Performance Analysis of Real Time Multi-platform Physics Simulations Design Document

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

Figure 1: *A) Project Inspiration [Sony 2008], (B) Rigid bodies [Facepunch Studios 2014], (C) Colliding Particles [Crytek 2011], (D) Non colliding particles [Epic Games 2014]*

Abstract

Realistic Real time physics simulation is highly sought after in interactive applications, especially games. Achieving high-accuracy while maintaining performance in often resource restricted environments(I.E a games console) requires the highest level of optimisations and often results in a trade-off with simulation speed against Accuracy. This project attempts to record and analyse the performance of various optimisations on a simulated scene. This will be taken further by applying the project to various different processing architectures. The scene that will be simulated is a large set of Bouncy balls, travelling down a hill.

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

1 Introduction

Realistic Real time physics simulation is highly sought after in interactive applications, especially games. Achieving high-accuracy while maintaining performance in often resource restricted environments(I.E a games console) requires the highest level of optimisations and often results in a trade-off with simulation speed against Accuracy. This project attempts to record and analyse the performance of various optimisations on a simulated scene. This will be taken further by applying the project to various different processing architectures. 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 Aims This project will attempt to create a physics solution that will run on various systems, along with a game engine with an appropriate interface to swap in other existing physics solutions. This will allow for fair comparisons and benchmarking of the physics solution created by this project.
The target platforms will be the Sony Playstation 3, Sony Playstation Vita and a conventional x86 multi-core PC architecture. Utilising the general purpose processing functionality of the graphics cards within the PC system for physics is also a possibility.

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 part 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:3)

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:2)

Refer to literature on the particular physics-based animation effect you want to synthesize (e.g., published articles, books, conference proceedings, web articles) provide a comprehensive review - and use the correct citation format

Related work should finish with a summary paragraph - emphasising the crucial similarities or differences between existing methods presented in the literature. For example: (1) you might want to modify the technique to run on the GPU; (2) or you combine different techniques from different authors; (3) or you are simplifying the algorithm to make it run faster.

3 Overview

Brief overview of the core principles and mechanism behind your effect. This should be reflected in your final implementation, so consider what you will actually be implementing. What components make up the effect and how are they connected.

4 Simulation

How does your simulation work and what are the reasons it is important. This is a decisive section to put together as it will help you in the rest of your physics-based animation development.

- Physics-Based Animation Description Provide a more detailed description of how the simulation functions.
- Formal Elements Description of your simulation using the formal elements. You should include the following sections.
- Interactive Describe the interactive pattern you are aiming for .
- Objectives What are the objectives of the simulation? You may have one overall objective and some sub-objectives.
- Procedures What are the actions performed in the simulation. Are there any specific technical workings?
- Rules What test cases have you defined to demonstrate the effectiveness of your technique? Define a list of simple tests - both showing the good and the bad parts of your approach.
- Resources What are the resources within the solution? Is it procedural? Does it require artist intervention? Is the solution scalable? e.g., large and small scenes?
- Boundaries What are the boundaries of the technique? This will probably be based on experiments - the technique might

only fit specific situations (e.g., outside/inside/small high detail situations or crowds of low-level detail).

- Outcome What is the outcome? Explain any possible issues? Complexity? Computational power? Remember, relate this back to your investigations, related work, and experience - what have other people said? Do you agree with them?

5 Description

- Screen Mock-Ups & Preliminary Screenshots Provide any screen mock-up of how your animation will look or preliminary screenshots. You can use whatever method you see fit to create the mock-up. Your mock-up doesn't have to be exactly how your simulation will look, and you can use in place graphics. Explain principles and concepts to the reader - so you can explain your idea.
- Dynamic In this section you should describe any user interactive controls for customizing or changing the simulation - to effect the outcomes of the effect (e.g., adding external forces, numbers of objects, gravity)
- Control Mechanism Describe the control principles. What controller mechanism have you gone for (i.e., random or from scripts)?
- Rules Reflect upon the rules defined in the formal elements, and how these may affect the controls of the simulation. Also consider any relationships between physical objects which might have an influence on the controls.
- Modes and Other Features Does your effect support multiple modes? Are there any other features (e.g. novel control mechanisms) which need to be taken into account?
- Test Scenes Does your simulation have different scenes (2D, 3D, different complexity, real-time, off-line)? If so, then design a variety of simple test cases to show the situations and demonstrate the viability of your technique.
- Flowchart Create an overview flowchart - and explain the interaction of the different components. For example, do you update the forces, then move forwards, then draw....

6 Technical Specification

> "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]

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.

- Technical Analysis This section analyses the technical requirements of your simulation. The goal here is to describe the main software development tasks, the risks involved, and any external libraries you are using.
- Major Software Development Tasks What are the major pieces of development to make the software work? Identify the main tasks from the simulation design, particularly items you feel will be difficult to implement. List these here so you can approach these tasks individually.
- Risks What are the risks in your development? Consider which pieces of functionality will be difficult to implement, and what are the options if you cannot achieve them.
- External Libraries Are you using any external libraries or resources to implement your simulation? If you are using any libraries outside the ones developed in the practical sessions, these will need to be described here.

Equations should be numbered and in the correct format, e.g., Equation 1 below:

$$\sum_{j=1}^{z} j = \frac{z(z+1)}{2} \tag{1}$$

7 Conclusion

The report should finish with a summary to give a brief overview of what the reader should remember most. What was most important?

References

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

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

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

CRYTEK. 2011. Cryengine - destrucction demo.

EPIC GAMES. 2014. Unreal engine 4 - particle demo.

FACEPUNCH STUDIOS. 2014. Garry's mod - source engine demo.

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

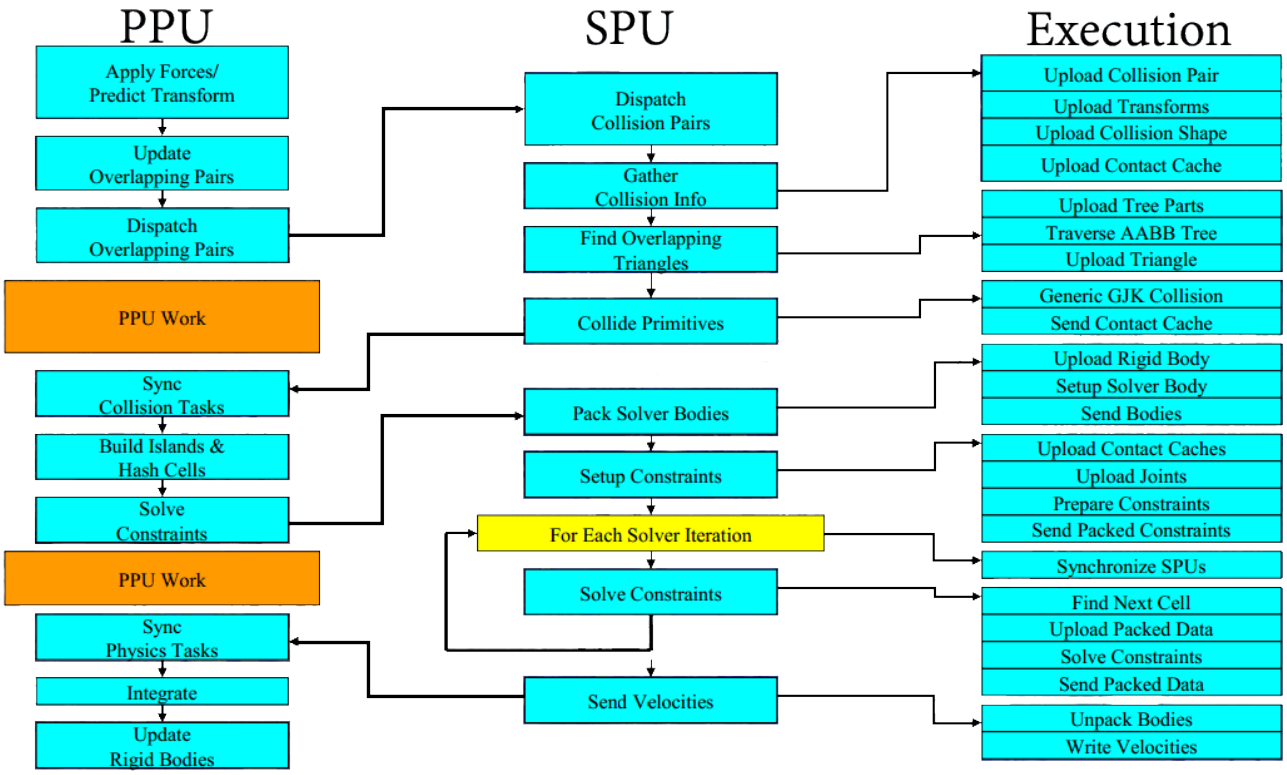

Figure 2: ***Bullet Physics PS3 Pipeline*** *- Requires Intermediate Data Swapping Between PPU and SPU [Coumans and Christensen 2008]*

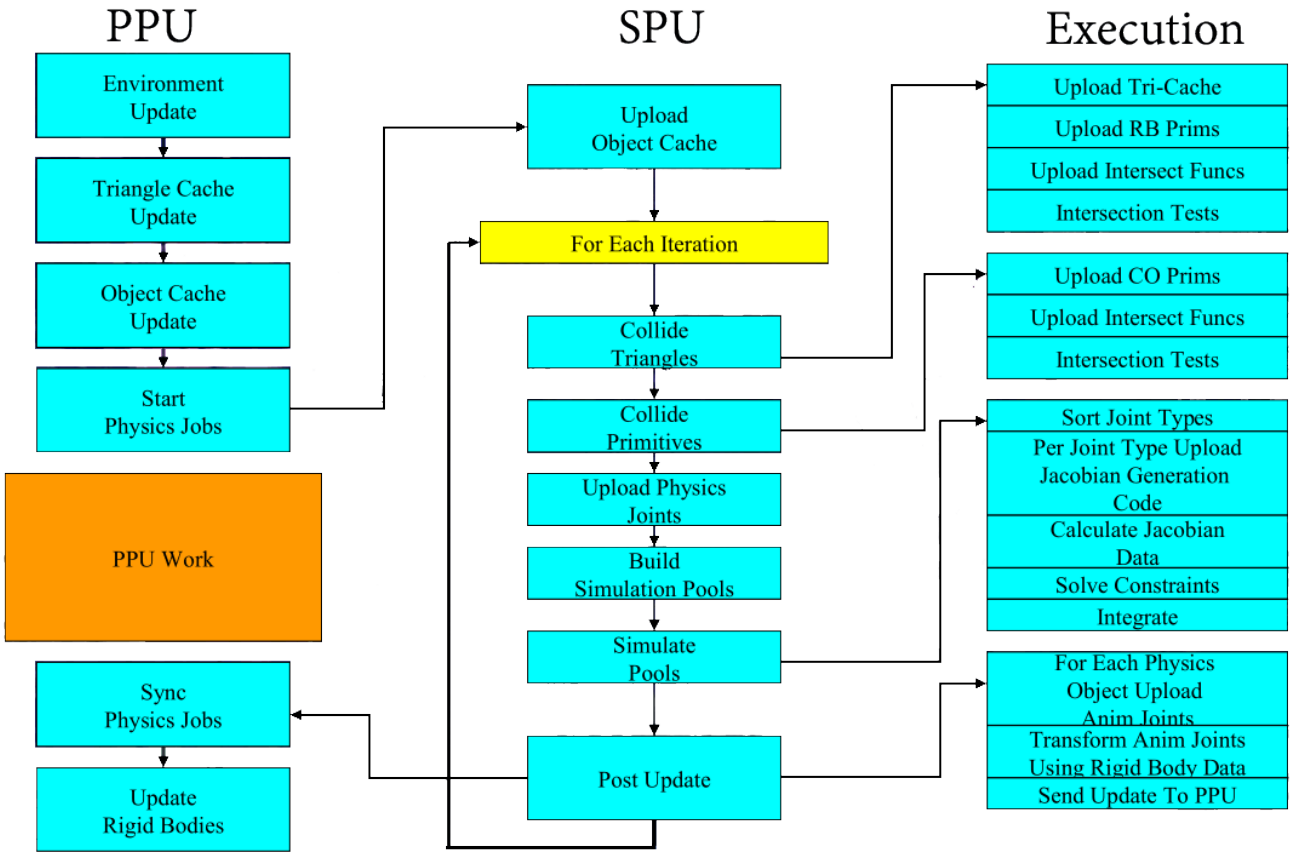

Figure 3: ***Insomniac Physics Pipeline*** *- No Intermediate Data Swapping Between PPU and SPU once the physics job has started [Coumans and Christensen 2008]*