Research Proposal: An Exploration of Optimisation Techniques for Vulkan-based Particle Systems

Robin Wragg

October 8, 2019

1 Introduction

Vulkan is a cross-platform Application Programming Interface for rendering 3D graphics, currently developed by the Khronos Group. It provides new opportunities for high-performance, realtime graphics compared to its precursor, OpenGL. Vulkan is a much more explicit API, potentially putting a greater mental load on the programmer but likely to give the programmer greater control over the graphics processor, which should limit executional uncertainty and provide high performance overhead.

These assumptions, as well as Vulkan-specific and particle-system-specific challenges, will be explored in this project by developing a performant, Vulkan-based particle system.

2 Literature Review

The trend of ever-increasing clock speeds is reaching its limit for the current state of the art in CPU fabrication. Gillespie (2009) of Intel states “One of the most significant challenges in the recent history of the microprocessor industry was the rise of inherent limitations to the ability to increase performance by driving up clock speed. [...] Multiple processing cores now run in parallel at relatively low clock speeds to provide high performance at low power.”

This posed a problem for programmers using older graphics APIs; Blackert (2016) found that “OpenGL is unable to efficiently use multiple threads for rendering commands which can mean that a rendering intensive program is limited by the single core performance of the CPU." In light of this, graphics API vendors moved toward multicore-friendly APIs, giving graphics programmers increased potential performance and new challenges to utilise the modern APIs effectively.

As one of those modern APIs, Vulkan removed the need for the graphics driver to handle high-level features such as state validation and error checking, and instead allowed the application to perform those actions as needed via an explicit API. This decreased non-essential traffic through the driver and reduced performance randomisation (Joseph 2016, p. 23).

Particle systems are a useful environment for experimenting with performance because they primarily function by rendering multiple similar objects per frame; the faster the render, the more objects are able to rendered without a noticeable drop in framerate, allowing improved visual results (Beeckler & Gross 2008). This quantitative nature makes them appropriate for quantifying the performance impact of any changes made to the program.

3 Methodology

C++ and Vulkan 1.1 will be used to produce a particle system which will serve as an environment for experimentation. The system will be designed to provide a non-trivial optimisation problem.

The relative change in graphical quality when adjusting the number of particles in the system will be observed in order to contribute to the understanding of how performance can affect graphical quality, but the overall graphical quality of the system (aside from the particle count) is less important; the focus of this project is on the optimisations, not on producing realistic particle effects.

Approaches to optimisation will be informed by the literature on parallel computing, CPU cache usage, bottlenecking, shader optimisation and so on. Profilers will be used to study the performance characteristics of the program to further inform what modifications should be made. Changes in performance will be noted by studying changes in the framerate and the number of particles renderable in a given time period, for a given computer.

Failed optimisation techniques will be reverted and new techniques will continually be attempted, in order to accumulate the beneficial optimisations and produce an optimal particle rendering pipeline.

4 Aims and Objectives

Test the knowledge on optimisation techniques found in the literature as well as speculative ideas, confirming their usefulness, and examining the degree to which they are useful by applying them to the particle system as a representation of a common use case.

References

Beeckler, J. S. & Gross, W. J. (2008), ‘Particle Graphics on Reconfigurable Hardware’, *ACM Transactions on Reconfigurable Technology and Systems* **1**(3), 1–27.

Blackert, A. (2016), Evaluation of multi-threading in Vulkan, PhD thesis, Linköping University.

Gillespie, M. (2009), Preparing for the Second Stage of Multi-Core Hardware: Asymmetric (Heterogeneous) Cores, Technical report, Intel.

Joseph, S. (2016), An Exploratory Study of High Performance Graphics Application Programming Interfaces, PhD thesis, University of Tennessee at Chattanooga.