Real-Time Compression of Dynamically Generated Images for Offscreen Rendering

Leah Emerson*
University of St. Thomas

Thomas Marrinan†
University of St. Thomas
Argonne National Laboratory

Abstract

Ultra-high-resolution visualizations of large-scale data sets are often rendered using a remotely located graphics cluster that does not have a connected display. In such instances, rendered images must either be streamed over a network for live viewing, or saved to disk for later viewing. This process introduces the additional overhead associated with transferring data off of the GPU device. We present early work on real-time compression of rendered visualizations that aims to reduce both the device-to-host data transfer time and the I/O time for streaming or writing to disk. By using OpenGL / CUDA interop, images are compressed on the GPU prior to transferring the data to main memory. Although there is a computation cost to performing compression, our results show that this overhead is more than offset by the reduced data transfer and I/O times.

Keywords: Image compression, offline rendering, ultra-high-resolution visualization, distributed rendering

1 Introduction

High Performance Computing centers provide researchers with immense computing power, but this comes at the cost of having output data generated in a remote location. When doing detailed visual analysis of large-scale data sets, HPC centers can be leveraged for distributed rendering in order to create ultra-high-resolution images. However, in order to view such images, they either must be streamed to a local display system in real-time or saved to disk then later downloaded to a local system. This process introduces two additional, and time-consuming, steps to the traditional rendering pipeline: 1) rendered images must be copied from GPU memory to main memory, and 2) images now in main memory must be written to some output device (e.g. streamed over the network or saved to disk). This overhead can significantly increase overall job time and inhibit the ability to achieve interactive frame rates when streaming for real-time viewing.

Often images are compressed to typical formats (e.g. jpeg, png, etc.) before saving to disk. While this does reduce the data size, and therefore decrease the I/O time, compression usually is computed on the CPU after the device-to-host memory transfer. This not only ignores the first source of overhead, but also can lead to non-negligible overhead for the compression itself. Instead, our work uses texture-based DXT1 compression [2], which can be easily parallelized for execution on the GPU. By executing compression on the GPU, we are able to add minimal computation overhead while greatly reducing the device-to-host memory transfer time in addition to decreasing the I/O time.

We have conducted initial experiments to demonstrate the effectiveness of on-device texture compression for offscreen rendering used by large-scale distributed rendering systems. Our results show that performing compression before device-to-host memory transfer reduces the time spent not rendering the visualization (and therefore limits the impact on rendering frame rates). Additionally, since images are compressed, a disk write or network send of the image is significantly faster than the raw uncompressed image.

*e-mail: leah.emerson@stthomas.edu
†e-mail: tmarrinan@stthomas.edu

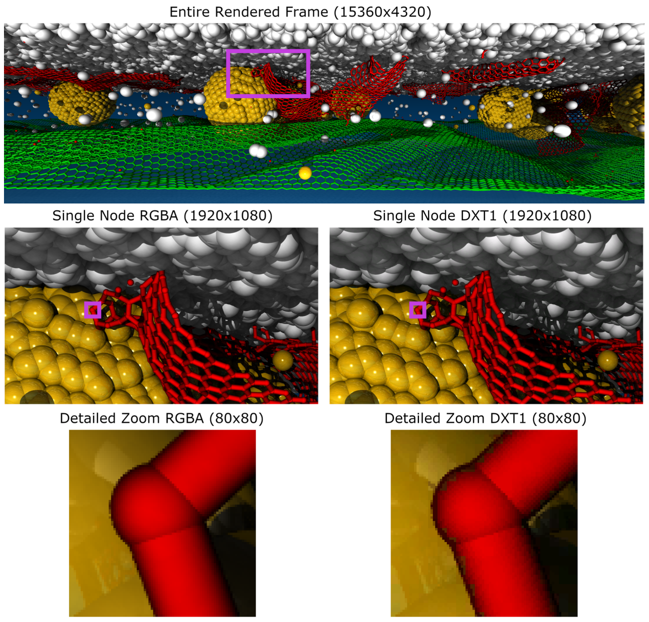
Entire Rendered Frame (15360x4320)
Single Node RGBA (1920x1080)
Single Node DXT1 (1920x1080)
Detailed Zoom RGBA (80x80)
Detailed Zoom DXT1 (80x80)

Figure 1: Comparison of visual integrity between raw RGBA pixels and a compressed DXT1 texture for an ultra-high-resolution visualization.

2 Related Work

Performing DXT1 compression on a GPU is not new. Castano created a CUDA implementation for real-time compression [1]. However, this work was more of a proof of concept on the capabilities of GPUs, and therefore did not detail motivation or provide authentic use cases.

Tokdemir and Belkasim [4] similarly have used the GPU for parallel image compression. In their work, they describe how compression using DCT has better performance on the GPU than the CPU. They indicate that increasing image size significantly hinders performance on the CPU, whereas the GPU performance is not affected.

Revanth and Narayanan [3] describe a distributed rendering process for achieving interactive frame rates when rendering data otherwise too large and complex to render on a single workstation. Geometry and screen space are both partitioned and assigned to individual rendering processes. A master process gathers all rendered portions for final display and interaction. Such a system would benefit from real-time image compression if the graphics cluster were a remote resource and images could not be displayed locally, but rather saved to disk or streamed via the network.

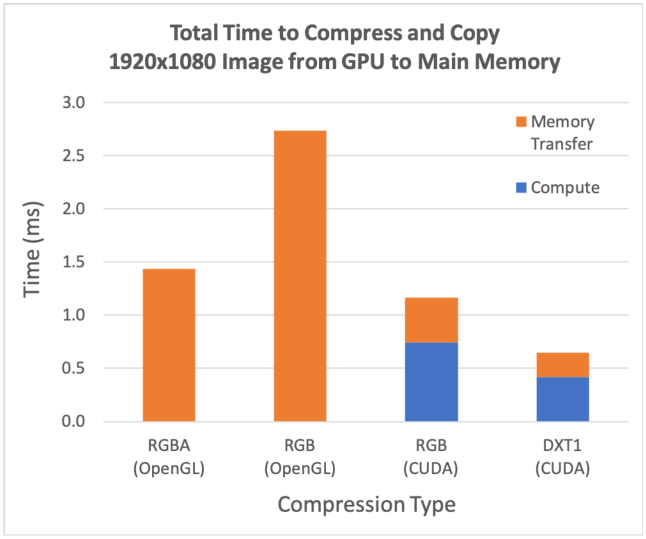

Figure 2: Computation and memory transfer performance for various compression techniques.

3 Real-time Compression on the GPU

Since images are being rendered by the GPU, we leverage OpenGL / CUDA interop to allow our CUDA Kernel to access (or copy) the data originally allocated via OpenGL. This eliminates the need to upload images from host memory to device memory. Our CUDA kernel is a parallel implementation of van Waveren's real-time DXT1 compression algorithm [5]. A rendered image is divided into tiles of 4×4 pixels. Each of these tiles is handled by a single thread. For each tile, our kernel computes the following:

- Find the min and max color
- Linearly interpolated between min and max to determine two other colors
- Assign each of the 16 pixels an index corresponding to one of the four colors (the color with the least error compared to that pixel's original color)
- Store min and max colors (in 16-bit RGB565 format) and pixel indices (2 bits per pixel)

A total of 64 bits are required to store each 4×4 tile (16 bits for each min and max color and 32 bits for the 16 indices). This results in an 8:1 compression ratio compared to raw RGBA pixels, which require 32 bits per pixel (512 bits for 16 pixels). While DXT1 is a lossy compression format, it maintains good image integrity for most images – especially those that are ultra-high-resolution. Fig. 1 shows a 15360×4320 rendering of a molecular dynamics simulation. Notice that even when looking at a very detailed subsection of the image, it is difficult to see the differences between RGBA and DXT1.

Since each thread on the GPU is only responsible for a 4×4 tile, the computation cost is quite low and scales well as image size increases.

4 Results

We used Argonne National Laboratory's Cooley cluster as a distributed rendering resource in order to test our real-time offscreen compression. Cooley has 126 nodes, each with 12 cores and a two-GPU NVIDIA Tesla K80, connected via FDR Infiniband. Each node is also conncted to a shared GPFS file system.

For our tests, 1920×1080 images are rendered to a texture in RGBA format on the GPU. Four methods were used to copy images from the GPU to main memory. We utilized OpenGL (via `glGetTexImage()`) to transfer the RGBA image, and CUDA (via `cudaMemcpy()`) to transfer the DXT1 image. Both the OpenGL and CUDA methods were utilized for "compressing" and transferring a RGB image. Fig. 2 shows the average total times to compress and transfer images to main memory. Note that DXT1 is 2.2x faster than RGBA, even with the added compute time.

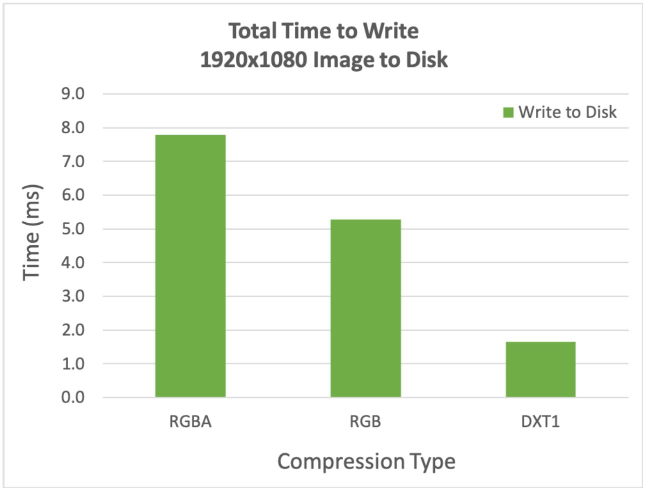

Figure 3: File write time for images stored in various formats.

Once in main memory, images must be output to some device. Fig. 3 shows write time for the three data formats when saved to Cooley's GPFS. Note that the DXT1 image was able to be saved to disk 4.7x faster than the raw RGBA image.

5 Conclusion

As our results have shown, DXT1 proves to be an effective algorithm for real-time compression of dynamically generated images. By using OpenGL / CUDA interop, we have decreased the overhead of transferring data from the GPU to main memory and reduced the data size for performing I/O.

In future work we would like to implement other compression algorithms in CUDA (especially lossless ones). We envision a smart compression algorithm that could perform the computation for a few different formats, then choose the most effective one prior to transferring the data to main memory. We are also interested in video compression using H.264/HEVC.

Acknowledgments

This research used resources of the Argonne Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC02-06CH11357.

References

[1] I. Castao. High quality DXT compression using CUDA, Feb 2007.

[2] K. I. Iourcha, K. S. Nayak, and Z. Hong. Fixed-rate block-based image compression with inferred pixel values, September 1999. US Patent 5,956,431.

[3] N. R. Revanth and P. J. Narayanan. Distributed massive model rendering. In *Proceedings of the Eighth Indian Conference on Computer Vision, Graphics and Image Processing*, ICVGIP '12, pp. 42:1–42:8, 2012. doi: 10.1145/2425333.2425375

[4] S. Tokdemir and S. Belkasim. Parallel processing of DCT on GPU. In *2011 Data Compression Conference*, pp. 479–479, March 2011. doi: 10.1109/DCC.2011.95

[5] J. M. P. van Waveren. Real-time DXT compression. 2006.