BREVE: a 3D Environment for the Simulation of Decentralized Systems and Artificial Life

Jon Klein
Complex Systems Group
Department of Physical Resource Theory
Chalmers University of Technology and
Göteborg University
SE-412 96 Göteborg, Sweden
and
Cognitive Science
Hampshire College
Amherst, MA 01002
jklein@hampshire.edu

Abstract

BREVE is a 3D simulation environment designed for simulation of decentralized systems and artificial life. While BREVE is conceptually similar to existing packages such as Swarm and StarLogo, the implementation of BREVE—which simulates both continuous time and continuous 3D space—is quite different such that the environment is suited to a different class of simulations. BREVE includes an interpreted object-oriented language, an OpenGL display engine, collision detection, as well as experimental support for articulated body physical simulation and collision resolution with static and dynamic friction. The ultimate goal of the system is to allow decentralized simulations to be implemented quickly and easily while providing a powerful framework to facilitate the construction of advanced artificial life simulations. BREVE is available for download at http://www.spiderland.org/breve.

Introduction

The field of artificial life focuses on replicating lifelike behaviors in computer simulations. The goals of this research range from creating novel artificial life-forms to building models which will help researchers better understand biological life. Artificial life can also be seen as a necessary first step toward the ambitious goals of artificial intelligence. Many artificial intelligence researchers have gradually shifted focus from huge monolithic artificial intelligence environments to studying intelligence as an emergent behavior of simple interactions.

The focus on emergent behaviors is a recurring theme in many artificial life and decentralized system simulations: carefully constructing the laws of a virtual world which dictate the low-level interactions can lead to the evolution of interesting high-level interactions and emergent behaviors.

The bulk of artificial life simulations to this point have focused on designing novel low-level rules in order to evolve novel lifelike behaviors and observing life as it *could* be. These types of experiments often use discrete time steps and agent states. Conway's famous game of life creates a lifelike cellular automaton in which each cell picks a new state based on the states of its neighbors (Gardner 1970). The Tierra system presented by Ray (1992) uses a virtual machine to evolve programs which compete for computational resources. The low-level rules can even correspond to specific problems or games in order to evolve solutions or strategies. Lindgren and Nordahl (1994) used an agent's performance in the Iterated Prisoner's Dilemma as a fitness function in order to evolve strategies for the game.

While most of these simulations use rules that are loose analogies of real life concepts of evolution and survival, other artificial life simulations have focused on the goal of implementing highly realistic environments in an attempt to evolve equally realistic simulated creatures. Sims (1994) used a realistic physics environment to evolve 3D creatures capable of walking, swimming and competing. Lipson and Pollack (2000) took this technique a step further by evolving physically realistic mobile robots in simulation and then building real-world copies. The technique of building realistic simulations does not necessarily involve physical simulation, but can also involve low-level simulation of biological systems. Yaeger (1993) used artificial neural networks to evolve virtual creatures capable of developing realistic survival strategies.

Despite the impressive artificial life simulations that have been presented through the years, they seem to be few and far between. The limiting factor in implementing many of these artificial life experiments is the amount of development required to simulate the virtual world. Many of the techniques associated with realistic physical and biological simulation are active fields of research and the approaches to implementing such techniques are not always clear-cut.

The BREVE simulation environment (available from http://www.spiderland.org/breve) is presented which provides a simple programmable framework for artificial life in which creating realistic 3D simulations is as easy as defining the state of the environment and the behaviors of the agents. An agent's behavior, for example, may be to query the positions of other objects and adjust its velocity accordingly. The "guts" of the simulation—graphical display; physical simulation; acceleration, velocity and positions of agents; integration and object management—are all managed by the BREVE engine.

Background

There are a number of existing packages designed for the implementation of decentralized system simulations. Many of these packages focus on more abstract types of simulations using discrete time steps and agent states. Though these packages can be quite powerful, they are ultimately not suited to realistic artificial life simulations.

One of the most popular packages for simulation of decentralized systems is Swarm (Minar, Burkhart, Langton & Askenazi 1996). While Swarm provides a sophisticated system for creating hierarchies of objects and triggering events, it does not provide frameworks for 3D simulations or visualization. The package is also somewhat complex—Swarm is a collection of libraries accessed through Objective-C or Java, and not a single integrated application. Writing simulations in Swarm thus involves a great deal of programming overhead.

Another popular package which makes writing simulations considerably easier is StarLogo (Resnick 1994). StarLogo provides a simple Logo-like language and an integrated environment for the simulation of multi-agent systems. The simulations are based around a number of agents on a 2D landscape of "patches". While the environment is easy to use and powerful for more abstract discrete models such as cellular automata, simulating complex 3D worlds and implementing features such as physical simulation is simply not possible.

BREVE is an integrated simulation environment which aims to greatly simplify the implementation of decentralized systems and advanced artificial life simulations. BREVE is designed from the ground up to simulate 3D worlds with continuous time and continuous agent states, and to support realistic physical simulation. In addition to the simulation internals provided by BREVE, the package also aims to be easy to use for non-programmers, and simultaneously flexible enough for more experienced users to interface with other languages, libraries and programs.

The "steve" Language

Simulations in BREVE are written using a simple interpreted object-oriented language called "steve". The steve language, specifically designed for implementation of 3D simulations, includes garbage collection, support for lists and 3D vectors as native types, as well as a set of included classes which interface with the simulation features of the BREVE engine. BREVE also offers a simple plugin architecture which allows users to interface with external libraries or languages and access them through steve.

steve is a procedural language and many of its features should look familiar to users familiar with C. A notable difference is the syntax of method calls which more closely resembles Objective-C. In steve, each argument of a method call is associated with a keyword. This means that arguments are not identified by their order, but instead by their keywords. In the following example, the method `move` takes a single vector argument with keyword `to`:

```
self move to (1, 0, 0).
```

The order of the keyword/argument pair makes no difference to the interpretation of the method call, but may affect its readability. The following two method calls are identical, but the first one reads more naturally:

```
self schedule method-call "print-status"
at-time 20.0.
self schedule at-time 20.0 method-call
"print-status".
```

steve provides a combination of dynamic and static type checking such that certain tasks involving type checking are delayed until runtime. Types must be defined for all variables, but method calls are not bound to return a specific type—different types may be returned depending on the context of the call (if the programmer is daring enough). Likewise, the class of an instance variable does not need to be specified in the code—looking up method calls is done at runtime.

The fact that steve is an interpreted language which delays several type checks and method lookups until runtime can cause speed concerns, but in practice, the bottleneck of BREVE simulations is most often the physical simulation and graphical rendering and not the execution of the steve code.

Included Class Hierarchy

Interacting with BREVE's built-in features is done using the included class hierarchy. The top level of the hierarchy is a class called Object which is the ancestor to all objects created in BREVE. The hierarchy is split into Abstract classes and Real classes. Abstract classes are defined as classes which have no physical representation in the simulated world, while Real classes are associated with specific physical objects.

The Real class includes Mobile and Stationary objects.

The functions of these classes are simple: Mobile objects represent agents which move around during the course of the simulation, while Stationary objects represent objects which are immobile over the course of the simulation such as a floor or obstacles in the world. The Mobile class is also the parent to the class MultiBody which is used to create articulated bodies made up of several links.

The most notable members of the Abstract hierarchy are the classes Control and Data. The class Control provides an interface to the BREVE engine and gives control over features such as camera and light settings, graphical rendering and the user interface. A subclass of Control, PhysicalControl, provides special features required during physical simulations. The Data class is a special class that can be saved to disk or sent over a network. The class is limited in that it may only hold instance variables of simple data types such as integers, floats or vectors but may not hold references to other instances. In order to save other instance types to disk or send them over the network, they need to be encoded into Data objects.

Events and Scheduling

Agent behaviors are written in steve as part of an object's definition. Each action an agent can take may be written as a separate method that gets called in response to certain events. There are several ways to trigger events within BREVE simulations.

- Events called every iteration

Most instances in BREVE have actions that need to be performed at every iteration of the engine. These actions could include changing velocity or acceleration, examining the environment for certain cues or interacting with other instances in the simulation.

These types of actions are triggered simply by the existence of an `iterate` method in the class definition. All instances of classes which implement an `iterate` method will have the method automatically called at each time step.

There is no way to control the order in which instances are iterated during each time step of the simulation. If iteration order is important for a set of instances, however, users can forgo the automatic iteration for these instances and instead use another instance's `iterate` code to manually call methods for instances in the desired order.

- Events scheduled for a specific time

Events can be scheduled to be triggered at a certain time using the method `schedule`:

```
  self schedule method-call ‘‘<method>’’
at-time <time>.
```

The scheduled method call takes no arguments.

- Events triggered by notifications

Because instances may appear and disappear over the course of a simulation, it can become difficult for instances to keep track of each other. In order to simplify the process of communicating with arbitrary instances, BREVE allows instances to post notifications about their current status using the method `notify`. These notifications then trigger actions for all instances that have registered themselves as observers using the method `observe`.

Using this technique, a new instance can simply register itself as an observer of the key players in the simulation, instead of requiring the existing instances to keep track of other instances which are created and destroyed over the course of the simulation.

This technique can be used not only for simple notifications in which one instance announces a status change to the rest of the instances, but may also be used to set up complex notification hierarchies. Physical instances in the world can be grouped into separate "flocks" by encompassing them in Abstract instances for which the physical instances observe notifications. The instances containing the flocks can then in turn be grouped into more general categories, and so forth. The result is a notification hierarchy that can effectively be controlled using a single instance.

- Events triggered by collisions

Events can be triggered whenever the simulation engine detects a physical collision between two objects. This type of callback handles the behavioral changes that occur when two objects collide, such as one object destroying the other or an exchange of information. The simulation first registers to catch collision events with a specific object type using the method `handle-collisions` which is available to all instances of subclasses of the class Real. When the physics engine detects a collision between two objects, the simulation is stepped forward to the precise time of impact before the collision callback is called.

- Events triggered through the user interface

Simulations can be set up to trigger events in response to user interaction. BREVE allows users to set up a global application menu, as well as contextual menus associated with each physical object in the simulation. This allows messages to be sent to a centralized controller instance, or to specific instances in the simulation at any time. Additionally, a simulation callback is executed when the user clicks on physical objects in the simulated world. The default behavior in this case is to select the object that the user clicks on, but this behavior can be overridden.

- `init` and `destroy`

If they are defined, the special methods `init` and `destroy` are automatically called for an instance when

it is created and when it is freed, respectively. These methods allow the instance to handle initialization and deinitialization, if required.

Running Simulations

Every simulation is controlled by a single "controller" object—an instance of the built-in class Control (or one of its subclasses). The controller instance is defined in the simulation source code file and is the only instance which is automatically created when a simulation is run. Other instances may, in turn, be created by the controller instance in the `init` method. The controller instance also receives a number of callbacks associated with user interaction, such as clicking on objects in the simulated world or selecting menu items, so in this respect, the controller instance can be thought of as the main thread of the simulation.

Physical Simulation

The physical simulation engine takes the instantaneous state of the objects in the world, along with the internal forces (such as joint torque) and external forces (such as gravity) acting upon them and computes the state of the objects at the following time step. The physical simulation engine in BREVE encompasses rigid body simulation, collision detection, collision response, as well as the general state of the simulated world and integration.

While the goal of the simulation engine is to provide realistic simulation, it does not aim to be a precise predictor of real-world behaviors. This distinction means that while the simulation must be "realistic", an emphasis is placed on performance rather than on accuracy. Collision impulses, for instance, are computed according to a fast algebraic algorithm rather than through a slower, but more accurate, simulation of the deformation of the colliding objects. In contrast to highly accurate systems which are expected to predict the motion of complex machinery or the results of a simulated car crash, it is acknowledged that small physical simulation errors will find their way into BREVE simulations. These errors are on the scale of variations due to imperfections in the ground. Care is taken, however, to enforce conservation of energy laws in order to prevent evolving agents from exploiting bugs in the physical simulation.

The physics engine currently implemented in BREVE is a work-in-progress and does have a number of limitations, but it is sufficient to simulate articulated creatures learning to walk on a flat plane such as those exhibited by Sims (1994).

Integration & Performance

Integration is done using either a simple Runge-Kutta 4th order integrator or a Runge-Kutta-Fehlman integrator with adaptive step-size control. The initial integration step for physical simulations is generally between .005 and .01 seconds. In addition to integration step, the speed of a simulation depends on a large number of other factors. The most important of which are the number of bodies in the simulation and the number of joints in each body, as well as the number of collisions that must be resolved at each time step. One of the demo simulations which features a single walking articulated body made up of 9 links can be run at more than 2x real-time on a 500 Mhz PowerPC Macintosh.

For simulations in which performance is critical, a command-line version of BREVE without graphics or user interface offers a boost in performance. The level of performance gain depends on the relative complexity of the computation and rendering required by the simulation. The best performance gains occur in simulations containing large numbers of objects with relatively simple behaviors; the worst performance gains occur when the rendering is relatively simple and computational needs are intensive, as is the case in most physical simulations.

Articulated Body Simulation

The articulated body simulation algorithm computes the physics of a single body made up of several individual links. The articulated body simulation provided by BREVE was implemented specifically for the simulation of biologically inspired articulated bodies such as insects, animals or simple robots.

The articulated body simulation implemented in BREVE is the $O(n)$ algorithm presented by Featherstone (1983) and described in detail by Mirtich (1996). The algorithm takes as input the state of an articulated body and the forces acting upon it, and gives as output the resulting acceleration of the body and its component joints. Integrating these values yields the velocities and, in turn, the positions of the simulated bodies at each time step.

Collision Detection

The collision detection algorithm implemented in BREVE is based on the Lin-Canny closest feature algorithm (1993) with many of the improvements recommended by Mirtich (1998).

The algorithm works in two passes: the pruning pass which examines objects' bounding boxes to quickly determine which object pairs may be overlapping, followed by the more detailed collision detection pass of all candidate object pairs. The pruning pass works simply by projecting the bounding box vertices of all of the objects in the simulation onto each axis, and sorting the resulting lists. By inspecting the positions of the sorted minima and maxima, it is possible to determine whether two objects are overlapping on a given axis—those overlapping on all three axes are considered to be collision candidates. Although performing this check is in the worst case $O(n^2)$, in practice objects move very little from one time step to the next, so the lists remain almost sorted

at all times and this step of the simulation approaches $O(n)$.

Object pairs are examined using the second collision detection pass if and only if their bounding box vertices overlap on all three axes. The second pass works by locating the closest feature pair between two objects and calculating the distance between them. As with the bounding box phase, this stage exploits coherence, or the fact that object positions change very little from one time step to the next such that the closest features for a given object pair are often already known from previous iterations.

Many simulations also require agents to query the positions of other objects in the simulation before determining what kind of action to take. Although agents are often only interested in nearby objects, the task of determining which objects are "nearby" requires examining the position of every other object in the simulation. Performing this type of check quickly becomes the bottleneck of simulations with a large number of agents, even if the interactions between neighboring objects are relatively simple. This is especially relevant when dealing with flocking behaviors, swarms, or implementing senses such as smell or hearing.

BREVE solves this problem by offering a feature called "neighbor detection" which uses a second pruning pass of the collision detection algorithm in order to find object pairs which are not at risk of colliding with each other, but which are below a certain per-object user-defined distance threshold that makes them "interesting". This simple technique means that many complex simulations that previously grew with complexity $O(n^2)$ can now be simulated in near $O(n)$ time, depending on the distribution of the objects in simulated space. In the worst case scenario in which all objects are overlapping one anothers' neighbor spaces, all object pairs would be registered as neighbors and the simulation would still run with $O(n^2)$ complexity.

Collision Response

The collision response algorithm implemented by BREVE is the Chatterjee and Ruina (1998) collision impulse algorithm in which collision impulses are estimated mathematically according to the positions and velocities of the colliding objects. The algorithm accounts for both static and dynamic friction.

At low speeds, contact "spring" forces are also applied in order to ensure that objects don't penetrate. These spring forces mean that fewer collisions need to be resolved and that larger integration steps can be taken in situations in which objects attain some sort of resting contact. In most physical simulation environments, spring forces are almost always avoided in favor of more accurate contact force algorithms, but due to the rather small number of situations that require any kind of contact force to be applied, the spring forces are adequate in the BREVE environment.

Examples

BREVE includes several demos which exhibit the power of the simulation engine and language. The demos implemented should be quite familiar to artificial life or decentralized systems researchers. The most notable difference is that in each case, the BREVE implementation requires only a few hundred lines of code. The more tedious steps of implementing these simulations, such as integration, physical simulation and graphical display are all handled automatically by the BREVE engine. The included demos also illustrate the diversity that BREVE is capable of. A few of the included demos are described below.

Flocking

The "Swarm" demo is more or less an implementation of Craig Reynolds classic "Boids" model (1987). Using simple urges, such as matching velocity with and maintaining distance from neighboring birds, simulated birds exhibit realistic flocking behaviors. These urges are accumulated with certain weights in order to determine the birds' instantaneous accelerations.

The BREVE implementation is quite simple: each bird, in its `iterate` method, queries its neighbors for their velocities and locations, and adjusts its own acceleration accordingly. Users can dynamically adjust the weights placed on each urge in order to evoke different kinds of flocking behaviors.

Evolution of Walking Behaviors

In the "Walker" demo, a physically realistic simulated creature learns to walk using a genetic algorithm. The demo is inspired by Sims' creatures, but is ultimately a far more simple implementation. The most notable difference is that the body of the creature is hard-coded with four limbs, with each limb composed of two sections. Instead of using a neural network to control the motion of the joints, each joint is controlled by a sine-based function of time with a number of variables controlling the wavelength of the sine cycle, a phase shift and an amplitude shift. The values of these variables are evolved for each joint.

The implementation in BREVE involves a single creature which tries different walking strategies for 20 seconds at a time, noting how far it moves using each strategy. After every four tests, the best two strategies are bred together to replace the worst two. After a short period of time, the creature learns to coordinate its limbs in order to walk.

Diffusion Limited Aggregation

Diffusion limited aggregation is a simple decentralized model of fractal growth (Witten & Sander 1981). The model begins with a single seed particle. Other particles

then float in from an arbitrarily long distance one at a time. When the random particles touch any part of the growing mass, they freeze to become part of the structure themselves.

The BREVE implementation of this algorithm is incredibly simple. In order to simplify the simulation and improve performance, the mass is made up of stationary objects and the same random walking object is reused. The incoming particle is implemented as an object with two simple behaviors: the first behavior which changes velocity randomly is performed at every time step and is implemented as the `iterate` method; the second behavior is implemented as a collision callback which is activated whenever a collision with the growing particle mass is detected. When a collision occurs, a new particle is attached precisely at the point of collision and the random walking object is moved out to a new starting point far away from the particle mass. After a short period of time the growing particle mass exhibits a fractal appearance.

Though the model is normally presented in 2D, the BREVE implementation can be switched to a 3D simulation simply by changing a variable corresponding to the permitted velocities of the random walking object.

Future Work

Future work on the BREVE environment involves refining its use as an artificial life experiment platform. Work is already in progress to integrate advanced networking capabilities, recurrent and feed forward neural network simulations and more complex terrains. Another important goal is to parallelize the physical simulation engine such that real-time simulations can be run including perhaps hundreds of complex physically simulated creatures instead of just a few at a time.

Once these goals have been achieved, the focus will turn to creating realistic artificial life simulations in which creatures with evolved morphology and behaviors compete for resources in a physically simulated world.

Other work is in progress which explores swarm-based evolution and computation techniques using simulated worlds in BREVE. One such experiment involves integration with the Push programming language used for genetic programming (Spector & Robinson 2002). An important goal of this research is to find alternatives to traditional evolutionary computation techniques which use centralized controls for tasks such as determining fitness and breeding. Swarm-based evolutionary computation may offer many advantages over traditional algorithms, such as more natural visualization and more intuitive solutions to problems such as maneuvering around local minima and maintaining diversity in the population.

Conclusion

The BREVE simulation environment is a unique environment which is well suited for simulations of artificial life and other decentralized systems. A special emphasis is placed on the goal of implementing highly realistic low-level simulations in order to evolve highly realistic high-level behaviors. Features such as the ability to simulate and display continuous 3D worlds, an easy to use object-oriented language, as well as physical simulation capabilities distinguish BREVE from existing simulation packages.

Acknowledgments Support was provided by the Defense Advanced Research Projects Agency and Air Force Research Laboratory (agreement number F30502-00-2-0611, Lee Spector, P.I.).

References

Chatterjee, A., and Ruina, A. 1998. A New Algebraic Rigid Body Collision Law Based On Impulse Space Considerations. *Journal of Applied Mechanics* 65:939-951.

Gardner, M. 1970. The fantastic combinations of John Conway's new solitaire game "Life". *Scientific American* 223:120-123.

Featherstone, R. 1983. The calculation of robot dynamics using articulated-body inertias. *International Journal of Robotics Research* 2:13-30.

Lin, M. 1993. *Efficient Collision Detection for Animation and Robotics*. Ph.D. thesis, University of California, Berkeley.

Lindgren, K., and Nordahl, M. G. 1994. Evolutionary dynamics of spatial games. *Physica D* 75:292-309.

Lipson, H., and Pollack, J. B. 2000. Automatic Design and Manufacture of Robotic Lifeforms. *Nature* 406:974-978.

Minar, N.; Burkhart, R.; Langton, C.; and Askenazi, M. 1996. The Swarm Simulation System: A Toolkit for Building Multi-Agent Simulations. Sante Fe Institute working paper 96-06-042, Sante Fe, NM.

Mirtich, B. 1996. *Impulse-based Dynamic Simulation of Rigid Body Systems*. Ph.D. thesis, University of California, Berkeley.

Mirtich, B. 1998. V-Clip: Fast and Robust Polyhedral Collision Detection. *ACM Transactions on Graphics* 17:177-208.

Ray, T. S. 1992. Evolution, Ecology and Optimization of Digital Organisms. Santa Fe Institute working paper 92-08-042, Sante Fe, NM.

Resnick, M. 1994. *Turtles, Termites, and Traffic Jams: Explorations in Massively Parallel Microworlds*. Cambridge, MA: MIT Press.

Reynolds, C. W. 1987. Flocks, Herds, and Schools: A Distributed Behavioral Model. *Computer Graphics* 21(4) (SIGGRAPH '87 Conference Proceedings): 25-34.

Sims, K. 1994. Evolving 3D Morphology and Behavior by Competition. In Brooks, R., and Maes, P., eds., *Artificial Life IV Proceedings*. Cambridge, MA: MIT Press.

Spector, L., and Robinson, A. 2002. Genetic Programming and Autoconstructive Evolution with the Push Programming Language. *Genetic Programming and Evolvable Machines*, 3(1):7-40.

Witten, T. A., and Sander, L. M. 1981. Diffusion-Limited Aggregation, a Kinetic Critical Phenomenon. *Physical Review Letters* 47:1400-1403.

Yaeger, L. 1993. Computational Genetics, Physiology, Metabolism, Neural Systems, Learning, Vision, and Behavior or PolyWorld: Life in a New Context. In Langton, C. G., ed., *Artificial Life III Proceedings*. Redwood City, CA: Addison Wesley Longman.