MVE-2 Applied in Education Process

Milan FRANK
University of West Bohemia
Univerzitní 8, BOX 314
Pilsen, Czech Republic
mfrank@kiv.zcu.cz

Libor VÁŠA
University of West Bohemia
Univerzitní 8, BOX 314
Pilsen, Czech Republic
lvasa@kiv.zcu.cz

Václav SKALA
University of West Bohemia
Univerzitní 8, BOX 314
Pilsen, Czech Republic
skala@kiv.zcu.cz

ABSTRACT

For years we have been developing a our project on MVE. MVE stands for Modular Visualization Environment. It is a user friendly modular environment using data flow paradigm for communication between user-created modules. The core of the system is based on pure .NET technology.

We find this environment useful in several application areas. This paper focuses on successful employment within the education process. We believe that MVE-2 can be a good entry point for programmers to learn how to develop plugin style software components and to cooperation between them.

This paper also discusses advantages of modern programming techniques available in .NET. We have found several .NET features very useful during design and development of the environment.

Keywords
.NET, MVE, Visualization, Plugin, Data Flow, Pipeline

1. INTRODUCTION

MVE-2 is our grass root effort to create a general and easy to use modular environment. It uses pipeline approach for problem decomposition. This paradigm is useful for both theoretical and practical purposes. Engaging our system in education leads our students naturally to perform clear and well defined problem decomposition as well as to follow good programming habits.

We have used the MVE-2 in the frame of subjects taught at University of West Bohemia (UWB), in separate student projects and as a tool for real research projects. The environment proved itself useful in all the previously mentioned application areas.

The fact that we started the project from scratch allowed us to choose whatever technology available. Our choice of .NET as a core technology provided us with numerous features, which allowed a solid and elegant design of the system.

The rest of the paper is organized as follows: Brief overview of the history of our visualization environment development is given in the following subsection. Section 2 gives basic description of the MVE-2 architecture and its differences from similar systems. Section 3 focuses on the ways students have contributed to the development and expansion of the system. Section 4 describes how MVE-2 is useful for our scientific effort.

1.1 MVE History

The idea of developing a new modular environment at UWB started in 1996 as a diploma thesis of Martin Roušal. This original MVE system was based on the Win32 API. The primary focus on visualization tasks and the choice of development environment lead to several drawbacks of the original design, such as fixed set of datatypes, simple pipeline execution, explicit memory management and problems with components created with different programming tools. This environment was used for several years in both education and research applications.

In year 2004 the Center of Computer Graphics and Data Visualization (CCGDV) has decided to

Project supported by Microsoft Research 2003-178
6FP EU 3DTV No 511568

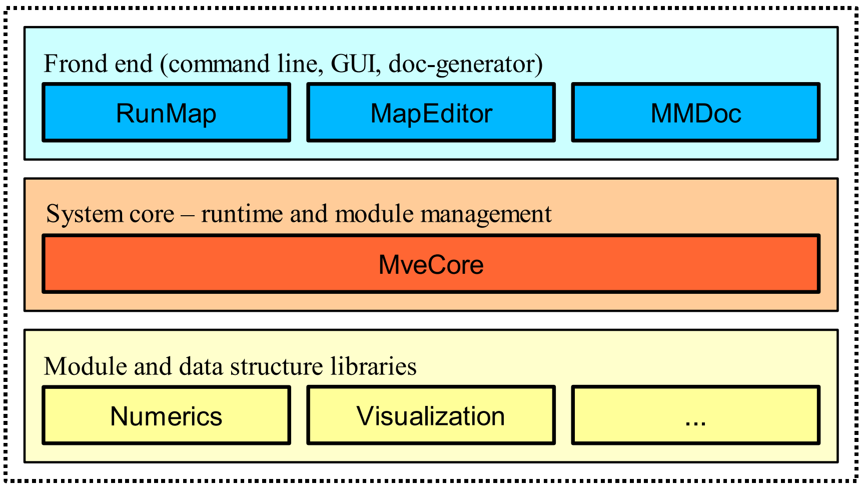

Frond end (command line, GUI, doc-generator)
RunMap
MapEditor
MMDoc
System core – runtime and module management
MveCore
Module and data structure libraries
Numerics
Visualization
...

Figure 1: This figure illustrates a general structure of MVE-project. Each sub-block represent one .NET assembly.

implement a new redesigned version of MVE. Based on extensive experience with the previous version, a new set of requirements has been set. The core of the system was designed and implemented by Milan Frank and his small team of MSc. students with supervision of prof. Václav Skala. The common set of data structures and basic modules for data visualization and computer graphics has been subsequently implemented by the team.

Furthermore, the MVE-2 development spreads beyond the boundaries of the core team as the system was used in subjects provided by the CCGDV for research purposes and in other areas.

2. SPECIFICATION

MVE-2 offers easy-to-use, data-flow based modular environment. Its primary users are researchers and students together with their projects. Our environment makes these projects compatible with each other with minimum additional effort. API (Application Program Interface) of a module is enforcing good programming habits, such as clear problem decomposition, precise comment writing and cooperation with other programmers. Using MVE-2 leads to less routine programming due to compatibility and reusability of existing modules. Therefore, users can concentrate on their particular problem and employ existing modules for marginal tasks.

When we were designing the MVE-2 system, we have attempted to achieve several goals. We wanted to create a well defined and understandable module API with following important features of the whole system:

- general core, ready for modules and data types from different application areas,
- module-maps with support for cycles and sub-branches,
- intuitive and friendly API for modules and data structures,
- automatic generation of basic GUI of a module,
- automatic generation of documentation for module libraries,
- built in XML export/import of all data types and module-maps. (very efficient way to check and modify data manually).

2.1 Core

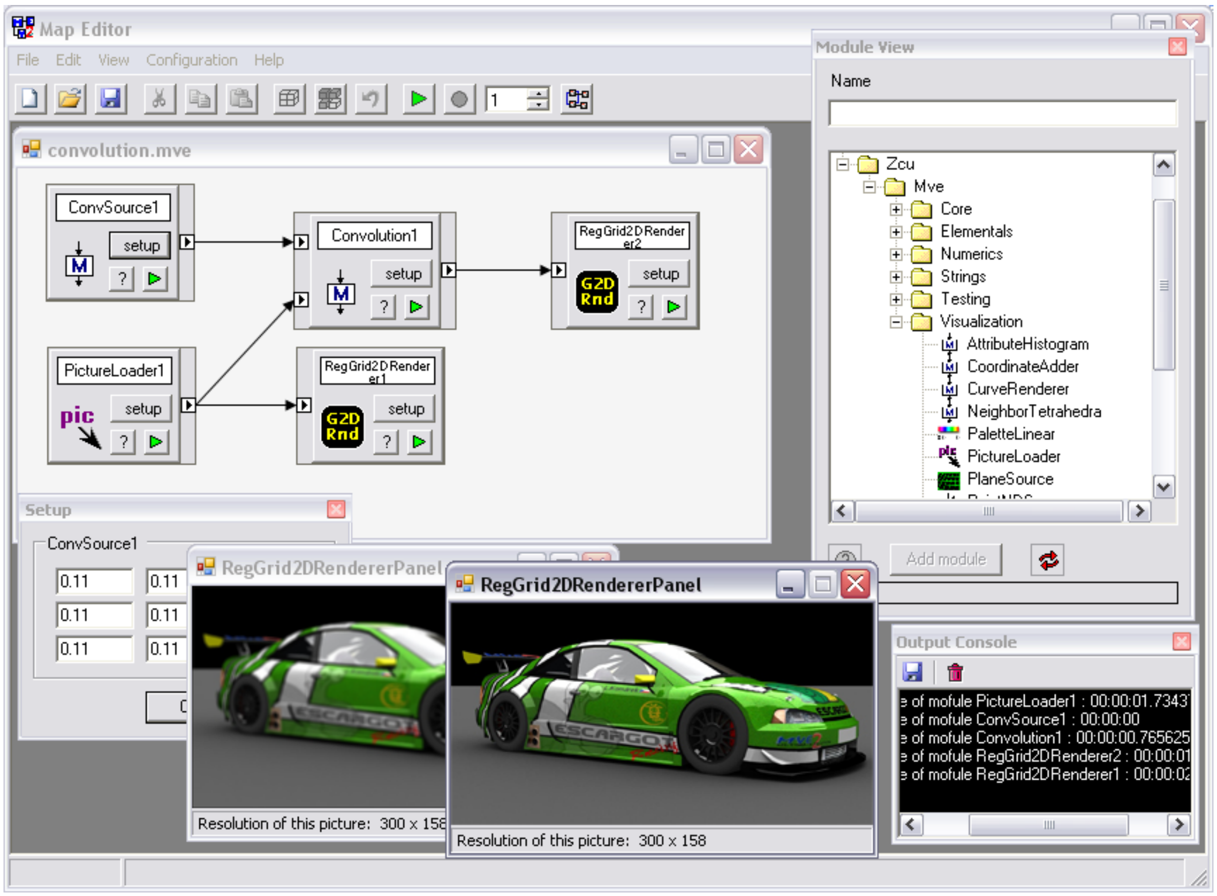
Main part of the environment is MveCore. It provides runtime and module management functionality. Capabilities of the core are accessed by two front-ends, a graphical interface that allows module map composition and execution, Map Editor, and a command line tool for executing existing maps stored in XML files.

One of the main advantages of the system is very simple implementation of a plain module. As well as this, power and high flexibility is available if required. This advantage is especially important for programmers at the beginning of their career. Another specialty of MVE-2 is execution mechanism of module network. Possibilities of module map topology are far beyond simple pipeline and reach closely towards complete visual programming. Many other advantages arise from use of pure managed environment (.NET).

2.2 MVE Front-end

MapEditor is a GUI front end of MVE-2. It allows intuitive module map editing, module configuration and execution. Figure 2 shows a screenshot of the GUI with a simple convolution application.

Located in the upper left corner is a module-map edit window with a simple pipeline using two sources (PictureLoader, ConvolutionMask) and two sinks (RegGrid2DRenderer). In the upper right corner there



ConvSource1

0.11	0.11
0.11	0.11
0.11	0.11

Resolution of this picture: 300 x 158

Resolution of this picture: 300 x 158

```
e of mofule PictureLoader1 : 00:00:01.7343
e of mofule ConvSource1 : 00:00:00
e of mofule Convolution1 : 00:00:00.765625
e of mofule RegGrid2DRenderer2 : 00:00:01
e of mofule RegGrid2DRenderer1 : 00:00:02
```

Figure 2: Screenshot of MapEditor. The GUI front end of MVE2. It shows a simple convolution scheme.

is a ModuleView dialog that contains list of available modules that are ordered according to namespaces. The standard output is redirected to the output console, which usually displays important messages from modules and core. In the example it displays the running times of modules. The two green cars in the center are the original and the filtered image rendered by renderers. In the bottom left corner a setup dialog of convolution source module is shown. User can define the convolution mask via this dialog.

2.3 Module libraries

The core and the front-end only create an empty space for modules. Without modules there is no useful functionality. Modules can be added into the MVE-2 system very easily. It is sufficient to copy an assembly (a .NET DLL) into a particular directory. All public subclasses of the MveCore.Module class are interpreted as modules.

Each assembly can also provide subclasses of MveCore.DataObject. It allows everyone to define their own custom data types that can be passed between modules via connections.

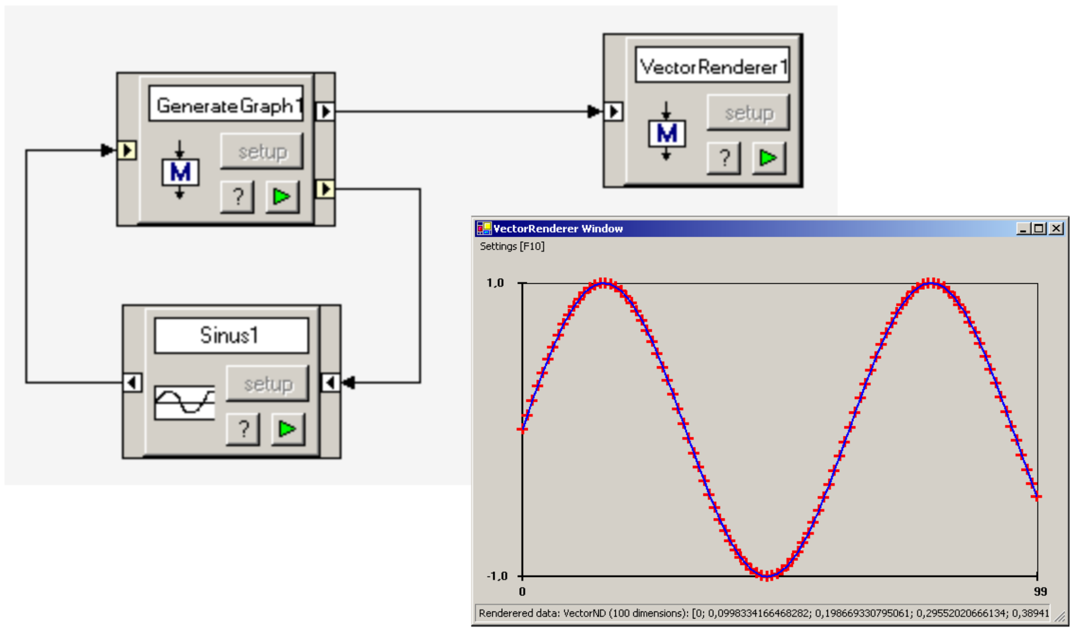
The assembly can contain (or call) any other code allowed by the .NET standard. If one has a project already written in the .NET environment, then it is usually very easy to provide a set of MVE-2 modules as wrappers for functionality of the code. These modules can serve as a "standard" interface and thus can be easily reused by many other researchers and developers.

2.4 Advanced pipeline examples

Execution mechanism implemented in the core can do more than simple pipeline execution. We support repeated execution, module driven execution and cycles. Any map can be run N-times. Module can run a subbranch to obtain its input data multiple times. It is also possible to create cycles in module map graph. See following examples:

Sinus (See Figure 3) is an example of sub-branch construction. Execution of Sinus module is controlled by GenerateGraph module. In this particular case the whole module map runs only once, while the Sinus module runs 100 times.

Counter (See Figure 4) is an example of DelayModule usage. The DelayModule acts as a single place memory with initialization. It returns data form previous (N-1) step. In the first step it returns data from initialization port. Thus it allows cycles in



Renderered data: VectorND (100 dimensions): [0; 0,0998334166468282; 0,198669330795061; 0,29552020666134; 0,38941

Figure 3: Simple example of module driven subbranch execution.

module-map graph. This example counts from zero to number of runs minus one. The DelayModules can be chained.

2.5 Module creation

As mentioned in the begging of the text, creation of a module is very simple. It is based on inheritance mechanism. Every subclass of MveCore.Module is interpreted by the MVE-2 core as a module.

There are only two methods that have to be subject to override. The first one is the constructor, which creates input and output ports and defines their names and accepted data types. The second one is the Execute method that represents the activity of the module.

The standard .NET property mechanism allows the module authors to easily provide configurable parameters of their modules. Every public read/write property of a standard datatype is automatically saved into and restored from the module map XML file, and it is also shown in a module GUI that is automatically generated for each module. These features are provided by the Module superclass, and don't require the user to write a single line of code.

There is also a set of advanced methods that can be called and set of events that can be handled by a module. These additional methods provide a possibility to create a module with advanced features, such as immediate reaction to incoming data, advanced module GUI creation, execution of a subbranch etc. This means good flexibility for a

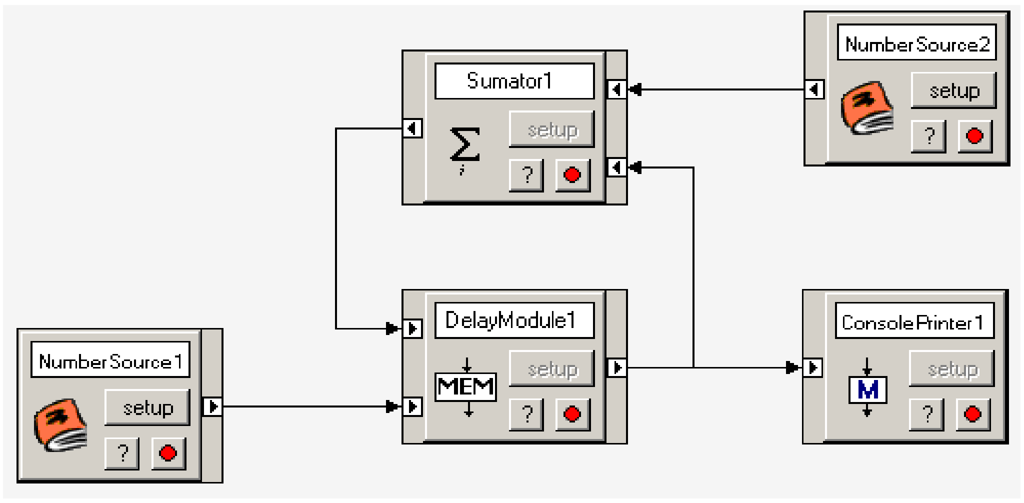


Figure 4: Simple example of loop with delay module.

```
public class Sinus : Zcu.Mve.Core.Module
{
  ScalarNumber y = new ScalarNumber();
  public Sinus()
  {
    AddInPort("in", typeof(ScalarNumber));
    AddOutPort("out", typeof(ScalarNumber));
  }

  public override void Execute()
  {
    ScalarNumber x = (ScalarNumber) GetInput("in");
    y.Val = Math.Sin(x.Val);
    SetOutput("out", y);
  }
}
```

Figure 5: Example of a simple module implementation. Note this is just a class derived from Core.Module class. It is than interpreted by core as a module.

module. Fortunately, in the beginning there is no need to even know about these methods.

Example of implementation of a simple module that calculates sinus of input value follows. Please note this is a complete C# source code one needs to create a MVE2 module. See Figure 5.

Creation of data type is similarly easy, only Core.DataObject is used instead Core.Module.

Documentation of a module library can be generated automatically by the MMDoc utility that is distributed with MVE-2 system. It uses the .NET attribute mechanism to obtain additional information from each module and data type, which describes the module behavior. This information includes description of module ports and configuration properties as well as general description of the task that is performed by the module. This information is also used by the GUI front-end to provide the user basic information about the modules in help dialogs and pop-ups.

3. MVE-2 IN EDUCATION PROCESS

There are two main ways how students get in touch with the system. Many students were asked to create modules to be integrated with the system and with one another. Small group of students was also involved at the development of the system itself and its peripheral tools, such as GUI frond end, automated documentation system, data structures and so on.

3.1 Student Contribution to the Core Development

Several volunteer MSc. and Bc. students were involved in the development of the core of the system. They were cooperating closely with a current project leader. Such involvement gave them feedback about their work and made them familiar with a developing model typical for small software companies. We believe it is a useful experience in a career of a programmer and will help them in seek for future employment.

Miroslav Fuksa was the first volunteer to be involved in the development. His contribution to the execution mechanism was very inspiring. For one developer it is not easy to keep in mind all the possibilities of such a complex algorithm as the module execution mechanism.

A huge contribution has been made by Zdeněk Češka. He is fully responsible for development of the GUI front end. Design of such complex subsystem gave him good practical experience about how to apply theoretical knowledge obtained in subjects of software engineering and knowledge of programming in .NET environment.

The whole of MMDoc subsystem was designed and implemented by Petr Dvořák. He also created a useful GUI front end of this subsystem. Such task made him familiar with several modern technologies such as .NET, XML, XSLT, CHM, etc. He proved himself to be able to apply such technologies in a real world application.

Very important task was to design and implement common data structures for data visualization. Miroslav Vavruška did significant contribution to this essential part of MVE.

Přemek Zítka was responsible for adding a useful feature. Thanks to his effort it is now possible to use automatically generated module GUI (setup dialog).

Using the standard .NET Framework PropertyGrid component it was possible to expose public properties of a module in a simple and elegant way.

3.2 Student Contribution to the Module Library Development

In the frame of Computer Graphics and Data Visualization subject taught at the UWB students were supposed to implement several modules that solve a particular task. These tasks included mesh displacement, elevation coloring, triangle mesh reduction, readers of several triangular formats, etc.

The results of their effort were interesting sets of modules. They were also supposed to provide a detailed documentation for their module libraries.

We believe such task give the student a basic idea about how to write useful pieces of code that can be integrated in some larger systems.

For example: Task chosen by student Jan Bárta was to create a reader and writer module for several standard geometry data files such as PLY, STL, TRI, CMX. Result of his work is clearly very useful and reusable by many other people.

Another nice task was to create a set of modules for generating a displacement mesh from a 2D picture. Jiří Skála took this work very seriously and the result of his effort is an example of what a module library should look like.

Mesh smooth and displace modules were designed and implemented by a team of Ondřej Kvasnička and Martin Pokorný. Their task was to create a set of modules to produce a mesh distorted by the intensity of applied texture. It was necessary to divide this task into several modules. It was interesting to see their feedback about how MVE helped them with the problem decomposition (and following composition) and programming cooperation.

Most of these modules are freely available at the MVE-2 website.

4. MVE-2 IN RESEARCH

As MVE-2 became stable it was employed in a number of real research projects that are carried by CCGDV PhD. students. This lead to benefits for all involved parties, MVE-2 has gained some useful modules, core developers received feedback about the performance of the core and researchers benefited from a easy to use and powerful tool for their projects, which allowed easier collaboration and code sharing.

So far three research topics were addressed using MVE-2: stripification of triangular models by Petr Vaněček, artificial hologram rendering and reconstruction by Martin Janda and Ivo Hanák, and space-time metric for dynamic mesh comparison by Libor Váša.

The first project carried by Petr Vaneček showed some performance drawbacks of the original data structures that were fixed in subsequent versions of MVE-2 by optimization of the visualization structures. Thanks to the efforts of Dr. Vaneček there is a fully functional and thoroughly tested support for triangle stripes and fans in the visualisation library provided with MVE-2.

The second project was the spatio-temporal metric implementation for dynamic mesh comparison by Libor Váša. This effort has benefited greatly from the available range of modules, while some more common functions were added to the visualization library. This allowed wide testing of the proposed algorithms on several kinds of source data, using loaders for various data types, and also the result was visualized using the modules provided by MVE-2. This project will continue using MVE-2 in the future in order to allow international cooperation with foreign universities that participate in providing source data.

The most recent application of MVE-2 in the research field is in the artificial holography research that is carried by Martin Janda and Ivo Hanak. They are developing modules for rendering scenes into artificial holograms and computer reconstruction of holograms. The environment allows this elementary team to cooperate easily, as each researcher works on his own module, while having a clearly defined interface to each other. They have also contributed to the core of the MVE-2 with some minor changes that improved the usability of the system for their specific purposes.

5. FUTURE DEVELOPMENT PLANS

Although the core of the system is not being actively developed anymore, the project is still growing by additions of modules and features into the MapEditor GUI. One of CCGDV MSc. students is currently working on a general rendering module that will utilise the D3DUT [4] for rendering. This rendering library developed also by CCGDV will allow platform independent rendering, which will enable full visualization pipelines on all platforms that allow compilation of the system and D3DUT.

In the near future, we would like to investigate the rewriting an area of the visualisation library so that it will utilize the new features of .NET 2.0. The generic data types of .NET 2.0 can be very useful and can simplify some algorithms quite significantly.

The system will be most likely be used in the data visualisation subject taught at the UWB, where students will contribute, develop and test modules as a part of their coursework. The system will also be used as a target platform for computer graphics related diploma theses.

The environment will be used by the holography and dynamic mesh researchers, who will contribute feedback and new modules to the system. This can give the users of the system the advantage of availability of state of the art algorithms within the environment.

The further in the future, our development plans include allowing parallel and distributed execution of module maps. Module libraries for volume data rendering and computational geometry tasks would also greatly improve the practical usability of the system, and we are currently looking for contributors or student leaders to develop such functionality for MVE-2.

6. CONCLUSION

We have described a modular system that is developed at UWB. Students at all levels of education have contributed to the system, which allowed them to learn a valuable lesson about modular programming.

The system is currently used in number of student and research projects, where the structure of the environment helps to clearly formulate and thus easier solve various kinds of problems.

The system uses .NET environment at its best. It enabled the system designers to implement desirable features, such as editable module properties, in a way that is not matched by any similar system in its elegance and simplicity.

7. ACKNOWLEDGMENTS

The authors of the paper would like to thank all previously mentioned contributors for their effort to improve MVE-2. We also thank to Angharad Savage for her careful spell checking of this paper.

8. REFERENCES

1. Schreder, W., Avila, L., Martin, K., Hoffman, W., Law, C.: *The VTK User's Guide.* Prentice Hall, New Jersey, 2001.
2. Váša, L., Skala, V.: A spatio-temporal metrics for dynamic mesh comparison. Subbmitted to AMDO 2006
3. Frank, M., Váša, L., Skala, V.: Pipeline approach used for recognition of dynamic meshes. Submitted to 3IA Limoges 2006
4. Home pages of D3DUT http://herakles.zcu.cz/research/d3dut/
5. *Home pages of VTK.* http://public.kitware.com/vtk/
6. *Home pages of MVE-2.* http://herakles.zcu.cz/research/mve2/