IJCSI International Journal of Computer Science Issues, Volume 15, Issue 4, July 2018
ISSN (Print): 1694-0814 | ISSN (Online): 1694-0784
www.IJCSI.org https://doi.org/10.5281/zenodo.1346021

F-Rep Designer 2.0 – Everything is a Code

Alexander Penev

Faculty of Mathematics and Informatics, University of Plovdiv "Paisii Hilendarski"
Plovdiv, Bulgaria

Abstract

This article discusses the visualization approach applied to the experimental geometric modeling system F-Rep Designer 2.0. This system uses the Functional Representation (F-Rep) scheme in order to describe the model. In our approach the scene/model, the ray tracing algorithm, the shading model, and all that is necessary for the visualization are compiled into executable code for the selected target platform (CPU, GPGPU, etc.), and the execution of this code generates the final image. This is done through a multistep process, as the changes of the model lead to an optimized process of change in the various intermediate representations wherever possible.

Keywords: *Computer Graphics, F-Rep, Visualization, Model Compilation, Retargeting.*

1. Introduction

Computer graphics have many applications in different spheres of today's world. Geometric modeling systems have been developing for years and nowadays there are many systems with different capabilities and application areas.

Most modern geometric modeling systems use combinations of the well-known representation schemes Boundary representation (B-Rep), Constructive Solid Geometry (CSG) and others. One of their main advantages – the existence of hardware implementation of the visualization – is becoming less important with the development of modern graphics hardware (mainly GPGPU). This opens up the possibility of using more powerful and informationally complete models such as Function representation (F-Rep) [5] scheme for modeling and interactive work with geometric modeling systems based mainly on such class of model representations.

The project F-Rep Designer [1] was created to provide a base for exploring the theory and practice of systems with this kind of scene representation. Unlike most other systems the main model of scene representation is only F-Rep. It also aims to create an entirely mesh-free geometric modeling system. F-Rep Designer 2.0 is a logical continuation of the principles in the previously described work [1].

The problems in the first implementation of the F-Rep based system are mainly related to productivity. This was due to the transformation and the interpretation of the whole model (because of the limitations of existing GPGPU compilers), and the incomplete use of modern graphics hardware. The second version of the system overcomes much of these performance issues. The system has improved interactivity due to:

- an incremental compilation of the entire model, the visualization algorithm, and everything that is needed to visualize the modeled scene;
- a suitable caching scheme for intermediate results;
- a better load of graphic hardware.

While working on the first version of F-Rep Designer, we noticed that much of the model as well as the other elements of the system can be combined and compiled together (to an executable code). This is not accidental, and in practice it can be extended even further so that everything that is necessary for the visualization is combined and compiled. It is due to the fact that the scene/ model has a well-defined semantics which defines a 2D/3D images. Algorithms for visualization interpret this semantics and "execute" it by constructing this image. The model semantics and the visualization algorithm define a calculation whose result is the final image. Therefore, all of these calculations can be combined and compiled (for another more specific and less abstract processor) and executed on a physical (or virtual) machine based on CPU, GPGPU, FPGA, ASIC, Quantum Computer, or any other future hardware.

2. Related Works

In recent years, due to the increasing computational power of computers, more and more geometric modeling systems

are created based on powerful representation schemes. Most of them are research systems, but in the past few years they have inspired several commercial systems that fill some gaps in the market of 3D modeling software. Only visualization the surface of the models or the presence of a constructive tree is no longer enough. For example, in order to materialize a solid with a 3D printer one needs more than its 3D triangle mesh. On the other hand, the classical representation schemes for volume description are often difficult to the user, do not provide powerful algorithms for transformation and editing, have low quality, take up too much memory, etc.

There is much theoretical and applied research that uses powerful representation schemes for volume description, 3D textures, etc. Most of it is based on modifications of implicit surfaces, signed distance functions, and other similar approaches that apply mathematical real functions for describing the volume of solids. The goal is to use these functions to perform all the classic set-theoretic operations and easily to expand the set of operations performed over the solids.

One of the most well-known systems is HyperFun [6]. It is a system based on F-Rep, having its own scene description language (SDL) and scene visualization algorithms. One of the main ideas is to include in F-Rep other representations (homogeneous hybridity) such as Voxels, Implicit surfaces, CSG (by using the so-called R-functions), Sweeping, etc.

Another modern system is Symvol for Rhino [7]. It is a commercial plugin relying on F-Rep as a basic concept for the construction and visualization of scenes based on the description of volumes. A constructive tree whose leaves are F-Rep functions is used, and R-functions are used for the constructive operations. There is also a variety of other operations, bounded blending, non-linear transformations (twist, taper, bend) and others.

Many other commercial plugins for creating FX effects work with internal geometry representations that are more complex than B-Rep, but in most cases they are converted to B-Rep so that the system fits in the host geometric modeling system.

3. F-Rep Basic Concepts

The Function representation schemes (F-Rep) are used for describing geometric objects (solids). F-Rep [5] represents a geometric object by a real continuous function *f* defined in a Euclidean space.

A real continuous function *f*, which describes a solid, is defined by:

$$f: X \to R,\ X \in E^n \tag{1}$$

This function induces a point set

$$S_G = \{X \in E^n \mid f(X) \geq 0\} \ \text{ in } \ E^n \tag{2}$$

A special case of the function *f* is when it gives the signed distance from a point X to the surface of the solid S_G.

Modeling by using such a function is more restrictive, but it also has some advantages (mainly for developing fast visualization algorithms like Sphere tracing [8], [19]).

There are many operations on objects of F-Rep [5]: set-theoretic, blending, offsetting, Cartesian product, metamorphosis, bijective and linear mapping, projection, etc. A fundamental advantage of F-Rep is its openness (extensibility) in view of the possibility for adding new primitives, operations, and relations. Among its big advantages is also the easy implementation of non-linear transformations and other complicated operations.

All geometric operations in F-Rep can be defined analytically. For example, the set-theoretic operations are implemented by using the so-called R-functions [2], [3], [4], [5] (see (3)-(5), for example). The use of the R-functions makes F-Rep more powerful. R-functions [2], [3] are real functions of real variables which inherit some properties of the logical functions (binary or ternary logic). For example, the conjunction is called the logical friend of the R-function

$$f_1 \wedge_a f_2 \equiv \frac{1}{1+a} \cdot \left(f_1 + f_2 - \sqrt{f_1^2 + f_2^2 - 2a \cdot f_1 \cdot f_2} \right) \tag{3}$$

Analogous functions exist for all other set-theoretic operations, for example:

$$f_1 \vee_a f_2 \equiv \frac{1}{1+a} \cdot \left(f_1 + f_2 + \sqrt{f_1^2 + f_2^2 - 2a \cdot f_1 \cdot f_2} \right) \tag{4}$$

$$\neg f \equiv -f \tag{5}$$

In practice we use the special cases $a=1$ ($min(f_1, f_2)$ for conjunction and $max(f_1, f_2)$ for disjunction) and $a=0$ ($f_1 + f_2 - \sqrt{f_1^2 + f_2^2}$ and $f_1 + f_2 + \sqrt{f_1^2 + f_2^2}$, respectively).

The contribution of R-functions to computer graphics, and F-Rep in particular, is the possibility for composing practical arbitrary solids (functions) based on other

simpler and already constructed functions or primitives as spheres, cylinders, cones, etc. In general, F-Rep provides an easy opportunity to incorporate elements from other representation schemes (not only ones from CSG by R-functions, but models can be parameterized easily, etc.) in itself. This means that F-Rep provides an approach for the realization of homogenized hybrid representations [9]. The inclusion of new operations and transformations (including non-linear) is also uniform and smooth.

4. F-Rep Designer 2.0 System

As we have already said, the F-Rep Designer 2.0 is a logical evolution/continuation of our previously described work [1].

4.1 Requirements and Implementation

F-Rep Designer 2.0 system is a prototype of an experimental geometric modeling system based entirely on the F-Rep representation scheme.

As a continuation of the previous work, it has the same goals and meets the same requirements. Geometric modeling system F-Rep Designer must be (and to a high degree is already implemented):

- interactive;
- F-Rep based – using only F-Rep for the scene model description;
- ray tracing based – using only Ray tracing for visualization;
- a hybrid system – using CPU/GPGPU/etc. simultaneously for visualization and other algorithms;
- mesh-free (in model, visualization, and user interaction) – triangle-mesh, B-Rep or similar structures are not used in any system operation stage;
- SDL-free – using neither its own SDL nor other well-known SDL;
- platform independent – this is achieved mainly by using platform-independent programming languages, libraries and systems such as C#, GTK#, Mono/MS.NET, OpenCL, etc.;
- plugin based, open, and extensible – all important subsystems are implemented as plugins that use well-designed programming interfaces. The system maintains lists of items that are registered in it. If necessary, the user chooses which ones to use at a given time.

The system architecture is organized similarly to the one in [9]. It is a system based on plugins and numerous interacting services.

The current realization (see Fig. 1 and Fig. 2) develops and complements the implementation of the previous version of the system. In Fig. 1 we see a model that contains three spheres and one R-function based intersection of two spheres (in the center of screen), and at the bottom right the visualization algorithm can be selected from the available ray tracing algorithms. On Fig. 2 we see a scene with a single sphere and a custom primitive defined by the user by an arithmetic expression.

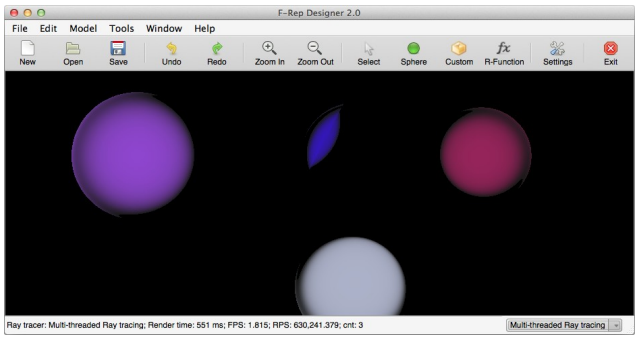

Fig. 1 F-Rep Designer's MVC based GUI

During the implementation we used object-oriented programming and some classic design patterns, such as Composition [17], Strategy [17], and architectural meta-pattern as Model-View-Controller (MVC) [18], etc.

In F-Rep Designer a dialect of F-Rep is used that we called F-Rep*. The difference is that we use $f(X)\leq 0$ for the interior and the surface of the solids (which imposes a similar change of sign when using R-Function, etc.), and we require that the solids be described by a distance function. For example, to describe a sphere we use $x^2+y^2+z^2-r^2\leq 0$. The R-function for intersection is ***max***, instead of ***min*** and vice versa.

The model is currently a composition of real functions of real parameters defined analytically. At this stage a constructive tree is not used on purpose, since the composition of functions carries enough information to construct such a tree, if necessary.

We created a simple application with GUI using GTK# and MVC design pattern. The View part of the system

visualizes the model from the user point of view. Visualization is performed by ray tracing. The calculation of the F-Rep function is performed by compiling functions to IL or by retargeting to OpenCL/CUDA/etc., compiling and executing on CPU/GPGPU/etc. (See Section 4.2 for more details)

An important feature of the implementation is that the visualization of the scene and the user interaction with it occur without polygonization and is fully mesh-free (B-Rep is not used in any form). This is done in order to simplify the system and to avoid possible problems arising from the B-Rep characteristics.

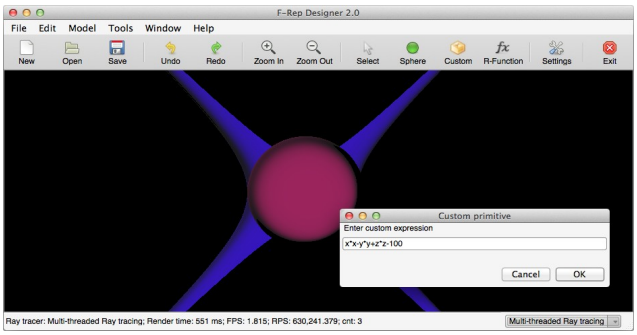

Fig. 2 One sphere and one custom primitive.

User interaction is based on the ray tracing algorithm used for visualization, working in ray casting mode. This allows selection of elements of the model and their transformation, removal, composition of the model elements by means of R-functions, and so on. The user can perform basic navigation through the scene.

4.2 Visualization Approach

The classical scheme of work in geometric modeling systems is shown in Fig. 3. During implementation, the MVC design pattern is often used. The Model describes a scene containing different geometric primitives (and other elements) and their visual characteristics (attributes, shaders, etc.). The final image is obtained by an algorithm called Visualization that crawls the Object space of the model or the Screen space of the image or both, analyzes them, and produces the image that the user sees on an output device (most often the computer system monitor). When the user want to changes some aspect of the image, he/she sends commands to the system (via the computer input devices) and the Controller(s) modifies the Model. This change leads to a new visualization process, and so on.

This classic approach has been proven in practice. It is applicable to different types of models, different visualization algorithms, and for different types of result images. It is very similar to "Interpreter" if we speak in terms of the Programming Languages – the "Interpreter" is the visualization algorithm that interprets/executes the model by applying its semantics in order to obtain the result of execution – the final image. This is done repeatedly. Various acceleration structures and techniques can be applied (most often at the expense of using more memory) to optimize the visualization process. This, however, has many disadvantages (which in case of the F-Rep model are essential). This approach is also useful if the graphic hardware is not programmable (but now most modern graphic hardware are highly programmable). In this case, the bulk of the calculations and algorithms are performed by the CPU, but the trend over the last 10 years is, if possible, to offload these calculations into a specialized graphics hardware.

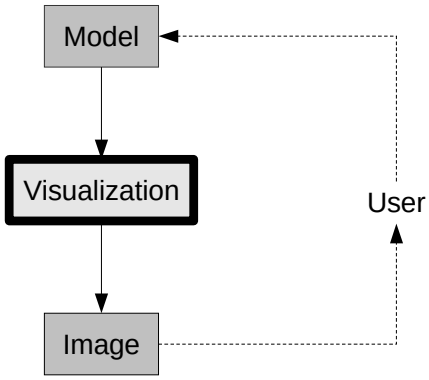

Fig. 3 Classic approach to visualization – an Interpreter.

Here are some arguments that let us to the visualization approach shown in Fig. 4.

First, F-Rep is based on an implicit representation of primitives, R-functions for set-theoretic operations, and others. Not all functions are known in advance and are often input as analytic expressions. This often requires an evaluation of these expressions on runtime. In the first realization of the F-Rep designer, we partially solved this problem by compiling the expressions into an executable code for the respective processor (CPU, GPGPU). Any change to the model, however, leads to a new compilation of the whole model, which in case of larger and more complicated models can lead to a delay and performance problems.

Second, the visual features such as color and other physical features are defined by shaders, which are once again functions. This fits in the previous case (for geometry). Shader features can be compiled for and

executed by a processor (CPU/GPGPU/etc.). They can be further optimized (domain-specific optimization) [10].

Third, the calculation of the various properties required for the visualization algorithms and the shading model, such as normal vectors, can be processed by the F-Rep model (functions). Moreover, these properties may be obtained in advance from the functions describing the geometry by automatic differentiation (AD). This produces a computable function that can be calculated by the graphics hardware or other processor.

Fourth, the visualization algorithm is also an executable code that can be compiled for the corresponding processor (CPU, GPGPU, etc.).

Fifth, combining all elements into one code allows optimization (including global optimization) to be applied to all system components (something that is almost impossible in the classic scheme). The visualization, geometry, shaders and etc., can also be optimized. The merging of the model with the visualization algorithm allows for optimizations on both of them simultaneously, for example, inlining of the geometry functions in the visualization algorithm, etc.

Sixth, acceleration structures, if needed, may also be wrapped or implemented in the form of an additional algorithms. This allows them to be easily changed to different target architectures (for better performance).

Seventh, an execution in a heterogeneous environment can be achieved through retargeting of the code. Different parts of the system are targeted to different processors.

Eight, some platforms may have a very different architecture than the CPU and GPGPU, even much different than the CPU and the GPGPU. The memory model may be different, even there might be no memory in the classical sense. The model and complexity of parallelism might be different. For example, we may use FPGAs or ASICs, even Quantum Computers or any other future (currently non-existent) hardware (including one whose architecture is not clarified). These differences in architectures can be overcome in two ways: either we rewrite (by hand) an algorithms specific for each platform, or we make a module that automatically transforms the algorithms into the new architectures. The second approach is more forward-looking (and more suited to research), and so we chose it in the new version of F-Rep Designer. This variant includes the first one, so its choice is not self-aiming.

All this led to the realization of our system with the architecture shown in Fig. 4.

When we changing the model, the visualization algorithm is triggered. The action of the algorithm is as follows:

1. Stage “Generate/Combine” – the F-Rep model (this is not an essential limitation and can be applied to other representations) is transformed into C# functions. Shaders are converted to C# functions. Other functions necessary for the visualization algorithm, such as the private derivatives of the model functions, are generated to calculate the normal vectors at different geometry points. The ray tracing algorithm is generated and combined with the other generated elements. All this is what we call Code-as-a-Model and on it is based whole “Compiler” approach;

2. Stage “Compile” – the C# code model is compiled to an intermediate .NET IL code;

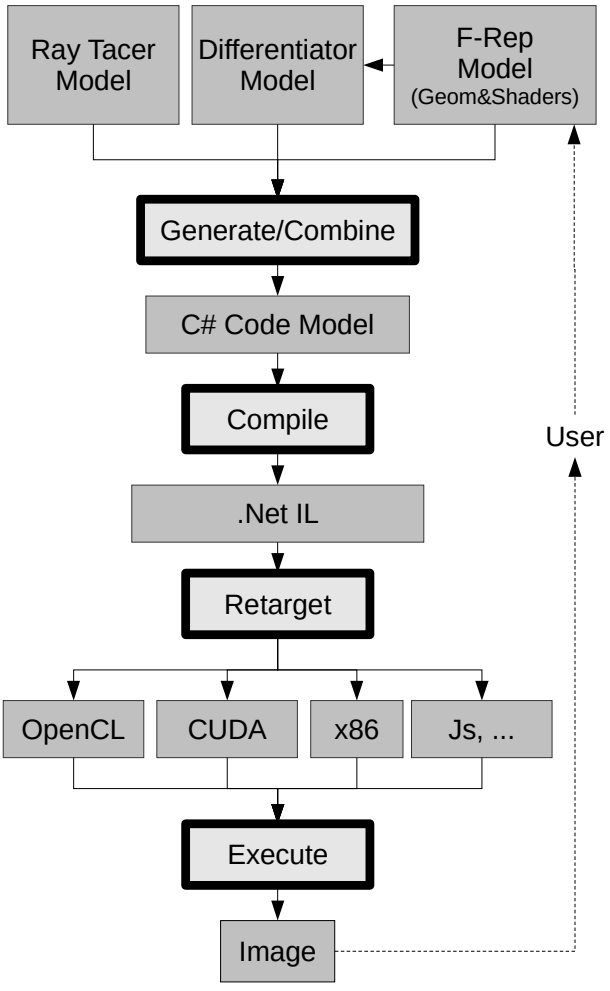

Fig. 4 Our approach to visualization – a Compiler.

3. Stage "Retargeting" – if the target platform is not .NET, the intermediate code is analyzed and is retargeted to the appropriate platform (special classes from plugins are responsible for transferring this code to an equivalent Target Platform code). Retargeting subsystem uses various auxiliary libraries and compilers (SolidOpt [12], CUDAfy.NET [13], Cloo [14], Bridge.NET [15], Cling [11], etc.). This stage may require additional compilation for the target platform by using its optimizing compilers;

4. Stage "Execute" – the native compiled code is executed, which generates the final image.

At any stage, an item may not be modified and may already exist (in cache) from a previous visualization. In this case, it is used by the cache.

In case of user commands, the parts of the system that change the F-Rep model are executed. Besides the other models that are generated by the F-Rep model are changed. This is done through special incremental controllers that only change parts of the compiled models at all stages of the visualization. For example, if we change only the color (or shader of an element of model/scene) it is not necessary to regenerate and recompile the entire scene. Only functions dependent on the modified features are regenerated. The remaining parts are kept the same and if necessary the caches of different levels are used. Also, if we add a new element to the scene – the incremental controller only adds a newly-generated function (if they do not yet exist), and an eventually existing incremental compiler recompile the required code. This is not yet available on all target platforms – for example Cling is an interactive compiler, but OpenCL platforms do not yet have similar tools. In this case, it is possible to work directly with the SPIR [16] representation, but this complicates the realization much more and at this stage it is envisaged that this opportunity will be experimented and realized in the future.

The retargeting subsystem offers an easy way to add new target platforms. The creation of transforming/retargeting algorithms is not difficult, because the generated code does not contain platform-dependent parts, i.e. we use only arithmetic expressions, simple data structures, and base control-flow statements. However, in general, creating a maximally effective mapping between two platforms is not a trivial task.

5. Results

The main result was the realization of a prototype of the system F-Rep Designer 2.0, which has the wanted characteristics and meets the our requirements. This proves that the proposed approach is possible and has some features and benefits that deserve further analysis, research and development.

The main advantages of new system are: F-Rep based model; Mesh-free implementation; Hybrid CPU/GPU/etc. implementation (many target platforms are supported by retargeting using simple transformation plugins).

A central feature of the implementation is that it uses the approach in which are combined and compiled the model (F-Rep, geometry), the functions for calculating the normal vectors, the shading model, and the ray tracer algorithm. Once this is done retargeting to OpenCL, CUDA, x86, etc. and from there to the executable code. This provides some significant advantages. For now, the main difficulty for this approach is that still there are some limitations in the use of OpenCL/CUDA due to the large generated programs, the slow compilation, etc.

6. Conclusions

F-Rep Designer is a system with high potential for research and applications. Some of the problems and weaknesses in the F-Rep Designer prototype that we became aware of are in the process of removal, but others still require the development and improvement of other systems (such as OpenCL, CUDA, etc.) and are not a subject of this study.

The Code-as-a-Model and Compilation approach gives great flexibility, expansion and power to the geometry modeling system. The F-Rep provides users with great expressive power.

Even though we have achieved some important results, for the development and application of F-Rep Designer there is still much to be done. In future we plan to extend and develop F-Rep Designer in several ways:

- Expansion of the retargeting subsystem with the new platforms (FPGA and others); improving existing plugins in order to make better use of the appropriate platforms/architectures;

- Expansion of ray tracing plugins and adding new visualization capabilities. Adding "preview" and/ or "progressive view" options to all available visualization algorithms to maximize interactivity while using the system;

- Add an incremental compilation for all platforms.

Acknowledgment

This work was partially supported by project FP17-FMI-008 of the Scientific Research Fund at University of Plovdiv “Paisii Hilendarski”.

References

[1] A. Penev, "F-Rep Designer", in International Conference "From DeLC to Velspace", 2014, pp. 239-247.
[2] V. Rvachev, Geometrical application of logic algebra, Kiev, Technika, 1967. (in Russian)
[3] V. Rvachev, R-Function theory and its applications, Kiev, Naukova Dumka, 1982. (in Russian)
[4] V. Shapiro, "Theory of R-functions and applications: a primer", Computer Science Technical Reports, TR91-1219, Cornell University, New York, 1991.
[5] A. Pasko, and V. Adzhiev, and A. Sourin, and V. Savchenko, "Function representation in geometric modeling: concepts, implementation and applications", The Visual Computer, Vol. 11, Issue 8,1995, pp. 429-446, doi:10.1007/BF02464333.
[6] V. Adzhiev, and R. Cartwright, and E. Fausett, and A. Ossipov, and A. Pasko, and V. Savchenko, "Hyperfun project: a framework for collaborative multidimensional f-rep modeling", Implicit Surfaces, Vol. 99, 1999, pp. 59-69.
[7] Uformia, "Symvol for Rhino", http://www.uformiaworld.com/products/symvol-for-rhino, (visited 15 June 2018).
[8] A. Penev, "One approach to describe geometric information", M.S. thesis, Faculty of Mathematics and Informatics, Plovdiv university “Paisii Hilendarski”, Plovdiv, Bulgaria, 1996. (In Bulgarian).
[9] A. Penev, "Open hybrid systems for geometric modelling", Ph.D. thesis, Faculty of Mathematics and Informatics, Plovdiv university “Paisii Hilendarski”, Plovdiv, Bulgaria, 2013. (In Bulgarian).
[10] H. Lesev, "Optimizing shading process in photorealistic graphic systems", in Proceedings of International scientific conference Informatics in the scientific knowledge, 2010, pp. 211-219.
[11] V G. Vassilev, and Ph. Canal, and A. Naumann, and P. Russo, "Cling – The New Interactive Interpreter for ROOT 6", Journal of Physics Conference Series 396(5), Volume: 396, 2012, DOI: 10.1088/1742-6596/396/5/052071.
[12] V. G. Vassilev, and A. Penev, and M. Vassilev, "SolidOpt – A Multi-Model Software Optimization Framework", International Journal of Computer Science Issues, Vol. 12, Issue 2, 2015, pp. 32-41.
[13] CUDAfy, "CUDAfy.NET", http://www.hybriddsp.com, (visited 20 April 2016).
[14] Cloo, "Cloo", https://sourceforge.net/projects/cloo/, (visited 15 June 2018).
[15] Object.NET Inc., "Bridge.NET", https://bridge.net, (visited 15 June 2018).
[16] Kronos Group Inc., "The SPIR Specification", https://www.khronos.org/registry/SPIR/specs/spir_spec-2.0.pdf, (visited 15 June 2018).
[17] E. Gamma, and R. Helm, and R. Johnson, and J. Vlissides, "Design patterns: elements of reusable object-oriented software", Boston MA, Addison-Wesley Longman Publishing Co. Inc., 1995.
[18] G. Krasner, and S. Pope, "A cookbook for using the model-view controller user interface paradigm in Smalltalk-80", Journal of Object-Oriented Programming, August/September, 1988, pp. 26-49.
[19] J. Hart, "Sphere tracing: a geometric method for the antialiased ray tracing of implicit surfaces", The Visual Computer, Vol. 12, Issue 10, 1996, pp. 527–545.

Alexander Penev received his PhD degree in Computer Science at the University of Plovdiv “Paisii Hilendarski”, Bulgaria. In 1996, he completed his MSc degree in “Mathematics – specialization Informatics” at the same institution. Alexander has over 20 years work experience at University of Plovdiv as an assistant professor. Currently, his research interests are in the area of computer graphics, visual programming languages design and implementation, software optimization, and others.