Accelerating Line-of-Sight queries for Terrain Processing using Region Based Visibility

Brian Salomon David Tuft Sean Hanlon Dinesh Manocha
University of North Carolina at Chapel Hill
{salomon, tuft, wanderer, dm}@cs.unc.edu

Abstract

We present an efficient algorithm for line-of-sight (LOS) computations in modeling and simulation applications. LOS queries are solved by performing point-to-point visibility computations in complex environments with moving entities. Our approach divides the environment into regions and uses region based visibility calculations to compute region-to-region visibility as a visibility table for each region. This table indicates portions of the environment that are definitely blocked from any point within a region. When answering an LOS query we examine visibility tables of the regions containing the endpoints. If the portions of the environment containing the endpoints are blocked from each other according to the RBV computation then we do not need to perform a ray cast test. We have implemented our algorithm and have demonstrated its performance on terrain and urban environments. We observe more than three times speedup based on our region based visibility implementation.

Keywords: Region Based Visibility, Line of Sight, Ray Tracing

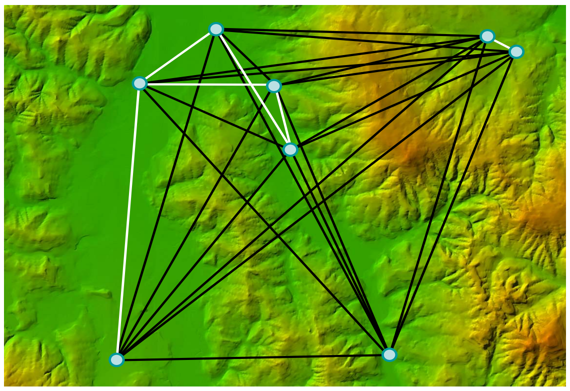
Figure 1: **LOS Between Multiple Entities.** *LOS between multiple entities is inherently an $O(n^2)$ algorithm. For the eight entities shown here there are 28 LOS queries. The blocked LOS queries are shown in black. The unblocked queries are shown in white.*

1 Introduction

Computer generated forces (CGFs) are computer systems that emulate the behavior of multiple entities and units in a complex environment. In these systems the tactical behaviors and decisions are either made by human operators or automated decision algorithms. Such systems are increasingly used in computer games, battlefield simulations, and training environments.

One of the recent challenges has been to perform these interactive simulations on complex terrain and urban environments. Any entry level simulation requires interaction between the entities. Representing the effects of terrain requires some method for computing an unbroken geometric line of sight (LOS) between any two given entities or locations. An LOS query simply requires determining whether two entities in the environment can see each other with respect to all sources of occlusion. Occlusion may be caused by environmental obstacles such as the terrain, man-made structures, atmospheric effects, or other simulation entities. These queries are used extensively in entity AI processing allowing entities to react to other entities within their visible range (or the range of various sensors). Figure 2 depicts two LOS queries. Query A intersects geometry and does not have LOS, while Query B does have LOS.

Although a single LOS query is a fairly simple geometric problem, LOS queries can account for upwards of 40% [15] of total simulation time. In a battlefield simulation with n entities the total number of LOS queries tends to grow as $O(n^2)$ [9]. This is demonstrated by Figure 1. Simulation designers must reduce the number of LOS queries and often employ filters and heuristics. Distance thresholds are commonly used to reduce the total number of LOS calls. This requires calculating entity-to-entity distances and reduces the number of queries by a constant factor but does not address the theoretical complexity. Thus, as the desired complexity of simulation increases, the fraction of CPU cycles used to compute LOS rises. A few thousand entities may require millions of LOS queries per time step. Moreover, advances in acquisition and modeling technologies have allowed simulation designers to create more complex environments, thereby increasing the number of obstacles against which an LOS query must be tested. Legacy combat simulation systems sample an LOS ray and test points along the ray against a terrain rather than computing exact interactions to cope with the number of LOS queries that must be resolved [12].

LOS and Ray Casting: LOS is typically solved as a point-to-point visibility problem using ray casting. A ray is traced through the environment and tested against obstacles. Simulators may account for atmospheric effects by tracking the distance the ray travels through a transparent medium such as smoke or fog. LOS algorithms can borrow many of the techniques used in ray tracing for image synthesis such as acceleration data structures (e.g. grid or kd-tree). Ray tracing acceleration algorithms often rely on spatial coherence between rays because the eye rays all have a common origin and are shot through the same plane. However, LOS requires tracing many non-coherent rays through the environment making it difficult to leverage more efficient ray tracing algorithms. Moreover, for many scenarios it is sufficient to simply know whether the line between the entities is blocked without computing the actual point of intersection or blocking object.

Region based visibility (RBV) algorithms have been developed in computer graphics as solutions for various problems. RBV

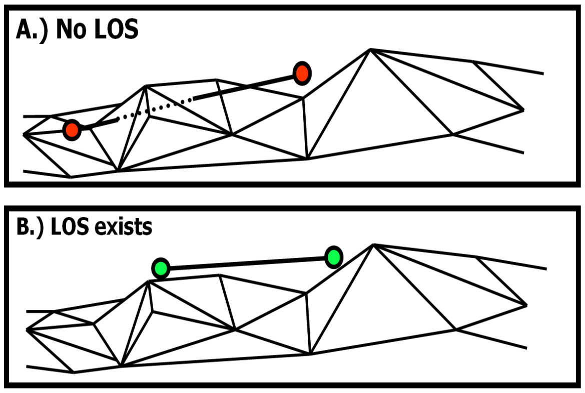

Figure 2: **LOS.** *To determine LOS, a line is drawn between two points. If the line intersects any geometry, LOS does not exist.*

algorithms determine the visible portion of the environment from a given region or cell. In this paper we show how RBV can be used to decrease the number of ray cast tests that must be performed.

Main Results: We employ RBV to reduce the number of LOS queries that must be resolved by a ray cast. The environment is divided into a set of regions using a coarse uniform grid. Rather than simply computing region-to-region visibility, we compute a visibility table for each region with cells that are much finer than those of the coarse region grid. RBV culling gives the greatest benefit when at least one of the entities is on or near the ground. Thus, the visibility table is a 2D grid of cells aligned to the terrain rather than 3D grid. The table for each region simply consists of one bit per cell indicating whether that cell is blocked from the region. Each region in 3D stores a table of visibility from itself to the entire terrain. This allows RBV culling for both aerial-to-ground and ground-to-ground LOS queries.

The runtime system uses the visibility tables to determine whether a ray cast is necessary. If one entity is in the blocked portion of the other entity's region according to that region's visibility table then we know the ray cast will find an intersection. Thus, the ray cast is skipped and the query is answered as blocked. Currently, our RBV culling only accounts for static geometry and cannot account for blockage due to entities. Such intersections would be found by the ray casting algorithm.

We have implemented our algorithm and integrated it into a next generation CGF. In test scenarios 70% to 90% of queries can be culled using our technique. We used a simple urban scene and a large dataset representing a 8km by 8km area around the Caspian Sea with buildings. Our algorithm reduced the average LOS query time from 6.2 microseconds to 2.7 microseconds. The storage overhead for visibility tables is approximately 300MB. However, this data is highly compressable, though our initial implementation does not use compression. The size of the tables is independent of the primitive or object count in the scene.

The rest of the paper is organized as follows: Section 2 reviews the relevant work in LOS, ray tracing, and RBV. In Section 3 we describe our approach to using RBV to assist in solving the LOS problem. Initial results are presented in Section 4 and concluding remarks in Section 5.

2 Previous Work

We outline some of the previous work in LOS, ray tracing, and RBV that is most related to our algorithm.

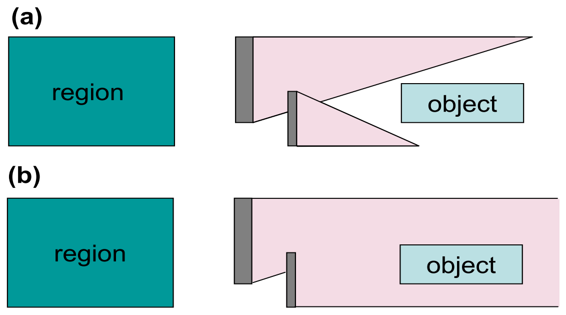

Figure 3: **Occluder fusion.** *In (a) neither of the individual umbras occlude the object. However, in (b) the fused umbra fully occludes the object.*

2.1 Line of Sight

An LOS query is presented as two entity locations expressed as 3D points. Current simulation systems typically perform ray cast tests to resolve LOS queries. Legacy systems used approximate algorithms that sample the line segment connecting the entities and compare sample points to a terrain representation and terrain features [12]. The ray cast test is similar to tracing a single ray for rendering purposes in that the primitives along the ray are intersected with the ray. The only significant difference is that the LOS test is actually a line segment test rather than a half-infinite ray. We still use the terms "ray" and "ray cast" to emphasize the connection to ray tracing.

The algorithm presented by Salomon, et al. [15] uses a GPU to render the LOS ray between two points as a line segment. Hardware occlusion queries are used to compare the line segment against a representation of the terrain in the depth buffer using an orthographic projection. This method culls rays with definite visibility and works best in scenarios in which most of the entities are visible to each other, such as open fields. In this case many LOS calls can be culled as visible. Non-culled queries are tested using ray casting.

Like this previous algorithm, our LOS algorithm is a culling approach. In fact, these algorithms are orthogonal and can be used in combination. While the previous algorithm conservatively accepts trivially visible queries, our new algorithm rejects trivially blocked queries using region based visibility.

2.2 Ray Casting and Ray Tracing

Ray casting was first proposed for image synthesis by Appel [2]. Appel used rays originating from the eye and passing through pixels on the image plane. In the 1960s scientists at MAGI used ray casting techniques developed for radiation simulation and applied them to generating images.

Whitted [21] extended ray casting to generate more realistic lighting effects using reflected, refracted and shadow rays. The term "ray tracing" describes this use of eye rays followed by secondary rays. In the decades since there has been extensive research into accelerating ray tracing computations. We cannot cover all of the literature in this broad field and refer the interested reader to a recent survey [19] for a more extensive coverage. However, efficient ray tracing and ray casting algorithms have been developed that historically have not been employed in simulation systems for LOS. The most important of these are spatial data structures such as bounding volume hierarchies, grids, and kd-trees. Our algorithm uses a kd-tree to accelerate ray casting.

Many recent ray tracing acceleration approaches built on top of spatial acceleration structures rely on ray coherence. For image synthesis all rays originate at the eye and thus the initial set of rays exhibit very high coherence. Secondary rays originating at highly specular surfaces may also exhibit high coherence. Early research using coherence includes [8], pencil tracing [18] and cone tracing [1]. One recent approach [20] leverages modern architectures by grouping rays into bundles and accelerates traversal and intersection with primitives for all rays simultaneously by taking advantage of SIMD instructions. Reshetov et al. [14] present a variation on beam tracing to exploit coherence in traversing a kd-tree spatial structure.

LOS has different coherence characteristics than traditional ray tracing for image synthesis. LOS queries are issued by AI simulators and other simulation steps for the various entities in an incoherent manner and thus tracing multiple rays simultaneously is not feasible. Furthermore, the design of existing simulators precludes a simple method for bundling queries after they are issued. In effect each query must be answered sequentially in the order issued.

2.3 Region Based Visibility

Given a subspace of a virtual environment, RBV algorithms compute a visible subset of the environment from the subspace. Typically the subspace is a convex polyhedral such as a box. RBV algorithms have many applications in computer graphics. One main use is to increase rendering speeds by culling invisible geometry. RBV has also been used to decrease network traffic for remote renderings. However, it has been shown that computing RBV exactly is an $O(n^4)$ problem [13]. As with ray tracing, this a very broad topic which has been extensively researched. We provide an extremely brief overview and refer readers to recent surveys [4, 3].

To alleviate the computational complexity three classes of algorithms have been developed: approximate, conservative, and aggressive. Conservative algorithms compute a superset of the actual visible set. Aggressive algorithms compute a subset of the visible set [11]. Approximate algorithms compute a possibly bounded estimate of the visible set which may be neither a subset nor a superset. Our aim is to conservatively cull LOS queries that are definitely blocked by the environment. Thus, we restrict ourselves to conservative algorithms.

Early work in region based visibility focused on using a single occluder [5, 16], or approximated multiple occluders [7]. In order to perform more effective visibility culling, umbras of multiple occluders must be joined to create larger occluders. The concept of merging the umbras of occluders is called occluder fusion. An example of occluder fusion is presented in Figure 3. In this figure, the occluders are represented by the gray bars. The object being viewed is not blocked by any one occluder alone. However, the occlusion umbra cast by both occluders together occludes the object. The algorithm presented in [6] was one of the first to effectively exploit occluder fusion in 3D.

Two practical RBV algorithms for 3D environments that perform occluder fusion are: ray space factorization [10] and a volumetric approach [17]. These RBV algorithms perform a spatial decomposition on the environment (e.g. octree or kd-tree). RBV is then computed for every spatial region. The ray space factorization algorithm performs computations in the space of possible rays emanating from a region and uses the GPU for acceleration. We present the volumetric approach of Schaufler et al. in more detail because we have employed it for our LOS algorithm.

The volumetric approach[17] requires all occluders to be submitted as closed volumes. A spatial subdivision hierarchy is imposed on the environment. The cells of the hierarchy are marked as interior if they are completely inside an occluder, exterior if they are completely outside all occluders and boundary if they are partially inside the occluders and are at the maximum level allowed level of subdivision.

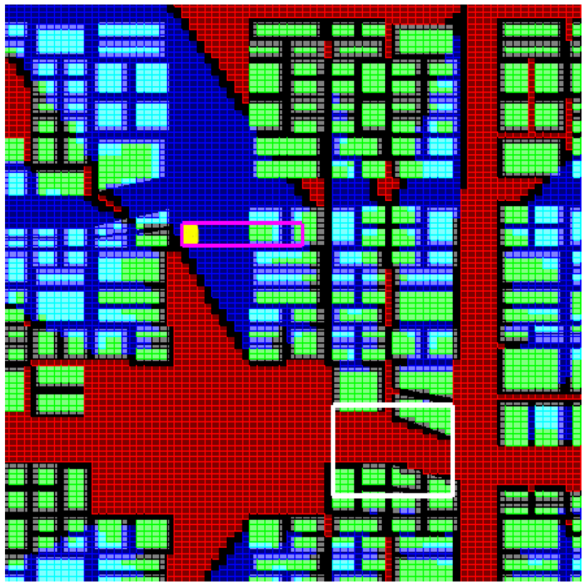

Figure 4: **Visualization of Volumetric Approach.** *This image shows a region in white, the leaf level of the hierarchy and an occluder. The red cells are exterior cells, the green cells are interior cells, and the blue cells are blocked cells. The yellow cell is current cell in the traversal. It is a blocked cell that is being used as an occluder. The extents have been enlarged to cover neighboring blocked and interior cells to create a larger occluder (magenta).*

To calculate occlusion for a region the subdivision is traversed in a hierarchical front to back manner. As the traversal proceeds each cell visited is marked as either visible or blocked with respect to the region. The traversal stops when it finds a node that can be used as an occluder. The authors show that blocked cells can be used, in addition to interior cells, as occluders to facilitate occluder fusion. Furthermore, rather than using just the cell as an occluder, the extents of the cell are enlarged to enclose neighboring cells which are also interior or blocked into a single occluder. An umbra is computed for the occluder as a set of planes and cells inside the umbra are marked as blocked. Figure 4 shows an illustration of the algorithm in progress. The output is a partitioning of the environment volume into conservatively blocked and potentially visible sets.

3 LOS USING RBV

As mentioned earlier, there are many techniques that reduce the complexity of ray casting for static scenes. These techniques are usually based on spatial subdivisions and coherent rays. Spatial subdivisions can be used to speed up LOS and, indeed, our LOS algorithm uses a kd-tree for ray casting. However, ray coherence methods do not apply well to LOS calculations. As previously discussed, LOS rays tend to be extremely incoherent. Even though many rays may originate at the same entity, they are not necessarily generated in an entity coherent order. Moreover, directions of LOS rays are extremely incoherent. Ray casting algorithms developed for ray tracing benefit from coherence in both ray origins and directions. While the kd-tree structure dramatically speeds up ray casting, we show that a large fraction of LOS rays can be culled from ray casting altogether with a simple table lookup. Thus, these techniques work in tandem to reduce the average LOS call time.

Conservative RBV can be used to accelerate LOS calculations for static environments with dynamic entities. We subdivide the environment into regions and then compute region-to-region visibil-

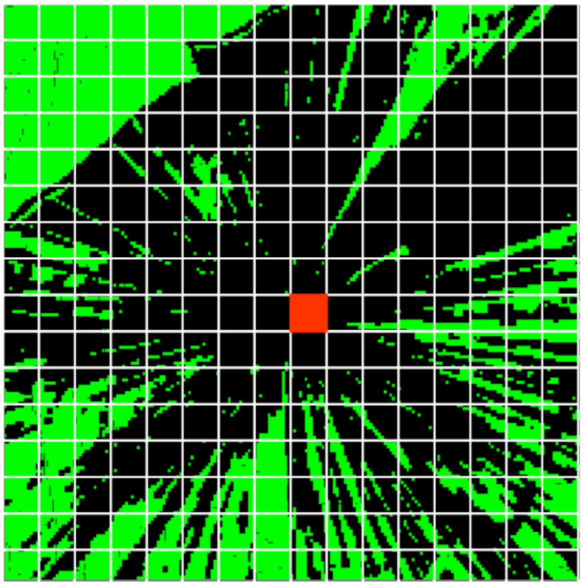

Figure 5: **Visible Cells.** *The white grid is a 2D slice of regions for which visibility is computed. The visibility table is shown in black and green. This visibility table corresponds to the red region. The cells in green represent the blocked portion of the visibility table.*

ity. If region B is blocked from region A there are no unobstructed rays from A to B. Therefore, any LOS query between an entity in A and an entity in B will be blocked. Each region stores its visibility to all other regions as a boolean table. The tables are then used to cull LOS queries at run time between regions that are blocked from each other.

We implemented both the ray space factorization [10] and volumetric [17] algorithms. We found that the results of the method of the former were highly sensitive to the order in which primitives are processed. Furthermore, we had difficulty with numerical precision issues leading to unfused umbras. Both of these problems arise from the fact that the method uses vertical slices of rays originating in the region and can only store one umbra per slice (or a small fixed number). This property can also lead to overly conservative results when occluders have significant complexity in the vertical direction. We chose to instead use the volumetric algorithm in our implementation

The main limitation of the chosen algorithm is that it requires volumetric occluders. Our test scenes included terrains and buildings that consisted of boxes. We imposed an octree on the environment and marked any cells within buildings or below the terrain as interior cells. We then applied the RBV algorithm to each region of the region grid using this octree. The algorithm uses hierarchical front-to-back traversal of the octree with occluder fusion and polygonal umbras to determine which nodes of the octree are blocked from the region. After running the RBV algorithm, the octree cells are marked as blocked or unblocked from the region. This is converted into our visibility table representation (discussed in the next section) by discovering which cells of the visibility table fall entirely within the blocked portion of the octree.

3.1 RBV Visibility Tables

Determining the resolution of the region subdivision presents a challenging tradeoff. Higher resolution allows increased culling because smaller regions are more likely to be blocked but storage of the visibility tables grows as $O(n^2)$ where n is the number of regions. We strike a compromise by using both a fine and coarse grid. We subdivide the environment into a coarse grid of regions. Each region stores a visibility table at a finer resolution. We call this finer resolution table the visibility table. Each cell of the visibility table represents a very small box in the environment. The cells store a single bit indicating whether the cell is blocked from the region according to the output of the RBV algorithm. The same set of cells is used for the visibility tables for all regions; only the boolean table is stored per region. It is important to note that the size of the visibility table depends on the overall size of the environment rather than the number of primitives. This is different from ray tracing techniques that generally depend upon the primitive count. In other words, adding geometric details to the terrain or buildings will not affect the storage size or runtime cost.

In current simulators the majority of entities are ground based and some entities are aerial based. This gives three cases for LOS queries: ground-to-ground, aerial-to-ground, and aerial-to-aerial. The aerial-to-aerial query is a fast query and is determined to be visible in most cases. The GPU culling method [15] will cull most of these queries. We do not use the RBV algorithm for aerial-to-aerial queries. The remaining aerial-to-ground and ground-to-ground queries have at least one entity on the terrain. This allows us to store the visibility table for each region as a 2D table at the level of the terrain. Assuming the z-axis points in the vertical direction, the visibility table is a uniform grid in x and y. The z extents of a cell is computed to bound the terrain surface within its xy-rectangle. The entire set of visibility tables represents a conservative approximation of the visibility from all points in 3D to the 2D terrain surface.

Figure 5 shows a visibility table in green and black for the region highlighted in red. The green pixels are cells of the visibility table that are blocked from the highlighted region, while the black pixels are not blocked.

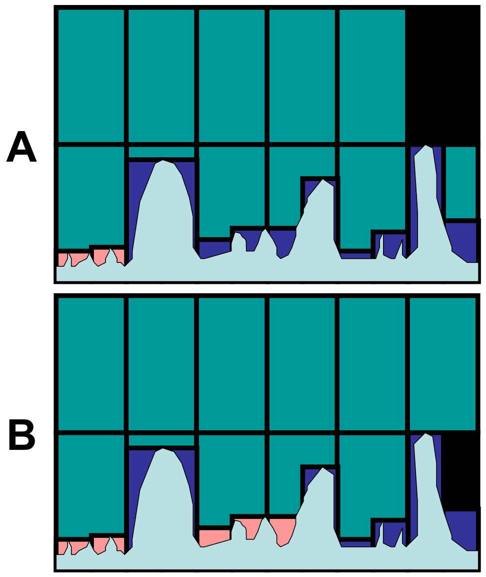

Figure 6: **Region Based Visibility Table.** *The dark blue and pink cells represent the visibility table for the region highlighted in black. These cells are smaller than the regions for which visibility is calculated. The cells making up the visibility table are 2D and the grid of regions are 3D. A) The visible cells are shown in dark blue and the blocked cells in pink for the aerial region in black. B) The same is show for the ground region in black.*

Figure 6 demonstrates how visibility tables are calculated. The heights of the cells in the visibility table are adjusted to the height of the terrain. Changes in visibility occur more frequently due to horizontal movement than vertical. Therefore our grid of regions is much coarser in the vertical direction. Figure 6 shows that the regions are taller than they are wide. These modifications reduce both the storage space and the computation time for the algorithm.

3.2 Run-Time

The run-time portion of the algorithm consists of calculating LOS between all entities in the simulation. Queries are culled first by the RBV lookup tables. Next they are culled by the GPU method described in [15]. If both of these cull tests pass, then the ray cast computation is performed.

The RBV cull test for ground-to-ground entities requires two table lookups. An LOS query contains two entities, e_1 and e_2. The first step is to identify the visibility table, V_1, of the region containing e_1. Similarly, we identify the relevant visibility table for e_2 which is labeled V_2. Then, we lookup the boolean value of the cell containing e_1 in V_2 and e_2 is looked up in V_1. If either of the lookups return false then the LOS query is culled. Because the RBV algorithm is conservative, one or both of these tables may be unblocked even though there is no visibility. The bidirectional test reduces the number of queries between blocked regions that are then ray casted due to the conservative nature of the algorithm. Both of these lookups are $O(1)$. For calculations between aerial-to-ground queries we use one table lookup. The ground unit is looked up in the visibility table of the region containing the aerial unit.

4 Results

We gathered results on two data sets. The first was a synthetic data set consisting of a flat terrain and dense tall buildings. Figure 7 shows this data set. The second scenario was an 8 by 8 kilometer region of terrain near the Caspian Sea with an urban environment consisting mostly of one and two story buildings.

In order to test this algorithm, entities were randomly distributed across the scenario. Entities moved in a random walk. LOS queries were computed between all pairs of entities. We timed all of the LOS queries with RBV culling on and again with RBV culling off. In the first scenario the query time went from 5 microseconds for each LOS call to 1 microsecond per call when the RBV based culling method was enabled. On the second scenario the average time per query was 6.2 microseconds. With culling, the average time dropped to 2.7 microseconds. In this scenario an average of 70% of queries were culled.

As this is a culling technique, results vary based on the distribution of the entities. If entities are close together they are less likely to be culled than when they are spread out. Also, the nature of the environment affects the performance of our algorithm. Greater culling can be achieved in densely occluded environments such as city blocks or mountainous terrain. However, the previous GPU culling algorithm performs well in cases of flat terrain making these two approaches complementary.

For the Caspian Sea scenario we found a region grid size of 64x64x7 worked well. Our visibility tables were 256x256. The total storage overhead for RBV culling was approximately 300MB.

The RBV pre-process generates a large table of 2D values for every region in the environment. There is a tradeoff in selecting a visibility table size. Although visibility tables entries can be represented by a single bit, this can still become quit large. However, when the visibility tables are too small, RBV does not achieve effective culling. We would like to employ compression techniques to reduce the storage overhead. However, we cannot allow decompression to cause the the table look-up time to approach the cost of a ray cast or we will lose the benefit of RBV.

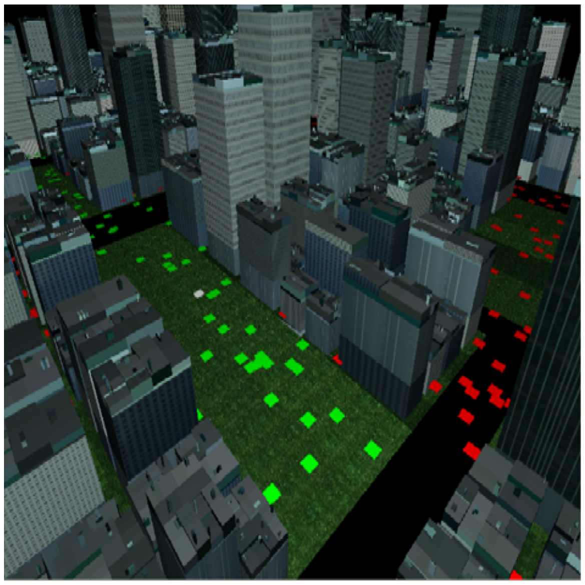

Figure 7: **Urban Environment.** *This simulated urban environment was a test simulation for region based visibility as a method for culling line of sight queries. The green squares are entities with LOS to the white entity. The red squares are entities without LOS to the white entity.*

5 Conclusion and Future Work

LOS is similar to ray tracing but its incoherence presents a different set of challenges. We have shown how RBV can be used to accelerate LOS computations by culling LOS queries. Our algorithm culls queries between regions of terrain that are blocked from each other using table lookups. Our implementation achieves significant speedups and has been incorporated in an existing simulation system.

There are several limitations to our algorithm and implementation that we would like to address in future work. Because RBV algorithms entail an extensive pre-process dynamic environments pose a challenge. Algorithms that recompute only portions of visibility tables could be developed. Also, the RBV algorithm we used requires volumetric occluders. This property may limit the types of environments that can benefit from this algorithm. We are investigating improved RBV algorithms. The visibility tables can be quite large with large homogenous regions. It should be possible to highly compress this data. We believe these limitations show promising directions for future work.

Acknowledgements: Our work was supported in part by ARO Contracts DAAD19-02-1- 0390 and W911NF-04-1-0088, NSF awards 0400134 and 0118743, ONR Contracts N00014-01-1-0067 and N00014-01-1-0496, DARPA/RDECOM Contract N61339-04-C-0043 and Intel. We would like to thank NVIDIA corporation for the hardware and driver support. We would like to thank Maria Bauer at RDECOM for the datasets and overall support for this project. We would also like to thank SAIC for their support.

References

[1] John Amanatides. Ray tracing with cones. In *Computer Graphics (SIGGRAPH '84 Proceedings)*, volume 18, pages 129–135, July 1984.

[2] Arthur Appel. Some techniques for shading machine renderings of solids. In *AFIPS 1968 Spring Joint Computer Conf.*, volume 32, pages 37–45, 1968.

[3] J. Bittner and P. Wonka. Visibility in computer graphics. *Journal of Environment and Planning B: Planning and Design*, 30(5), 2003.

[4] D. Cohen-Or, Y. Chrysanthou, C. Silva, and F. Durand. A survey of visibility for walkthrough applications. *IEEE Transactions on Visualization and Computer Graphics*, 2003.

[5] D. Cohen-Or, G. Fibich, D. Halperin, and E. Zadicario. Conservative visibility and strong occlusion for viewspace partitioning of densely occluded scenes. *Comput. Graph. Forum*, 17:C243–C253, 1998. Eurographics '98.

[6] F. Durand, G. Drettakis, J. Thollot, and C. Puech. Conservative visibility preprocessing using extended projections. *Proc. of ACM SIGGRAPH*, pages 239–248, 2000.

[7] C. Gottsman, O. Sudarsky, and J. Fayman. Optimized occlusion culling using five-dimensional subdivision. *Computer and Graphics*, 23(5):645–654, 1999.

[8] Paul S. Heckbert and Pat Hanrahan. Beam tracing polygonal objects. In *SIGGRAPH '84*, pages 119–127, 1984.

[9] D. L. Henderson. Modterrain: A proposed standard for terrain representation in entity level simulation. Master's thesis, Naval PostGraduate School, 1999.

[10] Tommer Leyvand, Olga Sorkine, and Daniel Cohen-Or. Ray space factorization for from-region visibility. *ACM Transactions on Graphics (TOG)*, 22(3):595–604, 2003.

[11] S. Nirenstein and E. Blake. Hardware accelerated visibility preprocessing using adaptive sampling. In *Eurographics Workshop on Rendering*, 2004.

[12] Line of Sight Technical Working Group. Line-of-sight compendium.

[13] Harry Plantinga and Charles R. Dyer. Visibility, occlusion, and the aspect graph. *Int. J. Comput. Vision*, 5(2):137–160, 1990.

[14] Alexander Reshetov, Alexei Soupikov, and Jim Hurley. Multi-level ray tracing algorithm. *ACM Trans. Graph.*, 24(3):1176–1185, 2005.

[15] Brian Salomon, Naga Govindaraju, Avneesh Sud, Russell Gayle, Ming Lin, and Dinesh Manocha. Accelerating line of sight computations using graphics processing units. In *24th Army Science Conference Proceedings*, 2004.

[16] C. Saona-Vázquez, I. Navazo, and P. Brunet. The visibility octree: A data structure for 3d navigation. *Comput. & Graphics*, 23(5):635–643, October 1999.

[17] G. Schaufler, J. Dorsey, X. Decoret, and F. Sillion. Conservative volumetric visibility with occluder fusion. *Proc. of ACM SIGGRAPH*, pages 229–238, 2000.

[18] Mikio Shinya, Tokiichiro Takahashi, and Seiichiro Naito. Principles and applications of pencil tracing. In *SIGGRAPH*, volume 21, pages 45–54, July 1987.

[19] Peter Shirley, Philipp Slusallek, Bill Mark, Gordon Stoll, and Ingo Wald. Introduction to real-time ray tracing. *SIGGRAPH Course Notes*, 2005.

[20] Ingo Wald, Carsten Benthin, Markus Wagner, and Philipp Slusallek. Interactive rendering with coherent ray tracing. In *Computer Graphics Forum (EUROGRAPHICS)*, volume 20, pages 153–164, 2001.

[21] Turner Whitted. An improved illumination model for shaded display. *Commun. ACM*, 23(6):343–349, 1980.