Computing approximate horizons on a GPU

Wenli Li, W. Randolph Franklin, Salles V. G. Magalhães
Rensselaer Polytechnic Institute, Troy, NY, USA
liw9@rpi.edu, mail@wrfranklin.org, vianas2@rpi.edu

ABSTRACT
We use an $O(n \log(n))$ quadtree-forest algorithm to compute approximate horizons at all points of a DEM, and achieve more than 30 times speedup on a GPU. The result of the algorithm is very close to that of the brute-force algorithm.

1. INTRODUCTION
Given a terrain represented as a Digital Elevation Map (DEM), the horizon at a point is the largest elevation angle to another point in all directions. To save time and space, the horizon can be approximated as a constant value in each of a number of sectors. Approximate horizons at all points of a terrain are useful in shading, visibility, and solar irradiation applications. For example, they can be used to render a shaded terrain with any sky radiance function [3]. They can be used to compute the occlusion regions of terrain quads for visibility culling [2]. And they can be used to estimate the global solar irradiance at all points of the terrain [4].

Stewart [3] proposed an $O(sn \log^2(n))$ algorithm to compute approximate horizons at all points of a terrain, where s is the number of horizon sectors and n is the number of terrain points. The algorithm approximates the horizon of a point by the highest point in each sector, and computes the horizons of all points sector by sector. It is parallel for the sectors and sequential for each sector, and not ideal for massively-parallel architectures like the GPU. Tabik et al. [4] proposed a parallel horizon algorithm for large terrains. It divides a terrain into blocks, computes 'near' horizons for each block using Stewart's algorithm, and computes 'far' horizons on a lower-resolution terrain using Stewart's algorithm. The blocks can be processed on multiple compute nodes and the sectors of a block can be processed on multiple compute cores. How to process a block on the GPU remains a problem.

This paper uses the brute-force algorithm and a quadtree-forest algorithm to compute approximate horizons on a GPU. The second algorithm is inspired by the Barnes-Hut algorithm for N-body simulation [1]. The algorithm divides the space in an octree such that each leaf node contains at most one body, and stores in each internal node the center of mass and total mass of the group of bodies. To approximate the gravitational force on a body, it recursively traverses the tree from the root. Let w be the width of an internal node and d be the distance between the body and the node's center of mass. If w/d is less than a threshold θ, the node is treated as a single body, otherwise the children of the node are examined. It takes $O(\log n)$ time to compute the force on one body and $O(n \log n)$ time on all bodies.

2. BRUTE-FORCE ALGORITHM
Stewart defined the approximate horizon as follows: At each point, divide the terrain into s sectors ($[0, 2\pi/s]$, $[2\pi/s, 4\pi/s]$, ...) and project the points in each sector onto a vertical plane through the bisector. The approximate horizon of the point in a sector is the largest elevation angle from the point to a projected point on the vertical plane. Therefore, the approximate horizon is constant in each sector. Stewart showed it is better to consider all points in a sector than points in a single direction for terrains with sharp spikes.

The brute-force algorithm computes the horizon of each point using each other point. The algorithm is slow but GPU-friendly, because each horizon is computed independently. We define the sectors as $[0, 2\pi/s)$, $[2\pi/s, 4\pi/s)$, ... and do not project the points in a sector onto a vertical plane, which is required for Stewart's algorithm but not for the brute-force algorithm. If the sectors are narrow, they may not contain a point until some distance, causing the horizon to be underestimated. Stewart solved the problem by checking a fixed number of points bordering a sector near the sector vertex. The number of bordering points is about s/π ($s/2\pi$ on one side). We deal with the problem by checking $s/2\pi$ points whose cells are intersected by the bisector. Only boundary points have undefined horizons, which can be set to 0 for rendering. The complexity of the algorithm is $O(n^2)$, where n is the number of points.

Algorithm 1: Brute-force algorithm

```
Data: a DEM
Result: approximate horizons at all points
foreach point $p$ do
    foreach sector $s$ do
        foreach point $q$ of a fixed number of points along the bisector of $s$ do
            update the horizon of $p$ in $s$ using $q$;
    foreach point $q$ do
        find the sector $s$ of $p$ containing $q$;
        update the horizon of $p$ in $s$ using $q$;
```

3. QUADTREE-FOREST ALGORITHM
The quadtree-forest algorithm uses quadtrees to approximate individual points like the 2D Barnes-Hut algorithm. The algorithm divides the terrain into blocks and constructs a largest-value quadtree for each block. To compute the horizon of a point, it recursively traverses each quadtree and visits the children of an internal node if $w/d > \theta$. A fixed-sized stack is used to simulate recursion on the GPU. The reason for using a quadtree forest instead of a quadtree is twofold: the first few levels of a quadtree would not be used as points, and a higher tree would require a larger stack and more stack operations. The algorithm builds the quadtrees in $O(n)$ time and computes the horizons in $O(n \log(n))$ time.

4. RESULTS
The algorithms are implemented in C++ as sequential programs on the CPU and CUDA programs on the GPU. The hardware includes an Intel Xeon E5-2660 v4 CPU and an NVIDIA GeForce GTX 1080 GPU. A 1024×1024 DEM is used as the data, and the result horizons have 64 sectors. Ta-

Algorithm 2: Quadtree-forest algorithm

```
Data: a DEM
Result: approximate horizons at all points
divide the DEM into blocks and create a largest-value
quadtree for each block;
foreach point $p$ do
    foreach sector $s$ do
        foreach point $q$ of a fixed number of points along the bisector of $s$ do
            update the horizon of $p$ in $s$ using $q$;
    foreach quadtree $t$ do
        push the root of $t$ onto a stack;
        while the stack is not empty do
            pop a node $n$ from the stack;
            foreach child $c$ of $n$ do
                if $c$ is not a leaf and $w/d > \theta$ then
                    push $c$ onto the stack;
                else
                    find the sector $s$ of $p$ containing the center of $c$;
                    update the horizon of $p$ in $s$ using the center of $c$;
```

Table 1: θ, running time (seconds), and accuracy (radians) of the quadtree-forest algorithm. Accuracy is the average absolute difference between the result and the result of the brute-force algorithm.

θ	Sequential time	CUDA time	Accuracy
0.2	125	3.82	0.012
0.1	334	9.35	0.005
0.05	997	27.9	0.002

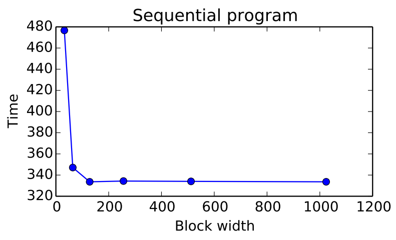
ble 1 shows θ and the running time (excluding file I/O) and accuracy of the quadtree-forest algorithm. When $\theta = 0.1$, the average absolute difference between the result and that of the brute-force algorithm is only 0.005 radians. Figure 1 shows block width (32, 64, ..., 1024) and the running time (excluding file I/O) of the quadtree-forest algorithm. The sequential program does not benefit from a quadtree-forest, but the CUDA program runs faster as block width decreases to 64. Table 2 shows the running time (excluding file I/O) and relative speedup of the brute-force algorithm and the quadtree-forest algorithm ($\theta = 0.1$ and block width = 1024 for the sequential program or 64 for the CUDA program). The quadtree-forest algorithm is significantly faster on both the CPU and the GPU. Figure 2 shows the relative area of the visible sky, calculated from the approximate horizon, at all points of the DEM.

5. CONCLUSIONS

We use a quadtree-forest algorithm to compute approximate horizons at all points of a DEM. The algorithm is not only faster but also more suitable for the GPU than some existing algorithms. We will look for alternative algorithms and applications of the approximate horizon.

Table 2: Running time (seconds) and relative speedup of the algorithms.

Algorithm	Sequential time	CUDA time	Speedup
Brute force	55278	984	56.1
Quadtree forest	334	9.34	35.7



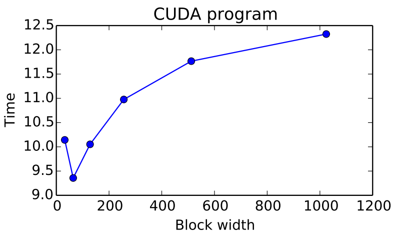


Figure 1: Block width and running time (seconds) of the quadtree-forest algorithm.

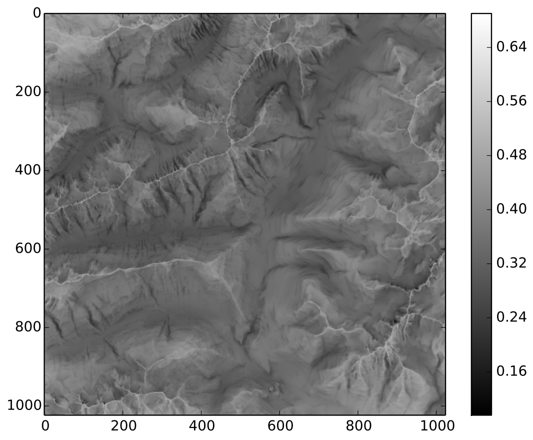


Figure 2: Relative area of the visible sky.

Acknowledgement. This research was partially supported by NSF grant IIS-1117277 and CAPES (Ciência sem Fronteiras).

6. REFERENCES

[1] J. Barnes and P. Hut. A hierarchical *O(N log N)* force-calculation algorithm. *Nature*, 324:446–449, Dec. 1986.

[2] A. J. Stewart. Hierarchical visibility in terrains. In J. Dorsey and P. Slusallek, editors, *Rendering Techniques '97: Proceedings of the Eurographics Workshop in St. Etienne, France, June 16–18, 1997*, pages 217–228. Springer Vienna, Vienna, 1997.

[3] A. J. Stewart. Fast horizon computation at all points of a terrain with visibility and shading applications. *IEEE Transactions on Visualization and Computer Graphics*, 4(1):82–93, Mar. 1998.

[4] S. Tabik, L. F. Romero, and E. L. Zapata. High-performance three-horizon composition algorithm for large-scale terrains. *International Journal of Geographical Information Science*, 25(4):541–555, Apr. 2011.