Proposed Project Topic

Interface for Motion Planning In Physically Realistic 3D Virtual Worlds

Kostas E. Bekris
Computer Science & Engineering Department
University of Nevada, Reno

1 Introduction

There has been great progress over the last few years in the design of physics engines [1]. Physics engines are computer programs that simulate Newtonian physics models, using variables such as mass, velocity, friction and wind resistance. They can simulate and predict effects under different conditions that approximate what happens in real life or in a fantasy world. Their main uses are in computer games and scientific simulation.

In order to be possible to easily construct, however, computer games and simulation environments, it is necessary to have a software tool that allows the effective construction of physically interesting scenes and objects. Such a tool is the equivalent of a map editor for traditional computer games. However, in the case of physically realistic games and simulation environments, such a tool should not just be limited on allowing the placement of objects in desired locations. It should also provide a graphical interface for defining the physical properties of new objects and how they operate in the world, such as their motion primitives and how they can act upon other world objects.

For example, in a racing game it would be interesting if the user can easily define new cars by selecting their modules (e.g., wheels, chassis, etc.), the way they are connected (e.g., two-wheel drive, four-wheel drive, tricycle, innovative configurations etc.) and their physical properties (e.g., friction, torque limits etc.). In games with articulated figures a user can use such a tool for designing new types of agents, such as spiders or legged creatures that move in weird and funny ways and which can change their environment (e.g., carry other objects). Articulated robots and manipulation arms correspond to another example of such moving systems with interesting physical properties.

2 Software Description

This proposal invites you to design a tool for graphically defining new objects that can be physically simulated by existing physics engines. The focus is especially on moving systems and on the physical properties that effect their motion and their interaction with the environment. For such systems, like those mentioned above, it is possible to use motion planning algorithms to automatically compute their path in a simulated world. Your tool should be able to interface with existing motion planning software so as to allow the computation of paths for newly designed physical systems in the simulated world.

2.1 Designing Physically-simulated Systems

Although the above examples of moving physical systems sound quite diverse, ranging from vehicles to legged creatures, they do share many common properties. They correspond to articulated systems compromised by 3D objects linked through joints. The joints may contain motors that allow the systems to move (e.g., the muscles in biological systems also correspond

to motors). The physical properties of the system depend on the physical properties of the 3D objects and the joints.

For example, a simple simulated car can be defined as 4 cylindrical objects (wheels) and 1 cuboid object (chassis) linked via 4 joints. For the objects we must typically define their geometry and their mass. For the joints we must define its type, the physical properties, such as the torque threshold, and where is it attached on the two objects that is linking. Figure 1 provides an illustration of two different types of joints defined in an open-source physics engine, the Open Dynamics Engine [2]. Once the user has defined all these properties, a simulated car has been defined.

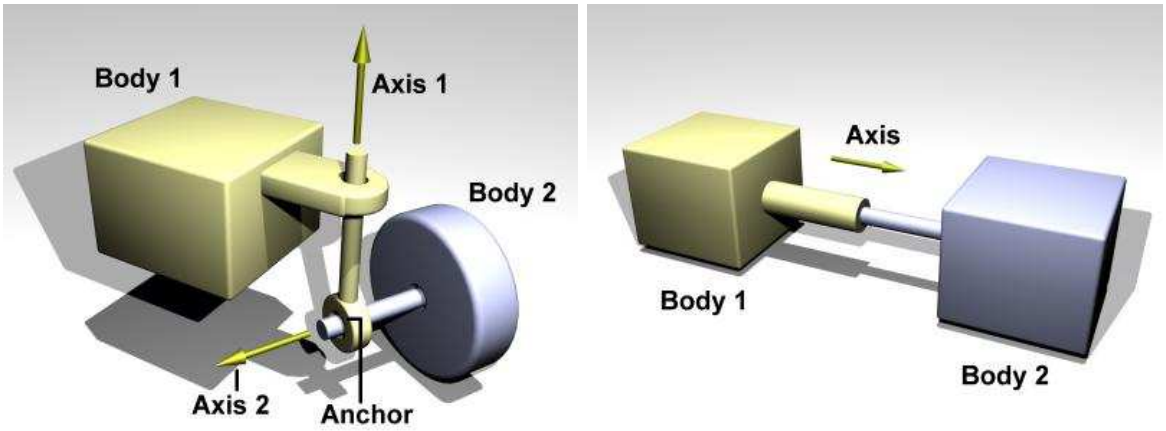

Figure 1: Two examples of joints defined in the Open Dynamics Engine [2]. The first one, called the hinge-2 joint, is typically used to connect the chassis of a car with the front wheels. The second one is a slider joint and often appears in robotic manipulators and certain legged systems.

Today, this process is typically done by coding the parameters of the system and potentially allowing the user to change this parameters manually later. The interface that is the focus of this project proposal should provide an easy way to *graphically* define what kind of 3D objects and joints compromise a new system, as well as the physical parameters of those modules.

2.2 Rendering

Note that once the physical modules (3D objects and joints) of a system have been defined, then each instantiation of a physical system can be rendered in a different way. For example, once a physical model of a car has been defined, then three dimensional meshes and textures can be used to render a car in an interesting way. In this way, you can have a basic physical model for a car and then render one car instantiation as a Ferrari and another one as a VW Beetle! The graphical interface should allow the user to easily select 3D meshes (in standard formats) for rendering the underlying physical 3D objects. It should also allow the storage, loading and alteration of physical models so that they can be easily reused and potentially rendered in a different way.

2.3 Integration with Other Modules

From the above discussion it becomes apparent that the overall end result should contain and/or interact with the following modules:

- A physics engine,

- visualization software (e.g., 3D manipulation of the camera, storing a hierarchical representation of the scene to be rendered - called a scene graph - handling geometric transformations, color, lighting, textures, transparencies, etc.),
- AI software (in particular motion planning algorithms),
- a model database (for storing and loading physical models and 3D meshes),
- and a user interface.

It is important to emphasize that the project does not require the design of a new physics engine or AI software or of visualization software. Instead, the use of existing open source code is recommended and promoted for these modules. For example, the Open Dynamics Engine [2], or the Newton Physics Engine [3] can be used as the underlying physics-based simulator. For motion planning, the OOPSMP software can be used [4]. For visualization, the OpenSceneGraph software [5] and many other alternatives provide the required functionality. Furthermore, for the model database, the simplest possible solution that facilitates the project is sufficient (just storing the models on separate files and potentially constructing an intuitive directory structure).

One important objective for the proposed software is the definition of an effective abstraction so that hooking up alternative physics-engines or motion planning software can be achieved with minimum effort. This means that a user who wants to add a new engine (e.g., like the popular in computer games Havok engine [1]) will have to write a small amount of code so that the proposed software can make use of additional functionality by the new engine (e.g., new types of joints, different types of parameters). This step will require proper software design so as to allow the software to be easily expendable.

2.4 Platforms and Dependencies

A secondary objective for the application is to be cross-platform (e.g., able to run on a PC with Windows or Linux or on a MAC with OS X). To achieve this objective, it might be helpful to use of cross-platform application frameworks (e.g., Qt) or makefile systems (e.g., CMake).

Nevertheless, it is also preferable if the amount of dependencies on other software is minimized. For example, if there is no need for a dependency on a GUI software or a visualization software like OpenSceneGraph but instead the program itself provides all the necessary functionality directly then this would be a strong advantage for the adaptation of the software by others.

3 Final Demo

A possible demo for this project could contain the following steps. The creation from scratch of a new physically simulated car (the definition of the five geometric objects, the four joints and their physical properties). After storing this model, it must be possible to reload it and render multiple different cars using the same physical model. A physical model of a truck can then be defined by altering the properties of the car model. Various tracks should then be rendered in the physically simulated scene. Once all these moving systems are defined, it should be possible to import and position various static obstacles like buildings. Then the user can call the motion planning software to sequentially compute paths for various vehicles between various start and goal configurations. Overall, the final result should look like a traffic simulator for a small city that makes use of physical simulation and autonomous planners. Note that it must be possible to define other types of systems than vehicles as well (e.g., articulated manipulation arms). The extendability property of the software must also be exhibited by showing the amount of code necessary to support a new physics engine.

4 Related Software

A recent plugin for Google Sketchup called SketchyPhysics [6] provides related functionality to the one described in this proposal. SketchyPhysics can provide some good ideas on how to design the interface. There are many YouTube videos on how someone can use SketchyPhysics to design new physically simulated systems. Nevertheless, the objective of this proposal is to design a system that will be integrable with motion planning code, will allow for a more detailed design than the one that SketchyPhysics allows and also provide the capability of easily supporting different physics engines and motion planning software. Finally, SketchyPhysics runs only on Windows.

References

[1] “Havok Physics by Havok.com Inc.” http://www.havok.com.

[2] “Open Dynamics Engine,” http://opende.sourceforge.net/mediawiki-1.6.10/index.php/Main_Page.

[3] “Newton Physics Engine,” http://www.newtondynamics.com/.

[4] “OOPSMP,” http://www.kavrakilab.org/OOPSMP.

[5] “OpenSceneGraph,” http://www.openscenegraph.org/projects/osg.

[6] “SketchyPhysics,” http://code.google.com/p/sketchyphysics2/.