Abstract

"From Dust" is a new game for Playstation 3, Xbox 360 and PC developed at Ubisoft Montpellier Studio. It features a fully dynamic world, where sand, water, lava and vegetation are constantly evolving and ruled by a realistic simulation.

While processing some mostly static terrain environments in video games is a well established subject, "From Dust" takes its unique set of challenges in the highly dynamic nature of the environment.

With our approach, it is possible to create a game that takes place in a fully procedural environment ruled by a complex simulation for current generation’s consoles.

1. Introduction

The world is made of a grid comprising several layers that store the information about rock, sand, fluids and vegetation for each cell. Cellular automatons are used to update these different layers.

For example, Navier-Stokes equations are used to compute the motion of fluids and update the fluid layer. Different rules are used to determine if vegetation should spread, burn, live, die, bloom or grow, and update the vegetation layer.

A full update of the simulated world is required at each frame no matter the complexity of the world, and as the gameplay depends on the simulation, the update is not only limited to the visible part of the map.

To achieve this huge amount of data processing, we iteratively enhance our code and methodologies, to use at their best the 3 target platforms.

2. CPU Compute shaders

All the simulation and rendering code is designed as SIMD CPU compute shaders [1] and runs natively on PS3 SPUs. These shaders use some of the objects as inputs and produce some outputs which will be either used by other shaders or by the GPU.

Objects can be buffers, streams, textures, vertex buffers or display lists. Compute shader jobs can often run concurrently, enabling the full usage of up to 8 processors, and run in parallel with the GPU processing.

3. Low-level Optimizations

Using a data-driven approach, we had to deal with a lot of features such as instruction latencies, in-order execution stalls, 256bits SIMD porting, cache-misses, cached memory aliasing, write-combining, memory prefetching, local-storage limitations, lock-free multi-threading, hyper-threading, load balancing.

Even if these 3 platforms are quite different, you can share some methodologies across them to reach a high level of performances, using only one version of the code.

We will explain how we did this, step by step.

4. Results

The game runs at 30 fps on consoles and 60 fps on a modern PC, using all the available CPUs. Even with the simulator running, lots of resources are still being available for the rest of the game.

We now have a set of methods and code design suitable to be used not only for physics, but also for rendering, AI animation ... adapted to multi-threaded architectures, with concepts shareable between CPU and GPU programmers.

References