DSL FOR BUILDING FRACTALS

Liviu IORDAN*, Arina PERETEATCU, Pavel ȚAPU, Cristian PRODIUS

1Department of Software Engineering and Automation, gr FAF-223, Faculty of Computers, Informatics and Microelectronics Technical University of Moldova, Chișinău, Republic of Moldova

*Corresponding author: Liviu Iordan, email: liviu.iordan@isa.utm.md

Tutor: **Mariana CATRUC**, university lecturer

Abstract. *The Scientific Conference paper introduces a Domain Specific Language (DSL) dedicated to simplifying fractal geometry through automated coding. This language aims to make fractal generation easier for both beginners and experts by providing intuitive syntax and functionalities for defining shapes and parameters. The DSL's development involved domain analysis, design principles, and iterative cycles. Results show its effectiveness in creating intricate fractal patterns for various purposes such as art, education, and science. Potential applications include artistic expression, educational exploration, and integration into scientific simulations. This project contributes to computational geometry, offering a valuable tool for fractal enthusiasts and researchers.*

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

Introduction

Fractals, intricate geometric structures characterized by self-similarity, have fascinated humanity for centuries, from ancient African architecture to modern art movements. Their ubiquity in nature and art underscores their profound influence across disciplines. In this paper, the intersection of fractal geometry with computer science is explored, emphasizing its relevance in contemporary research and creative endeavors.

Fractals, with their inherent ability to exhibit self-similarity across varying scales, serve as a captivating lens through which the convergence of art, nature, and scientific inquiry is explored. As one navigates through the rich tapestry of fractal imagery and its historical significance, the symbiotic relationship between mathematical abstraction and human creativity is unveiled. From the ancient origins of fractal motifs in indigenous architectural marvels to the avant-garde experimentation of 20th-century Surrealist artists, the allure of fractals has transcended cultural boundaries, captivating the imagination of scholars and artisans alike.

In this interdisciplinary exploration, the researchers aim to shed light on the transformative potential of fractal geometry within the realm of computer science. Through an in-depth analysis of fractal analysis techniques and their applications in data science, the study elucidates how these geometric marvels serve as a powerful tool for unraveling the complexities of real-world datasets. Moreover, the paper delves into the realm of domain-specific languages tailored for fractal generation, envisioning a future where intuitive programming interfaces empower individuals across diverse backgrounds to engage in the creation and exploration of fractal art and scientific inquiry.

Fractals in Art and Nature:

The historical and artistic significance of fractals is analyzed, tracing their roots in traditional African architecture to their adoption by Surrealist artists like Max Ernst. Notably, it discusses the breakthrough analysis of Jackson Pollock's works, highlighting how fractals captivate viewers and contribute to stress reduction.

Fractal Analysis and Data Science:

Fractal analysis intersects with data science, offering insights into complex datasets through techniques like feature extraction, data visualization, and time series analysis. It explores how fractal dimensions enhance machine learning models and aid in understanding intricate data structures.

Domain-Specific Language (DSL) for Fractal Geometry:

The proposal suggests developing a user-friendly DSL tailored for fractal generation, addressing the limitations of existing tools. The DSL aims to democratize fractal exploration by offering intuitive syntax, comprehensive grammar, and versatile features for customization.

Implementation Approach:

The provided DSL would be implemented as an internal DSL, seamlessly integrated with existing programming languages. Leveraging language design principles, the focus is on performance and scalability to handle complex fractal computations efficiently.

Impact and Benefits:

The development of a comprehensive Fractal DSL promises to revolutionize artistic expression, scientific exploration, and educational advancement. By lowering the entry barrier, the DSL empowers users across disciplines to engage meaningfully with fractal geometry, fostering innovation and interdisciplinary collaboration.

Grammar

The provided grammar is a context-free grammar (CFG) designed to describe the syntax of a programming language used for creating fractals. Fractals are complex geometric shapes that can be split into parts, each of which is a reduced-scale copy of the whole. This grammar defines the structure of programs written in this language, specifying how statements, commands, function calls, and conditional statements are organized. It facilitates the parsing and interpretation of code written in the fractal programming language.

Table 1

Grammar notations

Notation	Description
<foo>	non-terminal symbol
foo	terminal symbol in the grammar
|	separates alternative choices for a production rule
→	denote a production rule

$$G = (V_N, V_T, P, S)$$

V_N - syntactic categories or abstract components of the language's grammar.

V_T - terminal symbols represent the basic building blocks of the language, such as keywords, literals, punctuation marks, and logical operators

P - production rules define how the non-terminal symbols can be replaced by sequences of terminal and non-terminal symbols.

S - the start symbol denotes the beginning of a program written in the fractal programming language

V_N = { <program>, <statement>, <command>, <function_call>, <arguments>, <shape_function>, <recursive_function>, <value>, <string>, <number>, <shape>, <function_name>, <parameter_list>, <parameter>, <if_statement>, <else_statement>, <comparison>, <logical_operator>}

V_T = {size, color, background, speed, shape, depth, points, length, direction, edges, draw, 'STRING', 'NUMBER', '(', ')', ',', if, else, '==', '!=', '<', '>', '<=', '>=', and, or}

P = { <program> → <statement> | <statement> <program>
<statement> → <command> | <function_call> | <if_statement>
<command> → size <number> | color <string> | background <string>
| speed <number> | shape <string> | depth <number>
| points <number> | length <number> | direction <number>
| edges <number> | draw
<function_call> → <shape_function> | <recursive_function>
<shape_function> → <function_name> '(' <arguments> ')'
<recursive_function> → draw '(' ')'
<arguments> → <parameter_list> | ε
<parameter_list> → <parameter> | <parameter> ',' <parameter_list>
<parameter> → <value>
<value> → <number> | <string>
<string> → 'STRING'
<number> → 'NUMBER'
<shape> → triangle | koch | dragon | capital | fern | tree | star | snowflake | gardi | spiral
<function_name> → <shape>
<if_statement> → if <comparison> <logical_operator> <comparison> <command> <else_statement>
<else_statement> → else <command> | ε
<comparison> → <value> '==' <value> | <value> '!=' <value> | <value> '<' <value>
| <value> '>' <value> | <value> '<=' <value> | <value> '>=' <value>
<logical_operator> → and | or}

S = <program>

Program showcase

The following program showcases the creation of a snowflake fractal with ten edges, each composed of three smaller edges. The snowflake is rendered in yellow against a green background. The canvas size is set to 800 pixels in width and length, providing ample space for the intricate design. By setting the speed parameter to 0, the turtle drawing the fractal moves at the fastest possible speed, ensuring efficient rendering. Finally, the command "draw 0" specifies that the fractal should indeed be drawn, bringing the intricate snowflake pattern to life on the canvas.

size 800
color 'yellow'
background 'green'
shape 'snowflake'
depth 3
edges 10
speed 0
draw 0

Parsing tree

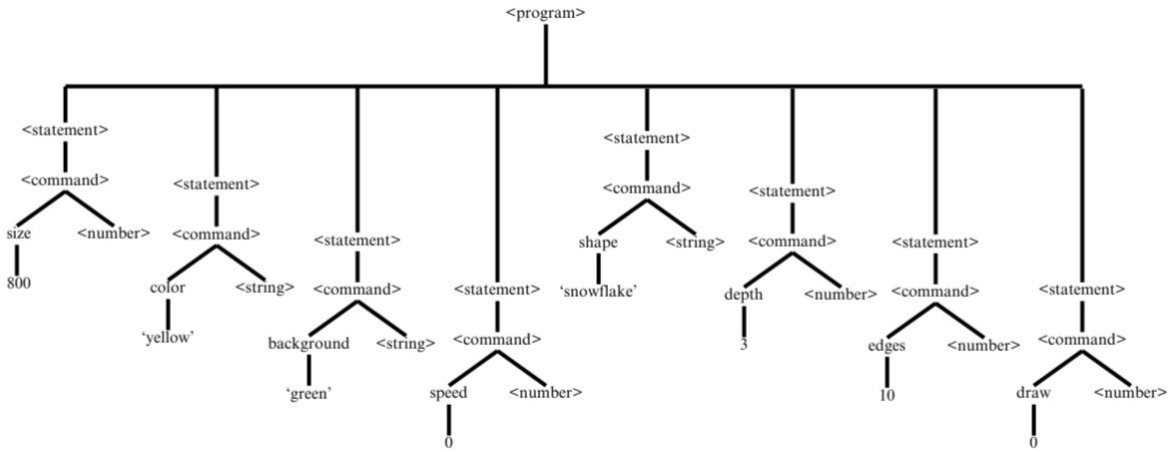

Figure 1. Parsing tree

The parsing tree delineates a series of statements, each encapsulating a specific command for defining and rendering fractals. These statements are integral components of a DSL designed to streamline the creation process while ensuring flexibility and expressiveness. Here's a breakdown of the key elements represented in the parsing tree: **<program>**: This serves as the root of the parsing tree, indicating the initiation point for defining fractals. It orchestrates a sequence of statements that collectively configure the properties and characteristics of the fractal to be generated. **<statement>**: Within the parsing tree, each statement encapsulates a distinct command responsible for configuring various aspects of the fractal. These statements collectively define the fractal's attributes, including its size, color, background, speed, shape, depth, edges, and the drawing action itself. **<command>**: The commands embedded within each statement dictate specific actions to be performed during fractal generation. These commands accept parameters such as numerical values or string inputs, enabling users to customize fractal attributes according to their preferences.

Attributes: Size: Specifies the size of the fractal, influencing its overall dimensions. **Color:** Determines the color scheme applied to the fractal, enhancing its visual appeal. **Background:** Sets the background color against which the fractal is rendered, providing contrast and context. **Speed:** Regulates the rendering speed of the fractal, facilitating smooth visualization. **Shape:** Defines the geometric shape utilized as the basis for fractal generation, offering versatility and creative freedom. **Depth:** Controls the complexity and intricacy of the fractal pattern by specifying the recursion depth. **Edges:** Specifies the number of edges or segments comprising the fractal, influencing its overall structure. **Draw:** Triggers the rendering process, instructing the system to generate and display the fractal based on the defined parameters.

Conclusions

In conclusion, this article has delved into the fascinating intersection of fractal geometry with computer science, highlighting its profound implications across various domains. From its historical and artistic significance to its practical applications in data science and beyond, fractals continue to captivate the imagination of scholars, artists, and researchers alike.

The development of a Domain-Specific Language (DSL) dedicated to simplifying fractal geometry represents a significant step forward in democratizing fractal exploration. By providing an intuitive platform for defining shapes and parameters, the DSL bridges the gap between complex mathematical concepts and user-friendly programming interfaces. This advancement not only empowers individuals across diverse backgrounds to engage meaningfully with fractal geometry but also opens up new avenues for artistic expression, scientific exploration, and educational advancement.

Through the comprehensive grammar and versatile features offered by the DSL, users can effortlessly create intricate fractal patterns for various purposes, including art, education, and scientific simulations. By lowering the entry barrier and abstracting away technical complexities, the DSL facilitates innovation and interdisciplinary collaboration in the captivating field of fractal geometry.

As we look toward the future, the potential impact of the Fractal DSL extends far beyond its initial development. It promises to revolutionize artistic expression, scientific inquiry, and educational exploration, unlocking new realms of creativity, discovery, and understanding. By fostering a community of fractal enthusiasts and researchers, the DSL paves the way for continued innovation and advancement in computational geometry and beyond.

In essence, the journey into fractal geometry through the lens of computer science is one of boundless possibility and profound insight. As we embark on this journey together, let us embrace the transformative power of fractals to inspire, educate, and illuminate the world around us.

References:

[1] Fractal Foundation: What are Fractals? [online], [accessed on 08.03.2024]: https://fractalfoundation.org/
[2] HowStuffWorks: How Fractals Work? [online], [accessed on 07.03.2024]: https://science.howstuffworks.com/math-concepts/fractals.htm
[3] HowStuffWorks: How DSL Works? [online], [accessed on 07.03.2024]: https://computer.howstuffworks.com/dsl.htm
[4] CosmosMagazine: Do fractals exist in nature? [online], [accessed on 06.03.2024]: https://cosmosmagazine.com/science/mathematics/fractals-in-nature/
[5] Eclipse: Forums [online], [accessed on 06.03.2024]: https://www.eclipse.org/forums/index.php/t/1107070/
[6] Duncans: Nature of Code Fractals [online], [accessed on 08.03.2024]: https://wp.nyu.edu/tischschoolofthearts-duncanfigurski/2021/04/06/nature-of-code-fractals/
[7] Ola Bini: Fractal Programming [online], [accessed on 09.03.2024]: https://olabini.se/blog/2008/06/fractal-programming/
[8] MANDELBROT, B. *The fractal geometry of nature*. Times Books, 1982
[9] MERNIK, M., HEERING, J., & SLOANE, A. M. When and how to develop domain-specific languages. In: *ACM Computing Surveys* (CSUR), 2005, 37(4), 316-344.
[10] FOWLER, M. *Domain-specific languages*. Addison-Wesley Professional, 2010
[11] AHO, A. V., LAM, M. S., SETHI, R., & ULLMAN, J. D. *Compilers: principles, techniques, and tools. Pearson Education.* Addison-Wesley, 2006