SIMLAB: Automatically Creating Physical Systems Simulators

Richard S. Palmer
Computer Science Department
Cornell University
Email: rick@cs.cornell.edu

James F. Cremer
Computer Science Department
University of Iowa
Email: cremer@cs.uiowa.edu

Abstract

SIMLAB, a software environment for creating simulators directly from computer-readable physics models, is based on the following concept: *creating physical systems simulators should be as simple as describing the underlying physics to a colleague.*

Rather than programming in a conventional programming language, a SIMLAB user expresses physics models (and thus simulators) directly in terms of the concepts, quantities, and equations familiar to a scientist or engineer.

The benefits of the SIMLAB approach include: 1) reducing the time and effort required to create simulators, 2) providing more understandable and reliable simulators, and 3) support for more sophisticated simulators, e.g., for multiple-domain problems, which have proved intractable in the past.

Introduction

Large physical systems simulators are currently specified at too low a level — they are specified by writing a computer program in a conventional programming language. As a result, creating a such a simulator can be a daunting task taking years to complete. This causes a shortage of physical systems simulation and analysis software. In particular, there is a significant time lag between development of hardware technology (such as new parallel architectures) and its use for physical systems simulation and analysis.

Existing simulation languages[Mit, 1986, Augustin *et al.*, 1967, Pritsker, 1986] are typically a thin veneer over FORTRAN: they are often, in fact, preprocessors for FORTRAN. In addition to providing a high-level/FORTRAN syntax, such systems typically provide subroutines for performing a variety of simulation operations, such as ODE integration. While these software systems are an important advance over unenhanced conventional programming languages, we envisage a more ambitious system: a simulator programming environment in which simulators are specified more or less directly as they are understood by a scientist or engineer. Our prototype system, SIMLAB, is a first step towards this goal. The input to SIMLAB looks much like the models of physics described in textbooks.

Existing special purpose simulation tools such as AUTOSIM[Mit,] and SD/FAST[Sym, 1990] generate simulation code for a particular physics model (rigid body dynamics in this case) using a particular formulation method (Kane's method). In contrast, the SIMLAB environment allows a *user* to define the physics model and formulation method. Thus, SIMLAB can be used to create special purpose simulators for a wide range of problems.

Besides the simulation languages described above, the *bond graph* formalism, developed by Paynter in the late 1950's[Paynter, 1961], has engendered a number of simulation systems[Ros, 1989, Broenink, 1990]. Bond graphs are essentially a generalization of electrical circuits. Analogs to electrical components, such as inductors, resistors, and capacitors are defined for various physical domains. As an example, in the mechanical domain, the spring is analogous to the capacitor, while the dashpot (shock absorber) is analogous to the resistor. Physical interaction between components is modeled by "power bonds," which model energy transfer between the components, and guarantee conservation of energy.

Transducers may be used to relate phenomena in different physical domains. As an example, a bond graph model of a hydraulic cylinder might include a transducer that transforms the hydraulic pressure into mechanical pressure on the cylinder clevis.

Bond graph based simulation systems are closer in spirit to the current SIMLAB prototype than the simulation languages described above. Both SIMLAB and bond graph systems support the notion of primitive objects with associated behavior (PRIMITIVES and *components*, respectively), and interactions between them (CONNECTIONS and *power bonds*). They differ mainly in that bond graph systems use a predefined class of equations and solution methods instead of allowing the user to define them. One might say that a given bond graph modeler corresponds to a single SIMLAB

model and formulation. The system designer has chosen these; the user has no choice.

To see the value and implications of creating simulators directly from models of physics, suppose a scientist has spent the last two years constructing a simulator for a diesel engine. The resulting program is a combined rigid-body/fluid-dynamics/combustion simulator. It simulates the pistons, connecting rods, and crankshaft as rigid bodies, and uses PDE models of turbulent fluid flow and combustion.

The scientist has defined models of the interaction between these various models of physical phenomena. For example, the force that the expanding gasses exert on the piston due to combustion must be represented. Weeks have been spent defining and implementing data structures, choosing algorithms, and incorporating pieces of various numerical packages such as LINPACK and ODEPACK. The simulator has been carefully designed, coded, tested, and debugged.

The result is a large body of intricate and carefully crafted simulation software. This is used to tune the design parameters of a new diesel engine, and proves to be of great value. Although the author is convinced of its correctness, it is difficult to convince others that the results may be relied on, because the simulator implementation itself is difficult for others to understand.

Depending on the implementer's level of effort and foresight, it may be possible to change a few parameters to modify the simulation to correspond to design changes such as larger pistons, stroke, or improved models of phenomena. It may even be possible to accommodate a change in the number of cylinders. The difficulty arises when the simulation must incorporate a heat flow analysis of the piston and block. Because this requirement did not exist in the original specification, it is very difficult to patch this additional phenomenon into the simulator: software must be modified and augmented in a multitude of places distributed throughout the program. In addition to requiring months of effort, the resulting program is less elegant, and because it is now more difficult to understand, the level of confidence in the results is reduced considerably.

Suppose, in contrast, that an implementer creates a simulator using the SIMLAB system. The initial stage is identical: one determines a mathematical model of the physical phenomena. In this case, however, the initial stage is much closer to the final stage: The input to SIMLAB is the mathematical model, together with high-level instructions for solving the resulting systems of equations. From these, the system creates a simulator as well as an editor (called the *scene generator*) for creating instances of the problem. Rather than requiring the user to define data structures and algorithms, combine various numerical packages, and write analysis and visualization routines, SIMLAB performs these steps automatically. This means that changing the model, for instance adding heat transfer to a simulation, requires only defining a *heat transfer* model, and defining the interactions between *heat transfer* and the various other phenomena. Again the simulator and related modules are created directly from the model. While the resulting simulator may be as intricate as a hand crafted one, the time, effort, and cost required to create it is reduced by orders of magnitude. In addition, the results are more credible, because the author has expressed the underlying mathematical model explicitly. The simulator has been automatically constructed from that model, which aids in convincingly defending the simulation results.

The process of creating a simulator is not the only aspect of simulation that is time consuming. In fact, the design process itself, i.e., creating the artifacts to be simulated, is the major purpose of most engineering and design departments. In the previous example, the design of the engine itself could easily require more effort than the design of the simulator that analyses it. The ability to quickly create and modify simulators is of limited value if it means that extant artifact designs must be discarded or laboriously re-created when a simulator is modified. Fortunately, the explicit representation of the physical model allows the simulator designer to specify translations directly in terms of the physical model itself.

A "simulation language" to *think* in The goal in creating a computer language or environment is to provide the scientist with the means to express the problem in a manner as close as possible to the way he or she *thinks* about the problem. Specifying a simulator should be, as nearly as possible, like describing the solution method to a colleague. Beyond providing semantics corresponding to the user's current mode of understanding the phenomena, we believe that our environment will provide an enhanced *understanding* of the engineering problems themselves. By way of comparison, the `goto` statement encourages a low-level difficult-to-understand style of programming, while the `while` loop construct encourages a structured style, which is more easily verified, understood, and explained. In a analogous manner, we believe that the SIMLAB environment for specifying simulators will make explicit the meaning and assumptions typically hidden in simulation software. This in turn can provide engineers and scientists with an improved model for *understanding* the process of creating software for physical systems modeling and simulation.

Ultimately, we believe the benefits of the SIMLAB approach will be:

1. Development of a high-level means of expressing simulation problems.
2. Improved understanding of physical phenomena and interactions.
3. Quickly constructed, reliable, and understandable simulators.
4. Simulators that can be quickly and easily modified.

5. Retargeting simulators to take advantage of new machine architectures.

Where we stand: the current state of SIMLAB

Our first steps toward automatic simulation generation have been to define the system architecture of SIMLAB and to implement the first prototype. Our architecture comprises three levels:

1. A high-level language for expressing and representing models of physics.
2. An intermediate level *algorithmic substrate* which implements computational forms of algebra, combinatorial topology, and control specification.
3. A meta-language for representing facts about program components, subroutines, and packages, which will allow algorithmic components to be automatically assembled.

We have implemented a prototype of the first, and have begun to investigate the second. We have yet to begin implementation of third level.

The "semantic gap" between SIMLAB's model language and the conventional languages it uses to create simulators suggests the need for an an intermediate language and associated software, which we refer to as the *algorithmic substrate*: a language in which to program mathematics and algorithms. Because we currently lack such a in intermediate level, all the simulators created so far contain some "hardwired" program components. These and other issues are described in more detail in our longer paper[Palmer and Cremer, 1991].

Examples

Here we present three example SIMLAB models and simulations performed using simulators produced from these models by the prototype SIMLAB system.

Particle dynamics

Figure 1 shows results from the the first kind of simulation done using SIMLAB. A simulator was produced from a model defining particles and a single type of connection — interparticle gravitational attraction.

More recently, we've used SIMLAB to generate simulators for electrostatic particle dynamics. These were the first SIMLAB generated simulators to support events. In particular, particles can be added and deleted during the course of simulations. Figure 2 shows a simulation involving a number of particles introduced at random time intervals at the left end of the image. The small circles represent wires having charges that vary in accordance with an alternating current. In addition, the particles are attracted upwards by a charged plate. This example was inspired by simulations used to design xerographic copiers.

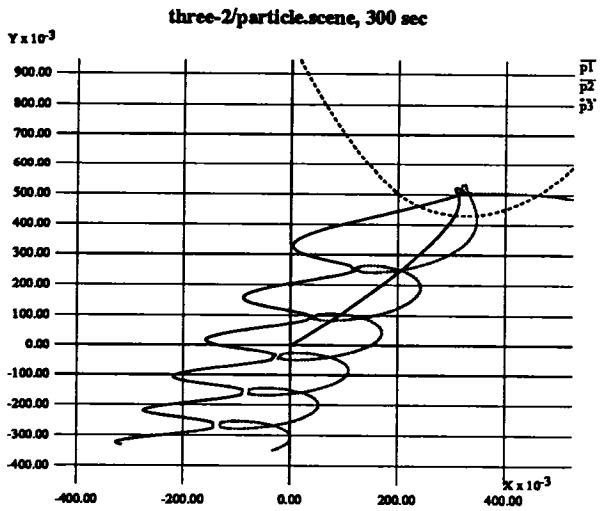

Figure 1: Trajectories of three particles. Gravitational attraction is the only force.

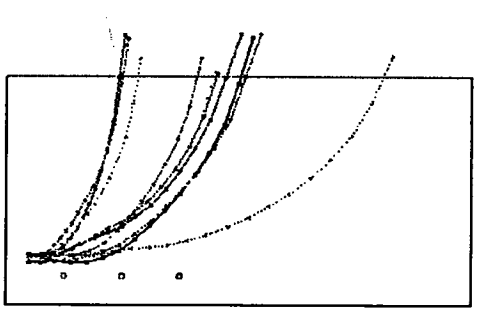

Figure 2: Electrostatic particle dynamics.

Rigid-body dynamics

One of our goals is to be able to use SIMLAB to produce the equivalent of the NEWTON rigid-body dynamics simulator from a concise, high-level specification. The NEWTON system is thousands of lines of LISP and FORTRAN code; it should be possible to generate most of that from a few-page-long SIMLAB model.

SIMLAB requires further development before that goal can be fully realized. However, our current progress is encouraging. We have generated one rigid-body dynamics simulator from a two-page model; its coverage, however, is only a subset of NEWTON's. The model defines the basic motion laws of rigid-body dynamics and allows for various kinds of hinges (in this model, holonomic constraints only) between bodies. It does not provide for collision detection and resolution, or contact formation and release. Part of the model is shown in Figure 3. The figure uses a more readable syntax than the LISP-flavored one used by the existing system.

Figure 4 shows the simplest rigid-body dynamics simulation — a block falling under the influence of gravity. The rotation seen is due an initial angular velocity, as specified in the scene. The rotation axis then changes due to the $\omega \times J\omega$ term in the dynamics equations.

Figure 5 shows a more complicated simulation. In this example two parallelepipeds are hinged together at the corners. In addition, one is "hinged" to the "world," i.e., a point on it is fixed in space. The parallelepipeds were hinged at their corners to induce the interesting spinning motion. The only external force is gravity.

Electric circuits

We have also used SIMLAB to generate very simple circuit simulators. From a model defining capacitors, resistors, inductors, ac-generators and current laws, SIMLAB produced a circuit simulator. Figure 6 graphs the sample charges and current in components of a simple circuit containing a resistor, inductor, ac-generator, and two capacitors.

model RigidBodyDynamics

objects
  RigidBody
    PROPERTIES
      m: real /* mass */
      V: real /* volume */
      J: $real^{3*3}$ /* inertia tensor */
      r: $real^3$ /* position */
      p: UnitQuaternion /* orientation */
      ...
      F_c: $real^3$ /* constraint force*/
      F_e: $real^3$ /* external force */

CONNECTIONS
  HmConstraint (object1, object2: RigidBody)
    ϕ /* Holonomic constraints*/
    ...
  BallAndSocket (c1, c2: $real^3$)
    inherits-from HmConstraint
    ϕ = r(object1) + c1 - r(object2) - c2
    ...
  AppliedForce(object:RigidBody)
    force: $real^3$

laws
  Resultant: for o:RigidBody,
    $F_i = \sum_{h \in HmConstraint\ s.t.\ h(o,*)} \text{force1(h)} - \sum_{h \in HmConstraint\ s.t.\ h(*,o)} \text{force2(h)}$
    $F_e = \sum_{f \in AppliedForce\ s.t.\ f(o)} \text{force(f)}$
  Newton: for o: RigidBody
    $m(o) \times \ddot{r}(o) = F_c(o) + F_e(o)$
  Euler: ...
  VirtualWork: for h:Hinge,
    o1=object1(h), o2=object2(h),
    force1 = $\frac{\partial \phi}{\partial r(o1)}^T \lambda$
    torque1 = ...
    ...

formulation schemes
  UnreducedNewtonEuler:
    MotionEquations =
      $\forall$o$\in$RigidBody: ApplyLaw(Newton, o)
      $\cup$
      $\forall$o$\in$RigidBody: ApplyLaw(Euler, o)
    ConstraintEquations =
      $\cup_{h \in \text{HmConstraint}} \phi(h)$

Figure 3: Rigid-body dynamics model

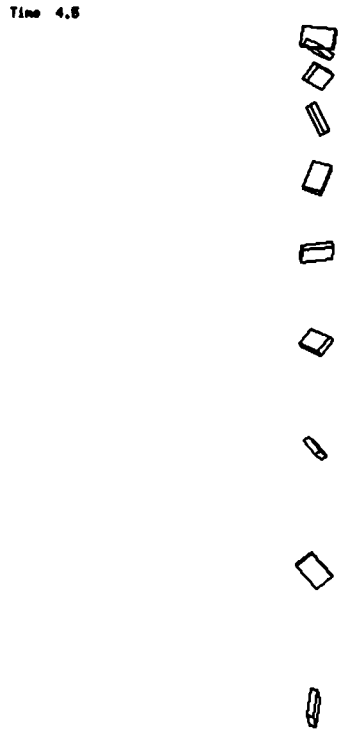

Figure 4: A single falling rigid body. Gravity is the only force.

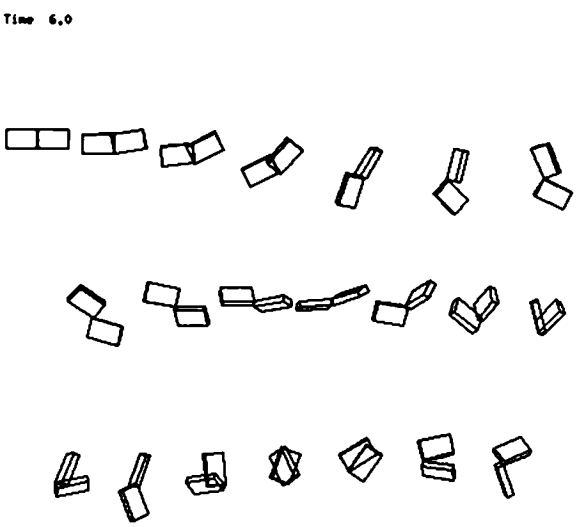

Figure 5: A double pendulum under the influence of gravity. The lower right rear corner of the rightmost block is fixed in space. The image sequence is left to right, top to bottom.

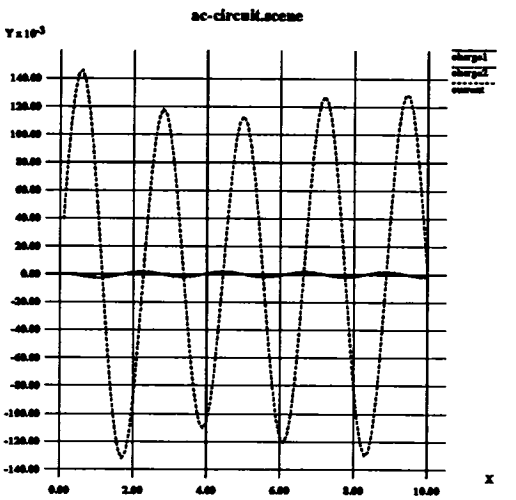

Figure 6: Sample charge, current trajectories for a simple circuit simulation. Although the graph of the current appears flat, it in fact oscillates correctly; it's simply below the resolution of this figure.

References

Donald C. Augustin, Mark S. Fineberg, Bruce C. Johnson, Robert N. Linebarger, F. John Sansom, and Jon C. Strauss. The csi continuous system simulation language (cssl). *Simulation*, 9:281–303, 1967.

Jan F. Broenink. *Computer-Aided Physical-Systems Modeling and Simulation: A Bond-Graph Approach.* PhD thesis, University of Twente, Enschede, The Netherlands, 1990.

Mitchell & Gauthier Assoc., Concord, Mass. *AUTOSIM.*

Mitchell & Gauthier Assoc., Concord, Mass. *ACSL: Advanced Continuous Simulation Language — User Guide and Reference Manual*, 1986.

Richard S. Palmer and James F. Cremer. Simlab: Automatically creating physical systems simulators. Technical Report 91–1246, Cornell University, November 1991.

Henry M. Paynter. *Analysis and Design of Engineering Systems.* The M.I.T. Press, Cambridge, Massachusetts, 1961.

A.A.B. Pritsker. *Introduction to Simulation and SLAM II.* Systems Publishing Corporation, 1986.

RosenCode Associates, Inc., Lansing, Mich. *The ENPORT Reference Manual*, 1989.

Symbolic Dynamics, Inc., Mountain View ,CA. *SD/FAST User's Manual*, 1990.