from Eurographics Rendering Workshop, June 1997

Hierarchical Visibility in Terrains

A. James Stewart

Dynamic Graphics Project
Department of Computer Science
University of Toronto
jstewart@dgp.toronto.edu

Abstract: This paper describes a hierarchical visibility technique that significantly accelerates terrain rendering. With this technique, large parts of the terrain that are hidden from the viewpoint are culled, thus avoiding the expense of uselessly sending them down the graphics pipeline (only to find in the z–buffer step that they are hidden). The hierarchical visibility technique has been implemented in a multiresolution terrain rendering algorithm and experimental results show very large speedups in some situations.

1 Introduction

In rendering a terrain, the greatest time is spent sending polygons down the graphics pipeline, where they are transformed, clipped, rasterized, shaded, and z–buffered.

This paper describes *hierarchical visibility*, a technique that efficiently culls terrain regions that are not visible from the current viewpoint. It avoids altogether the overhead of sending these regions down the graphics pipeline. The technique is hierarchical in the sense that terrain regions are grouped in an implicit quadtree and whole subtrees may be excluded from display by performing a simple test at the root of the subtree.

The hierarchical visibility technique has been combined with a multiresolution terrain rendering algorithm. Implementation results show large speedups when the visibility test is enabled.

2 Previous Work

Other work in terrain rendering has concentrated on reducing the load in the graphics pipeline by using larger, coarser triangles for distant parts of the terrain. For example, Lindstrom *et al* [1] vary the surface geometry of a regular square grid (RSG) to reduce the number of displayed triangles, while carefully maintaining bounds on the error between the reduced and full images. De Floriani and Puppo [2] and Scarlatos and Pavlidis [3] describe a hierarchical structure to represent triangulated irregular networks (TINs) at various resolutions. Gross, Gatti, and Staadt [4] locally vary the level of surface detail depending upon a measure of the surface's importance.

For general scenes, Koenderink and van Doorn [5] introduced the aspect graph, which describes the qualitatively different views of a scene; it has formed the basis for much of visibility research. For example, Platinga and Dyer [6] present an algorithm to compute the aspect graph for a polyhedral scene and describe bounds on its size, Gigus and Malik [7] describe an efficient algorithm to compute the aspect graph, and Gigus, Canny, and Seidel [8] describe how to efficiently compute the different views.

Coorg and Teller [9] use a subset of the aspect graph to identify changes in visibility; they also use an oct–tree to update this set efficiently and incrementally as the viewpoint moves through the scene. Earlier work by Teller and Séquin [10] uses a "stab tree" to describe cell–to–cell visibility in an architectural model. Teller [11] extends this with a description of the exact visibility through a sequence of portals. Chen and Wang [12] modify the standard BSP to minimize polygon splitting and, hence, the number of rendered polygons.

In other work, a hierarchical description of image space (a quad–tree sitting on top of the image pixels) is used to accelerate visibility computations. Meagher [13] uses a oct–tree to describe a solid object; each oct–tree cube inside the object is projected onto the finest enclosing quad–tree node of the image and refined as necessary. Greene, Kass, and Miller [14] partition a polygonal scene with an oct–tree. The polygons inside an oct–tree cube are not rendered if the projection of the cube falls inside a quad–tree node that is covered by a closer object.

In computational geometry, de Berg and Dobrindt [15] use a Kirkpatrick mesh decimation technique [16] to generate a hierarchy of different levels of terrain detail which fit together smoothly. Cole and Sharir [17] preprocess a terrain to answer ray shooting queries from a fixed point and from a fixed vertical line. Bern, Dobkin, Eppstein, and Grossman [18] perform ray shooting queries from vertical lines in terrains and also describe how to detect all qualitative changes in visibility experienced by a moving viewpoint on a straight trajectory through a general scene. These works require that the viewpoint trajectory be part of the input and, hence, don't apply to the typical case in terrain navigation where the viewpoint is controlled in real time by the user.

3 Terrain Rendering

This section describes a fairly simple terrain rendering algorithm and introduces the hierarchical terrain represention. The real contribution of this paper lies in the following Sections 4 and 5, which describe hierarchical visibility and its application to the terrain rendering algorithm.

In this paper, a terrain consists of a regular square grid of dimensions $0 \ldots N_x - 1$ in the x direction and $0 \ldots N_y - 1$ in the y direction. Each grid point, (x, y), has an associated height, $z(x, y)$ and a mesh covers the points $(x, y, z(x, y))$.

The grid is represented by quads at various resolutions, where a **quad** is a rectangular region in the $x - y$ plane. Let $Q^i_{x,y}$ be a quad of resolution i. At the finest resolution, 0, quad $Q^0_{x,y}$ is the square with vertices

$$(x, y), (x + 1, y), (x + 1, y + 1), (x, y + 1).$$

At resolution i, quad $Q^i_{x,y}$ is a $2^i \times 2^i$ square with vertices

$$(2^i x, 2^i y), (2^i(x+1), 2^i y), (2^i(x+1), 2^i(y+1)), (2^i x, 2^i(y+1)).$$

Quad vertices are clamped to the range $[0, N_x - 1] \times [0, N_y - 1]$, so it's possible to have a non–square quad at the terrain boundary. Quad $Q^i_{x,y}$ is the union of its four **subquads**,

$$Q^{i-1}_{2x,2y}, \quad Q^{i-1}_{2x+1,2y}, \quad Q^{i-1}_{2x+1,2y+1}, \quad \text{and} \quad Q^{i-1}_{2x,2y+1}.$$

The terrain rendering algorithm produces a set of disjoint quads whose union covers the whole terrain. Each quad is triangulated and the corresponding triangles (with 3D vertices) are sent down the graphics pipeline to produce an image. The goal of the algorithm is to produce a small set of quads whose image is perceptually close to the image of the set of all resolution–0 quads.

Each vertex, (x, y), has an associated **level of detail**, $\text{LOD}(x, y)$, which is the maximum resolution at which quads adjacent to (x, y) may be rendered. $\text{LOD}(x, y)$ is chosen to ensure that quads adjacent to (x, y) don't span more than some number of pixels when projected onto the screen. $\text{LOD}(x, y)$ is the logarithm of the distance between the viewpoint and the terrain point, $(x, y, z(x, y))$, plus a constant, τ. The constant, τ, is chosen by the user to strike a balance between accuracy and speed. A small τ yields high resolution images at the cost of rendering smaller (and hence more) quads.

The terrain rendering procedure starts with the quad $Q^k_{0,0}$ of coarsest resolution: $k = \lceil \log_2(\max(N_x, N_y) - 1) \rceil$. In general, to treat quad $Q^i_{x,y}$, the algorithm first tests whether the quad intersects the view frustum; if not, it returns. Then the algorithm tests whether the quad is visible (as will be discussed in the following sections); if not, it returns. Otherwise, $\text{LOD}()$ is computed at each vertex of $Q^i_{x,y}$. If the minimum such level of detail is greater than or equal to i, the quad is drawn at resolution i. Otherwise, the algorithm is called recursively for each of $Q^i_{x,y}$'s four sub–quads.

A quad is triangulated before being rendered. If the quad shares an edge with finer–resolution quads, that edge must be split and the quad's triangulation must incorporate the splitting vertices. This is commonly known as *anchoring* (see, for example, Baum *et al* [19]).

4 Visibility of a Single Point

This section and the next describe the hierarchical visibility technique and its application to the terrain rendering algorithm of Section 3.

Consider a single terrain point $p = (x, y, z(x, y))$. The **horizon** at p is the boundary between the visible sky and the terrain, as seen from p. The horizon in a particular azimuth direction[1], ϕ, can be represented by the terrain point, $h_p(\phi)$,

[1] 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 in the $-z$ direction.

that lies on the horizon in that direction. Let $R_p(\phi)$ be the vertical planar region that lies below the line through p and $h_p(\phi)$ and that lies to the side opposite p of the vertical line through $h_p(\phi)$. See Figure 1.

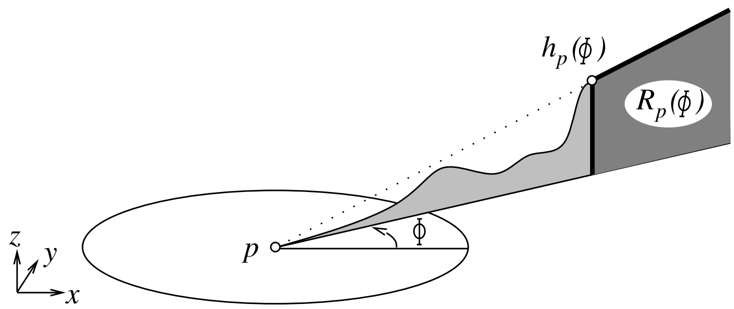

Fig. 1. A vertical cross section of the terrain in azimuth direction ϕ from point p. Point $h_p(\phi)$ lies on the horizon in direction ϕ. p is hidden if the viewpoint lies in region $R_p(\phi)$.

We make the following observation about visibility: *Point p is hidden if the viewpoint lies in $R_p(\phi)$.* Hence, $R_p(\phi)$ is called an **occlusion region**. Note that there may be viewpoints outside $R_p(\phi)$ from which p is also hidden (for example, viewpoints behind the first ridge in Figure 1), so this is a conservative visibility test.

We could exploit this observation by storing at each terrain point a representation of its horizon. Then, to test whether a quad is visible, we could simply test whether each of its four vertices is visible. If no vertex is visible, we would not render the quad.

One objection to this approach is that storing the complete horizon at each terrain point would consume huge amounts of memory. Another objection is that a quad — particularly a large, coarse resolution quad — might be partially visible while its four vertices are hidden. The test would incorrectly discard such a quad. To address these objections, we introduce hierarchical visibility.

5 Hierarchical Visibility

The key idea is to store a small set of occlusion regions with each quad, $Q^k_{x,y}$, of each resolution, k. For a particular quad, this set of occlusion regions describes the conservative visibility in the space around the quad.

In a preprocessing step, occlusion regions are computed at each of the terrain points. These are combined to form occlusion regions for quads of each resolution.

5.1 Occlusion Regions for Terrain Points

The range of azimuth angles around a point p is divided into s sectors, where sector i spans angles in $[\ \frac{2\pi}{s}\ i, \frac{2\pi}{s}\ (i+1)\ ]$. Each sector, i, of p stores a single 2D occlusion region, $R_p(i)$, which is defined by a 2D horizon point, $h_p(i)$.

The horizon in a sector, i, typically varies in elevation across the width of the sector. Let $h_{min}(i)$ be the point of minimum elevation on the horizon in sector i. Then $h_p(i)$, which defines $R_p(i)$, will be based on $h_{min}(i)$ as follows:

Referring to Figure 2, let r_i be a horizontal vector parallel to the centerline of sector i in the direction $\frac{2\pi}{s}\ (i+\frac{1}{2})$. Let π_i be the vertical plane that passes through the origin and embeds r_i. Let $\hat{p}$ be the perpendicular projection of p onto π_i.

$h_p(i)$ is the perpendicular projection of $h_{min}(i)$ onto π_i. The occlusion region, $R_p(i)$, is defined with respect to $h_p(i)$ as described in Section 4: $R_p(i)$ is the region of π_i below the line through $\hat{p}$ and $h_p(i)$ and to the side opposite $\hat{p}$ of the vertical line through $h_p(i)$.

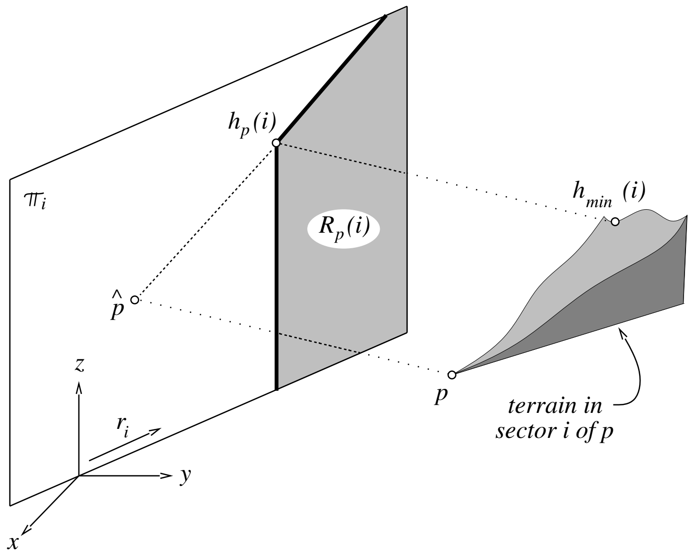

Fig. 2. The horizon point, $h_{min}(i)$, of minimum elevation in sector i is projected onto the plane, π_i, that passes through the origin and is parallel to the centerline of the sector. The occlusion region, $R_p(i)$, of the sector is shaded in the figure.

Given $R_p(i)$ for each sector $i \in \{0, 1, \ldots, s-1\}$, we determine whether p is visible as follows: Determine which sector, j, of p the viewpoint lies in. Project the viewpoint onto the corresponding plane, π_j. If the projected viewpoint falls within $R_p(j)$, p is hidden.

This visibility test is conservative: It might conclude that p is visible when, in fact, it is occluded by a point on the horizon that is higher that $h_{min}(i)$. This is fine, since we only discard a quad when it is not visible.

We're assuming that all points on the horizon of a sector have the same horizontal distance from the sector vertex, p. If, however, $h_{min}(i)$ is closer than all other horizon points, the viewpoint might project to $R_p(i)$ even when p is visible (i.e. the viewpoint is below the elevation of $h_{min}(i)$ and more distant from p, but still in front of the horizon). This can occur with wide sectors. The solution is simple: Let $h_p(i)$ have the same elevation as $h_{min}(i)$, but let its horizontal distance from $\hat{p}$ be that of the most distant horizon point from p.

5.2 Finding the Minimum–Elevation Horizon Point in a Sector

We determine the minimum–elevation point, $h_{min}(i)$ as follows: The horizon within a sector is approximated by a piecewise–constant elevation function. The sector is divided into many very narrow **subsectors**, each typically spanning $\frac{1}{2}$ a degree or less. In each subsector, the horizon elevation is taken to be constant and is defined by the terrain point of maximum elevation within that subsector. Narrow subsectors might not contain any terrain points, so we also consider terrain points that lie immediately adjacent to, but outside, the subsector. Then $h_{min}(i)$ is the terrain point defining the *minimum*–elevation one of these subsectors.

Let $h_p(i,k)$ be the terrain point of maximum elevation in subsector k of sector i of p. Several algorithms exist to determine $h_p(i,k)$ for every subsector, (i,k), of every terrain point, p. Here's one:

> For each point p, consider each other point q. Determine which subsector, (i,k), of p contains q and set $h_p(i,k) = q$ if the elevation of q is greater than that of the current $h_p(i,k)$.

Such an algorithm takes $\mathcal{O}(n^2)$ time in a terrain of n points. It is exorbitantly expensive for even moderately sized terrains (the author's implementation took 9.5 hours on a $100,000$–point terrain). Two other algorithms, due to Max [20] and to Cabral, Max, and Springmeyer [21], sample along the centerline of each sector. These algorithms take $\mathcal{O}(n^{1.5})$ time, but suffer from undersampling artifacts, since they miss high–elevation points that don't lie on the centerline. A recent algorithm by the author [22] computes the horizon points in $\mathcal{O}(n \log^2 n)$ time and doesn't suffer from undersampling.

The computation of $h_{min}(i)$, for all points and sectors, i, is performed in a preprocessing step using one of these algorithms. For example, the author's algorithm takes 1.3 hours for a 100,000–point terrain using 64 sectors with 4 subsectors each.

5.3 Occlusion Regions for Quads

A quad $Q^k_{x,y}$ is said to **cover** the terrain points (u,v) in the range $[2^k x, 2^k(x+1)] \times [2^k y, 2^k(y+1)]$. We would like to answer the query "Are *all* terrain points covered by quad $Q^k_{x,y}$ hidden from the viewpoint?" If the answer is affirmative, the quad need not be displayed.

Each quad, Q, stores occlusion regions $R_Q(i)$ for $i \in \{0, 1, \ldots, s-1\}$. For a particular i, we define $R_Q(i)$ to be the intersection of the occlusion regions $R_{(u,v)}(i)$ of all the terrain points, (u, v), covered by Q. Since each occlusion region, $R_{(u,v)}(i)$, is a 2D region in the *same* plane π_i, the intersection of these occlusion regions is well defined.

Among the terrain points covered by a particular quad, the viewpoint can appear in different sectors. We determine, for each of the four vertices of the quad, in what sector the viewpoint lies. Let $[i_{min}, i_{max}]$ be the smallest contiguous range (modulo s) that contains these sectors. Then, with respect to each terrain point covered by the quad, the viewpoint lies in some sector in the range $[i_{min}, i_{max}]$.

We can thus say that all terrain points covered by quad Q are hidden if the viewpoint lies in $R_Q(i)$ for *every* i in the range $[i_{min}, i_{max}]$.

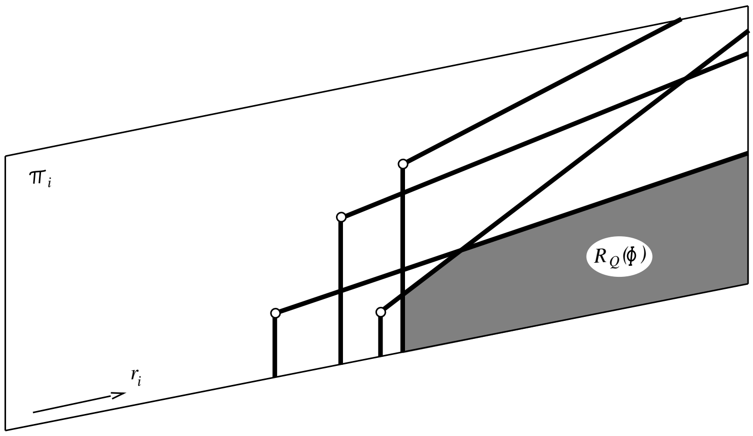

Fig. 3. Occlusion region $R_Q(i)$ of quad Q^0 is the intersection of the occlusion regions of all terrain points covered by Q^0. In the figure, the shaded region is the intersection of the occlusion regions of four terrain points.

As shown in Figure 3, the intersection of the convex occlusion regions is itself convex. Experiments have shown that, even in moderately sized terrains of 100,000 points, the occlusion regions are bounded by two to three edges on average and, very rarely (in less than 0.1% of regions) by a maximum of 16 edges. An occlusion region can be stored as a compact array of the 2D vertices of the convex region, ordered by increasing horizontal distance from the origin (the last point in the array is interpreted as the direction of the last edge, which extends to infinity). To test whether a projected viewpoint is inside such an occlusion region, it is sufficient to find the two vertices between which the projected viewpoint lies and to determine whether it is above or below the edge joining those two vertices.

A preprocessing step computes the occlusion regions of all the sectors of all the quads at all resolutions. The regions of quad Q^k are most efficiently computed by intersecting of the regions of Q^k's four subquads. Thus, occlusion regions are built up in order of coarsening resolution.

Quads of fine resolution cover a small area and there are typically a very large number of them. If memory is limited, the occlusion regions associated with these quads may be discarded. Experiments have shown that there is only a slight performance penalty for discarding occlusion regions corresponding to resolutions 0, 1, and 2. Since the average size of an occlusion region does not vary greatly with resolution, occlusion regions at resolution k typically use four times the memory of occlusion regions at resolution $k + 1$ and it is very advantageous to discard occlusion regions of low (fine) resolution.

5.4 Hierarchical Visibility for Terrain Rendering

A quad visibility test is used by the hierarchical terrain rendering algorithm of Section 3. As described in the section above, this is implemented by checking whether the projected viewpoint falls within every one of a small range of the quad's occlusion regions. If so, the function returns `false`. If not, or if no occlusion regions are stored at resolution i, the function returns `true`.

The two objections stated at the end of Section 4 have been addressed: Huge amounts of memory are not necessary because we can adaptively choose to store as many levels of occlusion regions as fit into memory, starting with the coarsest resolutions which require the least memory. Secondly, visibility in a large, coarse quad takes into account the visiblity of *all* points covered by the quad, not just the four vertices of the quad, so we will not mistakenly discard a quad.

The real advantage of hierarchical visibility is that large regions of the terrain can safely be discarded without even touching the underlying terrain points. When flying through a valley, for example, the valley walls typically occlude the vast majority of terrain points. Hierarchical visiblity detects this and greatly reduces the number of quads sent down the graphics pipeline.

6 Experimental Results

The hierarchical rendering algorithm, including the hierarchical visiblity test, was implemented in about 4,500 lines of C++ code using OpenGL on a Linux system. The preprocessing step was implemented in a separate program of about 2,500 lines. Horizons at individual terrain points were computed with an algorithm of the author [22] and passed to the preprocessing step to build the occlusion regions. Upon starting, the terrain rendering algorithm reads these occlusion regions from disk.

Experiments were performed on two terrains. One terrain contained 100,000 points and is shown in Figures 4, 5, and 6. The other terrain contained 472,360 points and is a superset of the smaller terrain, but with a very large, fairly flat

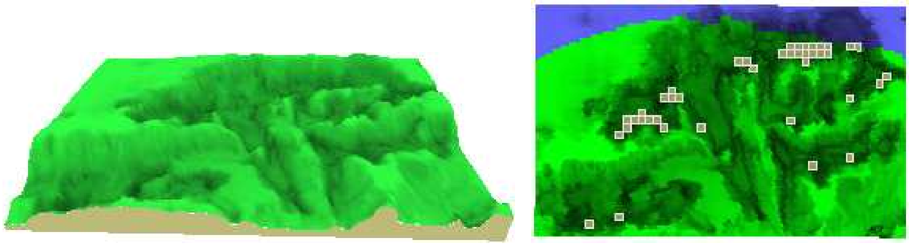

Fig. 4. *Viewpoint C:* A high–altitude view of a 250×400 terrain. On the left is the rendered image; on the right is an overhead view showing which parts were rendered. One visibility test culled each of the outlined regions on the right. There is little hidden from the viewpoint, so only about 2,000 of the 25,000 quads are culled. (Due to the distance of the viewpoint from the terrain, quads of size 2×2 were rendered.)

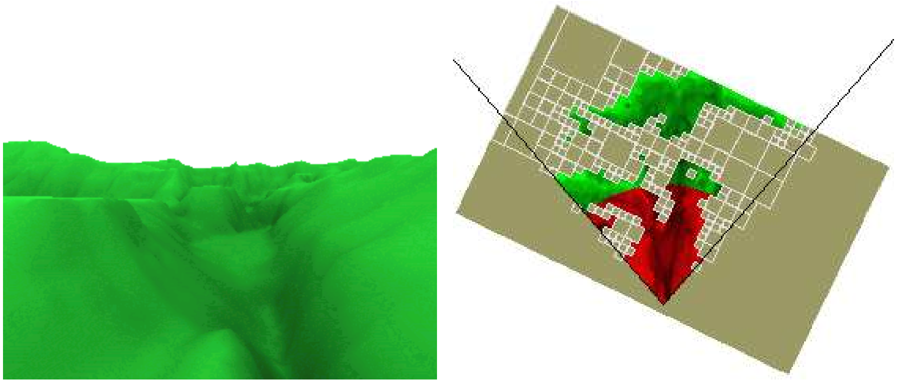

Fig. 5. *Viewpoint A:* A view from a valley in the center of Figure 4, looking straight ahead. Hierarchical visibility culls about 12,000 of the 21,000 quads that would otherwise be displayed. The hierarchical nature of the visibility tests is apparent on the right, where one such visibility test culled each of the outlined regions. The two black lines bound the view frustum.

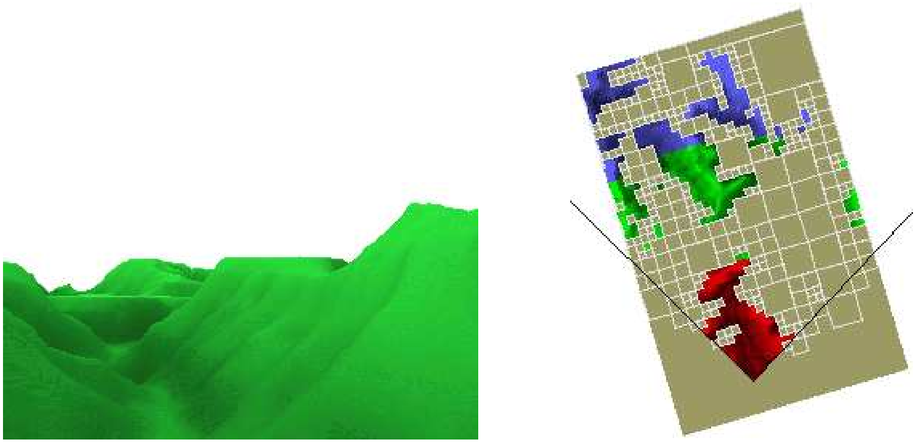

Fig. 6. *Viewpoint B:* A view from another valley, this one in the right of Figure 4, looking to the left. There are many fine–resolution quads that are hidden by the valley wall to the right of the viewpoint, so hierarchical visibility is very effective, culling about 50,000 of the 70,000 quads that would otherwise be displayed.

plateau on which there is little occlusion. Horizons of 64 sectors were used in both cases and visibility was tested for quads of size 8×8 and larger. The algorithm was run on two machines: a 166 MHz Pentium running Linux and rendering in software with Mesa (an OpenGL implementation) and a 250 MHz SGI High Impact rendering in hardware with OpenGL.

Figure 7 shows the rendering times with the visibility test enabled and disabled for six viewpoints. Viewpoints A, B, and C are in the 100,000–point terrain, while viewpoints D, E, and F are in the 472,360–point terrain.

Where there is a fair amount of occlusion (viewpoints A, B, and D) the visibility test saves a significant amount of time in both hardware and software rendering. Where there is little occlusion (viewpoints C, E, and F) there is no penalty for the visibility test in software rendering and only a slight penalty in hardware rendering.

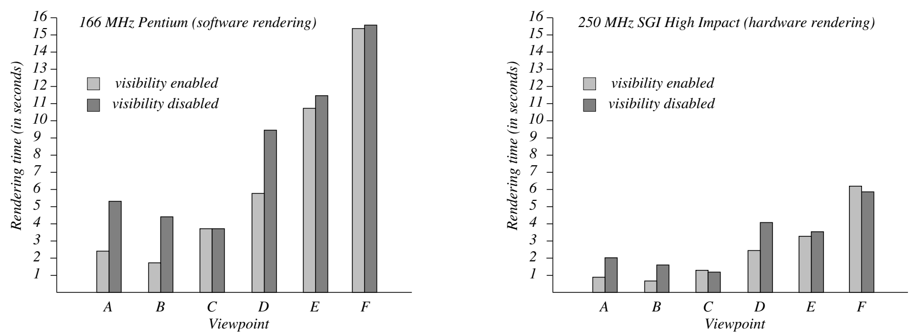

Fig. 7. Rendering times on two machines from various viewpoints with the visibility test enabled and disabled. Viewpoints A, B, C are in a 100,000–point terrain, while D, E, and F are in a 472,360–point terrain. Viewpoints A, B, and D have a fair amount of occlusion, while there is little occlusion from viewpoints C, E, and F.

Visibility tests were performed on quads of resolutions 3, 4, 5, 6, and 7 (from size 8×8 to size 128×128). The corresponding occlusion regions (64 regions for each quad) required a total of 4.1 Mb of memory for the small terrain and 20.2 Mb for the large terrain. Discarding the finest occlusion regions of resolution 3 brought the required memory down to 1.0 Mb and 5.4 Mb, respectively (the finest regions are also the most numerous). A terrain rendering application would typically load as many levels of occlusion regions as would comfortably fit into memory, starting from the coarsest level.

7 Discussion

Many issues remain to be addressed. The quad rendering could be accelerated by exploiting the “triangle strip” rendering primitive since, currently, each quad is rendered separately. It is an interesting and, I think, a difficult problem to render the visible parts of the terrain with the fewest triangle strips. Another, more general problem, is that larger terrains that are not stored in main memory require a real–time paging strategy. Such a strategy would have to consider level of detail and visibility.

The hierarchical visibility algorithm is of a different flavour from that of Coorg and Teller [9]. They exploit the coherence of viewpoint positions to update in real time a conservative visibility data structure. The algorithm of this paper precomputes and stores all the visibility information and doesn't exploit viewpoint coherence at all. These two algorithms take different approaches to the tradeoff between inter–frame computation time and memory requirements.

The experiments show that the acceleration due to hierarchical visibility varies with the amount of occlusion, just as one would expect. Hierarchical visibility is particularly effective when the viewpoint is in a valley (Viewpoints A and B) or very close to the ground (Viewpoint D). Hierarchical visibility seems to be an effective technique to accelerate low–level flight simulation and ground vehicle simulation.

8 Acknowledgments

I wish to thank the three reviewers, whose many good suggestions I've tried to incorporate. I also wish to thank Mike Caplinger of Malin Space Science Systems for providing the terrain data, which comes from Mars. This data 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.

References

1. P. Lindstrom, D. Koller, W. Ribarsky, L. F. Hughes, N. Faust, and G. Turner, "Real-Time, continuous level of detail rendering of height fields", in *Computer Graphics (SIGGRAPH '96 Proceedings)*. ACM SIGGRAPH, August 1996, pp. 109–118, held in New Orleans, Louisiana, 04-09 August 1996.
2. 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.
3. 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.
4. M. H. Gross, R. Gatti, and O. Staadt, "Fast multiresolution surface meshing", in *Proceedings of Visualization '95*, October 1995, pp. 135–142.
5. J. J. Koenderink and A. J. van Doorn, "The singularities of the visual mapping", *Biological Cybernetics*, vol. 24, pp. 51–59, 1976.

6. H. Plantinga and C. R. Dyer, "Visibility, occlusion, and the aspect graph", *International Journal of Computer Vision*, vol. 5, no. 2, pp. 137–160, November 1990.
7. Ziv Gigus and Jitendra Malik, "Computing the aspect graphs for line drawings of polyhedral objects", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 12, no. 2, pp. 113–122, February 1990.
8. Ziv Gigus, John Canny, and Raimund Seidel, "Efficiently computing and representing aspect graphs for polyhedral objects", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 13, no. 6, pp. 542–551, June 1991.
9. Satyan Coorg and Seth Teller, "Temporally coherent conservative visibility", in *Proc. 12th Annu. ACM Sympos. Comput. Geom.*, 1996, pp. 78–87.
10. Seth J. Teller and Carlo H. Séquin, "Visibility preprocessing for interactive walkthroughs", in *Computer Graphics (SIGGRAPH '91 Proceedings)*, Thomas W. Sederberg, Ed., July 1991, vol. 25, pp. 61–69.
11. Seth J. Teller, "Computing the antipenumbra of an area light source", in *Computer Graphics (SIGGRAPH '92 Proceedings)*, Edwin E. Catmull, Ed., July 1992, vol. 26, pp. 139–148.
12. H-M Chen and W-T Wang, "The feudal priority algorithm on hidden-surface removal", in *Computer Graphics (SIGGRAPH '96 Proceedings)*, August 1996, pp. 55–64, held in New Orleans, Louisiana, 04-09 August 1996.
13. Donald J. Meagher, "Efficient synthetic image generation of arbitrary 3–D objects", in *Proceedings of the IEEE Conference on Pattern Recognition and Image Processing*, June 1982, pp. 473–478.
14. Ned Greene, Michael Kass, and Gavin Miller, "Hierarchical Z-buffer visibility", in *Computer Graphics (SIGGRAPH '93 Proceedings)*, James T. Kajiya, Ed., Aug. 1993, vol. 27, pp. 231–238.
15. M. de Berg and K. Dobrindt, "On levels of detail in terrains", in *Proc. 11th Annu. ACM Sympos. Comput. Geom.*, 1995, pp. C26–C27.
16. D. G. Kirkpatrick, "Optimal search in planar subdivisions", *SIAM J. Comput.*, vol. 12, pp. 28–35, 1983.
17. R. Cole and M. Sharir, "Visibility problems for polyhedral terrains", *J. Symbolic Comput.*, vol. 7, pp. 11–30, 1989.
18. M. Bern, D. Dobkin, D. Eppstein, and R. Grossman, "Visibility with a moving point of view", *Algorithmica*, vol. 11, pp. 360–378, 1994.
19. Daniel R. Baum, Stephen Mann, Kevin P. Smith, and James M. Winget, "Making radiosity usable: Automatic preprocessing and meshing techniques for the generation of accurate radiosity solutions", in *Computer Graphics (SIGGRAPH '91 Proceedings)*, Thomas W. Sederberg, Ed., July 1991, vol. 25, pp. 51–60.
20. N. L. Max, "Horizon mapping: shadows for bump-mapped surfaces", *The Visual Computer*, vol. 4, no. 2, pp. 109–117, July 1988.
21. 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.
22. A. James Stewart, "Fast horizon computation for accurate terrain rendering", Tech. Rep. TR 349, Department of Computer Science, University of Toronto, June 1996, available from `http://www.dgp.toronto.edu/people/JamesStewart`