Using GPU Compute for Productivity and Play

Engin Cilasun
Avalanche Studios, engin.cilasun@avalanchestudios.se

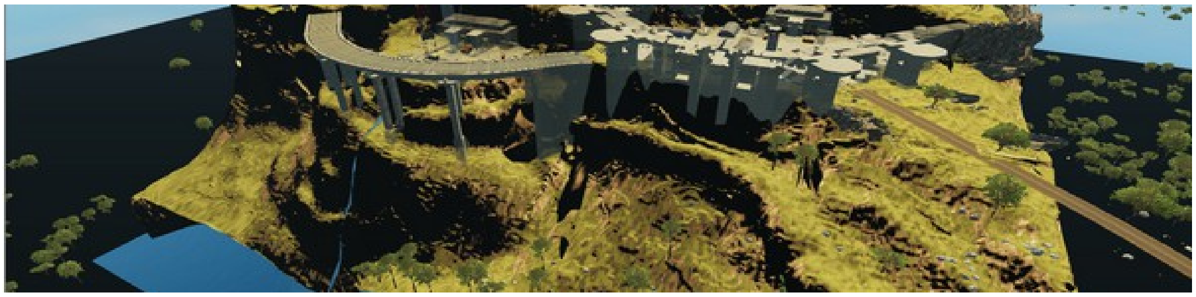

Figure 1: *A terrain patch in the editor environment*

Abstract

A summary of our GPGPU use in the production of Just Cause 3, starting from the editor tools, summarizing compiler pipeline and finally detailing some aspects of GPGPU in Just Cause 3 runtime.

Keywords: gpgpu, volume data editing, parallelism, sparse data storage

1 Background

As an integral part of all games developed by Avalanche Studios, large scale terrain and related custom editor tools have always been heavily used.

As Avalanche Studios have progressed towards a volumetric, non-height based representation of terrain, the data required to represent it grew to hundreds of gigabytes. This brought questions about whether we would be able to handle it fast enough entirely on the CPU. Therefore, to allow for faster iteration times, and to meet our delivery dates for Just Cause 3, we have decided to utilize the GPU throughout our new editor tools and compiler pipeline.

Here we detail how such a switch is possible from the CPU side, and how large data is arranged so it can be handled on the GPU, as well as give a brief sample on how one such GPU code operates on the data and what we output for the game to consume.

SIGGRAPH 2015 Talks, August 09 – 13, 2015, Los Angeles, CA.
ACM 978-1-4503-3636-9/15/08.
http://dx.doi.org/10.1145/2775280.2793104

2 Changing our mindset

Initially all pieces of Just Cause (and our other title's) game play code depended on some form of height data to exist given a two dimensional position on the terrain's projection plane. This, however convenient and fast, had its drawbacks, such as not being able to carve tunnels and generate overhangs.

It was a simple enough task to switch to volumetric representation, using a scalar field. This was adequate for our purposes since the terrain data could be represented at an even-higher resolution than before with a simple grid of scalar values.

When it comes to using this data on the GPU, however, we see that a simple grid is not the best fit for parallel access patterns.

Also this meant the gameplay code depending on height data could no longer sample a height value directly.

Looking at the requirement of runtime code and the data representation, the decision was to share some of the data that we compile for the graphics representation with the physics code so we could be memory efficient on modern consoles.

3 Choice of OpenCL

The choice to use OpenCL for our editor and compiler pipeline was made early on. C++AMP, DirectCompute, CUDA were all candidates at the time but OpenCL was the most adequate for what we were aiming at. OpenCL also has the necessary extension such as byte access buffers and some means to get debug output or debug it on the CPU, so we could see what our code was doing during development.

Some of the aforementioned APIs are also vendor bound, which made us skip their use in favor of utilizing a more wider range of hardware.