Published in **Innovation in Mathematics**. Proceedings of the Second International *Mathematica* Symposium, Rovaniemi, Finland, 29 June - 4 July 1997, V. Keränen, P. Mitic, A. Hietamäki (Ed.), pp. 129 - 136

Generating efficient 3D graphics animation code with OpenGL from object oriented models in Mathematica

Vadim Engelson[1], Peter Fritzson[1], Dag Fritzson[2]

[1] *Department of Computer and Information Science, Linköping University, S-58183 Linköping, Sweden, {vaden,petfr}@ida.liu.se*

[2] *SKF Nova AB, Chalmers Teknikpark S-41288 Göteborg, Sweden, dag@nov.skf.se*

Abstract

Traditionally 3D plots of parametric functions expressed in *Mathematica* are computed interpretively and saved in a static form before display. This causes low graphic performance. In this paper we describe an approach to generate efficient C++/Fortran90 code from such functions. This code is linked together with a powerful 3D browsing environment and uses OpenGL with possible hardware support. Thus flexibility of interactive exploration of 3D scenes and animation options become available for the end-user.

1 Introduction

1.1 The visualization problem

Numerical experiments based on mathematical models is one of the most prevalent classes of applications of high performance computers and workstations. A common problem is however to interpret and make use of numerical data produced from such experiments. High performance numerical programs usually generate vast amounts of data (in our applications 1-2 Gbyte or more). Currently we work with industrial applications such as mathematical modelling and simulation of Rolling Bearings in cooperation with SKF - the world's leading bearing manufacturer. Advanced 3D visualization or animation is very important *for* the end-users to help in interpreting the results.

Typically we visualize a parametric function F_v which can be expressed as $F_v(u) = F_m(F_d(u))$ i.e. it is a composition of functions. A common case is that F_d represented in tabular form (e.g. voluminous generated or measured data) and F_m is defined by the user in *Mathematica* . For example, a table contains position of body (x_i, y_i) relative to coordinate system C for time moments t_i. We need to visualize the acceleration vector movements for time period [t_0,t_n] in another coordinate system. Visualization requires efficient input of F_d, optimized evaluation of F_m, and hardware graphic acceleration for rendering 3D surfaces defined by F_v.

Universal visualization and animation tools available on the market are often insufficient for high performance industrial applications. Performance is not sufficient, options to call user-specified functions are limited or absent, options to configure visualization are limited. In practice, for most simulation application domains, specific application dependent visualization support is usually required.

In this paper we address the problem of visualization of mathematical models, particularly models expressed in the symbolic mathematical and programming language *Mathematica* [1] and an object-oriented extension of this language called *ObjectMath* [2,3,4].

1.2 The *Mathematica* system

The *Mathematica* system includes the built-in functions `Plot[`f`[`x`],`$\{x, min, max\}$`]` and `Plot3D[`f_2`[`x`,`y`],`$\{x, min, max\}$`,`$\{y, min, max\}$`]` which can generate 2D and 3D plots of the functions $f(x)$ and $f_2(x, y)$. These functions can be defined symbolically.

As result, creating a 3D picture of realistic application models e.g. of Rolling Bearings developed in cooperation with SKF takes far too long (more than an hour).

The performance of `Plot` and `Plot3D` is currently relatively low for the following reasons:

- The functions f and f_2 are computed symbolically and interpretively. They are not compiled to efficient machine code.
- Graphics is rendered through a universal interface (PostScript format). Hardware graphic acceleration is not used at all.

1.3 Related systems for output of *Mathematica* graphics

A *Mathematica* to VRML compiler. The *Mathematica* to VRML (Virtual Reality Modelling Language) compiler [5] allows generation of files with point coordinates which describe sets of vectors and polygon surfaces. Such files describe sets of graphical objects that can be explored using a VRML browser. The performance is relatively good because VRML browsers often use hardware graphic accelerators. This approach has the disadvantage of being rather static and not allowing dynamic aspects of graphics. Animations of realistic size cannot be implemented this way since a sequence of VRML files will occupy large amounts of space. Reading these files slows down the animation. If picture resolution (the level of detail) needs be changed, the plotted function must be recomputed on a more fine grained grid. This requires the generation of a new VRML file.

LIVE and similar systems. In these systems the graphics created by the `Plot3D` is converted to sets of 3D vectors and polygons and sent (via `MathLink`) to a graphical 3D browser. By using such a browser the users can rotate, zoom, change textures of the surfaces etc.

This approach allows/supports more interactivity between the *Mathematica* system and the browser. But the function computation is still performed within *Mathematica* and the speed is not sufficient.

Examples of these systems are the LIVE tool (from True-D Software) [6], MathLive and MathLive Pro tools (from Milo Hedge Ltd) [7] and Dynamic Visualizer (from Wolfram Research) [8]. In the Table 1 we try to identify the bottlenecks in the techniques described above.

Other graphical systems. Why not just buy an existing visualization package? Unfortunately, most available software known to us, e.g. GLView [9] and AVS [10], require that the geometry of visualized objects is specified outside the program and is read from an external file. The geometry of complex bearing models, specially with structural deformations, may consist of more than 10 - 50 thousand polygons; their coordinates should be rapidly computed and it would be highly inefficient to store them outside the visualizing program. Therefore these systems do not match our requirements.

The animator/visualizer should also include high-performance functions for reading data files produced from simulations and for computation of polygons. This option is absent in available systems (or is available, but inefficient).

Another problem is that no available visualization package accepts mathematical models as input.

The AVS supports construction of process networks. These networks may contain geometry viewers as well as components written by the user in C which, e.g., perform computations and fast I/O. For the moment the speed of communication between the processes is not high enough for our visualizations.

Techniques	Tools
Processor(s) running optimized code	*MAGGIE*
Hardware accelerated graphics	*MAGGIE* , VRML
Binary, fixed size data I/O	*MAGGIE*
Textual data I/O with parsing	VRML, Live
Traditional graphical standards (OpenGL)	VRML, Live, (*MAGGIE*)
Special universal graphics (*Mathematica* PostScript)	*Mathematica*
Mathematica internally compiled functions	*Mathematica* , VRML, Live
Mathematica interpreted functions	*Mathematica* , VRML, Live

Table 1: Relative performance comparison (in descending order) of different techniques. *MAGGIE* is our new tool proposed in the paper. LIVE represents also MathLive and Dynamic Visualizer in the table.

2 A code generation approach

The approach proposed in the paper uses *code generation* techniques to produce efficient implementations of visualization code.

The input to the code generator is an object-oriented mathematical model expressed in the *ObjectMath* [2,3,4] extension of *Mathematica* . The model contains parametric surface functions that describe the graphic objects to be visualized. The output of the tool is a system that contains a graphical interactive environment which allows the user to explore the objects and functions expressed in the model.

The current prototype implementation, called *MAGGIE* (Mathematical Graphical Generated Interactive Environment) has shown feasibility of this approach. In this implementation *Mathematica* functions are converted to C++ functions and an OpenGL-based graphical environment is constructed.

The following components are discussed in this paper:

- The *ObjectMath* compiler and code generator which needs *type declarations* to generate efficient code.
- The graphical *interactive environment*. The generated code is linked together with this environment.
- A *hierarchy of objects* is introduced to describe complex scenes, consisting of several 3D bodies described through sets of parametric functions. Every 3D body is defined as a collection of surfaces; every surface is defined by a parametric function $f(u, v) = (x, y, z)$ and ranges $u \in [u_{min}, u_{max}]$, $v \in [v_{min}, v_{max}]$.

  The functions we use have two input scalar parameters and give out a 3D vector that represents a point on the surface. An optional input scalar parameter is used as a coefficient of magnification (it is necessary in order to artificially enlarge relatively small variations of function f.).

  Initial properties of surfaces and objects (color, texture, position, rotation etc.) can be specified.
- For the variables that are *not bound* in the function definitions there exist several mechanisms to specify their numerical values, e.g. form-based and scale-based interactivity options.

2.1 The *ObjectMath* environment

ObjectMath is a high-level programming environment and modeling language for scientific computing, developed with partial support from the European Union Esprit3 PREPARE project. The *ObjectMath* language is an object- oriented extension to the computer algebra language *Mathematica* in a similar way as C++ is an extension to C. The language integrates object-oriented constructs such as classes, single and multiple inheritance, with computer algebra features.

The environment is currently being used for industrial applications in bearing simulation, as well as being evaluated for use in other areas. Objects are used to structure models that might consist of a large number of equations. Using *ObjectMath* , it is possible to model classes of equation objects, to support inheritance of equations, to model part-of relations between objects, and to solve systems of equations. The class specialization hierarchy is also used to generate specialized efficient numeric code from general symbolic equations and expressions.

2.2 The *ObjectMath* compiler

The mathematical model is augmented with suitable declarations (e.g. variable and function types) such that functions and equations can be transformed into executable code.

The system extracts all *ObjectMath* / *Mathematica* code necessary as the input for the compilation of required functions $f(u, v)$ and values of $u_{min}, u_{max}, v_{min}, v_{max}$. This code is optimized, and target code is generated.

We use C++ or Fortran90 as the output language. The code is then compiled by a C++ or Fortran90 compiler into machine code. This way we solve the problem of the performance bottleneck in function evaluation.

The geometrical properties of physical objects modeled in *ObjectMath* are usually specified through parametric surfaces. The functions and formulas that define such surfaces have the nice property that they can be used both for computation and visualization.

2.3 Graphical interactive environment

The graphical interactive environment includes all traditional tools for navigation, including zooming, scaling, viewpoint and object rotation, manipulating cameras and lights, showing several views simultaneously from different camera positions .

2.4 Using free variables for parameterization

All free variables which are not bound as parameters or local variables in the code of these functions can serve as input variables. For example in function definition `f[x_,y_]:=Sin[x z+y]` the variable `z` is not bound.

The code of generated function

```
real f(real x,real y) { return sin(x*z+y); }
```

cannot be changed at run time. However the values of free variables (such as variable the `z`) and interactivity properties can be changed. The following are few examples of different methods for update:

- A free variable can be set from numerical data provided from an input file. The user can update the file manually or generate (compute) it using some program. Note that the variable can be a multi-dimensional array. This way the visualization uses a simulation trace, results of other computations or measured data.

- An environment parameter that can be updated in a corresponding data entry window, via a textual entry or a slider. In this case the surface described by the model (e.g. $\sin(xz + y)$) can be changed interactively by adjusting the value of z. Design of appropriate graphical interface is discussed in [15].

- An environment parameter can also represent modeled time. In this case the animation can be shown with different time scales. The animation tool includes functionality for browsing forward and backward in time. The end-user can set preferred speed and stride values.

3 Technical aspects of the MAGGIE system

Efficient use of graphical hardware. PEX [11] and OpenGL [12] are two most distributed graphical standards (see Figure 1) used for programming 3D scientific visualization applications. Using graphical standards optimized for particular hardware is absolutely necessary if realistic visualization with both quality and speed is required.

The current implementation of our *BEAUTY* tool uses the PEX standard for rendering. PEX is supported by SUN and several other vendors. As an emerging *de facto* graphical standard, OpenGL is supported on a wider range of platforms including SGI machines and SUN UltraSparc computers.

There are many similarities between PEX and OpenGL [13]. In order to achieve better usability and portability of our visualization tools between PEX and OpenGL we are planning to implement a common graphical interface library called *COGRINT* (Common Graphical Interface). It will allow to use the same code based on both standards - PEX and OpenGL, since PEX is the current standard primarily used on SUN workstations, and OpenGL is the standard on most platforms including SGI workstations and Sun UltraSparc workstations.

Control over the level of detail The current implementation supports only a fixed level of detail. This causes unnecessary loss of performance when whole model is rendered, and deficiency of details when a high zooming level is used. Instead, the level of detail (i.e. the rendered grid granularity) should be automatically adapted.

Also, user preferences should be taken into account. A high zooming rate will automatically switch the tool to higher resolution.

Support for hierarchical grouping A multi-body system is typically structured as a hierarchy of objects which includes groups of elements which in turn can be groups or atomic objects. For example, a bearing consists of inner ring, outer ring

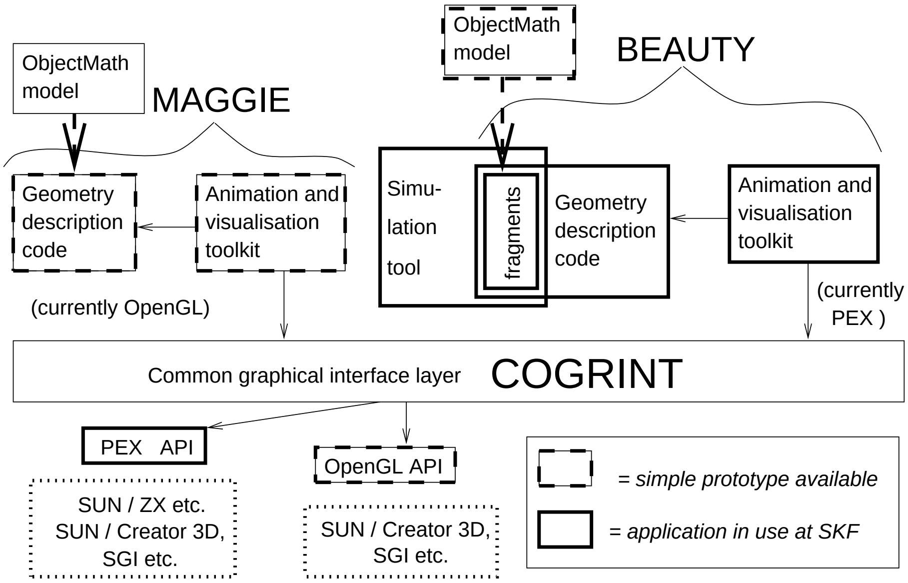

Figure 1: Layers of components in the visualization tools *BEAUTY* and *MAGGIE*

and a set of rollers, etc. The geometry of a roller in a model is represented as a parametric surface.

In this hierarchy attributes (such as parametric description of surface, position and orientation, color and resolution of grid) of groups are automatically propagated to their elements. These attributes can be changed interactively at run-time, except the parametric description of surface.

4 Conclusion and future work

This paper presents ongoing work to give more efficient and flexible ways to explore massive amounts of numerical data from simulations. We are looking for a compromise between the needs for high performance graphics and our desire too keep down the complexity of object-oriented modelling with ObjectMath. The system can be used both for small models and complex industrial scale applications.

Our future research will be directed to:

- Closer integration of the 3D graphical browser with the *Mathematica* environment.
- More automation and flexibility at code compilation stage, e.g. incremental compilation would be useful.
- Using the tool with various graphical environments, e.g. virtual reality modelling systems.
- Using the tool for computational steering[14], on-line, interactive control of the simulation via graphical environment.

References

[1] Wolfram, S. *The Mathematica Book*, Wolfram Media Inc., 1996.

[2] Fritzson, P., Viklund, L., Herber, J. and Fritzson, D. High-level Mathematical Modeling and Programming, *IEEE Software*, July 1995, pp. 77-86.

[3] Viklund, L. and Fritzson, P., ObjectMath: An Object-Oriented Language for Symbolic and Numeric Processing in Scientific Computing, *Scientific Programming*, 1995, Vol. 4, pp 229-250.

[4] ObjectMath Home Page, `http://www.ida.liu.se/labs/pelab/omath`

[5] A *Mathematica* to VRML compiler, `http://www.mathsource.com/cgi-bin/MathSource/Enhancements/Interfacing/Web/0208-336`

[6] LIVE tool, `http://www.wolfram.com/cgi-bin/MathSource/Enhancements/Interfacing/Graphics/0206-479`

[7] MathLive and MathLive Pro tools, `http://www.milohedge.com/products/mathlive/mathlive.html`

[8] Dynamic Visualizer, `http://www.wolfram.com/applications/visualizer/index.html`

[9] GLView, `http://www.viewtech.no/GLview.html`

[10] AVS, `http://www.avs.com`

[11] Tom Gaskins, *PEX Programming Manual*, O'Reilly, 1992.

[12] J. Neider et al., *OpenGL Programming Guide*, Addison-Wesley. 1993.

[13] Mark J. Kilgard, Comparing OpenGL to PEX, `http://www.sgi.com/Technology/openGL/mjk.intro/section3_5.html`

[14] Computational Steering Links, `http://www.cs.uoregon.edu/harrop/steering.html`

[15] Engelson, V., Fritzson, P., Fritzson, D., Automatic Generation of User Interfaces From Data Structure Specifications and Object-Oriented Application Models. In *Proceedings of European Conference on Object-Oriented Programming (ECOOP96)*, Linz, Austria, 8-12 July 1996, vol. 1098 of Lecture Notes in Computer Science, Pierre Cointe (Ed.), pp. 114-141. Springer-Verlag, 1996