Java Implementation of a MathML Rendering Engine

Anton N. Dragunov
Master of Science Degree Project

Department of Computer Science
Oregon State University
102 Dearborn Hall
Corvallis, Oregon 97331, U.S.A.
anton@cs.orst.edu

ABSTRACT

Current approaches to presentation of mathematics (on paper or in electronic format) have usability drawbacks that make learning and appreciation of mathematics challenging and often frustrating. In a framework of a big research project of identifying new approaches to communicating mathematical ideas in a highly usable and effective manner, we are building prototype software toolkits displaying documents with math content in various comprehensive ways. For the encoding of mathematics we use MathML, a standard XML-based markup language allowing specification of both facade and underlying semantic content of mathematical presentations, as well as providing certain possibilities for dynamics and interactivity in communication of math. The Java implementation of a MathML rendering engine which we use in our prototypes is the topic of the current project.

Keywords

Math Visualization, User Interface, MathML, XML, Java.

ACKNOWLEDGEMENTS

I want to thank all the people who helped me in my work:

- Dr. Jonathan L. Herlocker, my academic advisor who provided me with lots of invaluable ideas and comprehensive consultations while I was working on this project.
- Yuemei Sun, a graduate student of Computer Science Department of Oregon State University who did a great job in writing a set of Java classes for Unicode character output which I used in my project.
- NACSE, The Northwest Alliance for Computational Scientists and Engineers, the research center of OSU and its people for their understanding and supporting me with all the computing facilities I needed to work productively on this project.
- All my friends from all over the world who gave me tons of inspiration and advice in my work and studies in the United States.

Table of Contents

Chapter 1. INTRODUCTION

MOTIVATION

In this section we are giving the readers a general perspective on the problems we are dealing with in the framework of our big project on Math Visualization.

Why Bother?

Comprehending complex mathematics presents a frustrating challenge for many of us. Although mathematics follows well-defined and logical rules, there are enormous numbers of those rules that we must understand in order to comprehend much of today's mathematics. To communicate mathematics, we must process a symbolic language that can sometimes be very complex. Even as scientists with strong mathematics backgrounds, many of us find that we do not have the patience or the time to spend on processing the current complex mathematical explanations that we find in journal articles.

We can gain a lot of insight by examining mathematical presentations in documents (paper or electronic) from a usability perspective. For example, journal articles and textbooks often present enormously long chains of formulaic derivations spread over many pages of text (e.g. equation 1.1, 1.2, ..., 1.56...), yet, most of us are unable to clearly remember the equations (or the transitions between them) of more than one or two steps the first time that we see them. To understand a derivation, we are thus forced to frequently refer back and forth between different pages to refresh our memory of the derivation. Such a process is tedious, time-consuming, and distracting (resulting in losing your place in the document).

Consider consistency — an important facet of good user interface design. Upon reviewing both textbooks and journal articles, we find different symbols being used for the same mathematical concept. Perhaps worse, we also find the same symbol being used for multiple different concepts in different papers. These variations in the mapping between symbols and concepts can be exceptionally confusing and frustrating to the reader.

At the core of usability engineering is identification of the user interface's target users and their needs. With traditional non-dynamic presentations of mathematics, such as those in journals and textbooks, authors must identify (sometimes implicitly) the mathematical sophistication of their target audience, and provide explanations in sufficient detail for the target audience. For textbooks, the sophistication assumed depends on the level of students targeted, and for journal articles, the author often assumes that the readers will have the same mathematical sophistication as the author. Given these assumptions, many appropriate details may be omitted. For example, how often have you seen statements like "*...and here we use the standard formula for finding the roots...*" without actually showing the formula itself, but only displaying the result of its application? Or we might see statements like "*The author assumes that readers have at least [some level] of knowledge in [some math domain].*"

A problem arises in that not all readers have the appropriate level of expertise or sophistication assumed by the authors. The reader may not know what "the standard formula for finding the roots" is or may not have the appropriate level of knowledge in the specified math domain. For example, a usability expert working with intelligent user interfaces may need to read journal articles about machine learning without having the same mathematical background as a researcher in machine learning. With static presentations that are created once and then distributed to all, this cannot be helped. At the other extreme we might have a situation where the author tries to make the math content of the document be easily understood by a large audience, thus making the whole presentation of ideas unnecessarily long and irritating for people with extensive knowledge of math. However, when presenting math using a medium that supports interaction, such as a personal computer or a personal digital assistant, we can build a presentation of mathematics that adapts to the needs of each individual reader.

The time has come to design, develop, and integrate new approaches to presenting mathematics into our current means for transmitting mathematical knowledge in journal articles and textbooks. Several important factors have developed in support of this endeavor. We are seeing most journals provide all of their articles in an electronic form. The most notable example of this is the ACM Digital Library [1]. Electronic textbooks (or electronic accompaniments to traditional textbooks) are being developed. Thus the channels exist for the continued distribution of mathematical communications in digital form. In support of standardized and structured encoding of mathematics, a recent standard called MathML [24] has emerged that has been accepted by all major vendors of mathematical software. Software already exists to author and display equations in MathML, including stand-alone widgets such as WebEQ or MathPlayer [36], MathML authoring tools such as MathType [25] and IBM Techexplorer [17], and full web-browsing applications with native support of MathML such as Amaya [5] and Mozilla [27]. What remains is to identify the ways in which mathematics can be presented in a highly usable, intelligent, and adaptive fashion, while at the same time maintaining compatibility with existing means of presentation and display.

Integration With Document Standards

Given that we are unlikely to see publishers change their mode of distributing electronic documents, one challenge is to identify how to integrate dynamic math into existing digital document formats, most notably Adobe PDF and HTML. We believe that these challenges can be addressed, as Adobe Acrobat and all popular HTML browsers support plug-ins to extend their functionality. Such plug-ins could provide alternative viewing capability for documents containing embedded dynamic mathematics presentations. However, we do not address those challenges in this project report. Rather we focus on identifying how to present math in a more usable, dynamic, and an adaptive fashion.

Three Usability Issues

Now we summarize the problems which readers of math documents usually experience, and which we believe could be solved with a more interactive and intelligent interface.

1. Inability to view, compare, and contrast related mathematical artifacts within a single field of vision due to limitations of print/display medium.
   - The common way to reference equations in documents (numbers, hyperlinks) helps to save document space and ensures its consistency and integrity, but working in such an environment is inconvenient and error-prone for readers, since locating necessary piece of information in the text takes additional actions — switching between pages back and forth.
   - When a chain of related equations goes through many pages, comparing the equations to one another is difficult, since it, again, takes additional actions to switch between pages and increased concentration of mind to keep in memory much information about the equations previously viewed.
   - In many cases the separation of math artifacts and their textual explanations may cause difficulties in understanding of the mathematical ideas. Authors are almost free in naming the variables or arranging math terms in expressions in any way they find appropriate or convenient in the course of their presentations. Because of that even very common formulas might look quite unfamiliar to readers, who thus will be forced to seek for textual explanations in the surrounding text, which means that readers will have to put additional effort in and spend additional time on comprehending the presentation.
   - In some cases the notation used in general formulas is similar to that the author uses in his/her derivations (e.g., the author may use variable x, and the formula also uses x, but, generally speaking, these two x's describe completely different concepts), and the reader must always be warned about such a trap.
2. Mathematical explanations do not provide the right amount of detail for all potential readers. Something that is unclear is very hard to follow; clarity is the key to understanding.
   - Whether the explanations are clear or not depends on readers' level of preparation. Novice readers want to see all derivation steps and extended explanations of every transition, whereas expert readers want the explanations be brief, very high-level. It is desirable to make the document's level of sophistication be dynamically tailored for every particular reader.
   - There is always a trade-off between conciseness and detail in documents containing mathematical presentations. Making the presentation concise saves document space, reduces its creation time, and speeds up reading, particularly, for expert users. In contrast, providing math explanations in many details supports non-expert users, and ensures that ideas are not misunderstood.
   - To some extent any math document can be considered as a presentation in a mix of two languages: natural language and the language of math formulas. To be precise, the whole presentation usually follows the rules of a natural language, where math terms appear as nouns (playing roles of subjects or objects), with verbs and modifiers being taken from natural language. All educated readers understand the symbolic language of math formulas, since this language is almost universal. However, the readers might not be that much fluent in the language of the document, and this may result in misinterpretation of ideas. Even if the reader is perfectly pro-

ficient in the language of the document, textual explanations are slower to comprehend than their visual counterparts [34, 35]

3. Sometimes notation being used varies from author to author, from document to document.
   - In case if one document contains references to equations in another document differences in notation might be very much misleading.

The next section describes related work, followed by a description of our vision of a new approach to presentation of mathematics.

RELATED WORK

We have to emphasize that we were not aiming to develop another tool for symbolic formula derivation. There are many successful applications for symbolic math [21, 22, 26]. Also, we are not going to develop a new format for representation of mathematical knowledge, like OpenMath [2]. However, in our project we do not exclude the possibility to use the aforementioned tools and formats for doing actual formula derivations, format conversion, and math knowledge replesentation.

Probably, the most popular and very useful technique that electronic document format makes possible to exploit is *interactive illustrations*. Interactive illustrations became very popular in teaching. They have long been studied and proved to be successful [13, 32]. The idea of interactive illustration, basically, is to allow readers not only passively observe the illustration material (picture, graph, diagram, etc.), but also be able to "play" with it, by rearranging objects on the illustration, modifying parameters (in case if the illustration shows diagrams, curves, surfaces, or just tables of numbers), and immediately seeing the effects of such modifications. E.g., in [20], in the set of problems on plain geometry the users can modify pictures (stretch/ shrink circles and/or triangles, move intersection points, etc.) and see that the properties of geometric shapes that need to be proven remain the same for an arbitrary configuration of the shape conforming certain initial conditions.

To some extent, fragments of mathematics in a document can be considered as illustrations to the ideas presented in the surrounding text (the converse is also true). Still, application of the ideas of interactive illustrations to symbolic mathematics has not become widespread.

To enliven their presentations authors may develop small computer applications (which are sometimes interactive, as in [20]), create animations, or record short video fragments, thus providing *animated visualizations* for their documents/presentations. "Project Mathematics!" [30] by Tom M. Apostol and James F. Blinn (California Institute of Technology, Pasadena, CA) is an excellent example of this sort of work. Usually, all such animations follow a fixed scenario and are created "by hand" (meaning that the author has to compose the animation out of a set of frames) for some particular presentation only.

An attempt to make math presentations be able to become clearer as more and more readers work with them has been undertaken by Gabe Johnson's team in University of Colorado, Boulder. Their idea of a "virtual book" [23], in essence, was to put on the "walls" of an on-line book a set of "hooks" on which readers could "hang" multiple bulletin boards (on-line discussion systems). Using these boards the readers could ask questions or exchange their opinions about the text right inside the book, as if making "notes on margins"[1]. The system was expected to serve students to better understand course materials.

A big project, although primarily dealing with internal organization of math knowledge but also introducing innovative techniques for math presentation, is HELM — the Hypertextual Electronic Library of Mathematics [16] (University of Bologna, Italy). This is an attempt to create a sort of a repository of structured mathematical knowledge based on the use of XML/MathML. For math presentation the project team, too, was looking for "innovative interactive capabilities, such as a structure-aware form of selection and the possibility to conceal and disclose parts of the displayed document to change the level of detail". The new MathML rendering engine [29] developed in the framework of HELM was endowed with functionality which went beyond simple display of math in on-line documents.

[1] Almost the same idea, yet, under different name — *annotations*, is utilized in many web sites. Some annotations may be linear (single-level), and some can be organized into hierarchical trees (i.e., annotations to other annotations are possible), thus having right to be called *discussion* boards. For instance, such approach is used at http://www.lenta.ru/, a Russian web site for official news. The difference, however, is that annotations are usually created for the whole document, whereas in Gabe Johnson's project users can annotate specific portions of a document.

ActiveMath [6], is a relatively recent project (based on the use of OpenMath technology) aimed to facilitate presentation of mathematical knowledge by providing a user-adaptive learning environment. However, the ActiveMath software, in essence, is just a web-interface to a repository of math facts, with some features helping to learn mathematical ideas by giving formal definitions, examples and exercises.

Finally, we also have to mention a number of small software tools, which were created to facilitate math comprehension in academic world. FORMULA [14], Alged [2], ComputerMentor [11], Algebra Interactive! [3] are such tools, to mention a few. Some tools allow to "play" with math formulas (rearrange terms, simplify/factorize expressions, etc.), thus working as simplified versions of software for symbolic math with capabilities of doing numerical calculations, aiming, actually, at the same target as interactive illustrations do: answer a question "what if...?". In many cases such tools are designed to work with a "standalone math", not included into documents. Some tools (e.g., ComputerMentor and Algebra Interactive!) work as electronic teachers, explaining the material (particular algebraic or trigonometric problem) at various levels of detail.

All the projects we have just spoken about (with exception of HELM) were aimed to provide convenience of calculations and "playing around", and not convenience in navigation among equations and comprehension of math. User interface aspects were not prevalent; functionality was the pivotal issue.

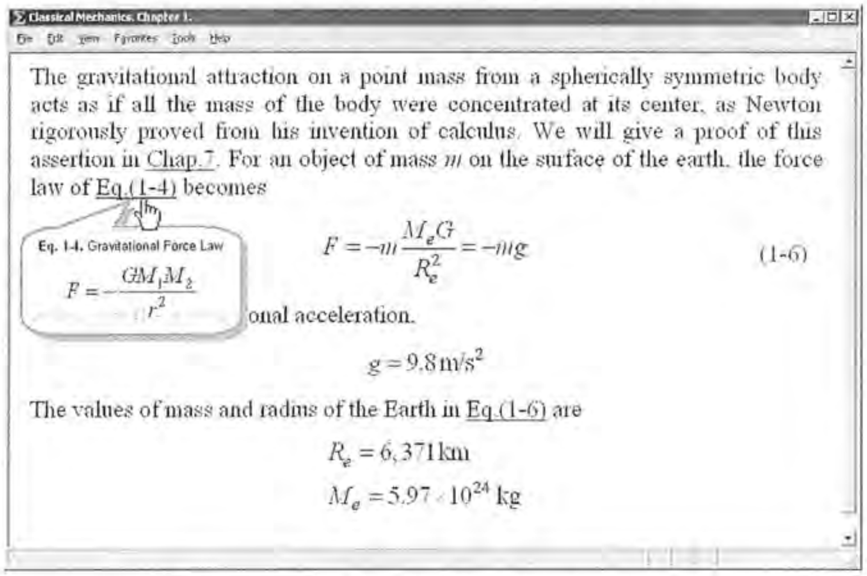
Chapter 2. THE BIG PICTURE

Here we want to identify the approaches to make the user interface of math presentations help reduce the amount of effort needed to grasp the ideas given in math expressions. In what follows we repeat the list the problems we mentioned in the Motivation section, and give a vision of how we might address them in our project. For illustrations we use "screenshots" displaying chapters taken from a physics textbook on classical mechanics [9].

THE VISION

1. Problem: Inability to view related mathematical artifacts within a single field of vision due to limitations of print/display medium.

   - References to equations will no longer require users to switch pages of an electronic document (although, old-fashioned technology of hyperlinks will be also honored). The referenced math expression will appear at the point of reference as a tooltip with extended functionality (Figure 1). By the extended functionality we mean


Classical Mechanics, Chapter 1.
File Edit View Favorites Tools Help

The gravitational attraction on a point mass from a spherically symmetric body acts as if all the mass of the body were concentrated at its center, as Newton rigorously proved from his invention of calculus. We will give a proof of this assertion in Chap.7. For an object of mass m on the surface of the earth, the force law of Eq.(1-4) becomes

$$F = -m\frac{M_e G}{R_e^2} = -mg \qquad (1\text{-}6)$$

Eq. 1-4. Gravitational Force Law

$$F = -\frac{GM_1M_2}{r^2}$$

onal acceleration.

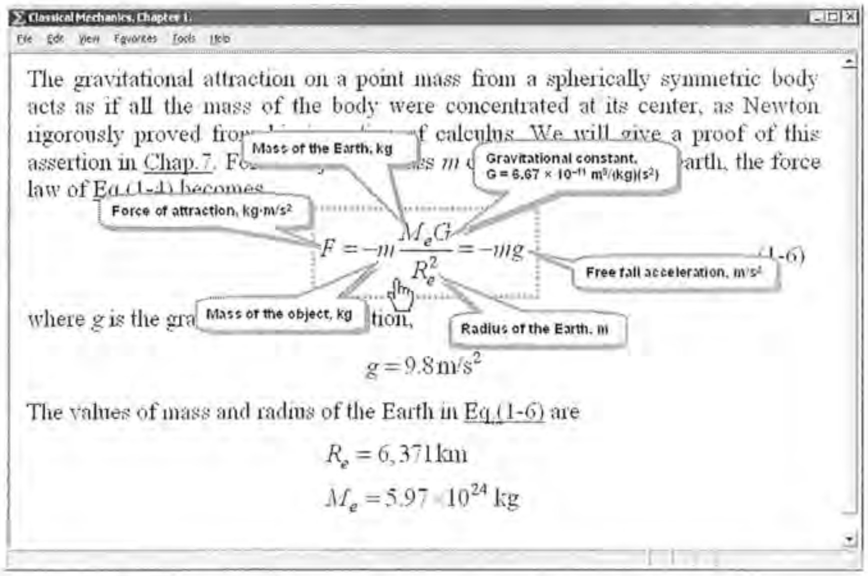
$$g = 9.8\,\mathrm{m/s^2}$$

The values of mass and radius of the Earth in Eq.(1-6) are

$$R_e = 6{,}371\,\mathrm{km}$$

$$M_e = 5.97\cdot 10^{24}\,\mathrm{kg}$$

Figure 1. Simple tooltip with referenced equation.


Classical Mechanics, Chapter 1.
File Edit View Favorites Tools Help
Mass of the Earth, kg
Gravitational constant, G = 6.67 × 10⁻¹¹ m³/(kg)(s²)
Force of attraction, kg·m/s²
Free fall acceleration, m/s²
Mass of the object, kg
Radius of the Earth, m

The gravitational attraction on a point mass from a spherically symmetric body acts as if all the mass of the body were concentrated at its center, as Newton rigorously proved fro f calculus. We will give a proof of this assertion in Chap.7. F s m arth, the force law of Eq.(1-4) becomes

$$F = -m\frac{M_e G}{R_e^2} = -mg \qquad (1\text{-}6)$$

where g is the gra tion.

$$g = 9.8\,\mathrm{m/s^2}$$

The values of mass and radius of the Earth in Eq.(1-6) are

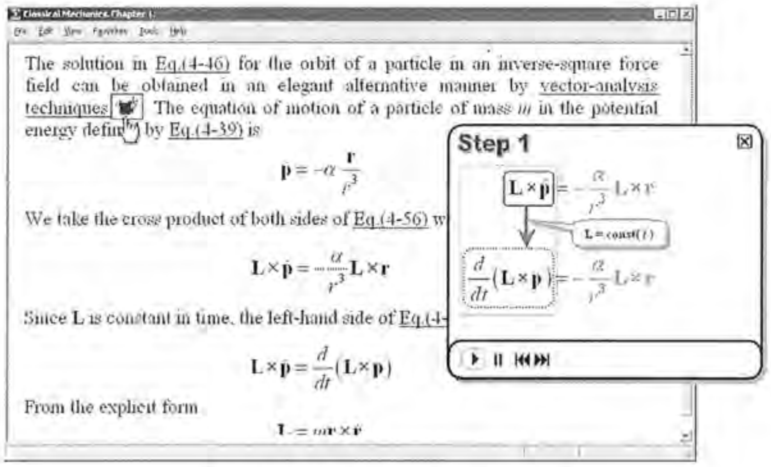
$$R_e = 6{,}371\,\mathrm{km}$$

$$M_e = 5.97\cdot 10^{24}\,\mathrm{kg}$$

Figure 2. Simple callouts with brief descriptions of math terms.

the ability of the tooltip to be "pinned" to the display surface (so that the tooltip will stay on the screen even after the user moves the mouse pointer away from the reference point), and the ability to highlight related math terms both in the equation and the context (surrounding text, or another equation) where the equation is used.

- Every math element (variable, index, operator, etc.) will be provided a brief description showing up as a tooltip (or a text in status bar) which will help readers know exactly what each term means, and/or what measurement units are used for it (Figure 2).
- Derivation chains will have several parallel modes of display: original print-friendly document repeating the format of ordinary paper-based presentation, and a "wizard" tool, activated by clicking on a link/button located at the start of the derivation, and allowing the user to go through the derivation step-by-step, with capabilities to show/hide derivation steps on the wizard's window (Figure 3). In a long chain of math derivation a user will be able to see the "history" of a certain variable or math term by highlighting the variable and/or related math artifacts throughout the whole derivation chain. This will help the user answer questions like "when was this variable first defined?" or "why does this variable/math term appear in the current equation at this particular place".
- The user will be able to see corresponding terms in one or more math expressions by automatic highlighting those terms in appropriate manner, and see the semantics of the correspondence: e.g., "$\sin\varphi = \varphi$ for small φ", or "$\mathbf{L}\times\dot{\mathbf{p}} = \frac{d}{dt}(\mathbf{L}\times\mathbf{p})$ since **L** is constant in time" (Figure 3).



Classical Mechanics. Chapter I.

The solution in Eq.(4-46) for the orbit of a particle in an inverse-square force field can be obtained in an elegant alternative manner by vector-analysis techniques. The equation of motion of a particle of mass m in the potential energy defined by Eq.(4-39) is

$$\dot{\mathbf{p}} = -\alpha\frac{\mathbf{r}}{r^3}$$

We take the cross product of both sides of Eq.(4-56) w

$$\mathbf{L}\times\dot{\mathbf{p}} = -\frac{\alpha}{r^3}\mathbf{L}\times\mathbf{r}$$

Since **L** is constant in time, the left-hand side of Eq.(4-

$$\mathbf{L}\times\dot{\mathbf{p}} = \frac{d}{dt}(\mathbf{L}\times\mathbf{p})$$

From the explicit form

$$\mathbf{L} = m\mathbf{r}\times\dot{\mathbf{r}}$$

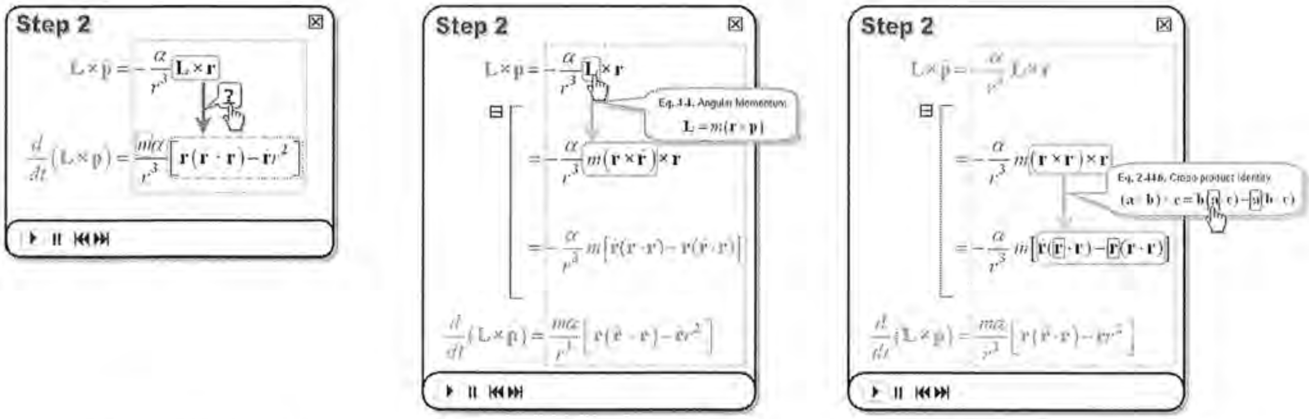
Step 1

$$\boxed{\mathbf{L}\times\dot{\mathbf{p}}} = -\frac{\alpha}{r^3}\mathbf{L}\times\mathbf{r}$$

$\mathbf{L} = \mathrm{const}(t)$

$$\frac{d}{dt}(\mathbf{L}\times\mathbf{p}) = -\frac{\alpha}{r^3}\mathbf{L}\times\mathbf{r}$$

Figure 3. Formula Derivation Wizard.


Step 2
Eq. 3.4. Angular Momentum
$\mathbf{L} = m(\mathbf{r}\times\mathbf{p})$
Eq. 2.44b. Cross product identity
$(\mathbf{a}\times\mathbf{b})\times\mathbf{c} = \mathbf{b}(\mathbf{a}\cdot\mathbf{c}) - \mathbf{a}(\mathbf{b}\cdot\mathbf{c})$

a) Shrunk transition.
b) Expanded transition, step 1.
c) Expanded transition, step 2.

Figure 4. Going through the formula derivation steps.

2. Problem: Mathematical explanations do not provide the right amount of detail for all potential readers.

   - Similarly to what is implemented in [36], our tool will provide the users with possibility to tailor the whole presentation to their level of knowledge by hiding/showing some derivation steps (Figure 4). The interface will include the feature to identify users' level of preparation and give the presentation with the appropriate level of details. For the task of such identification we need a way to classify the mathematical facts more or less universally, so that we could recognize similar patterns of users' behavior over heterogeneous math documents. For example, there may be the following classes of math transformations: 1) application of a formula (standard or derived earlier in the same presentation), 2) term rearrangement, 3) mutual term cancellation, 4) approximation, etc. There might be other classes of math transformations, as well as there might be sub-classes of the classes we have just mentioned. The general idea of learning the user's reading habits and determining his/her level of preparation is to keep track of how often (for every particular class of math transformations) the user looks into the details hidden by default, shrinks the transitions initially expanded, does no modifications to the level of details currently in use, or how many times the user goes through the same derivation during a certain period of time. To collect and use this information we have to identify users somehow and keep records about their behaviors in user profile data structures. In addition to the users' reading habits we also might need to analyze how those habits evolve, so that we could make more precise predictions on the level of details needed for the math presentation the user is about to work with. In any case, these learning features must be configurable and as much transparent for the user as possible.
   - Very often readers do not care about the correct wording and grammar, or do not have time to figure out what a particular word means in the current context, and are mostly concerned about grasping the general idea of how math terms transform. In such cases we need to use another universal language of symbols or gestures. E.g., to express the idea that in an equation some terms are going to be rearranged, simple arrows from initial positions of the terms to where they are going to move to may be much easier to understand than a bundle words saying "...rearranging the factors in the equation gives us..." Arrows, and even better, animations of the rearrangement, in fact, convey more information about how the terms move or transform, since they tell us not only that the rearrangement takes place and produces a certain result, but also show exactly which term goes where during the operation (Figure 5).

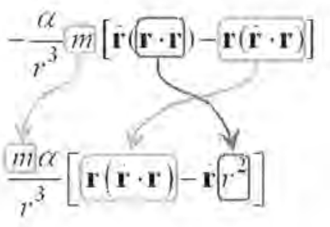


Figure 5. Visual explanation of term rearrangement.

3. Problem: Sometimes notation being used varies from author to author, from document to document.

   - Working with both presentation and content markup our tool will make possible to do automatic conversion of notation.

Chapter 3. IMPLEMENTATION

THE PROJECT

All tasks we have mentioned in the previous section require display of mathematics. Unlike the static display of equations in paper documents or their electronic versions in PostScript, PDF, MS PowerPoint, MS Word, or other similar formats, in our case we need to have all those formulas be able to interact with the user and the parts of the document they are included in. Not only the whole equation, but every math term of the equation must be separately accessible. The processing environment has to know how to link the equation and its terms to other equations and the surrounding text. This is because we inevitably need a tool capable of rendering the equations and providing them with certain interactivity.

MathML

MathML [24], a markup language designed for encoding math presentation in electronic documents is now being supported by more and more software tools dealing with symbolic math. Unlike OpenMath [2] markup language, which is designed for the representation of math content only, MathML has syntactic constructs both for presentation and content markup.

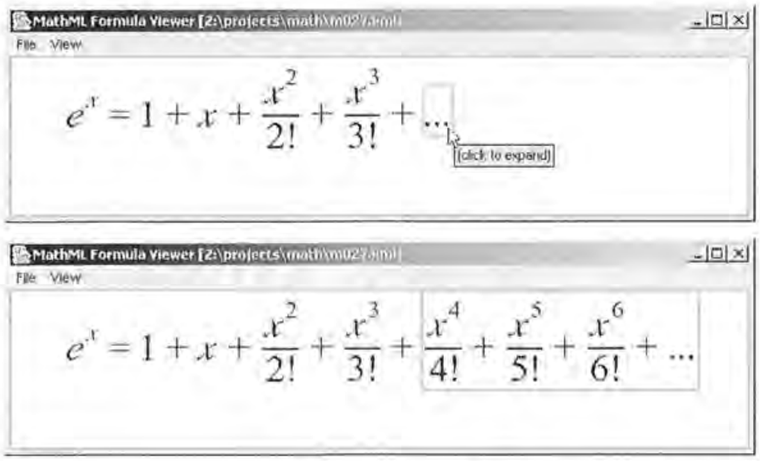
The possibility for dynamic and interactive display has initially been laid in the foundation of MathML. Adding a link to a MathML sub-expression is one basic kind of interactivity, which was provided in MathML recommendation in order to fully integrate MathML into XHTML, the Extensible Hypertext Markup Language [37]. For such integration it should be possible not only to embed MathML in XHTML, but also to embed XHTML in MathML. However, at present, the MathML specification does not permit any XHTML elements within a MathML expression, although this may be subject to change in a future revision of MathML.

In the current design of MathML it is assumed that XHTML elements — headings, paragraphs, lists, etc. — either do not apply in mathematical contexts (what, actually, does not agree with our vision in which we see mathematical artifacts be competent parts of the whole presentation), or MathML already provides equivalent or better functionality specifically tailored to mathematical content (tables, mathematics style changes, etc.). More information on linking in MathML see W3C's MathML recommendation, Chapter 7.1.4.

Many other kinds of interactivity cannot be easily accommodated by generic linking mechanisms. For example, in lengthy mathematical expressions, sometimes it is desirable to be able to shrink/expand expressions, e.g., allow a user to toggle between an ellipsis and a much longer expression that it represents (Figure 6). To create a mechanism for binding actions to expressions, MathML provides the `<maction>` element. This element accepts any number of sub-expressions as arguments, and, when given appropriate attributes, can give the math expression certain amount of dynamics.

a)

b)



$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \boxed{\ldots}$$

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \boxed{\frac{x^4}{4!} + \frac{x^5}{5!} + \frac{x^6}{6!} + \ldots}$$

Dynamic display of mathematical equations my include shrinking or expanding some math terms. In this example the ellipsis in Taylor series representing the function e^x (*a*) can be expanded showing three more terms of the series (*b*).

Such interactivity can be achieved by using a standard MathML construct: `<maction>` element with `toggle` attribute set.

Figure 6. Dynamic display made possible by MathML.

The MathML recommendation suggests several types of actions (i.e., "standard" attributes and their possible values for `<maction>` element): "toggle", "statusline" "tooltip", "highlight", "menu", whose meaning is clear from their names. However, this list, actually, is not *a standard*; it is provided in the recommendation only for illustration purposes. Recognized values and behaviors may vary from application to application, from one rendering tool to another.

MathML Rendering Tool

MathML rendering software already exists [5, 27, 29, 36]. However, none of the available software tools (at least those we are aware of) has all the features we need to implement the required dynamics and interactivity. In the framework of our "big" project we are still at the stage of identifying new ways to present mathematics in a more efficient, dynamic and highly usable manner. We do not yet know all the properties and behavioral patterns that our MathML rendering engine might need to have in the future. Even if we had software package with well-documented API that was designed for rendering MathML, this would not be enough, since it still might have certain limitations, which would complicate creation of the dynamic and interactive objects we need. For example, a MathML web-applet by Waterloo Maple, Inc. [21] does a very good job of displaying equations encoded in MathML, correctly rendering all special characters used in math notations, and even making them interactive by appropriate processing of `<maction>` element (e.g., the one with "toggle" attribute). However, there is no possibility to animate the terms in arbitrary way, nor is there any documented interface to access the math terms from the "outside" of the applet. The applet is a sort of a Thing-in-Itself living its own life and having very little interaction with the environment, except its simple reaction to mouse clicks.

We do not have to be bound by such limitations, and this is because we decided to concentrate on the design of our own MathML rendering engine which would have all the features we need. In some respect, we have to repeat much of the job done by other developers (MathML parsing, formula layout), but this needs to be done to provide us with solid ground in the future: a flexible and configurable tool for dynamic display of mathematics.

The features which we wanted our MathML rendering tool to have were quite "standard": the software had to

- correctly and quickly parse MathML code and do the layout of math on the rendering device;
- provide interface to access [and activate] each math term after the layout is done;
- enable the processing environment to manipulate with math terms (change their color, size, positions).

The development of this software became the core of the current project.

THE IMPLEMENTATION

The general scheme of processing MathML does not differ from processing of any other language (Figure 7-*a*).

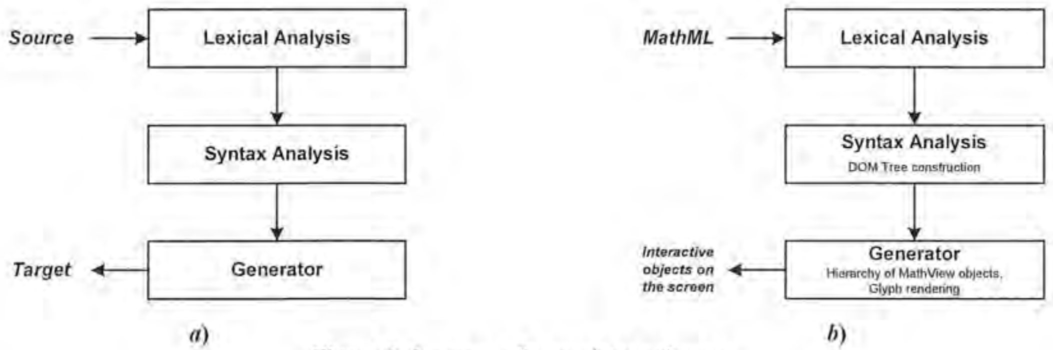


Figure 7. Structure of a translator software

In the case of processing MathML, the Generator does the job of creating an internal representation of math terms and performs rendering them on the output device (Figure 7-*b*). Lexical Analysis and most of the Syntactic analysis is done by the parser software (the exact distribution of jobs is actually dependent on the parser architecture; see discussion below).

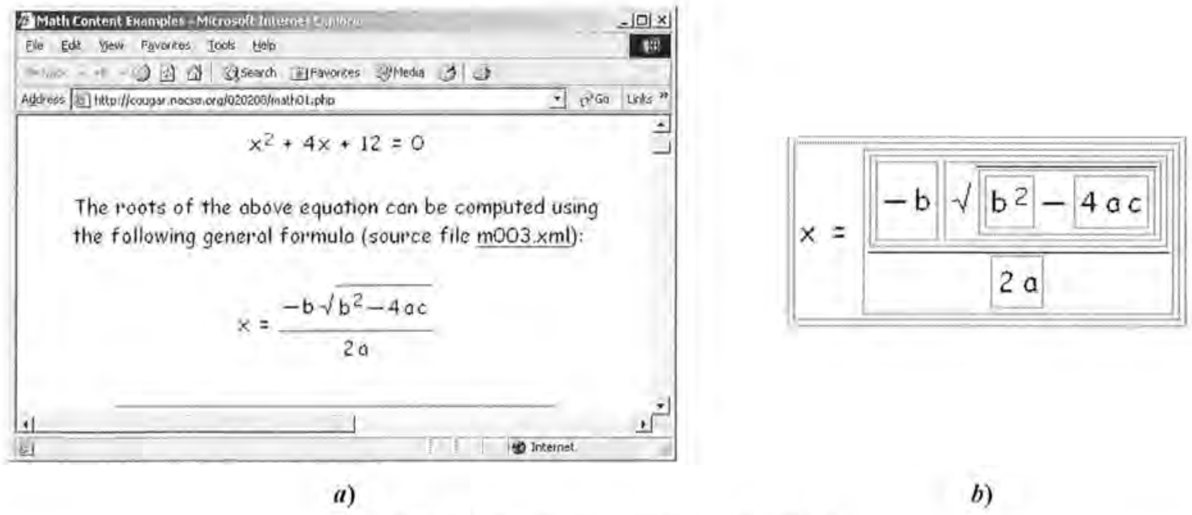
MathML Parsing

Since MathML, in essence, is a particular implementation of general-purpose XML, any good XML parser can be used to parse MathML. The word "good" means that there are some requirements for the parser that must be met to make it useful in parsing MathML.

1. The parser has to correctly process XML tags, and handle errors (should it find any) in a nice manner which means that the whole program should not crash because of a syntax error in the source XML code.
2. The parser must validate the document against its grammar defined in Document Type Declaration (DTD) specifying the valid grammar constructs of the document. With respect to MathML, the validation is important, since MathML uses many symbolic names like `&PlusMinus;`, and also there are some rules specifying how elements can be nested. E.g., the parser should generate an error if an `<mn>` element contains data other than plain text (no inner elements allowed in token elements, such as `<mn>`, `<mi>`, `<mo>`, etc.).
3. The parser must work reasonably fast. We cannot make our readers wait too much before the document is fully loaded and rendered on the screen.

Many XML parsers already exist. For the first experiments, we started with the DOM (Document Object Model) parser that comes with PHP 4.1.1 [30] The parser worked very fast, but had a significant drawback: it did not do the DTD-validation properly. Although we used the XML parser from PHP only for test purposes, its use gave us an insight on how the DOM structures are organized and how they could be used in programs. As expected, it turned out that the procedures for traversing DOM trees all have almost the same architecture, no matter what programming language is used.

Another reason to start with PHP was to quickly create an application which would be capable of parsing documents containing MathML and displaying math terms on the screen. The emphasis was on the XML parsing, and the screen layout of math terms was done by table elements (`<TABLE>` tags of HTML) nested and styled appropriately, so that the documents could be viewed in a web-browser. This approach could help us relay the task of elements' layout to the HTML rendering engine of a web-browser. Although the formula layout done in this way was not perfect (e.g., it did not do display of stretchy characters — the characters such as big parentheses, or long fraction lines), it was a good "test of concept", and formulated the requirements which a parser for MathML must meet. The screenshots of the application is shown on Figure 8-*a*. Figure 8-*b* shows the borders of the nested tables used to lay out the math expression.


Math Content Examples - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Search Favorites Media
Address http://cougar.nacse.org/020208/math01.php
Go Links
Internet
a)
b)

$x^2 + 4x + 12 = 0$

The roots of the above equation can be computed using the following general formula (source file m003.xml):

$$x = \frac{-b \sqrt{b^2 - 4ac}}{2a}$$

Figure 8. Simple "box" rendering of MathML.

We have to mention other XML parsers we came across doing our experiments with displaying mathematics in a web-browser.

There are two built-in DOM XML parsers in MS Internet Explorer 5+, **Microsoft.XMLDOM** and **Msxml2.DOMDocument**. Both work as ActiveX objects in JavaScript. The parsers work reasonably fast but again, do not do DTD validation appropriately. Also, they work only in MSIE 5+ and only on Win32 platform.

The Apache Software Foundation [7] has an initiative — The Apache XML Project [8] — that works on the tools for XML processing. Xerces Java XML parser is one of them. The software is written in Java and distributed freely under the Apache Software License. Xerces parsers are distributed as Java archive (`*.jar`) files; their source code is also available for download at http://xml.apache.org/dist/xerces-j/. The parsers are highly configurable, and they have a well-documented Application Program Interface (API).

First experiments with the parsers showed that the Xerces XML DOM parser was very good at validating XML documents against DTDs, although the parsing was done slower than that with the parser from PHP.

The PHP application described earlier was re-written to a JSP (Java Server Pages [18]) application. The program did the same processing of XML document with resolving correctly all the entities defined in the DTD of MathML. However, the program worked much slower, and the speed of parsing seemed to be not very much dependent on the size of the document. Later experiments, in which we used the same parser in Java application, showed the same unsatisfying result. One of the reasons is that MathML has a relatively complex grammar (`mathml2.dtd` plus 23 files with entity definitions), whose parsing takes about 1–3 seconds. The situation can be improved if we pre-parse the grammar at the time of MathML rendering engine initialization, and then use the "binary" representation of the grammar in the process of parsing the document. The possibility of such pre-processing is provided in the Xerces parser interface, but it is documented somewhat vaguely, which by now has prevented us from creating a reliable and efficient implementation of the grammar pre-parser.

DOM parsers create complete tree of DOM objects, which afterwards can be used to do the necessary manipulations on the document. Such manipulations may include not only reading the document's content, but also changing its structure. With such design, DOM parsers are more time- and resource-consuming (parsing process is full of memory allocations for the DOM tree nodes, and after parsing the whole DOM tree must be kept in memory), however, experiments with Xerces parser showed that DOM parser is almost as fast as an event-driven SAX (Simple API for XML) parser with the same functionality. The possibility to change document's structure might be necessary for creation of interactive MathML documents, or providing interactivity to existing documents.

The PHP and JSP application have been run on an Apache HTTP-server with PHP4.1.1. and Tomcat 4 JSP Servlet/JSP [33] running on Windows 2000 machine in NACSE (http://cougar.nacse.org/).

`MathView` Objects Hierarchy

Initially, we were creating our own hierarchical objects for displaying math terms. However, it turned out that Java classes already had appropriate structures for building the hierarchy — the Container-Component hierarchy (Java classes Component and Container, `java.awt` package), which fit nicely to the concept of "terminal" and "non-terminal" elements of math terms. The terminal elements are *identifiers*, *numbers*, *operators* and *plain text* encoded in MathML by the tags `<mi>`, `<mn>`, `<mo>`, `<plus/>`, `<minus/>` and `<text>`. Non-terminal elements are all other markup elements that are used to represent displayable objects (e.g., elements sub/superscripts, encoding fractions, etc.). Non-displayable elements (such as `<style>` or `<maction>`) are processed in such a way that they become attributes of displayable elements, or they are represented in auxiliary data structures responsible for dynamic features.

Every math expression encoded in MathML can be naturally thought of as tree structure. Each node in the tree corresponds to a particular layout schema, and its branches or child nodes correspond to its subexpressions. This structure nicely fits into the Container-Component hierarchy. The hierarchy of `MathView` objects descending from the `java.awt.Container` class is shown on Figure 9.

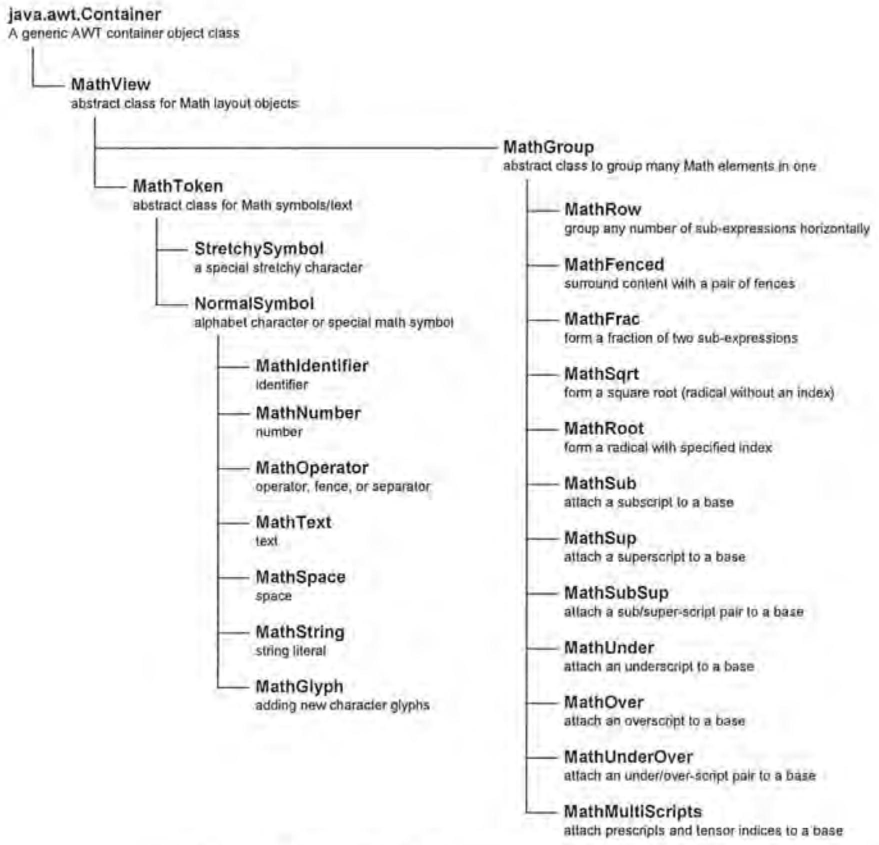
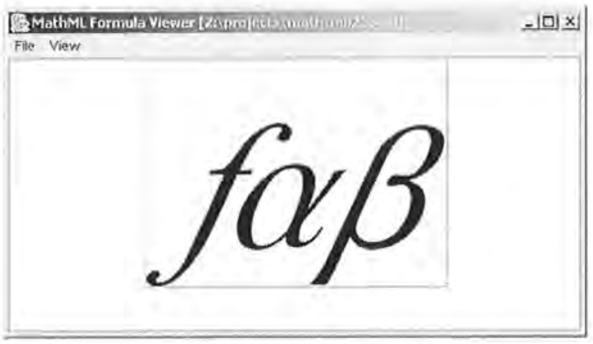

java.awt.Container
A generic AWT container object class
MathView
abstract class for Math layout objects
MathGroup
abstract class to group many Math elements in one
MathToken
abstract class for Math symbols/text
StretchySymbol
a special stretchy character
NormalSymbol
alphabet character or special math symbol
MathIdentifier
identifier
MathNumber
number
MathOperator
operator, fence, or separator
MathText
text
MathSpace
space
MathString
string literal
MathGlyph
adding new character glyphs
MathRow
group any number of sub-expressions horizontally
MathFenced
surround content with a pair of fences
MathFrac
form a fraction of two sub-expressions
MathSqrt
form a square root (radical without an index)
MathRoot
form a radical with specified index
MathSub
attach a subscript to a base
MathSup
attach a superscript to a base
MathSubSup
attach a sub/super-script pair to a base
MathUnder
attach an underscript to a base
MathOver
attach an overscript to a base
MathUnderOver
attach an under/over-script pair to a base
MathMultiScripts
attach prescripts and tensor indices to a base

Figure 9. Hierarchy of `MathView` objects (fragment).

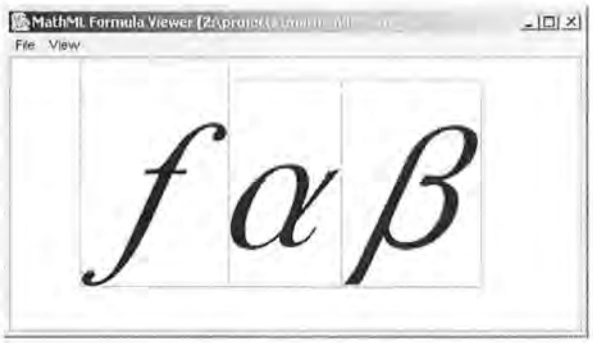
The hierarchy of displayable math objects is built after the parser creates the DOM tree. The DOM tree is traversed from the root node up to the every leaf (a terminal token element), and all math objects are created along the way. A small observation: the internal hierarchical structure of multiply-nested math elements is invariant for font-style, color, and many other properties defining the appearance of the element on the screen. So, the building of the hierarchy of the math terms, which requires many operations with the dynamic memory allocation (object creation and initialization) which are relatively time-consuming, is done only once. The arrangement and resizing of the elements according to the new font style/size/color is done when the math expression is about to be redrawn on the screen. During the *arrangement* stage, the tree of math objects is traversed recursively, and layout rules are applied to every element. The layout rules for math formulas are specified in every detail in Appendix G of Knuth's TEXbook [19]. Since the procedure of arranging the components deals (mostly) with arithmetic operations and does almost no memory allocations, the arrangement is done very quickly.

Glyph Rendering

However, the arrangement of math terms is not that simple, as it could be thought of at a glance. The principal problem is that we cannot consider every math term only as a rectangular object (as we did in the first experiments with PHP/JSP applications, when we used nested tables). For a professional-looking layout of math terms the layout procedures must always take into account the *contents* of the terms. Fortunately, Java provides all necessary classes and interfaces for access to and manipulations of font properties. For example, Figure 10 shows the placement of the math terms with (*a*) and without (*b*) taking into account their contents. As we can see, information about the contents (e.g., the *advance* property of the text string in a math token) is very important for proper placement of the characters.



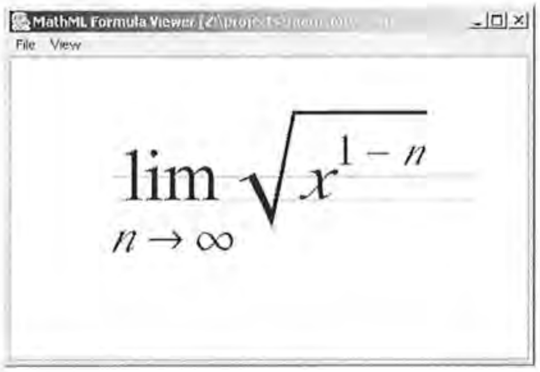
a) Content-dependent layout.



b) Simple "box" layout.

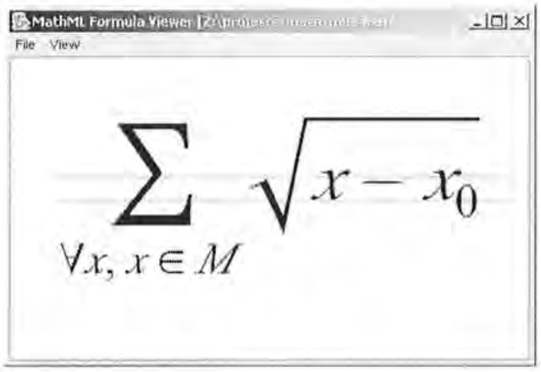
Figure 10. Layout of math terms (variables).

Another big issue arises from the fact that MathML has relatively small number of lexical constructs, which are to be used to represent a large variety of mathematical terms. As a result, the same constructs must be rendered differently, depending on the contents they have. For example, the same MathML element `<munder>` can be used both for encoding limits and summations (Figure 11 shows the source MathML code and how it is rendered in these two cases). In the former case the size of the font used for the token "lim" is the same as the font size for the element on this level of nesting and the baseline of "lim" is the same as the baseline of the whole expression, whereas in the latter case the summation symbol must be made about 150% of the default font size, and the centerline of the symbol must be aligned with the centerline of the following expression. Lots of other similar exceptions documented in [24] must be processed by a MathML rendering engine.



```
<math>
  <mrow>
    <munder>
      <mo>lim</mo>
      <mrow>
        <mi>n</mi>
        <mo>&rarr;</mo>
        <mo>&infin;</mo>
      </mrow>
    </munder>
    <msqrt>
      ...
    </msqrt>
  </mrow>
</math>
```

a) The baseline of "lim" operator is aligned with the baseline of the whole expression.



```
<math>
  <mrow>
    <munder>
      <mo>&sum;</mo>
      <mrow>
        <mo>&forall;</mo>
        <mi>x</mi>
        ...
      </mrow>
    </munder>
    <msqrt>
      ...
    </msqrt>
  </mrow>
</math>
```

b) The summation operator symbol is aligned according to the center line of the whole expression.

Figure 11. Content-dependent layout of math operators.

Symbol Output

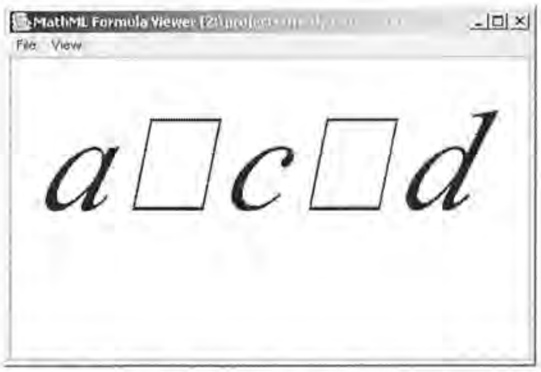
Since we are dealing with mathematical expressions, many special characters are to be used. Java SDK has a number of classes and interfaces to work with non-standard glyphs. The principal problem is that ordinary procedures for font output cannot be used in the task on output of special characters (Greek or "fractur" symbols, stretchy symbols, etc.) Due to the "platform-independecy", Java graphic text output capabilities are very limited, and special procedures must be used for non-standard font output. In this project, all symbol output is done using the classes operating with glyph and glyph vectors, which allows the user to see the symbols on the screen exactly as they are in the font files, not dependent on the current language settings of the system. For ordinary symbols from the first part of ASCII table this approach might seem too complicated, but this is the only way to put out stretchy characters such as big parentheses, brackets, or root symbols.

The general scenario of symbol output is as the following:

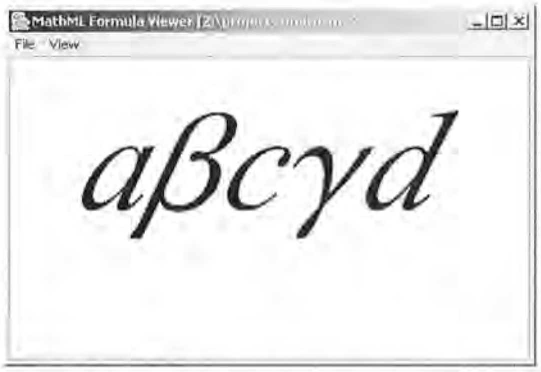
The `MathView` object is initialized with a certain contents — a string of Unicode characters. Before the drawing procedure is called, all `MathView` objects must be resized and laid out according to the current font size. The arrangement is done in the virtual procedure `arrange()`. For token objects (descendants from `MathToken` class) this procedure works with the glyphs representing the symbols, and for the non-terminal objects the procedure does the appropriate positioning of insider objects. In Both cases `arrange()` determines the new screen size of the `MathView` objects in the current graphics context.

For token objects the procedure `arrange()` takes the Unicode string, "looks" what characters are in it, creates a corresponding glyph vector, which is used to determine some font-related properties (e.g., baseline, advance, centerline, etc.). The glyph vector is then transformed to a Shape object, using which the procedure determines the visual sizes of the string, which are used to set the size of the `MathView` object itself. In non-terminal objects `arrange()` applies positioning rules depending on what class this object is an instance of.

If a Unicode string contains characters which must be taken from different fonts, the use of glyph vectors and their transformation to Shape objects is necessary, since it allows us to do a much nicer layout. Figure 12 shows the sample string containing a mix of Latin and Greek characters. When all characters are taken from the same font, some symbols might "drop" being rendered as empty rectangles (Figure 12-*a*). Using our approach with glyph output, all characters are rendered appropriately (Figure 12-*b*). Although it is very unlikely that a math variable encoded in a single MathML tag would contain a mix of characters, such approach makes the symbol output procedure universal.



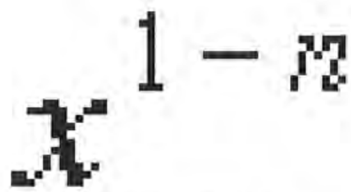
***a*)** All glyphs are taken from the same font



***b*)** Each glyphs is taken from an appropriate font.

Figure 12. Rendering of a Unicode string containing a mix of Latin and Greek characters

Shape-based output might create a small problem with the output of characters when the font size is small (e.g., 10pt). Most probably, this is the result of the double-to-integer approximation when the double-precision coordinates of the Shape are cast to the integer coordinates of the output graphics context (see Figure 13-*a*). The problem is still actual for the font sizes which are much bigger: the characters look ugly even when the font size is 14…16 pt, the size which is usually used in many document browsing tools. Enabling *antialiasing* in the rendering context solved the problem. Now even very small characters do not lose their pixels. The approximation is done by adding some grayscale pixels to the output image (Figure 13-*b*)


a) No antialiasing

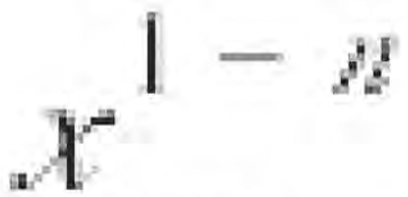

b) Antialiasing is on.

Figure 13. Use of antialiasing in glyph output.

Stretchy Characters

Stretchy characters are composed of several (1, 2, 3, or 4) parts. They usually have some "terminal" parts which are "glued" together with "extenders". Figure 14 shows examples of such characters. The procedure of output of these characters is very similar to that of normal symbols, except that with the stretchy characters the size of the symbols is known *before* the symbol is "arranged" and drawn on the screen. When the size is pre-defined, what is left for the drawing procedure is to arrange the "terminal" elements of the character and "glue" them with extender parts stretched appropriately. The stretching is usually done by putting as many extenders as needed next to each other (or one on top of the other) to achieve the desired size. In this project, however, we use another approach: the extenders are stretched by applying affine transformations to them. This allows us to make the whole character be precisely the size it needs to be. With simple repetition of extender parts such precision is generally impossible to achieve.

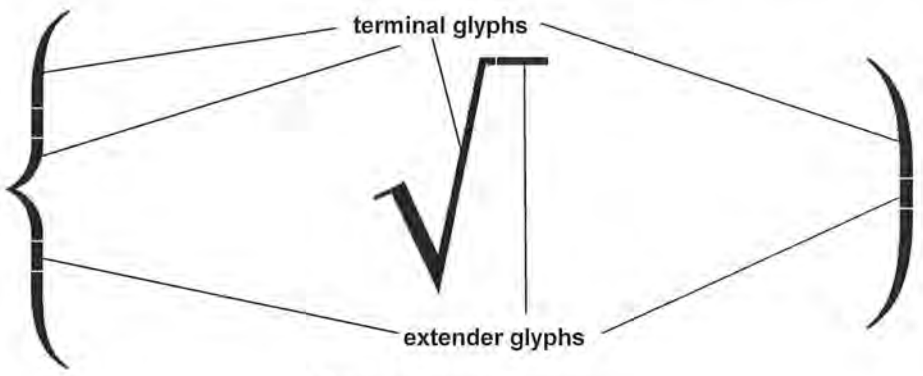
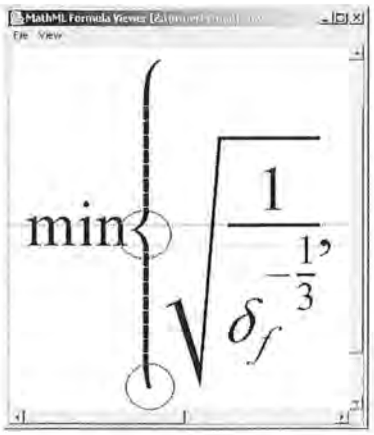
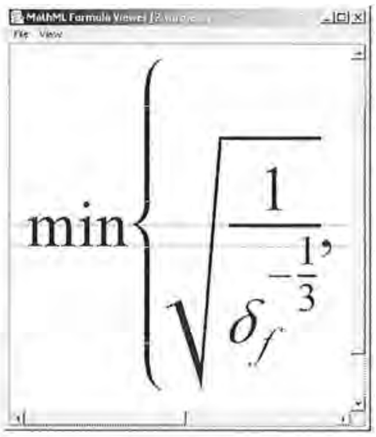


Figure 14. Stretchy characters.


a) Stretching is done by repetition of the extender glyph. Notice the inappropriate alignment of the center part of the left brace and a cut-off of its bottom part.


b) Stretching is done by stretching the extender glyph using affine transformations of the shape. The alignment and the size of the brace is precise.

Picture 15. Two ways of stretching the characters.

The output of stretchy characters is very complicated. Thanks to Yuemei Sun who did a great job of writing the general procedures for stretchy symbols output, based on which I wrote the subroutines that are now used in this project.

Object Activation

The principal reason why we started designing our own rendering engine is that no MathML renderers that we could find had any interactive features. At least, there were no documented API for creation of interactive formulas. At a glance, the MathML rendering software we developed might seem to have no difference from similar software already existing. However, we had to design our own rendering engine for the future tasks of making math content in on-line documents be dynamic and interactive.

Had the ultimate purpose of the program been only to display the formulas, the complex hierarchy of `MathView` objects would not have been needed. But for the purposes of making the math display dynamic it was necessary to have every math term be accessible in some way after the math expression is drawn on the screen. This is because we descend the class hierarchy of our objects from Java's Component-Container classes. Those classes can be easily extended to support handling of mouse events, and they provide us with possibility to access each component (math term) independently: every math term (terminal or non-terminal) can be re-drawn/hidden/shown/moved arbitrarily when needed.

The *activation* API of the math terms include methods for firing activation events in response to user's actions. In our first design we do the *synchronous highlights* of related math terms: when a mouse pointer is moved over a math term, the term is highlighted (changes color, or draws itself in a thin frame) and all or some related terms are also highlighted.

Every math term that needs to be identified is assigned a cross-reference identifier (*xref* parameter in the corresponding MathML tag). When the source document is parsed, the rendering engine creates additional data structures to store the information about all objects identified with *xref*.

For the purposes of synchronous highlight, we accept a simple model of a directed activation graph: an object may activate other objects, if there are directed arcs from a node representing that object to nodes representing the objects that must be activated synchronously with it (Figure 16).

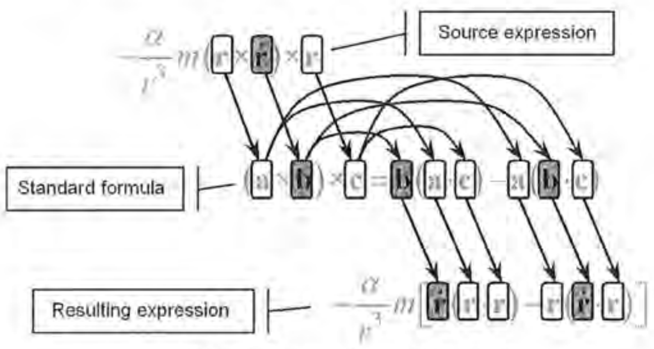
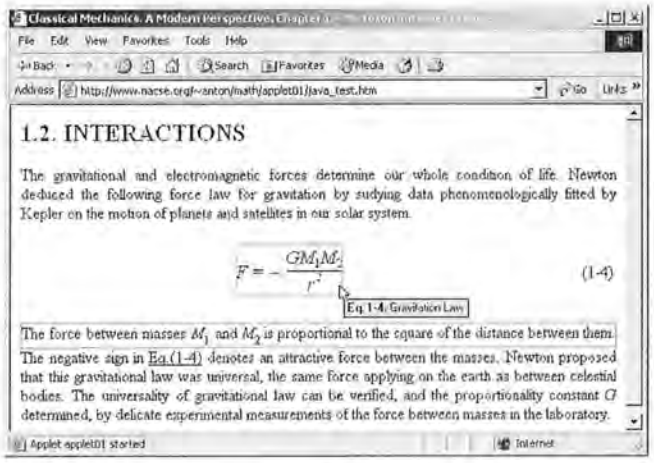


Figure 16. Activation Graph (fragment).

By doing in this way we can, e.g., highlight some variable in an equation, and (synchronously with that) highlight the corresponding fragment in the surrounding text containing, e.g., description of the variable or some other expression (Figure 17, Figure 18-*c*). Also, considering activation chains as a directed graph, we can assign certain properties to its edges, thus providing information about the semantics of the links between the terms. Such information might appear as callouts, or show up in the status bar of the window that displays the document.


Classical Mechanics. A Modern Perspective. Chapter 1
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://www.nacse.org/~anton/math/applet01/java_test.htm
Go Links
Eq 1-4: Gravitation Law
Applet applet01 started
Internet

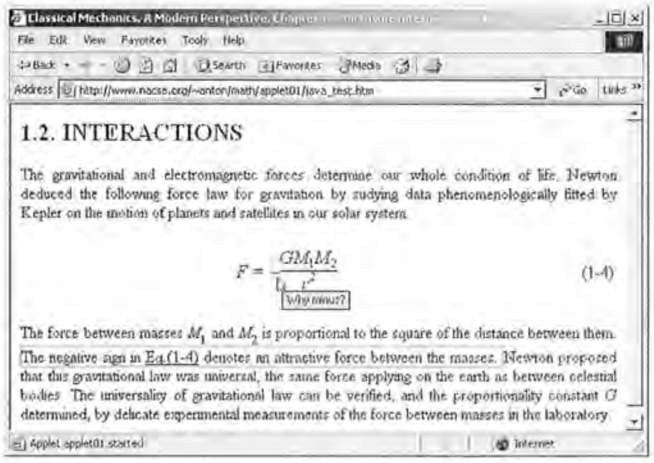
1.2. INTERACTIONS

The gravitational and electromagnetic forces determine our whole condition of life. Newton deduced the following force law for gravitation by sudying data phenomenologically fitted by Kepler on the motion of planets and satellites in our solar system.

$$F = -\frac{GM_1M_2}{r^2} \quad (1\text{-}4)$$

The force between masses M_1 and M_2 is proportional to the square of the distance between them. The negative sign in Eq.(1-4) denotes an attractive force between the masses. Newton proposed that this gravitational law was universal, the same force applying on the earth as between celestial bodies. The universality of gravitational law can be verified, and the proportionality constant G determined, by delicate experimental measurements of the force between masses in the laboratory.

a)


Classical Mechanics. A Modern Perspective. Chapter 1
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://www.nacse.org/~anton/math/applet01/java_test.htm
Go Links
Why minus?
Applet applet01 started
Internet

1.2. INTERACTIONS

The gravitational and electromagnetic forces determine our whole condition of life. Newton deduced the following force law for gravitation by sudying data phenomenologically fitted by Kepler on the motion of planets and satellites in our solar system.

$$F = -\frac{GM_1M_2}{r^2} \quad (1\text{-}4)$$

The force between masses M_1 and M_2 is proportional to the square of the distance between them. The negative sign in Eq.(1-4) denotes an attractive force between the masses. Newton proposed that this gravitational law was universal, the same force applying on the earth as between celestial bodies. The universality of gravitational law can be verified, and the proportionality constant G determined, by delicate experimental measurements of the force between masses in the laboratory.

b)

Figure 17. Synchronous highlight of math terms and their explanations.

In the current design every math term that can be activated has an identifier. Also, every term has a list of "child" elements that are prone to activation when their parent is activated. Encoding of such a construct can be done by adding to MathML tags attribute *children* whose value can be a set of id's of other math terms separated by commas (thus forming a sort of *adjacency list*,. However, this is not a proper solution since it adds to MathML tags new non-standard attributes not documented in W3C's MathML recommendation. Instead, the source document should have separate sections that would encode the behavior of the identified math terms.

Transition Encoding/Rendering

A similar graph model is used in the encoding of transitions. At present, transitions are encoded as an XML element with two sub-elements describing two states, s_0 and s_1 — before and after the transition, and additional sub-elements used for indication what math object from state s_0 becomes what in state s_1. For the encoding of the states and establishing links between related objects we use cross-reference identifiers of the math objects in the document.

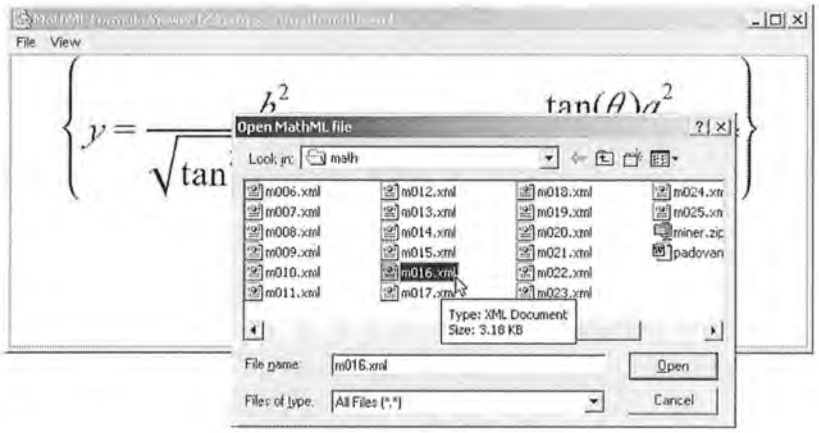
Such design gives us certain freedom in choosing how the transitions will be rendered. At this point in time we consider showing the transitions step-by-step (two states at a time with corresponding math elements highlighted and/or connected by arrows, see Figure 4), but in future we can use animations in addition to highlights in order to show how one math artifact becomes another, should this technique prove to result in a better user's satisfaction.

Since the activation chains and transitions are modeled as digraphs, their encoding can simply follow the XML rules for encoding graphs [15].

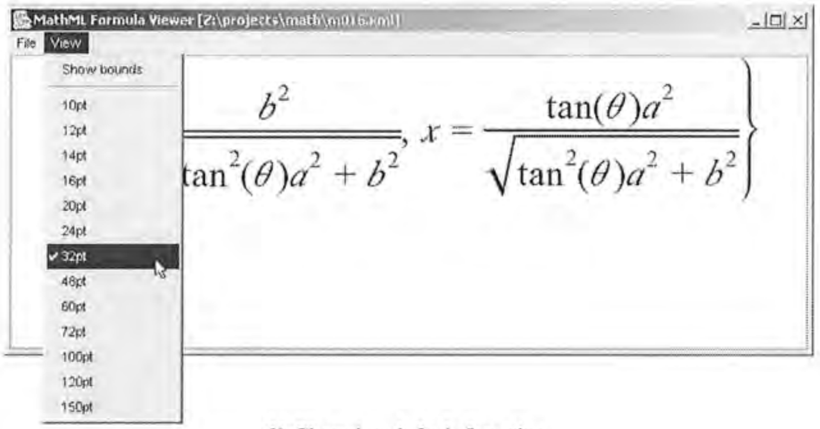
MATHML FORMULA VIEWER

Currently, the MathML rendering engine is implemented as a part of a MathML formula viewer — a Java application which allows loading math expressions encoded in MathML and viewing them with different default font sizes (Figure 18 – *a*, *b*). When the mouse pointer is moved over certain math terms, those terms and other terms related to them are synchronously highlighted (Figure 18-*c*).

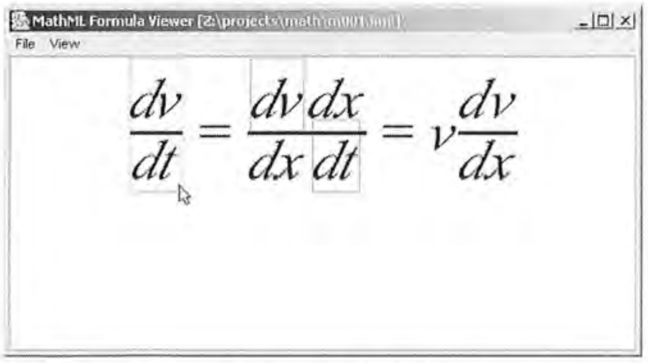
The same Java classes can be used in creating applets to include mathematics on web-pages (Figure 17).



a) Loading a MathML file for viewing.



b) Changing default font size.



$$\frac{dv}{dt} = \frac{dv}{dx}\frac{dx}{dt} = v\frac{dv}{dx}$$

c) Showing dependencies by synchronous highlight of the related math terms.

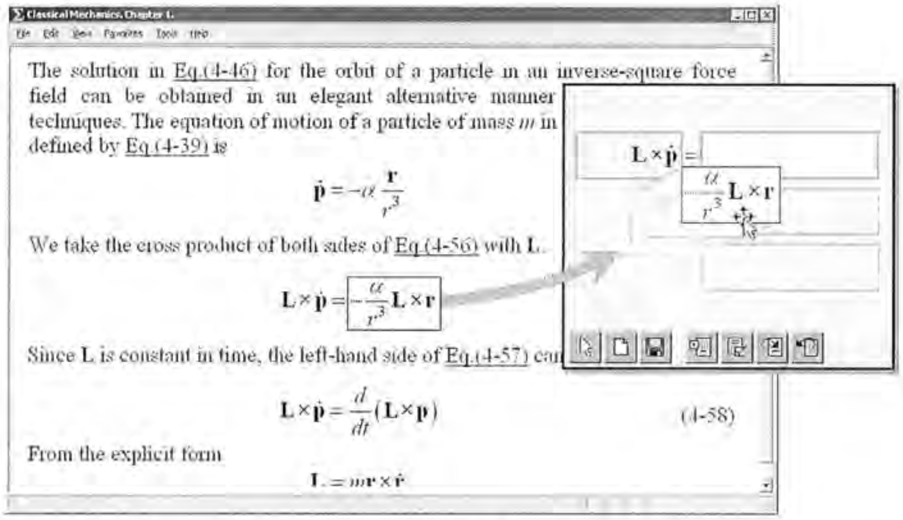
Figure 18. MathML formula viewer application.

FUTURE WORK

The MathML rendering software we have developed is not yet perfect. There are still many issues which need further consideration and implementation.

1. We need to implement rendering of all MathML constructs (all elements, all attributes, style rendering). Although providing support for additional MathML tags is rather a mechanical job of writing appropriate class definitions and overriding their `arrange()` subroutine, implementation of *style* rendering is not that simple. MathML recommendation suggests using Cascading Style Sheet (CSS) technology [11] in MathML documents. For us this means that in order to provide full support for styles in MathML we will have to use an appropriate (non-XML) parser to validate and parse the values of `style` attribute.

2. The MathML rendering engine will probably need to have an interface to serve as a MathML authoring tool. One of the useful feature we can think of is that the software provides possibility for drag-and-drop operations which can be used to move equations from one display frame to another (Figure 19). For authoring of the static math content we can use existing software tools like Amaya [5] or MathType [25], but new dynamic features we are going to introduce will require the math editor software to have additional editing capabilities corresponding to the display capabilities of our rendering engine. Also, in the task of authoring of the documents with interactive and dynamic content the authors will have to carefully consider syntactic aspects of the math presentations. In addition to providing nice layout of math expressions, the authors will have to ensure that their formulas and equations have appropriate internal structure with all terms properly annotated and no links broken. The authoring tool must be able to automatically check internal integrity of the documents being created.


Classical Mechanics. Chapter 1.
File Edit View Favorites Tools Help

The solution in Eq.(4-46) for the orbit of a particle in an inverse-square force field can be obtained in an elegant alternative manner techniques. The equation of motion of a particle of mass m in defined by Eq.(4-39) is

$$\dot{\mathbf{p}} = -\alpha\frac{\mathbf{r}}{r^3}$$

We take the cross product of both sides of Eq.(4-56) with **L**.

$$\mathbf{L}\times\dot{\mathbf{p}} = \boxed{-\frac{\alpha}{r^3}\mathbf{L}\times\mathbf{r}}$$

Since L is constant in time, the left-hand side of Eq.(4-57) can

$$\mathbf{L}\times\dot{\mathbf{p}} = \frac{d}{dt}(\mathbf{L}\times\mathbf{p}) \qquad (4\text{-}58)$$

From the explicit form

$$\mathbf{L} = m\mathbf{r}\times\dot{\mathbf{r}}$$

$\mathbf{L}\times\dot{\mathbf{p}} =$
$\frac{\alpha}{r^3}\mathbf{L}\times\mathbf{r}$

Figure 19. Authoring the Formula Derivation Wizard (prototype).

3. At some point in time we might be considering working with math knowledge repositories. To work with math content we will need to use efficient algorithms for rendering content markup (e.g., OpenMath or MathML). Although the user-interface features we are working with are primarily related to how the information is presented, in our project the presentation is tightly connected with the underlying mathematical content of the documents.

4. Up to now we did not concentrate too much on the efficiency issues. However, the time has come to start thinking of how to optimize the current implementation of our MathML parser. So far, the parsing time is unacceptably long. The procedures creating internal tree of `MathView` objects are also need to be optimized to reduce the amount of time required for their creation and initialization.

References

1. ACM Digital Library. [on-line] http://www.acm.org/dl/
2. Abbott, J., Diaz, A., Sutor, R.S. A report on OpenMath: a protocol for the exchange of mathematical information. In ACM SIGSAM Bulletin, Vol.30, No.1, pp. 21–24, 1996.
3. Algebra Interactive! [on-line] http://www.win.tue.nl/~ida/home.html
4. ALGED — Algebra Editor, Software (DOS). http://www.geocities.com/paris/6502/alged35
5. Amaya. W3C's Open Source Editor/Browser. http://www.w3c.org/Amaya/
6. Andres, E., Melis, E., et al. ActiveMath: System Description. Artificial Intelligence in Education 2001.
7. Apache Software Foundation, The. [on-line] http://apache.org/
8. Apache XML Project, The. [on-line] http://xml.apache.org/
9. Barger, V., Olsson, M., Classical Mechanics. A Modern Perspective. McGraw-Hill, Inc.
10. Bell, J.E., Doppelt, A.M., Hughes, J.F. Developing an Interactive Illustration: Using Java and the Web to Make It Worthwhile. Proceedings of 3D and Multimedia on the Internet, WWW and Networks, 1996.
11. Cascading Style Sheets, Level 2. W3C Recommendation. [on-line] http://www.w3.org/TR/REC-CSS2/
12. ComputerMentor (Win32). Available at http://www.computermentor.da.ru/
13. Exploratory Project. Department of Computer Science, Brown University. http://www.cs.brown.edu/exploratory/
14. FORMULA (DOS). Available at http://www.exponenta.ru/soft/others/formula/formula.asp
15. GXL, Graph eXchange Language. [on-line] http://www.gupro.de/GXL/
16. Hypertextual Electronic Library of Mathematics. [on-line] http://www.cs.unibo.it/helm/
17. IBM Techexplorer Hypermedia Browser. [on-line] http://www-3.ibm.com/software/network/techexplorer/
18. JSP (web-scripting technology). [on-line] http://java.sun.com/products/jsp/
19. Knuth, D.E., The TEXbook. Addison-Wesley Publishing Co., 1984. — ISBN: 0201134489.
20. Live Pictures for problems on Plain Geometry [on-line] http://zadachi.mccme.ru:8101/njava/
21. Maple. Waterloo Maple, Inc. [on-line] http://www.maplesoft.com/products/Maple8/index.shtml
22. Mathematica. Wolfram Research Inc. [on-line] http://www.wolfram.com/
23. Mathematical Discussion System. [2000]. [on-line] http://sourceforge.net/projects/mds/
24. MathML. W3C Recommendation. http://www.w3.org/Math/
25. MathType. [on-line] http://www.mathtype.com/
26. Matlab. The MathWorks, Inc. [on-line] http://www.mathworks.com/
27. Mozilla Open Source Web Browser. [on-line] http://www.mozilla.org/
28. Multimedia Tools for Communicating Mathematics. Springer Verlag, 2002 — ISBN: 3540424504.
29. Padovani L. A Stand-Alone Rendering Engine for MathML. MathML Conference, 2002. [on-line] http://www.mathmlconference.org/2002/presentations/padovani/
30. PHP (web-scripting technology). [on-line] http://www.php.net/
31. Project Mathematics! [on-line] http://www.projectmathematics.com/
32. Simpson, R.M., Spalter A.M., van Dam, A. Exploratories: An Educational Strategy for the 21st Century, in Proceedings of ACM SIGCSE '99, 1999.
33. Tomcat 4 Servlet/JSP. The Apache Jakarta Project. [on-line] http://jakarta.apache.org/tomcat/
34. Tufte, E.R. Envisioning Information. 7th Ed. Graphics Press, 1999.
35. Tufte, E.R. The Visual Display of Quantitative Information. 2nd Ed. Graphics Press, 2001.
36. WebEQ/Math Player. Design Science, Inc. [on-line] http://www.dessci.com/webmath/
37. XHTML, The Extensible Hypertext Markup Language. [on-line] http://www.w3.org/TR/xhtml1/