GIMME Fractals

Manual

Contents

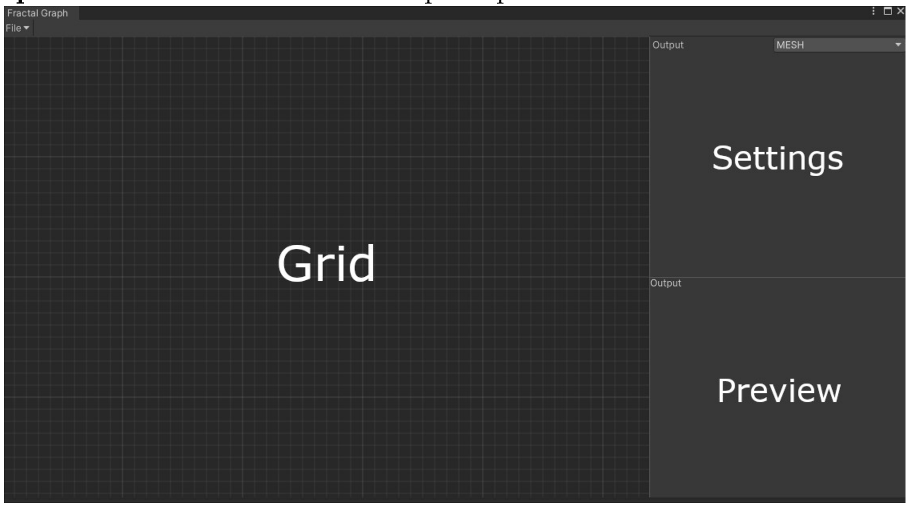
1 Dependencies and Setup

In order for the package to work, the following dependencies need to be installed either manually or via the package manager:

- **Burst 1.6.6** or above
- **Collections 1.2.4** or above
- **FBX Exporter 4.1.3** or above
- **Mathematics 1.2.6** or above (might work with lower versions; untested)
- **Unity UI 1.0.0** or above

2 Fractal Graph

To first open up the tool, go to **Window→Gimme Fractals→Fractal Graph**. A new editor window will open up:



The nodes for combining and creating fractals are going to be placed on the **Grid**, the output settings are on the right top panel (**Settings Panel**, and a preview is going to be shown at the right bottom panel (**Preview Panel**).

But before you can start placing nodes, you first have to create a new graph under **File→New**. A mesh output node is automatically created for you. However, you can change it to a different output type in the **Settings Panel**. At the moment two types of output are supported: **Mesh** and **Texture**.

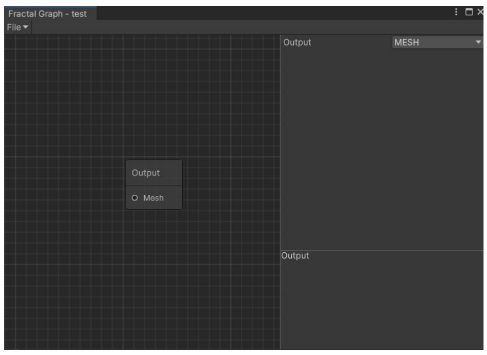


Figure 1: Mesh Output

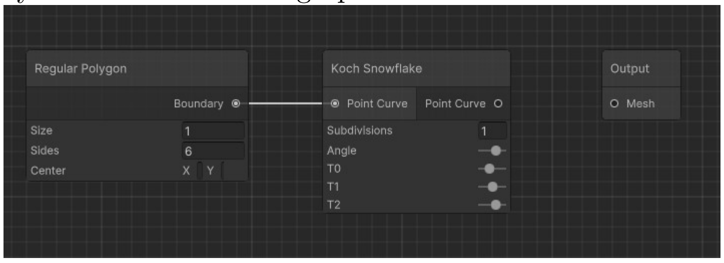
Leave the output to mesh for now. To create nodes, open up the context menu by right-clicking the **Grid**. For an explanation for the different types of nodes, head over to the next section (**Nodes**).

Create a regular polygon under **Create→Regular Polygon**. Set the sides to 6 and leave the rest of the values as they are. You'll see that the node has a **Boundary** as an Output. A boundary is a **Point Curve** that can be triangulated (not yet integrated).

Here an overview of the different I/O Types:

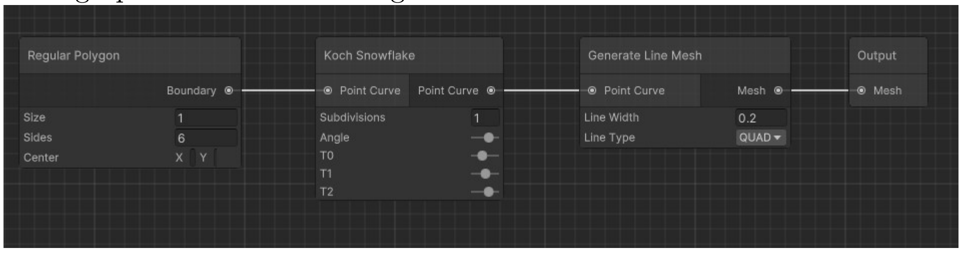
- **Point Curve**: A collection of points and edges
- **Boundary**: A point curve, that bounds a region (there are no open ends, i.e. a polygon)
- **Mesh**: The same as a Unity Mesh (also internally)
- **Texture**: A render texture (also internally)

Now create a Koch Snowflake Modifier under **Fractal→Boundary→Koch Snowflake**. Then connect the output of the polygon with the input of the newly created node. Your graph should look like this:

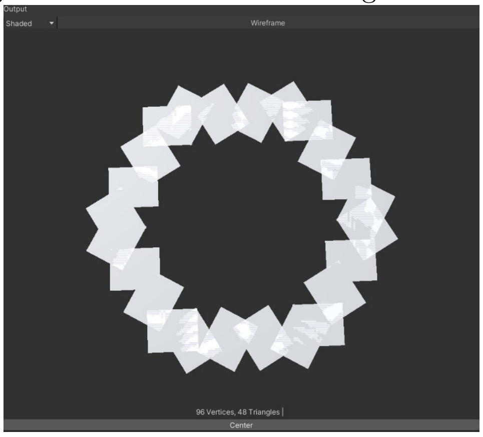


Now we need to create a mesh from the **Point Curve** (output of the Koch Snowflake). To do that, we create a new node under **Generate→Generate Line**. It will take a point curve and places geometry for every edge of the curve into a mesh. Connect the output of the **Koch Snowflake** to the **Generate Line Mesh** input. Then connect the output of the **Generate Line Mesh** node to the **Mesh Output**.

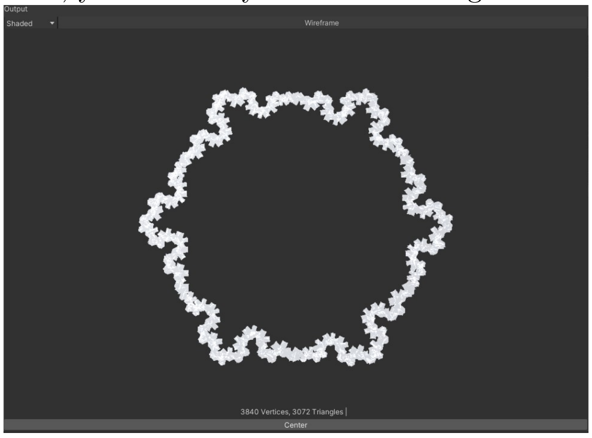
Your graph will look something like this now:



You should also see the generated mesh now in the **Preview Panel**. Right now it will look something like this:



Not very good yet, but after fiddling around with the values of the nodes however, you can easily create something resembling a snowflake:

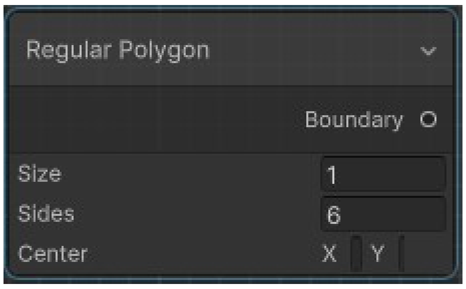


Multiple example fractal graphs are included in the folder **Sample-Graphs**, so you can see what is possible already.

3 Nodes

3.1 Create

Regular Polygon



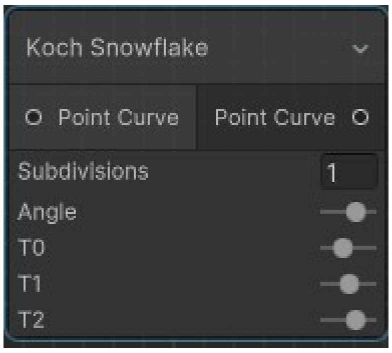
Creates a regular polygon (all sides have equal length) as a **Boundary**.

- **Size**: Size of the polygon (in m) [Obsolete in the future]
- **Sides**: How many edges the boundary has
- **Center**: A position offset to the **Boundary** [Obsolete in the future]

3.2 Fractal

3.2.1 Boundary

Koch Snowflake



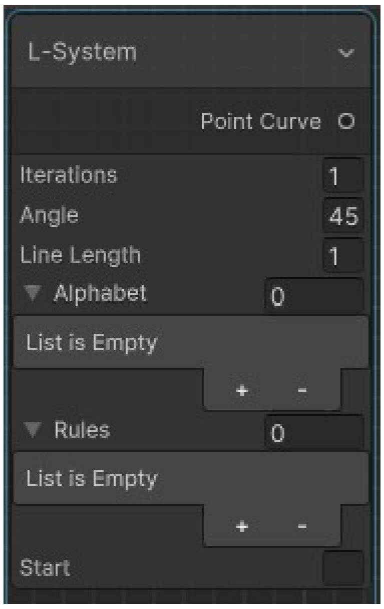
Subdivides every edges of an input **Point Curve** into three edges and moves the center point into the perpendicular direction (magnitude depends on angle). Three new vertices are created for this.

- **Subdivisions**: How often should the procedure be repeated?

- **Angle**: Determines the offset height of the center vertex
- T_0: The first edge cut-point
- T_1: The second edge cut-point
- T_2: The third edge cut-point
  When the angle is 60°, $T_0 = \frac{1}{3}$, $T_1 = \frac{1}{2}$, $T_2 = \frac{2}{3}$, this is the classical Koch Curve.

3.2.2 Line

L-System

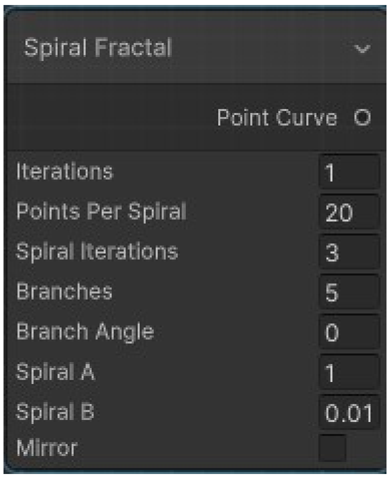

L-System
Point Curve
Iterations 1
Angle 45
Line Length 1
Alphabet 0
List is Empty
Rules 0
List is Empty
Start

This node allows you to create most Lindenmayer Systems. For a reference what cool things you can do with it, I recommend this site here: Paul Borke - L-System Notes

- **Iterations**: How often should **Start** be expanded?
- **Angle**: Determines how far lines are turned when encountering a **"Turn Left"** or **"Turn Right"** symbol.
- **Line Length**: How long is each line?
- **Alphabet**: The symbols and their meaning to be used in the rules.

- **Rules**: Each rule is a string, starting with a symbol from the alphabet followed by ":". Then a list of replacement symbols can be entered, separated by ",". E.g. a rule would be "X:F,+,-,F,X". This is working but suboptimal right now, and will be upgraded by a better system in the future (like tags for example)
- **Start**: The starting symbol sequence. Also separated by ",".

Spiral



Spiral Fractal

Point Curve

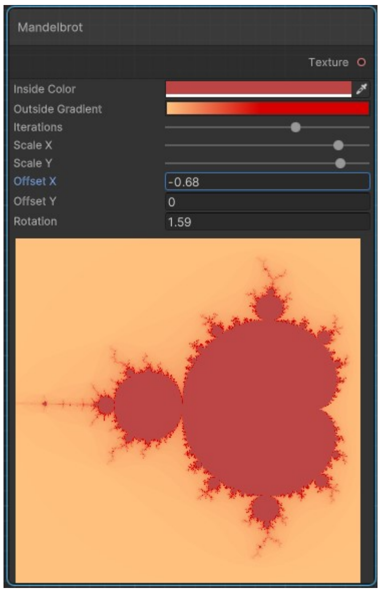
Iterations	1
Points Per Spiral	20
Spiral Iterations	3
Branches	5
Branch Angle	0
Spiral A	1
Spiral B	0.01
Mirror	☐

Creates a spiral fractal (spirals attached to spirals). The spiral type is Archimedean, and more information can be found on Wikipedia: Archimedean Spiral

- **Iterations**: How often should spirals be attached?
- **Points Per Spiral**
- **Spiral Iterations**: How many windings does the spiral have?
- **Branches**: How many spirals are attached to a larger spiral in each iteration?
- **Branch Angle**: Offsets the angle of the branch spirals attachment point
- **Spiral A**: The starting point of the spiral in relation to its center
- **Spiral B**: How quickly the spiral scales (in essence)
- **Mirror**: Alternates the orientation of the branches [Experimental]

3.2.3 Texture

Mandelbrot


Mandelbrot

Texture

Inside Color

Outside Gradient

Iterations

Scale X

Scale Y

Offset X -0.68

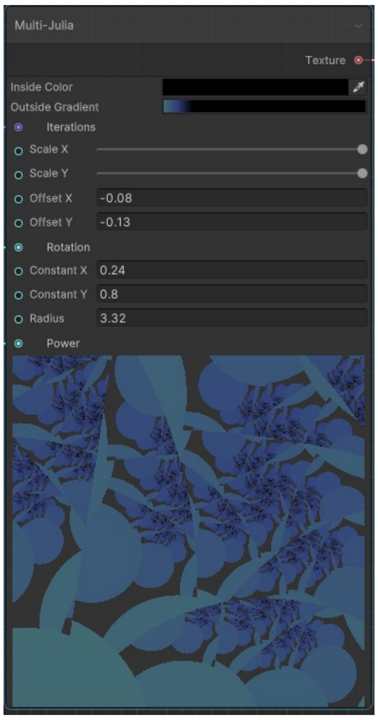
Offset Y 0

Rotation 1.59

Creates a Mandelbrot Texture (a classic).

- **Inside Color**: Color of the region that converges
- **Outside Gradient**: Color gradient of the region that diverges
- **Iterations**: How many times should the shader check if a value converges or diverges.
- **Scale - Offset - Rotation**: Should be self-explanatory

Multi-Julia


Multi-Julia

Texture

Inside Color

Outside Gradient

Iterations

Scale X

Scale Y

Offset X -0.08

Offset Y -0.13

Rotation

Constant X 0.24

Constant Y 0.8

Radius 3.32

Power

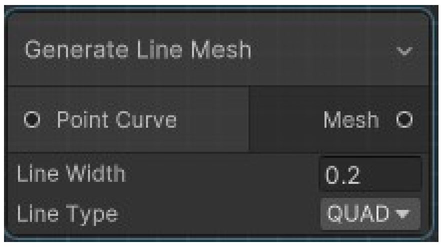
Creates a Texture, representing a Multi-Julia Set

- **Inside Color**: Color of the region that converges
- **Outside Gradient**: Color gradient of the region that diverges
- **Iterations**: How many times should the shader check if a value converges or diverges.
- **Scale - Offset - Rotation**: Should be self-explanatory
- **Constants**: Julia-Constant (in contrast to Mandelbrot where it depends on the pixel position)
- **Radius**: Radius of convergence / divergence
- **Power**: Takes the complex values to this power. Mandelbrot has a power of 2.

3.3 Generate

Generate Line

Creates a line mesh from a given **Point Curve**. Each edge of the curve is transformed into geometry.

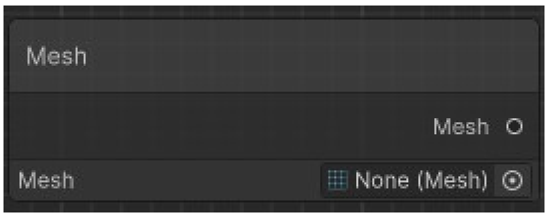


Line Width 0.2
Line Type QUAD

- **Line Width**: Thickness of each edge.
- **Line Type**: There are two modes right now: **QUAD** and **SIMPLE**. When choosing quads, each edge is simply replaced by a quadrilateral. It is simple enough to be put into a compute shader, which is also what happens. Simple mode on the other hand connects the edges together (not so simple actually). It can however not handle vertices that are connected to more than two other vertices (it will work, just not look correct). More sophisticated (read: computationally expensive) modes may be added in the future.

3.4 Input

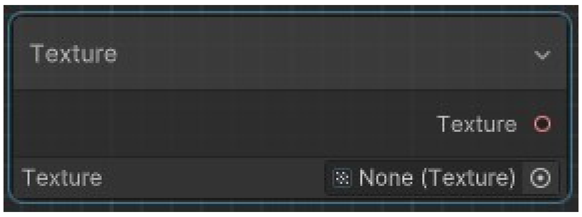
Mesh Input



Mesh None (Mesh)

Import your own mesh to e.g. spawn or combine them into fractals

Texture Input

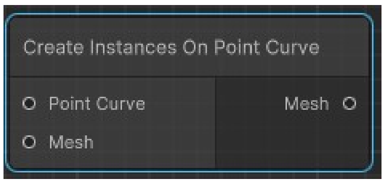


Texture None (Texture)

Import a texture to e.g. combine it with other fractal textures

3.5 Instancing

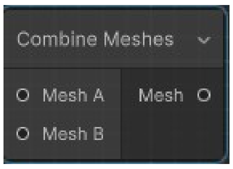
Generate Instances on Point Curve



Spawns an input mesh on each point of a **Point Curve** and combines them into a single mesh (CombineInstance). This allows you to for example to spawn fractals in fractal patterns or any mesh really.

3.6 Mesh

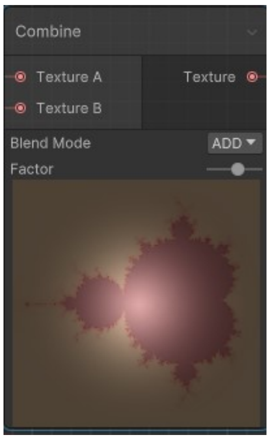
Combine



Simply combines two meshes into one.

3.7 Texture

Combine



Blend Mode ADD
Factor

Combines two textures together.

- **Blend Mode**: How should the textures be blended together. The modes are similar to other tools and programs (however, there are a few missing right now).
- **Factor**: How much should one texture be preferred to the other while blending. Default value is 0.5 (equal treatment)

4 Contact

For any questions or suggestions, you can reach me anytime by the following email-adress:

blenderfan@gmx.at

There is also a discord server, which is usually the fastest way to reach me:

Parable Games - Discord

Alternatively, you can also find some social media links and contact information on my website:

https://parable-games.com

5 Future Plans

Some additional features I plan to implement in the near future:

- 3D L-System
- Some form of UV-Mapping (likely very simple for starters)
- Noise Generator (Compute Shaders already exists in Gimme Cloud Shadows)
- Runtime Generation

Thank You!

Your purchase of **Gimme Fractals** enables me to continue developing code and techniques for game-development in an independent way!