S. Marks, J. Windsor, B. Wünsche, 'Collaborative Soft Object Manipulation for Game Engine-Based Virtual Reality Surgery Simulators', *Proceedings of Image and Vision Computing New Zealand 2007*, pp. 205–210, Hamilton, New Zealand, December 2007.

Collaborative Soft Object Manipulation for Game Engine-Based Virtual Reality Surgery Simulators

Stefan Marks
Email: dev.stefan.marks@gmx.net
Division for Biomedical Imaging and Visualization
Department of Computer Science
University of Auckland, New Zealand

John Windsor
Email: j.windsor@auckland.ac.nz
Advanced Clinical Skills Center
Department of Surgery
Faculty of Medical and Health Sciences
University of Auckland, New Zealand

Burkhard Wünsche
Email: burkhard@cs.auckland.ac.nz
Division for Biomedical Imaging and Visualization
Department of Computer Science
University of Auckland, New Zealand

Abstract

In this paper we analyse and evaluate the capabilities of popular game engines to simulate and interact with soft objects. We discuss how these engines can be used for simulated surgical training applications, determine their shortcomings and make suggestions how game engines can be extended to make them more suitable for such applications.

Keywords: game engines, surgical simulation, collaboration, deformation

1 Introduction

Training tools using virtual reality (VR) simulations are becoming increasingly important in healthcare, in particular for applications which involve complex procedures and tasks, such as surgical training. The drivers for this include the improvement of quality in medical care and training, the need to reduce errors and costs [1], the requirement of quality standards and assessment methods for the performance of medical staff, and the desire to increase patient safety.

Most commercially available simulators focus on training individuals on a limited range of surgical procedures. This restriction is a major drawback since cooperation is considered an important dimension of simulations [2]. Only very expensive simulators (mainly mannequins) are capable of addressing this dimension.

The second drawback of commercially available simulators is their high price, mainly due to the high-end hardware and specialised input and output devices (e.g. for force feedback). Only a limited number of training centers have the means to obtain simulators, so that surgeons and interns have to travel to the closest centre to get training.

A third drawback is the constant "reinvention of the wheel." All simulators require the basic components depicted in figure 1, which in general are developed independently by each vendor.

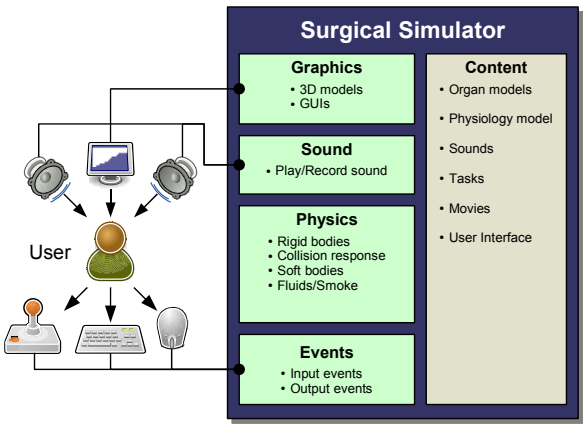

Figure 1: Functional components of a surgical simulator.

There have been attempts to create extendable frameworks for building surgical simulators (e.g. SPRING [3], GiPSi [4], SOFA [5], [6]). They all incorporate the above mentioned components and a variety of mathematical models for the physical simulation and interaction. But except for SPRING (ironically the oldest project in the list) they all lack the capability of networking with

other simulators in order to develop collaborative scenarios. Furthermore, sound support is not built into one of them.

Modern game engines are structured similar to surgical simulators (see figure 2) and also incorporate a networking and sound component. Game engines are well tested, robust and usually available at low costs. This paper evaluates the suitability of game engines for handling of soft objects in collaborative scenarios, an essential aspect of surgical simulation and training applications.

2 Game Engines

The use of games or game engines for medical education is a little explored research subject with many areas still undiscovered. One reason for this might be the incoherency of the seriousness of medicine and the playful, sometimes violent character of computer games. Nevertheless, game engines offer a vast pool of useful concepts and resources in technical as well as in educational aspects.

Projects like the "Serious Game Initiative" [7] focus on offering help to "organize and accelerate the adoption of computer games for a variety of challenges facing the world today." A subproject founded by this initiative is "Games for Health" [8], focusing mainly on games used in various health care sectors.

Previous authors have so far concentrated on applications where the game content was about learning facts, rather than tasks and procedures. For example, Wünsche et al. [9] have examined how game engines can be used for visualisation of medical datasets, and Mackenzie et al. [10] utilise a game engine for anatomical education. However, so far nobody has tried to simulate complex and cooperative procedures, involving the interactive manipulation of soft objects.

2.1 Design

A game engine is a complex software system necessary for developing and playing games. Two different games with the same underlying engine differ by the *game content*, i.e. graphics, sounds, storyline. Game engines build a bridge between this content and the underlying hardware. With the help of an operating system abstraction layer, the same game content can be run on many platforms (e.g. Windows, Linux, XBox) without change.

Modern game engines consist of all or a subset of functional blocks depicted in figure 2.

The *Graphics Engine* loads, displays, manipulates and manages all the data related to graphical content and visual effects. 3D models of landscapes, buildings, players and other characters and objects can be loaded, textured, lit, and animated.

All audible content like sound effects, ambient noise, and music, but also physical sound phenomena like occlusion or Doppler effect, is handled by the *Audio Engine*.

The *Physics Engine* implements advanced mathematical models for calculating rigid body simulations of arbitrarily shaped and articulated objects (e.g. vehicles, machines).

Artificial intelligence, provided by the *AI Engine*, is needed for enabling Non-Player Characters (NPC), the computer controlled antagonists or team members, to navigate, make decisions, and react to their environment.

The *Networking* component allows multiple players to interact over a network. This is achieved by sending the game state and actions of all players and NPCs over the network, such that each connected client sees the same instance of the virtual world.

The great flexibility of a game engine is achieved by *Scripting* languages that allow an immediate access to the functions of the engine.

The remaining parts of a game engine manage the memory and the efficient scheduling of processes (*Memory/Process Management*), receive the user input and other events, e.g from keyboards, mice, joysticks (*Event handling*), or can load media like music or videos (*Streaming*).

2.2 Advantages

Modern game engines utilise the latest graphics hardware while still being compatible with older technologies. This makes game engines capable of handling the same game content on hardware with different speed, memory and features.

Playing computer games is no longer an action for single players but has evolved into multiplayer gaming, bringing together several thousand players at the same time [11]. Consequently, many game engines offer carefully designed and tested network support that enables users at different physical locations anywhere in the world to cooperatively accomplish tasks. Built-in support for recording and playing sound over the network enables the players to communicate in a natural way to coordinate their actions. Textual input of messages serves as an alternative way.

The stability and reliability of game engines is secured by customers and developers worldwide,

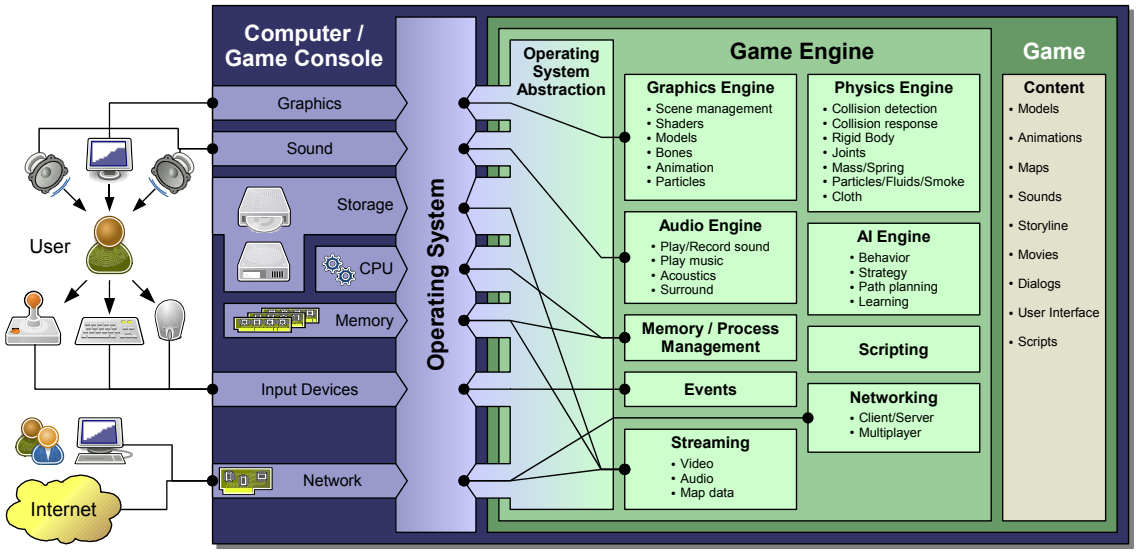

Figure 2: Functional blocks of a game engine.

who give feedback about errors and flaws. Fixes and patches for those errors are then developed, and returned to the customers.

All of these advantages are useful for developing biomedical and scientific simulations, which run on consumer level hardware.

The networking support enables the construction of collaborative scenarios and provides verbal and textual communication between participants. This can be used by a supervising trainer to observe an ongoing surgery simulation, to provide spoken or written feedback, to interact with participants and to change parameters of the simulation in accordance with the skills of the participants.

The large user base of game engines ensures that the underlying technology is regularly updated and increases the number of potential users of the simulation, which helps with making the application more stable. By allowing users to contribute new models and scenarios, the quality and diversity of simulations can be improved.

3 Methodology

3.1 Engine Selection

We started our selection of suitable game engines with an evaluation of an internet game engine database [12]. At the time of this evaluation (July 2007), this database contained 278 engines. We disregarded engines still in an early development state and those that were not developed or maintained any more. Engines without sound or other essential components were also removed from the list. Of the remaining engines, we selected those with in-built means of creating new game environments (maps). This requirement reduces the complexity of the editing process as with it there is no need for purchasing, installing and setting up external editors and necessary conversion tools, assumed the latter exist at all.

After the reduction of the original list by this selection process, we chose from the remaining engines those which were inexpensive and in our opinion most popular and widely distributed:

- Unreal Engine 2 [13]
- id Tech 4 [14]
- Source Engine [15]

3.2 Evaluation

A first step in our evaluation process is the creation of a simple multiplayer map with isolated physical objects. As a limited subset of surgical actions, the objects are pushed and pulled by the users. This demonstrates the ability of the engine for exact replication of the state of objects on the game server and the other connected users (clients). Failing this test would restrict the engine to playback of predefined animations triggered by the users. This would drastically decrease the degree of interactivity in cooperative scenarios.

In a second step, we connect several physical objects by constraints (joints) and evaluate the stability of the simulation when multiple users push and pull this construct. This step demonstrates the ability and stability of the physics engine in resolving constraints, a prerequisite for more complicated physical constructs, like mass-spring systems, the simplest representation of soft objects.

A third step is the evaluation of the extendability of the game engine itself by means of scripting or programming, called "modding." This facilitates the implementation of features which a game engine might lack, such as soft tissue simulation and new interaction tools.

4 Results

4.1 Unreal Engine 2

The Unreal Engine 2, used, e.g., by the game "Unreal Tournament 2004" (see figure 3), is capable of simulating physical objects connected by joints, but only in single player mode. It cannot duplicate the state of moving physical objects over the network.

We created a map including simple physical objects like a shelve and a table. In addition, we modified a freely available skeleton model[1] by adding joints to the arms, the legs and the skull.

When this skeleton is moved by the user on the server, the user on the client will not see this movement. In contrast the physical modelling of the skeleton and the collision detection with it are handled correctly on the server. The client player sees the skeleton still lying on the table, but is lifted up when walking over the skeleton, which in reality lies on the ground as correctly seen by the server user.

This asynchronism of the physics engine is not considered an error, as in 2003, the time of the release of the game, the physical simulation of game objects was not yet an important aspect of gameplay. Nevertheless, users wanted to create multiplayer maps with synchronised physical objects and thus developed a modification of the physics engine [16] written in the engine's scripting language UnrealScript. Due to the age of the Unreal Engine 2, this project has undergone no further improvement since 2005 and is now no longer available on servers.

[1]Skeleton model source: `http://artist-3d.com/free_3d_models/dnm/model_disp.php?uid=637`

Figure 3: Multiplayer test map for the Unreal Engine 2. The skeleton shows up at different locations on the server (left) and the client (right) when being moved in multiplayer mode.

4.2 id Tech 4

The id Tech 4 engine, represented by the game "Quake 4" was not able to fluently display physical object movements on the connected client. Objects moved with more than 25 frames per second on the server, but only with about 5 frames per second on the client. In rare cases, the same location update problems as for the Unreal Engine 2 occurred.

In addition, it was not possible to connect physical objects by joints, because the map editor did not implement the adding of joint constraints in the map. This lack was unexpected, as articulated structures like cranes with swinging load can be seen in the game "Doom 3," which also uses the id Tech 4 engine.

It is possible to access features of the game engine by scripting, but only to a limited extent. Adding a more sophisticated physics engine is not possible since the low level functionality of the engine is inaccessible.

4.3 Source Engine

With the purchase of a game of the "Half-Life 2" series, the user is authorised to download the SDK (Software Development Kit) for the Source Engine. This SDK enables the construction of new maps and even modification of the source code of the engine. The engine itself is constructed in a modular manner, so that new features can easily be added, e.g. the extension of the render engine for automatic brightness adaption in dark or bright scenes (High Dynamic Range), which was released with the expansion mission pack *Half-Life 2:Lost Coast*.

The games based on the Source Engine 2 are at the time of writing unique with respect to the use of physical objects for the completion of game objectives (e.g. building ramps, moving levers, building traps). The engine's advanced physics model is reflected in its ability to fluently simulate objects in our test map on the server and the client.

For the evaluation, we created a test map with a mass-spring system consisting of 4×4 partially intersecting spheres connected in a rectangular pattern by 24 springs (see figure 4). This physical

Figure 4: Cooperative manipulation of the mass-spring system.

construct can be grabbed by two or more users and being pushed and pulled. The simulation of the mass spring system was stable and well synchronised on server and client. With the custom texture and the shiny surface properties, it resembled the handling of a colon or a similar structure. For a more realistic appearance, the mass spring system should deform a single geometrical object, but this is not possible without extending the engine. Nevertheless, this extension is possible, since the included SDK allows the modification of the source code of the engine.

We also created a map with a heart model whose vessels are deformable by means of "skeletal animation" (see figure 5). To achieve this, the heart model is extended by a skeleton structure with bones positioned along the centres of the main vessels. As a result, movement of the skeletal bones deforms the mesh of the vessels. To achieve softer bending and thus a more natural appearance, the influence of the bones on the mesh linearly varies from 0.0 (no influence, blue colour in figure 5) at the beginning of the vessel to 1.0 (full influence, red colour) at the end.

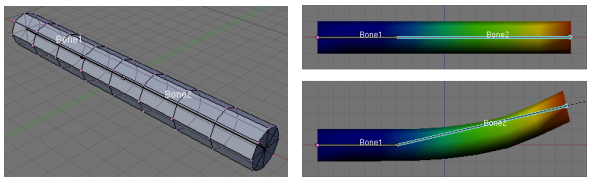

Figure 5: Soft deformation of a tube mesh by skeletal animation.

Figure 6 shows how the user can grab a tool and interactively deform the heart vessels with it. At the same time, other users are able to observe this action and to cooperatively interact with the same model.

Figure 6: Manipulation of a deformable heart model seen by the acting user (top) and the observing user (bottom).

5 Conclusion

All three evaluated engines (see results in table 1) allow the playback of animations and are hence suitable for implementing simple simulations based on prerecorded animations. The Unreal Engine 2 also allows the manipulation of articulated physical objects for a single user and is thereby suitable for standalone simulators. However, full collaborative interaction with physical objects is only possible with the Source Engine. Its physics engine allows the simulation of soft objects by skeletal animation or with mass-spring systems. In addition, the SDK also enables the extension of the engine with new features.

Since in current games soft objects are not important for the gameplay and are thus not implemented in the physics engines, this extensibility is a major selection criterion when using game engines for such simulations.

Development times for biomedical and scientific simulations can be reduced by using game engines that include editors for building custom maps. Without such editors, the integration of custom content is difficult and time consuming. Engines that allow users to create their own content also are more likely to be stable and well designed, as with increasing exploration of their capabilities in user created maps many bugs, errors and design flaws will eventually be reported and corrected.

6 Future Work

While the mesh-spring system implemented in the Source Engine allows the simulation soft objects, the process of manually inserting and editing the mesh is time consuming. We plan to investigate the integration of new mathematical models for soft objects, e.g. [17, 18, 19].

Towards the end of 2007, new game engines (CryEngine 2 [20], Unreal Engine 3 [21]) will be released. Their physics engines will be more flexible and offer more features, such as the simulation of cloth, fluids and smoke. This new functionality might also be useful for simulating deformable objects.

References

[1] L. T. Kohn, J. M. Corrigan, and M. S. Donaldson, Eds., *To Err is Human: Building a Safer Health System*. Washington, DC, USA: National Academy Press, Nov. 2000. [Online]. Available: http://www.nap.edu/catalog.php?record_id=9728

Feature	Unreal Engine 2	id Tech 4	Source Engine
Predefined animation	+	+	+
Single physical objects, single player	+	+	+
Single physical objects, multiplayer	–	– (5 fps)	+
Connected physical objects, single player	+	– (no constraints available)	+
Connected physical objects, multiplayer	–	– (no constraints available)	+
Extensibility	UnrealScript	limited	Source code

Table 1: Results of the evaluation of game engines.

[2] D. M. Gaba, "The future vision of simulation in health care," *Quality and Safety in Health Care*, vol. 13, no. Suppl 1, pp. i2–i10, Oct. 2004.

[3] K. Montgomery, C. Bruyns, J. Brown, S. Sorkin, F. Mazzella, G. Thonier, A. Tellier, B. Lerman, and A. Menon, "Spring: A General Framework for Collaborative, Real-time Surgical Simulation," *Studies in Health Technology and Informatics*, vol. 85, pp. 296–303, 2002.

[4] M. C. Çavuşoğlu, T. G. Göktekin, and F. Tendick, "GiPSi: A Framework for Open Source/Open Architecture Software Development for Organ Level Surgical Simulation," *IEEE Transactions on Information Technology in Biomedicine*, vol. 10, no. 2, pp. 312–322, Apr. 2006.

[5] J. Allard, S. Cotin, F. Faure, P.-J. Bensoussan, F. Poyer, C. Duriez, H. Delingette, and L. Grisoni, "SOFA – an Open Source Framework for Medical Simulation," in *Medicine Meets Virtual Reality (MMVR 15)*, Long Beach, USA, February 2007.

[6] S. Tuchschmid, M. Grassi, D. Bachofen, P. Früh, M. Thaler, G. Székely, and M. Harders, "A Flexible Framework for Highly-Modular Surgical Simulation Systems," in *Biomedical Simulation: Third International Symposium, ISBMS 2006, Zurich, Switzerland, July 10-11, 2006*, ser. Lecture Notes in Computer Science, vol. 4072. Heidelberg: Springer Berlin, Jul. 2006, pp. 84–92.

[7] Serious Games Initiative. (2007) Serious Games Initiative. [Online]. Available: http://www.seriousgames.org

[8] Serious Games Initiative. (2007) Games For Health. [Online]. Available: http://www.gamesforhealth.org

[9] B. C. Wünsche, B. Kot, A. Gits, R. Amor, J. Hosking, and J. Grundy, "A Framework for Game Engine Based Visualisations," in *Proceedings of Image and Vision Computing New Zealand 2005*, Nov. 2005. [Online]. Available: http://www.cs.auckland.ac.nz/~burkhard/Publications/IVCNZ05_WuenscheKotEtAl.pdf

[10] J. Mackenzie, G. Baily, M. Nitsche, and J. Rashbass, "Gaming Technologies for Anatomy Education," Online, May 2003. [Online]. Available: http://www.virtools.com/news/pdf/2004/CARET.pdf

[11] K.-T. Chen, P. Huang, and C.-L. Lei, "Game traffic analysis: An MMORPG perspective," *Computer Networks*, vol. 50, no. 16, pp. 3002–3023, Nov. 2006.

[12] DevMaster.net. (2007) 3D Game Engines Database. [Online]. Available: http://www.devmaster.net/engines/

[13] Epic Games. (2004) Unreal Engine 2. [Online]. Available: http://www.unrealtechnology.com/html/technology/ue2.shtml

[14] Wikipedia. (2007) id Tech 4 — Wikipedia, The Free Encyclopedia. [Online]. Available: http://en.wikipedia.org/wiki/Doom_3_engine

[15] Valve Corporation. (2004) Valve Source Engine Features. [Online]. Available: http://www.valvesoftware.com/sourcelicense/enginefeatures.htm

[16] J. Zepp. (2005) GoodKarma Physics Mod Beta 4. [Online]. Available: http://www.ataricommunity.com/forums/showthread.php?t=440477

[17] A. Henriques, B. Wünsche, and S. Marks, "An investigation of meshless deformation for fast soft tissue simulation in virtual surgery applications," *International Journal of Computer Assisted Radiology and Surgery*, vol. 2, no. Suppl 1, pp. S169–S171, June 2007.

[18] M. Müller, B. Heidelberger, M. Teschner, and M. Gross, "Meshless deformations based on shape matching," *ACM Transactions on Graphics*, vol. 24, no. 3, pp. 471–478, Jul. 2005.

[19] A. R. Rivers and D. L. James, "FastLSM: Fast Lattice Shape Matching for Robust Real-Time Deformation," in *SIGGRAPH '07: ACM SIGGRAPH 2007 papers*, vol. 26, no. 3. New York, NY, USA: ACM Press, Jul. 2007, p. 82.

[20] Crytek. (2002) CryEngine 2 Specifications. [Online]. Available: http://www.crytek.com/technology/index.php?sx=eng2

[21] Epic Games. (2006) Unreal Engine 3. [Online]. Available: http://www.unrealtechnology.com/html/technology/ue30.shtml