Application of physics engines in virtual worlds

Mark A. Norman*, Tim J. Taylor†

International Centre for Computer Games and Virtual Entertainment (IC-CAVE)
University of Abertay Dundee, Scotland

ABSTRACT

Dynamic virtual worlds potentially can provide a much richer and more enjoyable experience than static ones. To realise such worlds, three approaches are commonly used. The first of these, and still widely applied, involves importing traditional animations from a modelling system such as 3D Studio Max. This approach is therefore limited to predefined animation scripts or combinations/blends thereof. The second approach involves the integration of some specific-purpose simulation code, such as car dynamics, and is thus generally limited to one (class of) application(s). The third approach involves the use of general-purpose physics engines, which promise to enable a range of compelling dynamic virtual worlds and to considerably speed up development. By far the largest market today for real-time simulation is computer games, revenues exceeding those of the movie industry. Traditionally, the simulation is produced by game developers in-house for specific titles. However, "off-the-shelf" middleware physics engines are now available for use in games and related domains. In this paper, we report on our experiences of using middleware physics engines to create a virtual world as an interactive experience, and an advanced scenario where artificial life techniques generate controllers for physically modelled characters.

Keywords: virtual environments, physics engines, real-time

1. BACKGROUND

In the last few years, the emergence of middleware for game development (especially 3D engines for first-person shooter games) has dramatically reduced the cost of creating low-end virtual environments. A typical new consumer PC now has the ability to render millions of triangles per second. More importantly, the bulk of this effort is now done in the graphics card, leaving the CPU free to improve other aspects of the environment such as sound or physics. This newly available processing power, plus the difficulties of co-ordinating large games software projects, means that game developers are now starting to purchase physics engines (e.g. Havok[‡] and MathEngine[§]) as well as 3D engines. VR developers will also be able to exploit this technology.

Physics engines for games are distinct from scientific physics simulators due to their very different goals and constraints:

- The physics engine must run in real time, and it may have only a fraction of the CPU time (<20%).
- The processing load will be variable and unpredictable, especially in multi-player games where many moving objects will be under human control.
- During busy times in the simulation, it is preferable to reduce accuracy rather than performance, as the player is unlikely to notice small variations in physics but will be very sensitive to a drop in frame rate. However, this must be achieved without also causing gross inaccuracies under extreme load.
- Game developers need the ability to tune the engine's behaviour to match players' assumptions about reality.

* mark.norman@abertay.ac.uk; phone 44 1382 308909; fax 44 1382 308345; http://www.iccave.com; IC CAVE, University of Abertay Dundee, Bell Street, Dundee DD1 1HG, United Kingdom.
† tim.taylor@abertay.ac.uk; phone 44 1382 308959. Other contact details as for first author.
‡ http://www.havok.com
§ http://www.mathengine.com

Visualization and Data Analysis 2002, Robert F. Erbacher, Philip C. Chen, Matti Gröhn, Jonathan C. Roberts, Craig M. Wittenbrink, Editors, Proceedings of SPIE Vol. 4665 (2002) © 2002 SPIE · 0277-786X/02/$15.00

The commercial considerations for game engines include:

- Most games use rigid objects with realistic shape and density. Optimising for these objects will be more important than having the flexibility to simulate extreme situations (e.g. gaseous aliens or neutron star paperweights).
- Some common game genres have a well-defined set of typical objects and situations to resolve. If the genre is large enough then a physics engine may be optimised specifically for that market, e.g. the "Optimized Vehicle SDK" inside Havok Hardcore.
- Most game projects begin with the production of a demo to secure a publishing deal. If the API allows the rapid creation of a prototype that will be a selling point.

Over the last year or so, members of our research group have been using physics engines (mostly Havok) in a couple of applications. This paper takes the form of a project report in which we describe our experiences in rapidly developing demonstration systems to real deadlines, and highlight some of the issues learnt from this experience.

The outline of the rest of the paper is as follows. General design and implementation issues for both of the projects are discussed in the next section. The two projects ("Odbods" and "Creatures") are then introduced in the following two sections. For each one, we briefly describe the aims of the project, discuss specific implementation details, present some results, and finally discuss the issues which arose with using a physics engine to implement the project. In the penultimate section, we attempt to list some general issues to be aware of when using physics engines for the development of games and virtual reality applications, in light of our experiences with implementing both of the projects. In the final section we list some general conclusions from our work.

2. GENERAL DESIGN AND IMPLEMENTATION ISSUES

The personnel, software and hardware resources available to implement these two projects are listed below.

2.1. Project personnel

- One research associate with four years' experience in SCADA software engineering and six months in games programming.
- One research associate with one years' experience of using physics engines, and four years' experience with artificial life techniques.
- Two student artists (full-time during holidays, part-time during the Semesters).

2.2. Tools

- Microsoft Visual C++ 6.0.
- 3D Studio MAX® Release 3.
- Microsoft DirectX 8.
- Havok™ Physics Engine. Version 1.2 initially, then 1.3 and finally 1.5.
- Code from a previous project at Abertay University.

2.3. Development hardware

Programming: PC with 700MHz Pentium III, 256MB RAM, nVidia GeForce2 graphics card and Windows 98 SE.
Art #1: PC with twin 500MHz Pentium II, 512MB, GeForce2 and Windows NT4.
Art #2: PC with 700MHZ Pentium III, 512MB, 3DLabs Oxygen GVX1 and Windows NT4.

3. PROJECT 1: ODBODS

3.1. Description of project

The primary goals were to assess if middleware physics was ready for use in games, and to build and populate a virtual environment governed by a physics engine. There were additional requirements that had an influence on the project's design:

- We had an externally imposed goal of producing a technical demonstrator within eight weeks of the start of project (for a local exhibition).
- The developed system should also be capable of simulating the results of the "Creatures" project (Project 2).
- The rendering code was to serve as a test bed for research into illumination methods.

The Odbods application consisted of a number of creatures bouncing and colliding in a virtual environment. The setting was a fairly flat hilltop strewn with rocks. Slopes on either side showed the ability of the creatures to cope with irregular terrain. One creature (the frog) was under human control. The others were given simple AI to interact with the frog and trigger the correct sounds for collisions.

3.2. Implementation

Before starting coding, several commercial and open source 3D engines were assessed, but none of them justified the learning time and/or the cost, given that we could re-use existing code. Few of the free engines had tools for importing animated models from artists' modelling packages. However, the latest release of DirectX contained a plug-in for 3D Studio Max that would achieve 80% of what was required. Most importantly, it included the source code and could be extended to meet the project's needs.

The task list broke down as follows:

- Research and select middleware and APIs.
- Design the game.
- Gather, extend and debug tools. These included 3D Studio Max, the .X file format provided in Microsoft DirectX, and the .X export plug-in for Max.
- Code and debug the game world, renderer and creature AI.
- Learn to use tools and middleware.
- Liaise with student artists.
- Work-in-progress to be demonstrable at all stages.

3.3. Challenges in the project

- Deadlines: The initial project schedule gave about 11 weeks from start to first deliverable.
- Learning curve: During the first 11 weeks, we had to learn the essentials of physics, plugins for 3D Studio Max, animation and AI, as well as code the project and liaise with the student artists.

3.4. Factors in our favour

- Code re-use: Code for sound, input, configuration files and 3D graphics all came from a previous project. After physics, the largest new code modules were for animation and game logic.
- Artistic freedom: The wording of the primary goals left the team with a free hand on what sort of environment would be built.

3.5. The non-feature list

"Feature creep" is a recurring theme in the delay or failure of game development projects. To prevent this, a list of deliberate non-features was created:

- No multitexturing: Only one object (the foreground terrain) truly needed multitexturing. It would have been trivial to hack in a special case, however the code was likely to be re-used for another project afterwards. The cost of this omission is blurring of the terrain. This is painfully obvious in the screenshots but much less so when the camera is following the creatures around, hence the sense of immersion is not greatly diminished by this choice.
- Sound to be plain stereo, not 3D: The target PC's have modern sound cards and will play 3D sound at relatively little performance cost. The rocky environment could be full of echoes. However, physics was the priority.
- No skinned models: Had the project been started six months later, the necessary tools would have been available off-the-shelf. Knowing this, it was better to do without and wait.

In spite of these limitations, the average visitor to IC-CAVE was impressed by the application.

3.6. Results and discussion

At the beginning of the project the models were imported into the scene with minimal changes. They were scaled relative to each other and given sensible masses, but without considering the resulting object densities. Stability improved considerably in the second run once the creatures were shrunk to be on average one metre high. For example, the frog has the height and mass of a young child.

Shadows, though expensive to draw, were important to show that objects were touching, and were useful during development for ensuring that the physical and visual objects were aligned to one another.

The physics objects were extremely simplified versions of the graphical objects. For example, the visual mesh for one of the boulders contained 334 triangles, while its physics equivalent contained only 28. Nevertheless, from the reactions of visitors to IC-CAVE who saw the application, the simulation was meeting expectations. Screenshots from the finished application are shown in Figures 5 and 6.

4. PROJECT 2: CREATURES

4.1. Description of project

Our basic approach to evolving the morphology and controllers of virtual creatures using Havok is essentially similar to that pioneered by Karl Sims[4]. The following paragraphs give a brief overview of the techniques used. We then go on to discuss our experiences, good and bad, of using physics engines for this sort of application. For more details of the genetic algorithm, the genetic representation of the creatures, and the controllers used, the reader is referred to papers by Sims[4] and Taylor & Massey[5].

Each creature is described by a "genome" that contains information about the body shape and controller. Within the genome, the body shape is described by a directed graph, where each node represents an individual body part, and the connections between nodes describe how the parts are connected. Each node also describes an augmented neural network type controller for the corresponding body part. This representation provides modularity to the mapping from genotype (the description of the creature) to phenotype (the instantiation of the creature as a physical model and controller), and naturally leads to features such as duplication and recursion of body parts.

A run is started by randomly generating a population of genotypes. Each genotype in turn is translated into a physical creature, and then evaluated in a physically simulated environment for its performance at a given task. In our work with Havok we have been using a simple underwater environment with a simplistic model of fluid drag. A number of different criteria (or "fitness functions" in genetic algorithm jargon) have been used for scoring the success of each creature in its environment, but they all basically reward creatures for movement. The simplest such function would simply return the distance travelled by a creature's centre of mass during the evaluation period.

The first, randomly generated population of creatures typically performs poorly at the designated task, although a few, by chance, typically have some degree of success. Each creature is scored according to its performance, and when all creatures have been evaluated the population is ranked according to score. The best individuals are kept to form the basis of a new generation. This new population is filled up by adding mutated forms of these best genotypes and genetic crosses of pairs of genotypes (i.e. new genotypes were formed from the combination of parts from two different parent genotypes). This process is repeated over a number of generations.

The end result of the process is the automated production of a 3D virtual creature with autonomous behaviour. The user can dictate what kinds of behaviours evolve via the fitness function. This function is defined at a high level (e.g. the creature should move forward), so the user need not worry, in general, about specifying precise details.

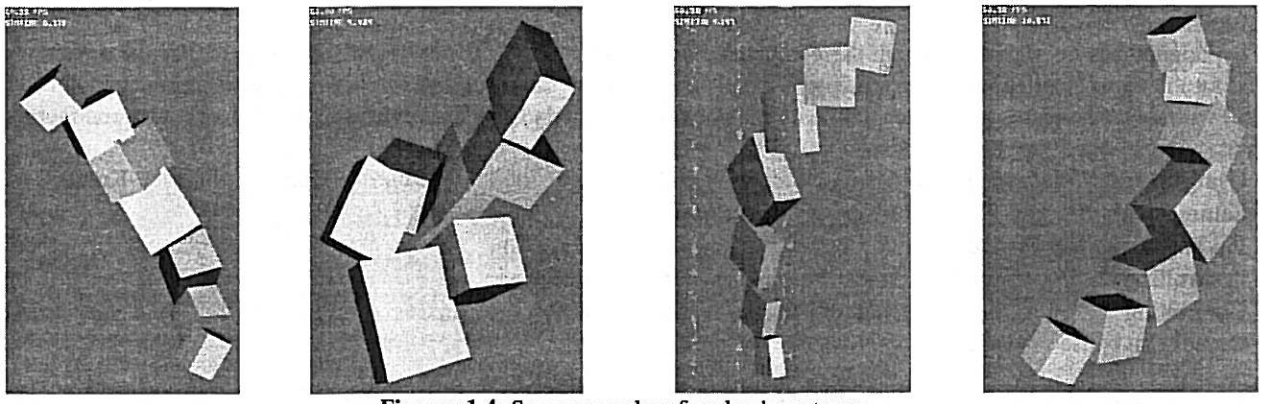
One of the nice properties of systems such as these is that each run generally produces a very different result due to the stochastic nature of the evolutionary process. We only selected for high-level behaviours such as the ability to move

forwards, but within the vast space of different creature designs describable with the genetic system used, there are countless forms that can competently perform such behaviours. The evolutionary process is therefore a tool for exploring interesting regions of this immense landscape of creature designs; it is a creative machine for generating suitable and interesting forms and behaviours, not limited by the preconceptions of a human designer's imagination.

It should be noted that the high level specification of the desired behaviour means that this sort of approach may not be suitable for applications where a very specific kind of behaviour is required. For example, this approach can produce a wide variety of creatures that can locomote on a ground plane, but if one were trying to produce a biped that walked specifically with a realistic human gait, the fitness function would have to be fairly explicit about the kinds of movements that were, and were not, to be rewarded when evaluating the evolving creatures. There are, however, ways in which this approach can be modified to cope better with situations like this. For a start, continuing with the example of evolving a realistic human gait, one would probably not be evolving the creature's morphology at all, but rather just evolving the controller for a fixed, bipedal body shape. The evolution of a realistic human gait could be simplified by specifying appropriate degrees of freedom, joint limits, muscle strengths, etc., for the body within which a controller is to be evolved. Indeed, others have had considerable success using just this kind of approach[3].

4.2. Implementation and results

By repeating this process, efficient creatures evolve reasonably quickly. With a typical population size of 250 creatures, and running the genetic algorithm for 30 generations, a single run takes between 8-12 hours to execute on a single 700MHz Pentium III PC. The precise duration depends on whether creatures are being evaluated once or multiple times in each generation (as explained below), and on how long each evaluation lasts (typically, around 10 seconds of simulated time). In the current work, visualisation of the evolving creatures is active throughout the entire evolutionary process, to enable the user to inspect the current progress of the system. Of course, the run times could be considerably shortened if visualisation was disabled during evolution. Indeed, this was demonstrated on a similar system by one of the authors [TT], using the MathEngine physics engine[5]. Example results are shown in Figures 1-4, 7 and 8.



Figures 1-4: Some examples of evolved creatures.

In the following section, we describe our experiences relating to using a physics engine while developing this application.

4.3. Discussion

One of the great advantages of using a physics package is that it is generally easy to get the basic application up and running rapidly and without detailed knowledge of the specific physics simulation techniques used. Havok, the particular package used in this work, has a well-integrated collision detection package, which further eased the effort required to develop the application.

Although Havok, MathEngine and similar packages are able to simulate a wide variety of situations without problems, there are still various circumstances under which they are likely to produce unrealistic behaviour. Some of these are described in the remainder of this section. Unfortunately, it is in the nature of evolutionary algorithms, such as that used in the present application, that such weaknesses will almost inevitably be encountered. This is because a very wide

variety of systems (the creatures), with very few constraints imposed on their design, are being generated, and simulated, at run-time. (In most other kinds of applications, it would be possible to avoid many of these problems, because the designer would generally be working with a single, well-defined system to be simulated.) A recent review article has tested the stability of the MathEngine, Havok and Ipion engines in a variety of situations[1,2]. Although these products are improving, the current situation is that, no matter which physics engine is used, it is likely that a certain number of stability checks will be required at run-time in any evolutionary system of this kind.

One feature, shared by most rigid body simulators, which has caused us particular problems, is the inability to deal with loops in a creature's morphology. Although the directed graph representation of the body shape does not allow for loops, they may arise during the simulation if, for example, one limb is in collision with another limb on the same creature. The earlier versions of Havok used in this project (v1.2 and v1.3) did not cope well with this situation; forces tended to accumulate around the loop and send the creature spinning off into infinity. However, more recent experience with the latest version of the engine, Havok Hardcore 1.5, suggests that it can cope much better with this type of situation.

In addition to evolving creatures in an underwater environment, we have also tried using a "dry land" environment by removing the fluid drag force and adding gravity and a ground plane. The interaction between the ground plane and an articulated creature with actuated joints caused some problems; sometimes these led to a gain of energy in the system, which eventually crashed the simulation. Adding a small amount of fluid drag improved the situation, and, again, the problem seems to have lessened somewhat with Havok Hardcore 1.5. However, we are still working on producing a reliably stable simulation for this kind of environment.

5. GENERAL RESULTS AND DISCUSSION

5.1. Parameter tweaking

Our experience has been that, although these engines work well for a wide variety of situations, in reality the developer really does need to have some understanding of the algorithms being used, in order to tweak appropriate parameters and to deal with degenerate cases. The resolution of such problems usually involves finding a good balance between factors such as:

- Using an appropriate integrator (e.g. Havok comes with a variety, including Euler, Midpoint and Runge-Kutta).
- Setting an appropriate integration step of the integrator.
- Finding appropriate values for the parameters of the constraint system.
- Controlling the maximum forces and torques exerted within the system.
- Controlling the range of masses of the bodies in the simulation (i.e. mixing very large masses and very small masses can cause problems).

5.2. Indeterminism

Another common problem we encountered with these packages is that the simulations are indeterministic. In other words, if an identical simulation is run multiple times, each one may behave slightly differently. The longer the simulation and the more interactions that occur, the worse this problem becomes. Unfortunately, the latest versions of the Havok and MathEngine APIs do not allow the programmer to set the seed for the random number generator used by the engines, so there is no way to overcome this problem. Even if this were possible, it is likely that an identical simulation run on different hardware platforms would produce different results. Of course, in some applications a small degree of indeterminism is unlikely to cause major problems, but in others, it might be more serious.

5.3. Realism

A physically simulated world will be closer to reality than traditional virtual worlds, but with current technology is still not a perfect match. For performance reasons the physical models are usually simpler than reality. On the bright side, you have freedom to deviate from the real world where this suits your game. You can selectively disable collision detection, use simpler friction, etc. - all very good for rapid prototyping.

5.4. Performance

The number of objects for the Odbods simulation is limited to less than 50 of medium complexity. Only a small number of creatures can be animated in real-time (typically around 5 creatures each consisting of 10 rigid bodies). It is also necessary to try and avoid extended sequences of collisions, unless you can tolerate the temporary performance drop that entails.

5.5. Technology not yet mature

Havok has undergone significant evolution and improvement between versions 1.2 and 1.5. However, even Havok 1.5 does not offer stability out of the box. To be fair, Havok 1.5 includes functions that will allow you to detect when the simulation has broken and take steps to recover to a valid state.

6. CONCLUSIONS

We have developed two kinds of applications based on general-purpose physics engines, an interactive experience and evolved creatures. For our two applications, which took around six man-months each, general-purpose engines perform well, though it took longer than expected to adjust the systems for stable simulation. We found that the physics engine increased our productivity, however they are not a substitute for expertise.

REFERENCES

1. J. Lander and C. Hecker, "Physics Engines, Part One: The Stress Tests", *Game Developer* **7** (no.9), pp. 15-20, 2000. (Available online at http://www.gdmag.com)
2. J. Lander and C. Hecker, "Physics Engines, Part Two: The Rest of the Story", *Game Developer* **7** (no.10), pp. 13-18, 2000. (Available online at http://www.gdmag.com)
3. T. Reil and C. Massey, "Biologically Inspired Control of Physically Simulated Bipeds", *Theory in Biosciences* **120**, pp. 1-13, 2001.
4. K. Sims, "Evolving Virtual Creatures", *Computer Graphics (SIGGRAPH 94 Proceedings)*, Andrew Glassner (editor), pp.15-22, ACM SIGGRAPH, New York, 1994.
5. T. Taylor and C. Massey, "Recent Developments in the Evolution of Morphologies and Controllers for Physically Simulated Creatures", *Artificial Life* **7** (no.1), pp. 77-87, 2001.

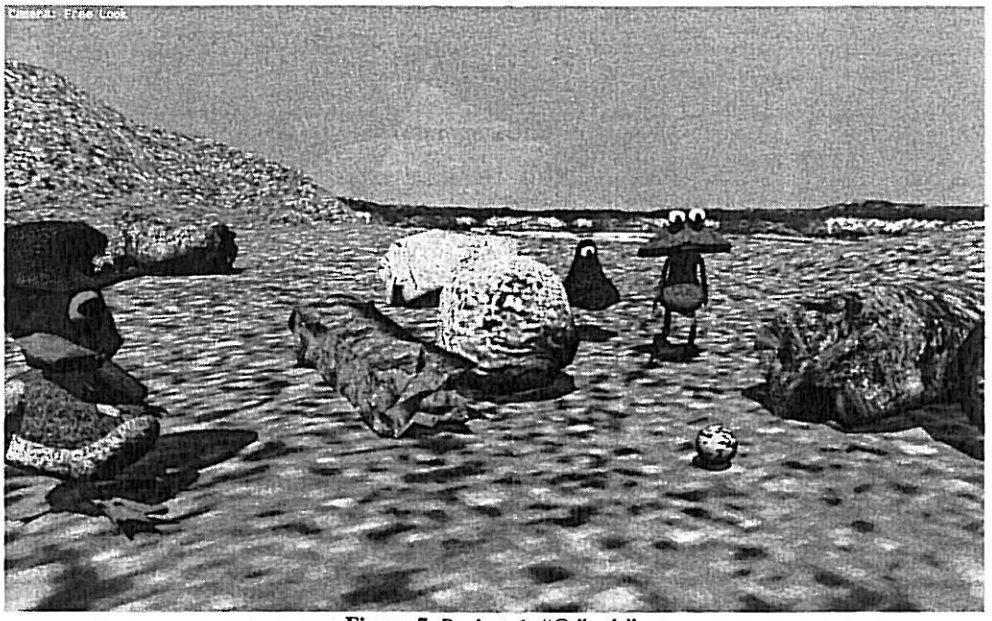


Figure 5: Project 1: "Odbods"

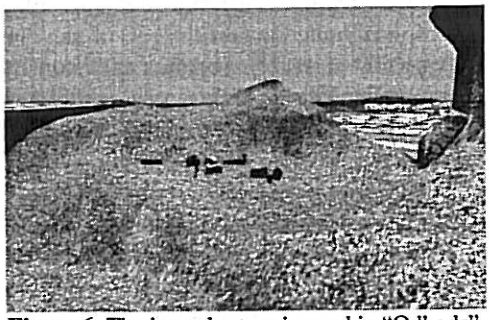


Figure 6: The irregular terrain used in "Odbods"

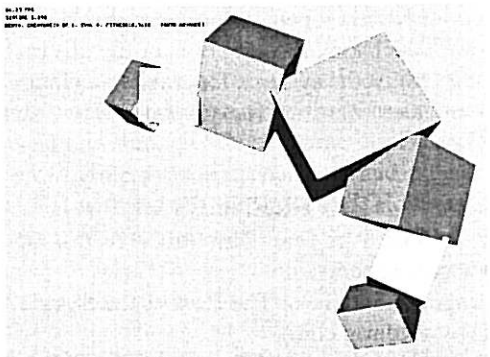


Figure 7: Project 2: "Creatures" – creature evolution version

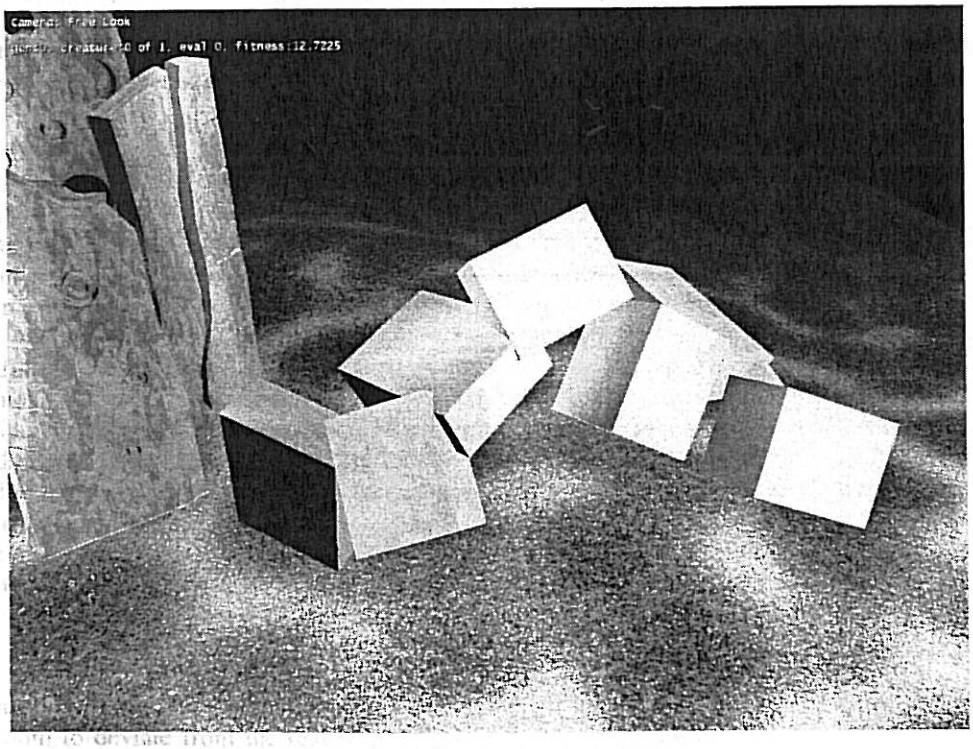


Figure 8: "Creatures" – presentation version