Performance Analysis of Real Time Multi-platform Physics Simulations Final Report

Sam Serrels
40082367@napier.ac.uk
Edinburgh Napier University
Physics-Based Animation (SET09119)

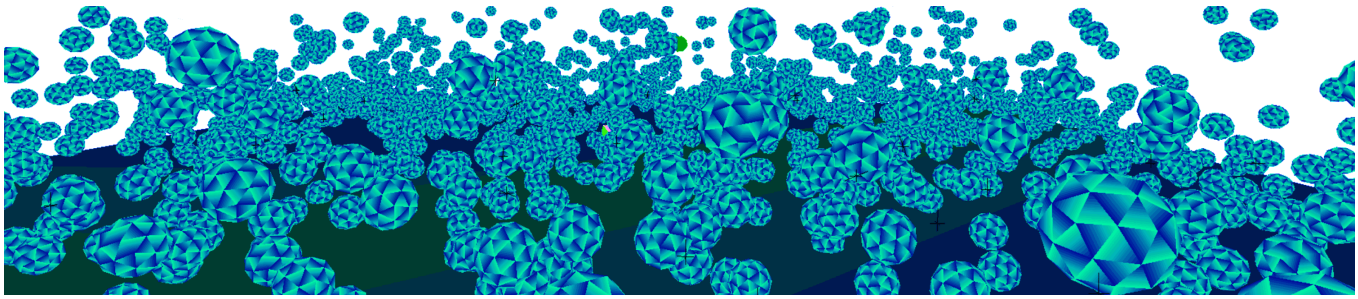

Figure 1: *Project screenshot*

Abstract

This project aimed to develop a physics engine and simulate a scene consisting of thousands of bouncing balls, running on various systems, such as the Sony Playstion 3. The performance of each system was measured as the systems were optimised and are compared in this final report.

Keywords: Multi-platform Physics, Optimisation, GPU, Cell, PS3

1 Introduction

Project Aims This project attempted to create a physics solution that would run on various systems, along with a game engine with an appropriate interface to swap in other existing physics solutions. The target platforms were the Sony Playstation 3, and a conventional x86 multi-core PC architecture.

The scene that will be simulated is a large set of Bouncy balls, travelling down a hill.

Physics Engines Large and complex video games tend to use 3rd party physics solutions, this vastly cuts down on the project development man-hours, and the maintenance thereafter. Third party physics solutions have the benefit of being battle tested out in the wild beforehand, so internal reliability is usually a given. A further benefit is that being developed solely for the purpose of being a "a good physics engine" by people who are usually experts in the field, large optimisations are already implemented. The problems arise in the implementation, the coupling of a physics engine and the existing codebase. While they are usually well coded, they are not tailor made to each game.

Optimising for Physics Engines Trying to regain performance from an external physics engine can be a hard task, diving into the source code requires expert knowledge of the inner-workings of the whole system. A common path is to shape the design of the game code to conform better to the demands of the physics engine and hope that the internal optimisations will be sufficient. Often enough, they are not.

Shipping on consoles Video games consoles have vastly varying hardware capabilities and architecture, writing robust code that is performant on all of them is a rarely possible. Code has to be redesigned for each system. The simple truth: more code means more problems, so throwing a large complex physics engine that you have no knowledge of the inter-working of into the mix, is a recipe for bad performance and bad code. However some physics systems do have separate versions for different architectures, assuming the interface is unchanged, games can swap in different libraries at compile-time. A further extension on to this would be to abstract all the physics so many different libraries can be plugged in.

Project Application The systems created in the project will be used to simulate a demo scene, picked to be processing intensive. The scene will be a large set of bouncy balls, bouncing down a sloped street. This will result in plenty of collisions, a large amount of data needing transferred every frame, and something interesting to watch. The inspiration for this came from a Tv advert [Sony 2008] where a similar thing was filmed in reality, with real bouncy balls and a real street.

Lack of Statistics Developers who make physics engines written and optimised for a single specific game don't usually publish much information on the performance. The results are always relative the specific game and have little meaning when applied to any other project. General Physics optimisations and platform specific optimisation methods are published, but are rarely included with performance statistics. This project aims to have a breakdown of all the optimisations taken with numerical performance statistics.

2 Related Work

Physics on the PS3 Some of the major physics libraries have been ported in some fashion to the Playstation 3.

Havok

The Havok physics engine is the most widely used 3rd party library for custom game engines and larger engines such as Unreal Engine 3. The library is a proprietary technology and information regarding the specific PS3 implementation is not publicly available.

Physx
The Nvidia Physx sdk was ported to the PS3, but it's use in released games is mostly undocumented. The PS3's "RSX" GPU does not include the CUDA architecture that modern Gpus use to further accelerate Physics.

Bullet
A fork of the open source physics engine "Bullet", was developed and released. Although not a popular PS3 engine, shipping with only a handful of games, it served mainly as a reference for other engines. As of 2012 this is no longer in active development.

Custom engines Unfortunately there is no good statistics on the ratio of games shipped with custom physics engines versus 3rd party physics. Documentation and relevant publications on the subject are also rare, the most documented physics implementations are the Bullet engine and the custom engine made by Insomniac games.

Insomniac's approach Insomniac games developed a custom solution for utilising the Playstations's SPUs as efficiently as possible, called SPU Shaders. Although not actually linked to shading or rendering, they follow a similar design to graphics shaders, by being modular self contained modules with one specific purpose.
Formally defined as:

- Fragments of code used in a larger system
- Code is injected at location pre-determined by system.
- Custom made for any particular system.
- Allows modifications of system data.
- Can feedback to other systems outside the scope of the current system.

[Acton 2007]

This system is used extensively for all SPU work in the game, but is also key the implantation of the physics engine. This allows for a modular physics system, tailor made to the SPU's limitations and strengths. The primary benefit is that it also becomes a standalone set of tasks, leaving the main processing thread to deal with other tasks. (See Fig:**??**)

Bullets's approach The Bullet PS3 implementation took more of a *port* approach rather than a *complete redesign*. This makes sense as Bullet is a Library rather than a single game solution, effort was made to keep the core codebase as untouched as possible to allow for greater stability across all the platforms. The main work was done in the area of scheduling and threading. The system was designed so that the SPU jobs could easily be reimplemented on another platform like the Xbox360 as CPU threads. The SPU code had it's major specific changes in the region of data management and loading, the physics processing code remains largely unchanged apart from some minor optimisations.

> "The Generic convex collision algorithm performs a bit less then specific algorithms.
> This generic approach requires Intermediate Data Swapping Between PPU and SPU and requires synchronization between SPUs" [Coumans and Christensen 2008]

(See Fig:**??**)

Physics on the Vita the Processing architecture of the Playstation Vita is similar to that of a common desktop processor with 4 cores, the difference being that the instruction set is ARM rather than X86. The gpu is designed for graphics, and graphics only, so all physics processing on the vita will take place on the GPU. At the current time the only documented Physics library for the Vita is Havok, with no public published articles regarding performance or implementation details.

Physics on the PC There are multitudes of different physics implementations on the traditional architecture of the PC, where the extra resources of the hardware mean that more focus can be placed on usability and implementation of the physics. It is more common to see physics engines compared against each other based on the complexity of the code rather than the runtime performance (this is also partly to do with the large hardware and capability differences between computer systems). Many reports have been published analysing the software design methodologies and ease of implementation of physics engines.

> "Using Software Quality Metrics which give an empirical indication of (among other things) the reusability, usability, and flexibility of a piece of softwarewe can begin to investigate ways of improving the games development process."[Christopher Mann 2012]

This project This project will take a similar approach to that of Bullet, by creating a generic physics engine that will run on multiple systems with as little as modification as possible, while still aiming for the best performance. The end result should be a program that uses good software design methodologies, meaning code that is easy to understand and modify. Platform specific modification can then be implemented and the change in performance against the change in code complexity can be measured.

3 Background / Related Work

4 Software Design

5 Results

6 Evaluation

7 Future Work

Scope of Physic capabilities The physics engine created for this project will be limited in scope and capabilities rather than a full general solver. The primary function will be Collision detection and realistic movement of single rigid-bodies. Constraints, animation, Inverse Kinematics, and advanced mass calculations are not planned to be implemented although the system should be designed to accommodate these systems in the future.

Simulation Scene The project will be simulating a large set of bouncing balls travelling down a hill. The ball will be solid object with mass, and will collide with each other and the surrounding geometry. the geometry will consist of flat shaded primitive shapes, to roughly emulate the real life scenery of the street in the advert.

Simulation Objectives The aim of the simulation is to measure the performance of the physics simulation, I.e the maximum amount of balls that can be simulated. Rendering the scene is also an objective, as a physics simulation that cannot be rendered, either by design or by taking up too many resources is of no use for a real-time application.

Simulation Variables The balls will be created somewhat randomly, to the simulation should look different every time it is run. Variables could include changing the size and bounciness e of the

balls, changing the design of the environment (street slope, length, frequency of objects...) and other physics variables, like time-step and gravity.

Rendering Rendering the output of the simulation should result in a visually interesting scene, hopefully as similar to the reference video as possible. The focus of this project is optimising the simulation, but an impressive looking output that also achieves a good level of performance (60+fps) would be a large bonus to the project.

Data output The simulations should be logging and metering all the data to file as well as rendering the scene, this will be used in the evaluation report.

8 Application Description

Figure 2: ***Expected visualisation of the output*** *- Background geometry will not be this detailed [Sony 2008]*

The scene will start with an initial amount off balls at the top of the hill. They will be dropped from a height or fired via some force to give them initial velocity. Over time more ball will be created at the top of the hill to keep the simulation going. Balls may be removed once they reach the bottom of the hill or after a set time, this will depend on the overall performance limitations.

User Interaction The user will be able to use the input controls to move the camera around in the scene, to get close up views of whatever they choose. There will also be controls to alter the variables in the scene and execute gameplay functions. Input methods will take the form of a controller / mouse and keyboard where applicable.

Gameplay functions Functions that the user can use could include: adding forces to the scene, adding groups of balls at a certain location, removing balls, firing balls, and resetting the simulation.

9 Technical Specification

Parallelism the key to success in implementing this project will be to heavily rely on multi-core/multi-thread/multi-processor software architecture. For the Playstation 3 this is an absolute necessity, as the SPUs are the only obvious way to implement physics. A parallel approach will benefit all systems, and designing for the small memory limits of the SPU will also benefit other platforms as smaller code and datasets can make use of the extensive caching available on Desktop Processors.

> "Old Processing model: Big semi truck. Stuff everything in. Then stuff some more. Then put some stuff up front. Then drive away.
> New SPU model: Fleet of Ford GTs taking off every five minutes. Each one only fits so much. Bucket brigade. Damn they're fast!" [Acton and Christensen 2008]

Optimisation Optimisation can be split into 2 large categories, algorithmic optimisations and data optimisations. Algorithmic optimisations is the process of monitoring all the operations done within a physics tick to spot redundant operations, or to swap slow operations for faster methods. This can be done initially by using profiling, but it can result in going down to the compiled assembler code to see if the compiler is misbehaving. This is often the case when trying to squeeze every process cycle out of an SPU due to it's highly vectorised SIMD based instruction set.

Data Optimisation This is the task of minimizing the transference and duplication of data. Moving Data takes time and can hog buses and caches that need to be used by other systems. In a panellised systems this can become a headache as the extra case of *when* items need to be accessed needs to be take into account, so that threads are not blocked by other threads.

Achieving Optimisation This project intends to measure the very best performance out of the different systems, getting to that level of performance will take a large amount of the development time. Real-world game projects have resource budgets and target to hits (i.e 60fps) and then they can ship the game. This project has no performances target, so the aim is to keep optimising until there is nothing left to optimise.

Software Design The project will be written as one solution with platform modules automatically swapped at compile time, rather than a separate project for each solution. this promotes shared code between the systems which reduces bugs and increases code quality. This method does have the downside of possibly having a large maze of "#ifdefs" but with proper inheritance and code design, the impact should be minimal an confined only to the interface classes.

Physics abstraction In this section, you should put together some of the technical details of your simulation. This will help when developing your physics-based simulation later.

External libraries The current list of libraries that project will require is as follows:

SDL - Simple DirectMedia Layer
Used for creating windows, polling input and general useful platform code for PC/mac/linux.

GLEW - The OpenGL Extension Wrangler Library
Used as the entry point into the OpenGl framework, used for detecting graphics capabilities.

GLM - OpenGL Mathematics
Maths Library used on Pc platforms.

10 Conclusion

In conclusion, this project should generate some new information about the physics capabilities of the Playstation 3 and Plystation Vita, along with an interesting interactive display of the simulation.

References

ACTON, M., AND CHRISTENSEN, E. 2008. Insomniacs spu best practices. Game Developers Conference.

ACTON, M. 2007. Spu shaders. Game Developers Conference.

CHRISTOPHER MANN. 2012. Game physics engineanalysis and development. *Honours Dissertation, Games Development, Edinburgh Napier University*.

COUMANS, E., AND CHRISTENSEN, E. 2008. Spu physics. Game Developers Conference.

SONY. 2008. Boucy balls advert http://www.youtube.com/watch?v=-zorv-5vh1a. *Accessed: Oct 2014*.