Horizon Occlusion Culling for Real-time Rendering of Hierarchical Terrains

Brandon Lloyd
Parris Egbert

Brigham Young University

Abstract

We present a technique to perform occlusion culling for hierarchical terrains at run-time. The algorithm is simple to implement and requires minimal pre-processing and additional storage, yet leads to 2-4 times improvement in framerate for views with high degrees of occlusion. Our method is based on the well-known occlusion horizon algorithm. We show how to adapt the algorithm for use with hierarchical terrains. The occlusion horizon is constructed as the terrain is traversed in an approximate front to back ordering. Regions of the terrain are compared to the horizon to determine when they are completely occluded from the viewpoint. Culling these regions leads to significant savings in rendering.

CR Categories and Subject Descriptors: I.3.3 [Computer Graphics]:Picture/Image Generation - Viewing Algorithms; I.3.7 [Computer Graphics]:Three-Dimensional Graphics and Realism - Hidden line/surface removal.

Additional Keywords: rendering algorithms, visibility, occlusion culling.

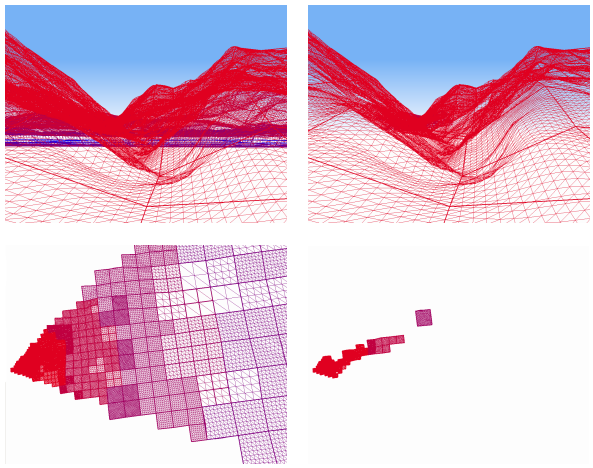

Figure 1: Top-left, view of a terrain without occlusion culling. Top-right, occlusion culling is enabled. The bottom row shows the terrain elements drawn for the corresponding views in the top row.

1. INTRODUCTION

Real-time navigation of detailed terrain models is important for many applications. Since the early days of computer graphics, the military has used flight simulators to train pilots. Flight simulators for commercial airliners soon followed. Simulator technology inevitably made its way into video games. Today real-time terrain navigation is routinely used in other applications such as visualizing a proposed road construction project or data captured by a satellite. The quality of these applications often depends on the size and detail of the terrain models.

The sheer size of terrain models presents several problems. Large models require a lot of memory. Models covering only a few square miles can consist of millions of polygons. Texture-maps for terrain models usually have a high resolution in order to provide sufficient detail at close proximity, which makes them very large. Often only part of the model can fit in memory at any time. Rendering large terrain models at real-time framerates can be challenging. While rendering hardware has made great advances in recent years, even the best hardware can render only a few million polygons per second. Memory bandwidth limitations restrict the amount of texture that can be rendered to a few hundred megabytes per second. These limitations are not a problem when the viewer is close to the ground looking down because large portions of the model fall outside the view frustum and may be culled away. However, when the viewer is looking at the model from high altitude or is looking out horizontally at ground level, arbitrarily large portions of the model may be visible. Since it is currently not possible to render millions of polygons with many megabytes or even gigabytes of texture at high frame rates, rendering large terrain models in realtime requires specialized algorithms.

One solution is to render the terrain using an adaptive level of detail (LOD). Features in the distance do not need to be rendered with the same fidelity as those that are close to the viewer. A smaller, coarser representation may be used for distant features without any noticeable degradation of image quality. Hierarchical terrain representations such a quadtrees are especially well-suited for adaptive LOD and have been used extensively in computer graphics and GIS systems. Quadtrees facilitate choosing a representation with an appropriate resolution for different parts of the model. Adaptive LOD can drop the number of polygons in a given frame from millions down to tens of thousands. Adaptive LOD yields similar exponential reductions in the amount of texture

{blloyd, egbert}@cs.byu.edu. 3366 TMCB, Provo, UT 84602

IEEE Visualization 2002 Oct. 27 - Nov. 1, 2002, Boston, MA, USA
0-7803-7498-3/02/$17.00 © 2002 IEEE

required for a given frame. This is because fewer, lower resolution texture maps can be used to cover large portions of the model.

While adaptive LOD dramatically reduces the polygon count for both high altitude views and horizontal views close to the ground, the polygon counts for horizontal views remain considerably higher than those for other views. This is a characteristic of terrain models. When viewed from above, all of the geometric detail becomes compressed in the viewing direction, so a coarser LOD may be used. Horizontal views require a finer resolution to capture the details of the terrain profiles which leads to a higher polygon count. This is unfortunate because these views tend to be very common in 3D terrain visualization applications. However, the rough features of a terrain that require an increased polygon count also afford an opportunity for optimization, since they often occlude large portions of the terrain model. Occluded regions need not be drawn because they make no contribution to the final image. Figure 1 shows the reduction in rendered geometry due to occlusion culling.

While some work has been done to perform occlusion culling for terrains, most algorithms pre-compute visibility and require significant pre-processing and storage. We present an algorithm to compute occlusions in terrains on-the-fly. It is based on a well-known technique. With a heightfield, anything below the horizon line is guaranteed to be occluded. By tracking the horizon line of the heightfield as it is rendered in front-to-back order, self-occlusions within the heightfield may be detected. The horizon effectively fuses the occlusion of many individual parts of the terrain. Terrain elements which fall completely below the horizon are culled. The rest of the terrain elements are rendered and are used to update the visiblity horizon as it sweeps through the scene. The contribution of this paper is the application of the occlusion horizon to hierarchical terrains. The algorithm requires minimal pre-processing and additional memory and is simple to implement.

After briefly discussing work related to this paper in Section 2, we give an overview of the terrain rendering algorithm we use in Section 3. Section 4 explains the details of the occlusion culling algorithm. In Section 5 we show the performance enhancements we obtained in rendering a large terrain model containing over 100 million triangles. Finally we make some concluding comments and discuss directions for future work.

2. RELATED WORK

A vast amount of research has been conducted in the area of visibility and occlusion culling. A recent paper by Cohen-Or et al. provides an excellent survey of visibility techniques [6]. Visibility algorithms can determine visibility either in a precomputation step or on-the-fly. Precomputed visibility algorithms pre-process a scene to determine which objects are visible from points or regions in space. At run time, only the objects in the pre-calculated set of visible objects for the current viewpoint are drawn. This can result in enormous savings when the size of the set of visible objects is small compared to the number of objects in the scene. The major drawbacks of pre-processing algorithms are that they are limited to static scenes, they can require a lot of storage space to store the visibility information, and the pre-processing step can be very time consuming. These algorithms have been successfully applied to indoor scenes because these environments are static and the potentially visible sets are usually small [1,22]. Cell-based precomputed visibility algorithms have been applied to outdoor scenes [7,23]. Schaufler et al. [20] present a general algorithm based on voxelization of occluders which can also be applied to terrains. Stewart [21] pre-computes visibility for hierarchical terrains. A representation of the visibility horizon is created for each vertex of the terrain in a pre-processing step. This information is used at run-time to determine whether a particular vertex is occluded from the current viewpoint.

Another class of occlusion algorithms calculates occlusions on-the-fly. Green et al. [13] and Zhang et al. [24] use hierarchical data structures to calculate occlusions on the pixel level. Bartz et al. [2] use hierarchical spatial subdivision and OpenGL features to detect occlusions, thus taking advantage of graphics hardware. Unfortunately, this approach is limited by expensive framebuffer reads. Hey et al. [14] use a low-resolution grid in combination with an occlusion buffer or z-buffer that is updated in a lazy manner in order to reduce the cost of expensive pixel-level queries. Coorg and Teller [9] classify large occluding polygons and use the support planes of these polygons to determine occlusion of other objects in the scene. Bitner et al. [3] propose a similar algorithm that uses a BSP data structure to merge occluders. Both of these methods require large polygons to be used as occluders. Since terrains typically consist of many small polygons, neither of the algorithms are suitable for terrain rendering.

Occlusion horizons can be used to render 2D-functions and heightfields. Downs et al. [12] show how to use an occlusion horizon for urban environments. Our algorithm also uses an occlusion horizon and thus is similar in many ways to theirs. However, we use an occluder representation that is better suited for large terrains. Our horizon representation is also simpler.

Terrain rendering is another area of research that has received much attention. Hierarchical terrain models may be built on top of a uniform rectangular grid in the form of a quadtree or a restricted quadtree, also called a bintree. These representations have the advantage of compact storage since connectivity and vertex positions are implicit in the regular grid. Quadtrees [5, 16, 19] and bintrees [11, 17] have been used to generate terrain models with an adaptive level of detail. Hierarchical terrains may also be built upon irregular triangle meshes. Since more freedom can be used in selecting the location of vertices, it is often possible to represent a terrain with fewer triangles with these methods than with those based on regular grids. Examples of these types of terrains include Delaunay triangulations [10,18] and view dependent progressive meshes [15]. Terrains can also be rendered efficiently using voxels [8].

3. TERRAIN RENDERING ALGORITHM

Our terrain visualization application is used to interactively view very large models stored on a central server. Texture and geometry data is transferred on demand to the client program over a network using the caching scheme described in [4]. For best performance it is important to minimize the amount of data sent over the network. On-the-fly occlusion culling is more appropriate for this type of application than precomputed visibility because no visibility information needs to be transferred to the client.

We render the terrain using a variation of quadtrees described by Cline and Egbert [5], primarily because it works well with the caching algorithm we are using. Other hierarchical terrain algo-

rithms could also have been used. One advantage of their method is that it eliminates the noticeable "pop" that occurs when the LOD for part of the terrain suddenly changes. Using a technique called quadtree morphing, or *q-morphing*, they perform a smooth morph between adjacent levels of detail. The morph is position-based and is therefore almost always unnoticeable because the morphing occurs when the viewer is in motion.

The algorithm uses a quadtree with small, regular grids at the nodes called *gridlets*. Gridlets have fixed spatial extents but may vary in resolution. To select which gridlets to draw for any particular view, the quadtree is traversed from the top down. Each gridlet is first checked against the view frustum for visibility. Gridlets lying completely outside the view frustum may be discarded along with all of their children. If the projected screen radius of a gridlet falls below a user specified threshold it is added to the drawing list. Otherwise the traversal continues with its children. Once a gridlet is placed in the drawing list, a screen-space error metric and the local surface roughness are used to compute an LOD parameter describing how finely the gridlet should be decimated in order to satisfy user specified error bounds. The LOD parameters for both the current gridlet and its parent are combined in such a way that the parameter varies smoothly as position changes. The integer part of the parameter is used to pick a subdivision level. The fractional part is used to morph from one subdivision level to the next. Interested readers are referred to [5] for the details of how the LOD parameter is computed.

The horizon occlusion culling algorithm is not restricted to this particular implementation of hierarchical terrains. Any algorithm that can bound the child geometry of a node in the hierarchy may be used.

4. HORIZON OCCLUSION CULLING

Horizon occlusion culling is incorporated in the visibility test of the rendering algorithm. After a gridlet passes the view frustum culling test, it is tested against the current visibility horizon. The horizon tracks the highest points in the screen space projection of the terrain rendered thus far. If the projection of the gridlet falls below the horizon, it is guaranteed to be hidden and can be culled along with all of its children. This can result in significant computational savings, especially when gridlets high in the quadtree can be culled. If any part of the gridlet lies above the horizon, the horizon is updated with the gridlet's occlusion profile and the gridlet is added to the drawing list. Generating the list of visible gridlets before rendering allows us to reorder the list in order to avoid texture memory thrashing by drawing gridlets with the same texture at the same time.

Algorithms based on the occlusion horizon consist of three major components:

1. A front-to-back ordering of the elements of the scene.
2. A representation of scene elements to be used for calculating occlusions. The elements themselves may be used, but they are often simplified for efficiency.
3. A representation for the visibility horizon.

We will now discuss in detail how each of these components are implemented in our algorithm.

4.1 Approximate Front-to-Back Traversal

The horizon occlusion algorithm requires that objects be rendered in front-to-back order. Due to the ordering imposed on the gridlets by the quadtree structure of the terrain, an approximate front-to-back ordering can be obtained simply by traversing the nodes in the correct order. We observe that a strictly front-to-back ordering is not necessary to obtain correct results. A correct traversal of the gridlet children need only ensure that each child is rendered before any of its siblings that it may occlude. As shown in Figure 2, the correct order depends on the slope of the line through the gridlet center, which in turn depends on which side of the vanishing point the gridlet center lies. For horizontal projections of axis-aligned terrain gridlets, the traversal ordering depends only on the quadrant of the gridlet center, relative to the viewpoint. The traversal orderings can be stored in a simple look-up-table for each quadrant.

4.2 Occluders for Terrain Elements

The visibility test is performed many times per frame and must be efficient. For this reason, the visibility tests are performed with simple approximations for the objects in the scene called *occludees* and *occluders*. The occludee is a conservative over-estimation of the extents of an object used for the visibility test. A conservative test may classify an occluded object as visible, which results in more processing time, but does not affect the final image. The occludee is conservative because whenever it falls completely below the horizon, so does the object, although there may be instances when part of the occludee lies above the horizon but the object itself does not. The occluder is a conservative under-estimate of the occlusion profile of the object, which is used to update the horizon.

A simple, compact occluder representation can be used for terrains. We use single line segments that approximate the edges of the gridlet. Each line segment requires only two height values and can be shared between adjacent gridlets. Only the top two or three of these segments have to be tested per gridlet, depending on how the gridlet is oriented relative to the viewer.

To produce occluder edges, we fit a line through the edge points of a gridlet using least-squared error. This line is shifted down so that all edge points lie above it to ensure that the occluder edge is conservative. We use the edge points at the finest decimation level for the gridlet to guarantee that the occluder edge valid for all levels of detail. Using just the corners of the gridlets would

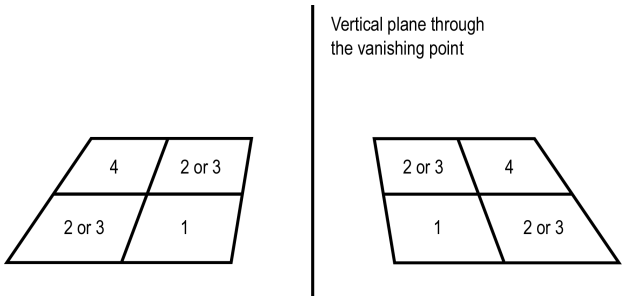

Figure 2: The gridlet child traversal orderings that ensure that no child is drawn after a child it occludes.

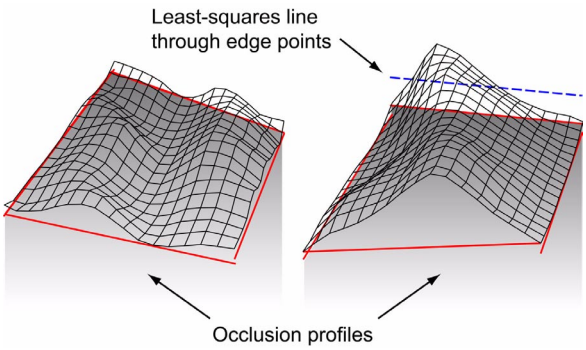

Figure 3: Gridlets with occluder edges and their occlusion profiles. The dashed line is least-squared error line through the edge points. The gridlet on the right is an example of a poor approximation of the actual gridlet edge by the occluder.

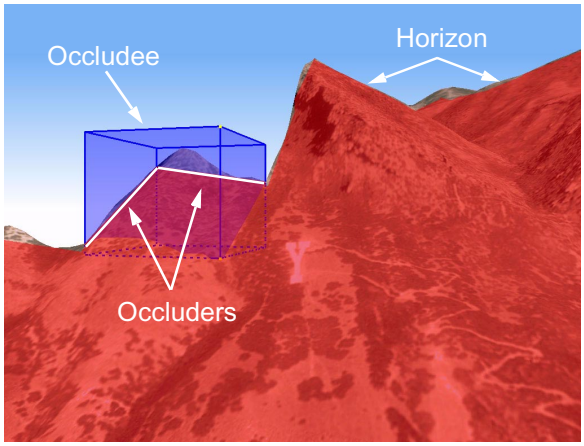

Figure 4: Visualization of horizon occlusion culling. The red mask represents the current approximation of the visibility horizon updated with current gridlet's occlusion profile.

produce an occluding edge that would work fine for the lowest LOD but morphing to a higher resolution may cause the edge to dip below the occluder line which would break the requirement that the occluder be conservative. A slightly better line may be obtained by using only those edge points that lie on the lower convex hull of the edge since these are the points that limit the position of the line.

Occluder edges do have some limitations. Occluder edges are unable to account for occlusion caused by raised portions in the center of the gridlet. There are also cases such as the one shown in Figure 3 for which this method produces a particularly poor approximation because a single line segment is used to represent a curved edge. To determine the performance of the approximation we compared the number of gridlet occlusions generated by using actual gridlet edges to the number of occlusions using occluder edges. We found that for paths through mountain canyons, where occlusion culling is most useful, the exact gridlet edges produced only 6% more occlusions on average than occluder edges. Overall, occluder edges work surprisingly well. The approximation seems to be a good trade off between accuracy, speed, and storage.

We use the axis-aligned bounding box of a gridlet for its occludee. Only the top-most edges of the bounding box need to be compared to the horizon. We tried using tighter bounding boxes created in a manner similar to that used for the occluder edges. We calculated the least-squared error plane for the gridlet vertices and shifted it up and down to form the top and bottom faces of a bounding prism for the gridlet. We found that the tighter bounding volumes made almost no difference in reducing the number of gridlets culled by the horizon visibility test. Occluders and the occludee for a gridlet are shown in Figure 4.

Other hierarchical terrain algorithms can use similar occluders and occludees. For example, the ROAM algorithm [11] is a hierarchical terrain that uses triangle bintrees. Triangles are split and merged to produce the final mesh. Each triangle is bounded by a pie-shaped "wedgie." This can be used as the occludee. Occluder edges can be constructed for each triangle in a manner similar to that used for gridlets.

4.3 Horizon Representation

To track the occlusion horizon, we discretize it into an series of vertical columns at the same resolution as the image. We store the height of each column in an array. It is important that the resolution of the discretized horizon be at least that of the image in order to avoid discretization errors. A horizon update is performed by scan-converting the projected occluder edges into the array, recording the height of the edge at each column whenever it is greater than the column's current height. The horizon visibility test scan converts occludee edges returning visible if the edge is higher than the horizon at any point.

4.4 Arbitrary Views

Downs et al. [12] perform all horizon visibility calculations using a horizontal projection. We use this idea to simplify the algorithm. This makes it easier to test objects against the horizon because it ensures that vertical edges in world space project to vertical edges in screen space, and thus do not need to be tested against the visibility horizon. Arbitrary views are handled by increasing the field of view of a horizontal projection to include the intersection of the original view frustum with the vertical image plane of the horizontal projection. Though the projection produces a warped image it does not change the occlusions, which depend only on view position. As shown in Figure 5, the field of view of the horizontal projection is only increased up to a useful limit. Parts of the scene that lie outside of the limit are rendered without horizon culling. Since the view is nearly straight down on these parts of the terrain, few occlusions occur anyway.

4.5 Comparison

Since both our algorithm and that of Downs et al. [12] are based on the idea of the occlusion horizon the main components have many similarities. The major differences are in the details and are due to the fact that their algorithm was designed specifically to handle

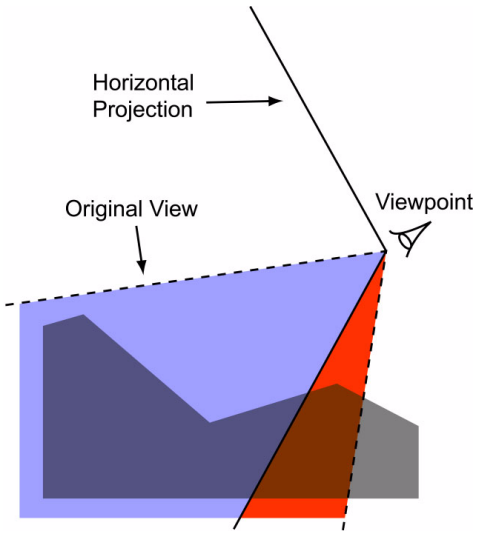

Figure 5: Horizontal projection used for arbitrary views. The region in blue is rendered with horizon occlusion culling. The region in red is rendered without the occlusion test.

urban environments while our algorithm is better suited for hierarchical terrains.

For front-to-back ordering, Downs et al. used an event queue representing the position of a plane which is swept away from the viewpoint through the scene. An event queue is necessary because buildings are located at arbitrary positions and lack the natural ordering of hierarchical terrains.

For occluders, Downs et al. used a collection of convex vertical prisms (CVPs) defined by a convex polygonal cross-section and a height value. CVPs approximate an object's inner volume. Buildings can often be represented with a small set of CVPs. Terrain elements, however, require larger sets of CVPs to represent their occlusion profiles, due to their arbitrarily oriented, sloping features. The larger number of CVPs necessary for terrain elements requires more time to process, narrowing the margin between the time to detect occluded elements and the time saved by culling them. In addition, for large terrain models, CVPs can require large amounts of memory. Occluder edges are a better representation for terrain elements because they require less memory and processing time.

Downs et al. used a hierarchy of horizontal segments arranged in a binary tree to represent the visibility horizon. This data structure can result in faster queries and updates to the visibility horizon. Unfortunately, it does not allow for arbitrarily oriented edges. Thus the occlusion profile approximated by a CVP must be further approximated by the largest rectangle that can fit inside the projected CVP. The top edge of this rectangle is used to update the horizon. Likewise, the occludee is the smallest rectangle that surrounds the projected bounding box of an object. The ability to use arbitrarily oriented edges results in larger occluders and tighter occludees. This is perhaps more important for terrains where sloped mountain tops are responsible for most of the occlusion. In our experiments, using the more accurate occlusion profiles consisting of arbitrarily oriented edges resulted in 15% more culled gridlets. Though scan converting edges is potentially slower than the tree structure, for our application it is actually faster to use the discretized horizon because fewer gridlets are drawn. With a slower processor or a faster graphics card, it may be faster to use the horizontal segment tree. The best horizon representation to use for any particular application depends on the relative speed of the processor and the graphics hardware and is probably best determined by experimentation. The main advantage of the discretized horizon is that it is extremely simple to implement and yields good results.

5. RESULTS

We have implemented the horizon occlusion culling algorithm in our terrain visualization program. Our terrain model is a 100 x 50 mile area surrounding Salt Lake City, Utah. It was created from DEMs and DOQs obtained from the USGS. The model contains over 100 million triangles and is texture mapped with imagery at one meter per pixel resolution totalling 8GB in size.

To measure the performance enhancements obtained using horizon occlusion culling, we created a predefined course through the model. This course is shown in Figure 7. Figure 6 shows the timing results we obtained by running our program on a PC with a 2.5GHz Pentium 4 processor and a GeForce 4 graphics card. We also ran the program on an SGI 320 with dual 500 MHz Pentium II processors and a PC with a 1 GHz Athlon processor and a GeForce 3 graphics card. The framerate graphs for each of these platforms were virtually identical varying only in scale due to the relative speed of the hardware. Thus we have included only the graph of the machine with the most up-to-date hardware. We measured typical improvements of twice the framerate for most views on the course with as much as a four times improvement for highly occluded views. Figure 9 shows a breakdown of the amount of time used to calculate which polygons to draw in a frame and the time it took to render them. The calculation time includes the time to perform the horizon occlusion culling algorithm. The algorithm consistently occupied about 40% of the total calculation time for each frame. Even though a large portion of the calculation time is spent in occlusion culling, the overall calculation time using the algorithm is less than without it. This is because much less geometry needs to be processed. The dramatic reduction in polygon count is shown in Figure 8. Without horizon occlusion culling 2-4 times as many polygons are drawn, none of which make any contribution to the final image.

We chose the course shown Figure 7 because it contains a wide range of view configurations. In part A, the course begins on a flat valley floor with mountains in the distance. The framerates are nearly the same with and without culling because there are few occlusions. Figure 9 shows that with occlusion culling turned on it actually takes more time to calculate which polygons to draw than to render them, due to the extra overhead of the algorithm which is producing little savings. At the end of A, the view approaches a canyon. The framerate steadily increases as the approaching mountains occlude more and more of the model. Part B flies through the canyon for some time. Here the results show that occlusion culling makes a dramatic difference because of the strong occlusion by the canyon walls. At the end of B, the view enters an open area and the framerate drops again. In the middle of part C, the view ascends up a slope which hides almost everything else, producing a rise in framerate. In part D, the view ascends the slope of a high mountain. The framerate rises both with and without culling because

many gridlets are culled by the view frustum. Finally, the view ends up on the top of a mountain overlooking the terrain. Here as in part A and the end of part B, few gridlets are completely occluded so occlusion culling yields lesser improvements.

6. CONCLUSIONS AND FUTURE WORK

In this paper we have presented a technique for calculating occlusions on-the-fly for hierarchical terrains using a visibility horizon. The algorithm is easy to implement and requires very little additional storage or precomputation. We obtained improvements in framerate by factors of 2-4 in view configurations such as mountain canyons where most of the terrain is occluded by mountains on either side.

Our model contains both urban and mountainous areas. In order to be able to render models of cities, we would like to extend the algorithm to handle urban environments. Such an extension should be fairly easy. The gridlet quadtree could be augmented to store building location information. Then the algorithm would proceed as in Downs et al. [12] using a plane sweep to process the buildings lying on each gridlet in the proper order. We would also like to explore better occluder representations for gridlets.

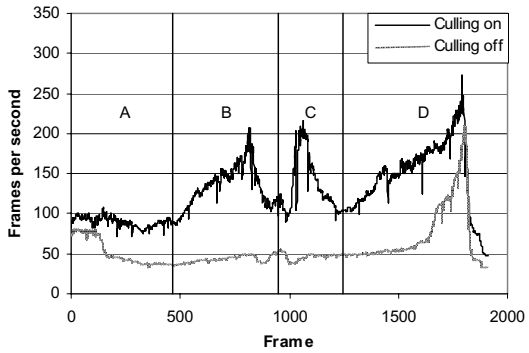

Figure 6: Frame rates achieved with and without horizon occulusion culling at 1024x768 pixels on a PC with a 2.5 GHz Pentium 4 processor and GeForce 4 graphics card.

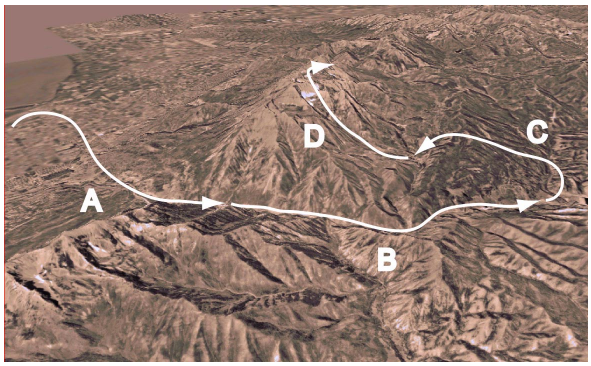

Figure 7: Path used for timing results. This path was chosen because it contained a wide variety of views including a flat valley (A), a narrow canyon (B), ascent and descent of slopes (C & D), and a mountain top (D).

References

[1] J. Airey, J. Rohlf, and F. Brooks Jr. Towards Image Realism with Interactive Update Rates in Complex Virtual Building Environments. *Symposium on Interactive 3D Graphics 1990,* pp. 41-50, 1990.

[2] D. Bartz, M. Meißner and T. Hüttner. OpenGL-assisted Occlusion Culling for Large Polygonal Models. *Computers & Graphics,* vol. 23, pp. 667-679, 1999.

[3] J. Bittner, V. Havran, and P. Slavik. Hierarchical Visibility Culling with Occlusion Trees. *Proceedings of Computer-Graphics International '98*, pp. 207-219, June 1998.

[4] D. Cline and P. Egbert. Interactive Display of Very Large Textures. *Proceedings of IEEE Visualization '98*, pp 343-350, October 1998.

[5] D. Cline and P. Egbert. Terrain Decimation through Quadtree Morphing. *IEEE Transactions on Visualization and Computer Graphics*, vol. 7 (1), pp. 62-69, 2001.

[6] D. Cohen-Or, Y. Chrysanthou, and C. T. Silva. A Survey of Visibility for Walkthrough Applications. *Proceedings of EUROGRAPHICS '00*, *Course Notes,* 2000.

[7] D. Cohen-Or, G. Fibich, D. Halperin, and E. Zadicario. Conservative Visibility and Strong Occlusion for Viewspace Par-

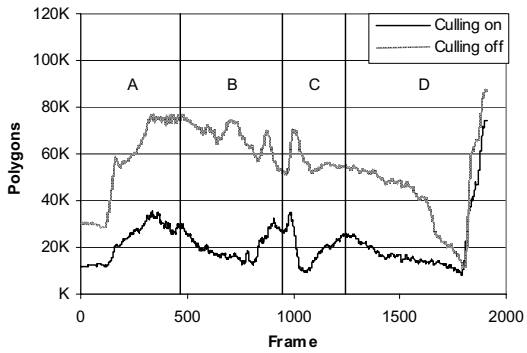

Figure 8: Number of polygons drawn each frame.

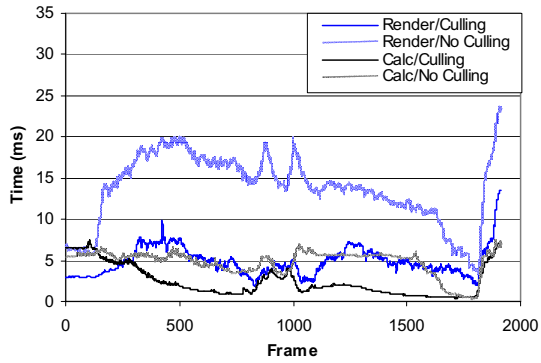

Figure 9: Breakdown and comparison of timing. This graph shows the time to calculate which polygons to draw and the time to render them for each frame. Horizon occlusion culling occupies 40% of the calculation time.

titioning of Densely Occluded Scenes. *Computer Graphics Forum*, vol. 17(3), pp. 243-253, 1998.

[8] D. Cohen-Or, E. Rich, U. Lerner, V. Shenkar. A Real-Time Photo-Realistic Visual Flythrough. *IEEE Transactions on Visualization and Computer Graphics,* vol. 2(3), pp. 255-265, 1996.

[9] S. Coorg and S. Teller. Real-time Occlusion Culling for Models with Large Occluders. *1997 Symposium on Interactive 3D Graphics,* pp. 83-90*,* April 1997.

[10] L. De Floriani and E. Puppo. Constrained Delaunay Triangulation for Multiresolution Surface Description. *Proceedings of Ninth IEEE International Conference on Pattern Recognition*, pp. 566-569, 1988.

[11] M. Duchaineau, M. Wolinsky, D.E. Sigeti, M.C. Miller, C. Aldrich, M.B. Mineev-Weinstein. ROAMing Terrain: Real-time Optimally Adapting Meshes. *Proceedings of the Conference on Visualization '97*, pp.81-88. Oct. 1997.

[12] L. Downs, T. Möller, and C. Sequin. Occlusion Horizons for Driving Through Urban Scenery. *2001 Symposium on Interactive 3D Graphics*, pp. 121-124, 256, March 2001.

[13] N. Greene, M. Kass, and G. Miller. Hierarchical Z-Buffer Visibility. *Computer Graphics (Proceedings of SIGGRAPH '93)*, pp. 231-238, 1993.

[14] H. Hey, R. Tobler, and W. Purgathofer. Real-Time Occlusion Culling with a Lazy Occlusion Grid. *Rendering Techniques 2001: 12th Eurographics Workshop on Rendering*. pp. 217-222, 2001.

[15] H. Hoppe. Smooth View-Dependent Level-of-Detail Control and its Application to Terrain Rendering. *IEEE Visualization '98*, pp. 35-42, Oct. 1998.

[16] P. Lindstrom, D. Koller, W. Ribarsky, L. Hodges, N. Faust, and G. Turner. Real-time, Continuous Level of Detail Rendering of Heightfields. *Computer Graphics (Proceedings of SIGGRAPH '96)*, pp. 109-118, 1996.

[17] R. Pajarola. Large Scale Terrain Visualization Using The Restricted Quadtree Triangulation. *Proceedings of IEEE Visualization '98*, pp. 19-26, 1998.

[18] B. Rabinovich and C. Gotsman. Visualization of Large Terrains in Resource-Limited Computing Environments. *IEEE Transactions on Visulaization and Computer Graphics*, pp. 95-102, 1997.

[19] S. Röttger, W. Heidrich, P. Slussallek, and H.-P. Seidel. Real-Time Generation of Continuous Levels of Detail for Height Fields. *Proceedings of the 6th International Conference in Central Europe on Computer Graphics and Visualization*, pp. 315–322, Feb. 1998.

[20] G. Schaufler, J. Dorsey, X. Decoret, and F. Sillion. Conservative Volumetric Visibility with Occluder Fusion. *Computer Graphics (Proceedings of SIGGRAPH 2000)*, pp. 229-238, July 2000.

[21] J. Stewart. Hierarchical Visibility in Terrains. *Eurographics Workshop on Rendering '97,* pp. 217-228, 1997.

[22] S. Teller and C. Sequin. Visibility Preprocessing for Interactive Walkthroughs. *Computer Graphics (Proceedings of SIGGRAPH '91)*, pp. 61-69, 1991.

[23] B. Zaugg and P. Egbert. Voxel Column Culling: Occlusion Culling for Large Terrain Models. *VisSym 2001-Eurographics/IEEE Symposium on Visualization*, pp. 85-93, May 2001.

[24] H. Zhang, D. Manocha, T. Hudson, and K. Hoff. Visibility Culling using Hierarchical Occlusion Maps. *Computer Graphics (Proceedings of SIGGRAPH '97),* pp. 77-88, 1997.