A FASTER SOLUTION TO THE HORIZON PROBLEM

J. DOZIER and J. BRUNO
Computer Systems Laboratory, University of California, Santa Barbara, CA 93106, U.S.A.

and

P. DOWNEY
Department of Computer Science, University of Arizona, Tucson, AZ 85721, U.S.A.

(Received 13 November 1979; revised 27 September 1980)

Abstract—An algorithm is developed for calculating the horizons for each point in a digital terrain grid in order \( N \) iterations, whereas all previous methods seem to be of \( O(N^2) \) time complexity. The new method makes horizon computations reasonable, and ought to improve the accuracy of surface climate models in rugged terrain.

Key Words: Terrain analysis, Solar radiation.

INTRODUCTION

In order to calculate incoming solar radiation over a digitized topographic grid, it is helpful to be able to determine local horizon information from the gridded data, to ascertain whether a given location at a given sun position is shaded from direct sunlight by surrounding terrain. Moreover, the horizon information is useful in calculations of atmospheric (longwave) radiation and diffuse solar radiation, because, at any location, the portion of the overlying hemisphere which is obscured by terrain is

\[
\frac{1}{2\pi} \int_{0}^{2\pi} \sin(h[\theta])d\theta
\]

where \( h[\theta] \) is the horizon angle in the direction \( \theta \).

These horizons, however, are difficult to compute, for unlike slope and azimuth, they cannot be calculated from information restricted to the immediate neighborhood of a point. For this reason most, studies of the surface energy budget over terrain grids have ignored the horizon problem. Those investigators who have calculated horizons have found that the procedures were computationally expensive and inconvenient. The new algorithm presented in this paper ought to remedy this situation.

BACKGROUND AND PREVIOUS APPROACHES

The most important use of an efficient horizon algorithm would be in the computation of the surface energy budget over digitized elevation grids. Such calculations are likely to become increasingly frequent, due to the availability of surface temperature data from remote sensing and of "Digital Terrain Tapes" from the National Cartographic Information Center, U.S. Geological Survey.

By comparing the surface temperatures predicted by an energy budget model with measured temperatures, one can derive sometimes information about surface thermal properties. Possible applications of such data/model combinations include mapping of surface geology (Pohn, Offield, and Watson, 1974; Watson, 1975; Kahle, 1976), surface soil moisture detection (Idso and others, 1975b; Rosema, 1975), and geothermal prospecting (LeSchack and Del Grande, 1976). In addition, calculations of solar radiation can be used for remote albedo measurements (Otterman and Fraser, 1976) and hence surface soil moisture (Idso and others, 1975a), or for automatic registration of satellite data to terrain grids (Horn and Bachman, 1978).

All of the papers cited previously have ignored horizons. Whereas for some types of terrain this would not lead to significant errors in energy budget calculations, it is obvious also that for rugged terrain, local horizons play an important role in surface energy budget. For example, manual measurement of surrounding horizon angles is a standard procedure for microclimate studies at single locations (for typical illustrations, see Brazel and Outcalt, 1973). For calculations of solar radiation over an area, we know of only four investigations which utilized horizon information.

Garnier and Ohmura (1968), in their mapping of solar radiation in the vicinity of Mont St. Hilaire, Quebec, seemingly determined manually horizon information from a topographic map, and then transferred these data into grid format. They also measured slopes and exposures manually. Williams, Barry, and Andrews (1972) designed a computer algorithm to calculate horizons for solar radiation studies in the alpine glacial terrain on Baffin Island, and Obled and Harder (1979) describe a method devised by Lacomme-Lahourguette which has been used for energy budget snowmelt studies in the French Alps. These two algorithms are similar, in that they consider the elevation at each of the grid points, and search along the line of the sun's azimuth for points high enough to block the sun. The methods are of order \( N^2 \) time complexity because, for all directions, every point is compared to every other point. An alternative, expressly two-dimensional method, derived by Dozier and Outcalt (1979), calculates the horizon values for the entire circle for every point in a single pass. The method remains \( O(N^2) \) because every point is compared to every other point, but it has an improvement to the
Williams and Lacomme-Lahourguette methods for energy budget studies because the time complexity is independent of the number of directions considered. Moreover, the horizons for a given terrain grid, once calculated, could be stored for future use.

**DESCRIPTION OF THE NEW ALGORITHM**

The method of calculating horizons presented in this section is of $O(N)$ time complexity. Thus it represents a substantial improvement to previously described approaches and thereby makes calculation of the horizons a reasonable undertaking. We first describe slow and fast one-dimensional algorithms and then show how to extended the one-dimensional situation to two dimensions.

We achieve these considerable computational economies through re-casting the problem in a somewhat ill-posed way. Our interest is the angle to the horizon from any point in any direction, but we formulate the problem by determining the coordinates of the points which form the horizons. Minor errors in the elevation grid can shift therefore the “answer”, the coordinates of the horizon point, by considerable distance, but minor errors do not cause much variation in the end result, which is the angle to the horizon.

**One dimension**

A nonnegative altitude function $A$ is defined on the points $0, 1, \ldots, N - 1$. The abscissa is specified by a distance function $D$ where $D(j)$ is the distance from point $j$ to some arbitrary origin and $D(j) > D(i)$ for $j > i$. The problem is to calculate two integer-valued functions, termed horizon functions, defined on the same points. The horizon function in the forward direction $H_f(i)$ satisfies:

1. For all $0 < i < N$, $0 < H_f(i) < i$ (hence $H_f(N-1) = N-1$);
2. For all $0 < i < N$, if $A(i) > A(j)$ for all $0 < j < i$, then $H_f(i) = i$ (i.e. if the altitude is higher than or equal to any other point in the forward direction, it is its own horizon);
3. For all $0 < i < N$, if $k$ is the largest value less than $i$ and greater than or equal to 0 such that $A(k) > A(i)$ and $|D(j) - D(i)| < |A(k) - A(i)|$ for all $0 < j < i$, $j \neq k$, then $H_f(i) = k$ (i.e. if two points in the forward direction are equally suitable as horizon candidates, the farthest is chosen).

Figure 1 gives an example of the one-dimensional horizon problem, with the following forward and backward horizon functions:

<table>
<thead>
<tr>
<th>$i$</th>
<th>$H_f(i)$</th>
<th>$H_b(i)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>11</td>
<td>6</td>
</tr>
<tr>
<td>8</td>
<td>11</td>
<td>6</td>
</tr>
<tr>
<td>9</td>
<td>10</td>
<td>6</td>
</tr>
<tr>
<td>10</td>
<td>11</td>
<td>6</td>
</tr>
<tr>
<td>11</td>
<td>11</td>
<td>6</td>
</tr>
<tr>
<td>12</td>
<td>13</td>
<td>11</td>
</tr>
<tr>
<td>13</td>
<td>13</td>
<td>11</td>
</tr>
<tr>
<td>14</td>
<td>15</td>
<td>11</td>
</tr>
<tr>
<td>15</td>
<td>15</td>
<td>11</td>
</tr>
</tbody>
</table>

The function $slope(i, j)$ for $0 < i < N$ and $0 < j < N$ is defined to be zero if $A(j) < A(i)$ and \[slope(i, j) = \frac{|D(j) - D(i)|}{|A(k) - A(i)|} \] otherwise. If $A(j) > A(i)$ then $slope(i, j)$ is the tangent of the acute angle that the line passing through points $i$ and $j$ makes with the horizontal axis, provided that the units for $D$ are the same as those for $A$.

Algorithm 1 in the Appendix correctly computes the horizon function in the forward direction for $N \geq 1$. This algorithm has time complexity $O(N^2)$. The reader is encouraged to try it on the data set in Figure 1.

A more efficient algorithm can be obtained by noting that, in testing whether $j$ is the horizon of point $i$, if $slope(i, j) > slope(i, H_f(i))$ then all of the points forward
of \( j \) need not be checked because clearly they are not "visible" from point \( i \), and \( H_{ij} = j \). Moreover, if \( \text{slope}(i,j) < \text{slope}(i,H_{ij}) \) then the points \( j, j + 1, \ldots, H_{ij} - 1 \) cannot be horizon points for \( i \) because point \( H_{ij} \) is "visible" from \( i \) under these circumstances. Finally, if \( \text{slope}(i,j) = \text{slope}(i,H_{ij}) \), then \( H_{ij} = H_{ij} \).

Algorithm 2 in the Appendix employs these economies to calculate the horizon function in the forward direction for \( N \gg 1 \).

It might not be obvious at first that this algorithm runs in \( O(N) \) time because we have two nested while clauses and the outer one is executed \( N - 1 \) times. Some thought reveals that the running time is proportional to the total number of times lines 7 and 9 are executed. Clearly line 9 is executed \( N - 1 \) times, once for each value of \( i \). A point \( k \) is "passed over" if line 7 is executed with the value of \( j \) set to \( k \), and if the conditional statement in line 7 turns out to be true. Subsequently \( k \) can no longer be a candidate for the horizon for points less than or equal to \( i \). It follows that each of the \( N \) points can be "passed over" only once. Therefore, the total number of times line 7 is executed is \( \leq 2N \). The improvement to the slower algorithm is dramatic: with a random altitude function and \( N = 32 \), the slower algorithm required 496 iterations, the faster algorithm only 59; for \( N = 2048 \), the slower algorithm required 2,096,128 iterations, the faster algorithm only 4081.

It is straightforward to modify the fast algorithm to compute the horizon function in the backward direction. The lines in Algorithm 2 which must be changed are given in Algorithm 2.1 in the Appendix.

Two dimensions

For the two-dimensional situation, the altitude function is defined over a regularly spaced \((N_1 + N_2 + 1)\) by \((M_1 + M_2 + 1)\) grid. Grid coordinates \((x,y)\) are integer-valued with \(-N_1 \leq x \leq N_2\) and \(-M_1 \leq y \leq M_2\), where \(\Delta x = \Delta y = 1\). The origin \((0,0)\) is located as close as possible (given the integer spacing) to the center of the grid. The objective is to compute a horizon function, which is defined on the same points: for any point in any direction \(\theta\), the horizon function is the \((x,y)\) coordinate of the point which forms the horizon in that direction.

To calculate such a function for all points, \((J_1 + J_2 + 1)\) profiles are passed through the grid. These profiles are parallel straight lines which intersect perpendicularly the \(x'\)-axis where \(-J_1 \leq x' \leq J_2\). Spacing between the profiles is 1, the same as the grid spacing, and the grid is rotated by an angle \(\theta\) (ccw) with respect to the profiles (Figure 2).

Let \((x,y)\) be a grid point before rotation. Then

\[
x' = x \cos \theta - y \sin \theta
\]
\[
y' = x \sin \theta + y \cos \theta
\]

are the grid coordinates after rotation. \(x'\) is rounded to its nearest integer value to determine which profile the grid point will fall upon, and \(y'\) (not rounded) is the distance along that profile from the arbitrary origin. If for a given \(x'\), the elevations are sorted into order with respect to their \(y'\) values, then the problem is reduced to its one-dimensional equivalent. Furthermore, because we can determine for any profile the horizons in both forward and backward directions, the grid can be rotated over a discrete set of \(0 \leq \theta < \pi\) to obtain the horizons over the entire circle. For the elevations of the points along the profiles, the value at the shifted point is used, rather than interpolating to determine the elevation at \((x', y')\) where \(x'\) is not rounded. In our experience, available digital terrain grids are smooth enough that interpolation would not change the results. Avoiding interpolation reduces the programming complexity, especially in a computing environment where the elevation grid is too large to be stored in main memory.

Because it is necessary that every \((x,y)\) in the original space be represented on one and only one profile, it is not possible to step through the coordinates in \((x', y')\) order and then assign nearest neighbors around rounding the \((x,y)\) transforms. This procedure would not guarantee that every \((x,y)\) would be selected. Instead a \textit{queue} is created for each profile, and the coordinates stepped through in \((x,y)\) order, assigning each coordinate to the appropriate queue.

The notation \((x, y) \rightarrow \text{queue}(x')\) means to place the coordinate pair \((x, y)\) at the end of the \(x'\) queue, where \(x'\) is rounded to the nearest integer. Similarly, given a vector \(C\) to hold pairs of \((x,y)\) coordinates, the notation \(C \leftarrow \text{queue} \left( x', N \right)\) indicates movement of \(X'\)queue (of length \(N\)) into \(C\) in the order in which the coordinates were placed in the queue. The random access function \(\text{get}(x,y)\) returns the value of the altitude function at \((x,y)\), and the horizon file \(H\_FILE\) stores the horizon points for each \(\theta\) for each point. Let \(H(\theta, x, y)\) contain the horizon coordinate for the point \((x,y)\) in the direction \(\theta\), and let \(H(\theta, x, y) \leftarrow H\_FILE\) signify writing this information at the appropriate address in \(H\_FILE\). The function \(\text{round}(\cdot)\) rounds to the nearest integer, and the
procedure \textit{horizon}(N, A, D, H_f, H_b) calculates the one-dimensional horizon function for both forward and backward directions by the previously described algorithm. \( S(x') \) is a vector holding the number of coordinate pairs in queue \( x' \).

Algorithm 3 to build the queues, and Algorithm 4, to calculate the horizon function for each queue, in the directions \( \theta \) and \( \theta + \pi \), are listed in the Appendix. If the horizon for the complete circle is desired, this last algorithm may be run at some \( \Delta \theta \) over the range \( 0 \leq \theta < \pi \). In these algorithms, the details of the queue management and random access input/output functions are omitted, because these depend on the size of the computer and the details of the operating system.

AN EXAMPLE

Figure 3 shows an example of the type of calculation possible with the horizon algorithm. Horizons were calculated for sun azimuth 148 deg. (from N) and elevation 21 deg. (from horizontal), corresponding to 9:30 a.m. on 1 January for the Kearsarge Pass area in the southern Sierra Nevada. The complete grid consists of 63,333 points (279 \times 227) at 100 m spacing. The figure shows the area which would be shaded by local horizons, about 24\% of the total area. The total number of iterations required was 115, 239, or an average of 1.8 per point. In Figure 3 edge effects, which are present along the southern boundary, have not been accounted for. Normally, it is necessary to run the horizon algorithm on an area which is larger than the actual area for which the horizons are desired. For most terrain an edge zone of 1 or 2 km is sufficient, if it is selected carefully by inspection of a topographic map. In the southern Sierra Nevada, maximum distances to horizons within a grid are typically about 10 km.

![Figure 3. Topographic map of Bullfrog Lake-Kearsarge Pass area in southern Sierra Nevada. Contour interval is 100 m. Shaded area represents terrain that would be obscured by local horizons at 9:30 a.m. on 1 January (sun elevation 21 deg., azimuth 148 deg. from north). Obscured terrain comprises 24\% of total area.](image-url)
A faster solution to the horizon problem

N = 16

Figure 4. Horizon diagrams for location on East Creek (located by an X on Figure 3) for Δθ of π/8, π/16, π/32, and π/64, corresponding to 16, 32, 64, and 128 divisions of circle. It seems that little is gained by dividing circle into more than 32 parts.

THE CHOICE OF Δθ

When calculating horizons over all directions, an important question is: what value of Δθ will give us a reasonable picture of the complete surrounding horizon? Figure 4 illustrates, for a single point, horizon diagrams for Δθ of π/8, π/16, π/32, and π/64 rad (i.e. for 16, 32, 64, and 128 horizon directions). The point, indicated by an X on Figure 3, is located on the floor of the steep mountain valley of East Creek, a tributary of Bubbs Creek in the Kings River drainage, South Fork. Figure 4, along with similar drawings for other locations, indicates that, in the southern Sierra Nevada, 32 directions for horizon computations (Δθ = π/16 = 11.25 deg.) are sufficient for radiation models, even though a smaller Δθ gives a more natural looking horizon.

SUMMARY

The new algorithm for calculating horizons over a digital terrain grid requires only order N iterations. Such a task now becomes computationally reasonable.

Acknowledgments—The work was supported by the National Aeronautics and Space Administration, Grant NSG 5362, and the National Oceanic and Atmospheric Administration, Grant 04-6-M01-78. We also appreciate some helpful comments by the reviewers.

REFERENCES


Horn, B. K. P., and Bachman, B. L., 1978, Using synthetic images to register real images with surface models: Comm. ACM, v. 21, no. 11, p. 914-924.


Otterman, J., and Fraser, R. S., 1976, Earth-atmosphere system and surface reflectivities in arid regions from LANDSAT MSS data: Remote Sensing Environ., v. 5, no. 4, p. 247-266.


APPENDIX

1. A straightforward but slow one-dimensional horizon algorithm (for the forward direction)

\[
\begin{align*}
1: & \quad H(N-1) := N-1; \\
2: & \quad i := N-2; \\
3: & \quad \text{while } (i \geq 0) \text{ do} \\
4: & \quad H(i) := i; \\
5: & \quad j := i+1; \\
6: & \quad \text{MAX_TAN} := 0; \\
7: & \quad \text{while } (j < N) \text{ do} \\
8: & \quad \quad \text{if } (\text{slope}(i,j) > \text{MAX_TAN}) \text{ then do} \\
9: & \quad \quad \quad H(i) := j; \\
10: & \quad \quad \quad \text{MAX_TAN} := \text{slope}(i,j); \\
11: & \quad \quad \text{end}
\end{align*}
\]
12: \( j := j+1; \)
13: end
14: \( i := i-1; \)
15: end

2. A faster one-dimensional algorithm (for the forward direction)
1: \( H(N-1) := N-1; \)
2: \( i := N-2; \)
3: while \( i > 0 \) do
4: \( j := i+1; \)
5: \( \text{found} := 0; \)
6: while \( \text{found}=0 \) do
7: if \( \text{slope}(i,j) < \text{slope}(j,H(j)) \) then \( j := H(j); \)
8: else do
9: \( \text{found} := 1; \)
10: if \( \text{slope}(i,j) > \text{slope}(j,H(j)) \) then \( H(i) := j; \)
11: else if \( \text{slope}(i,j) = 0 \) then \( H(i) := i; \)
12: else \( H(i) := H(j); \)
13: end
14: end
15: \( i := i-1; \)
16: end

2.1. Changes in fast algorithm for backward direction
1: \( H(0) := 0; \)
2: \( i := 1; \)
3: while \( i < N \) do
4: \( j := i-1; \)
5: end

3. Algorithm to build queues to hold profiles
1: if \( \theta < \pi/2 \) then do
2: \( y := -M_1; \)
3: while \( y < M_2 \) do
4: \( x := -N_1; \)
5: while \( x < N_1 \) do
6: \( x' := \text{round}(x \cos \theta - y \sin \theta); \)
7: \( (x,y) := \text{queue}(x'); \)
8: \( x := x+1; \)
9: \( S(x') := S(x')+1; \)
10: end
11: \( y := y+1; \)
12: end
13: end
14: else do
15: \( y := M_1; \)
16: while \( y > -M_2 \) do
17: \( x := -N_1; \)
18: while \( x < N_1 \) do
19: \( x' := \text{round}(x \cos \theta + y \sin \theta); \)
20: \( (x,y) := \text{queue}(x'); \)
21: \( x := x+1; \)
22: \( S(x') := S(x')+1; \)
23: end
24: \( y := y-1; \)
25: end
26: end

4. Algorithm to calculate horizon functions along profiles
1: if \( \theta < \pi/2 \) then do
2: \( J_1 := \text{round}(N_1 \cos \theta + M_1 \sin \theta); \)
3: \( J_2 := \text{round}(N_2 \cos \theta + M_1 \sin \theta); \)
4: end
5: else do
A faster solution to the horizon problem

6: \[ J_1 = \text{round}(-N_1 \cos \theta + M_1 \sin \theta); \]
7: \[ J_2 = \text{round}(-N_2 \cos \theta + M_2 \sin \theta); \]
8: end

9: \[ x' = -J_1; \]
10: while \( x' \leq J_2 \) do
11: \[ S(x') = \text{queue}(x', S(x')); \]
12: \[ j = 0; \]
13: while \( j < S(x') \) do
14: \[ (x, y) = C(j); \]
15: \[ H(j) = \text{get}(x, y); \]
16: \[ D(j) = x \cos \theta - y \sin \theta; \]
17: \[ j = j + 1; \]
18: end
19: \[ \text{hzh } (S(x'), A, D, H_j, H_b); \]
20: \[ j = 0; \]
21: while \( j < S(x') \) do
22: \[ (x, y) = C(j); \]
23: \[ H(0, x, y) = H_j(j); \]
24: \[ H(0, x, y) = \_FILE; \]
25: \[ H(\theta + n, x, y) = H_b(j); \]
26: \[ H(\theta + n, x, y) = \_FILE; \]
27: \[ j = j + 1; \]
28: end
29: \[ x' = x' + 1; \]
30: end