User-friendly language for programming efficient simulations

August 10 2016

Computer simulations of physical systems are common in science, engineering, and entertainment, but they use several different types of tools.

If, say, you want to explore how a crack forms in an airplane wing, you need a very precise physical model of the crack's immediate vicinity. But if you want to simulate the flexion of an airplane wing under different flight conditions, it's more practical to use a simpler, higher-level description of the wing.

If, however, you want to model the effects of wing flexion on the crack's propagation, or vice versa, you need to switch back and forth between these two levels of description, which is difficult not only for computer programmers but for computers, too.

A team of researchers from MIT's Computer Science and Artificial Intelligence Laboratory, Adobe, the University of California at Berkeley, the University of Toronto, Texas A&M, and the University of Texas have developed a new programming language that handles that switching automatically.

In experiments, simulations written in the language were dozens or even hundreds of times as fast as those written in existing simulation languages. But they required only one-tenth as much code as meticulously hand-optimized simulations that could achieve similar execution speeds.

"The story of this paper is that the trade-off between concise code and good performance is false," says Fredrik Kjolstad, an MIT graduate student in electrical engineering and computer science and first author on a new paper describing the language. "It's not necessary, at least for the problems that this applies to. But it applies to a large class of problems."

Indeed, Kjolstad says, the researchers' language has applications outside physical simulation, in machine learning, data analytics, optimization, and robotics, among other areas. Kjolstad and his colleagues have already used the language to implement a version of Google's original PageRank algorithm for ordering search results, and they're currently collaborating with researchers in MIT's Department of Physics on an application in quantum chromodynamics, a theory of the "strong force" that holds atomic nuclei together.

"I think this is a language that is not just going to be for physical simulations for graphics people," says Saman Amarasinghe, Kjolstad's advisor and a professor of electrical engineering and computer science (EECS). "I think it can do a lot of other things. So we are very optimistic about where it's going."

Kjolstad presented the paper in July at the Association for Computing Machinery's Siggraph conference, the major conference in computer graphics. His co-authors include Amarasinghe; Wojciech Matusik, an associate professor of EECS; and Gurtej Kanwar, who was an MIT undergraduate when the work was done but is now an MIT PhD student in physics.

Graphs vs. matrices

As Kjolstad explains, the distinction between the low-level and high-level descriptions of physical systems is more properly described as the

distinction between descriptions that use graphs and descriptions that use linear algebra.

In this context, a graph is a mathematical structure that consists of nodes, typically represented by circles, and edges, typically represented as line segments connecting the nodes. Edges and nodes can have data associated with them. In a physical simulation, that data might describe tiny triangles or tetrahedra that are stitched together to approximate the curvature of a smooth surface. Low-level simulation might require calculating the individual forces acting on, say, every edge and face of each tetrahedron.

Linear algebra instead represents a physical system as a collection of points, which exert forces on each other. Those forces are described by a big grid of numbers, known as a matrix. Simulating the evolution of the system in time involves multiplying the matrix by other matrices, or by vectors, which are individual rows or columns of numbers.

Matrix manipulations are second nature to many scientists and engineers, and popular simulation software such as MatLab provides a vocabulary for describing them. But using MatLab to produce graphical models requires special-purpose code that translates the forces acting on, say, individual tetrahedra into a matrix describing interactions between points. For every frame of a simulation, that code has to convert tetrahedra to points, perform matrix manipulations, then map the results back onto tetrahedra. This slows the simulation down drastically.

So programmers who need to factor in graphical descriptions of physical systems will often write their own code from scratch. But manipulating data stored in graphs can be complicated, and tracking those manipulations requires much more code than matrix manipulation does. "It's not just that it's a lot of code," says Kjolstad. "It's also complicated code."

Automatic translation

Kjolstad and his colleagues' language, which is called Simit, requires the programmer to describe the translation between the graphical description of a system and the matrix description. But thereafter, the programmer can use the language of linear algebra to program the simulation.

During the simulation, however, Simit doesn't need to translate graphs into matrices and vice versa. Instead, it can translate instructions issued in the language of linear algebra into the language of graphs, preserving the runtime efficiency of hand-coded simulations.

Unlike hand-coded simulations, however, programs written in Simit can run on either conventional microprocessors or on graphics processing units (GPUs), with no change to the underlying code. In the researchers' experiments, Simit code running on a GPU was between four and 20 times as fast as on a standard chip.

This story is republished courtesy of MIT News (web.mit.edu/newsoffice/), a popular site that covers news about MIT research, innovation and teaching.

Provided by Massachusetts Institute of Technology