# Batching Terrain Line-Of-Sight Tests on a GPU

Joseph Del Rocco
University of Central Florida
04.25.2008

## 1. INTRODUCTION

It is no surprise that highly parallel algorithms perform well on modern GPUs, since most GPUs are designed with thread and instruction parallelization in mind, running anywhere from a handful to literally 1000s of light-weight processing threads at a time. Algorithms that are highly parallelizable and exploit such parallelism, if implemented well, are able to execute extremely fast and often hide much of the latency involved in copying to/from the GPU device.

The same can be said for many relatively simple algorithms that can be batched together to perform a final result, since the batching process can assign each light-weight GPU thread to a single process in the batch, many of which will operate concurrently exploiting thread level parallelization. Examples of such batching algorithms include many line-of-sight(LOS) tests against a height-map to generate a view-shed, many LOS tests between different viewpoints in a search space, many geometric intersection/collision tests needing to be processed within a single execution frame and referenced throughout the frame by dependent behavior routines, ray tracing algorithms with many rays that combined produce a resulting image, etc. Indeed some of these batching algorithms may run so fast as to be considered useable by runtime applications such as simulations or games.

This paper specifically shows some results of performing many line-of-sight(LOS) checks against a height-map to create a view-shed, implemented with a common GPGPU sdk for a modern GPU. Such an algorithm could be used to show visual vulnerability, visualize aesthetics, etc [3]. Additional batched LOS applications include testing a single entities viewpoint against a number of other entities on a map, testing all entities against all other entities, etc. Such applications are useful for quickly culling out sets of entities are rendered with respect to a particular entity's viewpoint or acted upon via behavioral routines. Additionally, such runtime checks can sometimes be favorable over pre-computed visibility structures if the terrain is dynamic or destructible.

## 2. IMPLEMENTATION

There are several algorithms for testing line-of-sight(LOS) across a terrain, though most of them involve testing the sight line height against the height-at-terrain(HAT) at various points along the line. Most terrain LOS algorithms contain at least this routine and derivate only by how the terrain geometry is sorted, stored and which parts are compared against the line.

### 2.1 Height-Maps

A commonly used algorithm in simulations and games, presented by Jeffrey Roberto et al., is to test LOS against a *height-map* instead of the actual geometric terrain [2]. A height-map is simple a texture, at any specified resolution, that represents the average heights at various points across the terrain. These height values are typically normalized between 0-255 so that they can fit into a single byte per pixel of the texture, reducing its size. Note that the resolution of this height-map is arbitrary, though a higher resolution height-map has more pixels and therefore more terrain samples which provide a more accurate definition of the terrain. For example, a 512x512 height-map used in this experiment, represents the heights of a "small" portion of the Grand Canyon, where each pixel in the height-map maps roughly to an actual area of 40mx40m, and the heights correspond to an actual height range between 717m-2567m.

The general description of the height-map LOS testing algorithm is as follows:

Given a height-map that represents the actual terrain used, and two locations *A* and *B* within the 3D area of the terrain bounds and above the terrain, walk along the line *AB* and compare the height at the line with the height stored in the height-map. If the sampled height at the terrain is higher than the height of the line, then it can be determined that there is no visible LOS between the two positions *A* and *B* (and vice versa from *B* to *A*). Figure 1 shows the algorithm as described by Jeffrey Roberto et al.

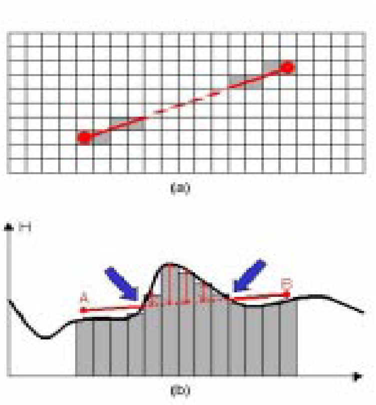

**Figure 1: Sampling height-map tests visibility [2].**

In this experiment, since a view-shed of an entire height-map is the end result of these batched LOS tests, the LOS lines are from $A$ to $B$ where $A$ is a position on the height-map denoting a specified viewpoint, and $B$ is a position on the height-map that is or is not viewable from viewpoint $A$. So the batching part of this is because viewpoint $A$ is testing LOS with every other point $B$ on the height-map, resulting in $n^2$ LOS tests, where $n$ is the width of the height-map.

## 2.2 Terrain vs. Entities

Because the height-map itself is an approximation of the terrain where a single pixel maps to a potentially much larger actual terrain region, this algorithm is also merely an approximation of LOS. More specifically, a negative LOS result could immediately be used for culling, however a positive LOS result may need to be tested further against other entities in the scene that are not represented by the height-map. Such testing is beyond the scope of this paper.

## 2.3 Bi-cubic Filtering

Depending on how interpolation occurs along a LOS segment, it is possible that multiple samples of a line may be compared against the same pixel of a height-map. Typically this does not affect the results of a given LOS test, it just wastes cycles by performing redundant comparisons (and potentially redundant gather loads on the GPU depending on implementation). However, there are instances where comparing heights along a line segment against the same height-map sample can produce varying results. For instance, if the position and slope of the actual terrain chunk is identical or very close to the LOS line segment being tested, then parts of the line segment will be under the height at terrain and parts of the line segment will be above the height at terrain. This is typically the undesirable result of aliasing. Consider Figure 2 below, if the comparison of the LOS line segment height with the sampled terrain height is made at the denoted intervals along the line (shown in the figure), then the line segment height is higher than the sampled terrain height. However, if those intervals were shifted slightly to the left (with respect to the figure), then the line segment height would be below the sampled terrain height, resulting in a negative LOS result.

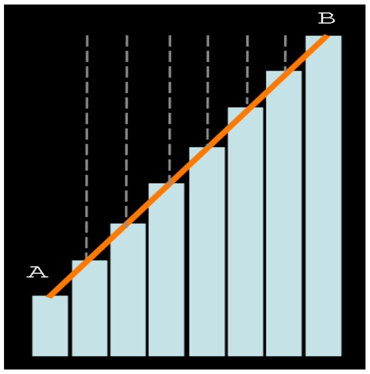

**Figure 2: Sampling period may affect results.**

In this implementation, a bi-cubic filtering step is added at the time of sampling from the height-map, which gives a much better approximation of the actual height terrain. This slightly adds to the complexity of the algorithm, but produces more accurate results. The overhead added is not considerable at all for the GPU, given its fast floating point computational ability.

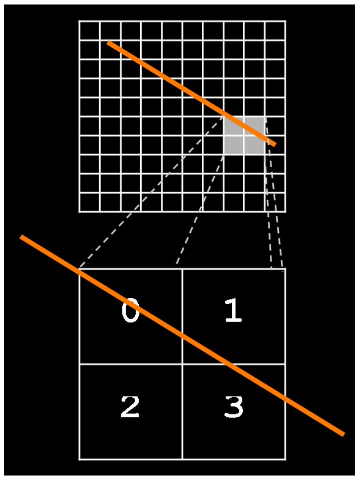

**Figure 3: Bi-cubic filtering of sampled results.**

## 3. EXPERIEMENTS & RESULTS

The LOS device implementation was written and tested with CUDA, thus for NVIDIA hardware, however the algorithm is succinct enough to be easily ported to Brook+ with minimum effort for ATI devices.

Two separate programs were written for testing, one to benchmark the LOS algorithm, written strictly in C as a simple console program to gather timings, and the other as a C++ windowed application to render the results in a 3D environment (using DirectX9). Note that both programs include the LOS device implementation functions via a header file, so the hardware implementation is identical in both programs.

The timing program was compiled and tested on a Linux box with Fedora 8 distribution with CUDA toolkit and sdk 1.1 installed, and an NVIDIA 8800GTX card with 768MB video RAM. Figure 4 and Table 1 show the timings obtained from running the LOS algorithm from one point on a height-map to every other point on the same height-map, for various size square maps with varying levels of height detail. Figure 5 and Table 2 show the average number of LOS checks possible per second. Notice that for the largest size height-map tested, the CPU results are not included since the time to complete is large.

The timings were obtained by running the program 100 times and taking an average of the times, since running time is completely dependant upon the viewing position and the height-map image itself.

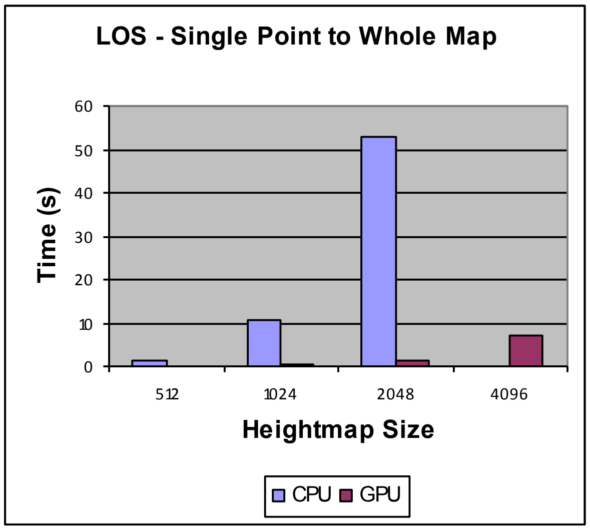

**Figure 4: Time(s) for one point to all other points.**

| | 512 | 1024 | 2048 | 4096 |
|---|---|---|---|---|
| CPU | 1.3586 | 10.753 | 52.7563 | ~ |
| GPU | 0.0444 | 0.3363 | 1.2896 | 7.2219 |

**Table 1: Exact timing results.**

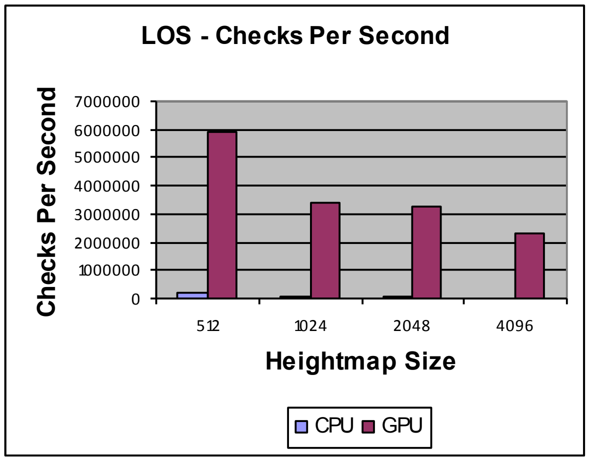

**Figure 5: Number of LOS checks per second.**

| | 512 | 1024 | 2048 | 4096 |
|---|---|---|---|---|
| CPU | 192949 | 82221 | 79503 | ~ |
| GPU | 5907152 | 3423604 | 3252478 | 2323115 |

**Table 2: Number of LOS checks per second.**

The visualization program was compiled and tested on a Windows box with Intel P4 3.2GHz processor and 2GB of system RAM, Windows XP OS, CUDA toolkit and sdk 1.1 installed, and an NVIDIA 8600GT card with 512MB video RAM. The geometry in the scene was rendered with standard DirectX9 fixed function pipeline routines. The height-map terrain itself as well as the LOS view-shed was rendered with indexed triangle list primitives (not strips).

A 128x128 height-map of a small portion of the Grand Canyon (heights ranging from 700m to 2500m and each pixel representing approximately 40mx40m of space), ran at 92 frames per second while performing the LOS test above which views roughly a quarter of the terrain every frame. Note that v-sync was disabled and there were no other entities in the scene other than the rendered height-map polygons, LOS view-shed polygons and 6 small polygons for the cursor.

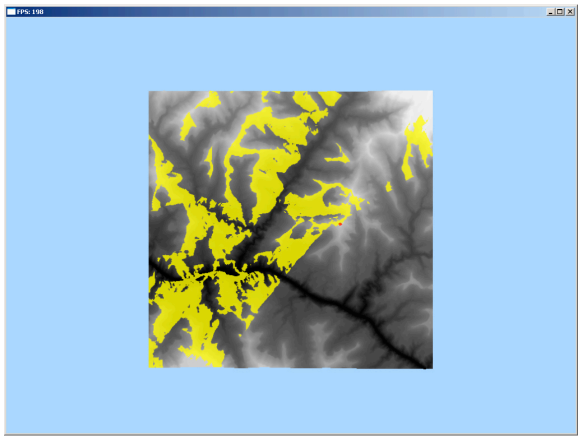

**Figure 6: Orthographic view showing rendered height-map & view-shed.**

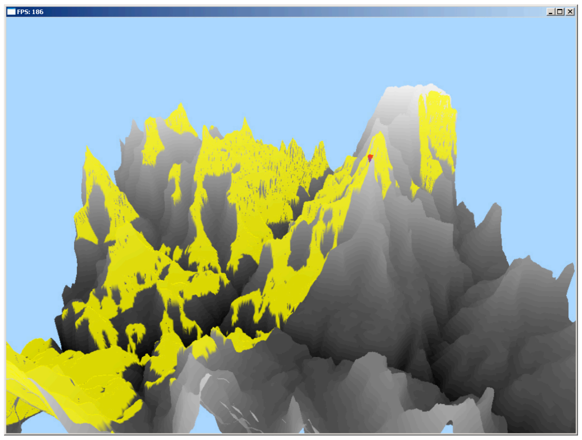

**Figure 7: Third person view showing rendered height-map, view-shed and cursor at viewpoint.**

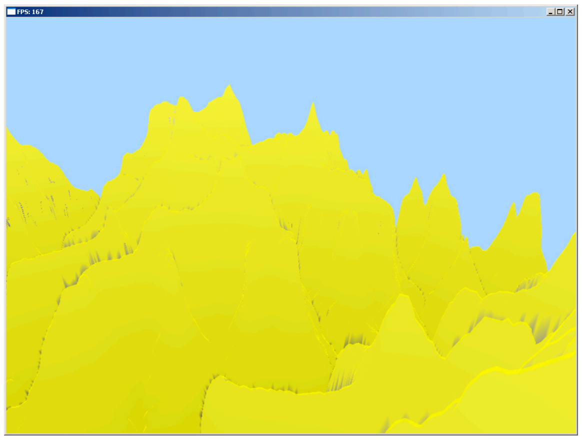

**Figure 8: Perspective view at the viewpoint.**

## 4. CONCLUSIONS & FUTURE WORK

Batching height-map LOS checks to execute concurrently is an ideal problem for massively parallel GPUs such as the NVIDIA G80 or ATI R670. Since each LOS check itself is simple in implementation, and fast because of the break-out option upon determining no LOS, and has a small memory footprint, a single LOS check can easily be adopted to fit within the resource constraints of a GPGPU environment.

This experiment shows just how quickly these checks can be computed on a GPU versus a CPU, and the quantity of checks that can be computed in a graphical real-time environment.

Future work that would complement this experiment is equivalent tests on ATI hardware, implemented with Brook+, as well as other comparable massively parallel GPUs. Additionally, more usage tests would be nice, including an implementation integrated into a more complete simulation or game with many other processing factors. Also, the GPU implementation in this experiment is naïve in the sense that there is little difference between it and the equivalent CPU implementation. Further GPU specific optimizations, including pre-fetching, vectorization (on certain GPUs), etc. are expected to show faster results.

## REFERENCES

[1] Govindaraju, N., S. Redon, M. Lin, and D. Manocha. *Accelerating Line of Sight Computation Using GPUs*, November 2005.

[2] Jeffrey Roberto de Beauclair Seixas, M.R. Mediano and M. Gattass. *Efficicient Line-of-Sight Algorithms for Real Terrain Data*, October 1999.

[3] Joseph K. Berry. Beyond Mapping III: Topic15 – Use Maps to Access Visual Vulnerability, BASIS Press 2007. http://www.innovativegis.com/basis/MapAnalysis/Topic15/Topic15.htm#Assess_visual_vulnerability

[4] USGS – The National Map Seamless Sever http://seamless.usgs.gov/website/seamless/viewer.htm

[5] Peter L. Guth. *MICRODEM* GIS mapping program, Department of Oceanography, U.S. Naval Academy. http://www.usna.edu/Users/oceano/pguth/website/microdem.htm