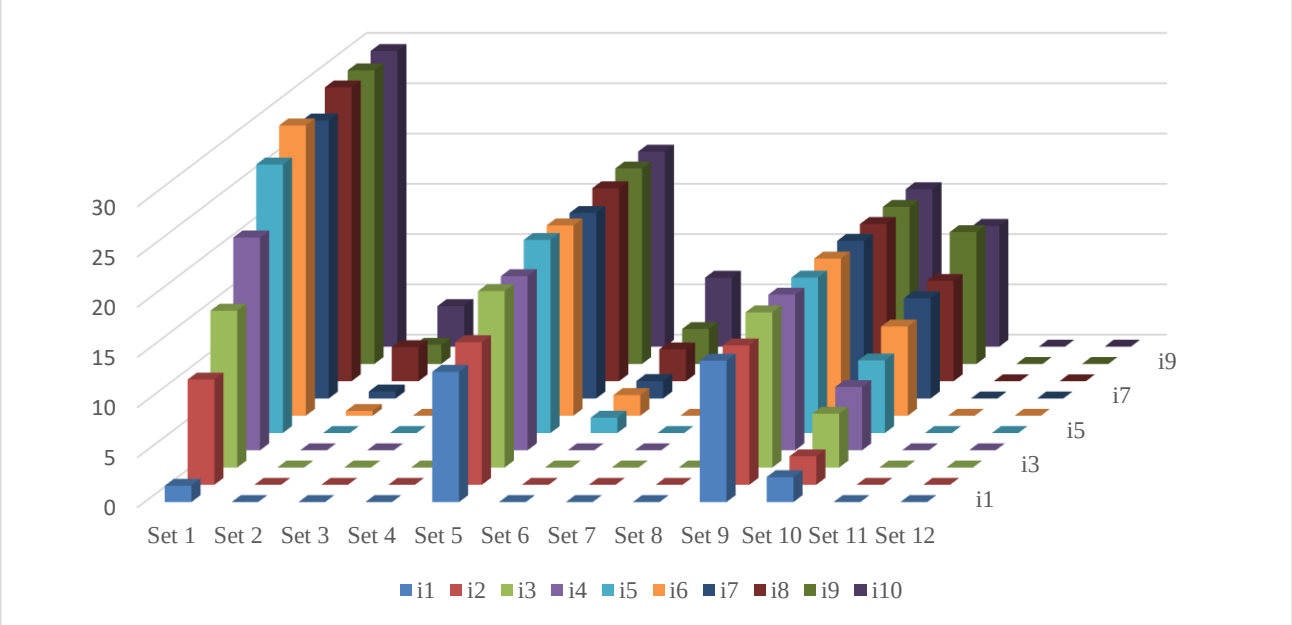


Рисунок В.21 – Графічне відображення показників втрачених біт патерновим методом при повороті

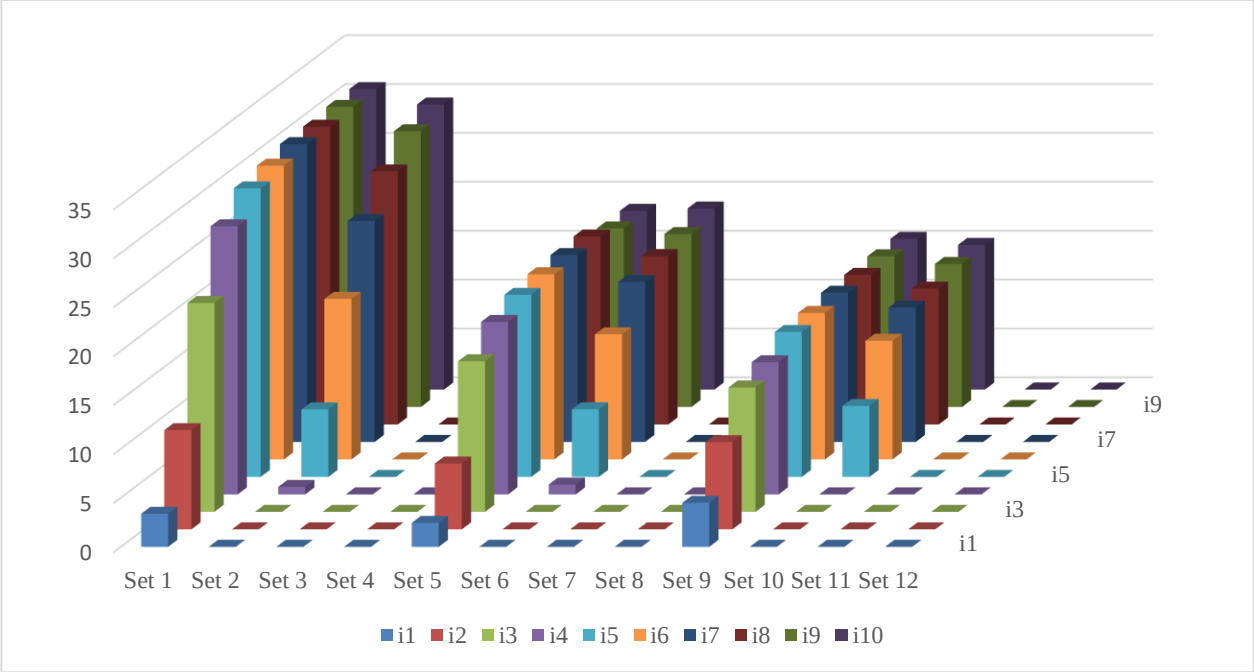


Рисунок В.22 – Графічне відображення показників втрачених біт бітовим методом при X-зсуві

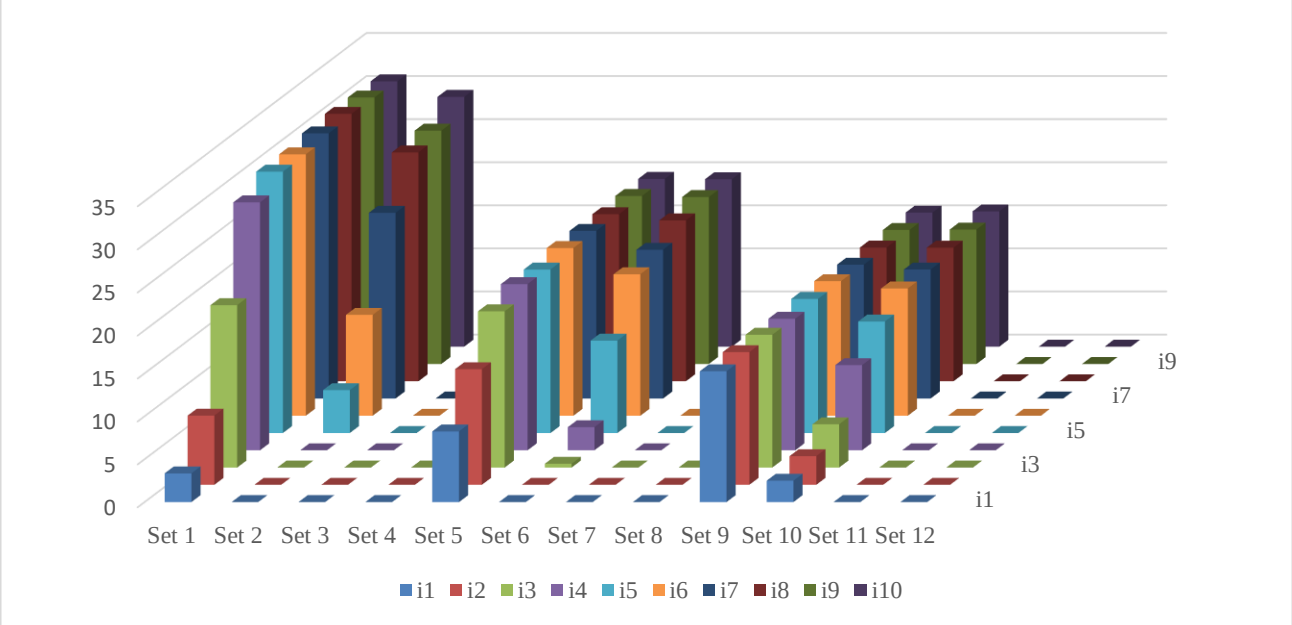


Рисунок В.23 – Графічне відображення показників втрачених біт патерновим методом при X-зсуві

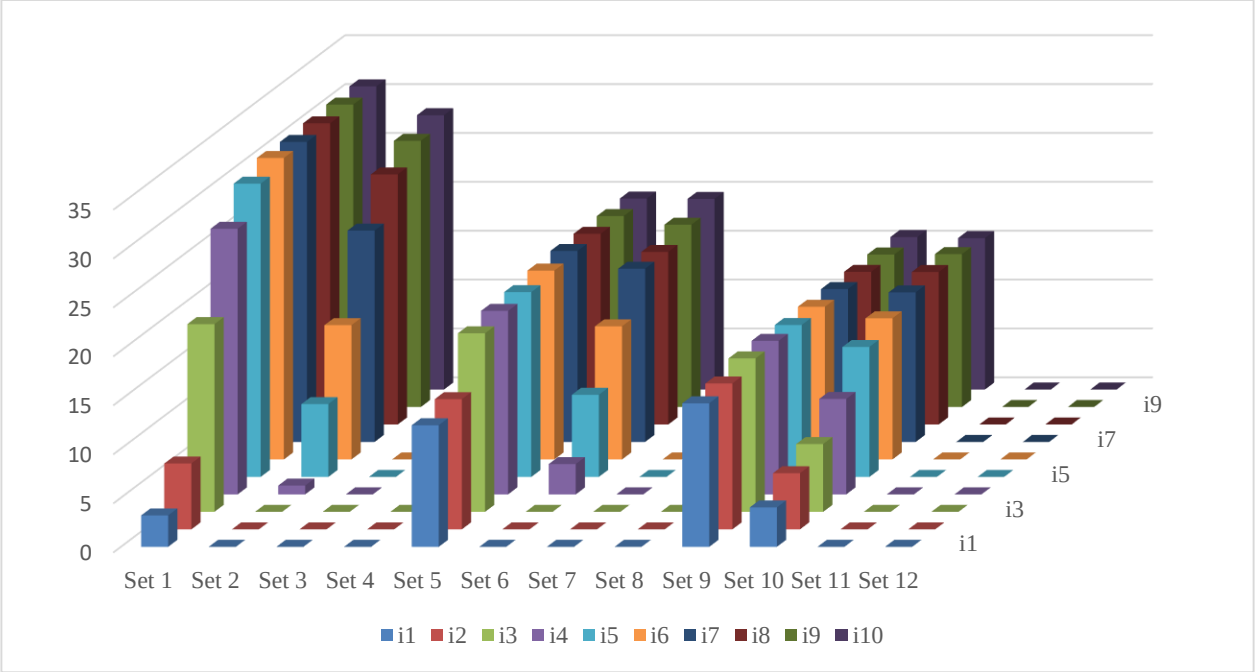


Рисунок В.24 – Графічне відображення показників втрачених біт бітовим методом при Y-зсуві

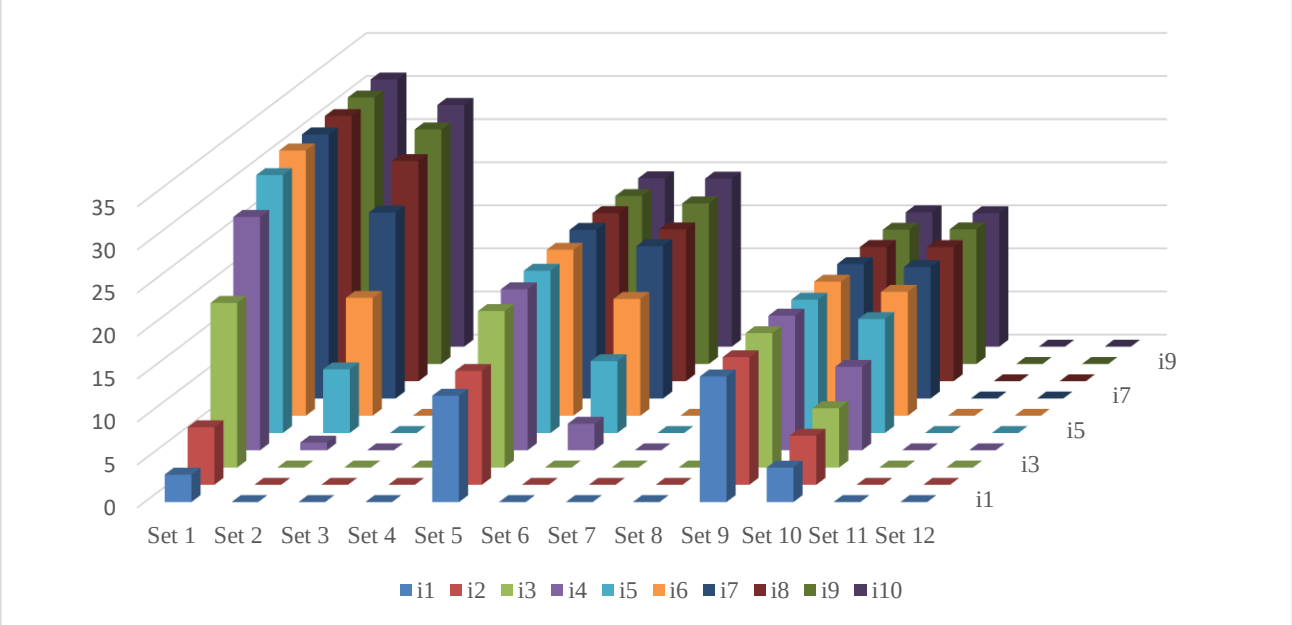


Рисунок В.25 – Графічне відображення показників втрачених біт патерновим методом при Y-зсуві

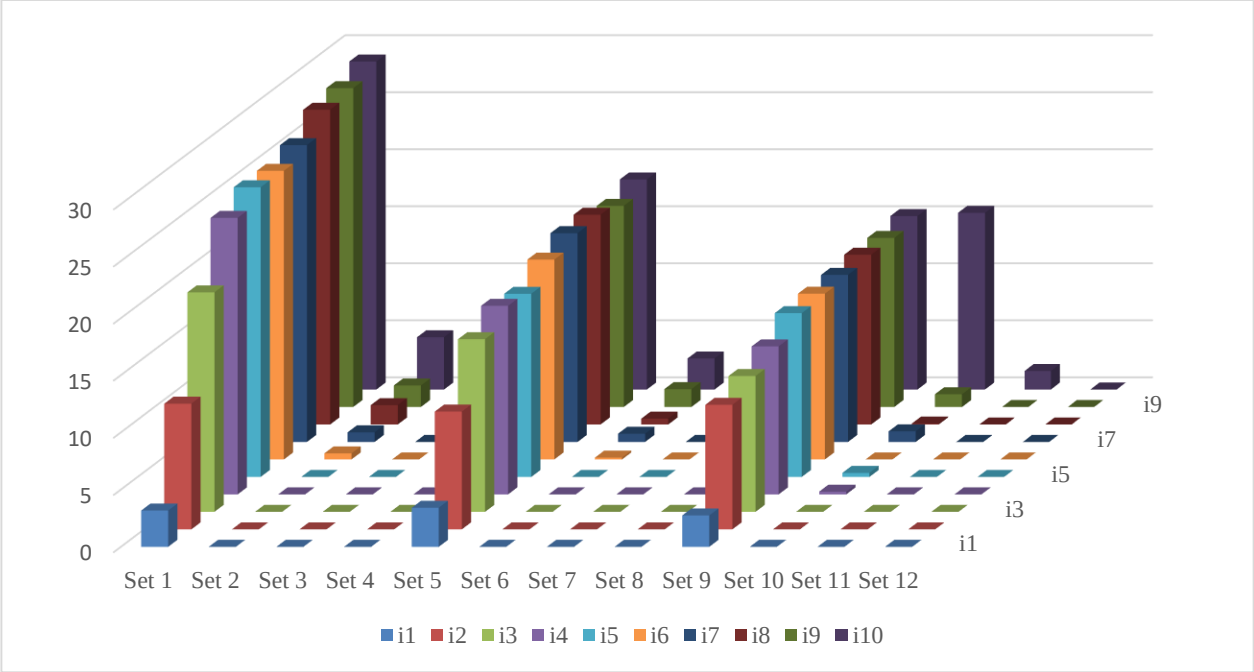


Рисунок В.26 – Графічне відображення показників втрачених біт бітовим методом при масштабуванні зменшення

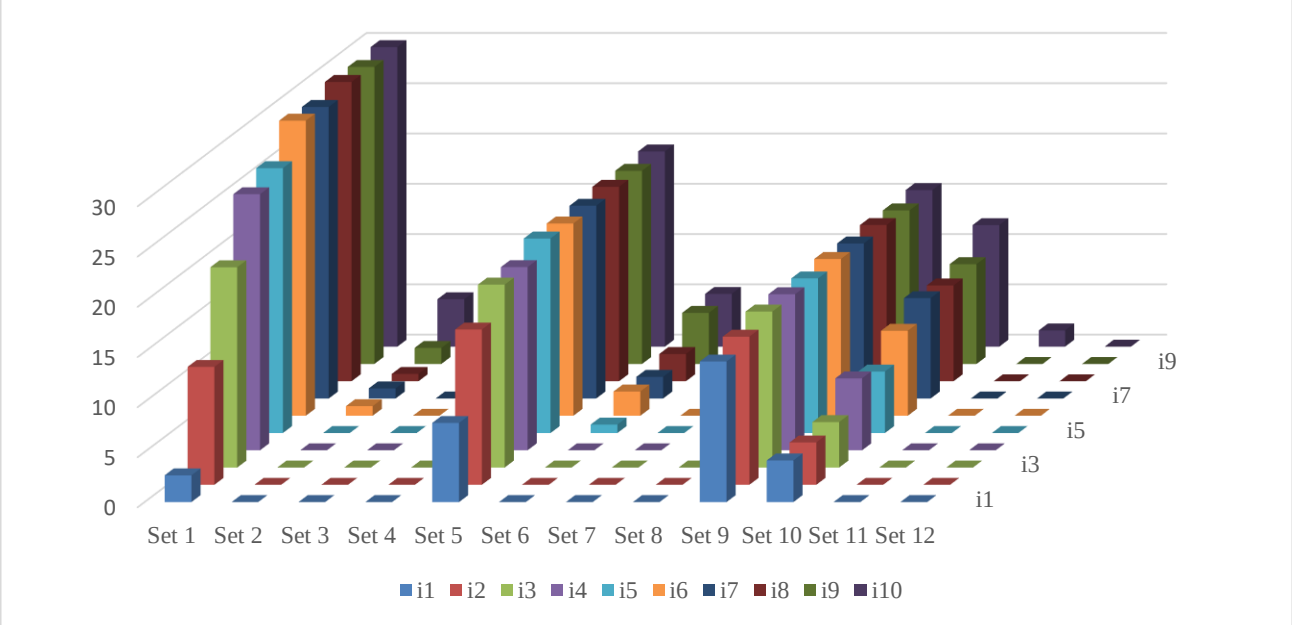


Рисунок В.27 – Графічне відображення показників втрачених біт патерновим методом при масштабуванні зменшення

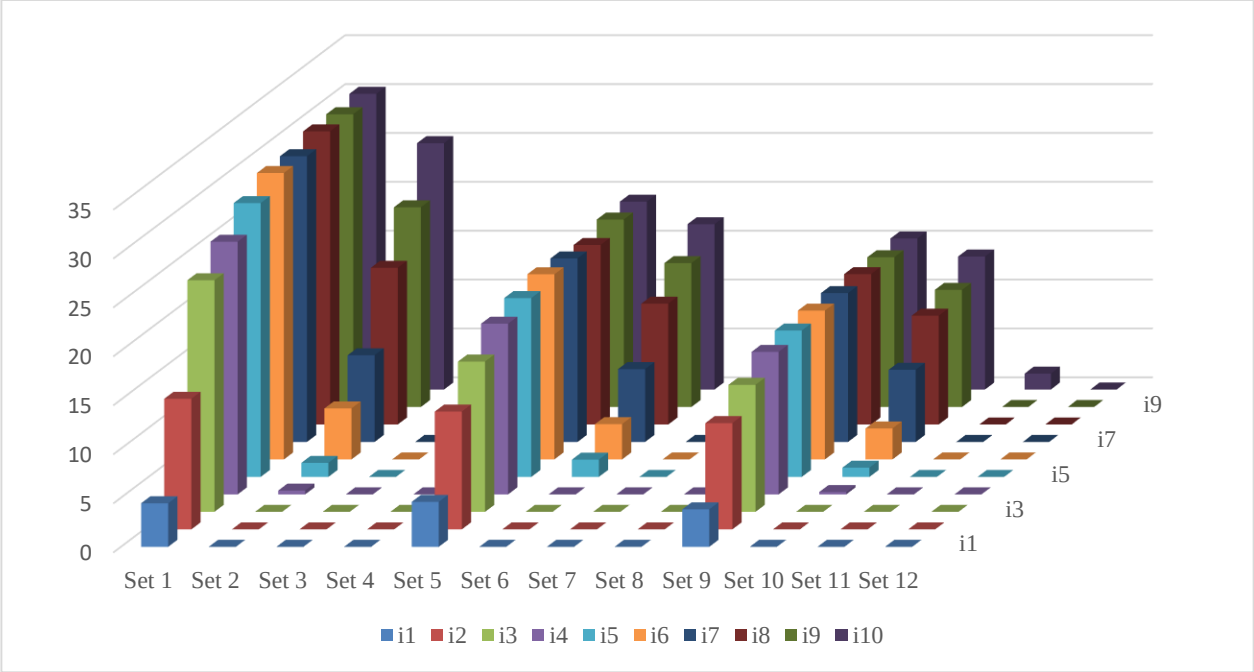


Рисунок В.28 – Графічне відображення показників втрачених біт бітовим методом при масштабуванні збільшення

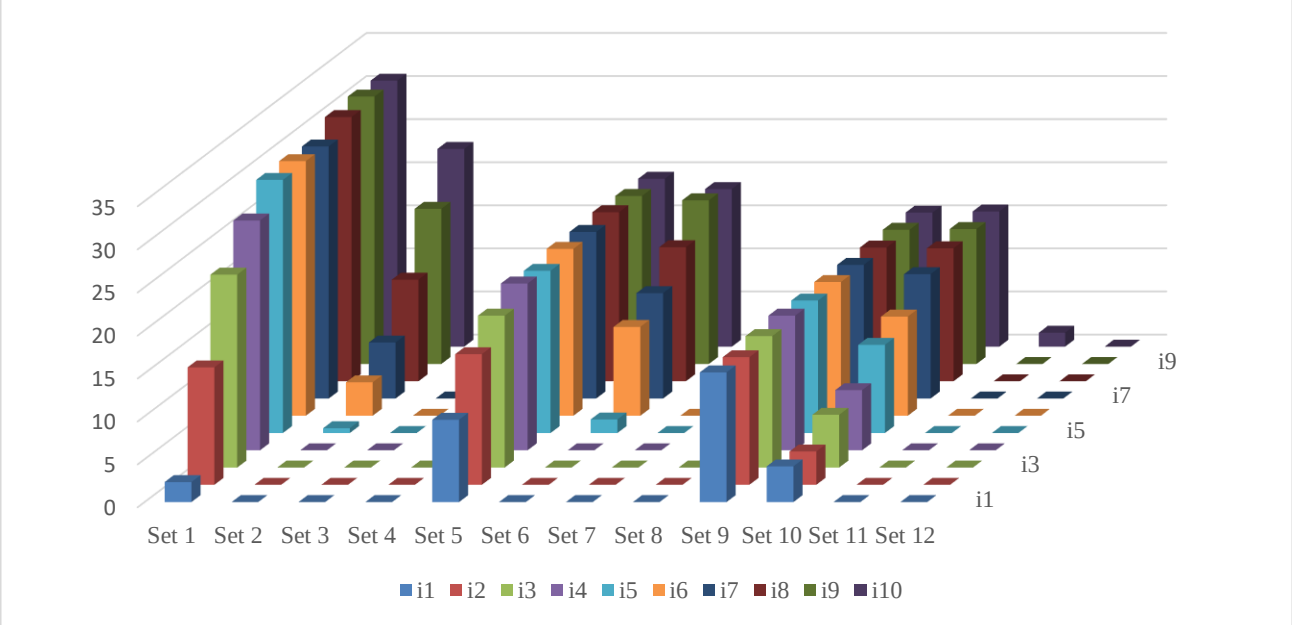


Рисунок В.29 – Графічне відображення показників втрачених біт патерновим методом при масштабуванні збільшення

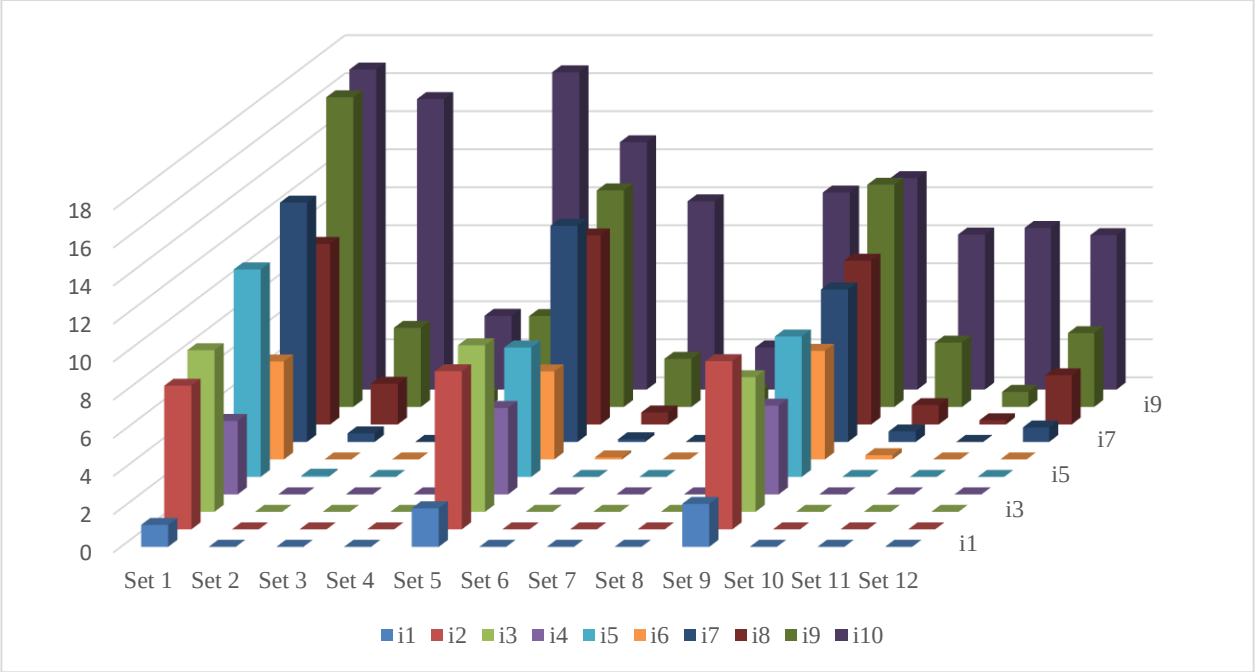


Рисунок В.30 – Графічне відображення показників втрачених біт бітовим методом при майже афінному перетворенні

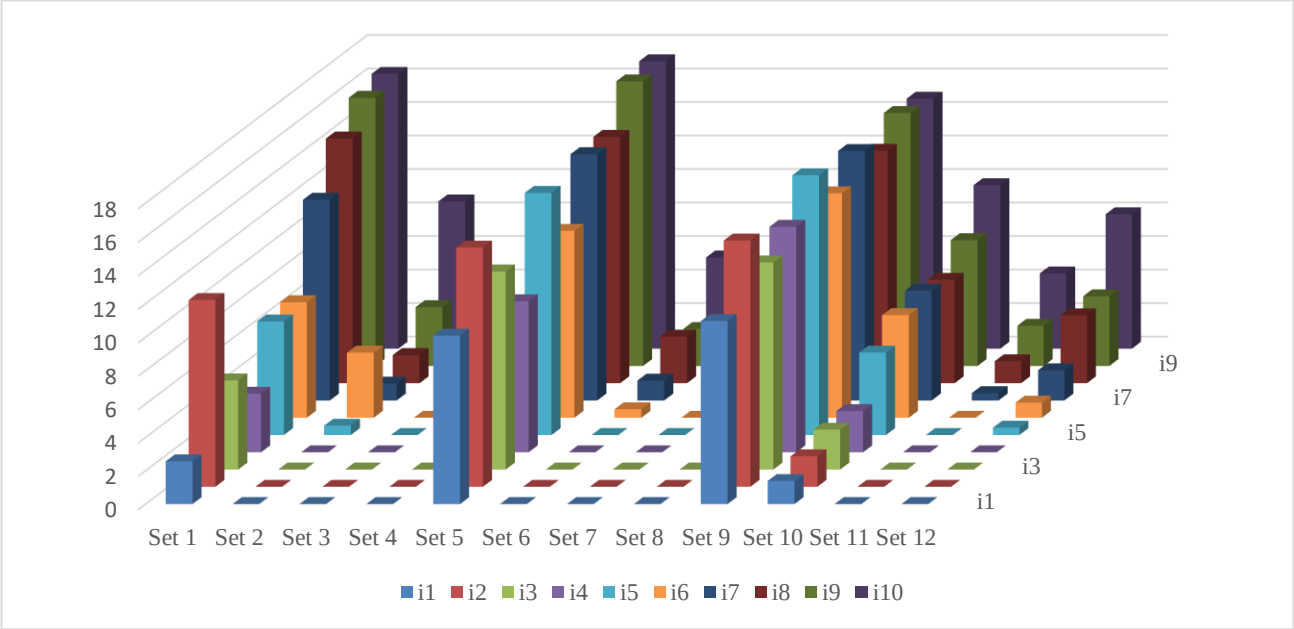


Рисунок В.31 – Графічне відображення показників втрачених біт патерновим методом при майже афінному перетворенні

ДОДАТОК Г

Експериментально отримані швидкісні характеристики

1) Табличне відображення середніх показників

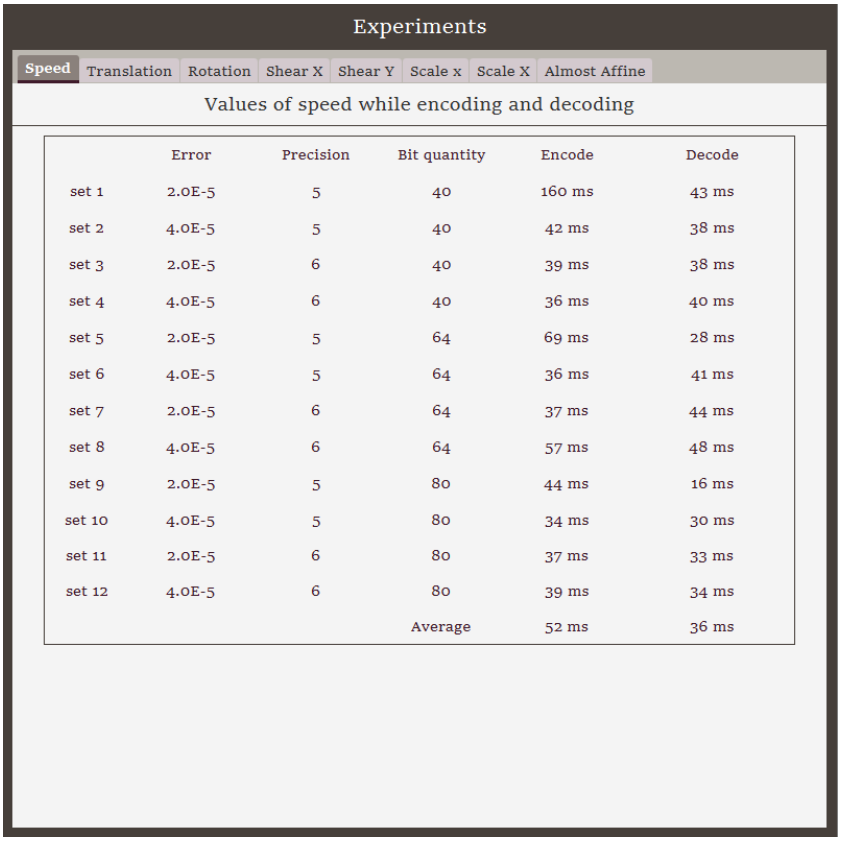
Experiments

Speed | Translation | Rotation | Shear X | Shear Y | Scale x | Scale X | Almost Affine

Values of speed while encoding and decoding

	Error	Precision	Bit quantity	Encode	Decode
set 1	2.0E-5	5	40	590 ms	293 ms
set 2	4.0E-5	5	40	202 ms	175 ms
set 3	2.0E-5	6	40	147 ms	137 ms
set 4	4.0E-5	6	40	135 ms	136 ms
set 5	2.0E-5	5	64	117 ms	158 ms
set 6	4.0E-5	5	64	93 ms	139 ms
set 7	2.0E-5	6	64	82 ms	155 ms
set 8	4.0E-5	6	64	99 ms	150 ms
set 9	2.0E-5	5	80	84 ms	133 ms
set 10	4.0E-5	5	80	95 ms	143 ms
set 11	2.0E-5	6	80	78 ms	148 ms
set 12	4.0E-5	6	80	83 ms	155 ms
			Average	150 ms	160 ms

Рисунок Г.1 – Часові показники при експериментах перенесення бітовим методом



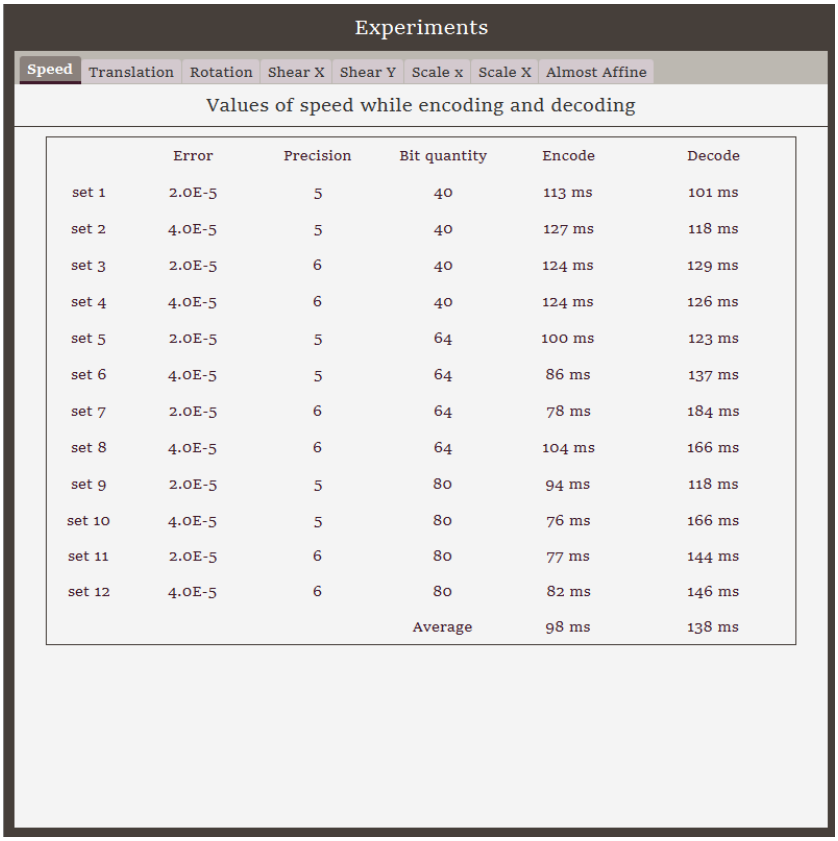
Experiments

Speed | Translation | Rotation | Shear X | Shear Y | Scale x | Scale X | Almost Affine

Values of speed while encoding and decoding

	Error	Precision	Bit quantity	Encode	Decode
set 1	2.0E-5	5	40	160 ms	43 ms
set 2	4.0E-5	5	40	42 ms	38 ms
set 3	2.0E-5	6	40	39 ms	38 ms
set 4	4.0E-5	6	40	36 ms	40 ms
set 5	2.0E-5	5	64	69 ms	28 ms
set 6	4.0E-5	5	64	36 ms	41 ms
set 7	2.0E-5	6	64	37 ms	44 ms
set 8	4.0E-5	6	64	57 ms	48 ms
set 9	2.0E-5	5	80	44 ms	16 ms
set 10	4.0E-5	5	80	34 ms	30 ms
set 11	2.0E-5	6	80	37 ms	33 ms
set 12	4.0E-5	6	80	39 ms	34 ms
			Average	52 ms	36 ms

Рисунок Г.2 – Часові показники при експериментах перенесення патерновим методом



Experiments

Speed | Translation | Rotation | Shear X | Shear Y | Scale x | Scale X | Almost Affine

Values of speed while encoding and decoding

	Error	Precision	Bit quantity	Encode	Decode
set 1	2.0E-5	5	40	113 ms	101 ms
set 2	4.0E-5	5	40	127 ms	118 ms
set 3	2.0E-5	6	40	124 ms	129 ms
set 4	4.0E-5	6	40	124 ms	126 ms
set 5	2.0E-5	5	64	100 ms	123 ms
set 6	4.0E-5	5	64	86 ms	137 ms
set 7	2.0E-5	6	64	78 ms	184 ms
set 8	4.0E-5	6	64	104 ms	166 ms
set 9	2.0E-5	5	80	94 ms	118 ms
set 10	4.0E-5	5	80	76 ms	166 ms
set 11	2.0E-5	6	80	77 ms	144 ms
set 12	4.0E-5	6	80	82 ms	146 ms
			Average	98 ms	138 ms

Рисунок Г.3 – Часові показники при експериментах повороту бітовим методом

Experiments

Speed | Translation | Rotation | Shear X | Shear Y | Scale x | Scale X | Almost Affine

Values of speed while encoding and decoding

	Error	Precision	Bit quantity	Encode	Decode
set 1	2.0E-5	5	40	26 ms	17 ms
set 2	4.0E-5	5	40	35 ms	29 ms
set 3	2.0E-5	6	40	44 ms	33 ms
set 4	4.0E-5	6	40	35 ms	33 ms
set 5	2.0E-5	5	64	27 ms	14 ms
set 6	4.0E-5	5	64	26 ms	32 ms
set 7	2.0E-5	6	64	33 ms	35 ms
set 8	4.0E-5	6	64	43 ms	32 ms
set 9	2.0E-5	5	80	34 ms	8 ms
set 10	4.0E-5	5	80	28 ms	25 ms
set 11	2.0E-5	6	80	32 ms	34 ms
set 12	4.0E-5	6	80	46 ms	36 ms
			Average	34 ms	27 ms

Рисунок Г.4 – Часові показники при експериментах повороту патерновим методом

Experiments

Speed | Translation | Rotation | Shear X | Shear Y | Scale x | Scale X | Almost Affine

Values of speed while encoding and decoding

	Error	Precision	Bit quantity	Encode	Decode
set 1	2.0E-5	5	40	117 ms	88 ms
set 2	4.0E-5	5	40	120 ms	102 ms
set 3	2.0E-5	6	40	120 ms	120 ms
set 4	4.0E-5	6	40	125 ms	124 ms
set 5	2.0E-5	5	64	87 ms	95 ms
set 6	4.0E-5	5	64	84 ms	108 ms
set 7	2.0E-5	6	64	78 ms	152 ms
set 8	4.0E-5	6	64	106 ms	173 ms
set 9	2.0E-5	5	80	74 ms	91 ms
set 10	4.0E-5	5	80	71 ms	111 ms
set 11	2.0E-5	6	80	66 ms	148 ms
set 12	4.0E-5	6	80	80 ms	153 ms
			Average	94 ms	122 ms

Рисунок Г.5 – Часові показники при експериментах X-зсуву бітовим методом

Experiments

Speed | Translation | Rotation | Shear X | Shear Y | Scale x | Scale X | Almost Affine

Values of speed while encoding and decoding

	Error	Precision	Bit quantity	Encode	Decode
set 1	2.0E-5	5	40	26 ms	14 ms
set 2	4.0E-5	5	40	26 ms	24 ms
set 3	2.0E-5	6	40	32 ms	31 ms
set 4	4.0E-5	6	40	39 ms	33 ms
set 5	2.0E-5	5	64	37 ms	11 ms
set 6	4.0E-5	5	64	35 ms	21 ms
set 7	2.0E-5	6	64	25 ms	32 ms
set 8	4.0E-5	6	64	31 ms	32 ms
set 9	2.0E-5	5	80	36 ms	8 ms
set 10	4.0E-5	5	80	58 ms	18 ms
set 11	2.0E-5	6	80	34 ms	38 ms
set 12	4.0E-5	6	80	30 ms	31 ms
			Average	34 ms	24 ms

Рисунок Г.6 – Часові показники при експериментах X-зсуву патерновим методом

Experiments

Speed | Translation | Rotation | Shear X | Shear Y | Scale x | Scale X | Almost Affine

Values of speed while encoding and decoding

	Error	Precision	Bit quantity	Encode	Decode
set 1	2.0E-5	5	40	134 ms	123 ms
set 2	4.0E-5	5	40	143 ms	131 ms
set 3	2.0E-5	6	40	160 ms	152 ms
set 4	4.0E-5	6	40	145 ms	148 ms
set 5	2.0E-5	5	64	99 ms	110 ms
set 6	4.0E-5	5	64	155 ms	133 ms
set 7	2.0E-5	6	64	111 ms	170 ms
set 8	4.0E-5	6	64	96 ms	161 ms
set 9	2.0E-5	5	80	92 ms	108 ms
set 10	4.0E-5	5	80	85 ms	135 ms
set 11	2.0E-5	6	80	87 ms	183 ms
set 12	4.0E-5	6	80	83 ms	173 ms
			Average	115 ms	143 ms

Рисунок Г.7 – Часові показники при експериментах Y-зсуву бітовим методом

Experiments

Speed | Translation | Rotation | Shear X | Shear Y | Scale x | Scale X | Almost Affine

Values of speed while encoding and decoding

	Error	Precision	Bit quantity	Encode	Decode
set 1	2.0E-5	5	40	35 ms	14 ms
set 2	4.0E-5	5	40	27 ms	26 ms
set 3	2.0E-5	6	40	34 ms	30 ms
set 4	4.0E-5	6	40	28 ms	32 ms
set 5	2.0E-5	5	64	35 ms	11 ms
set 6	4.0E-5	5	64	32 ms	20 ms
set 7	2.0E-5	6	64	32 ms	34 ms
set 8	4.0E-5	6	64	27 ms	33 ms
set 9	2.0E-5	5	80	44 ms	8 ms
set 10	4.0E-5	5	80	28 ms	16 ms
set 11	2.0E-5	6	80	33 ms	34 ms
set 12	4.0E-5	6	80	27 ms	32 ms
			Average	31 ms	24 ms

Рисунок Г.8 – Часові показники при експериментах Y-зсуву патерновим методом

Experiments

Speed | Translation | Rotation | Shear X | Shear Y | Scale x | Scale X | Almost Affine

Values of speed while encoding and decoding

	Error	Precision	Bit quantity	Encode	Decode
set 1	2.0E-5	5	40	125 ms	119 ms
set 2	4.0E-5	5	40	243 ms	125 ms
set 3	2.0E-5	6	40	135 ms	121 ms
set 4	4.0E-5	6	40	158 ms	133 ms
set 5	2.0E-5	5	64	91 ms	96 ms
set 6	4.0E-5	5	64	83 ms	132 ms
set 7	2.0E-5	6	64	88 ms	145 ms
set 8	4.0E-5	6	64	74 ms	167 ms
set 9	2.0E-5	5	80	96 ms	100 ms
set 10	4.0E-5	5	80	75 ms	141 ms
set 11	2.0E-5	6	80	73 ms	148 ms
set 12	4.0E-5	6	80	77 ms	148 ms
			Average	109 ms	131 ms

Рисунок Г.9 – Часові показники при експериментах масштабування для зменшення бітовим методом

Experiments

Speed | Translation | Rotation | Shear X | Shear Y | Scale x | Scale X | Almost Affine

Values of speed while encoding and decoding

	Error	Precision	Bit quantity	Encode	Decode
set 1	2.0E-5	5	40	32 ms	17 ms
set 2	4.0E-5	5	40	34 ms	34 ms
set 3	2.0E-5	6	40	31 ms	31 ms
set 4	4.0E-5	6	40	26 ms	30 ms
set 5	2.0E-5	5	64	26 ms	12 ms
set 6	4.0E-5	5	64	32 ms	33 ms
set 7	2.0E-5	6	64	38 ms	35 ms
set 8	4.0E-5	6	64	29 ms	30 ms
set 9	2.0E-5	5	80	28 ms	8 ms
set 10	4.0E-5	5	80	29 ms	25 ms
set 11	2.0E-5	6	80	39 ms	40 ms
set 12	4.0E-5	6	80	37 ms	33 ms
			Average	31 ms	27 ms

Рисунок Г.10 – Часові показники при експериментах масштабування для зменшення патерновим методом

Experiments

Speed | Translation | Rotation | Shear X | Shear Y | Scale x | Scale X | Almost Affine

Values of speed while encoding and decoding

	Error	Precision	Bit quantity	Encode	Decode
set 1	2.0E-5	5	40	110 ms	89 ms
set 2	4.0E-5	5	40	124 ms	109 ms
set 3	2.0E-5	6	40	128 ms	133 ms
set 4	4.0E-5	6	40	136 ms	142 ms
set 5	2.0E-5	5	64	84 ms	91 ms
set 6	4.0E-5	5	64	86 ms	117 ms
set 7	2.0E-5	6	64	81 ms	139 ms
set 8	4.0E-5	6	64	89 ms	134 ms
set 9	2.0E-5	5	80	76 ms	93 ms
set 10	4.0E-5	5	80	91 ms	152 ms
set 11	2.0E-5	6	80	74 ms	139 ms
set 12	4.0E-5	6	80	74 ms	137 ms
			Average	96 ms	122 ms

Рисунок Г.11 – Часові показники при експериментах масштабування для збільшення бітовим методом

Experiments

Speed | Translation | Rotation | Shear X | Shear Y | Scale x | Scale X | Almost Affine

Values of speed while encoding and decoding

	Error	Precision	Bit quantity	Encode	Decode
set 1	2.0E-5	5	40	30 ms	14 ms
set 2	4.0E-5	5	40	33 ms	26 ms
set 3	2.0E-5	6	40	31 ms	31 ms
set 4	4.0E-5	6	40	24 ms	31 ms
set 5	2.0E-5	5	64	31 ms	11 ms
set 6	4.0E-5	5	64	25 ms	24 ms
set 7	2.0E-5	6	64	35 ms	31 ms
set 8	4.0E-5	6	64	33 ms	32 ms
set 9	2.0E-5	5	80	31 ms	6 ms
set 10	4.0E-5	5	80	35 ms	20 ms
set 11	2.0E-5	6	80	23 ms	32 ms
set 12	4.0E-5	6	80	24 ms	32 ms
			Average	29 ms	24 ms

Рисунок Г.12 – Часові показники при експериментах масштабування для збільшення патерновим методом

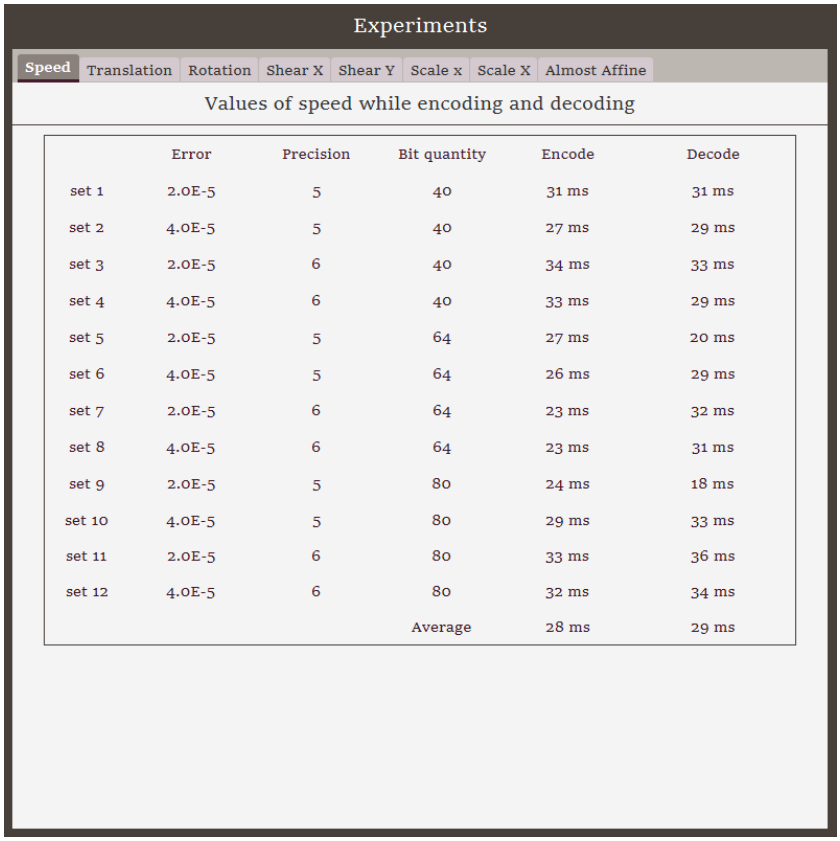
Experiments

Speed | Translation | Rotation | Shear X | Shear Y | Scale x | Scale X | Almost Affine

Values of speed while encoding and decoding

	Error	Precision	Bit quantity	Encode	Decode
set 1	2.0E-5	5	40	114 ms	107 ms
set 2	4.0E-5	5	40	119 ms	116 ms
set 3	2.0E-5	6	40	134 ms	116 ms
set 4	4.0E-5	6	40	120 ms	118 ms
set 5	2.0E-5	5	64	75 ms	109 ms
set 6	4.0E-5	5	64	81 ms	123 ms
set 7	2.0E-5	6	64	78 ms	127 ms
set 8	4.0E-5	6	64	91 ms	133 ms
set 9	2.0E-5	5	80	67 ms	116 ms
set 10	4.0E-5	5	80	66 ms	130 ms
set 11	2.0E-5	6	80	78 ms	182 ms
set 12	4.0E-5	6	80	75 ms	137 ms
			Average	91 ms	126 ms

Рисунок Г.13 – Часові показники при експериментах майже афінного перетворення бітовим методом



	Error	Precision	Bit quantity	Encode	Decode
set 1	2.0E-5	5	40	31 ms	31 ms
set 2	4.0E-5	5	40	27 ms	29 ms
set 3	2.0E-5	6	40	34 ms	33 ms
set 4	4.0E-5	6	40	33 ms	29 ms
set 5	2.0E-5	5	64	27 ms	20 ms
set 6	4.0E-5	5	64	26 ms	29 ms
set 7	2.0E-5	6	64	23 ms	32 ms
set 8	4.0E-5	6	64	23 ms	31 ms
set 9	2.0E-5	5	80	24 ms	18 ms
set 10	4.0E-5	5	80	29 ms	33 ms
set 11	2.0E-5	6	80	33 ms	36 ms
set 12	4.0E-5	6	80	32 ms	34 ms
			Average	28 ms	29 ms

Рисунок Г.14 – Часові показники при експериментах майже афінного перетворення патерновим методом

2) Узагальнення середніх значень часових показників

Таблиця Г.1 – Середній час приховування та вилучення під час експериментів

Перетворення\ Режим	Бітовий		Патерновий	
	Закодування, мс	Декодування, мс	Закодування, мс	Декодування, мс
Перенесення	150	160	52	36
Поворот	98	138	34	27
X-зсув	94	122	34	24
Y-зсув	115	143	31	24
Зменшення	109	131	31	27
Збільшення	96	122	29	24
Майже афінне	91	126	28	29

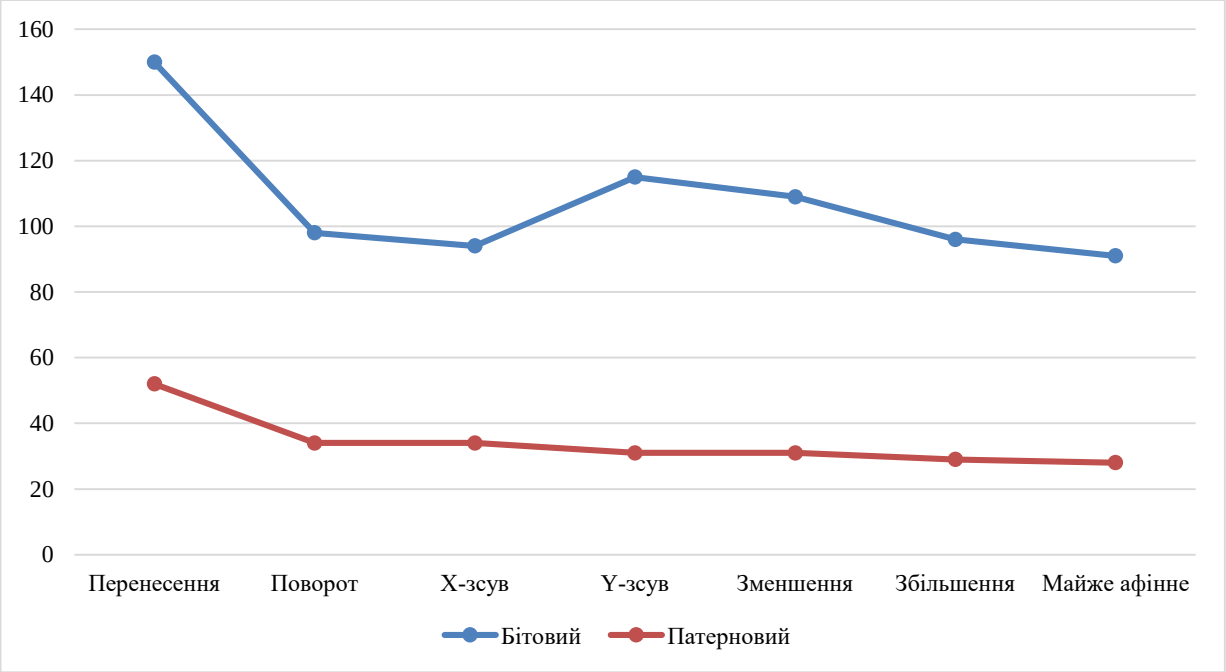


Рисунок Г.15 – Середні значення часу закодування

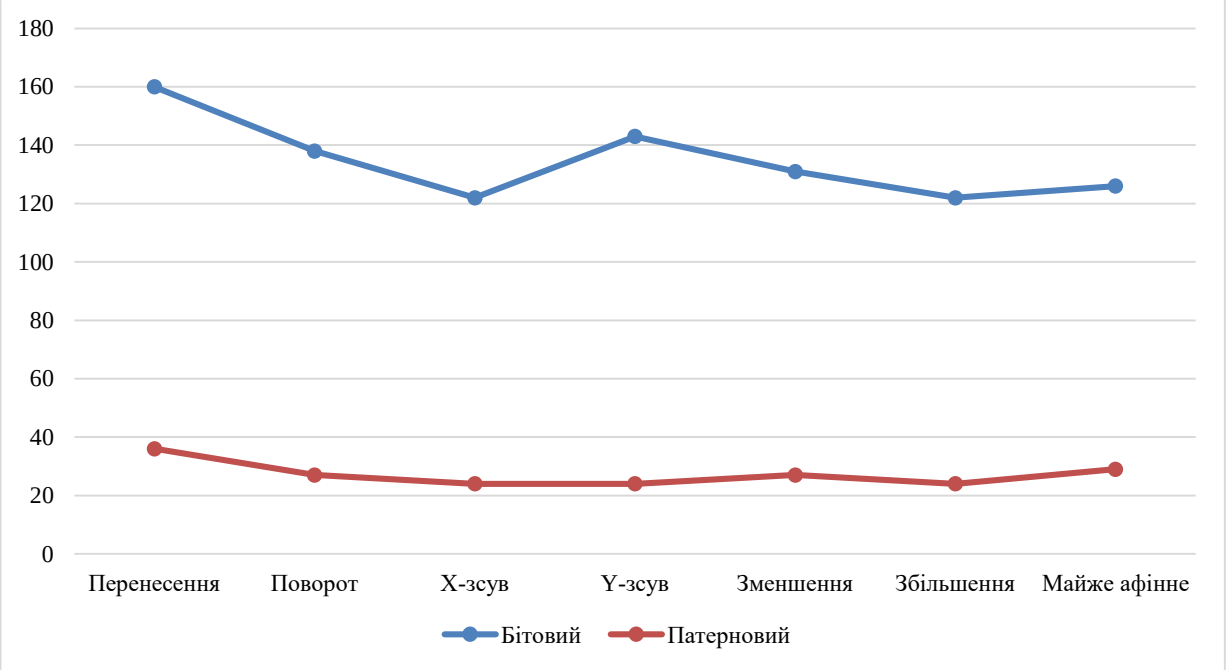


Рисунок Г.16 – Середні значення часу декодування

ДОДАТОК Д

Результати пошуку спотворень стеконтейнерів

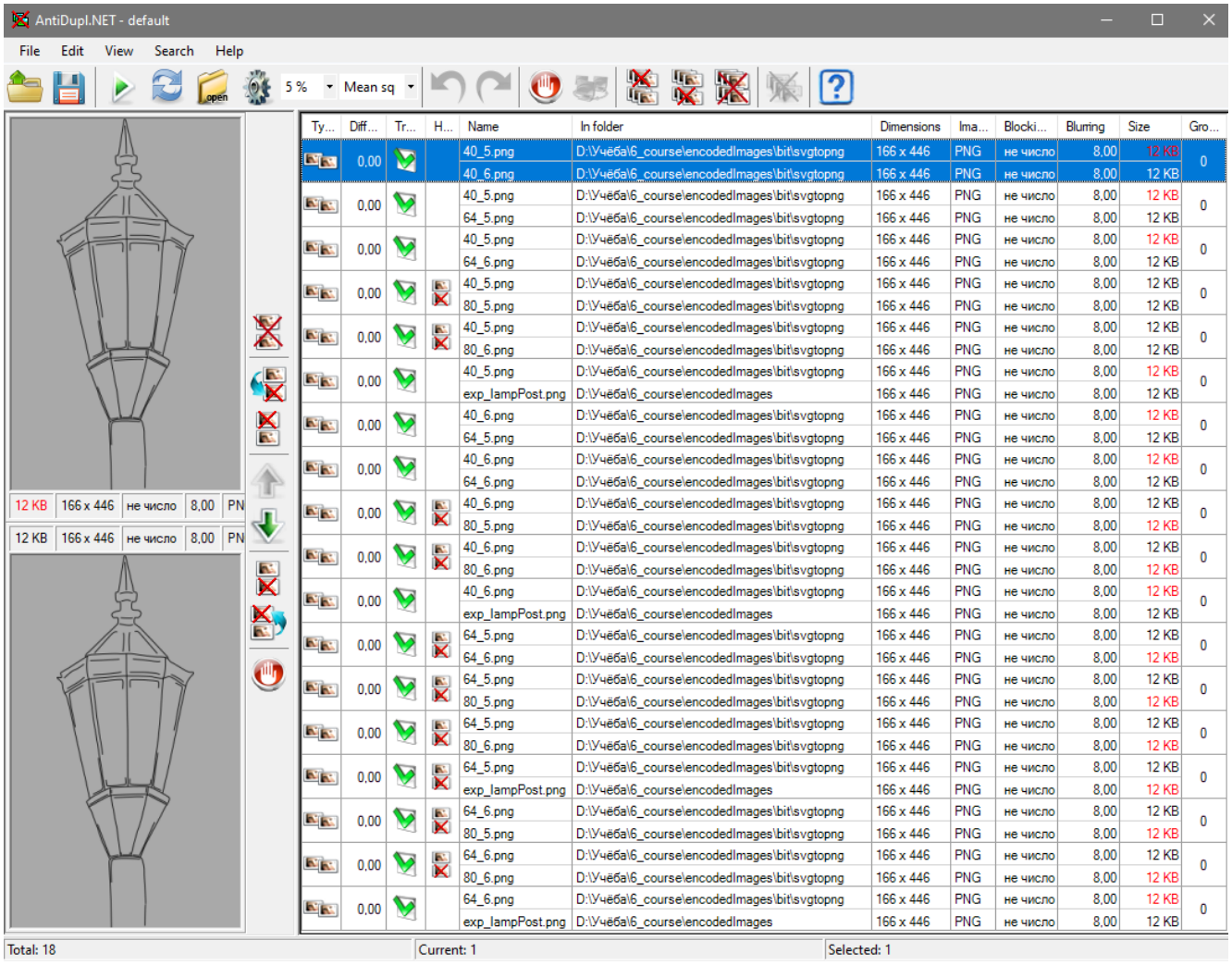


Рисунок Д.1 – Показники різниці стегоконтейнерів, отриманих після бітового закодування

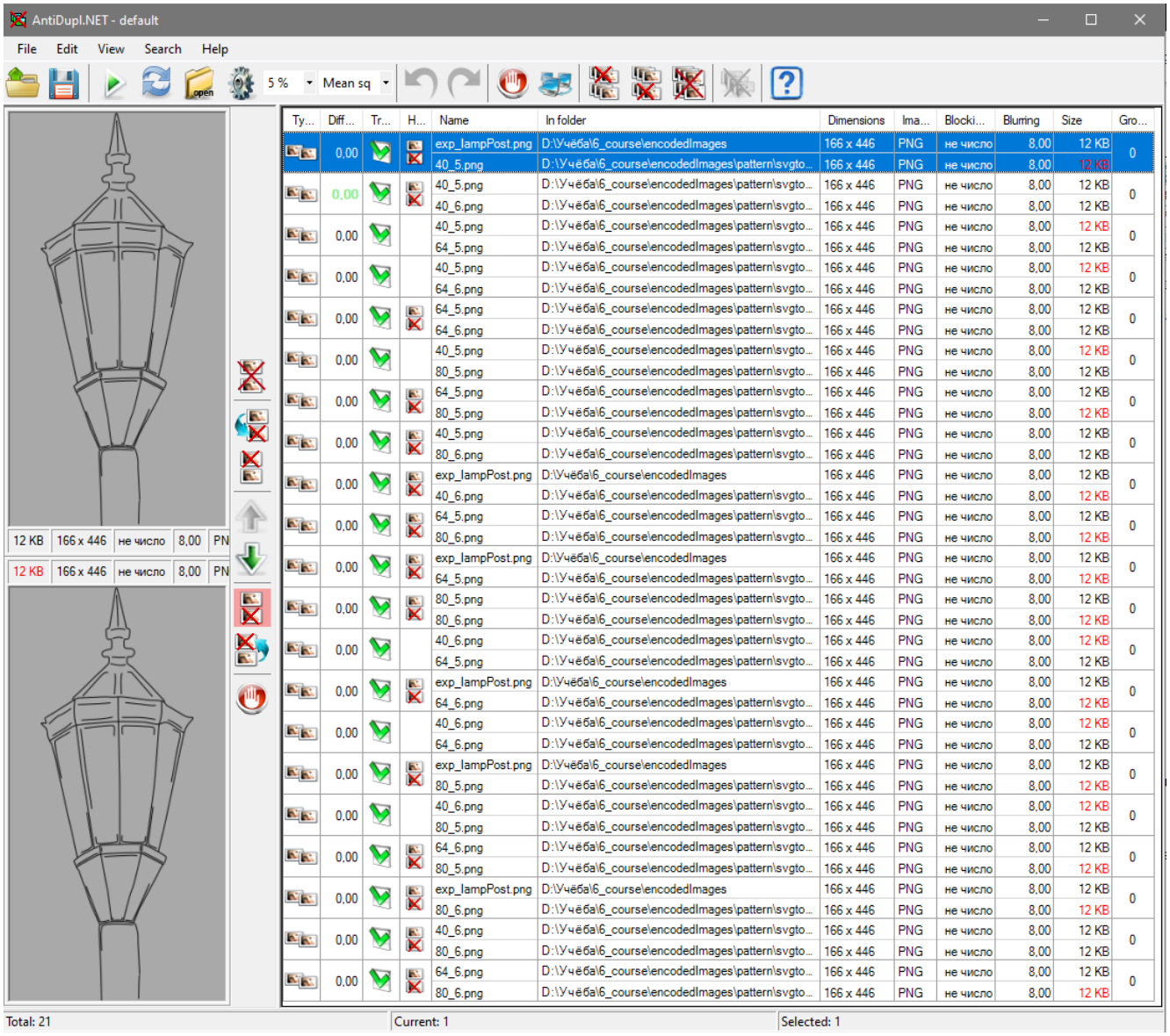


Рисунок Д.2 – Показники різниці стегоконтейнерів, отриманих після патернового закодування