3AI int.conf. Limoges, pp.219-224, ISBN 2-914256-08-6, France, 2006

Pipeline approach used for recognition of dynamic meshes

Milan Frank[*], *Libor Váša*[†], *Václav Skala*[‡]

(mfrank|lvasa|skala@kiv.zcu.cz)

Abstract

In this paper we discuss an application of our MVE-2 project in a research experiment. The MVE-2 stands for "Modular Visualization Environment version 2". It is our grass root effort to create a general and easy to use pipeline based environment. These days we are in the stage of first real-world research experiments. Following text contains an introduction to the system and its possibilities. Second part illustrates how our environment was used in a nontrivial recognition method research.

1 Introduction and motivation

Pipeline data processing is a very efficient and intuitive way to experiment with data and data processing methods. This approach pays off due to flexibility and straightforward reusability of already existing processing code.

2 MVE-2

Modular Visualization Environment 2 project has been started several years ago. It has been driven by enthusiasm about advantages of pipeline data processing and about modern programming techniques. It allowed us to create general intuitive environment that is friendly for the users as well as for the programmers of modules.

The whole project is based on the .NET technology, a major platform for Windows programming with hi-end programming features (garbage collector, metadata, ...).

Important features of the system include the following**:**

- general core, ready for modules and data types from various application areas,
- module-map that supports cycles and sub-branches,
- intuitive and friendly API for modules and data structures,
- automatic generation of basic module GUI
- automatic generation of module library documentation,
- very simple creation of modules
- 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 the *MveCore*. It provides runtime and module management. Functionality of the core is accessed by a GUI and a command line frond-end.

2.2 Front ends

MapEditor is a GUI front end of MVE-2. It allows intuitive module map editing, module configuration and execution. Screenshot of the GUI with convolution can be seen on Figure 1.

The program window contains an edit window with a simple pipeline using two sources (*PictureLoader*, *ConvolutionMask*) and two sinks (*RegGrid2DRenderer*). A ModuleView dialog contains list modules available, ordered according to namespace. The standard output is

Project supported by Microsoft Research 2003-178[•]

6FP EU 3DTV No 511568[†], MSMT Czech Republic project LC06008[‡]

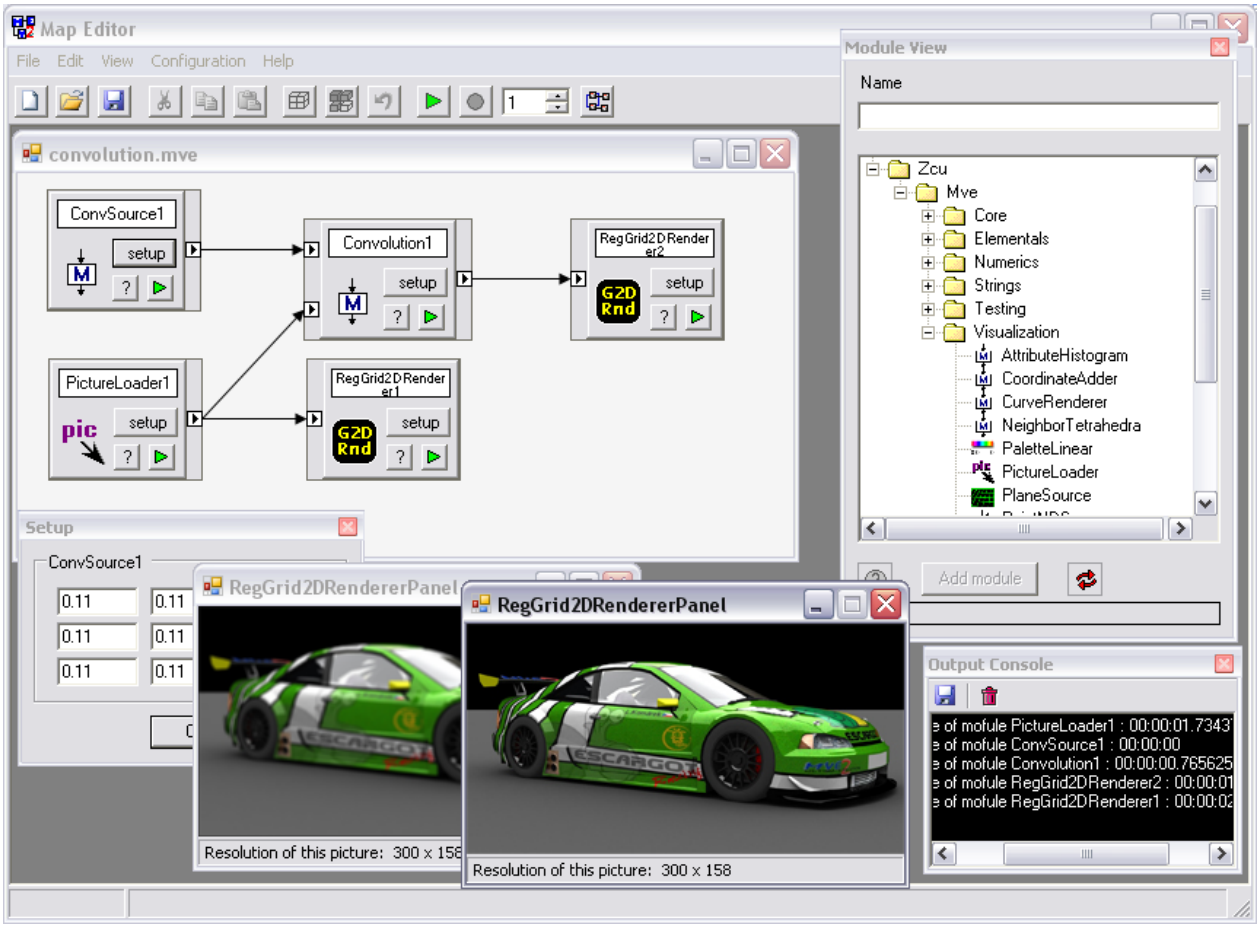


Figure 1: Screenshot of MapEditor. The GUI front end of MVE-2. It shows a simple convolution scheme.

redirected to the output console, which usually displays messages from modules and core. 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 the convolution source module is shown. User can define the convolution mask via this dialog.

The second front end of the system is a simple command line utility called **Runmap**. It allows execution of maps stored in XML files without any user interaction, which is useful for batch tasks.

2.3 Module libraries

The core and front end only creates an empty space for modules. Adding modules to the system is very simple and only requires the user to copy a .NET DLL that contains the modules into a directory where it is found by the core. All public subclasses of the *MveCore.Module* class are interpreted as modules and added into the GUI.

Adding a data structure to be passed between modules is similar, all public subclasses of *MveCore.DataObject* can be passed from one module to another.

2.4 Standard pipeline execution

If not specified otherwise, a standard pipeline execution is performed when a map is started. This involves determining terminal modules and propagating a request for data from the top of the map to the bottom. When all terminal modules have obtained input data and performed their task, then that map run is finished.

2.5 Advanced pipeline examples

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

Sinus is an example of **sub-branch** construction. Execution of Sinus module is controlled by GenerateGraph module. In this particular case the module-map runs only once while the Sinus module runs 100 times. (See Figure 2)

Counter 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, allowing cycles in module-map graph. This example counts from zero to number of runs minus one. The delay modules can be chained. (See Figure 3)

2.6 Module creation

Simple creation of modules is one of the most interesting features of our system. By inheriting a new class from the *MveCore.Module* abstract class, a fully functional module is created.

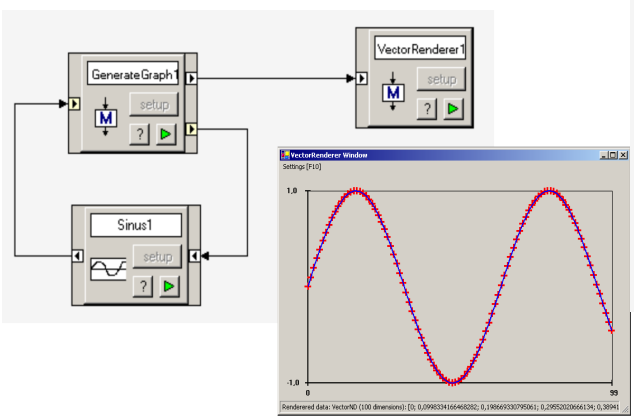
There are only two methods that have to be overridden. The first one is the *constructor*, which creates ports and defines their names and accepted data types. The second one is *Execute* method that represents the activity of module.

We are using features of the .NET system to provide comfort to module authors. For example any public property of a module is automatically displayed in a module setup dialog, and saved/restored with the module map. Adding a user-editable parameter is therefore a matter of exposing it using the property mechanism.

There is a set of advanced methods that can be called and set of events that can be handled by a module. These additional methods make it possible to create a module with advanced features, such as immediate reaction on incoming data, advanced module GUI creation, execution of subbranch etc.

2.7 Documenting MVE-2

Documentation for MVE-2 module libraries can be generated automatically using the MMDOC utility that is part of the system. It uses attribute classes and comments that describe modules, ports and data types, and generates electronic documentation in a number of formats (html, chm, ...). It can be also used to generate a list of uncommented entities (methods, modules etc.), thus enforcing careful commenting.



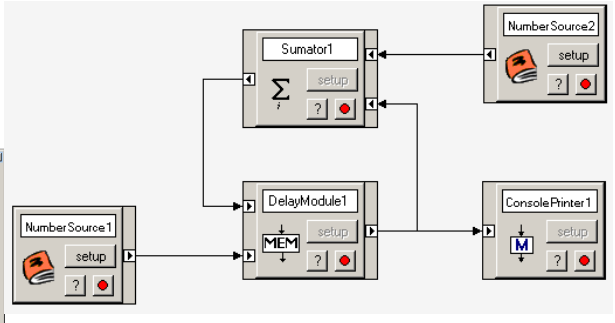


Figure 2: (left) Simple example of module driven subbranch execution.

Figure 3: (right) Simple example of loop with delay module.

3 Application of MVE-2 for AI and CG

The recognition task was one of the main topics of AI research in the past decades. Many efforts appeared in the field of voice recognition, image recognition and mesh recognition, as this task is crucial for understanding the environment. Recognition algorithms allow the AI to reduce the amount of information to be processed; it allows to understand the relations in the environment and to make correct decisions quickly.

The task we are addressing using MVE-2 is recognition of dynamic meshes, i.e. animations in surface representation. Our goal is to provide not only static information (i.e. like "the object in front of the camera is a human"), but also dynamic ("the object in front of the camera is a human, who is jumping"). This will not only allow the system to better analyze the scene in current time, but it will also help the system to predict future states of its environment.

The task of dynamic mesh extraction is one of the state of the art problems that is being investigated by many recent papers ([3], [4]), but for our purposes we can assume that the extraction was already performed. Our input is therefore a dynamic mesh M, that consists of *n* static meshes. Our task is to qualitatively evaluate the dynamic mesh and to produce information about it that will help an AI system to plan its actions.

Our approach is based on template comparison. We suppose that there is a library of dynamic meshes that represent actions known to the system. The information we are extracting is the correspondence of the given dynamic mesh M to the meshes present in the library. Namely we want to create a metric in the space of dynamic meshes, that will tell us which of the known animations is most similar to the one extracted from the environment of the system.

Our method is based on the approach used for static mesh comparison ([1], [2]), i.e. using the Hausdorff distance of two objects. We represent each dynamic mesh in E3 by a static tetrahedral mesh in 4D, subsequently we compute the approximate Hausdorff distance of given mesh to each of the library meshes, and finally we pick the one with the smallest distance. Following this scheme however requires addressing some non trivial issues, which will be briefly discussed in the following paragraphs.

3.1 Dynamic mesh representation

Our approach is to represent a dynamic triangle mesh by a static tetrahedral mesh in space-time. This can be easily done for meshes of constant connectivity (i.e. where each triangle corresponds to exactly one triangle in any frame of the animation). In such a case we can see the evolution of a triangle between two frames as a prism in 4D. We can now break this prism into three tetrahedra. If implemented carefully, this approach leads to consistent tetrahedral mesh representation, even though the faces of the 4D prism are non-planar.

Another issue to be addressed is the used units. We must use consistent units for all the meshes, and we must define relation between time and space units.

In order to unify space units we have decided to use relative lengths only, i.e. all sizes and positions are expressed as fractions of the body diagonal of the object. This allows us to measure spatial difference consistently for all meshes.

On the other hand, time can be measured absolutely and should never be scaled. The only thing that needs to be done is to relate the time units to spatial units in order to produce the Euclidean metric in space-time that will be needed for the Hausdorff distance computation.

The purpose of the space-time representation is to find how similar two animations are. In other words, distance in the space should represent difference between meshes. Therefore the distance represented by a single unit in each direction should represent equal difference. We wish to find a constant that will relate time (measured absolutely in seconds) and space (measured as a fraction of the main diagonal). We don't know the value of this constant, but we can do following considerations in order to estimate its value:

1. time span of 1/100s is almost unrecognizable for a human observer, while spatial shifts of 10% is on the limit of acceptability, therefore we expect the constant to be larger than 0.01/0.1 = 0.1
2. time spans of units of seconds are on the limit of acceptability, while spatial shift of 0.1% is almost unrecognizable, therefore we expect the constant to be smaller than 1/0.001 = 1000

Saying that, we can guess the value of the relation coefficient to be about 10, i.e. time span of 100ms is equal to spatial shift of 1%.

3.2 Implementation

We have implemented the proposed method in a set of MVE-2 modules. First, we have debugged a simple module for computation of a distance from a point to a tetrahedron in 4D. Constructing a module that composes a set of triangular meshes into a space-time tetrahedral mesh was very easy thanks to the generality of data structures provided by the Visualization library. It is also easy to use a variety of input formats.

In order to speed the computation up we have also constructed a module called `AnimationDistanceEvaluator` that encapsulates the distance evaluation from each vertex of one mesh to each tetrahedron of the other. This module provides a significant speedup of the process by using advanced acceleration techniques (spatial subdivision etc.), while it preserves reusability of code, because it calls public methods of the `PointToTetrahedronDistanceEvaluator` module.

A typical map may consist of two loops that compose two space-time tetrahedral meshes. For each of them a new point attribute is computed using the `PointToTetrahedronDistanceEvaluator` module that represents the distance from each point to the other mesh. A general `AttributeMax` module can then be used for computing the one-way mesh distance, and a general `ScalarMax` module finally produces the symmetric estimate of Hausdorff distance.

The resulting point attribute can also be used in other ways. We may display its value distribution by the standard `AttributeHistogram` and `CurveRenderer` modules. Such visualization helps when considering similarity of animations.

We can also transform this attribute into color attribute and display it using some MVE-2 renderer. It allows us to see exactly where and when the two animations are similar or distinct. Such information can be also very useful in many AI tasks, for example machine learning, where a trainee can see how precisely she follows some pattern.

4 Conclusions

We have shown a method for comparing dynamic meshes. This method can be used for a variety of AI applications, from animation recognition to automated learning or teaching. The implementation in the MVE-2 environment allows easy experimenting with the method in various setups and algorithms.

The current implementation is still not fast enough to compare moderately complex animations in real time, but we are still working on speeding the method up. We believe that the performance of the optimized

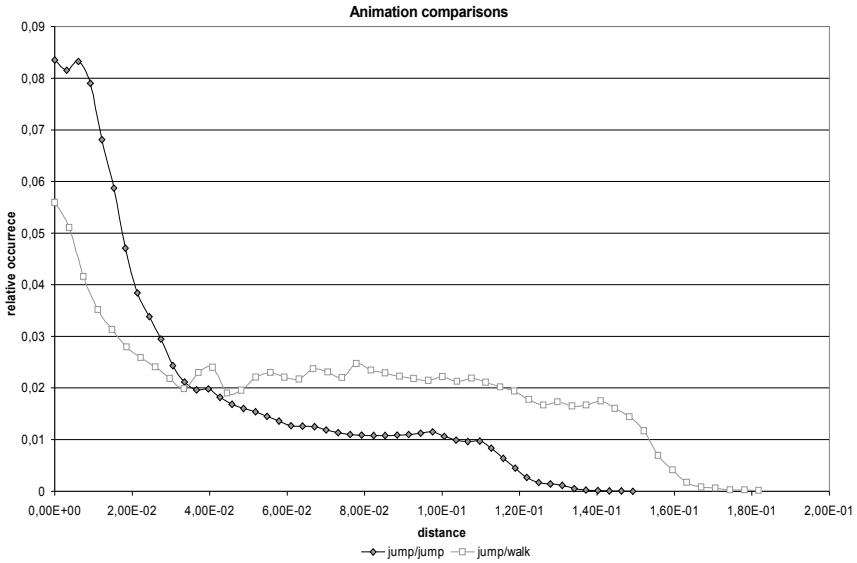


Figure 4: Distance distributions. Human jump ([3], [4]) sequence was used to show results of animation comparison. The jump/jump curve shows the distance distribution when comparing two jumps contained in the sequence (the animations are not equal!). The other curve shows the distance distribution when a jump sequence is compared to a walk sequence of the same length. It can be clearly seen that comparing the jumps produces more small distances, while comparing non-equal animations produces more bigger distances.

algorithm and future hardware will allow to employ our approach in real-time applications.

5 Acknowledgements

The authors of this paper would like to thank all the contributors of the MVE-2 system, who put lots of effort to make it better. Namely following people contributed significantly: Miroslav Vavruška, Petr Dvořák, Zdeněk Češka, Petr Vaněček and Václav Mikolášek.

6 References

1. Cignoni, P., Rochini, C., Scopigno, R.: Metro: measuring error on simplified surfaces. Technical Report B4-01-01-96, Istituto I.E.I. - C.N.R., Pisa, Italy, January 1996.
2. Aspert, N., Santa-Cruz, D., Ebrahimi, T.: Mesh: Measuring errors between surfaces using the hausdorff distance. In Proceedings of the IEEE International Conference on Multimedia and Expo, volume I, pages 705--708, 2002.
3. Anuar, N., Guskov, I.: Extracting Animated Meshes with Adaptive Motion Estimation. Proc. of the 9th International Fall Workshop on Vision, Modeling, and Visualization, 2004.
4. Sand, P., McMillan, L., Popovic, J.: Continuous Capture of Skin Deformation. ACM Transactions on Graphics. 22(3), pp. 578-586, 2003.
5. *Home pages of MVE-2.* http://herakles.zcu.cz/research/mve2/