DOMAIN SPECIFIC LANGUAGE FOR WEB GRAPHICS

Anastasia IAȚCO*, Marius PURICI, Vasile IGNAT, Andrei PĂGÂNU

Department of Software Engineering and Automatics, group FAF-202 , Faculty of Computers, Informatics and Microelectronics, Technical University of Moldova, Chisinau, Moldova

*Corresponding author: Anastasia Iațco, anastasia.iatco@isa.utm.md

Abstract. *The aim of this paper is to show a domain-specific language (DSL) that was designed special for web graphics. Moreover, this article describes the grammar of the DSL and how it will work, what are its functions and how this language will relieve the learning and implementing activity for everyone who is interested in web graphics.*

Keywords*: domain-specific language, grammar, web graphics.*

Introduction

The importance of the domain-specific languages in software engineering has significantly expanded in recent times, as a consequence of that fact, that they are designed in order to maintain a specific set of functions from a certain domain [1]. A DSL holds constructs which absolutely suit the issue space, allows people that have no relation with the specific domain acknowledge the general idea and makes it less complex to create a model of the final application.

This article has the purpose to explain the development of a language for creating web graphics, and at the same time reducing the complexity involved in tasks related to WEBGL and web graphics in general.

Web graphics in any websites is as significant as the content of the site. Graphics can distract, teach, or emotionally influence the user, and are decisive for coherence of illustration, and ease of utilization for interfaces [2].

A lot of people face some issues while learning and working, because they have insufficient experience that causes the appearance of some difficult moments. WEBGL is not the easiest library of JavaScript, and many people struggle with learning it, especially at the beginning.

The problem that the proposed DSL aims to solve is the inability of including interactive web graphics from the point of view of someone with limited experience in the field of computer graphics.

The way in which the domain-specific language aims to solve the given problem as follows: DSL will automate many things that are present in every WebGL JavaScript file. It will provide a much more simplified language and will significantly reduce the number of lines needed for mundane actions by cutting down on the definitions that a user would need to make by generalizing different parts of the process.

Specifications of the DSL

The fundamental features of the described domain specific language are:

- Create and manipulate 2D and 3D objects;
- Move the camera;
- Load images or obj files;
- Apply texture.

Reference grammar

Grammar is a set of rules that grants the programming language a special form. As a rule, a grammar is written in the following way: G = {V_T, V_N, P, S} where:

- "V_T" - set of terminals.
- "V_N" - set of non-terminal.
- "S" - start symbol.
- "P" - the set of production rules.

In order to have a greater perception of the grammar, in this paper are being specified particular notations. (Table 1. Metanotations).

Table 1

Metanotations

<symbol>	**symbol is a nonterminal**
symbol	**symbol** is a terminal (a part of a token).
[x]	x is optional; note that brackets in quotes '[' ']' are terminals
x*	zero or more occurence of x
{ }	used for grouping
|	separates alternatives

N = { <program> , <statement_list>, <statement>, <var_declaration> , <transform_command> , <apply_texture_command>, <transform_matrix_declaration> , <apply_transform_command>, <create_command>, <id>, <number_literal> , <object_type>, <object_prop>, <image_link>, <prop_assignment>, <prop_name>, <number_literal>, <apply_texture_command>, <transform_command_list>, <digit>, <alpha> }

T = { Object, Number, Texture, Create, Create Texture, Image, Triangle, Cube, Sphere, Pyramid, X, Y, Z, Radius, Scale, TranslateX, TranslateY, TranslateZ, RotateX, RotateY, RotateZ, ApplyTexture, matrix, Transform, 0..9, a..z, A..Z }

S = { program }

Rules:

P = { <program> → <statement_list>
<statement_list> → <statement><statement_list>
<statement_list> → <statement>
<statement> → <var_declaration> | <transform_command> |
<apply_texture_command> | <transform_matrix_declaration> |
<apply_transform_command> | <create_command>
<var_declaration> → **Object** <id> = <create_command> |
Number <id> = <number_literal> **Texture** <id> = <create_command>
<create_command> → **Create** <object_type> |
Create <object_type> { <object_prop> } | **Create Texture** { **Image** <image_link>}
<object_type> → **Triangle** | **Cube** | **Sphere** | **Pyramid**
<object_prop> → <prop_assignment> , <object_prop>
<object_prop> → <prop_assignment>
<prop_assignment> → <prop_name> = <id> | <prop_name> = <number_literal>
<prop_name> → **X** | **Y** | **Z** | **Radius** | **Scale**

<transform_command> → **TranslateX**(<id>) | **TranslateY**(<id>) |
TranslateZ(<id>) | **Scale**(<id>) | **RotateX**(<id>) | **RotateY**(<id>) | **RotateZ**(<id>) |
TranslateX(<number_literal>) | **TranslateY**(<number_literal>) |
TranslateZ(<number_literal>) | **Scale**(<number_literal>) |
RotateX(<number_literal> | **RotateY**(<number_literal>) |
RotateZ(<number_literal>)
<apply_texture_command> → **ApplyTexture**(<id>, <id>)
<transform_matrix_declaration> → **matrix** { <transform_command_list> }
<transform_command_list> → <transform_command> , <transform_command_list>
<transform_command_list> → <transform_command>
<apply_texture_command> → **Transform**(<id>, <id>)
<number_literal> → <digit> <number_literal>
<number_literal> → <digit>
<digit> → **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9**

<alpha> → **a** | **b** | **... z** | **A** | **B** | **... Z**
<id> → <alpha> <id>
<id> → <alpha> <digit>
<image_link> → <char>* }

Code example

A presentation of a code snippet, that was constructed using the DSL described in this paper is shown in Fig. 1:

```
matrix m1 {
    TranslateX(3),
    TranslateY(2),
    TranslateZ(1),
    Scale(2)
};

Object obj1 = Create Cube {
    X = 1,
    Y = 1,
    Z = 0
};

Texture t1 = Create Texture {
    Image = 'D:\img\texture.png';
};

Transform(obj, m1);
Transform(obj, TranslateX(1));
Transform(obj, RotateX(90));
ApplyTexture(obj, t1);
```

Figure 1. DSL code example

Derivation Tree

A derivation tree or parse tree is a graphical representation that illustrates the way in which strings in a language are being derived, taking in consideration the rules of the grammar [3]. Fig. 2 illustrates the derivation tree obtained from an example of code.

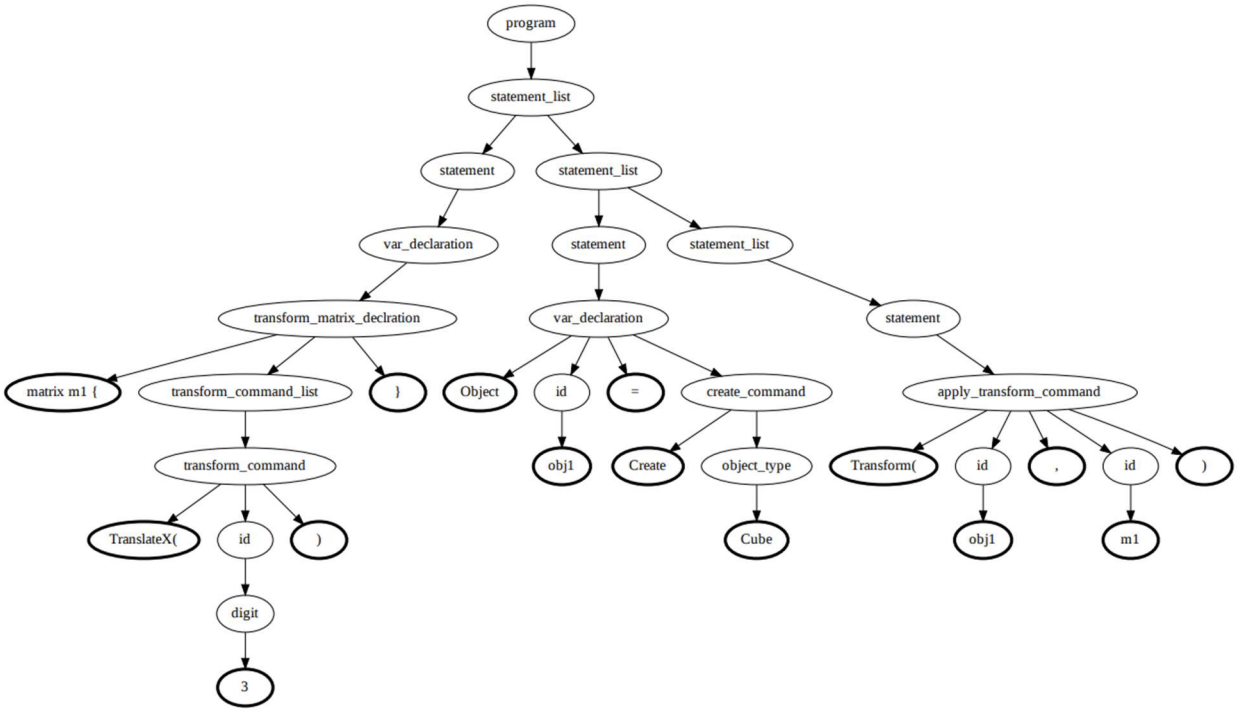

Figure 2. DSL Derivation Tree

Conclusion

This paper presented the Domain Specific Language for Web Graphics that is proposed to solve the problem of learning hard topics and unwillingness of some to work hard, and solution – make the learning process easier and more pleasant. The target niche is not so extensive, having relation on the students of IT faculties and people interested in web graphics.

To summarize everything that was written, the domain specific language that is being developed is a great alternative to that solutions which have already been discovered, but with more advantages and ascendancy sides.

References

1. JET BRAINS, Domain-Specific Languages [online] [accessed 25.02.2022] Available: https://www.jetbrains.com/mps/concepts/domain-specific-languages/
2. WEB DESIGN AND APPLICATIONS, Graphics, [online] [accessed 25.02.2022] Available: https://www.w3.org/standards/webdesign/graphics
3. SCIENCEDIRECT, Derivation Tree [online] [accessed 25.02.2022] Available: https://www.sciencedirect.com/topics/computer-science/derivation-tree