Efficient Ray Intersection for Visualization and Navigation of Global Terrain using Spheroidal Height-Augmented Quadtrees

Zachary Wartell, William Ribarsky, and Larry Hodges

College of Computing
Georgia Institute of Technology, Atlanta GA 30332-0280, USA

Abstract. We present an algorithm for efficiently computing ray intersections with multi-resolution global terrain partitioned by spheroidal height-augmented quadtrees. While previous methods support terrain defined on a Cartesian coordinate system, our methods support terrain defined on a two-parameter ellipsoidal coordinate system. This curvilinear system is necessary for an accurate model of global terrain. Supporting multi-resolution terrain and quadtrees on this curvilinear coordinate system raises a surprising number of complications. We describe the complexities and present solutions. The final algorithm is suited for interactive terrain selection, collision detection and simple LOS (line-of-site) queries on global terrain.

1 Introduction

The increasing computation power, memory and rendering rates coupled with efficient data organization make it feasible to interactively visualize global 3D terrain with varying resolutions down to a centimeter. Interactively rendering these large-scale terrain databases places increasing demands on the software system. Real-time level-of-detail management, efficient spatial subdivision and the use of an accurate two-parameter ellipsoidal coordinate system are a must.

This paper describes the impact of this geodetic coordinate system on quadtree spatial subdivision with respect to computing ray-terrain intersections. We extend a well-known ray-casting method for height-augmented quadtrees defined on Cartesian coordinates. The extension also handles multi-resolution terrain.

2 Background

Our terrain visualization software is VGIS [8]. VGIS uses automatic, continuous level-of-detail management for geometry and imagery. The data is partitioned into 32 spheroidal quadrilaterals called zones. Each zone contains its own quadtree. Each quadtree supports terrain at resolutions varying from 8km down to a centimeter. Currently, individual VGIS applications contain datasets with a mixture of resolutions with the higher detail insets covering the more important regions. The current maximum resolution data set is at 1 ft resolution.

To accurately model this global terrain, VGIS uses a two-parameter ellipsoidal coordinate system commonly used in geodesy [12].

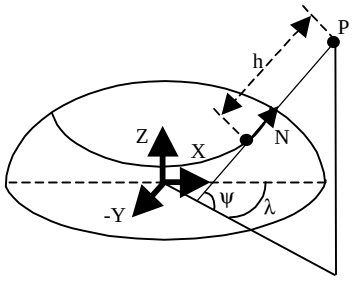

Fig. 1. Spheroidal Coordinate System

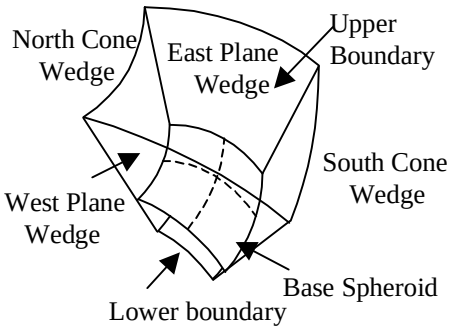

Fig. 2. Spheroidal Height-Quad

This two-parameter ellipsoidal coordinate system is based on a spheroid. A spheroid is subclass of ellipsoid created by rotating an ellipse about its major or minor axis. It is synonymous with "rotation ellipsoid" [4] ,"biaxial ellipsoid" [12] and "ellipsoid of revolution"[11]. Figure 1 illustrates this coordinate system. The two parameters are the spheroid's major semi-axis, a, along X and Y, and minor semi-axis, b, along Z. In this system longitude, λ, is equivalent to the longitude in polar coordinates; however, latitude, ψ, is the angle between the surface normal and the equatorial plane. Height, h, is measured parallel to the normal between the point in question, P, and the underlying surface point.

VGIS builds its terrain database and quadtree subdivision on this curvilinear coordinate system. In this quadtree the quads are bounded by meridians and parallels. This divides the spheroid into triangles at the poles and quadrilaterals elsewhere. (Note that since meridians are not geodesics, these quadrilaterals and triangles are not true spheroidal quadrilaterals and spheroidal triangles; however, for brevity we will ignore this distinction.) Finally each quad is augmented with a height attribute equal to the maximum spheroidal height of the contained data.

Using this terrain structure, we must provide an efficient method for finding arbitrary ray to terrain intersections. Such an algorithm serves as a basis for interactive terrain selection, collision detection and simple line-of-site queries.

While an efficient method for ray-casting through Cartesian coordinate height-augmented quadtrees is well-known [2], this method assumes that the bounding volumes are bounded by their Cartesian coordinates. Extending the algorithm to handle spheroidal based height-quadtrees for multi-resolution terrain poses a number of problems. We present our solutions in order of their generality with respect how terrain is modeled. First we address tracing through the spheroidal bounding volumes. The presented algorithm applies to terrain modeled either as

voxels, triangles, or bilinear patches. Next we address tracing through individual terrain elements. Here our solution is specific to terrain modeled as a regular triangle mesh. Third we address complications added by triangle models using multi-resolution data sets. Finally, we discuss surface continuity issues that are specific to VGIS's continuity preservation methods.

3 Traversing Spheroidal Height-Quads

Cohen et al.'s [2] method for efficient ray-terrain intersection, is similar in spirit to Bresenham line drawing. It traces the XY projection of a ray through the XY footprints of a height-augmented quadtree based on Cartesian coordinates. Upon entering a height-quad the entering and exiting z-coordinate of the ray is compared to the height of the quad. If the ray intersects the quad, the algorithm steps into the child quad at the next resolution level. Otherwise, the algorithm steps into the next quad at the same resolution level. The algorithm is so efficient that it is targeted towards real-time rendering of terrain. Figure 3 (see Appendix) illustrates the high-level functionality of the algorithm. The figure is a side view with the ray in red and 3 levels of recursive height-quads. Blue volumes are intersected by the ray. Solid black volumes are not intersected, but the ray does enter their X-Y footprint. Dash black volumes are not examined by the algorithm at all. The red volume is the lowest level intersected volume. This figure illustrates the recursive nature of the bounding volumes and of the algorithm.

Ideally, a spheroidal extension would use incremental integer calculations similar to Cohen's midpoint method. Unfortunately, while the basic high-level algorithm still applies, the midpoint technique that works so beautifully in the Cartesian setting appears to have no similarly efficient analog in this spheroidal case.

An exact analog would require a spheroid to plane mapping in which the spheroidal projection of a ray in 3-space maps to a line and in which the spheroid's quads are mapped to a regular square grid. The only common sphere to plane mapping that maps parallels, meridians and projected rays onto lines is the gnomonic projection [9, 236]. The gnomonic mapping centrally projects the sphere through the sphere center onto a plane. The plane is placed tangent to the sphere at an arbitrary intersection point. Unfortunately, this mapping projects spherical quads onto planar rectangles with varying sizes. As we examine rectangles farther and farther away from the plane-to-sphere intersection point, the rectangles' areas grow towards infinity. The gnomonic map will not allow us to translate the Cartesian algorithm to the spheroidal case.

A partial analog to the Cartesian algorithm would require a spheroid to plane mapping in which quads map to a regular square grid and a projected ray maps to a curve. A cylindrical mapping can map quads onto a regular square grid. Unfortunately, this mapping maps the projection of a 3D ray to a complicated curve containing multiple embedded transcendental functions (equation 3-1 [14]). While efficient methods for discretizing lines [6], ellipses [6], and cubics [15] are

known, a similarly efficient method of discretizing a curve of this complexity is not available.

Without an incremental integer approach for stepping through quads, we resort to floating-point computation of ray-quad boundaries. Unfortunately, there appears to be no closed form solution for solving t in terms of the latitude, ψ [14] for the cylindrical projection. This would be needed for computing the projected ray's quad intersections with closed form arithmetic. We therefore perform the ray-quad intersection tests in 3 dimensions where closed form solutions exist.

3.1 Bounding Surfaces

We begin by describing the bounding surfaces of a spheroidal height-quad. Generally these boundaries consist of 4 side boundaries formed by 2 plane wedges and 2 cone wedges, and of upper and lower boundaries formed by quadrilaterals on the normal expansion of the spheroid (Fig. 2). We now give the equations of these surfaces. They are derived in [14].

A longitude side boundary at longitude λ is a wedge of the plane:

$$\cos\lambda x + \sin\lambda y = 0 \tag{1}$$

A latitude side boundary at latitude ψ is a wedge of the cone:

$$\begin{aligned} &x^2 + y^2 - \tfrac{z^2}{m^2} + \tfrac{2zk}{m^2} - \tfrac{k^2}{m^2} = 0 \\ &\text{where} \\ &m = \tan\psi, k = Z - Xm, \\ &X = \tfrac{a\cos\psi}{\sqrt{1-e^2\sin^2\psi}}, Z = \tfrac{a\left(1-e^2\right)\sin\psi}{\sqrt{1-e^2\sin^2\psi}}, e^2 = \tfrac{a^2-b^2}{a^2} \text{(see [4])} \end{aligned} \tag{2}$$

For the upper boundary surface of a quad with maximum height, h, the true boundary surface is not amenable to analytic intersection computations [14]. Therefore we use the approximation spheroid, B_h:

$$B_h = \begin{cases} majorAxis = (a+h), minorAxis = \left(b + \tfrac{ha}{b}\right), \text{if } h \geq 0 \\ majorAxis = \left(a + \tfrac{hb}{a}\right), minorAxis = (b+h), \text{if } h \in \left(\tfrac{-b^2}{a}, 0\right) \end{cases} \tag{3}$$

The stipulation that $h \geq -b^2/a$ is necessary due to degenerate cases of the true boundary surface. In practice, surface terrain models well exceed this lower boundary because a and b are typically close while the minimum h is orders of magnitude greater than a or b. For example in the WGS-84 Earth datum, $-b^2/a$ is -6,335,439m while the minimum h is around -15,000m.

Finally, since we assume that traced rays begin outside the planet, it is sufficient to choose a single global lower bounding surface. We model the lower boundary simply as a sphere whose radius equals the distance from the spheroid

center to the closest terrain vertex. B_h is inappropriate here since it lies outside the true boundary while a lower boundary approximation must lie inside the true boundary.

4 The Algorithm

While the high-level principles of the Cohen algorithm (Section 3), apply to the spheroidal case, the details differ. The spheroidal algorithm is divided into two procedures. A user called procedure performs setup and zone traversal and a recursive procedure then recursively traverses through each zone's quadtree. The user called procedure first clips the ray to the volume bounded by a global upper boundary and the global lower boundary. The global upper boundary is the upper bounding surface, B_h, with height equal to the maximum global height. As part of this clipping, we compute, t_global_exit, the ray parameter value of the ray's global exit point. Next, we determine which zone contains the ray origin. Starting with this zone, we step through successive zones until either an intersection occurs or the ray exits the global boundaries. Zone traversal is quite similar to quad child traversal which is discussed in detail below. For each zone we call the recursive procedure to recursively traverse the zone's quadtree.

The recursive procedure must first determine whether the ray, which is assumed to enter a quad's side bounds, truly intersects the quad volume. Since the upper boundary is curved, it is insufficient to check the height of the ray's entering and exiting intersections with the side boundaries. Instead, we compute the ray's parameter values, t_in and t_out, at these side intersections and we compute the ray's intersection parameters, t_0 and t_1, with the quad's upper boundary surface (Fig. 4, see Appendix). If and only if these two parameter intervals overlap, then the ray has entered the height-quad volume and we step through the quad's children.

If the quad volume is intersected, the algorithm must traverse the quad's children and recurse at each child. The first encountered child is determined from two factors. The first factor is which side boundary of the parent the ray entered. The second factor is in which half-space of one of the parent's internal partition surfaces the entrance point lies. In Figure 5 (see Appendix) these internal partitions are shown in blue. They partition the quad into four sections. The latitude partition surface is a cone wedge stretching east-west. The longitude partition is a plane wedge stretching north-south. Knowing which side boundary is intersected and which internal partition half-space contains the entrance point, we know which child quad to visit. For example, in Figure 5 the ray (red) enters the west side boundary and the entrance point (marked by a red X) is north of the internal latitude partition. Hence, the ray enters the north-west child. Four side boundaries and two internal partitions yield eight combinations. Each combination maps to one child. By determining which combination occurs, the algorithm determines which child to visit. Note an exception arises when the current quad contains the ray origin. In this case, we visit the child containing

the ray origin. This child is determined by examining which half-spaces of both internal partitions contain the ray origin.

Having visited the first child, we must determine the other child quads intersected by the ray. Note that in the spheroidal case, a ray may intersect all four of a quad's children or may enter a quad twice. This can occur since a ray can have two intersections with a quad's latitude cone boundary. Given these complications we determine the next child quad by computing the ray's intersection with the current child's boundaries. Note these boundaries are subsets of the parent's boundaries and internal partitions. The child exit boundary is the child boundary whose ray intersection's ray parameter value is the smallest while still being greater than the child's entrance point's value. This exit point is illustrated by the second red X in Figure 5. So given the current child, we compute *t_child_out*, the ray parameter where the ray exits the child, along with *side_out*, the boundary of the child at this exit. With knowledge of *side_out*, we know what child is entered next. Child traversal terminates when either a child reports terrain intersection, all children are visited, or *t_child_in* of the current child is greater than *t_global_exit*, the ray intersection with global upper boundary.

5 Traversing Individual Terrain Elements

While the methods of the previous section apply to terrain regardless of the modeling method (voxel, bilinear patch, or triangles), the issues raised when traversing individual terrain elements are model dependent.

In ray-casting methods [2] the height-quad tree recurses down to the level of the smallest modeled terrain element. In regular triangle mesh methods, however, the height-quad tree typically does not recurse down to the level of the smallest modeled terrain element. Instead, a quad contains a fixed size matrix of triangles such as in Figure 6 (see Appendix). Within this block there is no further quadtree subdivision. This means that for triangle modeled terrain, once we trace a ray to a leaf quad, we must then separately trace the ray through that quad's block of triangles. Additionally, the modeling method affects the mathematical surface in between the sampled elevation points. If we render with ray-casting we might model the surface as set columnar voxels which project radially out from the zero-elevation surface. (Note on the spheroidal coordinate system, these voxels are not cubes as in traditional Cartesian based terrain.) Alternatively, we can define the surface to be a set of bi-linearly interpolated patches. This is the typical method of interpolating height fields in geodesy [3]. Unfortunately, while these are the most mathematically robust surface definitions, a practical polygon graphics pipeline based system must interpolate between sampled elevation points by treating these points as vertices of triangles.

Here we will focus on the triangle model. In order to minimize the number of triangles tested, we treat each triangle-pair as if it was contained in its own small height-quad and we then visit only those height-quads whose sides are intersected by the ray.

In Figure 7 (see Appendix), four triangle pairs are drawn in red on a part of a spheroidal quad in black. The blue arrows are extensions of the spheroid normal at the quad's terrain grid points. Triangle vertices are confined to these lines. Furthermore, the lines delineate plane wedges defining four-sided volumes (blue). Note, triangle edges are confined to these plane wedges. These four-sided volumes can serve a similar purpose to the high level height-quads. If the ray intersects the first triangle pair's volume, A, (in bold blue), we determine which of the 4 neighboring triangle-pairs to visit next by intersecting the ray with the volume's planar wedge sides. If the ray intersects the side shared by volume A and B, this immediately tells us to visit the triangle-pair volume B. Similarly if the ray next intersects the side shared by B and C, we step into volume C. At each volume we test for ray-triangle intersections with the triangles in that volume. We continue traversing and testing triangle pairs until either a triangle is intersected, the quad boundary is reached or t_volume_exit, the ray parameter at its exit from the current triangle-pair's volume, is greater than t_global_exit. Figure 6 (see Appendix) illustrates a typical pattern of examined triangle pairs in red.

Unfortunately the triangle model poses a theoretical problem that the other surface models do not have. Since the spheroidal height-quads are concave volumes, they will not contain all parts of the triangles whose vertices are contained in the volume and assigned to the quad. Specifically, the latitude conical boundaries do not contain all parts of the planar terrain triangles along this border. This problem is illustrated in Figure 8 (see Appendix). Figure 8 shows 3 terrain triangles in red at the corner of a quad whose east, north, south and lower boundaries are drawn in black. The upper triangle is assigned to the illustrated quad while the lower 2 triangles are assigned to the adjacent quad across the south border. The green highlighted portion of the lower 2 triangles is the portion of these lower triangles not contained in the adjacent quad.

It is important to note that the containment problem is fundamental to any recursive spheriodal partitioning. Using Bowring's theorem on normal sections [3], it is easy show that all spheroid partitionings, such as [1], [5], [7] and [10], have this problem [14].

The containment problem can potentially cause the ray intersection algorithm to miss an intersection. Referring to Figure 8, the ray could first pass over the adjacent southern quad without intersecting it and then enter the illustrated quad. If the ray is at a steep angle, it could then pierce the green area. Since the illustrated quad does not contain the triangles associated with this green area, the ray will exit the global lower boundary and the algorithm would falsely indicate no intersection occurred.

When using this algorithm for interactively pointing at and grabbing terrain, however, it has been our experience that such pathological cases never occur [13]. The reason is that each quad contains a relatively dense 128x128 triangle-pair block making the green area in Figure 8 extremely small. While the increasing curvature of the cone wedges at extreme latitude quads could exacerbate the containment problem, the increasing surface density of the triangles at these

extreme latitudes counteracts this effect. This increase in surface density occurs because the quad surface area grows smaller at extreme latitudes.

6 Managing Multiresolution Aspects of Terrain

While covering the general traversal of the high-level spheroidal quads and the specific traversal of triangle-modeled terrain elements, we glossed over how a multi-resolution terrain model interacts with the ray casting algorithm. A typical multi-resolution model such as VGIS stores terrain data in 2^nx2^n blocks at resolutions at varying powers of 2. For rendering purposes, the system then goes to great lengths to ensure that the rendered terrain is a continuous surface. The algorithm uses a visual error metric to render the lowest-detail level necessary to maintain visual quality while preserving mesh continuity [8].

As previously mentioned, contrary to ray-casting models where the recursive subdivision of height-quads continues down to individual voxels, in a regular mesh model a leaf quad contains a N by N array of triangle pairs called a block. Equally important, the quadtree is not a full tree. Instead a branch is only as deep as necessary to reach the highest resolution block available on disk. Moreover, while the complete quadtree is always in main memory, the actual triangle data is dynamically paged into main memory as dictated by the rendering algorithm. In Figure 9a, a flattened and zoomed in view of a single zone is shown. The outlines of the sub-quads existing in the zone are also shown. High resolution data is only available for the north-eastern most quad. This is indicated by the presence of the higher resolution quads in this region. Figure 9b shows the corresponding quadtree data structure.

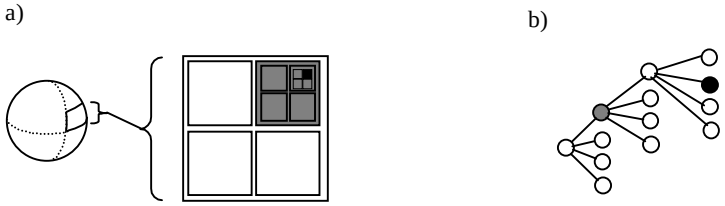

Fig. 9. Multi-resolution quadtree.

These complications lead to the following modification to the high-level quad traversal algorithm. We traverse the quad-tree as detailed in Section 4 until either the ray exits the tree or the ray enters a leaf quad. If the ray enters a leaf quad, we need to find the highest resolution in-memory block covering the quad. So if the entered quad's block is paged out, we must find the first quad ancestor whose block is paged in. A loop accomplishes this traversal. At each iteration we attempt to trace the ray through triangles of the ancestor quad. There are three possible results. Either the ray intersects a triangle, intersects no triangle, or the ancestor quad has no in-memory data available. The loop continues until

reaching an ancestor with data available. For example, assume in Figure 9 we have recursed down to and intersected the north-eastern most quad. This quad is shown as a solid square in the geometric diagram (9a) and a solid circle in the tree diagram (9b). Let's assume data for this quad is paged out. Then the ancestor loop climbs up the tree to the first ancestor that contains terrain data. In Figure 9 we assume this is the 2nd level quad which is tinted light grey.

When an ancestor with data is found, we should only trace through the rectangular subset of its block which covers the original leaf quad. We compute the boundaries of this subset using an incremental integer approach [14] since floating point methods allowed rounding errors that occasionally yielded invalid array indexes.

Whether the algorithm should only address paged-in data is application dependent. Therefore we also have a parameter controlling how ray traversal handles paged-out data. This allows the programmer to balance the accuracy of the terrain data used against the performance penalty of paging. The additional parameter, *min_level* ("minimum level") indicates the minimum tree depth of a quadnode that may be used during triangle traversal. We modify the high-level quad traversal algorithm as follows. Recall that high-level traversal continues until a leaf quad is encountered. Now instead of just using terrain data from the leaf quad's first ancestor with paged-in data, we add the constraint that we must stop at the ancestor whose depth equals *min_level*. If we reach this ancestor without finding paged-in terrain data, we must wait and page in this ancestor's terrain. Note it is possible that the leaf quad depth is less than *min_level*. This means that terrain data at the desired resolution does not exist on disk. We must make do with the terrain data from this leaf quad and page it in as needed. With this modification, setting *min_level* to zero will use only the data in primary memory and will never wait for new data to be paged-in (unless even the lowest resolution data is absent). Setting *min_level* to the maximum possible depth will page-in whatever data is necessary to ensure that ray traversal uses the greatest resolution data on disk. Setting *min_level* to some other value allows the programmer to balance the accessed terrain's resolution with the algorithm's real-time performance.

7 VGIS Surface Continuity

Surface continuity issues add further complications. When two adjacent terrain blocks have different resolutions the edges of triangles along the shared border will not match. When rendering, VGIS uses a set of rules to discard certain vertices and generate a triangle mesh using this vertex subset. This resulting mesh has no cracks along block borders. How and/or when should we apply such rules to the terrain traversed by the ray intersection algorithm? The answer depends on the application of the intersection algorithm.

In our current VGIS virtual reality application, we use the ray-terrain intersection for terrain selection with a hand-held virtual laser pointer [13]. Whenever, the ray crosses block boundaries during triangle-pair traversal we compute tem-

porary geometry for these polygons and test them for intersection with the ray. This method is simple and fast but it violates the spirit of the VGIS rendering algorithm. We continue to research how to modify VGIS's rendering rules for applications where the temporary geometry approach may be inappropriate.

8 Results and Conclusions

Figure 10 (see Appendix) illustrates our complete algorithm in operation. Here *min_level* is zero and we use the simplest continuity algorithm appropriate for fast interactive terrain selection. The application is running on a virtual workbench [13]. The red ray is a virtual laser pointer interactively manipulated by the user. In order to better distinguish the terrain from the quad outlines, the terrain has been altered to appear black-and-white. The yellow lines indicate the projection of the ray origin onto the spheroid and the point on the ray where it exits the global boundaries. The visited height-quads' upper boundaries are outlined in green, black, red and blue. Blue indicates the quad volume was intersected. Red indicates the quad was intersected and is a leaf. Green indicates the quad's polygon data was used for polygon traversal. Black indicates that while the quad side bounds were intersected the quad volume was not, i.e. the upper boundary was not pierced. The small streaks of green inside the red quads are the outlines of the triangles which were tested for intersection. In 10a, the planet is at a resolution such that the polygon data associated with the leaf quad (red) is not paged in. The algorithm visits ancestor quads until reaching the first quad (green) with polygon data covering the leaf quad. Figure 10b is similar to 10a, but it shows a further zoomed in view.

We are successfully using this algorithm for navigating global terrain on the virtual workbench [13]. The algorithm is fundamental to our navigation method. The user navigates with a virtual laser pointer used to grab the terrain when panning, rotating and zooming. Empirically the intersection algorithm has had no affect on framerate, as is desired for VR interaction. Asymptotically, the algorithm is equivalent to standard quadtree and octree traveral methods. It is linear in the number of pierced bounding elements.

To conclude we have described the impact of the geodetic coordinate system on quadtree spatial subdivision with respect to computing ray-terrain intersections. We presented a new set of efficient methods for tracing a ray over the terrain. These methods go beyond the work of Cohen, promoting a complete approach for global terrain in a multi-resolution spheroidal quadtree structure.

9 Future Work

There are several avenues of future work. First, the continuity issues have yet to be fully resolved for all uses of ray-terrain intersection. Next it is probably possible to switch from the spheroidal approach to the much simpler Cartesian approach when the algorithm reaches high detail quads. This is plausible because at some point the results of these two approaches will be the same due to the

finite precision of computer arithmetic. Finally, the algorithm can be extended to manage spheroidal octrees for partitioning aerial information.

10 Acknowledgements

This work was performed in part under contracts N00014-97-1-0882 and N00014-97-1-0357 from the Office of Naval Research. Support was also provided under contract DAKF11-91-D-004-0034 from the U.S. Army Research Laboratory. We thank Frank Jiang for help in setting up the workbench environment.

References

1. Borgefors, Gunilla. A hierarchical 'square' tesselation of the sphere. Pattern Recognition Letters 13 (1992), pages 183-188.
2. Cohen, Daniel, and Amit Shaked. Photo-Realistic Imaging of Digital Terrains. Eurographics '93, Volume 12, (1993), No. 3. Pg 363-373.
3. Hooijberg, Maarten. Practical Geodesy Using Computers. Springer. 1997.
4. Dragomir, V., D.Ghiţău, M. Mihăilescu, M.Rotaru. Theory of the Earth's Shape. Elsevier Scientific Publishing Company. Amsterdam. 1982.
5. Fekete, György, Rendering and Managing Spherical Data with Sphere Quadtrees. Proceedings of the First IEEE Conference on Visualization. Visualization '90. 1990. Pp. 176-86.
6. Foley, James D., Andres Van Dam. Fundamentals of Computer Graphics. Addison-Wesley. Reading, Mass. 1990.
7. Hwang, Sam C., Hyun S. Yang. Efficient View Sphere Tessellation Method Based on Halfedge Data Structure and Quadtree. Computer & Graphics, Vol. 17, No. 5 (1993), pages 575-581.
8. Lindstrom, Peter, David Koller, William Ribarsky, Larry Hodges, Nick Faust, and Gregory Turner. Real-Time Continuous Level of Detail Rendering of Height Fields. Computer Graphics (SIGGRAPH 96), pp. 109-118.
9. Maling, D.H. Coordinate Systems and Map Projections. London: George Philip and Son Limited. 1973.
10. Otoo, Ekow J., Hogwen Zhu. Indexing of spherical surfaces using semi-quadcodes. Advances in Spatial Databases. Third International Symposium, SSD '93 Proceedings, pages.510-529.
11. Smith, James R. Introduction to Geodesy. John Wiley & Sons, Inc. 1997.
12. Vaníček, Petr, Edward Krakiwksy. Geodesy: The Concepts. North-Holland Publishing Company. Amsterdam. 1982.
13. Wartell, Zachary, William Ribarsky, Larry Hodges. Third-Person Navigation of Whole-Planet Terrain in a Head-tracked Stereoscopic Environment. (to appear) Proceedings of IEEE Virtual Reality 1999 (March 13-17 1999, Houston TX).
14. Wartell, Zachary, William Ribarsky, Larry Hodges. Efficient Ray Intersection for Global Terrain using Spheroidal Height-Augmented Quadtrees. GVU Tech Report 98-45.
15. Watson, Ben, Larry Hodges. Fast algorithms for rendering cubic surfaces. Proceedings Graphics Interface '92 (May 11-15 1992, Vancouver, BC), 19-28.

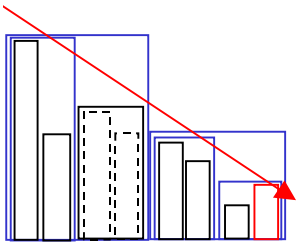

Fig. 3. General algor-ithm.

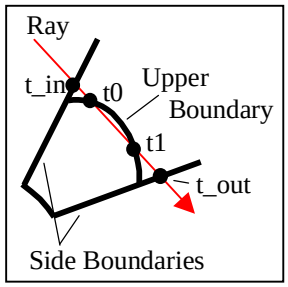

Fig. 4. Interesection test

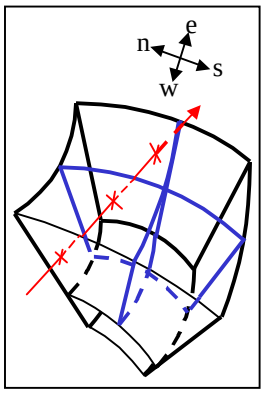

Fig. 5. Quad traversal.

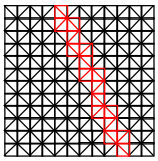

Fig. 6. Triangle grid.

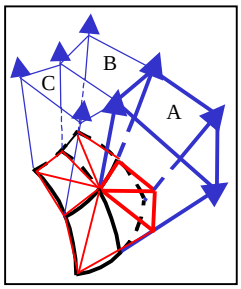

Fig. 7. Triangle traversal.

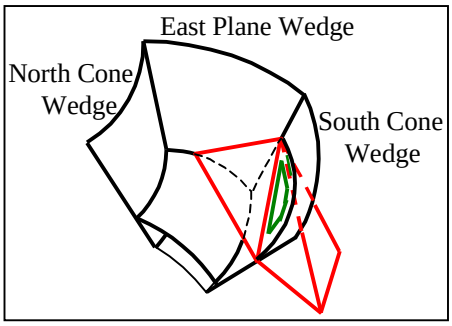

Fig. 8. Triangle containment problem.

a)

b)

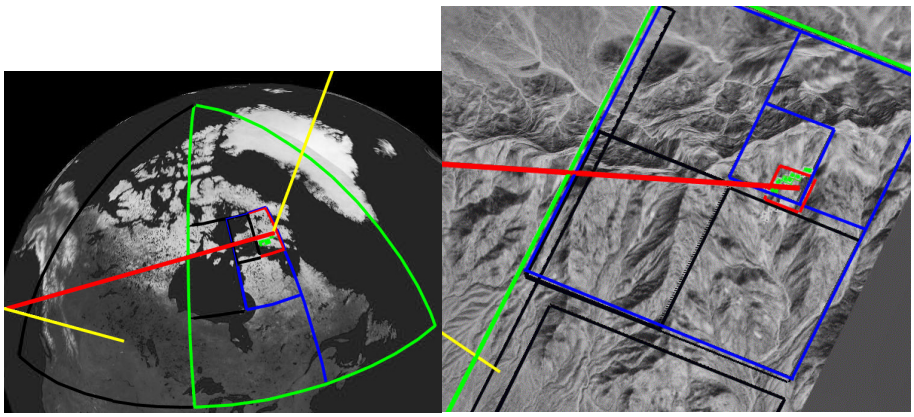

Fig. 10. Complete Algorithm.