Interactive Simulation and Visualization

22.05. 2014
Mathematical models used in scientific computing are becoming large and complex. In order to handle the size and complexity, the models should be better structured (using object-orientation) and visualized (using advanced user interfaces). Visualization is a difficult task, requiring a great deal of effort from scientific computing specialists. Currently, the visualization of a model is tightly coupled with the structure of the model itself. This has the effect that any changes to the model require that the visualization be redesigned as well. Our vision is to automate the generation of visualizations from mathematical models. In other words, every time the model changes, its visualization is automatically updated without any programming efforts.
The innovation is demonstrating this approach in a number of different situations, e.g. for input and output data, and for two- and three-dimensional visualizations. We show that this approach works best for object-oriented languages (*ObjectMath, C++, and Modelica*). We describe the design of several programming environments and tools supporting the idea of automatic generation of visualizations. Tools for two-dimensional visualization include an editor for class hierarchies and a tool that generates graphical user interfaces from data structures. The editor for class hierarchies has been designed for the *ObjectMath* language, an object oriented extension of the *Mathematica* language, used for scientific computing. Diagrams showing inheritance, part-of relations, and instantiation of classes can be created, edited, or automatically generated from a model structure. A graphical user interface, as well as routines for loading and saving data, can be automatically generated from class declarations in *C++* or *ObjectMath*. This interface can be customized using scripts written in *Tcl/Tk*. 
Mathematica includes highly flexible tools for visualization of models, but their performance is not sufficient, since Mathematica is an interpreted language. We use a novel approach where Mathematica objects are translated to C++, and used both for simulation and for visualization of 3D scenes (including, in particular, plots of parametric functions).

Traditional solutions to simulations of CAD models are not customizable and the visualizations are not interactive.

Mathematical models for mechanical multi-body simulation can be described in an object-oriented way in Modelica. However, the geometry, visual appearance, and assembly structure of mechanical systems are most conveniently designed using interactive CAD tools. Therefore we have developed a tool that automatically translates CAD models to visual representations and Modelica objects which are then simulated, and the results of the simulations are dynamically visualized.
We have designed a high performance OpenGL-based 3D-visualization environment for assessing the models created in Modelica.

These visualizations are interactive (simulation can be controlled by the user) and can be accessed via the Internet, using VRML or Cult3D technology. Two applications (helicopter flight and robot simulation) are discussed in detail and a section on integration of collision detection and collision response with Modelica models in order to enhance the realism of simulations and visualizations.

We compared several collision response approaches, and ultimately developed a new penalty-based collision response method, which we then integrated with the Modelica multi-body simulation library and a separate collision detection library.

We also present a new method to compress simulation results in order to reuse them for animations or further simulations. This method uses predictive coding and delivers high compression quality for results from ordinary differential equation solvers with varying time step.
An appropriate user interface technology should be used in each phase.

The development and use of software goes through several stages.
We use the word visualization in a broad sense. Our interpretation is the representation of data structures and data values on computer displays by means of two and three-dimensional graphical elements using an appropriate level of abstraction. The 3D visualization is a representation of three-dimensional scenes mapped onto a 2D display. Scientific visualization is a special case of visualization which usually means visual presentation of high volumes of numeric data defined over some continuous domain, such as time and/or space. Often computational results of scientific computing are displayed by scientific visualization tools (e.g. AVS, Data Explorer, and Vis5D).

The information used in scientific computing falls into two categories: descriptions of mathematical models and descriptions of data. When a mathematical model (at some level of abstraction) is represented graphically as a diagram by some tool it is usually not called visualization, but rather graphical model browsing and/or editing.
Interactive simulation and visualization

Most researchers who perform data analysis and visualization do so only after everything else is finished, which often means that they don't discover errors invalidating the results of their simulation until post-processing. A better approach would be to improve the integration of simulation and visualization into the entire process so that they can make adjustments along the way. This approach, called computational steering, is the capacity to control all aspects of the computational science pipeline. Recently, several tools and environments for computational steering have begun to emerge. These tools range from those that modify an application's performance characteristics (either by automated means or by user interaction) to those that modify the underlying computational application. A refined problem-solving environment should facilitate everything from algorithm development to application steering. The authors discuss some tools that provide a mechanism to integrate modeling, simulation, data analysis and visualization.
There are many other kinds of information that are important in software design, which are not covered in this work. Particular examples are documents and document structures, entity-relation diagrams, scenario diagrams, database visualization and diagrams of program execution paths. Here we discuss and compare several stages where interactive graphical environments can be used:

- model editing (in particular, browsing and editing of model component diagrams during the design stage),
- input data editing,
- visualization during execution (e.g. interactive control of simulations, execution monitoring, and computational steering)
- output visualization in the form of 2D graphs and 3D interactive animations.
Models and Graphical User Interfaces

We can use a variety of object-oriented languages (*ObjectMath*, *C++*, *Modelica*) as the basis for our tools. Object-oriented models have a number of advantages, mainly for the following reasons:

- object-orientation imposes concise, hierarchical structures on models and data;
- information necessary for graphical user interface design can be extracted from such structures;
- object oriented languages provide the means to attach auxiliary attributes to existing structures. This can be done by specialization through inheritance. Then these attributes can be used for graphical user interface generation. Such attributes do not interfere with the data properties used for normal computation (e.g. simulation).
User Interaction

Users are expected to be active agents in the process of design and use of scientific software. Therefore user interfaces play a dominant role. In descriptions of interactive tools we attempt to evaluate the quality of user interfaces. For this purpose typical user tasks are selected and user efforts required for performing these tasks are estimated. A good interface for visualization and editing should provide the following

- consistent and compact presentation of components (layout quality),
- ease of use in navigation and location of relevant subcomponents (navigation quality),
- consistent feedback from user actions, such as data editing or simulation steering (feedback quality).
Taxonomy of Visualizations

In this taxonomy seven kinds of visualizations are related to seven corresponding data types. Generation of some of visualizations in this taxonomy can be automated.

1-dimensional: This class of visualization and data structures includes textual documents and program source code. However, the models we consider are usually quite complex. Therefore, one-dimensional visualizations are not sufficient for presentation of such models. In particular, program source code has a well-defined structure. We suggest that this source code is converted into hierarchical structures with nodes (classes, objects, instances) and connections (relations between them). This structure is generated automatically from the textual representation and is visualized by our tools. For instance, ObjectMath models are represented by hierarchical diagrams.
2-dimensional: This kind of visualization includes geographical maps and plans, as well as 2-dimensional plots of functions (X-Y plots). This is a widespread way to present results from scientific computing applications. Most of the tools for scientific computing, e.g. Dymola, Mathematica, MathModelica, Beast have facilities for plotting variables in different ways and options for choosing variables to be plotted. The graphical user interface for selecting variables to be plotted is automatically generated from data structures, e.g. for Dymola, MathModelica and Beast.

3-dimensional: Items with volume, e.g. real world objects, and object designed using 3D CAD modeling tools are most naturally shown using 3-dimensional visualization. Both real and abstract objects (such as three dimensional plots of functions) can be viewed in this kind of visualization. Visualizations in three dimensions are often used for physics-based simulations. In this case, the structure of objects and their interrelations usually correspond to the structure of the mathematical model.
Such a model contains descriptions of each physical component (e.g. rigid body) which is visualized as a separate graphical object in 3D. Therefore, creation of such visualizations can be automated (see paper 5). There are relations between the visualized components (contacts and various motion constraints), which usually are not visualized explicitly as graphic elements, but which can easily be noticed when moving objects are observed.

There exist, however, 3-dimensional visualizations where the structure of graphical objects is completely different from the structure of mathematical model. For instance, this happens in scientific visualization tools used in computation fluid dynamics. Instead, the structure of graphical user interface for visualization control often corresponds to the structure and dimensionality of data.
Temporal: Visualization of time lines, historical information, and events are examples of temporal visualization. In our case, visualization of temporal data is just one feature for other visualization types, in particular 3-dimensional visualization. We use time in order to represent changes in objects and their relations during physics-based simulation. Output data of such simulations contains values of various variables at each time instant. Animation is used for presentation of such simulations.

Multi-dimensional: Tools working with objects with many attributes need multidimensional visualization. Such objects become points in n-dimensional space. Visualization tools map objects with these attributes to a 2- or 3-dimensional representation. In this thesis, work in this direction has been done with parametric functions of many parameters, which were defined in Mathematica. In our tool the way of mapping n-dimensional parametric functions to space coordinates and time can be selected interactively.
Tree: Tree-structured visualization is useful for structures with relations between parent and child nodes. A tree is a convenient way to represent data structures of a model. Furthermore, it is possible to automate creation of interactive visualizations based on data structures. This automation has been designed for C++, ObjectMath and MathModelica.

Networks and general graphs: Arbitrarily linked relations between nodes are conveniently visualized by networks and general graphs. This visualization is used where objects are related by connectors, for instance in Modelica and Dymola. In mechanical models joints and other contact elements are used as relations between rigid bodies. These relations can be generated using a graphical user interface. For instance, relations between bodies are specified using a CAD interface.
Three dimensional graphical user interfaces

Development of modern technologies has made it possible to apply three-dimensional visualization in many application areas. In particular 3D is used for visualization of computation results and for modeling real world objects, such as objects constructed using CAD tools. Due to development of graphic hardware 3D animation recently became widely available for the users working on average computers and therefore 4D data (three space dimensions and one time dimension) can be used for visualization.

3D visualization in scientific computing falls into three categories:

1. Visualization of numerical results
2. Visualization of fixed shapes
3. Volume visualization
1. **Visualization of numerical results**, where displayed shapes depend on a particular computation. These shapes might depend on some specific parameters, e.g. time. We assume that points in 3D are denoted as \((x,y,z)\). The set of displayed points in three-dimensional coordinate space can be expressed as

\[
\{(F_x(u,v), F_y(u,v), F_z(u,v))\},
\]

where \(\text{umin} < u < \text{umax} \) and \(\text{vmin} < v < \text{vmax}\).

Some components of a graphical user interface for such visualization can be generated automatically.
2. **Visualization of fixed shapes.** Each shape corresponds to a real world object which is modeled as a rigid body (e.g. by a **CAD** tool). Movement of the body is constrained by the laws of physics. This kind of visualization is also called physics-based visualization.

Visualization of results from physics-based simulations can be automated.

3. **Volume visualization**, where displayed shapes are isosurfaces computed from some volume data. This data can be the result of some other computations or measurements. The set of points displayed can be expressed as

$$\{ (x,y,z) / F(x,y,z) = 0 \}.$$ 

Rendering such visualizations is more difficult since it is hard to find the set of points and translate to 3D graphic primitives.
Interactive Visualization of Numerical Results of Computations Specified in Mathematica

Usually parametric surfaces are used for visualization of computational results when many inputs and outputs are involved in a computation. If two input and one output variables are used, the visualization of such a function is a surface composed by all points

\[ \{(x,y,F(x,y))\}, \]

where \( xmin < x < xmax, \ ymin < y < ymax. \)

If three input and three output variables are used, a dynamically changing surface can be composed from all the points

\[ \{(F_x(u,v,t), F_y(u,v,t), F_z(u,v,t))\}, \]

where \( umin < u < umax, \ vmin < v < vmax, \ tmin < t < tmax. \)
In general, an arbitrary function

\[ F : R^m \rightarrow R^n \]  

can be visualized using this method.

The limitations are:

- Input values for a number of dimensions \((m-3)\) dimensions should be fixed.
- Three dimensions are chosen from \(n\), whereas the other \(n-3\) dimensions are omitted.
Generation of visualization for functions with multiple arguments and multiple output values defined in *Mathematica*:
The *ObjectMath* Programming Environment

The *ObjectMath* programming environment is designed to be easy to use for application engineers, e.g. in mechanical analysis who are not computer scientists. It is interactive and includes a graphical browser for viewing and editing inheritance hierarchies, an application oriented editor for editing *ObjectMath* equations and formulae, the *Mathematica* computer algebra system for symbolic computation, support for generation of numerical code from equations, an interface for calling external functions, and a class library. The graphical browser is used for viewing and editing *ObjectMath* inheritance hierarchies. *ObjectMath* code is automatically translated into *Mathematica* code and symbolic computations can be done interactively in *Mathematica*.
The displayed tree in the graphical browser window shows the inheritance hierarchy of classes, the text windows show the edited class definition and the **Mathematica** window for symbolic computations, whereas the visualized object **Body1** is instantiated from a specialized **Sphere** class.
ObjectMath is both a language and a programming environment. The current ObjectMath language has recently been enhanced with features for multiple inheritance and modeling part-of relations between objects. Both of these features has turned out to be important in realistic application models. An early version of the ObjectMath language only supported single inheritance. The ObjectMath language is an hybrid modeling language, combining object-oriented constructs with a language for symbolic computation. This makes ObjectMath a suitable language for implementing complex mathematical models, such as those used in machine element analysis. Formulae and equations can be written with a notation that closely resembles conventional mathematics, while the use of object-oriented modeling makes it possible to structure the model in a natural way.
Object-Oriented Modeling

When working with a mathematical description that consists of hundreds of equations and formulae, for instance one describing a complex machine element, it is highly advantageous to structure the model.

A natural way to do this is to model machine elements as objects. Physical bodies, e.g. rolling elements in a bearing, are modeled as separate objects. Properties of objects like these might include a surface description, a normal to the surface, forces and moments on the body, and a volume. These objects might define operations such as finding all contacts on the body, computing the forces on or the displacement of the body, and plotting a three-dimensional picture of the body.
Abstract concepts can also be modeled as objects. Examples of such concepts are coordinate systems and contacts between bodies. The coordinate system objects included in the ObjectMath class library define methods for transforming points and vectors to other coordinate systems.

Equations and formulae describing the interaction between different bodies are often the most complicated part of problems in machine element analysis. This makes it practical to encapsulate these equations in separate contact objects. One advantage of using contact objects is that we can substitute one mathematical contact model for another simply by plugging in a different kind of contact object. The rest of the model remains completely unchanged. When using such a model in practice, one often needs to experiment with different contact models to find one which is exact enough for the intended purpose, yet still as computationally efficient as possible. The ObjectMath class library contains several different contact classes.
The use of inheritance facilitates reuse of equations and formulae. For example, a cylindrical roller element can inherit basic properties and operations from an existing general cylinder class, refining them or adding other properties and operations as necessary.

Inheritance may be viewed not only as a sharing mechanism, but also as a concept specialization mechanism. This provides another powerful mechanism for structuring complex models in a comprehensive way. Iteration cycles in the design process can be simplified by the use of inheritance, as changes in one class affects all objects that inherits from that class. Multiple inheritance facilitates the maintenance and construction of classes which need to combine different orthogonal kinds of functionality.

The part-of relation is important for modeling objects which are composed of other objects. This is very common in practice.
A CLASS declaration declares a class which can be used as a template when creating objects. ObjectMath classes can be parameterized. The ObjectMath INSTANCE declaration is, in a traditional sense both a declaration of class and a declaration of one object (instance) of this class. This makes the declaration of classes with singleton instances compact. An array containing a symbolic number of objects can be created from one INSTANCE declaration by adding an index variable in brackets to the instance name. This allows for the creation of large numbers of nearly identical objects, for example the rolling elements in a rolling bearing. To represent differences between such objects, functions (methods) that are dependent upon the array index of the instance can be used. The implementation makes it possible to do computations with a symbolic number of elements in the array.
Single Inheritance

In addition to classes describing bodies with different geometry depicted in the inheritance hierarchy, there are additional classes which describe interactions between bodies and coordinate systems. Note that the inheritance hierarchy usually is edited graphically so that the user does not have to write the class headers by hand.

An inheritance hierarchy of classes for modeling bodies with different geometries such as cylinders and spheres:

![Inheritance Hierarchy Diagram]

```
Body
  ↓             ↓
Sphere         Ring
  ↓             ↓
Body1          Cylinder
                  ↓
                     Body2
```

Multiple inheritance is useful when combining orthogonal concepts. Multiple inheritance hierarchy of bodies of different materials and geometries:

The filled lines denote single inheritance, whereas the dotted lines denote additional inheritance, i.e. we have multiple inheritance. Since material properties and geometry are orthogonal concepts there are no collisions between inherited definitions.
Single inheritance version of the material-geometry model:

The material equations describing elasticity or plasticity have to be repeated twice. This model structure is harder to maintain when changes are introduced into the model.
Another useful case of multiple-inheritance is shown below, where an integration method is inherited into classes from two separate inheritance hierarchies (multiple inheritance of a numerical integration method into two different classes):

Here to be used for integrating forces or volumes. One class contains contact equations; another contains volumes, moments and equilibrium equations.

The entities inherited from class `Integration_Method` will typically be a combination of entities such as procedural code, transformation rules.
Modeling Part-Of Relations

The part-of relation is important for modeling objects which are composed of other objects, also noting that this concept is orthogonal to the concept of inheritance which is used to represent specialization. For example, a bicycle contain parts such as wheels, frame, pedals, etc. A rolling bearing contain inner ring, outer ring, rolling elements, lubrication fluid, etc. The ObjectMath syntax for expressing composition using the part-of relation is exemplified below for a Bicycle class:

```
CLASS Bicycle(C,P)

... PART frontwheel INHERITS Wheel(P);
     PART rearwheel INHERITS Wheel(P);
     PART frame INHERITS Body;
... END Bicycle;
```
During the development of complex mathematical models there is often a need to explore different variants of solution strategies and formulations of equations. One would like to experiment with alternative ways of expressing equations and transformations within a certain class and still keep the previous version of the class definition in the model.

Each new variant of a class can of course be tried out by creating an entirely new model where all classes except one are identical compared to the previous model.
The ObjectMath environment consisting of a diagram editor window, a program text window and the start window:

```plaintext
model abc;

class Body ... end;
class Material ... end;
class Metal inherits Material ... end;
class Plastic(b,b) inherits Material ... end;
class Sphere inherits Body ... end;
class Cylinder inherits Body ... end;
class Wheel inherits Cylinder, Metal ... end;
class Bicycle inherits Body
    part tube[five] inherits Cylinder;
    part front inherits Wheel(t);
    part rear inherits Wheel;
end;
instance bike inherits Bicycle ... end;
```
The inheritance relations are numbered because the order of classes in case of multiple inheritance affects the program semantics.

The container for global objects (Global container) is used for two purposes. First it contains global variables, functions and equations which do not belong to any particular instance. Second, the icon of Global container is connected to all classes and instances that have no superclasses.
Operations of *ObjectMath* diagram editor

Menu choices of the *ObjectMath* class diagram editor. The alternatives leading to new dialogs are marked with ellipsis:
Virtual prototyping of complex systems presents interesting challenges, especially with regard to systems in which different languages are used to represent different parts. We describe here one approach to solving such a problem. In particular, we describe how to integrate the virtual test bed (VTB) solver engine with the Simulink solver. This produces a rich environment for virtual prototyping of power electronic applications due to the inherent mixture of circuital and control problems. The integration is conducted within the context of the resistive companion approach. We present here the theoretical foundation of the approach, and also suggest the generality and extensibility to other solver engines such as SPICE. The theory is then enriched with some examples that illustrate the approach including the use of the VTB high-level graphic user interface.
A Mathematica notebook with results of a symbolic integration, 2D and 3D plots. The notebook cell structure is made visible via brackets on the right side.
The screen shot of the MAGGIE tool with animation of a parametric surface:

... USING THE MATHEMATICA ENVIRONMENT FOR GENERATING EFFICIENT 3D GRAPHICS
Water surface after the stones fell down and the waves appeared. The stones are below the surface and we look at them from below. This is a screen shot from the animation sequence.
References

1. Interactive simulation and visualization, Johnson C.; Center for Sci. Comput. & Imaging, Utah Univ., Salt Lake City, UT, USA ; Parker, S.G. ; Hansen, C. ; Kindlmann, G.L., ...

