2013 IEEE/RSJ International Conference on
Intelligent Robots and Systems (IROS)
November 3-7, 2013. Tokyo, Japan

ReMod3D: A High-Performance Simulator for Autonomous, Self-Reconfigurable Robots

Thomas Collins, Nadeesha Oliver Ranasinghe, Wei-Min Shen

Abstract— **Three-dimensional, physics-based simulators are important to the field of self-reconfigurable robotics because they allow researchers to approximate the physical interactions and autonomous behaviors of large numbers of modules in a low-cost, safe, and highly-controlled manner. This paper presents a novel, high-performance, general-purpose simulator for autonomous, self-reconfigurable robots called ReMod3D (RM3D) that overcomes the speed and scalability limitations of existing self-reconfigurable simulators while, at the same time, allowing for realistic module structures, complex environments, and high physical simulation fidelity. While most existing self-reconfigurable simulators view modules as actuated physical bodies with programmable controllers, RM3D views them as embodied agents, defined not only by their physical *bodies* (links, joints, docks, sensors, actuators) but also by their *minds* (actions, percepts, behaviors, world models) and the noise inherent in the interaction between sensors, actuators, and the environment. RM3D also simulates inter-module dock connection breakage, something novel for self-reconfigurable robot simulators. Additionally, we present experimental evidence showing that this novel architecture makes RM3D well-suited to locomotion, manipulation, reconfiguration, and embodied intelligence research.**

I. Introduction

The successful development of an autonomous robotic system suitable for deployment in the real world requires the careful design and evaluation of both hardware and control software. Hardware failures, power limitations, and manufacturing costs are just a few of the difficulties inherent in performing design-build-test iterations for robotic hardware in the physical world. Building and evaluating robotic control software using real hardware is also challenging; unpredictable real-world environments may make validating the software difficult, and there is always a risk of software bugs causing damage to potentially expensive prototype hardware.

Developing robotic systems that are modular and self-reconfiguring amplifies these challenges. Such systems [1]–[7] often require a large number of physically-interacting modules to achieve reconfigurability, which increases hardware costs, power requirements, and the likelihood of hardware damage or malfunctions. In addition, many self-reconfiguring robotic systems, such as ATRON [1] and SuperBot [4], utilize sophisticated distributed control algorithms that may be difficult to develop, debug, and validate without the aid of a simulator. When hardware is in limited supply due to cost or time constraints, researchers may be unable to evaluate the effectiveness of control software on large populations of modules without simulation tools.

It is apparent, then, that three-dimensional, physics-based simulators are of vital importance to the field of self-reconfigurable robotics. Self-reconfigurable robot simulators must meet a number of criteria: (1) they must be highly *scalable* in terms of the number of modules that can be simulated simultaneously; (2) they must provide good *performance* (simulation speed), even on a single computer; (3) they must support *complex module structures* in order to represent real-world hardware faithfully; (4) they must have high *physical fidelity* so that there is a high degree of similarity between simulated module behavior and real-world module behavior as well as between simulated environments and real-world environments; (5) they must simulate the autonomous *docking and undocking* of modules and provide *inter-module communication* facilities that are aware of the connections between modules; (6) they must provide general and powerful mechanisms for specifying module *intelligence* (behaviors) in order to be useful in embodied intelligence research; (7) they must be highly *extensible* to support new simulated hardware and software.

It is in compliance with these requirements that we designed and built ReMod3D (RM3D). RM3D is a general-purpose, three-dimensional, physics-based simulator for autonomous, self-reconfigurable robots built in C++ that utilizes the NVIDIA® PhysX® physics engine[1]. RM3D is novel in terms of its scalable performance and fidelity, its ability to simulate dock connection breakage, and its architecture, making it well-suited to many diverse types of research in the field, including locomotion, manipulation, reconfiguration, and embodied intelligence. Figure 1 provides some example uses of RM3D.

This paper is organized as follows: in Section II, we discuss previous work related to RM3D. In Section III, we discuss the software architecture of the RM3D simulator. Section IV gives results and describes the novel simulation abilities provided by RM3D. Section V concludes with future research directions.

II. Related Work

Building simulators for self-reconfigurable robots is a challenging endeavor because it requires knowledge of robotics, 3D graphics, the inner workings of physics engines, and software engineering. This difficulty is heightened by the fact that the criteria for these simulators (Section I) are often

Thomas Collins, Nadeesha Oliver Ranasinghe and Wei-Min Shen are with Information Sciences Institute, The University of Southern California, Los Angeles, U.S.A. collinst@usc.edu, nadeeshr@usc.edu, and shen@isi.edu

[1] http://www.geforce.com/hardware/technology/physx

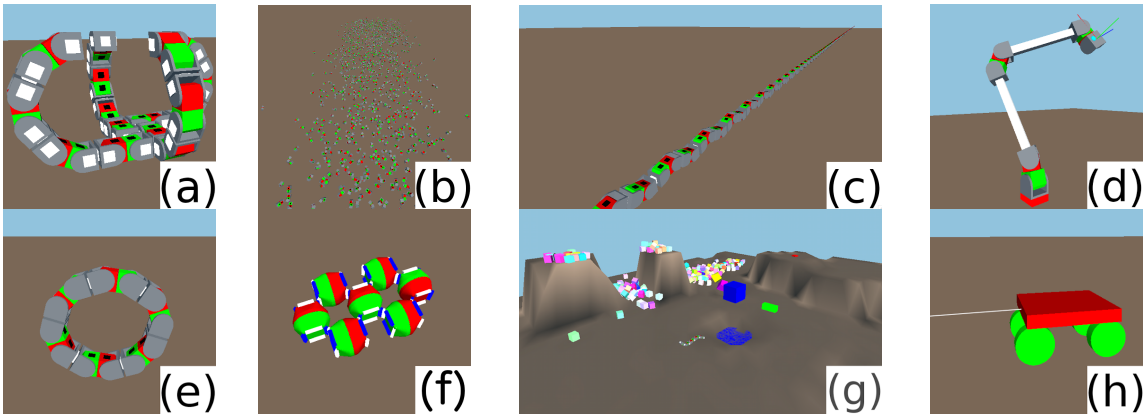

Fig. 1. Example uses of ReMod3D. From top left to bottom right: (a) a complex structure made of SuperBot [4] modules; (b) part of a simulation scene populated with 1000 autonomous SuperBot modules moving all their motors simultaneously; (c) part of a chain made of 1000 autonomous SuperBot modules connected front-to-back executing a sinusoidal snake motion; (d) a 9-DOF manipulator made of SuperBot modules and elongated docks; (e) a 6-module SuperBot rolling track; (f) 7 ATRON [1] modules connected in an 8-shape; (g) a 5-module SuperBot snake in a complex environment with a custom height field terrain, 1000 fluid particles in a pool, and over 200 rigid body obstacles; (h) a mobile robot (Wheelbot) with a visualized range sensor and four wheels, each of which has its own independent velocity drive. With the exception of (b) and (c), which were scalability tests, all simulations in this figure run in real-time (60 frames per second with a time-step of 1/60th of a second). Simulations (b) and (c) run at 1/10th real-time speed (3 frames per second with a time-step of 1/30th of a second). These experiments were run on a MacBook Pro with a 2.4 GHz Intel Core i7 and 8 GB of RAM.

difficult to meet simultaneously. For example, increasing physical fidelity tends to decrease performance, and allowing complex module structures often adversely affects scalability. In this section, we explore the existing general-purpose and specialized robot simulators that are related to RM3D with a particular focus on the three-dimensional physics engines they use and the module frameworks they support.

A. Physics Engine

The 3D physics engine of a simulator greatly affects its scalability, performance, its ability to support complex module structures, and its physical fidelity (criteria 1-4). The three most popular and powerful physics engines available to researchers today are the Open Dynamics Engine (ODE)[2], Bullet[3], and PhysX®.

Open Dynamics Engine. ODE is an open-source 3D rigid body dynamics solver. It has been used as the primary 3D physics engine in many robotics simulators including the popular mobile robot simulator Gazebo [8], the proprietary mobile robot simulator Webots [9], the swarm robot simulator ARGoS [10], and Robot3D [11] (part of the Replicator/Symbrion project [12]). Using ODE, these simulators have produced high-fidelity, three-dimensional simulations of smaller numbers of complex robots in realistic environments, which suits their intended purposes well. However, the authors have extensive experience with ODE in the form of the ODE-based Superbot Simulator [13]–[15][4] and have observed that its performance and fidelity rapidly break down as the number of autonomous modules increases, making it a poor candidate for the physics engine in a self-reconfigurable robot simulator. The Superbot Simulator suffers from extremely poor performance with as few as 20 SuperBot modules, and is completely unusable with 100 or more SuperBot modules due to collision detection failures and incredibly slow simulation speeds stemming from ODE's inefficiencies. ODE is also hampered by its lack of built-in GPU and multithreading support for physics calculations, its lack of built-in support for simulating particle systems such as fluids, and its lack of soft-body physics. For these reasons, we chose not to extend any ODE-based simulator for our purposes.

Bullet. In contrast to ODE, the open-source Bullet physics engine offers solid performance and fidelity in a way that is scalable, even with complex robotic modules in complex environments. Bullet also features multithreaded physics calculations, soft body physics, particle systems, and fluids. The most notable simulators making use of Bullet are the mobile robot simulators V-REP[5] and MORSE [16] as well as the Unified Simulator for Self-reconfigurable Robots (USSR) [17], a general-purpose self-reconfigurable robot simulator.

Both V-REP and MORSE simulate complex mobile robots in realistic environments. However, MORSE's scalability is limited by design; it is intended to support only approximately 15 autonomous modules simultaneously on a single computer at 20 frames per second. V-REP, on the other hand, is proprietary, which severely limits its extensibility. The USSR offers good performance and supports complex module structures, but its scalability and physical fidelity are its limiting factors. The USSR has been shown to simulate only up to 303 Odin modules [18] at 1/10th real-time speed on a 2.4 GHz dual-core MacBook Pro with 4 GB of RAM. Importantly, these simulated Odin modules are made of simple spheres, cylinders and cones, which are much easier for physics engines to simulate than more general convex meshes. Additionally, to the best of our knowledge, the USSR does not allow users to create realistic environments

[2]http://www.ode.org
[3]http://bulletphysics.org/wordpress/
[4]Videos available at http://www.isi.edu/robots/media-superbot.html
[5]http://www.coppeliarobotics.com

involving height field terrains and particle systems (debris, fluids) that can interact with simulated modules, limiting its fidelity. At the time of this writing, the USSR's physical fidelity and scalability limitations also stem from Bullet's primary downside: its lack of built-in multi-GPU support for accelerating physics calculations.

PhysX®. PhysX® is a commercial-grade physics engine developed by NVIDIA®. PhysX® has been used in a number of robot simulators, both general-purpose and specialized. These include USARSim [19], designed for simulating search and rescue operations with mobile robots; Microsoft Robotics Developer Studio[6], designed for high-fidelity mobile robot simulations in realistic environments; and SwarmSimX [20], a recent real-time, multi-robot simulator. PhysX® provides excellent scalability, performance and fidelity, native multi-GPU support, native multithreading support, soft body physics mechanisms, and GPU-accelerated particle, fluid and cloth simulations. These features make it the most well-suited of the three major physics engines for use in a self-reconfigurable robot simulator; however, none of the existing PhysX® simulators meet all the criteria for self-reconfigurable robot simulators, nor can they be easily modified to do so.

As [10] point out, USARSim's architecture was not designed to provide easy access to underlying models, meaning that it would be very difficult to extend it to support general-purpose docking mechanisms and connection-aware communication between modules in a way that facilitated further extensibility by users. As was the case with V-REP, the proprietary nature of Microsoft Robotics Developer Studio severely limits its extensibility. Finally, SwarmSimX was designed explicitly to model physically-independent multi-robot systems. Though its design makes it extensible and it is open-source, most of the simulator's core functionality would have to be dramatically altered to add self-reconfigurable functionality, saving minimal, if any, work.

B. Module Framework

The module framework provided by a self-reconfigurable robot simulator is the set of abstract software mechanisms that allows users to define new modules by specifying their properties and basic functionality and provides users with an application programming interface (API) to populate simulated scenes with them. The module framework is largely responsible for how well a simulator meets criteria 5-7 (Section I).

Existing self-reconfigurable simulators, such as the USSR, focus heavily on *body* research (locomotion, manipulation, reconfiguration) while providing only minimal support for *mind* research (reasoning, planning, learning), which is integral to self-reconfigurable robotics research. The USSR architecture views self-reconfigurable modules as physical robots with controllers. Such a simulation design is well-suited to locomotion, manipulation, and physical reconfiguration research, but it is not as well-suited to embodied intelligence research. For instance, to the best of our knowledge, the USSR does not provide any generalized capabilities for adding sensor and actuator noise, nor does it support sensors that do not return floating point values. This would make it difficult to simulate reasoning and planning under varying degrees uncertainty or to create abstract sensors (such as an obstacle sensor that returns a set of features about the nearest obstacle). These module intelligence limitations also limit the USSR's extensibility.

As discussed in subsequent sections, RM3D overcomes the speed and scalability limitations of existing self-reconfigurable simulators while, at the same time, allowing for realistic module structures, complex environments, and high physical simulation fidelity (criteria 1-4) primarily by utilizing the PhysX® engine, which, importantly, utilizes the supported GPUs on the user's computer to accelerate calculations.

The module framework of RM3D, which views modules as embodied agents - autonomous entities associated with a physical body (links, joints, docks), a model of the world (actions taken, observations), a behavior (program), a set of sensors, a set of percepts, a set of actuators, and a set of actions - facilitates extensibility and makes RM3D suitable for many diverse types of research in the field of self-reconfigurable robotics, including both *body* and *mind* (intelligence) research. The way in which RM3D's software architecture (Section III) makes use of this module framework demonstrates that RM3D does, in fact, meet criteria 5-7 for a self-reconfigurable robot simulator as well.

III. Software Architecture

Architecturally speaking, RM3D is comprised of a simulation core that manages populations of modules connected in arbitrary ways via their docks, an interface to the PhysX® engine, and an OpenGL[7] renderer. To run simulations, users instantiate *experiments*, which are C++ programs with a *main* function (to initiate the simulation) and a subclass of the *Experiment* class, which defines an *init* function and a *shutdown* function. The *init* function is used to create instances of existing module types, configure simulation environments (using the *Environment* class), and load module behaviors. The *shutdown* function is used to release dynamic memory. The simulation core calls these functions at the appropriate time. Figure 2 visualizes this architecture.

A. The Simulation Core

The simulation core contains the primary *step* function of the simulator as well as the renderer runloop. The *step* function allows modules to read their sensors, execute actions, send and receive messages, and execute their behaviors. It also manages all interactions between modules (such as messaging, docking and undocking). The *step* function interfaces with PhysX® to simulate the physical phenomena of a scene for a user-defined period of time, called a *time-step*, and update the simulation state. The *step* function is integrated with the renderer runloop such that one *time-step*

[6]http://www.microsoft.com/robotics

[7]http://www.opengl.org

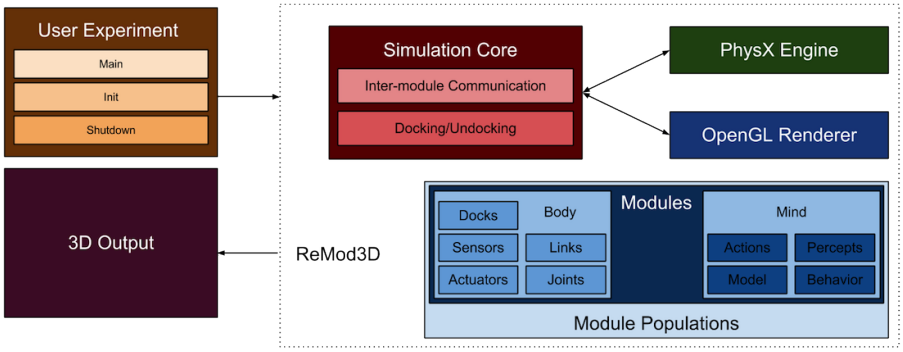

Fig. 2. The simplified software architecture of the ReMod3D simulator.

Algorithm 1: Simplified simulator step function

Input: None
Output: None

```
1  if program not paused then
2      foreach module do
3          module.step();
           /* Module reads sensors, executes
              behavior, sends/receives
              messages, executes actions,
              updates model.                 */
4      route messages to message boxes;
5      foreach pair of docked modules do
6          if either dock in connection no longer enabled
           then
7              destroy joint between modules;
8              update connections;
9      foreach pair of enabled, un-engaged docks do
10         if within distance/orientation tolerances then
11             create joint between modules;
12             update connections;
13     physicsEngine.simulate(time-step);
```

of time is simulated immediately before each renderer frame update. Thus, the most up-to-date simulation state is always available for the renderer to display to the user. Pseudocode for this function is given in Algorithm 1.

The simulation core handles inter-module communication by establishing message boxes for each module and updating which docks (of which modules) are connected to one another each time the *step* function is called. This allows RM3D to provide connection-aware communication between modules. More specifically, modules send messages to docks rather than to one another, and the simulation core routes these messages to the correct message boxes using each dock's pointer to the module that owns it. This facilitates homogeneity and anonymity amongst modules, which is important when designing distributed algorithms for self-reconfigurable robots. Particularly important is the *time-step* message propagation delay: if module i sends a message to one of its docks at time t, the module (if any) attached to that dock will receive the message at time $t+1$. If the receiving module (module j) in turn sends a message to one of its docks, the receiving module (module k) will receive the message at time $t+2$, and so on. When messages are read, they are consumed (deleted from the appropriate message box), but they can be read by the receiving module at any time after they become available. Likewise, message sending is done at the discretion of each module's behavior. By making the sending and receiving of messages stochastic and utilizing the built-in noise mechanisms of RM3D, users can easily model more complex, hardware-specific communication.

By continuously updating connection information, the simulation core is also able to facilitate run-time docking and undocking of modules. Docks have two primary attributes: an *engaged* status (currently connected to another dock) and an *enabled* status (able to connect). Whenever two enabled but un-engaged docks come acceptably close to one another (Euclidean distance tolerance) and are acceptably lined up with one another (dock frame orientation tolerance), they are automatically pulled together and connected with a joint. Since module behaviors can disable docks at any point via an action execution, the simulation core may find that two docks are still connected, even though one or both of them is no longer enabled. When this happens, the joint between them is destroyed, disconnecting them. Thus, RM3D supports a "magnetic" docking system that module behaviors can enable or disable on a per-dock basis.

B. Modules

In RM3D, each experiment is populated with one or more modules. Each module has a physical body defined in terms of PhysX® rigid bodies and joints, and a set of data structures representing their docks, actuators, sensors, actions, percepts, and models. Modules also have associated code, called a *behavior*, that they use to select what actions to take based on their current model of the world. Each module has a *step* function that is called within the main simulator *step* function. This function can be used to read sensor data, read pending messages, send new messages, execute a step of the module's behavior, update the module's model, execute

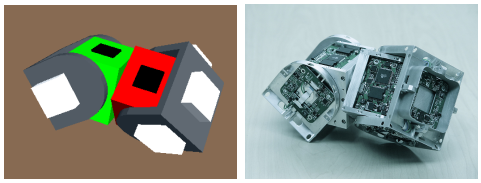

Fig. 3. Simulated (left) and actual (right) SuperBot modules.

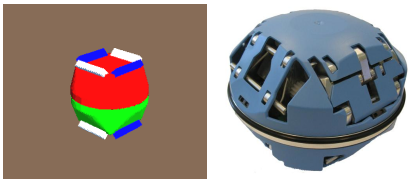

Fig. 4. Simulated (left) and actual (right) ATRON modules.

actions, etc. This function allows module creators to easily specify base functionality that applies to all instances of a module type, preventing the need for users to replicate it in all behaviors. Sensor values are read through percepts, which optionally corrupt readings according to any custom noise model. In a similar way, actuator commands are executed through actions that may add arbitrary noise to the command the actuator performs. The simulation core provides helper methods to populate the simulated scene with predefined modules and initially connect them (when necessary). This extremely general design is equally well-suited to *body* and *mind* research as it is based on the most fundamental ideas in artificial intelligence, machine learning, and modular, self-reconfigurable robotics.

C. Extensibility

Almost every aspect of RM3D was designed to be extensible. RM3D is primarily implemented in abstract and templated C++ base classes designed to provide general simulation functionality. Users create concrete subclasses of these base classes to implement functionality that is specific to a particular type of module or experiment. For example, the authors implemented the simulated SuperBot and ATRON modules shown in many of the figures in this paper as concrete subclasses of the *Module* base class. Sensors, actuators, percepts, and actions were all defined as subclasses of their respective base classes. Users can easily add new types of modules or extend existing modules using these base classes.

We achieved an additional level of extensibility in RM3D using the popular Boost[8] C++ framework. This framework adds dynamic data typing functionality to C++ in the form of the *boost::any* data type, which can hold a value of any C++ data type. Boost provides mechanisms to cast values to and from the *boost::any* type. The *boost::any* type was used throughout RM3D to ensure that we did not make any assumptions about the type of values that users would want to use in sensors, actuators, noise models, etc.

IV. Evaluation

A. Simulated Module Construction

RM3D is general enough to support all types of self-reconfigurable robots, including both chain-based and lattice-based modules. To demonstrate this, we simulated both SuperBot and ATRON modules.

SuperBot. In RM3D, each simulated SuperBot module consists of four rigid bodies connected by three joints. The four rigid bodies consist of ten simulated shapes, including two rectangular solids and eight convex meshes specified by a total of 624 3D data points. The simulated SuperBots are built to scale and have the same three actuated degrees of freedom (including joint limits) as the real SuperBot modules. Each module can be fitted with up to six docks (white) with which it can connect to docks on other SuperBot modules or on specialized elements in the environment. Of importance is the fact that the outer two rigid bodies (gray) are flush against the inner two rigid bodies (red and green), generating intra-module friction. Figure 3 shows a visual comparison between simulated and actual SuperBot modules.

ATRON. The simulated ATRON module consists of two symmetrical hemisphere-like rigid bodies (approximately built to scale), each composed of a single convex mesh shape specified using 276 3D data points, for a total of 552 3D data points per module. Like the real ATRON module, each simulated ATRON has an actuated, free-spinning revolute joint connecting its hemispheres. The simulated ATRON module has eight docks (four per hemisphere), with which it can attach to other ATRON modules. Four of these docks are male (blue) and four of them are female (white). RM3D ensures that docking only occurs between male and female docks. Figure 4 shows a visual comparison between simulated and actual ATRON modules.

B. Locomotion

Figure 1 shows several examples of simulated locomotion achieved in RM3D: snake (Figure 1(c),(g)), rolling track (Figure 1(e)), and wheeled (Figure 1(h)). RM3D has been used to achieve sinusoidal snake locomotion on chains of anywhere between 3 and 1000 simulated SuperBot modules connected front-to-back. RM3D has also been used to create a six-module rolling track that moves autonomously by utilizing a simulated accelerometer. The code for these gaits was adapted from code that controls actual SuperBot modules, and the same behavior was observed on the simulated SuperBot modules as was observed on the actual modules. This is evidence of RM3D's physical fidelity. It also illustrates RM3D's inter-module communication facilities because some of these gaits require synchronization amongst modules. RM3D was also used to achieve wheeled locomotion using both simulated ATRON modules (Figure 5(bottom)) and Wheelbot (Figure 1(h)) by controlling the wheel velocities of these modules.

C. Manipulation

As part of the DARPA Phoenix project [21], RM3D was used to create a 9-DOF manipulator using three Super-

[8]http://www.boost.org

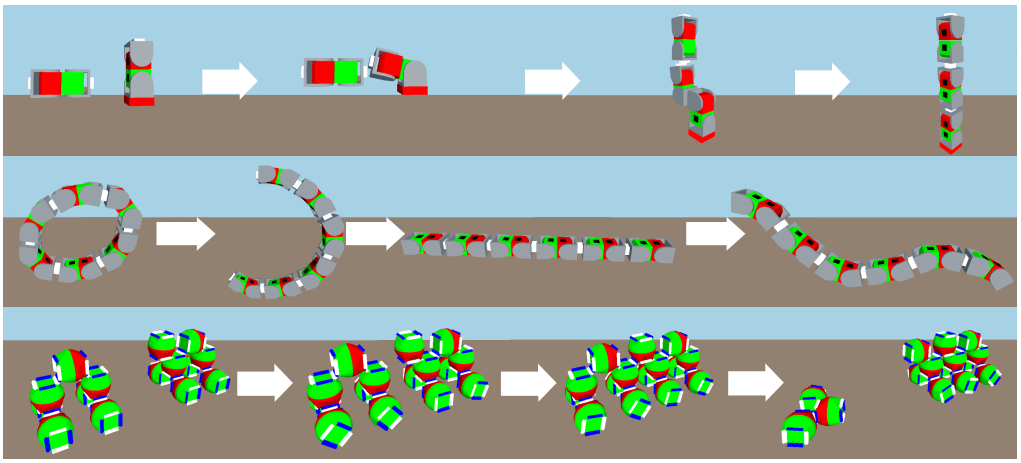

Fig. 5. Examples of self-reconfiguration in RM3D. Top: A single SuperBot module docking with nearby modules to form a 3-SuperBot module arm. Middle: A 6-SuperBot module rolling track reconfiguring into a snake. Bottom: Two ATRON cars docking before a 3-ATRON car undocks and drives away.

Bot modules connected by elongated docks (to extend the manipulator's reach). Both forward and inverse kinematics were applied successfully to the manipulator. We performed numerous experiments in which our arm was able to reach desired positions within 5mm of the intended target using the damped least squares inverse kinematics methodology [22]. This is further evidence of the simulator's physical fidelity. Figure 1(d) shows an image of the simulated SuperBot arm.

D. Self-Reconfiguration

Figure 5 shows three experiments illustrating the self-reconfiguration abilities of RM3D. The top sequence of images shows a SuperBot arm (docked to a base) docking with nearby SuperBot modules to make itself longer. The middle sequence of images shows a rolling track of SuperBot modules reconfiguring into a snake. To achieve this, one of the modules disables one of its engaged docks, causing the simulator to destroy the joint in which that dock was involved. This undocking begins the reconfiguration. The bottom sequence of images shows two ATRON cars docking with one another to produce a 15-module car. In the last image of the sequence, one of the male docks near the rear of the long car is disabled, and a mini-car of three ATRON modules undocks and drives away from the larger car.

E. Realistic Environments

Figure 1(g) illustrates RM3D's ability to support complex and realistic user-defined environments, a key component in physical simulation fidelity. In this example, a five-module SuperBot snake traverses a height field terrain specified using a 30×30 pixel grayscale image. Each pixel value corresponds (through a scaling factor) to a sample of the height field. Also shown in this figure are 1000 fluid particles in a pool and over 200 rigid body obstacles of various shapes and sizes. Users can easily add fluid and debris particle systems of any size as well as any number of obstacles (including cloths) to any environment. If the user's GPU(s) support PhysX®, they will automatically be utilized to accelerate all particle and cloth calculations. As an example, utilizing a mid-range NVIDIA® GeForce® GT 650M GPU, we were able to realistically simulate 20,000 moving particles of fluid at 51 frames per second, 50,000 moving particles of fluid at 15 frames per second, and 100,000 moving particles of fluid at 7 frames per second, all with a time-step of 1/60th of a second. In addition, 100,000 debris particles were simulated at 16 frames per second with the same time-step. See Figure 7 for an example of fluid simulation in RM3D.

F. Dock Connection Breakage

Docking mechanisms are an important aspect of the field of self-reconfigurable robotics. It is important for researchers and practitioners to be able to simulate not only docking and undocking using these mechanisms but also how much force they can withstand before they break. This enables researchers to validate gaits and reconfiguration strategies in terms of how much force they apply to the docks holding modules together. Importantly, performing this validation in simulation is safer and less costly than real-world experimentation because there is no risk of damaging robotic hardware. Figure 6 shows five spheres of high mass colliding with a grid of 45 connected SuperBot modules in zero gravity and the resulting dock connection breakage.

G. Scalability and Performance

In our extensive experiments, RM3D has proven to be very scalable, even with complex modules such as simulated SuperBot and ATRON modules. Figure 1 shows two such experiments done with SuperBot modules. The first experiment, shown in Figure 1(b), involved 1000 disconnected, autonomous SuperBot modules in a single simulated scene. Each SuperBot module executed a simple behavior that moved all of its motors between two states in an alternating pattern. The simulation ran at about 4 frames per second. We also performed an experiment (Figure 1(c)) in which we connected 1000 autonomous SuperBot modules front-to-back in a long snake. Each module executed a sinusoidal snake motion behavior. This simulation ran at 3 frames per second. Additionally, a simulation of 1500 disconnected ATRON

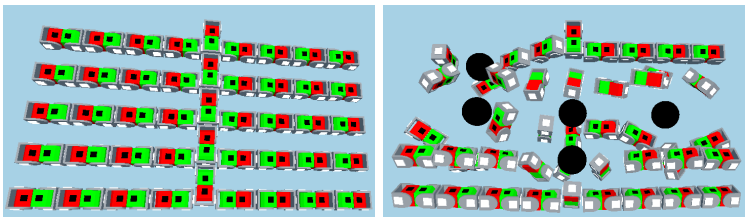

Fig. 6. Example of the novel RM3D dock connection breakage feature.

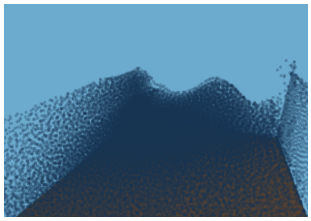

Fig. 7. A simulation of 40,000 moving fluid particles.

modules (each continuously spinning its middle motor) ran at 3 frames per second. These experiments were performed on a MacBook Pro with a quad-core 2.4 GHz Intel Core i7 processor and 8 GB of RAM with a simulation time-step of 1/30th of a second, meaning that they all ran at or over 1/10th real-time speed. Simulations of 200 autonomous SuperBot modules (disconnected), 275 autonomous ATRON modules (disconnected), and snakes of 150 autonomous SuperBot modules ran at 30 frames per second (real-time) on the same MacBook Pro with the same simulation time-step (1/30th of a second).

V. Conclusions and Future Work

In this paper, we presented a three-dimensional, high-performance self-reconfigurable robot simulator called ReMod3D. RM3D is novel in terms of its design, its scalable performance and fidelity, and its ability to simulate dock connection breakage. We presented the results of numerous experiments showing how RM3D meets all the criteria for a self-reconfigurable robot simulator and how it is applicable to wide range of *mind* and *body* research in the field.

We plan, as the next step, to perform a cross-validation of RM3D's simulation fidelity with actual physical hardware. We were able to successfully port code from real SuperBot modules to simulated ones and observe the same results. Next, we will implement a gait developed in RM3D on actual hardware. We also plan to add support for more types of physical docking mechanisms, such as docks with actuated clamps or teeth. Finally, we plan to optimize RM3D to take advantage of the multithreading capabilities of PhysX®. We are confident that this will increase RM3D's scalability and performance. The RM3D simulator (which runs on Mac OSX, Linux, and Windows) along with its source code and videos, are available at http://www.isi.edu/robots/remod3d.

References

[1] M. W. Jrgensen, E. H. stergaard, and H. H. Lund, "Modular atron: Modules for a self-reconfigurable robot," in *In Proc. of the 2004 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*. IEEE Computer Society Press, 2004, pp. 2068–2073.

[2] S. Murata, E. Yoshida, A. Kamimura, H. Kurokawa, K. Tomita, and S. Kokaji, "M-tran: self-reconfigurable modular robotic system," *Mechatronics, IEEE/ASME Transactions on*, vol. 7, no. 4, pp. 431–441, Dec.

[3] K. Kotay, D. Rus, M. Vona, and C. McGray, "The self-reconfiguring robotic molecule," in *Robotics and Automation, 1998. Proceedings. 1998 IEEE International Conference on*, vol. 1, May, pp. 424–431 vol.1.

[4] B. Salemi, M. Moll, and W.-M. Shen, "SUPERBOT: A deployable, multi-functional, and modular self-reconfigurable robotic system," Beijing, China, Oct. 2006.

[5] V. Zykov, P. Williams, N. Lassabe, and H. Lipson, "Molecubes extended: Diversifying capabilities of open-source modular robotics."

[6] M. Yim, K. Roufas, D. Duff, Y. Zhang, C. Eldershaw, and S. Homans, "Modular reconfigurable robots in space applications," *Auton. Robots*, vol. 14, no. 2-3, pp. 225–237, Mar. 2003. [Online]. Available: http://dx.doi.org/10.1023/A:1022287820808

[7] M. Yim, D. Duff, and K. Roufas, "Polybot: a modular reconfigurable robot," in *Robotics and Automation, 2000. Proceedings. ICRA '00. IEEE International Conference on*, vol. 1, pp. 514–520 vol.1.

[8] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *In IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2004, pp. 2149–2154.

[9] O. Michel, "Cyberbotics ltd. webots tm : Professional mobile robot simulation," *Int. Journal of Advanced Robotic Systems*, vol. 1, pp. 39–42, 2004.

[10] C. Pinciroli, V. Trianni, R. O"Grady, G. Pini, A. Brutschy, M. Brambilla, N. Mathews, E. Ferrante, G. Di Caro, F. Ducatelle, T. Stirling, A. Gutierrez, L. Gambardella, and M. Dorigo, "Argos: A modular, multi-engine simulator for heterogeneous swarm robotics," in *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, Sept., pp. 5027–5034.

[11] A. van Rossum. (2011) Robot3d, open source modular swarm robot simulation engine. [Online]. Available: https://launchpad.net/robot3d

[12] European Research Community. (2012, Jan.) Symbrion replicator. [Online]. Available: http://www.symbrion.eu/tiki-index.php

[13] N. Ranasinghe, J. Everist, and W.-M. Shen, "Modular robot climbers," San Diego, CA, Nov. 2007, iROS 2007 Workshop on Self-Reconfigurable Robots, Systems and Applications.

[14] M. Moll, P. Will, M. Krivokon, and W.-M. Shen, "Distributed control of the center of mass of a modular robot," Beijing, China, Oct. 2006.

[15] H. Chiu, M. Rubenstein, and W.-M. Shen, "'deformable wheel'-a self-recovering modular rolling track," Tsukuba, Japan, Nov. 2008.

[16] G. Echeverria, N. Lassabe, A. Degroote, and S. Lemaignan, "Modular open robots simulation engine: Morse," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, May, pp. 46–51.

[17] D. Christensen, D. Brandt, K. Stoy, and U. Schultz, "A unified simulator for self-reconfigurable robots," in *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*, Sept., pp. 870–876.

[18] R. F. M. Garcia, K. Stoy, D. J. Christensen, and A. Lyder, "A self-reconfigurable communication network for modular robots," in *Proceedings of the 1st international conference on Robot communication and coordination*, ser. RoboComm '07. Piscataway, NJ, USA: IEEE Press, 2007, pp. 23:1–23:8. [Online]. Available: http://dl.acm.org/citation.cfm?id=1377868.1377897

[19] S. Carpin, M. Lewis, J. Wang, S. Balakirsky, and C. Scrapper, "Usarsim: a robot simulator for research and education," in *Robotics and Automation, 2007 IEEE International Conference on*, April, pp. 1400–1405.

[20] J. Lächele, A. Franchi, H. H. Bülthoff, and P. Robuffo Giordano, "Swarmsimx: real-time simulation environment for multi-robot systems," in *Proceedings of the Third international conference on Simulation, Modeling, and Programming for Autonomous Robots*, ser. SIMPAR'12. Berlin, Heidelberg: Springer-Verlag, 2012, pp. 375–387. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-34327-8_34

[21] DARPA. Darpa phoenix satellite servicing. [Online]. Available: http://www.darpa.mil/our_work/tto/programs/phoenix.aspx

[22] S. R. Buss, "Introduction to inverse kinematics with jacobian transpose, pseudoinverse and damped least squares methods," IEEE Journal of Robotics and Automation, Tech. Rep., 2004.