НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

Кафедра системного програмування і спеціалізованих комп‘ютерних систем

«До захисту допущено»

Завідувач кафедри

_____ Віталій РОМАНКЕВИЧ
(підпис)

“___” червня 2020 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Спеціалізовані комп'ютерні системи»

зі спеціальності 123 «Комп'ютерна інженерія»

на тему: Програмні засоби ущільнення графічних даних

Виконав: студент IV курсу, групи КВ-63

Верьовочкін Олександр Сергійович _____

Керівник ст. викл. Дробязко Ірина Павлівна _____

Консультант з нормоконтролю, доц.каф.СПСКС, к.т.н. Клятченко Я.М._____

Рецензент_____

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2020 року

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

Кафедра системного програмування і спеціалізованих комп‘ютерних систем

Рівень вищої освіти – перший (бакалаврський)

Спеціальність 123 «Комп’ютерна інженерія»

За освітньо-професійною програмою «Спеціалізовані комп’ютерні системи»

ЗАТВЕРДЖУЮ
Завідувач кафедри

_____ Віталій РОМАНКЕВИЧ
(підпис)

«___» _____ 2019 р.

ЗАВДАННЯ
на дипломний проєкт студента
Верьовочкіна Олександра Сергійовича

1. Тема проєкту «Програмні засоби ущільнення графічних даних»
   Керівник проєкту старший викладач Дробязко Ірина Павлівна,
   затверджені наказом по університету від «_25_» __05_ 2020_ р. № 1181-С_

2. Термін подання студентом проєкту 25 травня 2020 р.

3. Вихідні дані до проєкту: див. Технічне завдання

4. Зміст пояснювальної записки:
   1) Аналіз предметної області;
   2) Аналіз існуючих алгоритмів ущільнення графічних даних;
   3) Вибір засобів розроблення;
   4) Розроблення програми ущільнення графічних даних.

5. Перелік графічного матеріалу:
   1) Програмні засоби ущільнення графічних даних. Схема структурна;
   2) Компоненти програми. Схема структурна;
   3) Компресія даних. Схема алгоритму;
   4) Декомпресія даних. Схема алгоритму;

5). Презентація проєкту.

6. Консультанти розділів проєкту*

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Клятченко Я.М., доц.		

7. Дата видачі завдання 25 жовтня 2019 р.

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1.	Вивчення літератури за тематикою проєкту	15.11.2019	
2.	Розроблення та узгодження технічного завдання	30.11.2019	
3.	Аналіз існуючих рішень	05.02.2020	
4.	Підготовка матеріалів першого розділу дипломного проєкту	05.03.2020	
5.	Підготовка матеріалів другого розділу дипломного проєкту	27.03.2020	
6.	Підготовка матеріалів третього розділу дипломного проєкту	15.04.2020	
7.	Підготовка графічної частини дипломного проєкту	10.05.2020	
8.	Оформлення документації дипломного проєкту	14.05.2020	
9.	Попередній огляд матеріалів диплому на кафедрі	25.05.2020	

Студент _____ Олександр ВЕРЬОВОЧКІН

Керівник проєкту _____ Ірина ДРОБЯЗКО

_____

АНОТАЦІЯ

Кваліфікаційна робота включає пояснювальну записку (56 с., 14 рис., 4 додатки)

Об’єктом розробки є програма ущільнення графічних даних.

Метою даного проєкту є розробка програмних засобів, що реалізують ущільнення графічних даних, а також їх відновлення до початкового стану без втрати якості зображень.

Під час розробки:

- проведено аналіз предметної області;
- проаналізовані існуючі алгоритми ущільнення графічних даних;
- розроблено алгоритми ущільнення та відновлення;
- розроблено програму ущільнення на основі створених алгоритмів.

Для розробки використано мову програмування C++, інтегроване середовище розробки Microsoft Visual Studio.

Ключові слова: ущільнення даних, графічні дані, C++, зображення, компресія, декомпресія.

ABSTRACT

Qualifying project includes an explanatory note (56 p., 14 pic., 4 applications)

The object of development is a program of compression of graphic data.

The aim of this project is to develop a software that implements the compression of graphic data without loss of image quality, as well as the restoration of compressed data to its original state.

During the development:

- the analysis of the subject area;
- the existing algorithms of compression of graphic data are analyzed;
- algorithms of consolidation and restoration are developed
- the program is developed on the basis of the created algorithms.

The C ++ programming language, an integrated Microsoft Visual Studio development environment, was used for development.

Keywords: compaction, graphic data, C ++, image, compression, decompression.

Поз.	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кількість аркушів	№ прим.	Примітки
			Документація загальна			
			Новорозроблена			
	А4	ІАЛЦ.045490.002 ТЗ	Програмні засоби	3		
			ущільнення графічних			
			даних.			
			Технічне завдання			
	А4	ІАЛЦ.045490.003 ТП	Програмні засоби	2		
			ущільнення графічних			
			даних.			
			Відомість технічного			
			проекту			
	А4	ІАЛЦ.045490.004 ПЗ	Програмні засоби	56		
			ущільнення графічних			
			даних.			
			Пояснювальна записка			

Змін.	Арк.	№ докум.	Підпис	Дата	ІАЛЦ.045490.001 ОА		
Розробив		Верьовочкін О.С.			Програмні засоби ущільнення графічних даних **Опис альбому**	Літ.	Аркуш 1 / Аркушів 2
Перевірив		Дробязко І.П.				НТУУ "КПІ" ФПМ КВ-63	
Н. контроль		Клятченко Я.М.					
Затвердив		Романкевич В.О.					

Поз.	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кількість аркушів	№ прим.	Примітки
	А3	ІАЛЦ.045490.005 Д1	Програмні засоби	1		
			ущільнення графічних			
			даних.			
			Схема структурна			
	А3	ІАЛЦ.045490.006 Д2	Компоненти програми.	1		
			Схема структурна			
	А3	ІАЛЦ.045490.007 Д3	Компресія даних.	1		
			Схема алгоритму			
	А3	ІАЛЦ.045490.008 Д4	Декомпресія даних.	1		
			Схема алгоритму			
		Диск CD-ROM	Текст пояснювальної	1		
			записки.			
			Графічний матеріал			

Змін.	Арк.	№ докум.	Підпис	Дата	ІАЛЦ.045490.001 ОА	Арк. 2

ЗМІСТ

					ІАЛЦ.045490.002 ТЗ		
Змін	Арк.	№ докум.	Підпис	Дата			
Розробив		Верьовочкін О.С.			Програмні засоби ущільнення графічних даних	Літ.	Аркуш 1 / Аркушів 4
Перевірив		Дробязко І.П.					
Н. контроль		Клятченко Я.М.			***Технічне завдання***	КПІ ім. Ігоря Сікорського, ФПМ КВ-63	
Затвердив		Романкевич В.О.					

1. НАЙМЕНУВАННЯ ТА ГАЛУЗЬ РОЗРОБКИ

Назва розробки: «Програмні засоби ущільнення графічних даних».

Галузь застосування: інформаційні технології, комп’ютерна графіка.

2. ПІДСТАВА ДЛЯ РОЗРОБКИ

Підставою для розробки є завдання на виконання роботи першого (бакалаврського) рівня вищої освіти, затверджене кафедрою системного програмування і спеціалізованих комп’ютерних систем Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського».

3. МЕТА І ПРИЗНАЧЕННЯ РОБОТИ

Метою даного проєкту є розробка програмних засобів, що реалізують ущільнення графічних даних, а також їх відновлення до початкового стану без втрати якості зображень.

4. ДЖЕРЕЛА РОЗРОБКИ

Джерелом інформації є технічна та науково-технічна література, технічна документація, публікації в періодичних виданнях та електронні статті у мережі Інтернет.

5. ТЕХНІЧНІ ВИМОГИ

5.1 Вимоги до програмного продукту, що розробляється:

- підримка файлів формату bmp;
- можливість ущільнення вхідного файлу;
- можливість повного відновлення ущільненого файлу;
- надання користувачу підказок для запуску програми.
- можливість роботи з програмою через командний рядок

5.2 Вимоги до апаратного забезпечення:

- оперативна пам’ять: 1 Гб.

5.3 Вимоги до програмного та апаратного забезпечення користувача:

- операційна система Windows.

6. ЕТАПИ РОЗРОБКИ

№ з/п	Назва етапів виконання дипломного проекту	Термін виконання етапів
1.	Вивчення літератури за тематикою проєкту	15.11.2019
2.	Розроблення та узгодження технічного завдання	30.11.2019
3.	Аналіз існуючих рішень	05.02.2020
4.	Підготовка матеріалів першого розділу дипломного проєкту	05.03.2020
5.	Підготовка матеріалів другого розділу дипломного проєкту	27.03.2020
6.	Підготовка матеріалів третього розділу дипломного проєкту	15.04.2020
7.	Підготовка графічної частини дипломного проєкту	10.05.2020
8.	Оформлення документації дипломного проєкту	14.05.2020
9.	Попередній огляд матеріалів диплому на кафедрі	25.05.2020

Поз.	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кількість аркушів	№ прим.	Примітки
			Документація загальна			
			Новорозроблена			
	А4	ІАЛЦ.045490.004 ПЗ	Програмні засоби	56		
			ущільнення графічних			
			даних.			
			Пояснювальна записка			
	А3	ІАЛЦ.045490.005 Д1	Програмні засоби	1		
			ущільнення графічних			
			даних.			
			Схема структурна			
	А3	ІАЛЦ.045490.006 Д2	Компоненти програми.	1		
			Схема структурна			

Змін.	Арк.	№ докум.	Підпис	Дата	ІАЛЦ.045490.003 ТП		
Розробив		Верьовочкін О.С.			Програмні засоби ущільнення графічних даних Опис альбому	Літ.	Аркуш 1 / Аркушів 2
Перевірив		Дробязко І.П.					
Н. контроль		Клятченко Я.М.				КПІ ім. Ігоря Сікорського ФПМ КВ-63	
Затвердив		Тарасенко В.П.					

Поз.	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кількість аркушів	№ прим.	Примітки
	А3	ІАЛЦ.045490.007 Д3	Компресія даних.	1		
			Схема алгоритму			
	А3	ІАЛЦ.045490.008 Д4	Декомпресія даних.	1		
			Схема алгоритму			

Змін.	Арк.	№ докум.	Підпис	Дата	ІАЛЦ.045490.003 ТП	Арк. 2

ЗМІСТ

ІАЛЦ.045490.005 Д1. Програмні засоби ущільнення графічних даних. Схема структурна;

ІАЛЦ.045490.006 Д2. Компоненти програми. Схема структурна;

ІАЛЦ.045490.007 Д3. Компресія даних. Схема алгоритму;

ІАЛЦ.045490.008 Д4. Декомпресія даних. Схема алгоритму.

Додаток 2. Фрагменти програмного коду

Додаток 3. Презентація проєкту

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ, ТЕРМІНІВ

LZ77, LZ78, LZO, LZS, LZW, LZC, LZMA – алгоритми ущільнення сімейства LZ

LZW – Lempel-Ziv-Welch – алгоритм Лемпеля – Зіва – Велча

RGB – red, green, blue – колірна модель

RLE – Run Length Encoding – алгоритм кодування повторів

Deflate – алгоритм ущільнення даних без втрат

JPEG – алгоритм ущільнення даних з втратами

BMP – Bitmap Picture – формат зберігання растрових зображень

ДКП – дискретне косинусне перетворення

ВСТУП

В наш час майже вся графічна інформація, незалежно від сфери її застосування, зберігається в цифровому вигляді на носіях даних. Майже кожна людина має смартфон з фотокамерою. Люди створюють величезну кількість фотографій, створюють картини в графічних редакторах, отримують зображення з надпотужних телескопів. Для багатьох дуже важливим є питання довготривалого зберігання цих даних. Звичайно, зараз можна майже нескінченно збільшувати обсяги пам’яті для зберігання, але вона потребує обслуговування. Іншим підходом є зменшення розмірів самих даних таким чином, щоб в потрібний момент їх можна було отримати в повному обсязі.

Для цього було створено багато алгоритмів ущільнення даних. Галузь відносно нова, особливо що стосується ущільнення графічних даних. Тому багато з існуючих алгоритмів можна покращувати або створювати нові. Цьому сприяє і стрімкий розвиток інформаційних технологій.

Завданням дипломного проєкту є розроблення програмних засобів, що орієнтовані на обробку і зменшення обсягу графічних даних. Алгоритм, який є основою такої програми, має забезпечувати значне зменшення обсягу даних з можливістю такого ж простого відновлення даних до початкового стану. Тобто, однією з головних вимог до розроблюваного засобів є забезпечення повного відновлення даних до їх початкового стану.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

В наш час комп’ютерна графіка набула дуже великої популярності як область діяльності, в якій комп'ютери поряд зі спеціальним програмним забезпеченням використовуються в якості інструменту для створення, редагування зображень, оцифровування візуальної інформації, отриманої з реального світу, з метою подальшої її обробки та зберігання. Комп’ютерна графіка знайшла своє застосування майже в будь-якій області діяльності людини – від наукової до побутової, від ділової до художньої. А переважна більшість зображень існує саме в цифровому форматі.

За способом представлення зображень комп'ютерну графіку зазвичай розділяють на векторну і растрову, хоча відокремлюють ще й фрактальну графіку [1].

1.1 Растрова графіка

Растрова графіка є частиною комп'ютерної графіки, яка має справу зі створенням, обробкою та зберіганням растрових зображень. Растрове зображення є масивом кольорових точок (пікселів). Обробка растрової графіки здійснюється растровими графічними редакторами. Інколи говорять про растрову графіку, маючи на увазі зображення, представлене у растровому форматі зберігання інформації [2], [3].

Растрова графіка завжди оперує двовимірним масивом (матрицею) пікселів. Кожному пікселю відповідає значення яскравості, кольору, прозорості або комбінація цих значень. Растровий образ має деяке число рядків і стовпців.

Без особливих втрат растрові зображення можна лише зменшувати, хоча деякі деталі зображення тоді зникнуть назавжди, що неможливо у

векторному поданні. Збільшення ж растрових зображень створює збільшені квадрати того чи іншого кольору, які раніше були пікселями (рис. 1.1) [1].

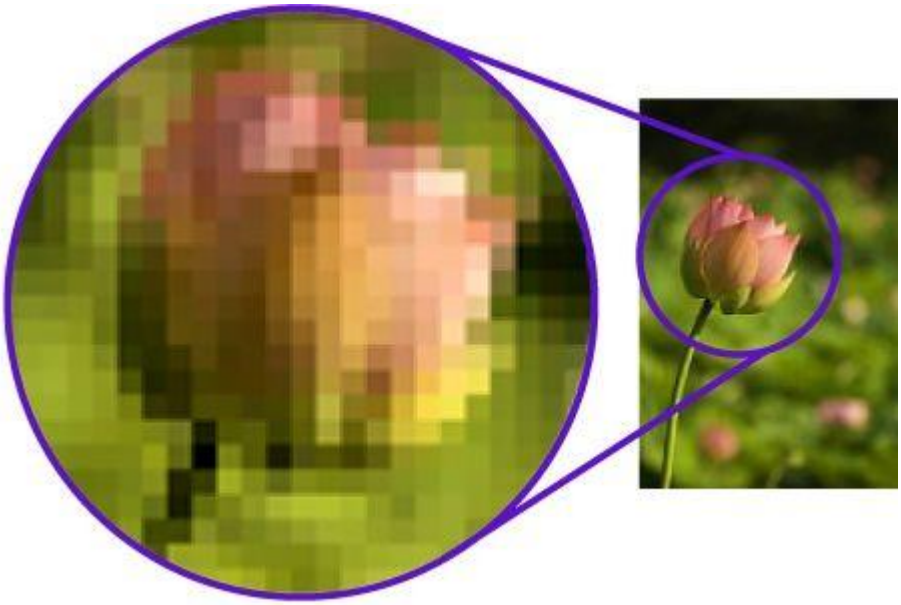

Рисунок 1.1 – Збільшення растрового зображення

Переваги растрової графіки:

- растрова графіка дозволяє створити практично будь-яке зображення, незалежно від складності;
- растрова графіка використовується сьогодні практично скрізь: від маленьких значків до великих рекламних плакатів;
- висока швидкість обробки складних зображень, якщо не потрібно масштабування;
- растрове представлення зображення є природним для більшості пристроїв введення-виведення графічної інформації (за винятком векторних пристроїв виводу), таких як монітори, матричні та струменеві принтери, цифрові фотоапарати, сканер;
- простота автоматизованого вводу (оцифрування) зображень, фотографій, слайдів, малюнків за допомогою сканерів, відеокамер, цифрових фотоапаратів;

- фотореалістичність: можна отримувати різні ефекти, такі як туман, розмитість, тонко регулювати кольори, створювати глибину предметів.

Недоліки растрової графіки:

- великий розмір файлів у простих зображень, тому що розмір файлу є пропорційним до площі зображення, роздільності і типу зображення і, переважно при хорошій якості, є великим;
- неможливість ідеального масштабування: растрове зображення має визначену роздільність і глибину представлення кольорів, ці параметри можна змінювати лише у визначених межах і, як правило, із втратою якості.
- неможливість виведення на друк на векторний графічний пристрій;
- складність управління окремими фрагментами зображення.

1.2 Векторна графіка

Векторна графіка – спосіб представлення об'єктів і зображень (формат опису) в комп'ютерній графіці, заснований на математичному описі елементарних геометричних об'єктів, зазвичай званих примітивами, таких як: точки, лінії, сплайни, криві Безьє, кола та багатокутники [4]. Об'єкти векторної графіки є графічними зображеннями математичних об'єктів. Об'єктам присвоюються деякі атрибути, наприклад, товщина ліній, колір заповнення. Малюнок зберігається як набір координат, векторів і інших чисел, що характеризують набір примітивів. При відтворенні об'єктів, що перекриваються, має значення їх порядок.

Зображення у векторному форматі дає простір для редагування. Зображення може без втрат масштабуватися, повертатися, деформуватися,

також імітація тривимірності в векторній графіці простіша, ніж в растровій (рис. 1.2).

Рисунок 1.2 – Демонстрація при збільшенні розміру векторного (зліва) і растрового (справа) зображення

Справа в тому, що кожне таке перетворення фактично виконується таким чином: старе зображення (або фрагмент) стирається, і замість нього будується нове. Математичний опис векторного малюнка залишається старим, змінюються тільки значення деяких змінних, наприклад, коефіцієнтів. Термін «векторна графіка» використовується для пояснення відмінностей від растрової графіки, в якій зображення представлено у вигляді графічної матриці [1].

При виведенні на матричні пристрої відображення (монітори) векторна графіка попередньо перетворюється в растрову графіку. Перетворення проводиться програмно або апаратно засобами сучасних відеокарт. Цю процедуру називають растеризацією [5].

Переваги векторної графіки:

- обсяг даних не залежить від реальної величини об'єкта, що дозволяє, використовуючи мінімальну кількість інформації, описати наскільки завгодно великий об'єкт. Наприклад, опис

окружності довільного радіусу вимагає завдання тільки 3 чисел, не враховуючи атрибутів;

- через те, що інформація про об'єкт зберігається в описовій формі, можна нескінченно збільшити графічний примітив при виведенні на графічний пристрій, наприклад, дугу кола, і вона залишиться при будь-якому збільшенні гладкою.
- параметри об'єктів можуть бути легко змінені. Зазвичай вказують розміри в апаратно-незалежних одиницях, які ведуть до найкращої можливої растеризації на растрових пристроях;
- спрощене редагування;
- деякі методи виготовлення зображень, вимагають саме векторний файл і не допускають растра. У їх числі: вишивка, виготовлення штампів і вивісок, гравірування.

Недоліки векторної графіки:

- не кожна графічна сцена може бути легко зображена у векторному вигляді – для подібного оригінального зображення може знадобитися опис дуже великої кількості примітивів з високою складністю, що негативно впливає на об’єм пам'яті і на час, необхідний для перетворення його в растровий формат для графічного виведення;
- переклад векторної графіки в растрове зображення досить простий, проте зворотний шлях, як правило, складний;
- перевага векторного зображення – масштабованість – пропадає, коли векторний формат відображається в растрі особливо малої роздільної здатності (наприклад, іконки 32 × 32 або 16 × 16).
- програми, пов'язані з растровою графікою, є на всіх ПК в усіх ОС. Для векторної графіки у середньостатистичного користувача є тільки браузер, який грає роль переглядача SVG.

1.3 Фрактальна графіка

Фрактал – об'єкт, окремі елементи якого успадковують властивості батьківських структур. Оскільки більш детальний опис елементів меншого масштабу відбувається за простим алгоритмом, описати такий об'єкт можна лише кількома математичними рівняннями.

Фрактали дозволяють описувати цілі класи зображень, для детального опису яких потрібно відносно мало пам'яті. З іншого боку, фрактали слабко застосовні до зображень поза цих класів. Найпростішим фрактальним об'єктом є фрактальний трикутник. Фрактальними властивостями володіють багато об'єктів живої і неживої природи. Звичайна сніжинка при збільшенні виявляється фрактальним об'єктом. Фрактальні алгоритми лежать в основі росту кристалів і рослин.

Властивість фрактальної графіки – моделювати образи живої природи обчисленням – часто використовують для автоматичної генерації незвичних ілюстрацій [1].

В даній роботі розглядатиметься растрове подання, бо саме такий спосіб дозволяє створювати фотореалістичні зображення найвищої якості.

1.4 Модель та глибина кольору

Модель кольору визначає подання різних кольорів спектру у вигляді набору числових характеристик певних базових компонентів. Зараз найбільш поширені такі моделі: RGB, CMYK, Lab, HSB, HSL та ін. Найширшого застосування в комп’ютерній техніці та програмах растрової графіки набула модель RGB. За цією моделлю колір кожного пікселю створюється завдяки накладанню трьох компонент – червоної (R), зеленої (G), та синьої (B) [6].

Важливим поняттям є глибина кольору. Його також називають бітовою глибиною. Це поняття визначає кількість бітів, які використовуються для представлення кольору одного пікселя растрового зображення. Найменше значення глибини кольору – 1. З такою глибиною можна відобразити чорно-біле зображення. Зазвичай в зображеннях використовується глибина 24. Її ще називають 24bit колір або TrueColor. В моделі RGB 24bit колір використовує по 8 бітів для представлення кожної з компонент. Всього можна задати $2^{24} = 16777216$ різних кольорів.

1.5 Обґрунтування теми дипломного проєкту

В наш час актуальною є проблема зберігання графічної інформації великого об’єму. Для зменшення її обсягів було створено багато алгоритмів ущільнення даних.

Завданням дипломного проєкту є, на основі аналізу існуючих алгоритмів та засобів ущільнення графічних даних, створити програмні засоби, які реалізуватимуть ущільнення даних без втрат після їх відновлення.

Розроблюваний програмний засіб має забезпечити істотне зменшення графічних даних за маленький проміжок часу. Крім того, до алгоритму компресії, який буде реалізовано в програмі, потрібно розробити алгоритм зворотної дії, який буде відновлювати ущільнені дані.

2 АНАЛІЗ ІСНУЮЧИХ АЛГОРИТМІВ

Незважаючи на те, що ємність запам’ятовуючих пристроїв постійно збільшується, ущільнення будь-якої цифрової інформації залишається дуже актуальним питанням. Графічна інформація сама по собі займає великі об’єми пам’яті та може містити багато надлишкової інформації. Тому не дивно, що було розроблено багато алгоритмів, зокрема для ущільнення зображень.

Всі алгоритми ущільнення зображень поділяються на два великих класи:

- Алгоритми ущільнення без втрат
- Алгоритми ущільнення з втратами

Суть ущільнення полягає в тому, щоб зменшити розмір файлу зображення і при цьому зберегти його візуальну якість.

Алгоритми першого класу забезпечують можливість повного відновлення ущільненого зображення до початкового. Тобто, має існувати алгоритм відновлення, що є зворотним до алгоритму ущільнення. Але цей критерій накладає обмеження на ступінь ущільнення даних. Тому виник клас алгоритмів ущільнення з втратами. Вони базуються на особливостях і недосконалості людського зору. Деякі характеристики зображення можна змінити так, що візуально різниця буде майже непомітною, а різниця в розмірах файлів – значною.

Великою проблемою комп’ютерної графіки є те, що не існує єдиного критерію для оцінки якості зображення. Тому для алгоритмів ущільнення з втратами зазвичай можна задавати ступінь втрати якості. Очевидно, що чим менша якість – тим більше буде ступінь ущільнення (рис. 2.1) [7].

Рисунок 2.1 – Різниця між зображенням, ущільненим алгоритмом без втрат (зліва), і зображенням, ущільненим алгоритмом з втратами (справа)

Взагалі, всі існуючі алгоритми ущільнення даних базуються на трьох методах зменшення надлишковості: зміна вмісту даних, зміна структури даних, та одночасне поєднання перших двох методів. Якщо при ущільненні даних змінюється лише їх структура, то вони можуть бути повністю відновлені.

2.1 Алгоритми ущільнення без втрат

Run Length Encoding

Run Length Encoding (RLE) – один з найстаріших і найпростіших алгоритмів ущільнення даних [7]. При застосуванні цього алгоритму до графічних даних, вони подаються як потік байтів. Растрове зображення, яке зберігається у вигляді матриці байтів, перетворюється на один ланцюг байтів – наприкінці першого рядка матриці «прикріплюється» початок другого і т. д. Ущільнення в цьому алгоритмі відбувається за рахунок того, що зустрічаються послідовності однакових байтів. Замість того, щоб записувати послідовно одне й те ж саме значення, алгоритм записує кількість повторів цього значення (лічильник) і один раз саме значення:

1 45 45 45 45 45 45 7 9 5 → 1 6 45 7 9 5

Проте треба якось розрізняти між собою лічильники і безпосередні значення. Це реалізовано наступним чином: закодована послідовність починається зі службового байту, перший біт якого дорівнює 1, якщо після нього буде записано «згорнуте значення», або 0, якщо далі будуть записані байти, що не є однаковими. Останні 7 бітів цього службового байту, у першому випадку, визначають кількість повторів «згорнутого значення», яке записано в наступному байті, або кількість байтів, яку треба зчитати до наступного службового байту (ці всі байти будуть різними) (рис. 2.2).

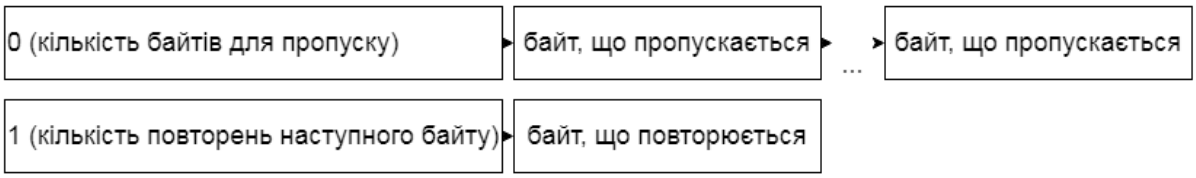

Рисунок 2.2 – Дві можливі послідовності в закодованій стрічці

Наведена вище стрічка буде закодована таким чином, як показано на рис. 2.3.

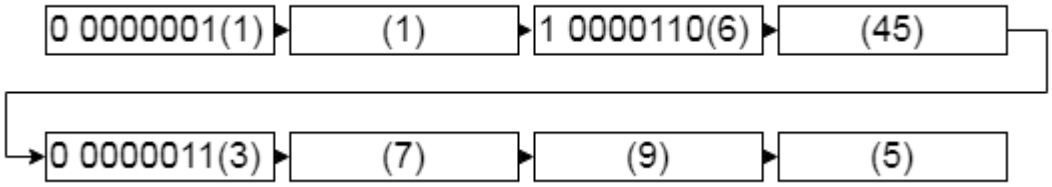

Рисунок 2.3 – послідовність байтів, закодована алгоритмом RLE

Початкові дані займали $8(\text{бітів}) * 10(\text{символів}) = 80$ бітів.

Після ущільнення дані стали займати 64 біти.

У найкращому випадку, цей алгоритм може зменшити розмір файлу в 64 рази. Але, якщо його модифікувати та замість одного службового байту використовувати два, то файл можна зменшити в 16384 рази. Треба розуміти,

що такий результат досягається лише за умови, що вихідний файл складається як мінімум з 32768 (2^{15}– саме 15 бітів будуть відповідати за кількість повторень значень в послідовності) і всі біти в ньому однакові. Тобто, це однокольорова картинка. У випадках з маленькими картинками, той факт, що службових байтів 2, а не 1, може викликати надлишковість, і сильно збільшити файл, а не зменшити.

RLE набув популярності в діловій та науковій графіці, він використовується для ущільнення простих зображень, наприклад, іконок програм.

Деякі алгоритми ущільнення даних використовують алгоритм RLE в якості свого останнього етапу. Інші етапи ущільнення призводять до того, що з’являється багато послідовностей з однаковими значеннями, які ефективно і швидко обробляються алгоритмом RLE.

Сімейство алгоритмів LZ*

LZ* – сімейство алгоритмів словникового ущільнення даних [7], [8]. Назву отримало за ініціалами двох дослідників – Абрахама Лемпела і Якоба Зіва, які розробили в 1970 році алгоритми LZ77 і LZ78. На базі LZ77 і LZ78, змінюючи і комбінуючи їх з іншими методами, дослідниками створено ряд інших алгоритмів – LZO, LZS, LZW, LZC, LZMA. Словниковими алгоритми називають тому, що в процесі ущільнення вони розбивають дані на частини, записують ці частини в «словник» та замінюють відповідні дані на індекси.

Найбільш поширеними є LZ77, LZ78 та LZW. Можна сказати, що алгоритми сімейства LZ* являють собою більш складне узагальнення простого й інтуїтивного способу стиснення даних, використовуваного в RLE.

Розглянемо докладніше алгоритм LZW, який розроблено у 1984 році Террі Велчем, як покращена модифікація алгоритму LZ78. Алгоритм формально можна описати наступними кроками:

1) Ініціалізація словника всіма односимвольними фразами. Вхідна фраза *W* ініціалізується першим символом повідомлення.
2) Читання чергового символу *X* з повідомлення. Якщо це символ кінця повідомлення, то видати індекс фрази *W* на вихід та закінчити роботу.
3) Інакше, якщо в словнику вже є фраза *WX*, тоді *W* := *WX*, перейти до кроку 2.
4) Інакше, видати індекс фрази W на вихід, записати в словник *WX*, *W* := *X*, перейти до кроку 2.

Головна ідея полягає в тому, що можна замінити фразу будь-якої довжини індексом, який буде складатися з одного або кількох байтів. Якщо ця фраза повторюється у вхідних даних декілька разів, то це дає ще більше виграшу.

Ще однією важливою особливістю є те, що сам словник не треба включати в закодовані дані. Алгоритм декодування може відновити його, імітуючи роботу алгоритму кодування, знаючи лише початкову ініціалізацію словника.

Розглянемо процес кодування вхідної послідовності на прикладі. Для спрощення, припустимо, що весь алфавіт складається лише з великих літер від A до Z, та символу закінчення повідомлення #.

Закодуємо повідомлення FAMFAMFAMFAM#.

Ініціалізація словника. Маємо 27 символів алфавіту, тому для кодування слів в словнику вистачить 5 бітів. В процесі ущільнення словник буде доповнюватись, і, починаючи з 33 слова, слова вже будуть кодуватись шістьма бітами. З цього моменту всі слова, що вже були записані в словнику до цього у вигляді 5 бітів, слід подавати на вихід у вигляді 6 бітів. Це обумовлено тим, що алгоритм декодування також з цього моменту почне зчитувати по 6 бітів в якості одного символу. Стан словника та буферу після ініціалізації:

Словник: 00000: #, 00001: **A**, 00010: **B**, … , 11010: **Z**.

W: F

Читання чергового символу – *X*: A

1) Фрази FA в словнику немає, тому видаємо індекс *W* на вихід, додаємо FA в словник, присвоюємо символ A буферу *W*, та переходимо до зчитування нового символу.

Вихід: 00110(F)

Словник: … , 11010(26): **Z**, 11010(27): **FA**.

W: A

Читання чергового символу – *X*: M

2) Фрази AM в словнику немає, тому видаємо індекс *W* на вихід, додаємо AM в словник, присвоюємо символ M буферу *W*, та переходимо до зчитування нового символу.

Вихід: 00110(F)00001(A)

Словник: … , 11010(26): **Z**, 11011(27): **FA**, 11100(28): **AM**.

W: M

Читання чергового символу – *X*: F

3) Фрази MF в словнику немає, тому видаємо індекс *W* на вихід, додаємо MF в словник, присвоюємо символ F буферу *W*, та переходимо до зчитування нового символу.

Вихід: 00110(F)00001(A)01101(M)

Словник: … , 11011(27): **FA**, 11100(28): **AM**, 11101(29): **MF**.

W: F

Читання чергового символу – *X:* A

4) Фраза FA в словнику вже є, тому переходимо до зчитування нового символу.

Читання чергового символу – *X:* M

5) Фрази FAM в словнику немає, тому видаємо індекс *W* на вихід, додаємо FAM в словник, присвоюємо символ M буферу *W*, та переходимо до зчитування нового символу.

Вихід: 00110(F)00001(A)01101(M)11011(FA)

Словник: … , 11011(27): **FA**, 11100(28): **AM**, 11101(29): **MF**,

11110(30): **FAM**.

W: M

Читання чергового символу – *X:* F

6) Фраза MF в словнику вже є, тому переходимо до зчитування нового символу.

Читання чергового символу – *X:* A

7) Фрази MFA в словнику немає, тому видаємо індекс *W* на вихід, додаємо MFA в словник, присвоюємо символ A буферу *W*, та переходимо до зчитування нового символу.

Вихід: 00110(F)00001(A)01101(M)11011(FA)11101(MF)

Словник: … , 11011(27): **FA**, 11100(28): **AM**, 11101(29): **MF**,

11110(30): **FAM**, 11111(31): **MFA**.

W: A

Читання чергового символу – *X:* M

8) Фраза AM в словнику вже є, тому переходимо до зчитування нового символу.

Читання чергового символу – *X:* F

9) Фрази AMF в словнику немає, тому видаємо індекс *W* на вихід, додаємо AMF в словник, присвоюємо символ F буферу *W*, та переходимо до зчитування нового символу.

Вихід: 00110(F)00001(A)01101(M)11011(FA)11101(MF)11100(AM)

Словник: … , 11011(27): **FA**, 11100(28): **AM**, 11101(29): **MF**,

11110(30): **FAM**, 11111(31): **MFA**, 100000(32): **AMF**.

W: F

Читання чергового символу – *X:* A

10) Фраза FA в словнику вже є, тому переходимо до зчитування нового символу.

Читання чергового символу – *X:* M

11) Фраза FAM в словнику вже є, тому переходимо до зчитування нового символу.

Читання чергового символу – *X:* #

12) Це символ кінця повідомлення. Видаємо на вихід індекс *W* та завершуємо роботу.

Вихід:

00110(F)00001(A)01101(M)11011(FA)11101(MF)11100(AM)011110(FAM)000000(#)

Початкові дані займали $5(\text{бітів}) * 13(\text{символів}) = 65$ бітів.

Після ущільнення дані стали займати $5 * 6 + 6 * 2 = 42$ біта.

Deflate

Алгоритм Deflate ([9] – [11]) реалізує компресію в два окремі етапи.

Перший етап – це ущільнення даних за допомогою алгоритму LZ77. Ідея, що лежить в основі алгоритму LZ77, полягає в заміні підрядка якомога більшої довжини посиланням на місце в рядку, де цей підрядок вже зустрічався. Посилання кодується трьома параметрами: зміщення (кількість позицій, на яку потрібно перемістити рядком назад, щоб потрапити на початок першого входження підрядка), довжина знайденого підрядка, значення наступного символу після знайденого підрядка.

Наприклад, кодування послідовності FAMFAMFAMFAM буде виглядати таким чином:

1) Перші три символи FAM будуть переписані у результуючу послідовність у такому ж вигляді.

2) Після того, як зчитування дійде до 4 символу, алгоритм розпізнає F, як підрядок, який вже зустрічався раніше. Він зчитає наступний символ та проведе пошук в пройденій частині підрядка FA. Результат буде позитивним. У підсумку, алгоритм визначить, що підрядок FAM – максимальний підрядок, що зустрічався раніше, наступний за ним символ це F, а зміщення відносно початку знайденого підрядка дорівнює 3 (рис. 2.4).

Рисунок 2.4 – Заміна підрядка на посилання

Результуюча послідовність буде тепер складатись з таких елементів: F, A, M, (3, 3, F).

3) В процесі зчитування даних далі вже підрядок FAMFAM буде ідентифіковано як той, що зустрічався раніше. Результуюча послідовність: F, A, M, (3, 3, F), (6, 6, ‘EOF’).

Для підвищення ефективності при роботі з великими даними був використаний принцип «ковзаючого вікна» – пошук однакових підрядків відбувається не у всій частині рядку до поточної позиції, а тільки серед обмеженої кількості останніх символів. Зазвичай розмір «вікна», тобто кількість символів, серед яких проводиться пошук, становить 2, 4 або 32 Кб. Але це також породжує певні недоліки. Два однакові підрядки, які знаходяться один від одного на відстані, що перебільшує розмір «вікна», не

будуть порівняні, тому друге входження підрядка не буде замінене посиланням на його перше входження. Крім того, максимально можлива довжина підрядків, що порівнюються, не може бути більшою ніж розмір «вікна».

Другий етап – кодування послідовності, отриманої після першого етапу, алгоритмом Хаффмана. Алгоритм Хаффмана – це жадібний алгоритм оптимального префіксного кодування алфавіту з мінімальною надлишковістю. Алгоритм Хаффмана перетворює дані в код Хаффмана, який розроблено у 1952 році Девідом Хаффманом. Ідея полягає в тому, що, знаючи ймовірність появи символів в повідомленні, можна закодувати ці символи кодами змінної довжини, де символи, ймовірність появи яких більша, будуть закодовані більш короткими кодами. При цьому код Хаффмана має властивість префіксності. Це означає, що кожне кодове слово не є префіксом іншого, тому їх можна однозначно декодувати, не зважаючи на різну довжину.

Класичний алгоритм Хаффмана отримує на вході таблицю частот входжень символів в повідомлення. Після чого на основі цієї таблиці будується дерево кодування Хаффмана за таким алгоритмом:

1) Символи вхідного алфавіту представляються списком листів дерева. Кожен лист має свою вагу, яка дорівнює або ймовірності виникнення відповідного символу в повідомленні, або кількості входжень.
2) Обираються два вузли з найменшою вагою (ті, що зустрічаються найменшу кількість разів).
3) Створюється вузол дерева, що є батьком цих двох листів. Його вага дорівнює сумі ваг листів. Одному ребру, що виходить із створеного вузла, ставиться у відповідність 0, іншому – 1. Новий вузол додається до списку листів дерева, а два його нащадки видаляються. Відповідно, список зменшиться на один елемент.

4) Кроки 2-3 повторюються до тих пір, поки в списку листів не залишиться лише один елемент. Він буде коренем дерева.

Розглянемо кодування на прикладі повідомлення TOBEORNOTTOBE.

Складемо список листів дерева з їх вагою:

T – 3, O – 4, B – 2, E – 2, R – 1, N – 1.

1) Найменшу вагу мають R та N. Створюємо новий вузол RN – 2

   Дерево:

   RN:

      0: R

      1: N

   Список листів: T – 3, O – 4, B – 2, E – 2, RN – 2.

2) Найменшу вагу мають B та E. Створюємо новий вузол BE – 4

   Дерево:

   RN:

      0: R

      1: N

   BE:

      0: B

      1: E

   Список листів: T – 3, O – 4, BE – 4, RN – 2.

3) Найменшу вагу мають T та RN. Створюємо новий вузол TRN – 5

   Дерево:

   TRN:

      0: RN:

            0: R

            1: N

      1: T

   BE:

      0: B

1: E

Список листів: TRN – 4, O – 4, BE – 4.

4) Найменшу вагу мають TRN та O. Створюємо новий вузол TRNO – 8

Дерево:

TRNO:

0: TRN:

0: RN:

0: R

1: N

1: T

1: O

BE:

0: B

1: E

Список листів: TRNO – 8, BE – 4.

5) Найменшу вагу мають TRNO та BE. Створюємо новий вузол TRNOBE – 12

Дерево:

TRNOBE:

0: TRNO:

0: TRN:

0: RN:

0: R

1: N

1: T

1: O

1: BE:

0: B

1: E

Список листів: TRNOBE – 12. Вузол TRNOBE і є коренем дерева.

Згідно цього дерева можна скласти код для кожного символу:

T – 001, O – 01, B – 10, E – 11, R – 0000, N – 0001.

Алгоритм Хаффмана має ряд суттєвих недоліків. Наприклад, для відновлення закодованого повідомлення декодер має отримати таблицю частот. Довжина цієї таблиці може збільшити закодовані дані настільки, що ступінь ущільнення сильно зменшиться, або взагалі стане нульовою. Крім того, для алфавітів, ентропія яких не перевищує 1 (наприклад, двійковий алфавіт), застосування алгоритму Хаффмана не має сенсу.

Алгоритм Deflate набув значної популярності і використовується для компресії будь-який даних. Саме він використовується в архіваторах ZIP, gzip, а також PNG-зображеннях.

2.2 Алгоритми ущільнення без втрат

JPEG

Алгоритм JPEG [7], [12], [14] – це відносно новий алгоритм, який наразі є найпопулярнішим алгоритмом ущільнення зображень. Фактично він є стандартом для повноколірних зображень. Алгоритм розроблено групою експертів підрозділу ISO – Joint Photographic Expert Group (JPEG) – спеціально для ущільнення 24-бітних зображень. Найбільш придатний для реалістичних зображень з плавними переходами кольору та яскравості. Алгоритм складається з декількох кроків:

1) Перехід від колірної моделі RGB до YCrCb. В цій моделі компонента Y відповідає за яскравість, Cr – хроматичний червоний колір, Cb – хроматичний синій колір. За рахунок того, що око людини більш чуттєве до яскравості ніж до кольору, можна ущільнювати масиви компонент Cr, Cb з більшими втратами, що дозволить отримати більший коефіцієнт

ущільнення. Після переходу до іншої моделі відбувається «проріджения». У кожному блоці 2x2 беруться компоненти Cr, Cb з кожного пікселя та замінюються на середні їхні значення.

2) Вихідне растрове зображення розбивається на матриці 8x8 пікселів. Через те, що на таких ділянках колір змінюється плавно, до них можна ефективно застосувати ДКП.
3) Над обробленими блоками 8x8 проводиться квантування. Кожен з коефіцієнтів матриці ділиться на певне число. Дробові значення округлюються до цілих. Після округлення багато коефіцієнтів матриці стають нулями, що дозволяє значно скоротити об’єм даних на наступному етапі. Налаштування якість-ущільнення, які задає користувач, найбільше визначаються саме цим етапом. Чим більше заданий ступінь ущільнення, тим більше значень в матриці перетвориться на нулі.
4) Зображення (яке має вигляд матриці) перетворюється в одновимірний масив діагональним обходом (рис. 2.5.) [13]. Після цього отриманий масив ущільнюється алгоритмом Хаффмана.

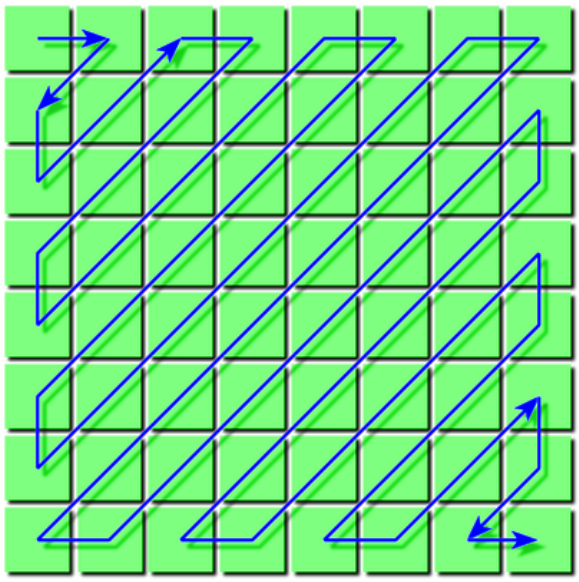

Рисунок 2.5 – Діагональний обхід матриці

Незважаючи на те, що алгоритм JPEG використовується майже всюди для роботи з фотореалістичними зображеннями та демонструє великий ступінь ущільнення, він має декілька суттєвих недоліків. Чим більший ступінь ущільнення, тим більш явно видно на зображенні так звані артефакти. Зображення ніби розділяється на квадрати. Також з'являються ореоли на межах різких переходів кольору.

Існує похідний від JPEG алгоритм – JPEG2000. Замість дискретного косинусного перетворення (ДКП), в ньому використовується технологія вейвлет-перетворення. Після компресії цим методом зображення залишається більш чітким та гладким, а розділення на квадрати і поява артефактів не спостерігається. При цьому розмір файлу при після використання цього алгоритму менший, ніж після використання JPEG, при однаковій якості зображення.

Представлений вище аналіз існуючих алгоритмів ущільнення графічних даних доводить необхідність написання алгоритму ущільнення та відповідно на його основі програми, яка дозволить здійснювати швидке зменшення обсягів графічних даних без втрат при наступному відновленні цих даних.

3 ВИБІР ЗАСОБІВ РОЗРОБЛЕННЯ

Мова програмування C++ – мова програмування високого рівня, яка також має властивості низькорівневих мов. Підтримує такі парадигми програмування як процедурне програмування, об’єктно-орієнтоване, узагальнене. C++ успадковано від мови C, тому вона має сумісність з C, але все ж таки C++ це не просто надмножина C. Так існують програми, написані мовою C, що не можуть бути трансльовані компілятором C++.

C++ – це мова зі статичною типізацією, тобто кожна змінна чи параметр функції зв’язується з конкретним типом при оголошенні та не може бути змінена пізніше. Це дозволяє генерувати найбільш простий машинний код, що суттєво впливає на швидкість роботи програми. До того ж в C++ існує потужний механізм приведення типів, який дозволяє значення змінних одного типу приводити до схожого типу. Наприклад, char до int. Основною рисою, що відрізняє C++ від її попередника C, є підтримка ООП. Мова C++ – машинно-незалежна, хоча й залежить від ОС, на якій компілюється. Наприклад, скомпільовані програми на Windows, не можна запустити на Linux, та навпаки. C++ має велику кількість бібліотек, що розширюють можливості програмістів та спрошують їх роботу. В основному, вони містять деякі алгоритми, також структури даних, в тому числі й динамічні контейнери, наприклад vector, list, тощо. C++ надає програмісту прямий доступ до адрес пам’яті, що дозволяє працювати з нею на дуже низькому рівні. З іншого боку, це може призвести до великої кількості помилок.

C++ застосовується для розробки операційних систем, драйверів пристроїв, прикладних програм, високоефективних серверів, ігор, вбудованих та хмарних систем, роботи з базами даних [15].

4 РОЗРОБЛЕННЯ ПРОГРАМИ УЩІЛЬНЕННЯ ГРАФІЧНИХ ДАНИХ

4.1 Опис алгоритму ущільнення графічних даних

В існуючій комп’ютерній графіці значну частину займають звичайні фотографії. Сучасні фотоапарати здатні створювати деталізовані та якісні зображення. З кожним роком роздільна здатність екранів збільшується, разом з цим збільшується і кількість пікселів в зображеннях.

Фотореалістичні зображення зазвичай містять багато плавних переходів (рис. 4.1).

Рисунок 4.1 – Зображення з плавними переходами

Тобто, якщо розглядати матрицю растрового зображення, то можна побачити багато послідовностей, що складаються з пікселів, близьких один до одного за значеннями компонент. Розглянемо найпоширеніший варіант зображень – 24-бітний колір з колірною моделлю RGB.

На рис. 4.2 можна побачити, що деякі пікселі візуально навіть не відрізняються, деякі зовсім однакові, а деякі відрізняються на маленьке значення.

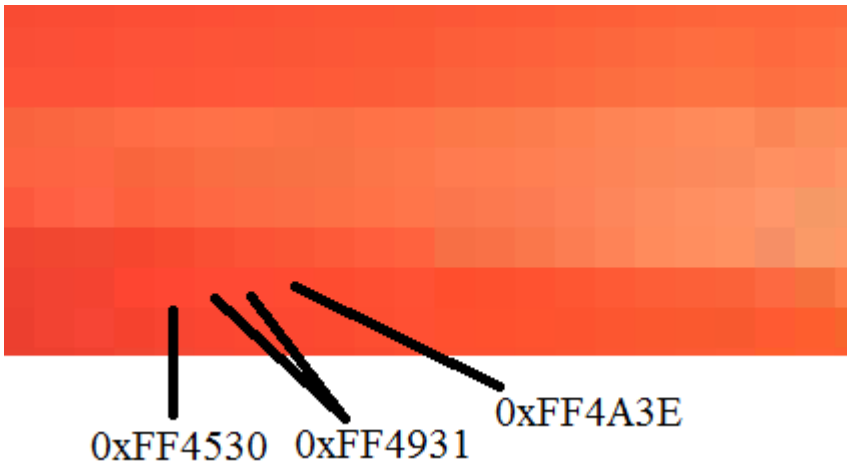

Рисунок 4.2 – зображення у наближеному вигляді

Саме плавні переходи, а також велика загальна кількість пікселів створюють таку властивість, коли пікселі, що знаходяться поруч, майже не відрізняються.

Основна ідея алгоритму полягає в тому, що схожі пікселі можна кодувати різницями відносно якогось одного пікселя. При цьому, різниці мають бути обмеженого значення, що дозволить записувати їх меншою кількістю біт, ніж звичайне значення пікселю. Зараз кожен піксель займає 24 біти, по 8 на кожну компоненту кольору. При кодуванні різницями, на кожен піксель можна витрачати, наприклад 12 бітів. Однак залишається незмінним піксель, відносно якого вираховується різниця. Крім того, додається ще певна кількість бітів, яка позначає, скільки саме наступних пікселів закодовано відносно поточного – лічильник.

Очевидно, що можна отримати більш вигідні послідовності, якщо розглядати зображення не як послідовність пікселів, а розкласти кожен з них на компоненти. Тобто, окремо закодувати послідовність синьої компоненти

кожного пікселя, окремо послідовність зеленої, та окремо послідовність червоної.

Наприклад, беручи до уваги лише 4 пікселі, що позначені на рисунку 4.2, послідовність пікселів FF 45 30 FF 49 31 FF 49 31 FF 4A 3E перетвориться на послідовності FF FF FF FF, 45 49 49 4A, 30 31 31 3E.

Також, для збільшення швидкості та простоти алгоритму, послідовності, що є результатом кодування якогось байту, але при цьому займають меншу кількість бітів, повинні мати довжину, що є дільником числа 8 – саме стільки бітів в одному байті. Наприклад, різниця між якимись двома байтами вхідної послідовності може складатись з 4 бітів, тоді одним байтом можна закодувати дві таких різниці. Якщо вони будуть складатись, наприклад, з 5 або 6 бітів, це створить необхідність в процедурі, яка при декодуванні повинна буде розділяти вхідні байти на різну кількість бітів і складати з них результуючі байти, а вже після цього починати сам алгоритм декодування. На зображеннях з великою кількістю пікселів це може значно вплинути на швидкість роботи алгоритму. Якщо ж кожен вхідний байт буде містити в собі два значення по 4 біти, то достатньо лише застосувати до нього порозрядну операцію І зі значеннями 240 (11110000) та 15 (00001111), після чого перше отримане значення змістити вправо на 4 розряди. Отримані два байти і будуть результатом. Такий підхід буде застосовуватись однаково до кожного такого вхідного байту, що значно спрощує та пришвидшує алгоритм.

Якщо розглядати одну з наведених вище послідовностей 16-розрядних даних 45 49 49 4A, то при кодуванні вона перетвориться в наступну:

45 – головний байт, від якого буде вираховуватись різниця з іншими байтами. 03 – лічильник, який вказує, яка кількість послідовних байтів відрізнялась від головного не більше ніж на задане значення. 4 – різниця між другим байтом в послідовності та головним. 4 – різниця між третім байтом в послідовності та головним. 5 – різниця між четвертим байтом в послідовності

та головним. В підсумку маємо такі байти: 45 03 4 4 5. Припустимо, що різниця складається з 4 біт, тому їх можна склеювати по дві – 45 03 44 50 (до 5 справа приклеєно 0).

З цього прикладу добре видно, що даний алгоритм треба застосовувати до підпослідовностей, де будуть схожими мінімум 6 байтів підряд. В інших випадках, через те, що один байт залишається без змін, а також додається байт лічильника, зменшення розміру даних немає, навпаки може відбутись збільшення.

Описаний вище алгоритм доцільно використовувати в комбінації з якимось іншим алгоритмом. Другим етапом кодування буде адаптивний алгоритм Хаффмана. Цей алгоритм, на відміну від класичного алгоритму Хаффмана, не потребує створення таблиці частот. При проході вхідною послідовністю він почергово додає зчитані символи до дерева, змінюючи в ньому ваги вузлів та, при необхідності, переставляє вузли. Необхідність в перестановці вузлів виникає тоді, коли порушується умова впорядкованості вузлів. Декодер не потребує таблиці частот, він також починає обробляти отриману послідовність, маюче пусте дерево, і виконує ті ж самі процедури, що і кодер – додавання зчитаного символу до дерева та оновлення дерева [16], [17].

Алгоритм роботи програми ущільнення складається з таких дій:

- Зчитування даних з вхідного файлу (файл формату bmp).
- Перший етап кодування – кодування даних розробленим алгоритмом.
- Другий етап кодування – кодування адаптивним алгоритмом Хаффмана.
- Запис закодованих даних у вихідний файл.

Алгоритм відновлення даних складається з таких дій:

- Зчитування даних з вхідного файлу (закодований файл).

- Перший етап декодування – декодування адаптивним алгоритмом Хаффмана.
- Другий етап декодування – декодування даних розробленим алгоритмом.
- Запис закодованих даних у вихідний файл.

Розроблений алгоритм кодування складається з таких кроків:

1) Змінні byte1, byte2 ініціалізуються першими двома байтами з вхідного потоку, лічильник ініціалізується значенням 2.
2) Якщо різниця byte2 та byte1 менша за задану межу, тоді до буферу додається знайдена різниця. Інакше перехід до кроку 4.
3) Якщо розмір буфера досяг максимального, перехід до кроку 4.
   Якщо лічильник менший розміру вхідного потоку (тобто, ще не кінець потоку), читання чергового байту з потоку в змінну byte2 і перехід до кроку 2. Інакше, перехід до кроку 4.
4) Якщо буфер різниць не пустий, у вихідний потік записується розмір буферу зі старшим бітом 1, значення змінної byte1 та «склеєні» попарно різниці з буферу.
   Якщо лічильник дорівнює розміру вхідного потоку, запис в вихідний потік числа 1, та значення змінної byte2. Кінець роботи.
   Інакше, значення змінної byte2 присвоюється змінній byte1. Читання чергового байту з потоку в змінну byte2. Очищення буферу.
5) Якщо різниця byte2 та byte1 не менша за задану межу, тоді до буферу додається значення byte1 та значення змінної byte2 присвоюється змінній byte1. Інакше, перехід до кроку 7.
6) Якщо розмір буфера досяг максимального, перехід до кроку 7.
   Якщо лічильник менший за розмір вхідного потоку, читання чергового байту з потоку в змінну byte2 і перехід до кроку 5. Інакше, перехід до кроку 7.

7) Якщо буфер різниць не пустий, в вихідний потік записується розмір буферу зі старшим бітом 0 та всі значення з буферу. Очищення буферу.

   Якщо лічильник дорівнює розміру вхідного потоку, запис в вихідний потік числа 1, та значення змінної byte2. Кінець роботи.

   Інакше, перехід до кроку 2.

При кодування треба враховувати декілька важливих деталей:

1) Через те, що у вихідному потоці лічильник схожих байтів або байтів, що відрізняються, займає один байт, його значення має бути в межах від 1 до 127. Старший біт є індикатором того, що наступні байти будуть або різницями, або оригінальними значеннями байтів з вхідного потоку. Можна використати значення 0, як 128, таким чином, максимальне значення лічильника та буферів, що наповнюються значеннями під час роботи алгоритму, буде 128.
2) Закодовані різниці мають містити знаковий біт, який буде визначати, що черговий байт більший або менший за основний. В одному закодованому байті треба розмістити два значення різниць, тому на кожне з них відводиться по 4 біти, один з яких є знаковим. Тому, межа різниці між двома байтами буде дорівнювати 8. Два байти можна назвати схожими один з одним, якщо різниця між ними менша за 8.

Розглянемо роботу алгоритму на прикладі послідовності 16-розрядних даних 05 05 05 06 04 05 14 14 14 20 30 40 50 60 (кожні два символи – це один байт):

1) byte1: 05

   byte2: 05

   Знаходимо різницю:

   $|\text{byte2} - \text{byte1}| = 0 < 8 \rightarrow \text{buf: } 0$

2) Читаємо наступний байт:

   byte2: 05

   Знаходимо різницю:

   $|\text{byte2} - \text{byte1}| = 0 < 8 \rightarrow \text{buf: } 0, 0$

3) Читаємо наступний байт:

   byte2: 06

   Знаходимо різницю:

   $|\text{byte2} - \text{byte1}| = 1 < 8 \rightarrow \text{buf: } 0, 0, 1$

4) Читаємо наступний байт:

   byte2: 04

   Знаходимо різницю:

   $|\text{byte2} - \text{byte1}| = 1 < 8 \rightarrow \text{buf: } 0, 0, 1, 9(00001001)$ додано одиничний біт який вказує на від'ємне значення.

5) Читаємо наступний байт:

   byte2: 05

   Знаходимо різницю:

   $|\text{byte2} - \text{byte1}| = 0 < 8 \rightarrow \text{buf: } 0, 0, 1, 9, 0$

6) Читаємо наступний байт:

   byte2: 14

   Знаходимо різницю:

   |byte2 – byte1| = F > 8

7) result: 85(1 000 0101 – лічильник з старшим бітом 1), 05(основний байт),00 (0000 0000),19(0001 1001), 00(0000 0000-приклеєний зайвий нуль, який при декодуванні не буде братися до уваги).

   byte1: 14

   byte2: 14

8) Знаходимо різницю:

|byte2 – byte1| = 0 < 8 → buf: 0

9) Читаємо наступний байт:

byte2: 14

Знаходимо різницю:

|byte2 – byte1| = 0 < 8 → buf: 0,0

10) Читаємо наступний байт:

byte2: 20

Знаходимо різницю:

|byte2 – byte1| = C > 8

11) result: 85 05 00 19 00 82(1 000 0010 – лічильник з старшим бітом 1), 14(основний байт), 00 (0000 0000)

byte1: 20

byte2: 30

12) Знаходимо різницю:

|byte2 – byte1| = 10 > 8 → buf: 20

byte1: 30

13) Читаємо наступний байт:

byte2: 40

Знаходимо різницю:

|byte2 – byte1| = 10 > 8 → buf: 20,30

byte1: 40

14) Читаємо наступний байт:

byte2: 50

Знаходимо різницю:

|byte2 – byte1| = 10 > 8 → buf: 20,30,40

byte1: 50

15) Читаємо наступний байт:

byte2: 60

Знаходимо різницю:

|byte2 – byte1| = 10 > 8 → buf: 20,30,40,50

byte1: 60

Кінець вхідного потоку.

16) result: 85 05 00 19 00 82 14 00 04(лічильник) 20 30 40 50 01 (лічильник останнього байту) 60(останній байт)
17) Кінець роботи.

Вхідна послідовність: 05 05 05 06 04 05 14 14 14 20 30 40 50 60

Результат після кодування: 85 05 00 19 00 82 14 00 04 20 30 40 50 01 60

У цьому випадку, ущільнена послідовність стала більшою, ніж початкова, через те, що вона дуже мала, та довжини послідовностей схожих байтів дуже короткі. В реальних фотографіях, особливо в тих, які мають багато плавних переходів, ці послідовності часто складаються з 128 і більше байтів. На цьому прикладі продемонстровано лише алгоритм кодування, зробити повну оцінку алгоритму за цими даними неможливо. Можна зазначити, що він не є ефективним при роботі з дуже маленькими зображеннями, а також у тих випадках, коли пікселі, що сильно відрізняються кольорами один від одного, будуть чергуватись групами в кількості від 2 до 5.

Розроблений алгоритм декодування складається з таких кроків:

1) Якщо лічильник менший розміру вхідного потоку, читання чергового байту з вхідного потоку в змінну count. Інакше, кінець роботи.
2) Якщо старший біт зчитаного байту дорівнює 0, перехід до кроку 3. Інакше перехід до кроку 4.
3) Читання чергового символу з вхідного потоку та запис його в вихідний потік. Зменшення count на 1. Якщо count більший за 0, повторення кроку 3. Інакше, перехід до кроку 1.

4) Зміна старшого біта змінної count на 0. Читання чергового символу з вхідного потоку в змінну main_byte. Запис main_byte до вихідного потоку.
5) Читання чергового байту з вхідного потоку. Запис у вихідний потік суми main_byte та старших 4 бітів зчитаного байту (з урахуванням знакового біту). Зменшення count на 1.
6) Якщо count більший за 0, запис у вихідний потік суми main_byte та молодших 4 бітів зчитаного байту (з урахуванням знакового біту). Зменшення count на 1.

   Якщо count більший за 0, повторення кроку 5. Інакше, перехід до кроку 1.

Розглянемо роботу алгоритму на прикладі послідовності, отриманої в результаті виконання алгоритму кодування: 85 05 00 19 00 82 14 00 04 20 30 40 50 01 60

1) count: 85

   старший біт: 1
2) count: 05

   Читаємо наступний байт:

   main_byte: 05

   result: 05
3) Читаємо наступний байт:

   buf: 00

   result: 05, 05

   count: 04
4) $\text{count} > 0 \rightarrow$ result: 05, 05, 05

   count: 03
5) Читаємо наступний байт:

   buf: 19

   result: 05, 05, 05, 06

count: 02

6) count > 0 → result: 05, 05, 05, 06, 04

   count: 01

7) Читаємо наступний байт:

   buf: 00

   count > 0 → result: 05, 05, 05, 06, 04, 05

   count: 00

8) Читаємо наступний байт:

   count: 82

   старший біт: 1

9) count: 02

   Читаємо наступний байт:

   main_byte: 14

   result: 05, 05, 05, 06, 04, 05, 14

10) Читаємо наступний байт:

    buf: 00

    count > 0 → result: 05, 05, 05, 06, 04, 05, 14, 14

    count: 01

11) count > 0 → result: 05, 05, 05, 06, 04, 05, 14, 14, 14

    count: 00

12) Читаємо наступний байт:

    count: 04

    старший біт: 0

13) count > 0

    Читаємо наступний байт:

    result: 05, 05, 05, 06, 04, 05, 14, 14, 14, 20

    count: 03

14) count > 0

    Читаємо наступний байт:

result: 05, 05, 05, 06, 04, 05, 14, 14, 14, 20, 30

count: 02

15) $count > 0$

Читаємо наступний байт:

result: 05, 05, 05, 06, 04, 05, 14, 14, 14, 20, 30, 40

count: 01

16) $count > 0$

Читаємо наступний байт:

result: 05, 05, 05, 06, 04, 05, 14, 14, 14, 20, 30, 40, 50

count: 00

17) Читаємо наступний байт:

count: 01

старший біт: 0

18) $count > 0$

Читаємо наступний байт:

result: 05, 05, 05, 06, 04, 05, 14, 14, 14, 20, 30, 40, 50, 60

count: 00

Кінець роботи.

Вхідна послідовність: 85 05 00 19 00 82 14 00 04 20 30 40 50 01 60

Результат після декодування: 05 05 05 06 04 05 14 14 14 20 30 40 50 60

4.2 Опис програми ущільнення даних

Програма складається з класу Compressor. Він має два загально-доступних методи:

- Compressor(const char* filename, MODE mode) – конструктор, який ініціалізує деякі поля початковими значеннями.

- void run(MODE mode) – цей метод приймає параметр користувацького enum типу MODE, який визначає, в якому режимі запущена програма – компресії чи декомпресії. Викликає один з приватних методів compress() або decompress() в залежності від режиму роботи.

Програма передбачає роботу з файлами формату bmp. Перед тим, як зчитувати з вхідного файлу значення, необхідно правильно зчитати заголовки файлу. Структура файлу bmp наведена в табл. 4.1 [18].

Таблиця 4.1 – Структура файлу bmp

Назва поля	Обов'язкове	Розмір	Призначення
Заголовок файлу (Bitmap file header)	так	14 байтів	Основна інформація про файл
Інформаційний заголовок (DIB header)	так	Фіксований розмір (існує 7 різних варіантів)	Детальна інформація про файл та формат пікселів
Додаткові бітові маски (Extra bit masks)	ні	12 або 16 байтів	Визначення формату пікселів
Таблиця кольорів (Color table)	ні	Нефіксований розмір	Опис кольорів, використаних в масиві пікселів
Розрив1 (Gap1)	ні	Нефіксований розмір	Структурне вирівнювання перед початком масиву пікселів

Продовження таблиці 4.1

Назва поля	Обов’язкове	Розмір	Призначення
Масив пікселів (Pixel array)	так	Нефіксований розмір	Значення пікселів зображення
Розрив2 (Gap2)	ні	Нефіксований розмір	Структурне вирівнювання перед початком ICC-профіля
ICC-профіль (ICC color profile)	ні	Нефіксований розмір	Дані, що характеризують пристрій кольорового вводу/виводу

У більшості 24-бітних зображень використовуються тільки обов’язкові поля та інформаційний заголовок BITMAPINFOHEADER.

Структура файлового заголовку наведена в табл. 4.2 [18].

Таблиця 4.2 – Структура заголовку файлу

Зміщення (hex)	Розмір	Призначення
00	2 байти	Ідентифікатор bmp файлу
02	4 байти	Розмір bmp файлу в байтах
06	2 байти	Зарезервовано. Значення залежить від додатку, в якому було створено зображення
08	2 байти	Зарезервовано. Значення залежить від додатку, в якому було створено зображення
0A	4 байти	Зміщення масиву пікселів в файлі.

Для збереження вмісту цього заголовку в классі Compressor створено структуру BMPFileHeader з відповідними полями.

Структура заголовку BITMAPINFOHEADER наведена в табл. 4.3 [18].

Таблиця 4.3 – Структура заголовку BITMAPINFOHEADER

Зміщення (hex)	Розмір	Призначення
0E	4 байти	Розмір цього заголовку (40 байтів)
12	4 байти	Ширина зображення в пікселях
16	4 байти	Висота зображення в пікселях
1A	2 байти	Кількість колірних площин (1)
1C	2 байти	Глибина кольору (кількість бітів на піксель)
1E	4 байти	Застосований метод ущільнення
22	4 байти	Розмір зображення
26	4 байти	Горизонтальна роздільна здатність зображення (кількість пікселів на метр)
2A	4 байти	Вертикальна роздільна здатність зображення (кількість пікселів на метр)
2E	4 байти	Кількість кольорів в палітрі, зазвичай 0
32	4 байти	Кількість основних кольорів, зазвичай 0

Для збереження вмісту цього заголовку в классі Compressor створено структуру BMPInfoHeader з відповідними полями.

Розглянемо, як виглядає вміст простого bmp файлу, за допомогою програми Hex Editor Neo. Це зображення розміром 5 на 7 пікселів наведено на рис. 4.3 у збільшеному вигляді, а його представлення у шістнадцятковому коді – на рис. 4.4.

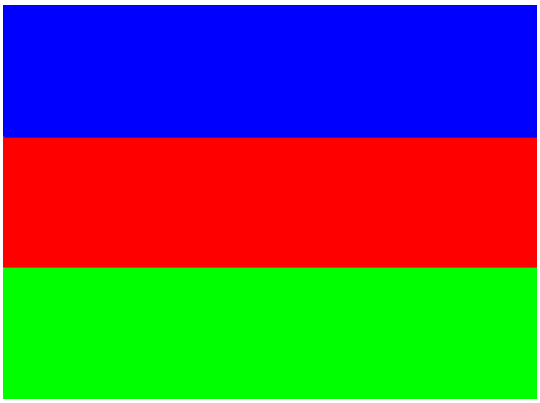

Рисунок 4.3 – Тестове зображення у збільшеному вигляді

0000005a	00 01	02 03	04 05	06 07	08 09	0a 0b	0c 0d	0e 0f	
00000000	42 4d	5a 00	00 00	00 00	00 00	36 00	00 00	28 00	BMZ.....6...(.
00000010	00 00	04 00	00 00	03 00	00 00	01 00	18 00	00 00	.....
00000020	00 00	24 00	00 00	00 00	00 00	00 00	00 00	00 00	..$.....
00000030	00 00	00 00	00 00	00 ff	00 00	ff 00	00 ff	00 00	.....я..я..я..
00000040	ff 00	00 00	ff 00	00 ff	00 00	ff 00	00 ff	ff 00	я...я..я..я..яя.
0000005a	00 ff	00 00	ff 00	00 ff	00 00	.. ..	.. ..	.. ..	.я..я..я.....

Рисунок 4.4 – Вміст файлу (шістнадцяткове подання)

Чорними прямокутниками виділено: ідентифікатор bmp файлу (424D), значення зміщення масиву пікселів (36), ширина (04), висота (03), глибина кольору (18). Прямокутниками різних кольорів виділені значення відповідних пікселів. Видно, що компоненти кольорів в кожному пікселі йдуть в послідовності BGR, а не RGB. До того ж, масив пікселів записано у файл знизу догори. Тобто, спочатку останній рядок, а наприкінці – перший. Треба зазначити, що поля заголовків, окрім ідентифікатора файлу, необхідно зчитувати, переставивши байти у зворотному порядку. Також, якщо кількість байтів в рядку зображення не буде кратна 4, то в форматі bmp кожен такий рядок доповнюється необхідною кількістю байт зі значенням 0.

Програма розпізнає файли лише з одним типом заголовку, з глибиною кольору 24, тому ще в конструкторі класу може відбутись завершення роботи, якщо якісь параметри не будуть співпадати.

Приватні поля класу Compressor:

- int _end_symbol – символ кінця закодованого потоку алгоритмом Хаффмана.
- int _esc_symbol – символ, після якого у закодованому потоці записано оригінальне значення закодованого байту.
- FILE* _compressed_file – вказівник для доступу до файлу. При роботі в режимі ущільнення ця змінна вказує на вихідний потік, при роботі в режимі відновлення – на вхідний.
- unsigned char _mask – маска для побітового вводу та виводу закодованих байтів в алгоритмі Хаффмана.
- int _bit – значення введеного або виведеного біта.
- FILE* _file_output – вказівник на файл – результат роботи програми.
- FILE* _file_input – вказівник на вхідний файл.
- string _filename – назва вхідного файлу.
- string _new_filename – назва вихідного файлу.
- BMPFileHeader _bmp_file_header – об’єкт структури заголовку bmp файлу.
- BMPInfoHeader _bmp_info_header – об’єкт структури інформаційного заголовку.
- size_t _max_difference – значення межі різниці між значеннями байтів в розробленому алгоритмі ущільнення. Ініціалізується значенням 8;
- vector<unsigned char> _blue – вектор для збереження байтів синьої компоненти пікселів з вхідного файлу.
- vector<unsigned char> _green – вектор для збереження байтів зеленої компоненти пікселів з вхідного файлу.

- vector<unsigned char> _red – вектор для збереження байтів червоної компоненти пікселів з вхідного файлу.

Приватні методи класу Compressor:

- void initBMPFileHeader() – метод зчитування з вхідного файлу байтів заголовку файлу та ініціалізація відповідної структури.
- void initBMPInfoHeader() – метод зчитування з вхідного файлу байтів інформаційного заголовку та ініціалізація відповідної структури.
- template <typename T>
  void fillField(T& field, const size_t size) – шаблонний метод для ініціалізації полів структур заголовків. Зчитує з файлу задану кількість байтів та записує їх в зворотному порядку в задане поле структури.
- void compress() – метод для ущільнення файлів. В ньому послідовно викликаються методи readImageForCompress(), encode() для кожного з векторів колірних компонент та huffmanEncode() – для кожного з вже закодованих векторів. Запис байтів в вихідний файл відбувається в методі huffmanEncode().
- void decompress() – метод відновлення ущільненого файлу. В ньому послідовно викликаються методи huffmanDecode(), в яких також відбувається зчитування даних з закодованого файлу, для кожного з векторів колірних компонент, decode() – для кожного з векторів та writeToDecompressedFile() – для формування відновленого файлу.
- void readImageForCompress() – метод побайтового зчитування масиву пікселів у відповідні вектори колірних компонент. Враховується «зміщення кінця рядка» – доповнення кожного рядка зображення байтами до кількості, кратній 4.

- void encode(const vector<unsigned char>& source, vector<unsigned char>& result) – метод, в якому реалізовано розроблений алгоритм ущільнення даних. Приймає в якості вхідного параметра вектор байтів і повертає вектор вже закодованих байтів.
- void decode(vector<unsigned char>& source, vector<unsigned char>& result) – метод, в якому реалізовано розроблений алгоритм декомпресії даних. Приймає в якості вхідного параметра вектор закодованих байтів і повертає вектор відновлених байтів.
- void writeToDecompressedFile(vector<unsigned char>& blue, vector<unsigned char>& green, vector<unsigned char>& red) – метод для запису відновлених байтів в файл, що є оригінальним зображенням. Враховується «зміщення кінця рядка» – доповнення кожного рядка зображення байтами до кількості, кратній 4.
- void initializeTree(HuffmanTree& tree) – метод початкової ініціалізації дерева Хаффмана. Створюється кореневий вузол та вузли для символів кінця потоку та символу наявності реального значення байту.
- void updateTree(HuffmanTree& tree, int byte) – метод оновлення дерева після зчитування нового символу. Викликає, при необхідності, метод rebuildTree(). Переставляє вузли в дереві для збереження впорядкованості за вагою.
- void rebuildTree(HuffmanTree& tree) – метод перебудови дерева, яких застосовується в тих випадках, коли вага вузлів досягла максимального значення.

- void huffmanEncode(vector<unsigned char>& source) – метод кодування даних алгоритмом Хаффмана. Послідовно зчитує байти з вхідного потоку, викликає методи encodeByte() та updateTree().
- void encodeByte(HuffmanTree& tree, unsigned int byte) – метод для кодування чергового символу. Визначає код Хаффмана для чергового символу та додає новий вузол до дерева.
- void outputBits(unsigned int code, int count) – метод виведення отриманого коду Хаффмана для чергового символу у вихідний файл.
- void huffmanDecode(vector<unsigned char>& dest) – метод декодування даних, закодованих алгоритмом Хаффмана. Викликає методи decodeByte() та updateTree(). Отримані значення байтів записує у вектор.
- int decodeByte(HuffmanTree& tree) – метод декодування окремого байта. Знаходить в дереві справжнє значення чергового байту.
- int inputBit() – метод зчитування чергового біта з вхідного файлу. Зчитує з вхідного потоку необхідний біт, використовуючи бітову маску.
- unsigned int inputBits(int bit_count) – метод зчитування заданої кількості бітів. Зчитує з вхідного потоку задану кількість бітів, використовуючи бітову маску.
- void addNewNode(HuffmanTree& tree, int byte) – метод для додавання нового вузла до дерева Хаффмана.
- void swapNodes(HuffmanTree& tree, int current_node_num, int new_node_num) – метод перестановки вузлів в дереві Хаффмана, який застосовується під час оновлення дерева черговим символом, та відновлення властивості впорядкованості.

4.3 Тестування розроблених засобів

Програма написана та скомпільована за допомогою інтегрованого середовища розробки Microsoft Visual Studio 2013.

Ноутбук, на якому проводиться тестування, має такі характеристики:

- Операційна система: Windows 7.
- Оперативна пам’ять: 2ГБ.
- Процесор: Intel Core 2 DUO CPU P8400

З точки зору користувача, розроблена програма повинна відповідати наступним основним вимогам:

- Ущільнювати або навпаки відновлювати вхідні дані в залежності від обраного режиму роботи.
- Повертати повідомлення-інструкцію, якщо параметри було введено неправильно. Це допоможе користувачу правильно запустити програму.
- Повертати повідомлення про помилку, програму було запущено правильно, але параметри є некоректними. Наприклад, якщо вхідний файл має формат, який програма не підтримує.

Програма запускається з командного рядка. Вона приймає 2 параметри. Перший вказує на режим роботи, другий – на ім’я вхідного файлу. При невірно вказаних параметрах на екран виводиться підказка (рис. 4.5).

```
Usage:
diploma compress [file_name]
diploma decompress [file_name]
```

Рисунок 4.5 – Способи запуску програми

Для вимірів часових характеристик роботи програми застосовано бібліотеку chrono. Виміри проводились таким чином: проводиться 20

незалежних прогонів програми з одними і тими ж самими даними. Всі часові характеристики збираються у відповідні масиви. Після цього, з масивів вилучаються по 3 максимальних та мінімальних значення та знаходиться середнє серед інших.

Для тестування обрано зображення різних розмірів із різним ступенем плавності переходів. Картинки різних розмірів необхідні для вимірювання залежності часу роботи від кількості даних. Картинки з різнем ступенем плавності – для вимірювання ступеня ущільнення. Найгірший варіант для розробленого алгоритму – по 2 підряд попарно різних пікселі – рис. 4.6.

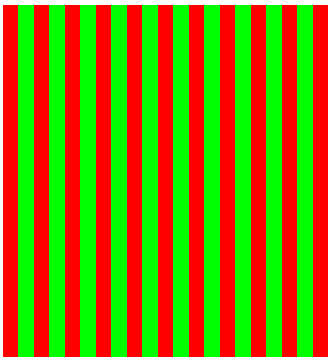

Рисунок 4.6 – Частина тестового зображення у збільшеному вигляді (реальний розмір – 960x297).

Найкращим варіантом є зображення білого кольору (розмір 1920x1080 – 1080p) – всі байти однакові.

Також додаємо до списку тестових зображень рис. 4.1 та рис. 4.7.

Рисунок 4.7 – Тестове зображення у зменшеному вигляді (розмір 1920x1080 – 1080)

Отже, маємо наступний набір тестових даних:

1) Зображення з попарно різними пікселями. Розмір 960x297 – 285210 пікселів.
2) Повністю біле зображення. Розмір 1920x1080 – 2073600 пікселів.
3) Зображення заходу сонця з плавними переходами. Розмір 512x320 – 163840 пікселів.
4) Абстрактне, менш плавне зображення. Розмір 1920x1080 – 2073600 пікселів.

Виміри часу для ущільнення кожного зображення представлені в табл. 4.4.

Таблиця 4.4 – Вимір часу виконання окремих етапів процесу компресії

№ зображення (розмір) / Метод	1 (285210)	2 (2073600)	3 (163840)	4 (2073600)
Constructor (початкова ініціалізація, зчитування заголовків)	0,001 с	0,001 с	0,001 с	0,001 с
readImageForCompress (зчитування масиву пікселів)	0,152 с	1,135 с	0,113 с	1,114 с
encode (blue)	0,082 с	0,613 с	0,049 с	0,628 с
encode (green)	0,084 с	0,588 с	0,049 с	0,624 с
encode (red)	0,083 с	0,595 с	0,050 с	0,659 с
huffmanEncode (blue)	0,203 с	1,452 с	0,152 с	2,091 с
huffmanEncode (green)	0,209 с	1,486 с	0,152 с	2,058 с
huffmanEncode (red)	0, 209 с	1,448 с	0,153 с	2,102 с
Загальний час	1,034 с	7,257 с	0,706 с	8,770 с

З цієї таблиці видно, що найбільше часу 60-75% витрачається на процес кодування алгоритмом Хаффмана. Це пояснюється тим, що в ньому проходить процес запису даних в файл. Процес кодування розробленим алгоритмом займає майже 2 секунди для зображень формату 1080p. По відношенню до часу роботи всієї програми, час роботи розробленого алгоритму становить 21-30%. Залишки часу витрачаються на зчитування даних з початкового файлу. Добре видно залежність між розміром файлу та часом зчитування.

В табл. 4.5 представлено інформацію про те, скільки байтів займає та чи інша послідовність після кодування розробленим алгоритмом. Початкова кількість байтів дорівнює кількості пікселів зображення, яка вказана в дужках біля його номера.

Таблиця 4.5 – Розміри бітових векторів після ущільнення розробленим алгоритмом

№ зображення (розмір) / Метод	1 (285210)	2 (2073600)	3 (163840)	4 (2073600)
encode (blue) в байтах	145878	1060921	100499	1356970
encode (green) в байтах	149142	1060921	102280	1282063
encode (red) в байтах	149142	1060921	113078	1357534

З цієї таблиці видно, що у середньому розмір ущільнених даних складає 52-60% від початкових даних, а коефіцієнт ущільнення майже 2.

У табл. 4.6 надані розміри в байтах вхідних файлів та розміри файлів після всіх етапів ущільнення.

Таблиця 4.6 – Порівняння розмірів файлів до і після ущільнення

№ зображення	Розмір до ущільнення (в байтах)	Розмір після ущільнення (в байтах)
1	855414	107509
2	6220854	416034
3	491574	268989
4	6220854	3347309

На прикладі другого зображення видно, що алгоритм Хаффмана значно зменшив розмір повністю білого зображення, відносно того, яким воно було після кодування розробленим алгоритмом. В інших випадках, алгоритм Хаффмана надав ще 15-20% ущільнення. В табл. 4.7 представлені часові характеристики при декомпресії.

Таблиця 4.7 – Вимір часу виконання окремих етапів процесу декомпресії

№ зображення (розмір) / Метод	1 (285210)	2 (2073600)	3 (163840)	4 (2073600)
Constructor (початкова ініціалізація, зчитування заголовків)	0,001 с	0,001 с	0,001 с	0,001 с
huffmanDecode (blue)	0,105 с	0,765 с	0,127 с	1,749 с
huffmanDecode (green)	0,122 с	0,768 с	0,134 с	1,615 с
huffmanDecode (red)	0,122 с	0,760 с	0,143 с	1,711 с
decode (blue)	0,064 с	0,470 с	0,038 с	0,556 с
decode (green)	0,065 с	0,470 с	0,037 с	0,533 с
decode (red)	0,065 с	0,470 с	0,037 с	0,528 с
writeToDecompressedFile (запис відновлених даних в файл)	0,161 с	1,175 с	0,096 с	1,354 с
Загальний час	0,730 с	5,603 с	0,631 с	8,1 с

Ці дані відповідають даним з таблиці часу при компресії. Найбільша частка припадає на алгоритм Хаффмана та запис розкодованих байтів до файлу.

Як бачимо, в розробленій програмі при виконанні процесу компресії/декомпресії найбільша частка часу витрачається на зчитування та запис у файли. Кожен з двох етапів кодування/декодування інформації відпрацьовується швидко, навіть у випадках, коли обсяг оброблюваних даних досить великий. В цілому, алгоритм і програма відповідають основним вимогам.

ВИСНОВКИ

В процесі роботи над дипломним проєктом проведено дослідження предметної області, ознайомлення з принципами зберігання графічної інформації в комп’ютері, методами та алгоритмами ущільнення графічних даних.

Розроблено програму, яка реалізує ущільнення графічних даних без втрати якості зображення. Ущільнення складається з двох етапів. Для першого етапу розроблено алгоритм, який базується на достатньо простому принципі – кодуванні схожих один на одного пікселів меншою кількістю бітів за рахунок виконання арифметичних операцій. Другий етап ущільнення – це компресія за допомогою адаптивного алгоритму Хаффмана.

Проведене тестування показало, що розроблені програмні засоби відповідають поставленим вимогам, працюють достатньо швидко, а поєднання розробленого алгоритму і алгоритму Хаффмана показує найкращі результати в тих випадках, де зображення мають багато однакових послідовностей пікселів. Чим їх більше, тим більше можна зменшити обсяг даних.

СПИСОК ВИКОРИСТАНИХ ЛІТЕРАТУРНИХ ДЖЕРЕЛ

1. Компьютерная графика. [Електронний ресурс] – Режим доступу: https://ru.wikipedia.org/wiki/Компьютерная_графика
2. Растрова графіка. [Електронний ресурс] – Режим доступу: https://uk.wikipedia.org/wiki/Растрова_графіка
3. Raster graphics. [Електронний ресурс] – Режим доступу: https://en.wikipedia.org/wiki/Raster_graphics.
4. Vector graphics overview. [Електронний ресурс] – Режим доступу: https://docs.microsoft.com/en-us/dotnet/framework/winforms/advanced/vector-graphics-overview?redirectedfrom=MSDN
5. Веторная графика. [Електронний ресурс] – Режим доступу: https://ru.wikipedia.org/wiki/Векторная_графика
6. Колірна модель. [Електронний ресурс] – Режим доступу: http://www.zhu.edu.ua/mk_school/mod/page/view.php?id=5029
7. Алгоритмы сжатия изображений. [Електронний ресурс] – Режим доступу: http://lib.ru/TECHBOOKS/ALGO/VATOLIN/algcomp.htm
8. Алгоритм Лемпеля — Зива — Велча. [Електронний ресурс] – Режим доступу: https://ru.wikipedia.org/wiki/Алгоритм_Лемпеля_—_Зива_—_Велча
9. DEFLATE Compressed Data Format. [Електронний ресурс] – Режим доступу: https://tools.ietf.org/html/rfc1951
10. Код Хаффмана. [Електронний ресурс] – Режим доступу: https://ru.wikipedia.org/wiki/Код_Хаффмана
11. Алгоритмы LZ77 и LZ78. [Електронний ресурс] – Режим доступу: https://neerc.ifmo.ru/wiki/index.php?title=Алгоритмы_LZ77_и_LZ78
12. Алгоритм JPEG. [Електронний ресурс] – Режим доступу: https://ru.wikipedia.org/wiki/JPEG

13. JPEG. Алгоритм сжатия. [Електронний ресурс] – Режим доступу: https://habr.com/ru/post/482728/
14. Алгоритм JPEG 2000. [Електронний ресурс] – Режим доступу: https://ru.wikipedia.org/wiki/JPEG_2000
15. C++. [Електронний ресурс] – Режим доступу: https://ru.wikipedia.org/wiki/C%2B%2B
16. Адапривное сжатие Хаффмана. [Електронний ресурс] – Режим доступу: http://mf.grsu.by/UchProc/livak/po/comprsite/pract_huffman_adapt.html
17. Метод Хаффмана. [Електронний ресурс] – Режим доступу: http://mf.grsu.by/UchProc/livak/po/comprsite/theory_huffman_adapt.html#huffman_adapt
18. BMP file format. [Електронний ресурс] – Режим доступу: https://en.wikipedia.org/wiki/BMP_file_format