Fast Horizon Computation at All Points of a Terrain with Visibility and Shading Applications

A. James Stewart

Abstract

A terrain is most often represented with a digital elevation map consisting of a set of sample points from the terrain surface. This paper presents a fast and practical algorithm to compute the horizon, or skyline, at all sample points of a terrain. The horizons are useful in a number of applications, including the rendering of self–shadowing displacement maps, visibility culling for faster flight simulation, and rendering of cartographic data. Experimental and theoretical results are presented which show that the algorithm is more accurate that previous algorithms and is faster than previous algorithms in terrains of more than 100,000 sample points.

Keywords

terrain, digital elevation map, horizon, skyline, visibility, shadows, rendering, GIS

I. Introduction

Terrains are most often represented with a digital elevation map which consists of a set of sample points from the terrain surface. The sample points are typically taken over a regular square grid and the terrain surface is interpolated between adjacent samples. Alternatively, sample points may be taken at arbitrary locations and the terrain surface formed from a triangulated irregular network of the points.

From a particular point on the terrain, the horizon, or skyline, is the boundary between the sky and the terrain as seen from that point. The horizon is different at different points on the terrain.

This paper describes an algorithm to compute the horizon at all sample points of a terrain. This is in contrast to most previous horizon algorithms which compute the horizon at a single sample point. Single–point algorithms cannot efficiently be scaled to all points, as will be discussed below.

The algorithm has been implemented and tested on terrains of up to 915,800 points, the largest available to the author. Experimental and theoretical results show that the algorithm is more accurate than previous algorithms for this problem and is faster in terrains of more than 100,000 points.

A. James Stewart, Dynamic Graphics Project, Department of Computer Science, University of Toronto, 10 Kings College Road, Toronto, Ontario, Canada, M5S 3G4. phone: (416) 978–5359. fax: (416) 978–5184. email: jstewart@cs.toronto.edu

A. Motivation

Horizons can be used in a number of applications. These include rendering cartographic data, rendering self–shadowing textures, accelerating flight simulation, placing radio transmission towers, visualizing scientific data, and path planning to avoid detection. These applications fall into two categories:

A.1 Shading Applications

Horizons can be used to increase accuracy and speed in shading applications, in which a terrain, displacement map[1], or 2D function surface is shaded under some lighting conditions. When rendered as a 2D image, the shading provides important visual cues to the 3D shape. One such cue is the presence of shadows, which are typically found in valleys or indentations of the surface.

To shade a Lambertian terrain,[2] the radiosity (i.e. intensity) of each sample point is computed and interpolation yields the radiosity of the terrain surface between sample points. If the surface is texture mapped — using photographs of the terrain, for example — the colour provided by the texture map is scaled by the surface radiosity.

Horizons allow efficient computation of a point's approximate radiosity. First, the point's direct primary irradiance (i.e. direct sky illumination) is computed as follows: Given a function, $L(\mathbf{x}, \theta, \phi)$, that defines the sky radiance from direction (θ, ϕ) toward sample point $\mathbf{x}$, the direct primary irradiance at $\mathbf{x}$ is computed by integrating $L(\mathbf{x}, \theta, \phi)$ over those directions above the horizon. Second, the direct primary irradiance is used in a local illumination model to produce the point's radiosity. Section VI-A provides details. Note that, once the horizons are computed, the terrain can quickly be shaded for *any* distribution of sky light, $L(\mathbf{x}, \theta, \phi)$, and, in particular, for any position of the sun. Fig. 1(a) shows a terrain shaded with this method using a "cloudy day" distribution of sky light.

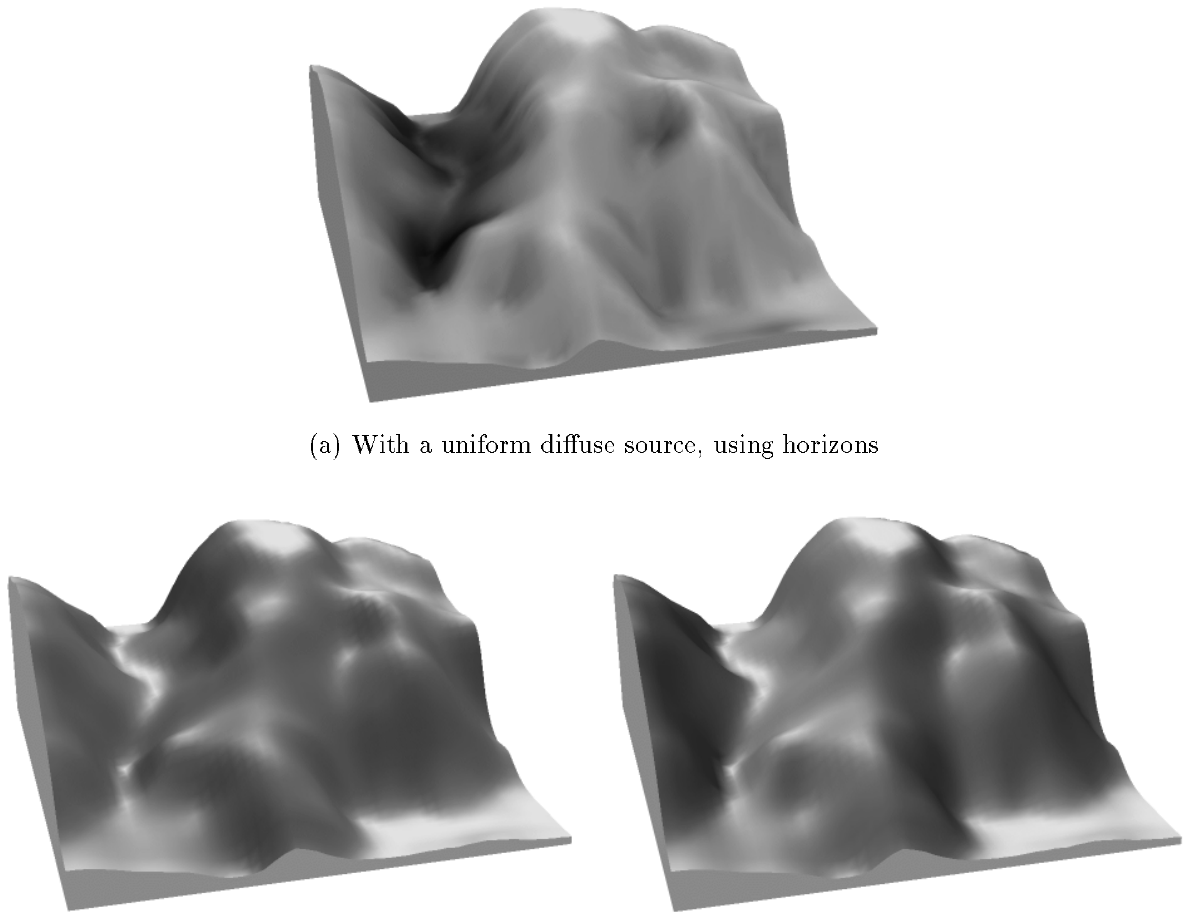
Most cartographic applications instead use a point light source to produce a shaded terrain. They shade a point, $\mathbf{x}$, of the terrain according to the "aspect," which is the dot product of two vectors: the surface normal at $\mathbf{x}$ and the direction from $\mathbf{x}$ to the light source. If the point light source is directly above the terrain (as the sun is at noon), flat surfaces are bright and sloped surfaces are darker as shown in Fig. 1(b). If the light source is not directly above, shadowing is present as shown in Fig. 1(c). Robertson [1] describes a spatial transform that efficiently determines where these shadows fall. Similar techniques are used by Miller [2] and Max [3].

Unfortunately, point–source lighting provides ambiguous perceptual cues to the surface shape. Specifically, both hilltops and valleys contain image intensity maxima and the human visual system may easily confuse one with the other [4]. However, under diffuse lighting conditions, valleys are darker than hills and the visual system does not make this confusion [5].[3]

[1]A displacement map is a texture that may be placed on a smooth surface to produce variations in surface height.

[2]A Lambertian, or diffuse, surface reflects light equally in all directions.

[3]Langer and Bulthoff [5] show that, for a common class of surfaces, perception of shape is more accurate under



(a) With a uniform diffuse source, using horizons

(b) With a point source directly overhead

(c) With a point source 10 degrees off vertical

Fig. 1. Three methods of shading a terrain. The top image uses a uniform diffuse hemispheric light source; its illumination is computed from horizons. The valley in the top image is shadowed by the hills on either side. The bottom images use a point light source; their illumination is computed from the local normal at each terrain point. In the bottom–right image, the right–facing slopes are shadowed since the light source is 10 degrees left of vertical. The most evident flaw in the two bottom images is the overly–bright valley, which is not present in the top image.

Horizons, which allow a terrain to be shaded efficiently under uniform diffuse illumination, can thus be used to produce images with better shape perception that current methods.

Other terrain shading techniques include shadow volumes (for example, Kaneda *et al* [6]) and ray tracing (for example, Coquillart and Gangnet [7]). However, shadow volumes only apply to point light sources and ray tracing is quite expensive if sampling is used to incorporate diffuse illumination [8].

A.2 Visibility Applications

Horizons can also be used to accelerate visibility applications. These applications need to know what parts of the terrain are visible from particular viewpoints.

For example, in placing radio transmission towers, one must determine what part of the

uniform diffuse lighting than under point source lighting. Experiments have not yet been performed to determine whether this observation extends to terrains, but Fig. 1 seems to support such an extension.

terrain is covered by, or is visible from, each tower. In path planning to avoid detection, one must determine a path that is not visible from any of a number of observers who are positioned on or above the terrain. In flight simulation, one would prefer to render only those parts of the terrain that are visible from the current viewpoint, since rendering invisible parts is wasteful and potentially time consuming. For further applications, the interested reader is directed to Nagy's comprehensive survey paper [9], which describes terrain visibility applications and algorithms. De Floriani and Magillo also provide a thorough survey [10] of visibility algorithms for triangulated terrains.

In all of these visibility applications, horizons can be used to determine which parts of the terrain are visible from a particular viewpoint, as follows: The key observation is that a sample point is hidden if the viewpoint lies beyond and below the horizon of that sample point. Given precomputed horizons, this observation yields a quick, conservative[4] visibility test which is much faster than the usual method of tracing a ray between the viewpoint and the sample point. In addition, an entire group of sample points can be tested for visibility by computing, in a preprocessing step, the lower envelope of all the points' horizons. Then the entire group is hidden if the viewpoint lies beyond and below this lower envelope in all directions in which it is visible from the group. Thus, a large area of the terrain can be classified with a single visibility test. Details are found in another paper [11], which describes hierarchical visibility testing in terrains. That paper uses horizons to eliminate from rendering almost all hidden parts of the terrain. As shown in Fig. 2, this can be quite effective, greatly reducing the number of rendered polygons.

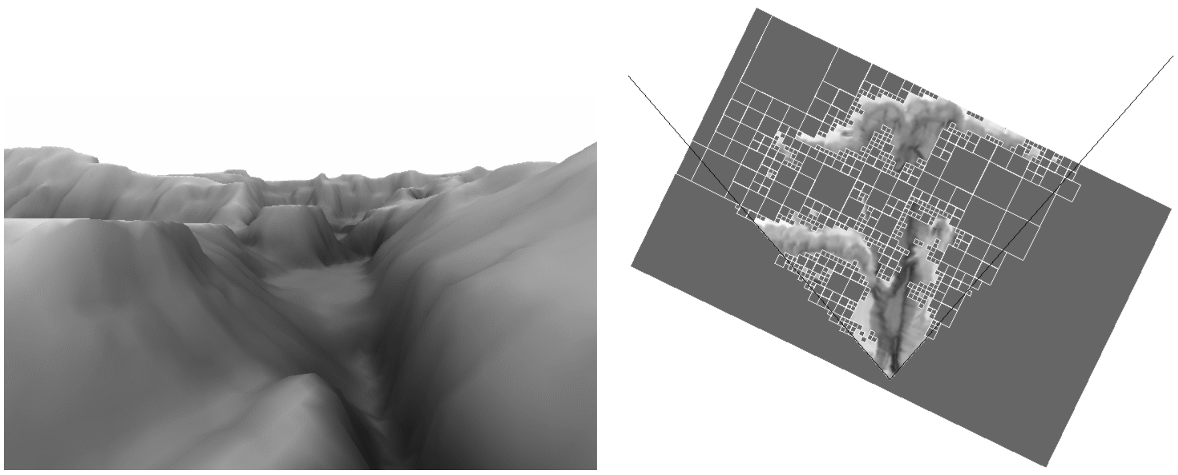

(a) Rendered terrain (b) Overhead view showing visibility

Fig. 2. A 100,000–point terrain rendered using horizons for visibility culling. The overhead view shows which parts of the terrain are rendered. The view frustum is shown as a 'V' with the viewpoint at its vertex. Each empty square region was culled with a single visibility test. Use of horizons eliminated 73% of the 54184 triangles that would otherwise have been rendered, reducing rendering time to 0.96 seconds from 2.44 seconds.

[4]The visibility test is conservative in that some hidden points will not be recognized as such, resulting in a slight overestimate of those parts that are visible.

II. Background

This paper describes an algorithm to compute the horizon at all sample points of a terrain. Closely related work has been done by Max [12], [13], who computes horizons for displacement–mapped surfaces. He considers eight directions around each sample point and determines the horizon elevation in each direction. Points that do not fall directly in the eight directions are not considered in determining the horizon. Upon rendering, a sample point is considered to be illuminated if the sun is visible from the point, which is determined by comparing the sun's elevation with the closest known horizon elevation. Intensity within penumbral shadows is determined according to the fraction of the sun's disc that lies above the horizon. Max also describes transformations from flat to curved surface patches.

The method presented in this paper differs significantly from that of Max. In building each horizon, we consider all sample points of the terrain, rather than only those that lie in particular directions. This results in more accurate horizons and, hence, more accurate rendering. It also enables us to produce soft shadows under nonuniform diffuse illumination. As Max states, his approach is not well suited to casting shadows from bumpy profiles, since this requires a much denser sampling of horizon elevations. In addition, terrains with sharp spikes will result in undersampling artifacts (like absent shadows) with his method. By considering *all* points of the terrain when building each horizon, we avoid these problems. However, considering all points for each horizon can be extremely expensive. Our goal is to do this quickly when the number of points is very large.

Horizon maps like those of Max are used by Cabral, Max, and Springmeyer [14] to compute bidirectional reflection functions for bump maps. They use 24 directions around each point and sample a triangulated terrain instead of discrete points. This is a more accurate and slightly more expensive procedure which reduces — but does not eliminate — undersampling artifacts. In what follows, this will be called the **CMS algorithm**. Section V compares the performance of CMS to the algorithm presented in this paper.

Cohen–Or and Shaked [15] use a similar sampling technique to partition a terrain into visible and invisible parts with respect to a particular viewpoint. A simple modification of their algorithm would produce the horizon from that viewpoint. Cohen–Or and Shaked model the terrain as a dense grid of n rectilinear prisms, the height of each being defined by a sample point centred on its upper face. They show that, with this dense representation, the algorithm need consider only $\mathcal{O}(\sqrt{n}\,)$ directions to avoid undersampling. In related work, Lee and Shin [16] perform similar sampling to determine which parts of a terrain are visible on the screen.

Other work on terrains appears in the computational geometry literature. If the terrain is modelled as a polygonal mesh on n points, computing the horizon for a *single* point is equivalent to computing the upper envelope of a set of $\mathcal{O}(n)$ segments [17]. The horizon has complexity $\mathcal{O}(n\ \alpha(n))$ [5] and can be computed in optimal time $\mathcal{O}(n \log n)$ [18].

[5] $\alpha(n)$ is the extremely slowly growing inverse of Ackermann's function and can be considered constant for any

A hierarchical model can represent the terrain at various resolutions [19], [20], [21]. For this multiresolution model, De Floriani and Magillo present an algorithm to compute the horizon for a single point at different resolutions in time $\mathcal{O}(n \log n)$ [22]. Their algorithm allows the horizon to be updated efficiently with changing levels of detail.

Cole and Sharir [23] describe how to preprocess a terrain in order efficiently to answer ray shooting queries from a fixed point and from a fixed vertical line. Bern, Dobkin, Eppstein, and Grossman [24] also discuss ray shooting queries from vertical lines in terrains. However, none of these works considers computing the horizons at *all* sample points of a terrain. As will be discussed in Section II-B, simply using any one of these single–point algorithms to compute the horizon at all points is far too expensive.

A. Approximate Horizons

For the applications discussed in Section I-A, the horizon must be determined at each of the n sample points. It is not clear how any of the results described above can be extended to compute the horizon at each of n points in subquadratic total time, or whether this is even possible when each horizon is represented as a polygonal chain of size $\Theta(n\ \alpha(n))$ in the worst case.

To achieve subquadratic running time, we approximate the horizon of each sample point: The horizon is divided into s sectors, each spanning $\frac{2\pi}{s}$ radians. In each sector, the horizon is assumed to have constant elevation, which is the maximum of the elevations of all sample points that appear in the sector. The user can choose s to control the accuracy of the approximation.

The methods of Max [13] and of Cabral, Max, and Springmeyer [14] can be thought of as constructing the approximate horizon: Each method divides the horizon into a number of sectors and, in each sector, determines the elevation of the horizon by sampling only those terrain points or terrain edges that fall on the centreline of the sector.

Consider a terrain consisting of a single, tall, narrow spike. It is clear that these two methods will miss the spike if it does not fall exactly on the centreline of a sector, which will result an underestimated horizon elevation and, upon rendering, a missing shadow. However, the approach taken by this paper will use the narrow spike to define the horizon of the entire sector, which may be much wider than the spike. Upon rendering, this might cause shadows to appear where they should not. In Section VI we will demonstrate that the undersampling artifacts of the first two methods are more severe than the artifacts produced by the new method.

B. Analysis of Other Horizon Algorithms

The approximate horizons at each of n sample points can be computed in a straightforward manner in $\mathcal{O}(n^2)$ time, as follows: For each point p, consider each other point q. Determine which sector of p contains q and store in that sector the elevation of q if it is greater than

practical n.

the maximum elevation stored so far. Such an algorithm is extremely expensive for even moderately sized terrains: The author's implementation took 9.5 hours on a $100,000$–point terrain and would have taken 33 days on a $915,800$–point terrain if run to completion. In what follows, this will be called the **naive algorithm**.

Other, more complicated, algorithms that treat each point individually will be even slower. Applying those single–horizon algorithms (e.g. [18], [22]) to each of n points would take $\mathcal{O}(n^2 \log n)$ time. Due to the complexity of the single–horizon algorithms, the constant in front of the $n^2 \log n$ running time will be much larger than that of the naive algorithm. However, the sampling methods of Max [13] and Cabral, Max, and Springmeyer [14] only take time $\mathcal{O}(s\, n^{1.5})$, since each of the n points must sample the centreline of s sectors and the average length of a centreline is $\mathcal{O}(\sqrt{n}\,)$. The algorithm of Cohen–Or and Shaked [15] uses $\mathcal{O}(\sqrt{n}\,)$ sampling directions and, hence, runs in $\mathcal{O}(n^2)$ time when applied to all points of the terrain. However, it has a very small constant like that of the naive algorithm.

This paper presents a faster algorithm to compute the approximate horizon consisting of s sectors at each of n sample points in total time $\mathcal{O}(s\, n\, (\log^2 n + s))$ and space $\mathcal{O}(n \log n)$. The algorithm can be restricted to finding the horizon in a single sector for all n points in time $\mathcal{O}(n\, (\log^2 n + s))$. This is useful if, for example, the sun sets in a fixed sector and we want to know at what elevation the sun disappears from the view of each sample point.

Experimental results in Section V show that, as the number of points increases, the new algorithm's empirical execution time grows much more slowly than that of the CMS and naive algorithms. Thus, the new algorithm is better suited for large terrains (in particular, those of more than 100,000 sample points).

The next two sections describe the horizon computation algorithm. Following that, implementation results are presented.

III. Preliminaries

The input consists of a set of n sample points $p_1, \ldots, p_n$, where p_i has coordinates (x_i, y_i, z_i); coordinate z_i is the height of the point. No two points project to the same location in the x–y plane. It is not necessary that the points lie on a regular (x_i, y_i) grid.

Throughout this discussion, a vertical direction is parallel to the z axis and a horizontal direction is parallel to the x–y plane. The positive z direction is "up." Refer to Fig. 3 for the following definitions.

The **azimuth angle** of a vector v in R^3 is the angle of the projection of v onto the x–y plane, measured counterclockwise from the positive x axis when viewed from above. The space around a point p is divided into s regions based on azimuth angle: The **sector $\boldsymbol{i}$ of $\boldsymbol{p}$**, denoted $\boldsymbol{\sigma_{i,p}}$, is the region of R^3 reachable from p by (non–vertical) vectors having azimuth angle in the closed range $[\ \frac{2\pi}{s}\ i, \frac{2\pi}{s}\ (i+1)\ ]$. Note that adjacent sectors intersect on their common boundary.

Let $\boldsymbol{r_i}$ be a horizontal vector pointing in the direction $\frac{2\pi}{s}\ (i + \frac{1}{2})$. Let $\boldsymbol{\pi_i}$ be the vertical plane that passes through the origin and embeds r_i. Let the notation $\widehat{\boldsymbol{v}}_{\boldsymbol{i}}$ denote the projection of a vector v onto π_i.

With respect to a point p, the **elevation angle** of a point q that lies in $\sigma_{i,p}$ is the angle between r_i and the projection $\widehat{pq}_i$ of the vector pq (angle Θ in Fig. 3). The angle has the same sign as the z component of $\widehat{pq}_i$. This definition is slightly different from the usual one and is necessary in the algorithm that follows. Finally, the **approximate horizon** of a point p is a sequence $e_{0,p}, e_{1,p}, \ldots, e_{s-1,p}$, where $\boldsymbol{e_{i,p}}$ is the elevation of the maximum–elevation point in $\sigma_{i,p}$. In this model, the horizon is a piecewise–constant function of azimuth angle.

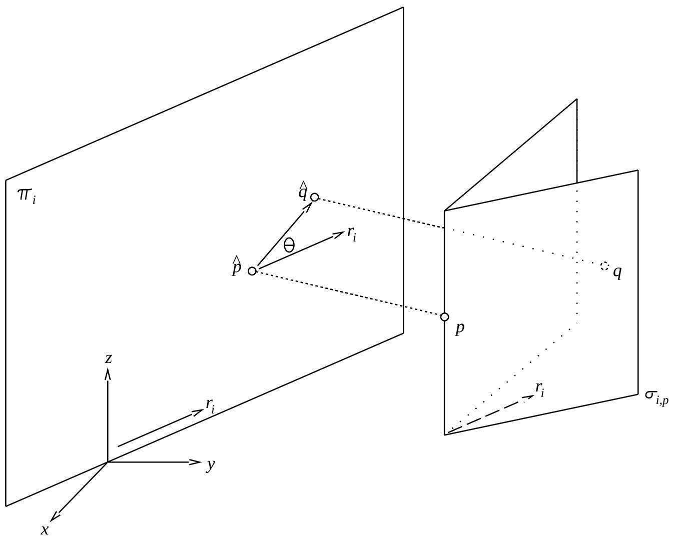


Fig. 3. A point, p, of the terrain is shown, along with sector $\sigma_{i,p}$. Point q lies in the sector. The horizontal vector r_i bisects the sector and is parallel to the vertical plane π_i, which passes through the origin. The projections of p and q onto π_i are denoted $\widehat{p}$ and $\widehat{q}$, respectively. The elevation angle of q with respect to p is denoted Θ.

IV. The Algorithm

The horizon algorithm iterates over each of the s sectors. In the i^{th} iteration, the horizon elevation in sector i is calculated for each of the n sample points. Upon completion, the horizon is known for all points. This is in contrast to the algorithms described in Section II-B, which compute the entire horizon of one point before moving on to the next point.

The remainder of Section IV describes the i^{th} iteration of the horizon algorithm.

A. New Coordinate System

To simplify what follows, we'll transform the sample points from an x–y–z coordinate system to an $a^{\perp}$–$b^{\perp}$–z coordinate system. Refer to Fig. 4 for the following definitions. Let a be a horizontal vector in the direction $\frac{2\pi}{s}\ i$ and let b be a horizontal vector in the direction $\frac{2\pi}{s}\ (i+1)$. These vectors define the extreme directions of the i^{th} sector. Let $a^{\perp}$ be a horizontal vector $\frac{\pi}{2}$ radians clockwise of a. Similarly, let $b^{\perp}$ be a horizontal vector $\frac{\pi}{2}$ radians clockwise of b.

Order the sample points in the direction $a^{\perp}$ and let j denote the position of point p in the

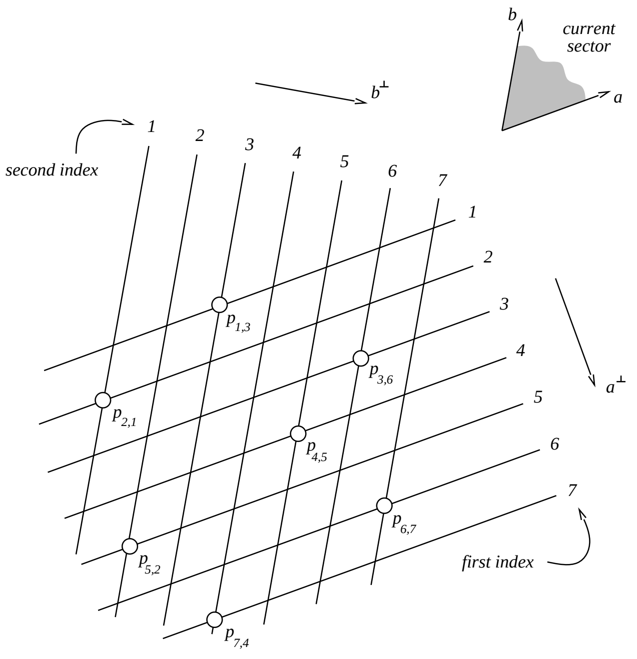


Fig. 4. An overhead view of a terrain of seven sample points is shown. Vectors a and b bound the current sector, i. $\sigma_{i,p}$ is the region above and to the right of a point p, delimited by the lines (parallel to a and b) passing through p. Each point is doubly indexed with its position in the $a^\perp$ and $b^\perp$ orderings.

ordering. Similarly, order the points in the direction $b^\perp$ and let k denote p's position in the ordering. In both orderings, the first point has position 1. We will attach these subscripts to p, as in $p_{j,k}$. Note that one subscript is sufficient to distinguish a point and the notation $p_{j,*}$ or $p_{*,k}$ may be used without ambiguity.

B. Algorithm Overview

Each sample point, p, has an associated data structure, $\mathcal{D}_p$, which stores sample points from $\sigma_{i,p}$ (recall that $\sigma_{i,p}$ denotes the i^{th} sector of p). Over the course of the i^{th} iteration, sample points are inserted into $\mathcal{D}_p$. As soon as $\mathcal{D}_p$ contains all the sample points in $\sigma_{i,p}$, it is queried for $e_{i,p}$, the maximum elevation of points in $\sigma_{i,p}$.

Points are processed in order of increasing first index. This ensures that all points that lie in $\sigma_{i,p}$ are processed before p itself, since all points in $\sigma_{i,p}$ have a smaller first index than that of p. Processing a particular point, p, involves two steps:

1. For all points, q, in whose i^{th} sector p lies, p is inserted into $\mathcal{D}_q$. In Fig. 4, the points in the region below and left of p have p in their i^{th} sector. These are exactly the points whose first index is greater than that of p and whose second index is less than that of p.
2. $\mathcal{D}_p$ is queried for elevation $e_{i,p}$, which is stored. Due to the processing order, $\mathcal{D}_p$ contains all points in $\sigma_{i,p}$ when this query is made.

C. The Data Structure

Each sample point stored in $\mathcal{D}_p$ is projected onto the plane π_i. $\mathcal{D}_p$ maintains on π_i a **covering set** of upper convex hulls[6] whose union covers all of the projected points. See Fig. 5.

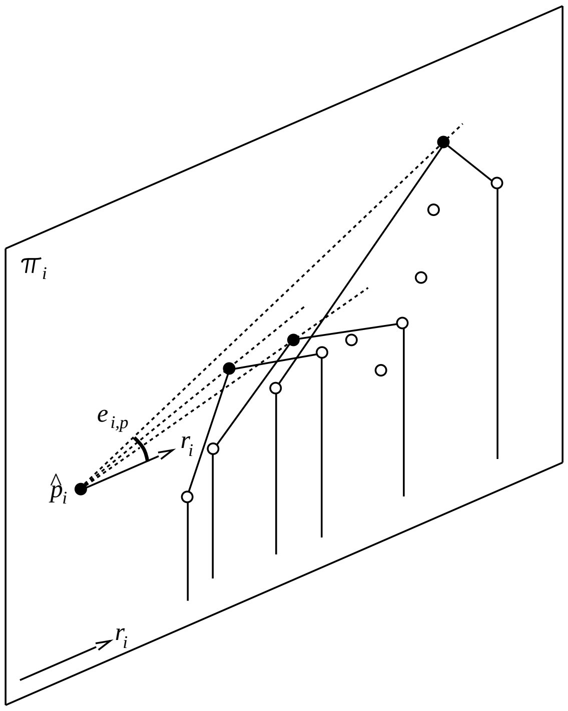


Fig. 5. Plane π_i is shown with projection $\widehat{p}_i$ of the point p and projections (unlabelled in the figure) of the points that are stored in $\mathcal{D}_p$. Three upper hulls form a covering set whose union covers all of the projected points. The elevation $e_{i,p}$ is that of the steepest of the three tangents from $\widehat{p}_i$ to the hulls.

To insert point q into $\mathcal{D}_p$ its projection, $\widehat{q}_i$, is computed. Then $\widehat{q}_i$ is inserted into one of the hulls, possibly causing the hull to be modified. The choice of hull will be described below, along with the reason for having more than one hull.

To determine the maximum elevation of the points stored in $\mathcal{D}_p$, the tangent between $\widehat{p}_i$ and each of the hulls is computed. The maximum of the tangent elevations is equal to the maximum of the point elevations, since each point in $\mathcal{D}_p$ is covered by at least one of the hulls, whose vertices are points in $\mathcal{D}_p$.

The difficulty in making the algorithm efficient lies in Step 1 above, in which p is inserted into the $\mathcal{D}_q$ of every point q having p in its i^{th} sector. Since there are $\mathcal{O}(n)$ such points and insertion into a hull takes $\mathcal{O}(\log n)$ time [25], that step could take $\mathcal{O}(n \log n)$ time for p alone. The i^{th} iteration could thus take $\mathcal{O}(n^2 \log n)$ time, which is worse than the previous algorithms described in Section II-B.

To make Step 1 more efficient, the algorithm maintains a complete, static, binary tree of

[6]An **upper convex hull** is the upper chain of a convex hull with two extreme edges extending vertically downward to $-\infty$. While the hull may contain many points on its interior, we only store the points that define the hull's boundary.

n leaves[7] in which each node stores a *single* upper hull that lies in the plane π_i. The leaf nodes are labelled $\mathcal{L}_1, \ldots, \mathcal{L}_n$ from left to right.

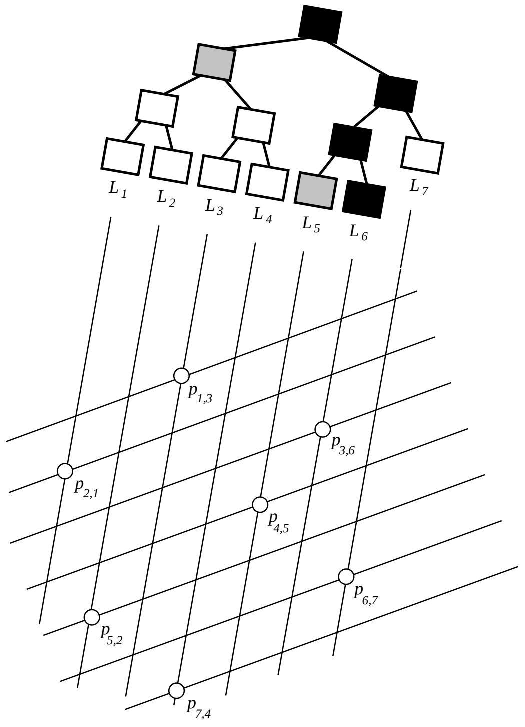


Fig. 6. A complete binary tree of n leaves, each node containing a single upper hull. Each point $p_{*,k}$ is associated with leaf $\mathcal{L}_k$ of the tree. The covering set of $\mathcal{D}_{p_{*,k}}$ consists of those upper hulls on the root–to–$\mathcal{L}_k$ path. For example, the black nodes contain the covering set of $\mathcal{D}_{p_{*,6}}$. To add point $p_{*,6}$ to the covering sets of sectors $\mathcal{D}_{p_{*,1}}$, $\mathcal{D}_{p_{*,2}}$, $\ldots$, $\mathcal{D}_{p_{*,5}}$, it is sufficient to insert it into the upper hulls of nodes (shown in gray) to the immediate left of the root–to–$\mathcal{L}_6$ path.

Using the second index (from the $b^{\perp}$ ordering), point $p_{*,k}$ is associated with leaf $\mathcal{L}_k$. The covering set of $\mathcal{D}_{p_{*,k}}$ consists of the upper hulls stored in the nodes on the root–to–$\mathcal{L}_k$ path. See Fig. 6. With this tree of hulls, insertion and elevation query can be efficiently implemented:

Elevation Query: The maximum elevation of points in $\mathcal{D}_{p_{*,k}}$ is that of the maximum–elevation tangent between $\widehat{p}_i$ and each of the hulls on the root–to–$\mathcal{L}_k$ path.

Insertion: Point $p_{j,k}$ must be inserted into the $\mathcal{D}_{q_{s,t}}$ of all points, $q_{s,t}$ for which $s > j$ and $t < k$. These $q_{s,t}$ are below and left of $p_{j,k}$ in Fig. 6. We will remove the restriction that $s > j$. This means that $p_{j,k}$ will, in addition, be inserted into the $\mathcal{D}_{q_{s,t}}$ of points *above* and left of $p_{j,k}$ in Fig. 6. However, this will have no effect on the computed elevations because those $\mathcal{D}_{q_{s,t}}$ above $p_{j,k}$ will have been queried for their elevations *before* $p_{j,k}$ is inserted, since points are processed in order of increasing first index.

Now, point $p_{j,k}$ must be inserted into those $\mathcal{D}_{q_{s,t}}$ for which $t < k$: $\mathcal{D}_{q_{*,1}}, \mathcal{D}_{q_{*,2}}, \ldots, \mathcal{D}_{q_{*,k-1}}$. To do this efficiently, the projection, $\widehat{p_{j,k}}_i$, is inserted into the hull of *every left child* on the

[7]In a complete binary tree, all levels are full except the bottommost, which is filled from left to right. Since the tree is static (it doesn't change shape), it can be stored in an array $A[1 \ldots N]$ such that the two children of node $A[i]$ are $A[2i]$ and $A[2i+1]$.

root–to–$\mathcal{L}_k$ path. This has the effect of inserting $p_{j,k}$ into $\mathcal{D}_{q_{*,1}}, \mathcal{D}_{q_{*,2}}, \ldots, \mathcal{D}_{q_{*,k-1}}$ since one of the left children lies on each root–to–$\mathcal{L}_t$ path for $1 \leq t \leq k-1$.

D. Algorithm Summary and Analysis

In summary, the algorithm iterates over the s sectors, computing in the i^{th} iteration the maximum elevation, $e_{i,p}$, for each of the n sample points, p. The i^{th} iteration is performed as follows:

1. Sort the sample points by $a^{\perp}$ and $b^{\perp}$ to determine the two indices of each point.
2. For each point, $p_{j,k}$, in order of increasing first index:
   (a) Insert $\widehat{p_{j,k}}_i$ into the upper hull of every left child on the root–to–$\mathcal{L}_k$ path.
   (b) Compute the tangents from $\widehat{p_{j,k}}_i$ to each of the upper hulls on the root–to–$\mathcal{L}_k$ path. Record the elevation of the highest tangent in $e_{i,p_{j,k}}$.

Insertion of a point into a hull of n points can be accomplished in $\mathcal{O}(\log n)$ time [25]. The tangent to a hull of n points can be computed in $\mathcal{O}(\log n)$ time [26]. For implementation details, a very readable discussion of point insertion into convex hulls appears in Section 3.7 of O'Rourke's text [27].

Since a complete binary tree of n leaves has height in $\mathcal{O}(\log n)$, each of steps 2a and 2b above takes $\mathcal{O}(\log^2 n)$ time. The $\mathcal{O}(n \log n)$ time for sorting is subsumed by that of processing the points, so each of the s iterations takes $\mathcal{O}(n \log^2 n)$ time. A modification described in Section IV-G will increase the running time by $\mathcal{O}(ns)$ per iteration. Thus, the total running time of the algorithm is $\mathcal{O}(s\, n\, (log^2 n + s))$.

The key idea of the algorithm is that a point can be added to the covering sets of $\mathcal{D}_{p_{*,1}}, \mathcal{D}_{p_{*,2}}, \ldots, \mathcal{D}_{P_{*,k-1}}$ by inserting it into only $\mathcal{O}(\log n)$ upper hulls. Since each sample point is added to only $\mathcal{O}(\log n)$ upper hulls, the storage bound is $\mathcal{O}(n \log n)$.

E. Space Reduction

To reduce space requirements and increase speed, a leaf node $\mathcal{L}_k$ of the tree is marked as **inactive** once the corresponding point $p_{*,k}$ has been processed. The hull associated with the inactive node is discarded, since it is not required after the elevation of $p_{*,k}$ is computed. An internal node is marked as inactive and its hull discarded when its two children become inactive. In Step 2a above, $p_{j,k}$ is *not* inserted into inactive hulls.

To reduce space further, note that points are only inserted into hulls of tree nodes that are left children. All right children and the root have empty hulls. Thus, only the left children need to be stored, which can be done with a heap–ordered array [28]. This requires a bit of care with array indices when traversing the root–to–$\mathcal{L}_k$ path, but reduces the size of the tree by half. For example, in the author's implementation this would save about eight megabytes in a terrain of a million points, since each hull requires eight bytes plus 12 bytes per stored point and a tree of a million leaves has about two million nodes.

F. Degenerate Situations

Care must be taken with sample points that fall exactly upon the boundary of a sector; this occurs frequently with grid data. We must ensure that such points are included in the sector; otherwise, we may underestimate the sector elevation. Consider Fig. 7. When sorting the points by $a^\perp$ to determine their processing order, points that have the same projection onto $a^\perp$ must be sorted by their projection onto $-a$. This ensures that points lying on the more clockwise boundary of the sector of a point p are processed, and hence added to the sector, before p is processed. When sorting the points by $b^\perp$, points that have the same projection onto $b^\perp$ must be sorted by their projection onto $+b$. This ensures that points lying on the more counterclockwise boundary of the sector of a point p have a larger second index than p, and are thus added to the sector of p when they are processed (recall that a point with second index k is added to $\mathcal{D}_{p_{*,1}}, \mathcal{D}_{p_{*,2}}, \ldots, \mathcal{D}_{p_{*,k-1}}$).

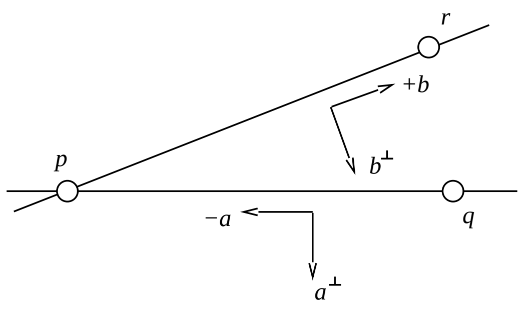


Fig. 7. Points q and r fall on the clockwise and counterclockwise boundaries, respectively, of a sector of p. In order that q is stored in the sector, it must be processed before p and so must precede p in the ordering by $a^\perp$. Although r already precedes p in the $a^\perp$ ordering, it must *follow* p in the $b^\perp$ ordering if it is to be stored in p's sector.

G. Narrow Sectors

Care must be taken when the sector angle $\frac{2\pi}{s}$ is small. Narrow sectors will often not intersect a sample point until some distance from the sector vertex. If closer sample points are higher than the first intersected point, the elevation will be underestimated. See Fig. 8.

The solution is to test the elevations of a fixed number of sample points that border the sector close to the sector vertex, in addition to determining the maximum elevation of the $\mathcal{O}(\log n)$ upper hulls in the usual way. Suppose that the sample points are regularly spaced a distance d apart. In the worst case, the sector will not intersect a sample point until it becomes about this wide. Since the sector has width d at about distance $d \,/\sin(\frac{2\pi}{s})$ from the vertex, only $2\,/\sin(\frac{2\pi}{s})$ sample points bordering the sector must be checked. For large s, $\sin(\frac{2\pi}{s}) \approx \frac{2\pi}{s}$ and the number of bordering points is approximately $\frac{s}{\pi}$. This solution is simply an application of the CMS algorithm in a small, bounded area around each point.

This additional work adds $\mathcal{O}(ns)$ time to the running time of each of the s iterations of the algorithm. However, the constant in front of ns is relatively small. For example, if 64 sectors are used, only 20 additional sample points must be checked. As each point takes

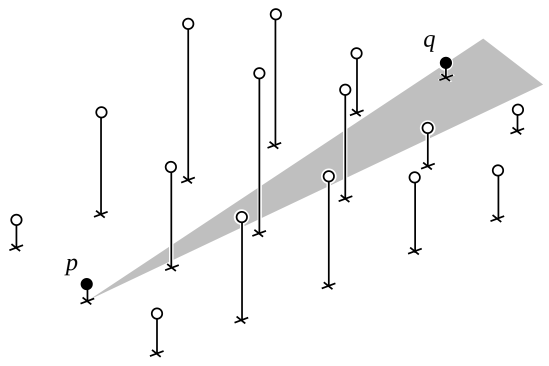


Fig. 8. Sample points are shown on top of vertical segments that indicate their heights with respect to the x–y plane. The shaded area of the x–y plane is a sector of p that has its elevation incorrectly determined by point q, the only point to fall within the sector. The elevation should be determined by the high points closer to p.

about 9.9×10^{-7} seconds to test (which was experimentally determined), this takes only 19 minutes for a $915,800$–point terrain on which the algorithm takes 2.9 hours to run. That is, only 11% of the time is spent checking these points.

V. Experimental Results: Running Times

The algorithm was implemented in 1200 lines of C++ code. It was tested on terrains of between 1000 points and 915,800 points, the largest available to the author. Running times on a 166 MHz Pentium PC running Linux with 64 Mb RAM are summarized in Table I. For comparison, the CMS algorithm was implemented and tested on the same scenes. The naive $\mathcal{O}(n^2)$ algorithm, described in Section II-B, was also implemented and tested on some of the same scenes (its time for the largest scene is extrapolated from the smaller scenes). The algorithms computed horizons of 64 sectors.

Fig. 9 shows the running times on a log / log graph, in which the line slopes correspond to the exponents of the algorithms' running times. This graph shows that, in practice, the new algorithm exhibits almost linear growth in execution time, while the CMS algorithm exhibits approximately $\mathcal{O}(n^{1.5})$ growth, as expected. The new algorithm outperforms the CMS algorithm on terrains of more than about $100,000$ points (e.g. a 316×316 terrain), which are considered small by today's standards.

Memory requirements are a concern with large terrains. Experiments with the new algorithm show that the average number of points stored in each upper hull of the binary tree was less than two for all terrains tested. Since the average hull size is so small in practice, it is neither necessary nor desirable to use a sophisticated data structure to store the hulls. A simple sorted array for the points of each hull uses very little space and achieves sufficient speed. In fact, the times reported in Table I are for an algorithm that uses a simple sorted array for each hull. A more sophisticated data structure, like a concatenable queue [25], is unnecessary and uses far more space. Using a concatenable queue, the algorithm ran out of memory for the largest terrain.

TABLE I

Running Times (in seconds)

Terrain Size	New Algorithm	CMS Algorithm	$\mathcal{O}(n^2)$ Algorithm
1,000	5	2	4
3,136	18	6	
10,000	61	24	331
40,000	302	176	
100,000	911	893	34,100
315,844	3,320	5,570	
915,800	10,600	25,800	2,850,000

VI. Experimental Results: Rendering Quality

The horizon allows us to compute the direct sky illumination of each terrain point. For accurate rendering, we should also consider reflected illumination that arrives at each point from other parts of the terrain. In this case, the direct sky illuminations computed from the horizons can serve as the initial irradiances for a progressive radiosity algorithm.

A. Illumination Model

To avoid the expensive radiosity computation, we will use an illumination model developed by Stewart and Langer [29]. This model, which provides a coarser approximation of the radiosity of a point, only considers direct primary illumination. The model exploits the following observation: In a terrain under uniform diffuse illumination, bright points on peaks tend to see other bright points on peaks, while dark points in valleys tend to see other dark points in valleys. That is, points tend to see other points of approximately the same radiance. Exploiting this heuristic, we start with the standard radiosity equation

$$R(\mathbf{x}) \;=\; \frac{\rho}{\pi}\int_{\mathcal{V}(\mathbf{x})} L_{sky}(\mathbf{u})\;\mathbf{N}(\mathbf{x})\cdot\mathbf{u}\;d\Omega \;+\; \frac{\rho}{\pi}\int_{\mathcal{H}(\mathbf{x})\backslash\mathcal{V}(\mathbf{x})} R(\Pi(\mathbf{x},\mathbf{u}))\;\mathbf{N}(\mathbf{x})\cdot\mathbf{u}\;d\Omega$$

where $\mathbf{x}$ is a surface point, $L_{sky}(\mathbf{u})$ is the radiance of the source in direction $\mathbf{u}$ (independent of position $\mathbf{x}$), $\mathbf{N}(\mathbf{x})$ is the surface normal, $\mathcal{H}(\mathbf{x}) = \{\mathbf{u} : \mathbf{N}(\mathbf{x})\cdot\mathbf{u} > 0\}$ is the hemisphere of outgoing unit vectors, $\mathcal{V}(\mathbf{x})$ is the set of unit directions in which the sky is visible from $\mathbf{x}$, $d\Omega$ is an infinitesimal solid angle, and $\Pi(\mathbf{x},\mathbf{u})$ is the surface point visible from $\mathbf{x}$ in direction $\mathbf{u}$. We assume diffuse reflection and replace incoming radiance, $R(\Pi(\mathbf{x},\mathbf{u}))$, from the directions $\mathcal{H}(\mathbf{x})\backslash\mathcal{V}(\mathbf{x})$ in which the terrain is visible with the point's own outgoing radiance, $R(\mathbf{x})$. An algebraic manipulation yields

$$R(\mathbf{x}) \;\equiv\; \frac{\rho\,\frac{1}{\pi}\,\mathbf{N}(\mathbf{x})\cdot\int_{\mathcal{V}(\mathbf{x})} L_{sky}(\mathbf{u})\;\mathbf{u}\;d\Omega}{1 \;-\; \rho\,(1-\frac{1}{\pi}\mathbf{N}(\mathbf{x})\cdot\int_{\mathcal{V}(\mathbf{x})}\;\mathbf{u}\;d\Omega)}\;. \tag{1}$$

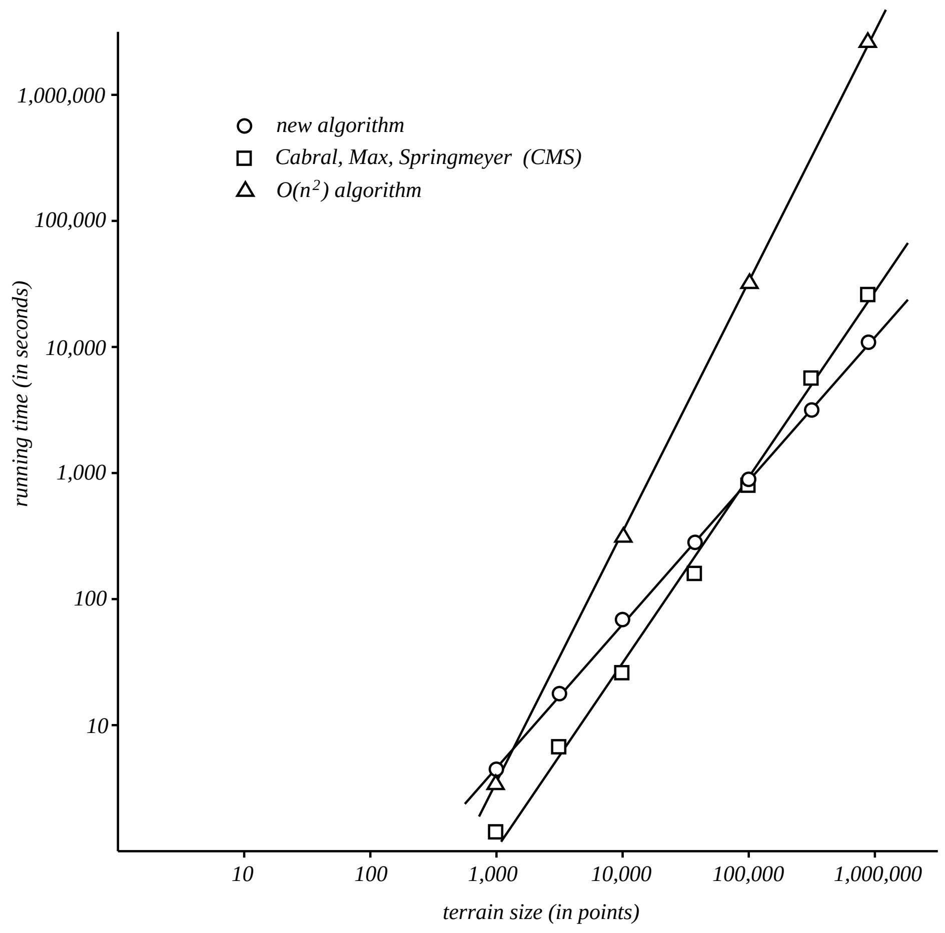


Fig. 9. A log / log graph of the running times of the three algorithms. If the line interpolating the results for a particular algorithm has slope k, that algorithm runs in time $\mathcal{O}(n^k)$ on inputs of size n. These experimental results show that the new algorithm runs in time $\mathcal{O}(n^{1.1})$, the CMS algorithm runs in time $\mathcal{O}(n^{1.6})$, and the straightforward algorithm runs in time $\mathcal{O}(n^{2.0})$. The new algorithm is faster than the others for terrains of more than 100, 000 points.

Further discussion and experimental justification can be found in the paper [29].

To model nonuniform sky illumination, the hemisphere of sky directions is discretized into 256 s equal–area regions: Azimuth angle ϕ is divided into s sectors corresponding to the s sectors of the horizon; each sector of the hemisphere is divided by elevation angle θ into 256 equal–area regions. Each region has a constant illumination which is defined by the user.

Under the approximate illumination model, the radiosity is computed quickly using Equation 1 with lookup tables — indexed by horizon elevation and sector — for the two integral terms. The author's rendering program computes radiosity for about 3600 terrain points per second with horizons of 64 sectors.

B. Rendered Images

Both the new algorithm and the CMS algorithm have problems in terrains with tall, narrow spikes. The CMS algorithm may miss the spikes (generating no shadows at all), while the new algorithm may generate too dark a shadow, since it uses the highest point in a sector to define the elevation across the whole sector.

Fig. 10 shows six renderings of a terrain of 100×100 sample points, viewed from directly above. All points have height zero, except a single point in the upper–left corner which has height 50 and represents a tall, narrow spike. The area light source is 30 degrees above the horizon. The images differ in the algorithm used to compute the horizons (CMS algorithm or new algorithm) and in the number of sectors making up the horizon (64, 125, or 250).

The top row of Fig. 10 demonstrates the algorithms at low horizon resolution. The sampling directions are clear. The CMS algorithm misses a large area of the penumbra due to undersampling, while the new algorithm produces too dark a penumbra due to its assumption that the single spike spans the whole sector.

The middle and bottom rows demonstrate higher horizon resolutions. At 125 sectors, the new algorithm produces a reasonable penumbra, while the CMS algorithm is still misses large parts. At 250 sectors, the penumbrae are identical near the spike (since the new algorithm uses CMS–like sampling in a small, bounded region around each point; recall Section IV-G), but the new algorithm produces a smoother and more complete penumbra farther from the spike.

Fig. 11 shows a 221×202 Martian terrain rendered from 150–sector horizons generated by the new algorithm. The sun is an area source and diffusion in the Martian atmosphere is modelled with an small background illumination from all directions. Fig. 12 shows an overhead view of the same area over the course of a day. The horizons needed to be computed only once to render all of the images.

VII. Summary

An efficient algorithm has been presented that computes the approximate horizon, consisting of s sectors, at all n points of a terrain in time $\mathcal{O}(s\, n\, (\log^2 n + s))$. The horizons, which need to be computed only once, can be used determine the direct sky illumination of each sample point under any distribution of sky light. The direct sky illumination can be used to render the terrain with an approximate illumination model or can be used as initial input to a progressive radiosity algorithm. In addition, the horizons provide visibility information that can be used to accelerate terrain rendering and other visibility applications. The algorithm requires only 1200 lines of C++ code and is available by email from the author.

There are two criteria for evaluating this new algorithm with respect to the existing CMS algorithm: its speed and the quality of images generated from its horizons. The new algorithm is faster than the CMS algorithm, both in experimental and theoretical analysis, once the size of the terrain exceeds about $100,000$ points. Since the execution time of the new algorithm grows more slowly than that of the CMS algorithm ($\mathcal{O}(n^{1.1})$ versus $\mathcal{O}(n^{1.6})$ in experiments), it becomes more attractive as the terrain size becomes larger.

With respect to image quality, the new algorithm avoids absent shadows at the risk of introducing spurious shadows. From the images shown, this appears to be a good tradeoff.

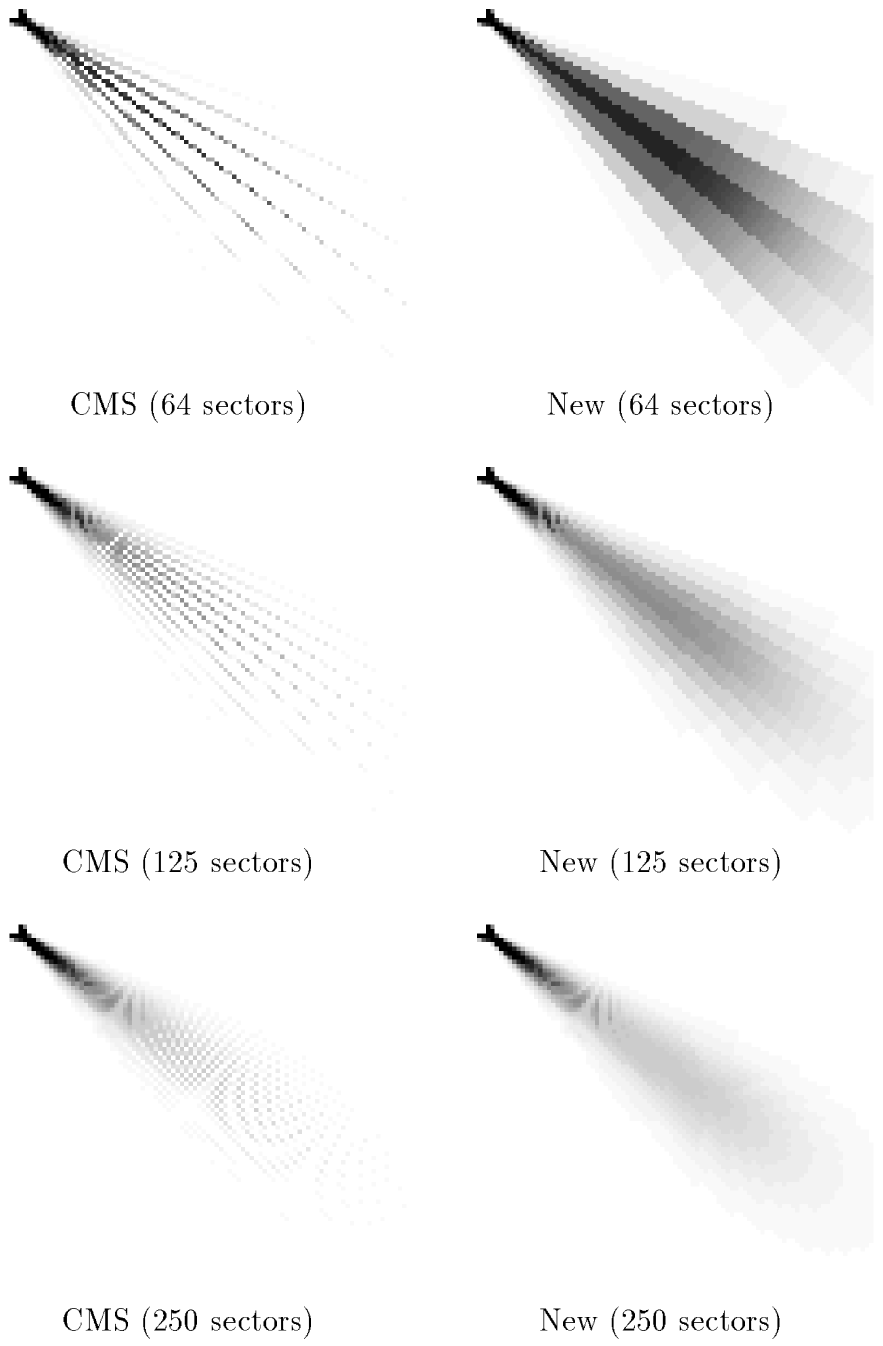


Fig. 10. A flat, 10, 000–point terrain containing a single spike in the upper–left corner, shaded using horizons computed by the CMS and new algorithms, viewed from directly above. At low horizon resolution, the CMS algorithm misses shadows, while the new algorithm generates too dark a shadow from a single point. At higher horizon resolutions the algorithms produce identical penumbrae near the spike, while the new algorithm produces a smoother and more complete penumbra farther from the spike. The Moiré pattern near the spike occurs because CMS–like sampling sometimes includes the spike in more than one sector, resulting in less visible sky and a darker terrain point.

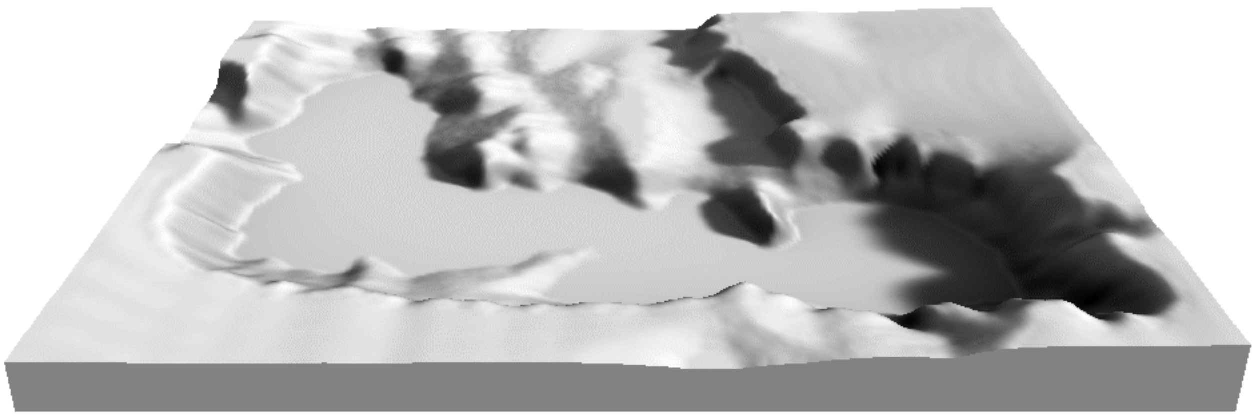


Fig. 11. A Martian terrain of 221 × 202 sample points rendered with the sun 30 degrees above the right horizon. A shadow is cast from the plateau in the upper–left and the silhouette of the ridge is apparent in the shadow. A soft penumbra, which results from the area light source, appears on the right of the basin floor.

The algorithm is easy to parallelize on the obvious "per sector" basis, since iterations of the algorithm are independent. It would be interesting to explore a parallel algorithm that exploits the parallelism inherent in the tree of convex hulls.

Acknowledgements

The author wishes to thank Mike Caplinger of Malin Space Science Systems for providing the Mars dataset, and wishes to thank Bob Kanefsky of the NASA Ames Research Center and Doug Obrien of the Canada Centre for Remote Sensing for providing pointers to other datasets. The author wishes to express sincere thanks to the reviewers, whose comments were quite helpful, and to Michael Langer, for suggestions on the manuscript.

The Martian terrain data provided by Michael Caplinger of Malin Space Science Systems was processed from the data on the Planetary Data System cd–rom volume MG_2007, "Mars Mosaicked Digital Image Model (MDIM) and Digital Terrain Model (DTM), version 2.0," assembled by Eric Eliason, Raymond Batson, and Anthony Manley. This cd–rom is available from the National Space Science Data Center, Code 633, Goddard Space Flight Center, Greenbelt, MD 20771.

James Stewart's work is supported by the Information Technology Research Centre of Ontario, the Natural Sciences and Engineering Research Council of Canada, and the University of Toronto.

References

[1] P. K. Robertson, "Spatial transformations for rapid scan-line surface shadowing", *IEEE Computer Graphics and Applications*, vol. 9, no. 2, pp. 30–38, Mar. 1989.

[2] G. S. P. Miller, "The definition and rendering of terrain maps", in *Computer Graphics (SIGGRAPH '86 Proceedings)*, Aug. 1986, vol. 20, pp. 39–48.

[3] Nelson L. Max, "Atmospheric illumination and shadows", in *Computer Graphics (SIGGRAPH '86 Proceedings)*, Aug. 1986, vol. 20, pp. 117–24.

[4] V. Ramachandran, "Perception of shape from shading", *Nature*, vol. 331, no. 6152, pp. 163–165, 1988.

[5] M. S. Langer and H. H. Bülthoff, "Do humans perceive judge shape from shading better on sunny days or on cloudy days?", Tech. Rep., NEC Research Institute, 1997, NECI Technical Report 97-130.

[6] K. Kaneda, F. Kato, E. Nakamae, T. Nishita, H. Tanaka, and T. Noguchi, "Three dimensional terrain modeling and display for environmental assessment", in *Computer Graphics (SIGGRAPH '89 Proceedings)*, July 1989, vol. 23, pp. 207–214.

[7] S. Coquillart and M. Gangnet, "Shaded display of digital maps", *IEEE Computer Graphics and Applications*, vol. 4, no. 7, pp. 35–42, July 1984.

[8] J. T. Kajiya, "The rendering equation", in *Computer Graphics (SIGGRAPH '86 Proceedings)*, August 1986, vol. 20, pp. 143–150.

[9] G. Nagy, "Terrain visibility", *Computers and Graphics*, vol. 18, no. 6, pp. 763–773, 1994.

[10] L. De Floriani and P. Magillo, "Visibility algorithms on triangulated terrain models", *International Journal of Geographic Information Systems*, vol. 8, no. 1, pp. 13–41, 1994.

[11] A. James Stewart, "Hierarchical visibility in terrains", in *Eurographics Rendering Workshop*, June 1997, pp. 217–228.

[12] N. L. Max, "Shadows for bump-mapped surfaces", in *Advanced Computer Graphics (Proceedings of Computer Graphics Tokyo '86)*, Tsiyasu L. Kunii, Ed. 1986, pp. 145–156, Springer-Verlag.

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

[14] B. Cabral, N. L. Max, and R. Springmeyer, "Bidirectional reflection functions from surface bump maps", in *Computer Graphics (SIGGRAPH '87 Proceedings)*, July 1987, vol. 21, pp. 273–281.

[15] D. Cohen-Or and A. Shaked, "Visibility and dead–zones in digital terrain maps", *Computer Graphics Forum (Eurographics '95)*, vol. 14, no. 3, 1995.

[16] C-H. Lee and Y. G. Shin, "An efficient ray tracing method for terrain rendering", in *Pacific Graphics '95*, August 1995.

[17] M. J. Atallah, "Dynamic computational geometry", in *Proceedings of the 24th Annual IEEE Sympososium on Foundations of Computer Science*, 1983, pp. 92–99.

[18] J. Hershberger, "Finding the upper envelope of n line segments in $O(n \log n)$ time", *Information Processing Letters*, vol. 33, pp. 169–174, 1989.

[19] L. De Floriani and E. Puppo, "Constrained Delaunay triangulation for multiresolution surface description", in *Proceedings of the Ninth IEEE International Conference on Pattern Recognition*, 1988, pp. 566–569.

[20] L. Scarlatos and T. Pavlidis, "Hierarchical triangulation using terrain features", in *Proceedings of the First 1990 IEEE Conference on Visualization (Visualization '90)*, 1990, pp. 168–175.

[21] M. de Berg and K. Dobrindt, "On levels of detail in terrains", in *Proceedings of the 11th Annual ACM Symposium on Computational Geometry*, 1995, pp. C26–C27.

[22] L. De Floriani and P. Magillo, "Horizon computation on a hierarchical triangulated terrain model", *The Visual Computer*, vol. 11, pp. 134–149, 1995.

[23] R. Cole and M. Sharir, "Visibility problems for polyhedral terrains", *Journal of Symbolic Computing*, vol. 7, pp. 11–30, 1989.

[24] M. Bern, D. Dobkin, D. Eppstein, and R. Grossman, "Visibility with a moving point of view", *Algorithmica*, vol. 11, pp. 360–378, 1994.

[25] F. P. Preparata and M. I. Shamos, *Computational Geometry: An Introduction*, Springer-Verlag, New York, NY, 1985.

[26] M. H. Overmars and J. van Leeuwen, "Maintenance of configurations in the plane", *Journal of Computer Systems Science*, vol. 23, pp. 166–204, 1981.

[27] J. O'Rourke, *Computational Geometry in C*, Cambridge University Press, New York, 1994.

[28] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*, MIT Press, Cambridge, MA, 1990.

[29] A. J. Stewart and M. S. Langer, "Towards accurate recovery of shape from shading under diffuse lighting", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, September 1997.

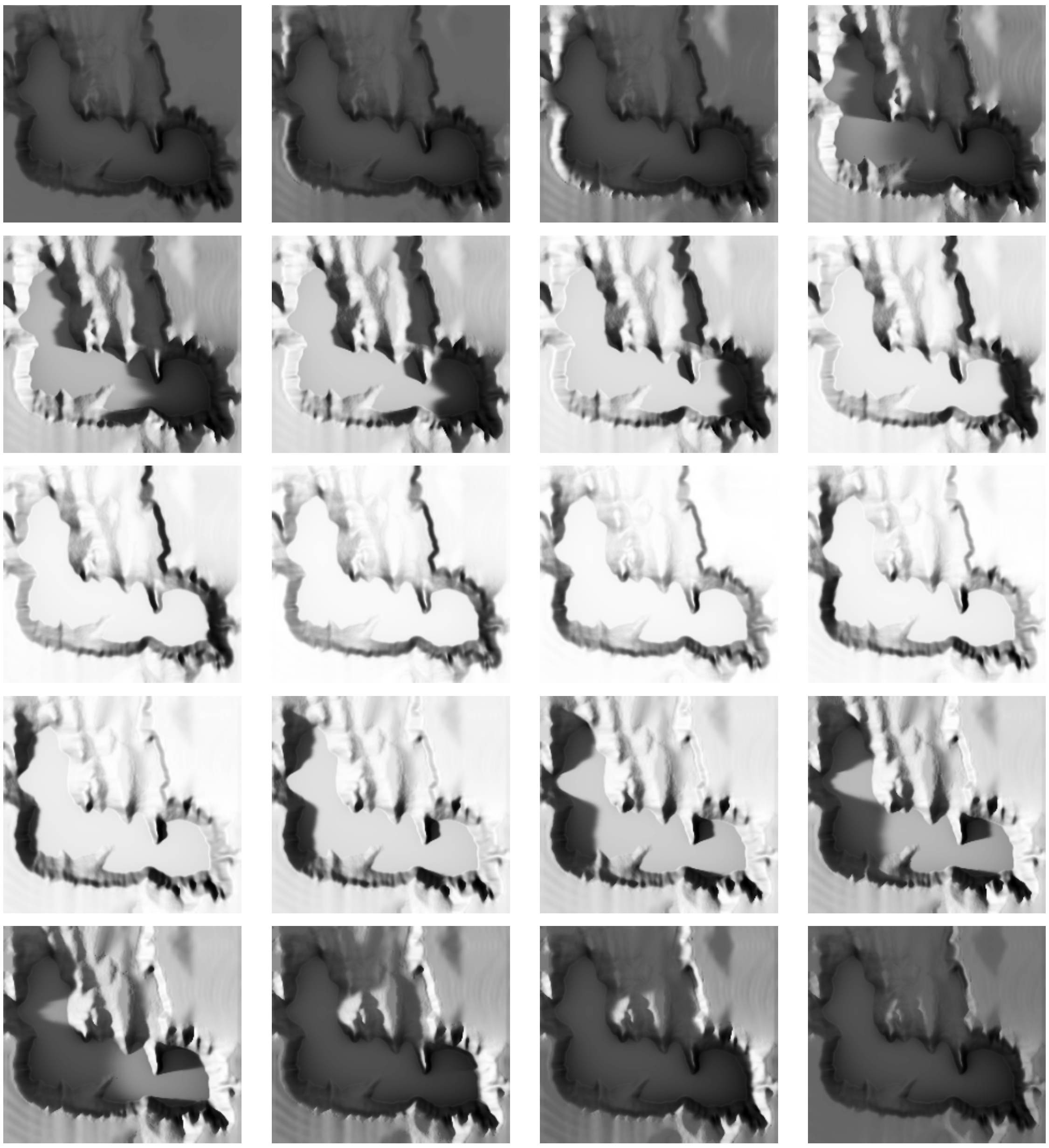


Fig. 12. The area of Fig. 11 shown from above during the course of a day. In the upper–left image, the sun is near the right horizon. In the lower–right image, the sun is near the left horizon. The horizons only needed to be computed once for all of the images.