MOVIE-BASED MULTIMEDIA MATRIX LIBRARY

Dmitry Vazhenin and Alexander Vazhenin

Graduate School Department of Information Systems
The University of Aizu, Aizu-Wakamatsu, Fukushima, 965-8580, Japan
E-mail: {d8052102,vazhenin}@u-aizu.ac.jp

ABSTRACT

The paper describes a library supporting effective programming and design of matrix algorithms and programs. The important feature of proposed library is the visual algorithm representation using a movie-based approach. The user has a deal with special multimedia objects, each of which can generate an executable code as well as produce animation frames. These objects build an algorithmic skeleton representing the steps of computation. In this paper, we show the main features of movie-based programming as well as describe the movie-based matrix library. Examples of the library usage are also presented.

1. INTRODUCTION

Nowadays, visual programming has become a standard environment, and many applications include a big variety of attractive functions with icons, pictures, animations, sound and other multimedia components [1]. This is because visual applications provide to the user reliable understanding as well as effective manipulating with the complex objects. Some of them (Visual C++, VisualWorks, etc.) typically consist of browsers for manipulating text, combined with a GUI editor and a rudimentary application skeleton generator. The Limnor system can be as an example of the last development in this area [2].

The other directions are focused to offer substantially higher expressiveness than conventional textual programming by means of software visualization and a visual programming language. The following examples represent this approach. Tanimoto [3] states that "Data Factory" indicates visual dataflow environment. In this model, users can control icons prepared with mathematical operations in the layout where mathematical methods are connected to others like belt conveyers in the factory. JAVAVIS was developed as a tool to support teaching object-oriented programming concepts with Java [4]. This tool monitors a running Java program and visualizes its behavior with two types of UML diagrams, which are de-facto standards for describing the dynamic aspects of a program, namely object and sequence diagrams. We can characterize most of the mentioned systems as very special. They are mostly focused on solving specific problems.

A multimedia approach for interactive specifications of applied algorithms and data representations is based upon a collection of computational schemes represented in the "film" format [5]. In [6], some modifications to this approach, called the Movie-based Multimedia Environment for Programming and Algorithm Design (MMEPAD), are shown. The movie-based programming process is manipulated with special movie-program objects (MP-objects) that generate *automatically* a part of an executable code as well as producing movie frames, which are *adequate* for the code generated.

The matrix operations are often used in many scientific and practical computations. That is why the visual representations of matrix data and algorithms are very important in order to choose a corresponding method as well as design matrix algorithms. Traditionally, the matrix software consists of well-maintained libraries available commercially or electronically in the public-domain, other libraries distributed with texts or other books, individual subroutines tested, as well as individual or electronic sources which may be hard to use or come without support. As shown in [7], there is a need for tools to help users in picking the best algorithm and implementation for their numerical problems, as well as in getting expert advises on how to tune them. Authors suggested for using special text-based forms called templates. This approach can be considered as a good idea to help in choosing the suitable algorithm. Nevertheless, essential problems with understanding computational schemes presented still exist. This research is devoted to design the new the movie-based matrix library including objects having mentioned-above features.

This paper has the following structure. In section 2, features are shown of movie-based programming technology. In section 3, we discuss also main stages of the movie/program design and debugging. Section 4 contains description of movie-based matrix library. Some remarks and examples of the movie-based library usage are shown in section 5. Conclusion and future work are discussed in section 6.

2. MOVIE-BASED PROGRAMMING ENVIRONMENT

The Movie-Based representations of computational methods and algorithms consider a correspondence between algorithmic movie frames and problem solution steps. Accordingly, each frame should visualize/animate a corresponding step of a program/algorithm execution. We define such a frame as the Movie-Program Frame or MP-frame. The *Movie-based Programming* is in manipulating with special objects generating a part of an executable code as well as producing MP-frames, which are adequate to the code generated. The key point of this concept is based a ***Movie-Program Skeleton*** or ***MP-skeleton*** including components supporting these dualistic features (Figure 1).

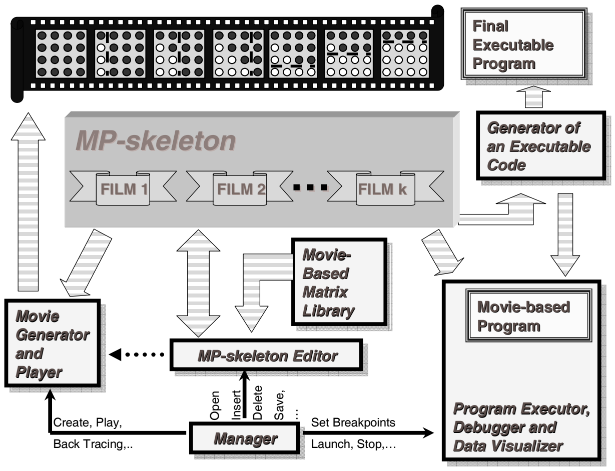


Figure 1. Movie-based Programming Environment

The Movie-based software architecture includes five main program modules. The *MP-skeleton Editor* allows users to manipulate with MP-components specifying movie-program parameters. *Movie Generator and Player* is to produce an algorithmic movie by generating a basic MP-frames sequence. This sequence can be extended by additional animations/sounds in order to improve movie presentation. It is also possible to generate and play movie fragments together with editing operations. The *Generator of an Executable Code* is to create a program from the MP-skeleton. There are two possible types of MP-programs. The *Final Executable Program* is generated according to the target machine requirements. The *Movie-based Program* can be implemented and debugged under control of the *Program Executor, Debugger and Data Visualizer*. The *Manager* controls all system operations and data access procedures. The movie-based matrix library is a collection MP-components allowing the user to use complete components for the program/algorithm design

3. MP-SKELETON DESIGN STAGES

Figure 2 depicts an example of an algorithmic movie showing computations on matrices and containing 13 MP-frames. Each MP-frame highlights and flashes some elements of parameterized matrices. This means that operations or formulas should be defined on corresponding matrix elements and/or sub-structures like sub-matrix, column, and rows. Different operations can be coded by different colors/sounds/animations. Special ***Control Lines I1, I2*** and ***J1, J2*** are used to simplify the computational scheme understanding and for possible references.

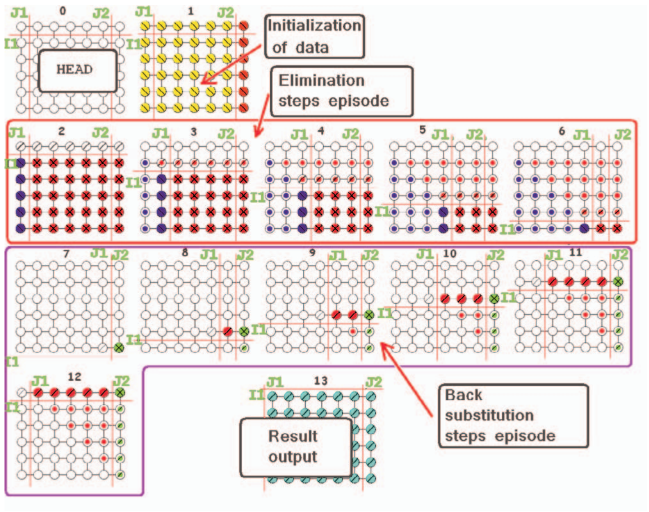
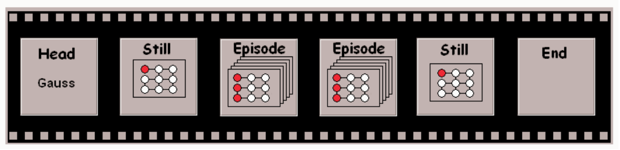


Figure 2. Gaussian Elimination Movie Example

Usually, any MP-skeleton consists of a set of ***MP-films***. There exists one *main MP-film*. Other films can be activated by using a special calling mechanism. Each MP-film consists of a set of ***MP-stills*** each of which is a scene defining how to generate MP-frames and program. The user should specify parameters of this generation by manipulating with the MP-still objects like ***MP-nodes***, ***MP-structures***, ***Control Lines*** as well as ***MP-formulas*** defining operations on these objects. As shown in Figure 2, a MP-skeleton for the Gaussian Elimination Movie contains four MP-stills.



a). Gaussian Elimination Skeleton

$$A[\bullet,\bullet] = A[\bullet,\bullet] - A[\mathrm{mat1}_{I1}-1,\bullet] * \frac{A[\bullet,\mathrm{mat1}_{J1}-1]}{A[\mathrm{mat1}_{I1}-1,\mathrm{mat1}_{J1}-1]};$$

$$A[\bullet,\bullet] = 0$$

b). Gaussian Elimination MP-formulas

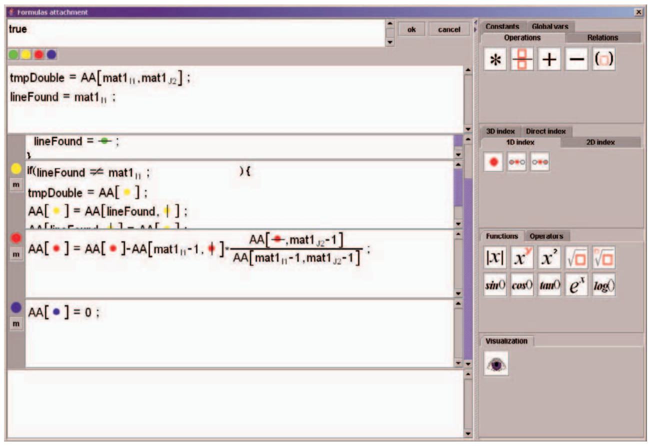
Figure 3. Gaussian Elimination MP-Skeleton

The algorithmic MP-skeleton design process includes specifications of MP-objects like MP-stills, control lines, structures (scalars, vectors and matrices) as well as definitions of behaviors of these elements during transitions of MP-frames. The process of MP-still creation/debugging should be implemented as follows. Firstly, the user should specify a set of structures (scalars, vectors, matrices, etc.) needed to realize algorithm. Secondly, for each structure, it is necessary to introduce ***control lines*** and define their positions and behaviors by inputting special index expressions called ***I-formulas*** according to the rules shown in Figure 4. Finally, the user should specify nodes activities by coloring corresponding domains and substructures. As shown in Figure 2, the domain configuration in changing according to the control lines positions.

CF_ID – *Control Line Name*	Example
Initialization Rule: **CF_ID = <expression>**	J1 = 1;
Transition Rule: **if (<condition>) then CF_ID = <expression> else CF_ID = <expression>**	***if*** (J1<J2) ***then*** J1++; ***else*** J1= J2;
Episode Rule: **if (<condition>) then*{Generate Next Frame}*; else *{Finish Episode}*;**	***if*** (J1 == J2)

Figure 4. I-formulas semantic rules

Computational formulas or ***C-formulas*** are necessary to specify operations on colored MP-nodes. We define a C-formula as a subprogram containing a sequence of arithmetical and logical expressions to specify some local nodes activities. Each C-formula includes the following components: ***MP-expressions***, ***Control structures, Regular text***. ***MP-expressions*** are to specify data access and operations on MP-nodes. The C-formula notation is to the conventional mathematical expressions (Figure 5.).



```
tmpDouble = AA[mat1_I1, mat1_J2] ;
lineFound = mat1_I1 ;

lineFound = ● ;
}
if(lineFound ≠ mat1_I1 ; ){
tmpDouble = AA[ ● ] ;
AA[ ● ] = AA[lineFound, ● ] ;

AA[ ● ] = AA[ ● ]-AA[mat1_I1-1, ● ]* AA[●, mat1_J2-1] / AA[mat1_I1-1, mat1_J2-1] ;

AA[ ● ] = 0 ;
```

Figure 5. C-Formula Attachment Interface

As shown in Figure 5, we are enhancing C-formulas by using special multimedia attributes like images, symbols and tables in order to improve the formula perception. ***Control structures*** are used to point branch conditions. ***Regular text*** can be comments and/or a custom code, which extends formula capabilities.

In most cases, film program consists of several stills and structures. Each still contains a set of traversal schemes specifying colored domains in corresponding structure. Each schema has its own color and formula sequence attached to this color. The same nesting scheme is always reflected in the corresponding program source code. Each still produces one or several static frames representing skeleton steps of computation and hiding formulas.

4. MATRIX LIBRARY COMPONENTS AND USAGE

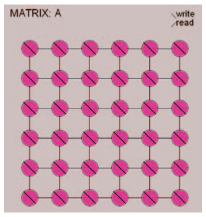
The efficiency and convenience of the most of programming systems depends on variety of embedded packages and libraries. Those tools can significantly reduce the software design expenses as well as help in understanding of computational methods. In this section we describe a collection of movie-based matrix routines.

The library contains the matrix film set supporting basic matrix-vector and matrix-matrix operations. It includes ***matrix multiplication algorithms*** of different kind, ***matrix transposition stills***, etc. The other group of MP-stills and episodes is oriented in creating solvers of systems of linear algebraic equations (SLAE). It includes ***direct SLAE solvers,*** for example, a set of the Gaussian Elimination and LU-decomposition algorithms as well as ***iterative solvers***.

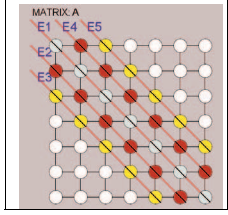
Matrix Generators and Service Functions are a special set of standardized MP-stills including movie-based algorithms for obtaining matrices with given types and features. These matrices are necessary in developing and evaluating many matrix algorithms and. In other words, it is a set of the movie-based matrix generators. This group includes also some service functions like data dump, visualization, print, etc. Table 1 contains several typical examples of matrix generators and services as well as corresponding C-formulas. The left table column shows the matrix structure color representation implemented according to control lines positions. The right column demonstrates C-formulas, which can be implemented on the corresponding matrix elements. We show also traditional mathematical expressions to compare them with our representation. It is possible to see that the usage of images can improve the formula perception and understanding.

As shown in Figures 2 and 3, the first MP-still of the Gaussian Elimination Algorithm is a matrix generator. The last MP-still can be used, for example, for printing results. It is necessary only to import them from the library. The user can also extend the library content by including his/her own stills and films.

Table 1. Examples of Matrix Generators

Frame image	Description and C-formulas
Generate full matrix 	***Hilbert Matrix*** A[●] = 1.0 / (⊖ + ⦶ +1.0) ; $a_{ij} = \frac{1}{i+j+1}; \quad i,j = 0,1,\cdots,N-1$
	Row-Column Matrix if (⊖ ≥ ⦶) A[●] = ⊖ +1 ; else A[●] = ⦶ +1 ; $a_{ij} = \begin{cases} i+1 & \text{for } i > j \\ j+1 & \text{for } i \le j \end{cases}; \quad i,j = 0,1,\cdots,N-1$

	Print matrix as a table
	if (⦶ < MATRIX$_M$)
	🖨 ("%lf", A[●]);
	else
	🖨 ("%lf\n", A[●]);

Generate a band matrix 	***Band Matrix defined by five Control Diagonal Lines*** A[●] = 1; A[●] = 2; A[●] = 3; **E1:** $a_{ij} = 1, \; j = 0,1,\dots,N-1; \; i=j$ **E3, E5:** $a_{ij} = 2, \; a_{ji} = 2, \; j = 1,\dots,N-1; \; i=j-1$ **E2, E4:** $a_{ij} = 3, \; a_{ji} = 3, \; j = 2,\dots,N-1; \; i=j-2$

Any movie-based library component can easily be imported or inserted into the user's application skeleton. To import MP-films, the user can simply insert the selected MP-film into his/her application and specify corresponding call-still on order to bind formal and real film parameters.

Import operations on stills and episodes have some specifics because of necessity in redefining variables and structures names, sizes, etc (Figure 5).

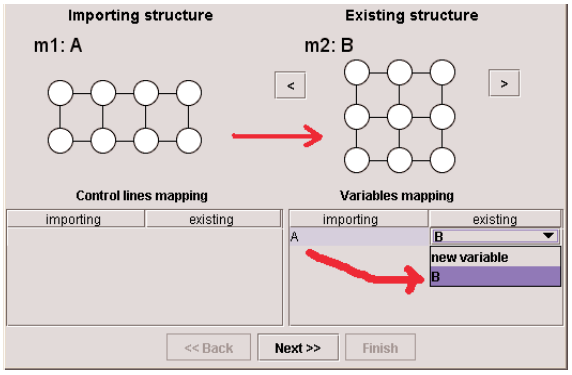


Figure 5. Example of Mapping Imported Structure

The user should specify a mapping of imported entities and real variables and structures defined in the user's MP-film. Importantly, all formulas will be transformed according to the user's film notations. If an inserted MP-still or episode has an extended set of variables and structures, all unspecified components will be added to the user's film component set.

5. CONCLUSION

The proposed movie-based library can significantly help in designing and debugging matrix algorithms. It combines an executable code generation with the visual representation of algorithms and programs. It seems also very attractive to use such environment for education purposes. The library is open and allows adding/designing new matrix algorithms. The programmer can easier understand relations between a real application and algorithm used for it. Designing a formula sequence the user should operate only with visual objects specifying variable names and index expressions as multimedia symbols. The results of testing confirm that the presented system can be used not only as an algorithm demonstration tool but also as a programming tool.

The system presented is realized on Java platform. It generates C/C++ programs and can export movies in the Macromedia Flash Animation format.

6. REFERENCES

[1] J. Stasko, J. Dominique, M. Brown, and B. Price, *Software Visualization: Programming As a Multimedia Experience*, The MIT Press, 1998.

[2] *Limnor Tutorial.* ©2003 Longflow Enterprises Ltd. http://www.limnor.com/.

[3] S. Tanimoto, "Programming in a Data Factory," *Proc. of Human Centric Computing Languages and Environments*, Auckland, pp. 100-107, 2003.

[4] R. Oechsle, and T. Schmitt, "JAVAVIS: Automatic Program Visualization with Object and Sequence Diagrams Using the Java Debug Interface (JDI)," *LNCS*, Springer-Verlag, Vol. 2269, pp. 1-15, 2002.

[5] N. Mirenkov, A. Vazhenin, R. Yoshioka, Ts. Ebihara, at al., "Self-Explanatory Components: A New Programming Paradigm," *Int. Jour. of Soft. Eng. and Knowledge Eng.,* vol. 11, no. 1, pp. 5-36, 2001.

[6] D. Vazhenin, A. Vazhenin, and N. Mirenkov, "Movie-based Multimedia Environment for Programming and Algorithms Design," *LNCS*, Springer-Verlag, Vol. 3333, Part III, pp. 533-541.

[7] Z. Bai, D. Day, J. Demmel, J. Dongarra, M. Gu, A. Ruhe, and H. Vorst, "Templates for Linear Algebra Problems," LNCS, Springer-Verlag, Vol. 1000, Springer-Verlag, 1996.