Modular Visualisation Environment MVE

Michal Roušal[1], **Václav Skala**[2]
Department of Computer Science and Engineering
University of West Bohemia
Univerzitní 8, Box 314, 306 14 Plzeň
Czech Republic
E-mail: rousal@students.zcu.cz, skala@kiv.zcu.cz

Abstract

The interpretation of various data resulting from simulation and real world is an important of a disparate range of sciences. The field of data visualisation seeks to address this problem by providing appropriate analysis via graphical representations. Various software tools now exist that provide suitable environments for a range of interpretational tasks. Modular Visualisation Environments are one from them. These provide a visual programming environment in which the user can manipulate data and create various graphical representations. A major disadvantages of visualisation via standard MVEs are, that the systems are not easy to use, too complex, huge and very expensive. To address these problems, a new system is presented. This system builds upon recognised benefits of MVEs, simplifying the visualisation, module creation process and introducing a framework that allows complex data (volumetric data in our case, for example) to be easily investigated.

1. Introduction

Data visualisation deals with the exploration and presentation of data, primarily through graphical representation. In the last years the problem of visualising data felt into three main categories of software solutions.

Possible solutions for data visualisation

Graphical Libraries – These range from low level libraries such as OpenGL to higher level examples such as NAG Graphical libraries. A major disadvantage of these solutions is their time requirements, because user has to learn how to use the libraries and need the time for IMPLEMENTATION of data visualisation.

Turnkey Systems - (systems with closed set of modules) Useful for one specific problem. The user is provided with a visualisation solution quickly and easily but their options are subsequently restricted.

Modular Visualisation Environments (MVEs) – these systems fall between the two previous options. They give user a default set of modules. By connecting these modules together the user can visually construct a program to visualise their data. The modules are equivalent to functions in high-level programming language. These systems should allow the user with unforeseen visualisation requirements to write new modules.

Although MVEs are designed to overcome the disadvantages of both graphical libraries and turnkey systems, offering a very flexible environment, learning how to use these products is not simple. Access to these systems for a novice user is as complex as graphical libraries for the non-programmer. The novice user must undergo an extensive period of experimenting with the systems to enable their

[1] This work was supported by the Ministry of Education of the Czech Republic - project VS 97155
[2] Supported by the Academy of Sciences of the Czech Republic project - A2030801

full potential to be realised. Once familiar with operation of MVEs, a further disadvantage is encountered. The learning curve is derived from the bottom-up method employed to create visualisations. The users start their visualisation from nothing, using "small" visualisation modules to build a complex network.
A great advantage of MVEs is their extensibility. User with some knowledge of programming can "easily" create new module to solve a specific problem and add this module to existing set of modules.

Our approach

In any group of programmers, where are people working on wide range of problems, they always need a piece of code one from another one. This situation is also in our computer graphics group.
For example, people need to load volumetric data, generate triangle mesh from this data, reduce the mesh and render it on screen or write the output into a file. This is more complicated, because there is no common programming language, so everyone uses his favourite programming environment (VC++, Delphi, C++ Builder).
To solve this problem we have decided to use some MVE to put all pieces of our work together. Nevertheless all existing systems are very expensive to buy them and programming of new modules is always with some restrictions (programming language etc.). So we concluded to create our own MVE which will be exactly what we need.
Our system is designed for MS Windows platform and uses one of many possibilities how to create modular system under this operating system.
We are using standard DLLs (dynamic load libraries), this solution enables to write modules in any programming environment which can create DLLs for MS Windows operating system.
The system consists of the Editor, representing visual and interactive part of MVE, and runtime part, which manages running and controlling the whole computation. Runtime itself is also integrated to the editor, to enable running designed net of modules directly.

2. MVE

2.1 Modules

It is possible to imagine module like a function in high-level programming language. It can have some data inputs and some data outputs (Figure 1). One special input for each module is its settings. For example: the module for extraction of triangles from volumetric data will have one input "volume data", one output "triangular mesh" and its settings. It can be the density of the material to compute iso-surfaces.

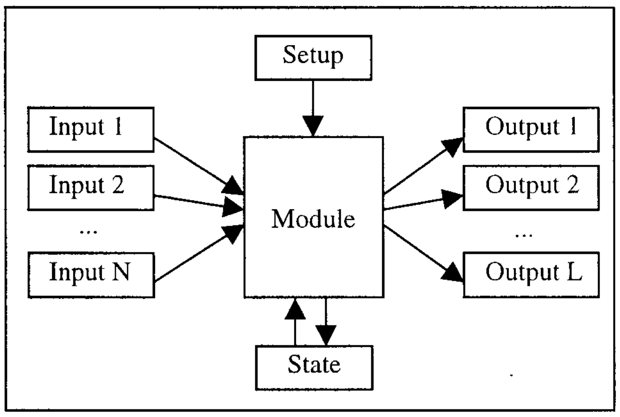


Figure 1

There are many possibilities how to represent modules in MS Windows operating system from simple set of functions across the object representation to usage of COM/DCOM or CORBA standards.
We don't use COM/DCOM or CORBA standards, because they are too complicated, and so unusable for a normal user. In regard of the conditions we have mentioned above (MS Windows platform, no common programming language) we have decided to use set of functions to represent the modules. It is a simple solution, but its open for any further improvements and the users who are not so skilled in programming can implement their modules more easily.
Each module consists of five functions: One function for the executive part of a module

(whole computation or visualisation), one function for the settings of a module and three functions to free memory allocated during the module execution or its set-up data.
All the functions must be named according to defined rules. The names of the functions consist of the name of the module they belong to and the suffix to specify the object of the function.

Module functions
ModulNamer_SETUP_FUNC
ModulNamer_MAIN_MODULE_FUNC
ModulNamer_FREE_SETUP_DATA
ModulNamer_FREE_DATA
ModulNamer_FREE_STATE

Table 1

These functions are named according to Table 1, only *ModuleName* sub-string must be replaced by correct module name.
Functions for freeing memory are important, because if memory is allocated in DLL library it must be freed by function from the same DLL in the MS Windows environment.
In contrast to other MVEs we give users, who are creating modules, full control over the settings of the modules in the setting function. It can be done automatically by analysing the set of module parameters, but we think, that in modern programming environments its is very easy to create any setting dialog.

2.2 Editor

In the editor users can manipulate with modules from the list of available modules, connect them together, set any parameters needed for the proper function of the modules and to create his own application visually. The user can also run his application inside the editor to check if everything is working correctly.
The whole environment is designed to be easy to understand for everyone. Users can simply drag modules on the workspace, connect them together by dragging output of one module to corresponding input of another module and then run this designed application (Figure 2) just by pressing one button. Of course it is possible to save or load the whole net of modules with all the settings.
Adding new modules into the database of the editor is very simple. Users need only to register their DLL library containing the modules. System itself analyses if the selected DLL is usable (if it contains any modules), load all the modules (information about the modules) from the DLL and add them into the list from which users can select them.

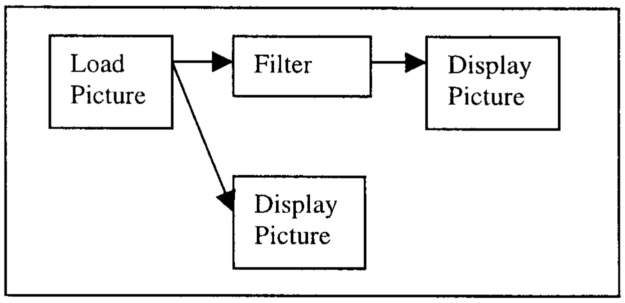


Figure 2

Every DLL library for our MVE must contain the function, which returns the information (name of the module, number and data type of module inputs and outputs), about the modules that it contains. Inside the editor and the runtime part is this information converted into the object representation of modules. This representation is used for any further work with the modules in MVE. At first the object is created for the runtime, then by using this one the object for graphic representation of the module inside the editor.
We have chosen this way to eliminate possible problems (new version of modules but old info file, unusable DLL without info file, etc.) with modules and description of modules stored apart.

2.3 Run-time

This part of MVE is responsible for running modules, transferring data between modules and controlling the whole computation.
Modules can run sequentially or in parallel on one computer. We also thought about distributed computation, but because our MVE was especially composed for visualisation of large data sets like volumetric data, we still have not implemented this. It is complicated to

transfer such large data over the network and it can consume more time then the computation itself on one computer with more processors and fast network usable for this kind of computation is very expensive, currently 100Mb/s Ethernet is used.

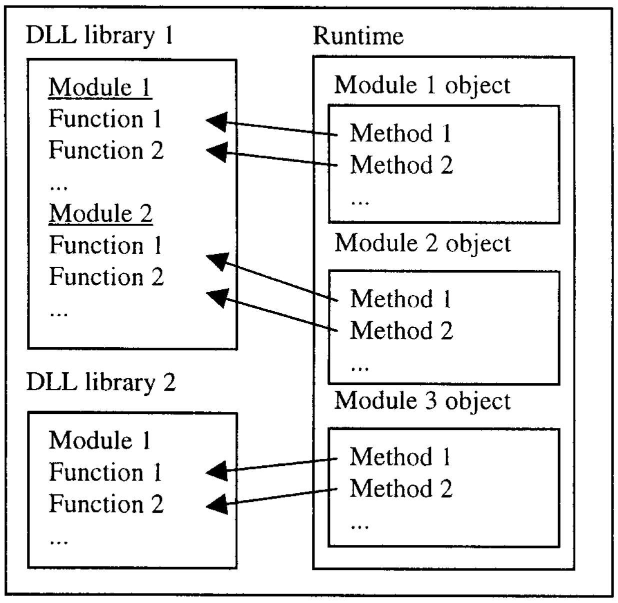


Figure 3

As we have mentioned above all the functions of modules from the DLL library are mapped into the object representation (Figure 3).

The internal representation of modules that are connected into the net, see figure 4, and ready for the execution.

Each module can have only one data on each of its outputs. It means that before the next execution of the designed net of modules all old outputs must be marked as used or already deleted.

For the execution of the designed net users can choose from two possibilities, sequential or parallel execution.

The process of controlling module execution is very simple and can be split to three steps.

In the first step we select modules that can be executed, it means modules with no input data (loaders of data or data generators). We add references to all this modules into a list, which contains all modules that ready for execution.

The second step is different for sequential and parallel execution.

For sequential execution we take all modules from list mentioned before one by one and execute them sequentially. When any module finish, we update list of modules ready to execution. If there is no module ready to execute in the list, the computation process is finished.

For parallel execution we execute all modules that are ready for execution in parallel using threads. In this case is execution of modules a little bit more complicated. For each module in list we create a thread which is used to execute the module. Now we can execute all modules, which are ready, in one time. The operating system automatically distributes the computation of modules to all available processors. Only modules, which are used for data visualisation on the screen of computer (renderers), must be executed in standard way without threads, because of a restriction in MS Windows operating system and MFC[3].

In the last step are freed all data which has been generated during execution of the module net. This can be also done immediately after the execution of the module(s) which were working with that data.

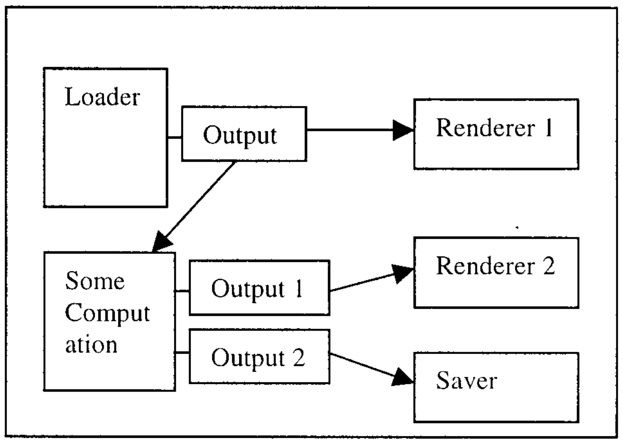


Figure 4

3. Future work

For future we want to improve computation control system to enable some construction from high programming languages (cycle, condition etc.).

The most important thing on which we are working on is an improvement of the MVE into distributed system.

[3] Microsoft Foundation Class Library

Some improvements to the editor will be done to give users more adjustable environment and higher level of interactivity. Some changes in the module section are needed in order to enable automatic update of visualised scene after any change in setting of any module, which affects the visualised data.

4. Conclusion

We have presented the modular visualisation system MVE, which was created for visualisation of large data sets but it usable for any graphics or non-graphics modular application.
Its power is in its simplicity, which gives to all users strong tool for their work in this area of interest. It is useful when more people are working on any project where they need to share some parts of their code.

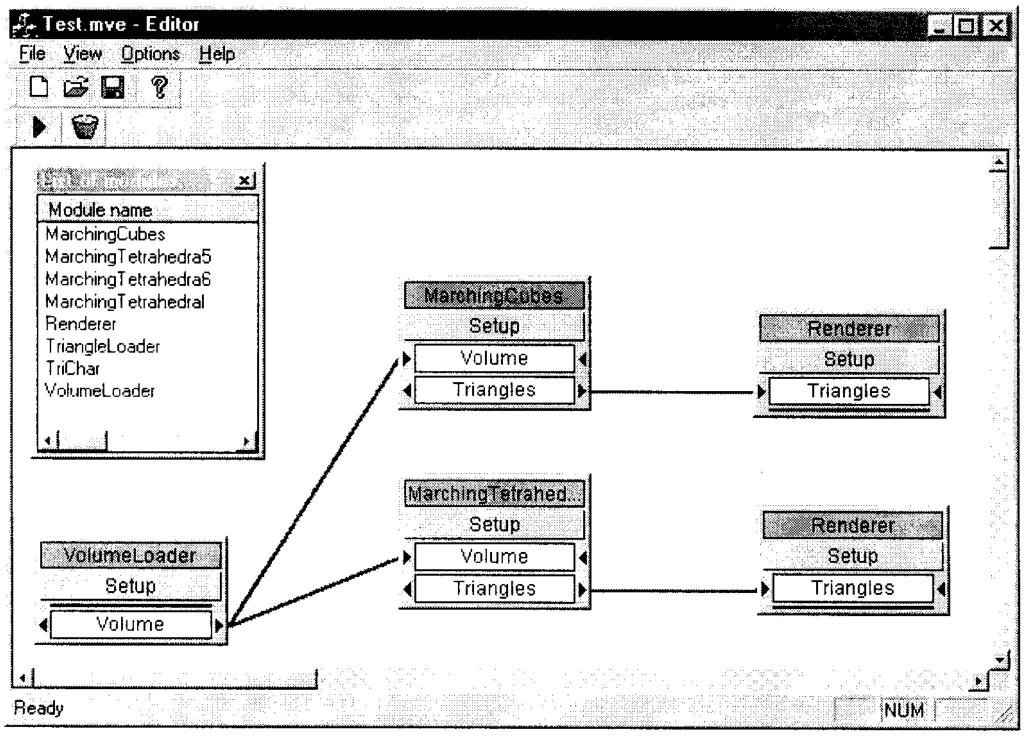
5. Acknowledgement

The authors would like to thanks to all who contributed to this work, especially to MSc. and PhD. students, who prepared others modules and use the MVE, at the University of West Bohemia in Plzen, to all who have stimulated this work and development. Theirs invaluable critical comments and suggestions improved the manuscript significantly.

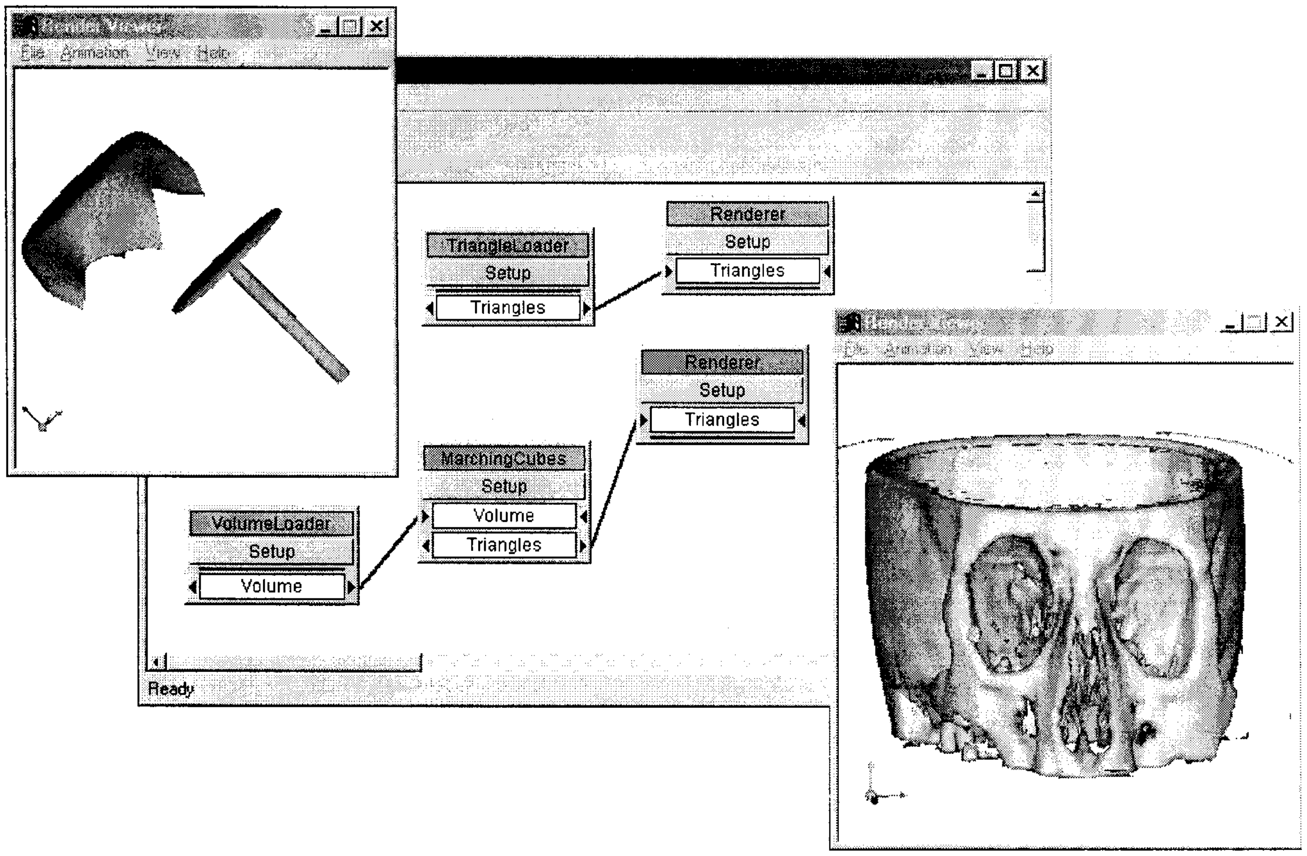
References

[1] Stead,G.A.: Simplifying the Visualization Process, The University of Leeds, 1995
[2] Skala,V.&others: Computer Graphics and Visualization in Parallel and Distributed Environment, Technical Report, Computer Sci.Dept., Univ.of West Bohemia, Plzen, 1999
[3] Chen,M., Kaufmann,A.E., Yagel,R.(Ed.): Volume Graphics, Springer Verlag, 2000
[4] Skala,V.(Ed.): WSCG'99 Int.Conf. on Computer Graphics, Visualization and Interactive Digital Media'99, Plzen, 1999

6. Appendices



A.1. This picture shows the editor with designed net of modules for volume data visualisation with two different methods (MarchingCubes & MarchingTetrahedra)



A.2. How can system look after the execution of modules, if visualised simple triangular mesh and volume data.