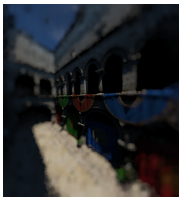
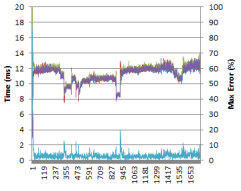
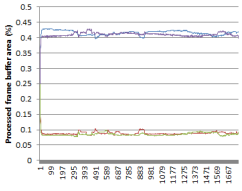
High Performance Graphics (2014)
Jonathan Ragan-Kelley and Ingo Wald (Editors)

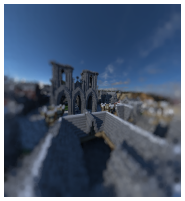
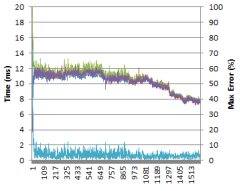
RayTracing Irregulaly Distributed Samples on Multiple GPUs

Takahiro Harada[†1] and Masahiro Fujita[‡2]

[1]Advanced Micro Devices, Inc. [2]Light Transport Entertainment Research



(a) 0.26M tris, 60fps (b) Time and max. deviation (a) (c) Area of frame buffer (a) (d) 6.7M tris (e) Time and max. deviation (d)

Figure 1: *Ray traced on four AMD Radeon™ HD 7970 GPUs. Average of max deviations of time are 3.3% and 3.7% for (b),(d).*

Abstract

Ray tracing has a freedom to choose sample location on a frame buffer. As we do not have to place a sample for each pixel as we do for raster graphics, we can increase or decrease the sample density depending on the importance. In this paper, we present a method to ray trace irregularly distributed samples on multiple GPUs. Our target is an interactive application which requires low latency rendering. A challenge of using multiple GPUs is we cannot use alternate frame rendering (ARF) introducing a few frame latency. To achieve a low latency rendering, a frame buffer is split once in a frame and all the GPUs are used to render a single frame. We developed a low overhead load balancing method for irregularly distributed samples, which is the major contribution of this paper, to exploit the computation power of all the GPUs.

Overview A sampling pattern is precomputed and loaded to the renderer (Poster). The pattern includes sample locations and k-nearest samples for each pixel (k=8). Also we store the radius of bounding circle of pixels each sample can affect. A frame rendering starts with a mater thread splitting the frame buffer into m areas using the algorithm described below, where m is the number of GPUs. Once the frame buffer split is sent to slaves, parallel rendering starts and each GPU executes 1. Find samples affects to the assigned area using bounding circle stored for each sample, 2. Compute color at samples using ray tracing, 3. Compute pixel color in the area by interpolating color of k-nearest samples, 4. Send the frame buffer and work statistics to the mater.

Frame Buffer Split Algorithm The proposed split algorithm uses the statistics from the rendering of the previous frame. Each GPU reports area of processed frame buffer, number of samples processed, and computation time for the work. The algorithm first computes processing speed p_i for each GPU. Then it computes ideal time to finish the work with the perfect load balancing $t = N/\sum p_i$, where N is the total number of samples. Number of assigned samples for each GPU is computed by $n_i = t \times p_i$. A cumulative distribution function (CDF) of the sample distribution on the screen in the previous frame is built (Poster). n_i is used to search for the split of frame buffer from the CDF.

Results The method is implemented in an OpenCL ray tracer running on four AMD Radeon HD 7970 GPUs and the results rendering 732x800 images are shown in Fig. 1. The maximum deviations of computation time on 4 GPUs are at 3.3% and 3.7% in average for scene Fig. 1(a) and (d).

† takahiro.harada@amd.com
‡ syoyo@lighttransport.com