MINISTRY OF EDUCATION, CULTURE AND RESEARCH OF REPUBLIC OF MOLDOVA
TECHNICAL UNIVERSITY OF MOLDOVA
FACULTY OF COMPUTERS, INFORMATICS AND MICROELECTRONICS
DEPARTMENT OF SOFTWARE ENGINEERING AND AUTOMATICS

LaTeX: A DSL for building fractals.

Project report

Mentor: prof., Catruc Mariana
Students: Iordan Liviu, FAF-223
Pereteatcu Arina, FAF-223
Țapu Pavel, FAF-223
Prodius Cristian, FAF-223

Chișinău, 2024

Abstract

This report introduces a Domain Specific Language (DSL) tailored for creating fractals developed by the students: Pereteatcu Arina, FAF-223; Țapu Pavel, FAF-223; Iordan Liviu, FAF-223, from Technical University of Moldova, which will be addressing the complexities associated with manual coding in fractal geometry. The purpose of the project is to provide a specialized language that simplifies the process of generating fractal patterns, catering to both novice and experienced users. The main objectives include designing an intuitive syntax and semantics for the DSL, implementing functionalities for defining fractal shapes and parameters, and validating the correctness and effectiveness of the DSL through testing and user feedback. Methodologies employed in the project include domain analysis of fractal geometry, DSL design principles, and iterative development cycles. The obtained results showcase the effectiveness of the DSL in generating intricate fractal patterns for artistic, educational, and scientific purposes. Potential applications of the DSL include artistic expression, educational exploration of fractal geometry, and integration into scientific simulations. Through this project, we aim to contribute to the field of computational geometry and provide a valuable tool for fractal enthusiasts and researchers.

Keywords: Fractals, Domain Specific Language (DSL), Computational Geometry, Syntax and Semantics

Content

Introduction

Fractals, those endlessly intricate and visually captivating geometric patterns, have long been a source of fascination and inquiry across disciplines ranging from mathematics and physics to art and computer science. Defined by their self-similar structure, fractals possess a unique allure that transcends traditional notions of shape and dimensionality. From the mesmerizing complexity of the Mandelbrot set to the elegant simplicity of the Sierpinski triangle, fractals embody a profound mathematical beauty that continues to inspire exploration and creativity.

In our project, we embark on a comprehensive exploration of the domain of fractal geometry, with a specific focus on addressing the challenges inherent in the creation and manipulation of these complex structures. At the core of our endeavor lies the development of a specialized Domain Specific Language (DSL) tailored specifically for building fractals. This innovative language, designed with meticulous attention to the nuances of fractal design, aims to democratize access to fractal generation while offering a powerful and expressive toolset for users of all backgrounds.

The motivation for choosing this topic is deeply rooted in our collective fascination with fractals and their potential applications across a wide range of domains. Fractals, with their innate beauty and mathematical richness, have found utility in fields as diverse as computer graphics, digital art, natural sciences, and even finance. However, despite their ubiquity, the creation of fractals often involves complex mathematical calculations and intricate algorithms, presenting a significant barrier to entry for many enthusiasts and researchers.

Our project seeks to address this challenge by introducing a dedicated DSL for building fractals, thus democratizing access to fractal generation and empowering users to explore and create complex patterns with ease. The degree of novelty and relevance of our topic lies in its focus on addressing a specific problem within the domain of fractal geometry – the lack of a specialized language tailored to the needs of fractal creators.

The overarching objectives of our project encompass not only the design and implementation of the DSL itself but also a comprehensive exploration of the methodologies and techniques employed in its development. Through rigorous domain analysis, we aim to gain a deep understanding of the intricacies of fractal geometry, identifying key concepts, patterns, and challenges that inform the design of our DSL.

Furthermore, our project aims to contribute to the growing field of computational geometry by providing a practical and versatile tool for fractal exploration and creation. By leveraging cutting-edge research in language design, software engineering, and computational mathematics, we seek to push the boundaries of what is possible in the realm of fractal art and science.

The significance of our project lies not only in its technical innovations but also in its potential societal impact. By democratizing access to fractal generation tools, we hope to inspire a new generation of artists, educators, and researchers to explore the beauty and complexity of fractal geometry. Furthermore, the development of a specialized DSL for fractal generation has implications beyond the realm of art and aesthetics. Fractals have found applications in diverse fields such as data compression, signal processing, and even cryptography. By providing a user-friendly platform for fractal exploration, we aim to unlock new avenues for research and innovation across multiple disciplines.

Our project represents a multidisciplinary endeavor that draws upon insights from mathematics, computer science, and art. Through collaboration and interdisciplinary exchange, we aim to develop a comprehensive understanding of fractal geometry and its applications. By leveraging cutting-edge research in language design, software engineering, and computational mathematics, we strive to push the boundaries of what is possible in the realm of fractal art and science.

In the subsequent chapters of this report, we will delve into the details of our project plan, beginning with the design of the grammar for the DSL. This foundational step lays the groundwork for the subsequent phases of our project, including the implementation of lexical analyzers and symbol tables, the validation of our DSL through testing and user feedback, and our submission of a scientific paper on the state of the art in fractal geometry. Through these efforts, we aim to contribute to the advancement of computational geometry and provide a valuable tool for fractal enthusiasts and researchers alike.

1 st Midterm

Before delving into the intricacies of our project, it's essential to lay a solid foundation by thoroughly understanding the domain in which we operate and identifying the specific problem we aim to address. In this chapter, we will transition from the initial problem statement outlined in the project proposal to a comprehensive problem description that encapsulates the complexities and nuances of the issue at hand.

Domain analysis involves examining the context, scope, and constraints of the problem space. It entails understanding the various factors, stakeholders, and existing systems or processes relevant to the domain in which our project operates. By conducting a thorough domain analysis, we gain valuable insights into the landscape within which our solution will be situated, allowing us to tailor our approach accordingly.

Following domain analysis, we will proceed to identify and define the specific problem that our Domain-Specific Language (DSL) aims to solve. This entails dissecting the overarching problem statement into its constituent components, elucidating the underlying challenges, and delineating the desired outcomes. By clearly defining the problem, we establish a solid framework upon which to build our DSL solution, ensuring alignment with the needs and objectives of stakeholders.

In the subsequent sections, we will delve into the intricacies of domain analysis, exploring the key domains relevant to our project, and conduct a detailed problem identification process to refine our understanding of the issues we seek to address. Through this comprehensive analysis, we aim to lay the groundwork for the development of an effective and impactful DSL solution tailored to the specific needs of the domain.

1.1 Domain Analysis

Fractals, emerging at the nexus of mathematics and art, transcend conventional boundaries to offer profound insights into the structure of natural phenomena and human artifacts. While often appreciated for their aesthetic appeal, fractals serve as powerful mathematical tools for describing, measuring, and predicting complex systems in various domains of science. This article explores the genesis of four renowned fractals and elucidates their key properties, which underpin their utility across diverse scientific disciplines.

Fractals captivate the imagination with their captivating imagery, bridging the realms of mathematics and art to showcase the inherent beauty and complexity of natural forms. Beyond mere visual allure, fractal geometry serves as a fundamental framework for understanding the intricate patterns observed in nature, from coastlines to mountain ranges. The origins of fractal geometry can be traced back to the meticulous efforts of British cartographers grappling with the elusive nature of coastline measurements, laying the groundwork for a revolutionary approach to geometric analysis.

1.1.1 Fractals' Properties

Fractals exhibit two key properties that set them apart from classical geometric shapes: self-similarity and non-integer dimension.

Self-Similarity

Self-similarity is a defining characteristic of fractals, manifesting as the repetition of similar patterns at different scales within a fractal structure. This property means that when you zoom in or out on a fractal, you observe the same or similar shapes recurring at progressively smaller or larger magnifications. This recursive nature gives fractals their intricate and infinitely detailed appearance. To illustrate self-similarity, consider the Koch Snowflake. At each iteration of the fractal construction process, smaller triangular patterns are added to the original triangle's sides, creating a self-similar structure. No matter how much you zoom in on any part of the Koch Snowflake, you'll always see smaller triangles nested within larger ones, repeating the overall shape ad infinitum.

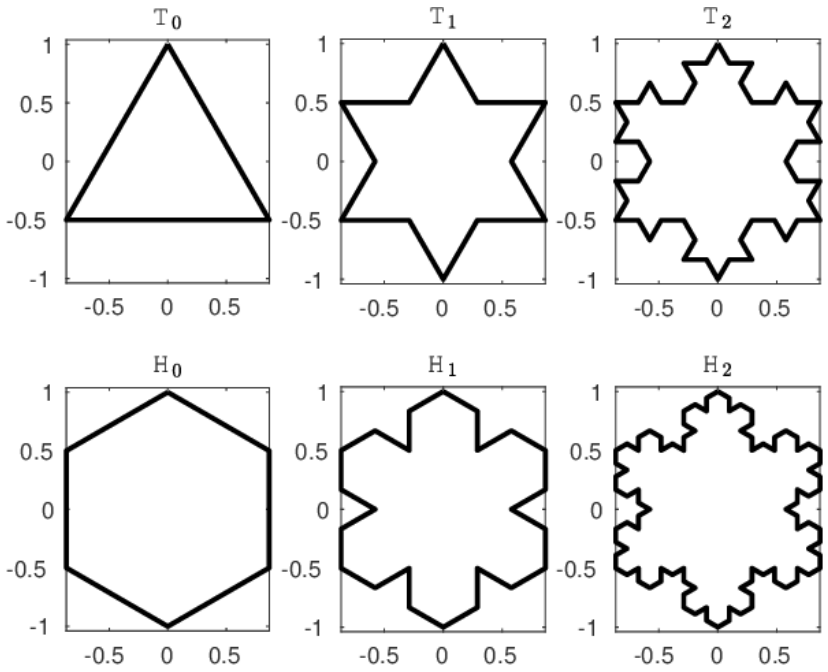

Figure 1.1.1 - Koch Snowflakes on different bounds.

Non-Integer Dimension

Fractals challenge traditional geometric notions of dimensionality by exhibiting non-integer dimensions, unlike the familiar integer dimensions of Euclidean geometry (0D for points, 1D for lines, 2D for shapes like squares, and 3D for solids like cubes). Fractal dimensionality falls between these integer values, reflecting the intricate and often irregular nature of fractal shapes. To understand non-integer dimensionality, consider the coastline paradox. Traditional geometry suggests that coastlines should have a finite length, but as you measure them with increasingly finer resolution, their apparent length increases indefinitely. This

phenomenon arises from the fractal nature of coastlines, which exhibit intricate detail at all scales. Fractal dimensionality provides a way to quantify this complexity, expressing it as a non-integer value between the dimensions of a line (1D) and a plane (2D).

In summary, the properties of self-similarity and non-integer dimensionality are fundamental to the definition and characterization of fractals. They imbue fractals with their unique visual appeal and underpin their utility across various scientific domains, from mathematics and physics to biology and computer science.[1]

1.1.2 Types of fractals

In this report, two widely recognized categories of fractals will be introduced: complex number fractals and Iterated Function System (IFS) fractals, among numerous others. [1]

Complex Number Fractals

Complex number fractals, pioneered by Gaston Maurice Julia and popularized by Benoit Mandelbrot, unveil mesmerizing patterns on the complex plane. The iconic Mandelbrot set, generated through iterative algorithms, delineates regions of convergence and divergence, offering a visual manifestation of complex dynamics.

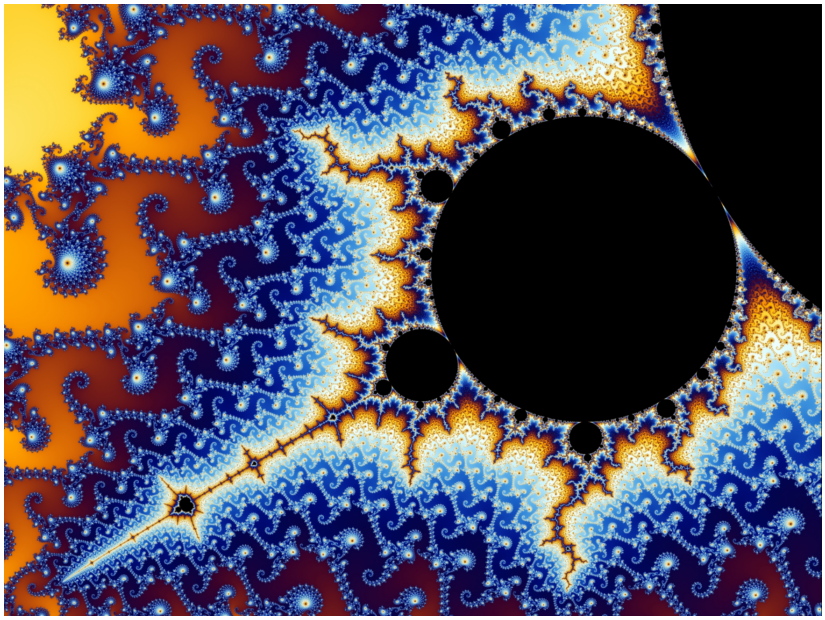

Figure 1.1.2 - Mandelbrot Set.

Julia sets, intimately linked with the Mandelbrot set, epitomize the intricate interplay between mathematical rigor and aesthetic elegance, revealing the boundless creativity inherent in fractal geometry.

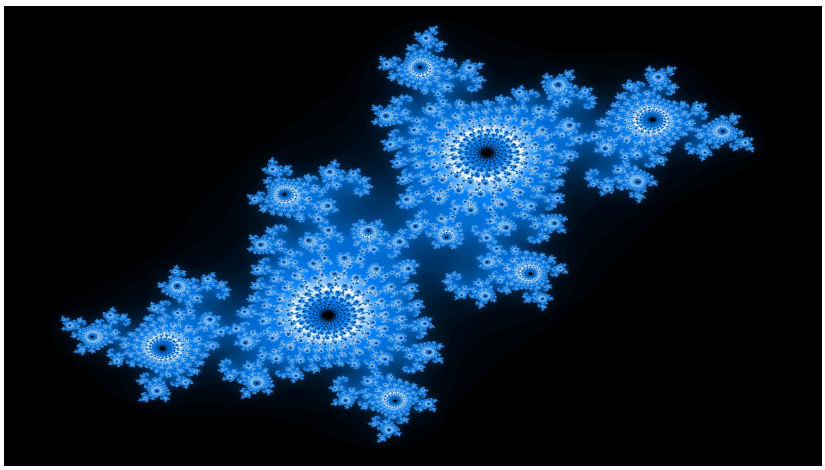

Figure 1.1.3 - Julia Set.

Iterated Function System Fractals

Iterated Function System (IFS) fractals are generated through basic plane transformations including scaling, translation, and rotation. The process of creating an IFS fractal involves the following steps:

- Defining a set of plane transformations.
- Drawing an initial pattern on the plane, which can be any shape.
- Applying the defined transformations to the initial pattern.
- Iteratively applying the transformations to the resulting image, combining it with the original pattern each time.
- Repeating this process as many times as desired, theoretically infinite times.

Some of the most well-known IFS fractals include the Sierpinski Triangle and the Koch Snowflake.

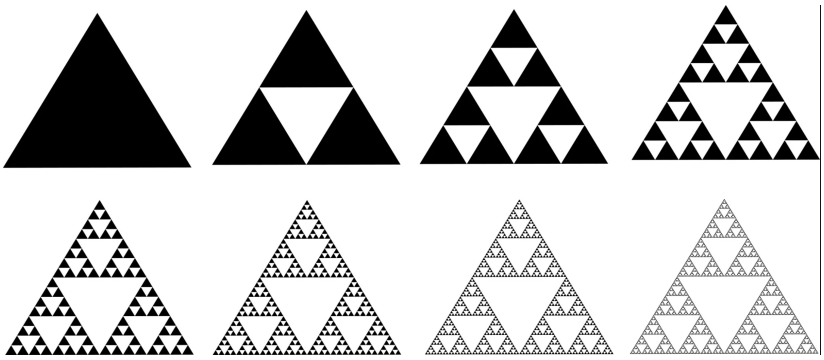

Figure 1.1.4 - Sierpinski triangles with different depths.

1.1.3 Fractals' Applications

Fractal geometry has permeated various scientific domains, including astrophysics, biological sciences, computer graphics,[1] as well as aspects of nature and art[2], establishing itself as a vital technique in each field.

Fractals in astrophysics

The exact number of stars in our skies remains a mystery, prompting curiosity about their origins and placement in the vast Universe. Astrophysicists suggest that the fractal nature of interstellar gas holds the key to this enigma. Fractal distributions, akin to the intricate patterns seen in smoke trails or billowy clouds, exist in both terrestrial and celestial realms. Turbulence shapes these formations, imbuing them with irregular yet repetitive patterns that defy conventional geometric descriptions.

Fractals in the Biological Sciences

Traditionally, biologists depicted natural phenomena using Euclidean representations, such as sine waves for heartbeats and cones for conifer trees. However, the advent of fractal geometry revolutionized this approach. Biological systems exhibit intricate substructures repeated across multiple levels, akin to fractal patterns. Chromosomes, for instance, display a tree-like architecture with numerous 'mini-chromosomes,' suggesting a fractal nature. Some biologists argue that the fractal properties of DNA sequences could illuminate evolutionary relationships in animals, hinting at a future where fractal geometry aids in creating comprehensive models of natural patterns and processes.

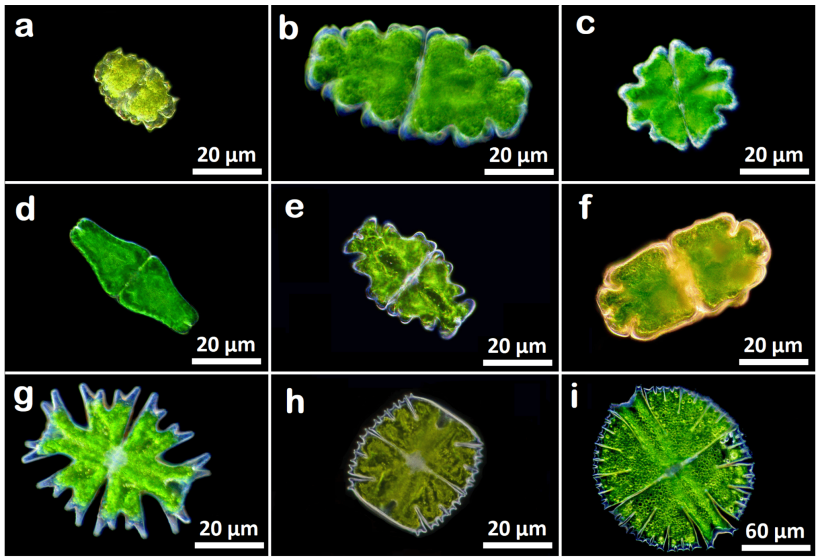

Figure 1.1.5 - DIC microscopy images of unicellular algae of the class Zygnematophyceae. (a) Euastrum bidentatum, (b) Euastrum oblongum, (c) Euastrum verrucosum, (d) Euastrum ansatum, (e) Euastrum humerosum, (f) Euastrum crassum, (g) Micrasterias americana, (h) Micrasterias truncata and (i) Micrasterias rotata.

Fractals in computer graphics

Fractals play a pivotal role in computer science, particularly in image compression. Fractal algorithms compress graphics files to a fraction of their original size, facilitating efficient storage and transmission. Computer graphic artists utilize fractal forms to craft intricate textured landscapes and models, enabling the creation of realistic scenes. Fractals are integral to special effects in Hollywood movies and

television advertisements, lending authenticity to lunar landscapes, mountain ranges, and coastlines. From the iconic ”Genesis effect” in ”Star Trek II - The Wrath of Khan” to the depiction of the dreaded ”Death Star” in ”Return of the Jedi,” fractals bring mathematical precision to modeling natural and artificial environments.

Figure 1.1.6 - Death star II

Fractals in nature

The discovery of fractals marked a significant breakthrough in our understanding of nature. Many natural forms, previously perceived as irregular, have been revealed to exhibit fractal properties. From the intricate patterns of river deltas and fern leaves to the geometric structures of Romanesco cauliflower and snowflakes, fractals permeate various aspects of our natural world. Even familiar elements like mountains, coastlines, and clouds exhibit fractal characteristics, challenging traditional geometric perceptions. Interestingly, Professor Shaun Lovejoy from McGill University predicts that fractals will revolutionize weather prediction, as meteorological phenomena demonstrate fractal patterns. Furthermore, elements within the human body, from proteins to neurons, exhibit fractal shapes, highlighting the ubiquity of fractals in nature.

Figure 1.1.7 - Fractals in nature

Fractals in art

While classical geometric figures hold conceptual beauty, they often lack the visual diversity found in natural phenomena. Fractal geometry has deep roots in ancient and traditional African architecture and art, where circular structures are nested within larger circles and triangles are nested within proportionally larger triangles. In the 20^{Th} century, Surrealist artists, particularly in the 1920s, embraced fractal forms through techniques like mental automatism, resulting in artworks rich with fractal accents. Max Ernst, among others, incorporated fractal patterns into his works, notably utilizing decalomania to create intricate fractal designs. In 1999, a breakthrough occurred in the art world when scientists conducted a fractal analysis of Jackson Pollock's works. The research revealed that Pollock's art intricately reproduces mathematical figures, captivating viewers' attention and even reducing stress levels, underscoring the profound influence of fractals in art.

Figure 1.1.8 - ”The Great Wave off Kanagawa” by Hokusai

1.1.4 Fractal Analysis and Data Science Relation

The relationship between fractal analysis and data science is multifaceted, with both fields complementing each other in various aspects:

- Understanding Complex Data: Fractal analysis offers a framework for comprehending intricate data structures and patterns. Data science deals with vast and intricate datasets, and fractal analysis aids in identifying self-similarity, scaling properties, and patterns within the data. By employing fractal analysis methods, data scientists can uncover the underlying structure of the data.
- Feature Extraction: Fractal analysis empowers data scientists to extract meaningful features from datasets. Fractal dimensions, for instance, can quantify the complexity or irregularity of patterns in data, serving as features for further analysis. These features enhance the predictive capabilities of machine learning models and unveil hidden relationships or anomalies in the data.
- Data Visualization: Fractal analysis contributes to visualizing and representing complex datasets in intuitive and informative ways. Data visualization is pivotal in comprehending data and conveying insights effectively. Fractals, with their visually captivating and self-replicating patterns, offer a unique and visually rich representation of data.
- Time Series Analysis: Fractal analysis techniques, such as fractal dimensions and Hurst exponent, prove beneficial in analyzing time series data. Data scientists often grapple with time-dependent data, like stock prices or weather data. Fractal analysis aids in identifying long-term dependencies, trends, or self-similar patterns in such data, facilitating forecasting, anomaly detection, and time series modeling.
- Dimensionality Reduction: Data science frequently deals with datasets of high dimensions, posing challenges in analysis and insight extraction. Fractal analysis techniques aid in reducing data dimensionality by identifying relevant features and minimizing noise or redundancy. This fosters more

efficient and accurate data analysis and modeling.

To further delve into the relationship between fractals and data science, exploring specific examples and detailed explanations, one may refer to academic articles or books that delve into the intersection of these fields. Publications by recognized researchers in fractal analysis and data science often provide in-depth insights and illustrative examples to elucidate the connection between fractals and data science.[3]

1.2 Benefits of a Domain-Specific Language (DSL) in the Fractal Geometry Domain

- Tailored Functionality A DSL designed specifically for building fractals would offer tailored functionality that caters to the unique requirements of fractal creation. By focusing solely on fractals, the DSL can provide specialized tools, functions, and syntax optimized for generating, manipulating, and visualizing fractal patterns.
- Simplified Fractal Creation A DSL streamlines the process of fractal creation by abstracting away complex mathematical concepts and algorithms. Users, even those without extensive mathematical or programming backgrounds, can leverage the DSL's intuitive syntax to generate intricate fractal designs. This simplification lowers the barrier to entry, allowing a wider range of enthusiasts, artists, educators, and researchers to explore and create fractals.
- Increased Productivity The specialized nature of a fractal DSL enables users to work more efficiently. With built-in functions and libraries tailored for fractal operations, users can quickly prototype, iterate, and refine their fractal designs. This increased productivity empowers users to focus on the creative aspects of fractal art or research, rather than getting bogged down by technical implementation details.
- Domain-Specific Abstractions A fractal DSL abstracts complex fractal concepts into higher-level abstractions, making it easier for users to express their ideas concisely. By providing domain-specific constructs for defining fractal structures, transformations, and rendering parameters, the DSL fosters clearer and more expressive code that accurately represents the user's intentions.
- Seamless Integration A well-designed fractal DSL seamlessly integrates with existing tools and workflows, enhancing interoperability and compatibility. Users can incorporate fractal generation directly into their projects, whether they are working on digital art, scientific simulations, educational materials, or software applications. This seamless integration fosters collaboration and facilitates the incorporation of fractal elements across various domains and disciplines.

Overall, a DSL tailored for building fractals offers numerous benefits, including simplified fractal creation, increased productivity, and enhanced expressiveness, making the exploration and creation of fractals more accessible and enjoyable for a broader audience.

1.3 Problem Description

These mesmerizing structures, known as fractals, exhibit fascinating properties such as self-similarity at varying scales, offering profound insights into the underlying mechanisms governing natural phenomena

and artistic expressions alike. However, despite their ubiquitous presence, the creation and exploration of fractals present formidable challenges and limitations when approached through traditional programming languages or existing fractal generation tools.

Manual coding of fractals using languages like Python, Java, or C++ demands a deep understanding of complex mathematical concepts and algorithms, deterring individuals without extensive expertise from engaging meaningfully with fractal geometry. The technical intricacies involved in implementing fractal algorithms, such as the Mandelbrot set or the Julia set, pose significant barriers for non-programmers and enthusiasts seeking to explore fractal patterns and phenomena.

Existing fractal generation tools and software, while offering functionality in specific areas like L-Systems, visualization, or fractal frameworks, suffer from several shortcomings that impede effective fractal exploration and creation. These tools often lack the comprehensive functionalities and flexibility required to experiment with diverse algorithms, parameters, and combinations, restricting users' ability to visualize and express intricate fractal patterns fully.

Moreover, the current landscape of fractal generation tools is fragmented, with scattered tools and approaches making it challenging for beginners and non-programmers to navigate and explore fractal geometry effectively. Users are often confronted with steep learning curves, complex interfaces, and limited documentation, further exacerbating the accessibility and usability issues associated with existing tools.

Identified Challenges and Complexities include:

- Limited Accessibility and Usability: The technical complexities inherent in manual fractal coding using traditional programming languages deter non-programmers and enthusiasts from engaging with fractal geometry. Existing tools often require coding knowledge or involve navigating complex interfaces, posing significant barriers for users across different skill levels.
- Restricted Flexibility and Customization: Many existing tools constrain users to predefined functionalities and parameter sets, inhibiting exploration of intricate variations and personal artistic expression. The lack of flexibility and customization options limits users' ability to experiment with diverse algorithms, parameters, and combinations freely.
- Fragmented Landscape: The current landscape of fractal generation tools is fragmented, with scattered tools and approaches making it challenging for users to navigate and explore fractal geometry effectively. The absence of a centralized and accessible platform hinders beginners and non-programmers from engaging meaningfully with fractal exploration.

2 Problem Analysis

Fractals, intricate mathematical structures characterized by self-similarity at varying scales, have fascinated scientists, artists, educators, and enthusiasts for decades. However, traditional programming languages and existing fractal generation tools present significant challenges for users seeking to explore and create fractal patterns:

- Steep Learning Curve: Mastering traditional programming languages and existing fractal generation tools demands deep mathematical knowledge and coding expertise, excluding individuals without advanced technical skills from meaningful engagement with fractal geometry.
- Limited Flexibility: Current tools often impose restrictions on users, limiting their ability to explore diverse algorithms, parameters, and combinations for creating intricate fractal patterns. This lack of flexibility hinders artistic expression and scientific exploration within the realm of fractal geometry.
- Accessibility Barrier: Non-programmers and beginners face substantial obstacles in navigating complex interfaces and understanding intricate mathematical concepts, preventing them from engaging effectively with fractal exploration and visualization.

2.0.1 Project Vision

Our project aims to address these limitations by creating a user-friendly Fractal Domain-Specific Language (DSL) that offers intuitive, built-in functions and libraries, abstracting away complex algorithms and mathematical concepts. By enabling high flexibility for exploring diverse algorithms, parameters, and combinations, the Fractal DSL will lower the technical barrier, allowing wider audiences to effortlessly engage with fractal creation and exploration.

2.0.2 Target Audience:

Artists seeking unique and expressive visuals through fractal patterns.

- Educators aiming to create engaging and interactive learning experiences with fractals in classrooms and educational settings.
- Researchers exploring new scientific applications of fractal geometry in fields such as physics, biology, computer science, and nature.
- Enthusiasts fascinated by the beauty and complexity of fractals, seeking to explore and create fractal patterns for personal enjoyment and exploration.

2.0.3 Key Questions for Analysis:

- What are the specific pain points and needs of each target audience regarding existing fractal generation tools?
- How would a Fractal DSL address these pain points and fulfill the unmet needs of users?

- What technical and design considerations are crucial for creating a user-friendly and effective Fractal DSL?
- What potential challenges and limitations might arise in developing and implementing a Fractal DSL, and how can they be addressed?
- How could a successful Fractal DSL impact the fields of art, science, education, and the public's understanding of fractal geometry?

2.0.4 Conclusion

The creation of a user-friendly Fractal DSL presents an opportunity to democratize fractal exploration and creation, empowering users across different skill levels to engage meaningfully with fractal geometry. By addressing the challenges associated with traditional programming languages and existing fractal generation tools, the Fractal DSL has the potential to revolutionize artistic expression, scientific exploration, and educational advancement within the realm of fractal geometry.

2.1 Problem Statement

In the realm of fractal geometry, enthusiasts, artists, educators, and researchers face significant challenges in creating and exploring fractal patterns using traditional programming languages and existing fractal generation tools. These challenges include steep learning curves, limited flexibility, and accessibility barriers, which hinder meaningful engagement with fractal geometry for a diverse range of users. As a result, there is a pressing need for a user-friendly Domain-Specific Language (DSL) tailored specifically for fractal generation. This DSL aims to address the limitations of existing tools by offering intuitive functions, high flexibility, and a lower technical barrier, empowering users to effortlessly create, explore, and visualize intricate fractal patterns with unprecedented ease and customization.

3 Solution Analysis

3.1 Solution Proposal

Fractals, with their intricate patterns and self-similarity across scales, have captivated mathematicians and artists alike for decades. However, creating fractals traditionally demands a deep understanding of mathematical concepts and algorithms, limiting accessibility. Our proposed solution, a domain-specific language (DSL) tailored for fractal generation, aims to democratize fractal creation by offering an intuitive platform for users of all expertise levels.

3.1.1 Key Features of the Fractal DSL

- Simplified Syntax: Our DSL offers a streamlined syntax with predefined statements, reducing the learning curve for users.
- Comprehensive Grammar: The language incorporates a comprehensive grammar, facilitating clear expression of fractal algorithms and designs.
- Versatility: Our DSL supports various fractal shapes, colors, rotations, and transformations, enabling diverse creations.
- Ease of Use: With its user-friendly interface, Our DSL empowers both experts and novices to generate stunning fractals effortlessly.
- Customization: Users can control parameters such as iterations, size, and colors, allowing for personalized fractal designs.

3.1.2 Implementation Approach

Our DSL will be implemented as an internal DSL, embedded within a host language for seamless integration and execution. Leveraging principles of language design and parsing techniques, we will develop a robust compiler/interpreter for our DSL, ensuring efficient translation of user code into executable fractal images. The implementation will prioritize performance and scalability to handle complex fractal computations efficiently.

3.1.3 Impact and Benefits

- Accessibility: Our DSL lowers the entry barrier for fractal creation, empowering a broader audience to explore and appreciate fractal geometry.
- Educational Tool: Our DSL serves as an educational resource, fostering understanding of mathematical concepts and algorithms through practical application.
- Artistic Expression: By democratizing fractal generation, our DSL promotes artistic expression, facilitating the creation of captivating digital art and animations.
- Research Advancement: Our DSL accelerates research in fractal geometry by providing a versatile

platform for experimentation and exploration.

- Cross-Disciplinary Applications: The versatility of our DSL extends its utility beyond mathematics and computer science, finding applications in fields such as art, design, and education.

In conclusion, the development of a comprehensive Fractal DSL holds the potential to revolutionize artistic expression, scientific exploration, and educational advancement within the realm of fractal geometry, empowering users across different skill levels to engage meaningfully with this mesmerizing world of mathematical beauty.

3.2 Solution Statement

- Key Components: Our DSL comprises a set of predefined statements including size, color, angle, iterations, shape, move, scale, rotate, mirror, draw, and save.
- Expected Impact: Our DSL is anticipated to democratize fractal generation, making it accessible to users with varying levels of expertise and fostering innovation in fractal art and mathematics.
- Lexical Considerations: Our DSL's syntax adheres to formal grammar rules, ensuring clarity and consistency in code expression. Tokens encompass alphanumeric characters, arithmetic operators, and punctuation symbols.

Our DSL represents a paradigm shift in fractal generation, offering a powerful yet accessible tool for enthusiasts, artists, educators, and researchers alike. Through its intuitive design and versatile features, our DSL stands poised to redefine the landscape of fractal exploration and creativity.

4 Grammar

The grammar provided serves as the foundation for defining the syntax of our DSL, a domain-specific language tailored for generating fractal images. Comprising non-terminal symbols, terminal symbols, production rules, and a designated start symbol, this grammar delineates the permissible structure and syntax of our DSL programs. Non-terminal symbols represent placeholders for various components of our DSL code, while terminal symbols encompass keywords, literals, and alphanumeric characters fundamental to the language. Production rules dictate how non-terminal symbols can be replaced or expanded, defining the syntactic rules for constructing valid our DSL programs. With a clear and concise specification, this grammar ensures consistency and correctness in our DSL code expression, facilitating the interpretation and execution of our DSL programs.

G = (VN, VT, P, S)

VN - finite set of non-terminal symbols.

VT - finite set of terminal symbols.

P - finite production rules.

S - start symbol.

S = { <program> }

VN = { <program>, <statement>, <size_statement>, <color_statement>, <angle_statement>, <
<rotate_statement>, <mirror_statement>, <axis>, <draw_statement>, <save_statement>, <fil

VT = { repeat, times, start, with, shape, circle, square, triangle, polygon, color, bac

P = {   <program> → <statement> | <statement> <program>
<statement> → <size_statement> | <color_statement> | <angle_statement> | <iterations_sta
<size_statement> → size <value>
<color_statement> → color <value>
<angle_statement> → angle <value>
<iterations_statement> → iterations <value>
<shape_statement> → shape <shape>
<move_statement> → move <value> <value>
<scale_statement> → scale <value>

<rotate_statement> → rotate <value>

<mirror_statement> → mirror <axis>

<axis> → x | y

<draw_statement> → draw

<save_statement> → save <filename>

<filename> → <string>

<shape> → circle | square | triangle | polygon

<value> → <digit> | <digit> <value> | <string>

<digit> → 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<string> → <char> | <char> <string>

<char> → [A-Z] | [a-z] | [0-9] | = | . | , | [|] | ’ ’ | ’_’

}

4.1 Our DSL Parser

The Parser class serves as a crucial component in our DSL interpreter, responsible for analyzing the syntax of our DSL code and executing corresponding actions to generate fractal images. Initialized with a lexer and a fractal object, the parser sequentially processes tokens obtained from the lexer. Upon encountering an invalid syntax, the parser raises an exception to maintain code integrity. The parsing process involves matching tokens against expected token types, ensuring syntactic correctness. Within the parse_commands method, the parser iterates through tokens until reaching the end of the input. For each command encountered, parse_command identifies the command type and triggers the corresponding action in the fractal object. For instance, commands like ’size’, ’color’, ’background’, ’speed’, ’shape’, and ’draw’ are parsed and appropriately executed, updating relevant parameters in the fractal object. This systematic parsing ensures the accurate interpretation and execution of our DSL code, facilitating the creation of intricate fractal designs.

```
class Parser:
    def __init__(self, lexer, fractal):
        self.lexer = lexer
        self.fractal = fractal
        self.current_token = self.lexer.get_next_token()

    def error(self):
        raise Exception("Invalid syntax")

    def match(self, expected_token_type):
```

```
        if self.current_token.type == expected_token_type:
            self.current_token = self.lexer.get_next_token()
        else:
            self.error()

    def parse(self):
        self.parse_commands()

    def parse_commands(self):
        while self.current_token.type != "EOF":
            self.parse_command()

    def parse_command(self):
        command = self.current_token.value.lower()
        if command == 'size':
            self.match("SIZE")
            size_value = self.current_token.value
            self.match("NUMBER")
            self.fractal.set_size(int(size_value))
        elif command == 'color':
            self.match("COLOR")
            color_value = self.current_token.value
            self.match("STRING")
            self.fractal.set_color(color_value.strip("'").strip('"'))
        elif command == 'background':
            self.match("BACKGROUND")
            background_value = self.current_token.value
            self.match("STRING")
            self.fractal.set_background(background_value.strip("'").strip('"'))
        elif command == 'speed':
            self.match("SPEED")
            speed_value = self.current_token.value
            self.match("NUMBER")
            self.fractal.set_speed(int(speed_value))
        elif command == 'shape':
            self.match("SHAPE")
            shape_value = self.current_token.value
            self.match("STRING")
            self.fractal.set_shape(shape_value.strip("'").strip('"'))
        elif command == 'draw':
```

```
            self.match("DRAW")
            draw_value = self.current_token.value
            self.match("NUMBER")
            self.fractal.draw()
        else:
            self.error()
```

Listing 4.1.1 - Our DSL Parser

4.2 Our DSL Lexer

The Lexer class functions as an essential component in our DSL interpreter, responsible for breaking down input text into individual tokens, each representing a meaningful component of our DSL syntax. Upon initialization with input text, the lexer processes the text character by character, identifying and categorizing tokens based on predefined rules. The Token class defines the structure for tokens, encapsulating a type and a corresponding value. This structure aids in the organization and interpretation of tokens within our DSL interpreter. Within the Lexer class, the get_next_token method serves as the main functionality for tokenization. It iterates through characters in the input text, skipping whitespace characters and identifying the type of each token encountered.

- If the current character is alphabetic, the lexer identifies keywords by extracting consecutive alphabetic characters until a non-alphabetic character is encountered. Based on the keyword, the lexer assigns an appropriate token type, such as ”SIZE”, ”COLOR”, ”BACKGROUND”, etc., and creates a corresponding token object.
- If the current character is a digit, the lexer identifies and extracts consecutive digits to form a numeric token, assigning the token type as ”NUMBER” and creating a token object with the numeric value.
- If the current character is a single quote (”) or double quote (”), the lexer identifies and extracts characters enclosed within the quotes to form a string token. The token type is set as ”STRING”, and a token object with the string value is created.
- If the current character does not match any predefined token rules, the lexer raises an exception to indicate an invalid character in the input text.

Throughout the tokenization process, tokens are appended to a list maintained by the lexer, enabling access to the generated tokens for further processing. By systematically parsing input text and generating tokens according to predefined rules, the lexer lays the foundation for accurate interpretation and execution of our DSL code within the interpreter.

```
class Token:
    def __init__(self, type, value):
        self.type = type
        self.value = value
```

```
5
6       def __str__(self):
7           return f"Token({self.type}, {self.value})"
8
9
10  class Lexer:
11      def __init__(self, text):
12          self.text = text
13          self.pos = 0
14          self.tokens = []
15
16      def error(self):
17          raise Exception("Invalid character")
18
19      def get_tokens(self):
20          return self.tokens
21
22      def get_keyword(self):
23          keyword = ""
24          while self.pos < len(self.text) and self.text[self.pos].isalpha():
25              keyword += self.text[self.pos]
26              self.pos += 1
27          return keyword
28
29      def get_next_token(self):
30          while self.pos < len(self.text) and self.text[self.pos].isspace():
31              self.pos += 1
32
33          if self.pos >= len(self.text):
34              return Token("EOF", None)
35
36          current_char = self.text[self.pos]
37
38          if current_char.isalpha():
39              keyword = self.get_keyword()
40              if keyword.lower() == 'size':
41                  token = Token("SIZE", keyword)
42              elif keyword.lower() == 'color':
43                  token = Token("COLOR", keyword)
44              elif keyword.lower() == 'background':
```

```
                token = Token("BACKGROUND", keyword)
            elif keyword.lower() == 'speed':
                token = Token("SPEED", keyword)
            elif keyword.lower() == 'shape':
                token = Token("SHAPE", keyword)
            elif keyword.lower() == 'scale':
                token = Token("SCALE", keyword)
            elif keyword.lower() == 'rotate':
                token = Token("ROTATE", keyword)
            elif keyword.lower() == 'draw':
                token = Token("DRAW", keyword)
            else:
                self.error()
            # self.pos += len(keyword)
            self.tokens.append(token)
            return token

        if current_char.isdigit():
            num = ""
            while self.pos < len(self.text) and self.text[self.pos].isdigit():
                num += self.text[self.pos]
                self.pos += 1
            token = Token("NUMBER", int(num))
            self.tokens.append(token)
            return token

        if current_char in ['\'', '"']:
            quote_char = current_char
            string = ""
            self.pos += 1
            while self.pos < len(self.text) and self.text[self.pos] != quote_char
    :
                string += self.text[self.pos]
                self.pos += 1
            self.pos += 1
            token = Token("STRING", string)
            self.tokens.append(token)
            return token

```

```
83          self.error()
```

Listing 4.2.1 - Our DSL Lexer

4.3 Examples

In illustrating the practical application of our DSL, two examples are presented showcasing its capabilities in generating fractal images from user-defined inputs. In the first example, the user provides our DSL code snippet specifying parameters such as size, color, and shape to define the desired fractal. Upon execution, our DSL interpreter parses the input, generates the corresponding fractal image, and displays it to the user. Subsequently, in the second example, another our DSL code segment is supplied, this time configuring different parameters to create a distinct fractal pattern. Once interpreted, the resulting fractal image is generated based on the specified parameters and presented to the user. These examples vividly demonstrate the flexibility and ease of use afforded by our DSL in creating diverse and visually captivating fractal designs.

Generation of a Sierpinski triangle using our DSL.

```
size 800
color '#9ACB34'
background '#6534CB'
shape 'triangle'
speed 0
draw 0
```

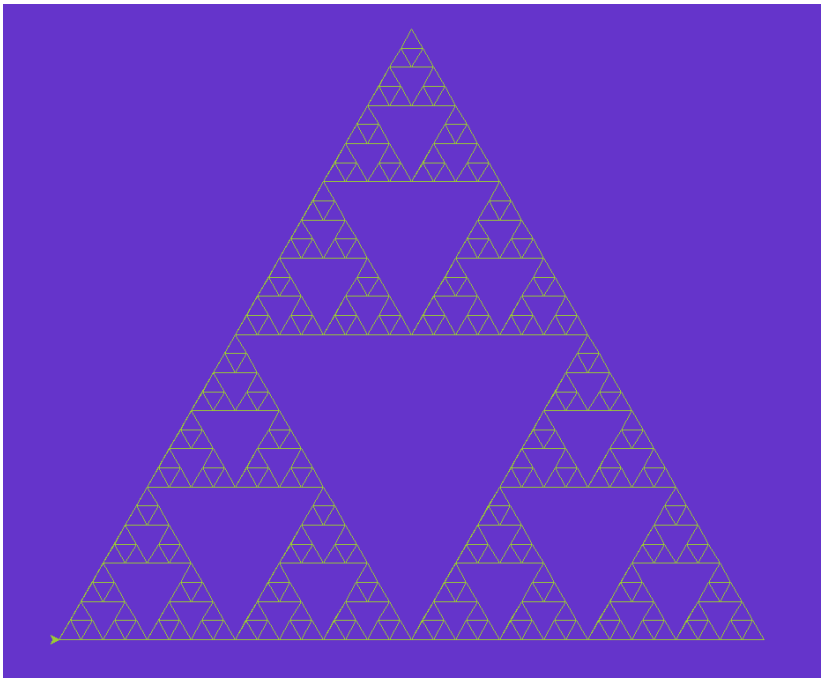

Figure 4.3.1 - Example 1. Sierpinski Triangle.

Generation of the Dragon Curve using our DSL.

```
size 800
color 'blue'
background 'red'
shape 'dragon'
speed 0
draw 0
```

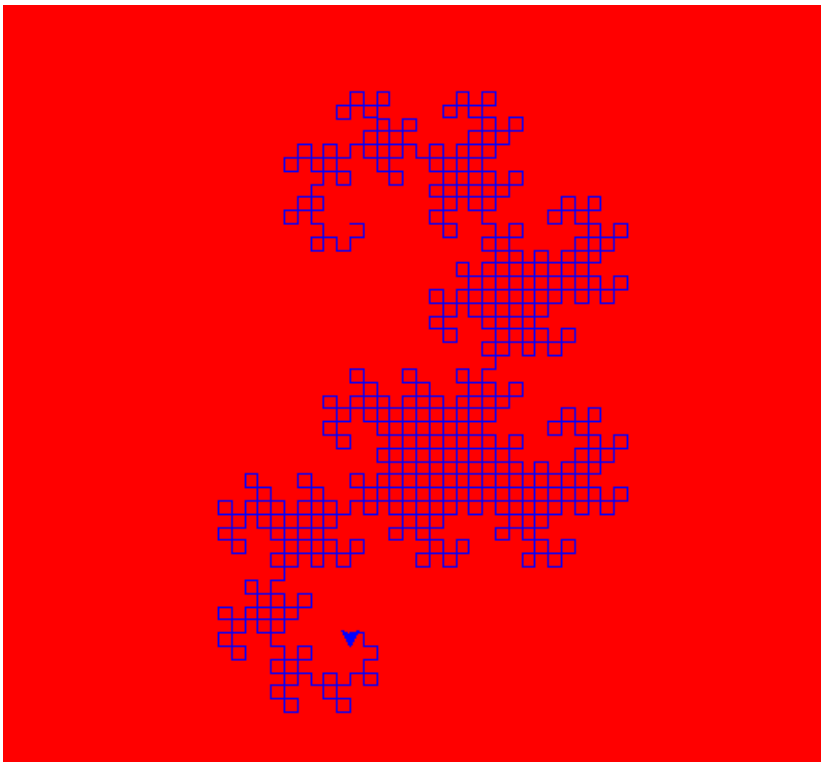

Figure 4.3.2 - Example 2. Dragon Curve.

5 Midterm 2

Conclusions

This report has presented a comprehensive exploration of the domain of creating a domain-specific language (DSL) for generating fractals. Through in-depth problem analysis, we formulated a clear problem statement, outlining the need for an intuitive and accessible language for fractal generation. To address this, extensive domain analysis was conducted, examining the mathematical principles, algorithms, and visualization techniques fundamental to fractal design.

Based on our analysis, a solution was proposed for a DSL that would incorporate essential language constructs and features vital for fractal generation. We meticulously crafted a solution statement, detailing the envisioned capabilities and syntax of the proposed language.

However, due to limitations in our expertise with parsing tools such as ANTLR, we were unable to fully realize the lexer and parser components. These components are essential for translating the human-readable DSL code into machine-executable instructions.

Future plans

To complete the development of this DSL, it's imperative to acquire the necessary knowledge and skills in parser generation and lexer design. Further research into ANTLR and similar tools, along with a deeper understanding of compiler concepts, would be essential for successfully implementing these components.

This endeavor has highlighted the value of a DSL for fractal generation, establishing a strong foundation for future work. By addressing the current limitations, this project has the potential to provide a powerful and approachable tool for artists, mathematicians, and enthusiasts, empowering them to explore the intricate world of fractals.

Bibliography

[1] fractal org. ”Fractals: Useful Beauty”. Accessed February 15, 2024. https://www.fractal.org/Bewustzijns-Besturings-Model/Fractals-Useful-Beauty.htm.

[2] DESA pl. ”FRACTALS: THE UNITY OF SCIENCE AND ART”. Accessed February 15, 2024. https://desa.pl/en/stories/fractals-the-unity-of-science-and-art/.

[3] Siavash Mohammadi. ”Relation between fractal analysis and data science”. Accessed February 15, 2024. https://www.researchgate.net/post/Relation_between_fractal_analysis_and_data_science.

[4] Our team repository on github

Accessible always https://github.com/XRRRA/FRACTAL-DSL.