РЕФЕРАТ

Пояснювальна записка: ___ с., ___ рис., ___ табл., __ дод., ___ джерел.

Об'єкт розробки: алгоритми фрактального стиску даних в мультимедійних файлах.

Мета кваліфікаційної роботи: аналіз можливості фрактального стиснення аудіо даних, розробка відповідного алгоритму та його програмної реалізації.

У вступі розглядається аналіз та сучасний стан проблеми, конкретизується мета кваліфікаційної роботи та галузь її застосування, наведено обґрунтування актуальності теми та уточнюється постановка завдання.

У першому розділі проведено аналіз предметної галузі, визначено актуальність завдання та призначення розробки, розроблена постановка завдання, задані вимоги до програмної реалізації, технологій та програмних засобів.

У другому розділі виконано аналіз існуючих рішень, обрано платформу для розробки, виконано проєктування і розробка програми, наведено опис алгоритму і структури функціонування програми, визначені вхідні і вихідні дані, наведені характеристики складу параметрів технічних засобів, описаний виклик та завантаження програми, описана робота програми.

В економічному розділі визначено трудомісткість розробленої інформаційної системи, проведений підрахунок вартості роботи по створенню програми та розраховано час на його створення.

Практичне значення полягає в тому, що в даний час найбільш перспективною і розвинутою групою алгоритмів стиснення мультимедійної інформації є фрактальні алгоритми. У них використовується принципово нова ідея – не близькість даних в локальній області, а подібність різних за розміром областей зображення. Так за допомогою стандартних прийомів обробки зображень, таких, як виділення країв і аналіз текстурних варіацій, зображення ділиться на сегменти і кодується за допомогою деякого стискаючого афінного перетворення. В результаті виконання даної роботи розроблена програма реалізує фрактальний алгоритм стиску аудіо даних.

Актуальність даного програмного продукту визначається тим, що існуючі методи стиснення звукових даних або забезпечують недостатні коефіцієнти стиснення, або ведуть до істотної втрати даних, що в свою чергу пов'язано з погіршенням споживчої якості інформації.

Список ключових слів: ФРАКТАЛ, АУДІО ДАННІ, АЛГОРИТМ, КОМПРЕССІЯ, ДЕКОМПРЕССІЯ, ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ.

ABSTRACT

Explanatory note: ___ pp., ___ fig., ___ table, __ appendix, ___ sources.

Object of development: algorithms of fractal data compression in multimedia files.

The purpose of the qualification work: analysis of the possibility of fractal compression of audio data, development of an appropriate algorithm and its software implementation.

The introduction considers the analysis and current state of the problem, specifies the purpose of the qualification work and its scope, provides a justification for the relevance of the topic and clarifies the problem.

In the first section the analysis of the subject area is carried out, the urgency of the task and purpose of development are defined, the statement of the task is developed, requirements to software realization, technologies and software are set.

The second section analyzes existing solutions, selects a platform for development, designs and develops the program, describes the algorithm and structure of the program, determines the input and output data, provides characteristics of the parameters of hardware, describes the call and download of the program, describes the program.

The economic section determines the complexity of the developed information system, calculates the cost of work to create a program and calculates the time for its creation.

Of practical importance is that currently the most promising and advanced group of algorithms for compressing multimedia information are fractal algorithms. They use a fundamentally new idea - not the proximity of data in the local area, and the similarity of different sized areas of the image. Thus, using standard image processing techniques, such as edge selection and analysis of texture variations, the image is divided into segments and encoded by some compressive affine transformation. As a result of this work, the developed program implements a fractal algorithm for compressing audio data.

The relevance of this software product is determined by the fact that the existing methods of audio data compression either provide insufficient compression ratios, or lead to significant data loss, which in turn is associated with a deterioration in the consumer quality of information.

List of key words: FRACTAL, AUDIO DATA, ALGORITHM, COMPRESSION, DECOMPRESSION, SOFTWARE.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

АЧХ – Амплітудно-частотна характеристика;

ЖЦПЗ – життєвий цикл програмного забезпечення;

ПЗ – програмне забезпечення;

СКВ – середньоквадратичне відхилення;

ADPCM– Adaptive Differential PCM, адаптивні диференціальні PCM;

ATRAC – Adaptive TRansform Acoustic Coding, адаптивне акустичне перетворення;

IFS – Iterated Function System, система ітерованих функцій;

MPEG – Motion Picture Expert Group, експертна група з кінофільмів;

PCM – Pulse Code Modulation, імпульсно-кодова модуляція.

ЗМІСТ

РЕФЕРАТ……...……. 3
ABSTRACT……. 4
СПИСОК УМОВНИХ ПОЗНАЧЕНЬ……... 5
ВСТУП…… 8
РОЗДІЛ 1. . АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАВДАННЯ……... 10
1.1. Загальні відомості з предметної галузі …….. 10
1.1.1. Загальні відомості та стандарти зберігання мультимедійних даних..... 40
1.1.2. Огляд існуючих методів стиснення звукових даних..... 12
1.1.3. Основні поняття теорії фракталів..... 19
1.2. Призначення розробки та галузь застосування…….. 29
1.3. Підстава для розробки……..…….. 29
1.4. Постановка завдання……..… 30
1.5. Вимоги до програми або програмного виробу……..……. 30
1.5.1. Вимоги до функціональних характеристик…….…… 30
1.5.2. Вимоги до інформаційної безпеки…… 31
1.5.3. Вимоги до складу та параметрів технічних засобів…… 31
1.5.4. Вимоги до інформаційної та програмної сумісності ……..…… 31
РОЗДІЛ 2. ПРОЄКТУВАННЯ ТА РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ ….……..…… 32
2.1. Функціональне призначення програми ……. 32
2.2. Опис застосованих математичних методів……..... 32
2.2.1. Метод фрактального стиснення зображень..... 32
2.2.2. Математичні основи теорії фрактального стиснення..... 34
2.3 Опис використаної архітектури та шаблонів проектування..... 37
2.4. Опис використаних технологій та мов програмування…..….... 40

2.5. Опис структури програми та алгоритмів її функціонування … 44
2.5.1. Алгоритм стиснення..... 44
2.5.2. Алгоритм декомпресії..... 48
2.5.3. Метод фрактального стиснення аудіо..... 49
2.5.3.1.Ідея фрактального стиснення аудіо..... 49
2.5.3.2.Визначення коефіцієнтів стискаючих перетворень..... 50
2.5.4. Алгоритм стиснення..... 54
2.5.5. Алгоритм декомпресії..... 55
2.5.5. Програмна реалізація алгоритмів..... 56
2.6. Обґрунтування та організація вхідних та вихідних даних програми…….. 61
2.7. Опис розробленого програмного продукту……. 61
2.7.1. Використані технічні засоби……..… 61
2.7.2. Використані програмні засоби…….. 62
2.7.3. Виклик та завантаження програми……... 62
2.7.4. Опис інтерфейсу користувача……..…… 62
2.7.5. Дослідження роботи розроблених алгоритмів та їх програмної реалізації..... 64
РОЗДІЛ 3. ЕКОНОМІЧНИЙ РОЗДІЛ……... 73
3.1. Розрахунок трудомісткості та вартості розробки програмного продукту…….. 73
3.2. Розрахунок витрат на створення програми……... 76
ВИСНОВКИ…… 78
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ…….. 80
Додаток А. Код програми……. 81
Додаток Б. Відгук керівника економічного розділу……... 108
Додаток В. Перелік файлів на диску…… 109

ВСТУП

Розвиток сучасних телекомунікаційних мереж характеризується збільшенням частки мультимедійного трафіку. Важливою складовою мультимедійного трафіку є аудіо інформація, і зокрема мовна інформація. Проблема, пов'язана з великим обсягом для їх передачі та зберігання, з'явилася при роботі і на робочих станціях, і на персональних комп'ютерах.

На теперішній час вже розроблено велику кількість різних алгоритмів архівації звукової інформації. Оцифрований звук з високою якістю вимагає величезних витрат дискової пам'яті. Спроби скоротити обсяг файлів, використовуючи стандартні архіватори, не призводять до значного виграшу через специфічність звукових даних. Проте, домогтися досить значного рівня стиснення аудіо інформації вдається при використанні спеціальних методів, заснованих на аналізі структури даних і наступним стисканням з деякими втратами.

Стандартні алгоритми оборотного стиску не дозволяють заощадити більше ніж 10-20% від загального обсягу файлу. Алгоритми, що враховують структуру звукозапису, дозволяють трохи підвищити ефективність стиснення, але також не дуже значно. Не буде перебільшенням сказати, що жоден з оборотних алгоритмів не дозволяє домогтися хоча б двократного зменшення необхідного простору. Алгоритми стиснення з втратами забезпечують стиснення приблизно в 4 рази, що супроводжується незначним погіршенням якості звучання. Інші алгоритми враховують, що джерелом або адресатом звуку є людина. Так формат MP3 орієнтується на не вельми голосову, а слухову систему людини і використовує так званий психоакустичний підхід. Ідентичність вихідної та кінцевої звукової хвилі в цьому форматі взагалі не потрібна. Зате він забезпечує їх однакове сприйняття людиною. При використанні формату MP3 досягається ступінь стиснення в 10-12 разів.

Таким чином, існуючі методи стиснення звукових даних або забезпечують недостатні коефіцієнти стиснення, або ведуть до істотної втрати даних, що в свою чергу пов'язано з погіршенням споживчої якості інформації.

В даний час найбільш перспективною і розвинутою групою алгоритмів стиснення мультимедійної інформації є фрактальні алгоритми. У них використовується принципово нова ідея - чи не близькість кольорів в локальній області, а подібність різних за розміром областей зображення. Так за допомогою стандартних прийомів обробки зображень, таких, як виділення країв і аналіз текстурних варіацій, зображення ділиться на сегменти і кодується за допомогою деякого стискаючого афінного перетворення. Відновлення зображення відбувається за допомогою багаторазового застосування цього афінного перетворення. При цьому коефіцієнт стиснення у фрактальних алгоритмів варіюється в межах 2-2000.

Метою даної кваліфікаційної роботи є аналіз можливості фрактального стиснення аудіо даних, розробка відповідного алгоритму і його програмної реалізації.

Під час виконання роботи викладається опис існуючих методів стиснення звукових даних, основні поняття, описується метод фрактального стиснення зображень і математичне обґрунтування його використання для стиснення аудіо даних, виконано програмну реалізацію даного алгоритму.

В результаті виконання даної роботи розроблена програма, що реалізує фрактальний алгоритм стиску аудіо даних.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1. Загальні відомості з предметної галузі

1.1.1. Загальні відомості та стандарти зберігання мультимедійних даних

В даний час найбільш поширені три стандарти зберігання відеоданих: MPEG-1, MPEG-2 і MPEG-4. В рамках перших двох форматів існують також формати зберігання звукової інформації - Layer-1, Layer-2 і Layer-3. Ці три звукових формату визначені для MPEG-1 і незначними розширеннями використовуються в MPEG-2. Всі три формати схожі один на одного, але використовують різні рівні компромісу між стисненням і складністю. Рівень Layer-1 - найбільш простий, не вимагає значних витрат на стиск, але і дає незначний ступінь стиснення. Рівень Layer-3 - найбільш трудомісткий і забезпечує найкраще стиснення. Останнім часом цей формат завоював величезну популярність. Його часто називають MP3. Така назва пов'язана з розширенням звукових файлів, що зберігаються в цьому форматі.

Основна ідея, на якій базуються всі методики стиснення аудіо сигналу з втратами, - зневага тонкими деталями звучання оригіналу, що лежать поза межами які сприймає людське вухо. Тут можна виділити кілька моментів.

Рівень шуму. Звуковий стиск базується на простому факті - якщо людина перебуває поруч з голосно виючою сиреною, то навряд чи він почує розмову неподалік розміщених людей. Причому це відбувається не тому, що людина звертає велику увагу на гучний звук, а в більшій мірі від того, що людське вухо фактично втрачає звуки, що лежать в тому ж діапазоні частот, що і більш гучний звук. Цей ефект називається маскуючим, він змінюється з різницею в гучності і частоті звуку.

Другим моментом є поділ смуги звукових частот на підполоси, кожна з яких далі обробляється окремо. Програма кодування виділяє найгучніші звуки в

кожній смузі і використовує цю інформацію для визначення прийнятного рівня шуму для цієї смуги. Кращі програми кодування враховують також вплив сусідніх смуг. Дуже гучний звук в одній смузі може вплинути на маскуючий ефект і на прилеглі смуги.

Ще одним моментом кодування є використання психоакустичної моделі, що спирається на особливості людського сприйняття звуку. Стиснення з використанням цієї моделі засновано на видаленні свідомо нечутних частот з більш ретельним збереженням звуків, що добре розрізняються людським вухом. На жаль, тут не може бути точних математичних формул. Сприйняття звуку людиною - складний, до кінця не вивчений процес, тому вибір методів стиснення виконується на основі аналізує прослуховування і порівняння по-різному стислих звуків групами експертів. Зате тут є практично необмежені можливості в сфері поліпшення психоакустичних моделей. Більшість існуючих алгоритмів для кодування людського голосу засноване на високій передбачуваності такого сигналу - універсальні алгоритми стиснення MPEG зі змінним успіхом намагаються застосувати цей прийом.

Ще одним прийомом стиснення є використання так званого суміщеного стерео. Відомо, що слуховий апарат людини може визначити напрямок лише середніх частот - високі і низькі звучать як би окремо від джерела. Значить, ці фонові частоти можна кодувати в моно сигнал. Крім усього цього для стиснення використовується відмінність в складності потоків в каналах. Наприклад, якщо в правому каналі якийсь час повна тиша, це "зарезервоване" місце використовується для підвищення якості лівого каналу або туди "впихнути" необхідні біти, які не влізли в потік трохи раніше. На останній стадії стиснення використовується алгоритм стиснення Хаффмана. Цей процес дозволяє поліпшити ступінь стиснення для відносно однорідних сигналів, які погано стискаються за допомогою описаних вище прийомів. На основі описаних ідей будуються алгоритми стиснення, що дозволяють досягати ступеня компресії 10: 1 або вище, практично без втрати в якості звучання. При кодуванні задають необхідний рівень компресії, а алгоритми стиснення

домагаються необхідного значення рівня стиснення за рахунок втрати якості. Необхідний рівень стиснення зазвичай вказують у вигляді величини потоку даних (bit rate), що вимірюється в Кбіт/сек.

1.1.2. Огляд існуючих методів стиснення звукових даних

На сьогоднішній день відомі наступні методи стиснення даних:

1. Кодування форми сигналу - ІКМ, дельта модуляція, ДІКМ, АДІКМ, дискретне косинусне перетворення, субполосне кодування і т.д.

2. Кодування параметрів мовного сигналу - вокодування.

3. Гібридне або параметричне кодування - піввокодери.

Залежно від реалізованого алгоритму кодування можна виділити наступні кодеки, які отримали найбільше поширення:

– PCM (Pulse Code Modulation) - ІКМ (імпульсно-кодова модуляція). Найбільш широко поширений алгоритм, який реалізує лінійне кодування рівня сигналу (величина коду прямо пропорційна рівню) без стиснення. Розрізняють знаковий 8 bit PCM (коди рівнів -128 - 127), беззнаковий 8 bit PCM (коди рівнів 0 - 255), знакові 16 bit PCM формату LSB (для платформ DOS, OS/2, Windows на основі процесорів Intel, порядок проходження байтів в багатобайтові даних молодший - старший), 16 bit PCM формату MSB (для платформ Macintosh, Amiga на основі процесорів Motorola порядок проходження байтів старший - молодший);

– CCITT a-Law/u-Law (μ -Law) - нелінійні PCM. Принцип нелінійного кодування заснований на тому факті, що суб'єктивне сприйняття гучності звуку людиною має логарифмічну залежність від його рівня. Використання логарифмічної компресії дозволяє стискати дані за рахунок погіршення якості кодування відліків сигналу високих рівнів. При цьому досягається коефіцієнт стиснення рівний двом. Види кодування A-Law/u-Law (μ-Law) відрізняються функціями відображення лінійної шкали рівнів вихідного сигналу на шкали рівнів стисненого сигналу (наприклад, вихідних 16 bit кодів на стислі 8 bit) і

поширені з одного боку в Європі A-Law і з іншого - в США і (u-Law (μ -Law)). Функція виду u-Law (μ-Law)), яка виробляє логарифмічне стиснення і відображає вихідний рівень сигналу величиною $-1 \le x \le 1$ на стиснений величиною $-1 \le y \le 1$, має вигляд:

$$y(x) = \text{sign}(x) \cdot \frac{\ln(1 + \mu|x|)}{\ln(1 + \mu)} \quad . \tag{1.1}$$

Функція, яка виробляє зворотне перетворення експоненціального експандування, записується у вигляді:

$$x(y) = \frac{\text{sign}(x)}{\mu}[(1 + \mu)^{|y|} - 1] \quad , \tag{1.2}$$

де $0 < \mu \le 255$ □□□ параметр функцій відображення, що визначає ступінь їх нелінійності.

При обробці рівнів сигналів, амплітуда яких відмінна від одиниці, для шкал несиметричних щодо нуля (наприклад при використанні беззнакових кодів), для отримання цілочисельних результатів, наведені вище функції відображення повинні бути модифіковані шляхом введення мультиплікативний і адитивних коефіцієнтів, які виробляють масштабування і зрушення функцій, а також шляхом використання функції виділення цілої частини. Так, наприклад, функція відображення початкового знакових 16 bit кодів на стислі беззнакові 8 bit представляється у вигляді:

$$y(x) = \text{trunc}\{2^7[\text{sign}(x) \cdot \frac{\ln(1 + \frac{\mu|x|}{2^{15}})}{\ln(1 + \mu)} + 1]\} \tag{1.3}$$

– ADPCM (Adaptive Differential PCM) - адаптивні диференціальні PCM. Дані алгоритми кодування ґрунтуються на тому факті, що послідовно йдуть відліки сигналу, що, як правило, за рівнями незначно різняться. Для кодування подальшого відліку використовується значення не власне його рівня, а його приріст у порівнянні з попереднім. Такий підхід дозволяє переходити від 8 bit кодів рівня до 1, 2, 4 або 6 bit кодами збільшень. В адаптивних (самоналагоджувальних) алгоритмах набір допустимих збільшень задається не апріорно, а визначається на основі кодованих даних. До числа найбільш поширених кодеків даного сімейства відносяться: Microsoft ADPCM, IMA ADPCM, DSP Group True Speech, GSM 6.10;

– ATRAC (Adaptive TRansform Acoustic Coding) розділяє 16-бітний 44,1 кГц цифровий аудіо сигнал на 52 частотні діапазони (після швидкого перетворення Фур'є). Діапазони з низькими частотами передаються більш точно, ніж з високими. Алгоритм використовує психоакустичне кодування, де застосовується ефект маскування і поріг чутності звуку, в результаті чого частина інформації може бути відкинута і виходить потік даних має розмір в 1/5 оригінального. Кожен канал обробляється незалежно (портативний MD привід Sony MZ-1 використовує один чіп ATRAC кодера/декодера на канал);

– MPEG (Motion Picture Expert Group) - сімейство алгоритмів, які увійшли в стандарт MPEG-1 стиснення аудіовідепотоків. Розділ, що відноситься до аудіо, визначає три рівні компресії (Layer I - III). Загальна структура процесу кодування однакова для всіх рівнів. Для кожного рівня визначено свій формат запису біт-потоку і свій алгоритм декодування. Алгоритми MPEG засновані в цілому на вивчені властивості сприйняття звукових сигналів слуховим апаратом людини (тобто кодування виробляється з використанням так званої "психоакустичної моделі"). Алгоритм кодування: Вхідний цифровий сигнал спочатку розкладається на частотні складові спектра. Потім цей спектр очищається від свідомо нечутних складових - низькочастотних шумів і найвищих гармонік, тобто фактично фільтрується. На наступному етапі проводиться значно складніший психоакустичний аналіз

чутного спектру частот. Це робиться, в тому числі з метою виявлення і видалення "замаскованих" частот (частот, які не сприймаються слуховим апаратом на увазі їх приглушення іншими частотами). Після всіх цих маніпуляцій з цифрового аудіо сигналу виключається більше половини інформації. Потім, в залежності від рівня складності використовуваного алгоритму, може бути також проведений аналіз передбачуваності сигналу. Крім цього, базуючись на тому, що людське вухо здатне розрізняти напрямок звучання тільки середніх частот, то в разі, коли кодується стерео сигнал, його можна перетворити в поєднаний стерео (joint stereo). Це означає, що фактично відбувається відділення верхніх і нижніх частот і їх кодування в моно варіанті (середні частоти залишаються в режимі стерео). Далі, в разі появи, наприклад, "тиші" в одному з каналів, "порожнє" місце заповнюється інформацією або підвищує якість іншого каналу, або просто не помістилася до цього. На довершення до всього проводиться стиснення вже готового біт-потоку спрощеним аналогом алгоритму Гоффмана (Huffman), що дозволяє також значно зменшити займаний потоком обсяг.

Комплект MPEG-1 передбачено для кодування сигналів, оцифрованих з частотою дискретизації 32, 44.1 і 48 КГц. Як було зазначено вище, комплект MPEG-1 має три рівні (Layer I, II і III). Ці рівні мають відмінності в яке забезпечується коефіцієнті стиснення і якості звучання одержуваних потоків. Layer I дозволяє сигнали 44.1 КГц/16 біт зберігати без відчутних втрат якості при швидкості потоку 384 Кбіт/с, що становить 4-х кратний виграш в займаному обсязі; Layer II забезпечує таку ж якість при 194 Кбіт/с, а Layer III - при 128 (або 112). Виграш Layer III очевидний, але швидкість компресії при його використанні найнижча (треба відзначити, що при сучасних швидкостях процесорів це обмеження вже не помітно). Фактично, Layer III дозволяє стискати інформацію в 10-12 разів без відчутних втрат в якості.

Окремо також слід сказати про MPEG-2 AAC, який є технологічним приймачем MPEG-1. Існує кілька різновидів цього алгоритму: Homeboy AAC, AT & T a2b AAC, Liquifier AAC, Astrid/Quartex AAC і Mayah AAC. Найбільш

висока якість звучання в порівнянні с MPEG-1 Layer III забезпечують дві передостанні реалізації. Всі наведені різновиди алгоритму AAC не є сумісними між собою.

Так само, як і в комплекті аудіо стандартів кодування MPEG-1, в основі алгоритму AAC лежить психоакустичний аналіз сигналу. Разом з тим, алгоритм AAC має в своєму механізмі безліч доповнень, спрямованих на поліпшення якості вихідного аудіо сигналу. Зокрема, використовується інший тип перетворень, поліпшена обробка шумів, змінений банк фільтрів, а також поліпшений спосіб запису вихідного біт-потоку. Крім того, AAC дозволяє зберігати в закодованому аудіо сигналі т.зв. "водяні знаки" (Watermarks) - інформацію про авторські права. Ця інформація вбудовується в біт-потік при кодуванні таким чином, що знищити її стає неможливо, не зруйнувавши цілісність аудіо даних. Ця технологія (в рамках Multimedia Protection Protocol) дозволяє контролювати поширення аудіо даних (що, до речі, є перешкодою на шляху поширення самого алгоритму і файлів, створених за допомогою нього). Слід зазначити, що алгоритм AAC не є назад сумісним MPEG-2 AAC не сумісний з рівнями MPEG-1 не дивлячись на те, що він являє собою продовження (доопрацювання) MPEG-1 Layer I, II, III.

MPEG-2 AAC передбачає три різних профілю кодування: Main, LC (Low Complexity) і SSR (Scaleable Sampling Rate). Залежно від того, який профіль використовується під час кодування, змінюється час кодування і якість одержуваного цифрового потоку. Найвищу якість звучання (при самій повільній швидкості компресії) забезпечує основний Main профіль. Це пов'язано з тим, що профіль Main включає в себе всі механізми аналізу та обробки вхідного потоку. Профіль LC спрощений, що позначається на якості звучання одержуваного потоку, сильно відбивається на швидкості компресії і, що більш важливо, декомпресії. Профіль SSR також є спрощеним варіантом профілю Main.

Говорячи про якість звуку, можна сказати, що потік AAC (Main) 96 Кбіт/с забезпечує якість звучання, аналогічне потоку MPEG-1 Layer III 128 Кбіт/с. При

компресії AAC 128 Кбіт/с, якість звучання відчутно перевершує MPEG-1 Layer III 128 Кбіт / с;

– Windows Media Audio (WMA) від Microsoft. Алгоритм WMA дозволяє потокове відтворення (stream playback). Якість WMA (якщо говорити про WMA 7.0 і 8.0) при швидкості потоку 64 Кбіт/с практично не поступається якості MPEG-1 Layer III 96-128 Кбіт/с. Для зберігання потоку в форматі WMA використовується універсальний потоковий файловий формат .ASF (Advanced Audio Streaming), що прийшов на заміну .WAV. Файли .WMA призначені виключно для зберігання аудіо даних. На сьогоднішній день існує кілька версій WMA: v1, v2, v7 і v8, v9. Версія 7 відрізняється від молодших побратимів підтримкою бітрейтів до 192 Kbps (на відміну від 164 Kbps в v1 і v2), кілька гіршою якістю кодування і іншою структурою даних. Версія 8 відрізняється від усіх попередніх явно переробленої психоакустичною моделлю кодека, за рахунок чого якість кодування сильно зросла. Так, при 64 Kbps WMA v8.0 на не сильно вимогливих до якості композиціях (поп-музика, наприклад) майже не відрізняється від MP3 при 128 Kbps;

– Wavelet алгоритм, заснований на використанні вейвлет-перетворення. Вейвлет-обробка сигналів забезпечує можливість досить ефективного стиснення сигналів та їх відновлення з малими втратами інформації, а також рішення задач фільтрації сигналів. Даний алгоритм стиснення заснований на розкладанні вихідного сигналу на елементарні компоненти (вейвлети), що мають локалізацію як в частотній так і в тимчасовій області. Для його реалізації часто використовують алгоритм Малла (швидке вейвлет-перетворення).

На рис. 1.1 представлена діаграма реалізації швидкого багатокрокового алгоритму Малла на основі вейвлет-фільтрації. Тут сигнал представлений 1000 підрахунками і схематично показані АЧХ узгоджених фільтрів нижніх частот (L) і верхніх частот (H). З цієї діаграми легко простежити за процесом декомпозиції і реставрації сигналу.

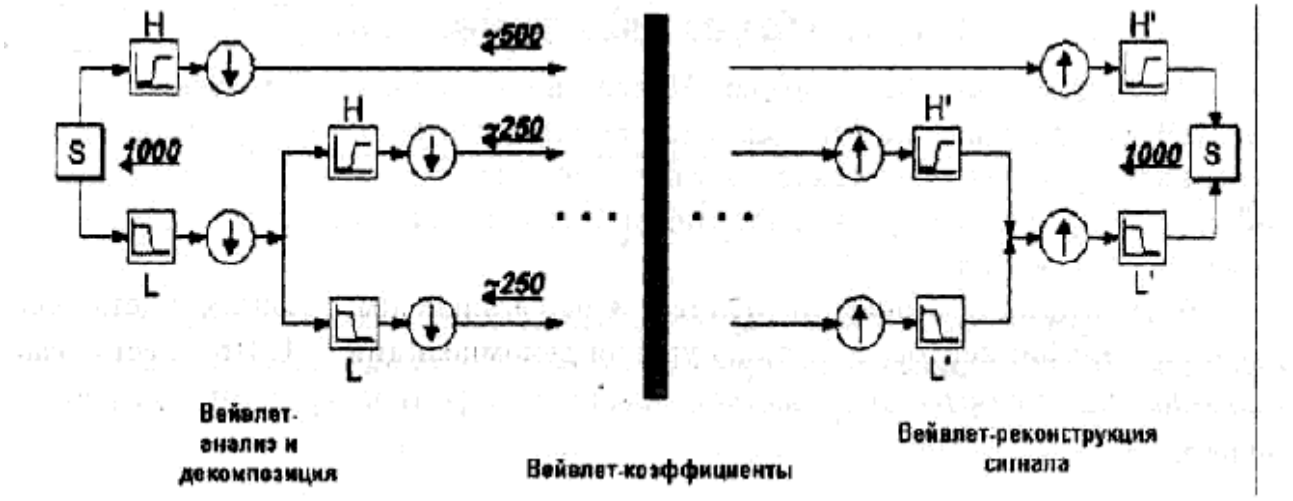

Рис. 1.1. Структура багатокрокового алгоритму Малла при декомпозиції і реставрації сигналу

В результаті цього процесу вихідний сигнал S розкладається на вейвлет-компоненти аж до заданого рівня декомпозиції, після чого, в ході реконструкції, відновлюється до наближеного сигналу S '. Ступінь наближення залежить від рівня декомпозиції та реконструкції. Нульовий рівень відповідає точному відновленню сигналу S. Рис. 1.2 показує звичайну діаграму розкладання (зверху-вниз) і реконструкцію (знизу-вгору) сигналу S.

На цій діаграмі коефіцієнти апроксимації сигналу позначені як C, а деталізуючи коефіцієнти як D. Цифри вказують на рівень декомпозиції та реконструкції сигналу (нульовий рівень окремо не зазначається, це є сам сигнал S).

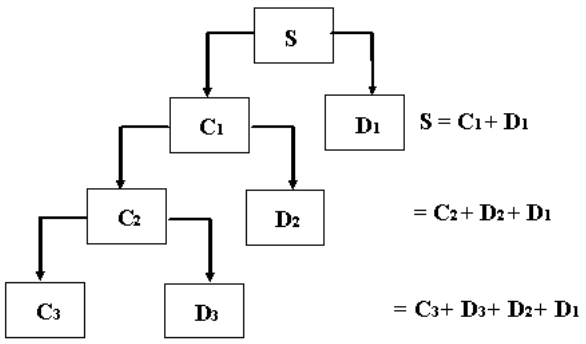

Рис. 1.2. Структура вейвлет-перетворення сигналу

В результаті використання вейвлет-перетворення коефіцієнт стиснення досягає 80% при задовільній якості відновленого сигналу [6].

Поєднання звичайних методів компресії даних і психоакустичного підходу дозволяє домогтися ступеня стиснення звукових даних до 10-12 разів при прийнятній якості звучання. Тобто, існуючі методи стиснення аудіо на даний момент не забезпечують високих коефіцієнтів стиску.

1.1.3. Основні поняття теорії фракталів

Фрактал походить від латинського прикметника "fractus" і в перекладі означає що складається з фрагментів, а відповідне латинське дієслово "frangere" означає розбивати, тобто створювати неправильні фрагменти. Це назва була введена в 70-х роках XX ст. ученим Бенуа Мандельброт, що зробив величезний вплив на виникнення і розвиток цієї галузі і пробудили загальний інтерес до фрактальної фізики. Згідно Мандельброту, «фракталом називається структура, що складається з частин, які в якомусь сенсі подібні цілому». Метричні характеристики, такі як довжина і площа, не мають для фракталів сенсу, тому що вони нескінченні.

Фрактали ― об'єкти, що володіють нескінченною складністю, що дозволяють розглянути стільки ж своїх предметів поблизу, як і здалеку. Будь-які їхні фрагменти, як нескінченно малі, так і нескінченно великі, по будові нічим не відрізняються один від одного. Фрактали можливі не тільки на площині, а й у просторі. Фрактальні структури зустрічаються і в природі.

Прикладами фракталів можуть служити крона дерева, силует гірського пасма, прикордонні і берегові лінії, пори в хлібі, дірки в деяких сортах сиру, частинки в порошках. Поверхня Місяця, виявляється, поблизу виглядає так само, як і здалеку, тільки розміри кратерів інші. Земля - класичний приклад фрактального об'єкта. З космосу вона виглядає як куля. Якщо наближатися до неї, ми виявимо океани, континенти, узбережжя і ланцюги гір. Будемо розглядати гори ближче ― стануть видні ще більш дрібні деталі: шматочок

землі на поверхні гори в своєму масштабі настільки ж складний і нерівний, як сама гора. І навіть ще більш сильне збільшення покаже крихітні частинки ґрунту, кожна з яких сама є фрактальним об'єктом. Фрактальну структуру має також Всесвіт.

Нижче наведено кілька прикладів відомих фракталів, що демонструють зачаровує красу цієї області людських інтересів, яка дивним чином поєднує природу, математику, комп'ютер і мистецтво (рис. 1.3-1.4):

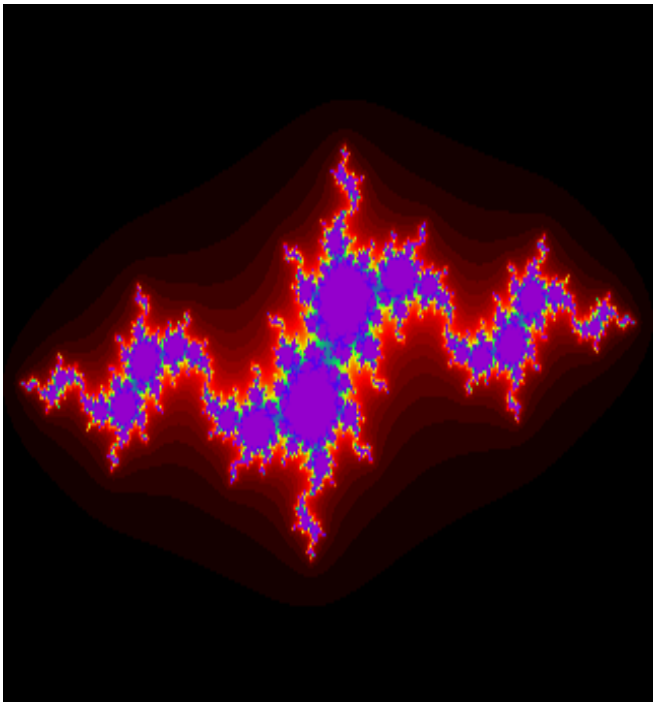

Рис. 1.3. Безліч Жюліа

Безліч Жюліа: колір кожної точки залежить від того, скільки ітерацій комплексної функції може бути зроблено, поки точка z не вийде за межі кола адіуса r (| z |> r):

$$f(z) = a(z2 + b, \quad (1.4)$$

де z = x + iy - комплексне число, відповідне точці (x, y).

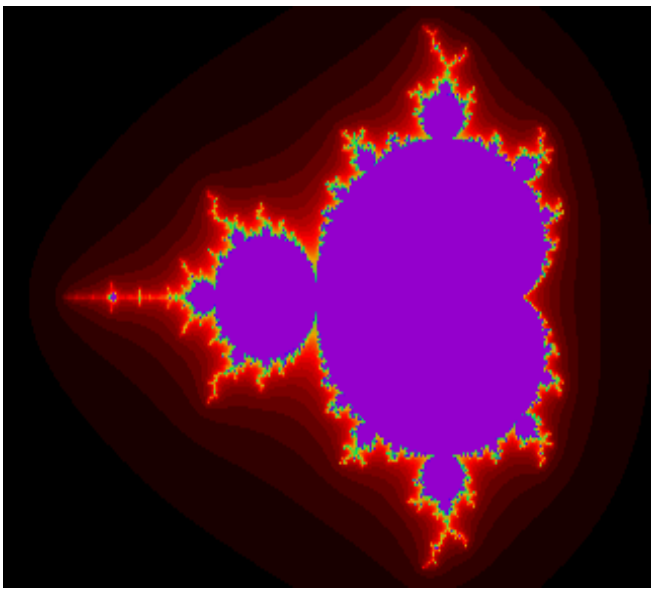

Рис. 1.4. Безліч Мандельброта

Безліч Мандельброта: картинка виходить за допомогою тієї ж процедури, що і вище. Різниця полягає в тому, що початкове значення для точки z береться завжди рівним нулю, а точці з координатами (x, y) на зображенні відповідає комплексний параметр b = x + yi.

Для того, щоб уявити все різноманіття фракталів, зручно вдатися до їх загальноприйнятої класифікації.

Геометричні фрактали. Фрактали цього класу самі наочні. У двомірному випадку їх отримують за допомогою деякої ламаної (або поверхні в тривимірному випадку), званої генератором. За один крок алгоритму кожен з відрізків, що становлять ламану, замінюється на ламану-генератор у відповідному масштабі. В результаті нескінченного повторення цієї процедури виходить геометричний фрактал.

Розглянемо один з таких фрактальних об'єктів - триадную криву Коха (рис. 1.5):

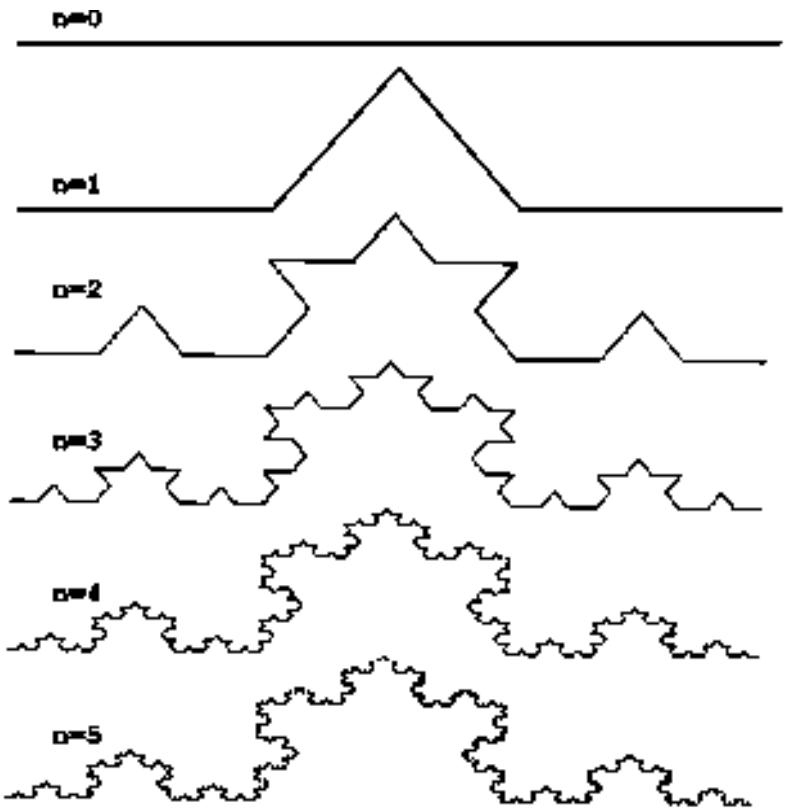

Рис. 1.5. Побудова триаідної кривої Коха

Побудова кривої починається з відрізка одиничної довжини (рис. 1.5) ─ це 0-е покоління кривої Коха. Далі кожна ланка (в нульовому поколінні ─ один відрізок) замінюється на утворюючий елемент, позначений на рис.1.5 через n=1. В результаті такої заміни виходить наступне покоління кривої Коха. У 1-му поколінні ─ це крива з чотирьох прямолінійних ланок, кожна довжиною по 1/3. Для отримання 3-го покоління проробляються ті ж дії ─ кожна ланка замінюється на зменшений утворюючий елемент. Отже, для отримання кожного наступного покоління всі ланки попереднього покоління необхідно замінити зменшеним утворюючим елементом. Крива n-го покоління при будь-якому кінцевому n називається предфракталом. На рис.1.5 представлені п'ять поколінь кривої Коха. При n, що прагне до нескінченності, крива Коха стає фрактальним об'єктом.

Для отримання іншого фрактального об'єкта потрібно змінити правила побудови. Нехай утворюючим елементом будуть два рівних відрізка, з'єднаних

під прямим кутом. У нульовому поколінні замінимо одиничний інтервал на цей утворюючий елемент так, щоб кут був зверху. Можна сказати, що при такій заміні відбувається зміщення середини ланки. При побудові наступних поколінь виконується правило: найперша зліва ланка замінюється на утворюючий елемент так, щоб середина ланки зміщалася вліво від напрямку руху, а при заміні таких ланок напрямку зсуву центрів відрізків повинні чергуватися. На рис. 1.6 представлені кілька перших поколінь і 11-е покоління кривої, побудованої за вищеописаним принципом. Гранична фрактальна крива (при n, що прагне до нескінченності) називається драконом Хартера-Хейтуея.

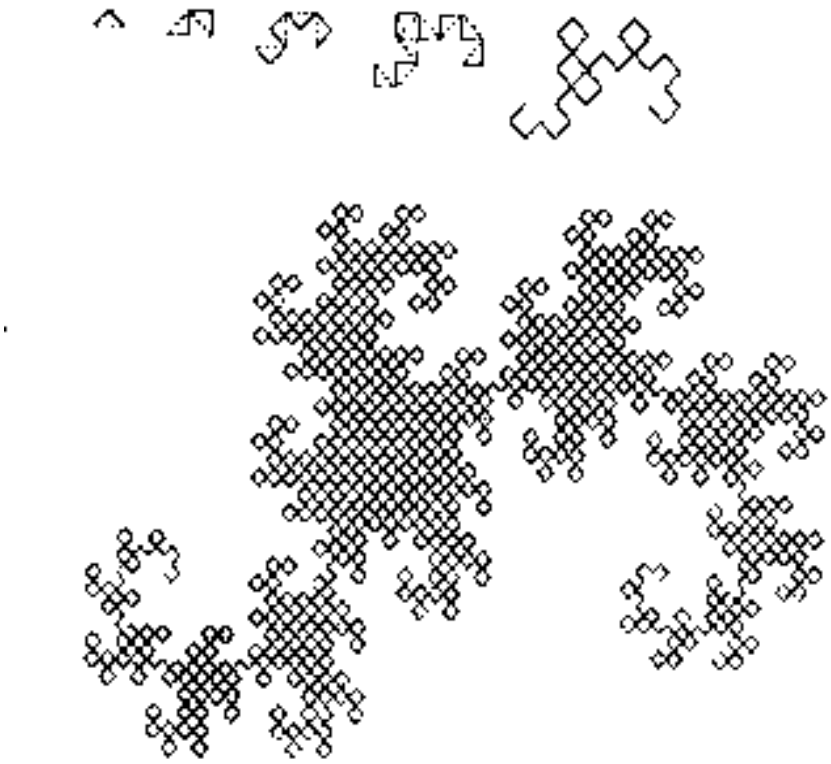

Рис. 1.6. Побудова "дракона" Хартера-Хейтуея.

У комп'ютерній графіці використання геометричних фракталів необхідно при отриманні зображень дерев, кущів, берегової лінії. Двомірні геометричні фрактали використовуються для створення об'ємних текстур (малюнка на поверхні об'єкта).

Алгебраїчні фрактали. Це найбільша група фракталів. Отримують їх за допомогою нелінійних процесів в n-мірних просторах. Найбільш вивчені двомірні процеси.

Відомо, що нелінійні динамічні системи володіють декількома стійкими станами. Той стан, в якому опинилася динамічна система після деякого числа ітерацій, залежить від її початкового стану. Тому кожний стійкий стан (аттрактор) володіє деякою областю початкових станів, з яких система обов'язково потрапить в розглянуті кінцеві стану. Таким чином, фазовий простір системи розбивається на області тяжіння аттракторів. Якщо фазовим є двомірний простір, то, фарбуючи області тяжіння різними кольорами, можна отримати колірний фазовий портрет цієї системи (ітераційний процес). Змінюючи алгоритм вибору кольору, можна отримати складні фрактальні картини з химерними кольоровими візерунками. Несподіванкою для математиків стала можливість за допомогою примітивних алгоритмів породжувати дуже складні нетривіальні структури.

Як приклад розглянемо безліч Мандельброта (рис. 1.7 і рис. 1.8).

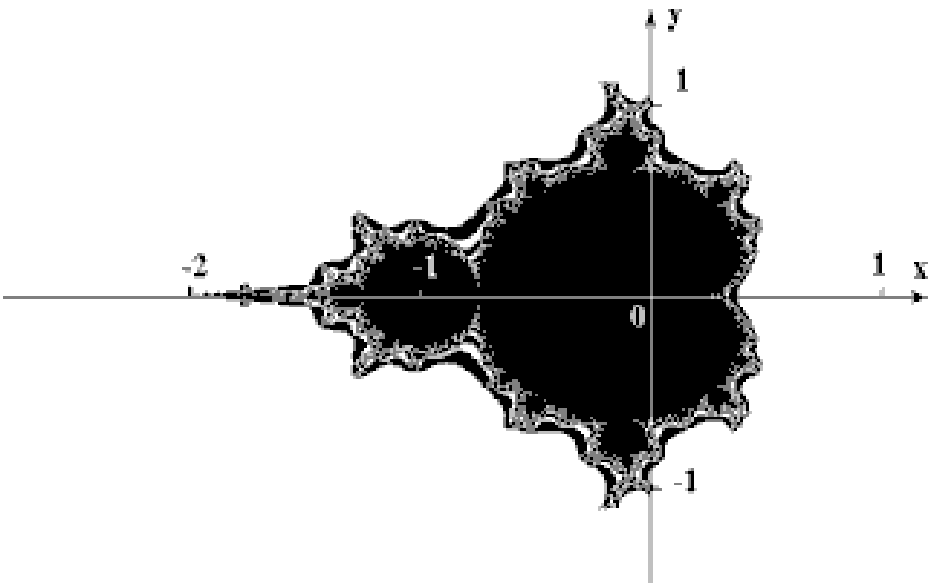

Рис. 1.7. Безліч Мандельброта

Алгоритм його побудови досить простий і заснований на простому

ітеративному вираженні:

$$Z [i + 1] = Z [i] * Z [i] + C, \quad (1.5)$$

де Z [i] і C ─ комплексні змінні.

Ітерації виконуються для кожної стартової точки C квадратної або прямокутної області підмножини комплексної площини. Ітераційний процес продовжується до тих пір, поки Z [i] не вийде за межі кола радіуса 2, центр якої лежить в точці (0,0), (це означає, що аттрактор динамічної системи знаходиться в нескінченності), або після досить великої кількості ітерацій (наприклад, 200-500) Z [i] зійдеться до якої-небудь точки кола. Залежно від кількості ітерацій, протягом яких Z [i] залишалася всередині кола, можна встановити колір точки C (якщо Z [i] залишається в колі протягом досить великої кількості ітерацій, ітераційний процес припиняється, і ця точка растра забарвлюється в чорний колір).

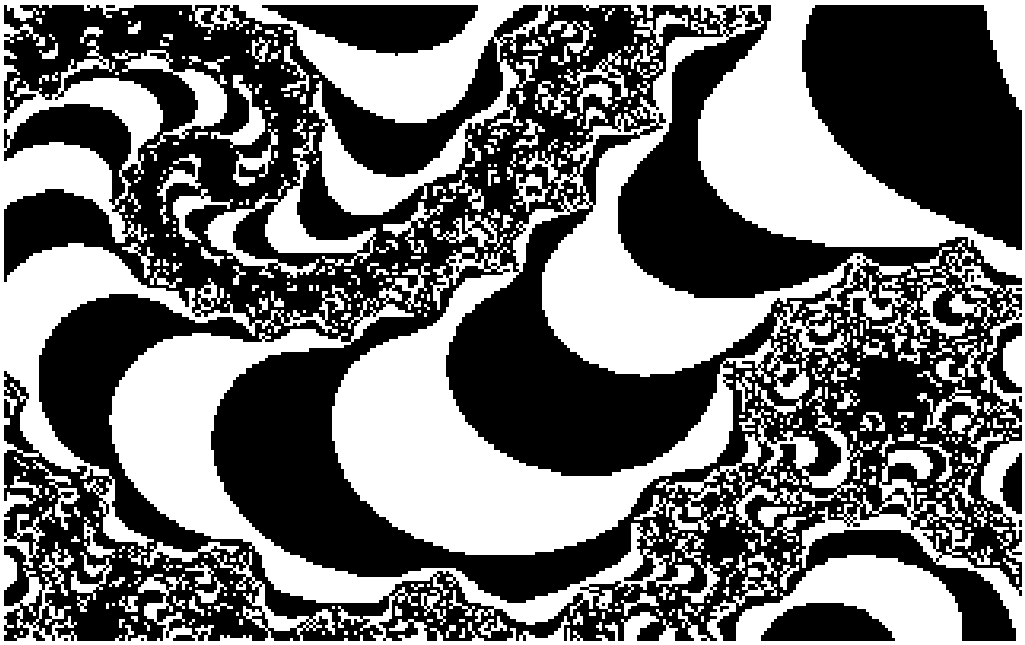

Рис. 1.8. Ділянка кордону безлічі Мандельброта, збільшений в 200 раз

Вищеописаний алгоритм дає наближення до так званої безлічі

Мандельброта. Безлічі Мандельброта належать точки, які протягом нескінченного числа ітерацій не йдуть в нескінченність (точки, що мають чорний колір). Точки, що належать кордону безлічі (саме там виникають складні структури), йдуть в нескінченність за кінцеве число ітерацій, а точки, що лежать за межами безлічі, йдуть в нескінченність через декілька ітерацій (білий фон).

Стохастичні фрактали. Ще одним відомим класом фракталів є стохастичні фрактали, які виходять в тому випадку, якщо в ітераційне процесі випадковим чином змінювати будь-які його параметри. При цьому виходять об'єкти, дуже схожі на природні несиметричні дерева, порізані берегові лінії і т.д. Двовимірні стохастичні фрактали використовуються при моделюванні рельєфу місцевості і поверхні моря. Типовий представник цього класу фракталів «Плазма» (рис. 1.9)

.

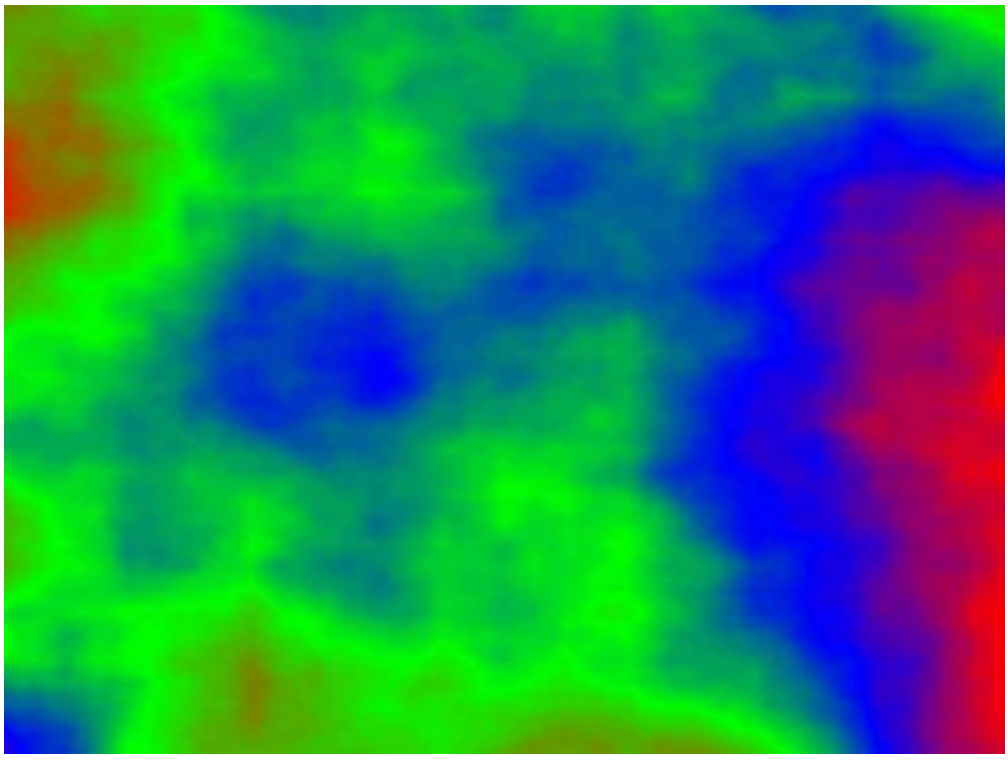

Рис. 1.9. Фрактал «Плазма»

Для його побудови візьмемо прямокутник і для кожного його кута визначимо колір. Потім знаходимо центральну точку прямокутника і

розфарбовуємо її в колір, рівний середньому арифметичному кольорів по кутах прямокутника плюс деяке випадкове число. Чим більше випадкове число - тим більше «рваним» буде малюнок. Якщо ми тепер скажемо, що колір точки це висота над рівнем моря - отримаємо замість плазми - гірський масив. Саме на цьому принципі моделюються гори в більшості програм. За допомогою алгоритму, схожого на плазму будується карта висот, до неї застосовуються різні фільтри, накладаємо текстуру і фотореалістичні гори готові (рис. 1.10).

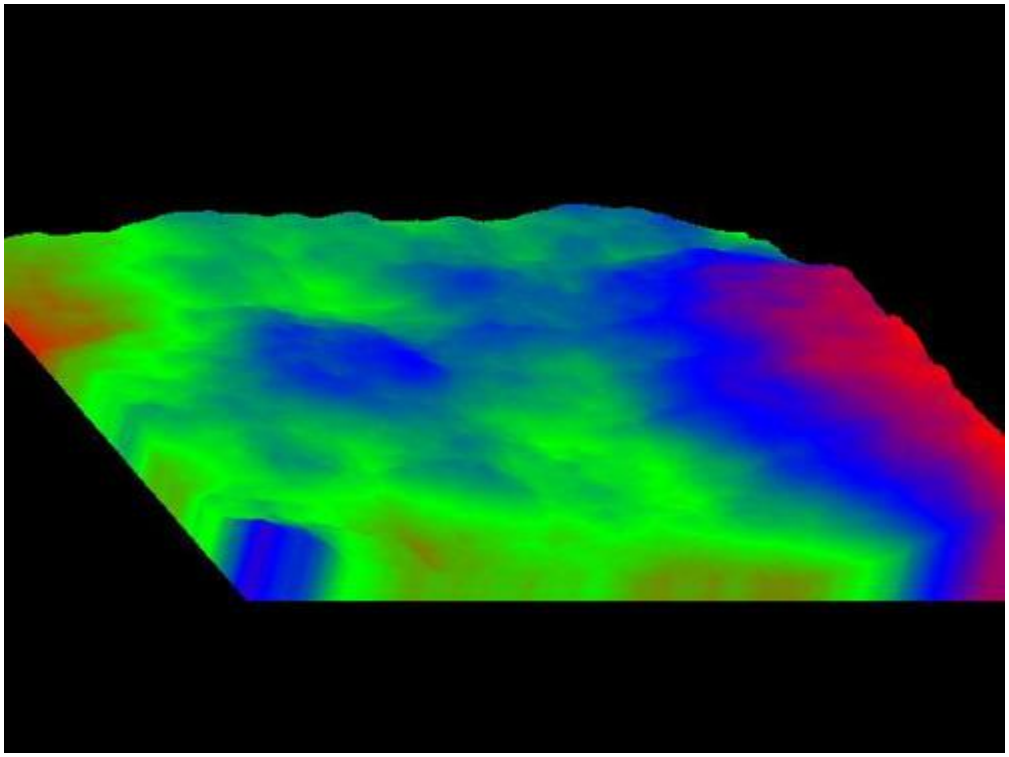

Рис. 1.10. Крта висот за допомогою фракталу

Фрактальна розмірність. У загальному випадку фрактальна розмірність є експонентою скейлінга (статичної залежності) деякого властивості об'єкта від його лінійних розмірів (масштабу виміру) і визначається наступним виразом:

$$I \sim LDf, \tag{1.6}$$

де: Df - фрактальная розмірність;

I - інтенсивність певного вимірюваної властивості об'єкта;

L - масштаб довжини (L = S1/2 для двовимірного випадку, де S - це площа поверхні ділянки об'єкта, яка відповідає певній інтенсивності властивості I).

Якщо розмірність Df відрізняється від розмірності простору (2 для плоских, 3 для об'ємних об'єктів), то такий об'єкт називають фракталом.

Однією з найбільш простих і наочних характеристик є розмірність Хаусдорфа-Безиковича (геометрична розмірність) df, яка показує, як змінюється маса об'єкта при зміні масштабу довжини:

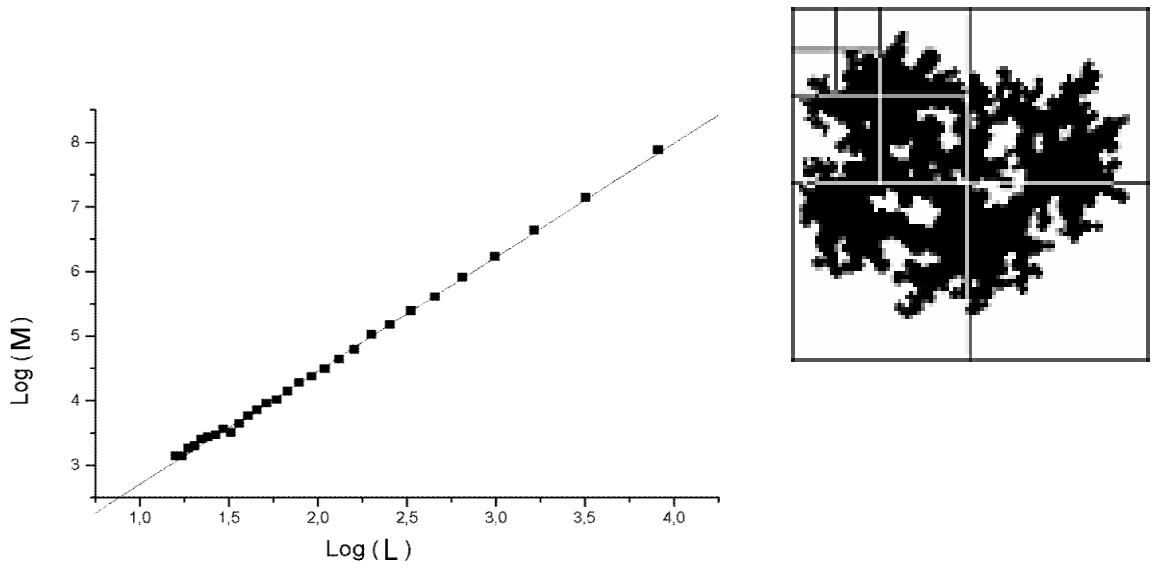

Рис. 1.11. Залежність маси фрактального об'єкта при зміні масштабу довжини

Розмірність df відповідає кутовому коефіцієнту прямолінійною залежності log (M) від log (L).

Фрактальну розмірність лінії D обчислюють, використовуючи залежність

$$L/a = (R/a)\, D, \qquad (1.7)$$

де:

L - лінійний розмір об'єкта;

a - довжина ланки ламаної лінії, яка апроксимує лінію кордону окремих кластерів;

R - сумарна довжина цієї ламаної лінії для всіх кластерів в об'єкті.

1.2. Призначення розробки та область застосування

Розвиток сучасних телекомунікаційних мереж характеризується збільшенням частки мультимедійного трафіку. Важливою частиною мультимедійного трафіку є аудіо інформація, і в тому числі мовна інформація. Проблема, що пов'язана з великим об'ємом для їх передачі і зберігання, з'явилася при роботі і на робочих станціях, і на персональних комп'ютерах. Відома велика кількість різноманітних алгоритмів стиску аудіо даних, але вони не забезпечують достатні коефіцієнти стиску, або призводять до суттєвої втрати даних, що в свою чергу пов'язано з погіршенням якості інформації. Таким чином, розробка нових підходів до стиску аудіо даних є актуальною.

В даний час найбільш перспективною і розвинутою групою алгоритмів стиснення мультимедійної інформації є фрактальні алгоритми. У них використовується принципово нова ідея - чи не близькість кольорів в локальній області, а подібність різних за розміром областей зображення. Так за допомогою стандартних прийомів обробки зображень, таких, як виділення країв і аналіз текстурних варіацій, зображення ділиться на сегменти і кодується за допомогою деякого стискаючого афінного перетворення. Відновлення зображення відбувається за допомогою багаторазового застосування цього афінного перетворення. При цьому коефіцієнт стиснення у фрактальних алгоритмів варіюється в межах 2-2000.

В результаті виконання даної роботи розроблена програма реалізує фрактальний алгоритм стиску аудіо даних.

1.3. Підстава для розробки

Підставою для розробки кваліфікаційної роботи бакалавра на тему «Розробка та програмна реалізація засобами мови С++ фрактального алгоритму стиснення аудіо даних» є наказ по Національному технічному університету «Дніпровська політехніка» від __.__. 2021р. № ____-_.

1.4. Постановка завдання

Метою роботи є розробка методів і засобів стиску аудіо даних з використанням фрактального підходу і дослідження їх ефективності.

Для реалізації мети були поставлені такі задачі:

1. Провести аналітичний огляд методів стиску аудіо даних.
2. Вивчити математичні основи геометрії фракталів та методів фрактального стиску зображень.
3. Розробити математичне обґрунтування можливостей фрактального стиску аудіо даних.
4. Розробити алгоритми фрактального стиску аудіо даних та їх програмні реалізації.
5. Дослідити ефективність фрактального стиску аудіо даних.

Для виконання теми кваліфікаційної роботи необхідно на основі існуючого методу фрактального стиснення растрових графічних зображень розробити математичне обґрунтування можливості фрактального стиснення аудіо, алгоритм стиснення і його програмну реалізацію; провести дослідження ефективності алгоритму (його швидкодії, ступеня стиснення і якості сигналу) від параметрів алгоритму і стиснення цифрового звуку.

1.5. Вимоги до програми або програмного виробу

1.5.1. Вимоги до функціональних характеристик

Розроблена програма має реалізовувати алгоритми фрактального стиснення аудіо даних. Консольний додаток повинен компресувати дані з файлу формату .wav та виконувати їх декомпресію з найменшими втратами та мінімальним часом на виконання даних операцій.

1.5.2. Вимоги до інформаційної безпеки

Для надійної роботи додатку необхідно:

— використовувати ліцензійне програмне забезпечення на сервері;

— здійснювати захист від вірусів на сервері;

— здійснювати захист від несанкціонованого доступу;

— застосовувати на сервері джерело безперебійного живлення для захисту від перепадів напруги або збоїв у живленні.

1.5.3. Вимоги до складу та параметрів технічних засобів

Необхідні мінімальні технічні вимоги для роботи з даним програмним продуктом мають такі характеристики:

1. Процесор 1.3 ГГц.
2. Оперативна пам’ять: 1 Гб.
3. Операційна система: Windows 8.1 64 Bit, Windows 8 64 Bit, Windows 7 64 Bit Service Pack 1.
4. Роздільна здатність монітору: 1280x800.
5. Наявність засобів вводу (клавіатура, мишка).

Вільне місце на жорсткому диску: 10 Мб.

1.5.4. Вимоги до інформаційної та програмної сумісності

Програмний код для реалізації даної кваліфікаційної роботи має бути створений за допомогою мови програмування C++ у середовищі розробки програмного забезпечення Code::Blocks. Програма має бути откомпільована та запускатися за допомогою виконуваного файлу в усіх версіях родини операційних систем Windoows.

РОЗДІЛ 2
ПРОЄКТУВАННЯ ТА РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ

2.1. Функціональне призначення програми

Розроблена програма реалізовує алгоритми фрактального стиску аудіо даних, який реалізований на основі існуючого методу фрактального стиснення растрових графічних зображень.

2.2. Опис застосованих математичних методів

2.2.1. Метод фрактального стиснення зображень

У фрактальному стисненні використовується принципово нова ідея - подоба різних за розміром областей зображення.

Розглянемо деяке зображення (картину). Якщо це зображення складається з відносно невеликого числа самоподібних фракталів, тоді зображення може кодуватися (стискатися) за допомогою запам'ятовування коефіцієнтів відповідних афінних перетворень (рис.2.1) і може бути зібрано, відновлено в ході ітераційного процесу подібно до того, як збирається трикутник Серпінського або крива Коха.

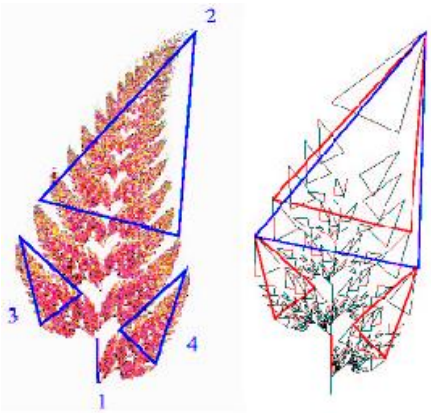

Рис. 2.1. Виділення подібних областей 2, 3, 4 в зображенні листа: стрижень 1 - це також трикутник.

Фрактальное стиснення - це пошук подібних областей в зображенні і визначення для них коефіцієнтів афінних перетворень. Істотною є умова, що кожне таке афінне відображення є стискаючим, тобто зменшує зображення («більше в меншу»). Завдяки даній умові, теорема Банаха про нерухому точку гарантує складання зображення в ході декомпресії, тобто в ході ітераційного процесу. Набір стискаючих відображень, що беруть участь в ітераціях, називають системою ітеруємих функцій (Iterated Function System - IFS).

Зазвичай комп'ютерні зображення виражаються у вигляді пікселів в сітці. Всі графічні формати в даний час емулюють сітку пікселів, яка необхідна, щоб відобразити образ на моніторі або надрукувати на принтері. Особливості такого способу зрозумілі: більше пікселів - краще якість, але більше файл.

Ітераційний спосіб зберігання, або стиснення зображень передбачає використання його рекурсивних особливостей. Це дозволяє стиснути 921 Кб BMP в ітераційне зображення, яке потребує всього 10 Кб. Ітераційні зображення можуть розглядатися як з низьким дозволом, так і з більш високим, ніж вихідне. Звичайно, подібне можна зробити і з пиксельними зображеннями, тільки вони потім стають розмитими. Якщо ж масштабувати зображення, стислим ітераційним способом, то буде видно все більш дрібні деталі.

Використання IFS для стиснення звичайних зображень засновано на виявленні локальної самоподібності (коли ознакою самоподібності володіють області зображення, а не вона сама), на відміну від фракталів, де спостерігається глобальне самоподоба і знаходження IFS не надто складно. За алгоритмом Барнслі [20] відбувається виділення в зображенні пар областей, найменша з яких подібна до більшої, і збереження декількох коефіцієнтів, що кодують перетворення, що переводить більшу область в меншу. Потрібно, щоб безліч «менших» областей покривало все зображення. При цьому в файл, який кодує зображення, будуть записані не тільки коефіцієнти, що характеризують знайдені перетворення, а й місце розташування і лінійні розміри «великих» областей, які разом з коефіцієнтами будуть описувати локальну самоподобу кодованого зображення. Відновлювальний алгоритм в цьому випадку повинен

застосовувати кожне перетворення не до всієї безлічі точок, одержаних на попередньому кроці алгоритму, а до деякої їх підмножини, що належить області, відповідної застосованому перетворенню.

2.2.2. Математичні основи теорії фрактального стиснення

Метричний простір.Безліч M називається метричним простором, якщо задано відображення ρ: M M → R таке, що

$$\rho(a, b) \geq 0 \text{ і } \rho(a, a);$$
$$\rho(a, b) = \rho(b, a);$$
$$\rho(a,b) \leq \rho(a,c) + \rho(c,b). \qquad (2.1)$$

Функція ρ називається метрикою. По суті, вона характеризує відстань між точками простору M.

Для метричного простору використовують позначення <M, ρ>.

Простір <M, ρ> називається повним, якщо будь-яка фундаментальна послідовність {xn} $\in$ M сходиться до деякій точці x0 $\in$ M, тобто

$$\lim_{n\to\infty, m\to\infty} \rho(x_n, x_m) = 0 \Rightarrow \exists x_0 \left(\lim_{n\to\infty} \rho(x_n, x_0) = 0 \right). \qquad (2.2)$$

Теорема Банаха про нерухому точку. Розглянемо метричний простір <M, ρ>.

Відображення ω: M → M називається стискає, якщо існує число s, 0 <s <1, таке, що для будь-яких двох точок a, b $\in$ M

$$\rho(\omega(a), \omega(b)) < s \cdot \rho(a, b). \qquad (2.3)$$

Точка а ∈ М називається нерухомою для відображення ω, якщо ω (а) = а.

Теорема (Банах). Нехай <М, ρ> повний метричний простір, а ω М → М стискуюче відображення. Тоді ω має єдину нерухому точку х0 в М, яка є межею ітераційного процесу

$$x_0 = \lim_{n\to\infty} \omega_n(a) \tag{2.4}$$

$$\omega_n(a) = (\omega \circ \ldots \circ \omega)(a), \tag{2.5}$$

де а ∈ М довільна точка.

Метрика Хаусдорфа. Розглянемо метричний простір <М, ρ>.

Нехай К (М) безліч всіх компактних (тобто замкнутих і обмежених) підмножин простору М.

Визначимо метрику Хаусдорфа на безлічі К (М):

$$\rho_H(K,K') = \max\left\{\sup_{x\in K'} \rho(x,K), \sup_{x\in K} \rho(x,K')\right\}, \tag{2.6}$$

$$\text{Де } \rho(x,K) \equiv \inf_{y\in K} \rho(x,y) \tag{2.7}$$

– відстань від точки х до безлічі К.

Теорема. Якщо <М, ρ> повний метричний простір, то <К(М), ρН> повний метричний простір.

Слідство. Якщо дано відображення W: K(M)→K(M), певне в такий спосіб

$$W(K) \equiv \bigcup_{i=1}^{N} \omega_i(K), \tag{2.8}$$

де ωi: М → М (i = 1, ..., N) відображення простору М, і воно є стискає щодо метрики Хаусдорфа, тобто

$$\rho_H(W(K),W(K')) < s \cdot \rho_H(K,K') \tag{2.9}$$

для деякого s, 0<s<1, і будь-яких К, К'$\in$ К(М), то воно має нерухому точку $К_0 \in$К (М):

$$K_0=W(K_0)\ . \tag{2.10}$$

Ця точка єдина і може бути отримана в результаті ітерацій

$$K_0 = \lim_{n\to\infty} W_n(A), \tag{2.11}$$

де А $\in$ К (М) довільне компактне безліч.

Можна показати, що в разі стискання ω_i: М→М (i=1, ..., N) стискання є і W.

Афінне перетворення - композиція лінійного перетворення і паралельного перенесення.

IFS - Система ітераційних функцій - це система, що складається з набору афінних перетворень w_1, w_2, ..., w_n, що працюють в просторі Хаусдорфа <К(М),ρН>.

PIFS - Система сегментованих ітераційних функцій - це система, що складається їх набору афінних перетворень $w_1, w_2, \ldots, w_n$, що працюють в просторі Хаусдорфа. Причому, кожне w_i перетворює тільки певну частину простору, позначену Мμ, тобто w_i: Мμ→М.

Таким чином, ми маємо математичне обґрунтування для побудови різних алгоритмів фрактального стиснення. Набір відображень ω_i: М→М (i = 1, ..., N) в даному випадку - це необхідна для фрактального стиснення система ітеруємих функцій.

2.3. Опис використаної архітектури та шаблонів проектування

Життєвий цикл програмного забезпечення – це безперервний процес, який починається з моменту прийняття рішення про необхідність створення ПЗ і закінчується в момент його повного вилучення з експлуатації.

Існує декілька підходів при визначенні фаз та робіт життєвого циклу програмного забезпечення (ЖЦПЗ), кроків процесу програмування, каскадна і спіральна моделі. Але всі вони містять загальні основні компоненти: постановка завдання, проектування рішення, реалізація, обслуговування.

В даному проекті використаний процес програмування по Райлі.

Процес програмування включає чотири кроки:

— постановка задачі, тобто отримання адекватного уявлення про те, яке завдання має виконати програма;

— проектування рішення вже поставленого завдання (загалом, таке рішення є менш формальним, ніж остаточна програма);

— кодування програми, тобто переклад спроектованого рішення в програму, яка може бути виконана на машині;

— супровід програми, тобто безперервний процес усунення в програмі неполадок і додавання нових можливостей.

Програмування починається з того моменту, коли користувач, тобто той, хто потребує програми для вирішення завдання, викладає проблему системного аналітику. Користувач і системний аналітик спільно визначають постановку задачі. Остання потім передається алгоритмісту, який відповідає за проектування рішення. Рішення (або алгоритм) представляє послідовність операцій, виконання яких призводить до вирішення завдання. Оскільки алгоритм часто не пристосований до виконання на машині, його слід перевести в машинну програму. Ця операція виконується кодувальником. За наступні зміни в програмі несе відповідальність супроводжуючий програміст. І системний аналітик, і алгоритміст, та кодер, і супроводжуючий програміст - всі вони є програмістами.

В разі великого програмного проекту число користувачів, системних аналітиків і алгоритмістів може виявитися значним. Крім того, може виникнути необхідність повернутися до попередніх кроків в силу непередбачених обставин. Все це служить додатковим аргументом на користь ретельного проектування програмного забезпечення: результати кожного кроку повинні бути повними, точними і зрозумілими.

Одним з найбільш важливих кроків програмування є постановка задачі. Вона виконує функції контракту між користувачем і програмістом (програмістами). Як і юридично погано складений контракт, погана постановка задачі марна. При гарній постановці завдання як користувач, так і програміст ясно і недвозначно представляють завдання, яку необхідно виконати, тобто в цьому випадку враховуються інтереси як користувача, так і програміста. Користувач може планувати використання ще нествореного програмного забезпечення, спираючись на знання того, що воно може. Гарна постановка завдання служить основою для формування її рішення.

Постановка завдання (специфікація програми), по суті, означає точне, повне і зрозуміле опис того, що відбувається при виконанні конкретної програми. Користувач зазвичай дивиться на комп'ютер, як на чорний ящик: для нього неважливо, як працює комп'ютер, а важливо, що може комп'ютер з того, що цікавить користувача. При цьому основна увага фокусується на взаємодії людини з машиною.

Д. Райлі пропонує для постановки задачі користуватися стандартною формою, яка забезпечує максимальну точність, повноту, ясність і включає:

— найменування завдання (схематичне визначення);

— загальний опис (короткий виклад завдання);

— введення;

— висновок;

— помилки (явно перераховані незвичайні варіанти введення, щоб показати користувачам і програмістам ті дії, які зробить машина в подібних ситуаціях);

— приклад (хороший приклад може передати сутність завдання, а також проілюструвати різні випадки).

Проектування рішення програмування є найбільш важким. На даній стадії постановка задачі має бути перетворена на алгоритм. Тому алгорітміст повинен володіти достатнім досвідом програмування і підходити до кожної нової задачі, спираючись на твердо встановлену методику проектування. Щоб уникнути помилок в програмах, алгорітмісти повинні використовувати ретельно розроблені процедури конструювання, засновані на правилах логічного висновку.

Завдання проектувальника – створення алгоритму, що виконує функції сполучної ланки між постановкою завдання і готова для виконання програмою. Перевірку створеного алгоритму, тобто наскільки останній відображає постановку завдання, здійснює системний аналітик. У силу цього і системний аналітик, і проектувальник повинні вміти читати і розуміти алгоритм. Кожен алгоритм записується на деякій псевдомова. Алгоритми, звані також псевдокод, не можуть бути виконані ні на якому комп'ютері.

Кодування алгоритму полягає в перекладі алгоритму в програму. Для створення повної, точної та зрозумілої програми необхідні відповідні методи запису програм. На відміну від природних, мови програмування створені спеціально для такого подання рішення завдання, яке може бути виконано комп'ютером.

Перш ніж завершити роботу, кодіровщік повинен переконатися, що програма відповідає псевдокод. Потім системний аналітик, алгоритміст і, що найголовніше, користувач повинні протестувати і підтвердити, що вона працює правильно. Після цього можна вважати, що програма готова для передачі користувачеві в комплекті з усією необхідною документацією.

Однак на цьому програмування не закінчується, далі слідує крок супроводу. Справа в тому, що в програмі можуть бути помилки, зумовлені або неадекватною постановкою завдання, або тим, що проект не задовольняє постановці задачі або програма не відповідає проекту. Яка б не була причина,

користувач має право вимагати коригування програми, оскільки він не уявляв, що програма буде працювати таким чином. Виправлення помилок є однією з головних завдань супроводу програм. Інший не менш важливим завданням супроводу програм є її модифікація, тобто додавання в програму нових можливостей або зміна існуючих. Користувач може змінити вимоги до роботи програми, що, у свою чергу, призведе до необхідності її переписати. Складність операцій із супроводу програми залежить від типу змін, які повинні бути зроблені: у гіршому випадку може знадобитися повна переробка програми від постановки до кодування. Зазвичай на супровід програми витрачається більше часу, ніж на її створення.

Останньою складовою процесу програмування є документування. Воно включає широкий спектр описів, що полегшують процес програмування і збагачують результуючу програму. Постійне документування має становити невід'ємну частину кожного кроку програмування. Постановка завдання, проектні документи, алгоритми і програми - все це документи. Внутрішня документація, зазначена безпосередньо в програму, полегшує читання коду.

Відповідно до моделі, представленої в цьому проекті, програмування можна розділити на чотири кроки: постановку задачі, проектування рішень, кодування програми, супровід програми. Додатково модель включає документування програми як дії, які необхідно виконувати протягом всього процесу програмування.

2.4. Опис використаних технологій та мов програмування

Для програмної реалізації даної інформаційної системи була використана мова програмування C++. За середовище розробки програмного забезпечення був обраний CodeBlocks.

Мова C++ є в даний час найбільш поширеною та перспективною мовою промислового програмування. Вона містить найбільш повний набір властивостей і можливостей, вироблених всією історією розвитку мов

програмування. До істотних характерних властивостей C++ слід віднести насамперед потужну підтримку об'єктно-орієнтованого підходу до розробки програм та механізм параметризації типів і алгоритмів. Широкий діапазон типів і розвинені можливості побудови користувацьких типів дозволяють адекватно відобразити особливості предметної області; суворі правила поводження з константними типами сприяють надійності програм. Підвищення надійності створюваних програм служить простий і гнучкий апарат керування винятковими ситуаціями. Розвинені схеми перетворення та приведення типів дозволяють забезпечити достатній компроміс між строгою типізацією і ефективністю виконання програм. Кошти явного управління областями дії ("простору імен") надають зручний механізм структурування великих програм.

C++ є прямим наступником мови C і фактично включає його як підмножину. Тим самим, C++ цілком містить добре зарекомендувала себе традиційну модель обчислень мови, у тому числі, розвинений общеалгоритмический базис, широкі можливості конструювання нових типів і гнучкі засоби роботи з пам'яттю, включаючи арифметику над покажчиками. Ця обставина забезпечує збереження в актуальному стані мільйони рядків програмного тексту, розробленого на C, і дає додаткові гарантії широкого використання C++.

Крім широкої поширеності і популярності, в тому числі, і у вітчизняній практиці програмування, мова C++ служить технологічною основою перспективною парадигми, яка виникла в недавній час, - узагальненого програмування. Основним інструментом реалізації узагальненого програмування на мові C++ служить механізм шаблонів і переконливим прикладом використання цієї парадигми є Стандартна Бібліотека Шаблонів (Standard Template Library, STL), розроблена А. Степановим та М. і ввійшла в 1994 р. до складу стандартної бібліотеки C++.

Ще одна обставина, що обумовило вибір мови C++ прийняття наприкінці 1998 р. Міжнародного Стандарту ANSI/ISO мови. Факт стандартизації для такого великого, складного і сучасної мови, як C++, важко переоцінити. Якщо

говорити коротко, C++ стає інструментом промислового програмування в загальносвітовому масштабі. Прихильність все більшого числа корпоративних розробників програмного забезпечення до використання типових рішень і стандартизованих інструментальних засобів дає тверду упевненість в успішних перспективах C++, принаймні, на найближчі десять-п'ятнадцять років (приблизний термін зміни поколінь мов програмування).

CodeBlocks - це вільна кросплатформова середа, що заповнює нішу між «дорослими» системами для великих проектів, типу Eclipse, Visual Studio, Net Beans, і убогими по функціоналу, але спритними блокнотами типу Scintilla, причому переваги і тих, і інших складаються і дозволяють використовувати дану систему як для написання невеликих проектів для вбудованих додатків, так і для програмування додатків для PC під Windows, Linux і MacOs.

Основні характерні особливості середовища:

— кроссплатформова IDE з відкритим кодом, заснована на бібліотеці wxWidgets;

— компактне ядро і розширення функціоналу за допомогою безлічі плагінів;

— вбудований інтерфейс під безліч компіляторів і тулчейнов;

— безліч візардів для швидкого створення шаблону проекту як для різноманітних мікропроцесорних архітектур (AVR, ARM, PowerPC), так і для бібліотек та тулкітів під ПК: GTK, Qt, WxWidgets, OpenGL ітд;

— компактна та інтуїтивно зрозуміла структура меню, що забезпечує швидке налаштування середовища.

Відмінною особливістю середовища розробки CodeBlocks є те, що в ньому реалізована підтримка безлічі компіляторів. Це дуже зручно, наприклад, коли вам потрібно відкрити вже написану під якийсь компілятор раніше програму. Або, спробувати відкомпілювати свій проект за допомогою різних компіляторів. Кожен компілятор має свої особливості і обсяг видається після компіляції файлів. Для кожного проекту ви можете вибрати свій компілятор: вибрати можна як при створенні проекту, так і вже в створеному проекті через

властивості. Список підтримуваних компіляторів наведено нижче.

- MinGW / GCC C / C ++;
- GNU ARM GCC Compiler;
- GNU AVR GCC Compiler;
- GNU GCC Compiler for PowerPC;
- GNU GCC Compiler for TriCore;
- Digital Mars C / C ++;
- Digital Mars D (з деякими обмеженнями);
- SDCC (Small device C compiler);
- Microsoft Visual C ++ 6 ;
- Microsoft Visual C ++ Toolkit 2003;
- Microsoft Visual C ++ 2005/2008 (з деякими обмеженнями);
- Borland C ++ 5.5;
- Watcom;
- Intel C ++ compiler;
- GNU Fortran;
- GNU ARM;
- GNU GDC.

Також в CodeBlocks є підтримка багатопрофільних проектів, підтримка робочих просторів. Є можливість імпорту проектів, створених в середовищі Dev-C ++, імпорт проектів та робочих просторів Microsoft Visual Studio.

Інтерфейс середовища забезпечує наступні функції:

- підсвічування синтаксису;
- згортання блоків коду;
- автодоповнення коду;
- браузенр класів;
- скриптовой движок Squirrel;
- планувальник під кілька користувачів;
- підтримка плагінів Devpack (installation packages for Dev-C ++);
- плагін wxSmith (a wxWidgets RAD tool);

— налагодження.

2.5. Опис структури програми та алгоритмів її функціонування

2.5.1. Алгоритм стиснення

Комплексні функції

$$\omega = az, \quad \omega = e^{i\varphi}, \quad \omega = z + c \tag{2.12}$$

реалізують власне стиснення (розтягнення) з різним коефіцієнтом уздовж осей x і y, поворот навколо 0 на кут φ і перенесення на вектор = (Re c, Im c). З лінійної алгебри та аналітичної геометрії відомо, що будь-яке афінне перетворення евклідової площини виду

$$\omega(x, y) = (x, y) \cdot \begin{pmatrix} \alpha & \beta \\ \gamma & \delta \end{pmatrix} + \begin{pmatrix} \lambda \\ \mu \end{pmatrix} \tag{2.13}$$

є композицією стиснення (розтягування), повороту і перенесення. Іншими словами, перетворення ω в своїй комплексній формі є композиція комплексних функцій виду (2.13).

Однак в теорії фрактального стиснення перевагу віддано дійсної формі для системи ітеруємих функцій. Це пов'язано з тим, що зображення на екрані описується не тільки координатами (x, y) пікселів, але і, наприклад, яскравістю зображення. А це третій параметр, і, отже, виникає необхідність від описів на площині переходити до описів в просторі.

У зв'язку зі сказаним зручно розглядати 3-мірне афінне перетворення виду:

$$\omega(x,y,z)=(x,y,z)\cdot\begin{pmatrix}\alpha & \beta & 0\\ \gamma & \delta & 0\\ 0 & 0 & \eta\end{pmatrix}+\begin{pmatrix}\lambda\\ \mu\\ \nu\end{pmatrix}, \tag{2.14}$$

де координата z є координата яскравості зображення, і, отже, коефіцієнт η, 0<η<1, що характеризує зменшення (відображення має бути стискає) яскравості в η разів.

Нехай задано квадратне зображення, що містить N×N пікселів. Позначимо через $\Omega=(0,N]\times(0,N]\in R^2$ безліч точок зображення. Розбиваємо спочатку Ω на безліч з r попарно рівних квадратних блоків

$$R_1,\ldots,R_r$$

$$R_i\cap R_j = 0 \quad (i\neq j),\ \bigcup_{i=1}^{r} R_i=\Omega \tag{2.15}$$

де Ri є квадратний $p\times p$ піксельний фрагмент зображення, а потім розіб'ємо зображення на нове безліч з d квадратних блоків D1, ..., Dd, де $D_i\subseteq\Omega$ представляють квадратні $2p\times 2p$ (можливо пересічні) піксельні фрагменти і

$$\bigcup_i D_i=\Omega \tag{2.16}$$

.

Блоки R_i іменуються ранговими, а блоки D_i - доменними. Доменний блок в 2 рази більше рангового.

Головне завдання стиснення - це побудова системи ітеруємих функцій (IFS)

$$w_i : D_{j(i)} \to R_i, \quad w_i(D_{j(i)}) = R_i \quad (i=1,\ldots,r) \tag{2.17}$$

виду (2.14), де індекс j (i) означає номер того доменного блоку, який зіставляється у фрактальному алгоритмі рангового блоку R_i. Знаходження

коефіцієнтів афінних стискань відображень ω_i і їх збереження в файл вирішує завдання стиснення (компресії).

Візьмемо блок R_i. Треба знайти для нього найбільш «подібний» блок D_j (і) (рис. 2.2.):

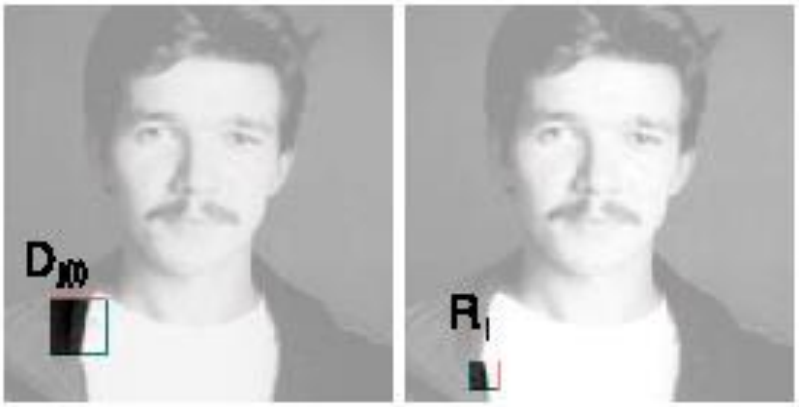

Рис. 2.2. Подібні блоки

Будемо в блоках D_j пікселі брати через один (як по х, так і по у). Це дозволяє вважати, що в D- блоках і R- блоках по однаковому числу пікселів - р.

При перекладі D- блоку Dj в R- блок R_i за допомогою перетворення (2.14), крім перенесення на вектор (λ, μ), використовуються матриці

$$\begin{pmatrix} \alpha & \beta \\ \gamma & \delta \end{pmatrix}, \tag{2.18}$$

які є композиціями обов'язкового стиснення в 2 рази, повороту блоку D_j на 00, 900, 1800 або 2700 і дзеркального відображення (щодо вертикалі або горизонталі). Це дає всього 8 варіантів для вибору матриці (2.14), (рис. 2.3).

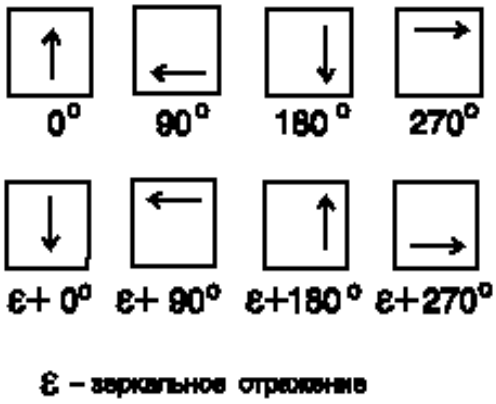

Рис. 2.3. Варіанти для матриці (2.14)

Позначимо через $r^i_{\alpha,\beta}$, $d^i_{\alpha,\beta}$ $(\alpha,\beta=1,\ldots,p)$ яскравості пікселів відповідно в блоці R_i і в блоці $D_j(i)$, підданої афінних перетворень (2.14).

Будемо зрушення по яскравості - число v_i в (3*) обчислювати за формулою:

$$v_i = \frac{1}{p^2} * \sum_{\substack{\alpha=0 \\ \beta=0}}^{p} (r^i_{\alpha,\beta} - \eta_i \cdot d^i_{\alpha,\beta}), \tag{2.19}$$

а відстань між блоками:

$$\rho(R_i, D_{j(i)}) = \sum_{\substack{\alpha=0 \\ \beta=0}}^{p} (\eta_i d^i_{\alpha\beta} + v_i - r^i_{\alpha\beta})^2, \tag{2.20}$$

де число η_i часто беруть рівним 0,75.

Підбір блоку $D_j(i)$ полягає в мінімізації відстані між блоками $\rho(R_i, D_j)$. Знаходимо блок $D_j(i)$ з найменшою відстанню і зберігаємо коефіцієнти відповідного перетворення ω_i в файл. Перебираючи рангові блоки послідовно, обчислюємо коефіцієнти всіх стискаючих афінних перетворень $\omega_1,\ldots,\omega_r$, що утворюють шукану систему ітеруємих функцій.

2.5.2. Алгоритм декомпресії

Процес декомпресії, відновлення зображень полягає в проведенні ітерацій системи $\omega_i (i=1,\ldots,r)$ до стабілізації отриманого зображення (рис. 2.4).

Рис. 2.4. Збірка (декомпресія) зображення.

Декомпресія алгоритму фрактального стиснення надзвичайно проста. Необхідно провести кілька ітерації двовимірних афінних перетворень, коефіцієнти яких були отримані на етапі компресії.

В якості початкового може бути взято будь-яке зображення (наприклад, абсолютно чорне), оскільки відповідний математичний апарат гарантує нам збіжність послідовності зображень, одержуваних в ході ітерацій IFS, до нерухомого зображення (близького до початкового). Зазвичай для цього

достатньо 16 ітерацій [18].

2.5.3. Метод фрактального стиснення аудіо

2.5.3.1. Ідея фрактального стиснення аудіо

У фрактальному стисненні аудіо будемо використовувати ідею, засновану на подобі різних за розміром послідовностей відліків звукового сигналу.

Розглянемо деякий сигнал. Якщо цей сигнал складається з відносно невеликого числа (самоподібних) фракталів, тоді сигнал може кодуватися (стискатися) за допомогою запам'ятовування коефіцієнтів відповідних афінних перетворень (рис. 2.5) і може бути зібраний, відновлений в ході ітераційного процесу.

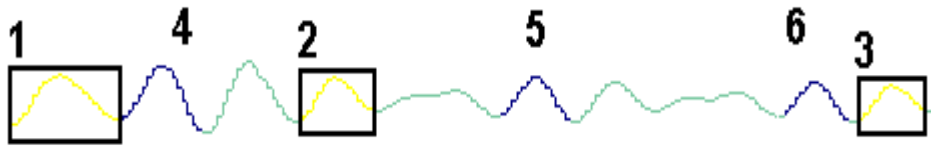

Рис. 2.5. Виділення подібних послідовностей 1, 2, 3 і 4, 5, 6 в звуковому сигналі

Використання IFS для стиснення звукових сигналів засноване на виявленні локального самоподібності (коли ознакою самоподібності мають послідовності відліків сигналу, а не він сам), на відміну від фракталів, де спостерігається глобальна самоподоба і знаходження IFS не надто складно. За аналогією з алгоритмом Барнслі [18] відбувається виділення в сигналі пар послідовностей, менша з яких подібна до більшої, і збереження декількох коефіцієнтів, що кодують перетворення, що переводить більшу послідовність у меншу. Потрібно, щоб безліч «менших» послідовностей покривало весь сигнал. При цьому в файл, який кодує сигнал, будуть записані не тільки коефіцієнти,

що характеризують знайдені перетворення, а й місце розташування і лінійні розміри «великих» послідовностей, які разом з коефіцієнтами будуть описувати локальну самоподобу кодованого сигналу. Відновлювальний алгоритм в цьому випадку повинен застосовувати кожне перетворення не до всієї безлічі точок, одержаних на попередньому кроці алгоритму, а до деякої їх підмножини, що належить послідовності, відповідної застосованого перетворення.

2.5.3.2. Визначення коефіцієнтів стискаючих перетворень

Нехай заданий звуковий сигнал з N відліків. Позначимо через безліч відліки сигналу. Розбиваємо спочатку Ω на безліч з r попарно рівних послідовностей

$$R_1,\ldots,R_r$$

$$R_i \cap R_j = 0 \quad (i \neq j), \quad \bigcup_{i=1}^{r} R_i = \Omega \tag{2.21}$$

де R_i є послідовність відліків сигналу довжиною p.

А потім розіб'ємо сигнал на нове безліч з d послідовностей

$$D1, ..., Dd, \tag{2.22}$$

де $D_i \subseteq \Omega$ представляють послідовності відліків (можливо пересічні) довжиною $2p$ і

$$\bigcup_i D_i = \Omega \quad . \tag{2.23}$$

Послідовності R_i є ранговими, а послідовності D_i - доменними. Доменна послідовність в 2 рази більше рангової.

Головне завдання стиснення - це побудова системи ітеруємих функцій (IFS)

$$w_i : D_{j(i)} \to R_i, \quad w_i(D_{j(i)}) = R_i \quad (i = 1, ..., r) \quad , \qquad (2.34)$$

де індекс j(i) означає номер тієї доменної послідовності, яка зіставляється у (фрактальному) алгоритмі рангової послідовності R_i.

У загальному вигляді афінне перетворення ω: $X^n \to X^n$ завжди може бути записано як ω = Ax + b, де $A \in X^{n \times n}$, це $n \times n$ матриця і $b \in X^n$, це вектор зміщення. У випадку з одновимірним звуковим сигналом ми маємо справу з лінійною частиною перетворення

$$\omega = sx + o, \qquad (2.34)$$

де s це масштабний коефіцієнт, а o - зсув за рівнями відліків послідовності сигналу.

При фрактальному кодуванні відображення ω підбирається таким чином, щоб мінімізувати відстань між вектором x та його образом в деякій метриці.

Відповідно до теореми Банаха, якщо ω - стискає відображення і (X, ρ) - повний метричний простір з метрикою ρ, тоді послідовність, побудована за правилом

$$x_{k+1} = \omega(x_k), \qquad (2.35)$$

сходиться для довільної точки $x_0 \in X$ до єдиної нерухомої точки $x_\omega \in X$ перетворення ω:

$$x_\omega = \omega(x_\omega). \qquad (2.36)$$

Тому для відновлення сигналу необхідно запам'ятати лише відображення ω.

Неважко бачити, що коли множина відображень ω таке, що

$$L = \max_i |s_i| < 1, \text{ то, } \rho(x, y) \le L \cdot \rho(\omega(x), \omega(y)), \qquad (2.37)$$

де ρ метрика, породжена нормою l^∞.

Таким чином, для збіжності ітераційного процесу (2.37) досить контролювати параметр s_i, щоб $s_i < 1 \ \forall i = 1, ..., r$. Проте, необхідно відзначити, що умова $L < 1$ не є необхідною для збіжності процесу відновлення.

Взагалі кажучи, x_ω не завжди буде точно збігатися з образом $\omega(x)$, але, при виконанні деяких умов оператором ω можна гарантувати, що x_ω буде «майже» також близько до x, як і $\omega(x)$.

Відповідну оцінку відстані між x і $\omega(x)$ дає наступна теорема (Collage theorem) [18].

Теорема: $\rho(x, x_\omega) \le \frac{1}{1-L} \rho(x, \omega(x))$, де $L < 1$- константа Ліпшиця стискає оператора ω в деякій метриці ρ.

У пропонованому далі алгоритмі як критерій близькості $\omega(x)$ до вихідного вектору сигналу x використовується евклідова метрика. А саме, потрібно знайти таке відображення ω, що $\rho(\omega(x), x)$ мінімально. Це равносильно завданню: знайти для кожної R-послідовності $R_i, i = 1, ..., r$ таку пару $(\omega_i, D_{j(i)})$, що

$$\sigma_{R_i}(\omega_i, D_{j(i)}) = \sum \left[\omega_i \left(D_{j(i)} \right) - R_i \right]^2 = \sum_\alpha (s_i d_\alpha^i + o_i - r_\alpha^i)^2 \to \min \qquad (2.37)$$

де α = 1, 2, ..., p.

При вирішенні цього завдання для будь-яких варіантів $D_{j(i)}$ і ω_i можна підібрати оптимальні параметри s_i і o_i, використовуючи метод найменших

квадратів для лінійної регресії, мінімізувавши суму

$$\sigma_{R_i} = \sum_{\alpha} (s_i d_\alpha^i + o_i - r_\alpha^i)^2 \qquad (2.38)$$

Функція (2.38) має мінімум в тих точках, в яких приватні похідні від σ_{R_i} по параметрам s_i і o_i звертаються в нуль. В результаті диференціювання і елементарних перетворень для визначення параметрів отримуємо систему двох лінійних рівнянь з двома невідомими s_i і o_i звідки отримуємо, що

$$s_i = \frac{p\sum_{k=1}^{p} d_k \cdot r_k - \sum_{k=1}^{p} d_k \cdot \sum_{k=1}^{p} r_k}{p \cdot \sum_{k=1}^{p} d^2{}_k - \left(\sum_{k=1}^{p} d_k\right)^2}$$

$$o_i = \frac{1}{p} \cdot \left(\sum_{k=1}^{p} r_k - s_i \cdot \sum_{k=1}^{p} d_k\right) \qquad (2.39)$$

Зокрема, якщо $s_i = 0$, то $o_i = \frac{1}{p}\sum_{k=1}^{p} d_k$. Таким чином середньоквадратична помилка перетвореної послідовності від її вихідного стану

$$\sigma_{R_i}^2 = \frac{1}{p} \cdot \left[\sum_{k=1}^{p} r_k{}^2 + s_i \cdot \left(s_i \cdot \sum_{k=1}^{p} d_k{}^2 - 2 \cdot \sum_{k=1}^{p} r_k \cdot d_k + 2 \cdot o_i \cdot \sum_{k=1}^{p} d_k\right) + o_i \cdot \left(o_i \cdot p - 2 \cdot \sum_{k=1}^{p} r_k\right)\right] \qquad (2.40)$$

Завдання пошуку D-послідовності $D_{j(i)}$ і його афінного відображення може бути вирішена різними способами. Наприклад, за допомогою алгоритму Фішера [3], повний перебір і т.д.

2.5.4. Алгоритм стиснення

Алгоритм упаковки коефіцієнтів перетворення в файл зводиться до перебору всіх рангових послідовностей і підбору для кожної відповідної їй доменної послідовності. Нижче наводиться схема цього алгоритму.

Крок 1. Вибір довжини рангової послідовності p.

Крок 2. Розбиття сигналу S на рангові та доменні послідовності

R = BreakToRanges (S);

D = BreakToDomains (S);

Крок 3.

```
For (i = 1; i ≤ num_ranges; i ++)
{
    For (j = 1; j ≤ num_domain; j ++)
    {
        Compute s, o;
        If (<min_error)
        {
            min_error =;
            best_domain = j;
            best_s = s;
            best_o = o;
        }
    }
    Save_coefficient_for_range (best_domain, best_s, best_o);
}
```

На рис. 2.6 наведена структурна схема кодера програми.

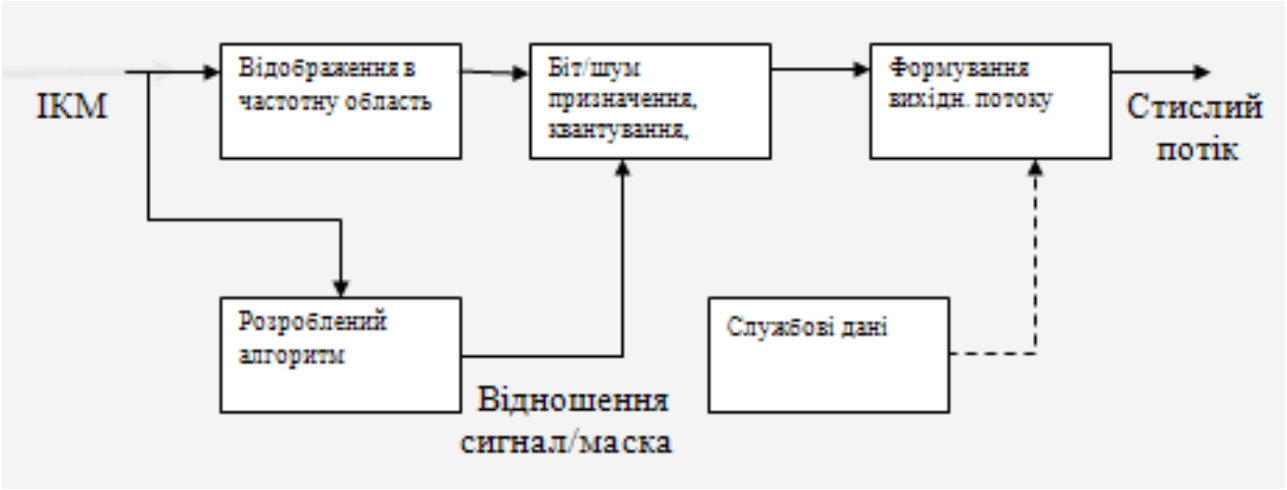

Рис. 2.6. Структурна схема кодера програми

2.5.5. Алгоритм декомпресії

Декомпресія алгоритму фрактального стиснення надзвичайно проста. Необхідно провести кілька ітерацій афінних перетворень, коефіцієнти яких були отримані на етапі компресії.

В якості початкового може бути взятий абсолютно будь-який сигнал (наприклад, нульовий), оскільки відповідний математичний апарат гарантує нам збіжність послідовності сигналів, які надходять в ході ітерацій IFS, до нерухомого сигналу (близького до початкового). Зазвичай для цього достатньо 16 ітерацій.

Прочитаємо з файлу коефіцієнти всіх послідовностей.

Створимо нульовий сигнал S0 потрібного розміру.

Until (сигнал не стане нерухомим)

{

R = BreakToRanges (S0);

D = BreakToDomains (S0);

For (i = 1; i ≤ num_ranges; i ++)

{

For (j = 1; j ≤ size_range; j ++)

{

```
            Rij =;
        }
    }
    S0 = JoinRanges (R);
    }
```

На рис. 2.7 наведена структурна схема декодера програми.

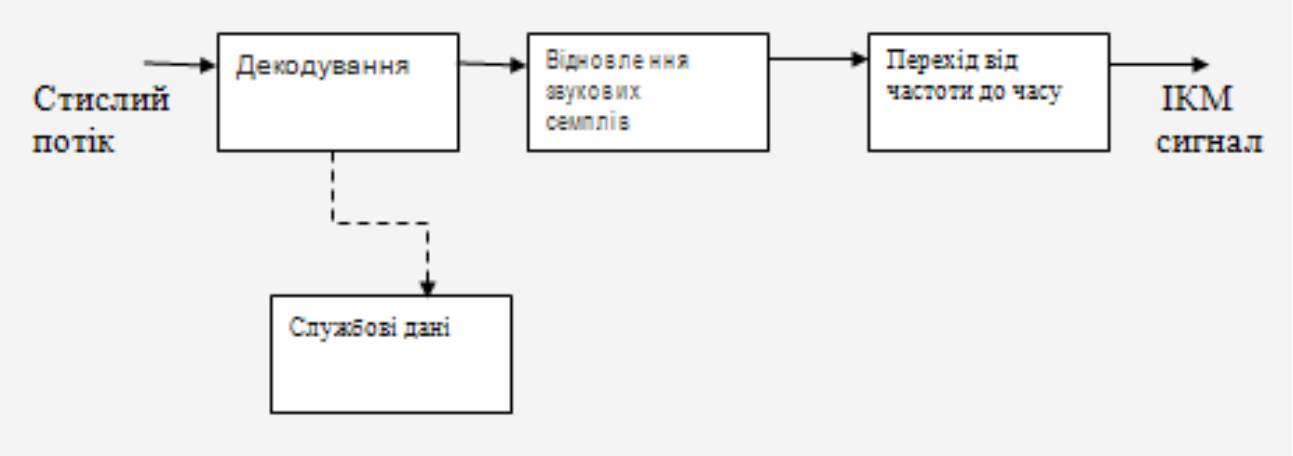

Рис. 2.7. Кодер (а) и декодер (б) MPEG.

2.5.5. Програмна реалізація алгоритмів

Процес компресії/декомпресії звукової інформації в програмі реалізований через класи CFractalCompress і CFractalDecompress відповідно.

```
//Коефіцієнти IFS
struct FractalCoef
{
  unsigned long index:16;
  long scale:8;
};
```

```
// Клас CFractalCompress: Клас для стиснення аудіоданих
class CFractalCompress
{
public:
 //Конструктор
    CFractalCompress(char*FileName,      //ім'я .wav файлу, що потрібно стиснути
                     char* FNameRec,  //ім'я стиснутого файлу
                    long size_block); //розмір рангового блоку
   //Процедура стиснення даних
    void CompressData();
    virtual ~CFractalCompress();
private:
   // ім'я стиснутого файлу
   char* m_FNameCmprs;
   unsigned long m_SamplesPerSec;
   //Довжина вихідних даних
    long m_count_data;
   //Вихідні дані
    long* m_data;
   //Кількість ранговых послідовностей
   long m_count_blocks;
   //Кількість доменних послідовностей
    long m_count_domains;
   //розмір рангового блока
    long m_size_of_block;
   //Рангові послідовності
    long** m_blocks_data;
   //Доменні послідовності
    long** m_domains_data;
```

```
  //Масив коеффіцієнтів IFS
  FractalCoef* m_out;
  //Ініціалізація масивів
  void InitData();
  void InitOutRange();
  //Розбиття вихідного сигналу на рангові послідовності
    bool Break2Blocks(void);
  // Розбиття вихідного сигналу на доменні послідовности
    bool Break2Domains(void);
  //Розрахунок відстані між послідовностями
  float RMS(long index_d,long index_b, float* v);
  //Розрахунок здвигу по значенню рівня відліку сигналу
  float CalculateV(long index_d, long index_b);
  //Знахождення найбільш відповідних коефіцієнтів, що переводять
  //i_domain доменну послідовність в i_block рангову
  void BestCoeffForDomain(long i_domain,long ,float* rms,float* best_v);
  //Запис отриманих коефіцієнтів в файл
  bool WriteCoefsToBinFile(char* FileName);
};

//Класс CFractalDecompress: Клас для декомпресіїи аудіоданих
class CFractalDecompress
{
public:
  //Конструктор
    CFractalDecompress(char*FileNameCoefs,//ім'я файлу для декомпресії
                  char* FNameRec);   //ім'я вихідного .wav файлу
    virtual ~CFractalDecompress();
  //Процедура декомпресії даних
  void DecompressData();
```

```
private:
  //ім'я файлу для декомпресії
  char* m_FNameRec;
  unsigned long m_SamplesPerSec;
  //Довжина вихідних данни
  long m_count_data;
  //Кількість ранговых послідовностей
  long m_count_blocks;
  //Кількість доменних послідовностей
  long m_count_domains;
//розмір рангового блока
    long m_size_of_block;
  //Відновлений  сигнал
  float* m_out_data;
  //Рангові послідовності
  float** m_blocks_data;
  //Доменні послідовності
    float** m_domains_data;
     //Масив коеффіцієнтів IFS
  FractalCoefs* m_in;
  //Ініціалізація масивів
  void InitData();
  void InitInputRange();
  //Розбиття вихідного сигналу на рангові послідовності
  bool Break2Blocks(void);
  // Розбиття вихідного сигналу на доменні послідовности
    bool Break2Domains(void);
  //Збір ранговых послідовностей в один масив послідованості
  bool JoinBlocks(void);
  // Відновлення index-ой рангової послідовності
```

```
    void ReconstructBlock(long index);
    //Видалення нулів в кінці сигналу
    void DeleteZeroEnd();
    //Запис .wav файлу
    bool WriteToFile(char* FileName);
    //Зчитування коефіцієнтів з файлу
    bool ReadCoefsFromBinFile(char* FileName);
};
```

Також використовується допоміжний клас CWav для читання файлів формату .wav:

```
//Клас CWav:  Клас для читання даних з .wav файлу
class CWav
{
private:
   // дані в беззнакову вигляді
   unsigned long * m_data;
   // дані в знаковому вигляді
   long * m_out_data;
   // Довжина послідовності
   unsigned long m_count;
   unsigned long m_SamplesPerSec;
   char * m_FName;
   // Ініціалізація масивів
   void InitData ();
public:
   CWav (char * FileName);
   ~ CWav ();
   // Читання .wav файлу
   bool ReadFile ();
```

```
    // Отримання рівнів відліків сигналу
    long * GetData ();
    // Отримання довжини даних
    unsigned long GetCountData ();
    unsigned long GetSamplesPerSec ();
};
```

2.6. Обґрунтування та організація вхідних та вихідних даних програми

Програма працює з 8-ми бітними одноканальними аудіоданими формату wav (Microsoft Waveform Audio).

Константні значення для заголовка .wav файлу:

- count_iteration (ітерація підрахунку) = 16;
- fmHeadSize (розмір)= 16;
- wFormatTag (тег формату) = 1;
- Channels (канали)= 1;
- nBlockAlign (вирівнювання блоку)= 1;
- nBitsPerSample (біти за зразок)= 8.

2.7. Опис роботи розробленого програмного продукту

2.7.1. Використані технічні засоби

Для розробки програмного забезпечення використовувався комп’ютер з наступними параметрами:

- операційна система Windows 8;
- процесор Dual Core Intel з частотою 2.8 GHz;
- оперативна пам'ять 4GB ОЗУ;
- відеокарта nVidia GeForce 8600/9600GT;
- Radeon HD2600/3600;

– DirectX версії 9.0c;
– жорсткий диск 16 GB.

2.7.2. Використані програмні засоби

Описаний метод фрактального стиснення звукових сигналів був запрограмований на мові C ++ із застосуванням об'єктно-орієнтованого підходу. Програма працює з 8-ми бітними одноканальними аудіоданими формату wav (Microsoft Waveform Audio).

2.7.3. Виклик та завантаження програми

Для установки програмного забезпечення на свій ПК, потрібно папку Fractal скопіювати на його жорсткий диск і відкрити файл Fractal.exe.

2.7.4. Опис інтерфейсу користувача

Розроблена програма забезпечує роботу використаних алгоритмів та зберігає стиснені та декомпресовані дані в окремих файлах для їх подальшої перевірки та дослідження. При цьому інтерфейс додатку досить простий та інтуїтивно зрозумілий і не містить зайвих функцій та забезпечує лише функцію компресії (стиснення) обраного файлу або декомпресії обраного файлу.

Програма використовує консольний додаток, який завантажується при відкритті файлу Fractal.exe (рис. 2.8).

Рис. 2.8. Завантажений додаток

Користувач вручну вводить ім’я файлу з розширенням .wav та шлях до його розташування. Система починає процес його компресії за відповідним розробленим фрактальним алгоритмом (рис. 2.9).

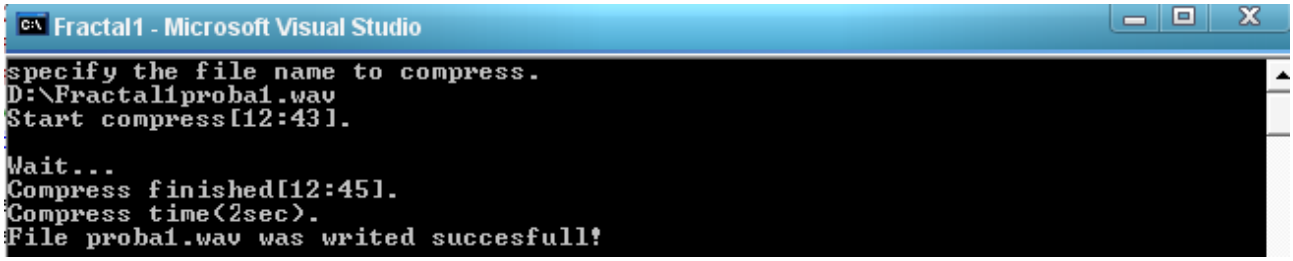

Рис. 2.9. Результат роботи програми в режимі компресії

Для декомпресії отриманого файлу, необхідно у відповідне поле (рис. 2.10) ввести ім’я файлу з розширенням .sfc та шлях до його розташування. Система починає процес його декомпресії за відповідним алгоритмом (рис. 2.11):

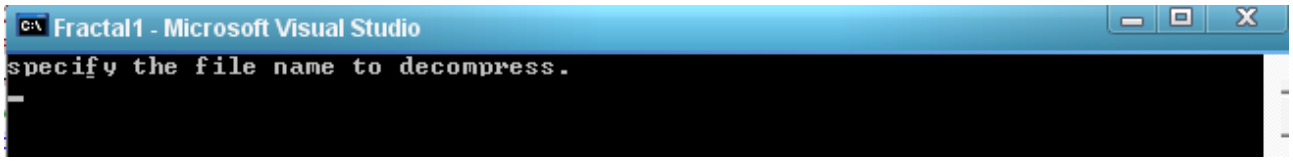

Рис. 2.10. Виконання декомпресії файлу

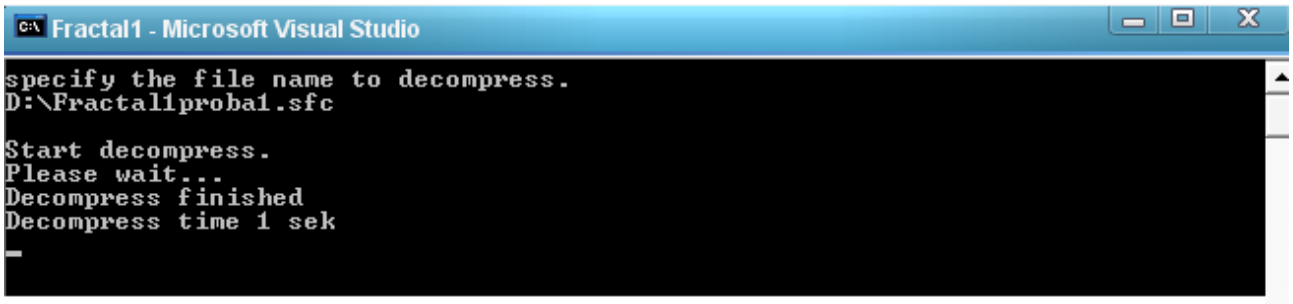

Рис. 2.11. Результат роботи декомпресії файлу

2.7.5. Дослідження роботи розроблених алгоритмів та їх програмної реалізації

Для оцінки відхилення закодованого 8-бітного одноканального звукового сигналу від вихідного використовувалося співвідношення

$$G(y,x)=\frac{\sqrt{\frac{1}{N}\cdot\sum_{i=1}^{N}(y_i-x_i)^2}}{\sqrt{N\cdot\sum_{i=1}^{N}(x_i)^2}}\,. \qquad (2.41)$$

Алгоритм оперує з послідовностями однакового розміру. Розмір послідовностей фіксований від початку роботи алгоритму до кінця. Також, з метою скорочення числа зберігаються в файл коефіцієнти для рангової послідовності, був зафіксований масштабний коефіцієнт s зі значенням 0,75 (аналогічно алгоритму стиснення зображень [6]).

Перевага даного алгоритму полягає в тому, що при відповідному виборі розмірів оброблюваних послідовностей забезпечується рівномірна якість кодування всього сигналу. Недоліком алгоритму є малий коефіцієнт стиснення і великий час компресії.

Так, в наступній діаграмі показана залежність часу компресії від обсягу кодованого аудіо файлу.

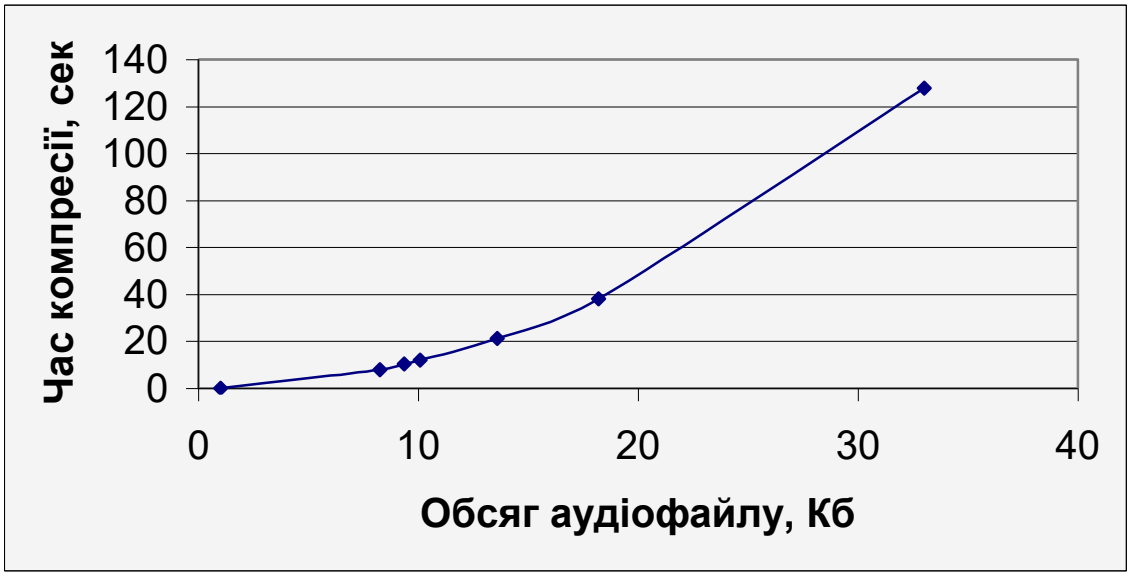

Рис. 2.12. Діаграма залежності часу компресії від обсягу кодованого аудіо файлу

Залежність часу декомпресії від розміру аудіо файлу має наступний вигляд (рис. 2.13):

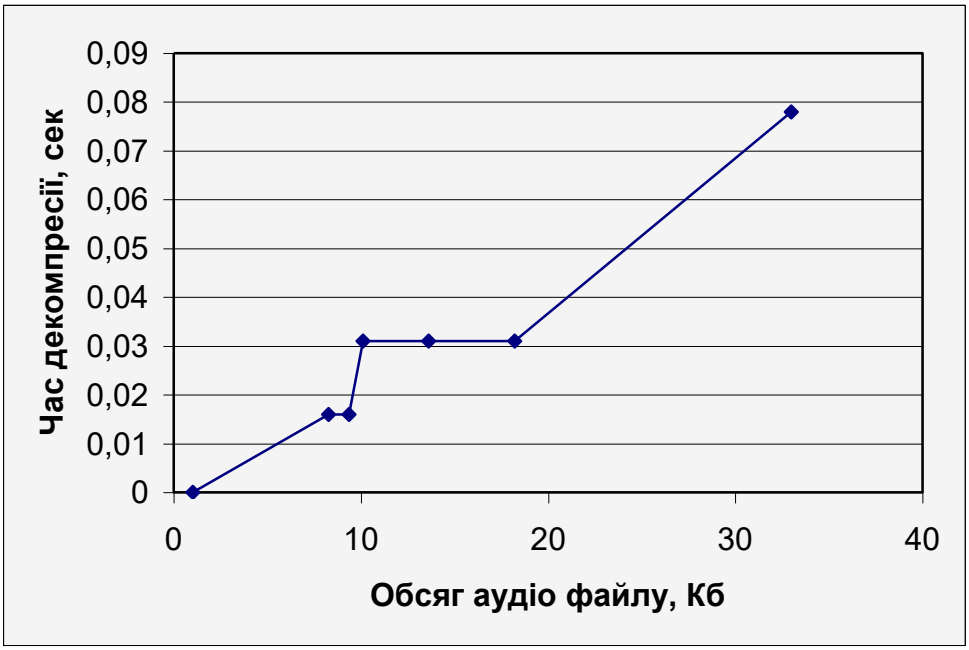

Рис. 2.13. Залежність часу декомпресії від розміру аудіо файлу

Залежність часу компресії аудіо файлу обсягом 13,7 КБ від розміру рангової послідовності має наступний вигляд (рис. 2.14):

Рис. 2.14. Залежність часу компресії аудіо файлу обсягом 13,7 КБ від розміру рангової послідовності

Для складного вихідного сигналу, що представляє собою суму трьох сигналів (прямокутного, трикутного і синусоїдального), ітераційний процес збіжності до нерухомого сигналу в процесі декомпресії з початковим синусоїдальним сигналом (рис. 2.14) представлений на рис. 2.15, де y_k це початковий сигнал після k-й ітерації.

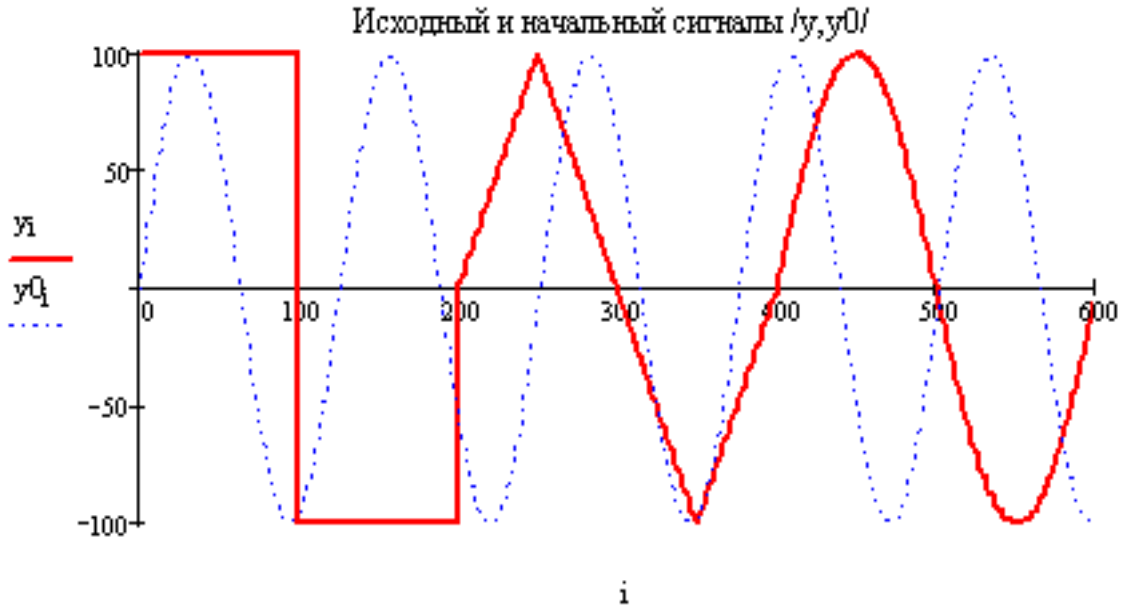

Рис. 2.14. Початковий сигнал

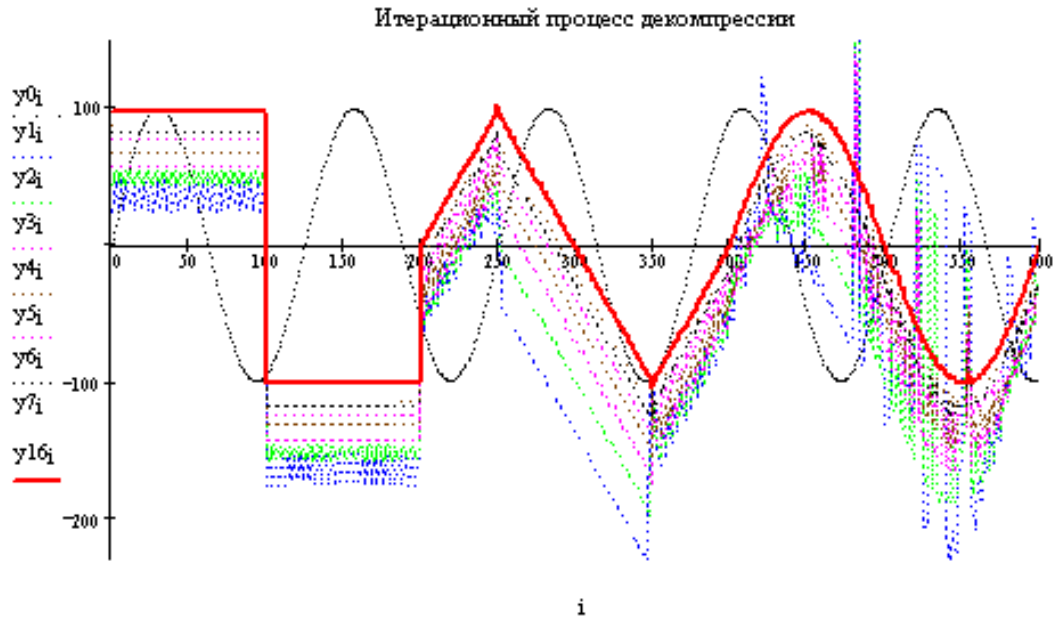

Рис. 2.15. Декомпресія початкового сигналу (рис. 2.14)

Робота програми з реальними звуковими даними представлена нижче.

Для досліджування сигналів на різних стадіях були взяті два наступних аудіо сигнали:

1. Перший сигнал: 8 біт, моно, з частотою дискретизації 22 222 Гц, 14 027 відліків. Характеризується відносно малою різницею значень між двома послідовними відліками. Склад сигналу: стандартна мелодія з Windows (рис. 2.16).

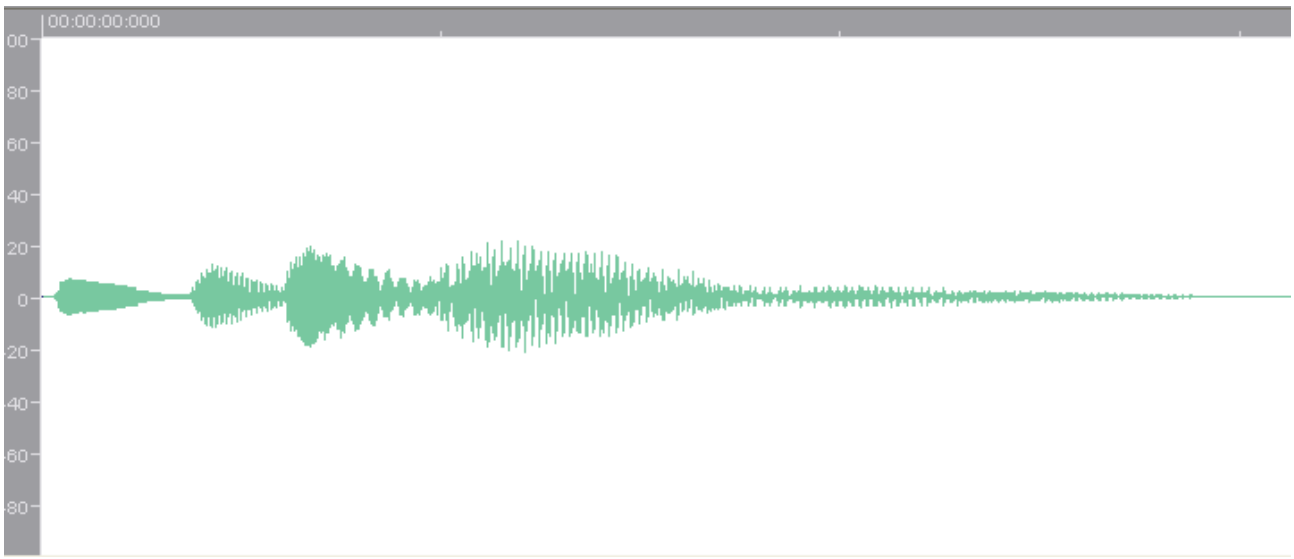

Рис. 2.16. Вхідний сигнал 1

На рис. 2.17 зображено відновлений сигнал після його компресії та декомпресії при розмірі рангової послідовності p = 4. Коефіцієнт стиснення: 1.33. Середньоквадратичне відхилення (СКВ): 0.188.

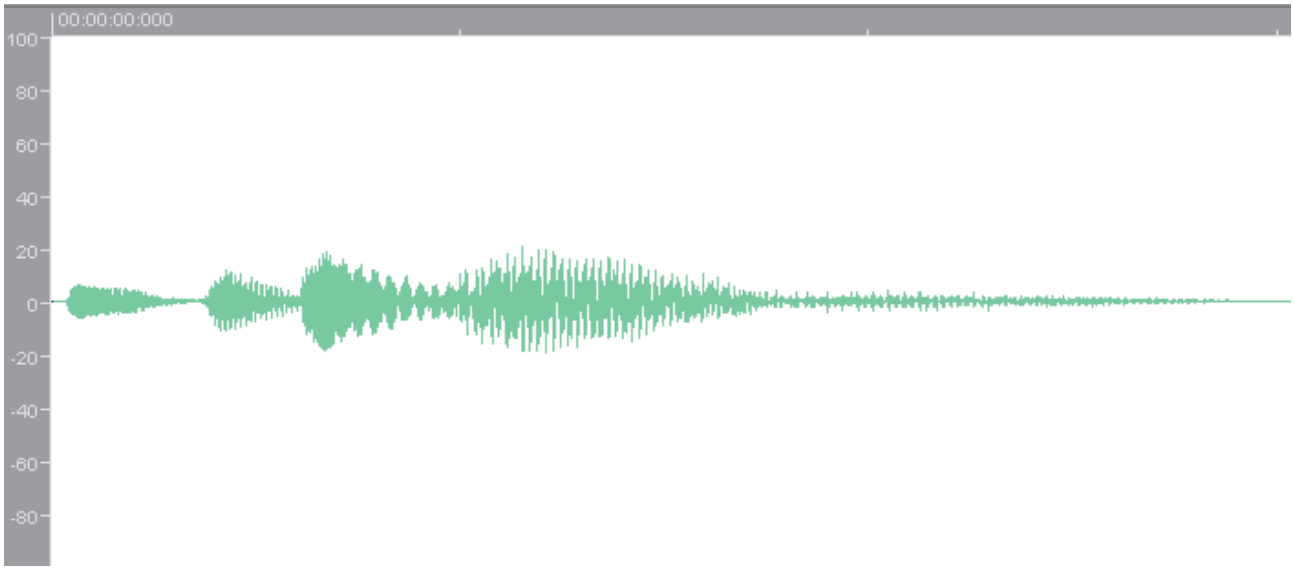

Рис. 2.17. Відновлений сигнал при розмірі рангової послідовності p = 4

На рис. 2.18 зображено відновлений сигнал після його компресії та декомпресії при розмірі рангової послідовності p = 6. Коефіцієнт стиснення: 2, СКВ: 0.238.

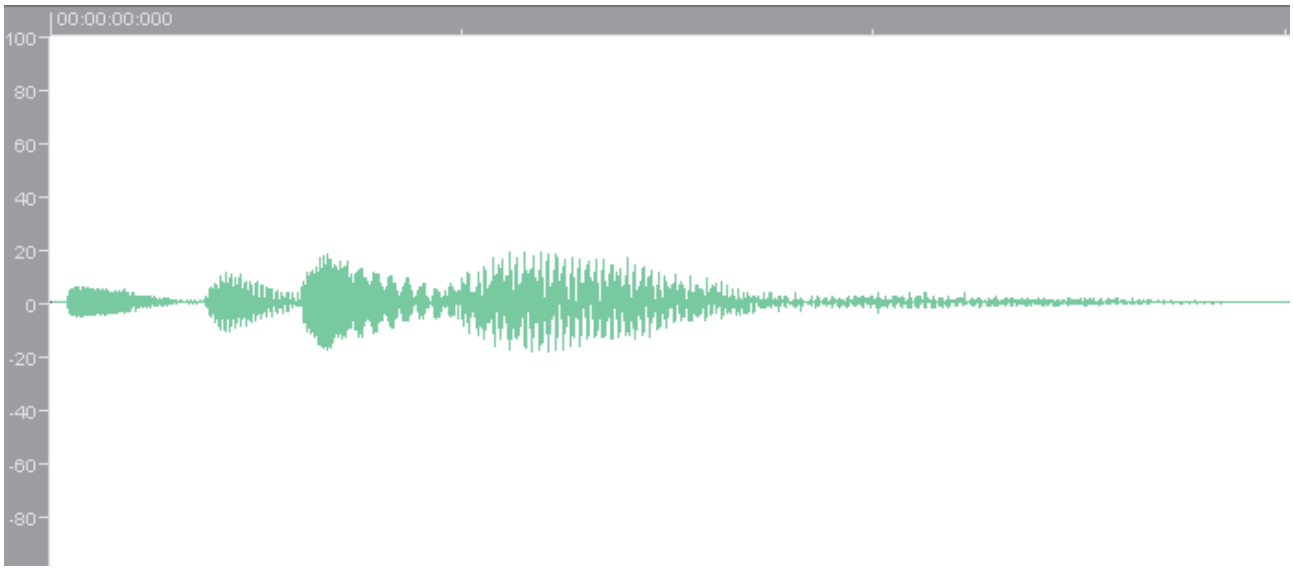

Рис. 2.18. Відновлений сигнал при розмірі рангової послідовності p = 6

На рис. 2.19 зображено відновлений сигнал після його компресії та декомпресії при розмірі рангової послідовності p = 8. Коефіцієнт стиснення: 2.7. СКВ: 0.319.

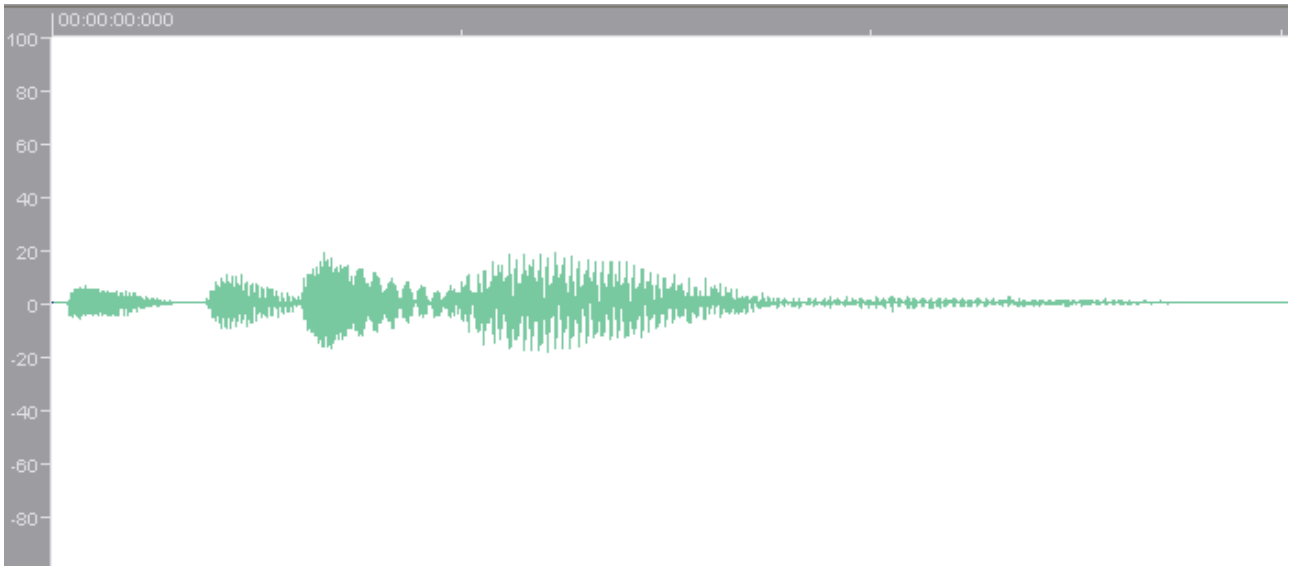

Рис. 2.19. Відновлений сигнал при розмірі рангової послідовності p = 8

На рис. 2.20 зображено відновлений сигнал після його компресії та декомпресії при p = 10. Коефіцієнт стиснення: 3.3. СКВ: 0.43.

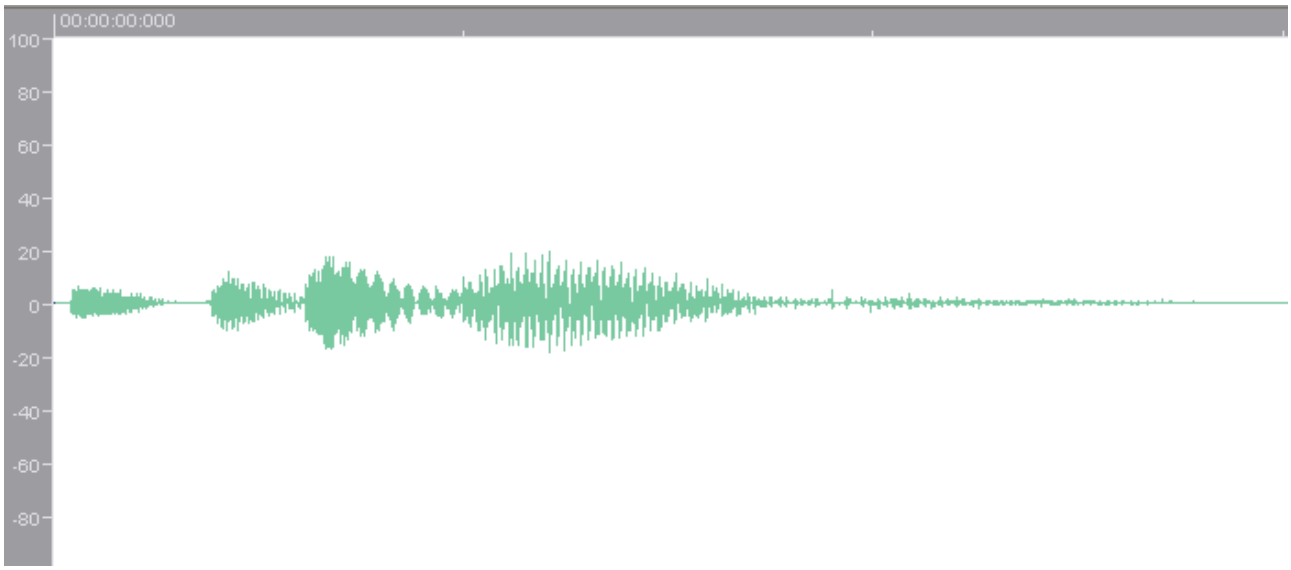

Рис. 2.20. Відновлений сигнал при розмірі рангової послідовності p = 10

2. Другий сигнал: 8 біт, моно, з частотою дискретизації 8 192 Гц, 14 000 відліків. Характеризується відносно великою різницею значень між двома послідовними відліками. Склад сигналу: хорова пісня (рис. 2.21) .

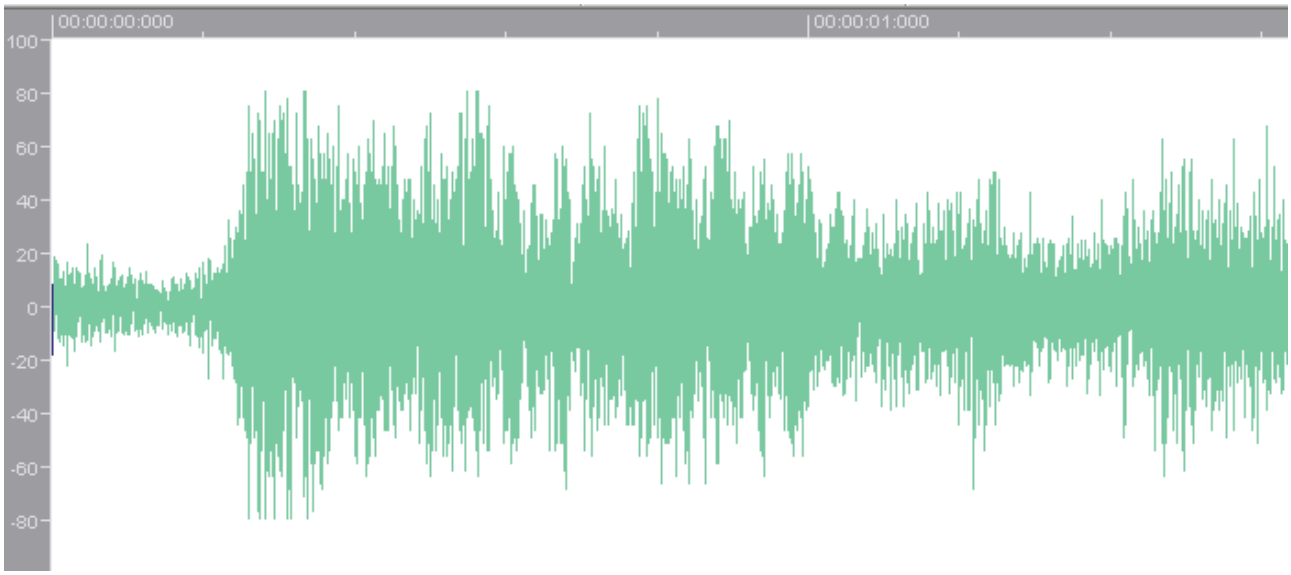

Рис. 2.21. Рис. 2.16. Вхідний сигнал 2

На рис. 2.22 зображено відновлений сигнал після його компресії та декомпресії при розмірі рангової послідовності p = 4. Коефіцієнт стиснення: 1.34. Середньоквадратичне відхилення (СКВ): 0.114.

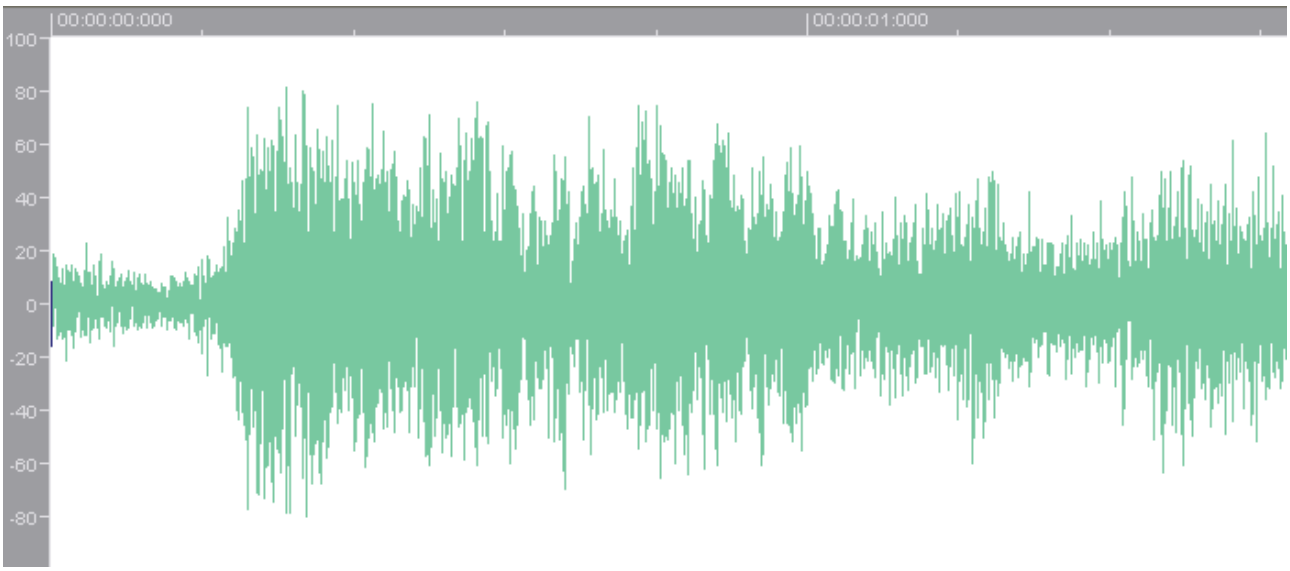

Рис. 2.22. Відновлений сигнал при розмірі рангової послідовності p = 4

На рис. 2.23 зображено відновлений сигнал після його компресії та декомпресії при розмірі рангової послідовності p = 6. Коефіцієнт стиснення: 2. СКВ: 0.297.

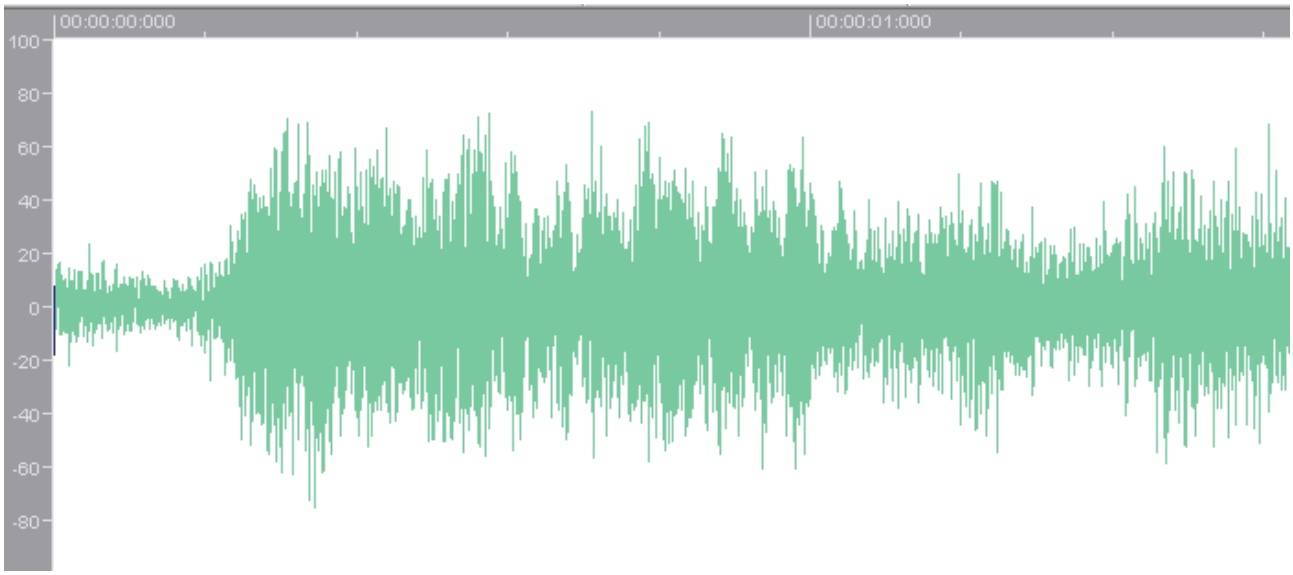

Рис. 2.23. Відновлений сигнал при розмірі рангової послідовності p = 6

На рис. 2.24 зображено відновлений сигнал після його компресії та декомпресії при розмірі рангової послідовності p = 8. Коефіцієнт стиснення: 2.7. СКВ: 0.551.

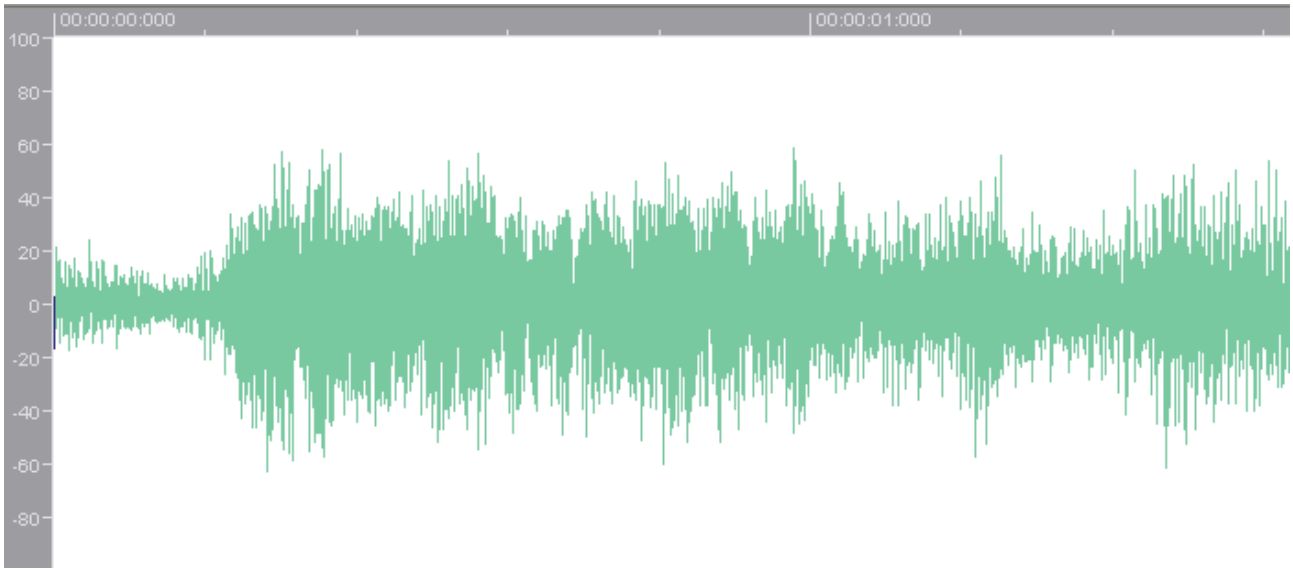

Рис. 2.24. Відновлений сигнал при розмірі рангової послідовності p = 8

На рис. 2.24 зображено відновлений сигнал після його компресії та декомпресії при розмірі рангової послідовності p = 10. Коефіцієнт стиснення: 3.33. СКВО: 0.863.

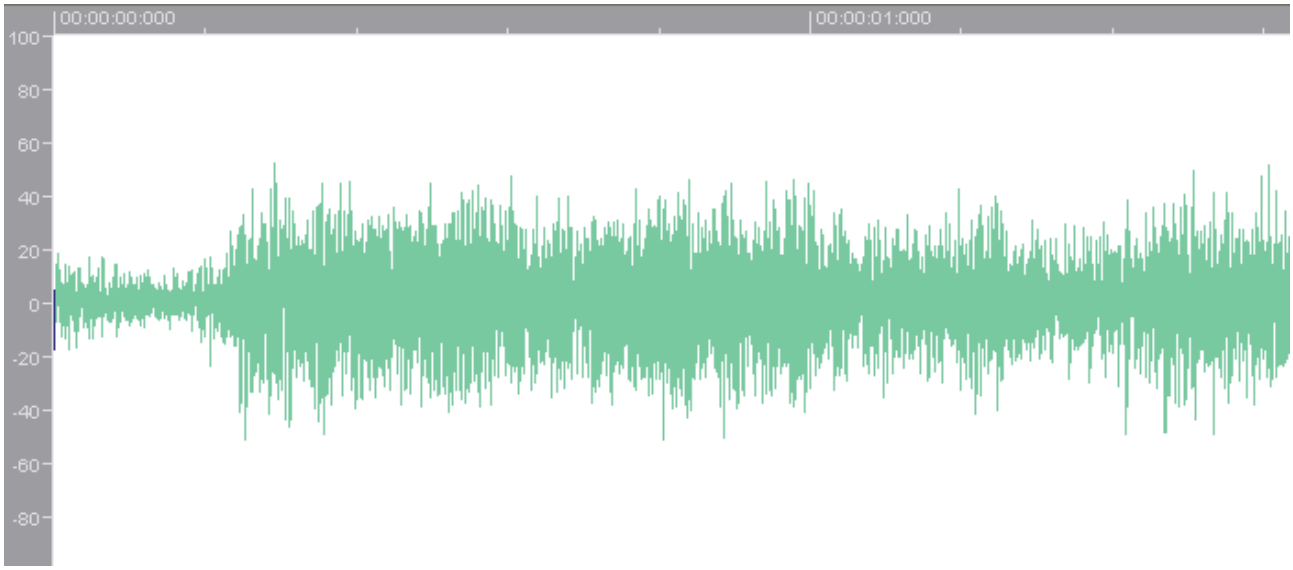

Рис. 2.24. Відновлений сигнал при розмірі рангової послідовності p = 10

Таким чином, з рис. 2.12 – 2.14, 2.16 – 2.24 слід, що зі збільшенням розміру доменного (рангового) блоку збільшується коефіцієнт стиснення, знижується якість відновленого сигналу і зменшується час його стиснення. Крім того, візуальний аналіз осцилограм рис. 2.16 – 2.24 і відповідних їм показників спотворення сигналу в циклі стиснення - декомпресія (СКВ), свідчать про задовільну якість відновлених сигналів при вибраних параметрах стиснення.

РОЗДІЛ 3
ЕКОНОМІЧНИЙ РОЗДІЛ

3.1. Розрахунок трудомісткості та вартості розробки програмного продукту

Вхідні дані:

- передбачуване число операторів - 1500;
- коефіцієнт складності програми - 2;
- коефіцієнт корекції програми в ході її розробки - 0,08;
- годинна заробітна плата програміста, грн / год - 50.

Нормування праці в процесі створення ПЗ істотно ускладнено в силу творчого характеру праці програміста. Тому трудомісткість розробки ПЗ може бути розрахована на основі системи моделей з різною точністю оцінки.

Трудомісткість розробки ПЗ можна розрахувати за формулою:

$$t = t_u + t_a + t_n + t_{отл} + t_д, \text{ людино-годин,} \quad (3.1)$$

де to - витрати праці на підготовку й опис поставленої задачі (приймається 50),

ti - витрати праці на дослідження алгоритму рішення задачі,

ta - витрати праці на розробку блок-схеми алгоритму,

tп - витрати праці на програмування по готовій блок-схемі,

tотл - витрати праці на налагодження програми на ЕОМ,

tд - витрати праці на підготовку документації.

Складові витрати праці визначаються через умовне число операторів у ПЗ, яке розробляється.

Умовне число операторів (підпрограм):

$$Q = q \times C \times (1+ p) \text{ , людино-годин,} \qquad (3.2)$$

де q - передбачуване число операторів,

С - коефіцієнт складності програми,

р - коефіцієнт кореляції програми в ході її розробки.

$$Q = 1500 \cdot 2 \cdot (1 + 0{,}08) = 3240 \text{ людино-годин.}$$

Витрати праці на вивчення опису задачі ti визначається з урахуванням уточнення опису і кваліфікації програміста:

$$t_u = \frac{QB}{(75...85)K} \text{ , людино-годин,} \qquad (3.3)$$

де В - коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі; В=1.2 ... 1.5,

k - коефіцієнт кваліфікації програміста, обумовлений від стажу роботи з даної спеціальності.

Витрати праці на розробку алгоритму рішення задачі:

$$t_u = \frac{3240 \cdot 1{,}3}{80 \cdot 1{,}2} = 44 \text{ , людино-годин.}$$

Витрати на складання програми по готовій блок-схемі:

$$t_a = \frac{Q}{(20...25)K} \quad \text{людино-годин.} \qquad (3.4)$$

$$t_a = \frac{3240}{22 \cdot 1{,}2} = 123 \text{ людино-годин.}$$

Витрати на складання програми по готовій блок-схемі:

$$t_n = \frac{Q}{(20...25)K} \quad \text{людино-годин.} \tag{3.5}$$

$$t_n = \frac{3240}{23 \cdot 1{,}2} = 117 \text{ людино-годин.}$$

Витрати праці на налагодження програми на ЕОМ:

$$t_{отл} = \frac{Q}{(4...5)K} \quad \text{людино-годин.} \tag{3.6}$$

$$t_{отл} = \frac{3240}{5 \cdot 1{,}2} = 540 \text{ людино-годин.}$$

Витрати праці на підготовку документації:

$$t_д = t_{др} + t_{до}, \text{ людино-годин,} \tag{3.7}$$

де tдр - трудомісткість підготовки матеріалів і рукопису.

$$t_{др} = \frac{Q}{(15...20)K}, \text{ людино-годин.} \tag{3.8}$$

$$t_{др} = \frac{3240}{18 \cdot 1{,}2} = 150 \text{ людино-годин,}$$

tдо - трудомісткість редагування, печатки й оформлення документації

$$t_{до} = 0{,}75 \cdot t_{др}, \text{ людино-годин.} \tag{3.9}$$

$$t_{до} = 150 + 73 = 223 \text{ людино-годин.}$$

Отримуємо трудомісткість розробки програмного забезпечення:

$$t = 50 + 44 + 123 + 117 + 540 + 223 = 1097 \text{ людино-годин.}$$

3.2. Розрахунок витрат на створення програми

Витрати на створення ПЗ Кпо включають витрати на заробітну плату виконавця програми Зз/п і витрат машинного часу, необхідного на налагодження програми на ЕОМ.

$$K_{ПО} = З_{ЗП} + З_{МВ}, \text{ грн,} \tag{3.10}$$

Заробітна плата виконавців визначається за формулою:

$$З_{ЗП} = t \cdot C_{ПР}, \text{ грн,} \tag{3.11}$$

де t - загальна трудомісткість, людино-годин,

Спр - середня годинна заробітна плата програміста, грн/година.

$$Ç_{ÇÏ} = 1097 \cdot 50 = 54850 \text{ ãðí.}$$

Вартість машинного часу, необхідного для налагодження програми на ЕОМ:

$$З_{МВ} = t_{отл} \times C_{М}, \text{ грн,} \tag{3.12}$$

де tотл - трудомісткість налагодження програми на ЕОМ, год,

Смч - вартість машино-години ЕОМ, грн/год.

Визначені в такий спосіб витрати на створення програмного забезпечення є частиною одноразових капітальних витрат на створення АСУП.

$$З_{МВ} = 540 \times 5 = 2700 \text{ грн.}$$

$$Êïï = 2700 + 54850 = 57550 \text{ ãðí.}$$

Очікуваний період створення ПЗ:

$$T = \frac{t}{B_k \cdot F_p} \text{ міс.} \qquad (3.13)$$

де Bk - число виконавців,

Fp - місячний фонд робочого часу (при 40 годинному робочому тижні Fp=176 годин).

$$T = \frac{1097}{1 \cdot 176} = 6.2 \text{ міс.}$$

Висновки: визначено трудомісткість розробленого додатку (1097 люд-год), проведений підрахунок вартості роботи по створенню програми (57550 грн.) та розраховано час на його створення (6,2 міс).

ВИСНОВКИ

Метою даної кваліфікаційної роботи є аналіз можливості фрактального стиснення аудіо даних, розробка відповідного алгоритму і його програмної реалізації.

Під час виконання роботи викладається опис існуючих методів стиснення звукових даних, основні поняття, описується метод фрактального стиснення зображень і математичне обґрунтування його використання для стиснення аудіо даних, виконано програмну реалізацію даного алгоритму.

В результаті виконання даної роботи розроблена програма, що реалізує фрактальний алгоритм стиску аудіо даних.

В даній роботі:

1. Показано, що використання фрактального методу стиснення є можливим і перспективним способом компресії звукової інформації.

2. Розроблено алгоритм фрактального стиснення аудіо даних та його програмна реалізація.

3. Досліджено залежність ефективності запропонованого алгоритму (його швидкодії, ступеня стиснення і якості сигналу) від параметрів алгоритму і компресованого цифрового звуку.

Описаний метод фрактального стиснення звукових сигналів був запрограмований на мові C ++ із застосуванням об'єктно-орієнтованого підходу. Програма працює з 8-ми бітними одноканальними аудіоданими формату wav (Microsoft Waveform Audio).

В економічному розділі визначено трудомісткість розробленого додатку (1097 люд-год), проведений підрахунок вартості роботи по створенню програми (57550 грн.) та розраховано час на його створення (6,2 міс).

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Бусигін Б.С., Коротенко Г.М., Коротенко Л.М. Прикладна інформатика. Підручник для студентів комп'ютерних спеціальностей. – Дніпропетровськ: Видавництво НГУ, 2004. – 559 с. URL: http://www.programmer.dp.ua/book-ua-k01.php. дата звернення: 15.03.2021.
2. Бусыгин Б.С., Дивизинюк М.М., Коротенко Г.М., Коротенко Л.М. Введение в современную информатику. Учебник. – Севастополь: Издательство СНУЯЭиП, 2005. – 644 с. / URL: http://www.programmer.dp.ua/book-ru-k02.php. дата звернення: 15.01.2021.
3. Гуц А.К., «Комплексный анализ и информатика», ОмГУ, Омск 2002.
4. Глобальная сеть рефератов – URL: https://otherreferats.allbest.ru/radio/ 00312640_0.html/ . Дата звернення 26.02.20210
5. Д.Ватолин, А.Ратушняк «Методы сжатия данных»: **UML:** http://compression.graphicon.ru. дата звернення: 15.01.2021.
6. Документація. Звіти у сфері науки і техніки. Структура і правила оформлення : ДСТУ 3008-95. – Чинний від 1996–01–01. – К. : Держстандарт України, 1996. – 39 с.
7. ДСТУ 2394-94 Інформація та документація. Комплектування фонду, бібліографічний опис, аналіз документів. Терміни та визначення. – Чинний від 01.01.1995. - Київ : Держстандарт України, 1994. – 88 с.
8. Майданюк В. П. Методи і засоби комп’ютерних інформаційних технологій. Кодування зображень. Навчальний посібник. – Вінниця: ВДТУ, 2001. – 63 с.
9. Методичні вказівки з виконання економічного розділу в дипломних проектах студентів спеціальності “Комп’ютерні системи” / Укладачі О.Г. Вагонова, Нікітіна О.Б. Н.Н. Романюк – Дніпропетровськ: Національний гірничий університет. – 2013. – 23с.
10. Офіційний сайт середовища розробки Visual Studio Code URL:

https://code.visualstudio.com/docs дата звернення: 15.01.2021.

11. Снопок А.Б. «Фракталы», Киевский Естественно-Научный лицей №145, Киев 2003.

12. Скорочення слів в українській мові у бібліографічному описі. Загальні правила та вимоги : ДСТУ 3582-97. – Чинний від 1998–07–01. – К. : Держстандарт України, 1998. – 24 с. – (Державний стандарт України).

13. Усенко И.В., «Применение генетического алгоритма фрактального сжатия изображений», Омский государственный университет, 1999.

14. A.E. Jacquin, «Fractal image coding based on a theory of iterated contractive image transformations» in Proceedings of SPIE Visual Communications and Image Processing ’90, vol. 1360, pp. 227-239 1990.

15. IDE MS Visual Studio URL: https://studfile.net/preview/5994722/page:7/ дата звернення: 5.05.2021.

16. G.E. Qien, Z. Baharav, S. Lepsqy, E. Karnin. A new improved collage theorem with appplications to multiresolution fractal image coding. In Proc. ICASSP, 1994.

17. Mark Chapman and George I. Davida. 1997. Hiding thehidden: A software system for concealing ciphertextas innocuous text. InProceedings of the First International Conference on Information and CommunicationSecurity, volume 1334, pages 335–345, Beijing.

18. M.F. Barnsley, «Fractals Everywhere». London: Academic Press Inc., 1988.

19. Fisher Y. «Fractal Image Compression: Theory and Applications», Springer-Verlag, New York, 1994.

ДОДАТОК А

КОД ПРОГРАМИ

Файл «FractalCompress.h»

```
#ifndef FRACTALCOMPRESS_H
#define FRACTALCOMPRESS_H

#include "Common.h"
#include "wav.h"

//Коэффициенты IFS
struct FractalCoef
{
  unsigned long index:16;
  long scale:8;
};
//Класс для сжатия аудиоданных
class CFractalCompress
{
public:
  //Конструктор
        CFractalCompress(char*FileName,   //имя .wav файла, подлежащего сжатию
              char* FNameRec,  //имя сжатого файла
             long size_block); //размер рангового блока
  //Поцедура сжатия данных
        void CompressData();
        virtual ~CFractalCompress();
private:
  //имя сжатого файла
  char* m_FNameCmprs;
  unsigned long m_SamplesPerSec;
  //Длина исходных данных
        long m_count_data;
```

```
  //Исходные данные
        long* m_data;
  //Количество ранговых последовательностей
  long m_count_blocks;
  //Количество доменных последовательностей
        long m_count_domains;
  //размер рангового блока
        long m_size_of_block;
  //Ранговые последовательности
        long** m_blocks_data;
  //Доменные последовательности
        long** m_domains_data;
  //Массив коэффициентов IFS
  FractalCoef* m_out;
  //Инициализация массивов
  void InitData();
  void InitOutRange();
  //Разбиение исходного сигнала на ранговые последовательности
        bool Break2Blocks(void);
  //Разбиение исходного сигнала на доменные последовательности
        bool Break2Domains(void);
  //Расчет расстояния между последовательностями
  float RMS(long index_d,long index_b, float* v);
  //Расчет сдвига по значению уровня отсчета сигнала
  float CalculateV(long index_d, long index_b);
  //Нахождение наиболее подходящих коэффициентов, переводящих
  //i_domain доменную последовательность в i_block ранговую.
  void BestCoeffForDomain(long i_domain,long ,float* rms,float* best_v);
  //Запись полученных коэффициентов в файл
  bool WriteCoefsToBinFile(char* FileName);
};
#endif // #ifndef FRACTALCOMPRESS_H
**********************************************************************
```

Файл «FractalCompress.cpp»

```
#include "FractalCompress.h"

CFractalCompress::CFractalCompress(char*FileName, char* FNameRec,long size_block)
{
  CWav wav(FileName);
  m_FNameCmprs = FNameRec;
  if(wav.ReadFile())
  {
    m_count_data = wav.GetCountData();
    m_data = wav.GetData();
    m_SamplesPerSec = wav.GetSamplesPerSec();
    m_size_of_block = size_block;
    m_count_blocks = long(m_count_data/m_size_of_block);
          m_count_domains = long((m_count_data-m_size_of_block*2)/2);
    InitData();
    InitOutRange();
  }
  else
  {
    printf("Can't create CFractalCompress!!!\n");
  }
}

CFractalCompress::~CFractalCompress()
{
        delete [] m_data;
  for(long i = 0; i < m_count_blocks; i++)
  {
    delete [] m_blocks_data[i];
  }
  delete [] m_blocks_data;
  for(i = 0; i < m_count_domains; i++)
  {
    delete [] m_domains_data[i];
  }
```

```
  delete [] m_domains_data;
  delete [] m_out;
}
/****************************************
*Description: Initialize m_blocks_data
*            and m_domains_data.
****************************************/
void CFractalCompress::InitData()
{
  //blocks
  m_blocks_data = new long*[m_count_blocks];
        for(long i = 0; i < m_count_blocks; i++)
        {
                m_blocks_data[i] = new long[m_size_of_block];
        }
  for( i = 0; i < m_count_blocks; i++)
        {
                for(long j = 0; j < m_size_of_block; j++)
                {
                        m_blocks_data[i][j] = 0;
                }
        }
  //domains
  m_domains_data = new long*[m_count_domains];
        for(i = 0; i < m_count_domains; i++)
        {
                m_domains_data[i] = new long[m_size_of_block];
        }
  for( i = 0; i < m_count_domains; i++)
        {
                for(long j = 0; j < m_size_of_block; j++)
                {
                        m_domains_data[i][j] = 0;
                }
        }
```

```
}
/*******************************************
*Description: Initialize m_out_index,
*            m_out_transform and m_out_scale.
*******************************************/
void CFractalCompress::InitOutRange()
{
  m_out = new FractalCoef[m_count_blocks];
  for(long j = 0; j < m_count_blocks; j++)
        {
    m_out[j].index = 0;
    m_out[j].scale = 0;
        }
}
/*****************************************
*Description: Break sound data to blocks
*            with size m_size_of_block.
*****************************************/
bool CFractalCompress::Break2Blocks(void)
{
        bool result = true;
        try
        {
                long k = 0;
                for(long i = 0; i < m_count_data-m_size_of_block+1; i = i +
m_size_of_block, k++)
                {
                        for(long j = 0; j < m_size_of_block; j++)
                        {
                                m_blocks_data[k][j] = m_data[i+j];
                        }
                }
        }
        catch(...)
        {
```

```
                result = false;
        }
        return result;
}
/****************************************
*Description: Break sound data to domains
*           with size 2*m_size_of_block.
****************************************/
bool CFractalCompress::Break2Domains(void)
{
        bool result = true;
        try
        {
                long k = 0;
    float sum = 0;
                for(long i = 0; i < 2*m_count_domains; i+=2, k++)
                {
                        for(long j = 0; j < 2*m_size_of_block; j+=2)
                        {
                                m_domains_data[k][j/2] = m_data[i+j];
                        }
                }
        }
        catch(...)
        {
                result = false;
        }
        return result;
}
/****************************************
*Description: Compress data with
*           fractal method.
****************************************/
void CFractalCompress::CompressData()
{
```

```
    bool error = false;
    if(Break2Blocks() == false)
    {
            std::cout<<"Can not break data to blocks!\n";
            error = true;
    }
    if(Break2Domains() == false)
    {
            std::cout<<"Can not break data to domains!\n";
            error = true;
    }
    if(error != true)
    {
clock_t start, finish;
double  duration;
start = clock();
char tmpbuf[128];
_strtime(tmpbuf);
std::cout<<"Start compress["<<tmpbuf<<"].\n";
std::cout<<"Wait...\n";
long best_index = 0;
float cur_sdvig = 0;
float best_sdvig = 0;
float rms_min = MAXIM;
float rms_cur = 0;
for(long i_block = 0; i_block < m_count_blocks; i_block++)
{
  rms_min = MAXIM;
  best_index = 0;
  best_sdvig = 0;
  rms_cur = 0;
  for(long i_domain = 0; i_domain < m_count_domains; i_domain++)
  {
    BestCoeffForDomain(i_domain,i_block,&rms_cur,&cur_sdvig);
    if(rms_cur < rms_min)
```

```
          {
            best_index = i_domain;
            best_sdvig = cur_sdvig;
            rms_min = rms_cur;
          }
        }
        m_out[i_block].index = best_index;
        m_out[i_block].scale = long(floor(best_sdvig));
      }
      finish = clock();
      duration = (double)(finish - start) / CLOCKS_PER_SEC;
      _strtime(tmpbuf);
      std::cout<<"Compress finished["<<tmpbuf<<"].\n";
      std::cout<<"Compress time("<<duration<<" sec).\n";
      if(true == WriteCoefsToBinFile(m_FNameCmprs))
      {
        std::cout<<"File "<<m_FNameCmprs<<" was writed succesfull!\n";
      }
          }
}
/****************************************
*Description: Compute RMS.
****************************************/
float CFractalCompress::RMS(long index_d,long index_b, float* v)
{
  float rms = 0;
  (*v) = CalculateV(index_d,index_b);
  for(long i = 0; i < m_size_of_block; i++)
  {
    rms += pow(0.75*m_domains_data[index_d][i] + (*v) - m_blocks_data[index_b][i],2);
  }
  return rms;
}
/****************************************
*Description: Find the best coeffs for this
```

```
*           domain and block.
*****************************************/
void CFractalCompress::BestCoeffForDomain(long i_domain,  //index of domain
                                          long i_block,   //index of block
                                          float* rms,     //return
                                          float* best_v)
{

  float rms_cur = 0;
  float v = 0;
  rms_cur = RMS(i_domain,i_block, &v);
  (*best_v) = v;
  (*rms) = rms_cur;
}
/*****************************************
*Description: Calculate sdvig.
*****************************************/
float CFractalCompress::CalculateV(long index_d, long index_b)
{
  float sum = 0;
  for(long i = 0; i < m_size_of_block; i++)
  {
    sum += (m_blocks_data[index_b][i]  - 0.75*m_domains_data[index_d][i]);//
  }
  return sum/m_size_of_block;
}
/*****************************************
*Description: Write fractal coefs
*           to binary file.
*****************************************/
bool CFractalCompress::WriteCoefsToBinFile(char* FileName)
{
  bool res = true;
  try
  {
```

```
    FILE* fout;
    if((fout = fopen(FileName,"wb")) != NULL)
    {
      fwrite(&"SFC",1,3,fout);//HeadID
      fwrite(&m_count_data,1,4,fout);
      fwrite(&m_size_of_block,1,4,fout);
      fwrite(&m_SamplesPerSec,1,4,fout);
      unsigned long index = 0;
      long scale = 0;
      for(long i = 0; i < m_count_blocks; i++)
      {
        index = m_out[i].index;
        fwrite(&index,1,2,fout);
        scale = m_out[i].scale;
        fwrite(&scale,1,1,fout);
      }
    }
    fclose(fout);
  }
  catch(...)
  {
    std::cout<<"Can not write to file"<<FileName<<"!!!\n";
    res = false;
  }
  return res;
}
```

Класс CFractalDecompress

Файл «FractalDecompress.h»

```
#ifndef FRACTALDECOMPRESS_H
#define FRACTALDECOMPRESS_H

#include "Common.h"
#include "wav.h"
//Константные значения для заголовка .wav файла
```

```
const long count_iteration = 16;
const unsigned long fmHeadSize = 16;
const unsigned short wFormatTag = 1;
const unsigned short Channels = 1;
const unsigned short nBlockAlign = 1;
const unsigned short nBitsPerSample = 8;

//Коэффициенты IFS
struct FractalCoefs
{
  unsigned long index:16;
  long scale:8;
};
//Класс для декомпрессии аудиоданных
class CFractalDecompress
{
public:
  //Конструктор
        CFractalDecompress(char*FileNameCoefs,//имя файла для декомпрессии
                char* FNameRec);   //имя выходного .wav файла
        virtual ~CFractalDecompress();
  //Поцедура декомпрессии данных
  void DecompressData();
private:
  //имя файла для декомпрессии
  char* m_FNameRec;
  unsigned long m_SamplesPerSec;
  //Длина исходных данных
  long m_count_data;
  //Количество ранговых последовательностей
  long m_count_blocks;
  //Количество доменных последовательностей
  long m_count_domains;
  //размер рангового блока
        long m_size_of_block;
```

```
  //Воостановленный сигнал
  float* m_out_data;
  //Ранговые последовательности
  float** m_blocks_data;
  //Доменные последовательности
        float** m_domains_data;
  //Массив коэффициентов IFS
  FractalCoefs* m_in;
  //Инициализация массивов
  void InitData();
  void InitInputRange();
  //Разбиение исходного сигнала на ранговые последовательности
  bool Break2Blocks(void);
  //Разбиение исходного сигнала на доменные последовательности
        bool Break2Domains(void);
  //Сборка ранговых последовательности в один массив
  bool JoinBlocks(void);последовательности
  //Воостановление index-ой ранговой последовательности
  void ReconstructBlock(long index);
  //Удаление нулей в конце сигнала
  void DeleteZeroEnd();
  //Запись .wav файла
  bool WriteToFile(char* FileName);
  //Чтение коэффициентов из файла
  bool ReadCoefsFromBinFile(char* FileName);
};
#endif //#ifndef FRACTALDECOMPRESS_H
```

Файл «FractalDecompress.cpp»

```
#include "FractalDecompress.h"
#include "wav.h"

CFractalDecompress::CFractalDecompress(char*FileNameCoefs, char* FNameRec)
{
  m_FNameRec = FNameRec;
```

```
  m_count_blocks = 0;
  m_count_data = 0;
  m_count_domains = 0;
  m_SamplesPerSec = 0;
  ReadCoefsFromBinFile(FileNameCoefs);
  InitData();
}

CFractalDecompress::~CFractalDecompress()
{
  delete [] m_out_data;
  for(long i = 0; i < m_count_blocks; i++)
  {
    delete [] m_blocks_data[i];
  }
  delete [] m_blocks_data;
  for(i = 0; i < m_count_domains; i++)
  {
    delete [] m_domains_data[i];
  }
  delete [] m_domains_data;
  delete [] m_in;
}
/*****************************************
*Description: Initialize m_domains_data, m_blocks_data.
*****************************************/
void CFractalDecompress::InitData()
{
  m_out_data = new float[m_count_data];
  for(long i = 0; i < m_count_data; i++)
  {
    m_out_data[i] = float(sin(i/20.0)*100);//
  }
   //blocks
  m_blocks_data = new float*[m_count_blocks];
```

```
        for(i = 0; i < m_count_blocks; i++)
        {
                m_blocks_data[i] = new float[m_size_of_block];
        }
  for( i = 0; i < m_count_blocks; i++)
        {
                for(long j = 0; j < m_size_of_block; j++)
                {
                        m_blocks_data[i][j] = 0;
                }
        }
  //domains
  m_domains_data = new float*[m_count_domains];
        for(i = 0; i < m_count_domains; i++)
        {
                m_domains_data[i] = new float[m_size_of_block];
        }
  for( i = 0; i < m_count_domains; i++)
        {
                for(long j = 0; j < m_size_of_block; j++)
                {
                        m_domains_data[i][j] = 0;
                }
        }
}
/*******************************************
*Description: Initialize m_in_index,
*           m_in_transform and m_in_scale.
*******************************************/
void CFractalDecompress::InitInputRange()
{
  m_in = new FractalCoefs[m_count_blocks];
  for(long j = 0; j < m_count_blocks; j++)
        {
                m_in[j].index = 0;
```

```
        m_in[j].scale = 0;
            }
    }
    /****************************************
    *Description: Break sound data to blocks
    *           with size m_size_of_block.
    ****************************************/
    bool CFractalDecompress::Break2Blocks(void)
    {
        bool result = true;
        try
        {
            long k = 0;
            for(long i = 0; i < m_count_data-m_size_of_block+1; i = i +
m_size_of_block, k++)
            {
                for(long j = 0; j < m_size_of_block; j++)
                {
                    m_blocks_data[k][j] = m_out_data[i+j];
                }
            }
        }
        catch(...)
        {
            result = false;
        std::cout<<"Error: Can not break to blocks\n";
        }
        return result;
    }
    /****************************************
    *Description: Break sound data to domains
    *           with size 2*m_size_of_block.
    ****************************************/
    bool CFractalDecompress::Break2Domains(void)
    {
```

```
		bool result = true;
		try
		{
		  long k = 0;
			for(long i = 0; i < 2*m_count_domains; i+=2, k++)
			{
				for(long j = 0; j < 2*m_size_of_block; j+=2)
				{
					m_domains_data[k][j/2] = m_out_data[i+j];
				}
			}
		}
		catch(...)
		{
			result = false;
	    std::cout<<"Error: Can not break to domains!!!\n";
		}
		return result;
	}
	bool CFractalDecompress::JoinBlocks(void)
	{
	  bool result = true;
		try
		{
			long k = 0;
			for(long i = 0; i < m_count_blocks*m_size_of_block; i = i +
m_size_of_block, k++)
			{
				for(long j = 0; j < m_size_of_block; j++)
				{
					m_out_data[i+j] = m_blocks_data[k][j];
				}
			}
		}
		catch(...)
```

```
        {
                result = false;
    std::cout<<"Error: Can not join data!!!\n";
        }
        return result;
}

void CFractalDecompress::DecompressData()
{
  try
  {
    std::cout<<"Start decompress.\nPlease wait...\n";
    //
    clock_t start, finish;
    double  duration;
    start = clock();
    //
    for(long z = 0; z < count_iteration; z++)
    {
      Break2Blocks();
      Break2Domains();
      for(long i_block = 0; i_block < m_count_blocks; i_block++)
      {
        ReconstructBlock(i_block);
      }
      JoinBlocks();
    }
    DeleteZeroEnd();
    WriteToFile(m_FNameRec);
    //
    finish = clock();
    duration = (double)(finish - start) / CLOCKS_PER_SEC;
    std::cout<<"Decompress finished\n";
    std::cout<<"Decompress time("<<duration<<" sec)\n";
  }
```

```
  catch(...)
  {
    std::cout<<"Error: Can not decompress this data!!!\n";
  }
}

void CFractalDecompress::ReconstructBlock(long index)
{
  try
  {

    long i_domain = m_in[index].index;
    for(long i = 0; i < m_size_of_block; i++)
    {
      m_blocks_data[index][i] = (0.75*m_domains_data[i_domain][i]+ m_in[index].scale);
    }
  }
  catch(...)
  {
    std::cerr<<"Error: Can not reconstruct block"<<index<<"!!!\n";
  }
}

void CFractalDecompress::DeleteZeroEnd()
{
  float avg = 0;
  for(long i = 0; i < m_count_blocks*m_size_of_block; i++)
  {
    avg += m_out_data[i];
  }
  avg = avg/(m_count_blocks*m_size_of_block);
  for(i = m_count_blocks*m_size_of_block; i < m_count_data; i++)
  {
    m_out_data[i] = avg;
  }
```

```
}

bool CFractalDecompress::WriteToFile(char* FileName)
{
  bool res = true;
  FILE* fout;
  try
  {
    if((fout = fopen(FileName,"wb")) != NULL)
    {
      fwrite(&"RIFF",1,4,fout);//HeadID
      const long size_fm_data_head = 36;
      unsigned long headSize = m_count_data + size_fm_data_head;//13984;
      fwrite(&headSize,1,4,fout);
      fwrite(&"WAVE",1,4,fout);//HeadWav

      fwrite(&"fmt ",1,4,fout);//fmHeadID
      fwrite(&fmHeadSize,1,4,fout);

      fwrite(&wFormatTag,1,2,fout);
      fwrite(&Channels,1,2,fout);
      fwrite(&m_SamplesPerSec,1,4,fout);
      fwrite(&m_SamplesPerSec,1,4,fout);
      fwrite(&nBlockAlign,1,2,fout);
      fwrite(&nBitsPerSample,1,2,fout);

      fwrite(&"data",1,4,fout);//HeadDataID
      fwrite(&m_count_data,1,4,fout);
      unsigned long tmp = 0;
      for(long i = 0; i < m_count_data; i++)
      {
        tmp = long(floor(m_out_data[i]))+128;
        fwrite(&tmp,1,1,fout);
      }
      fclose(fout);
```

```
    }
  }
  catch(...)
  {
    std::cout<<"Can not write to file: "<<FileName<<"!!!\n";
    res = false;
  }
  return res;
}

bool CFractalDecompress::ReadCoefsFromBinFile(char* FileName)
{
  bool res = true;
  try
  {
    FILE* fin;
    if((fin = fopen(FileName,"rb")) != NULL)
    {
      char Id[4] = "";
      fread(&Id,1,3,fin);//HeadID
      if(!strcmp(Id,"SFC"))
      {
        fread(&m_count_data,1,4,fin);
        fread(&m_size_of_block,1,4,fin);
        fread(&m_SamplesPerSec,1,4,fin);
        m_count_blocks = long(m_count_data/m_size_of_block);
              m_count_domains = long((m_count_data-m_size_of_block*2)/2);
        InitInputRange();
        unsigned long index = 0;
        long scale = 0;
        for(long i = 0; i < m_count_blocks; i++)
        {
          fread(&index,1,2,fin);
          m_in[i].index = index;
          fread(&scale,1,1,fin);
```

```
            m_in[i].scale = scale;
          }
        }
        else
        {
          std::cerr<<FileName<<"is not .SFC file!!!\n";
          res = false;
        }
      }
      fclose(fin);
    }
    catch(...)
    {
      std::cout<<"Can not read from file"<<FileName<<"!!!\n";
      res = false;
    }
    return res;
}
```

Класс CWav

Файл «Wav.h»

```
#ifndef WAV_H
#define WAV_H

#include "common.h"

//Класс для чтения данных из .wav файла
class CWav
{
private:
  //данные в беззнаковом виде
  unsigned long* m_data;
  //данные в знаковом виде
  long* m_out_data;
  //Длина последовательности
```

```
  unsigned long m_count;
  unsigned long m_SamplesPerSec;
  char* m_FName;
  //Инициализация массивов
  void InitData();
public:
  CWav(char* FileName);
  ~CWav();
  //Чтение .wav файла
  bool ReadFile();
  //Получение уровней отсчетов сигнала
  long* GetData();
  //Получение длины данных
  unsigned long GetCountData();
  unsigned long GetSamplesPerSec();
};
#endif //#ifndef WAV_H
```

Файл «Wav.cpp»

```
#include "wav.h"

struct tagHead
{
  char ckID[5];
  unsigned long ckSize;
  char ckWav[5];
}wavHead;

struct tagDataHead
{
  char ckID[5];
  unsigned long ckSize;
}wavHeadData;

CWav::CWav(char* FileName)
```

```
{
  m_FName = FileName;
  m_SamplesPerSec = 0;
  m_count = 0;
}

CWav::~CWav()
{
     delete [] m_data;
}

bool CWav::ReadFile()
{
  bool result = true;
  FILE* fin;
  if((fin = fopen(m_FName,"rb"))!=NULL)
  {
    fread(&wavHead.ckID,1,4,fin);
    const char chekID[] = "RIFF";
    if(strcmp(wavHead.ckID,chekID))
    {
      printf("Not a RIFF format file\n");
      result = false;
    }
    fread(&wavHead.ckSize,1,4,fin);
    fread(&wavHead.ckWav,1,4,fin);
    if(strcmp(wavHead.ckWav,"WAVE"))
    {
      printf("Not a WAVE file\n");
      result = false;
    }
    const long dist_to_SPR = 12;
    unsigned long nextseek = ftell(fin);
    fseek(fin,nextseek+dist_to_SPR,0);
    fread(&m_SamplesPerSec,1,4,fin);
```

```
    const dist_to_data = 8;
    nextseek=ftell(fin) + dist_to_data;
    fseek(fin,nextseek,0);
    fread(&wavHeadData.ckID,1,4,fin);
    fread(&wavHeadData.ckSize,1,4,fin);
    if(!strcmp(wavHeadData.ckID,"data")) //Data Chunk
    {
      m_count = wavHeadData.ckSize;
      InitData();
      for(long i = 0; i < m_count; i++)
      {
        fread(&m_data[i],1,1,fin);
      }
      for( i = 0; i < m_count; i++)
      {
        m_out_data[i] = m_data[i] - 128;
      }
      fclose(fin);
    }
    else
    {
      result = false;
    }
  }
  else
  {
    printf("Error opening .WAV file [%s]\n",m_FName);
    result = false;
  }
  return result;
}

long* CWav::GetData()
{
  return m_out_data;
```

```
}

unsigned long CWav::GetCountData()
{
  return m_count;
}

unsigned long CWav::GetSamplesPerSec()
{
  return m_SamplesPerSec;
}

void CWav::InitData()
{
  m_data = new unsigned long[m_count];
  m_out_data = new long[m_count];
  for(long i = 0; i < m_count; i++)
  {
    m_data[i] = 0;
    m_out_data[i] = 0;
  }
}
```

Использование классов CFractalCompress и CFractalDecompress происходит в процедуре main():

```
#include "Common.h"
#include "FractalCompress.h"
#include "FractalDecompress.h"
////////////////////////////////////////////////////////////////////////////
int main( int argc, char* argv[] )
{
        using std::cout;
        using std::cin;
  if(argc <= 5)
  {
```

```
    if(!strcmp(argv[3],"-c"))
    {
      long range_size = atol(argv[4]);
      CFractalCompress fcSound(argv[1],argv[2],range_size);
      fcSound.CompressData();
    }
    else if(!strcmp(argv[3],"-r"))
    {
      CFractalDecompress fcSound(argv[1],argv[2]);
      fcSound.DecompressData();
    }
    else
    {
      std::cerr<<"Wrong operation was set!\n";
    }
  }
  else
  {
    std::cerr<<"Too much parameters!\n";
  }
  return 0;
}
//wavsounds\chimes8.wav compress_coef\chimes14.sfc -c 14
//compress_coef\chimes14.sfc compress_coef\\chimes14.wav -r
```

Часто используемые заголовочные файлы были вынесены отдельно:

Файл «Common.h»

```
//Common include files
#ifndef COMMON_H
#define COMMON_H

#include <stdio.h>
#include <iostream>
#include <fstream>
#include <math.h>
```

```
#include <stdlib.h>
#include <conio.h>
#include <time.h>
const int  MAXIM = 9999999;
#endif // #ifndef COMMON_H
```

ДОДАТОК Б

ВІДГУК КЕРІВНИКА ЕКОНОМІЧНОГО РОЗДІЛУ

ДОДАТОК В

ПЕРЕЛІК ФАЙЛІВ НА ДИСКУ

Ім’я файлу	**Опис**
Пояснювальні документи	
Диплом_ doc	Пояснювальна записка до кваліфікаційної роботи. Документ Word.
Диплом_.pdf	Пояснювальна записка до кваліфікаційної роботи в форматі PDF
Програма	
Program.rar	Архів. Містить коди програми і откомпільовану програму
Презентація	
Презентація_.ppt	Презентація кваліфікаційної роботи