Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

кафедра Информатики

Дисциплина: Программирование

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе

на тему

Фракталы в среде Unity

БГУИР КП 1-40 04 01

Студент: гр. 953505 Басенко К. А.

Руководитель: ассистент кафедры информатики Клыбик П. М.

Минск 2020

Содержание

Введение…………………………………………………………………………..3

1. Анализ предметной области………………………………………………….4
   1. Теория……………………………………………………………………..4
   2. Анализ существующих аналогов………………………………………..6
   3. Постановка задачи……………………………………………………......9
2. Разработка программы для просмотра фракталов…………………………10
   1. Виды реализуемых фракталов………………………………………….10
   2. Требования к программе………………………………………………..14
   3. Принцип работы программы…………………………………………...15
3. Методика работы с полученной программой……………………………...16

Заключение……………………………………………………………………....18

Исходный код……………………………………………………………………19

Введение

***Фракталы вокруг нас повсюду, и в очертаниях гор, и в извилистой линии морского берега. Некоторые из фракталов непрерывно меняются, подобно движущимся облакам или мерцающему пламени, в то время как другие, подобно деревьям или нашим сосудистым системам, сохраняют структуру, приобретенную в процессе эволюции.***

Х. О. Пайген и П. Х. Рихтер.

Геометрия, которую мы изучали в школе и которой пользуемся в повседневной жизни, восходит к Эвклиду (примерно 300 лет до нашей эры). Треугольники, квадраты, круги, параллелограммы, параллелепипеды, пирамиды, шары, призмы - типичные объекты, рассматриваемые классической геометрией. Предметы, созданные руками человека, обычно включают эти фигуры или их фрагменты. Однако в природе они встречаются не так уж часто. Действительно, похожи ли, например, лесные красавицы ели на какой-либо из перечисленных предметов или их комбинацию? Легко заметить, что в отличие от форм Эвклида природные объекты не обладают гладкостью, их края изломаны, зазубрены, поверхности шероховаты, изъедены трещинами, ходами и отверстиями. "Почему геометрию часто называют холодной и сухой? Одна из причин заключается в ее неспособности описать форму облака, горы, дерева или берега моря. Облака - это не сферы, горы - не конусы, линии берега - это не окружности, и кора не является гладкой, и молния не распространяется по прямой. Природа демонстрирует нам не просто более высокую степень, а совсем другой уровень сложности", - этими словами начинается "Фрактальная геометрия природы", написанная Бенуа Мандельбротом. Именно он в 1975 году впервые ввел понятие фрактала - от латинского слова fractus, сломанный камень, расколотый и нерегулярный. Оказывается, почти все природные образования имеют фрактальную структуру. Что это значит? Если посмотреть на фрактальный объект в целом, затем на его часть в увеличенном масштабе, потом на часть этой части и т. п., то нетрудно увидеть, что они выглядят одинаково. Фракталы самоподобны - их форма воспроизводится на различных масштабах.

Открытие фракталов произвело революцию не только в геометрии, но и в физике, химии, биологии. Фрактальные алгоритмы нашли применение и в информационных технологиях, например, для синтеза трехмерных компьютерных изображений природных ландшафтов, для сжатия (компрессии) данных (см. "Наука и жизнь" № 4, 1994 г.; №№ 8, 12, 1995 г.; № 7, 1998 г.). Далее мы убедимся, что понятие фрактала тесно связано с еще одним не менее любопытным явлением - хаосом в динамических системах.

1. Анализ предметной области

1.1 Теория

Фрактал (лат. fractus — дробленый, сломанный, разбитый) — сложная геометрическая фигура, обладающая свойством самоподобия, то есть составленная из нескольких частей, каждая из которых подобна всей фигуре целиком. В более широком смысле под фракталами понимают множества точек в евклидовом пространстве, имеющие дробную метрическую размерность (в смысле Минковского или Хаусдорфа), либо метрическую размерность, строго большую топологической.

**Термин**

Следует отметить, что слово «фрактал» не является математическим термином и не имеет общепринятого строгого математического определения. Оно может употребляться, когда рассматриваемая фигура обладает какими-либо из перечисленных ниже свойств:

* Обладает нетривиальной структурой на всех шкалах. В этом отличие от регулярных фигур (таких, как окружность, эллипс, график гладкой функции): если мы рассмотрим небольшой фрагмент регулярной фигуры в очень крупном масштабе, он будет похож на фрагмент прямой. Для фрактала увеличение масштаба не ведёт к упрощению структуры, на всех шкалах мы увидим одинаково сложную картину.
* Является самоподобной или приближённо самоподобной.
* Обладает дробной метрической размерностью или метрической размерностью, превосходящей топологическую.

Многие объекты в природе обладают фрактальными свойствами, например, побережья, облака, кроны деревьев, кровеносная система и система альвеол человека или животных.

Фракталы, особенно на плоскости, популярны благодаря сочетанию красоты с простотой построения при помощи компьютера.

Многие объекты в природе обладают фрактальными свойствами, например, побережья, облака, кроны деревьев, кровеносная система и система альвеол человека или животных.

Фракталы, особенно на плоскости, популярны благодаря сочетанию красоты с простотой построения при помощи компьютера.

**История**

Первые примеры самоподобных множеств с необычными свойствами появились в XIX веке (например, множество Кантора). Термин «фрактал» был введён Бенуа Мандельбротом в 1975 году и получил широкую популярность с выходом в 1977 году его книги «Фрактальная геометрия природы».

**Классификация**

Алгебраические фракталы:

* Множество Мандельброта
* Множество Жюлиа
* Бассейны (фракталы) Ньютона
* Биоморфы

Геометрические фракталы:

* Кривая Коха (снежинка Коха)
* Кривая Леви
* Кривая Гильберта
* Ломаная (кривая) дракона (Фрактал Хартера-Хейтуэя)
* Множество Кантора
* Треугольник Серпинского
* Ковер Серпинского
* Дерево Пифагора

## Применение

### Экономика: *Анализ рынков*

Последнее время фракталы стали популярным инструментом у трейдеров для анализа состояния биржевых рынков.

### Естественные науки

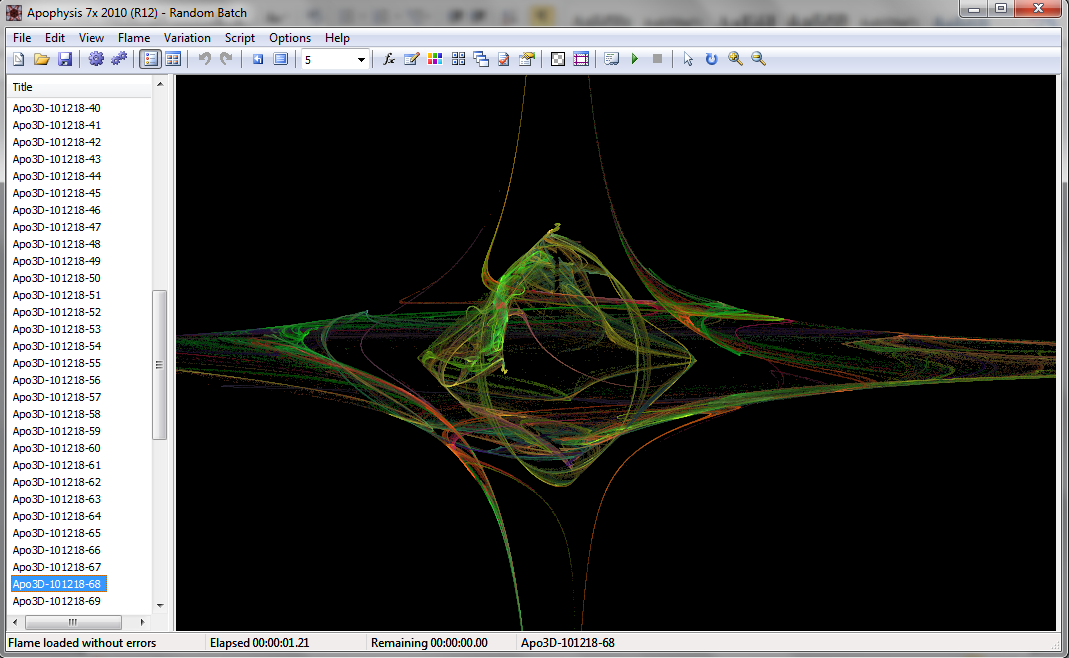
В физике фракталы естественным образом возникают при моделировании нелинейных процессов, таких, как турбулентное течение жидкости, сложные процессы диффузии-адсорбции, пламя, облака и т. п. Фракталы используются при моделировании пористых материалов, например, в нефтехимии. В биологии они применяются для моделирования популяций и для описания систем внутренних органов (система кровеносных сосудов).

1.2. Анализ существующих аналогов

Сегодня в глобальной сети интернет можно найти множество программ для работы с фракталами. Есть все, начиная простым построением фракталов и заканчивая фрактальным сжатием изображений. Нас интересует построение фракталов, поэтому рассмотрим следующий перечень программ: *Apophysis 7x, Fractal Fripperies Fractal Explorer, FractalEditor, FractInt*

***Первая программа Apophysis 7x.***

Интерфейс выглядит так:



Плюсы программы:

* Программа бесплатная
* Огромная коллекция фракталов
* Множество настроек
* «Режим моделирования» фракталов.

Минусы:

* Относительно долгая прорисовка фракталов большого разрешения.

***Вторая программа Fractal Fripperies Fractal Explorer.***

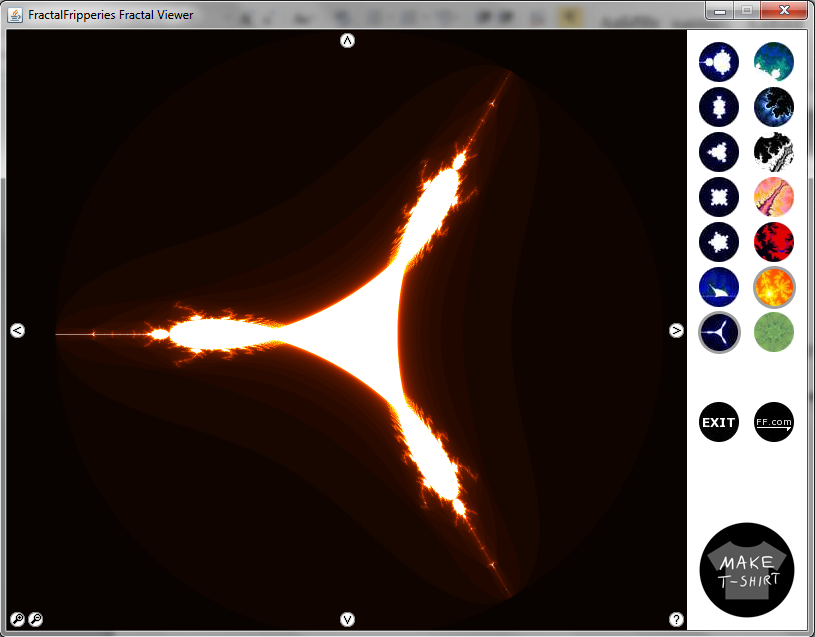
Плюсы программы:

* Интересная организация и дизайн
* Четкая отрисовка фракталов
* Возможность многократно приближать фракталы
* Предусмотрена возможность заказа майки с сгенерированным фракталом

Минусы:

* Ограниченная коллекция фракталов

Интерфейс программы:



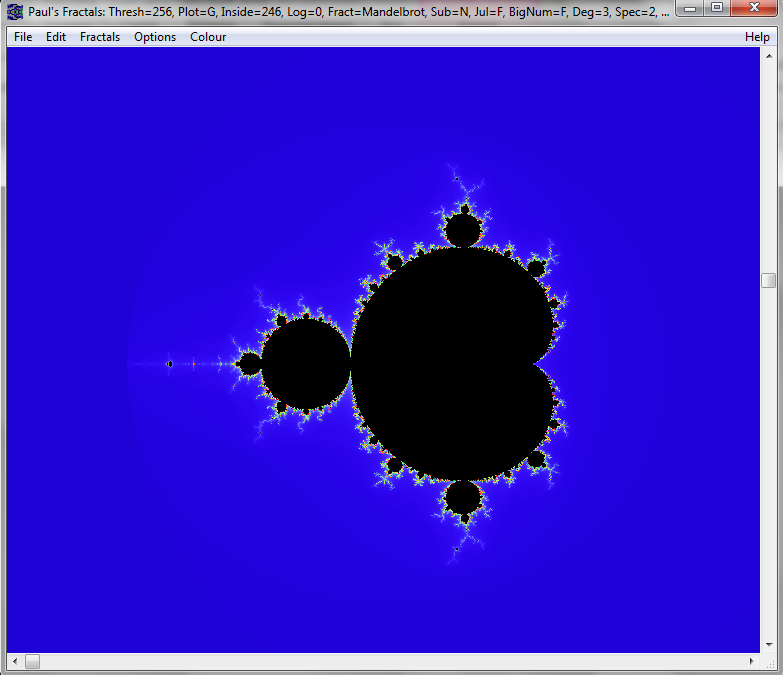
***Третья программа Manpwin***

Плюсы:

* Приятный интерфейс
* Хороший функционал
* Интересная работа с цветом
* Множество настроек

Минусы:

* Не обнаружены



***Четвертая программа FractInt***

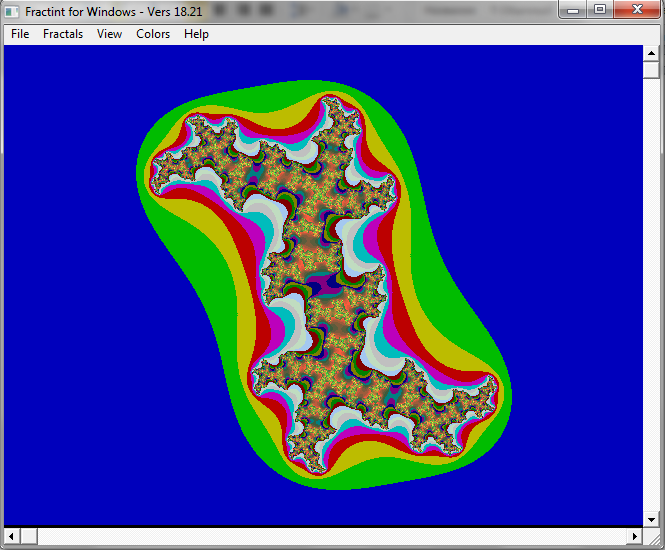
Плюсы:

* Быстрая скорость работы
* Большая коллекция фракталов
* Наличие настроек

Минусы:

* Не обнаружены

Интерфейс:



1.3 Постановка задачи

В задачу курсового проекта входит:

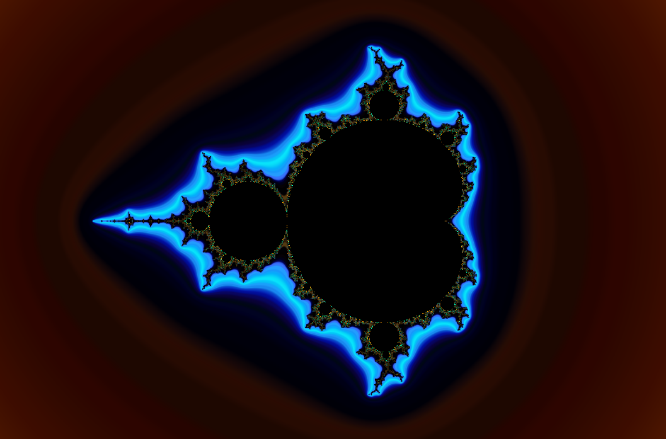
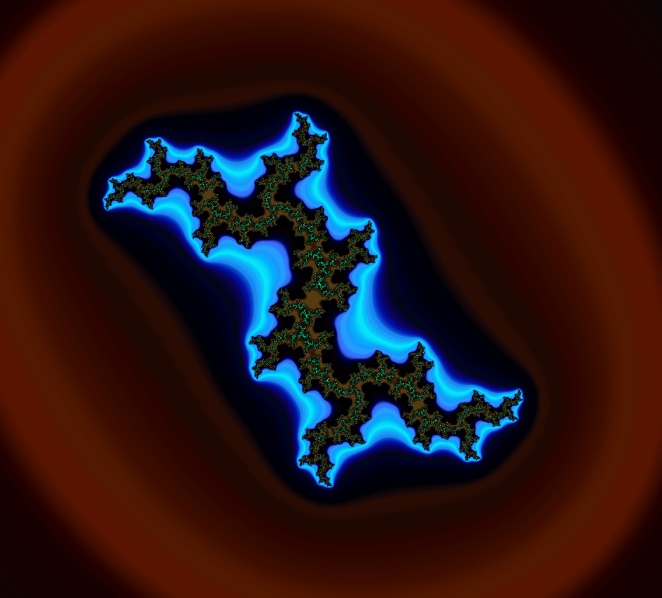
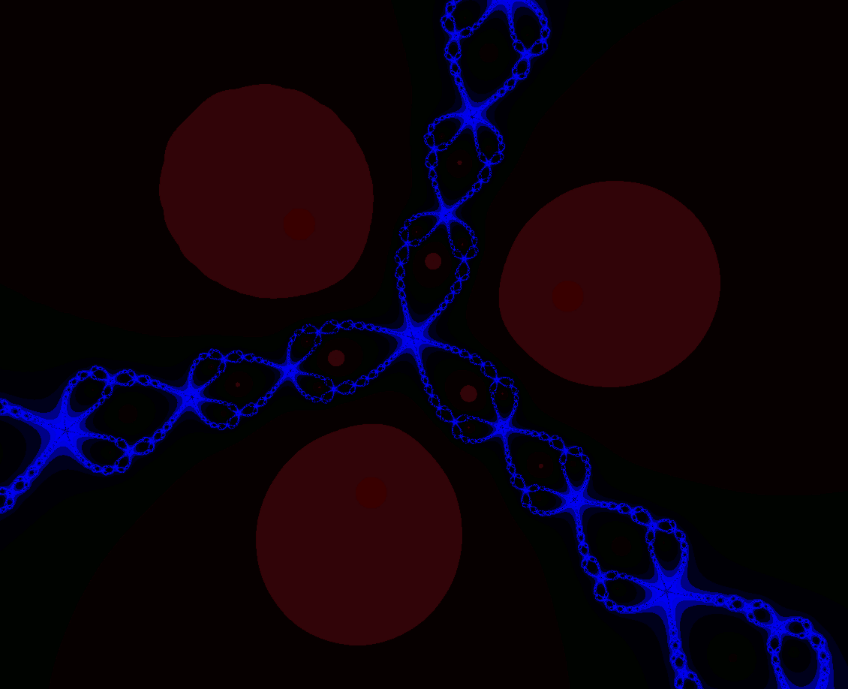
* Разработать программное средство с помощью Unity, способное выполнять построение фракталов.
* Программное средство должно выполнять следующие функции:
  + Построение фрактала
  + Изменение параметров, соответствующих построенному фракталу.
  + Сброс параметров фрактала.
* Реализовать понятный и удобный для пользователя интерфейс программного средства.

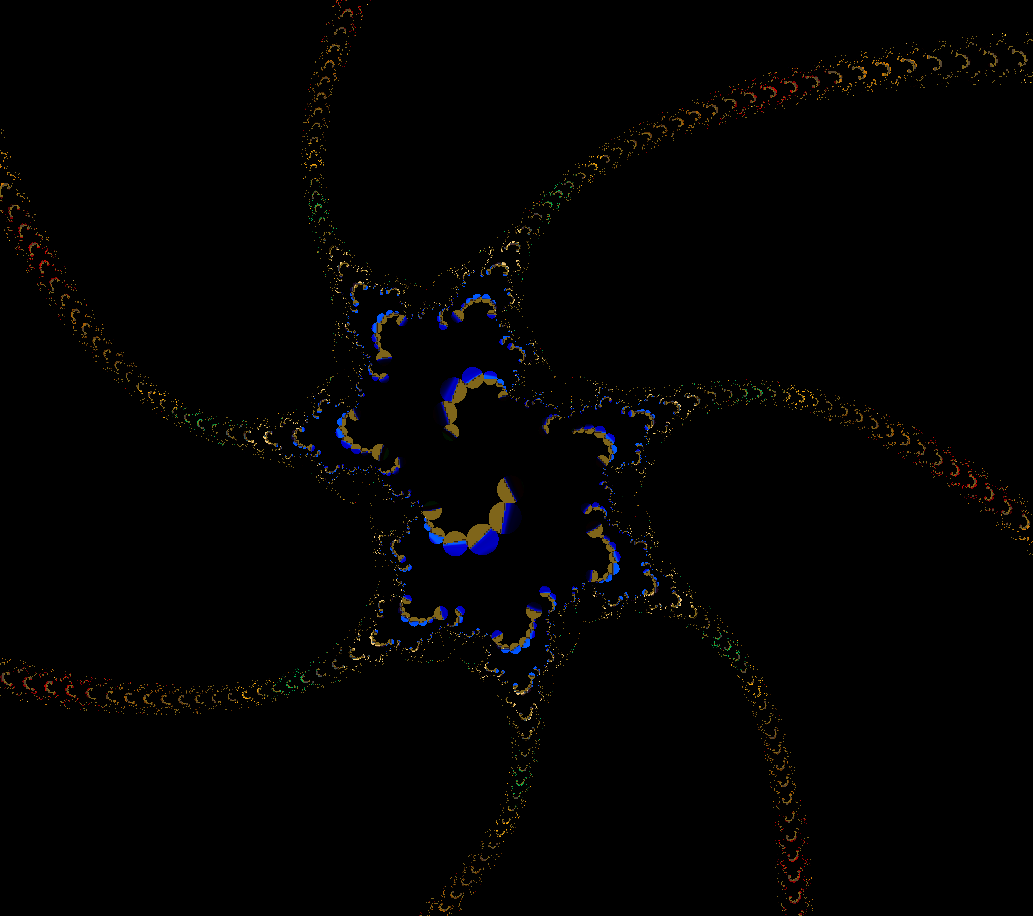
Таким образом, задача данного курсового проекта сводится к разработке программного средства, способного строить простые геометрические и алгебраические фракталы и выводить их на экран, а также обеспечению возможности дальнейшей работы с полученным изображением фрактала, например в среде AdobePhotoshop для получения конечного изображения, необходимого для создания, например логотипа, витража и т.д.

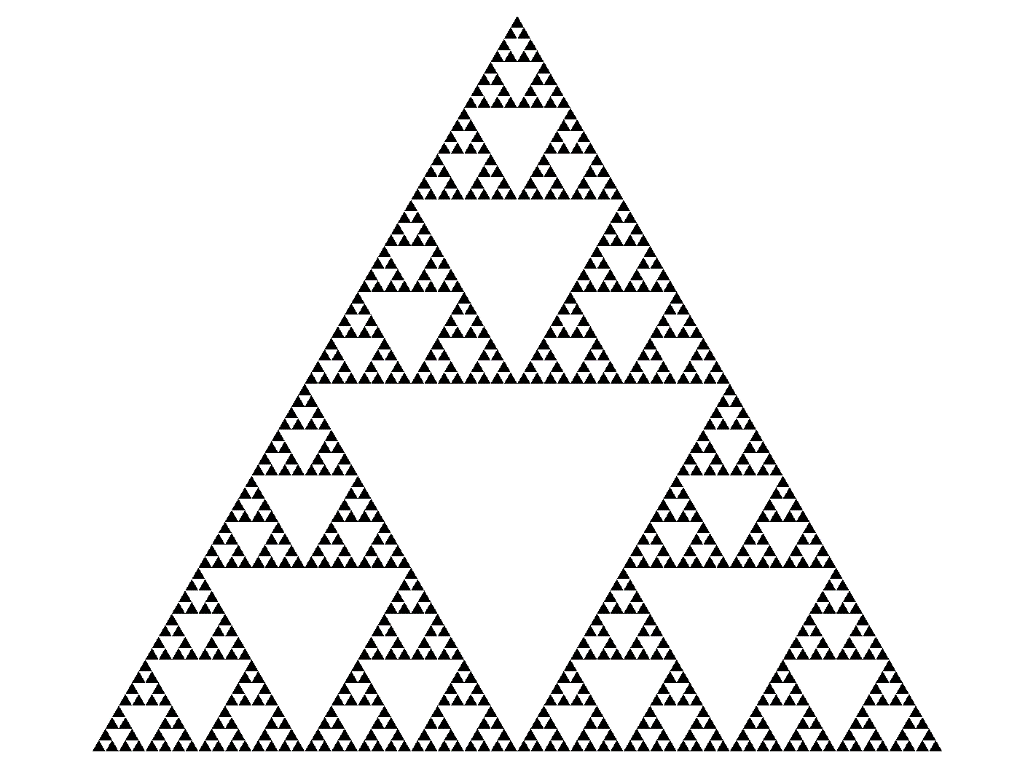
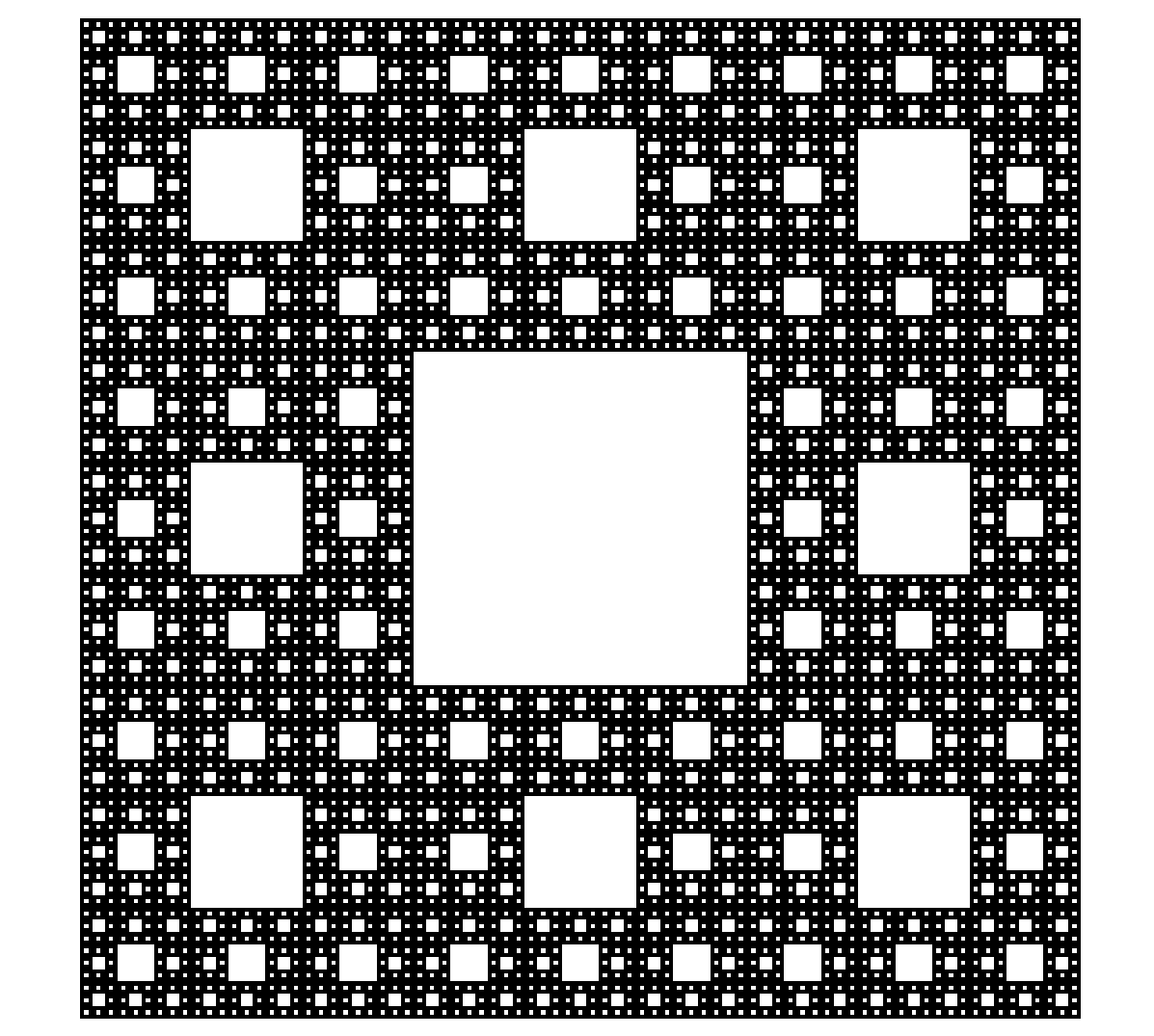
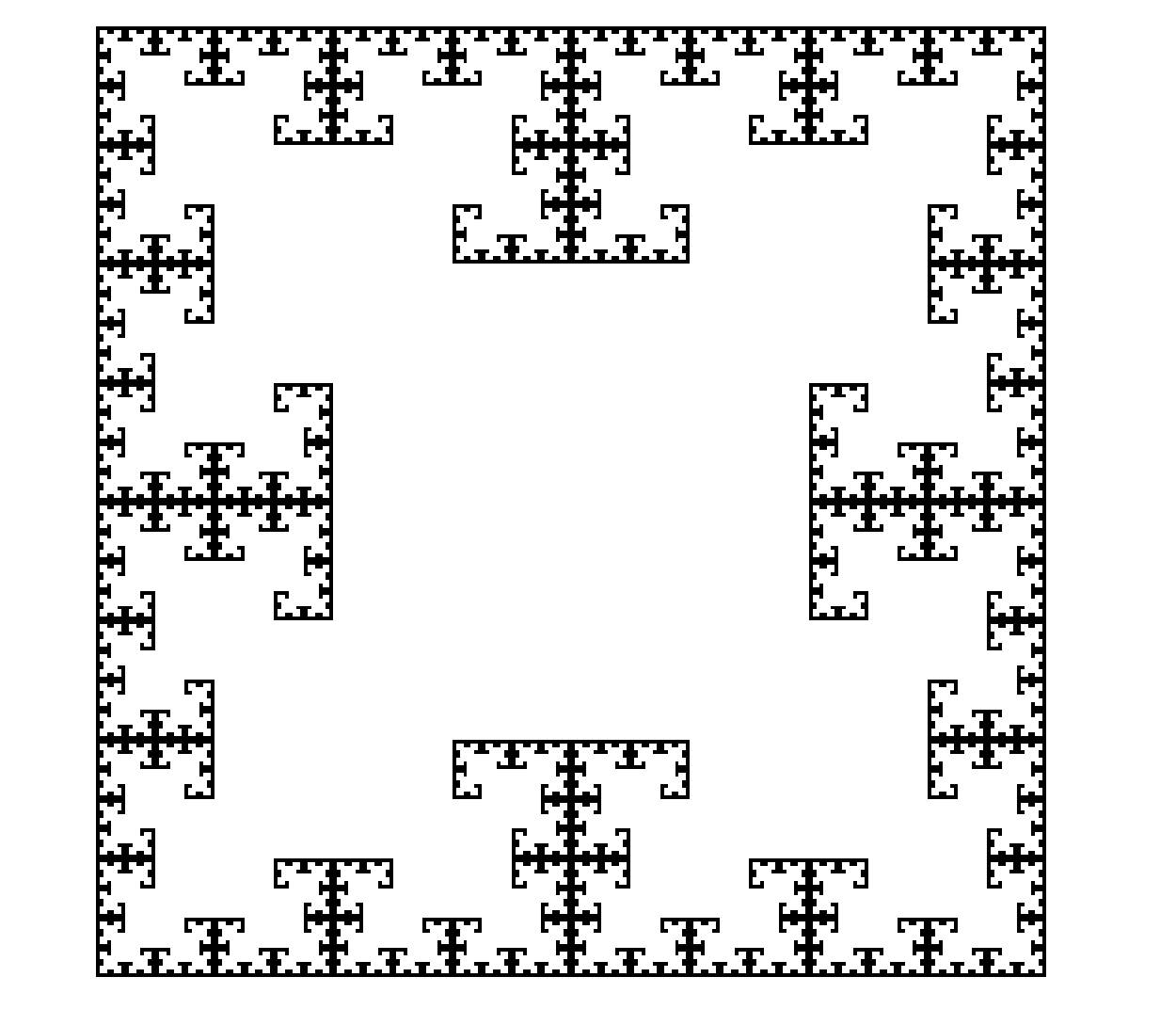
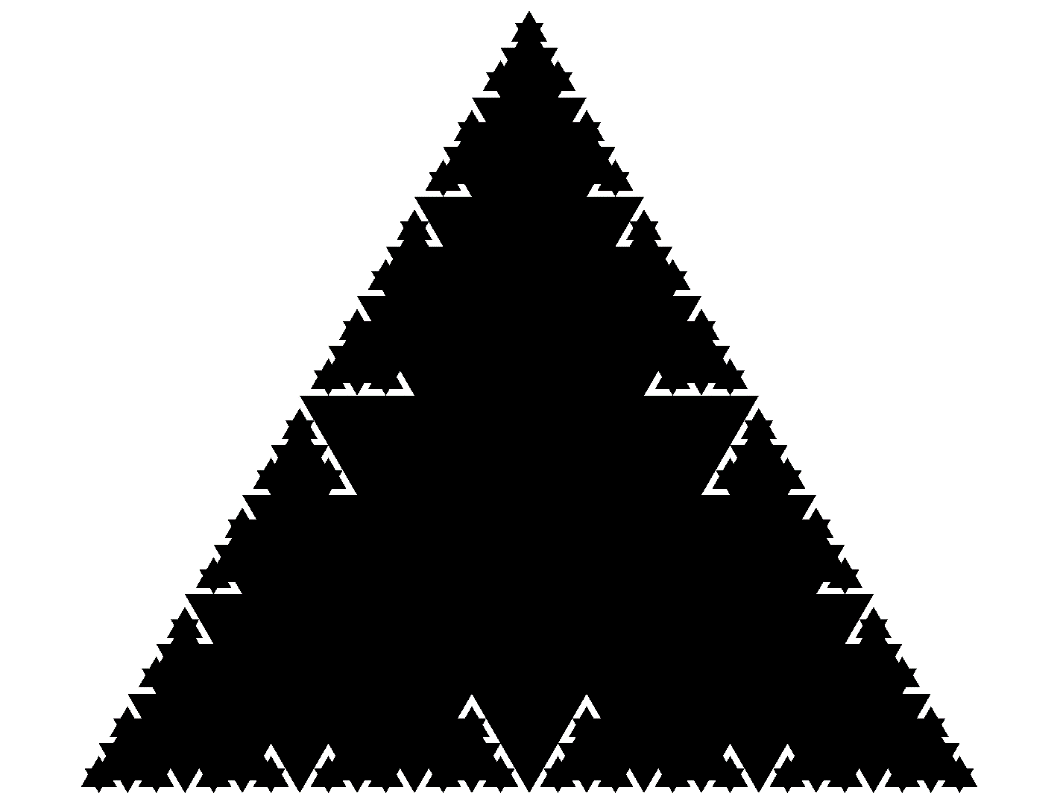
2. Разработка программы для просмотра фракталов

2.1 Виды реализуемых фракталов

В программе планируется реализовать следующие виды простых геометрических фракталов:

*  Множество Мандельброта
*  Множество Жюлиа
*  Фрактал Ньютона
* «Неправильный» фрактал Ньютона



*  Треугольник Серпинского
*  Квадрат Серпинского
*  T-квадрат
*  «Елочка»

2.2 Требования к программе для построения фракталов

Необходимо создать программу, используя Unity, которая смогла бы строить фракталы и предоставлять возможность дальнейшей работы с нарисованным фракталом. В связи с этим к программе предъявляются следующие требования:

1. Интерфейс.

Интерфейс программного средства будет содержать рабочую область, в которой будет отображаться сам фрактал. Так же будет отдельное меню, в котором будут находиться некоторые параметры, соответствующие данному фракталу.

Так же должно быть меню, в котором будут находиться пункты, соответствующие выполняемым функциям. Так же в процессе вызова некоторых функций, пользователю будет представлена возможность ввода дополнительных параметров настроек изображения.

1. Функциональность.

Данная программа должна выполнять следующие требования:

* + Предоставлять пользователю выбор фрактала для построения из меню
  + Изменять некоторые параметры построенного фрактала.
  + Сброс настроек фрактала.

1. Оптимальность используемых алгоритмов. Все используемые алгоритмы были протестированы на оптимальность, однако дальнейшее их улучшение имеет смысл.
2. Совместимость. Данное программное средство совместимо с операционными системами семейства Windows.
3. Среда разработки. Средой разработки данного программного средства выбран Unity. Выбор был сделан на основе желания поработать с чем-то новым.

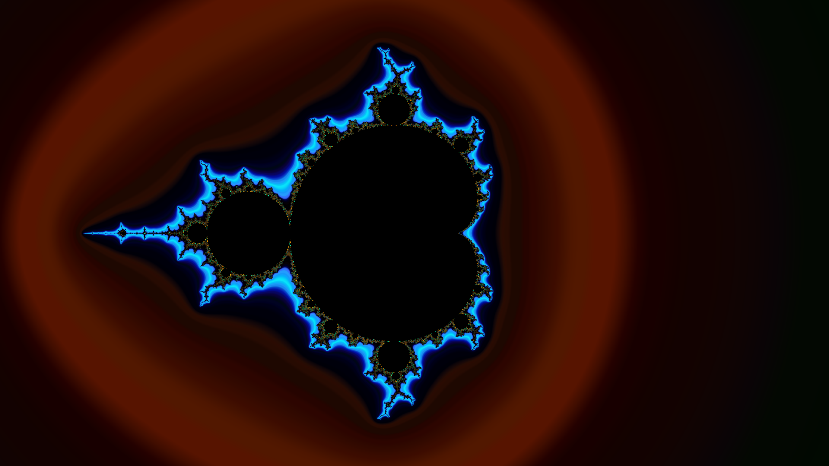
2.3 Принцип работы

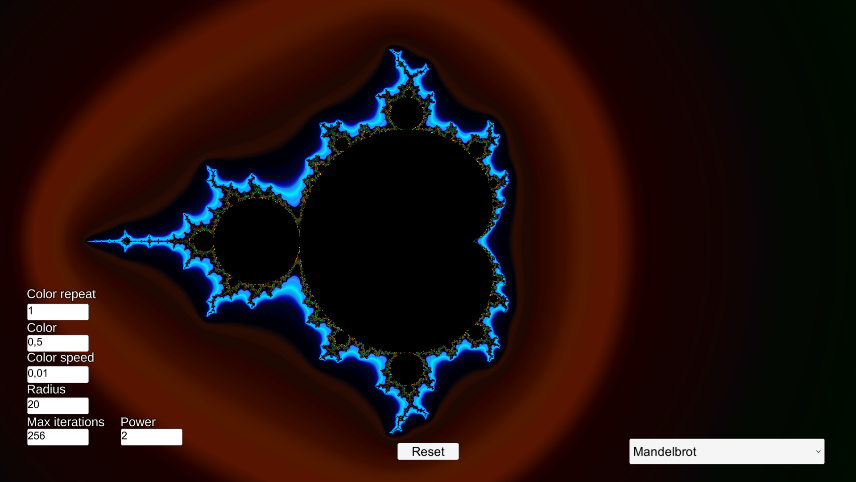
Фракталы прорисовываются с помощью шейдеров, наложенных на материал для картинки(canvas).

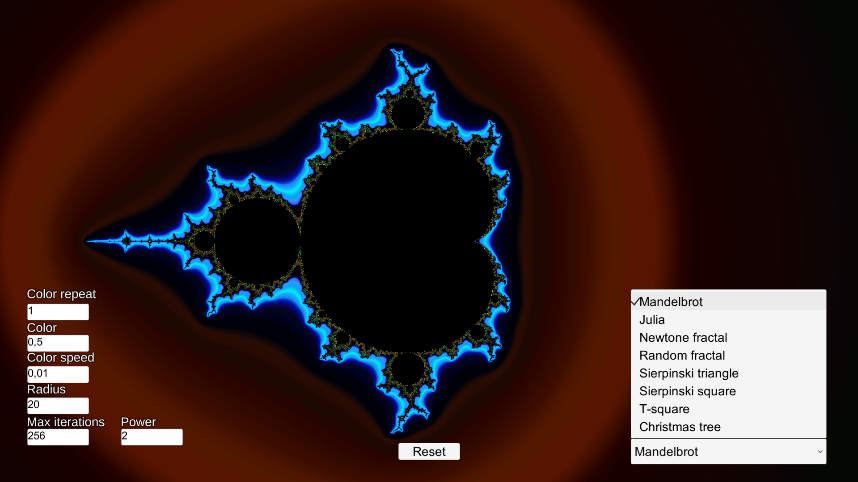
Пользователь выбирает фрактал, нажимает на его, выполняется обработка события, в ходе которого текущий шейдер заменяется на шейдер нужного нам фрактала. Рисуется выбранный фрактал в рабочей области.

В программе имеются вспомогательные функции: настройка некоторых параметров фракталов, сброс параметров фрактала к начальным.

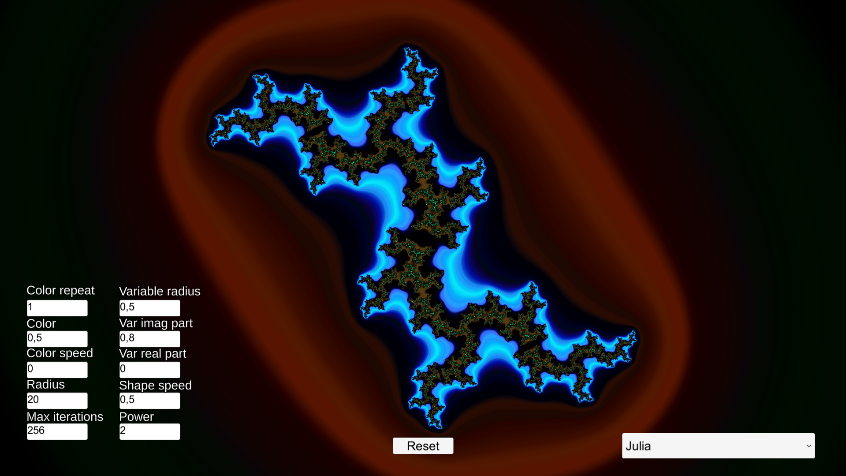
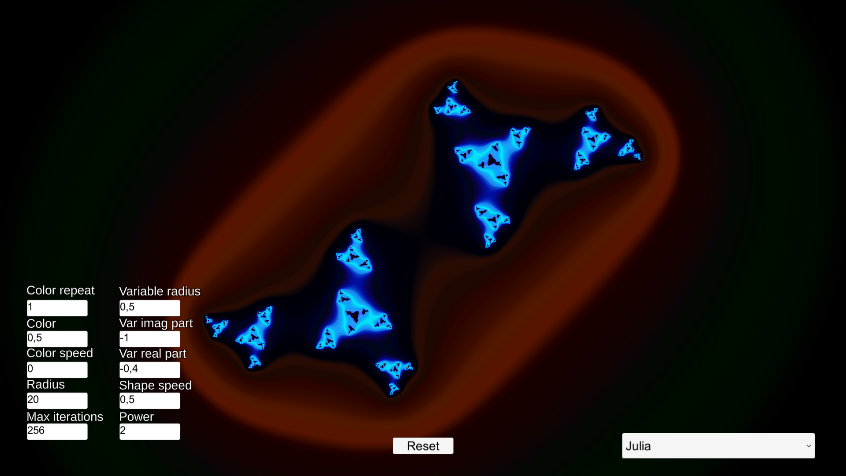
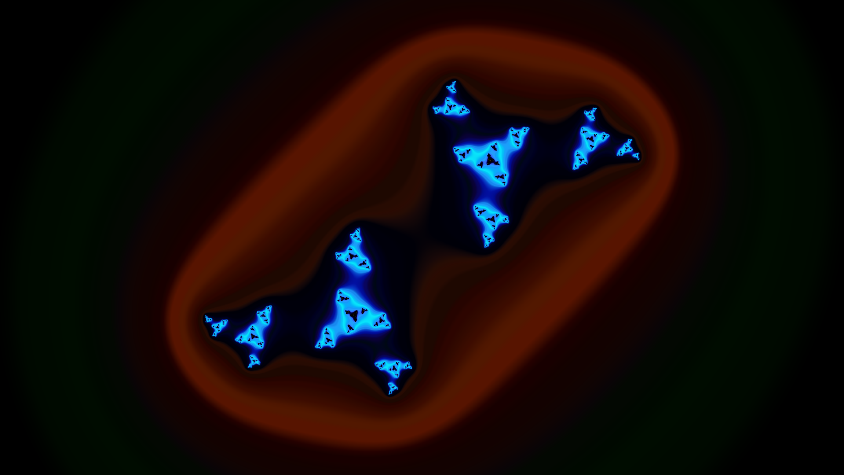
3. Методика работы с программой

Интерфейс данной программы состоит из меню параметров, которое по желанию можно вызвать на пробел, и рабочей области, где рисуются фракталы. При запуске программы нас будет встречать множество Мандельброта.

Меню выбора фракталов и параметров текущего фрактала выглядит так:



Продемонстрируем работу с программой:

Выберем пункт множество Жюлиа и изменим пару начальных параметров.

Дальше пускаем в бой фантазию.

Заключение

В результате работы над курсовым проектом было разработано программное средство для построения фракталов.

В ходе работы над проектом было изучено много информации по теме фракталов и фрактальной математики. Были рассмотрены принципы построения геометрических фракталов, и выбраны наиболее интересные и оптимальные варианты для их последующей реализации в алгоритмах программы.

Для каждого фрактала были созданы отдельные шейдеры, которые содержали свои методы и переменные для построения выбранного фрактала.

Также были рассмотрены аналоги данной программы: Apophysis 7x, Fractal Fripperies Fractal Explorer, Fractal Editor, FractInt. Проанализированы их плюсы и недостатки, и выбраны именно те качества, наличие которых хотелось в данной программе. Это понятный пользователю интерфейс, возможность выбора фрактала из списка для дальнейшего его построение.

Были изучены возможности работы с графикой в Unity, а именно работа с шейдерами для материала для картинок(canvas).

Разработанное программное средство представляет собой законченный продукт, готовый к использованию. Однако при желании функционал программы можно расширить, добавив новые функции, например сохранение полученного изображения в файл, можно написать алгоритмы для построения новых фракталов, так как в программе реализованы далеко не все возможные фракталы. Так же в качестве улучшения можно рассматривать переход от 2D изображений фракталов к 3D. Многие из фракталов очень красивы, поэтому можно рассматривать использование программы в качестве заставки на рабочий стол.

Исходный код

Шейдер множества Мандельброта.  
  
Shader "Fractals/Mandelbrot"

{

    Properties

    {

        \_MainTex("Texture", 2D) = "white" {}

        \_Area("Area", vector) = (0, 0, 4, 4)

        \_MaxIter("Max iterations", float) = 100

        \_Color("Color", range(0, 1)) = .5

        \_Repeat("Repeat", float) = 1

        \_Speed("Speed", range(0, 1)) = .01

        \_Radius("Radius", float) = 20

         \_Power("Power", float) = 2

    }

        SubShader

        {

            // No culling or depth

            Cull Off ZWrite Off ZTest Always

            Pass

            {

                CGPROGRAM

                #pragma vertex vert

                #pragma fragment frag

                #include "UnityCG.cginc"

                struct appdata

                {

                    float4 vertex : POSITION;

                    float2 uv : TEXCOORD0;

                };

                struct v2f

                {

                    float2 uv : TEXCOORD0;

                float4 vertex : SV\_POSITION;

                };

                v2f vert(appdata v)

                {

                    v2f o;

                    o.vertex = UnityObjectToClipPos(v.vertex);

                    o.uv = v.uv;

                    return o;

                }

                sampler2D \_MainTex;

                float4 \_Area;

                float \_MaxIter;

                float \_Color;

                float \_Repeat;

                float \_Speed;

                float \_Radius;

                float \_Power;

                float AngleOf(float2 num)

                {

                    float R = num.x;

                    float I = num.y;

                    if ((R > 0) && (I >= 0))

                    {

                        return (atan(I / R));

                    }

                    else if ((R < 0) && (I >= 0))

                    {

                        return (3.14159 - atan(abs(I / R)));

                    }

                    else if ((R < 0) && (I < 0))

                    {

                        return (3.14159 + atan(abs(I / R)));

                    }

                    else if ((R > 0) && (I < 0))

                    {

                        return (2 \* 3.14159 - atan(abs(I / R)));

                    }

                    else if ((R == 0) && (I > 0))

                    {

                        return (3.14159 / 2);

                    }

                    else// if ((R == 0) && (I < 0))

                    {

                        return ((3 / 2) \* 3.14159);

                    }

                }

                float2 powi(float2 z)

                {

                    float angle = AngleOf(z);

                    return (pow(length(z), \_Power) \* float2(cos(\_Power \* angle), sin(\_Power \* angle))); //z in power \_Power

                }

                float2 f(float2 z)

                {

                    return powi(z);

                }

                fixed4 frag(v2f i) : SV\_Target

                {

                    float2 c = (\_Area.xy +  (i.uv - .5) \* \_Area.zw);

                    float2 z;

                    float iter;

                    for (iter = 0; iter < \_MaxIter; iter++)

                    {

                        z = f(z) + c;

                        if (length(z) > \_Radius)

                        {

                            break;

                        }

                    }

                    if (iter >= \_MaxIter)

                    {

                        return 0;

                    }

                    iter -= log2(log(length(z)) / log(\_Radius)); //double exponential interpolation

                    return tex2D(\_MainTex, float2(sqrt(iter / \_MaxIter) \* \_Repeat + (\_Time.y - 2) \* \_Speed, \_Color));

                }

                ENDCG

            }

        }

}

Шейдер множества Жюлиа.  
  
Shader "Fractals/Julia"

{

    Properties

    {

        \_MainTex("Texture", 2D) = "white" {}

        \_Area("Area", vector) = (0, 0, 4, 4)

        \_MaxIter("Max iterations", float) = 100

        \_Color("Color", range(0, 1)) = .5

        \_Repeat("Repeat", float) = 1

        \_Speed("Speed", range(0, 1)) = .01

        \_Radius("Max radius of C num", float) = 20

        \_Power("Power", float) = 2

        \_R("Real part", float) = 0

        \_I("Imaginary part", float) = .8

        \_ShapeSpeed("Shape speed", float) = .5

        \_CircleRadius("Radius variable of circle", float) = .5

    }

        SubShader

        {

            // No culling or depth

            Cull Off ZWrite Off ZTest Always

            Pass

            {

                CGPROGRAM

                #pragma vertex vert

                #pragma fragment frag

                #include "UnityCG.cginc"

                struct appdata

                {

                    float4 vertex : POSITION;

                    float2 uv : TEXCOORD0;

                };

                struct v2f

                {

                    float2 uv : TEXCOORD0;

                    float4 vertex : SV\_POSITION;

                };

                v2f vert(appdata v)

                {

                    v2f o;

                    o.vertex = UnityObjectToClipPos(v.vertex);

                    o.uv = v.uv;

                    return o;

                }

                sampler2D \_MainTex;

                float4 \_Area;

                float \_MaxIter;

                float \_Color;

                float \_Repeat;

                float \_Speed;

                float \_Radius = 20.;

                float \_Power;

                float \_R;

                float \_I;

                float \_ShapeSpeed;

                float \_CircleRadius;

                float \_R2;

                float \_I2;

                float AngleOf(float2 num)

                {

                    float R = num.x;

                    float I = num.y;

                    if ((R > 0) && (I >= 0))

                    {

                        return (atan(I / R));

                    }

                    else if ((R < 0) && (I >= 0))

                    {

                        return (3.14159 - atan(abs(I / R)));

                    }

                    else if ((R < 0) && (I < 0))

                    {

                        return (3.14159 + atan(abs(I / R)));

                    }

                    else if ((R > 0) && (I < 0))

                    {

                        return (2 \* 3.14159 - atan(abs(I / R)));

                    }

                    else if ((R == 0) && (I > 0))

                    {

                        return (3.14159 / 2);

                    }

                    else// if ((R == 0) && (I < 0))

                    {

                        return ((3 / 2) \* 3.14159);

                    }

                }

                float2 powi(float2 z)

                {

                    float angle = AngleOf(z);

                    return (pow(length(z), \_Power) \* float2(cos(\_Power \* angle), sin(\_Power \* angle))); //z in power \_Power

                }

                float2 f(float2 z)

                {

                    return powi(z);

                }

                fixed4 frag(v2f i) : SV\_Target

                {

                    float2 c = \_Area.xy + (i.uv - .5f) \* \_Area.zw;

                    float2 RI;

                    if (\_ShapeSpeed != 0)

                    {

                        \_R2 = \_CircleRadius \* sin(\_Time.y \* \_ShapeSpeed);

                        \_I2 = \_CircleRadius \* cos(\_Time.y \* \_ShapeSpeed);

                        RI = float2(\_R2, \_I2);

                    }

                    float2 z = c;

                    float iter;

                    for (iter = 0; iter < \_MaxIter; iter++)

                    {

                        if ((\_I != 0) || (\_R != 0))

                        {

                            z = f(z) + float2(\_R, \_I);

                        }

                        else

                        {

                            z = f(z) + RI;

                        }

                        if (length(z) > \_Radius)

                        {

                            break;

                        }

                    }

                    if (iter >= \_MaxIter)

                    {

                        return 0;

                    }

                    iter -= log2(log(length(z)) / log(\_Radius)); //double exponential interpolation

                    float4 col = tex2D(\_MainTex, float2(sqrt(iter / \_MaxIter) \* \_Repeat + (\_Time.y - 2) \* \_Speed, \_Color));

                    return col;

                }

                ENDCG

            }

        }

}

Шейдер фрактала Ньютона

Shader "Fractals/Newtone fractal"

{

Properties

{

\_MainTex ("Texture", 2D) = "white" {}

\_Area("Area", vector) = (0, 0, 4, 4)

\_MaxIter("Max iterations", float) = 100

\_Color("Color", range(0, 1)) = .5

\_Repeat("Repeat", float) = 1

\_Speed("Speed", range(0, 1)) = .01

\_Radius("Radius", float) = 20

\_Power("Power", float) = 2

\_R("Real part", float) = 0

\_I("Imaginary part", float) = .8

\_ShapeSpeed("Shape speed", float) = .5

\_CircleRadius("Radius variable of circle", float) = .5

}

SubShader

{

// No culling or depth

Cull Off ZWrite Off ZTest Always

Pass

{

CGPROGRAM

#pragma vertex vert

#pragma fragment frag

#include "UnityCG.cginc"

struct appdata

{

float4 vertex : POSITION;

float2 uv : TEXCOORD0;

};

struct v2f

{

float2 uv : TEXCOORD0;

float4 vertex : SV\_POSITION;

};

v2f vert (appdata v)

{

v2f o;

o.vertex = UnityObjectToClipPos(v.vertex);

o.uv = v.uv;

return o;

}

sampler2D \_MainTex;

float4 \_Area;

float \_MaxIter;

float \_Color;

float \_Repeat;

float \_Speed;

float \_Radius;

float \_Power;

float \_R;

float \_I;

float \_ShapeSpeed;

float \_CircleRadius;

float AngleOf(float2 num)

{

float R = num.x;

float I = num.y;

if ((R > 0) && (I >= 0))

{

return (atan(I / R));

}

else if ((R < 0) && (I >= 0))

{

return (3.14159 - atan(abs(I / R)));

}

else if ((R < 0) && (I < 0))

{

return (3.14159 + atan(abs(I / R)));

}

else if ((R > 0) && (I < 0))

{

return (2 \* 3.14159 - atan(abs(I / R)));

}

else if ((R == 0) && (I > 0))

{

return (3.14159 / 2);

}

else// if ((R == 0) && (I < 0))

{

return ((3 / 2) \* 3.14159);

}

}

float2 powi(float2 z, float power)

{

float angle = AngleOf(z);

return (pow(length(z), power) \* float2(cos(power \* angle), sin(power \* angle))); //z in power \_Power

}

float2 f(float2 z, float power)

{

return powi(z, power) - 1;

}

float2 df(float2 z, float power)

{

return power \* powi(z, power - 1);

}

float2 cmult(float2 a, float2 b)

{

return float2(a.x \* b.x - a.y \* b.y, a.x \* b.y + a.y \* b.x);

}

float2 cdiv(float2 a, float2 b)

{

return float2((a.x \* b.x + a.y \* b.y) / (b.x \* b.x + b.y \* b.y), (-a.x \* b.y + a.y \* b.x) / (b.x \* b.x + b.y \* b.y));

}

fixed4 frag(v2f i) : SV\_Target

{

float2 c = \_Area.xy + (i.uv - .5f) \* \_Area.zw;

float2 z = c;

float iter;

float iter2;

float2 w;

float2 roots[3];

float \_R2;

float \_I2;

float2 RI;

if (\_ShapeSpeed != 0)

{

\_R2 = \_CircleRadius \* sin(\_Time.y \* \_ShapeSpeed);

\_I2 = \_CircleRadius \* cos(\_Time.y \* \_ShapeSpeed);

RI = float2(\_R2, \_I2);

}

for (iter = 0; (int)iter < (int)\_MaxIter; iter++)

{

w = z;

z -= cmult(cdiv(f(z, \_Power), df(z, \_Power)), float2(1, 0));

if ((\_I != 0) || (\_R != 0))

{

z -= cmult(cdiv(f(z, \_Power), df(z, \_Power)), float2(\_R, \_I));

}

else

{

z -= cmult(cdiv(f(z, \_Power), df(z, \_Power)), RI);

}

if (length(w - z) < \_Radius)

{

break;

}

}

if (iter == \_MaxIter)

{

return 0;

}

iter -= (length(z) - \_Radius) / (\_Radius \* \_Radius - \_Radius); //linear interpolation

return tex2D(\_MainTex, float2(pow(iter / \_MaxIter, 1.1) \* \_Repeat + (\_Time.y - 2) \* \_Speed, \_Color));

}

ENDCG

}

}

}

Шейдер «Неправильного» фрактала Ньютона.

Shader "Fractals/Random fractal"

{

Properties

{

\_MainTex("Texture", 2D) = "white" {}

\_Area("Area", vector) = (0, 0, 4, 4)

\_MaxIter("Max iterations", float) = 256

\_Color("Color", range(0, 1)) = .5

\_Repeat("Repeat", float) = 1

\_Speed("Speed", range(0, 1)) = .01

\_Radius("Radius", float) = 20

\_Power("Power", float) = 2

\_R("Real part", float) = 0

\_I("Imaginary part", float) = .8

\_ShapeSpeed("Shape speed", float) = .5

\_CircleRadius("Radius variable of circle", float) = .5

}

SubShader

{

// No culling or depth

Cull Off ZWrite Off ZTest Always

Pass

{

CGPROGRAM

#pragma vertex vert

#pragma fragment frag

#include "UnityCG.cginc"

struct appdata

{

float4 vertex : POSITION;

float2 uv : TEXCOORD0;

};

struct v2f

{

float2 uv : TEXCOORD0;

float4 vertex : SV\_POSITION;

};

v2f vert(appdata v)

{

v2f o;

o.vertex = UnityObjectToClipPos(v.vertex);

o.uv = v.uv;

return o;

}

sampler2D \_MainTex;

float4 \_Area;

float \_MaxIter;

float \_Color;

float \_Repeat;

float \_Speed;

float \_Radius = 20.;

float \_Power;

float \_R;

float \_I;

float \_ShapeSpeed;

float \_CircleRadius;

float AngleOf(float2 num)

{

float R = num.x;

float I = num.y;

if ((R > 0) && (I >= 0))

{

return (atan(I / R));

}

else if ((R < 0) && (I >= 0))

{

return (3.14159 - atan(abs(I / R)));

}

else if ((R < 0) && (I < 0))

{

return (3.14159 + atan(abs(I / R)));

}

else if ((R > 0) && (I < 0))

{

return (2 \* 3.14159 - atan(abs(I / R)));

}

else if ((R == 0) && (I > 0))

{

return (3.14159 / 2);

}

else// if ((R == 0) && (I < 0))

{

return ((3 / 2) \* 3.14159);

}

}

float2 powi(float2 z, float power)

{

float angle = AngleOf(z);

return (pow(length(z), power) \* float2(cos(power \* angle), sin(power \* angle))); //z in power \_Power

}

float2 f(float2 z, float power)

{

return powi(z, power) - 1;

}

float2 df(float2 z, float power)

{

return power \* f(z, power - 1);

}

float2 cmult(float2 a, float2 b)

{

return float2(a.x \* b.x - a.y \* b.y, a.x \* b.y + a.y \* b.x);

}

float2 cdiv(float2 a, float2 b)

{

return float2((a.x \* b.x + a.y \* b.y) / (b.x \* b.x + b.y \* b.y), (-a.x \* b.y + a.y \* b.x ) / (b.x \* b.x + b.y \* b.y));

}

fixed4 frag(v2f i) : SV\_Target

{

float2 c = \_Area.xy + (i.uv - .5f) \* \_Area.zw;

bool end = false;

float iter2;

float \_R2, \_I2;

if (\_ShapeSpeed != 0)

{

\_R2 = \_CircleRadius \* sin(\_Time.y \* \_ShapeSpeed);

\_I2 = \_CircleRadius \* cos(\_Time.y \* \_ShapeSpeed);

}

float2 roots[3];

roots[0] = float2(1, 0);

roots[1] = float2(-.5, sqrt(3) / 2);

roots[2] = float2(-.5, -sqrt(3) / 2);

float2 z = c;

for (int iter = 0; iter < \_MaxIter; iter++)

{

if ((\_R != 0) || (\_I != 0))

{

z -= cmult(cdiv(f(z, \_Power), df(z, \_Power)), float2(\_R, \_I));

}

else

{

z -= cmult(cdiv(f(z, \_Power), df(z, \_Power)), float2(\_R2, \_I2));

\_R = \_R2;

\_I = \_I2;

}

for (int i = 0; i < 3; i++)

{

if (length(z - roots[i]) < \_Radius)

{

iter2 = iter;

iter = \_MaxIter;

end = true;

break;

}

}

}

if (end)

{

iter2 -= log2(log(length(z)) / log(\_Radius));

return tex2D(\_MainTex, float2(sqrt(iter2 / \_MaxIter) \* \_Repeat + (\_Time.y - 2) \* \_Speed, \_Color));

}

return 0;

}

ENDCG

}

}

}

Шейдер треугольника Серпинского.

Shader "Fractals/Sierpinski triangle"

{

Properties

{

\_MainTex ("Texture", 2D) = "white" {}

\_Area("Area", vector) = (0, 0, 4, 4)

\_MaxIterSerp("Max iterations", float) = 7

\_V1("Vertex 1", vector) = (10, 0, 0, 0)

\_V2("Vertex 2", vector) = (0, 5.77350269189625, 0, 0)

\_V3("Vertex 3", vector) = (0, -5.77350269189625, 0, 0)

}

SubShader

{

// No culling or depth

Cull Off ZWrite Off ZTest Always

Pass

{

CGPROGRAM

#pragma vertex vert

#pragma fragment frag

#include "UnityCG.cginc"

struct appdata

{

float4 vertex : POSITION;

float2 uv : TEXCOORD0;

};

struct v2f

{

float2 uv : TEXCOORD0;

float4 vertex : SV\_POSITION;

};

v2f vert (appdata v)

{

v2f o;

o.vertex = UnityObjectToClipPos(v.vertex);

o.uv = v.uv;

return o;

}

sampler2D \_MainTex;

float4 \_Area;

float \_MaxIterSerp;

float4 \_V1;

float4 \_V2;

float4 \_V3;

float g(float2 a, float2 b, float2 d)

{

return ((d.x - a.x) \* (b.y - a.y) - (d.y - a.y) \* (b.x - a.x));

}

bool f(float2 a, float2 b, float2 c, float2 d)

{

return (g(a, b, c) \* g(a, b, d) >= 0);

}

float2 triagCenter(float2 a, float2 b, float2 c)

{

return float2((a.x + b.x + c.x) / 3, (a.y + b.y + c.y) / 3);

}

bool inTriag(float2 a, float2 b, float2 c, float2 d)

{

return (f(a, b, c, d) && f(b, c, a, d) && f(c, a, b, d));

}

float2 mPoint(float2 a, float2 b)

{

return float2((a.x + b.x) / 2, (a.y + b.y) / 2);

}

fixed4 frag (v2f i) : SV\_Target

{

float2 c = \_Area.xy + (i.uv - .5f) \* \_Area.zw;

\_MaxIterSerp += 1;

float2 v1 = \_V1.xy;

float2 v2 = \_V2.xy;

float2 v3 = \_V3.xy;

//temp vertexes

float2 tv1 = v1;

float2 tv2 = v2;

float2 tv3 = v3;

if (!inTriag(v1, v2, v3, c))

{

return 1;

}

float iter;

for (iter = 0; (int)iter < (int)\_MaxIterSerp; iter++)

{

if (inTriag(v1, v2, v3, c))

{

tv1 = mPoint(v2, v3);

tv2 = mPoint(v1, v2);

tv3 = mPoint(v1, v3);

if (inTriag(tv1, tv2, tv3, c)) //middle triag

{

return 1;

}

if (inTriag(v1, tv2, tv3, c)) //upper triag

{

v2 = tv2;

v3 = tv3;

}

else if (inTriag(tv1, tv2, v2, c)) //right triag

{

v1 = tv2;

v3 = tv1;

}

else if (inTriag(tv1, tv3, v3, c)) //left triag

{

v1 = tv3;

v2 = tv1;

}

else

{

return 0;

}

}

}

return 0;

}

ENDCG

}

}

}

Шейдер квадрата Серпинского.

Shader "Fractals/Sierpinski square"

{

Properties

{

\_MainTex ("Texture", 2D) = "white" {}

\_Area("Area", vector) = (0, 0, 4, 4)

\_MaxIterSerp("Max iterations", float) = 10

\_V1("Vertex 1", vector) = (-10, 10, 0, 0)

\_V2("Vertex 2", vector) = (10, 10, 0, 0)

\_V3("Vertex 3", vector) = (10, -10, 0, 0)

\_V4("Vertex 4", vector) = (-10, -10, 0, 0)

}

SubShader

{

// No culling or depth

Cull Off ZWrite Off ZTest Always

Pass

{

CGPROGRAM

#pragma vertex vert

#pragma fragment frag

#include "UnityCG.cginc"

struct appdata

{

float4 vertex : POSITION;

float2 uv : TEXCOORD0;

};

struct v2f

{

float2 uv : TEXCOORD0;

float4 vertex : SV\_POSITION;

};

v2f vert(appdata v)

{

v2f o;

o.vertex = UnityObjectToClipPos(v.vertex);

o.uv = v.uv;

return o;

}

sampler2D \_MainTex;

float4 \_Area;

float \_MaxIterSerp;

float4 \_V1;

float4 \_V2;

float4 \_V3;

float4 \_V4;

bool inTriag2(float2 a, float2 b, float2 c, float2 d)

{

return (sign((a.x - d.x) \* (b.y - a.y) - (b.x - a.x) \* (a.y - d.y)) ==

sign((b.x - d.x) \* (c.y - b.y) - (c.x - b.x) \* (b.y - d.y))) &&

(sign((c.x - d.x) \* (a.y - c.y) - (a.x - c.x) \* (c.y - d.y)) ==

sign((b.x - d.x) \* (c.y - b.y) - (c.x - b.x) \* (b.y - d.y)));

}

bool inSquare(float2 v1, float2 v2, float2 v3, float2 v4, float2 d)

{

return (inTriag2(v1, v2, v4, d) || inTriag2(v2, v3, v4, d) || inTriag2(v1, v2, v3, d) || inTriag2(v1,v3,v4, d));

}

float2 twothird(float2 a, float2 b)

{

return (2 \* a + b) / 3;

}

fixed4 frag(v2f i) : SV\_Target

{

float2 c = \_Area.xy + (i.uv - .5f) \* \_Area.zw;

\_MaxIterSerp += 1;

//vertexes

float2 v1 = \_V1.xy;

float2 v2 = \_V2.xy;

float2 v3 = \_V3.xy;

float2 v4 = \_V4.xy;

//temp vertexes

float2 tv1 = v1;

float2 tv2 = v2;

float2 tv3 = v3;

float2 tv4 = v4;

//more temp vertexes

//vertexes for transformation

float2 ttv1 = v1;

float2 ttv2 = v2;

float2 ttv3 = v3;

float2 ttv4 = v4;

float iter = 0;

if (!inSquare(v1, v2, v3, v4, c))

{

return 1;

}

for (iter = 0; (int)iter < (int)\_MaxIterSerp; iter++)

{

tv1 = twothird(twothird(v1, v2), twothird(v4, v3));

tv2 = twothird(twothird(v2, v1), twothird(v3, v4));

tv3 = twothird(twothird(v3, v4), twothird(v2, v1));

tv4 = twothird(twothird(v4, v3), twothird(v1, v2));

if (inSquare(tv1, tv2, tv3, tv4, c)) //center

{

return 1;

}

else if (inSquare(v1, twothird(v1, v2), tv1, twothird(v1, v4), c)) //north-west

{

ttv2 = twothird(v1, v2);

ttv3 = tv1;

ttv4 = twothird(v1, v4);

}

else if (inSquare(twothird(v1, v2), twothird(v2, v1), tv2, tv1, c)) //north

{

ttv1 = twothird(v1, v2);

ttv2 = twothird(v2, v1);

ttv3 = tv2;

ttv4 = tv1;

}

else if (inSquare(twothird(v2, v1), v2, twothird(v2, v3), tv2, c)) //north-east

{

ttv1 = twothird(v2, v1);

ttv3 = twothird(v2, v3);

ttv4 = tv2;

}

else if (inSquare(tv2, twothird(v2, v3), twothird(v3, v2), tv3, c)) //east

{

ttv1 = tv2;

ttv2 = twothird(v2, v3);

ttv3 = twothird(v3, v2);

ttv4 = tv3;

}

else if (inSquare(tv3, twothird(v3, v2), v3, twothird(v3, v4), c)) //south-east

{

ttv1 = tv3;

ttv2 = twothird(v3, v2);

ttv4 = twothird(v3, v4);

}

else if (inSquare(tv4, tv3, twothird(v3, v4), twothird(v4, v3), c)) //south

{

ttv1 = tv4;

ttv2 = tv3;

ttv3 = twothird(v3, v4);

ttv4 = twothird(v4, v3);

}

else if (inSquare(twothird(v4, v1), tv4, twothird(v4, v3), v4, c)) //south-west

{

ttv1 = twothird(v4, v1);

ttv2 = tv4;

ttv3 = twothird(v4, v3);

}

else if (inSquare(twothird(v1, v4), tv1, tv4, twothird(v4, v1), c)) //west

{

ttv1 = twothird(v1, v4);

ttv2 = tv1;

ttv3 = tv4;

ttv4 = twothird(v4, v1);

}

else

{

return 0;

}

v1 = ttv1;

v2 = ttv2;

v3 = ttv3;

v4 = ttv4;

}

return 0;

}

ENDCG

}

}

}

Шейдер T-квадрата.

Shader "Fractals/T-square"

{

Properties

{

\_MainTex ("Texture", 2D) = "white" {}

\_Area("Area", vector) = (0, 0, 4, 4)

\_MaxIterFBS("Max iterations", float) = 256

\_V1("Vertex 1", vector) = (-10, 10, 0, 0)

\_V2("Vertex 2", vector) = (10, 10, 0, 0)

\_V3("Vertex 3", vector) = (10, -10, 0, 0)

\_V4("Vertex 4", vector) = (-10, -10, 0, 0)

}

SubShader

{

// No culling or depth

Cull Off ZWrite Off ZTest Always

Pass

{

CGPROGRAM

#pragma vertex vert

#pragma fragment frag

#include "UnityCG.cginc"

struct appdata

{

float4 vertex : POSITION;

float2 uv : TEXCOORD0;

};

struct v2f

{

float2 uv : TEXCOORD0;

float4 vertex : SV\_POSITION;

};

v2f vert (appdata v)

{

v2f o;

o.vertex = UnityObjectToClipPos(v.vertex);

o.uv = v.uv;

return o;

}

sampler2D \_MainTex;

float4 \_Area;

float \_MaxIterFBS;

float4 \_V1;

float4 \_V2;

float4 \_V3;

float4 \_V4;

bool inTriag2(float2 a, float2 b, float2 c, float2 d)

{

return (sign((a.x - d.x) \* (b.y - a.y) - (b.x - a.x) \* (a.y - d.y)) ==

sign((b.x - d.x) \* (c.y - b.y) - (c.x - b.x) \* (b.y - d.y))) &&

(sign((c.x - d.x) \* (a.y - c.y) - (a.x - c.x) \* (c.y - d.y)) ==

sign((b.x - d.x) \* (c.y - b.y) - (c.x - b.x) \* (b.y - d.y)));

}

bool inSquare(float2 v1, float2 v2, float2 v3, float2 v4, float2 d)

{

return (inTriag2(v1, v2, v4, d) || inTriag2(v2, v3, v4, d) || inTriag2(v1, v2, v3, d) || inTriag2(v1, v3, v4, d));

}

float2 threefouth(float2 a, float2 b)

{

return (3 \* a + b) / 4;

}

float2 onehalf(float2 a, float2 b)

{

return (a + b) / 2;

}

fixed4 frag(v2f i) : SV\_Target

{

float2 c = \_Area.xy + (i.uv - .5) \* \_Area.zw;

float iter;

float2 v1 = \_V1.xy;

float2 v2 = \_V2.xy;

float2 v3 = \_V3.xy;

float2 v4 = \_V4.xy;

float2 v1v2;

float2 v2v3;

float2 v3v4;

float2 v1v4;

float2 cntr;

if (!inSquare(v1, v2, v3, v4, c))

{

return 1;

}

for (iter = 0; (int)iter < (int)\_MaxIterFBS; iter++)

{

v1v2 = onehalf(v1, v2);

v2v3 = onehalf(v2, v3);

v3v4 = onehalf(v3, v4);

v1v4 = onehalf(v1, v4);

cntr = (v1 + v2 + v3 + v4) / 4;

if (inSquare(

threefouth(threefouth(v1, v2), threefouth(v4, v3)),

threefouth(threefouth(v2, v1), threefouth(v3, v4)),

threefouth(threefouth(v3, v4), threefouth(v2, v1)),

threefouth(threefouth(v4, v3), threefouth(v1, v2)),

c)) //center

{

return 1;

}

else if (inSquare(v1, v1v2, cntr, v1v4, c)) //north-east

{

v2 = v1v2;

v3 = cntr;

v4 = v1v4;

}

else if (inSquare(v1v2, v2, v2v3, cntr, c)) //north-west

{

v1 = v1v2;

v3 = v2v3;

v4 = cntr;

}

else if (inSquare(cntr, v2v3, v3, v3v4, c)) //south-west

{

v1 = cntr;

v2 = v2v3;

v4 = v3v4;

}

else if (inSquare(v1v4, cntr, v3v4, v4, c)) //south-east

{

v1 = v1v4;

v2 = cntr;

v3 = v3v4;

}

else

{

return 0;

}

}

return 0;

}

ENDCG

}

}

}

Шейдер «Елочки».

Shader "Fractals/Christmas tree"

{

Properties

{

\_MainTex ("Texture", 2D) = "white" {}

\_Area("Area", vector) = (0, 0, 4, 4)

\_MaxIterFBT("Max iterations", float) = 256

\_V1("Vertex 1", vector) = (10, 0, 0, 0)

\_V2("Vertex 2", vector) = (0, 5.77350269189625, 0, 0)

\_V3("Vertex 3", vector) = (0, -5.77350269189625, 0, 0)

}

SubShader

{

// No culling or depth

Cull Off ZWrite Off ZTest Always

Pass

{

CGPROGRAM

#pragma vertex vert

#pragma fragment frag

#include "UnityCG.cginc"

struct appdata

{

float4 vertex : POSITION;

float2 uv : TEXCOORD0;

};

struct v2f

{

float2 uv : TEXCOORD0;

float4 vertex : SV\_POSITION;

};

v2f vert (appdata v)

{

v2f o;

o.vertex = UnityObjectToClipPos(v.vertex);

o.uv = v.uv;

return o;

}

sampler2D \_MainTex;

float \_MaxIterFBT;

float4 \_Area;

float4 \_V1;

float4 \_V2;

float4 \_V3;

float2 mPoint(float2 a, float2 b)

{

return float2((a.x + b.x) / 2, (a.y + b.y) / 2);

}

bool inTriag(float2 a, float2 b, float2 c, float2 d)

{

return (sign((a.x - d.x) \* (b.y - a.y) - (b.x - a.x) \* (a.y - d.y)) ==

sign((b.x - d.x) \* (c.y - b.y) - (c.x - b.x) \* (b.y - d.y))) &&

(sign((c.x - d.x) \* (a.y - c.y) - (a.x - c.x) \* (c.y - d.y)) ==

sign((b.x - d.x) \* (c.y - b.y) - (c.x - b.x) \* (b.y - d.y)));

}

float2 threefouth(float2 a, float2 b)

{

return (3 \* a + b) / 4;

}

fixed4 frag(v2f i) : SV\_Target

{

float2 c = \_Area.xy + (i.uv - .5) \* \_Area.zw;

float2 v1 = \_V1.xy;

float2 v2 = \_V2.xy;

float2 v3 = \_V3.xy;

//temp vertexes

float2 tv1 = v1;

float2 tv2 = v2;

float2 tv3 = v3;

if (!inTriag(v1, v2, v3, c))

{

return 1;

}

float iter;

for (iter = 0; (int)iter < (int)\_MaxIterFBT; iter++)

{

tv1 = mPoint(v2, v3);

tv2 = mPoint(v1, v2);

tv3 = mPoint(v1, v3);

if (inTriag(tv1, tv2, tv3, c) || inTriag(threefouth(v1, tv1), threefouth(v2, tv3), threefouth(v3, tv2),c)) //center triag or up-down center

{

return 0;

}

else if (inTriag(v1, tv2, tv3, c)) //upper triag

{

v2 = tv2;

v3 = tv3;

}

else if (inTriag(tv1, tv2, v2, c)) //right triag

{

v1 = tv2;

v3 = tv1;

}

else if (inTriag(tv1, tv3, v3, c)) //left triag

{

v1 = tv3;

v2 = tv1;

}

}

return 1;

}

ENDCG

}

}

}

Скрипт выхода из программы.

using UnityEngine;

public class Exit : MonoBehaviour

{

// Update is called once per frame

void Update()

{

if (Input.GetKey("escape"))

{

Application.Quit();

}

}

}

Скрипт движения по фракталу.

using System;

using System.Collections;

using System.Collections.Generic;

using System.Collections.Specialized;

using System.IO;

using System.Security.Cryptography.X509Certificates;

using System.Text.RegularExpressions;

using UnityEngine;

public class MovingInFract : MonoBehaviour

{

public Material mat;

public Vector2 pos;

public float scale = 3f;

public float smooth = 0.3f;

private Vector2 smoothPos;

private float smoothScale;

private float LogRegressSerp(float x) //Logarithmic regression for Sierpinski fractals

{

return 9.4514624177f - 1.4328380519f \* (float)Math.Log(x);

}

private float LogRegressFBS(float x) //Logarithmic regression for fractal by Squares

{

return 11.6665200498f - 1.4695802036f \* (float)Math.Log(x);

}

private float LogRegressFBT(float x)//Logarithmic regression for fractal by Triangles

{

return 8.648433152526f - 1.435240568838f \* (float)Math.Log(x);

}

private void UpdateShader()

{

smoothPos = Vector2.Lerp(smoothPos, pos, smooth);

smoothScale = Mathf.Lerp(smoothScale, scale, smooth);

float aspect = (float)Screen.width / (float)Screen.height;

float scaleX = scale;

float scaleY = scale;

if (aspect > 1f)

{

scaleY /= aspect;

}

else

{

scaleX \*= aspect;

}

mat.SetVector("\_Area", new Vector4(smoothPos.x, smoothPos.y, scaleX, scaleY));

try

{

mat.SetFloat("\_MaxIterSerp", LogRegressSerp((float)scale));

}

catch

{

//yep, void

}

try

{

mat.SetFloat("\_MaxIterFBS", LogRegressFBS((float)scale) + .9f);

}

catch

{

//yep, void

}

try

{

mat.SetFloat("\_MaxIterFBT", LogRegressFBT((float)scale) + .9f);

}

catch

{

//yep, void

}

}

private void HandleInputs()

{

if (Input.GetKey(KeyCode.KeypadPlus))

{

scale /= 1.01f;

}

if (Input.GetKey(KeyCode.KeypadMinus))

{

scale /= .99f;

}

if (Input.GetKey(KeyCode.A))

{

pos.x -= .01f \* scale;

}

if (Input.GetKey(KeyCode.D))

{

pos.x += .01f \* scale;

}

if (Input.GetKey(KeyCode.W))

{

pos.y += .01f \* scale;

}

if (Input.GetKey(KeyCode.S))

{

pos.y -= .01f \* scale;

}

}

// Update is called once per frame

void FixedUpdate()

{

HandleInputs();

UpdateShader();

}

}

Скрипт вызова меню параметров.

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class ChangeMenu : MonoBehaviour

{

public GameObject Menu;

// Start is called before the first frame update

void Start()

{

Menu.SetActive(false);

Cursor.visible = false;

}

// Update is called once per frame

void Update()

{

if (Input.GetKeyDown(KeyCode.Space))

{

if (Menu.activeInHierarchy) //activeSelf or activeInHierarchy

{

Menu.SetActive(false);

Cursor.visible = false;

}

else

{

Menu.SetActive(true);

Cursor.visible = true;

}

}

}

}

Скрипт считывания параметров с полей ввода.

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.UI;

using System.Linq;

public class Params : MonoBehaviour

{

public Material mat;

public InputField InputFieldIter;

public InputField InputFieldColor;

public InputField InputFieldColorRepeat;

public InputField InputFieldColorSpeed;

public InputField InputFieldShapeSpeed;

public InputField InputFieldRadius;

public InputField InputFieldVariableRadius;

public InputField InputFieldVariableRealPart;

public InputField InputFieldVariableImaginaryPart;

public InputField InputFieldPower;

public InputField InputFieldVertex1x;

public InputField InputFieldVertex1y;

public InputField InputFieldVertex2x;

public InputField InputFieldVertex2y;

public InputField InputFieldVertex3x;

public InputField InputFieldVertex3y;

public InputField InputFieldVertex4x;

public InputField InputFieldVertex4y;

// Update is called once per frame

void Update()

{

ReadIter();

ReadRadius();

ReadColor();

ReadColorSpeed();

ReadColorRepeat();

ReadShapeSpeed();

ReadVariableRadius();

ReadVariableRealPart();

ReadVariableImaginaryPart();

ReadPower();

ReadVertex1();

ReadVertex2();

ReadVertex3();

ReadVertex4();

}

private void ReadIter()

{

try

{

float iter;

string input = InputFieldIter.text;

int i = input.IndexOf('.');

if (i != -1)

{

char[] buf = input.ToCharArray();

buf[i] = ',';

input = new string(buf);

}

if (float.TryParse(input, out iter))

{

mat.SetFloat("\_MaxIter", iter);

}

}

catch

{

Debug.Log("Failing read \_MaxIter");

}

}

private void ReadColor()

{

try

{

float color;

string input = InputFieldColor.text;

int i = input.IndexOf('.');

if (i != -1)

{

char[] buf = input.ToCharArray();

buf[i] = ',';

input = new string(buf);

}

if (float.TryParse(input, out color))

{

mat.SetFloat("\_Color", color);

}

}

catch

{

Debug.Log("Failing read \_Color");

}

}

private void ReadRadius()

{

try

{

float Radius;

string input = InputFieldRadius.text;

int i = input.IndexOf('.');

if (i != -1)

{

char[] buf = input.ToCharArray();

buf[i] = ',';

input = new string(buf);

}

if (float.TryParse(input, out Radius))

{

mat.SetFloat("\_Radius", Radius);

}

}

catch

{

Debug.Log("Failing read radius");

}

}

private void ReadColorSpeed()

{

try

{

float ColorSpeed;

string input = InputFieldColorSpeed.text;

int i = input.IndexOf('.');

if (i != -1)

{

char[] buf = input.ToCharArray();

buf[i] = ',';

input = new string(buf);

}

if (float.TryParse(input, out ColorSpeed))

{

mat.SetFloat("\_Speed", ColorSpeed);

}

}

catch

{

Debug.Log("Failing read color speed");

}

}

private void ReadColorRepeat()

{

try

{

float ColorRepeat;

string input = InputFieldColorRepeat.text;

int i = input.IndexOf('.');

if (i != -1)

{

char[] buf = input.ToCharArray();

buf[i] = ',';

input = new string(buf);

}

if (float.TryParse(input, out ColorRepeat))

{

mat.SetFloat("\_Repeat", ColorRepeat);

}

}

catch

{

Debug.Log("Failing read color repeat");

}

}

private void ReadShapeSpeed()

{

try

{

float ShapeSpeed;

string input = InputFieldShapeSpeed.text;

int i = input.IndexOf('.');

if (i != -1)

{

char[] buf = input.ToCharArray();

buf[i] = ',';

input = new string(buf);

}

if (float.TryParse(input, out ShapeSpeed))

{

mat.SetFloat("\_ShapeSpeed", ShapeSpeed);

}

}

catch

{

Debug.Log("Failing read shape speed");

}

}

private void ReadVariableRadius()

{

try

{

float VariableRadius;

string input = InputFieldVariableRadius.text;

int i = input.IndexOf('.');

if (i != -1)

{

char[] buf = input.ToCharArray();

buf[i] = ',';

input = new string(buf);

}

if (float.TryParse(input, out VariableRadius))

{

mat.SetFloat("\_CircleRadius", VariableRadius);

}

}

catch

{

Debug.Log("Failing read variable radius");

}

}

private void ReadVariableRealPart()

{

try

{

float VariableRealPart;

string input = InputFieldVariableRealPart.text;

int i = input.IndexOf('.');

if (i != -1)

{

char[] buf = input.ToCharArray();

buf[i] = ',';

input = new string(buf);

}

if (float.TryParse(input, out VariableRealPart))

{

mat.SetFloat("\_R", VariableRealPart);

}

}

catch

{

Debug.Log("Failing read variable real part");

}

}

private void ReadVariableImaginaryPart()

{

try

{

float VariableImaginaryPart;

string input = InputFieldVariableImaginaryPart.text;

int i = input.IndexOf('.');

if (i != -1)

{

char[] buf = input.ToCharArray();

buf[i] = ',';

input = new string(buf);

}

if (float.TryParse(input, out VariableImaginaryPart))

{

mat.SetFloat("\_I", VariableImaginaryPart);

}

}

catch

{

Debug.Log("Failing read variable imaginary part");

}

}

private void ReadPower()

{

try

{

float Power;

string input = InputFieldPower.text;

int i = input.IndexOf('.');

if (i != -1)

{

char[] buf = input.ToCharArray();

buf[i] = ',';

input = new string(buf);

}

if (float.TryParse(input, out Power))

{

mat.SetFloat("\_Power", Power);

}

}

catch

{

Debug.Log("Failing read \_Power");

}

}

private void ReadVertex1()

{

try

{

float v1x;

float v1y;

string input1 = InputFieldVertex1x.text;

int i = input1.IndexOf('.');

if (i != -1)

{

char[] buf = input1.ToCharArray();

buf[i] = ',';

input1 = new string(buf);

}

string input2 = InputFieldVertex1y.text;

i = input2.IndexOf('.');

if (i != -1)

{

char[] buf = input2.ToCharArray();

buf[i] = ',';

input2 = new string(buf);

}

if (float.TryParse(input1, out v1x) && float.TryParse(input2, out v1y))

{

mat.SetVector("\_V1",new Vector4(v1x, v1y, 0f, 0f));

}

}

catch

{

Debug.Log("Failing read \_V1");

}

}

private void ReadVertex2()

{

try

{

float v2x;

float v2y;

string input1 = InputFieldVertex2x.text;

int i = input1.IndexOf('.');

if (i != -1)

{

char[] buf = input1.ToCharArray();

buf[i] = ',';

input1 = new string(buf);

}

string input2 = InputFieldVertex2y.text;

i = input2.IndexOf('.');

if (i != -1)

{

char[] buf = input2.ToCharArray();

buf[i] = ',';

input2 = new string(buf);

}

if (float.TryParse(input1, out v2x) && float.TryParse(input2, out v2y))

{

mat.SetVector("\_V2",new Vector4(v2x, v2y, 0f, 0f));

}

}

catch

{

Debug.Log("Failing read \_V2");

}

}

private void ReadVertex3()

{

try

{

float v3x;

float v3y;

string input1 = InputFieldVertex3x.text;

int i = input1.IndexOf('.');

if (i != -1)

{

char[] buf = input1.ToCharArray();

buf[i] = ',';

input1 = new string(buf);

}

string input2 = InputFieldVertex3y.text;

i = input2.IndexOf('.');

if (i != -1)

{

char[] buf = input2.ToCharArray();

buf[i] = ',';

input2 = new string(buf);

}

if (float.TryParse(input1, out v3x) && float.TryParse(input2, out v3y))

{

mat.SetVector("\_V3",new Vector4(v3x, v3y, 0f, 0f));

}

}

catch

{

Debug.Log("Failing read \_V3");

}

}

private void ReadVertex4()

{

try

{

float v4x;

float v4y;

string input1 = InputFieldVertex4x.text;

int i = input1.IndexOf('.');

if (i != -1)

{

char[] buf = input1.ToCharArray();

buf[i] = ',';

input1 = new string(buf);

}

string input2 = InputFieldVertex4y.text;

i = input2.IndexOf('.');

if (i != -1)

{

char[] buf = input2.ToCharArray();

buf[i] = ',';

input2 = new string(buf);

}

if (float.TryParse(input1, out v4x) && float.TryParse(input2, out v4y))

{

mat.SetVector("\_V4",new Vector4(v4x, v4y, 0f, 0f));

}

}

catch

{

Debug.Log("Failing read \_V4");

}

}

}

Скрипт выбора фрактала, его построения и задание начальный параметров.

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.UI;

public class DropdownChange : MonoBehaviour

{

public Dropdown dropdown;

public Material mat;

public MovingInFract script;

public InputField InputFieldIter;

public InputField InputFieldColor;

public InputField InputFieldColorRepeat;

public InputField InputFieldColorSpeed;

public InputField InputFieldShapeSpeed;

public InputField InputFieldRadius;

public InputField InputFieldVariableRadius;

public InputField InputFieldVariableRealPart;

public InputField InputFieldVariableImaginaryPart;

public InputField InputFieldPower;

public InputField InputFieldVertex1x;

public InputField InputFieldVertex1y;

public InputField InputFieldVertex2x;

public InputField InputFieldVertex2y;

public InputField InputFieldVertex3x;

public InputField InputFieldVertex3y;

public InputField InputFieldVertex4x;

public InputField InputFieldVertex4y;

public GameObject AlgNeed;

public GameObject AlgVar;

public GameObject GeometryVar;

public GameObject Vertex4;

public void Dropdown\_IndexChange()

{

string shader = dropdown.options[dropdown.value].text;

mat.shader = Shader.Find("Fractals/" + shader);

switch (shader)

{

case("Mandelbrot"):

{

SetMandelbrot();

SetMandelbrotInput();

break;

}

case("Julia"):

{

SetJulia();

SetJuliaInput();

break;

}

case("Newtone fractal"):

{

SetNewtonefractal();

SetNewtonefractalInput();

break;

}

case("Random fractal"):

{

SetRandomfractal();

SetRandomfractalInput();

break;

}

case("Sierpinski triangle"):

{

SetSierpinskitriangle();

SetSierpinskitriangleInput();

break;

}

case("Sierpinski square"):

{

SetSierpinskisquare();

SetSierpinskisquareInput();

break;

}

case("T-square"):

{

SetTsquare();

SetTsquareInput();

break;

}

case("Christmas tree"):

{

SetChristmastree();

SetChristmastreeInput();

break;

}

}

}

// Start is called before the first frame update

void Start()

{

SetMandelbrot();

SetMandelbrotInput();

}

private void SetMandelbrot()

{

mat.shader = Shader.Find("Fractals/Mandelbrot");

mat.SetVector("\_Area", new Vector4(0f, 0f, 4f, 4f));

mat.SetFloat("\_MaxIter", 256f);

mat.SetFloat("\_Radius", 20f);

mat.SetFloat("\_Color", .5f);

mat.SetFloat("\_Power", 2f);

mat.SetFloat("\_Speed", .01f);

mat.SetFloat("\_Repeat", 1f);

script.scale = 5f;

script.pos = new Vector2(0f, 0f);

}

private void SetMandelbrotInput()

{

AlgNeed.SetActive(true);

AlgVar.SetActive(false);

GeometryVar.SetActive(false);

InputFieldIter.text = "256";

InputFieldColor.text = "0,5";

InputFieldColorRepeat.text = "1";

InputFieldColorSpeed.text = "0,01";

InputFieldRadius.text = "20";

InputFieldPower.text = "2";

}

private void SetJulia()

{

mat.SetVector("\_Area", new Vector4(0f, 0f, 4f, 4f));

mat.SetFloat("\_MaxIter", 256f);

mat.SetFloat("\_Radius", 20f);

mat.SetFloat("\_Color", .5f);

mat.SetFloat("\_Power", 2f);

mat.SetFloat("\_Speed", .01f);

mat.SetFloat("\_Repeat", 1f);

mat.SetFloat("\_R", 0f);

mat.SetFloat("\_I", .8f);

mat.SetFloat("\_ShapeSpeed", .5f);

mat.SetFloat("\_CircleRadius", .5f);

script.scale = 5f;

script.pos = new Vector2(0f, 0f);

}

private void SetJuliaInput()

{

AlgNeed.SetActive(true);

AlgVar.SetActive(true);

GeometryVar.SetActive(false);

InputFieldIter.text = "256";

InputFieldColor.text = "0,5";

InputFieldColorRepeat.text = "1";

InputFieldColorSpeed.text = "0,01";

InputFieldRadius.text = "20";

InputFieldPower.text = "2";

InputFieldVariableRadius.text = "0,5";

InputFieldVariableImaginaryPart.text = "0,8";

InputFieldVariableRealPart.text = "0";

InputFieldShapeSpeed.text = "0,5";

}

private void SetNewtonefractal()

{

mat.SetVector("\_Area", new Vector4(0f, 0f, 4f, 4f));

mat.SetFloat("\_MaxIter", 256f);

mat.SetFloat("\_Color", .7f);

mat.SetFloat("\_Speed", .01f);

mat.SetFloat("\_Repeat", 1f);

mat.SetFloat("\_Radius", .0001f);

mat.SetFloat("\_Power", 3f);

mat.SetFloat("\_R", 1f);

mat.SetFloat("\_I", 0f);

mat.SetFloat("\_ShapeSpeed", .5f);

mat.SetFloat("\_CircleRadius", .5f);

script.scale = 5f;

script.pos = new Vector2(0f, 0f);

}

private void SetNewtonefractalInput()

{

AlgNeed.SetActive(true);

AlgVar.SetActive(true);

GeometryVar.SetActive(false);

InputFieldIter.text = "256";

InputFieldColor.text = "0,7";

InputFieldColorRepeat.text = "1";

InputFieldColorSpeed.text = "0,01";

InputFieldRadius.text = "0,0001";

InputFieldPower.text = "3";

InputFieldVariableRadius.text = "0,5";

InputFieldVariableImaginaryPart.text = "0";

InputFieldVariableRealPart.text = "1";

InputFieldShapeSpeed.text = "0,5";

}

private void SetRandomfractal()

{

mat.SetVector("\_Area", new Vector4(0f, 0f, 4f, 4f));

mat.SetFloat("\_MaxIter", 256f);

mat.SetFloat("\_Color",.6f);

mat.SetFloat("\_Repeat", 1f);

mat.SetFloat("\_Speed", .0f);

mat.SetFloat("\_Radius", .2f);

mat.SetFloat("\_Power", 2f);

mat.SetFloat("\_R", 3.79f);

mat.SetFloat("\_I", 0.87f);

mat.SetFloat("\_ShapeSpeed", .5f);

mat.SetFloat("\_CircleRadius", .5f);

script.pos = new Vector2(-0.9287205f, -1.076767f);

script.scale = 21f;

}

private void SetRandomfractalInput()

{

AlgNeed.SetActive(true);

AlgVar.SetActive(true);

GeometryVar.SetActive(false);

InputFieldIter.text = "256";

InputFieldColor.text = "0,6";

InputFieldColorRepeat.text = "1";

InputFieldColorSpeed.text = "0";

InputFieldRadius.text = "0,2";

InputFieldPower.text = "2";

InputFieldVariableRadius.text = "0,5";

InputFieldVariableImaginaryPart.text = "0,87";

InputFieldVariableRealPart.text = "3.79";

InputFieldShapeSpeed.text = "0,5";

}

private void SetSierpinskitriangle()

{

mat.SetVector("\_Area", new Vector4(0f, 0f, 4f, 4f));

script.pos = new Vector2(0f, 4.960936f);

script.scale = 19f;

mat.SetVector("\_V1", new Vector4(0f, 10f, 0f, 0f));

mat.SetVector("\_V2", new Vector4(5.77350269189625f, 0f, 0f, 0f));

mat.SetVector("\_V3", new Vector4(-5.77350269189625f, 0f, 0f, 0f));

}

private void SetSierpinskitriangleInput()

{

AlgNeed.SetActive(false);

AlgVar.SetActive(false);

GeometryVar.SetActive(true);

Vertex4.SetActive(false);

InputFieldVertex1x.text = "0";

InputFieldVertex1y.text = "10";

InputFieldVertex2x.text = "5.77350269189625";

InputFieldVertex2y.text = "0";

InputFieldVertex3x.text = "-5.77350269189625";

InputFieldVertex3y.text = "0";

}

private void SetSierpinskisquare()

{

mat.SetVector("\_Area", new Vector4(0f, 0f, 4f, 4f));

script.pos = new Vector2(0f, 0f);

script.scale = 37f;

mat.SetVector("\_V1", new Vector4(-10f, 10f, 0f, 0f));

mat.SetVector("\_V2", new Vector4(10f, 10f, 0f, 0f));

mat.SetVector("\_V3", new Vector4(10f, -10f, 0f, 0f));

mat.SetVector("\_V4", new Vector4(-10f, -10f, 0f, 0f));

}

private void SetSierpinskisquareInput()

{

AlgNeed.SetActive(false);

AlgVar.SetActive(false);

GeometryVar.SetActive(true);

Vertex4.SetActive(true);

InputFieldVertex1x.text = "-10";

InputFieldVertex1y.text = "10";

InputFieldVertex2x.text = "10";

InputFieldVertex2y.text = "10";

InputFieldVertex3x.text = "10";

InputFieldVertex3y.text = "-10";

InputFieldVertex4x.text = "-10";

InputFieldVertex4y.text = "-10";

}

private void SetTsquare()

{

mat.SetVector("\_Area", new Vector4(0f, 0f, 4f, 4f));

script.pos = new Vector2(0f, 0f);

script.scale = 38f;

mat.SetVector("\_V1", new Vector4(-10f, 10f, 0f, 0f));

mat.SetVector("\_V2", new Vector4(10f, 10f, 0f, 0f));

mat.SetVector("\_V3", new Vector4(10f, -10f, 0f, 0f));

mat.SetVector("\_V4", new Vector4(-10f, -10f, 0f, 0f));

}

private void SetTsquareInput()

{

AlgNeed.SetActive(false);

AlgVar.SetActive(false);

GeometryVar.SetActive(true);

Vertex4.SetActive(true);

InputFieldVertex1x.text = "-10";

InputFieldVertex1y.text = "10";

InputFieldVertex2x.text = "10";

InputFieldVertex2y.text = "10";

InputFieldVertex3x.text = "10";

InputFieldVertex3y.text = "-10";

InputFieldVertex4x.text = "-10";

InputFieldVertex4y.text = "-10";

}

private void SetChristmastree()

{

mat.SetVector("\_Area", new Vector4(0f, 0f, 4f, 4f));

script.pos = new Vector2(0f, 4.960936f);

script.scale = 18f;

mat.SetVector("\_V1", new Vector4(0f, 10f, 0f, 0f));

mat.SetVector("\_V2", new Vector4(5.77350269189625f, 0f, 0f, 0f));

mat.SetVector("\_V3", new Vector4(-5.77350269189625f, 0f, 0f, 0f));

}

private void SetChristmastreeInput()

{

AlgNeed.SetActive(false);

AlgVar.SetActive(false);

GeometryVar.SetActive(true);

Vertex4.SetActive(false);

InputFieldVertex1x.text = "0";

InputFieldVertex1y.text = "10";

InputFieldVertex2x.text = "5.77350269189625";

InputFieldVertex2y.text = "0";

InputFieldVertex3x.text = "-5.77350269189625";

InputFieldVertex3y.text = "0";

}

}