Parallel Object-Oriented Modeling and Visualization in OpenMV Environment

Victor Ivannikov, Sergei Morozov, Vitaly Semenov, Oleg Tarlapan

Institute for System Programming of the Russian Academy of Sciences (ISP RAS)

Moscow, Russia

Reinhard Rasche, Thomas Jung

Research Institute for Computer Architecture and Software Technology (GMD FIRST)

Berlin, Germany

Abstract

OpenMV (Open Modeler&Visualizer) is a programming environment intended for development of a wide range of applications, such as geometry modeling, simulation, computational mathematics, scientific visualization, computer graphics. Developed applications have a common open architecture that includes an object-oriented kernel being invariant with respect to various areas and problems, unified graphic user interface and class libraries specific for considered application areas. Functionality of an OpenMV application is determined mainly by completeness of included libraries and semantics of their classes.

The object-oriented kernel supports representation of final graphic scene as a composition of connected typed distributed data and algorithms and provides uniform mechanisms for composing complex scenes and scenarios from separate instances of library classes as well as for their serial and parallel interpretation.

Combining "entity-relationship" paradigm and an original object-oriented approach to modeling and visualization, OpenMV offers more flexibility, extensibility and reusability than traditional visualization and animation systems based on data flow paradigm and allows users to develop complex integrated parallel applications for essentially different areas on the same conceptual, methodological, instrumental and programming basis.

The research was funded by Russian Foundation of Basic Research (98–01–00321), German Ministry of Research and Technology (01 IR 701 — VolVis) and INTAS (96–0778).

***Keywords**: Mathematical modeling, Scientific visualization, Object-oriented programming, Parallel computing, CAD/CAM/CAE systems.*

1. INTRODUCTION

Currently mathematical modeling and visualization play central role in the analysis of complex phenomena and are successively applied in both science and industry. Development of modeling and visualization applications, such as CAD/CAM/CAE systems, was always a difficult task connected with integrating different-purpose components, providing a wide functionality, implementing convenient user interface, porting into different parallel computer platforms. Some of these problems have been partially resolved in visualization and animation systems utilizing open modular architectures and exploiting data flow paradigm [1]. Applicability of object-oriented paradigm to development of reusable visualization and animation software has been also actively examined and a lot of promising approaches and architectures have been proposed and probed [2].

Indeed, using an open object-oriented system, we hope that it may be relatively easy expended by appropriate set of specific classes and, thus, may be suited for considered application areas, particular problems and technical requirements. For example, we expect that complete CAD/CAM/CAE applications integrating needed components for geometry modeling, physical simulation, mathematical problem solving, visualization, rendering may be developed by this way. Nevertheless, we foresee serious obstacles to use many widespread systems for creation of complex applications mainly because of not sufficient generality of architectures and flexibility of mechanisms by means of objects may interact with each other. Mentioned aspects have principal value for integrating different-purpose components as well as for their parallel implementation within the same application imposing strong requirements upon uniform representation of different kinds of objects, general mechanisms for their interaction, common rules of manipulation by them. For scientific visualization applications these aspects concern many actual problems connected with computational steering, collaboration, high-performance computing [3].

Main drawbacks of data flow paradigm usually applied in visualization and animation systems are inter-connectivity mechanisms specifying rules for composition of separate objects into a scene and ways by means of which objects interact with each other while the scene is constructed. This paradigm predetermines serial composition of techniques applied to intermediate data and excludes the other possible ways of object's interaction. Nevertheless, such capabilities may be useful enough for composing and interpreting

complex scenarios encountered in meaningful particular applications and, thus, may provide needed generality for customizing open system to specific problems. It is essential that semantics of objects and their interactions in applications to be developed may not be priory known and the system architecture should provide uniform inter-connectivity and interaction mechanisms without any concretizing implemented data and techniques.

In our research we follow to "entity-relationship" paradigm that seems to be more natural and complete to suit generality and flexibility requirements imposed by integrating goals. Extending traditional data flow paradigm, it permits more sophisticated object's interaction schemes to be important for some topics of geometry modeling, computational mathematics, scientific visualization considered in the paper. The paradigm is successively combined with object-oriented approach within OpenMV architecture to develop complex integrated parallel applications for modeling and visualization.

In section 2 we present object analysis and design for modeling and visualization applications and describe underlying principles of the OpenMV kernel. Section 3 is devoted to some aspects of unified development of integrated parallel applications. In section 4 an example of complex modeling and visualization parallel scenario is considered to illustrate achieved advantages. In conclusions we address to more detailed information about current status of OpenMV and outdraw area of potential applications.

2. AN OBJECT-ORIENTED KERNEL FOR MODELING AND VISUALIZATION

We think final graphic scene of an application as a composition of connected typed data taking part in all processes of application, including modeling, visualization and rendering, as well as algorithms constructing, transforming, deleting these data and realizing mentioned above processes. We distinguish passive data-objects that control only own behavior and active algorithm-objects that can govern behavior of other objects through message passing in classic object-oriented style. Construction of final scene involves defining instances of data and algorithm classes, setting relationships between them, composing scenario from separate data and algorithm objects and its interpretation. An approach based on subdivision of active and passive objects reproduces to some extent Bailin's methodology known as object-oriented requirement specification [4] and has been successively applied to development of mathematical software [5].

Consider an object-oriented kernel of OpenMV environment in more details.

Basic abstract class is ***Object*** that expresses arbitrary data and algorithms taking part in modeling and visualization. Objects may be read, written in/out files, created, connected, transformed, viewed, copied, distributed over processes and deleted as application runs. To manipulate uniformly by different kinds of objects and to provide kernel functionality ***Object*** encapsulates identification key, version number, logical displacement in scene (will be explained later) and numbers of processes over which it has been distributed. These attributes are shared by all scene objects, classes of which are derived from ***Object***.

Besides common attributes each concrete object $\boldsymbol{obj \in Object}$ has own set of attributes defining its internal state and behavior as well as a set of typed links. Links are external ports of objects by means of which they may connect with other ones. A type of separate link $\boldsymbol{Link \subseteq Object}$ predetermines potential capability of the object ***obj*** to connect with any other ones $\boldsymbol{lobj \in LinkObject}$, type of which satisfies to link type or, by another words, is its subtype $\boldsymbol{LinkObject \subseteq Link}$.

Availability of connections in a scene means functional dependence of its objects and, consequently, necessity of their joint consideration and analysis. Being set each connection defines usage relations between a main object having a link and auxiliary objects involving to it. We consider single and multiple connections. Single connection defines one usage relation between pair of objects considered as main and auxiliary ones. Multiple connection should be used when one main object interacts with a subset of auxiliary objects of the same generic type through one link $\boldsymbol{lobj\_i \in LinkObject\_i \subseteq Link}$, $i = 1,...,n,$. The number of objects n involved into multiple connection may be arbitrary and depends only on particular scenario realized in an application.

To differ ways by which objects may interact, all links and connections corresponding them are classified as input, output and mixed (input/output). Having links and participating in connections, a main object uses data and methods of auxiliary objects and, therefore, may change states of connected objects. It is suggested that the main object is capable to change states of auxiliary objects via output and mixed links, and is not capable to influence on input objects. The main object depends only on input objects and mixed objects and does not depend on output objects. The main object and auxiliary objects involved into mixed connections are mutually dependent. Thus, described dependence relations based on classification of links of interacting objects may be established.

To develop parallel applications we need to refine the concept of object from implementation viewpoint. We differ objects that may be located only at any of the processes initiated by the application and objects that may be distributed over a set of the processes. Local objects are implemented by traditional methods of sequential

programming. An implementation of global objects is based essentially on parallel programming. For brief we omit details connected with possible techniques for data partition and consider that each global object may be scattered and gathered in accordance with some generic rule specifying a subset of processes over which the object to be distributed and a way of its geometric, physical, logic or any else partition.

Encapsulating described behavior and properties of objects, the class ***Object*** specifies the following groups of methods common for its concrete instances:

- creation (construct, destroy, copy; read, write in/out file),
- identification (identify an object, its class, parent class, version; verify whether the object belongs to given type),
- inter-connectivity (get number of links, their classes, types; connect objects),
- parallel processing (transmit (send, receive) a local object, distribute a global object),
- scene information (get parameters of a scene, logical arrangement of an object in a scene).

The basic OpenMV concepts are also data-objects ***dat*** $\in$ ***Data*** $\subset$ ***Object*** and algorithm-objects ***alg*** $\in$ ***Algorithm*** $\subset$ ***Object***. Abstract classes ***Data***, ***Algorithm*** are derived from ***Object*** and inherit its behavior and properties. The class ***Data*** expresses entity of various data encountered in applications of modeling and visualization. Its instances may control only own behavior and, therefore, may have only input links. The class ***Algorithm*** represents various algorithms, transforms, operations, auxiliary utilities realizing all processes in modeling and visualization applications. The algorithms are active objects that control both own behavior and behavior of auxiliary objects (not only data) connected via their links. The algorithms may not have input links, but necessarily have output and/or mixed links.

Essential distinction of algorithms consists in their activity that is realized when appropriate events occur in a scene. In these cases the scene activates appropriate method for running algorithms. It is suggested that all connections of algorithms have been preliminary set to activation moment. In opposite case the algorithm is considered as data-object and is not activated to eliminate possible error situations. While an algorithm is running, it sends messages to connected objects to get states of input and mixed objects, to perform needed operations over them, to update mixed objects and to construct output objects.

Figure 1 gives an example of object interaction in OpenMV. The example illustrates how typed data and algorithms may be connected and interacted via typed links. A hierarchy of classes specifies inheritance relations between classes of data and algorithms. A scene diagram specifies a particular scheme of connecting and interacting objects. In the scene diagram data and algorithm instances are shown as ovals and rectangles correspondingly. Links of the objects are marked by points. Connections between objects are shown as arrows.

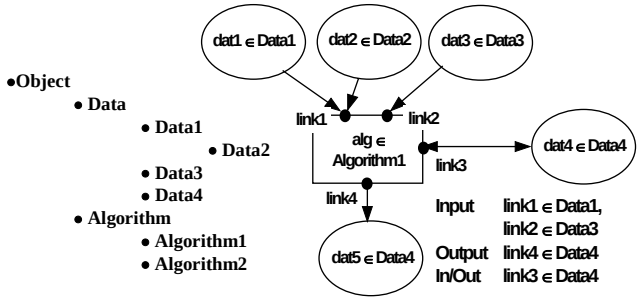

Figure 1: Class hierarchy and interaction of typed objects in OpenMV

Note that the accepted way to direct arrows corresponds to data flows in the scenario. This circumstance reveals similarity with traditional data flow diagram widely used in visualization and animation systems. Nevertheless, the scenario diagram expresses more general paradigm "entity—relationship" that, to our opinion, is more preferable in view of capabilities to specify more sophisticated kinds of object interaction. The visualization problem discussed below gives an example of data interaction that could not be represented by data flow diagram but it is naturally specified in terms of our approach.

Finally, class ***Scene*** is a container class supporting ordered representation (composition) of all data and algorithm objects inserted in a scene and providing a wide functionality needed for developed applications. A resulting picture is constructed by intermediate manipulating scene objects and by activating a conveyor consisted of separate algorithms of the scene. Algorithms of the scene conveyor are preliminary ordered and, then, are activated sequentially or in parallel to solve partial modeling and visualization problems. As the algorithms are being performed, objects of the scene interact with each other, which results to construction of new objects and updating existing objects.

Instances of the class ***Scene*** are used for representation of scenes as compositions of data as well as for specification and interpretation of complex modeling and visualization scenarios represented as compositions of data and algorithms. The concept of scene envelops both data and algorithms because iterative character of construction of a final picture generated usually by means of gradual correcting data properties, refining set of applied modeling and visualization techniques, adjusting their parameters, setting convenient views. Enumerated actions are very

closely connected with each other and are usually required to reproduce studied phenomenon by more adequate and more expressible way.

Having described representation of a scene, one can modify it until desirable picture will be produced. The same representation may be used for dynamic simulation and animation of obtained results. This capability takes place because a scenario once composed may be over and over again applied to semantically equivalent data sets generated in an application as well as inputted into it from files or received from other processes.

Functionality of the class ***Scene*** is provided by abstracting data and algorithms specific for particular applications to be developed. The class defines following groups of methods:

- creation (registry class of an object, construct an object of given class, update version of an object, copy, delete),
- identification (get an object by identification key, select objects belonging to given type),
- inter-connectivity (connect an object with given auxiliary objects, connect objects automatically),
- scenario interpretation (get physical and real time, iteration number; plan a scenario (rank the scene, arrange its objects), analyze latent objects; run a separate algorithm and whole scenario),
- processing application events.

Discuss some features of manipulating whole scene and interpretation of its corresponding scenario.

To simplify the connecting procedure a special mechanism is provided to connect objects automatically. The mechanism is based on type analysis performed for all objects of a scene and inclusion of acceptable objects into connections marked as automatic. All acceptable objects are included in multiple connections. Only the first selected objects are included in single connections. This technique is very useful in cases when a type of a object predetermines some semantic actions that should be performed automatically. For example, it would be convenient to draw geometric objects automatically just after their construction and inclusion into a scene. In this case the procedure drawing whole scene may be implemented as an algorithm having input multiple link of the geometric object type ***GeometryData*** $\subset$ ***Data***. Every time when a new geometric object is constructed, it is connected to the algorithm and may be automatically drawn without any additional efforts.

Specifying scenario, an user arranges the algorithms in logical order in which the scene should activate them. Because this purpose may be difficult, the scene permits to order algorithms automatically. As classified links of objects correspond to dependence relations arising between objects, the scene is capable to analyze connections between objects, to determine character of their dependence and to arrange them in desirable logical order. Arrangement methods provided by the class ***Scene*** are based on scene ranking. Ranking aims to assign to each object a pair of integer numbers (rank and index) pointing its place in a scenario diagram. Similar diagrams display objects of a scene as geometric primitives connected by arrows and are often used in visualization and animation systems as effective tools for graphic representation of scenarios and their visual programming.

Ranking establishes some logical order at which lower rank objects don't depend on higher rank objects, and conversely, objects having a higher rank are dependent only on lower rank objects. Ranking procedure is easily formalized and implemented. After ranking has been completed, objects having the same rank are indexed to determine their full positions in a scenario diagram.

Capability of a scene to rank and to arrange objects is very important for correct interpretation of a scenario because the algorithms should be sequentially activated in logical order, resulting to passing data through an algorithm conveyor and to generating a final picture. If this order was disturbed, there may occur different kinds of errors connected with attempts to activate algorithms without prepared input data.

If a scenario is executed repeatedly, latent objects may exist at some iterations. Latent objects are the objects whose state is not changed at the current iteration. If input objects of an algorithm have not been changed at the current iteration of a scenario, activation of the algorithm cannot result to changes of output objects and, therefore, has not any meaning. Latency analysis permits to exclude redundant events activating latent algorithms and, thus, to increase efficiency of whole scenario interpretation. A scene accomplishes latency analysis by comparing versions of input objects with their versions stored at the previous iteration. Latency analysis should be performed as a scenario is interpreted.

Finally, the class ***Scene*** provides a method for processing typical events connected with OpenMV functionality. Such events are directly dispatched to appropriate methods of the scene.

The considered object-oriented kernel is general and flexible enough to be used for development of various applications at the same conceptual, methodological, instrumental and programming basis. Indeed, to apply the kernel one should not modify it any time to suit to a particular problem, semantics of specific entities, their relationships, and possible details of program implementation.

Underlying mechanisms of inter-connecting and interaction of objects implemented by the kernel allow to compose and to interpret sophisticated scenarios encountered in real applications without any serious modifications. In most cases creation of an application is reduced to development of an applied class library for representation of used scientific data and for realization of modeling and visualization methods specific for a particular area. Taking into account benefits from object orientation of the kernel, the implementation of applied libraries is significantly simplified through exploiting principles of inheritance and polymorphism in development of data and algorithm classes.

A complete OpenMV application includes the object-oriented kernel, expanded class libraries and an unified graphic user interface. C++ language, graphic library OpenGL and message passing interface MPI have been used as standard implementation tools permitting to port developed applications into different platforms.

The current version of OpenMV runs under UNIX/X Window. The developed interface provides unified dialogs for composing scenarios and their interpretation, scene view windows, menus and toolbars. For clearness and convenience the hierarchy of registered classes, the current scenario diagram, the filtered list of scenario objects are displayed in the dialogs. Edition of separate objects is also performed through an unified dialog allowing the user to set desirable values of their public attributes and to connect objects via links. Performed unification of the interface does not hinder its specialization for particular purposes.

3. A PARALLEL EXTENSION OF THE KERNEL

Let's discuss possibilities of development of parallel applications with usage of the described object-oriented kernel. We follow to practically important scheme of asynchronous parallel computing governed by master (main) process and realized by slave (auxiliary) processes. Performing user's commands, the master composes, plans a scenario, distributes it over processes and manages them as the scenario runs. Performing master commands, the slaves load own partial scenarios and interpret them sequentially in accordance with user's one. Being started partial scenarios allocated on master and slave processes run concurrently. Problems connected with data distribution, communication, synchronization, deadlocking, load balancing are well known [6]. To resolve some of them following communication scheme has been developed and implemented. This scheme is oriented on parallel interpretation of user's scenario composed in usual sequential manner and is intended for high-performance computing at massively parallel systems and workstation clusters.

Composing a scenario, the user assigns to algorithms numbers of processes on which they should run. The master plans (potentially, schedules) the scenario by means of allocating local algorithms on assigned processes and distributing global algorithms over them (in addition to ranking and arrangement applied in sequential processing). In view of the data required for algorithms may be allocated by another way, the master reallocates and redistributes them in accordance with given allocation of algorithms. If objects cannot be reallocated in view of connections with already allocated objects, the master inserts additional transmitting and distribution instructions to partial scenarios to create appropriate versions of the objects on desirable processes. The transmitting instructions are instances of the class ***Transmission*** $\subset$ ***Algorithm*** that realize sending and receiving operations for local objects. The distribution instructions are instances of the class ***Distribution*** $\subset$ ***Algorithm*** that realize redistribution of global objects over assigned processes in accordance with the specified generic rule. All these instructions are interpreted within partial scenarios as usual global algorithms. To guarantee that the applications are free of deadlocks, the instructions are intermediately inserted forward places corresponding to algorithms that need appropriate data. The transmission instructions are always assigned to process pairs matching sending and receiving operations.

In consequence of the reallocation several versions of the same objects may arise in different processes. To maintain compatible representations of both user's and partial scenarios, the master stores virtual versions of all remote objects independently from their real location. A virtual version of the object is an instance of its class storing only most common attributes specified in ***Object***. Usage of virtual versions allows the master to manage simultaneously by different versions of objects avoiding redundant storing all data encapsulated by real versions. Virtual algorithms cannot be activated but are necessarily processed by the master.

Described actions are connected with planning the user's scenario. At this stage the master prepares partial scenarios and load them into slave processes by sending appropriate commands and objects. To run it the master sends commands for interpretation beginning to all slaves and then runs own partial scenario. This sequence of actions is repeated in dynamic modeling and animation.

At each iteration the partial scenarios are performed asynchronously, overlapping computations and communications. The necessary synchronization point is a begin of partial scenarios. Potential synchronization points are transmission instructions, distribution instructions and global algorithms. In dependence on particular parallel programming implementations the potential points may be omitted and should not be specially controlled. For example, transmission instructions synchronize pair of

processes in result of sending and receiving appropriate local objects. Nevertheless, current MPI implementation allows to exclude these points from the scenarios by assigning strict object-connected tags to messages. The master should not synchronize them specially if the appropriate tag control discipline is accomplished within global algorithm implementation.

To simplify development of parallel applications based on the described "master—slave" scheme some extensions of the kernel have been performed. Because of essential differences in ways by which the master and slave processes communicate, special classes ***SceneMaster*** $\subset$ ***Scene*** and ***SceneSlave*** $\subset$ ***Scene*** have been developed to suit to corresponding master and slave applications. These classes are derived from the ***Scene*** inheriting its functionality and, partially, implementation. The class ***SceneMaster*** provides additional methods for transmitting local objects, distributing global objects and redefines methods for planning and interpreting scenarios taking into account needed management of parallel slave processes. The class ***SceneSlave*** provides the only additional method for dispatching master commands. A structure of the master and slave applications as well as the scheme of their communications are shown in figure 2.

An important advantage of the extended object-oriented kernel is that it allows to exploit simultaneously different kinds of parallelism within the same modeling and visualization application.

- Fine-grain parallelism can be realized by usage of parallel computational methods (rendering techniques or something else) and by their implementation as the global kernel algorithms.
- Coarse-grain (task or scenario) parallelism can be exploited by distributing user's scenario over processes and by its parallel interpretation.
- Data flow parallelism can be also realized in cases when the same scenario should be executed repeatedly for semantically equivalent data sets. In particular, a conveyor can be constituted from separate algorithms allocated on different processes. A technique connected with periodic running the same scenario on different processes for different data sets can be also implemented.

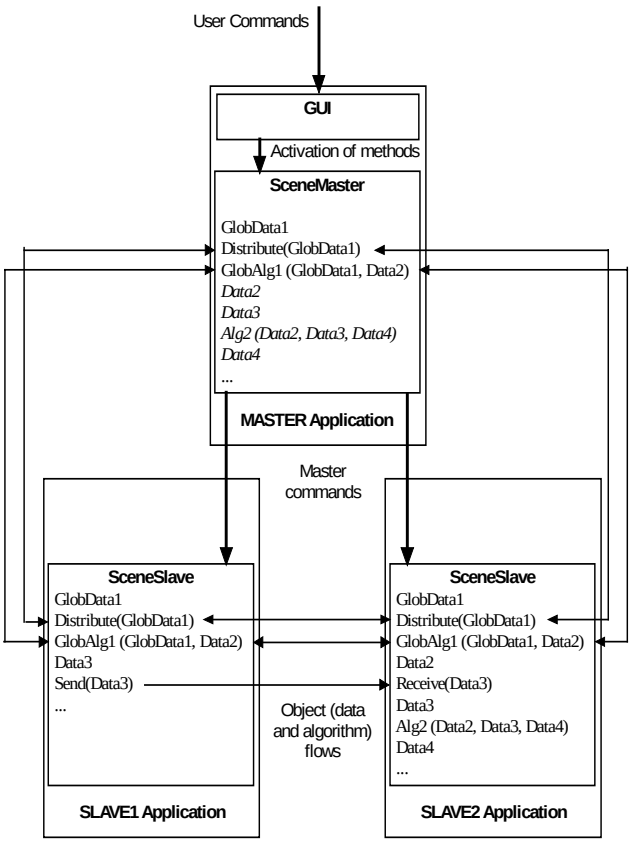

Figure 2: Structure of a parallel application

4. AN EXAMPLE OF PARALLEL COMPLEX SCENARIO

The following example illustrates how a complex parallel scenario integrating different-purpose components can be composed and interpreted within the OpenMV environment extended by applied class libraries. The presented scenario is intended for studying behavior of generator with nonlinear inertia. The model of the generator is a system of ordinary differential equations (ODE) with strange attractor [7].

To study behavior of the model and to determine attractive domain in phase space it is convenient to construct phase trajectories corresponding to different initial conditions of appropriate Cauchy's problem. A set of initial conditions can be derived by composing a complex solid in interesting domain in accordance with constructive solid geometry model (CSG) [8] and by generating vertices on its boundary representation. Then, a set of Cauchy's problems can be numerically solved and obtained results can be visualized by coloring phase trajectories in accordance with derivative magnitudes. A surface corresponding to given magnitude can be also extracted to reveal properties of the ODE function. Isosurface extraction can be performed by different processes in parallel with construction of trajectories. The described parallel modeling and

visualization scenario is given by Figure 4, obtained final image is shown in Figure 3.

Discuss achieved advantages in our approach, restricting ourselves by a fragment of the scenario connected with geometry modeling. CSG model specifies complex solid in terms of set-theoretic operations (union, subtraction, intersection) over elementary solids (box, sphere, cylinder, cone, prism, torus). CSG model can be developed in the scope of the OpenMV environment by natural way. Elementary solids and set-theoretic operations should be considered as passive objects represented by the classes ***CSGElement*** $\subset$ ***CSGData*** $\subset$ ***Data*** and ***CSGOperation*** $\subset$ ***CSGData*** $\subset$ ***Data*** correspondingly. Solids have not links. Operations have pairs of input links by means of which they connect with operands. Because operands may be both geometry elements and operations, the links have type ***CSGData***.

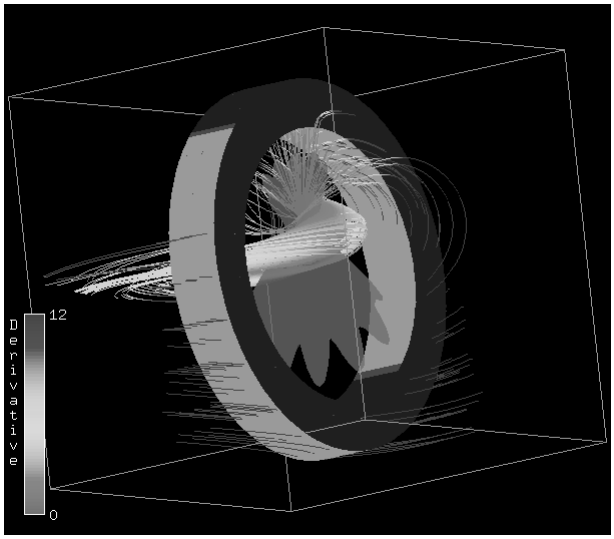

Figure 3: Final image generated in OpenMV environment

Interacting with each other both CSG elements and operations are capable to classify points relatively themselves. For elements such classification is based on simple geometric relations assigned to each element type. For operations the classification is based on Boolean analysis of similar results obtained for each operand. Resulting operation corresponds to complex solid, passes needed functionality for point classification and can be used as a part of more sophisticated scenarios.

Being developed in the scope of the OpenMV environment, the scenario can be flexibly recomposed for particular problems by modifying separate fragments, data and algorithms. The only restriction is that changed objects should satisfy to link typing. This feature guarantees that same scenario can be correctly applied in combination with new data and algorithm types extending developed class libraries. Summarizing achieved capabilities in object-oriented development of applied class libraries, composing and modifying complex parallel scenarios, OpenMV provides a wide reusability for both implemented software components and composed scenarios.

5. CONCLUSION

Thus, general topics connected with parallel object-oriented modeling and visualization in OpenMV environment have been considered. The environment provides general and flexible tools to integrate modeling and visualization components and to develop complex parallel applications for different science and industry areas at the same conceptual, methodological, instrumental and programming basis.

Having being initially developed as general-purpose scientific visualization system, OpenMV offers class libraries for representation of different kinds of scientific data, such as meshes, fields, geometry primitives, solids, scales, palettes, images, and for realization of many visualization and rendering techniques, including methods for extraction of isolines and isosurfaces, construction of streamlines, glyph sets, generating slices, pseudo-coloring. Availability of such libraries simplifies development of numerous applications connected with solving computational mechanics problems. Nevertheless, an area of potential applications of OpenMV is significantly wider and may envelop mathematical modeling, integrated CAD/CAM/CAE systems, VRML technologies, animation systems. Planned future works will be focused on creation of complete applications in closed areas. The examples of applications can be found in [9].

6. REFERENCES

[1] Ribarsky W. et al. Object-Oriented, Dataflow Visualization Systems — A Paradigm Shift? // Proceedings IEEE Visualization ’92, pp. 384–387, October 1992.

[2] Object-Oriented and Mixed Programming Paradigms: new directions in computer graphics / P. Wisskirchen (ed.), Springer, 1996.

[3] Jern M. Information Visualization — Trends in the late 90s // Proceedings GraphiCon’96, Vol. 1, pp. 91–131, Saint-Petersburg, 1996.

[4] Bailin S.C. An Object-Oriented Requirements Specification Method, Comm.ACM, Vol. 32, No. 5, 1989, pp. 608–623.

[5] Semenov V.A. Object systematization and paradigms of computational mathematics // Programming and Computer Software, No. 4, 1997.

[6] Lewis T.G. Foundations of Parallel Programming, IEEE CS Press, 1994.

[7] Anichenko V.S. Complex oscillating in simple systems — Moscow.: Science, 1990.

[8] Requicha A.A.G. and Voelcker H.B., Solid Modeling: Current Status and Research Directions. // IEEE Computer Graphics and Applications, Vol. 3, No. 7, 1983, pp.25–37.

[9] The ISP RAS Scientific Visualization Group Home Page, http://www.ispras.ru/~3D

Authors:

Prof. Victor Ivannikov, Corresponding member of RAS, ISP RAS director

Full Dr. Vitaly Semenov, ISP RAS staff, leading scientist

Dr. Sergei Morozov, ISP RAS staff, research scientist

Dr. Oleg Tarlapan, ISP RAS staff, research scientist

Address:

Institute for System Programming of the Russian Academy of Sciences (ISP RAS)

Bolshaya Kommunisticheskaya st., 25, Moscow, 109004, Russia

e-mail: {ivan,sem,serg,oleg}@ispras.ru

Dr. Reinhard Rasche, GMD FIRST staff

Dr. Thomas Jung, GMD FIRST staff

Address:

Research Institute for Computer Architecture and Software Technology (GMD FIRST)

Rudower chaussee, 5, D–12489, Berlin, Germany

e-mail: {drahnier,tj}@prosun.first.gmd.de

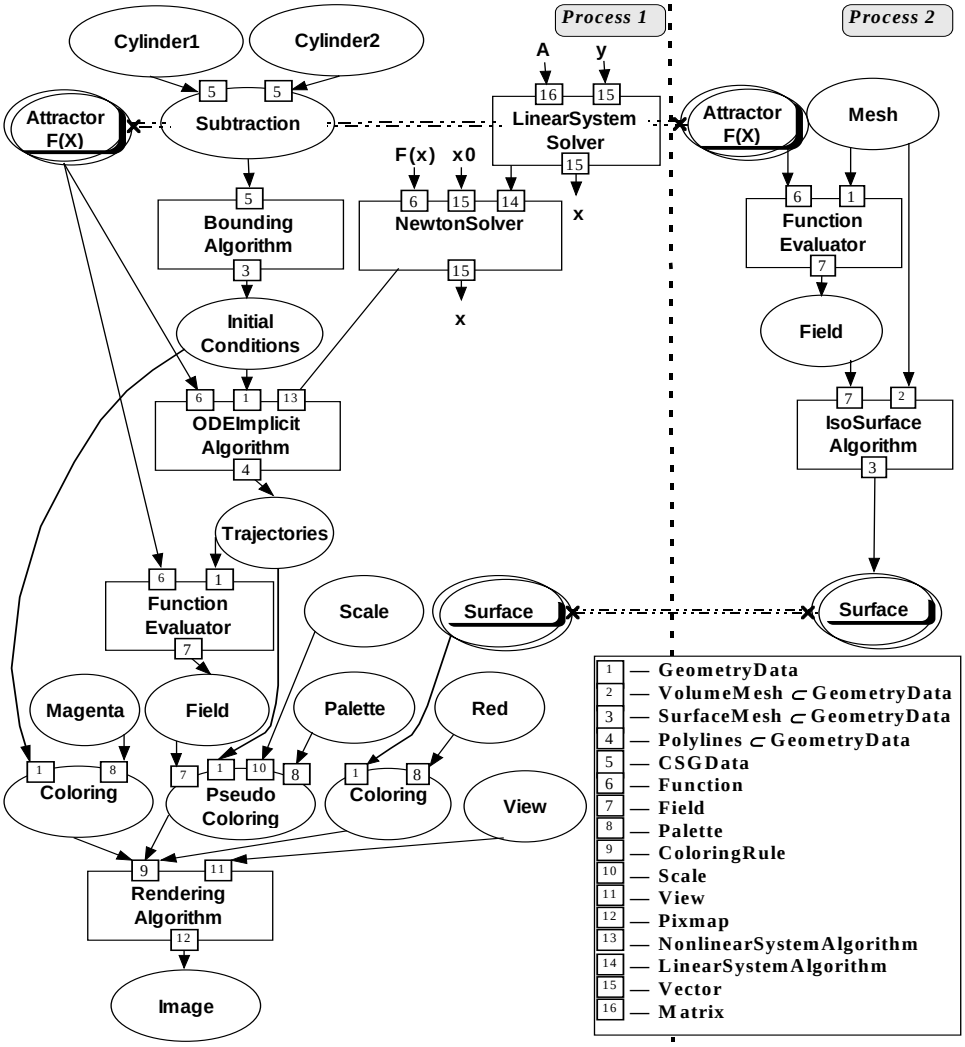

Figure 4: An example of parallel modeling and visualization scenario