**Контрольное домашнее задание. 3 модуль.**

**2019-2020 уч. год.**

**У вас есть две возможности:**

1. **Делать КДЗ общепотоковое с проведением peer-review. Все вопросы по нему задаете в форме авторам -** [**https://docs.google.com/forms/d/e/1FAIpQLSeYQbkwSAWRMloohq2rifIOfqg-Vhkgp1J89tgsQoQppY-5cQ/viewform**](https://docs.google.com/forms/d/e/1FAIpQLSeYQbkwSAWRMloohq2rifIOfqg-Vhkgp1J89tgsQoQppY-5cQ/viewform) **Ответы тут:** [**https://docs.google.com/spreadsheets/d/1RHMdYMisdgpMbhC0VOpfn0NVN9uITOrukV5wV0AkCSU/edit?usp=sharing**](https://docs.google.com/spreadsheets/d/1RHMdYMisdgpMbhC0VOpfn0NVN9uITOrukV5wV0AkCSU/edit?usp=sharing)
2. **Делать КДЗ в парах внутри 196 группы по фракталам с обязательным использованием гита (gitlab).**

**Изменить свой выбор после 22.02.20 (то есть после заполнения гугл-формы о распределении в пары) – НЕЛЬЗЯ.**

**До 21:00 22.02.20 вы заполняете гугл-форму** [**https://docs.google.com/forms/d/e/1FAIpQLSeM2JcMGmhBVFs030M7Z-k1sKxaQo1breAnT0AYoCBfb65HxQ/viewform?usp=sf\_link**](https://docs.google.com/forms/d/e/1FAIpQLSeM2JcMGmhBVFs030M7Z-k1sKxaQo1breAnT0AYoCBfb65HxQ/viewform?usp=sf_link) **в которой указываете, какой вариант КДЗ вы выполняете (1 или 2) и вашу команду из двух человек (если выбрали 2 вариант). Заполнить достаточно только одному человеку. Если вы выбираете 1 вариант, то форму можно не заполнять.** [**https://docs.google.com/spreadsheets/d/1YlNmoWqZe60\_esoBChR1KL0y5Wnml5KelcCag6t38vo/edit#gid=856702387**](https://docs.google.com/spreadsheets/d/1YlNmoWqZe60_esoBChR1KL0y5Wnml5KelcCag6t38vo/edit#gid=856702387) **– здесь лежат ответы на форму, ваш вариант, это номер строки в которую записался ваш ответ (да, нумерация вариантов будет с 2).**

**Далее будет описана только возможность 2), и все что тут написано относится только к ней.**

**На выполнение контрольного домашнего задания (КДЗ) отводится четыре недели.**

**Дата выдачи: 20 февраля 2020 года.**

**Дата сдачи: 17 марта 2020 года 23:00.**

**Предусмотрена защита КДЗ на одном из семинарских занятий 18 марта 2020 года.**

**Оценка КДЗ состоит из двух частей: программная реализация (сюда входит и использование стека технологий) и защита на семинаре.**

**Работа осуществляется в парах – по 2 человека.**

**Формула оценки КДЗ:**

**0,8\* Опрограммная\_реализация + 0,2 \* Озащита**

**Уровни мастерства программного инженера:**

1. **Используете Git (Gitlab) – ОБЯЗАТЕЛЬНО ДЛЯ ВСЕХ!**
2. **Используете логирование (библиотека NLog) – НЕОБЯЗАТЕЛЬНО, НО ЖЕЛАТЕЛЬНО!**
3. **Автотесты – НЕОБЯЗАТЕЛЬНО, НО ЖЕЛАТЕЛЬНО!**
4. **Настроили CI и автотесты при merge request (мердж реквестах) – НЕОБЯЗАТЕЛЬНО, НО ЖЕЛАТЕЛЬНО!**

**Без пункта 1 оценить работу и вклад каждого члена команды будет невозможно, увы. НО! Отсутствие пунктов 2, 3 и 4на оценку не влияет! Мы все-таки программировать учимся, но эти навыки вам пригодятся в дальнейшем и есть прекрасный шанс потренироваться. Но за 2, 3 и 4, от меня вам отдельный почет и уважение!**

В этом задании требуется разработать программу, которая строит изображения четырех фракталов (каждый должен разработать по 2 фрактала). Будут использованы фракталы, построение которых естественным образом осуществляется с помощью рекурсивных процедур. Фракталы и их построение описаны в разделе 4.

1. **Постановка задачи**

Разработать оконное приложение (Windows Forms Application **или** WPF (изучается самостоятельно)), позволяющее:

1. Отрисовывать четыре вида фракталов, которые определены в индивидуальном варианте. Описание фракталов представлено ниже.
2. Предоставлять пользователю выбор текущего фрактала для отрисовки.
3. Предоставлять пользователю возможность устанавливать количество шагов рекурсии (её глубину - количество рекурсивных вызовов). При изменении глубины рекурсии фрактал должен быть автоматически перерисован. Следите за переполнением стека.
4. Автоматически перерисовывать фрактал при изменении размеров окна. Окно обязательно должно быть масштабируемым. Вы можете задать минимальный и максимальный размер окна. Максимальным считается размер окна, соответствующий размеру экрана, а минимальным размером окна считается половинный размер экрана (как по длине, так и по ширине).
5. Предоставлять пользователю возможность выбора двух цветов startColor и endColor. Цвет startColor используется для отрисовки элементов первой итерации, цвет endColor - для отрисовки элементов последней итерации. Цвета для промежуточных итераций должны вычисляться с использованием линейного градиента. <http://qaru.site/questions/222000/generate-color-gradient-in-c>

Например,



Рис. 1. Градиент цвета

Желтый цвет используется для отрисовки элементов первой итерации. Синий - для отрисовки элементов последней итерации.

Промежуточные цвета вычисляются исходя из начального и конечного значений цвета и номера итерации **(не генерируются случайным образом).**

1. Сообщать о некорректном вводе данных, противоречивых или недопустимых значениях данных и других нештатных ситуациях во всплывающих окнах типа MessageBox.
2. Должна быть предусмотрена возможность сохранения фрактала в виде картинки (формат выбрать самостоятельно).
3. Предусмотреть возможность изменения масштаба фрактала для его детального просмотра. Увеличение должно быть 2, 3 и 5-кратным
4. Предусмотреть возможность перемещения изображения, в т.ч. при увеличенном изображении (пункт 8).
5. *На интерфейсе может быть предусмотрена дополнительная функциональность на Ваше усмотрение (например, сохранение фрактала и его последующее восстановление – использовать сериализацию и т.д.).*

Для того, чтобы выделить базовую структуру алгоритма построения фракталов следует разбить процедуру рисования на небольшие действия, и те из них, которые окажутся одинаковыми для разных фракталов, поместить в базовый (родительский) класс. Различающиеся действия поместить в производные классы, создав по одному производному классу для каждого фрактала.

Определить в базовом классе **Fractal** следующие члены:

* поля:
  + длина отрезка (сторона квадрата, треугольника, радиус окружности и т.д.) на первой итерации
  + **startColor** (см. п. 5)
  + **endColor** (см. п. 5)
  + текущий уровень рекурсии
  + максимальный уровень рекурсии, заданный пользователем.
* метод **Draw** (может быть абстрактным или виртуальным, зависит от архитектуры ваших классов)

В зависимости от типа фрактала в классы-наследники следует добавить необходимые члены класса, указанные при описании фрактала. Допустимо добавлять другие члены класса.

**Ограничения:**

1. В программной реализации не использовать вспомогательные компоненты и сторонние библиотеки, не входящие в стандартную библиотеку (кроме градиента и NLog).

2. Не использовать массивы типа **object[]**.

1. **Работа с Git**

**Используемый веб-сервис:** gitlab.com

**Обязательные условия при использовании СКВ и системы управления проектом:**

1. Проект должен иметь имя <number>\_P\_HW3\_Fractals, например 0\_P\_HW1\_ Fractals. Number – номер варианта. Можете создать проект и запушить из студии первый комит в maste (только первый разрешаю в мастер). Не забудьте поставить расширение для gitlab.

Изображение выглядит как снимок экрана, ноутбук, компьютер, монитор

Автоматически созданное описание



Изображение выглядит как снимок экрана, компьютер

Автоматически созданное описание



Изображение выглядит как снимок экрана

Автоматически созданное описание



Изображение выглядит как снимок экрана

Автоматически созданное описание

1. Преподавателя добавить в свой проект как владельцев (contributors) (поставить права maintainer). Мой ник – mkgordenko.

Изображение выглядит как снимок экрана

Автоматически созданное описание



Изображение выглядит как снимок экрана

Автоматически созданное описание



1. В настройках проекта **ОБЯЗАТЕЛЬНО(!!!)** отключить рассылку уведомлений по e-mail.
2. Для выполнения задания вы должны должен создать проект в gitlab и заполнить доску (issues/board) с первоначальными задачами (tasks) (примерно 30-50% от общего предполагаемого количества задач (tasks) в проекте). Задачи (tasks) должны вытекать из требований к программе и быть декомпозированы. После начала работы, по мере проработки проекта и создания задач членами команды, досоздаются необходимые.

Изображение выглядит как снимок экрана

Автоматически созданное описание



Изображение выглядит как снимок экрана

Автоматически созданное описание



Изображение выглядит как снимок экрана

Автоматически созданное описание

Изображение выглядит как снимок экрана

Автоматически созданное описание

1. Каждая задача должна быть назначена определенному члену команды, иметь конкретный срок окончания, быть оцененной по времени, иметь определенный лейбл (основные используемые лейблы – новая функциональность, улучшение, баг) в зависимости от задачи и вехи проекта (mail stone) (версия 1, версия 2).
2. Ваш проект имеет две вехи (mail stone): версия 1 и версия 2. В версию 1 должна войти работоспособная программа с двумя фракталами от двух членов команды. В версию 2 должна войти полностью работоспособная программа. После того, как версия 1 завершена, вы должны смерджить (merge) её в ветку (branch) master, тоже самое и с версией 2.
3. Сделать .gitignore и не грузить в репозиторий лишние файлы. В репозитории должен быть только исходный код, возможно, еще файл README.
4. При разработке необходимо соблюдать гитфлоу[[1]](#footnote-1) (без ветки testing).

Изображение выглядит как снимок экрана

Автоматически созданное описание

1. Репозиторий используется для хранения любого дерева веток, но подразумевается наличие, как минимум, двух: master и develop. В вашем случае, в master будет лежать версия проекта 1, а затем версия проекта 2. В develop лежит текущий разрабатываемый код.

Изображение выглядит как снимок экрана

Автоматически созданное описание



Изображение выглядит как снимок экрана

Автоматически созданное описание



1. В develop в ветках feature ведётся разработка - с ними можно делать что угодно.
2. **Выгрузка в master должна быть напрямую запрещена (настроить в gitlab).** В нее можно только вливать изменения из develop с предварительным мердж реквестом (в гитхабе это называется пул реквест).

Изображение выглядит как снимок экрана

Автоматически созданное описание



Изображение выглядит как снимок экрана

Автоматически созданное описание

Изображение выглядит как снимок экрана

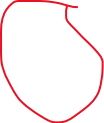
Автоматически созданное описание



1. Под каждую задачу (где нужно писать код и что-то комитить и пушить) создавать отдельную ветку из develop.

Изображение выглядит как снимок экрана

Автоматически созданное описание



1. Коммиты необходимо делать осознано, с понятными другим людям (!) комментариями (что было сделано по факту).
2. После пуша в удаленный репозиторий создавать мердж реквесты в develop ~~и настроить в проекте аппрувы и ревью кода~~ (в бесплатной версии нельзя настроить аппруверов, поэтому, когда делаете мердж реквест, то назначаете другого члена команды в качестве ревьюера - вы пишете комментарий с его упоминанием в тексте с просьбой сделать ревью - как ревьюер поставит свой аппрув (напишет комментарий, что все хорошо) вы (и только вы) делаете мердж в develop. Ревьюерам: при необходимости вам можно и нужно реджектить пул-реквест (через комментарии и упоминание девелопера) и отправлять его доработку.

*задача -> создать ветку из задачи -> закоммитить -> запушить в фича-ветку-> мердж реквест -> ревью -> мердж при успешном ревью*

Изображение выглядит как снимок экрана

Автоматически созданное описание



Изображение выглядит как снимок экрана

Автоматически созданное описание

1. Показывать, сколько времени было реально потрачено на задачу (по сравнению с тем, которое было оценено заранее).
2. Обратите внимание на вкладки Project->Cycle Analytics, Repository->Graph, Repository-Charts. Посмотрите, каким образом отслеживается статистика по проекту и по каждому участнику проекта.
3. **Удаляйте фича-бранчи при мердже.**

Так может выглядеть ваша доска в процессе работы (а может и по-другому):

Изображение выглядит как снимок экрана

Автоматически созданное описание

1. **Логирование NLog – простая, но очень полезная вещь!**

Отдельный плюс в карму за использование.

В реальной жизни не всегда есть возможность сделать debug и отладить программу. Более того, когда вы запускаете отладку, то вы должно воспроизвести произошедшую ситуацию, а это не всегда возможно, может быть просто непонятно почему, например, вылетело исключение. Чтобы задокументировать поведение программы используется такой прием, как логирование.

Конечно, мы могли бы написать свой собственный логер, но это не такая тривиальная задача, как может показаться.

Рекомендую посмотреть статью: <https://habr.com/ru/post/98638/> Здесь описано зачем вообще нужно логирование. Кто будет делать этот пункт, то статья для вас must have!

Итак, есть разные библиотеки, которые позволяют сохранять поведение вашей программы. Будем использовать NLog.

Как подключить библиотеку NLog и начать работать с ней <https://subreal-team.com/c-установка-и-настройка-nlog/> Я проверяла туториал, честно) Все описано хорошо и с картиночками.

Единственный неочевидный момент:

*«Важный момент, конфигурационный файл NLog.Config должен находиться в директории программы, поэтому необходимо выставить свойство файла «Копировать в выходной каталог» в статус «Всегда копировать». Это позволит упростить и автоматизировать применение изменений конфигурации логгирования»*

Кликаете по файлу NLog.config правой кнопкой мыши и выбираете «Свойства».

Изображение выглядит как снимок экрана

Автоматически созданное описание

1. **Тесты**

Представьте, что вы пишете код, и постоянно приходится его запускать и проверять. Это не всегда удобно и оправданно. А если у вас будет 10 функций, а если 100, а если 1000? И каждый раз вводить все вручную и проверять? Нет, конечно. Давно были придуманы сценарии, которые можно прогонять в полуавтоматическом или автоматическом режиме. Сейчас предлагаю посмотреть только unit-тесты, или тестирование отдельных функций и компонент приложения. В вашем случае, можно протестировать работу некоторых методов. Например, того, что фрактал создается. Интерфейс такими тестами мы не проверим. <https://habr.com/ru/post/169381/> вот тут хорошо рассказано, что это и с чем едят.

<https://docs.microsoft.com/ru-ru/visualstudio/test/walkthrough-creating-and-running-unit-tests-for-managed-code?view=vs-2019> официальный хороший туториал, как я люблю, с картинками и примерами кода.

1. **недоCI + тесты**

Тут я немного погорячилась)) на этот проект, не то, чтобы бессмысленно делать CI, но в общем-то, бессмысленно. Просто нам развертывать то нечего по итогу) Но зато таким способом можно автоматически прогонять тесты при каждом пуше. На конфигурацию не очень много времени. Лично у меня ушло 15 минут.

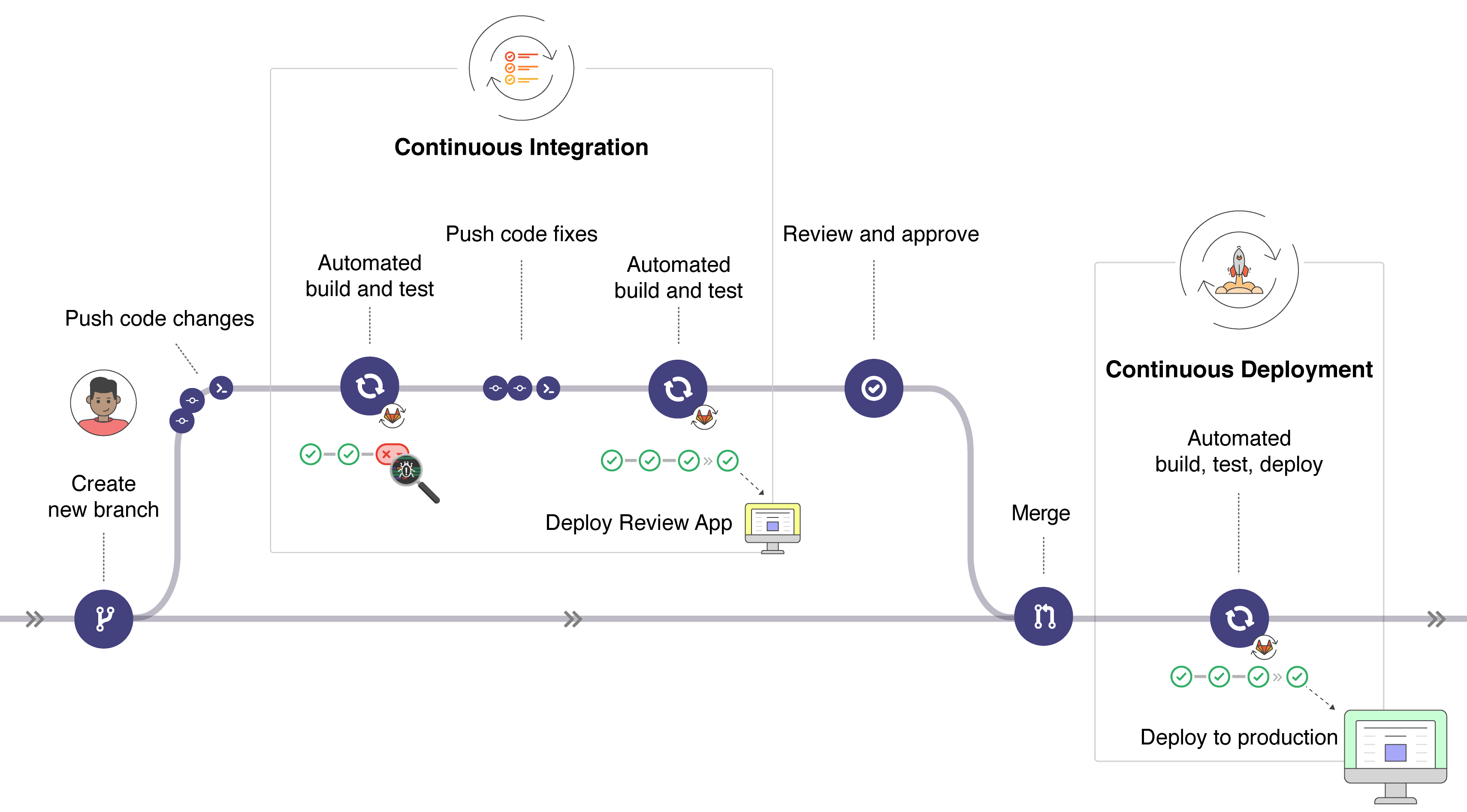
Вам понадобится официальный туториал (осторожно, английский), но там опять картинки, скриншоты и немного текста. Проверяла в точности по нему – все хорошо.

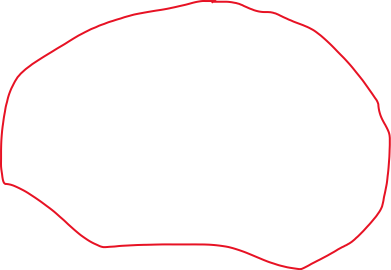
<https://docs.gitlab.com/ee/ci/introduction/>

<https://docs.gitlab.com/ee/ci/quick_start/README.html>

<https://docs.gitlab.com/ee/ci/quick_start/README.html#configuring-a-runner>

В идеале, хочется у вас увидеть вот такой процесс.





1. **Фракталы**

Фракталами называют математические множества, обладающие свойством самоподобия: любая часть фрактала подобна всему фракталу целиком. Термин «фрактал» был введён Бенуа Мандельбротом в 1975 году и получил широкую известность с выходом в 1977 году его книги «Фрактальная геометрия природы». Однако многие фракталы были известны и ранее.

1. ***Обдуваемое ветром фрактальное дерево.***

Фрактальное дерево иногда также называют Пифагоровым деревом. Первые итерации построения этого фрактала показаны на рис. 2.

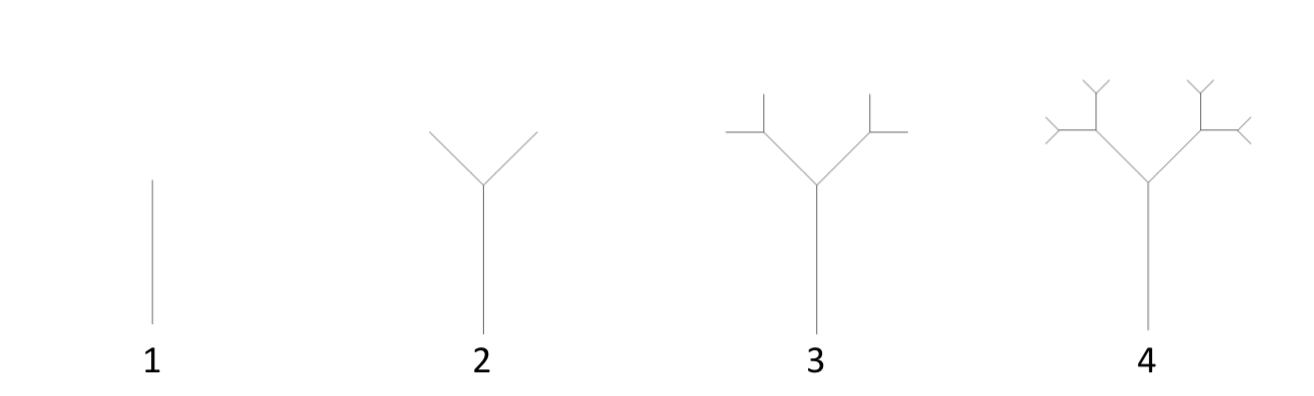


Рис. 2. Первые итерации построения фрактального дерева.

Процесс построения обычного фрактального дерева начинается с рисования вертикальной линии определённой длины. Затем к верхнему концу этой линии добавляют две линии меньшей длины, направленные под углом 45° направо и налево от направления исходной линии. И далее процесс продолжается – на концах каждой из линий добавляют ещё две линии и так далее. Теоретически, такой процесс построения фрактала должен продолжаться бесконечно. На практике его обычно ограничиваются конечным числом этапов (рис. 3).

Если использовать углы другой величины, то можно построить **обдуваемое ветром дерево Пифагора** (рис. 4).

Алгоритм :

1) Строим вертикальный отрезок.

2) Из верхнего конца этого отрезка рекурсивно строим еще 2 отрезка под определенными углами.

3) Вызываем функцию построения двух последующих отрезков для каждой ветви дерева.

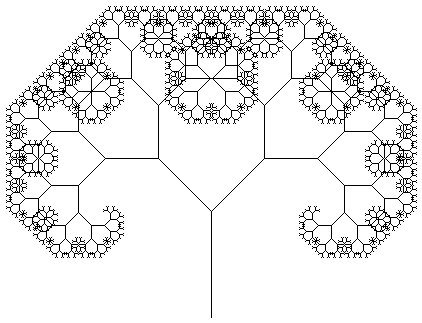


Рис. 3. Фрактальное дерево.

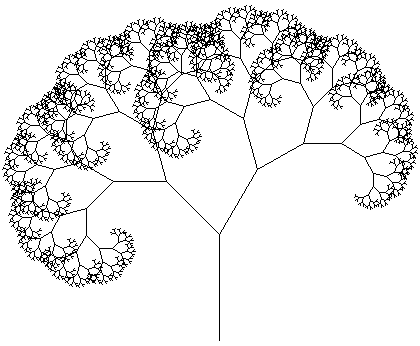


Рис. 4. Обдуваемое ветром фрактальное дерево.

При реализации фрактала учесть, что пользователь должен иметь возможность:

* задавать коэффициент, определяющий отношение длин отрезков на текущей и последующей итерации (с учетом разумных ограничений)
* угол наклона первого отрезка (с учетом разумных ограничений)
* угол наклона второго отрезка (с учетом разумных ограничений)

***2. Кривая Коха (рис. 5).***

Для построения этой кривой следует взять отрезок и заменить его центральную треть ломаной, повторяющей форму равностороннего треугольника. Затем, каждый из получившихся отрезков подвергается такому же преобразованию.

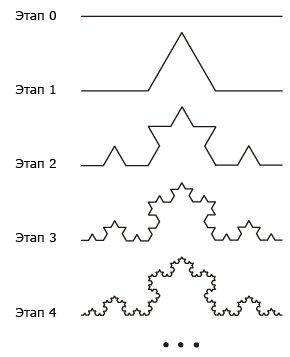


Рис.5. Первые итерации построения кривой Коха.

***3. Ковер Серпинского (рис. 6)***

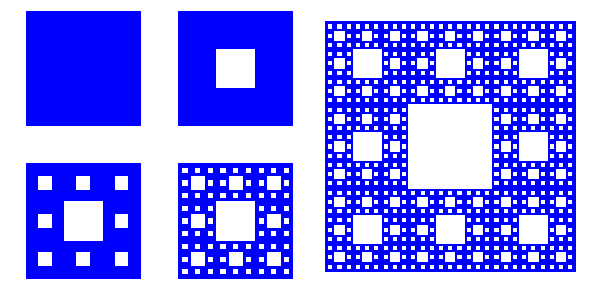


Рис. 6. Первые итерации построения ковра Серпинского

Построение Ковра Серпинского начинается с квадрата. На первом шаге квадрат разбивается сеткой 3 × 3 ячейки и центральная ячейка закрашивается. Затем процесс рекурсивно повторяют для 8 оставшихся ячеек.

***4. Треугольник Серпинского***

Чтобы получить этот фрактал, нужно взять равносторонний треугольник, затем достроить в нём треугольник, образованный средними линиями, затем в каждом из трех угловых треугольников сделать то же самое, и т. д. (рис. 7).

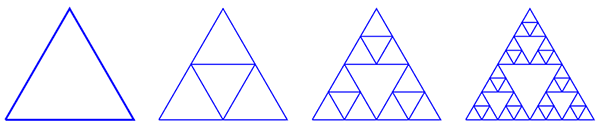


Рис. 7. Первые итерации другого способа построения треугольника Серпинского.

***5. Множество Кантора***

Способ построения этого множества следующий. Берётся отрезок прямой единичной длины (*[0,1]*). Затем он делится на три равные части, и вынимается средний отрезок (*[1/3, 2/3]*). Это первый шаг итерационной процедуры. На втором шаге подобной процедуре деления на три равные части и последующего удаления середины подвергается каждый из двух оставшихся отрезков. На рис. 8 первые шесть шагов процедуры.



Рис. 8. Множество Кантора

При реализации фрактала учесть, что пользователь должен иметь возможность определять расстояние между отрезками, отрисованными на разных итерациях.

***6. Т-квадрат***

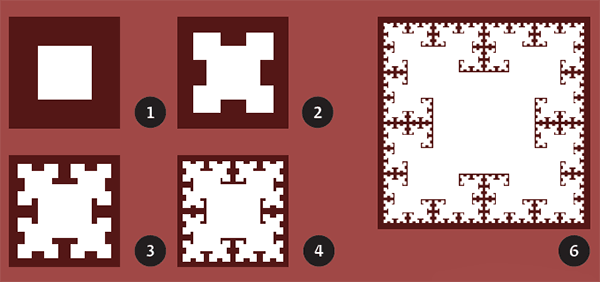


Рис. 9. Первые итерации построения Т-квадрата

Построение начинается с заполненного цветом единичного квадрата. Первый шаг: закрасить в центре квадрат со стороной 1/2. Затем нужно мысленно разделить квадрат на 4 одинаковых квадрата и в центре каждого из них закрасить квадрат со стороной 1/4. Дальше каждый из этих 4 квадратов снова делится на 4 части, всего получится 16 квадратов, и с каждым из них нужно рекурсивно проделать описанную выше процедуру (рис. 9).

***7. Н-фрактал***

Всё начинается с фигуры в виде буквы Н, у которой вертикальные и горизонтальные отрезки равны. Затем к каждому из 4 концов фигуры пририсовывается ее копия, уменьшенная в два раза. После к каждому концу (их уже 16) пририсовывается копия буквы Н, уменьшенная уже в 4 раза. И так далее.

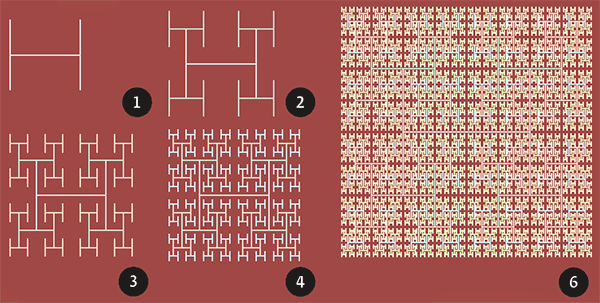


Рис. 10. Первые итерации построения Н-квадрата.

***8. Круговой фрактал***

Круговой фрактал — класс геометрических (конструктивных) фракталов, построенных многократным вписыванием в окружность других окружностей меньшего радиуса.

В окружность радиуса R вписывают семь окружностей радиуса R/3 таким образом, чтобы они все касались, но не пересекали друг друга. В каждую из этих семи окружностей вписываются по семь окружностей R/9 и т. д (рис. 11).

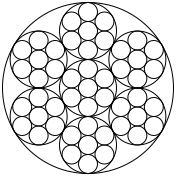
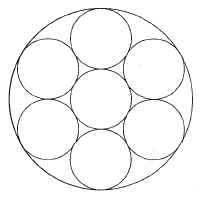
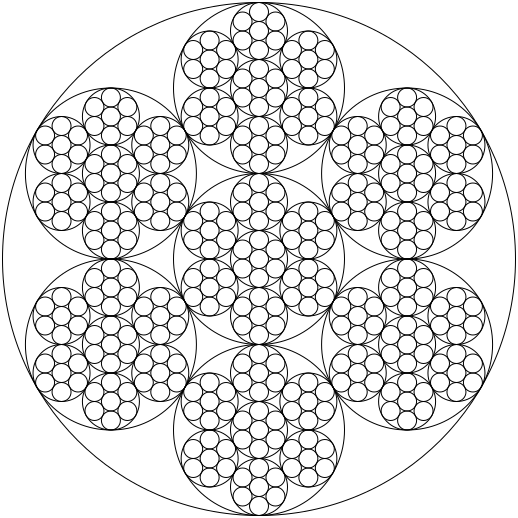
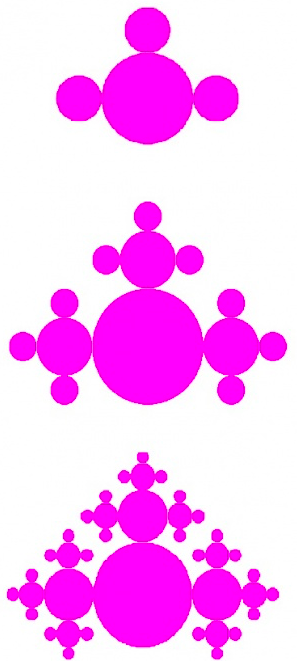
 

Рис. 11. Первые итерации построения кругового фрактала.



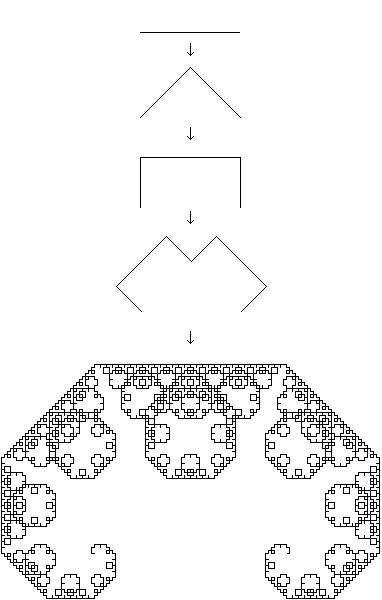
***9. Квазиклевер***

Изначально имеем круг некоторого радиуса (назовем его родителем). Затем слева, сверху и снизу от родителя касаются его круги-потомки с радиусом вдвое меньше, а далее для каждого последующего потомка процедура повторяется по 3-м точкам касания (4-я точка касания с родителем).

Данную процедуру демонстрирует рис. 12 ниже.

Рис. 12. Первые итерации построения квазиклевера.

При реализации фрактала учесть, что пользователь должен иметь возможность задавать, какая из точек касания не будет задействована при первой итерации. К примеру, на рис. 12 не задействована восточная точка касания (справа).

***10. С-Кривая Леви***

Для построения С-кривой Леви необходимо взять половину квадрата вида ᐸ, а затем каждую сторону фигуры заменить таким же фрагментом, повторяя эту операцию (рис. 13).

Рис. 13. Первые итерации построения С-кривой Леви.

***11. Кривая Гильберта***

Кривая Гильберта — это непрерывная кривая, заполняющая пространство. Эти кривые также являются фракталами, они самоподобны; если вы увеличите масштаб и внимательно посмотрите на часть кривой более высокого порядка, то вы увидите, что она выглядит так же, как сама кривая.

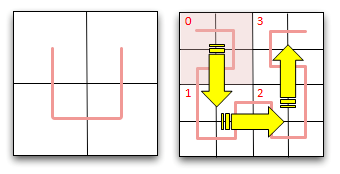
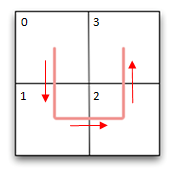


Рис. 14. Первые итерации построения С-кривой Леви.

Основным элементом кривой Гильберта является П-образный элемент (рис. 14). Здесь у нас квадратная сетка 2x2. Начнем с левого верхнего угла и проведем веревку через другие три квадрата сетки, закончив в правом верхнем углу. Теперь представьте, что мы удваиваем размер сетки, получая сетку 4x4. Мы можем представить эту сетку 4x4 в виде набора 4 = 2x2 сеток, каждая из которых имеет размер 2x2.

Мы хотим пересечь сетку более высокого уровня в таком порядке 0->1->2->3 (это показано большими желтыми стрелками), а затем внутри этой сетки используем все тот же П-образный шаблон обхода для каждой из сеток меньшего уровня.

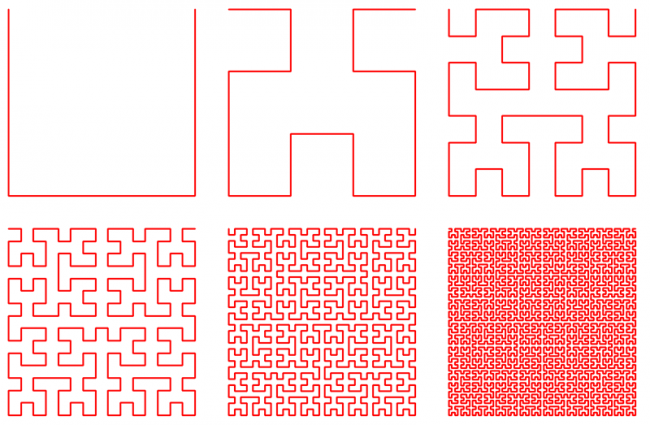


Рис. 15. Первые итерации построения кривой Гильберта.

***12. Фрактал “Центр масс треугольника”***

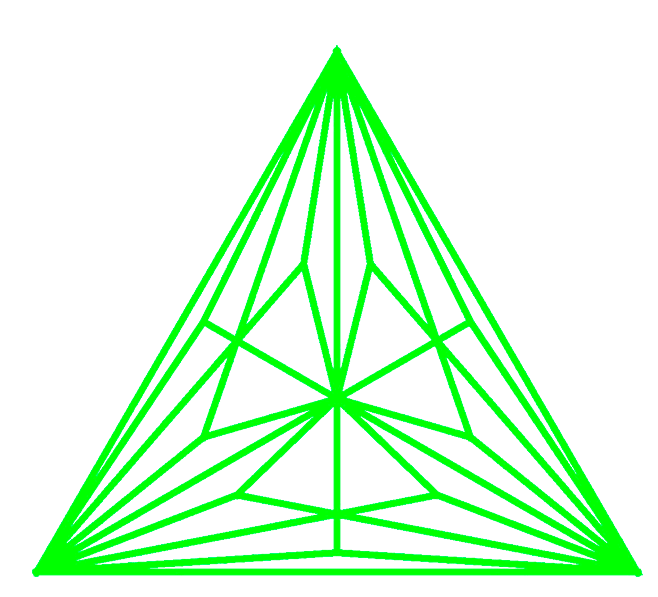
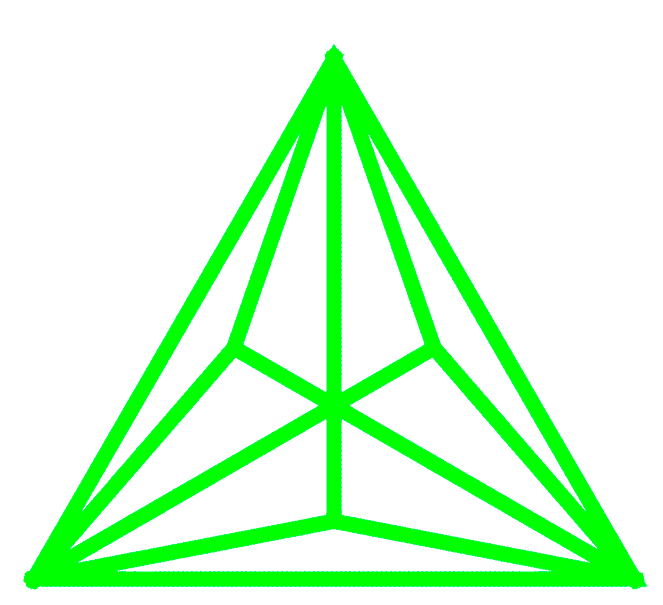
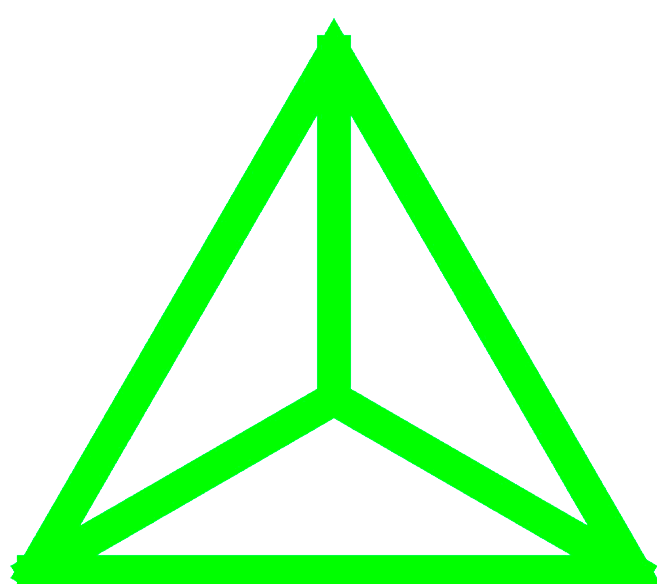


Рис. 16. Первые итерации построения фрактала “Центр масс треугольника”.

Данный фрактал строится на основе равностороннего треугольника.

Построение:

Находится центр масс произвольного треугольника (точка пересечения медиан треугольника), далее к нему проводятся отрезки из вершин треугольника, тем самым разбивая его на три новых треугольника. После этого процедура применяется рекурсивно к полученным треугольникам.

# 

1. <https://www.umgum.com/gitlab-workflow> [↑](#footnote-ref-1)