Федеральное государственное бюджетное образовательное учреждение

высшего образования

Санкт-Петербургский государственный университет

Ионина Ольга Анатольевна

**РАЗРАБОТКА ГЕНЕРАТОРОВ ФРАКТАЛЬНЫХ ТЕКСТУР ДЛЯ СОЗДАНИЯ КОНТЕНТА В СРЕДЕ PROCESSING**

Выпускная квалификационная работа по направлению подготовки

035300/50.03.01 «Искусства и гуманитарные науки»

Профиль подготовки «Сложные системы в природе и обществе»

Научный руководитель: Князева Ирина Сергеевна,

кандидат физ.-мат. наук,

доцент СПбГУ

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ подпись, дата

Санкт-Петербург

2018

СОДЕРЖАНИЕ

Введение …………………..……………………………………………………....3  
Глава 1. Обзор предметной области ……………………………………………..4  
1.1. Генеративное искусство и креативное программирование .……………..4  
Креативное программирование …………………..……….……………………..5  
1.2. Среда разработки Processing ……………………………..…………………..7  
Глава 2. Фрактальная графика ……...……………………………..……………..9

2.1. Фрактальные множества ………………………………….…………………9

Фрактальная размерность ………………………………………………….….....9

Cамоподобие и скейлинг ………………………………………………..…..…..11

2.2. L-система……………………….....……………………………………..…..13

Формальное определение описания DOL системы и ее операций………...…..14

Turtle интерпретация L систем…………………….…………………………….15

2.3. Результаты применения алгоритма L-система для генерации контента в Processing ………………..……………………………………………………….17  
2.4. Агрегация ограниченная диффузией ………………………………….…..23

Формальное описание алгоритма DLA ……………………………….……..…24

Классическая модель DLA……………………………………………...…….…24

2.5. Результаты применения алгоритма L-система для генерации контента в Processing ………………………………………………………………………...26

Заключение……………….………………………………..………………….….30

Библиография ………………………………...….……………..…………....…..31

**Введение**

Данная работа посвящена применению таких алгоритмов, как  L-система и агрегация ограниченная диффузией, при создании графического контента, обладающего фрактальными свойствами, в среде Processing.

В настоящее время наблюдается рост интереса к компьютерной графике, основанной на компьютерном коде, и также известной как генеративная графика. Она широко применяется в различных прикладных областях, таких как архитектура, веб-дизайн, промышленный дизайн, графический дизайн и визуализация данных, видеопроизводство, а также в искусстве. Графические редакторы с пользовательским интерфейсом, такие как Adobe Photoshop, CorelDraw, AutoCad, Adobe After Effects и др. интуитивно понятны для использования, однако их функционал ограничен заранее определенным набором инструментов программы, а создание сложных паттернов требует много времени. Компьютерный код требует знания синтаксиса и логики программирования, однако перед пользователем открывается широкий спектр возможностей. Изображение создается не вручную, а путем перевода визуальной идеи в набор правил, а затем ее реализации на языке программирования. Такая программа может не только создавать изображение, но и генерировать визуальные миры с возможностью изменения параметров.

Алгоритмы генерации фракталов широко применяются в компьютерной графике и позволяют создавать большое разнообразие текстур и структур. Использование математических методов для создания изображений объединяет науку с искусством. Бенуа Мандельброт, заметил, что «благодаря фрактальной геометрии мы узнаем о том, что некоторые из наиболее сухих и холодных разделов математики скрывают за внешней суровостью целый мир чистой пластичной красоты, доселе неведомой».[3]

В задачи данной работы вошло изучение специфики среды Processing, изучение вышеупомянутых алгоритмов, написание кода для графической анимации с использованием этих алгоритмов.

**Глава 1. Обзор предметной области**

* 1. **Генеративное искусство и креативное программирование**

Филлип Галантер, исследователь генеративного искусства, преподаватель и художник предлагает следующее определение:

«Генеративное искусство относится к любой практике искусства, где художник использует такие средства, как набор правил естественного языка, компьютерную программу, машины или другие процессуальные изобретения, которое приводятся в движение с некоторой степенью автономии, которая способствует или приводит к завершению произведения искусства.» [9]

Хаос, фракталы, степенные законы и другие разделы науки о сложных системах повлияли на искусство. Художественный отклик на сложность охватывает различные сферы искусства, включая графику, музыку, скульптуру, инсталляции, видео, анимацию. Сложность в искусстве – это вопрос содержания, а не сложности техники. Науку о сложных системах можно рассматривать как источник вдохновения, революционную интеллектуальную силу, которая имеет последствия для других дисциплин, таких как искусство и философия.

Филипп Галантер пишет [10] об использовании генеративных методов в различных сферах:

* Электронная музыка и алгоритмическая композиция. Музыканты используют всевозможные генеративные процессы, клеточные автоматы, фракталы, L-системы и рандомизацию для создания музыкальных партитур и тонкой модуляции исполнения и тембра.
* Компьютерная графика и анимация. Исследования компьютерной графики в течение десятилетий внесли свой вклад в развитие генеративного искусства. Примерами генеративной графики являются шум Перлина для синтеза изображений дыма, огня, использование L-систем для увеличения реалистичности изображений растений и для генерации с помощью этого алгоритма целых лесов и долин, а также использование физических моделирования для создания анимаций, которые отображают поведение реального мира, и которые не требуют от аниматора кропотливой отрисовки каждой детали. Эти усилия принесли результаты, которые выходят далеко за рамки исследовательского сообщества.
* Промышленный дизайн и архитектура. Практика проектирования всегда включает использование параметрического программного обеспечения для создания новых форм в предметном дизайне и строительстве.

**Креативное программирование**

«Многие считают, что программирование предназначено только для людей, которые хорошо разбираются в математике и других технических дисциплинах. Одна из причин, по которой программирование остается в области этого типа личности состоит в том, что технически мыслящие люди обычно создают языки программирования». [18]

Генеративная графика – это создание изображений с использованием кода. Изображение создается не вручную, а путем перевода визуальной идеи в набор правил, а затем ее реализации на языке программирования. Такая программа может не только создавать изображение, но и генерировать визуальные миры с возможностью изменения параметров. [11]

Ранним альтернативным языком программирования был Logo, разработанный в конце 1960-х годов Сеймуром Папертом как язык программирования для детей. Logo позволил детям программировать многие различные носители, включая роботизированную черепаху и графические изображения на экране.

Визуальные языки программирования (также называемые графическими языками программирования) обеспечивают альтернативный способ мышления с помощью кода. Написание программы с на языке визуального программирования чем-то похоже на создание диаграмм вместо написания текста. [17]

На сегодняшний день существует множество платформ визуального программирования в сфере искусств – Max MSP, Pure Data , vvvv, TouchDesigner, OpenFramework, Isadora и другие. Некоторые из них не используют синтаксис языков программирования как таковой, но используют уже готовые патчи и так называемы ноды-кабели, соединяющие их.

Эти программные среды вызывают энтузиазм у тысяч музыкантов и визуальных художников, которые используют его как базу для создания аудио и визуального программного обеспечения для создания уникального контента. Точно так же как графические пользовательские интерфейсы открыли возможности для вычисления для миллионов людей, альтернативные среды программирования позволяют художникам и дизайнерам работать непосредственно с программным кодом.

* 1. **Cреда разработки Processing**

Processing – это программное обеспечение с открытым исходным кодом, среда разработки и язык программирования, созданный в 2001 году выпускниками группы Джона Маеды Computation and Aestethics в медиалаборатории Массачусетского технологического университета. Processing распространяется бесплатно и подходит для различных операционных систем, таких как GNU/Linux, Mac OS X, Windows, Android и ARM.

Processing относится к концепциям программного обеспечения к принципами визуальной формы, движения и взаимодействия. Он объединяет язык программирования, среду разработки и методику преподавания в унифицированную систему. Processing был создан для обучения основам компьютерного программирования в визуальном контексте. Студенты, художники, дизайнеры и исследователи используют его для обучения, прототипирования и производства. [18]

Язык Processing - это язык текстового программирования, специально разработанный для генерации и обработки изображений. Processing стремится к достижению баланса между простотой и расширенными возможности. Начинающие могут начать писать свои собственные программы уже после нескольких вводных уроков, более продвинутые пользователи могут использовать библиотеки с дополнительными функциями. Система облегчает изучение компьютерной графики и методов взаимодействия, включая векторную и растровую графику, обработку изображений, цветовые модели, события мыши и клавиатуры, сетевые коммуникации и объектно-ориентированное программирование. Библиотеки легко расширяют возможности Processing для генерации звука, отправки и получения данных в разнообразных форматах, а также для импорта и экспорта 2D и 3D форматов.

Processing основывается на языках Java и JavaScript, но имеет упрощенный синтаксис, однако весь синтаксис этих языков может быть использован и соответственно скомпилирован. Некоторые характеристики:

* позволяет криптографию высокого уровня для обмена важной личной информацией;
* приложения могут использовать интернет-протоколы;
* позволяет динамически распределять оперативную память компьютера;
* поддерживает международные символы;
* имеет высокую производительность с помощью оптимизаторов;
* простота изучения.

Основными лингвистическими элементами, используемыми в Processing, являются константы, переменные, процедуры, классы и библиотеки, а основными операциями являются арифметические, логические, комбинаторные, реляционные и классификационные, организованные под конкретные грамматические и синтаксические правила. Эти элементы используют вычислительный характер компьютера, в то же время обеспечивают создание логических шаблонов. Таким образом, можно утверждать, что Processing предполагает, что дизайн может быть создан посредством манипулирования арифметическими и логическими шаблонами, и может стать результатом этих манипуляций.[12, 20,22]

**Глава 2. Фрактальная графика**

**2. 1. Фрактальные множества**

Французский математик Жан Перрен в 1906 году заявил, что «кривые не имеющие касательных, являются общим правилом, а гладкие кривые, такие как окружность, - интересным, но весьма частным случаем». Такие кривые являются недифференцируемыми и называются фракталами. [6]

Термин фрактал (от латинского fractus – дробный) предложил Бенуа Мандельброт, основываясь на теорию дробной размерности Хаусдорфа, предложенной в 1919 году:

Фракталом называется множество, размерность Хаусдорфа-Безиковича которого строго больше его топологической размерности. [3]

Однако позже Мандельброт сузил свое определение и предложил называть фракталом структуру, состоящую из частей, которые в каком-то смысле подобны целому. [5]

Федер считает, что первое определение при всей своей точности ограничительно и не исключает многие фракталы, встречающиеся в физике, и предлагает нестрогие определения термина, что для наших целей достаточно. Он предлагает рассматривать фракталы, как множество точек вложенных в пространство.

**Фрактальная размерность**

Длина гладкой кривой равна произведению прямолинейных отрезков, умещающихся на эту кривую на длину этого отрезка :, где N – количество отрезков.

Когда длина шага стремится к нулю, длина стремится к конечному пределу – длине L рассматриваемой кривой. В случае фракталов произведение стремится к бесконечности, так как если стремится к нулю, то можно учесть все более мелкие извивы фрактала. Однако это стремление к бесконечности происходит по некоторому определенному степенному закону от . Иными словами, существует некоторый критический показатель , называемый размерностью Хаусдорфа, при котором произведение останется конечным. При показателях меньших чем произведение обращается в бесконечность, а при больших, чем , - стремится к нулю. Тогда справедливо следующее определение размерности Хаусдорфа:

[6]

В качестве примера возьмем кривую Коха (рис. 1). На средней трети некого отрезка прямой (инициатор) строится равносторонний треугольник. В результате длина увеличится на 4/3 от исходного отрезка – инициатора, и будет называться генератором. Если повторить процедуру еще раз на каждой средней трети прямолинейных отрезков, мы получим кривую длиной . Повторяя процедуру много раз, мы можем получить кривую, длина которой будет стремиться к бесконечности. При n-й итерации число шагов будет пропорционально , а выбранный шаг , тогда размерность будет равняться:

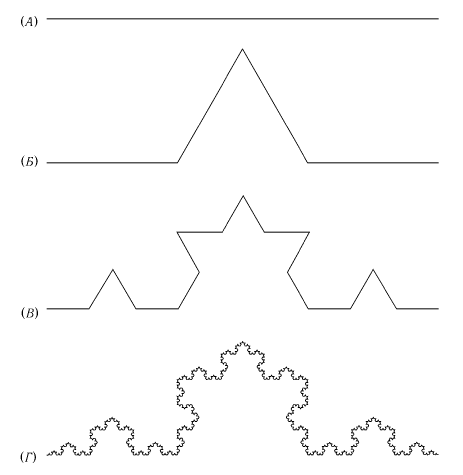


Рис. 1 Кривая Коха. (A) Инициатор, (Б) генератор, (В) cледующая итерация, (Г) аппроксимация высокого порядка к кривой. [6]

Таким образом, размерность Хаусдорфа-Безиковича для кривой Коха больше ее топологической размерности, мы заключаем, что кривая Коха есть фрактальное множество. Её размерность лежит между 1 и 2 эта кривая является большим нежели одномерный объект, но и не дотягивает до двумерного, так как не покрывает какую-либо область на плоскости. [5,6]

**Самоподобие и скейлинг**

Размерность Хаусдорфа-Безиковича называют также размерностью подобия. [2]

Шредер пишет[6]:«Самоподобный объект выглядит неизменным и после увеличения, и после уменьшения его размеров. Так в турбулентных потоках крупные вихри рождают меньшие, те в свою очередь, – еще меньшие и так(почти) ad infinitum. В общем случае, одно из наиболее заметных следствий самоподобия – объекты с необычайно тонкой структурой, которые ныне называются фракталами».

В качестве примера, рассмотрим также кривую Коха. На рисунке 2 изображены несколько множеств из последовательности которая аппроксимирует кривую Коха . Множество является пределом этой рекурсивной последовательности.

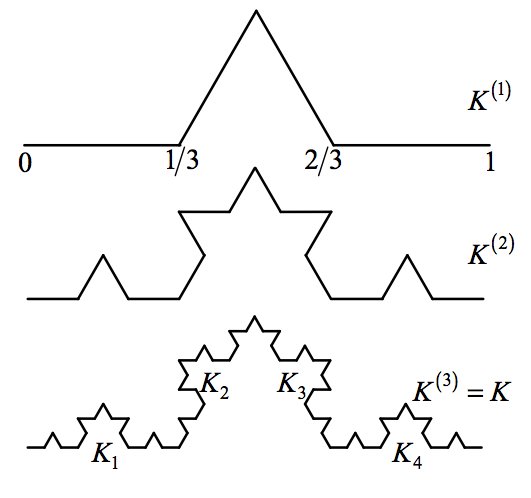


Рис. 2. Кривая Коха.

Из рис. 2 видно, что . Заметим, что каждое  конгруэнтно  с масштабным коэффициентом . Более строго, существует отображение , где каждое представляет собой композицию сдвига, поворота и масштабирования, таким образом мы можем переписать выражение .

Кривая Коха самоподобна. Она состоит из мелких деталей, которые являются точными маленькими репликами полной формы. Если взять, скажем, левую треть изображения и увеличить её в 3 раза, то получившееся изображение будет идентично полной кривой Коха. [8]

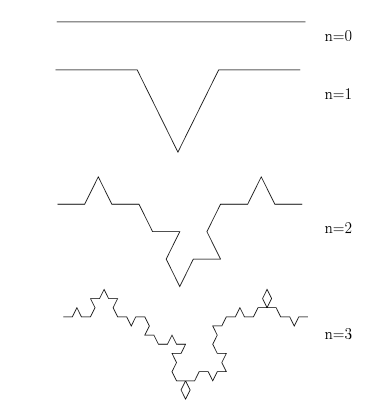


Рис. 3. Cтатистически самоподобная кривая Коха. [8]

Если при выполнении итераций произвольно выбирать направление для изгиба в каждом сегменте линии, кривые будут отличаться друг от друга и не являются точно самоподобными (рис. 3). Небольшая копия кривой при увеличении очень похожа на полную кривую, но это не точная копия. Это свойство неточной самоподобия иногда называют статистическим самоподобием. Идея состоит в том, что такой фрактал сделан не из точных копий самого себя, а из более мелких частей, которые имеют те же статистические свойства, что и целое. [8]

**2.2. L-система**

L-система, как новый механизм перезаписи строк, была предложена Аристидом Линденмайером в 1968 году как теоретическая основа для изучения развития простых многоклеточных организмов, которая в дальнейшем также стала применяться в исследованиях высших растений.

Другое направление исследований сделал в 1984 году Элви Рэй Смит, который предложил L-системы в качестве инструмента для синтеза реалистичных изображений растений и указал на связь между L-системами и концепцией фракталов, введенными Бенуа Мандельбротом.  [15]

Простейшая разновидность L-систем - так называемая DOL система (детерминированная, контекстно независимая). Она включает в себя *алфавит*, *аксиому* и *правила*.

Предположим строки(слова) состоящие из двух букв *a* и *b*. Каждая буква ассоциируется с правилом переписывания. Правило *a* → *ab* означает, что буква *a* будет заменена выражением *ab,* а правило *b* → *a* означает, что буква *b* будет заменена на *a*. Процесс переписывания начинается с выделенной строки, называемой аксиомой. Предположим это будет строка, которая содержит единственную букву *b*. На первом шаге переписывания аксиома *b* будет заменена по правилу *b* → *a.* На следующем шаге *a* будет заменена по правилу *a* → *ab.* Слово *ab* состоит из двух букв, каждая из которых будет заменена по установленным правилам при следующем переписывании. Так *а* будет заменена *ab*, *b* заменена *a*, результатом будет слово *aba*. Таким же образом слово *aba* будет заменено на *abaab*, что при следующем переписывании даст *abaababa*, затем *abaababaabaab* и так далее. (рис. 4)

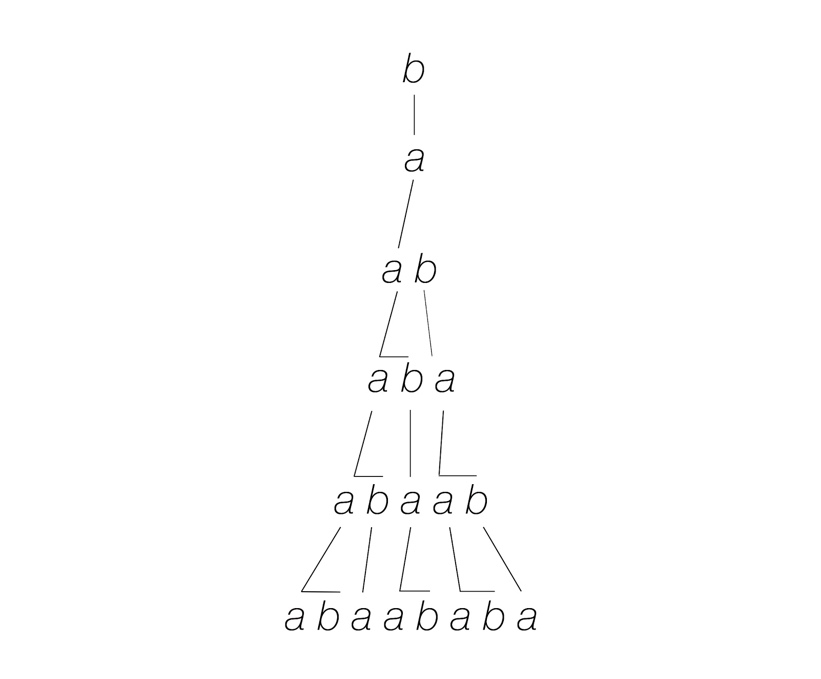


Рис. 4. Схема переписывания простейшей DOL системы.

**Формальное определение описания DOL системы и ее операций**

Предположим V означает алфавит, V\* - множество всех слов из V, и V+ множество всех непустых слов из V. Строка L системы упорядочивается тремя переменными G = V, ω, P, где V это алфавит системы, ω ∈ V + непустое слово, называемое аксиомой, и  P ⊂ V × V ∗ как конечный набор правил. Правило (*a*, χ) ∈ P записывается как *a* → χ. Буква *a* и слово χ называется предшественником и преемником этого правила соответственно. Предполагается, что для каждой буквы *a* ∈ V существует хотя бы одно слово χ ∈ V\* при котором *a* → χ. Если для предшественника *a* ∈ V не задано явное правило , предполагается, что правило идентичности *a* → *a* принадлежит множеству P. Контекстно независимая OL-система является детерминированной (DOL-система) тогда и только тогда, когда для каждого *a* ∈ V существует ровно один χ ∈ V\* такой, что *a* → χ. [16]

Для моделирования более высоких установок требуется более сложная интерпретация графических методов L-систем. Первые результаты в этом направлении были опубликованы в 1974 году Фритерсом и Линденмайером, а также Хогвегом и Хеспером. В обоих случаях L-системы использовались в основном для определения ветвящейся топологии смоделированных растений. Геометрический аспекты, такие как длины сегментов линии и значения угла были добавлены на следующем этапе обработки. Результаты Hogeweg и Hesper были впоследствии расширены Смитом, которые продемонстрировали потенциал L-систем для реалистичного синтеза изображений. [16]

**Turtle интерпретация L систем**

Turtle графика – это принцип организации компьютерной графики, основанный на метафоре воображаемой черепахи, которая перемещается по экрану и оставляет за собой нарисованные следы заданного размера и цвета. Как правило ассоциируется с языком LOGO, но применяется и в других языках. [23]

Положение системы определяется тремя переменными *x, y, α,* где *x* и *y* определяют координаты черепахи, а угол *α* определяет направление в котором она двигается. Имея размер шага d и коэффициент приращения угла δ, черепаха может реагировать на команды представленные следующими символами (рис. 5a):

**F** Двигайся вперед на шаг длиной . Положение системы изменится с , где и . А также между точками и рисуется линия.

**f** Двигайся вперед на шаг длиной без рисования линии.

**+** Поверни налево на угол . Следующее положение . Положительная ориентация углов – против часовой стрелки.

**−** Поверни направо на угол . Следующее положение .

Учитывая строку ν, начальное состояние черепахи *(x0, y0, α0)* и фиксированный набор параметров и , Turtle интерпретацией ν будет фигура (набор линий) нарисованные черепахой в ответ на строку ν. (рис. 5b)

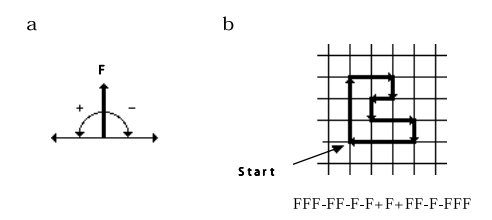


Рис. 5 a) Turtle интерпретация символов F, +, − .

б) Интерпретация cтроки. Приращение угла δ равно 90◦. [16]

В Processing мы можем имитировать turtle графику c помощью функций translate(), rotate(), line(). Ниже представлены символы и соответствующие им команды, предлагаемые Д. Шиффманом. [21]

|  |  |  |
| --- | --- | --- |
| **F** | **line(0,0,0,len); translate(0,len);** | Рисование линии заданной длины.  Перемещение на отрезок заданной длины. |
| **+** | **rotate(angle);** | Поворот на заданный угол против часовой стрелки. |
| **-** | **rotate(-angle);** | Поворот на заданный угол по часовой стрелке. |
| **[** | **pushMatrix();** | Сохранение текущего положения. |
| **]** | **popMatrix();** | Возврат в сохраненное поолжение. |

Наша работа основана на базовом коде, описывающем L-систему, представленный как приложение к книге Д. Шиффмана «Nature of code». [21] Пример доступен в приложении Processing в разделе File/Examples/Contributed Examples/The Nature of Code/chp08\_fractals/NOC\_9\_09\_LSystem.

Код, описывающий L-систему организован в три класса:

**Rule** – класс, задающий предшественника и преемника в строке правил.

**LSystem** – класс, описывающий итерацию нового поколения.

**Turtle** – класс для управления чтением предложения L-системы и выполнения его инструкций для рисования на экране.

Наши эксперименты касаются графической интерпретации L-системы, а также предложений, описывающих их. На рис. 6 представлен результат исходного кода.

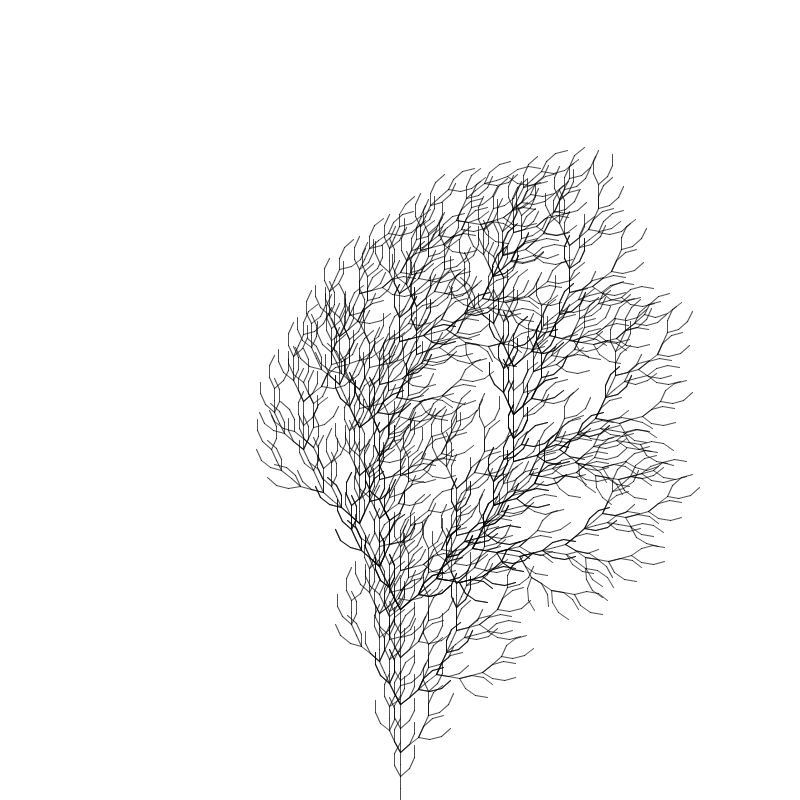


Рис. 6. Результат базового кода, описывающего L-систему (Приложение 1).

**2.3. Результаты применения алгоритма L-система для генерации контента в Processing**

В ходе нашего исследования мы провели множество экспериментов, касающихся графического отображения L-системы, а также описывающих их предложений, и выбрали некоторые интересные на наш взгляд результаты.

Прежде всего введем высокое разрешение экрана:

size(1200, 600);

Изменим правило на:

ruleset[0] = **new** Rule('F', "F[[F[F-][+F]]]++F--F++F--F");

Применим функцию random() к углу поворота, таким образом система будет носить стохастический характер, а также мы получим анимацию:

**else** **if** (c == '+') {

rotate(random(theta));

}

**else** **if** (c == '-') {

rotate(random(-theta));

}

Зададим частоту кадров:

frameRate(0.5);

Дополнительно к рисованию линий добавим рисование кривых Безье.

beginShape();

vertex(30, 20);

bezierVertex(x, y, len, 0, 35, 20);

endShape();

А также изменим коэффициент роста c 0,5 на 0,3 :

turtle.changeLen(0.3);

На рис. 7, 8 представлены результаты изменений.



Рис.7. Результат для предложения F[[F[F-][+F]]]++F--F++F--F с кривыми Безье и случайным углом поворота.

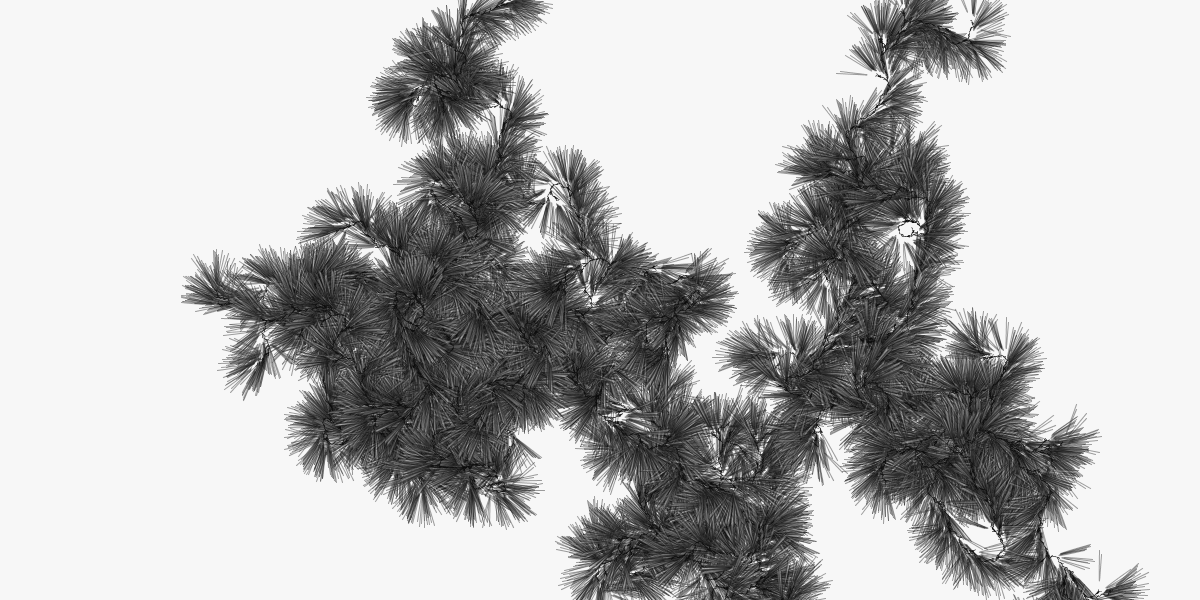
****

Рис.8. Результат для предложения F[[F[F-][+F]]]++F--F++F--F с кривыми Безье и случайным углом поворота.

Применив цвет, получим новый вариант:

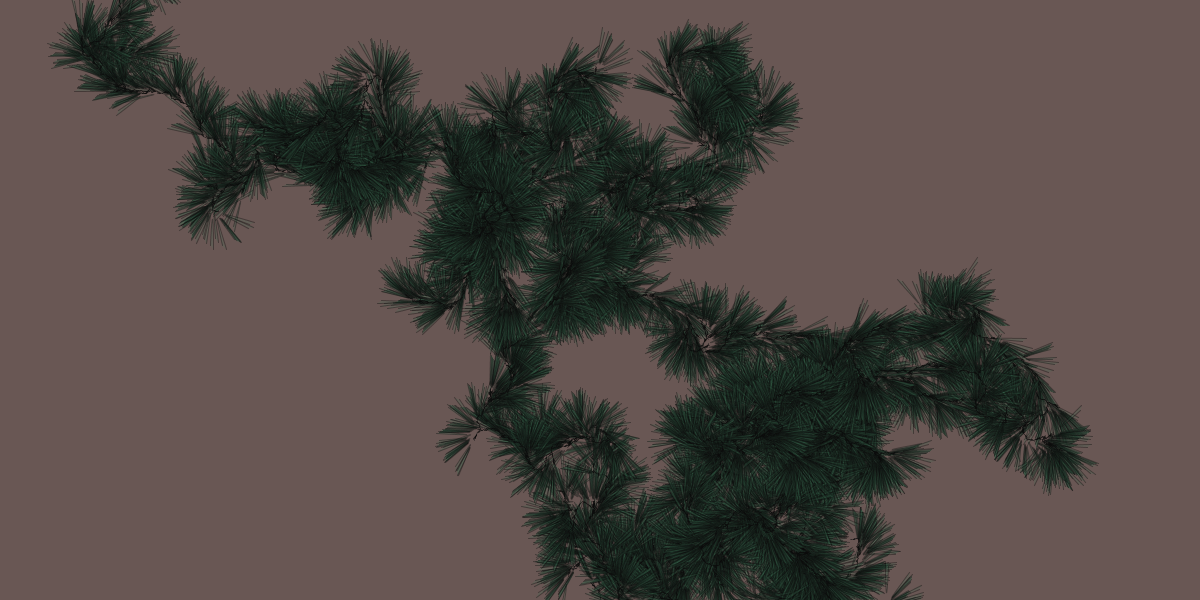
****

Рис.9. Результат для предложения F[[F[F-][+F]]]++F--F++F--F с кривыми Безье и случайным углом поворота

Изменив цвет, форму кривой Безье, а также предложение, получим следующий паттерн, изображенный на рис.10.

Форма кривой:

beginShape();

vertex(30, 20);

bezierVertex(0, 0, random(len), len\*20, 30, 20);

endShape();

Предложение:

ruleset[0] = **new** Rule('F', "FF[F-][+F]++F--F++F--F");

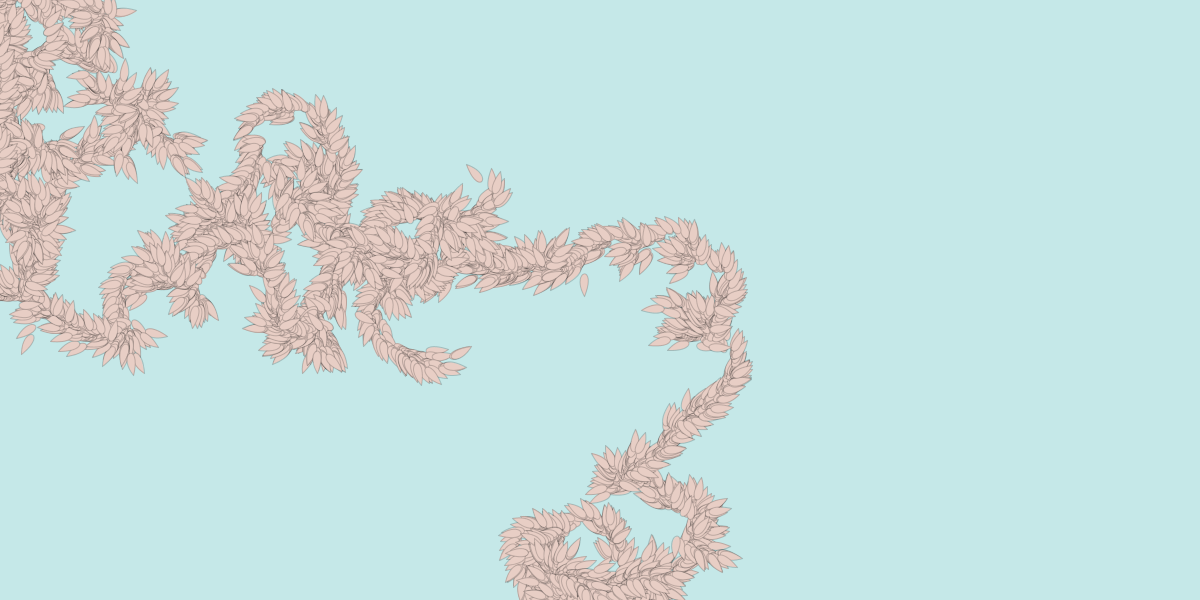
****

Рис.10. Результат для предложения FF[F-][+F]++F--F++F--F с кривыми Безье и случайным углом поворота.

Изменение расстояниямежду элементами дает новый рисунок, напоминающий следы на снеге. (Рис. 11)

translate(len\*20, 0);

****

Рис.11. Результат для предложения FF[F-][+F]++F--F++F--F с кривыми Безье и случайным углом поворота и увеличенным интервалом между элементами.

Рисование эллипсов с функцией random(), примененной к диаметру, а также новое предложение даст новые рассеяные паттерны.

ellipse(0,0,random(len),random(len));

ruleset[0] = **new** Rule('F', "FF[[-F[+F][-F]]++F]--F++F--F");

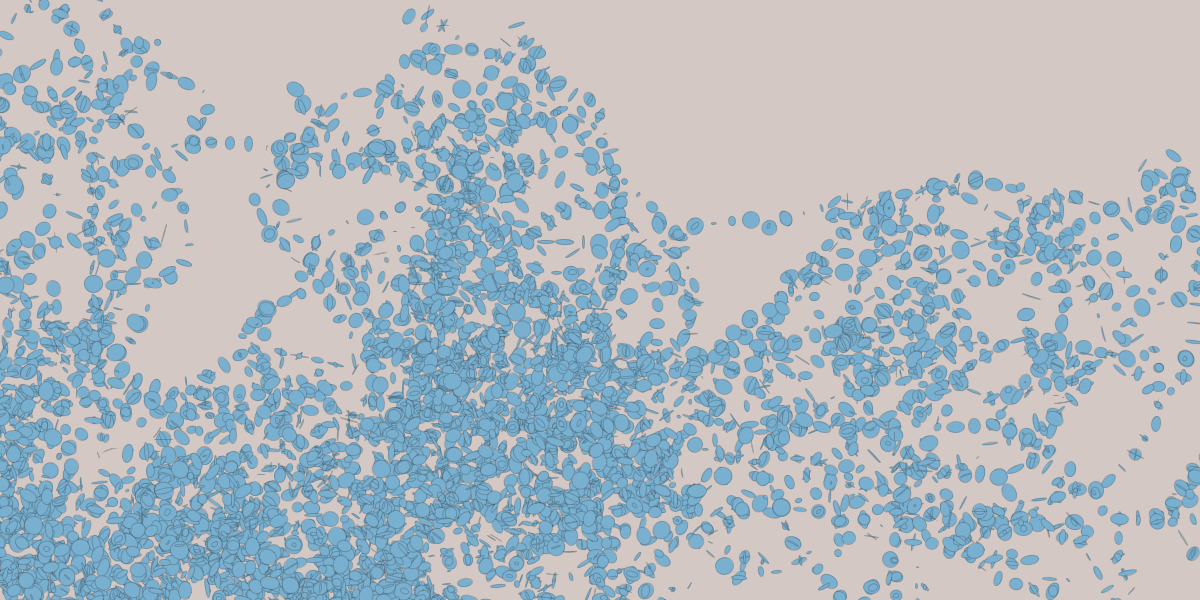
****

Рис.12. Результат для предложения FF[[-F[+F][-F]]++F]--F++F--F с кривыми Безье и случайным углом поворота и увеличенным интервалом между элементами.

На рис.10 приведен результат для нового предложения, в которое добавлена новая буква С, которая идентифицирует рисование кругов. F будет отвечать за рисование кривых Безье – листьев. Эту команду мы уже применяли при генерации рис.7, 8.

ruleset[0] = **new** Rule('F', "F+[-FC]-FC");

**else** **if** ( c == 'C') {

fill(188,68,92);

ellipse(0,0,len,len);

}

Необходимо также отметить, что изменение коэффициента также влияет на изображение. Он регулирует изменение размера элементов при каждом клике мыши в процентном соотношении. В данном случае он составляет 0.5.

turtle.changeLen(0.5);

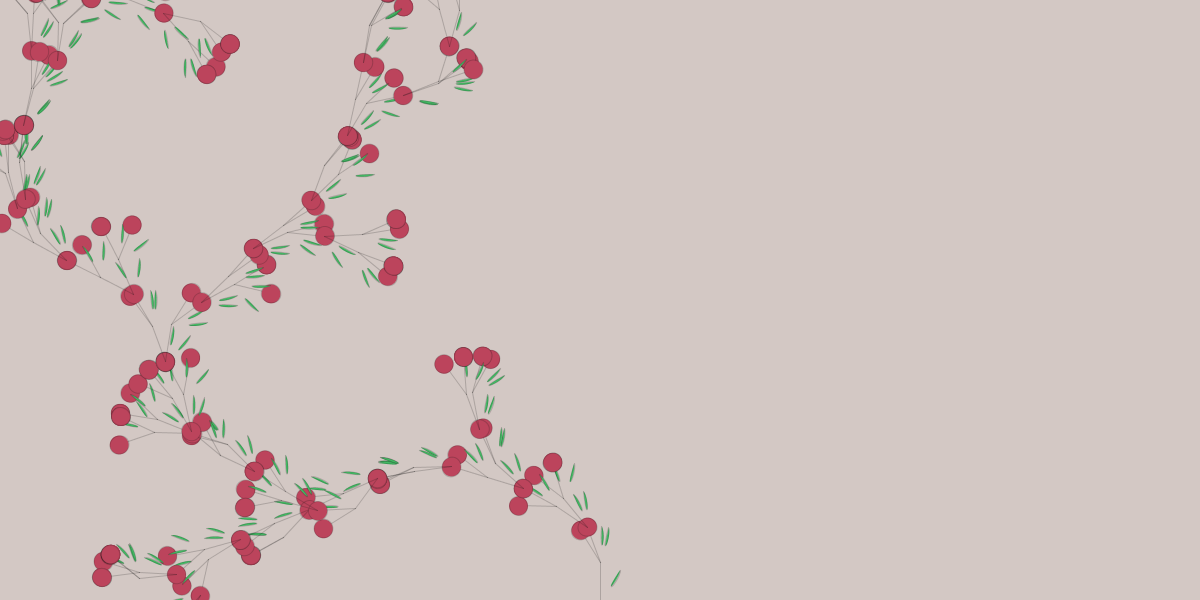


Рис.13. Результат для предложения F+[-FC]-FC с кривыми Безье, эллипсами и случайным углом поворота.

На рис. 14 изображен результат для предложения F+F-FF-F+F-FF-F+F и аксиома –F с углом поворота 90 градусов. Так с помощью L системы можно получить симментричный паттерн. А используя различные графические элементы и их комбинации, можно получить множество других рисунков, напоминающих орнамент.

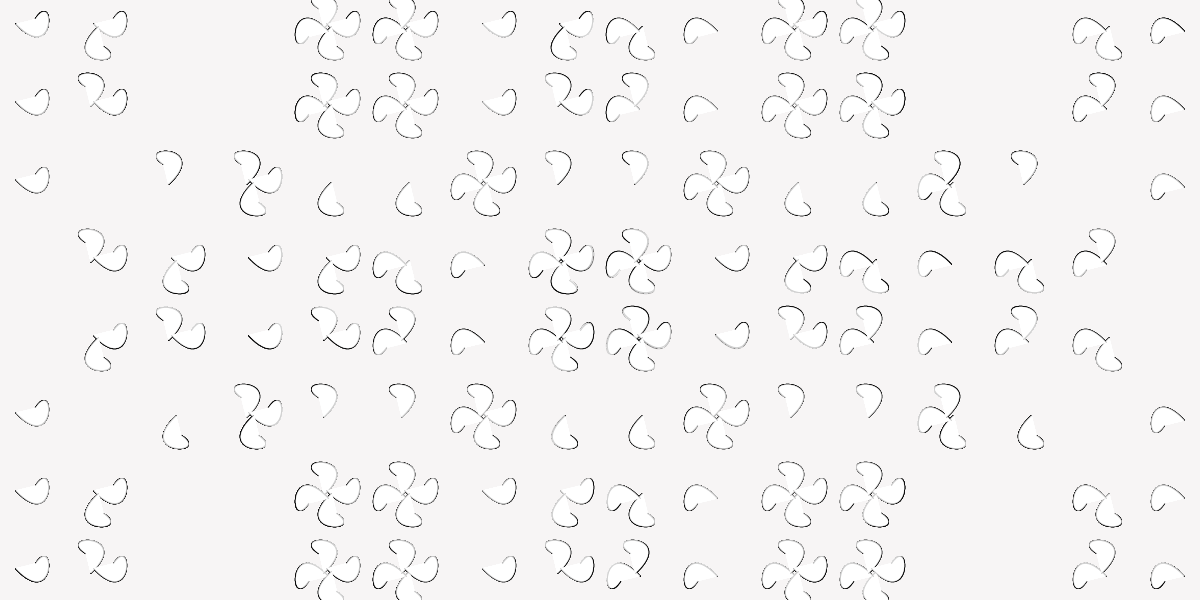
****

Рис. 14. Предложение F+F-FF-F+F-FF-F+F и аксиома –F с углом 90 градусов.

**2.4. Агрегация ограниченная диффузией**

Алгоритм агрегация ограниченная диффузией (Diffusion limited aggregation, далее DLA) был предложен Уиттеном и Сандером в 1981 году [19] как модель формирования паттерна и пример самоорганизованной критичности. Этот алгоритм моделирует большое количество физических систем, таких как

кристаллизация в растворе, осаждение металла при электролизе, электрический разряд при пробое диэлектрика, рост минеральных дендритов, бактериальные колонии, диффузионные процессы в жидкостях, кластеры при высыхании на стекле коллоидного раствора. [1]

Суть алгоритма достаточно проста. Предположим существует некоторое облако случайно блуждающих частиц, в центре него находится агрегатор. При приближении частицы к агрегатору, она прилипает к нему. С течением времени частицы образуют конгломерат, обладающий фрактальными свойствами. (Рис. 15)

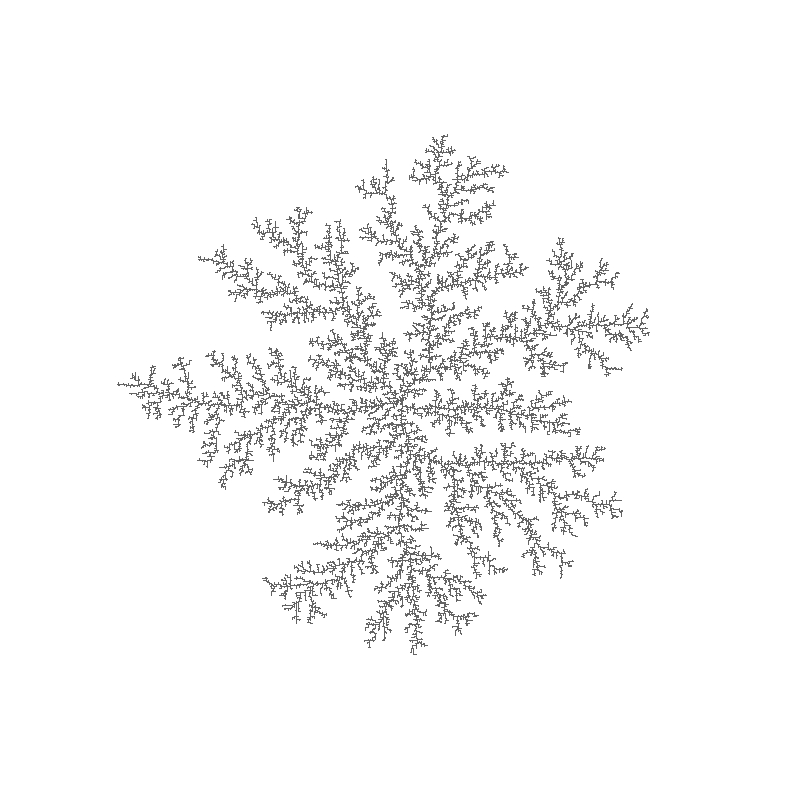


Рис. 15 Компьютерная модель алгоритма DLA

**Формальное описание алгоритма DLA**

Федер предлагает следующее описание процесса ограниченной диффузией агрегации.

Случайное блуждание частиц описывается уравнением диффузии. Пусть – плотность блуждающих частиц. Тогда уравнение диффузии представимо в виде:

Коэффициент диффузии в общем случае определяется соотношением Эйнштейна для частиц, перемещающихся на шаг *a* во всех направлениях со скоростью Г.

Если станционарное состояние (скорость) , тогда уравнение диффузии сводится к уравнению Лапласа:

.

[6,19]

Фрактальная размерность двумерной реализации DLA приближается к 1, 71. [13]

**Классическая модель DLA**

На рис. 16 проиллюстрирован процесс реализации DLA на двумерной дискретной решетке. Предполагается, что процесс построения начинается с одной частицы, помещенной в центре, остальные частицы присоединяются к нему последовательно. Положение второй частицы определяется случайным образом. Она начинает случайное блуждание по координатной сетке. C вероятностью ¼ она будет перемещаться на единицу вверх, вниз, вправо или влево. Она продолжает блуждание до тех пор, пока не окажется рядом с агрегатором, или частицей, которая уже входит в него. В этот момент частица останавливается и соединяется отрезком с агрегатором, и становится его частью. После этого происходит бросание новой частицы. Процесс повторяется многократно. [1]

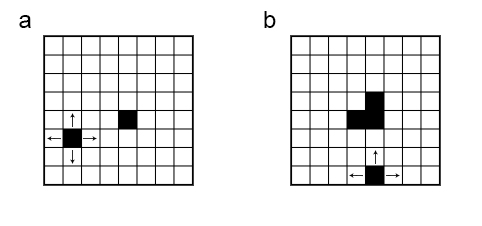


Рис. 16 a) Частица, инициализированная как агрегатор и введенная вторая частица, двигающаяся случайно. b) Четвертая частица движется случайным образом к агрегатору из трех частиц.

Наши эксперименты в Processing основаны на коде, реализованном на языке JavaScript Д. Шиффманом в видеоуроке, опубликованном на ресурсе YouTube. Запускается в приложении Processing в режиме p5.js, или в любом другом компиляторе JavaScript.[7]

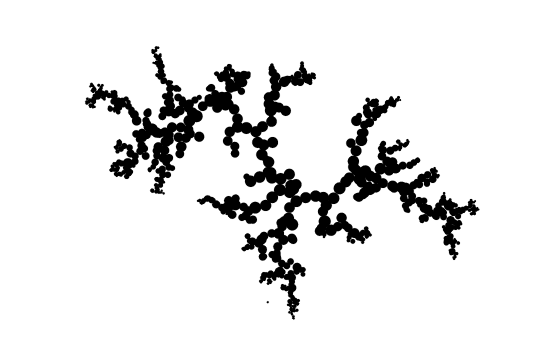


Рис. 17. Визуализация DLA, полученная с помощью базового кода (Приложение 2).

Код, описывающий DLA-систему, разделен на несколько функций:

**walk** – описывает скорость роста агрегатора, а также задает ограничение роста (функция constrain()).

**checkStuck** – описывает процесс присоединения частиц к агрегатору.

**show** – регулирует все графические отображения.

**randomPoint** – задает броуновское движение частиц.

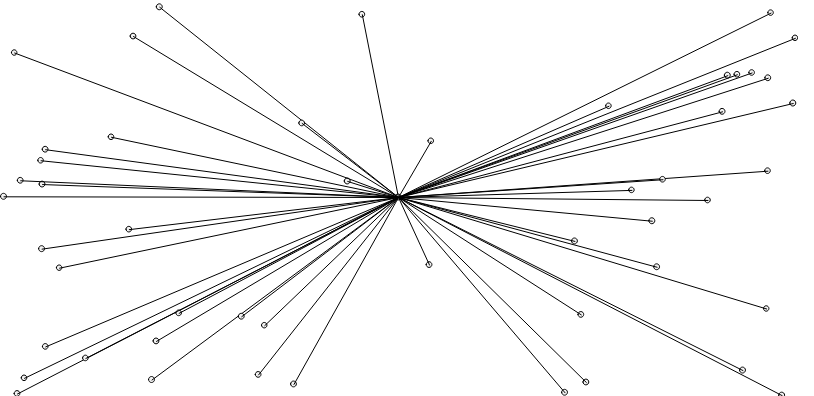
**distSq** – коэффициент прилипания.

На рис. 17 представлен конечный результат базового кода (Приложение 2).

**2.5. Результаты применения алгоритма DOL для генерации контента в Processing.**

Данный алгоритм представляет интерес прежде всего в качестве интересной абстрактной анимации. Однако отдельные кадры тоже обладают эстетической привлекательностью. Рассмотрим как изменение некоторых параметров будет влиять на изображение.

На рис. 18-20 представлены кадры анимации броуновского движения частиц, соединенных линиями с агрегатором.

Рис. 18. Визуализация броуновского движения частиц, соединенных линиями с агрегатором.

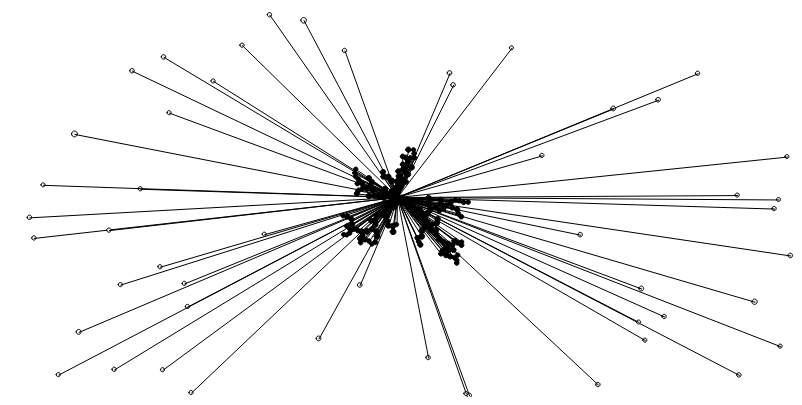


Рис. 19. Визуализация броуновского движения частиц, соединенных линиями с агрегатором и рост агрегатора.

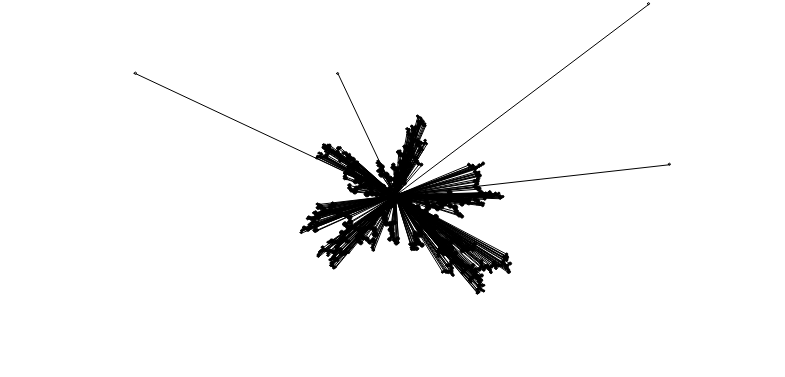


Рис. 20. Визуализация броуновского движения частиц, соединенных линиями с агрегатором и рост агрегатора.

Также можно поэкспериментировать и соединить блуждающие точки между собой, что даст новую интересную структуру (рис. 21). Программный код для этой операции:

var connectedWalkers = [];

var halfSize = Math.floor(walkers.length / 2);

**for** (var i = 0; i < halfSize; i++) {

var walkerFrom = walkers[i],

walkerTo,

to;

**do** {

to = halfSize + Math.floor(random(halfSize))

} **while** (connectedWalkers.indexOf(to) !== -1);

connectedWalkers.push(to);

walkerTo = walkers[to];

strokeWeight(1);

line(walkerFrom.pos.x, walkerFrom.pos.y, walkerTo.pos.x, walkerTo.pos.y);

}

}

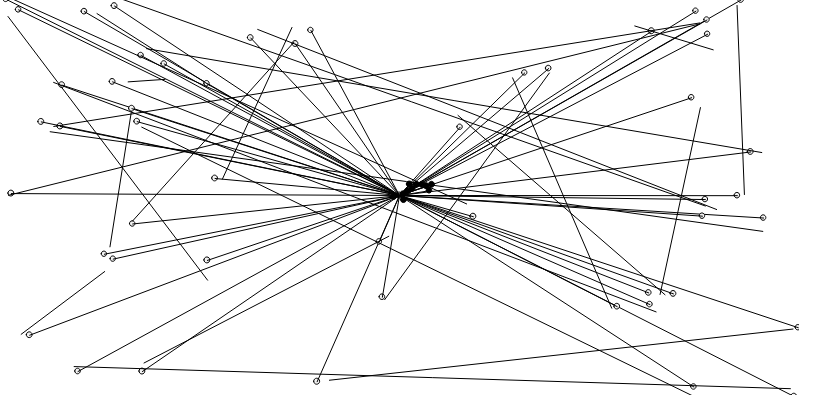


Рис. 21. Линии, соединяющие блуждающие точки между собой и агрегатором.

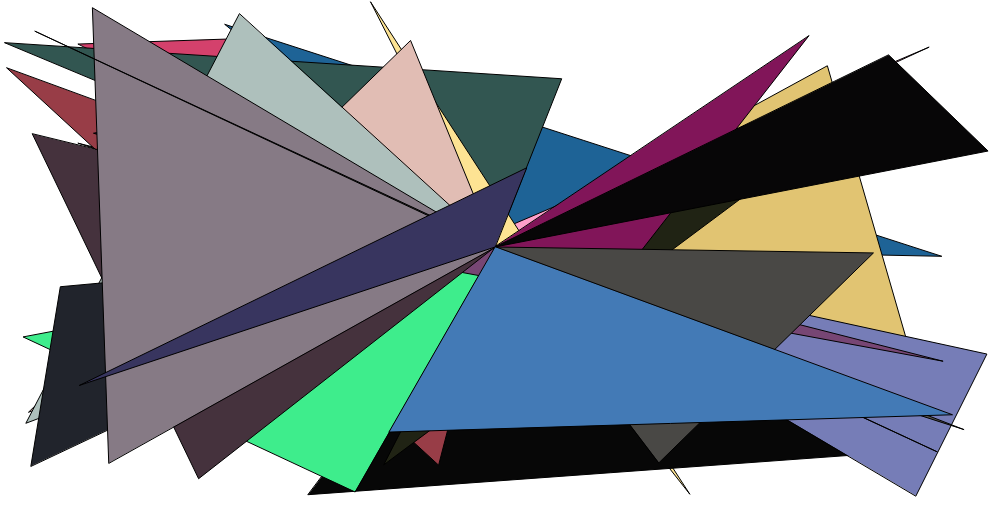


Рис. 22. Треугольники, соединяющие блуждающие точки между собой и агрегатором.

Нарисуем треугольники, соединив блуждающие точки с центром экрана и назначим для них случайный цвет (рис. 22):

fill(random(360),random(100),random(100),50);

triangle(walkerFrom.pos.x, walkerFrom.pos.y, walkerTo.pos.x, walkerTo.pos.y, width/2, height/2);

**Заключение**

В настоящей работе были исследованы возможности применения алгоритмов L-система и DLA при создании контента в среде Processing. Эти алгоритмы представляют большой интерес для производства разнообразной графики. А параметрические возможности Processing позволяют путем изменения некоторых коэффициентов создавать новые паттерны и текстуры. Результатом данной работы стал графический контент, основанный на этих алгоритмах, который может быть использован в дизайне, а также для создания анимаций, иллюстрирующих избранные алгоритмы – L-систему и DLA. Что представляет интерес в качестве популяризации науки. Не смотря на то, что создатели утверждают, что Processing прост для применения, все же для создания впечатляющего контента, от пользователя требуются глубокие знания специфики объектно-ориентированного программирования и синтаксиса Java и JavaScript.

**Библиография**

1. Батюков А. М., Ампилова Н. Б. Модифицированный алгоритм моделирования агрегации ограниченной диффузией. [Электронный ресурс] // URL: https://cyberleninka.ru/article/v/modifitsirovannyy-algoritm-modelirovaniya-agregatsii-ogranichennoy-diffuziey (дата обращения: 25.04.2018)
2. Кроновер Р. М. Фракталы и хаос в динамических системах. Основы тео-
3. Мандельброт Б. Б. Фрактальная геометрия природы. М.: Институт компьютерных исследований, 2002
4. рии. М.: Постмаркет, 2000
5. Федер Е. Фракталы. М.: Мир, 2001
6. Шредер, М. Фракталы, хаос, степенные законы. Миниатюры из бесконечного рая. Ижевск: Регулярная и хаотическая динамика, 2001

# Coding Challenge #34: Diffusion-Limited Aggregation [Электронный ресурс] // URL: https://www.youtube.com/watch?v=Cl\_Gjj80gPE (дата обращения: 25.04.2018)

1. Feldman D. P. Chaos and fractals. An elementary introduction. Oxford, 2012
2. Galanter P. Generative art and rules-based art vague terrain [Электронный ресурс] // Philip Galanter. [сайт]. URL: <http://philipgalanter.com/downloads/vague_terrain_2006.pdf> (дата обращения:23.04.2018)
3. Galanter P. What is Generative Art? Complexity Theory as a Context for Art Theory [Электронный ресурс] // Philip Galanter. [сайт]. URL: http://philipgalanter.com/downloads/ga2003\_what\_is\_genart.pdf (дата обращения:23.04.2018)
4. Generative gestaltung [Электронный ресурс] // URL: http://www.generative-gestaltung.de/ (дата обращения:20.04.2018)
5. Greenberg I. Processing Creative Coding and Computational Art. New York, 2007
6. Meakin P., Tolman S., Blumen A. Diffusion-limited aggregation.
7. Proceedings of the Royal Society of London. Series A, Mathematical and PhysicalSciences, Vol. 423, No. 1864, Fractals in the Natural Sciences (May 8, 1989), pp. 133-148
8. Prusinkiewicz P, Lindenmayer A. The Algorithmic Beauty of Plants. New York, 1990
9. Prusinkiewicz P., Hanan J. Lindenmayer Systems, Fractals, and Plants [Электронный ресурс] // URL: http://algorithmicbotany.org/papers/lsfp.pdf (дата обращения: 1.04.2018)
10. Reas C. Form + code in design, art and architecture. New York, 2010
11. Reas C., Fry B. Processing: a programming handbook for visual designers and artists Casey Reas Ben Fry. Cambridge, MA, 2007
12. Sander L. M. Diffusion-limited aggregation: A kinetic critical phenomenon? [Электронный ресурс] // URL: http://www.thp.uni-koeln.de/krug/teaching-Dateien/SS2012/Sander2000.pdf (дата обращения:20.04.2018)
13. Shiffman D. Learning Processing A Beginner’s Guide to Programming Images, Animation, and Interaction. Burlington, 2008
14. Shiffman D. The Nature of Code [Электронный ресурс] // URL: <http://natureofcode.com/book/> (дата обращения: 16.04.2018)
15. Terzidis K. Algorithms for Visual Design Using the Processing Language. Indianapolis, 2009
16. Thornburg D. D. On Logo And Turtles[Электронный ресурс] // Compute! : сетевой журнал. URL: https://www.atarimagazines.com/compute/issue34/068\_1\_FRIENDS\_OF\_THE\_TURTLE.php (дата обращения: 1.04.2018)

Приложение 1

*// The Nature of Code*

*// Daniel Shiffman*

*// http://natureofcode.com*

LSystem lsys;

Turtle turtle;

void setup() {

size(600, 600);

Rule[] ruleset = **new** Rule[1];

ruleset[0] = **new** Rule('F', "FF+[+F-F-F]-[-F+F+F]");

lsys = **new** LSystem("F", ruleset);

turtle = **new** Turtle(lsys.getSentence(), height/3, radians(25));

}

void draw() {

background(255);

fill(0);

translate(width/2, height);

rotate(-PI/2);

turtle.render();

noLoop();

}

int counter = 0;

void mousePressed() {

**if** (counter < 5) {

pushMatrix();

lsys.generate();

turtle.setToDo(lsys.getSentence());

turtle.changeLen(0.5);

popMatrix();

redraw();

counter++;

}

}

**class** **LSystem** {

String sentence;

Rule[] ruleset;

int generation;

LSystem(String axiom, Rule[] r) {

sentence = axiom;

ruleset = r;

generation = 0;

}

void generate() {

StringBuffer nextgen = **new** StringBuffer();

**for** (int i = 0; i < sentence.length(); i++) {

char curr = sentence.charAt(i);

String replace = "" + curr;

**for** (int j = 0; j < ruleset.length; j++) {

char a = ruleset[j].getA();

**if** (a == curr) {

replace = ruleset[j].getB();

**break**;

}

}

nextgen.append(replace);

}

sentence = nextgen.toString();

generation++;

}

String getSentence() {

**return** sentence;

}

int getGeneration() {

**return** generation;

}

}

**class** **Rule** {

char a;

String b;

Rule(char a\_, String b\_) {

a = a\_;

b = b\_;

}

char getA() {

**return** a;

}

String getB() {

**return** b;

}

}

**class** **Turtle** {

String todo;

float len;

float theta;

Turtle(String s, float l, float t) {

todo = s;

len = l;

theta = t;

}

void render() {

stroke(0, 175);

**for** (int i = 0; i < todo.length(); i++) {

char c = todo.charAt(i);

**if** (c == 'F' || c == 'G') {

line(0, 0, len, 0);

translate(len, 0);

} **else** **if** (c == '+') {

rotate(theta);

} **else** **if** (c == '-') {

rotate(-theta);

} **else** **if** (c == '[') {

pushMatrix();

} **else** **if** (c == ']') {

popMatrix();

}

}

}

void setLen(float l) {

len = l;

}

void changeLen(float percent) {

len \*= percent;

}

void setToDo(String s) {

todo = s;

}

}

Приложение 2

*//Daniel Shiffman*

**var** tree = [];

**var** walkers = [];

**var** maxWalkers = 50;

**var** iterations = 50;

**var** radius = 6;

**var** shrink = 0.998;

**var** hu = 5;

**function** setup() {

createCanvas(1200,600);

colorMode(HSB);

tree[0] = **new** Walker(width/2, height/2);

**for**(**var** i = 0; i < maxWalkers; i++){

walkers[i] = **new** Walker();

radius \*= shrink;

}

}

**function** draw() {

background(255);

**for**(**var** i = 0; i < tree.length; i++){

tree[i].show();

}

**for**(**var** i = 0; i < walkers.length; i++){

walkers[i].show();

}

**for**(**var** n = 0; n < iterations; n++) {

**for**(**var** i = walkers.length - 1; i >=0; i--){

walkers[i].walk();

**if** (walkers[i].checkStuck(tree)){

walkers[i].setHue(hu % 360);

hu += 0.5;

tree.push(walkers[i]);

walkers.splice(i, 1);

}

}

}

**var** r = walkers[walkers.length - 1].r;

**while**(walkers.length < maxWalkers && radius >1) {

radius \*= shrink;

walkers.push(**new** Walker());

}

}

**function** Walker(x, y) {

**if** (arguments.length == 2){

**this**.pos = createVector(x,y);

**this**.stuck = **true**;

}**else** {

**this**.pos = randomPoint();

**this**.stuck = **false**;

}

**this**.r = radius;

**this**.walk = **function**(){

**var** vel = createVector(random(-10,10), random(-10,10));

**this**.pos.x = constrain(**this**.pos.x, 3, width);

**this**.pos.y = constrain(**this**.pos.y, 3, height);

**this**.pos.add(vel);

}

**this**.checkStuck = **function**(others){

**for**(**var** i = 0; i < others.length; i++){

**var** d = distSq(**this**.pos, others[i].pos);

**if** (d < (**this**.r \* **this**.r \* 4)){

**this**.stuck = **true**;

**return** **true**;

**break**;

}

}

**return** **false**;

}

**this**.setHue = **function**(hu){

**this**.hu = hu;

}

**this**.show = **function**(){

stroke(0);

strokeWeight(0.1);

**if** (**this**.stuck){

fill(**this**.hu, 283,0,95,50);

}**else**{

fill(0, 50);

}

ellipse(**this**.pos.x, **this**.pos.y, **this**.r\*2, (**this**.r\*2));

}

}

**function** randomPoint(){

**var** i = floor(random(4));

**if** (i === 1) {

**var** x = random(width);

**return** createVector(x, height);

} **else** **if** (i === 2){

**var** x = random(width);

**return** createVector(x, 0);

} **else** **if** (i === 3){

**var** y = random(height);

**return** createVector(0, y);

}**else** {

**var** y = random(height);

**return** createVector(width, y);

}

}

**function** distSq(a, b){

**var** dx = b.x - a.x;

**var** dy = b.y - a.y;

**return** dx\*dx + dy\*dy;

}