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

[15] R. Cole and M. Sharir, "Visibility problems for polyhedral terrains", *J. Symbolic Comput.*, vol. 7, pp. 11–30, 1989.

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

[17] M. H. Overmars and J. van Leeuwen, "Maintenance of configurations in the plane", *J. Comput. Syst. Sci.*, vol. 23, pp. 166–204, 1981.

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

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

"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] M. Caplinger, "Planetary society marslink project", Tech. Rep., Malin Space Science Systems, available from http://barsoom.msss.com/http/ps/intro.html.

[2] 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.

[3] A. J. Stewart and M. S. Langer, "Towards accurate recovery of shape from shading under diffuse lighting", in *Proceedings of the 1996 IEEE Conference on Computer Vision and Pattern Recognition*, June 1996, pp. 411–418.

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

[5] 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.

[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] M. J. Atallah, "Dynamic computational geometry", in *Proc. 24th Annu. IEEE Sympos. Found. Comput. Sci.*, 1983, pp. 92–99.

[10] J. Hershberger, "Finding the upper envelope of $n$ line segments in $O(n \log n)$ time", *Inform. Process. Lett.*, vol. 33, pp. 169–174, 1989.

[11] L. De Floriani and E. Puppo, "Constrained Delaunay triangulation for multiresolution surface description", in *Proc. Ninth IEEE International Conference on Pattern Recognition*, Los Alamitos, California, 1988, pp. 566–569, CS Press.

[12] L. Scarlatos and T. Pavlidis, "Hierarchical triangulation using terrain features", in *Proceedings of the 1st 1990 IEEE Conference on Visualization, Visualization '90*, IEEE Service Center, Piscataway, NJ, USA (IEEE cat n 90CH2914-0), 1990, pp. 168–175, IEEE.

[13] M. de Berg and K. Dobrindt, "On levels of detail in terrains", in *Proc. 11th Annu. ACM Sympos. Comput. Geom.*, 1995, pp. C26–C27.

used as initial input to a progressive radiosity algorithm. The horizon computation can also be used to build horizon maps that are used in conjunction with bump maps to render self–shadowing textures. The algorithm is easy to implement in only 1100 lines of C++ code.

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 clearly 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 even more attractive as the terrain size becomes very large.

Regarding 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, especially considering that the CMS algorithm will have to take fewer samples in larger terrains to maintain a competitive speed.

The best horizon algorithm would probably be a blend of the CMS algorithm and the new algorithm. In this "best algorithm," the new algorithm presented in this paper would quickly build a relatively coarse, but complete, horizon for each sample point of the terrain. The CMS algorithm would then refine the horizon in those pairs of adjacent sectors between which a sharp change in elevation is detected. This approach is analogous to that of progressive refinement.

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 (Malin Space Science Systems) for providing the Mars dataset, and wishes to thank Bob Kanefsky (NASA Ames Research Center) and Doug Obrien (Canada Centre for Remote Sensing) for providing pointers to other datasets.

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,

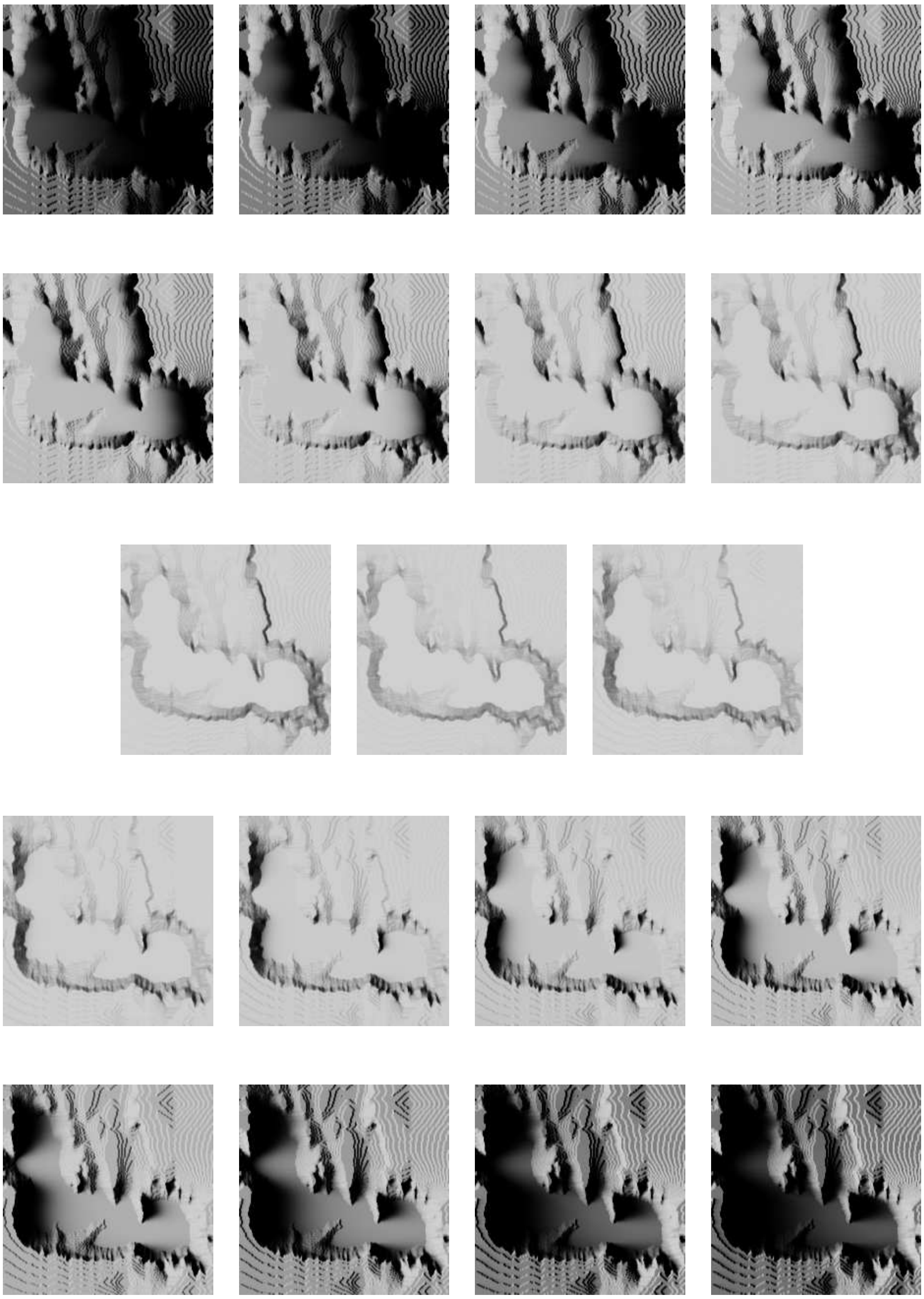

Fig. 10. The area of Fig. 9 shown from above during the course of a day. In the upper–left image, the sun is near the right horizon. In the lower–right image, the sun is near the left horizon. The horizons only needed to be computed once for all of the images. The eight–bit resolution of the height data results in "contour line" artifacts.

Note that the CMS algorithm will be slower than the new algorithm for terrains of more than about $100,000$ points. For such terrains, the CMS algorithm will have to reduce is sampling rate — at a cost to image quality — to remain competitive with the new algorithm.

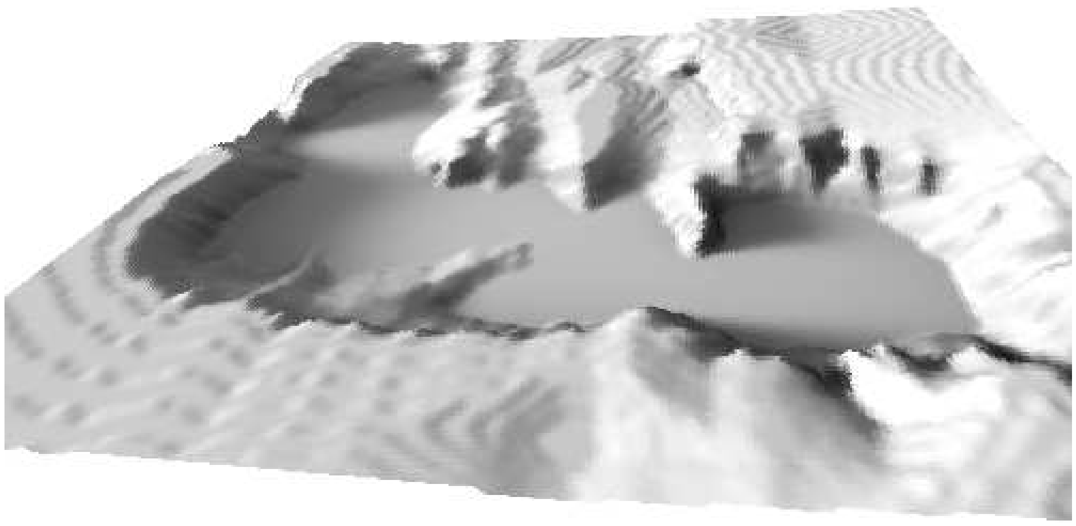

Fig. 9. A Martian terrain of $214 \times 214$ sample points rendered with the sun 30 degrees above the left horizon. The eight–bit resolution of the height data results in apparent contour lines. Note the soft shadows.

Fig. 9 shows a $214 \times 214$ Martian terrain rendered from 64–sector horizons generated by the new algorithm. The sky intensity in a particular view direction was modelled as $R_{sun} \cos^{20} \theta$, where $\theta$ is the angle between the sun direction and the view direction. This model approximates (very coarsely) the diffusion that occurs in the Martian atmosphere.

Fig. 10 shows an overhead view of the same area over the course of a day. Note the soft shadows due to diffuse, nonuniform illumination. The horizons only needed to be computed 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 only need to be computed once, are used determine the direct primary illumination of each sample point under any distribution of sky illumination. The direct primary illumination can be used to render the terrain with an approximate radiosity model or can be

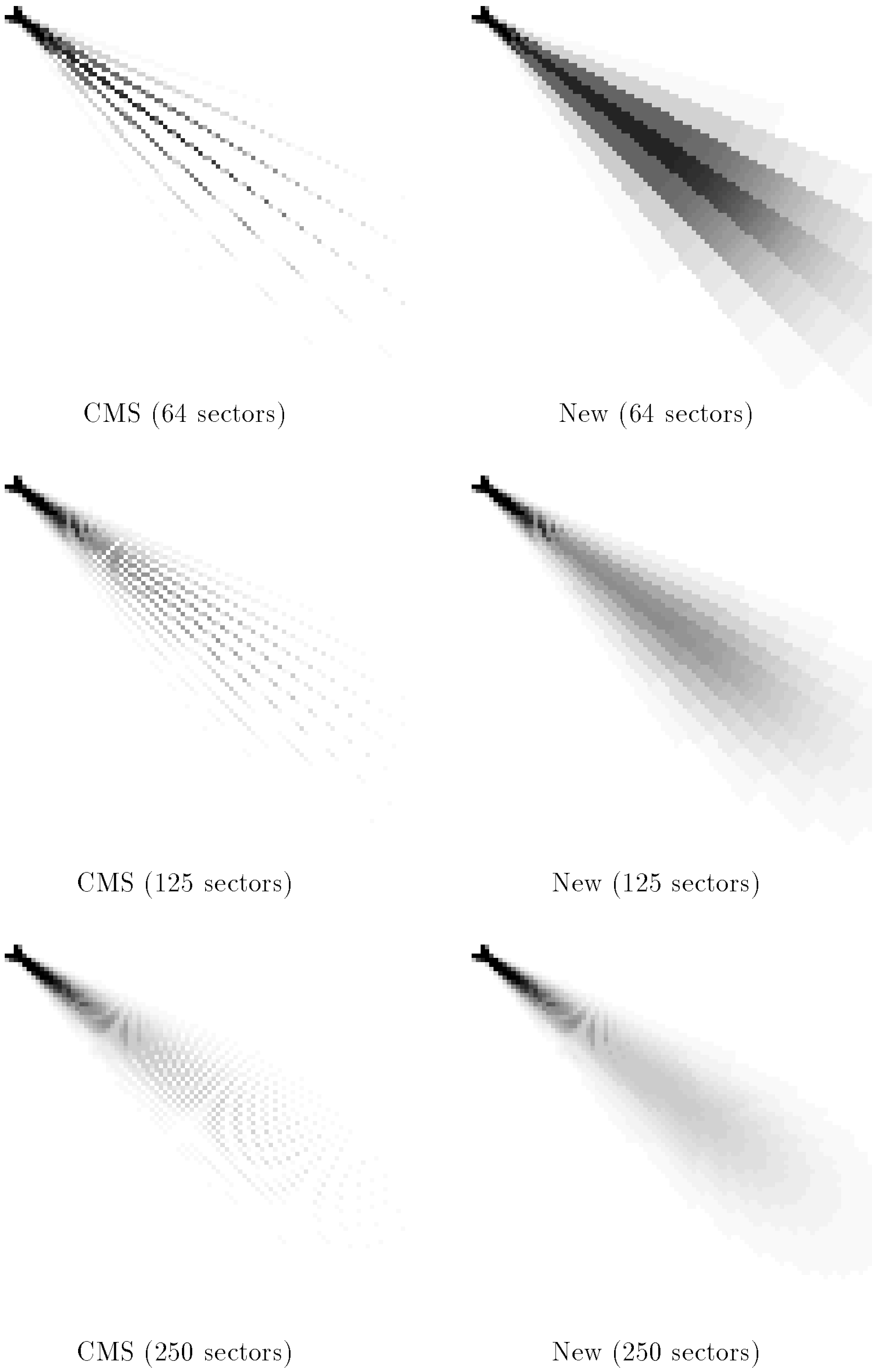

Fig. 8. A flat 10, 000–point terrain containing a single spike in the upper–left corner, rendered using horizons computed by the CMS and new algorithms, viewed from directly above. At low horizon resolution, the CMS algorithm misses shadows, while the new algorithm generates too dark a shadow from a 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.

into 256 $s$ equal–area regions: Azimuth angle $\phi$ is divided into $s$ sectors corresponding to the $s$ sectors of the horizon; each sector of the hemisphere is divided by elevation angle $\theta$ into 256 equal–area regions. Each region has a constant illumination which is defined by the user.

The radisoity $L(x)$ is computed quickly using lookup tables indexed by horizon elevation and sector. The author's rendering program computes $L(x)$ for about 3600 terrain points per second.

## *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. 8 shows six renderings of a terrain of $100 \times 100$ 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). See the Web page `http://www.dgp.toronto.edu/people/JamesStewart` for the actual images, which are of higher quality than could be printed in this paper.

The top row of Fig. 8 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-E), but the new algorithm produces a smoother and more complete penumbra farther from the spike. The Moiré patterns are due to the CMS–like sampling in discrete directions.

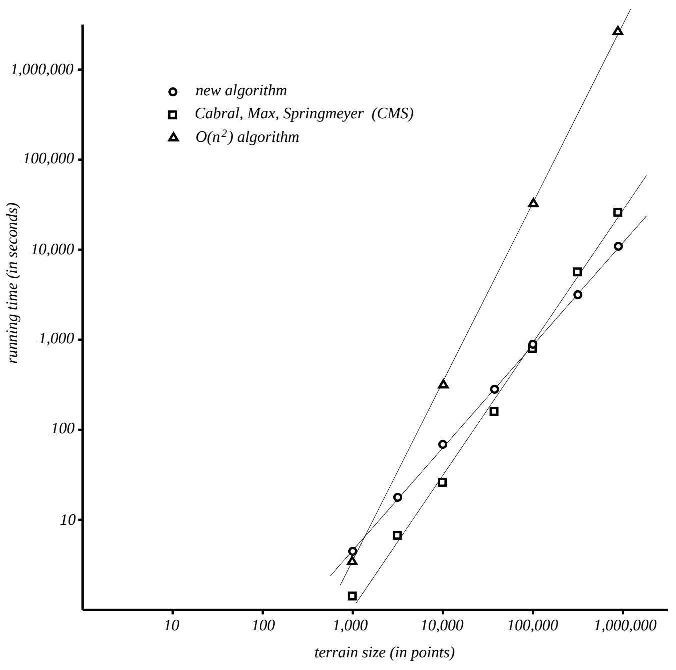

Fig. 7. A log / log graph of the running times of the three algorithms. If the line interpolating the results for a particular algorithm has slope $k$, that algorithm runs in time $\mathcal{O}(n^k)$ on inputs of size $n$. These experimental results show that the new algorithms runs in time $\mathcal{O}(n^{1.1})$, the CMS algorithm runs in time $\mathcal{O}(n^{1.6})$, and the straightforward algorithm runs in time $\mathcal{O}(n^{2.0})$. This shows that the new algorithm easily outpaces the others as the terrain size grows beyond about $100,000$ points. These experimental results correlate very well with the theoretical asymptotic running times, indicating that asymptotic analysis is useful in comparing algorithms for this problem.

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 $\mathbf{u}$ ($\Pi$ denotes "projection"). We 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

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

Further discussion and experimental justification can be found in the paper by Stewart and Langer [3]. While this local illumination model assumes diffuse reflection, another model could be constructed that includes a specular component. In particular, a bump map could be used to define a normal at each sample point.

To model nonuniform sky illumination, the hemisphere of sky directions is discretized

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 primary illumination of each terrain point. For accurate rendering, we should also consider reflected illumination that arrives at each point. In this case, the direct primary 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 a global illumination model developed by Stewart and Langer [3]. 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, 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 observation, we start with the standard radiosity equation

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

where $\mathbf{x}$ is a surface point, $\mathbf{N}(\mathbf{x})$ 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 the individual sample points near the centreline of each sector.

The tree of convex hulls can be thought of as a hierarchical data structure, since the left child of a node stores the upper convex hull of all the sample points in the right subtree of the node. High–level nodes in the tree cover many sample points and this covering is refined at lower levels in the tree. The tree of convex hulls can also be thought of, perhaps more accurately, as being similar to a structure for storing partial sums. It would be interesting to see whether this similarity of structure could be exploited in a parallel algorithm.

## V. Experimental Results: Running Times

The algorithm was implemented in 1100 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 straightforward $\mathcal{O}(n^2)$ algorithm, described in Section II-A, 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. 7 shows the running times on a log / log graph, in which the line slopes correspond to the exponents of the algorithm's 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 $316 \times 316$ terrain).

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 (Section IV-A) 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 [18], is unnecessary and uses far more space. Using a concatenable queue, the algorithm ran out of memory for the largest terrain.

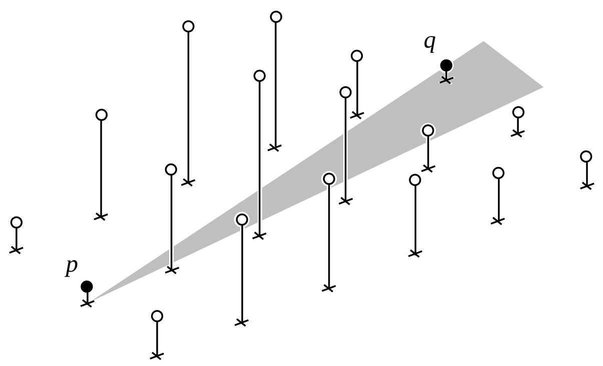

Fig. 6. A set of sample points are shown on top of vertical segments that indicate their heights with respect to the $x$–$y$ plane. The shaded area of the $x$–$y$ plane is a sector of $p$ that has its elevation incorrectly determined by point $q$, the only point to fall within the sector. The elevation should be determined by the high points closer to $p$.

$\mathcal{O}(\log n)$ 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 about $9.9 \times 10^{-7}$ 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 (i.e. 11% of the time is spent checking these points).

## *F. Convex Hulls vs. Quad Trees*

An alternative to the tree of convex hulls might be a quad tree having cell boundaries aligned with sector boundaries. However, the higher–level nodes in the quad tree would usually overlap sector boundaries, making them useless in computing the highest–elevation point near the boundaries of a particular sector. Thus, any algorithm using quad trees would have to inspect the lowest–level quad tree nodes near the sector boundaries. The work involved in doing this is equivalent to that done by the CMS algorithm, which inspects

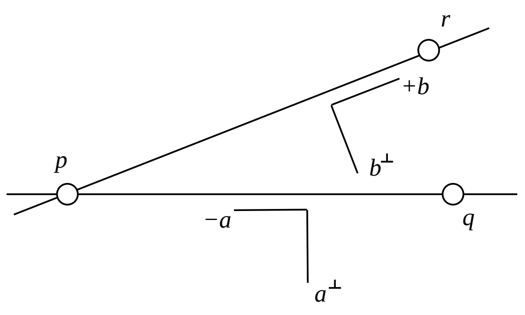

Fig. 5. Points $q$ and $r$ fall on the clockwise and counterclockwise boundaries, respectively, of a sector of $p$. In order that $q$ is stored in the sector, it must be processed before $p$ and so must precede $p$ in the ordering by $a^{\perp}$. Although $r$ already precedes $p$ in the $a^{\perp}$ ordering, it must *follow* $p$ in the $b^{\perp}$ ordering if it is to be stored in $p$'s sector.

## *D. 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. 5. 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 the sectors $\Sigma_{*,1}$ to $\Sigma_{*,k-1}$).

## *E. 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 and, hence, the irradiance will be overestimated (see Fig. 6).

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 root–to–$\mathcal{C}_k$ path. Each insertion can be done in time $\mathcal{O}(\log n)$ [18] for a total time of $\mathcal{O}(\log^2 n)$.

## *B. Correctness and Analysis*

Recall the Observation. Step 2 adds $p_{j,k}$ to the covering sets of all sectors $\Sigma_{s,t}$ for which $t < k$. Those sectors $\Sigma_{s,t}$ for which $s > j$ remain to be treated, since sectors are treated in increasing order of first index. Thus, when $p_{s,t}$ is processed later in the course of the algorithm, all points $p_{j,k}$ for which $s > j$ and $t < k$ will have been added to the covering set of $\Sigma_{s,t}$. Thus, Step 1 will compute the correct elevation $e_i$ when $p_{s,t}$ is processed.

The key idea of the algorithm is that a point can be added to the covering sets of sectors $\Sigma_{*,1}, \Sigma_{*,2}, \ldots, \Sigma_{*,k-1}$ 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 clearly $\mathcal{O}(n \log n)$.

The time bound for the steps above is clearly $\mathcal{O}(n \log^2 n)$ for each of the $s$ iterations. A modification described in Section IV-E will increase the running time by $\mathcal{O}(ns)$ per iteration. Thus, the total running time of the algorithm is $\mathcal{O}(s\ n\ (log^2 n + s))$.

## *C. Space Reduction*

To reduce space requirements and increase speed, a leaf node $\mathcal{C}_k$ 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 2 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 in a heap–ordered array [19]. This requires a bit of care with array indices when traversing the root–to–$\mathcal{C}_k$ 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.

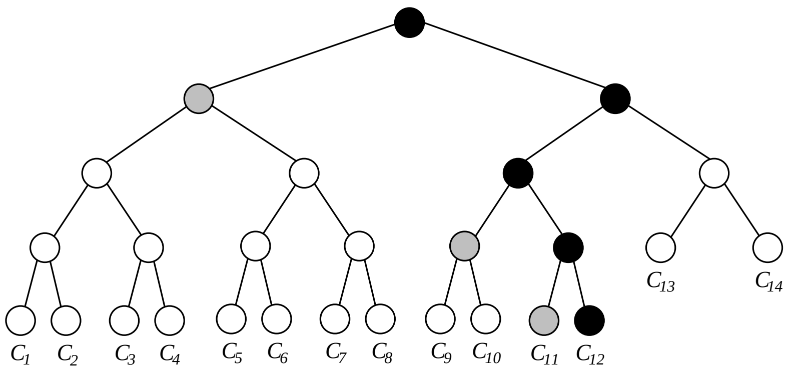

Fig. 4. A complete binary tree of upper hulls in which leaf $\mathcal{C}_k$ corresponds to sector $\Sigma_{*,k}$. The covering set of sector $\Sigma_{*,k}$ consists of those upper hulls on the root–to–$\mathcal{C}_k$ path. For example, the black nodes contain the covering set of sector $\Sigma_{*,12}$. To add point $p_{*,12}$ to the covering sets of sectors $\Sigma_{*,1}, \Sigma_{*,2}, \ldots, \Sigma_{*,11}$, it is sufficient to add it to the upper hulls (shown in gray) to the immediate left of the root–to–$\mathcal{C}_{12}$ path.

## *A. Algorithm Description*

The algorithm maintains a complete, static, binary tree[3] in which each node stores an upper hull that lies in the plane $\pi_i$ (see Fig. 4). The leaf nodes are labelled $\mathcal{C}_1, \ldots, \mathcal{C}_n$ from left to right. The covering set of sector $\Sigma_{*,k}$ consists of the upper hulls stored in the nodes on the root–to–$\mathcal{C}_k$ path, which is of length $\mathcal{O}(\log n)$.

Recall that the algorithm iterates over the $s$ sectors, computing in the $i^{th}$ iteration the maximum elevation $e_i$ in sector $\sigma_i$ of each of the $n$ sample points. The $i^{th}$ iteration is performed as follows:

Initially, sample points are sorted to determine the indices $j$ and $k$. They are then processed in order of increasing first index, $j$. Point $p_{j,k}$ is processed as follows:

1. The tangents from $\widehat{p_{j,k}}_i$ (the projection of $p_{j,k}$ onto $\pi_i$) to each of the upper hulls on the root–to–$\mathcal{C}_k$ path are computed. The elevation of the highest tangent is recorded as the elevation $e_i$ of the horizon in sector $\Sigma_{j,k}$. Each tangent takes $\mathcal{O}(\log n)$ time to compute [17] for a total time of $\mathcal{O}(\log^2 n)$.
2. Point $\widehat{p_{j,k}}_i$ is added to the covering sets of sectors $\Sigma_{*,1}, \Sigma_{*,2}, \ldots, \Sigma_{*,k-1}$. This is accomplished by inserting $\widehat{p_{j,k}}_i$ into the upper hull of every left child on the

[3]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 $A[2i+1]$.

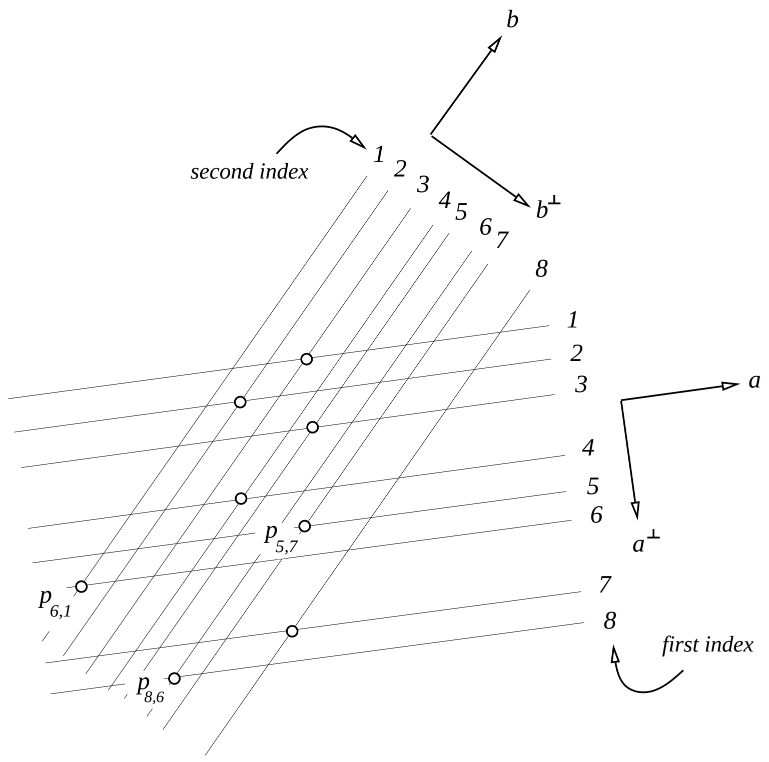

Fig. 3. A top view of the sample points is shown. The sector $\Sigma_{j,k}$ of a sample point $p_{j,k}$ is the region above and to the right of the point, delimited by the lines passing through the point. According to the Observation, point $p_{5,7}$ lies in exactly the sectors $\Sigma_{6,1}$ and $\Sigma_{8,6}$.

Order the sample points in the direction $a^\perp$ and let $j$ denote the position of point $p$ in the 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. For convenience in what follows, 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.

For notational convenience, let $\Sigma_{j,k}$ denote sector $\sigma_i$ of point $p_{j,k}$. This notation refers to a specific sector whose apex embeds a specific point. The index $i$ of the sector is implicit in the notation $\Sigma_{j,k}$ and will be clear from the context.

The following observation is exploited to efficiently generate the covering sets. See Fig. 3 for an example.

**Observation** *Point $p_{j,k}$ lies in exactly those sectors $\Sigma_{s,t}$ for which $s > j$ and $t < k$.*

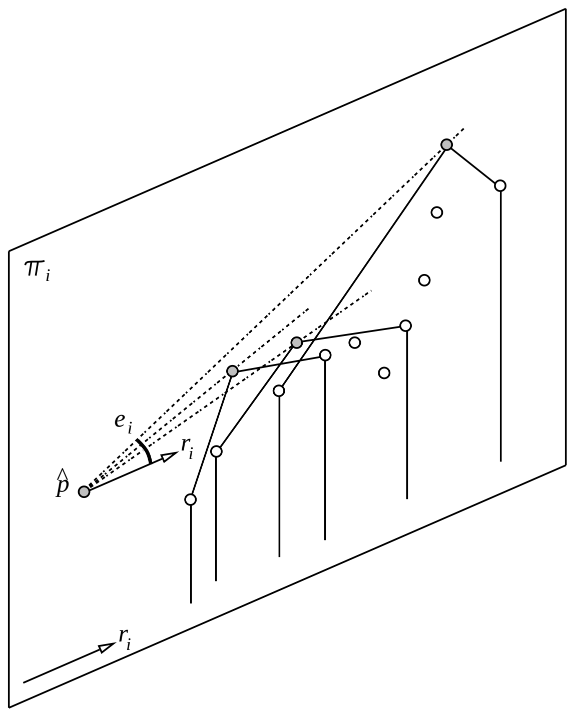

Fig. 2. Plane $\pi_i$ is shown with projection $\hat{p}$ of the point $p$ and projections $\widehat{q_j}_i$ (unlabelled in the figure) of the points $q_j$ that fall within sector $\sigma_i$ of $p$. Three upper hulls form a covering set whose union contains all the $\widehat{q_j}_i$. The elevation $e_i$ of the horizon of $p$ is the elevation of the highest of the three tangents from $\hat{p}$ to the upper hulls.

$\mathcal{O}(\log n)$ upper convex hulls[2] is built whose union contains all of the points $\widehat{q_j}_i$ (see Fig. 2). The tangent between $\hat{p}$ and each of the upper hulls is computed in $\mathcal{O}(\log n)$ time per tangent [17]. The elevation of the highest tangent is recorded in $e_i$. Given the covering set of upper hulls, this procedure takes $\mathcal{O}(\log^2 n)$ time per sample point per sector. It remains to efficiently construct the covering sets.

Refer to Fig. 3 for the following definitions. We will transform the sample points from an $x$–$y$–$z$ coordinate system to an $a^\perp$–$b^\perp$–$z$ coordinate system in which the sector boundaries are perpendicular to the $a^\perp$ and $b^\perp$ axes. 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 sector $\sigma_i$. 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 $b$.

[2]An **upper convex hull**, or simply **upper 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.

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 $e_0, \ldots, e_{s-1}$ of $s$ elevation angles, where $e_i$ is the maximum of the elevations of the points that fall within sector $\sigma_i$ of $p$. In this model, the horizon is a piecewise–constant function of azimuth angle.

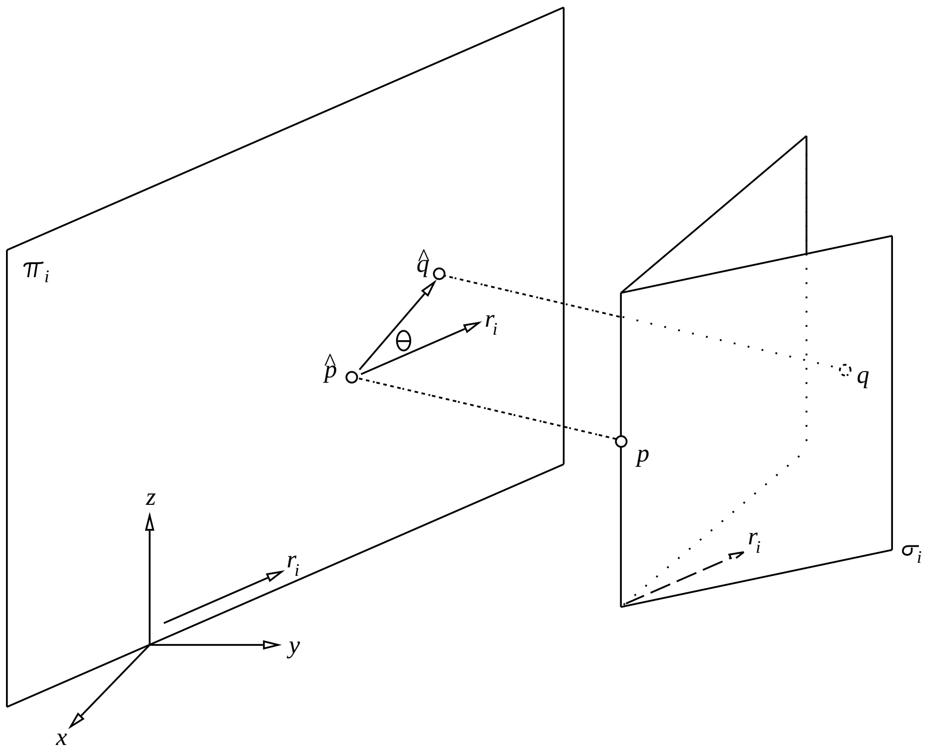

Fig. 1. Shown is a point $p$ of the terrain and the sector $\sigma_i$ of $p$; point $q$ falls within the sector. The horizontal vector $r_i$ bisects the sector and is parallel to the vertical plane $\pi_i$, which passes through the origin. The projections of $p$ and $q$ onto $\pi_i$ are denoted $\hat{p}$ and $\hat{q}$, respectively. The elevation angle of $q$ in sector $\sigma_i$ of $p$ is denoted $\Theta$.

## IV. The Algorithm

The algorithm iterates over the $s$ sectors. Each sector is treated independently (in effect, the algorithm starts anew for each sector). In the $i^{th}$ iteration, the maximum elevation $e_i$ in sector $\sigma_i$ is calculated for each of the $n$ sample points.

For a particular point $p$, this is accomplished as follows. All sample points $q_j$ that fall within sector $\sigma_i$ of $p$ are projected onto points $\widehat{q_j}_i$ in the plane $\pi_i$. A **covering set** of

$\mathcal{O}(n \log n)$. 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.

The algorithm is not difficult to implement: It took the author less than a week. The most complicated operations required are to insert a point into a convex hull and to compute a tangent from a point to a convex hull. Experimental results in Section V show that, as the number of points increases, the new algorithm's execution time grows much more slowly than that of the CMS algorithm. Thus, the new algorithm is better suited for very large terrains.

The next two sections describe the horizon computation algorithm. Following that, implementation results and rendered images 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.

Refer to Fig. 1 for the following definitions. 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."

The **azimuth angle** of a vector $v$ in $\Re^3$ 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 $\sigma_i$ of a point** $p$ is the region of $\Re^3$ 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 $r_i$ be a horizontal vector pointing in the direction $\frac{2\pi}{s}\ (i+\frac{1}{2})$. Let $\pi_i$ be the vertical plane that passes through the origin and embeds $r_i$. Let the notation $\hat{v}_i$ denote the projection of a vector $v$ onto $\pi_i$. See Fig. 1.

The **elevation angle** of a point $q$ that falls within sector $\sigma_i$ of a point $p$ is the angle between $r_i$ and the projection $\widehat{pq}_i$ of the vector $pq$ (angle $\Theta$ in Fig. 1). The angle has the

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 [4] and of Cabral, Max, and Springmeyer [5] can also 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 in 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. 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.

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 the maximum elevation stored so far. Such an algorithm is exorbitantly expensive for even moderately sized terrains: The author's implementation took 9.5 hours on a $100,000$–point terrain and would have taken 792 hours (i.e. 33 days) on a $915,800$–point terrain if run to completion.

Note that other, more complicated, algorithms that treat each point individually will be even slower (for example, [10] and [14] would run in $\mathcal{O}(n^2 \log n)$ time). However, the sampling methods of Max [4] and Cabral, Max, and Springmeyer [5] only take time $\mathcal{O}(sn^{1.5})$, since each of the $n$ points must consider $\mathcal{O}(\sqrt{n}\,)$ other points in each of $s$ directions around it.

This paper presents a much 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 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 [5] 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. This will be called the *CMS* approach. In Section V the performance of CMS is compared to the approach presented in this paper.

Shaded terrains have also been generated using shadow volumes (for example, Kaneda *et al* [6]) and ray tracing (for example, Coquillart and Gangnet [7]). Shadow volumes only apply to point light sources, and ray tracing can be expensive if sampling is used to incorporate diffuse illumination [8].

Related 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 [9]. The horizon has complexity $\mathcal{O}(n\ \alpha(n))$ [1] and can be computed in optimal time $\mathcal{O}(n \log n)$ [10]. Alternatively, a hierarchical model can be used to represent the surface at various resolutions [11], [12], [13]. In this model, the horizon can be computed for a single point at different resolutions in time $\mathcal{O}(n \log n)$ [14].

Cole and Sharir [15] describe how to preprocess a terrain in order to answer efficiently ray shooting queries from a fixed point and from a fixed vertical line. Bern, Dobkin, Eppstein, and Grossman [16] also discuss ray shooting queries from vertical lines in terrains. However, none of these works in computational geometry considers computing the horizons at *all* sample points of a terrain.

## *A. Approximate Horizons*

For accurate rendering, 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 $\mathcal{O}(n\ \alpha(n))$.

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

ity computation is desired, the horizon method provides the important initial irradiance of each point, which will speed up the radiosity computation considerably.

This paper describes how to compute efficiently the horizon at every sample point of a terrain. The method has been implemented and tested on terrains of up to 915, 800 points, the largest available to the author. On the largest terrain, the method took 2.9 hours on a 166 MHz Pentium PC, which compares quite favorably to two other methods (described in the next section) which took 7.2 hours and 33 days, respectively.

Images have been generated using an approximate illumination model [3] that requires only the direct primary illumination. From the horizon information, the radiosity of the sample points can be computed using this model at a rate of 3600 points per second for a discretized distribution of sky illumination.

## II. Background

Closely related work has been done by Max [2], [4], 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. When 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. Supersampling on pixels is performed to diminish aliasing artifacts. 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

**Abstract**

To render a terrain or a self–shadowing texture map whose topography is represented by many sample points, we must determine what part of the sky is visible from each sample point. The direct primary illumination of each point can then be computed for any distribution of sky illumination. This paper presents an efficient and practical method to compute the horizon, or skyline, at all sample points of a terrain.

From the horizons, the radiosity of each point can be determined in two ways: either quickly, using an approximate lighting model that relies upon only the direct primary illuminations, or more accurately, with a progressive radiosity algorithm that uses the direct primary illuminations as the initial point irradiances, for a considerable savings in time.

The new horizon computation method is compared with other methods by Max and by Cabral, Max, and Springmeyer. Experimental results are reported which show that the new method yields more accurate rendered images and takes much less time to process large terrains.

Additional images and animations for this paper are available at
`http://www.dgp.toronto.edu/people/JamesStewart`

**Keywords**

terrain, self shadowing texture map, shadows, displacement map, rendering, horizon, skyline

## I. Introduction

Terrain rendering is used in texture mapping, flight simulation, scientific visualization, and cartographic applications. For example, otherwise unobtainable views from the surface of Mars have been generated from terrain data [1]. Also, self–shadowing textures have been generated with a combination of bump maps and horizon maps [2].

To render a terrain or texture map for which the topography is represented by many sample points of various heights, we must compute the total light received by each sample point (its irradiance) in order to determine how much is emitted (its radiosity). The largest contribution to the irradiance of a sample point usually comes from the sky, although smaller contributions come from other surfaces of the terrain.

To determine the direct primary illumination from the sky, it is necessary to determine the horizon, or boundary, of the visible sky. However, each sample point usually sees a different horizon.

Once the horizon is computed for each sample point, we can quickly compute the direct primary illumination of each point for *any* distribution of sky light. As shown in Section VI, the direct primary illumination is sufficient to render realistic and informative images without resorting to an expensive radiosity step. However, if a progressive radios-

# Fast Horizon Computation for Accurate Terrain Rendering

A. James Stewart

Dynamic Graphics Project, Department of Computer Science, University of Toronto, 10 Kings College Road, Toronto, Ontario, Canada, M5S 1A4. phone: (416) 978–5359. fax: (416) 978–5184. email: jstewart@cs.toronto.edu