Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ Информатики, искусственного интеллекта и систем управления

КАФЕДРА Теоретической информатики и компьютерных технологий

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОЙ РАБОТЕ по дисциплине

Алгоритмы компьютерной графики

НА ТЕМУ:

Применение фрактальных алгоритмов для масштабирования береговых линий на цифровых картах местности

Студент ИУ9-51Б _____ **М.В.Куценков**
(Группа) (Подпись, дата) (И.О.Фамилия)

Руководитель курсовой работы _____ **А.Б.Домрачева**
(Подпись, дата) (И.О.Фамилия)

2023 г.

Оглавление

Введение

Расположение береговой линии и изменение её положения во времени имеет большое значение для специалистов, занимающихся прибрежными районами. Береговая линия может изменяться со временем из-за природных факторов, таких как приливы, эрозия, затопление или изменение уровня моря. Также береговая линия может быть важной зоной для различных видов деятельности, таких как туризм, рыболовство, судоходство и добыча нефти и газа. Исследование береговой линии может помочь в понимании изменения её положения во времени и его влияния на окружающую среду, содействовать в планировании и управлении указанными видами деятельности.

Известно, что береговая линия, как и многие другие природные объекты, является самоподобным, то есть масштабно-инвариантным объектом, не имеющим характерной длины. Поэтому, изменяя рассматриваемый масштаб изображения, невозможно определить, к какому изображению относится какой масштаб, так как оба они выглядят статистически одинаково. Из-за этого свойства открывается возможность масштабировать цифровые изображения береговой линии, получая новые детали рельефа, что позволяет проводить более глубокие исследования местности.

Цель работы – анализ и сравнение стохастических методов фрактальной интерполяции применительно к береговым кривым на цифровой карте местности.

1. Теоретические сведения

1.1. Описание предметной области

Согласно Водному Кодексу Российской Федерации [1], **водный объект** - природный или искусственный водоем, водоток либо иной объект, постоянное или временное сосредоточение вод в котором имеет характерные формы и признаки водного режима.

Водные объекты подразделяются на поверхностные и подземные.

К поверхностным водным объектам относят:

- моря и их отдельные части (проливы, заливы, бухты, лиманы);

- водотоки: реки, ручьи, каналы;

- водоемы: озера, пруды, обводненные карьеры, водохранилища;

- болота;

- природные выходы подземных вод: родники, гейзеры;

- ледники, снежники.

Береговая линия (граница поверхностного водного объекта) определяется:

– для моря – по постоянному уровню воды, а в случае периодического

изменения уровня воды – по линии максимального отлива;

– для реки, ручья, канала, озера, обводненного карьера – по среднемноголетнему уровню воды за период, когда они не покрыты льдом;

– для пруда, водохранилища – по нормальному подпорному уровню воды;

– для болота – по границе залежи торфа на нулевой глубине.

Согласно Межгосударственному Стандарту цифровой картографии [2], **цифровая карта;** ЦК (**цифровая карта местности**; ЦКМ): Цифровая картографическая модель, содержание которой соответствует содержанию карты определенного вида и масштаба. Классификация цифровых карт соответствует общей классификации карт, например: цифровая топографическая карта, цифровая авиационная карта, цифровая геологическая карта, цифровая кадастровая карта и другие.

Цифровая карта - основа информационного обеспечения автоматизированных картографических систем (АКС) и географических информационных систем (ГИС), может являться результатом их работы.

Способы сбора пространственной информации, с помощью которых создаются цифровые карты:

- оцифровка аналоговых картографических произведений (например, бумажных карт);

- фотограмметрическая обработка данных дистанционного зондирования;

- полевая съёмка (например, геодезическая тахеометрическая съёмка или съёмка с использованием приборов систем глобального спутникового позиционирования);

- камеральная обработка (в помещении) данных полевых съёмок и иные методы.

1.2. Описание исходных данных

В качестве исходных данных в данной работе будут использоваться изображения, полученные спутником Landsat 5.

Landsat – американская программа дистанционного зондирования Земли (ДЗЗ), в рамках которой было запущено 9 спутников. Это самый продолжительный проект съёмки земли из космоса [3].

Спутник Landsat 5 был запущен 1 марта 1984 года и находился под контролем Национального управления по аэронавтике и исследованию космического пространства (NASA) и Геологической службы США (USGS) до июня 2013 года, попав в книгу рекордов Гиннесса как самая длительная миссия спутникового наблюдения Земля, проработав более 28 лет. Он является полной копией спутника Landsat 4, оснащён двумя типами сканеров, обеспечивающими съёмку земной поверхности с различным пространственным и спектральным разрешением - MSS (Multispectral Scanner System – мультиспектральный сканер) и TM (Thematic Mapper – тематический картограф) [4].

Мультиспектральный сканер сканирует Землю вертикально, строго в надир, поперек направления полета спутника. На поверхности Земли захватывается полоса шириной 185 км. Съемка проводится в четырех спектральных зонах:

1. 0,50-0,60 мкм (видимый зеленый);

2. 0,60- 0,70 мкм (видимый красный);

3. 0,70-0,80 мкм (ближний инфракрасный);

4. 0,80-1,10 мкм (ближний инфракрасный).

Разрешение на местности составляет 80 м для оптического диапазона и 240 м для теплового, масштаб изображения - 1:400 000.

Тематический картограф сканирует Землю также при вертикальном направлении оптической оси, с такой же, как у мультиспектрального сканера, шириной захвата на местности - 185 км, в семи спектральных зонах:

1. 0,45-0,52 мкм (видимый синий);

2. 0,52- 0,60 мкм (видимый зеленый);

3. 0,63-0,69 мкм (видимый красный);

4. 0,76-0,90 мкм (ближний инфракрасный - NIR);

5. 1,55-1,75 мкм (коротковолновой инфракрасный - SWIR);

6. 2,08-2,35 мкм (тепловой – TIR/LWIR);

7. 10,4-12,4 мкм (коротковолновой инфракрасный – SWIR2).

Разрешение на местности составляет 30 м для первых пяти и седьмого каналов и 120 м для шестого. Масштаб изображения – 1:150000 (для шестого канала – 1:600) [5].

Период повторной съёмки спутника (интервал повторения) – 16 дней, период обращения – 99 минут, что влечёт собой оборот около 14.5 орбит в сутки.

Данные снимков хранятся в формате GeoTIFF, который представляет собой набор растровых данных в формате TIFF, включая метаданные о географической привязке.

Важный фактором в пользу использования данных спутников Landsat является свободный доступ к коллекциям данных. В эти коллекции также входят вспомогательные файлы, одним из которых является снимок оценки качества пикселей (Pixel Quality Assessment Band – QA_PIXEL), содержащий статистику, полученную из данных изображения, и маску облачных областей для данной сцены.

Для создания снимка оценки качества (далее – QA Band) пикселей используется результат применения алгоритма CFMask, подающийся на вход в приложение оценки качества (Quality Assessment Application), рассчитывающее все поля в файле снимка QA Band. QA Band состоит из беззнаковых 16-битовых чисел в таких же измерениях как остальные снимки сцены. Выделяются биты для классификации типов поверхности земли и артефактов данных. Нулевой бит считается наименее значащим. Для каждого типа классификации, состоящего из двух битов, предоставляется уровень уверенности: 00 – алгоритм не определил это условие, 01 – алгоритм имеет низкую уверенность в существовании этого условия (от 0% до 33%), 10 – алгоритм имеет среднюю уверенность в существовании этого условия (от 34% до 66%), 11 – алгоритм имеет высокую уверенность в существовании этого условия (от 67% до 100%). Для типов классификации, состоящий из одного бита, 1 считается выполнением этого условия, а 0 – его отсутствием.

Таблица 1 – описание структуры битов в снимке QA Band [6,с.12].

Номер битов	Описание флага	Значения
0	Заполнение	0 для данных изображения, 1 для заполняющих данных
1	Расширенное облако	0 если облако не расширено или нет облака, 1 если облако расширено
2	Не используется	Не используется
3	Облако	0 если уверенность в наличии облака не высокая, иначе - 1
4	Тень облака	0 если уверенность в наличии тени облака не высокая, иначе - 1

5	Снег	0 если уверенность в наличии снега или льда не высокая, иначе - 1
6	Чисто	0 если установлены биты облака или расширенного облака, иначе - 1
7	Вода	0 для земли или облака, 1 для воды
8-9	Уверенность в наличии облака	00 для отсутствия уверенности, 01 для низкой уверенности, 10 для средней уверенности, 11 для высокой уверенности
10-11	Уверенность в наличии тени облака	00 для отсутствия уверенности, 01 для низкой уверенности, 10 для средней уверенности, 11 для высокой уверенности
12-13	Уверенность в наличии снега или льда	00 для отсутствия уверенности, 01 для низкой уверенности, 10 для средней уверенности, 11 для высокой уверенности
14-15	Не используется	Не используется

Так как биты 3 и 4 устанавливаются в 1 тогда и только тогда, когда биты 8-9 и 10-11 устанавливаются в 11 соответственно, то для того, чтобы получить маску облачных областей изображения, необходимо получить все пиксели изображения, для которых установлены биты 3 или 4 (большая уверенность), либо биты 10-11 или 12-13 установлены в виде 10 (средняя уверенность). В итоге получается маска, в которой каждый установленный пиксель с вероятностью от 34% до 100%, с точки зрения алгоритма, анализирующего снимка, является облаком или тенью облака. Получение этой информации важно для анализа береговых линий, так как облака перекрывают линию обзора спутника, не позволяя сделать в месте их присутствия снимок поверхности земли, что приведёт к трудностям при дальнейшей обработке изображения, если этот факт не учесть.

1.3. Описание водных растровых индексов

Для последующего анализа изображения с целью извлечения береговой линии необходимо с помощью осуществления преобразований над значениями пикселей разных спектральных каналов получить растровую поверхность, на основании которой можно будет классифицировать пиксели изображения на водные объекты или объекты, содержащие влагу, и объекты на суше. Такие растровые поверхности называются водными индексами.

Рассмотрим некоторые водные индексы.

Водный индекс **WRI** (Water Ratio Index) используется для оценки содержания влаги в растительном покрове, рассчитывается по формуле:

$$WRI = (GREEN + RED)/(NIR + SWIR2), \qquad (1)$$

где *GREEN* – значения на втором канале, *RED* – значения на третьем канале , *NIR* – значения на четвёртом канале , *SWIR2* – значения на седьмом канале.

Результатом является растр поверхности со значениями пикселей от 0 до 3, значения от 1 и выше – водные объекты или объекты, содержащие влагу.

Нормализованный разностный водный индекс **NDWI** (Normalized Difference Water Index) определяет количество влагозапаса в растительном покрове, которое взаимодействует с поступающим солнечным излучением, чувствителен к изменениям влажности. Вычисляется по формуле:

$$NDWI = (NIR - SWIR2)/(NIR + SWIR2), \qquad (2)$$

где *NIR* – значения на четвёртом канале, *SWIR2* – значения на седьмом канале.

Данный индекс необходим для того, чтобы обнаруживать поверхностные воды среди заболоченной местности; измеряет степень покрытия поверхностными водами. Значения индекса колеблются в диапазоне от -1 до 1. Считается, что водные объекты принимают значения от 0,2 до 1, объекты, не содержащие влагу, принимают значения меньше 0.

Модифицированный нормализованный разностный водный индекс **MNDWI** (Modified Normalized Difference Water Index) позволяет эффективно подавлять и удалять шумовые эффекты с поверхности, почвы и растительности. Является более эффективным по сравнению с NDWI, в частности в случаях, когда какие-либо строения находятся рядом с водным объектом. Индекс рассчитывается по формуле:

$$MNDWI = (GREEN - SWIR2)/(GREEN - SWIR2), \qquad (3)$$

где *GREEN* – значения на втором канале, *SWIR2* – значения на седьмом канале.

Значения этого индекса лежат в диапазоне от -1 до 1. Вода имеет значения больше 0.

Естественно, существуют различные индексы, не описанные выше, также позволяющие классифицировать изображения для выделения береговой линии, в том числе индекс **AWEI**, обладающий ещё большей точностью, чем индекс MNDWI, но при этом использующий значения из пяти спектров снимка и меняющийся в зависимости от наличия факторов, способных повлиять на точность его вычисления, таких, как снег или лёд. Отметим, что индекс WRI использует значения из четырёх спектров снимка, двое из которых являются видимыми спектрами, индекс NDVI использует два спектра, оба из которых не являются видимыми, а индекс MNDWI использует два спектра, один из которых является видимым. Из этого следует, что при стремлении использовать как можно меньше спектров снимка одновременно, что обусловлено большим размером каждого из спектров снимка, следует использовать индекс MNDWI, так как его использование требует обращения к одному из спектров снимка, не входящих в видимый свет (SWIR2). Безусловное же использование трёх спектральных снимков видимого света обусловлено стремлением на их основе изобразить цветное (RGB) изображение снимка.

1.4. Сведения из теории фракталов

Несмотря на широкое распространение, понятие фрактала не имеет четкого и строгого определения. Приведём наиболее простое и краткое определением фракталов. **Фракталом** называется структура, состоящая из частей, которые в каком-то смысле подобны целому. Однако это определение не дает полного представления о разнообразии объектов, которые относят к фракталам. Рассмотрим также определение, которое дал Бенуа Мандельброт, математик, исследовавший фракталы и введший этот термин в научный оборот: «фракталом называется множество, размерность Хаусдорфа-Безиковича которого строго больше его топологической размерности» [9].

Это определение строго в математическом плане, но это является его существенным недостатком, поскольку оно требует определения еще и понятий размерности (топологической и хаусдорфовой), при этом оно исключает многие классы фрактальных объектов, встречающиеся в различных областях естествознания.

Мандельброту принадлежит ещё и более общее и менее формальное определение: «Все фигуры, которые я исследовал и назвал фракталами, в моем представлении обладали свойством быть нерегулярными, но самоподобными».

Таким образом, при характеристике фрактала центральным понятием является самоподобие. Можно сказать, что фрактальный объект статистически единообразен в широком диапазоне масштабов. В идеальном случае (математический фрактал) такое самоподобие приводит к тому, что фрактальный объект оказывается инвариантным относительно масштабных изменений пространства (растяжений и сжатий).

Рассмотрим существующую классификацию фракталов.

Геометрические фракталы получают с помощью некоторой ломаной (в двухмерном случае) или поверхности (в трёхмерном пространстве), называемой генератором. За один шаг алгоритма построения каждый из отрезков, составляющих ломаную, заменяется на ломаную-генератор, в соответствующем масштабе. В результате бесконечного числа повторения этой процедуры, получается геометрический фрактал. В качестве примера геометрического фрактала можно привести триадную кривую Коха.

Построение кривой Коха начинается с отрезка единичной длины – это нулевое поколение кривой. Затем каждое звено (один отрезок в нулевом поколении) заменяется на образующий элемент. В результате формируется новое поколение кривой Коха.

В первом поколении кривая состоит из 4-х прямолинейных звеньев, каждое из которых длиной по $\frac{1}{3}$. Для получения дальнейших поколений аналогичные шаги повторяются. Значит, создавая новые последующие поколения, все звенья предыдущего требуется заменить уменьшенным образующим элементом. Кривая n-го поколения при любом конечном n называется **предфракталом**. Рисунок 1 отражает 3 поколения кривой. При стремлении n к бесконечности кривая Коха является фрактальным объектом [10].

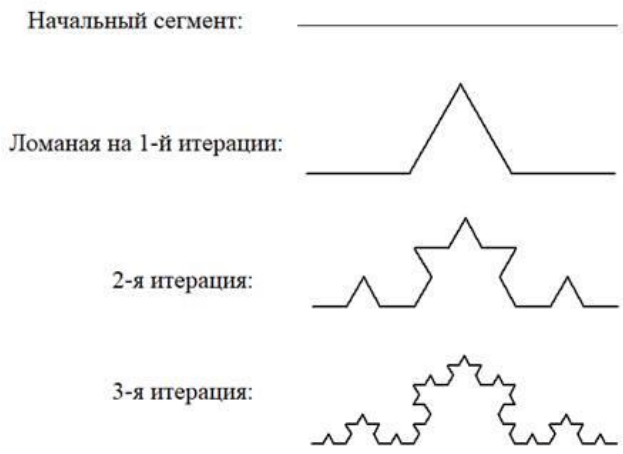

Рисунок 1 – построение кривой Коха

Для получения иных фрактальных объектов изменяются правила, на основании которых производится построение. Применение геометрических фракталов в компьютерной графике позволяет получить изображение природных объектов: деревьев, кустов, береговой линии и так далее.

Ещё одна группа фракталов, являющаяся самой крупной – **алгебраические фракталы**. Данный класс фракталов получают с помощью нелинейных процессов в n-мерных пространствах. Наиболее изучены и распространены процессы двухмерного случая. В качестве примера можно рассмотреть множество Мандельброта, представленное на рисунке 2. Алгоритм его построения основан на итеративном выражении:

$$A[i+1] = A[i] * A[i] + C, \qquad (4)$$

где $A[i]$ и C – комплексные переменные [10].

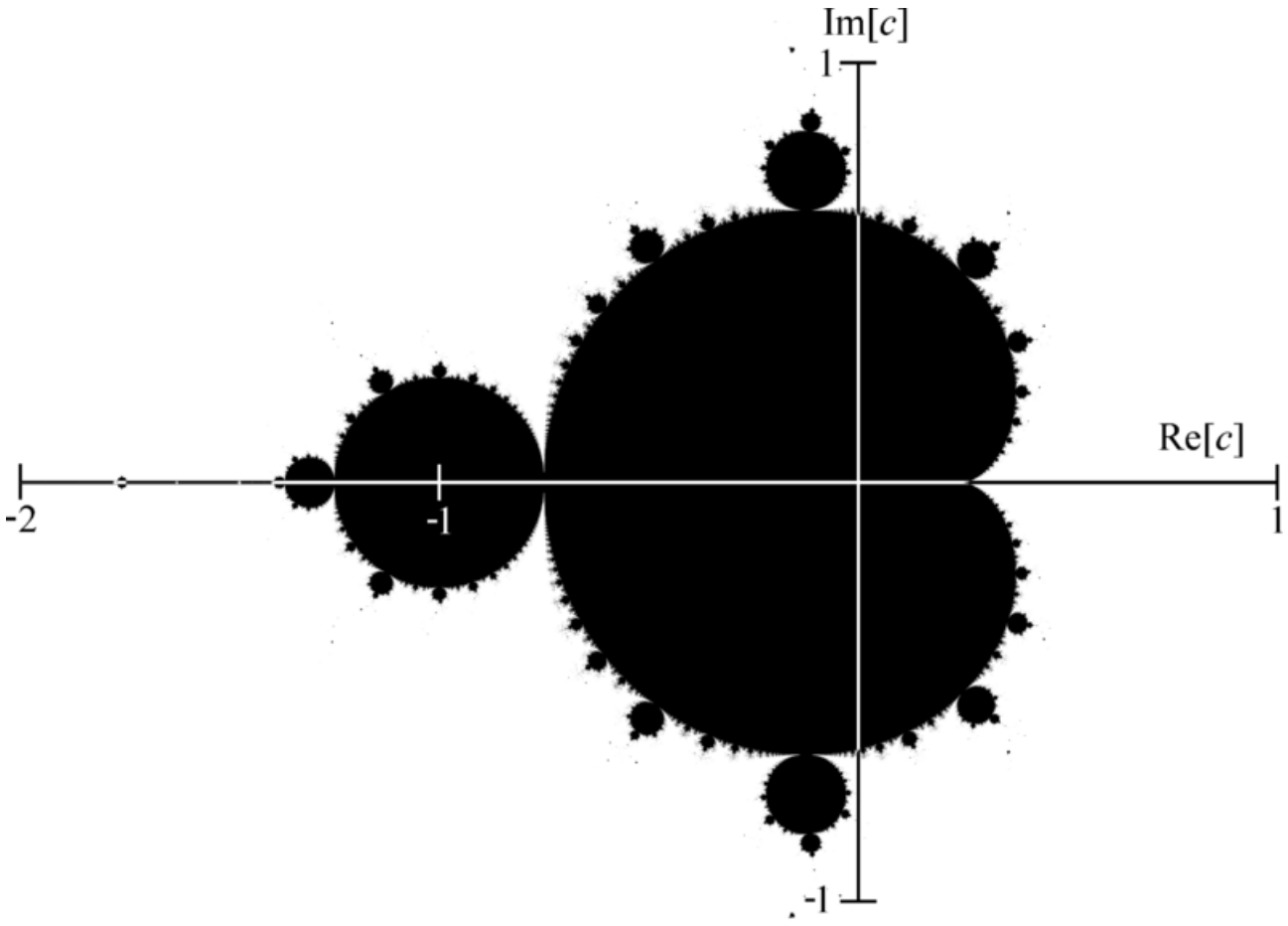

Рисунок 2 – Множество Мандельброта

Стохастические фракталы – это класс фракталов, которые получаются в случае случайного изменения различных параметров в итерационном процессе. Этот класс представлен нерегулярными фракталами, то есть полученными в ходе работы некоторого недетерминированного алгоритма. Их основное отличие от регулярных состоит в том, что свойства самоподобия справедливы только после соответствующего усреднения по всем статистически независимым реализациям объекта. При этом увеличенная часть фрактала точно не идентична исходному фрагменту, но их статистические характеристики совпадают [9]. В этом случае получаются объекты в значительной степени похожие на природные - асимметричные растения, изрезанные береговые линии, горы и так далее. Двумерные стохастические фракталы применяются для моделирования рельефа местности и поверхности моря.

1.5. Задача стохастической фрактальной интерполяции применительно к масштабированию изображений

Интерполяция – нахождение промежуточных значений функции по имеющемуся дискретному набору её значений.

Пусть функция $f(x)$ задана набором точек (x_i,y_i) на интервале $[a,b]$:

$y_i = f(x_i), i = 0,1, \dots, n, a \le x_i \le b$ (5)

Задача интерполяции – нахождение функции $F(x)$, принимающую в точках x_i те же значения y_i. Следовательно, **условие интерполяции**:

$F(x_i) = y_i$ *(6)*

Предполагается, что среди значений x_i нет одинаковых. Точки x_i называются **узлами интерполяции**.

Если поиск $F(x)$ ведётся только на отрезке $[a,b]$ – то это задача интерполяции, а если за пределами первоначального отрезка, то это задача экстраполяции [11].

Задача нахождения **интерполяционной функции** $F(x)$ имеет множество решений, так как через заданные точки x_i, y_i можно провести бесконечно много кривых, каждая из которых будет графиком функции, для которой выполнены все условия интерполяции. Интерполяционную функцию также иногда называют **интерполянтом**.

Естественно, в общем случае кривые, соответствующие береговым линиям, не представляются функциями. Поэтому, для выполнения строгого определения интерполяции необходимо осуществлять разбиение изображения на области, внутри которых уже будет возможно представить кривую в виде функции.

Разбиение изображения на области является одним из этапов **фрактального сжатия графической информации**, впервые предложенного М. Барнсли в 1987 году. Это метод сжатия с потерей качества (lossy), в котором самоподобные части изображения используются для уменьшения объёма дискового пространства, занимаемого файлом [12]. Алгоритм, разработанный Барнсли, по его утверждению, позволяет сжимать информацию в 500-1000 раз [10]. Фрактальный алгоритм преобразует самоподобные части изображения в так называемые фрактальные коды, которые можно использовать для воссоздания исходного изображения. В результате этого преобразования делает изображение не зависящим от разрешения (scale independent), что позволяет применять к нему фрактальное масштабирование, в том числе для повышения разрешения, которое является фрактальной интерполяцией[10].

Рассмотрим преимущества использования метода фрактального сжатия графической информации:

- возможность высокой степени сжатия;
- быстрое восстановление изображения;
- независимость от разрешения;

Главным недостатком метода является низкая скорость кодирования изображений алгоритмами.

Применительно к интерполяции изображений следует рассмотреть фракталы, задаваемые системами итерируемых функций. **Система итерируемых функций** (СИФ, Iterated Functions System - IFS) представляет собой систему функций из определённого фиксированного класса функций, отображающих одно многомерное пространство на другое. Принято считать СИФ системой из n сжимающих отображений w_i на полном метрическом пространстве [13]. Рассматриваемая двумерная СИФ отображает пространство $\mathbb{R}^2$ в себя и сходится в процессе итерации системы:

$$W(x) = \bigcup_{i=1}^{n} w_i(x) \quad \{w_i: \mathbb{R}^2 \to \mathbb{R}^2 \mid i = 1, \dots, n\} \qquad (6)$$

Цифровое изображение может быть сохранено как набор параметров преобразований СИФ и может быть с лёгкостью восстановлено или расшифровано в целях дальнейшего использования или вывода. Хранение коэффициентов преобразований СИФ влечёт высокую степень сжатия и хорошее качество восстановленных изображений.

Рассмотрим общий алгоритм фрактального кодирования [12]:

1. Прочесть бинарное изображение.

2. Преобразовать его в изображение в градации серого (greyscale image).

3. Разделить изображение на малые квадратные блоки без пересечения, называемые блоками диапазона (range block).

4. Ввести большие квадратные блоки с пересечением, называемые блоками домена (domain block). Размер блока домена в два раза больше размера блока диапазона.

5. Для каждого блока диапазона найти соответствующий блок домена, который достаточно близко похож на этот блок диапазона относительно некоторой метрики.

6. Записать сжатые данные в форме локальных кодов СИФ.

7. Применить алгоритм сжатия данных для получения сжатого кода СИФ.

Так как качество закодированного изображения напрямую зависит от размера сетки, формируемой диапазонами, то эти диапазоны можно задавать адаптивно. Тогда размер сетки может быть адаптирован к деталям изображения, используя больший размер сетки для менее детализированных изображений, благодаря чему уменьшается размер изображения, но сохраняется его качество. Наиболее часто используемыми способами адаптивного секционирования являются дерево квадратов (Quadtree), HV разбиение и треугольное разделение (triangular subdivision) [13].

Сжатое изображение может быть восстановлено с помощью итерирования СИФ поверх полученного в результате фрактального сжатия изображения:

$$I \approx |W| = \lim_{n\to\infty} W^{on}(x), \qquad (7)$$

где I – изначальное изображение, W – отображение из (6), o – фактор яркости.

Результат расшифровки не зависит от используемого изображения из-за стремления n к бесконечности, но фрактал может сходится за большее или меньшее количество итераций. Чем более похоже используемое изображение на расшифрованное, тем быстрее СИФ сходится.

Рассмотрим две важные теоремы.

Теорема 1 (Банаха о неподвижной точке).

Пусть (X, d) – полное метрическое пространство с сжимающим отображением T: $X \to X$. Тогда у отображения T существует единственная неподвижная точка x^* из X, то есть $Tx^* = x^*$.

Теорема 2 (Коллажа).

Пусть (X, d) – полное метрическое пространство. Преположим, что L – непустое компактное подмножество X и пусть $\varepsilon > 0$. Выберем СИФ $\{X, w_1, w_2,\ldots, w_n\}$ с коэффициентом сжатия s, $0 \leq s < 1$ (коэффициент сжатия СИФ определяется максимум коэффициентов сжатия отображений w_i).

Положим $h(L, \bigcup_{n=1}^{N} w_n(L)) \leq \varepsilon$, где $h(\cdot, \cdot)$ – хаусдорфова метрика. Тогда $h(L, \mathrm{A}) \leq \frac{\varepsilon}{1-\mathrm{s}}$, где A – аттрактор (устойчивое состояние) СИФ.

Неформально: если L близко к стабилизации системой итерируемых функций, то L близок к тому, чтобы быть аттрактором этой СИФ.

Из этих теорем можно сделать вывод, что при построении СИФ в процессе фрактального кодирования достаточно стремиться к минимизации $d(f, T(f))$, то есть метрики между исходным изображением f и коллажем $T(f)$ – приближением первого порядка неподвижной точки, что в результате минимизирует $d(f, f_f)$, что является целью алгоритма. Метрику $d(f, T(f))$ называют ошибкой коллажа. Вычислительная сложность фрактального представления значительно уменьшается при минимизации ошибки коллажа вместо метрики между исходным изображением и аттрактором $d(f, f_f)$.

Классически СИФ задаются несколькими простыми преобразованиями, а именно их коэффициентами. Часто для этого используются аффинные преобразования, но для построения СИФ помимо них могут использоваться и другие классы простых геометрических преобразований, например, проективные или квадратичные преобразования на плоскости[10].

Также можно использовать методы стохастической фрактальной интерполяции, основанные на построении стохастических фрактальных структур, например, основанные на модели случайного блуждания [15].

2. Анализ алгоритмов стохастической фрактальной интерполяции

2.1. Рандомизированный метод на основе систем итерируемых функций

Одним из способов построения стохастических фракталов является рандомизированный метод на основе систем итерируемых функций.

Построение фракталов на основе СИФ происходит следующим образом [10]:

Пусть для некоторого фрактала требуются N аффинных преобразований. На начальном шаге рандомизированного алгоритма выбирается одна точка.

Общий шаг алгоритма:

1. Случайным образом выбирается одно из N аффинных преобразований;

2. Образуется новая точка путем применения к предыдущей выбранного аффинного преобразования;

3. Новая точка изображается;

4. Выполняется присваивание: новая точка будет рассматриваться в качестве предыдущей для следующего шага алгоритма.

Общий шаг можно повторять заданное число раз либо до тех пор, пока детали изображения не станут мельче заданной величины. Таким образом, в отличие от детерминированного алгоритма на основе IFS на каждом шаге рандомизированного алгоритма обрабатывается только одна точка и применяется только одно аффинное преобразование.

Применение СИФ является достаточно простым средством получения фрактальных структур.

2.2. Алгоритм, использующий L-системы

Понятие L-систем, тесно связанное с фракталами, появилось в 1968 году благодаря Аристриду Линденмайеру. Изначально L-системы были введены при изучении формальных языков, а также использовались в биологических моделях селекции. С их помощью можно строить многие известные фракталы, например, снежинку Коха и ковер Серпинского.

Некоторые другие классические построения, например кривые Пеано (работы Пеано, Гильберта, Серпинского), также укладываются в эту модель. Также L-системы открывают путь к бесконечному разнообразию новых фракталов, что и послужило причиной их широкого применения в компьютерной графике для построения фрактальных деревьев и растений.

Для графической реализации L-систем в качестве подсистемы вывода используется принцип черепашей графики. При этом точка (черепашка) движется по экрану дискретными шагами, как правило, прочерчивая свой след, но при необходимости может перемещаться без рисования. Имеются три параметра (x, y, α), где (x, y) — координаты черепашки, α — направление, в котором она смотрит. Черепашка способна интерпретировать и выполнять последовательность команд, задаваемых кодовым словом, буквы которого читаются слева направо.

Кодовое слово представляет собой результат работы L-системы и может включать следующие буквы:

- F – команда переместиться вперед на один шаг, прорисовывая след;
- b – команда переместиться вперед на один шаг, не прорисовывая след;
- [– команда открыть ветвь;
-] – команда закрыть ветвь;
- + – команда увеличить угол α на величину θ;
- — – команда уменьшить угол а на величину θ.

Размер шага и величина приращения по углу θ задаются заранее и остаются неизменными для всех перемещений черепашки.

Если начальное направление движения α (угол, отсчитываемый от положительного направления оси X) не указано, то можно положить α равным нулю.

Формально, L-система состоит из алфавита, слова инициализации, называемого **аксиомой** или **инициатором**, и набора порождающих правил, указывающих, как следует преобразовывать слово при переходе от итерации к итерации. L-система является параллельной системой переписывания.

Символы +, —,], [не обновляются, а просто остаются на тех местах, где они встретились. Обновление букв в данном слове предполагается одновременным, то есть все буквы слова одного уровня обновляются раньше любой буквы следующего уровня [16,23с].

Когда мы встречаем символ [(открыть ветвь), мы запоминаем положение и направление черепашки, то есть переменные (x, у, α), и возвращаемся к этим установкам позднее. Для хранения троек (x, у, α) используется стек. Когда ветвь закрывается, переменным (x, у, α) присваиваются значения, считанные с вершины стека, и эти значения удаляются из стека.

Таким образом, ветвление задается двумя символами:

- [– открыть ветвь: Сохранить (x, у, α) на вершине стека.

-] – закрыть ветвь: Присвоить переменным (x, у, α) значения, считанные с вершины стека, после чего удалить их из стека.

Если для любого символа в L-системе есть ровно одно правило переписывания, то она называется детерминированной. В противном случае, если для каждого из правил выбрана определённая вероятность, то она называется стохастической.

Пример грамматики для стохастической L-системы:

- переменные: 0, 1;
- константы: «[»,«[»;
- аксиома: 0;
- правила переписывания:
  1) 1 → 11;
  2) 0 (0.5) → 1[0]0;
  3) 0 (0.5) → 0.

Можно заметить, что рядом с символом в левой части переписывания в скобках указана вероятность.

Использование стохастических L-систем позволяет построить множество стохастических фрактальных структур. И хоть классические примеры грамматики стохастической L-системы реализует изображения растений, что обоснованно видом деятельности Линденмайера, возможно также получить изображение фрактальной структуры, похожей на береговую линию, что, однако, потребует некоторого экспериментирования с заданием формальной грамматики для получения необходимой формы фрактала.

2.3. Алгоритм на основе модели броуновского движения

Хорошим классом фрактальных объектов в плане моделирования естественных объектов являются фракталы, случайный характер которых присущ им изначально из-за связи со случайными процессами. Фрактальное броуновское движение — случайный процесс, широко распространенный в природе.

Можно построить рандомизированную снежинку Коха, добавляя на каждом шаге равносторонние треугольники, обращенные как внутрь, так и наружу.

Если на каждом шаге мы сделаем выбор направления внутрь или наружу случайным, то в итоге получим фрактальную кривую, которая может использоваться для моделирования береговых линий островов [16,254с].

На рисунке 3 приведена кривая Коха, построенная таким способом.

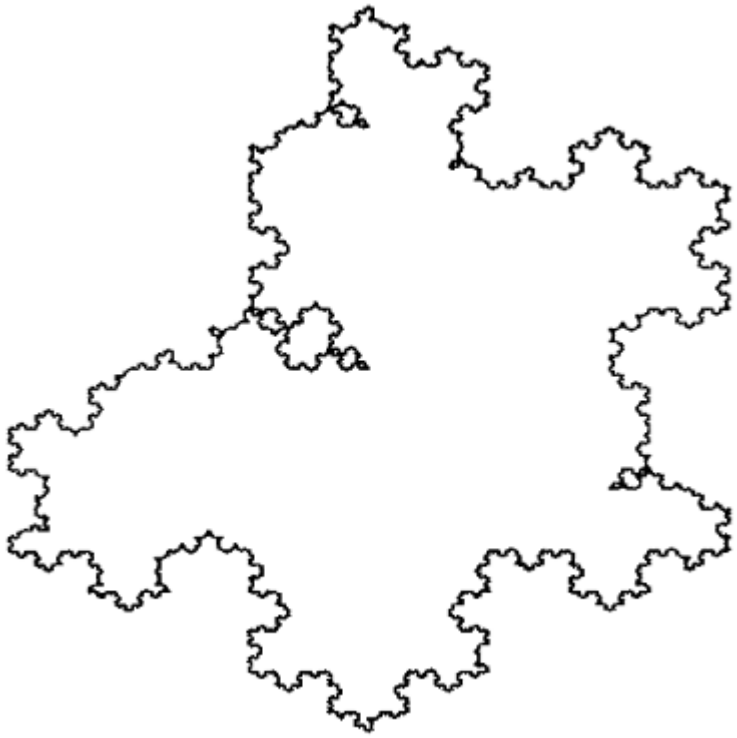

Рисунок 3 – рандомизированная кривая Коха

В 1827 году, когда шотландский ботаник Роберт Броун обнаружил, что маленькие частицы, взвешенные в жидкости, совершают непрерывное беспорядочное движение. В 1905 году Альберт Эйнштейн объяснил это движение хаотическими столкновениями с молекулами окружающей среды. Норберт Винер в 1923 году построил первую удовлетворительную с математической точки зрения модель выборочных реализаций.

На сегодняшний день по вопросу броуновского движения имеется обширная литература.

Простейшей дискретной аппроксимацией броуновского движения является одномерное случайное блуждание. В этом случае частица первоначально располагается в точке $x_0 = 0$ на прямой. Частица совершает единичный шаг вправо или влево в зависимости от случайного выбора. Случайное блуждание происходит итеративно. $\forall n = 1,2,3,...$ положим

$$x_n = x_{n-1} \pm 1. \qquad (8)$$

Более точным приближением к реальному броуновскому движению является замена шагов ±1 случайными величинами g_n, имеющими гауссовское

(нормальное) распределение. После первого шага частица находится в положении $x_1 = x_0 + g_1$, а после n шагов — в положении $x_n = x_{n-1} \pm g_n$.

Гауссовское случайное блуждание легко реализуется на компьютере. Единственная сложность — необходимость генератора гауссовских случайных чисел. Если имеется генератор равномерно распределенных на отрезке [0,1] случайных чисел, то вполне приемлемое приближение можно получить, используя формулу:

$$g = \sum_{i=1}^{12} u_i - 6 \qquad (9)$$

Утверждения о броуновском движении, относящиеся к одномерному случаю, имеют соответствующие аналоги для случая двух и большего числа измерений. Тогда Двумерный вариант броуновского движения можно определить по аналогии с одномерным случаем.

Таким образом, описанную модель броуновского движения можно использовать для создания фрактальных объектов, похожих на береговую линию.

2.4. Модифицированный муравей Лэнгтона

Муравей Лэнгтона – двумерный клеточный автомат, который также можно считать машиной Тьюринга с 2 символами и 4 состояниями. Данная структура представлена бесконечной плоскостью с разбиением на клетки, которые могут быть окрашены в белый или чёрный цвет. На одной из клеток находится муравей, имеющий направление взгляда. Муравей выполняет движение по простому алгоритму: если он находится на белой клетке, то он перекрашивает её в чёрный цвет, поворачивается на 90 градусов направо и перемещается на клетку, находящуюся перед ним; если он находится на чёрной клетке, то он перекрашивает её в белый цвет, поворачивает на 90 градусов налево и перемещается в клетку, находящуюся перед ним [17].

Несмотря на простоту описанных правил и произвольную конфигурацию окраски плоскости, после некоторого периода движения, имеющего произвольный вид, муравей начинает строить периодическую конструкцию, перемещаясь по диагонали. После этого момента его поведение уже не станет хаотичным. Иллюстрация данной «магистрали» приведена на рисунке 4.

Рисунок 4 – Магистраль муравья Лэнгтона.

Большой интерес вызывает возможность дальнейшей модификации правил, которым следует муравей, что приводит к различным рисункам, многие из которых представляют собой фрактальные объекты.

Наглядным способом модификации правил автомата является добавление нового состояния и переопределение уже имеющихся так, чтобы правило перехода из одного состояния в следующее было записано непосредственно перед правилом перехода из следующего состояния (кроме, конечно, последнего состояния, из которого осуществляется переход в первое), что также позволяет кратко описывать поведение муравья с помощью слова над алфавитом из двух (иногда трёх) букв.

Рассмотрим простейший пример модификации правил муравья Лэнгтона:

1. Если находится на белой клетке, повернуть направо и покрасить клетку в чёрный цвет (состояние 1)

2. Если находится на чёрной клетке, повернуть налево и покрасить клетку в красный цвет (состояние 2)

3. Если находится на красной клетке, повернуть направо и покрасить клетку в белый цвет (состояние 0).

Стоит отметить, что переход на клетку вперёд после смены цвета клетки подразумевается по умолчанию и не указывается в алгоритме.

На основании правил поворота из состояния строится строка из символов L (означает поворот налево) и R (означает поворот направо) длины, равной количеству состояний автомата. Так, для вышеописанной модификации правил строковым представлением является RLR. На рисунке 5 представлен результат движение муравья по правилам RLR [18].

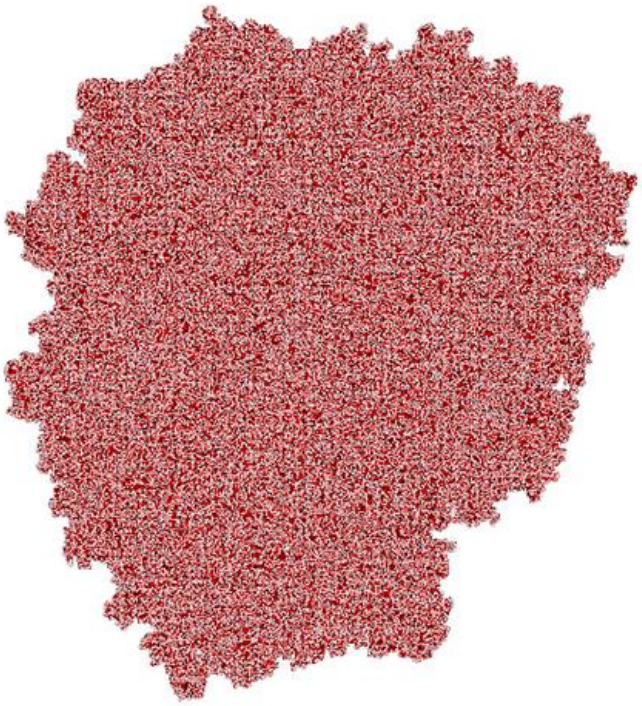

Рисунок 5 – визуальное представление движения муравья RLR

Путь муравья, следующего правилам LLRR, представляющий собой изображение, похожее на лилию, изображён на рисунке 6, а хронологическое сравнение путей муравья, следующего правилам RLLR, представляющих собой изображения почти прямоугольного фрактального объекта, изображён на рисунке 7.

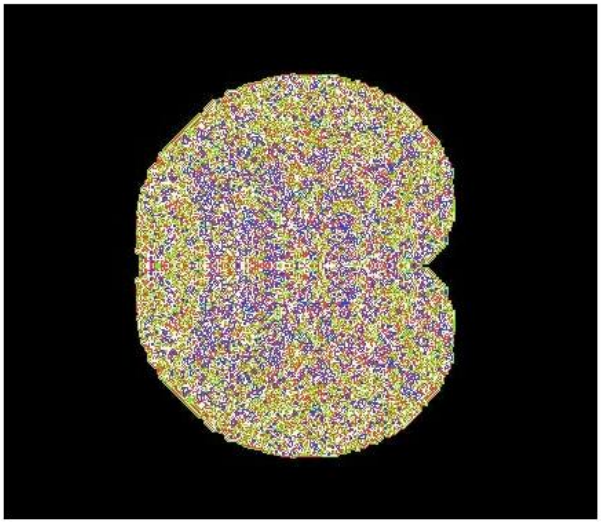

Рисунок 6 – визуализация пути муравья LLRR

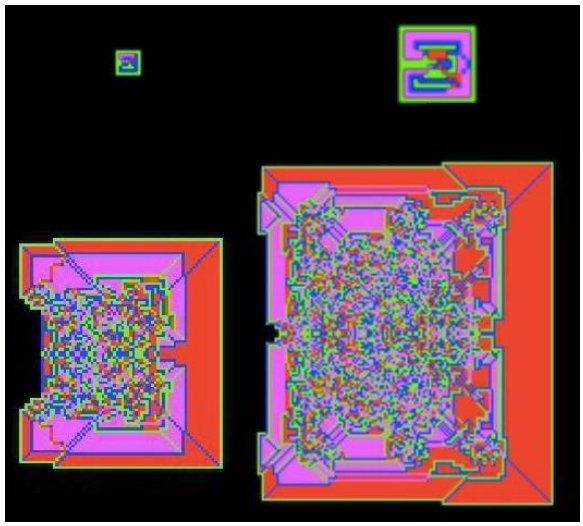

Рисунок 7 – хронология пути муравья RLLR

Аналогичная RLR картина получается при применении правил RRLRLRLLLRLLLRLRRLRRL, результат которого изображён на рисунке 8.

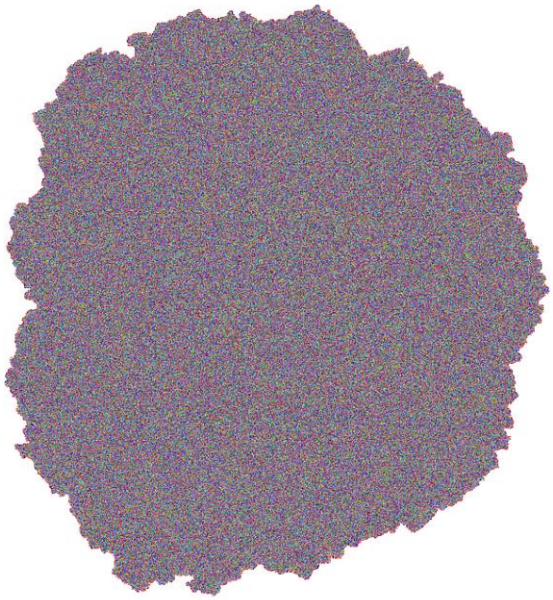

Рисунок 8 – визуализация пути муравья RRLRLRLLLRLLLRLRRLRRL

Однако некоторые из конфигураций правил дают в результате изображение, которое не является в полном объёме фрактальным объектом, например визуализация пути RLLRLRRLLRL, имеющая хаотическую область в центральной части, но также и несколько магистралей, одна из которых даже породила пирамидоподобную картину. Результат применения данного пути изображён на рисунке 9.

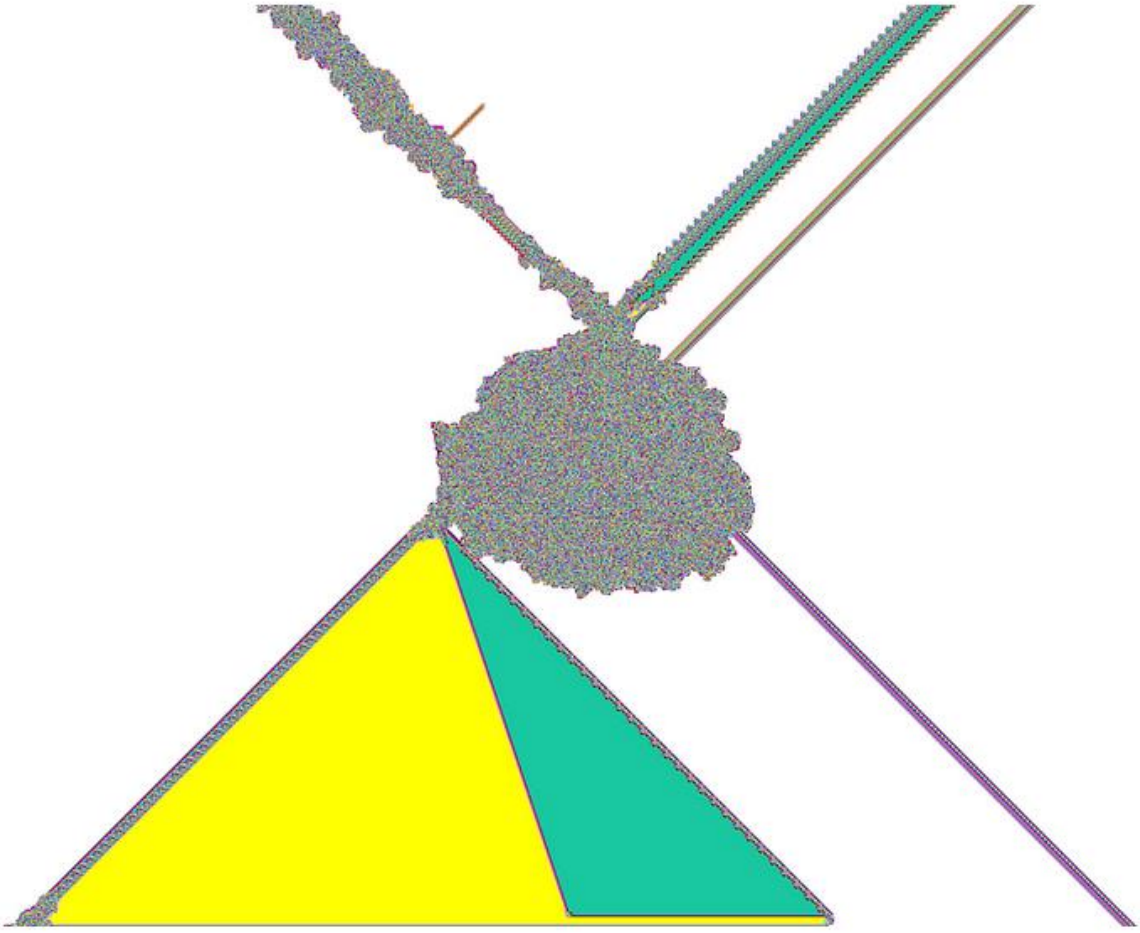

Рисунок 9 – визуализация пути муравья RLLRLRRLLRL

Иногда в постановку правил также добавляется возможность обозначения перехода, в результате которого муравей не повернётся и просто двинется вперёд, обозначаемая буквой N. Также существуют различные модификации самой плоскости, на которой движется муравей, самой распространённой из которых является гексагональная плоскость, наличие которой влечёт возможность поворотов на 60 и 120 градусов налево и направо, обозначаемые символами L1, L2, R1, R2.

Хотя данный автомат является детерминированным, в качестве алгоритма для построения стохастических фракталов можно использовать генератор произвольной строки определённой длины описанного формата, по которой будет работать автомат. Например, реализован такой генератор для автомата с 4096 состояниями [19].

Применительно же для построения стохастических фракталов, в том числе береговой линии, данную модель можно использовать, пренебрегая фактом наличия цвета в плоскости, используя только данные о передвижении муравья. Однако проблемой является возможность появления областей, нарушающих фрактальную структуру, как, например, на рисунке 9. Прогнозирование таких ситуаций может вызвать трудности из-за того, что даже классический муравей Лэнгтона до сих пор является нерешённой задачей.

2.5. Алгоритм, использующий диаграммы Вороного

Диаграммой Вороного конечного множества точек P на плоскости – такое разбиение плоскости, при котором каждая область этого разбиения образует множество точек, более близких к одному из $p(i) \in P$, чем к любому другому элементу P[20]. Для диаграммы Вороного имеется взаимно-однозначное соответствие с триангуляцией Делоне: одну структуру можно построить по другой за линейное время.

Рассмотрим построение фрактальной диаграммы Вороного, обладающей свойством самоподобия. Итерационный фрактальный процесс создаёт всё более плотные диаграммы Вороного с каждой итерацией. Фрактальная ячейка определяется как объединение ячеек диаграммы Вороного с одинаковым цветом. Используя больший набор точек, меньшая диаграмма Вороного изображается внутри каждой из исходных областей [21].

Иллюстрация процесса построения фрактальной диаграммы Вороного представлена на рисунке 10.

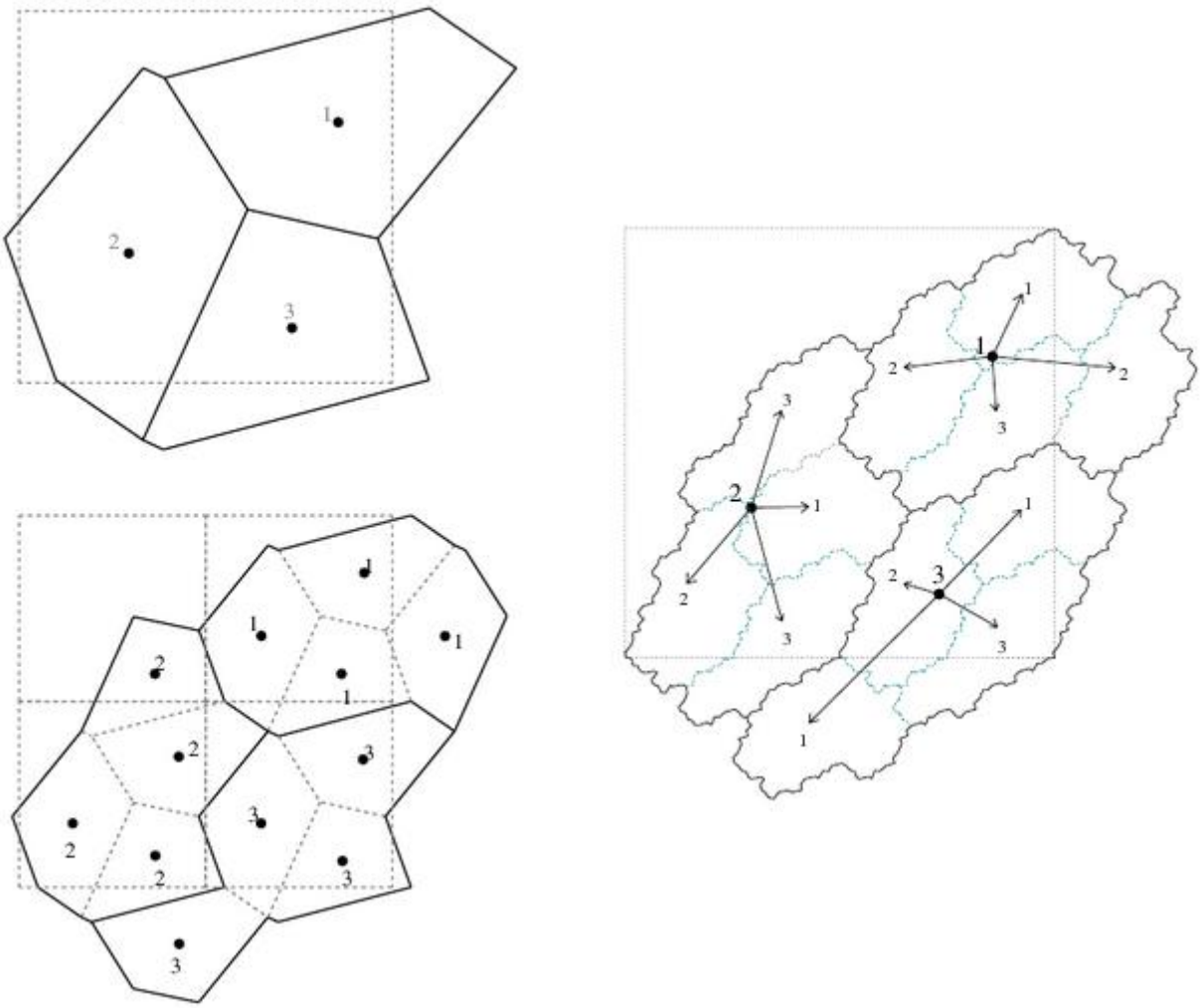

Рисунок 10 – итерационный процесс построения
фрактальной диаграммы Вороного

В результате такого построения получается фрактальный объект, представляющий собой грубые поверхности, такие как нерегулярные камни и сложные поверхности с произвольной топологией. Ещё одним возможным способом использования фрактальных диаграмм является создание искуственных молекул, имеющих связи во множестве точек, которые могут быть использованы в рамках химических симуляций. Данный метод возможно использовать для интерполяции береговой линии при условии корректного определения границ исходной диаграммы.

3. Программная реализация

3.1. Описание стека технологий

Для решения поставленной задачи был выбран язык Python на платформе Linux Mint. Плюсом языка является наличие множества библиотек для работы с данными, в том числе и данными спутниковых снимков, а также подробных руководств о работе с ними. Главным недостатком выбранного языка является невозможность распараллеливания кода, требующего больших вычислительных затрат. Это приводит к тому, что при обработке больших по объёму снимков, выполнение определённых этапов работы программы будет занимать очень много времени, в результате чего выполнение программы прерывается пользователем вручную.

Тестовые наборы данных представлены в отдельных папках, в каждой из которой содержатся снимки 1, 2, 3, 5 каналов, переименованные для краткости в «B1.TIF», «B2.TIF», «B3.TIF», «B5.TIF». Также возможно наличие снимка QA Band, переименованного в «QA.TIF». Выбор конкретного тестового набора данных для запуска осуществляется с помощью указания названия папки в качестве аргумента командной строки.

Для получения данных из файлов в формате .TIF используется библиотека rioxarray, являющаяся расширением библиотеки xarray для работы с геопространственными данными с помощью библиотеки rasterio.

Также для работы с данными используется библиотека numpy.

Для вывода растровых изображений используется модуль библиотеки earthpy.plot, содержащая обёртки функций библиотеки для визуализации данных matplotlib.

3.2. Загрузка данных и маскирование облачных областей снимков

Программа получает в качестве аргумента командной строки название папки с тестами и формирует на их основе список относительных путей к файлам снимков в указанном формате, и открывает каждый снимок, добавляя их в структуру *allBands*. В качестве аргумента командной строки программа может принимать ключ «*-ds*», а также, возможно, следующее за ним число, чтобы определить, какую часть изображения нужно отбросить слева и справа, сверху и снизу, что позволяет получить увеличенное изображение, обладающее меньшим размером, что критически важно для отображения цветного изображения на основе полученных снимков, так как в противном случае, используя снимки оригинального размера, возникает завершение программы из-за недостатка памяти. Переменная *clipSize*, устанавливаемая этим ключом, равная 4 по умолчанию, определяет, какая часть изображения будет отброшена по следующему принципу: берётся срез изображения без $\frac{1}{clipSize}$ части по каждому измерению в обоих направлениях. Так, при использовании ключа с значением 4, которое задано по умолчанию, будет получено увеличенное изображение с длиной и шириной, уменьшенными в два раза. Программа поддерживает значения *clipSize* больше 2 и меньше либо равные 6.

Листинг 1 – функция загрузки данных снимка

```
def open(path):
    if len(sys.argv) > 1:
        if "-ds" in sys.argv[2:]:  # означает downscale, обрезаем снимок до меньших размеров в центре
            band = rxr.open_rasterio(path, masked=True).squeeze()
            height = band.shape[0]
            width = band.shape[1]
            croppedBand = band.rio.clip_box(minx=band.x[int(width / clipSize)].values, miny=band.y[int(height / clipSize * (clipSize - 1))].values,  maxx=band.x[int(width / clipSize * (clipSize - 1))].values, maxy=band.y[int(height / clipSize)].values)
            return croppedBand
        return rxr.open_rasterio(path, masked=True).squeeze()
```

Пример отображения полученных снимков представлен на рисунке 11.

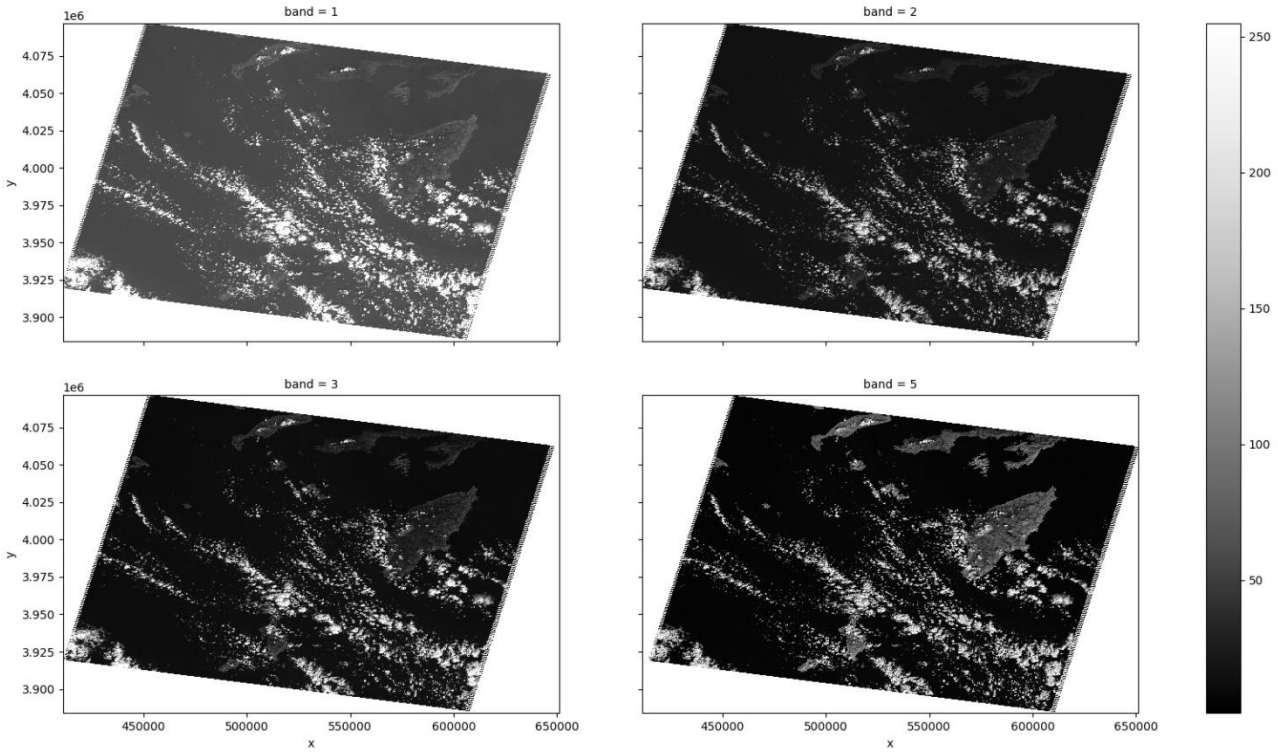

Рисунок 11 – полученные снимки из папки test3, не имеющей файла QA.TIF

На рисунке 12 показан пример цветного изображения, получаемого при запуске программы с увеличением изображения.

Рисунок 12 – полученное цветное изображение из папки test4 с использованием ключа «-*ds*» с числом 2,4

Облачные области нарушают корректность данных при построении водного индекса, поэтому следует исключить их с помощью наложения маски. В качестве аргумента командной строки программа может принимать ключ «*-qa*», получая который программа осуществит получение снимка QA Band из соответствующего файла в папке, построит маску облачных областей снимка с помощью классификации имеющихся пикселей и применит эту маску к каждому спектральному снимку. В результате этого преобразования все пиксели, лежащие в облачных областях, не имеют значения. На рисунке 13 показан пример маскирования облаков на двух разных снимках одной и той же географической территории, сделанных в разное время.

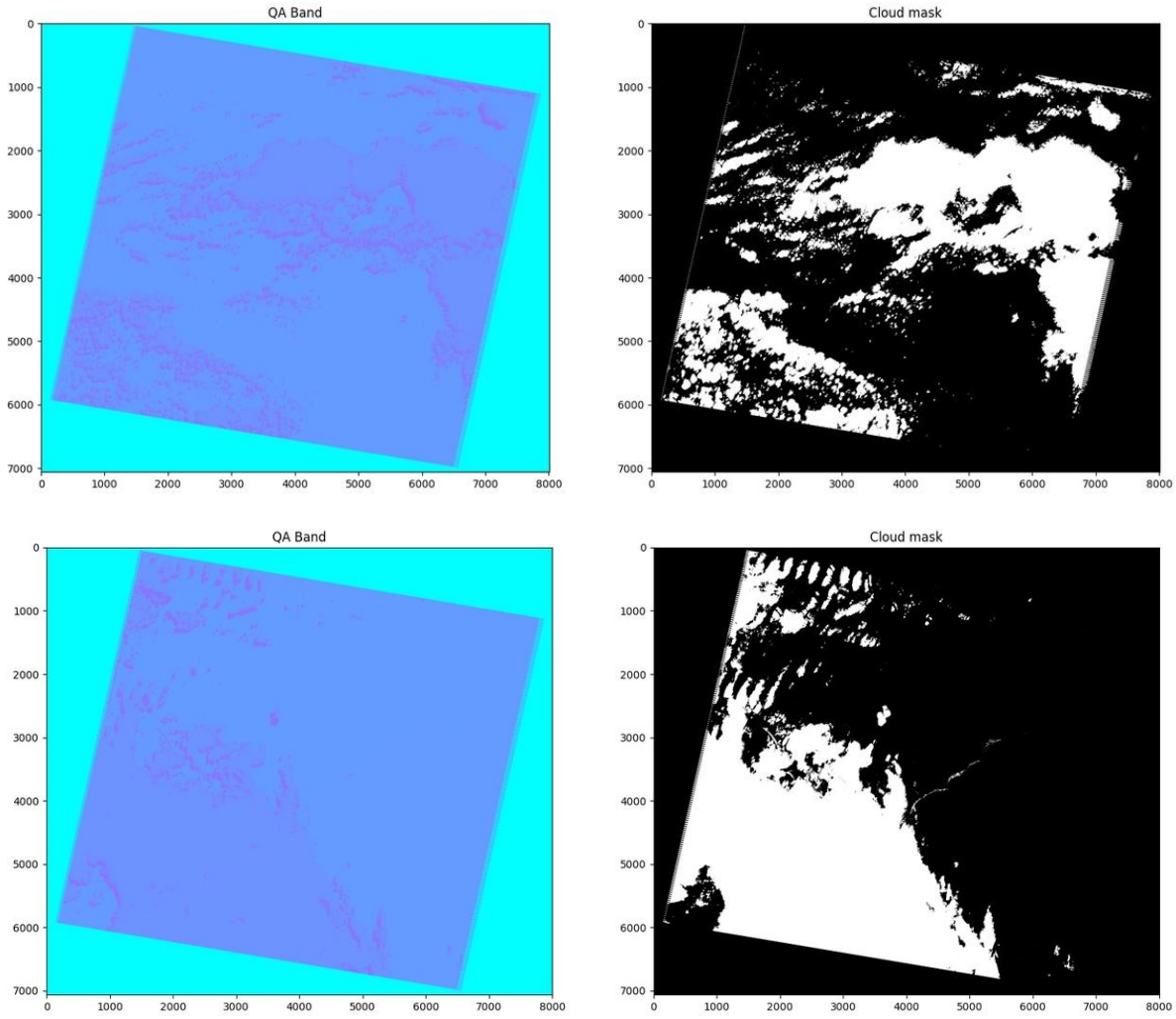

Рисунок 13 – полученные маски облачных областей для тестов из папок test1 и test2 с использованием ключа «*-qa*».

На рисунке 14 показаны цветные изображения областей двух снимков из рисунка 13 до и после маскирования облаков с использованием комбинации ключей «*-qa*» и «*-ds*» со значением по умолчанию.

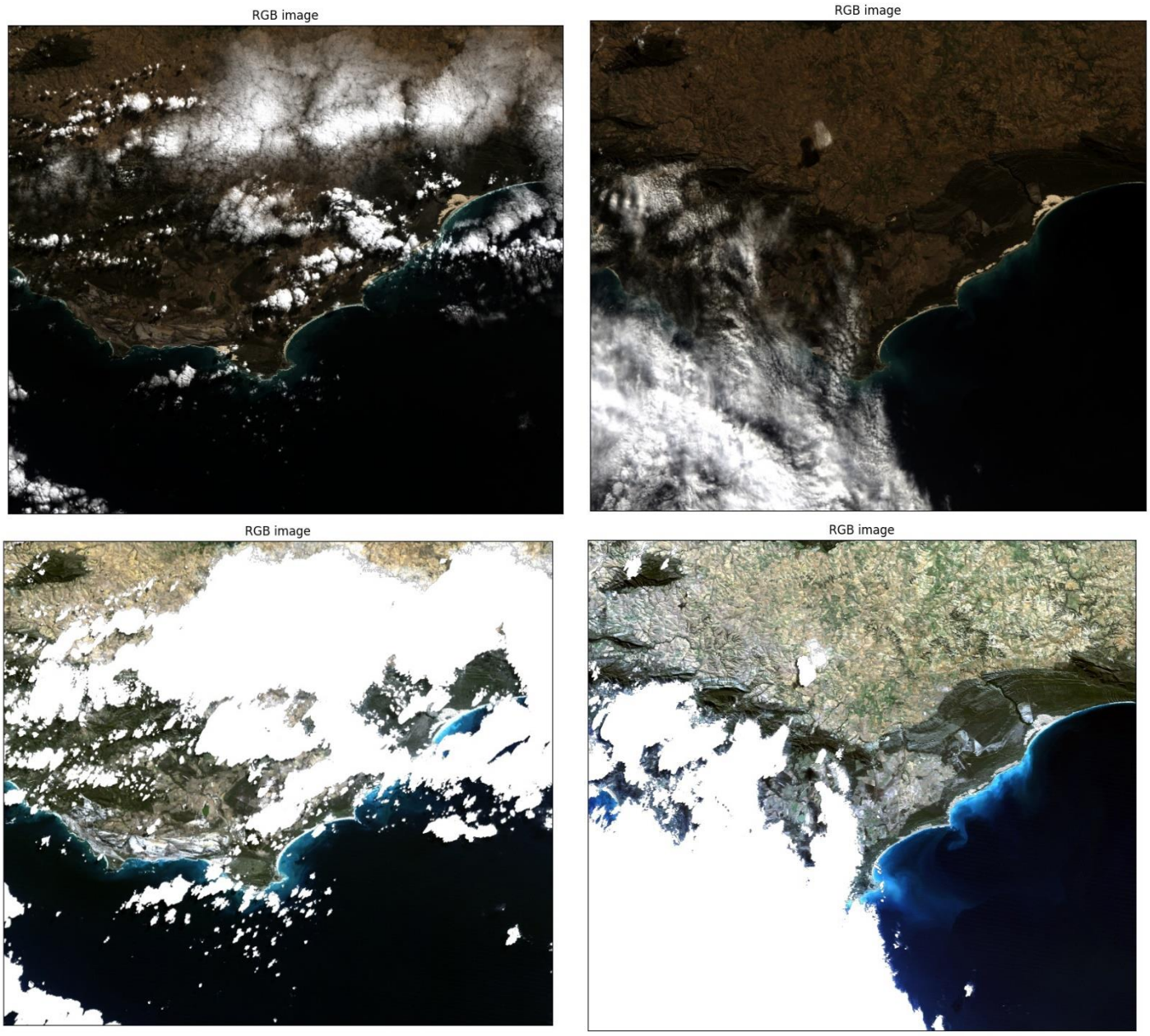

Рисунок 14 – цветные изображения, получаемые
комбинацией использования ключей.

Таким образом, наглядно продемонстрировано достаточно точное качество определение снимка QA Band и построенной на его основе маски облачных областей.

3.3. Построение водного индекса и выделение береговой линии

В качестве водного индекса по описанным ранее причинам и стремлением использовать меньшее количество памяти используется индекс MNDWI, считающийся по формуле (3). Представление этой формулы в коде:

```
MNDWI = ((allBands[1] - allBands[3]) / (allBands[1] + allBands[3])).values
```

Такие индексы многомерного используются из-за нумерации элементов с нуля, а также из-за того, что фактически пятый снимок является четвёртым по порядку из загруженных. Пример построенного индекса MNDWI для снимков, на одном из которых не проводилось маскирование облаков, а на другом проводилось, представлен на рисунке 15.

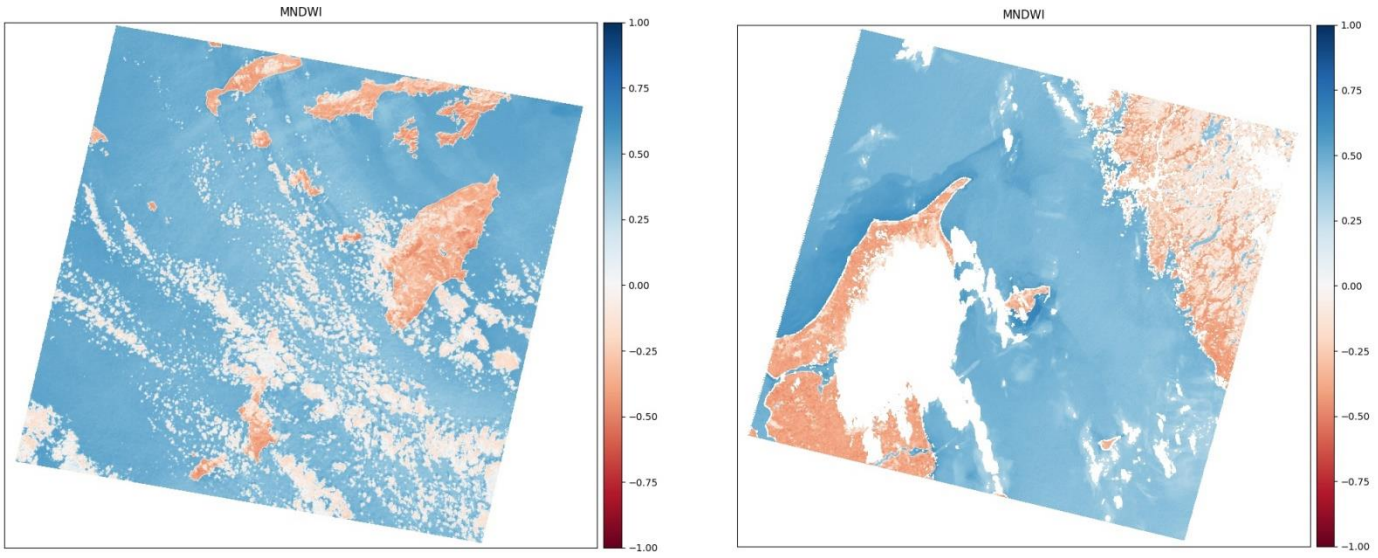

Рисунок 15 – построение индекса MNDWI.

Нетрудно заметить, что облачные структуры на левом снимке, также учитываемые индексом из-за отсутствия маскирования, имеют похожую на участки суши структуру, имея достаточно близкие к нулю и отрицательны значения, будут ошибочно классифицированы как участки суши и на них будет высчитываться береговая линия.

Можно отметить, что недостатком используемой цветовой схемы «*RdBu*» является то, что близкие к нулю значения имеют белый цвет, как и пиксели без значения, что может затруднить визуальное восприятие индекса.

Для выделения береговой линии было реализовано несколько алгоритмов, выбор которого регулируется заданием переменной *thresholdStep*, отвечающий за шаг цикла при первичном обходе индекса. Также условия классификации принадлежности определённого пикселя береговой линии, то есть близости значения к нулю, задаётся значениями *shorelineUpperThreshold* и *shorelineLowerThreshold*, равными 0,3 и -0,3 соответственно по умолчанию. Получение точек береговой линии при первичном обходе задаётся функцией getThreshold(matrix, minV, maxV, step).

Листинг 2 – функция getThreshold

```
def getThreshold(matrix, minV, maxV, step):
    tv = []
    for i in range(0, matrix.shape[0], step):
        for j in range(0, matrix.shape[1], step):
            if minV <= matrix[i][j] <= maxV:
                tv.append((i, j))
    return tv
```

Данная функция вызывается в виде getThreshold(MNDWI, shorelineLowerThreshold, shorelineUpperThreshold, thresholdStep) и возвращает список пар координат пикселей, подозрительных на принадлежность к береговой линии. Далее для каждого пикселя проверяется условие береговой линии, согласно которому, рядом с ним должны находиться как минимум по одному пикселю, которые можно классифицировать как часть суши и как часть водного объекта соответственно. Для этой проверки используется функция getNeighbors(matrix, i, j, len=1), возвращающая значения всех пикселей, находящихся на расстоянии $\leq$len от пикселя на позиции (i, j).

Листинг 3 – проверка условия береговой линии

```
shorelines = np.zeros((MNDWI.shape[0], MNDWI.shape[1]))
thresholdedValueCoords = getThreshold(MNDWI, shorelineLowerThreshold, shorelineUpperThreshold, thresholdStep)
startingIndexes = []
for index, tuple in enumerate(thresholdedValueCoords):
    x = tuple[0]
    y = tuple[1]
    neighbors = getNeighbors(MNDWI, x, y, 2)
```

```
if any(val > shorelineUpperThreshold for val in neighbors) and any(
 val < shorelineLowerThreshold for val in neighbors):
        shorelines[x][y] = 1
        startingIndexes.append((x, y))
```

Листинг 4 – функция getNeighbors

```
def getNeighbors(matrix, i, j, len=1):
   ns = matrix[max(i - len, 0):min(i + len + 1, matrix.shape[0]),
      max(j - len, 0):min(j + len + 1, matrix.shape[1])].flatten()
   return ns
```

В результате указанных преобразований формируется двумерное бинарное изображение shorelines, в котором значение пикселя равное 1 означает принадлежность данного пикселя в водном индексе к береговой линии.

Далее, в зависимости от значения *thresholdStep* может возникать необходимость проводить дополнительные преобразования для более полного выделения береговой линии. Если *thresholdStep* = 1, то условие принадлежности пикселя береговой линии уже заведомо проверено для всех пикселей индекса и дополнительных преобразований не требуется. Естетсвенно, этот способ является неэффективным по времени вычислений, так как осуществляется проверка каждого пикселя, что особенно тщетно в областях, имеющих обширные скопления суши или воды.

Так, например, для выделения береговой линии на индексе размером 7061 на 8011 пикселей (в сумме 56565671 пикселей) с шагом равным единице потребовалось 769,63 секунд, было отмечено 75226 пикселей. Для индекса размером 3731 на 4107 пикселей (в сумме 15323217 пикселей), полученного увеличением исходных снимков с помощью ключа «*-ds*», потребовалось 217,83 секунд, было отмечено 51179 пикселей.

Если же *thresholdStep* больше 1, то необходимо достроить береговую линию на основании уже полученных исходных точек. Стоит отметить, что в случае с большими значениями шага, береговые линии мелких географических структур могут быть не представлены ни одной из исходных точек, что приведёт к их потере.

Естественным алгоритмом, способном осуществить данное построение, является обход вдоль точек, соответствующих условию береговой линии, с внесением данных в бинарное изображение, начиная с исходных точек. Существенной проблемой при реализации данного алгоритма является глубина рекурсии, превышающая максимальный размер глубины рекурсии для языка Python. Реализация же данного обхода через очередь с переменной *thresholdStep* равной 16 показала неудовлетворительные результаты: количество точек очень медленно набирается в ходе итерационного процесса, что можно обосновать тем фактом, что при обходе из пикселя в очередь добавляются все соседние пиксели, в том числе и те, которые не являются пикселями береговой линии по условию, а также уже обработанные до этого, и такие пиксели обрабатываются многократно.

В качестве компромисса между описанных двух подходов к извлечению береговой линии был реализован алгоритм с шагом *thresholdStep* = 2, не использующего рекурсивный или итеративный обход. Вместо этого для всех соседних точек каждой исходной точки, полученной с помощью первичного обхода сеткой, проверяется условие береговой линии и, при соответствии ему, осуществляется внесение этой точки в бинарное изображение, а также проверка условия береговой линии для всех пикселей со значением, близким к нулю, рядом (*len* = 2) с новой точкой.

Листинг 5 – окончание выделения береговой линии, функция getThresholdedNeighbors(matrix, i, j, len=1), возвращающая индексы соседних пикселей, имеющих близкие к нулю значения, функция markShoreline(matrix, i, j, first=False) для *thresholdStep* = 2

```
for coords in startingIndexes:
    neighbors = getThresholdedNeighbors(MNDWI, coords[0], coords[1], 1)
    for n in neighbors:
        markShoreline(MNDWI, n[0], n[1], True)
```

```
def getThresholdedNeighbors(matrix, i, j, len=1):
    ns = []
    for dx in range(0 - len, 1 + len):
        for dy in range(0 - len, 1 + len):
            rangeX = range(0, matrix.shape[0])
            rangeY = range(0, matrix.shape[1])
            (newX, newY) = (i + dx, j + dy)
            if (newX in rangeX) and (newY in rangeY) and (dx, dy) != (0, 0) and
(shorelineLowerThreshold <= matrix[newX][newY] <= shorelineUpperThreshold):
                ns.append((newX, newY))
    return ns

def markShoreline(matrix, i, j, first=False):
        if thresholdStep == 2:
            neighbors = getNeighbors(matrix, i, j, 2)
            if any(val > shorelineUpperThreshold for val in neighbors) and
                                                any(val < shorelineLowerThreshold for val in neighbors):
                shorelines[i][j] = 1
                thrNeighbors = getThresholdedNeighbors(matrix, i, j, 2)
                for t in thrNeighbors:
                    if matrix[t[0]][t[1]] != 1:
                        ns = getNeighbors(matrix, t[0], t[1], 2)
                        if any(val > shorelineUpperThreshold for val in ns) and any(
                                val < shorelineLowerThreshold for val in ns):
                            shorelines[t[0]][t[1]] = 1
```

Реализация данной функции показала хороший результат. Для тех же индексов, на которых тестировалась проверка с шагом один, она даёт ускорение более, чем в 2 раза: для индекса размером 3731 на 4107 пикселей этап первичного обхода занял 55,43 секунды, а дополнительное построение заняло 51,68 секунд, что в сумме даёт 107,11 секунд; для индекса размером 7061 на 8011 пикселей этап первичного обхода занял 193,38 секунд, а дополнительное построение заняло 107,38 секунд, что в сумме даёт 300,76 секунд. Причём чем больше размер индекса, тем больше данная реализация выигрывает по скорости.

На рисунке 16 изображены некоторые береговые линии, полученные на основе индекса MNDWI. Стоит отметить, что при большой длине и ширине изображений может возникнуть трудность рассмотрения элементов построенного бинарного изображения, поэтому была реализована функция, создающая новое бинарное изображение, в котором в определённой области каждого пикселя все пиксели также закрашены, что утолщает видимые точки, но в данных примерах эта функция не используется.

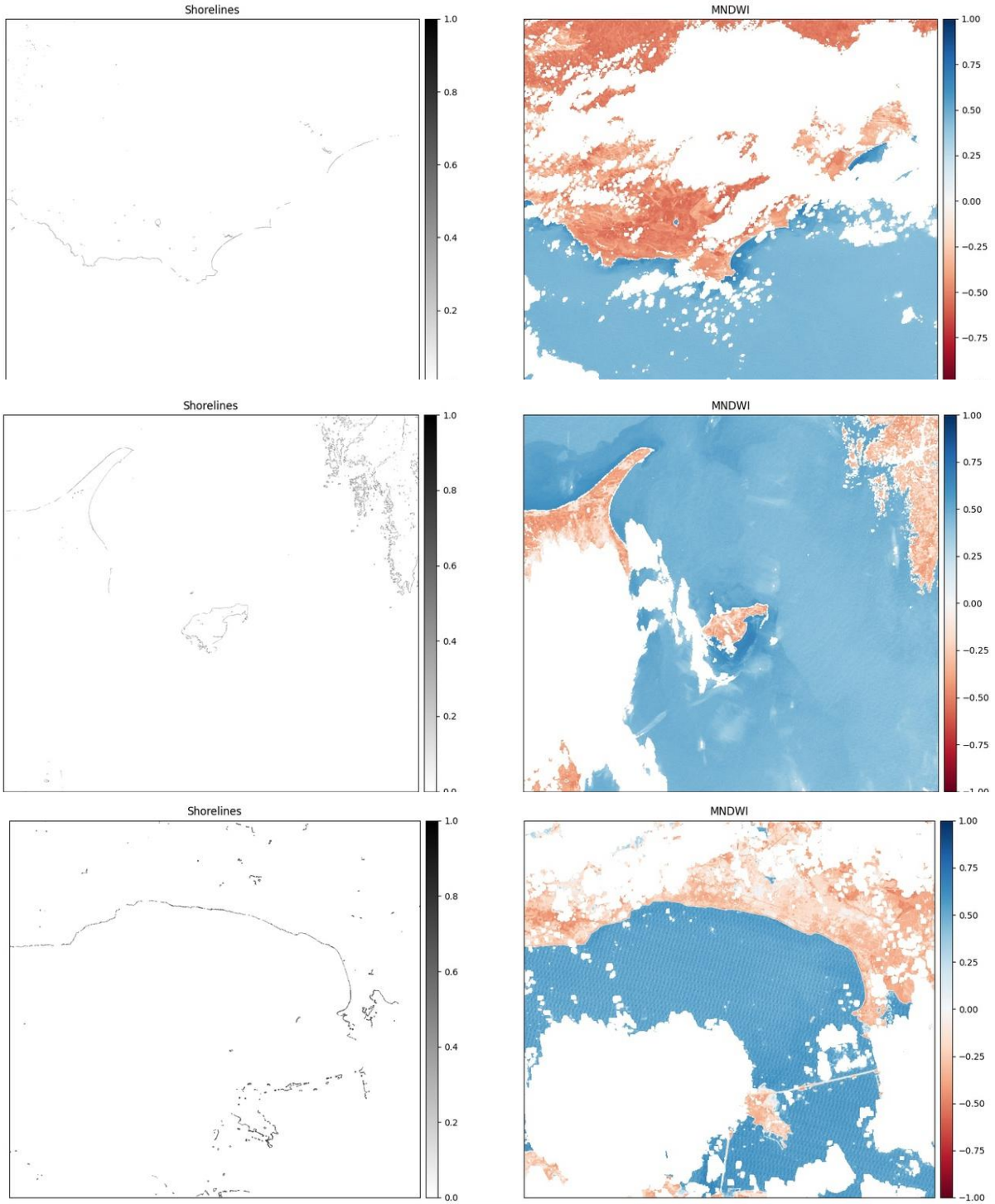

Рисунок 16 – примеры выделения береговой линии

3.4. Реализация рандомизированного метода на основе систем итерируемых функций

В теоретических сведениях было установлено, что фрактальное сжатие является времязатратным процессом. Поэтому перед тем, как реализовывать этот процесс, производится уменьшение разрешения полученного изображения береговых линий. При тестировании на водном индексе размером 2355 на 2671 пикселей общее время, потраченное на выделение береговой линии, составило 29,69 секунд, а время, потраченное на сжатие изображения, составило 729,80 секунд. При этом на восстановление изображения потребовалось 22,29 секунд.

Для уменьшения разрешения используется функция shrink(image, factor).

Листинг 6 – функция shrink

```
def shrink(image, factor):
    res = np.zeros((image.shape[0] // factor, image.shape[1] // factor))
    for i in range(res.shape[0]):
        for j in range(res.shape[1]):
            res[i, j] = np.mean(image[i * factor:(i + 1) * factor, j * factor:(j + 1) * factor])
    return res
```

Побочным эффектом данного преобразования является преобразование бинарного изображения в изображение в режиме градации серого, что проиллюстрировано на рисунке 17.

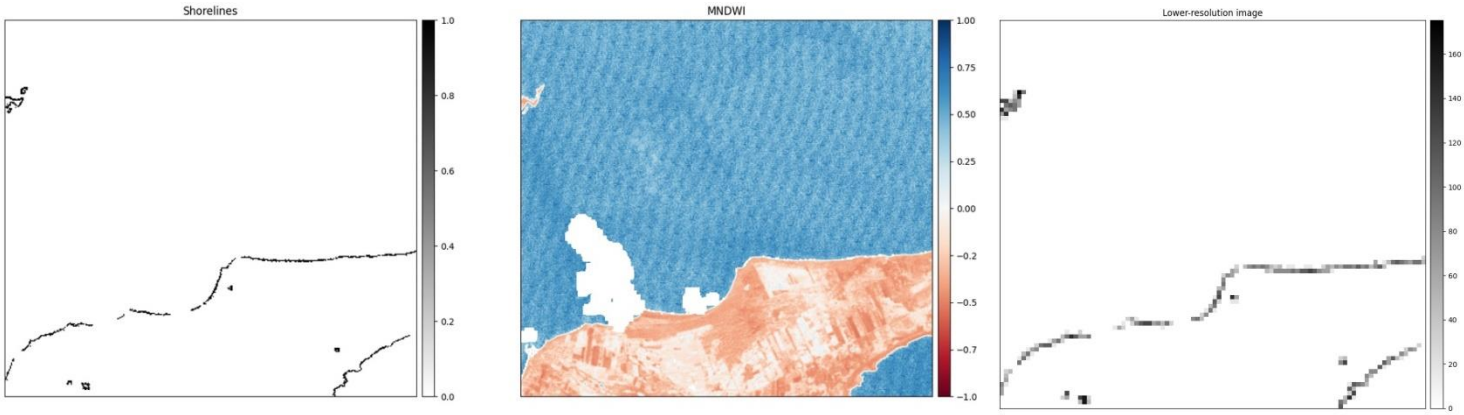

Рисунок 17 – понижение разрешения изображения береговых линий

Далее осуществляется сжатие с помощью функции compress(image, srcSize, destSize, steps) и масштабирование с помощью функции decompress(transformations, srcSize, destSize, steps, iters=8).

Листинг 7 – функции compress и decompress

```
def compress(image, srcSize, destSize, steps):
    transformations = []
    segmentation = getSegmentation(image, srcSize, destSize, steps)
    print("Starting compression of image after segmentation")
    for i in range(image.shape[0] // destSize):
        transformations.append([])
        for j in range(image.shape[1] // destSize):
            transformations[i].append(None)
            minDist = float('inf')
            block = image[i * destSize: (i + 1) * destSize, j * destSize: (j + 1) * destSize]
            for k, l, dir, angle, tr in segmentation:
                dist = np.sum(np.square(block - tr))
                if dist < minDist:
                    minDist = dist
                    transformations[i][j] = (k, l, dir, angle)
    return transformations


def getSegmentation(image, srcSize, destSize, steps):
    print("Starting segmentation of image")
    dirs = [-1, 1]
    angles = [0, 90, 180, 270]
    transforms = [[dir, angle] for dir in dirs for angle in angles]
    sizeFactor = srcSize // destSize
    segments = []
    for i in range((image.shape[0] - srcSize) // steps + 1):
        for j in range((image.shape[1] - srcSize) // steps + 1):
            seg = shrink(image[i * steps: i * steps + srcSize, j * steps: j * steps + srcSize], sizeFactor)
            dir, angle = random.choice(transforms)    # недетерминированный выбор преобразования
            segments.append((i, j, dir, angle, transform(seg, dir, angle)))
    return segments
```

```
def decompress(transformations, srcSize, destSize, steps, iters=8):
    sizeFactor = srcSize // destSize
    height = len(transformations) * destSize
    width = len(transformations[0]) * destSize
    iterations = [np.random.randint(0, 256, (height, width))]
    image = np.zeros((height, width))
    for iter in range(iters):
        print("Doing decompression iteration", iter)
        for i in range(len(transformations)):
            for j in range(len(transformations[i])):
                k, l, dir, angle = transformations[i][j]
                S = shrink(iterations[-1][k * steps: k * steps + srcSize, l * steps: l * steps + srcSize], sizeFactor)
                D = transform(S, dir, angle)
                image[i * destSize: (i + 1) * destSize, j * destSize: (j + 1) * destSize] = D
        iterations.append(image)
        image = np.zeros((height, width))
    return iterations

def transform(image, dir, angle):
    return rotate(flip(image, dir), angle)



def flip(image, dir):
    return image[::dir, :]

def rotate(image, angle):
    res = np.copy(image)
    for _ in range(angle, 0, -90):
        res = np.rot90(res)
    return res
```

В качестве преобразований используются комбинация поворотов и отражений. В данной реализации используется 8 итераций СИФ, хотя на практике изображение сходится и за меньшее количество шагов. На рисунке 18 представлены два результата восьми итераций на основе изображения, полученного на рисунке 17, а также исходное бинарное изображение береговой линии, изображение с уменьшенным разрешением и бинарное изображение, полученное в результате преобразования изображения получившегося в результате итерационного процесса.

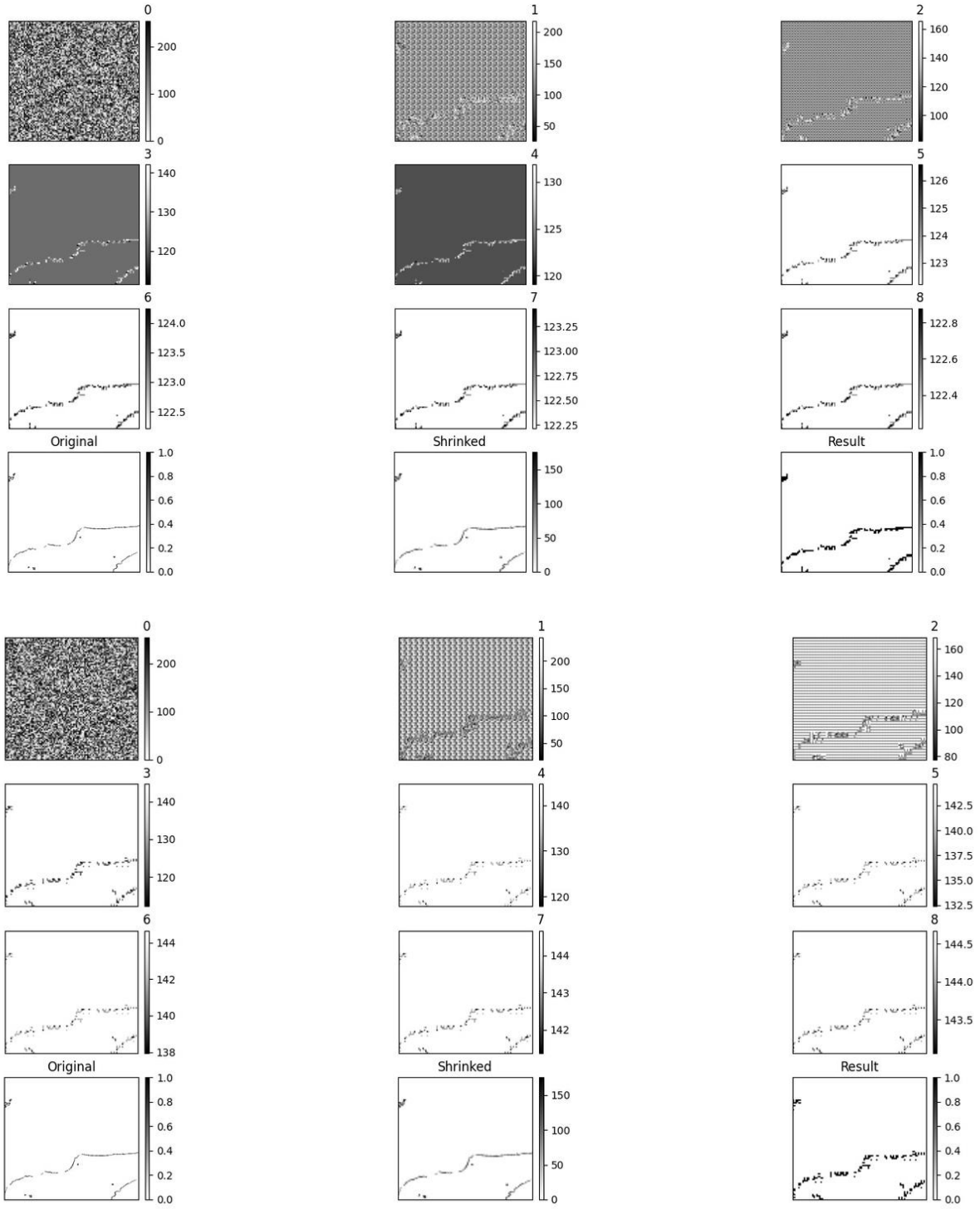

Рисунок 18 – два результата работы программы

Стоит отметить, что различие шагов итерации и итогового результата мотивировано недетерминированной натурой стохастического алгоритма. Также при рассмотрени этого рисунка нагляден тот факт, что зачастую система сходится раньше, чем за 8 итераций.

Рассмотрим результат используемого алгоритма, изображённого на рисунке 19, поближе.

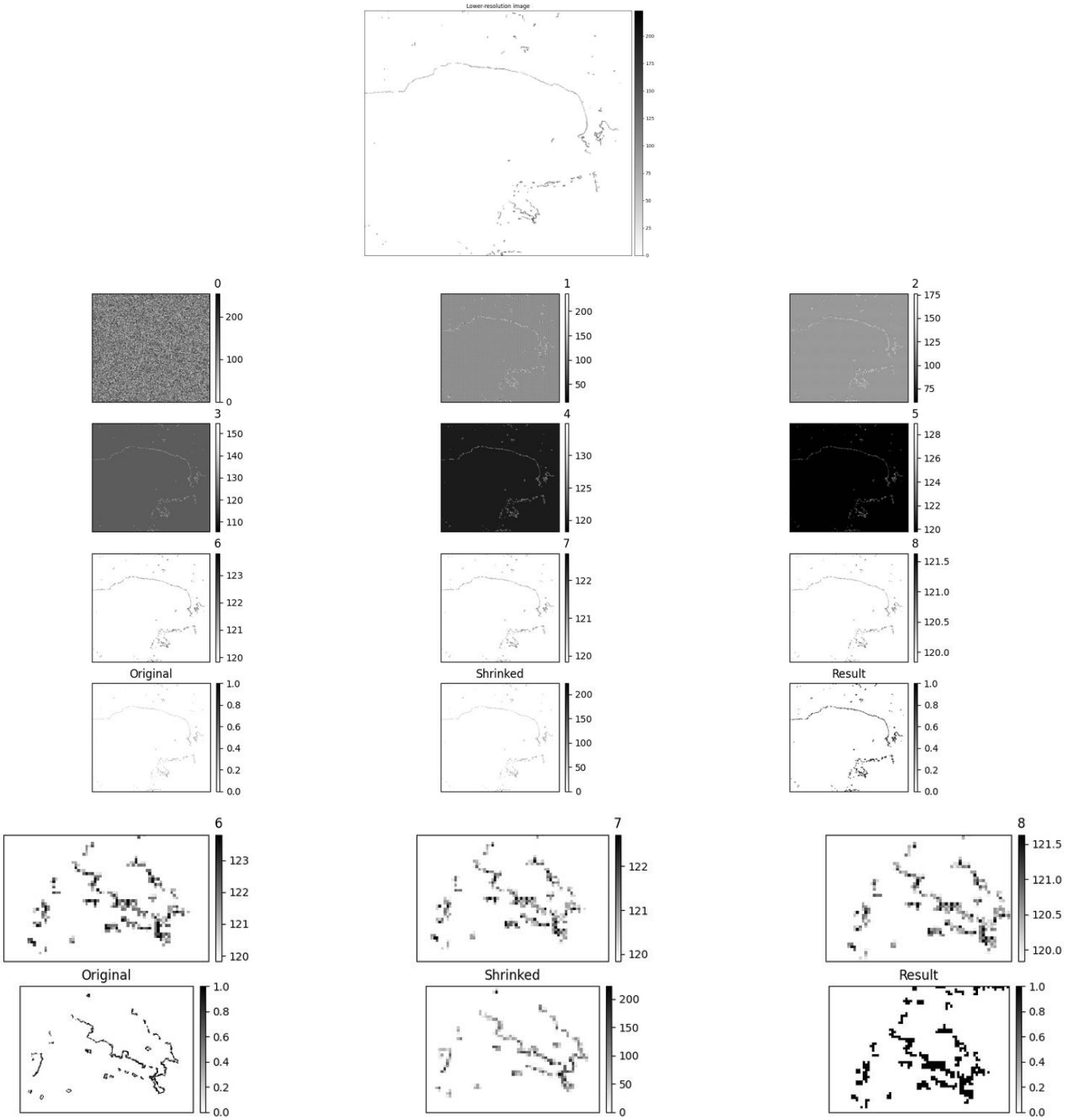

Рисунок 19 – увеличенный результат работы программы.

Рассмотрев последние итерации системы и результат и сравнив его с исходным изображением с уменьшенным разрешением, можно сделать вывод, что реализованный алгоритм вполне достоверно воссоздал его и изобразил новые детали рельефа.

Заключение

В ходе работы были изложены основные теоретические сведения о предметной области, изучены и реализованы на практике этапы анализа береговых линий на основе снимков дистанционного зондировния Земли, рассмотрены методы фрактальной интерполяции и их применимость к полученным береговым линиям, реализован рандомизированный метод на основе систем итерируемых функций.

Использование алгоритмов фрактального масштабирования изображений – это достаточно новая область, на данный момент не существует общепринятого стандартного решения данной задачи. Эти современные алгоритмы зачастую очень сложные, но позволяют получить многообещающие результаты.

Реализованный алгоритм достаточно точно работает на небольших изображениях, но для больших изображений фрактальное кодирование занимает длительное время, что затрудняет тестирование и получение результатов. Модификация алгоритма фрактального кодирования, в частности, усложнение метода разбиения изображения на самоподобные области, позволит сделать данный процесс более быстродейтсвующим, при этом возможно даже повышение качество получаемых в результате изображений.

Список литературы

1. Водный Кодекс Российской Федерации [Электронный ресурс] http://pravo.gov.ru/proxy/ips/?docbody=&nd=102107048

2. Картография цифровая. Термины и определения. [Электронный ресурс] https://docs.cntd.ru/document/1200009569

3. Landsat Science [Электронный ресурс] https://landsat.gsfc.nasa.gov/

4. Дистанционное зондирование Земли при эколого-геологических исследованиях 4. Системы съемки спутников типа Landsat [Электронный ресурс] http://www.geol.vsu.ru/ecology/ForStudents/4Graduate/RemoteSensing/Lection04.pdf

5. Описание инструментов спутников Landsat-4 и Landsat-5 [Электронный ресурс] https://earth.esa.int/eogateway/missions/landsat-4-and-landsat-5#instruments-section

6. Landsat Thematic Mapper (TM) Collection 2 (C2) Level 1 (L1) Data Format Control Book (DFCB) [Электронный ресурс] https://d9-wret.s3.us-west-2.amazonaws.com/assets/palladium/production/s3fs-public/atoms/files/LSDS-1415_Landsat4-5-TM-C2-L1-DFCB-v3.pdf

7. Морозова В. А. Расчет индексов для выявления и анализа характеристик водных объектов с помощью данных дистанционного зондирования / В. А. Морозова // Современные проблемы территориального развития : электрон. журн. – 2019. – № 2. [Электронный ресурс] https://terjournal.ru/2019/id85/

8. Виды водных индексов и их применение [Электронный ресурс] https://innoter.com/articles/vidy-vodnykh-indeksov-i-ikh-primenenie/

9. Иудин Д.И., Копосов Е.В. Фракталы: от простого к сложному [Электронный ресурс] https://www.nngasu.ru/unesco/resources/Fractals.PDF?utm_source=google.com&utm_medium=organic&utm_campaign=google.com&utm_referrer=google.com

10. Применение компьютерной графики для решения экономических и

инженерных задач : учебное пособие / А.М. Горюнова, Т.В. Омельченко, П.Н. Омельченко, Д.А. Муслимов; Оренбургский гос. ун-т. – Оренбург : ОГУ, 2018. [Электронный ресурс] http://elib.osu.ru/bitstream/123456789/12876/1/%D0%9F%D1%80%D0%B8%D0%BC%D0%B5%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5%20%D0%BA%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%BE%D0%B9%20%D0%B3%D1%80%D0%B0%D1%84%D0%B8%D0%BA%D0%B8.pdf

11. Учебное пособие по курсу "Численные методы в оптике". 3. Интерполяция [Электронный ресурс]: http://aco.ifmo.ru/el_books/numerical_methods/lectures/glava3.html

12. T.Janani, Shennes Mathew, S.Deepa. SURVEY OF FRACTAL IMAGE COMPRESSION TECHNIQUES [Электронный ресурс] https://acadpubl.eu/jsi/2018-118-20/articles/20a/86.pdf

13. Óscar Belmonte1, Sergio Sancho1and, José Ribelles1. Multiresolution Modeling Using Fractal Image Compression Techniques [Электронный ресурс] https://www.researchgate.net/publication/220862187_Multiresolution_Modeling_Using_Fractal_Image_Compression_Techniques

14. Chi Wah Kok, Wing Shan Tam. Fractal Image Interpolation: A Tutorial and New Result [Электронный ресурс] https://pdfs.semanticscholar.org/d27e/94037ce61ecb16bcaf2260f6509d6f864927.pdf

15. Robert Małysz, Convergence of trajectories in fractal interpolation of stochastic processes. [Электронный ресурс] https://www.sciencedirect.com/science/article/pii/S0960077905004698

16. Р. М. Кроновер. Фракталы и хаос в динамических системах. Основы теории. [Электронный ресурс] http://pzs.dstu.dp.ua/ComputerGraphics/bibl/fractal.pdf

17. Муравей Лэнгтона — загадочный клеточный автомат. [Электронный ресурс] https://habr.com/ru/post/599275/

18. Michel Khalife, Yussef Shehadeh. Creation of Fractal Imagery Based on the Template of Langton’s Ant. [Электронный ресурс]

https://medium.com/@yussefshehadeh/creation-of-fractal-imagery-based-on-the-template-of-langtons-ant-9eeba353ac9b

19. Randomly-Generated 4096-State Langton's Ant Patterns. [Электронный ресурс] https://adasba.github.io/adasba-portfolio-site/cellular_automata/langtons_ant.html

20. И.Захаркин. Диаграмма Вороного и её применения. [Электронный ресурс] https://habr.com/ru/post/309252/

21. Tomasz Dobrowolski. Generating Fractal Tiles using Voronoi Diagrams [Электронный ресурс] https://www.researchgate.net/publication/4263590_Generating_Fractal_Tiles_using_Voronoi_Diagrams