Efficient Depth-Extreme Retrieval for Use in Stereoscopic Applications

Kevin Bensema, Dr. Zachary Wartell
{kbensema, zwartell}@uncc.edu

Kevin Bensema, Grove City College.
Dr. Zachary Wartell, UNCC.

Introduction

Stereoscopic displays are used to visualize many 3-D environments. Two images are rendered, one from the perspective of each eye, and a virtual image is composed from these. If this image is too close to or too far from the viewer, optical problems result. To correct for this, the nearest and farthest points in a scene must be computed. We explore GPU programming methods to efficiently compute these values.

Background

Colin Ware did experiments to determine what model of eye separation to use in stereoscopic displays. His scenes consisted of “large continuous surfaces” and he used 100 samples from the depth buffer for his nearest/farthest point adjustments. However, this provides a good enough approximation only in certain cases, such as uncomplicated scenes.

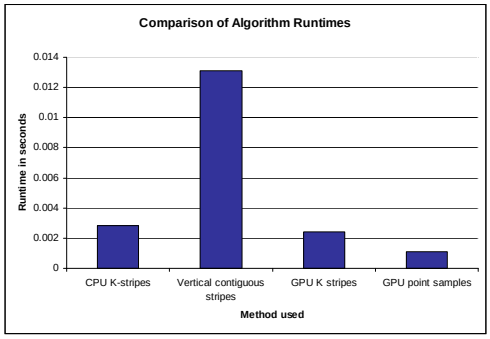
Wartell implemented a massive, whole-planet stereoscopic environment. One hundred samples would not have been enough to provide a reasonable approximation. Instead he read every twentieth row to main memory, where approximations to the minimum and maximum values were found. This provided a working estimation, but it scales with pixel read back speeds, a statistic which is not increasing anywhere near the rate that GPU performance is.

Research

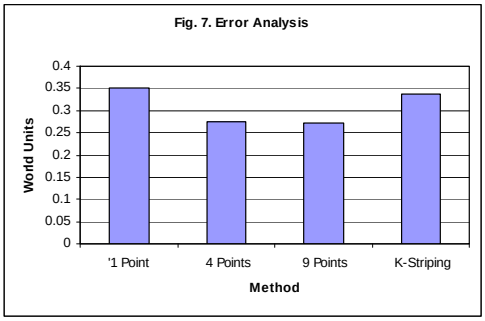
- We implemented and timed Wartell’s algorithm to use as a baseline.
- We realized that running as many of the comparisons on the GPU as possible would limit the amount of data that had to be sent over the system bus.
- We found a way to allow fragment shaders – small GPU programs – to access the depth buffer via a texture.
- Several versions of Wartell’s striping algorithm were implemented on the GPU.
  - A straightforward version
  - Two versions that attempted to gain a speed increase by cache-friendly programming.
- A separate algorithm, based on taking evenly spaced point samples from the depth buffer.
- We ran timing code comparing all of the shader-based algorithms against each other, and Dr. Wartell’s algorithm. Tests were run on four different video cards: the 6800GT, 7950GT, 8800GTS (G80), and the workstation FX4500 card.
- Due to timer precision issues, we timed 100 runs of each algorithm, and then divided that time by 100 to get an accurate runtime.
- A simulated cityscape was implemented and used as a test bed to verify that our algorithms were indeed sufficiently accurate enough to use.

Impact

- **The results of our timing are shown below.**



- **The GPU version of Dr. Wartell's algorithm did indeed run faster than the CPU version.**
- **We found that the contiguous methods that attempted to be faster by cache-friendly programming failed.**
  - **Vertical contiguous stripes ran slower than the CPU methods.**
  - **A Horizontal one was even worse**
- **Point Sampling methods ran the fastest.**
- **Error analysis showed that Point Sampling was also more accurate than stripes.**



Graph of measurement error

Conclusions

- We determined that while running implementations of the striping algorithms put forward by Wartell in his whole-planet environment run much faster on the GPU than on the CPU, but the GPU-based point sampling ran faster still.

- Error analysis showed that the point sampling method also returned more accurate data than the striping methods.

Future Work

In the future, we plan to implement our algorithm in the CAVE stereoscopic display in our labs to further test reliability and practical speed improvements. NVIDIA and ATI have developed SDKs-CUDA and STREAM, respectively-that allow mostly standard C code to run on certain video cards. We intend to explore the use of one of these development kits to implement the adjustment algorithm entirely on the GPU, thus eliminating most or all CPU-GPU communication overhead.