Accelerating Virtual Texturing Using CUDA

Charles-Frederik Hollemeersch, Bart Pieters,
Peter Lambert, and Rik Van de Walle

Abstract

Virtual texturing is a promising technique to improve the visual quality of real-time rendering applications such as simulations and games. By selectively loading parts of the texture dataset, virtual texturing allows for higher resolution texturing than possible with traditional texturing techniques. However, virtual texturing also adds a significant overhead to the renderer. First, there is the task of determining the working set of the current frame. Secondly, there is the need to upload the streamed data. Furthermore, the data that needs to be uploaded may also not be available in the desired texture format. Hence, the data will need to be converted to this format.

Although virtual texturing subsystems are expensive, they lend themselves well to data parallel processing. The fact that much of this data will need to be further processed on the GPU by the renderer makes implementing these tasks on the GPU even more attractive since the data will be available close to its end use. In this poster we present how we implemented some of the components needed for virtual texturing using CUDA. They work together with our OpenGL based renderer, to offer a complete virtual texturing system.

The first such component we implemented using CUDA is the resolver, the component that determines the frame's working set. This subsystem works by analyzing a rendered buffer that contains the texture pages needed by the corresponding pixel. This buffer is prepared using a traditional OpenGL based renderer and a special pixel shader. We then analyse this buffer using a CUDA kernel that determines what pages are present in the buffer and masks the pages used by the current frame. A second kernel then packs the list of used pages. This way, a compact list of pages is acquired that can then be asynchronously transferred to the CPU. On the CPU side, these pages are then presented to a LRU based cache manager who replaces pages and places new loading requests. Our system allows an efficient working set determination without any unnecessary CPU to GPU transfers or synchronizations.

The second component of our system implemented in CUDA is the page uploading pipeline. When pages are loaded by our system, they are decompressed to RGBA data and placed in page-locked memory by a separate loading thread. They are then asynchronously transferred to the GPU without further CPU intervention. Once the pages are available on the GPU, mipmaps are then generated for the uploaded pages. These mipmaps will be used by our renderer to achieve trilinear texture filtering. After the mipmaps are generated, a second kernel is invoked that compresses the pages using a real-time DXT encoder. Finally, the pages are transferred to the OpenGL cache texture for use by our renderer. To optimize the page uploading process even further, we make use of multiple streams so CPU to GPU uploading and DXT compression can run in parallel.

From our results, we can conclude that CUDA based working set determination and page uploading, together with OpenGL based rendering and page translation table generation, allow high quality, filtered, virtual texturing to be easily and efficiently implemented on today's hardware.