



Fractal-O-Mat Fractal Fun Resources

Fractal 1: Draw

Choose fractal: Sierpinski triangle

Choose iteration: 5

Fractal 1: Music

Choose key note: C

Choose scale: Major

Play Fractal 1:

Fractal 2: Draw

Choose fractal: Dragon curve

Fractal 2: Music

Choose key note: C

Choose scale: Major

Play Fractal 2:

Generate Fractals

Choose order of phrases: 4

Reset

Translation between fractal images and music

Using Grammatical Framework to translate between self-similar fractals, and how the fractals can be interpreted as music

DATX02-19-26, Supervisor: Krasimir Angelov

H. ANDERSSON, A. BERGSTEN, B. BRANDSTRÖM,
G. ENGSMYRE, E. KNOPH, E. MEIJER

Cover: A screen shot of the finished product. On the left (Fractal 1) is the graphical representation of the fifth iteration of the Sierpinski triangle, and on the right (Fractal 2) is the fifth iteration of the Dragon curve. A fourth order phrase has been highlighted in red in the left fractal, and the corresponding phrase in the right fractal is marked in red as well.

Helena Andersson, Alfred Bergsten, Boel Brandström, Gustav Engsmyre, Eli Knoph, Edvin Meijer
Department of Computer Science and Engineering
Chalmers University of Technology

Abstract

This bachelor's thesis concerns the translation between certain types of self-similar fractals, and the interpretation of said fractals as music. The fractals used in this project are the Dragon curve, the Sierpinski triangle, the Hilbert curve, the Gosper curve, and the Koch square snowflake. All these fractals can be described by *Lindenmayer systems*, or *L-systems*, consisting of an alphabet, an axiom and a set of rules. These systems can be viewed as a type of formal grammar, and thus the programming language *Grammatical Framework*, or *GF*, can be used to generate strings that represent the fractal images. GF is based on functional programming and is used for translation between languages – natural as well as formal. With GF it is possible to translate between the fractal images (e.g. from a Koch square snowflake to a Sierpinski triangle). The fractals and the translation between them are visualised in a GUI in the form of a web application. Information about the fractals (such as *phrases*, the coordinates for the different line segments that the fractal images consist of etc.) is temporarily stored in a data structure constructed for this project. The instructions for graphically rendering the fractal images are used to create a musical representation of the fractals. A straight line in the graphical representation means "play note", and turns mean "raise/lower the pitch". The pitch is changed with respect to the angle of the turn, and the duration of the notes is decided stochastically. In order to make the music harmonic, all pieces of music are created from a certain scale, e.g. C-major.

Keywords: fractals, music, Grammatical Framework, L-systems, formal grammar.

Sammandrag

Detta kandidatarbete behandlar översättning mellan vissa typer av självliknande fraktalbilder samt tolkningen av dessa fraktaler som musik. Fraktalerna som använts i projektet är Drakkurvan, Hilbertkurvan, Sierpinskis triangel, Gospers kurva och von Kochs fyrkantiga snöflinga. Alla dessa fraktaler kan beskrivas av *Lindenmayersystem*, eller *L-system*, vilka består av ett alfabet, ett axiom och en uppsättning regler. Dessa system kan ses som en sorts formell grammatik och därmed kan programmeringsspråket *Grammatical Framework*, eller *GF*, användas för att generera strängar som representerar fraktalbilderna. GF är baserat på funktionell programmering och används för översättning mellan språk – naturliga såväl som formella. Med GF är det även möjligt att översätta mellan fraktalbilderna (exempelvis från von Kochs fyrkantiga snöflinga till Sierpinskis triangel). Fraktalerna och översättningen mellan dem visualiseras i ett grafiskt användargränssnitt i form av en webapplikation. Information om fraktalerna (så som *fraser*, koordinaterna för de olika linjerna som bygger upp fraktalbilderna, m.m.) lagras temporärt i en datastruktur som konstruerats för detta projekt. Instruktionerna för det grafiska genererandet av fraktalbilderna används för att skapa en musikalisk representation av fraktalerna. En rät linje i den grafiska representationen betyder ”spela tonen” och svängar betyder ”höj/sänk tonhöjden”. Tonhöjden ändras med avseende på svängningsvinkeln och tonlängden bestäms stokastiskt. För att göra musiken harmonisk skapas alla musikstycken utifrån en viss skala, exempelvis C-dur.

Nyckelord: fraktaler, musik, Grammatical Framework, L-system, formell grammatik.

Contents

1	Background	1
1.1	Fractals	1
1.2	Grammars and Lindenmayer systems	2
1.3	Grammatical Framework	4
1.3.1	Simulated recursion in GF	4
1.3.2	Phrases in GF	4
1.4	MIDI files	6
1.5	Turtle graphics	6
1.6	Web server architecture	6
2	Task	8
3	Scope	9
3.1	Ethical aspects	9
3.2	Fractals used	9
3.3	Musical input	10
3.4	Stochastic rules	11
3.5	Grammar input	11
4	Method	12
4.1	Initial stage	12
4.1.1	Prototype	13
4.1.2	Exploring existing L-systems for music	14
4.2	GF generation	15
4.3	Data structure	16
4.4	Parsing	18
4.4.1	From string to tree	18
4.4.2	Musical interpretation	19
4.4.3	Virtual piano input affecting graphical representation	20
4.5	Web application	21
4.5.1	Client-server architecture	21
4.5.2	Drawing and translating between fractals	22
5	Result	24
6	Discussion	29
6.1	Generating strings in GF	29

6.2	Data structure	29
6.3	Musical observations	31
6.4	Translating fractal fragments	31
6.5	Taking the project further	31
A	Appendix: Musical terms	33
B	Appendix: GF-grammars in the project	34
	References	37

1

Background

Fractals have always been fascinating to mathematicians, programmers and people in general. There are multiple ways of generating fractals, and these ways vary among different fractal types, but self-similar fractals are possible to define using a *grammar* (see Section 1.2), similar to how a grammar can define a natural language. Grammatical Framework (see Section 1.3) is a programming language mainly used for translating between natural languages by defining a thorough grammar for each of the languages one wishes to translate between. Fractals can also be described by a grammar, and if these grammars are defined in Grammatical Framework translation between fractals is possible. Since fractal images and music are both two dimensional, with music having the properties of both pitch ("height") and duration ("length"), the fractal images can be interpreted as music.

To be able to translate between fractal images and/or music is mainly of academic interest. It is a way to possibly extend the use of Grammatical Framework and examine fractals from a different point of view. It can also be used to help people further understand fractals by comparing them to each other, study their grammars and visualise their construction. The musical interpretation can also be used to help composers by letting them be inspired by the fractal nature of music. The product might also be used to create or inspire multimedia art. Therefore, three main target groups were identified for this project: people interested in different applications of Grammatical Framework, students trying to understand fractals and people composing music or creating multimedia art.

The main purpose of this project is to examine how Grammatical Framework can be used as a tool for translating between different fractal images, and how said fractal images can be interpreted as music. Furthermore, a secondary purpose of the project is to examine how to make the final product interactive, accessible, and interesting for the target groups mentioned above.

1.1 Fractals

Benoit B. Mandelbrot [1, p15] gives the following definition of a fractal:

“A *fractal* is by definition a set for which the Hausdorff Besicovitch dimension strictly exceeds the topological dimension.”

However, as Kenneth Falconer [2] states in his book on the topic, this definition and many others prove unsatisfactory as they do not apply for certain sets that should be considered fractals. Instead, Falconer [2, pXXVIII] argues, a set F can be thought of as a fractal if:

- “ F has a fine structure, that is, detail on arbitrarily small scales.”
- “ F is too irregular to be described in traditional geometrical language, both locally and globally.”
- “Often F has some form of self-similarity, perhaps approximate or statistical.”
- “Usually, the ‘fractal dimension’ of F (defined in some way) is greater than its topological dimension.”
- “In most cases of interest, F is defined in a very simple way, perhaps recursively.”

For this text, the exact definition is not very relevant. All examples of fractals considered in the project are self-similar and defined recursively. According to Mandelbrot [1], a set F is called *self-similar* if it is made up of disjoint subsets that are all similar to F .

1.2 Grammars and Lindenmayer systems

A grammar is, according to Aarne Ranta, defined as: “a set of rules for analysing and producing text and speech” [3, p3]. Ranta writes that for natural languages (e.g. Swedish), grammars are a consequence of the language they describe. For formal languages, the language is defined by its grammar and has thus been designed with a specific purpose.

The types of grammar that are used in this project are all formal grammars. A formal grammar consists of three components: an *alphabet*, an *axiom* and a set of *rules* that should be applied to the axiom and the following iterations of it, as stated by Ullman and Hopcroft in [4]. The alphabet is – according to [4] – the set of characters that can be used in the particular grammar. Among many things it can be letters, descriptions of drawing operations or musical notes, depending of what type of grammar it is. Ullman and Hopcroft further mention that the axiom is the initial “sentence” consisting of characters from the alphabet. The characters of the axiom will then be substituted with other characters from the alphabet. The substitution will depend on the grammatical rules, stating which characters should be changed into which characters, see (1.1) for example.

A Lindenmayer system, or *L-system*, is a type of formal grammar that uses *rewriting* for generating fractals – described by James Hanan and Przemysław Prusinkiewicz in [5]. Hanan and Prusinkiewicz state that rewriting is a process where the axiom is changed to a more complex string by iteratively replacing parts of it. In an L-system, all possible rules are applied at every iteration [6]. An example of an L-system is:

1. Background

$$\begin{aligned}\text{Alphabet} &: A, B, r, l \\ \text{Axiom} &: A \\ \text{Rules} &: A \rightarrow BlAlB, B \rightarrow ArBrA\end{aligned}\tag{1.1}$$

The system in (1.1) takes the initial string "A" and replaces the A with $B l A l B$, generating the new string " $B l A l B$ ". In this string, the A is replaced with $B l A l B$ and the B is replaced with $A r B r A$, generating the string " $A r B r A l B l A l B l A r B r A$ ". The iterations continue as follows:

$$\begin{aligned}0 &: A \\ 1 &: BlAlB \\ 2 &: ArBrAlBlAlBlArBrA \\ 3 &: BlAlBrArBrArBlAlBlArBrAlBlAlBlArBrAlBlAlBrArBrArBlAlB \\ &\vdots\end{aligned}$$

The string can then be interpreted differently depending on what operations A , B , r and l represent. To know how to translate a string from an L-system, the following example of interpretations can be used:

$$\begin{aligned}A &: \text{draw a line and move forward} \\ B &: \text{draw a line and move forward} \\ r &: \text{turn right} \\ l &: \text{turn left}\end{aligned}$$

The above interpretation of the L-system presented in (1.1) will generate a fractal called the *Sierpinski triangle*. A string generated from a grammar in Grammatical Framework could thus be interpreted in another programming language, e.g. Python, to draw a fractal or generate musical notes [7]. A possible way to interpret the alphabet as music could be as follows:

$$\begin{aligned}A &: \text{play the current tone} \\ B &: \text{play the current tone} \\ r &: \text{raise pitch} \\ l &: \text{lower pitch}\end{aligned}$$

Since A and B both have the same interpretation in these examples, they will henceforth be replaced with the letter F .

1.3 Grammatical Framework

Grammatical Framework (GF) is a programming language, developed in 1999, used for defining different types of grammars in a programming context [3]. It is – according to Ranta [3] – mainly used for natural languages and serves as a tool for translating between them by defining complex grammatical structures for each language. Ranta writes that the design of GF is based on typed functional programming languages such as Haskell, but since the users of GF vary from programmers to linguists it needs to be accessible to people of various backgrounds. Though the main application of GF lies within natural languages, it can also be used for other purposes where a grammar is a core component. One example is fractals based on Lindenmayer systems (see Section 1.2).

Ranta describes that when translating between two languages/grammars in GF, one defines an *abstract syntax* where the axiom and the rules for the grammar are defined. The abstract syntax is inherited by one or many *concrete syntaxes*, where the different alphabets are defined [3]. A system with one abstract syntax and several concrete syntaxes is called a *multilingual grammar* [3]. If there are operations that are usable in more than one concrete syntax, Ranta writes that *resource modules* could be used. [3] states that a resource module defines these operations and that it can be opened by a concrete grammar that thus can use the operations.

1.3.1 Simulated recursion in GF

To achieve a fractal by GF generation the defined grammar needs to be called in a recursive manner. However, recursion is not supported by any type of grammar and will therefore be simulated by repeatedly calling a function on itself, any number of times. An example of recursion in GF reads as follows:

```
c(s(s(s(s z))))
```

This call would have a recursive depth of 4, where the function for simulating the recursion is called *s*, the function for describing the axiom is called *z*, and the whole expression – where the *c* function describes the composing of the string – generates a full fractal.

1.3.2 Phrases in GF

In natural languages, sentences can be divided into smaller components called *clauses* and *phrases*. The phrases of a sentence can be represented in a *phrase tree*, see Figure 1.1. A phrase can consist of either a group of words or a single word building a meaningful unit within a clause; furthermore, there are different categories of phrases such as *noun phrases* and *verb phrases* [8]. The GF function *brackets* can split a sentence or clause into phrases, marking the phrases within round brackets and a number to indicate the order of the phrase. The order of a phrase gives information about on what level in the phrase tree the phrase belongs.

Even though the strings representing the fractals in this project do not consist of words, GF can still divide these strings into phrases using the bracket function. An example of this can be seen in Figure 1.2, where the string " $F r F r F l F l F l F r F r F$ ", representing the third iteration of the Sierpinski triangle, is split into different phrases where each phrase consists of smaller sub-phrases. When using the bracket function with this string (generated using the Sierpinski concrete grammar in Appendix B), the output in GF reads:

```
(N:2
  (N:1 (N:0 F) r (N:0 F) r (N:0 F)) l
  (N:1 (N:0 F) l (N:0 F) l (N:0 F)) l
  (N:1 (N:0 F) r (N:0 F) r (N:0 F))
)
```

Each phrase is marked – within round brackets – with an N and the order of the phrase. All F are of order zero, and each phrase of order one consists of multiple sub-phrases of order zero, and so on. For the Sierpinski triangle the number of sub-phrases of order $n - 1$ in a phrase of order n is three; however, this varies among the different fractals.

Basic constituent structure analysis of a sentence:

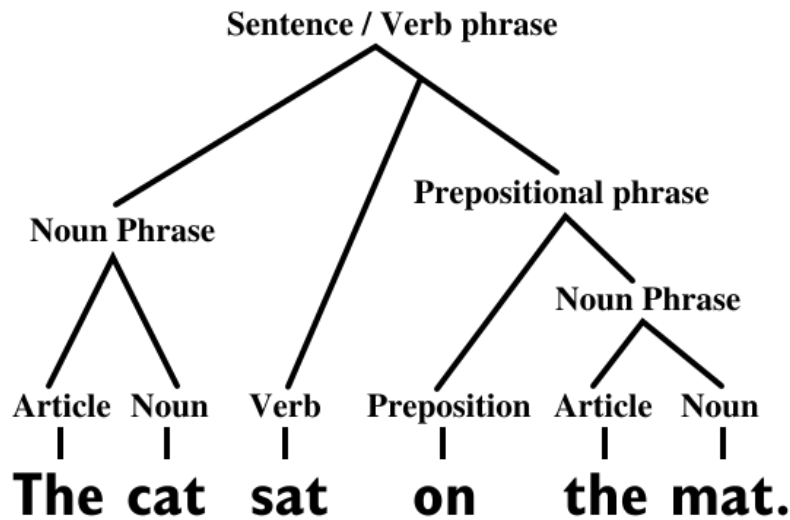


Figure 1.1: An example of a phrase tree for natural languages, where the sentence "The cat sat on the mat." is divided into phrases and sub-phrases. From [9]. Reproduced with permission.

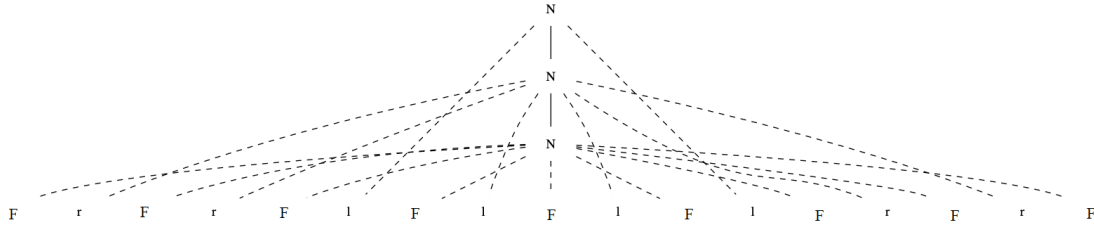


Figure 1.2: The string "*F r F r F l F l F l F l F r F r F*" represented by a phrase tree, where the string is split into different phrases. The different levels in the tree (marked with *N*) represent different orders of phrases. Each phrase consists of smaller sub-phrases, where order zero *F* build up higher order phrases.

1.4 MIDI files

MIDI (Musical Instrument Digital Interface) is a communication protocol invented in 1983. It is used as a tool for communication between digital musical instruments [10]. According to [10], there are three major parts of the MIDI system: a message format, a storage format and a physical connector. The message format is used for communication between musical instruments and computers by translating the music into programming language, making it possible to e.g. change recorded music by using a computer, states [10]. The file extension *.mid* is an industry standard that can be used by many different music programs [10]. To connect the musical instruments to each other, or to a computer, [10] describes that physical connectors such as MIDI cables are used. [10] further describes that MIDI messages are received through the in-port and MIDI data is transmitted through the out-port to the computer.

1.5 Turtle graphics

Turtle is a graphical tool, mainly developed for introducing programming to children in the LOGO programming language [11]. According to [11], it is based on the user giving commands such as *LEFT 45* and *FORWARD 20*. The first command will change the starting angle of the Turtle by 45 degrees and the second will draw a line with the length 20 steps [11]. Turtle graphics is mainly used for primitive graphical representations, such as lines and simple geometrical shapes, as stated in [11].

1.6 Web server architecture

A web server – as described by Simon Collin in [12] – can either refer to a physical computer which has the task of storing web server software and files, or a software that is running on a physical server. Collin states that whenever a web browser visits a website, a request is made to the web server which returns a file that tells the web browser how to render the website and how interactions from the user should be handled.

1. Background

Client-server architecture is a software design pattern which describes how two computers can communicate with each other, according to Craig Larman in [13]. The task of the server is to perform some service and listen for requests made to it [13]. Larman further states that the client (often a web browser or a UI) will send a request to the server specifying a service to be performed.

According to the official documentation of WebSockets by Ian Fette and Alexey Melnikov [14], WebSockets describe a protocol for performing two-way communication between a client and a host. Fette and Melnikov further specify that the connection is a two step process which begins with a *handshake* between the host and the client followed by the messages to be sent. A new connection does not need to be established for each new message, but the same message-frame is reused for all messages until the connection is closed [14].

The JSON file format is designed for sending data between computers [15]. The format is easily read by humans, and at the same time practical for computers to generate and read [15].

2

Task

The aim of the project was to use GF as a tool to explore how certain fractals constructed from L-systems relate to one another, as well as how these fractal images can be interpreted as music. A secondary goal was to create a user interface that would allow a broad spectrum of users (as mentioned in Chapter 1), including those lacking knowledge of fractals and/or GF, to explore and visualise the fractal relations.

In order to succeed with the task, a few main problems were identified. There had to be feasible a way to represent the fractals graphically as well as a way to translate between different types of fractals, using GF and L-systems. To enable this, the best suited format of the GF-output had to be decided, as well as a way to parse the output and interpret it as music. The next step was to find a suitable way to enable storing and accessing of the information generated in GF. Furthermore, the implementation of the user interface had to be decided and there needed to be a way to take the information generated in GF and visualise it in a suitable way. The level of user interactivity had to be decided as well, and the aim was to incorporate a virtual piano into the user interface. In that way, the user could input notes or pieces of music, and thereby have some sort of impact on the rendering of the fractal images and/or musical interpretation of said images.

3

Scope

3.1 Ethical aspects

At an early stage ethical aspects were deemed irrelevant for this project. The only thing to consider is not to use up too much research funds that could have better use elsewhere, which affected the musical input (see Section 3.3).

3.2 Fractals used

The fractals used in this project are limited to those that can be represented by an L-system (see Section 1.2), since the main purpose is to translate between different fractals and not to generate complex ones. Furthermore, the fractals are also limited to ones where the difference between two consecutive iterations can be described by replacing one line segment with a set of lines matching the first iteration, see Figure 3.1 for example. Therefore, the fractals included in the scope are the *Sierpinski triangle* (Figure 3.2), the *Dragon curve* (Figure 3.3), the *Gosper curve* (Figure 3.4), the *Koch square snowflake* (Figure 3.5), and the *Hilbert curve* (Figure 3.6). Conversely, fractals such as the *Mandelbrot set* (Figure 3.7) were not included in the scope.

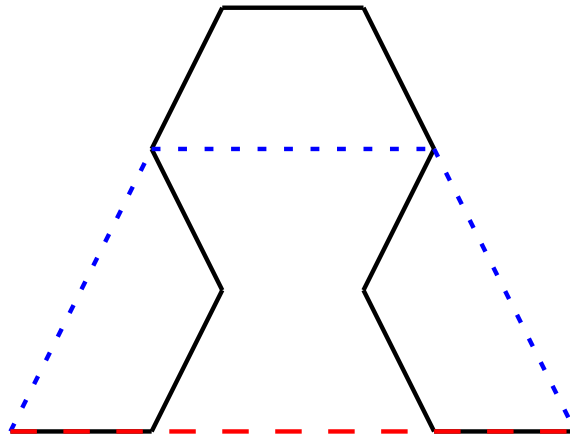


Figure 3.1: The first two iterations of the Sierpinski triangle. The red dashed line represents the axiom ($n = 0$), the blue dotted line is the first iteration ($n = 1$) and the black line is the second iteration ($n = 2$). Note that the second iteration is generated by changing all line segments in the first iteration to a self-similar piece.

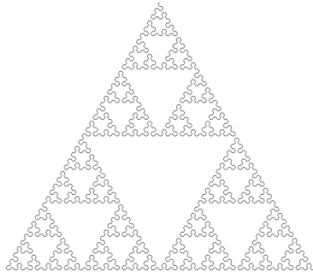


Figure 3.2: An example of the Sierpinski triangle drawn from instructions generated in GF with the grammars defined in Appendix B.

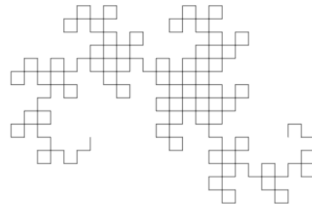


Figure 3.3: An example of the Dragon curve, eight iterations drawn from instructions generated in GF with the grammars defined in Appendix B.

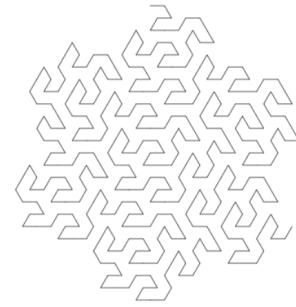


Figure 3.4: An example of the Gosper curve, three iterations drawn from instructions generated in GF with the grammars defined in Appendix B.

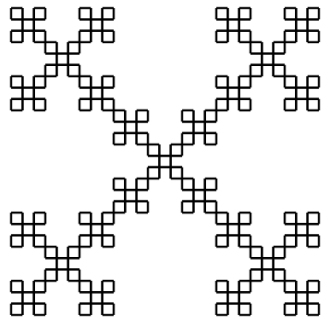


Figure 3.5: An example of the Koch square snowflake drawn from instructions generated in GF with the grammars defined in Appendix B.

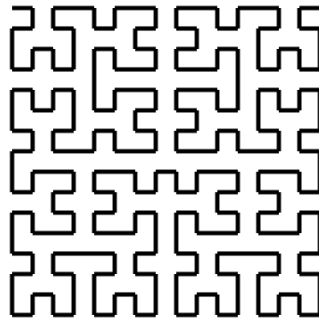


Figure 3.6: An example of the Hilbert curve drawn from instructions generated in GF with the grammars defined in Appendix B.

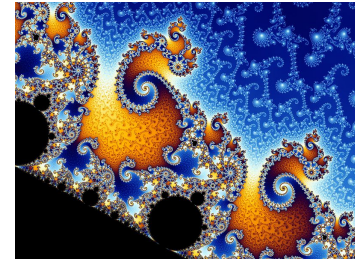


Figure 3.7: An example of the Mandelbrot set. This fractal was omitted from the scope. From [16]. CC-BY-SA.

3.3 Musical input

As stated in Section 3.1, it is important not to use up too much research funds that could have better use elsewhere. Therefore, a virtual keyboard was deemed most suitable for this project, as opposed to a physical digital keyboard with a MIDI connection (see Section 1.4). No other musical instruments are taken as input since that will have an impact on both the expenses and the complexity of the project.

3.4 Stochastic rules

Stochastic rules were not used for the L-systems in the project. However, at quite an early stage the musical representation was made stochastic in terms of the note length. The stochastic note length was added to make the music more interesting to listen to, instead of having the same duration for all notes.

3.5 Grammar input

The user is not able to define their own grammars and give them as input in the GUI. The grammars that are used are predefined in GF and not open for change from the front end.

4

Method

The fractals in this project are represented by strings generated from grammars in Grammatical Framework. The strings are the blueprints for drawing the fractals – and creating the music – generated by the grammar. In order to save, store and easily access needed information about the fractals, a data structure was created. After creating a simple prototype of the GUI, the project’s first step regarded generating the strings and translating from one fractal image to another. The second step concerned parsing the strings – turning the commands described in the GF alphabet into drawing instructions and MIDI-generation instructions. The third step involved drawing the fractal images and playing the music as well as developing the user interface (a web application). When these three tasks were successfully completed, further development of user interactivity was examined. This included a virtual keyboard where the user can provide input to alter the rendering of the fractal images.

4.1 Initial stage

Before the development of the product began, a prototype (see Section 4.1.1) was created in order to get an idea of how the finished product might behave and look like. A suitable structure of the data flow was also decided (see Figure 4.1).

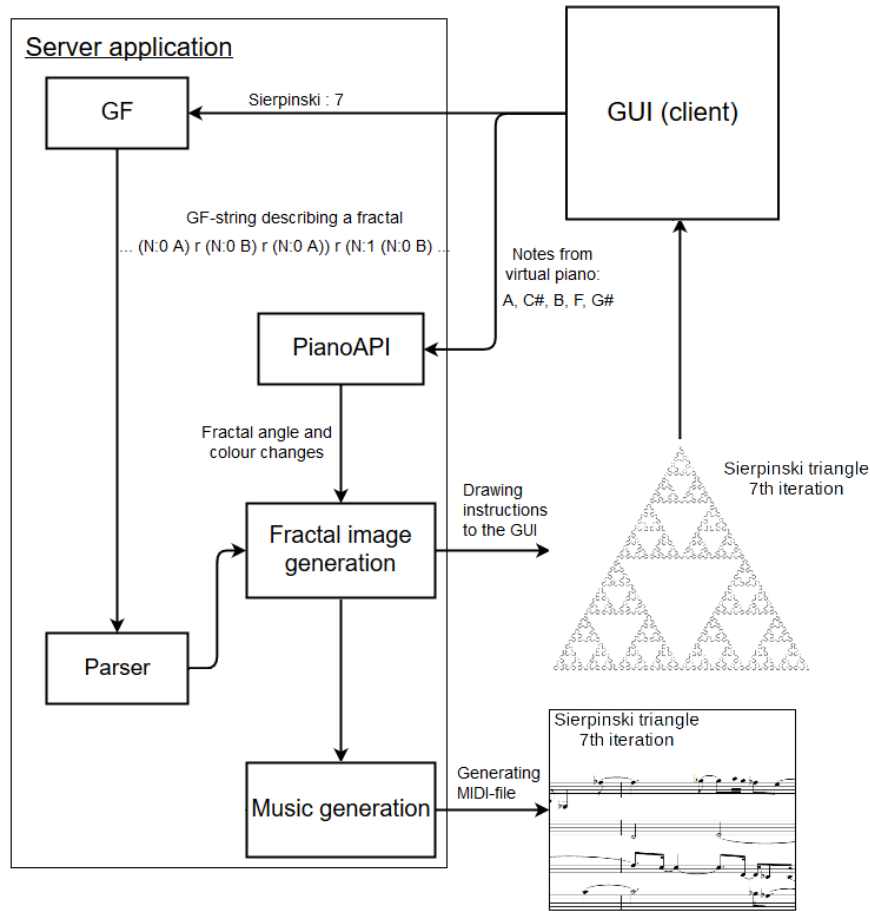


Figure 4.1: An overview of data flow through the application.

4.1.1 Prototype

Early on in the project, a simple prototype (see Figure 4.2) of the product was created. The main purpose of this prototype was to understand how the Turtle implementation would work in a web application. The prototype was programmed with Python and Brython. The user can play a few notes on the keyboard, and these notes will then appear in the leftmost grey box. The notes played by the user will be the axiom for the rules used to build the figure drawn in the prototype. The user can also enter how many iterations they want to perform. The rules used in the prototype are not derived from any existing L-system, and hence do not represent a fractal. The prototype is available at [17].

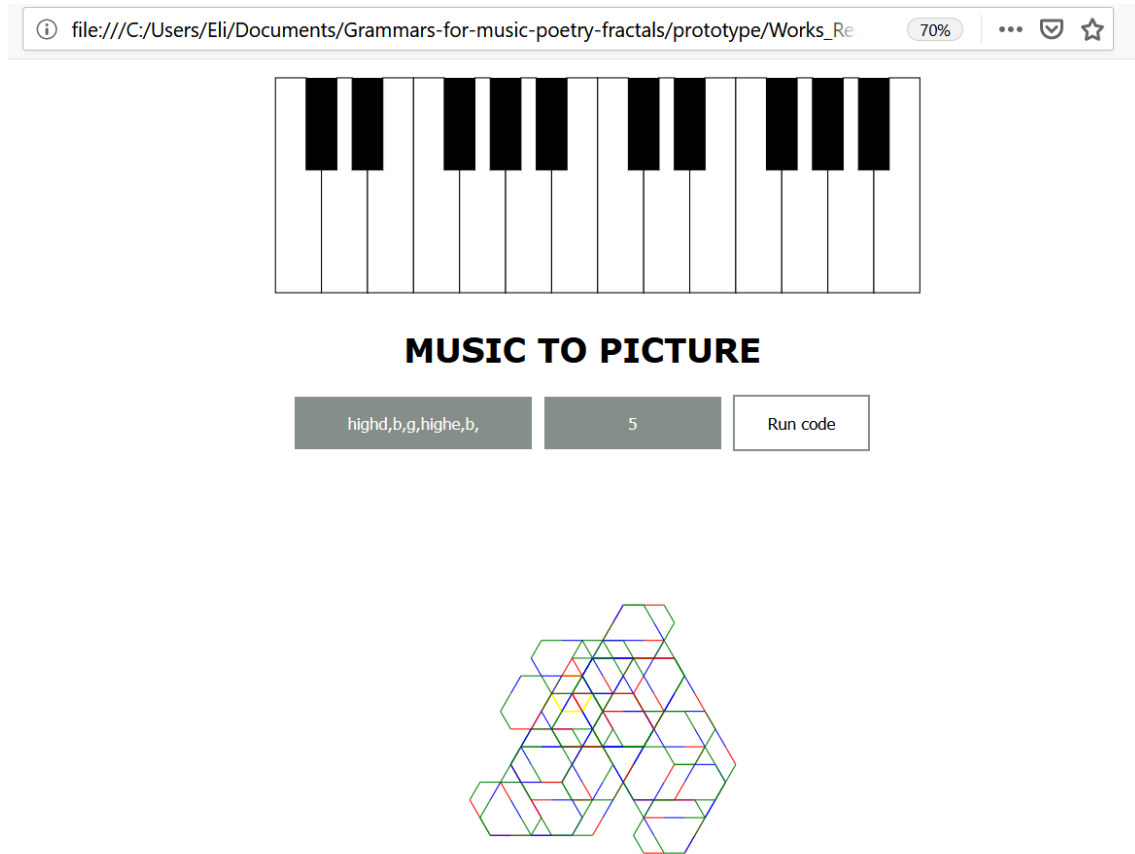


Figure 4.2: A snapshot from the prototype. The notes played by the user appear in the leftmost grey box. The user can also input the number of iterations they want the prototype to perform (thus influencing the complexity of the rendered image). The code for the piano (CSS and HTML) was adapted from [18]. The prototype is available at [17].

4.1.2 Exploring existing L-systems for music

As music can be viewed as being two-dimensional, with notes having both the property of pitch and duration, an interpretation of the strings as music is possible. There are a few existing L-systems describing music as well as musical interpretations of systems describing fractals [19] [20]. Prusinkiewicz [19] describes the following interpretation, which inspired the final choice of implementation in this project:

- A starting tone and scale is defined.
- The image is walked through as if drawing it using Turtle.
- Any horizontal movement is interpreted as playing a note with length proportional to the length of the corresponding line segment and with a pitch determined by the y-coordinate of the line segment.

There are, however, some limitations to this interpretation. Firstly, it only applies to fractals that only use 90°-angles, such as the Dragon curve or the Koch square snowflake. A solution to this limitation, suggested by Stelios Manousakis [20], is

to interpret sloped line segments as glissandi (see Appendix A). However, glissandi is unfortunately not supported by the *MIDIutil* Python library used for generating MIDI files (see Section 1.4). Secondly, the music generated with the interpretation above has very repetitive rhythm. Since the line segments of the fractals in this project all have the same length, only one note length will appear and the only intervals that appear in each part are of either one or two semitones. The final implementation chosen for the project is described in Section 4.4.2.

4.2 GF generation

Through a GF grammar in the form of an L-system, strings corresponding to a certain fractal and a specific iteration are generated. The strings are then interpreted and represented graphically; furthermore, the musical interpretation is based upon the graphical representation.

The rules of the fractal L-systems are applied iteratively in GF, thus generating strings that represent the two-dimensional growth of the fractals. As long as the grammars for the different fractal images implement the same abstract grammar in GF, translation between said images is possible.

Already existing grammars for fractals (available at [21]) provided the starting point for this project. Below is an example of the concrete grammar for the Sierpinski triangle from [21]:

```
concrete Sierpinski of Graftal = {
  lincat N = {a : Str; b : Str} ;
  lincat S = {s : Str} ;

  lin z = {a = A; b = B} ;
  lin s x = {a = x.b ++ R ++ x.a ++ R ++ x.b; b = x.a ++ L ++ x.b ++ L ++ x.a} ;
  lin c x = {s = "newpath 300 550 moveto" ++ x.a ++ "stroke showpage"} ;

  oper A : Str = "0 2 rlineto" ;
  oper B : Str = "0 2 rlineto" ;
  oper L : Str = "+60 rotate" ;
  oper R : Str = "-60 rotate" ;
}
```

These concrete grammars are constructed to generate a file in Postscript, which is an unsuitable format for the visualisation in this project. Therefore Postscript-specific functions and linearisations, as well as the alphabet, were modified to suit the implementation in this project. A resource model was also created to make the concrete grammars more compact. The Sierpinski triangle grammar in the new format is shown below:

```
concrete Sierpinski of Graftal = open Operations in {
  lincat N = {a : Str; b : Str} ;
```

```

lincat S = {s : Str} ;

lin z = {a = F; b = F} ;
lin s x = {a = x.b ++ L ++ x.a ++ L ++ x.b; b = x.a ++ R ++ x.b ++ R ++ x.a} ;
lin c x = {s = "ang:60" ++ x.a} ;
}

```

A full list of the concrete grammars for the different fractals, as well as the abstract grammar and the resource module, used in this project are available in Appendix B. The alphabet used for fractal description in the project is presented in table 4.1.

Table 4.1: The different operations available in the defined grammars.

Operation	Example
Draw forward	F
Turn right	r
Turn left	l
Set the turn angle of the fractal	ang:60

An example of a string generated with the GF bracket function (see Section 1.3.2), in this case the second iteration of the Sierpinski triangle, follows:

```

(S:3 ang:60 (N:2 (N:1 (N:0 F) r (N:0 F) r (N:0 F)) l
(N:1 (N:0 F) l (N:0 F) l (N:0 F)) l (N:1 (N:0 F) r (N:0 F) r (N:0 F))))

```

Since the fractals used in this project only consist of straight line segments and turns, the required alphabet is small. The strings generated in GF are sent to a parser to be interpreted into something more meaningful for the other components in the product.

4.3 Data structure

In order to store and access necessary information about the fractals, a data structure (see Figure 4.3) enabling this was created. The data structure is based on a tree-structure filled with *Nodes*. In this project, each Node contains a *LineSegment* that is created using a graphicless Turtle adaption (further described in Section 4.4.1), but could theoretically contain any arbitrary object. The LineSegment has a start position, an end position, a colour, a note length and a boolean to tell whether or not to start a new musical part (see Figure 4.4).

The Nodes are added iteratively to arraylists called *Layers*. The Layers are divided into different segments called *ListSets* where the number of ListSets per Layer depends on both the order of the Layer and which fractal is being studied. Each ListSet is doubly linked and holds a reference to both its parent and its children. If the Layer is of order n , then each ListSet in the Layer corresponds to a phrase of order n . The number of phrases of order n depends on how many children per parent the fractal has. In Figure 4.3 for example, the number of children per parent

is three, and thus the number of ListSets per Layer of order n in a fractal with highest iteration h is 3^{h-n} . The number of children per parent depends on the rules of the grammar.

References to all Layers are stored in a separate list (not part of the tree), making it possible to retrieve a direct reference to any Layer without having to traverse through the tree. This improves the performance and simplifies translation between fractals. An example of how the data structure is used for the Sierpinski triangle can be seen in Figure 4.5.

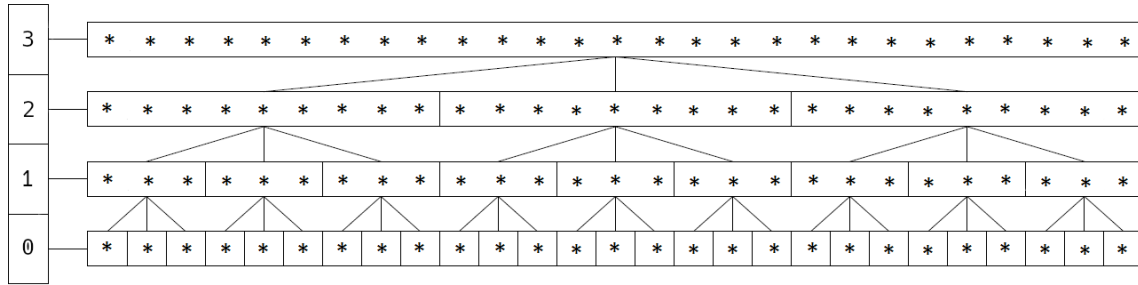


Figure 4.3: The data structure used in the project consists of a list of Layers, where each Layer has an order n , where in this example $n = 0, 1, 2, 3$. Each Layer contains all the information needed to form a full fractal, and the Layers are divided into ListSets representing phrases in the fractal. The number of ListSets per Layer depends on the order n of the Layer, as well as the number of children per parent in the fractal. The Layers in the list are also linked to each other in a tree, such that each Node (marked with an asterisk) in the Layer belongs to a ListSet which is connected to one parent and/or several children. In this example the number of children per parent is three; however, this varies among the different fractals.

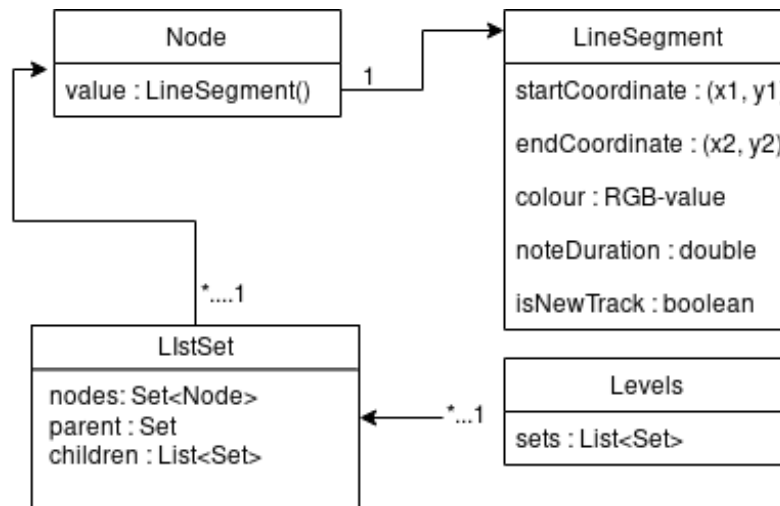


Figure 4.4: UML-diagram of the classes that fill the different parts of the data structure in Figure 4.3.

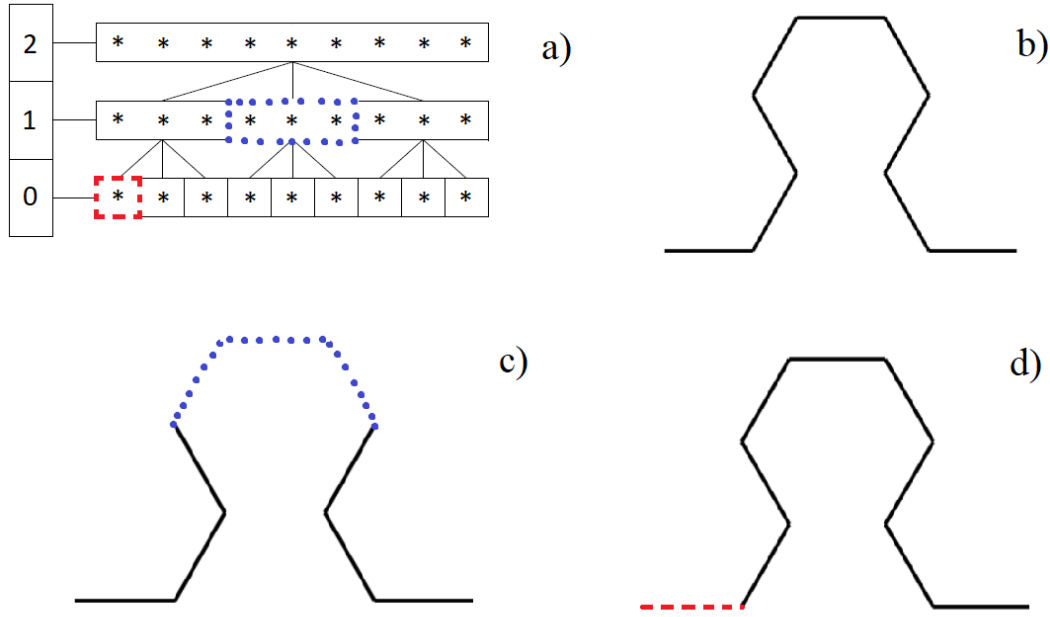


Figure 4.5: The second iteration of the Sierpinski triangle, shown in **b)**. The Layers in **a)** all contain nine Nodes, where each Node contains information according to Figure 4.4. The Layer $n = 2$ consists of one single ListSet with all nine Nodes, the Layer $n = 1$ consists of three ListSets with three Nodes in each ListSet, and for $n = 0$ there are nine ListSets which all contain only one Node. In **a)**, the ListSet marked with blue dots represents the phrase marked with blue dots in **c)**. The ListSet marked with dashed red in $n = 0$ corresponds to a phrase of order zero, marked with dashed red in **d)**. For $n = 2$, the ListSet of Nodes represents a phrase of order two, and the graphical representation of such a phrase is identical to **b)**.

4.4 Parsing

The information in the data structure described in Section 4.3 is used both when the product generates music, as well as for the graphical representation of the fractals. A string that represents a fractal is generated by GF, and said string is then parsed by a *Parser* written in Python.

4.4.1 From string to tree

The data structure is filled with Nodes containing LineSegments that, for convenience when generating the music, each have an associated note length. In order to generate the LineSegments from the strings generated in GF, an adaptation of Turtle was created. The Turtle adaption is called *HiddenTurtle* and only keeps track of its position, without drawing anything. *HiddenTurtle* is not an adaptation of existing Turtle libraries in Python – it was created for this project. The way in which the LineSegments are created and stored in the data structure is done as follows:

- A base angle α and the depth of the tree (corresponding to the highest iteration h of the fractal) are defined. α is the number given by the *ang: α* constant (see table 4.1).
- When a *right* or *left* command is given, the HiddenTurtle changes its angle by $\pm\alpha$.
- When an opening round bracket "(" is found, a new ListSet corresponding to the depth of insertion is created in the Layer.
- When a *forward* command is given, the current position of the HiddenTurtle is stored and the HiddenTurtle then moves a predefined distance.
- A LineSegment is created between the stored position and the current position of the HiddenTurtle and is stored in the data structure.
- When a closing round bracket ")" is found, the ListSet containing all data within the round brackets is ended.

When the tree is filled this way, each Layer of order n consists of the information in the Layer of order $n - 1$; however, additional information can be added to the Layer depending on the fractal. In Figure 4.3 Layer two corresponds to a fractal of iteration four, using nine different ListSets each consisting of three LineSegments.

4.4.2 Musical interpretation

From the information stored in the data structure, a musical representation is created. The highest order Layer in the data structure is interpreted as a MIDI file (see Section 1.4) in the following way:

- A key note, a musical scale and a maximum length of the musical parts (see Appendix A) are defined.
- Each LineSegment in the image is interpreted as a note.
- The first LineSegment in the image is interpreted as the key note of a random note length between a sixteenth and a whole note, all equally likely.
- In order to determine the following notes, the angle θ between the two LineSegments corresponding to the notes is taken into consideration. The interval is $\theta/30^\circ$ semitones, rounded down to the closest integer. Left turns raise the pitch and right turns lower it.
- The note is then fitted to the predefined musical scale by changing the tone to the closest one in the musical scale. If the two closest tones are equally close, the lower one will be chosen.
- If the length of the musical part will exceed the predefined maximum length if the note is added, a new musical part is created starting over at the key note.

In order to play the music from the web application, the MIDI format is not suitable. Therefore, a *WAV file* containing the same music is also created using the program *TiMidity++*. In Section 4.4.3, the impact of the user input from the virtual piano on the graphical representation of the fractals is described. Since the musical interpretation is based on said graphical representation, the input from the virtual piano will indirectly affect the musical interpretation of the fractals.

4.4.3 Virtual piano input affecting graphical representation

As there were no predefined rules of how piano input should affect fractal visualisation, a system interpreting the notes of a piano as alterations to the drawing operations was defined. The turning angle α for right and left turns between LineSegments, as well as the LineSegment colour are changed according to table 4.2 depending on which piano keys are pressed.

Table 4.2: Different changes to the drawing instructions for the fractal images depending on which piano key is pressed. The changes are stored in a list; thus, pressing a new key on the virtual keyboard does not overwrite previous commands.

Note	Right turning angle	Left turning angle	Colour
C ₃	-1°		
C ₃ [#]			#FF0000 (red)
D ₃		-1°	
D ₃ [#]			#FF8000 (orange)
E ₃	+1°		
F ₃			#FFFF00 (yellow)
F ₃ [#]		+1°	
G ₃			#80FF00 (bright green)
G ₃ [#]	-2°		
A ₃			#00FF00 (green)
A ₃ [#]		-2°	
B ₃			#00FF80 (turquoise)
C ₄	+2°		
C ₄ [#]			#00FFFF (baby blue)
D ₄		+2°	
D ₄ [#]			#0080FF (bright blue)
E ₄	-3°		
F ₄			#0000FF (blue)
F ₄ [#]		-3°	
G ₄			#7F00FF (purple)
G ₄ [#]	+3°		
A ₄			#FF00FF (pink)
A ₄ [#]		+3°	
B ₄			#FF007F (magenta)

By using the rules stated in table 4.2, piano input from the user interface read by the *Piano* class in Python will generate three lists:

- A list of colours.
- A list of right angle changes.
- A list of left angle changes.

These three lists are then added to the *TreeFiller* class through the Parser and are interpreted when generating the Nodes in the data structure. The changes of colour

and turning angle are applied to information in the data structure as follows:

- For a list of c colours and l LineSegments:
The list of l LineSegments is divided into c parts, with $(l/c) + (l \bmod c)$ LineSegments in each part. The LineSegments in the first part are given the first colour, those in the second part are given the second colour and so on.
- For a list of a_r right angle changes and t_r right turns:
The list of t_r right turns is divided into a_r parts, with $(t_r/a_r) + (t_r \bmod a_r)$ right turns in each part. The right turns in the first part of the list are changed by the first right angle change, those in the second part of the list are changed by the second right angle change and so on.
- For a list of a_l left angle changes and t_l left turns:
Same as for the right turns, but with a_l instead of a_r and t_l instead of t_r .

4.5 Web application

The web application consists of a server and a client. The server (see Section 1.6) holds GF, the generation of strings, and all Python code. To achieve the goal of making the product interactive, an already existing JavaScript piano (available at [22]) was integrated into the web application. The virtual piano was stripped down of JavaScript code and CSS styling deemed redundant for this project. Instructions for playing the virtual piano from the computer keyboard are presented in Figure 4.6.

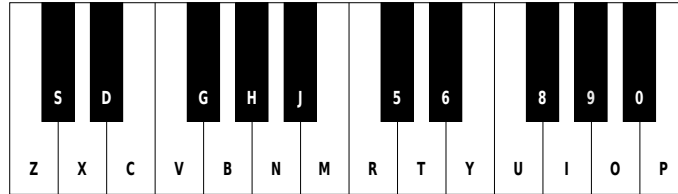


Figure 4.6: Instructions for playing the virtual piano on the web application from the users computer keyboard. For example, the note C_3 can be played either by clicking the leftmost key on the virtual piano, or by pressing the z -key on the computer keyboard. To be able to play from the computer keyboard enables the user to play more complicated pieces of music, since multiple keys can be pressed at once. The setup of how the computer keys (on a Swedish QWERTY keyboard) correspond to the virtual piano keys was made to resemble the fingering used when playing a real piano.

4.5.1 Client-server architecture

The architecture used in this product consists of a web server, which holds the fractal generation, and a client (the web application with the GUI, as seen in Figure 4.1). After taking input from the user, the client sends a request to the server to generate the commands needed to draw a fractal. The server generates and returns a command to the client with details regarding the coordinates where the front end should draw lines, which colours they should be, and other information regarding

the fractal.

For the client to communicate with the server, WebSockets are used. If a client's request to connect to the server is successful, a connection between the server and the requesting client is established. Once the connection has been established, the client will send a JSON file to the server containing what sort of operation it wants the server to perform, as well as the data required for the operation.

4.5.2 Drawing and translating between fractals

The fractal images on the web application are rendered using the HiddenTurtle (Section 4.4.1) for coordinate calculation and JavaScript for drawing. In order to make the fractals clickable – and to be able to compare different phrases (see Section 1.3.2) in the fractal images – an HTML canvas is used as drawing surface, and to register mouse events. Since the fractals vary in size, a scaling method was developed. The scaling is implemented by finding the maximum and minimum x and y coordinates of the fractal and dividing the size of the canvas with the distance between the minimum and maximum coordinate in the fractal.

The translation between fractals was for this project interpreted as comparing their phrases. When translating phrases between fractals, the user chooses which order of phrase they want to highlight in the fractal. When the user clicks the canvas, the coordinates of the cursor are compared to the coordinates of the LineSegments stored in the data structure. Through the data structure, corresponding phrases in the fractal images can be found and highlighted in both fractal images (see Figure 4.7)

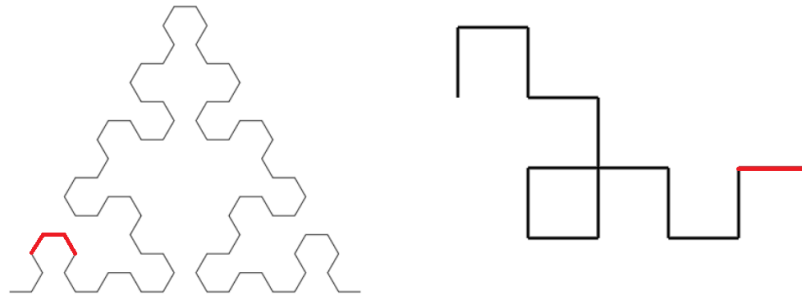


Figure 4.7: The fourth iteration of the Sierpinski triangle and the Dragon curve. The second sequential phrase of order one is highlighted in red in both of the fractal images.

If the number of sub-phrases per phrase in the first fractal is the same – or less – as in the second fractal, then when the user selects a sub-phrase in the first fractal the sub-phrase that comes in the same sequential order in the corresponding GF string is highlighted in the other fractal (e.g. in Figure 4.7 the second sub-phrase of order one is highlighted). Conversely, if the first fractal has a higher number of sub-phrases

per phrase, the selection of a corresponding sub-phrase has to be done differently. Should the user select a phrase in the first fractal with a higher sequential order than the number of sub-phrases in the second fractal, then the sub-phrase that is highlighted in the second fractal is calculated using the modulo operation.

5

Result

One of the main goals of the project was to make the result accessible to the general public, hence the developed product is a web application. There are three pages on the web application: *Fractal-O-Mat* (see Figure 5.1), the front page, *Fractal Fun* (see Figure 5.2), for exploring fractals, and *Resources* (see Figure 5.3) which contains useful links. The front page contains an explanatory image that shows the main components and structure of the project, as well as how the components interact with each other. On the front page, the user can also download this bachelor's thesis report. To make the web application more user friendly, *tooltips* and other explanatory features were added to several of the elements (as seen in Figure 5.4 and 5.5) on the Fractal Fun page. The features of the product make it accessible to most people, regardless of their operative system or what knowledge they may – or may not – have about fractals, GF, music theory, etc.

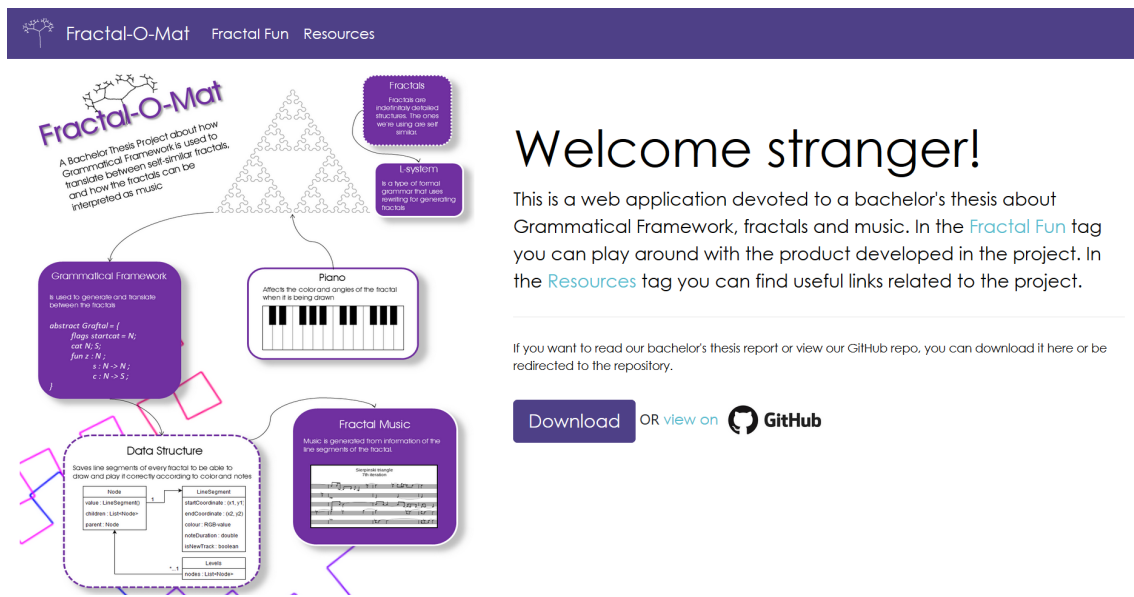


Figure 5.1: The front page of the web application consists of an explanatory image of the general structure and main components of the project, as well as some information to help the user navigate through the application. It is also possible to download this bachelor's thesis report or inspect the code base of the application on GitHub [23]. The front page is accessed by clicking on *Fractal-O-Mat* in the navigation menu.

On the Fractal Fun page, the user can choose two fractals that they want to generate

5. Result

(*Fractal 1* and *Fractal 2* in Figure 5.2) as well as which iteration of the fractals should be generated. The number of iterations (the choosable number depends on the fractal) decides how complex the fractal images are going to be. The two fractal images (in Figure 5.2 the fifth iteration of the Sierpinski triangle and the Dragon curve) are drawn on the web application. By choosing what order of phrases the user wants to study in the *Choose order of phrases* drop down list and then clicking the left fractal, some parts of it are highlighted in red. These parts represent a phrase of the selected order in the left fractal, and the corresponding phrase is also highlighted in the right fractal (see Section 4.5.2).

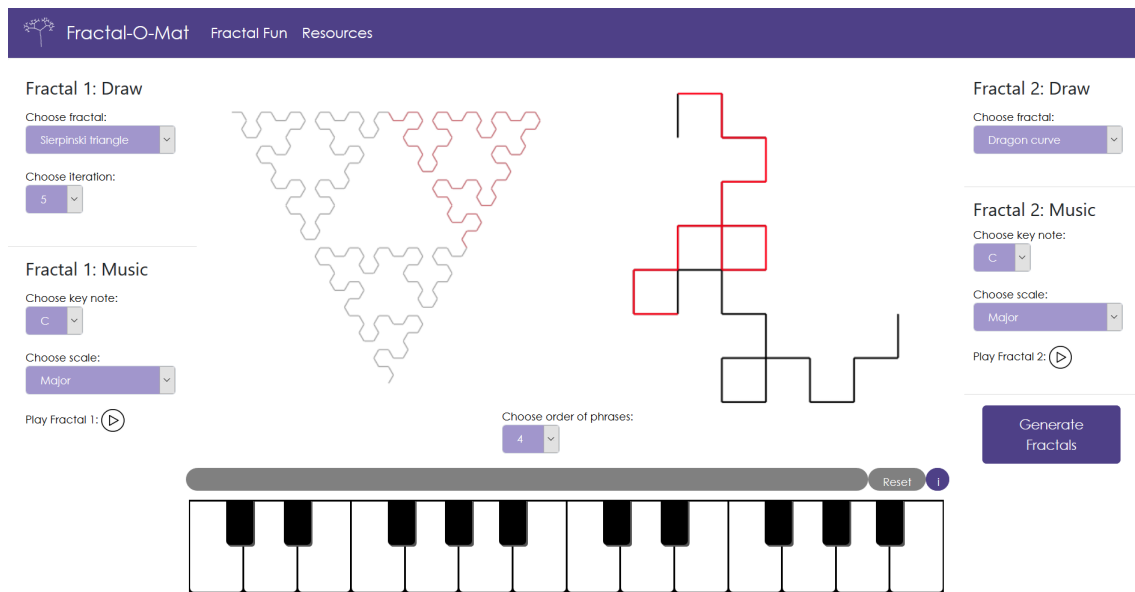


Figure 5.2: A screenshot from the web application where the user has chosen to generate the fifth iteration of the Sierpinski triangle and the Dragon curve. The fractals are chosen below *Fractal 1: Draw* to the upper left and *Fractal 2: Draw* to the upper right, whereas the iteration is only chosen below Fractal 1: Draw. The user has chosen to highlight phrases of order four and then clicked the left fractal. The corresponding phrase is highlighted in red in the right fractal. The page is accessed by clicking on Fractal Fun in the navigation menu.

The user also has the option to listen to a musical interpretation (see Section 4.4.2) of the fractal images, by clicking the play button for each of the fractals (see Figure 5.2 or 5.7), after choosing a desired musical scale and a key note in drop down lists. The types of musical scales that are available are the major, major pentatonic, minor, minor harmonic, minor pentatonic and blues scales. The key notes span an octave. Figure 5.6 shows the sheet music for a musical interpretation of the fourth iteration of the Sierpinski triangle.

The web application also contains a virtual piano keyboard spanning two octaves (see Figure 5.2 and 5.7). This allows the user to play a piece of music by clicking the piano keys or pressing certain keys on their computer keyboard (see Figure 4.6). Depending on what notes are played, the rendering of the fractal images is affected according to the rules in table 4.2.

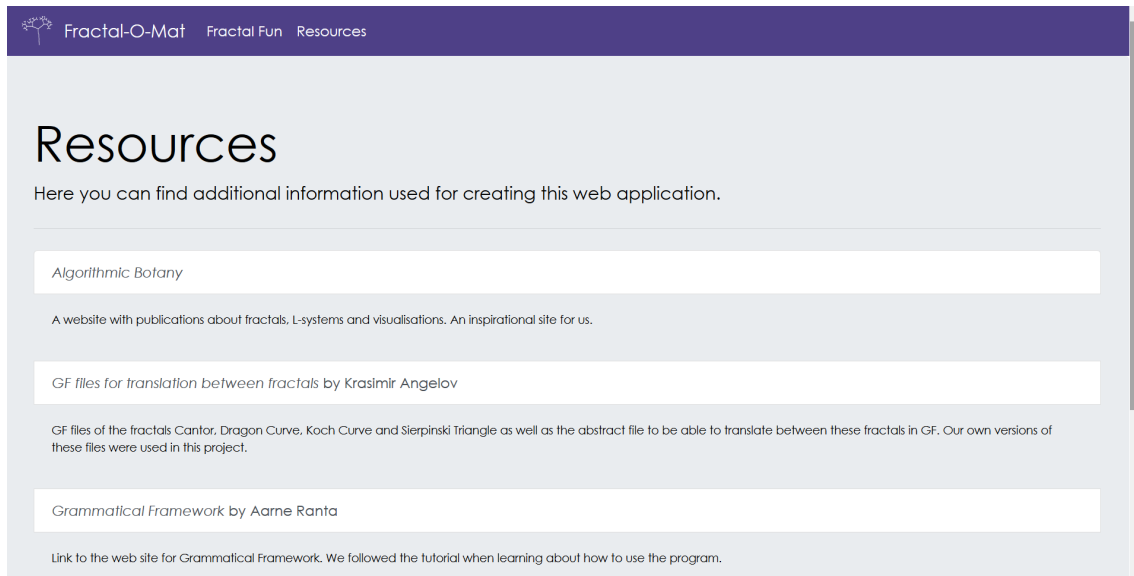


Figure 5.3: The Resources page provides links to information and inspiration regarding the project. The page is accessed by clicking on Resources in the navigation menu.

Fractal 1: Draw

Choose fractal:

Choose a fractal to draw in the left canvas

Choose iteration:

Higher iterations give more complex fractals

Figure 5.4: When hovering the drop down lists, tooltips will appear. The tooltips describe the functions of the respective drop down lists.

5. Result

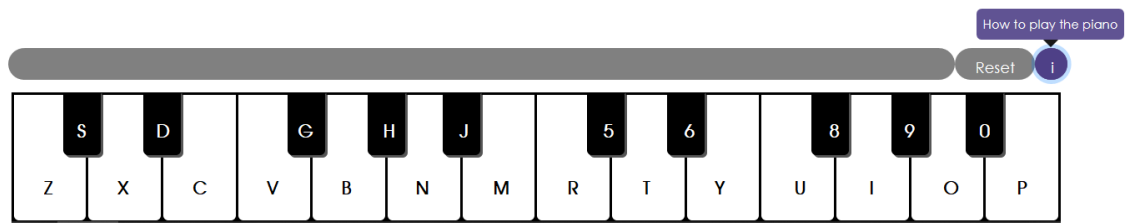


Figure 5.5: The tooltip for the virtual piano keyboard appears when hovering the purple information button. By clicking the button, a help piano will appear showing what keys on the computer keyboard the user should press to play the virtual piano.

Score

Sierpinski triangle - fourth iteration

Maestro Fractal

Track 1

Track 2

Track 3

Track 4

A screenshot of a sheet music score titled 'Sierpinski triangle - fourth iteration' by 'Maestro Fractal'. The score is displayed on four tracks, labeled 'Track 1' through 'Track 4'. Track 1 is in bass clef and 4/4 time, showing a complex melodic line with many beamed sixteenth notes. Tracks 2, 3, and 4 are in treble clef and 4/4 time, showing a more sparse accompaniment with fewer notes and some rests.

Figure 5.6: Sheet music for the music generated from the fourth iteration of the Sierpinski triangle. Music is first generated as a MIDI file and then converted to sheet music by an external program.

5. Result

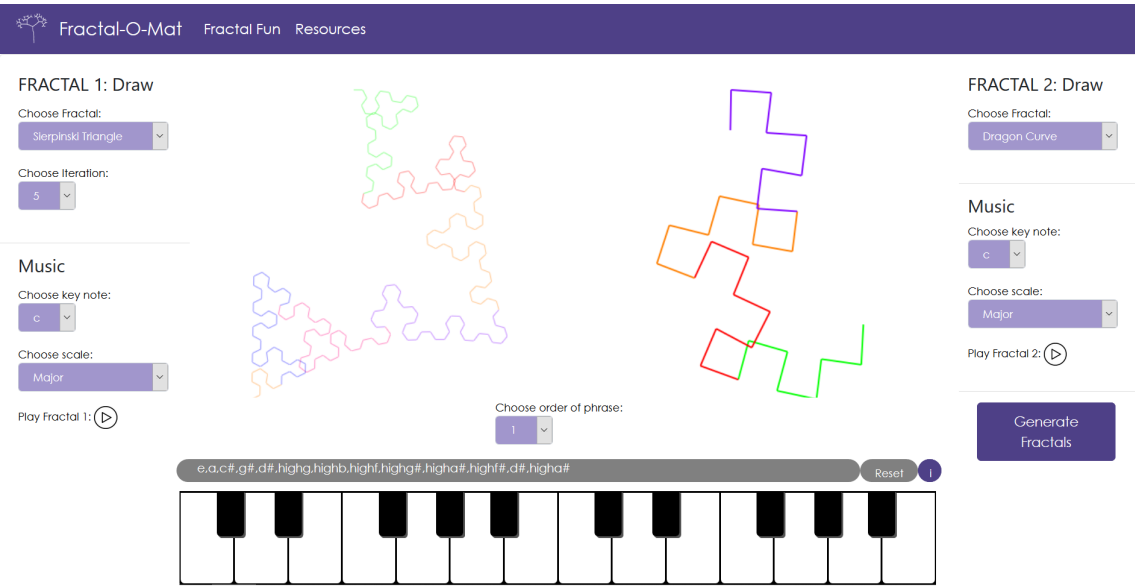


Figure 5.7: By playing different keys on the virtual piano before clicking the *Generate Fractals* button, the rendering of the fractals is changed according to table 4.2.

6

Discussion

The goal of making information about fractals, and how they relate to one another, accessible to a broad spectrum of users was accomplished through an interactive web application. However, the client-side code was written in JavaScript with no frameworks such as *ReactJS*, *Vue.js*, or *AngularJS*. Using a framework would have made it easier to develop and read the code, but learning new web frameworks was deemed too time-consuming for this project.

6.1 Generating strings in GF

In the original project description, one of the major parts of the project concerned translating between different fractals. As discussed in Section 1.3, fractal translation is something that can be done using GF. However, when it comes to the fractals used in this project, the translation that takes place is in reality nothing more than comparing same order phrases in different fractals. Since the axiom and the rules are built into the grammar and never change, the only thing about the first fractal that actually affects the rendering of the second fractal is the number of iterations the first fractal goes through. Compared to natural languages, the translation between fractals is not translation in the sense one is used to thinking about it as the first fractal hardly impacts the creation of the second at all.

Some of the fractals grow quickly – for the Gosper curve the growth per iteration n is proportional to 7^n – the strings become very long very fast. Since generating strings is very time-consuming, this raises the question about the usefulness of this method to the project. Instead the strings could be generated only once and stored to be used, as opposed to being generated anew each time a fractal is to be drawn. An even better solution would be to use a mathematical description of the fractals instead of strings.

6.2 Data structure

The original scope of the project did not include the Hilbert curve, since the Hilbert curve fractal is constructed in a slightly different manner from the other fractals in the project. The axiom of the L-system for the Hilbert curve is not a line, which caused the resulting string representing the fractal to be incompatible with the earlier versions of the data structure. The Dragon curve also caused some issues, since the fractal has a straight line segment connecting its phrases that is not part of

any phrase. The issue with the Dragon curve was resolved and the Hilbert curve could be included in the scope by reworking the data structure so it functioned as described in this section.

As mentioned in Section 4.4.1, a Layer of order n contains all the information in the Layer of order $n - 1$. For some fractals, additional information is added to the Layer of order n . The additional information is what binds together the ListSets in the Layer of order $n - 1$. The c function, described in Section 1.3.1, provides the information regarding how the ListSets are combined. For example, in Figure 6.1 the Dragon curve of iteration five is presented. The fractal consists of four phrases of order three, marked with lines in red, blue, green and pink, respectively. The dashed lines, corresponding to the graphical representation of the forward command, are not part of the phrases. The forward command is therefore part of the additional information that is added between the Layers. For fractal specific information regarding the binding of the ListSets, see Appendix B.

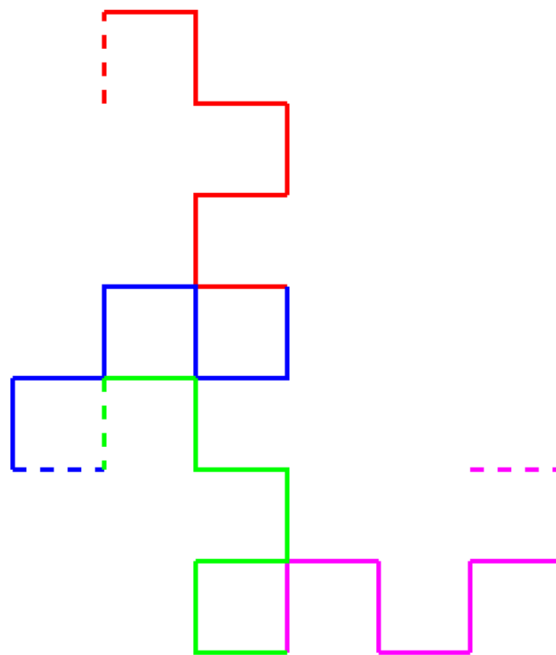


Figure 6.1: Iteration five of the Dragon curve. This fractal image consists of four phrases of order three, drawn with lines in red, blue, green and pink, respectively. The dashed lines between the phrases are the graphical representation of the additional information – in this case an extra forward command – that is added between the Layers.

6.3 Musical observations

Since the strings grow exponentially with the iterations and the music files are set not to exceed a certain duration, the music often gets many different musical parts (see Appendix A). For example, the musical interpretation of the Gosper curve quickly grows to almost 30 musical parts. This – in combination with each note having an assigned random note length – means that after a note has started playing, there is rarely more than a sixteenth-note before another note starts playing which makes the music quite blurry. Due to the blurriness, it is difficult to hear which fractal is being interpreted in a given piece of music.

Depending on the musical scale, it is also possible that the music gets stuck in one certain chord. For instance, if the musical scale is set to be the *A minor harmonic scale* the allowed tones are: A, B, C, D, E, F and G \sharp . The subset of tones containing only the tones B, D, F and G \sharp make up a G \sharp (or B, D or F) diminished seventh chord and since the distances between these tones are all multiples of 3 semitones, any interpretation of an image containing only angles between 90° and 119° will get stuck in this chord as soon as one of these tones is played.

6.4 Translating fractal fragments

An alternative method for comparing phrases in one fractal to the phrases in a different fractal (as opposed to the method described in Section 4.5.2) is by using the data structure to extract a phrase from one of the fractal strings. GF can subsequently be used as a tool for translating the phrase to the corresponding phrase in the other fractal string. This method was considered early on in the project, as it would fit the original project description slightly better than the current solution. For time constraint reasons, the work on this method was discontinued since the functionality of the product was prioritised over a product more dependent on GF. As mentioned in Section 1.3.2 and 4.5.2, different fractals have different number of sub-phrases per phrase. When comparing phrases between the fractals, it is not obvious how this problem should be solved and the solution presented in Section 4.5.2 is just one of many interpretations of how the phrases in the different fractals correspond to each other.

6.5 Taking the project further

There is a multitude of options for developing the project further. The graphical representation could be extended, for instance adding more complex rules for how the piano affects the rendering of the fractal images, more possibilities for user interactivity (such as zooming and rotating), etc. If the web application could register the rests (see Appendix A) in the user input from the virtual piano, said rests could have some impact on the rendering of the fractal images, e.g. gaps in the fractal images. The musical interpretation of a fractal could also be played

simultaneously as the fractal is being drawn, and the drawing could then be done in time with the music.

A further development could be letting the user provide their own L-systems for generating fractals. The data structure could also be adapted so that it will be compatible with even more fractals.

A

Appendix: Musical terms

The musical terms used in the report are meant to be understood as follows:

- **Pitch/tone** – The frequency of the sound wave.
- **Note** – A tone with a certain duration.
- **Key/scale** – A set of pre-defined tones on which the music is based. In a scale, the tones are usually ordered from the first to the seventh with the first tone being lower than the second, the second being lower than the third, etc. The seventh tone is lower than the double frequency of the first tone.
- **Rest** – Can be viewed as a silent note.
- **Glissando** – A note with continuously, strictly increasing or decreasing pitch.
- **Interval** – The distance between two tones. The interval can be measured in semitones.
- **Semitone** – The interval between two tones is a semitone if their frequencies are f_1 and $f_2 = \sqrt[12]{2}f_1$. This is the smallest interval used in western music and the smallest interval that is supported in the MIDIutil library (not counting notes of the same pitch).
- **Octave** – The interval between two tones is an octave if their frequencies are f_1 and $f_2 = 2f_1$.
- **Part** – A sequence of notes and rests. Multiple parts can be played simultaneously.
- **Note length** – Note lengths are defined only relative to each other: a half note is defined as half a whole note, a quarter note is defined as a quarter of a whole note, etc. The main note lengths are named after integer powers of $1/2$. A tempo can later be defined, usually by quarter notes per minute, to give the note lengths absolute values.

B

Appendix: GF-grammars in the project

The abstract grammar (see Section 1.3) adapted from [21] and used in the project follows:

```
-- "The L-system is a grammar formalism which is used to describe
-- graftals (recursive graphical objects). It is an interesting
-- coincidence that every L-System grammar could be redefined
-- as PMCFG grammar. This demo shows how to generate graftals
-- using GF. The output from every concrete syntax is a string
-- to be read in Python.
```

```
abstract Graftal = {
  flags startcat = N;
  cat N; S;
  fun z : N ;
    s : N -> N ;
    c : N -> S ;
}
```

In addition to the abstract grammar, a resource module with the three operations in table 4.1 were defined:

```
resource Operations = {
  oper
    F : Str = "F" ;
    R : Str = "r" ;
    L : Str = "l" ;
}
```

From the abstract grammar, five concrete grammars were developed that open the resource module. The grammars for the Sierpinski triangle, the Koch square curve (that later was modified into the Koch square snowflake) and the Dragon curve were all adapted from existing GF files at [21]. Conversely, the grammar for the Gosper curve and the Hilbert curve were implemented from scratch. The concrete grammars follow:

```

concrete Sierpinski of Graftal = open Operations in {
  lincat N = {a : Str; b : Str} ;
  lincat S = {s : Str} ;

  lin z = {a = F; b = F} ;
  lin s x = {a = x.b ++ L ++ x.a ++ L ++ x.b;
            b = x.a ++ R ++ x.b ++ R ++ x.a} ;
  lin c x = {s = "ang:60" ++ x.a} ;
}

```

```

concrete Koch of Graftal = open Operations in {
  lincat N = {f : Str} ;
  lincat S = {s : Str} ;

  lin z = {f = F} ;
  lin s x = {f = x.f ++ R ++ x.f ++ L ++ x.f ++ L ++ x.f ++ R ++ x.f} ;
  lin c x = {s = "ang:90" ++ x.f ++ R ++ x.f ++ R ++ x.f ++ R ++ x.f } ;
}

```

```

concrete Dragon of Graftal = open Operations in {
  lincat N = {a : Str; b : Str} ;
  lincat S = {s : Str} ;

  lin z = {a = ""; b = ""} ;
  lin s x = {a = x.a ++ L ++ x.b ++ F ++ L; b = R ++ F ++ x.a ++ R ++ x.b} ;
  lin c x = {s = "ang:90" ++ F ++ x.a } ;
}

```

```

concrete Gosper of Graftal = open Operations in {
  lincat N = {a : Str; b : Str} ;
  lincat S = {s : Str} ;

  lin z = {a = F; b = F} ;
  lin s x = {a = x.a ++ L ++ x.b ++ L ++ L ++ x.b ++ R ++ x.a ++ R ++ R
            ++ x.a ++ x.a ++ R ++ x.b ++ L;
            b = R ++ x.a ++ L ++ x.b ++ x.b ++ L ++ L ++ x.b ++ L ++
            x.a ++ R ++ R ++ x.a ++ R ++ x.b} ;
  lin c x = {s = "ang:60" ++ x.a} ;
}

```

```

concrete Hilbert of Graftal = open Operations in {

```



```
lincat N = {a : Str; b : Str} ;
lincat S = {s : Str} ;

lin z = {a = ""; b = ""} ;
lin s x = {a = L ++ x.b ++ F ++ R ++ x.a ++ F ++ x.a ++ R ++ F ++ x.b ++ L;
          b = R ++ x.a ++ F ++ L ++ x.b ++ F ++ x.b ++ L ++ F ++ x.a ++ R} ;
lin c x = {s = "ang:90" ++ x.a} ;
}
```

References

- [1] B. B. Mandelbrot, *The fractal geometry of nature*. New York, United States: W.H. Freeman and company, 1983, p. 15.
- [2] K. Falconer, *Fractal Geometry : Mathematical Foundations and Applications*, 3rd ed. University of St Andrews, United Kingdom: John Wiley & Sons, Incorporated, 2013, p. xxviii.
- [3] A. Ranta, *Grammatical Framework: Programming with Multilingual Grammars*, 1st ed. Stanford University, United States: CSLI Publications, 2011.
- [4] J. Hopcroft, J. Ullman, and R. Motwani, *Introduction to Automata Theory, Languages, and Computation*, 3rd ed. Pearson Education, 2014, [Online]. Available: <https://www.dawsonera.com/abstract/9781292056166> Accessed on: 2019/02/12.
- [5] P. Prusinkiewicz and J. Hanan, *Lindenmayer Systems, Fractals, and Plants*, ser. Lecture addendums in Biomathematics. New York, United States: Springer Science & Business Media, 1989, vol. 79, [Online]. Available: <https://books.google.se/books?id=J6fxBwAAQBAJ&printsec=frontcover#v=onepage&q&f=false>. Accessed on: 2019/02/10.
- [6] P. Prusinkiewicz and A. Lindenmayer, *The Algorithmic Beauty Of Plants*, 2nd ed. New York: Springer-Verlag, 1996, [Online]. Available: <http://algorithmicbotany.org/papers/#abop> Accessed on: 2019/02/04.
- [7] D. Shiffman, *The Nature of Code*. Cambridge MA: The Nature of Code, 2012, [Online]. Available: <https://natureofcode.com/book/>. Accessed on: 2019/02/04.
- [8] U. Teleman, S. Hellberg, and E. Andersson, *Svenska Akademiens grammatik*. Stockholm, Sweden: Svenska Akademien, 1999, [Online]. Available: https://svenska.se/SAG_Volym_1.pdf, Accessed on: 2019/05/09.
- [9] A. Moos, "Basic constituent structure analysis English sentence", 2014, [Electronic image]. Available: https://commons.wikimedia.org/wiki/File:Basic_constituent_structure_analysis_English_sentence.svg Accessed on: 2019/05/16.
- [10] R. Moog, "Midi: Musical instrument digital interface", *Journal of Audio Engineering Society*, vol. 34, no. 5, pp. 394–404, May 1986, [Online]. Available: <https://moogfoundation.org/wp-content/uploads/5267.pdf>, Accessed on: 2019/04/26.

- [11] R. Goldman, S. Schaefer, and T. Ju, “Turtle geometry in computer graphics and computer aided design”, *Computer-Aided Design*, vol. 36, pp. 1471–1482, Dec. 2004. DOI: <https://doi.org/10.1016/j.cad.2003.10.005>, [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0010448504000521> Accessed on: 2019/05/13.
- [12] S. Collin, *Setting Up a Web Server*. Boston, United States: Digital Press, 1997, [Online]. Available: <https://library-books24x7-com.proxy.lib.chalmers.se/assetviewer.aspx?bookid=549&chunkid=943466268&rowid=9> Accessed on: 2019/05/07.
- [13] C. Larman, *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development*, 3rd ed. Upper Saddle River, New Jersey, USA: Prentice Hall, 2004, p. 471, [Online]. Available: <https://www.utdallas.edu/~chung/SP/applying-uml-and-patterns.pdf> Accessed on: 2019/05/05.
- [14] I. Fette and A. Melnikov, *Rfc6455: The websocket protocol*, (2011) [Online]. Available: <https://tools.ietf.org/html/rfc6455>, Accessed on: 2019/04/29.
- [15] *The json data interchange syntax*, ECMA-404, (2017)[Online]. Available: <https://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf>, Accessed on: 2019/04/09.
- [16] W. Beyer, “Partial view of the Mandelbrot set.” 2005, [Electronic image]. Available: https://commons.wikimedia.org/wiki/File:Mandel_zoom_11_satellite_double_spiral.jpg Accessed on: 2019/02/13.
- [17] A. Bergsten et al. (2019), *Prototype*, [Online]. Available: <https://github.com/Gurr1/Grammars-for-music-poetry-fractals/tree/prototype> Accessed on: 2019/02/15.
- [18] E. Jarrell (2018), *Create a piano app with javascript*, [Online]. Available: <https://hackernoon.com/create-a-piano-app-with-javascript-97dbad1ff28c> Accessed on: 2019/02/14.
- [19] P. Prusinkiewicz, “Score generation with lsystems”, in *Proceedings of the 1986 International Computer Music Conference*, 1986, pp. 455–457, [Online]. Available: <http://algorithmicbotany.org/papers/score.icmc86.pdf> Accessed on: 2019/03/04.
- [20] S. Manousakis, “Musical l-systems”, Master’s thesis, Sonology, The Royal Conservatory, The Hague, Netherlands, 2006, [Online]. Available: http://www.modularbrains.net/support/SteliosManousakis-Musical_L-systems.pdf Accessed on: 2019/02/15.
- [21] K. Angelov, “Gf runtime system”, Licentiate Thesis, Department of Computer Science, Engineering, Chalmers University of Technology, and Gothenburg University, Gothenburg, Sweden, 2009, [Online]. Available: <http://www.cse.chalmers.se/~krasimir/lic-thesis.pdf> Accessed on: 2019/04/25.
- [22] P. Coles (2013), *Html5 javascript piano*, [Online]. Available: <https://github.com/mrcoles/javascript-piano> Accessed on: 2019/04/16.

- [23] A. Bergsten et al. (2019), *Fractal-o-mat*, [Online]. Available: <https://github.com/Gurr1/Grammars-for-music-poetry-fractals> Accessed on: 2019/05/14.

Pictures only

- [9] A. Moos, "Basic constituent structure analysis English sentence", 2014, [Electronic image]. Available: https://commons.wikimedia.org/wiki/File:Basic_constituent_structure_analysis_English_sentence.svg Accessed on: 2019/05/16.
- [16] W. Beyer, "Partial view of the Mandelbrot set." 2005, [Electronic image]. Available: https://commons.wikimedia.org/wiki/File:Mandel_zoom_11_satellite_double_spiral.jpg Accessed on: 2019/02/13.

Code only

- [17] A. Bergsten et al. (2019), *Prototype*, [Online]. Available: <https://github.com/Gurr1/Grammars-for-music-poetry-fractals/tree/prototype> Accessed on: 2019/02/15.
- [18] E. Jarrell (2018), *Create a piano app with javascript*, [Online]. Available: <https://hackernoon.com/create-a-piano-app-with-javascript-97dbad1ff28c> Accessed on: 2019/02/14.
- [22] P. Coles (2013), *Html5 javascript piano*, [Online]. Available: <https://github.com/mrcoles/javascript-piano> Accessed on: 2019/04/16.
- [23] A. Bergsten et al. (2019), *Fractal-o-mat*, [Online]. Available: <https://github.com/Gurr1/Grammars-for-music-poetry-fractals> Accessed on: 2019/05/14.