Laboratoire d'Architecture des Processeurs
IN-F Ecublens
CH-1015 Lausanne
http://lap.epfl.ch

Semester Project
Winter 2020

Accelerating Ray-Tracing on an FPGA

DOBIS Andrew IN BA5

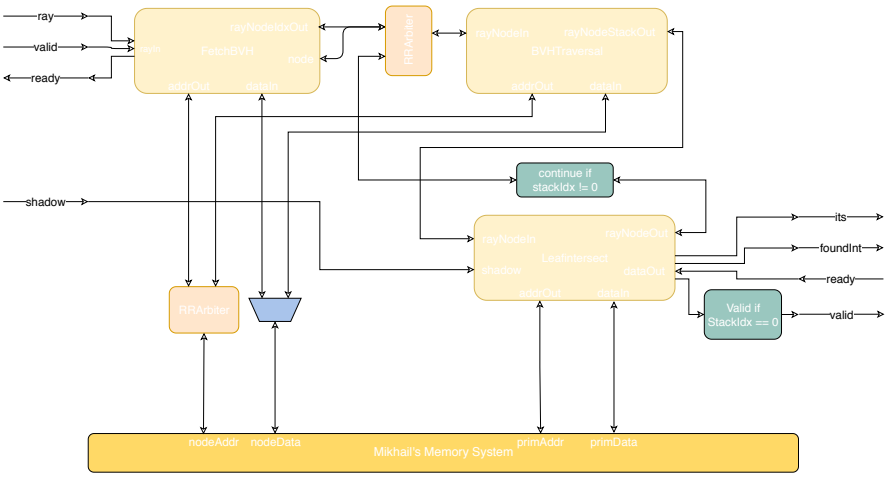
Introduction

Real-time Ray-Tracing has been the holy grail of computer graphics for many years, up until Nvidia released their RTX line of GPUs, using their new Turing Architecture, containing what they call RT-Cores. This new addition to the Nvidia GPUs enabled them to do what is know as Hybrid Rendering, which is a mixture of two rendering techniques Rasterization (direct conversion of mesh primitives into a bitmap that can then be displayed onto a screen) and Ray-Tracing. This method allowed them to render a couple of rays per pixel per frame, when needed, in a Real-Time application, but nothing more. This is due to the high irregularity in the memory access patterns of a standard Ray-Tracer, which causes a very large memory bottleneck due to the way that GPUs utilize their memory, which can potentially be solved using Mikhail's memory system.

Goals

- Design and implement an accelerator for the Nori Ray-Tracer.
- Verify the accelerator's functionality in simulation
- Run the accelerator along side the Software with and without Mikhail's Memory System.

Summary



The accelerator was implemented in Chisel3 to simplify the interfacing with Mikhail's Memory System (also in Chisel). The solution is separated into 3 different modules, FetchBVH, BVHTraversal and LeafIntersect. Given a ray, It is first associated with the root of the scene (in FetchBVH), the scene is then traversed until a leaf has been reached (BVHTraversal) and finally the leaf is tested to any possible intersection with the primitives it contains (LeafIntersect). Whence a primitive is intersected with, data about the mesh associated to the primitive is output. This solution has only partially been verified in simulation, however, its resource usage is mainly linked to two submodules used to computed the different intersections which add up to about 20% of the total resources available on the target platform (Xilinx ZC706 SoC).

Future considerations

The solution's traversal module still has an odd behavior given certain rays, where the state output from the control sub-moudle is incoherent (not delayed when it should be), so the modules should all be reverified in simulation. An other optimization that should be done would be to cache the root of the tree (and possibly other shallow nodes) to avoid having to refetch them for every ray. This would also get rid of the contention between FetchBVH and BVHTraversal when fetching nodes in memory. After that the accelerator should be run on the targeted plateform with and without Mikhail's Memory System to be able to verify the performance enhancements, if any. Finally the solution could be adapted to function on a much larger, and more realistic, scale, such as on Amazon F1 instances.