A 3D Simulator of Multiple Legged Robots Based on USARSim

Marco Zaratti, Marco Fratarcangeli, and Luca Iocchi

Dipartimento di Informatica e Sistemistica
Università "La Sapienza", Rome, Italy
Via Salaria 113 00198 Rome Italy
lastname@dis.uniroma1.it

Abstract. This paper presents a flexible 3D simulator able to reproduce the appearance and the dynamics of generic legged robots and objects in the environment at full frame rate (30 frames per second). Such a simulator extends and improves USARSim (Urban Search and Rescue Simulator), a robot simulator in turn based on the game platform Unreal Engine. This latter provides facilities for good quality rendering, physics simulation, networking, highly versatile scripting language and a powerful visual editor. Our simulator extends USARSim features by allowing for the simulation and control of legged robots and it introduces a multi-view functionality for multi-robot support. We successfully tested the simulator capabilities by mimicking a virtual environment with up to five network-controlled legged robots, like AIBO ERS-7 and QRIO.

1 Introduction

Robotic simulation is very important in developing robotics applications, both for rapid prototyping of applications, behaviors, scenarios, and for debugging purposes of many high-level tasks. Robot simulators have been always used in developing complex applications, and the choice of a simulator depends on the specific tasks we are interested in simulating. Moreover, simulators are also very important for robotic education: in fact, they are powerful teaching tools, allowing students to develop and experiment typical robotic tasks at home, without requiring them to use a real robot.

2D simulators are widely used to evaluate the behaviors of robotic applications, they are very effective for many kinds of robots and applications, and are easy to use and to customize. However, there are cases in which a 2D simulator is not sufficient. For example, for mobile robots with higher mobility than wheels (e.g., legged or snake-like robots) and in 3D environments, a 2D simulator may be too simplistic to correctly model some behaviors.

A 3D simulator for mobile robots must also correctly simulate the dynamics of the robots and of the objects in the environment, thus allowing for a correct evaluation of robot behaviors in the environment. The required accuracy of dynamics simulation depends on the particular behavior we are interested in evaluating. Moreover, real-time simulation is important in order to correctly model interactions among the robots and between the robots and the environment.

G. Lakemeyer et al. (Eds.): RoboCup 2006, LNAI 4434, pp. 13–24, 2007.

Since simulation accuracy is computationally demanding, it is often necessary an approximation to obtain real-time performance.

Another important feature of a robotic simulator is easy integration of different robotic platforms, different scenarios, different objects in the scene, as well as support for multi-robot applications.

Finally, visual realism is not fundamental for robotic simulation, since may be not adeguate to experiment and evaluate low-level sensor processes, such as image processing. However, visual realism usually has a minimum impact on the performance of the simulator (since most of the computation can be demanded to graphics adapters of the PCs), while a simulator with visual realism can be more attractive.

3D Robotic Simulators. There exist already several simulators that handle the issues discussed above. As our aim is to address 3D physics simulation, let us see how this feature is integrated in the simulators that have been used within the RoboCup Four Legged league and in general for 3D modeling of complex robots. There are several factors that make realistic robotics simulation hard to achieve. In order to represent a valid tool for the robotics researcher, the simulator must fulfill a number of requirements:

Flexibility: the simulator must allow for the simulation of different robots, not even know *a priori*, as well sensors and actuators. The generic virtual environment where the robots are placed, should be easy to model as well;

Physics Realism: to obtain plausible results, interaction among robots and between robots and the virtual environment must be carefully modeled through the physical laws of rigid body dynamics;

Visual Realism: the appearance of the whole system must be as accurate as possible to guarantee consistent sensor readings (e.g., images, audio);

Efficiency: simulation must be carried out in the most efficient way, hopefully in real-time, with a visualization frame rate of 30 frames per second;

Modularity: it must be easy to add and modify the features of the environment and of the robots, including the sensors input/output;

Effective Control: the simulator should be flexible enough to be easily interfaced to the same programming code that is used on the real robots.

The Asura Team[1] provides a development kit, namely the ASURA RoboCup Software, aiming at reproducing the Four-legged League environment. Such a development kit permits to develop strategies and sensor acquisition and processing. However, it lacks in flexibility since it can simulate only the AIBO ERS-210 robot in the RoboCup framework and it does not permit dynamics simulation, leading to a poor representation of the virtual system.

Zagal and Ruiz-del-Solar introduced UCHILSIM [7] in 2004. Such a simulator reproduces with high fidelity the dynamics of AIBO motions and its interactions with the objects in the game field. Physical simulation is carried out through

[1] www.asura.ac.

the open-source physical engine Open Dynamics Engine[2] (ODE) and objects (e.g., robots) are defined in the VRML standard. The goal of this project was to became a standard framework for learning complex AIBO behaviors. This simulator was quite promising but it seems to be no longer developed.

Gazebo [3] is a multi-robot 3D simulator with graphical interface and dynamics simulation (through ODE). It is able to simulate a wide range of sensors and it comes with models of existing robots even if the simulator does not allow to define complex objects (e.g., dummies for rescue arenas, moving people in the scene, a ball in the soccer field). The robots and sensors can be controlled by the Player [2] server or controllers can be written using a library provided with the simulator. Simulated environment are described in XML and new robot/sensor models can be created as plug-ins. Simulation of legged robots is supported but not extensively used in the current release.

SimRobot [4] by Laue *et al.* simulates arbitrary user-defined robots in three-dimensional space. To allow an extensive flexibility in building accurate models, a variety of different generic bodies, sensors and actuators has been implemented and specified in XML. The robot controllers are directly linked with the simulator library to produce an executable file. Furthermore, the simulator follows a user-oriented approach by including several mechanisms for visualization, direct actuator manipulation, and interaction with the simulated world. Dynamics is simulated through the ODE engine.

Webots[3] is a commercial general purpose mobile robotics simulation software. It uses ODE to simulate dynamics and it has an extensive library of actuators, sensors and robots like Aibo, Lego Mindstorms, Khepera, Koala and Hemisson. While the mechanical features of the robots are well defined, the main limitation of this simulator resides in the poor quality of the 3D graphical representation of the virtual environments, including robots and in the lack of adequate modeling tools.

USARSim (Urban Search and Rescue Simulator) [6,1] is a robot simulator based on the industrial game engine Unreal Engine[4]. It simulates the reference test arenas developed by National Institute of Standards and Technology (NIST) and robots intended for the Urban Search & Rescue (USAR) tasks. Since Unreal Engine has been deployed for the development of networked multi-player 3D games, it solves many of the issues related to modeling, animation and rendering of the virtual environment. It is a complete game development framework targeted at today's mainstream PC, it provides tools to rapidly develop objects and environment (Unreal Editor) and it is possible to define the behavior of the objects through an ad-hoc script language (Unreal Script). Dynamics of rigid bodies is transparently handled by the Karma physical engine [5]. As in Gazebo, robot controllers use a TCP/IP interface to control the robots and so may be programmed in any language that supports networking. The Client/Server architecture can be advantageously used to carry out complex computations on dedicated machines decoupling the simulation from intelligence processing.

[2] Open Dynamics Engine www.ode.org.
[3] www.cyberbotics.com/products/webots/.
[4] www.unrealtechnology.com.

However, the current version of USARSim is not able to simulate legged robots, like AIBOs, and it has several limitations like the total number of joints allowed for each robot, a limited support for multi-robot scenarios and an approximative collision handling.

Discussion. From the analysis of the existing simulators it appears that a general 3D simulator for legged robots with good dynamics simulation, multi-robot support, realistic appearance, and easy-to-use editing tools is not currently available. USARSim is the most promising, since it already implements many required features and it can be easily extended. Unreal Engine has a significant industrial support and a simulator based on it will benefit from new releases of this engine as soon as they are available, with minimum effort. Unreal Engine uses a different physical engine (Karma) than other simulators (using ODE). To our knowledge there are no comparative studies between these two engines and we believe the choice of ODE is given only by its open-source code. Unfortunately, the current version of Unreal Engine (and USARSim) makes a partial (and sometimes incorrect) use of the Karma engine. Finally, USARSim has been chosen as the standard simulator for a new RoboCup Rescue simulation league.

In this paper we present a 3D simulator based on USARSim[5] that allows for modeling complex legged robots (such as quadrupeds and humanoid ones) and for simulating their interaction. The performance of the system allows for simulating in real-time up to five of these robots in the environment. The simulator presented in this paper extends USARSim by introducing some important features: 1) it allows for the simulation and control of legged robots (four-legged, humanoids, etc.) 2) it introduces a multi-view functionality for multi-robot support. Moreover, we fixed a few problems in the use of the physical engine Karma, that was not fully and correctly integrated in the Unreal Engine.

We successfully tested the simulator by implementing AIBO ERS-7 and QRIO robots, controlling five of them at full frame rate (30 fps). The simulator is actually in use for the development of our team competing in the RoboCup Four-Legged League.

2 3D Multi-robot Simulator Architecture

In order to use a simulator for multi-robot applications, it is important to provide an effective interface to multiple robot control programs. A typical choice (e.g., Gazebo [3]) is to run the simulator as a server allowing robot control programs to act as clients. The robot control programs receive from the simulator data emulating sensor readings and send commands for the actuators. The simulator server manages interaction among robots and between robots and objects in the environment, maintaining an up to date representation of the world. For a realistic simulation, it is thus important that the simulation is asynchronous with respect to robot control programs. Moreover, the simulator must process data in real-time, otherwise it will feed robot control programs with unrealistic data.

[5] The simulator and demo videos are available from `www.dis.uniroma1.it/~spqr`.

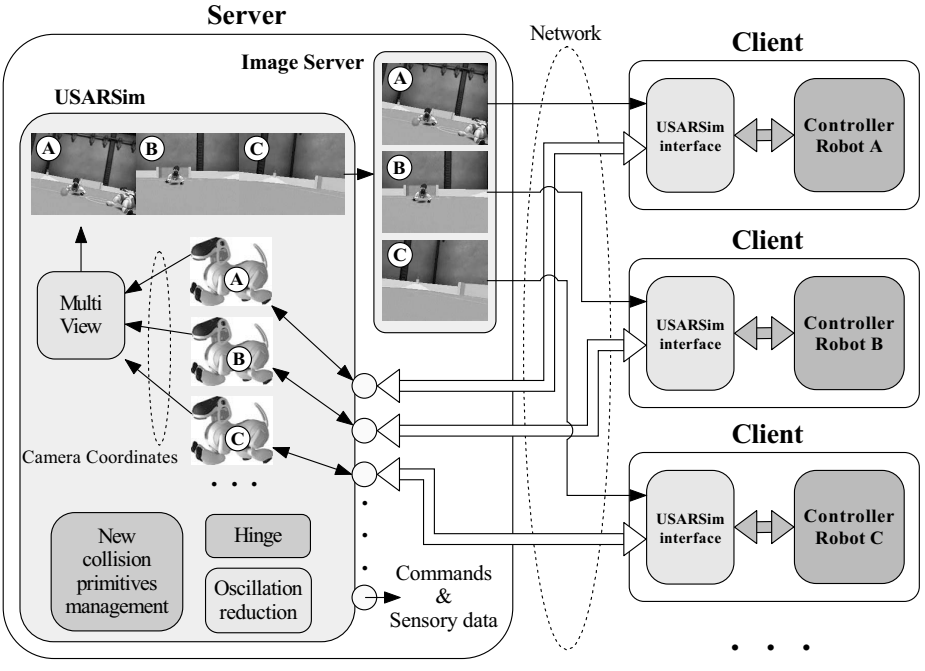

Fig. 1. Multi-Robot architecture

When using a 3D simulator with realistic modeling of appearance and dynamics of the environment, computational time is a main requirement and therefore it is usually necessary to run this process on a different machine with respect to robot control programs. Therefore, a networked client/server architecture should be used for 3D multi-robot simulation. The server machine runs the 3D simulator as a multi-client server. Other machines connected through TCP/IP act as robot control programs.

The choice of USARSim as the basis for the simulator described in this paper has required the implementation of another module, to overcome the problems due to the fact that the underlying game engine is not designed to act as a multi user server on a single machine. In fact, it is possible to use USARSim for multi-robot applications as long as only one robot is provided with a video camera. To simulate a multi-robot system with one camera per robot, it is necessary to run one USARSim process (actually an Unreal client) for each robot. Moreover, it is possible to run only one client per machine. This solution requires too many resources for a single simulation: for example, if robot control programs are separated from simulation we may need up to $2\,n$ machines for an n-robot simulation. Otherwise it is possible to use a single machine switching among robot cameras every t seconds, where t depends on the simulation load on the machine. However, also this solution is not advisable because it offers extremely low image acquisition rates, and it is not possible to access multiple cameras

at the same time. The solution proposed in this paper allows for using a single machine for the simulation in presence of multiple robots with multiple cameras without the aforementioned drawbacks.

The architecture of the multi-robot simulator we have implemented is depicted in Figure 1.

The USARSim interface module on the clients manages all the communications with USARSim. It was created to translate or modify both outgoing commands and incoming sensory data. This interface allows to adapt USARSim to already existing controllers with minimal changes. It also allows to preprocess sensory data if necessary (e.g., apply distortions introduced by real robot cameras).

3 3D Multiple Legged Robot Simulator

In this section, we describe the main modifications and extensions that we implemented in USARSim in order to allow the simulation of legged robots like quadrupeds, bipeds, hexapods and so on. We conclude by presenting an example with two different robots, Sony AIBO ERS-7 and Sony QRIO, operated by independent controllers, coexisting and interacting in the same environment.

Oscillations of Rigid Parts. The first problem we faced was due to the oscillation of all the rigid parts of the robots. The amplitude of the oscillations was $\pm 2mm$ on X, Y and Z axis and it has been observed only when the simulation has been run in networking mode, when server and clients were running on separate machines. The cause was in the replication mechanism, used by Unreal Engine to synchronize server and clients. We fixed this issue in a straightforward way by using in the USARSim code a different data type not subject to network optimizations.

Collision Handling. A collision primitive is an invisible volume with a simple shape (e.g. a box, a cylinder), embedding a 3D mesh and it is used to simplify the collision detection process with other meshes. It is also useful to define dynamic properties like center of mass and inertia tensor. A well-defined collision handling is crucial for a plausible physical simulation and so the correct definition of the collision primitives.

In the original USARSim, each single part of a robot is defined by the class KDpart where, beside other information, it is specified the shape of the robot part and a default collision primitive. When the simulator loads a robot in the virtual scene, the original USARSim assembles all its parts with the corresponding collision primitives on the fly. The process however is prone to a sneaky Unreal Engine bug causing the loading of the same default collision primitive for each mesh. To bypass this bug, our approach has been to write a script, derived from KDpart, for each part of the modeled robots. In this way, meshes and collision primitives have not to be defined at run-time (with a new static-mesh properties) but it is sufficient to specify them in the definition script of the robot.

Because this major change, our implementation lost compatibility with the robots deployed with the original USARSim. To be able to continue to use these

robots, we defined new scripts for each one of them, including the fine-tuning of physical parameters like mass, center of mass, inertia tensor and friction.

Hinge Joint. Original USARSim defines one motorized joint to connect the different parts of the robot, the CarWheelJoint (Fig. 2). Such a joint is provided by Karma physics engine integrated in the Unreal Engine and allows two or three DOFs, depending on its configuration. We did not use this joint because for our objectives, only one DOF was needed. USARSim allows for locking the suspension DOF, leading to a one-DOF joint, however the resulting joint is not stable enough for rigid parts, and it leads to instability of the simulation. Furthermore, with the CarWheelJoint, it is rather difficult to obtain the relative rotation angle among joined parts. Such angles cannot even be set precisely, since there is at least an error of ± 0.5 degrees, and this error drift away (i.e. increase) over time.

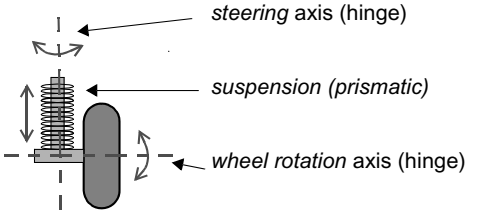

Fig. 2. CarWheelJoint [5]

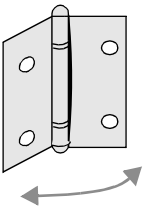

Fig. 3. Hinge Joint [5]

For these reasons, we modified USARSim in order to use another joint model, the Hinge (Fig. 3), also provided by Karma engine and not suffering from the aforementioned issues. Hinge allows one DOF, may be controlled in angle, angular velocity and torque, the maximum allowed precision is 0.0055 degrees, and it implements a feedback mechanism providing a stable control of the angle among joined parts.

Original USARSim allows up to 16 joints for each robot. This bound have been increased by modifying the size of some internal structures in the simulator code.

Multiple Views Support. Unreal Engine allows only one robot camera to be accessed at each frame. If more robots are placed in the virtual environment, only one robot controller may be correctly feed with image data. To allow the simulation of multiple robots, we attempted to interleave images obtained by accessing a different camera at every frame, but this solution is not feasible since simulation and controller modules run asynchronously. The solution provided by the original USARSim is to use the Unreal Engine multi-player support. Each robot runs on a different computer with its graphical client and a central server handles the whole simulation. This solution is obviously not suitable for multi-robot development since it requires too much hardware.

To solve this problem, we introduced a special kind of robot, namely the MultiView. MultiView collects camera locations and orientations from each camera

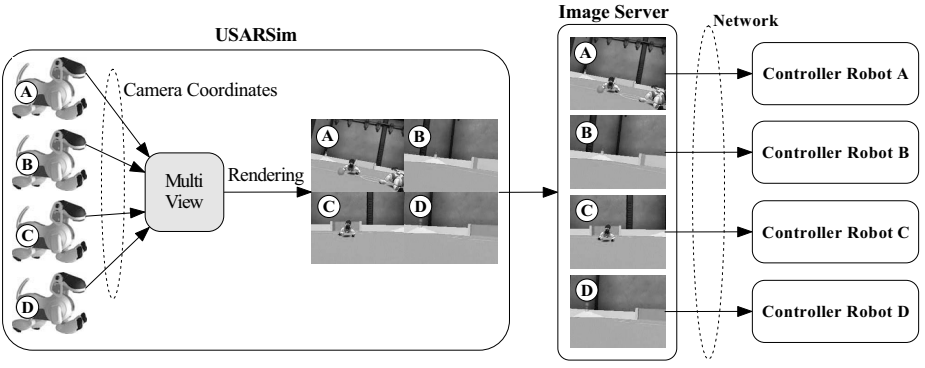

Fig. 4. Each robot camera is rendered by MultiView. Those images are then collected and splitted by ImageServer which sends them to the controllers.

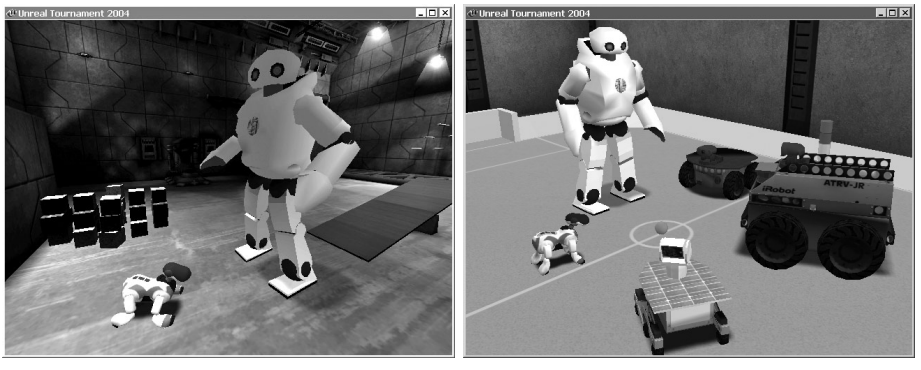

Fig. 5. Multi-Robot example

on the robots and renders each robot view in a different subview. This image mosaic is then grabbed by ImageServer, a thread running on the same machine where the USARSim server resides, by direct access to the Direct3D frame buffer. The views from each robot are extracted from the mosaic and sent across the network to the corresponding controllers (Fig. 4). This solution allowed us to simulate and control multiple robots using cameras by running the simulator on a single machine. The only limitation is in the reduced resolution of the images. Observe that this is not a problem, since simulation can not anyway be used to validate image processing. Moreover, for robots with low-resolution cameras (such as AIBOs), actual resolution can be obtained.

AIBO Sensors. To simulate the AIBO ERS-7 we also added 3 new sensors: a simplistic instant acceleration sensor, a contact sensor and a more flexible IR distant sensor than the one already defined in the original USARSim. In

particular, our IR distant sensor permits to set the maximal bound of the error magnitude in function of the measured distance.

MultiView and the client/server architecture allow to mix easily different robots in the same simulation. Figure 5 shows an example of two different robots, an AIBO and a QRIO humanoid robot, in the same map and handled remotely by different controllers.

4 Run-Time Environment Management

The flexibility of Unreal Engine and Unreal Script allowed us to define the behavior of the virtual environment in real-time.

We define the complex behavior of the objects in the virtual environment through events and triggers. An event is casted when an object, like a robot, comes in contact with a trigger, that is an invisible volume that can be placed anywhere in the map. Each event corresponds to an action like, for example, an affine transformation applied to an object (e.g., open and close a door or a passage), can turn on and off lights or motors. An event can activate users own script routines, permitting endless possibilities. Events can be chained, scheduled, dispatched to many objects or randomly generated.

To manually control objects and robots in the simulation environment, we defined new Unreal Tournament client console commands able, for example, to reset the simulation, to change lighting conditions, to transfer objects and robots from one place to another and so on.

Console commands can also be embedded into the script code defining the behavior of the robots. For example, it would be possible for a robot to modify the simulation laws in order to let it fly from one place to another.

5 Results

This section provides an investigation on physical behavior and the overall performance of our simulator. The testing machine has an AMD Athlon XP Burton 3000+ CPU with 1Gb DDR400 RAM and the nVidia FX 5900XT graphic adapter.

Physical Simulation. A plausible physical behavior is a primary concern if we want to test and simulate algorithms operating on real robots. USARSim uses Karma physics engine which is designed for video-games and not for robotics simulation. This means that the realism of the physical simulation is always sacrificed in order to achieve a smooth rendering frame rate (at least 30 fps). However, the accuracy of the simulation is not severely compromised because the approximations introduced by the Karma engine are comparable to the measurement errors due to the real robot sensor and actuator noise. Thus, it is not crucial to precisely quantify the approximation error, but it is important to qualitatively estimate the behavior of the robots involved in the simulation. Figures 6 and 7 show the results of the two principal interactions that may happen on the RoboCup field: ball kicking and AIBO collisions. We experienced a

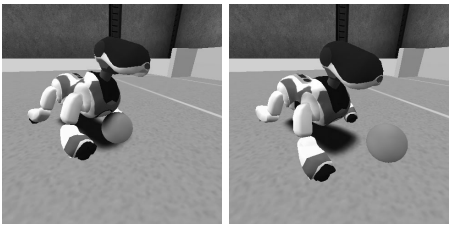

Fig. 6. AIBO kicking the ball

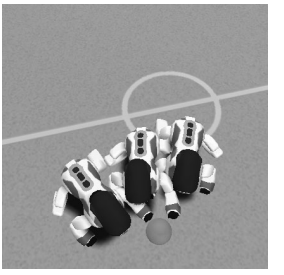

Fig. 7. Three AIBOs collision

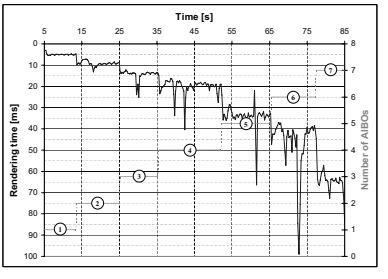

Fig. 8. AIBO creation stress test

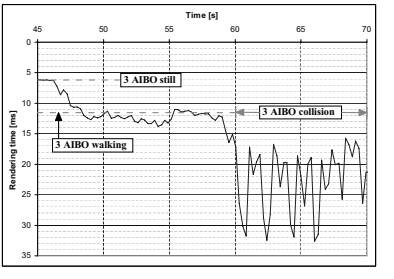

Fig. 9. 3 AIBOs collision

visually convincing behavior of the physical simulation and we can conclude that it is adequate both for a RoboCup simulation and for more general robot-robot and robot-environment interactions.

Performance. The first experiment, made to assess the simulator performance, determines the maximum number of supported AIBOs at the same time in the map using only the test bed PC. Each AIBO is created and set in walking state to make full use of the physics engine. The graph in Fig. 8 shows that the performance is still acceptable with five AIBOs, since the visualization frame rate is still greater than 25 fps.

The second test stresses directly the physics engine making AIBOs to collide with each other. When collisions occur, more than 75% of CPU time is devoted to collision detection and response. The graph in Fig. 9 corresponds to the collision of 3 AIBOs. It is very jagged and each performance drop corresponds to a contact between two or more robots. This result confirms that collision handling is the computation bottleneck.

In conclusion, the simulator can sustain at full frame rate five complex robots, three of which can collide at the same time.

6 Discussion

In this paper we have described the implementation of an extremely flexible simulator for multiple legged-robots. It supports rigid-body dynamics, realistic 3D environments, client/server architecture and real-time rendering. The simulator handles up to five legged robots at 30 frames per second on a middle-class hardware. Furthermore, it is coded in Unreal Script, the scripting language of Unreal Engine. This means that, although Unreal Engine is not open source, our extension is open and will be shared through the community, therefore everyone can access its script code, change its behavior, add new robots, sensors and any other functionality that may be required employing little effort.

As stated before, the choice of using Unreal Engine solves many of the main practical problems faced during the implementation of a robot simulator. However, such an engine has been devised primarily for games, not for robotic simulation. Thus, designers chose to sacrifice physical realism to obtain smoother animation and they bounded the physical time step to the visualization frame rate (i.e., the time step is equal to the frame rate), whereas the simulation and visualization could be clearly separated.

However, the advantages of using an industrial product are in the great development support they provide and in the availability of many effective tools to create contents such as scenarios and objects. For example, the Unreal Editor allows for easy creation of generic environments (Fig. 10), even with scripted objects reacting to changes in the system.

Moreover, improvements to the engine will directly reflect to improvements in the simulator with little effort, while obtaining significant advantages. For example, we intend to upgrade the simulator to use Unreal Engine 3 as soon as it will be available. This will dramatically enhance rendering quality, physics simulation, script and net code.

Applications. The primary use of this simulator is to evaluate the behavior of legged robots in a dynamic environment, such as RoboCup soccer. The advantages of using this simulator are evident in multi-robot contexts. As described above, the simulator is able to simulate in real-time up to 5 robots. We believe that with increasing CPU power it will soon possible to simulate a 4 vs. 4 game.

We have used the simulator to evaluate different situations, such as an attacker robot against three defenders and a goalie, two attacker robots against two defenders and a goalie. A first important process is debugging and refining plans, i.e., evaluating if the robots take the correct decisions according to the current state of the game. Note that a 2D simulator would have some limitations in this process: for example, partial occlusions of the ball, contacts between robots cannot be realistic modeled in 2D. A second process is to evaluate coordination strategies: e.g., position of the robots that are not in possess of the ball, position of the defenders, and decisions about when and how to pass.

In order to make development more effective we have implemented a USARSim interface (as described in Section 2), thus the same control code can be run connected to the simulator or on the real robot. This allows for a fast and effective development of many tasks by the students of our group.

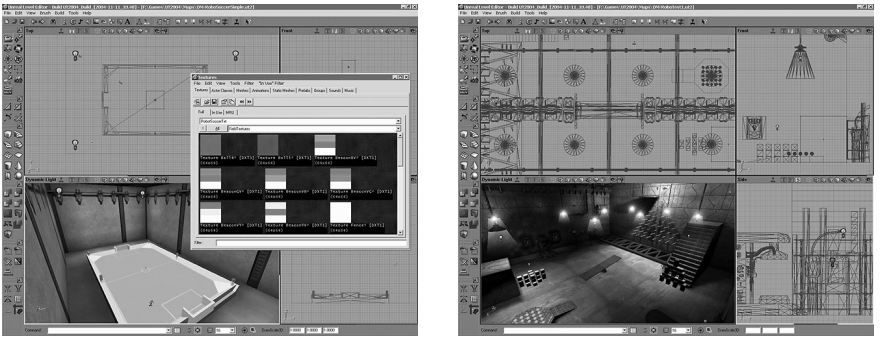

Fig. 10. Unreal Editor used to create RoboCup soccer field and a test arena

Future works. The extensions described in this paper will be integrated in the USARSim simulator. Moreover, we are planning to make some further improvements like enhancing sensor data message handling and making simpler and more rapid the creation of new robots. A further major task will be the modification of the Unreal Tournament deathmatch code in such a way to provide tools to interact with the environment during the simulation run; for example, we intend to implement tools like a game controller interface and a virtual referee placing the robots for the RoboCup soccer setting.

References

1. Carpin, S., Wang, J., Lewis, M., Birk, A., Jacoff, A.: High fidelity tools for rescue robotics: results and perspectives. In: Proc. of RoboCup Symposium (2005)
2. Gerkey, B., Vaughan, R.T., Howard, A.: The Player/Stage Project: Tools for Multi-Robot and Distributed Sensor Systems. In: Proceedings of the 11th International Conference on Advanced Robotics (July 2003)
3. Koenig, N., Howard, A.: Gazebo - 3D Multiple Robot Simulator with Dynamics (2006), `http://playerstage.sourceforge.net/gazebo/gazebo.html`
4. Laue, T., Spiess, K., Röfer, T.: SimRobot - A General Physical Robot Simulator and its Application in RoboCup. In: Proc. of RoboCup Symposium. Universität Bremen (2005)
5. Mathengine: MathEngine Karma User Guide (March 2002)
6. Wang, J., Lewis, M., Gennari, J.: A game engine based simulation of the NIST Urban Search & Rescue arenas. In: Proceedings of the 2003 Winter Simulation Conference (2003)
7. Zagal, J.C., Ruiz-del, J.: Solar. UCHILSIM: A Dynamically and Visually Realistic Simulator for the RoboCup Four Legged League. In: Nardi, D., Riedmiller, M., Sammut, C., Santos-Victor, J. (eds.) RoboCup 2004. LNCS (LNAI), vol. 3276, Springer, Heidelberg (2005)