Sense Hardware Backup Algorism of 3D Game Engine

Henrik Aamisepp, Daniel Nilsson
Department of Computer Science, Lund Institute of Technology, Canada

Abstract

The aim this master thesis is to find out if is possible to Integrate haptic hardware support in the source 3D game engine Crystal Space. Integrating haptic support would make it possible to get a haptic representation of 3D geometry in Crystal Space and therefore take advantage of all the benefits a 3D game engine provides, when building haptic applications, An implementation of the support should be as low-cost as Possible by taking advantage of available source haptic API alternatives.

The thesis report presents an evaluation of available haptic APIs and comes up with a design and an implementation. The solution has been implemented as a Crystal Space plugin by using modified parts of the e-Touch open module API. The plugin makes it possible to utilize the Phantom haptic device to touch and feel the 3D environments in a Crystal Space application. Two demo applications have also been constructed to show the capabilities of the plugin.

1. Introduction

At the Department of Design Sciences at Lund of Technology there is a division for rehabilitation engineering called Certec. Among other things they do Research on how to make computers accessible to blind and visually impaired people through the use of haptic technology. This means that they try to make it possible to use computers by using the sense of touch to navigate through a 3D interface. The force feedback hardware that is available in stores today cannot provide the accuracy that is needed for such a task. However there is available hardware that can be used, such as the Phantom haptic device from Sensable Technologies, Inc. The problem is that this hardware is very expensive and is therefore not available to a great extent. This in turn means that there are only a few APIs available for creating haptic environments and they do not always provide the same possibilities to easily create good 3D applications as for example a game engine would. The APIs are not designed to be integrated with other systems that can provide this functionality either.

The wish is that it should be possible to build large 3D environments that the user can navigate in and use the haptic device to touch the surroundings. There kinds of environments are very common in modern 3D computer game and thus it would be desirable to have support for haptic devices in a 3D game engine. This would make it possible to create 3D environments using existing 3D map editors, or perhaps even to use existing 3D game maps that can be downloaded form the Internet.

This master thesis is a collaboration between Certec and the Department of Computer Science at Lund Institute of Technology. The idea is to add support for haptics in the open source game engine Crystal Space, by using parts of available haptic APIs, such as e – Touch from Novint Technologies, Inc. or GHOST from Sensable Technologies Inc. It is required that the solution should support the Phantom haptic device, but it would be advantage if there also was support for other force feedback devices.

2. Haptics

2.1 Introduction to haptics

Haptics originates from the Greek word haptesthai ("to touch") and is a science that studies the sense of touch. In the computer world, haptics deals with using the sense of touch to control and interact with a computer application. To be able to do this, special input/ output devices, such as joysticks, wheels, data gloves or more advanced devices are connected to the computer.

The feedback from these devices is delivered as felt sensations to the user's hand or other parts of his body. The user can then interact with this feedback and

control the application according to the sensations he experiences. By using a haptic device, three dimensional virtual objects do not only have to exist as graphics, but they can also be represented in haptics as physical objects. This opens up a lot of possibilities for many different kinds of applications. For example, haptics have been utilized to train people in surgery and operation of machines in hostile environments. Additional examples of the use of haptics include the entertainment business, where haptics has been used to give the player force feedback of the events occurring in a computer game. Haptics can also help to guide blind and visually impaired people in computer applications.

2.2 History of haptics

One may think that the development of haptic devices and the applications adherent to them originates from the development of virtual reality, but this is actually incorrect. The first devices that one may state as haptic devices come from the teleoperation systems of the 1950s and 60s . In a teleoperation system, the user controls and manipulates objects from a remote location. These systems often use bilateral Master-Slave Manipulators (MSM), in which a master device follows and interacts with a remote slave device. A Class example of the field of application for MSM is within dangerous or unhealthy environments, like the handling of unclear waste or underwater operations. In these cases some type of a mechanical arm is used as the master unit and a smaller similar reproduction of this used as the slave device. These systems used to be all mechanical, but they were later made electrical which enabled the operator to work further away from the site. It also allowed the introduction of so called servomanipulators, which could give force feedback to the slave device.

3. Evaluation of haptic APIs

In this chapter an evaluation of GHOST and e- Touch will be made and a conclusion will be drawn whether e-Touch API, GHOST API or another solution will be used. Reachin API will not be evaluated the decision has been taken that its software license costs too much to be used in this project.

3.1 GHOST

In this project we have used GHOST version 3.1. However, version 4.0 of GHOST has recently been released. The new version does not provide any speed improvements, but it does provide a new gstDeviceIO class that can be used to create non- GHOST controlled servo loops. It should also include several bug fixes and make it easier to directly access the Phantom. This version was not available at Certec so we have not been able to try if this would have been easier to use for integrating haptics in Crystal Space or give other evaluation results.

A drawback for GHOST is of course that a software license costs money. However to buy a license is not as expensive as buying a Reachin API license. A GHOST License costs 2500€ compared to a Reachin license that costs beyond 11000€. A haptic API that has a reasonably cost or is completely free of charge would obviously be best suited for this project.

One problem with GHOST is that because it is not a free API, the source code is closed. This means that it is not possible to make changes where the haptics loop is calculated. For example, it is not possible to change the way collision detection, culling or force calculation is made. One way to introduce self-calculated forces however is to use a special force field node that is calculated in every update. This way it is possible to assign any forces at any time. Actually this is how e-Touch is originally constructed. The e- Touch driver class uses a scene with only a gstPhantom node and a gstForceField node. This is why, as mention before, e Touch is not really a completely independent API.

3.2 e- Touch

One of the main problems that we observed in e-Touch was the difficulty to disconnect the haptics part, which we wanted to take advantage of, from the graphics part. As mentioned before the central object in e-Touch is the user class. This class starts both a graphic manager part and a haptic manager part. But if one only wants to use the haptics part it is no just to remove the start of a graphic manager. Some graphics calls were buried deep down in the code so it took a lot of time to understand and remove this. Hopefully the graphics part will be better disconnected from the haptics part in future releases of e-Touch.

Another issue that can be said to be of concern is that e-Touch is still in the bera stadium. The current release used in this project is version 1.0.0 beta3. When looking through the code one notices a few hacks and comments at certain places. This means that everything may not yet have been thoroughly tested and structured in the best way.

As it stands now e- Touch currently works on Windows NT, Windows 2000 and Windows XP platforms. GHOST on the other hand also supports the Red Hat Linux platform. This is kind of a disadvantage for e- Touch because Crystal Space also supports the Linux platform and the plugin cold then perhaps have been made to support more platforms.

4. Game engines

4.1 Introduction to game engines

When creating a 3D game one needs software that handles the virtual environment and renders it to the screen. There is also need for software that handles for example physics and collision detection. These tasks are handled by the 3D game engine. One can say that the game engine is what powers the game. It is a platform upon which the game is built and it provides functionality that is common to all games, such as the things mentioned above and also the access to various hardware devices, for example keyboard, mouse, joystick, graphics accelerator, network card and so on. The advantage of a 3D game engine is that it is not necessary to create a new game engine for every new game. Instead a developer can reuse the same game engine over and over again. This saves the game developer from a lot of work when creating new games, but also makes it easier to enhance or add features to the game engine itself. The game engine performs a lot of time consuming tasks, so every enhancement that can be made to the engine may provide for new possibilities for the game developer.

4.2 Crystal Space

The game engine that is used in this project is called Crystal Space. It is a free 3D game development kit written in C++ and it is being developed as an open source project. Crystal Space has most of the features that are needed in a 3D game engine, but it is still under development, which means that it is constantly being improved with new features. Since it is an open source project with hundreds of people working on it all around the world the documentation of the system sometimes is worse than desirable which can make Crystal Space a quite difficult system to use. However it is the open source development approach that makes Crystal Space worth looking at in the first place. Crystal Space falls under the GNU copyleft license for libraries, which means that anyone is allowed to use it in their products and even to sell their products using Crystal Space provided that Crystal Space itself remains free. This approach makes the Crystal Space development kit desirable to game developers because it is free of charge and since it is open source they can even make changes in the game engine itself to customize it to their own game.

As mentioned above Crystal Space has many features. For example, it supports six degrees of freedom, colored lighting, mip-mapping, sectors and portals, mirrors, procedural textures, particle systems, OpenGL rendering, collision detection and physics, and it also has a flexible plugin system.

The remains of this chapter will provide an explanation to how the Crystal Space game engine functions. The default main loop is explained and also how it controls the execution using the event system. How a 3D environment is built up using meshes and also how this environment can be rendered to the screen are other topics that will be given an explanation here. There is also an explanation of the collision detection system as well as the dynamics system that are used in Crystal Space since they are important to our project. To get a more comprehensive knowledge about Crystal Space one should examine the user's manual [1]. The game engine is updated fairly often so it can also be useful to check out the homepage [13] for the latest news. There is also a Crystal Space developer homepage [14] with a forum and mailing lists.

5. Design

When we started to look at how to inergrate haptics with the game engine, we decided quite fast that a plugin to Crystal Space was the best way to do it. Implementing the haptics part as a plugin divides up the functionality in a very nice way so that the haptics does not affect other functionalities in the game engine.

This chapter is a detailed description of the design of the plugin, but it also presents the problems that we encountered in the development stage and how we chose to solve them.

5.1 Task

The intention with the plugin is that it should provide a haptic representation of the geometry contained within the Crystal Space making it possible to feel the environment of the camera and changes in the environment such as movements of dynamic objects. When the camera is moved in the application the

phantom should give the sensation of moving with the camera. The phantom workspace can be thought of as being attached to the camera and oriented in the same direction. When an object is moved in the application, for example due to gravity, and collides with the phantom pointer, the movement should be felt through the phantom. This should also work in the other direction, that is, if the user presses the phantom pointer against an object that is a dynamic object, the object should start to move if the force is sufficient. Furthermore there should be a possibility to set different haptic properties for object. By modifying these properties the user will be able to feel a variety of sensations.

5.2 Program structure

5.2.1 GHOST and e-Touch parts used

After the evaluation of GHOST and e-Touch we decided only to use a small part of their functionality and to implement the rest ourselves. For example, the part of the plugin that keeps track of all the haptic objects and the part manages the force collection from the objects are not taken from the APTs.

As in e Touch, we use a small GHOST scene graph to be able to communicate with the phantom device. The scene graph contains a gstPhantom node, which is needed to be able to retrieve the status of the phantom and also to be able to change its tsatus. In our case it is the position of the stylus that is retrieved from the phantom device and it is the position of the phantom workspace that needs to be changed when the camera moves around in the 3D environment. There is also need for a gstForceField in the scene graph. The force field is needed to be able to send forces directly to the phantom because GHOST is not used for the haptic calculations.

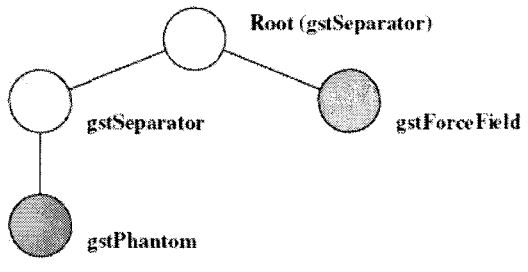

Figure . The small GHOST scene grapb used in the plugin to communicate with the pbantom.

Other things that we also use from e- Touch are modified versions of the haptic objects classes. These classed are used to represent the objects and they contain the force calculation algorithms. To make these classes fit into the rest of our plugin we examined them thoroughly. We then removed as much code as possible that was not needed for the force algorithm to work and made a few changes to create a more suitable interface. There are also other parts of the e- Touch API that are used but they have not been modified in any way.

5.3 Threads and processes

When the plugin is used together with Crystal Space there are two different processes running parallel to each other. There is the graphics loop that is run by the game engine and the servo loop, which is needed to update the haptic device with forces. As mentioned before the Phantom needs a very high update rate whereas the demands on the frame rate are not that high. The servo loop is therefore run at a fairly constant speed while the graphics are run without a minimum frame rate, which in practice means as fast as possible.

To communicate with each other the two loops use shared variables that are protected through the use of semaphores for mutual exclusion. The semaphore ensures that only one thread at a time is able to read or write a shared variable. If one thread is inside a critical region and the other one should try to take the semaphore at the same time, it will have to wait until the semaphore is released again before it can access the shared variable.

5. References

[1] J. Tyberghein, A. Zabolotny, E. Sunshine, T. Hieber, M. Ewert, S. Galbraith, 2003, "Crystal Space Open source 3D Game Toolkit Documentation", Edition 96.003.1 for Crystal Space 96.003.

[2] 2000, "GHOST SDK Programmer's guide", version 3.1, Sensable Technologies, Inc., Woburn MA

[3] 2001, "e-Touch Programmer's Guide", beta release 1.0, Novint Technologies, Albuquerque NM.

[4] 2001, "GHOST SDK API Reference", version 3.1, Sensable Technologies, Inc., Woburn MA.

[5] 2001, "e-Touch reference manual", version 1.0.0 beta 3, Novint Technologies, Albuquerque NM.