PARALLEL APPROXIMATE VIEWSHED COMPUTATION

Christoph Fünfzig
Le2i, University of Burgundy
Dijon, France
c.fuenfzig@gmx.de

ABSTRACT

Visibility computation on heightfield grids is important for geo information and simulation tasks like signal propagation and flight surveillance. We describe a new parallel primitive to compute an approximate viewshed inside a ray cone given by two point-vector pairs. For a given number of points along the edge, it computes the largest view heights on line segments parallel to the given edge. The primitive is an approximation, as it happens that the selected ray segments are not in a common plane. We analyze and compare the primitive's performance on different OpenCL devices in terms of sampling resolution on the edge and orthogonal to the edge. In applications, the primitive can be used to compute the approximate viewshed of a polygon or candidate viewpoints for covering the polygon.

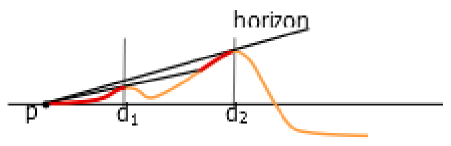
KEYWORDS

digital elevation model, DEM, viewshed, geometric approximation.

1. INTRODUCTION

Digital terrain models (DTM) can be represented in two different forms, as a triangulated irregular network (TIN) and as a regular height grid (DEM). With the acquisition by airborne laser altimetry, or stereo-photogrametry, especially the latter are available with good coverage in high resolutions. Many simulation and planning problems for electromagnetic wave propagation are based on the ray tracing model with linear rays of geometric optics. For planning applications also geometric information is required, i.e., the region visible from a view point, the terrain coverage.
Computing the visible horizon for a given grid point was considered in [Max 1988], which is computing the maximum angle or highest terrain point of an unblocked ray for each possible ray direction. The resulting horizons for each grid point are used for attenuating the incident light due to self-shadowing of the terrain. An extension is viewshed computation, which determines the region (Figure 1, bottom), for which an unblocked ray from the viewpoint exists. It computes all visible cells for each ray direction not just the upmost one before the sky, as shown in Figure 1, top.

Overview. We present and analyze an efficient parallel primitive (Figure 2), which allows to approximate the viewshed from a conical region defined by two points with associated vectors. The primitive allows to compute an approximation of the viewshed from a star-shaped polygon on the terrain by using the primitive for each polygon edge and direction vectors from a central point (Figure 3). Concerning visibility, observer and observed points can be exchanged. With the same primitive, it is possible to compute a set of viewpoints for covering the polygon.



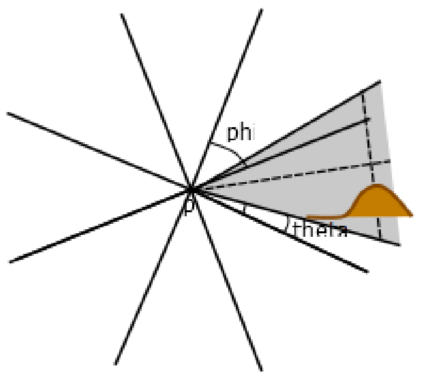


Fig. 1. Viewshed from a single point *p*.

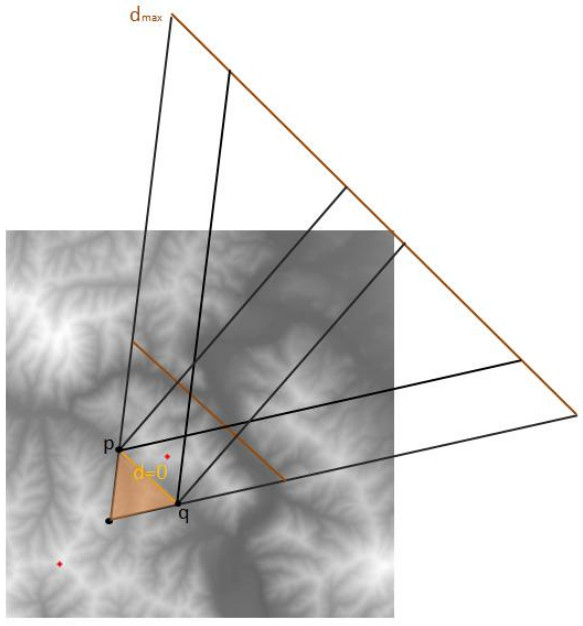


Fig. 2. Primitive defined by two point-vector pairs (p, r_p)(q, r_q) defining a planar cone.

After a short look at previous work in Section 2, we present the new parallel primitive and its OpenCL implementation in Section 3. It works on terrain tiles stored in processor memory. Section 4 gives results of our OpenCL implementation and before conclusions in Section 5.

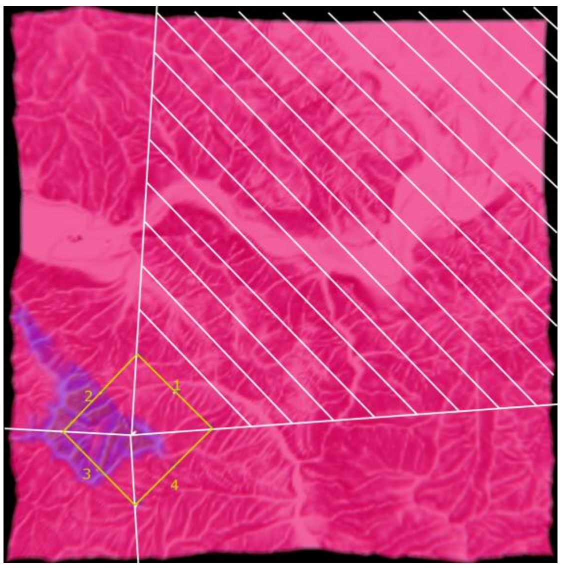


Fig. 3. Application of primitive to quadrangle. Each line segment is sampled in a given resolution.

2. PREVIOUS WORK

Special data structures like trees and index lists can be used for sequential ray traversal of grids [Fuenfzig 2007].
For parallel ray traversal, there are two different ways. Firstly, the computation for a single ray can be split in a fine grained way based on columns (for mostly horizontal rays) or rows (for mostly vertical rays). In this splitting scheme, an intersection is calculated with a cell in a column given by the thread index. The first or last intersection parameter then must be extracted from the array of intersection parameters using parallel reduction, see [Blelloch 1993]. Secondly, different rays can be traversed in parallel. Each thread traverses the terrain cells along the ray in a loop like the sequential algorithm does. This is the common organization of raytracing a grid with many threads [Aila 2012][Schiffer 2013]. In [Aila 2009][Aila 2012], the optimal scheduling of rays and ray intersection on NVidia GPUs is considered in detail.
In [Strnad 2011], the parallel viewshed computation in CUDA is extended from single points to sets like a scattered group of points, a path, or a rectangle. This is achieved by ray determination followed by ray traversal adding to the viewshed. As in [Fuenfzig 2007], nearest and bilinear reconstruction are considered. Furthermore, the viewsheds can be combined with logical operations.
For the new parallel primitive in this work, we need to traverse all rays across an edge with directions in a given cone, as shown in Figure 4. A parameterization of such rays was called the two-line parameterization for lightfields [Levoy 1996].

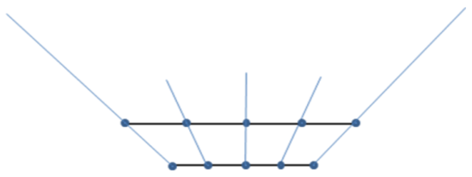


Fig. 4. Lightfield defined by two parallel line segments.

3. PRIMITIVE FOR APPROXIMATE VIEWSHED COMPUTATION

In this section, we give details of the approximate viewshed primitive. For each point p_i, i=1,...,n, on the line segment pq and given directions r_p, r_q, the cone $c(s,d)=p_i+d(sr_p+(1-s)r_q)$, $d\in\mathrm{R}^{\geq 0}$, $s\in[0,1]$ is traversed on offset line segments $c(s,d)$, $d=d^*$. The directions r_p and r_q are scaled such that $r_p\cdot(q-p)^\perp=1$ and $r_q\cdot(q-p)^\perp=1$. Note that the line segments $c_d(s):=c(s,d)$, $s\in[0,1]$ for different cone origins p_i are collinear and overlap.

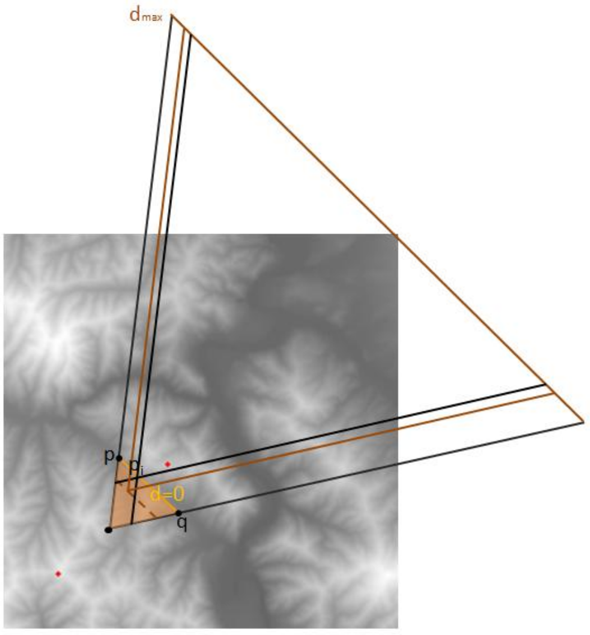


Fig. 5. Traversal of planar cone along offset line segments in the inside region (dashed) and in the outside region (normal).

In a sequential version, it is sufficient to traverse the cone with origin given by $p+ur_p$, $q+vr_q$ for $u,v\in\mathrm{R}^{\geq 0}$ to visit all cone points once. Altogether the number of visited points is the number of cone points. In the inside region with $d<0$, a lowest point is searched, and the offset lines with $d>0$ are traversed from near $d=d_{\min}\leq 0$ to distant $d=d_{\max}$. Note that $d_{\min}$ is cut off by a user-specified value, and $d_{\max}$ is bounded by the terrain extent. For each point p_i, i=1,...,n, we store an sequence of visible points as their height value *height[i,d]* and parameter *dist[i,d]:=s*.

The kernel performs two nested loops, the first for t and the second for all points p on the line segment p_i+dr_p, p_i+dr_q. We compute the height $z_{i,t}$ for each point $p_{i,t}$ and select a point $p_{i,t}$, such that $(z_{i,t}-z_{i,t-1})(\mathrm{a}(p_{i,t-2}, p_{i,t-1}, p_{i,t}))$ is maximum. Herein, the function $\mathrm{a}(r_1, r_2, r_3):=(r_2-r_1)(r_3-r_2)\ )/|r_2-r_1||r_3-r_2|$ denotes the cosine of the enclosed angle of successive ray segments. Finally, we store the selected point sequence of increasing values *height[i,d]* and corresponding parameter *dist[i,d]*.

4. PERFORMANCE IN OPENCL

We have implemented the viewshed primitive using OpenCL 1.1 without any modifications for the different devices. We refer the reader to [Fang 2011] for an overview of different optimizations especially for GPU devices.
The terrain dataset is uploaded once as a read-only array of single-precision floating point height values. The output arrays *height* and *dist* are retrieved as pinned memory for the Intel devices [Shen 2013]. Data transfer times are not significant for the small input/output sizes here. Figure 6 shows a color-coding of the viewshed

results for different queries. Additionally in the bottom row, the angle cosines of successive ray segments are shown.

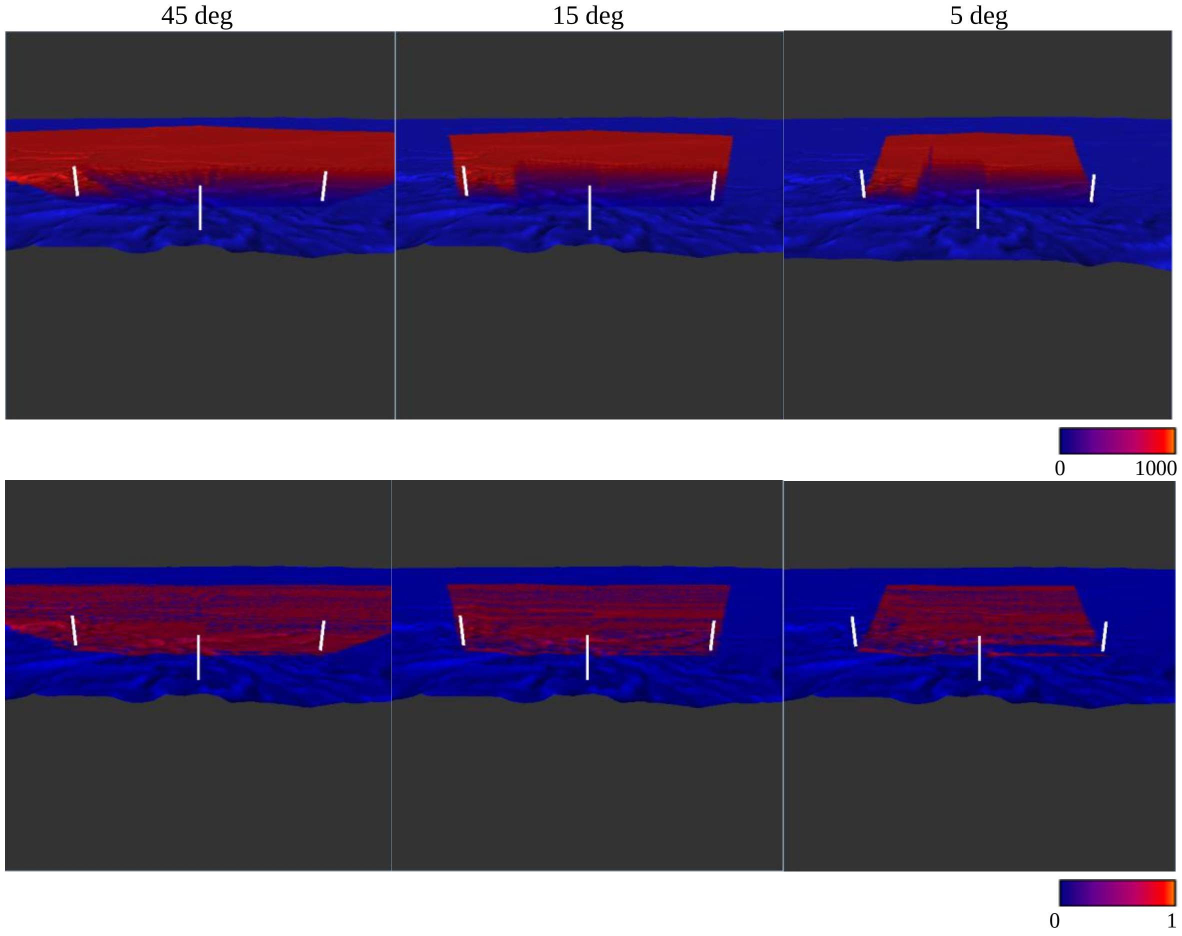


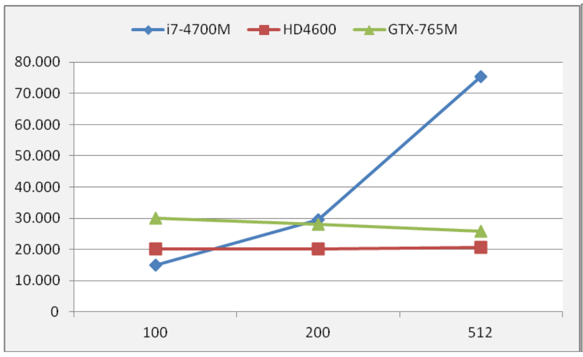
Fig. 6. Approximate viewsheds defined by two point-vector pairs on Utah terrain (3080*3582 points): In the top row, heights of line-of-sights from lowest terrain points outward are color-coded. In the bottom row, the scalar product values a(r_1, r_2, r_3) are depicted.

4.1 Devices

In our comparison, we use three devices listed in Table 2. The CPU device is an Intel i7-4700 MQ mobile processor with 2.4GHz and 8GB DDR3-RAM. Integrated is a GPU device Intel HD4600 with 800MHz and access to the CPU memory system. Finally, the dedicated GPU device is an NVidia Geforce GTX-765M with 862MHz core clock and 2GB DDR5-RAM.

	100 × 50	200 × 50	512 × 50	100 × 200	200 × 200	512 × 200
i7-4700M (buffer)	0.734,13.972	0.807,27.684	1.154, 70.633	1.159,14.979	1.084,29.546	1.027,75.274
HD 4600 (buffer)	1.365,19.371	1.500,19.361	1.018,19.675	0.927,20.064	1.394,20.266	1.455,20.562
GTX-765M (buffer)	1.479,28.924	1.170,27.118	1.669,24.721	2.040,29.948	1.237,28.142	1.452,25.944

i7-4700M (image)	0.784,22.581	0.656,45.008	0.696,114.851	0.720,24.053	0.761,47.697	0.659,122.060
HD 4600 (image)	0.769,19.387	0.675,19.555	0.732,19.861	0.738,19.998	0.734,20.249	0.641,20.682
GTX-765M (image)	1.674,27.245	1.649,27.403	1.578,27.639	1.659,28.241	1.589,28.354	1.675,28.487



buffer object

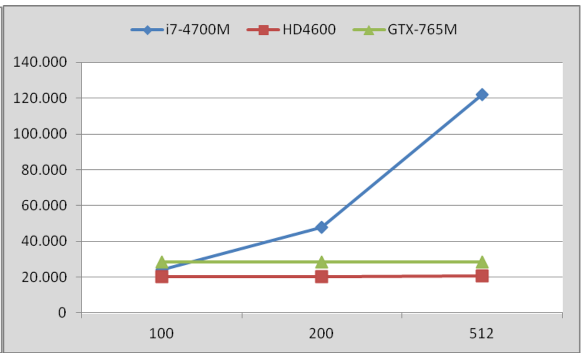


image object

Table1. Execution times (query upload, computation in ms) for different viewshed resolutions.

Table 1 gives the upload time for the input data and the kernel computation time (in milliseconds). It contains results for viewshed resolutions 100, 200, 512 (maximum work group size available on Intel HD4600) and 50, 200 view samples. In this way, the whole computation for an edge can be performed by one kernel invocation. The first three rows give the times when using a buffer object storing the terrain heights. Then follow the times for an image object of format CL_R|CL_FLOAT with the terrain heights.

	i7-4700M (Haswell)	HD 4600 (Haswell)	NV GTX-765M (Kepler)
OpenCL driver	3.0.1.10878 (CL 1.2)	10.18.10.3355 (CL 1.2)	327.62 (CL 1.1)
processor/memory clock (MHz)	2400/2800 DDR3	800/2800 DDR3	862/4000 DDR5
#proc-elements	8	140	768 (4 SMX)
#comp-units	4	20	96
#workgroups	1024	512	1024
#workitems	1024	512	1024

Table 2. Device capabilities according to *clGetDeviceInfo* [Fang 2011][Junkins 2014].

The OpenCL kernel consists of two successive loops, where the first traverses the cone's inside region, and the second traverses the cone's outside region, which writes the visible view heights to the output array in global memory. In our experiments, the storage of the terrain dataset as buffer was always faster than the one-component image. Nevertheless, a special sampler for the bilinear reconstruction of image values can be selected by flags. For the NVidia device, special care with image creation & upload is necessary to achieve the same performance as with a buffer object (as mentioned on the khronos mailinglist).

The iCore7 device shows increasing computation time with increasing resolution and number of view samples. For the GPU devices, the measured kernel time, is roughly the same for different resolution and number of samples, which hints at good load balancing. The GPU devices are fastest for resolution *r*=512 due to a sufficient number of thread units. Note that we used global memory for the height array throughout, from which large and arbitrarily shaped sections are accessed. We think that a sufficient number of thread units and direct access to the height array in system memory is the reason for the superior performance of the integrated HD4600 GPU device in the benchmark.

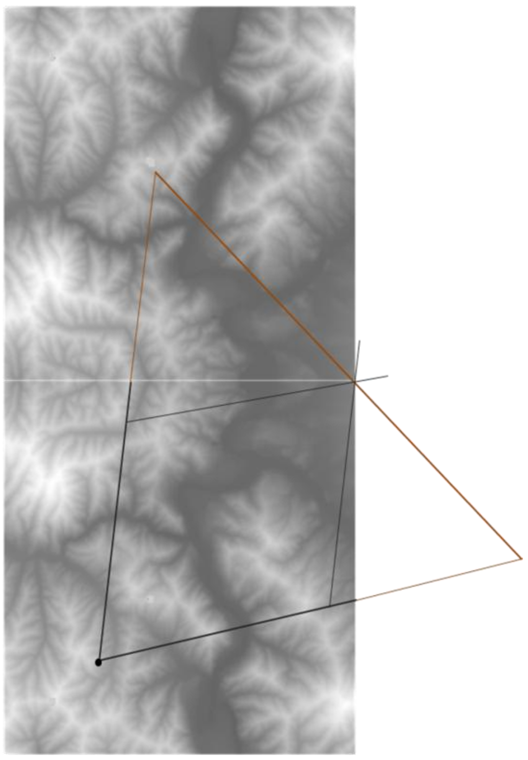


Fig. 7. Queries reaching to neighbor tiles.

5. CONCLUSION

In this article, we presented a new parallel primitive to approximately compute the viewshed across an edge of a polygonal region. It bundles the queries from all terrain points inside the cone defined by two point-vector pairs. We implemented the primitive using OpenCL, where the DEM grid is transferred as array of single-precision floating-point values.

In our comparison, we have used three devices, a CPU device with an integrated GPU device and a dedicated GPU device. The primitive is significantly faster than using a traversal, where each ray is encoded separately in the input. It can be employed on several polygon edges in parallel, occupying one work group per edge. For output, the kernel fills an array of size *resolution*×*num*, where *num* is the number of view samples. For tiled terrains, the query can run across the tile's border as shown in Figure 7. Note that the viewsheds on the tile's border line need to be combined into the first query result inside the brown-marked region.

ACKNOWLEDGMENT

The Utah SRTM dataset (1arcsec/30m grid) is freely available from the USGS.

REFERENCES

T. Aila, S. Laine, T. Karras 2009: *Understanding the Efficiency of Ray Traversal on GPUs*, High Performance Graphics 2009.

T. Aila, S. Laine, T. Karras 2012: *Understanding the Efficiency of Ray Traversal on GPUs - Kepler and Fermi Addendum*, Poster High-Performance Graphics, June 2012.

G.E. Blelloch 1993: *Prefix sums and their applications*, Carnegie Mellon University, Technical Report 1993.

Ch. Fuenfzig, T. Ullrich, D.W. Fellner, Ed Bachelder 2007: *Empirical Comparison of Data Structures for Line-Of-Sight Computation*, IEEE Workshop on Intelligent Signal Processing, Madrid, Spain, Oct 2007.

S. Junkins 2014: *The Compute Architecture of Intel® Processor Graphics Gen7.5*, Intel, Aug 2014.

M. Levoy, P. Hanrahan 1996: *Light field rendering*, Proc. ACM SIGGRAPH 1996, pp.31–42.

N.L. Max 1988: *Horizon mapping: shadows for bump-mapped surfaces*, The Visual Computer, vol. 4, no. 2, March 1988, pp.109–117.

D. Strnad 2011: *Parallel terrain visibility calculation on the graphics processing unit*, Concurrency and Computation Practice and Experience, vol. 1, no. 23, 2011, pp. 2452-2462.

T. Schiffer, D. Fellner 2013: *Raytracing – Lessons learned and future challenges*, IEEE Potentials, vol. 32, no. 5, 2013, pp. 34–37.

J. Fang, A.L. Varbanescu, H. Sips 2011: *Comprehensive Performance Comparison of CUDA and OpenCL*, Proceedings of the 2011 International Conference on Parallel Processing (ICPP '11), pp. 216-225.

J. Shen, J. Fang, H. Sips, A.L. Varbanescu 2013: *Performance traps in OpenCL for CPUs*, Proceedings of the 21st Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP'13), Washington DC, USA, 2013, pp.38–45.