ABSTRACT
In the presented work, we discuss the Movie-Based approach to design and represent of matrix algorithms including a lot of identical operations on matrix elements. These operations usually transform an initial matrix structure into a matrix with a given structure. Therefore, it is possible to represent a matrix algorithm as a series of frames or iterations reflecting the stages of this process. In other words, this is a series of matrix data representations. The key-point of this approach is the presence of special multimedia movie-program objects (MP-objects) having possibility to generate an executable code as well as produce movie frames, which are adequate to the code generated. Both movie and program can synchronously be generated and debugged. The important feature of the debugging process discussed is that debugging operations can be implemented in any stage of the movie/program design. Some examples are shown of typical matrix algorithms.

KEY WORDS
Multimedia Information Systems, Visual Programming, Movie-based Representation of Methods and Information Processes, Matrix Computations

1. Introduction
Within the past years, visual programming has been increasingly attracting attention, mainly because of its promise to make programming easier. This supposes also allowing laity to encroach on the domain of computer experts to solve problems with the computer. The market has already discovered the fascination of visual programming, and various new programming environments have been declared to be "visual". 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. The authors pointed that the Limnor is the world’s first generic purpose no-coding programming system. It can be used to create computer software without using any computer languages. The system has rich possibilities in designing GUI interfaces like kiosks, client/server platforms using language independent environment with relatively simple computational part [1].

Although some might call such environments visual, visual programming has to offer substantially higher expressiveness than conventional textual programming by means of software visualization and a visual programming language [2,3]. The following few examples confirm these concepts.

S. Tanimoto proposed the Data Factory, which is an experimental visual programming environment based on a form of computation called the "factory model" [4]. This software supports investigations into the use of the factory model in explaining computing concepts and in exploring the possibilities for programs and program styles in the factory model.

The animated visual 3D programming language SAM (Solid Agents in Motion) for parallel systems specification and animation was proposed in [5]. A SAM program is a set of interacting agents synchronously exchanging messages. The SAM objects can have an abstract and a concrete, solid 3D presentation. While the abstract representation is for programming and debugging, the concrete representation is for animated 3D end user presentations.

A system called JAVAVIS was developed as a tool to support teaching object-oriented programming concepts with Java [6]. The 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 the most of mentioned systems as very special. They are mostly focused on solving specific problems.

Multimedia approach for interactive specifications of applied algorithms and data representations is based upon a collection of computational schemes represented in the "film" format proposed in [7,8]. In the presented work, we discuss a variant of this approach called the Movie-Based Programming. The programming process is in manipulating with special movie-program objects (MP-objects) generating automatically a part of an executable code as well as producing frames, which are adequate to the code generated. Both movie and program can synchronously be generated and debugged.
To test our approach, we consider various computational methods to be realized in movie-based format. Matrix computations are the heart of most applications in scientific computing. They are under our special attention because of presence of well-maintained matrix software libraries available commercially or electronically in the public-domain as well as other libraries distributed with texts or other books, etc. 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. We can help in solving this problem.

In Section 2, we discuss the Movie-based Programming Paradigm, and show main elements of Movie-based Multimedia Representation of matrix algorithms. The third section describes the main user’s activities to design/debug MP-skeleton. In Section 4, the movie-program generation process and debugging procedures are shown. Section 5 describes movie-based examples of typical matrix algorithms. The last section contains conclusion and future research topics.

2. **Movie-based Representation of Computations and Data**

2.1. **Movie-based Representation of Matrix Computations and Data**

An important feature of matrix processing is that there are a lot of identical operations on matrix elements. These operations usually transform an initial matrix structure into a matrix with a given structure. For example, a variety of direct methods to solve systems of linear algebraic equations (SLAE) can be reduced to a limited set of computational schemes providing transformations of to triangular, diagonal, bi-diagonal, etc. matrices.

Another issue is how zones of computation can be reduced. The computation reduction is because of number of zeroed elements is increased during the elimination of unknowns. Operations on these elements can be avoided by implementing row-by-row, column-by-column, element-by-element eliminations. Therefore, it is possible to represent a matrix algorithm as a series of frames or iterations reflecting the stages of this process. In other words, this is a series of matrix data representations.

Iterative implementation schemes are often based on wave-fronts of computation along diagonals or matrix sides from one boundary to another and back. Wave-fronts can be of a line type or more sophisticated 2-D structures. As a rule, computation in all nodes of a wave-front can be performed in parallel. The computation related to different positions of the wave-front is performed sequentially. In addition to wave-fronts schemes, checker board or multi-color-board techniques are used. In these schemes the computation is performed on all nodes of one color, then on all nodes of another color, and so on. The computation in all nodes of the same color can be performed in parallel. The computation related to different colors is performed sequentially.

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 user has a deal with special multimedia movie-program objects (MP-objects). Each such an object is to generate an executable code as well as produce movie frames, which are adequate to the code generated. Figure 1 depicts a simple example of an algorithmic movie showing computations on matrices and containing six MP-frames. Each MP-frame highlights and flashes some elements of a parameterized matrix. This means that operations or formulas should be defined on corresponding matrix elements and/or sub-structures like sub-matrix, column, and rows. As shown in figure, special horizontal and vertical Control Lines i1, i2 and j1, j2 are used to simplify the computational scheme understanding as well as divide a whole matrix into colored domains. This allows distinguishing the matrix elements activities because the different operations can be coded by different colors/sounds/animations.

Importantly, control lines are changing their positions during transitions MP-frames. In the next section, we show how to program control lines behaviors. Accordingly, the colored areas also change their configuration. This allows the flexibility of programming the matrix elements activities in time.

2.2. **Movie-based Components**

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 proposed concept is a Movie-Program Skeleton or MP-skeleton including components having these dualistic features. Usually, any MP-skeleton consists of a set of MP-films. Similar to the traditional programming, each film can be considered as a procedure or function. 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. Usually, a still is a photograph of a scene from a movie (cinema film). In our system, such a scene is used for the frames-code generation. The user should specify parameters of this generation by operating 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, there exist several types of MP-stills. According to how the MP-frames will be produced, we distinguish the following types of MP-stills.

- **A single MP-still** corresponds to one computational step in the MP-movie and produces one MP-frame.
- **A still series** or **EPISODE-still** produces the set of MP-frames reflecting a complete fragment of a problem solution.
- The **HEAD-still** contains description of data structures and variables used in a current film.
- The **END-still** is to finalize a film.
- The **IF-still** is to skip or process selected groups of stills. The user should specify a logical conditional expression as well as mark stills that will be processed for true and false cases correspondingly.
- The **WHILE-still** is to repeat the processing of stills marked while a condition is true.
- The **CALL-still** is to pass processing to other MP-films. In this case, the END-still will return control to the parent film.

Each still also contains a set of internal objects. A **MP-node** is an elementary solid multimedia part (cell) of 3D-space. MP-nodes activities are coded by colors/sounds/animations. A **MP-structure** is a set of MP-nodes joined according to the structure type like scalars, linear and matrix arrays, grids, trees, etc. For matrix computing, the MP-structure is matrix data, and each MP-node corresponds to a single matrix element. **Control lines/structures** are used to address nodes, and/or show dependences between nodes.
3. MP-skeleton Design Stages

Accordingly, the movie-based program design includes four stages:
- Creating the algorithmic MP-skeleton,
- Attaching Formulas,
- Generating, Executing and Debugging Movie-based Program,
- Exporting algorithmic movie and program to the target machine.

The important feature of the debugging process discussed is that debugging operations can be implemented in any stage of the movie/program design. In the following sections, we will discuss this process in more details.

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.

As was pointed above, the control lines and/or structures can be used for the better explanation and understanding of method presented. This allows, for example, showing directions or placement of computations. Importantly, control lines point also placements and sizes of substructures and domains inside 2D-structures.

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. Finally, the user should specify nodes activities by coloring corresponding domains and substructures.

**Index formulas** or *I-formulas* are used to define control lines activities in order to update control lines positions during frame transitions (Figure 3).

![Figure 3: The MP-still Editor Window](image-url)
Each I-formula consists of control line names, basic arithmetical operations and branch conditions. It can be attached to a control line on a particular still. Therefore, each control line can have several index formulas inside different stills. As shown in

Figure 3, I-formulas can also be used for specification of conditions to finish MP-episodes. For example, it is possible to specify end-conditions of MP-episode as an equality of the control line positions with the structure boundaries as shown in Figure 1.

Operations related to I-formulas debugging can be implemented simultaneously with editing operations. In any time, the system can animate control lines movement and results of the I-formulas execution. It is also possible to provide frame-to-frame browsing of MP-episodes. The interface for inputting I-formulas includes not only text-based expressions but also icon-oriented subset to simplify some typical control line movements.

During specifications of the node activities, some difficulties appear when domain/substructure shapes will be irregular or complex. This situation can be avoided by decomposing such areas to a set of regular domains having a standard shape. It is important to note, that these domains are depending on the control lines placement, and change their configuration according to control lines movement during MP-frames transitions.

Algorithmic skeleton shows data structures and some activities on these structures. In order to specify these activities, it is necessary to attach arithmetical and/or logical formulas to the skeleton. In the next section, we show operations related to formula attachment and debugging.

4. C-Formulas Attachment/Verification

As was shown above, an MP-skeleton shows data structures and some activities on these structures. **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 very close to the conventional mathematical expressions. We are enhancing them by using special multimedia attributes like images, symbols and tables in order to improve the formula perception (Figure 4.) **Control structures** are used to point branch conditions. **Regular text** can be comments and/or a custom code, which extends formula capabilities.

The MP-structure size parameters used in movies may often be different from real ones needed for problem solution. The user should specify not only name and type of a MP-structure but also its dimensions for both movies and program. Those parameters can be either static or dynamic in a program, i.e. may be changed during computations.

Information stored in MP-stills is used to generate MP-frames as well as an executable code. During code generation, some components (scanning loops, variables, etc.) will be simply transferred in the final code defined by a target system. MP-formulas will be converted to the final code after additional verification.

To generate MP-frames, the MP-templates are also used to form images and other graphical information. Calculations using MP-formulas attached can also be implemented, and a movie will be generated representing only one possible case of a MP-program execution obtained according to the real data. It is also possible to generate a movie from a MP-skeleton with non-complete formulas and conditions. In this case, MP-frames with images can only be generated. The user may randomize or specify directly branches needed for IF- and WHILE-stills. As was mentioned, the movie and program have different size parameters of MP-structures. This leads that a movie and program will have/reflect different numbers of MP-frames.

During formula attachment, a debugging scheme allows visualizing and controlling all references to the structure elements. This allows debugging film structure and formulas activity during design-time. The **formula tracing technique** is used visualizing nodes referred by a formula on a particular frame (Figure 5.)

Each C-formula is parsed in order to extract indices of nodes where data access is performed. Those nodes are marked as active with 'read', 'write' and 'read-write' access type. This allows visualizing any wrong access even before program execution (Figure 5b.)
The Movie-based Program is adequate to the corresponding movie behavior. The Run-Time Debugging makes possible verifying a movie-based program data-flow using special breakpoints. When such a breakpoint achieved, the program stops, and the executor invokes the data visualization tool. Information provided to the user includes a global frame and still numbers and a frame number inside episode. He/she can choose to either continue/terminate execution, or return to design stages.

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

5. Examples of Movie-based Matrix Algorithms

Figure 6 depicts examples of movies showing Matrix Multiplication and Gaussian Elimination Algorithms.

The Matrix Multiplication algorithm is represented using one EPISODE-still. Each MP-frame of this still shows calculations of one element of the result matrix C. The row of matrix A (grey colored elements) is used as the leading zone forming the scanning loop to calculate dot products. C-formula should be specified for grey colored elements. After parsing, it will have the following text-representation:

\[
C[\text{mat1}_I\text{1}][\text{mat2}_J\text{1}] += A[\text{mat1}_I\text{1}][\text{col()}] * B[\text{col()}][\text{mat2}_J\text{1}];
\]

In this formula, function \text{col()} returns the column number of the corresponding grey-colored element. This algorithm uses three matrix structures \text{mat1}, \text{mat2} and \text{mat3}, each of which has one matrix variable A, B, and C correspondingly. Index variables of \text{mat1}_I\text{1} and \text{mat2}_J\text{1} correspond to the control lines I1 and J1.

The Gaussian Elimination Movie contains two episodes. The first episode is the Gaussian Elimination Procedure (Figure 6b, Frames 2-6) implementing matrix triangulation. Back Substitution (Figure 6b, Frames 7-12) Episode represents obtaining the final solutions. C-formulas for elimination are shown in Figure 5b.

6. Conclusion and Future Work

The Movie-based Programming allows manipulating MP-objects, automatically and synchronously generating movie frames and adequate executable code. Debugging operations can be implemented simultaneously with editing of the MP-skeleton. The C-formula debugging environment provides additional possibilities to collect and visualize a history of references to the structures and data. The Run-Time Debugging makes possible verifying a movie-based program data-flow using corresponding breakpoints. The system can generate C/C++ programs and can export movies in the Macromedia Flash Animation format.

Our further works will be oriented in designing movie-based linear algebra library as well as include other structure types like trees, stacks, graphs, etc.

References


Figure 6. Algorithmic Movies Examples

a). Matrix Multiplication Movie

b). Gaussian Elimination Algorithm Movie