Domain-specific Languages as Tools for Teaching 3D Graphics

Kęsik Jacek[1], Nowakowski Kamil[2] and Żyła Kamil[1]
[1]*Institute of Computer Science, Lublin University of Technology, 36b Nadbystrzycka St., Lublin, Poland*
[2]*Independent consultant, Lublin, Poland*

Keywords: 3D Graphics, Domain-specific Languages, Model-Driven Engineering, Modeling Shaders, Teaching.

Abstract: Model-driven engineering is constantly gaining importance, expanding to domains varying from the Web to the 3D graphics. Domain-specific languages besides contributing to the development process can be used in a didactic process conducted not only in schools. Thus this paper introduces new domain-specific language and discusses its usage in teaching construction of shaders and materials while working with 3D graphics. It presents the authors stance regarding the usefulness of domain-specific languages in education of 3D graphics development.

1 INTRODUCTION

During a process of creating 3D computer visualizations, as well as games, developers use specialized tools, including self-made ones. One of these tools is materials editor, which main purpose is defining optical properties of lighted surfaces on the 3D stage. Each material can be described by a set of parameters which allows to obtain and customize effects like - reflections, mattness, surface distortions etc. These sets are then passed to a specially designed program running on a graphics card, called shader, which purpose is to color and illuminate particular pixels on the screen (de Carvalho, Gill and Parisi, 2004).

Graphics card is a device specialized in image processing, vertex transformations (e.g. animation of 3D models of plants), triangles rasterization, etc. Moreover, shaders are usually the subject of strong parallelization performed on many graphics processors. As the result a programmer or a graphic designer needs to know low-level architecture and commands specific for this environment. As can be expected only limited number of people is able to work on mentioned graphical effects without the help of specialized tools, usually designed for specific purpose. Inexperienced programmers face major difficulties with reading and understanding shaders written in specialized language.

These factors cause the need of seeking for higher-level solutions, that are able to ease-up and speed-up work of people involved in 3D graphics and to extend this group by lowering requirements concerning shader design knowledge. Thus the idea of utilizing model-driven engineering (MDE) concepts during teaching and development process, in the 3D graphics domain, is quite attractive.

This paper is organized in three main sections. First presents evolution of languages for 3D graphics leading to the usage of domain-specific languages (DSLs). Next one introduces new DSL for building shaders and materials from its usage in a didactic process point of view. Last one describes authors' experiences concerning incorporation of MDE in a teaching process.

2 3D GRAPHICS AND MDE

One of the most programmatically complex tasks in 3D graphics development is generation of materials and effects using shaders - subprograms uploaded and executed by graphic card unit. It is almost natural application for DSL languages, as typical 3D developers not necessarily possess high level programming skills required for that task. It has been already noticed by main players on the market and suitable DSLs are implemented into their 3D development environments.

Initially shaders were programmed in low-level language similar to Assembler, where programmer worked on particular processor instructions, registers, banks, etc. Also, performing micro-optimizations by "tricks" was common practice

Jacek K., Kamil N. and Kamil Ż..
Domain-specific Languages as Tools for Teaching 3D Graphics.
DOI: 10.5220/0004884404980503
In *Proceedings of the 2nd International Conference on Model-Driven Engineering and Software Development* (MODELSWARD-2014), pages 498-503
ISBN: 978-989-758-007-9

(Bailey and Cunningham, 2012). Listing 1 presents exemplary part of simple shader, and listing 2 presents the same part after simple optimization by replacing few processor instruction with one, which is equivalent to: (A - 0.3) × 2.5 = A × 2.5 + (-0.75). It is easy to imagine, the code complication level of a typical, optimized shader.

Listing 1. Part of exemplary shader.

```
def             c0, 0.3, 2.5, 0 , 0

textld          r0, t0
sub             r0, r0, c0.x
mul             r0, r0, c0.y
```

Listing 2. Part of exemplary shader after optimization.

```
def             c0, -0.75, 2.5, 0 , 0

textld          r0, t0
mad             r0, r0, c0.y, c0.x
```

The need for more programmer friendly syntax was obvious. Next step in evolution of languages for programing shaders has been triggered by popularity of C / C++ languages. Their programmer friendly syntax was inspiration for High Level Shader Language (HLSL) for DirectX (Feinstein, 2013), OpenGL Shading Language (GLSL) for OpenGL (Rost et. al., 2010) and Cg language for both platforms. They became popular as they speed-up development process, ease-up maintaining code and decrease number of code version iterations needed to obtain final effect. Listing 3 presents exemplary shader written using C / C++ like language.

Listing 3. Basic shader written in C / C++ like language.

```
void main (void)
{
   vec4 base = texture2D(text, uv);
   gl_FragColor.rgb = mix(
        gl_Fog.color.rgb,
        base.rgb * ambient,
        fog_param );
   gl_FragColor.a = base.a;
}
```

Currently, in parallel to the above mentioned languages, state of the art (sharing in some degree MDE concepts) solutions exist. They are developed and published along with well-known commercial and open source projects for 3D graphics manipulation and game design. The nature of such projects - releasing to the broad audience of users, implies the need for maximal ease of converting designer's concepts to the shader language. The approaches worth mentioning are presented below.

Blender's Node Editor. The concept of designing materials covering the objects has been made available to basic level users by utilizing Node Editor. It allows to graphically mount visual effects applied to the material. The whole process of designing effect is reduced to connecting graphical units (nodes) and adjusting parameters. The connection is kept in tree model allowing for many outputs and single input of a node. The graphical representation of a node allows for in-place presentation of the effect acquired at the specific point (Valenza, 2013). The example of whole concept is presented in the figure 1. The nodes available for designer are grouped into sets dedicated to different aspects of material creation. The user of Node Editor is required only to possess knowledge about specific material creation, without the need to understand low or high level shader language.

3DS Max Slate Material Editor. The Slate Material Editor is designed according to the same concepts, thus the overall look of it is similar to the Node Editor. One can argue which one borrows concepts from which. The design elements are divided into Material, Map and Controller units, where material is the main object, controlled by connecting sets of other type objects. Final solution is connected to the graphical object of choice. As in the Node Editor case, the user is required only to possess knowledge about specific material creation. His actions are applied instantly to the connected graphical element but a render pass is required to actually see the changes on an object (Murdock, 2011).

UDK Editors. The Unreal Development Kit provides several DSLs as an aid in game development. The Material Editor is another example of "connect objects" approach to material creation, resulting in creation of specific shader. It is similar in user operation to other approaches. What's more, the created materials can be used by a set of additional tools e.g. Cascade Particle Editor allowing for, simple in operation, conducting of complicated task like preparing system of particle emitters (Doran, 2013). UDK is also equipped with Kismet DSL - much more complex approach. Due to its general game control purpose, not fixed on developing shaders, it is not considered in this list.

All of these approaches share the same concepts of connecting elements and behaviours well known to 3D object designers, thus it is quite simple for them to develop materials accordingly to their needs. The similarity and utilization of common concepts allows for simplification of explaining used

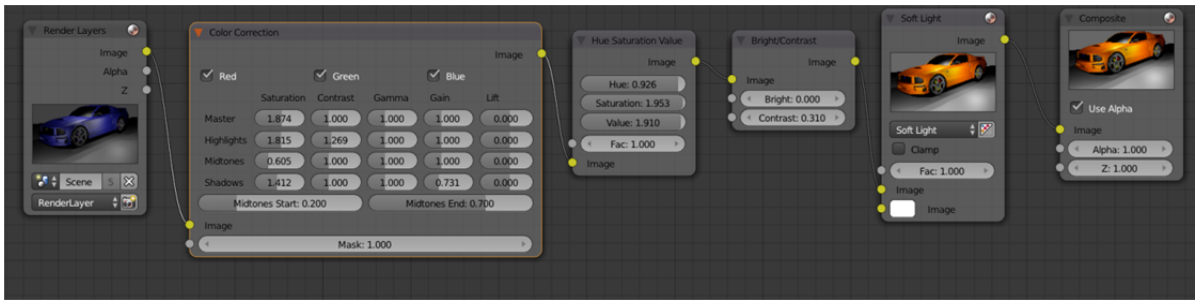

Figure 1: Example of a node type of DSL for defining materials for 3D object. Solution utilized in Blender.

approaches, thus improving education process both in classroom and tutorial approach.

3 ROLE OF SKINSHADER IN EDUCATION OF GAME DESIGN

Language introduced in this chapter has been created as a response for needs of team developing massive multiplayer online role play game, based on the state of art 3D engine (Barok engine). Despite of existence of proven tools, noticeable part of their users were not able to create advanced shaders due to insufficient knowledge. Another factor was rapid upgrades in the engine forced by arousing needs of developed game. Standard game control script language would become too unstable to be used by inexperienced developers. The goal of developed graphical DSL - SkinShader was to provide simple high-level solution that could be used either for game control or for creating advanced shaders as well as training inexperienced team members.

Language components were inspired by a solution provided with Blender for building materials. Its main concepts are units (nodes) performing particular actions and connections, between them, determining flow of information. Another key concept is master unit, special case of regular unit that gathers all connections from other units in order to orchestrate execution of model and code generation. Each unit has input and output of specific type - it means that unit can compute input values only of types predefined for the particular unit, the same applies for the output of the unit. Such mechanisms allow to minimize errors among beginners and speed-up obtaining fully functional results exemplifying theoretical topics. Another advantage is the optical similarity to widely used Entity-Relationship Diagrams (ERD), which makes the look (figure 2) of language more familiar (Nowakowski, 2013).

One of the key aspects during teaching process is curiosity of trainees, who very often invent ideas and applications beyond the imagination of the language creator. Also the ability of adjusting solution according to technological changes and development requirements, decides of its usability. Introduced DSL was designed keeping in mind its flexibility and expandability, thus it can be expanded by defining new components using syntax similar to JSON and implemented scripting engine (Lua).

A clearly arranged interface allows for quick learning of beginners and also does not slow down the ones possessing expertise in its usage. Ability to group elements eases navigation in complex designs while built-in schema evaluator takes the burden of checking the correctness of connections. It is especially important in the early learning stage, when simple mistakes are most likely to occur.

Another important issue is the quality of generated code. Optimization is very important in developing shaders where every millisecond counts. Presented DSL has been equipped with optimization mechanisms allowing for generation of fast working shaders. The resulting code is thus made even more unfriendly to editing due to random variable names but its hand edition is not assumed.

A gross of actions is kept behind the stage. The developer does not need the overall, high level knowledge about developing and optimizing shaders to create complex materials. Using the primary mathematical operations and visual representation of calculation process, the developer can construct a technologically advanced code, consistent with many standards.

While the general purpose of SinShader creation was to make available conceptual creation of material shaders, irrespective of technology and architecture changes, it has proven to be an important aid in teaching of material creation concepts. It allows for explanation of an approach on a conceptual level while on the other hand enables

Figure 2: Interface of SkinShader editor.

presentation of working solution. All that without diving into nuances of shader-specific languages.

4 MDE IN A TEACHING PROCESS

MDE and DSLs are not commonly used concepts in regular academic education curricula. Current techniques are mostly focused on teaching methods of object programming in languages of the 3rd generation. Introduction of MDE (and especially DSLs) allows, in specific applications, for rapid progress to concepts explanation and implementation without the need for prior acquaintance to specificity of general purpose language (GPL) doing its work on lower level. Students specializing in specific branch are not forced to first learn the actual GPL used by teacher to explain concepts, while in their future career they might be using completely different one. It also minimizes issues that due to teachers' abilities and attitude, students have to learn set of different 3rd generation languages during one specialization, which is not directly dedicated to teaching of these languages.

Teaching of DSL concepts has been introduced by authors in the course of web-design, where the standard development method has been substituted with MDE approach (Parreiras, 2012). While seeking to confirm the statements presented above a set of surveys has been conducted among students during conducing of the course. The surveys have shown (figure 3) a large interest in such teaching methods, as 70% of students have rated the new method positively, while 82% declared comprehension of presented concepts. Nevertheless, the overall willingness to utilize MDE methods in web development was quite low: 36% of respondents. It can be explained by parallel declaration of high knowledge of web languages, thus exchanging it to a new concept did not seem profitable. The students who declared lower level of web languages knowledge were presenting more positive attitude. The negative bias in students attitude could be also created by limitations and drawbacks of presented solution, which was at the moment of conducting surveys in not fully mature stage.

Authors performed also a short informal evaluation of the SkinShader. Besides developers team, language was also introduced to members of student organization dealing with 3D graphics and animation. At first, students have been acquainted with the tool by presenting the step-by step process of simple tasks development. During the process they were encouraged to ask questions and propose next-step solutions. In the following phase students were asked to implement minor changes in already developed solutions. They were allowed to ask for help with performing specific tasks of their concept. The last stage was the singlehandedly development of specific real life concept of limited complexity,

for the purpose of 3D visualisation of historic city of Lublin.

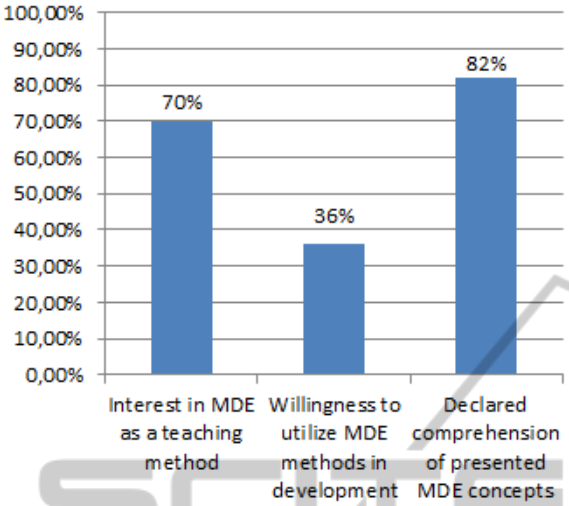

Figure 3: Results of survey on MDE concepts in web design.

In general, SkinShader as a tool for creating advanced shaders in real life projects, has been received positively. Students highlighted its simplicity and suggested that they would like to learn this language during regular classes. 80% of students were able to accomplish their tasks without the need of significant help. 53% of respondents were able to accomplish task without the help. Moreover, the surveys have shown (figure 4) a large interest in such teaching method, as 73% of students have rated the new method positively, while 87% declared comprehension of presented concepts. The overall willingness to utilize MDE methods in real life 3D graphics projects was quite high: 53% of respondents. Authors believe that this enthusiasm might be related to the state of knowledge of respondents, as they were less experienced in low-level programming of domain problems, then respondents working on Web technologies.

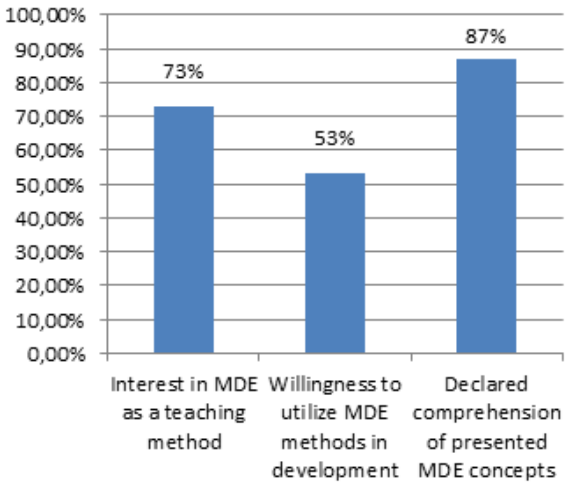

Figure 4: Results of survey on MDE concepts in 3D graphics design.

More detailed research will be conducted after introducing concepts of DSLs for modeling 3D graphics and animation into curricula. Nevertheless, the overall summary of surveys was encouraging to further introducing of MDE concepts into another courses, especially of not highly programmer focused branches.

5 CONCLUSIONS

Model-Driven Engineering is a very specific branch of software engineering, where opposite opinions concerning its usefulness, aside from didactic process, are fighting against each other. Particular MDE-like solution have chances to be widely accepted only when provides fair amount of flexibility and usability exceeding the cost of learning new solution. Also, personal preferences of developers play major role. It can be argued that developers possessing fair knowledge on solving domain problems will less likely switch to using technology that can potentially limit their freedom.

DSL for 3D graphics leverages programming to the conceptual modeling, which allows usage of domain experts' knowledge and rapid prototyping. Nevertheless, stating that complete newbie in particular domain can grasp all rules of language adjusted to the specific domain is a bit risky. In other word person pretending to use particular domain language has to know basic concepts of this domain. The learning process is though, in authors opinion, significantly faster than the regular one.

The presented SkinShader possess several features encouraging to use it as a training language in 3D development course. These features are common in the whole category of MDE solutions.

First the automation of low-level operations rises the abstraction level, minimizing the need for such specific knowledge. Developer can obtain the optimized solution not even being aware of the optimization process behind the scene. Secondly it provides the graphical platform independence, reliving the developer from the burden of attending to nuances of different types of graphics cards. The whole weight of this process is on the generator side, also assuring its implementation wherever it is needed. And as a final one, the rapid development of

materials and their shaders clearly supports the development process allowing for onsite testing of different approaches without losing time for coding it into specific shader syntax. All of that creates a developing environment on a conceptual level, especially suitable for teaching purposes.

From the education point of view the introduction of MDE is starting to prove successful, thus MDE aspects are more often introduced into educational programs. The student can fully concentrate on conceptual aspects of learned domain without the distraction of learning specific language syntax. It applies to 3D design as well. Authors can state that SkinShader DSL used by them in educational process has proven successful, what has been confirmed by opinion of students involved in 3D design projects. The presented solution has been usually described as comprehensible, elastic and worth learning.

REFERENCES

Bailey, M. and Cunningham, S. (2012). *Graphics Shaders: Theory and Practice*. CRC Press.

de Carvalho, G. N. M., Gill, T. and Parisi, T. (2004). X3D programmable shaders. In *Proceedings of the ninth international conference on 3D Web technology*, ACM, New York, (pp. 99 - 108). doi:10.1145/985040.985055

Doran, J. P. (2013). *Mastering UDK Game Development*. Packt Publishing.

Feinstein, D. (2013). *HLSL Development Cookbook*. Packt Publishing.

Murdock, K. L. (2011). *3ds Max® 2012 Bible*. Chapter 15, John Wiley & Sons.

Nowakowski, K. (2013). *Dokumentacja Barok Engine*. Retrieved October 23, 2013, from http://barokengine.com/dokumentacja/

Parreiras, F. S. (2012). *Semantic Web and Model-Driven Engineering*. Wiley.

Rost, R. J., Licea-Kane, B. M., et al. (2010). *OpenGL Shading Language (3rd Edition)*. Addison-Wesley.

Valenza, E. (2013). *Blender 2.6 Cycles: Materials and Textures Cookbook*. Packt Publishing.