Visibility and Concealment Algorithms for 3D Simulations

Christian J. Darken
MOVES Institute and Computer Science Department
Naval Postgraduate School
Monterey, CA 93943
831-656-2095
cjdarken@nps.navy.mil

Keywords: Terrain Reasoning, Visibility, Perceptual Modeling, Cover, Concealment, Line-Of-Sight, Hiding

ABSTRACT: *Many military simulations and computer entertainment products share a need to model the ability of individual entities (men, tanks, planes, etc.) to see one another and to hide from one another in a 3D virtual environment. The traditional line-of-sight (LOS) visibility model can cause serious problems with hiding behavior. Three alternative visibility models and corresponding algorithms for computing the locations of concealment opportunities are described. Under additional assumptions, these algorithms can compute locations that provide cover from direct fire as well. Our experience implementing one of these algorithms is described.*

1. Background

Commercial computer game technology, with its gorgeous, first-person views of highly detailed environments, and custom-built systems aiming at a similar degree of visual fidelity are increasingly under consideration for military training applications. An unintended consequence of providing such a clear view of the environment is to highlight flaws in the behavior of the software agents that simulate soldiers.

Simulated soldiers must have the ability to hide. No matter how realistic the soldier appears at first glance, the illusion of reality will quickly be shattered if it cannot duck behind a nearby rock, tree, or vehicle to escape hostile fire. In the world of computer entertainment, where soldiers are almost as ubiquitous as in military simulations, products are often touted on the basis of their AI, and sometimes specifically on their ability to exploit cover. Nevertheless, we have found that the best of these products are highly flawed with regard to cover, often running by excellent nearby sources of cover to exploit an inferior one in the distance. Furthermore, when cover is reached, the agent may not conceal its entire body from view. This paper concerns approaches to solving these two problems.

How serious are these problems? Accurate simulation down to the individual combatant level of resolution is a goal of many training and analytic simulations. For training applications, failure to exploit cover clearly has the potential to destroy the immersiveness of the simulation and the emotional involvement of the participants. For analytic applications, it would seem that inefficiency in exploiting local cover might result in inaccurate results. As an extreme example, consider an assault on a unit that starts in cover. The entire burden of finding cover is on the assaulting side, and if their ability to hide is poor, their success rate will be unrealistically low.

When failures of AI such as those above are noted, the layman's lament is that today's software agents are "still not smart enough". We believe that the real problem here is not that agents are not *smart* enough, but that they are not *perceptive* enough. Software agents generally have far less information about their world than any human player gets from looking at the screen. The problem is not that agents are doing the wrong thing with the information they have, but rather that they have hopelessly little information to begin with. We believe that, at least in part, the reason why more information is not provided is because algorithms for generating and exploiting it have not been available.

In this paper, we explore algorithms for determining visibility and finding opportunities to hide that can efficiently exploit varying amounts of environmental data. The most simple and computationally inexpensive of them is capable of avoiding the worst gaffes one sees in current generation agents in the entertainment sector. The most complex is a rigorous, complete calculation of all opportunities to hide in a given region. The emphasis in this work is not on psychological fidelity, but rather on locating concealment as efficiently as possible given a certain type and amount of environmental data. The component techniques used in the algorithms have their origins in the fields of computer graphics, computer vision, robotics, and computational geometry. Much of this material has been described previously by Morgan [10].

2. Related Work

The traditional method applied in military applications [5] [6] [8] to determine the visibility of an entity from a particular viewpoint is to trace ("cast") the ray between the viewpoint and a single point located somewhere, typically top center, on the entity. This ray is known as a "line of sight" (LOS). Inspection of released source codes, e.g. [12], and informal experimentation convinces us that LOS is used as a visibility model for many commercial entertainment products as well.

As pointed out by Champion, et. al. [11], the visibility of an entity is really a question of the visibility of the entity's surface, and not of a single point. The obvious problem is that it may be the case that only a tiny region of the surface that includes the LOS target point can be obscured while most of the entity is visible. This leads to the behavior described previously, where an agent "hides" in a position that is largely visible to a known threat. Champion's approach was to empirically determine the *average* fraction of surface that is visible as a function of range and type of vegetation. The estimated visible surface area is then used to estimate the detection probability using a modified version of the empirical formula of Johnson and Lawson [13]. We will present algorithms that can be used to determine the *actual* fraction of surface visible for agents situated in a 3D environment.

Concealment opportunities are computed in [5] [6] [7] by determining the visibility of an entity at a square grid of locations. For the complex geometry and constrained computational environment typical of entertainment products, Liden [4] describes a technique where visibility is computed only at a set of hand-selected navigation nodes (waypoints).

3. Modeling Visibility

In this section, we consider three different approaches for determining what parts of one entity are visible from the viewpoint of an observer idealized as a single point (a more accurate approach would require representing two viewpoints, one for each eye). The algorithms for computing concealment opportunities presented later correspond to these visibility models.

In a 3D simulation, the environment consists of a set of polygons. Likewise, the physical form of an entity is represented as a set of polygons.

3.1. Object Space Approach

The models of each entity as well as of the environment consist of sets of polygons that exist in an abstract mathematical space we call "object space". Whereas alternative approaches always involve approximation, the goal of the object space approach is to calculate precisely which parts of the model are visible from a given viewpoint.

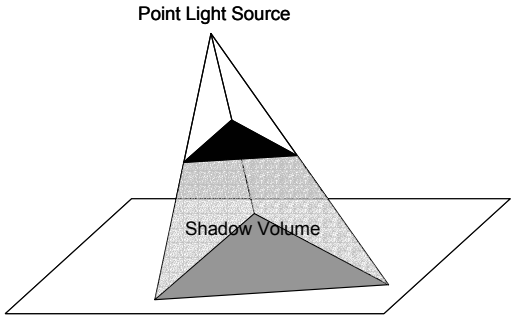

Figure 1: The shadow volume of a single triangle.

Imagine that all sources of light in the environment are removed except for a point source placed at the viewpoint of the observer. Then the parts of the entity model that are lit up correspond to those parts visible from the viewpoint. There are existing algorithms for efficiently computing shadow volumes (that part of the total volume of the environment that is in shadow) that can be directly applied to this calculation [1] [2]. The shadow volume is represented by the polygonal faces that bound it, and the parts of the model that are visible will constitute some of the faces. (This assumes that the optimization that only allows polygons that face away from the light source to cast shadows is not implemented.) If the shadow volume algorithm is implemented such that polygons that are part of the entity are labeled, then the parts of the entity's polygons that are used as shadow volume faces are visible. For example, the black triangle in Figure 1 is a face of the shaded shadow volume, and is therefore visible to a viewpoint positioned at the light source.

Figure 2: Illustration of the visibility models overlaid on a small part of a scene. The object

space approach computes the exact portion of the entity's surface visible from the viewpoint. In the picture, this portion is indicated by a white boundary line.

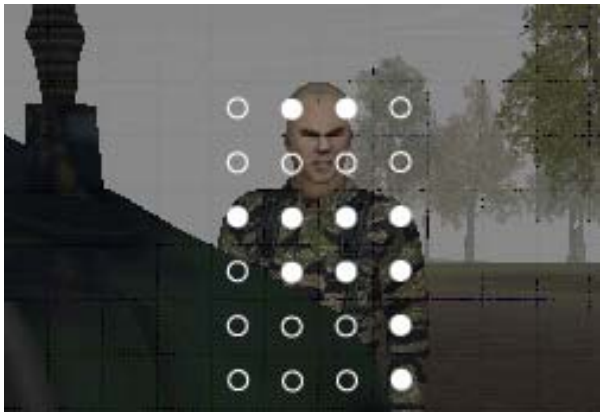

Figure 3: The raster approach computes a binary image where pixels whose center ray intersects the entity (white disks) are distinguished from those that do not (white circles). Note that the rays at the center of the two circles at the cheeks just miss intersecting the entity.

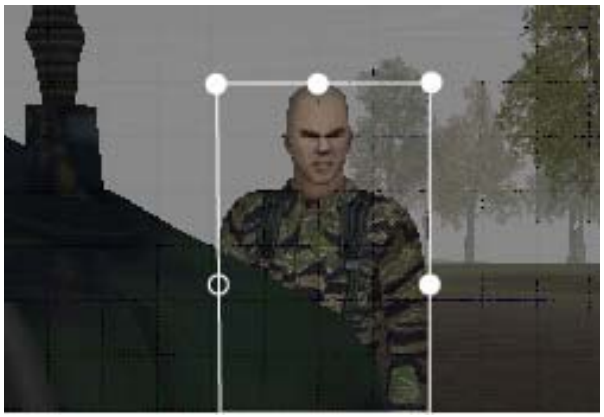

Figure 4: The multiple ray casting approach checks a selected set of rays for intersection with the entity. Here the selected rays include the one traditionally used for line-of-sight (LOS) at the top center of the entity, and several others at the corner and sides of a box centered around the entity. Note that for some box sizes, such as that above, the approach may result in a determination that the entity is visible when it actually is not.

3.2. Raster Approach

Another approach to determining what parts of an entity are visible is to perform the early steps of rendering a window just large enough to include the entire entity model from the viewpoint. The entire set of polygons in the scene is pared down to those that could possibly be visible using standard techniques (back face, view frustum, and occlusion culling), and then rasterized, i.e. put in the form of a discrete pixel matrix [15]. An extra buffer (matrix of bits the same size as the raster) may be used to record which pixels are part of the entity model. All pixels flagged as being part of the entity after all polygons have been rasterized correspond to visible points on the surface of the entity.

3.3. Multiple Ray Casting Approach

Instead of computing the visibility of the entity over a regular grid (i.e. a raster), an alternative approach is to compute the visibility of multiple representative points instead. Some obvious candidates include the position of the feet, the head, and these same points displaced horizontally half the width of the model to either side. The visibility of these points is determined by constructing the ray that originates at the viewpoint and determining whether it intersects any polygon of the environment. This functionality is typically provided by the engine used to render the environment. If the ray between the viewpoint and the target point on the entity does not intersect any polygon that is not part of the entity, then the target point is visible.

4. Concealment Algorithms

In this section, we present algorithms for computing a set of positions where an entity will be completely concealed from one or more observer viewpoints by objects in the environment. Each concealment algorithm computes concealment that corresponds to one of the visibility models presented above, i.e. shadow volume concealment corresponds to object space visibility, depth map concealment corresponds to raster visibility, and sensor grid concealment corresponds to multiple ray casting visibility. We make assumptions where necessary to simplify the presentation. Many of these assumptions can be relaxed.

Note that to be useful, concealed positions must be simultaneously:

1. Invisible to the observer.
2. "Standable" e.g. not be located in the air, or on a surface so far from horizontal that we cannot maintain our position there.
3. Unobstructed, i.e. the entity must be able to occupy the position without intersecting any solid object.

We do not discuss algorithms for obstruction testing (usually called collision testing) or for determining whether a line between two points in the environment intersects objects in the environment, since these

capabilities are typically provided by a 3D simulation engine.

Obstacles in the environment may render some or all of the concealed positions computed inaccessible to the entity in the sense that there is no feasible path from the entity's current location. Checking accessibility is beyond the scope of this paper. See Reece [5] for a discussion of the most salient path planning algorithms.

An important parameter of the first two algorithms is the approximate maximum distance that we are willing to travel to a concealed position. We call this parameter the "maximum distance of interest". The notion is that concealment that is further away from us than this distance is not interesting and not worth computing. This is a practical assumption which saves computing time and allows the application of computer graphics algorithms that we make to be in a context very similar to the usual one of rendering a view of the environment.

4.1 Shadow Volume Concealment Algorithm

The goal of the shadow volume algorithm is an exact computation of the entire set of concealed positions within a local region of the environment. For the sake of specificity, we make several modeling assumptions in the course of developing the algorithm. We believe that these assumptions match current generation systems well, and that the algorithm should be fairly straightforward to extend to systems based on similar but different assumptions.

The principle of the shadow volume algorithm is simple. If you imagine that the environment is completely unlit, and a point source of light is located at the viewpoint of each observer, then every location that is lit up is visible to the observer. Similarly, the unlit ("shadowed") locations are not visible. Efficient algorithms for shadow volume computation have been developed for computer graphics applications [1] [2]. For our application, there is no need to actually render the shadows. We will merely take the shadow volume, in the form of one or more connected regions bounded by polygons, and process it further to include standability and accessibility considerations, as discussed above.

The algorithm begins just as if we were preparing to render the environment from the point of view of the observer. A square-based pyramid is constructed with its apex at the observer's eye and it's base behind all of the geometry of the environment, i.e. effectively at infinity. The edges of the pyramid are chosen so as to be center the base on the entity's feet and to include the entire region of environment that is of interest, i.e. all points within the maximum distance of interest as defined above.

Let S be the "standable" set, i.e. the union of the set of all polygons we can stand on (either polygons must be tagged, or else we must sort through all polygons in the level according to some rule, for example, excluding those of excessive pitch). The key modeling assumption here is that the entity can stand in a certain location without falling if and only if its location is in S.

The next step is to compute the shadow volume, V. Typically, this potentially highly complex region is returned to us stored in a BSP (binary space-partitioning) tree. Efficient algorithms exist for computing Boolean operations (union, intersection, and complement) on regions stored in this form [3].

Let O be the set of all obstacles, i.e. the union of all polyhedra that are checked during collision detection. For concealment computation, points in O and V have the same interpretation: we do not want the the entity to intersect either. Therefore the set B, defined as O union V, the set of "bad" points, is computed. The concealed positions are the set of locations and orientations of our entity such that it does not intersect B. In robotics, this is known as the set of free configurations, and algorithms to compute it are known [9]. Since this computation is complex, one might consider computing a close, conservative approximation by solving the corresponding 2D problem instead. This can be done by approximating the model of the entity by a vertical cylinder with a polygonal base. S can be cut apart and projected on a plane, and likewise with the set of points in B that are potentially reachable by the cylinder. The problem has now been reduced to finding free configurations of the polygon in a plane, and can be solved with simpler algorithms such as those in Berg et. al. [14].

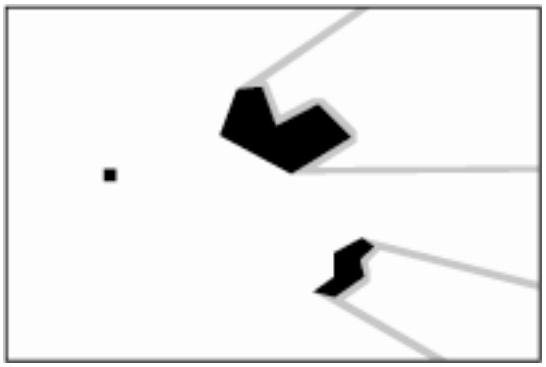

Figure 5: A 2D concealment computation. The black square is located at the viewpoint. The irregular polygons are objects in the environment.

The shadows cast by these objects are gray and include their white interiors, which represent concealed positions assuming the entity is represented by a bounding square the same size as the one at the viewpoint.

4.2. Depth Map Concealment Algorithm

One displeasing characteristic of the shadow volume algorithm is that the complexity of the concealed location representation grows with the number of edges in the scene. The depth map algorithm removes this dependence by rasterizing (putting into the form of a discrete matrix) the representation of the surface that blocks the observer's view. The price paid for this simplification is that the algorithm is able to provide only a discrete subset of concealed positions rather than the whole set.

This algorithm involves constructing a depth map from the point of view of the observer. If there are multiple observers, we select a centrally-located one to begin with, and consider the others below. The construction of such a map is a standard part of rendering a 3D environment for a player. The depth map is used to determine the parts of the environmental geometry that will be visible at each pixel location. Each polygon that could possibly influence a pixel is rasterized, and if it is the closest polygon to the viewpoint so far, the distance to the polygon (the depth) is recorded.

This algorithm begins like the shadow volume algorithm, by constructing a viewing pyramid centered on the feet of the entity large enough to include the region of the environment in which we wish to search for concealment opportunities. Instead of dealing with the full complexity of the environment's geometry in this region, the pyramid is rasterized. Conceptually, we can imagine intersecting the viewing pyramid with a plane perpendicular to the line connecting the observer's eye and the entity's feet. This square is then evenly divided along both axes to produce an array of smaller squares. The number of divisions determines the resolution of the technique. In ordinary rendering, each of the small squares corresponds to a single pixel. For our current use, the ray originating in the eye of the observer and passing through the center of each square is used to test for concealment along the ray. Rather than analyzing each ray individually, a more efficient approach is used.

For computer graphics applications, the z-buffer algorithm [15] is used to compute the distance at which each ray described above intersects a polygon in the environment. This is the closest distance at which the entity might hide while being behind the first object intersected by the ray. The matrix containing these distances is our depth map. Note that some rays may not intersect any object. The depth at these rays is infinite. For all but the coarsest buffer resolutions, the z-buffer algorithm can be expected to be faster than a sequential analysis of individual rays because it processes each polygon in the environment only once.

The next step is to determine the size of the silhouette of the entity at various relevant distances. At a minimum, we compute the silhouette size at the entity's actual range from the observer minus the radius of the region of interest. For better performance, the silhouette size at multiple ranges can be considered. The silhouette at the selected ranges is approximated as a rectangular grid of "pixels". For each possible placement of a silhouette over the depth map, the maximum of the silhouette's range and all depths under the silhouette is calculated. If the resulting distance is less than infinity, then the ray that passes through the bottom of the silhouette may contain a concealment opportunity at some range greater than this result. This ray is constructed starting at the calculated minimum distance and is then intersected with the set of "standable" surfaces. If an intersection exists, it is tested for possible obstruction. The generated position is concealed from the viewpoint of the representative observer we started with. If there are multiple observers, the position is now tested for visibility to the others using one of the visibility models previously described.

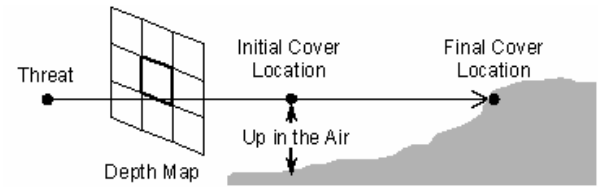

Figure 6: Intersecting the ray with the "standable" surface to find a concealed position.

Consider the example depth map below. This depth map corresponds to a view of a flat surface on which two roughly rectangular obstacles are standing at depths of approximately 40 and 60 units. "inf" stands for "infinite" and corresponds to sky. The horizon is at 100 units. Assume that the entity is currently located at a range of 60 units, and that the maximum distance of interest is 10 units. Then the closest cover opportunity of interest would be at 50 units. Assume that the silhouette of the entity has a width of two pixels and a height of three at a depth of 50 units. Consider the 2x3 rectangle outlined in black. If the entity were located with its feet along the ray passing through the bottom center of this rectangle at any distance nearer than 63 units, part of the entity would be exposed to view. Thus ray casting to find a concealed position will begin at 63 units. Consider the

gray rectangle. The maximum of the values inside the rectangle is 42, but not only would a location at this distance be outside the maximum distance of interest, but at this range the entity might not fit behind the rectangle being checked. For this reason, ray casting for this rectangle will begin at range 50.

inf	inf	inf	inf	inf	inf	inf	inf	inf	inf
inf	inf	inf	inf	inf	inf	inf	inf	inf	inf
inf	inf	inf	inf	inf	inf	inf	inf	inf	inf
inf	inf	inf	inf	inf	inf	57	inf	inf	inf
inf	inf	inf	42	inf	58	59	60	inf	inf
100	100	40	41	100	63	62	100	100	100
65	65	42	40	65	60	61	65	65	65
45	45	40	40	45	45	45	45	45	45
30	30	30	30	30	30	30	30	30	30
20	20	20	20	20	20	20	20	20	20

4.3. Sensor Grid Concealment Algorithm

Like the previous approach, the sensor grid approach works by looking for concealed positions by sampling the area around the entity. Unlike the previous approach, the specific concealment opportunities investigated are not determined by a square grid placed over the view of the observer, but rather by a grid with circular symmetry fixed with respect to the entity. This grid acts like a set of sensors testing whether the observer is visible from their location, thus the name of the approach.

The first step in the algorithm is the calculation of the absolute horizontal coordinates of each sensor from the relative coordinates of the grid template. A ray can then be traced vertically starting from the height of the entity's feet to intersect the closest standable surface. If this surface is too high or two low such that it is likely to be inaccessible, the sensor can be removed from further consideration. Visibility is tested by tracing multiple rays to the entity as if it were located at the sensor, for example to the eyes and four corners of a bounding square drawn from the perspective of the observer. A sensor is hidden if no ray from any observer viewpoint reaches the entity.

If any sensors are hidden, these represent concealed positions. These positions can then be tested to ensure that they are "standable" and unobstructed.

5. Implementation

The sensor grid approach was implemented on top of America's Army version 1.6. In addition to using the grid locations as candidate concealed positions, they were used to plan a path to the concealed position, by searching for paths between the entity's location and concealment that are either straight lines or two-segment paths via the position of a second sensor. The concealment testing and motion planning running on top of the standard game proved to be nearly instantaneous on our tests on a laptop with 1.7 GHz Pentium IV and Geforce4 440 Go graphics card. Informal testing showed successful hiding in environments featuring various types of objects including as rocks, trees, vehicles, and buildings.

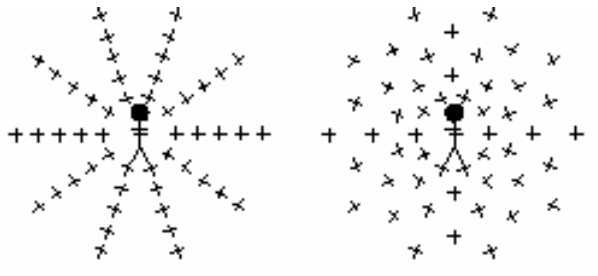

Figure 7: Construction of the sensor grid. (Left) Five circles of sensors uniformly spaced in depth and angle. (Right) The same sensors staggered in angle to form the sensor grid that was implemented. Note that the area closest to the entity is most densely covered by sensors (approx. 1.5m spacing close in versus 3m at the edge of the grid).

6. Extensions and Optimizations

6.1 Extension: Large Numbers of Observers

Generally, there will be more than one observer for the algorithms to deal with. When the number of observers is sufficiently small and their positions are known, we can represent them by the list of locations of their eyes. When there is a large number of observers or we only have a rough estimate of their position, one possible approach is to select a small set of representative viewpoints to deal with, for example, two viewpoints on the edge of the observer-occupied region of the space and one in the middle.

6.2 Extension: Partially-Transparent Polygons

We have assumed that polygons in the environment are either blocking or not blocking in their entirety. But a traditional trick is to model objects with a lacy, open structure (e.g. a tree branch) as a single polygon overlaid with a partially-transparent (alpha) texture. Taking partially-transparent polygons into account is straightforward for the depth map and sensor grid approaches, though it naturally involves extra computation to determine whether the precise location where a ray intersects a polygon is transparent. In the case of the shadow volumes, there is an additional,

possibly large, increment in the complexity of the geometry of the shadowed region. As environmental models become larger and involve greater numbers of smaller polygons, there is a trend towards modeling even fine structures with polyhedra instead of transparent textures. This will make the expedient solution of treating partially-transparent polygons as either completely blocking or completely transparent increasingly acceptable.

6.3 Extension: Computing Cover Opportunities

"Cover" implies protection from fire while in the covered position, whereas "concealment" merely implies that one cannot be seen. With only very rare exceptions (e.g. bullet-proof glass), all covered positions are concealed, but not vice versa. When the assumption that some subset of environment polygons block fire is an acceptable approximation, all of the algorithms we present can be applied to compute positions that provide cover from direct fire, i.e. from projectiles that travel on trajectories that approximate straight lines. This is accomplished simply by running the algorithms on the set of fire-blocking polygons, instead of the set of opaque polygons.

When the algorithms are applied to the computation of cover, the assumption that fire is blocked by polygons is at best an approximation. Polygons are infinitely thin and cannot themselves block fire. A completely realistic computation of cover would therefore involve the details of the velocity and material of the projectile, the stopping power of the material, and the length over which the trajectory intersects the material. The algorithms described in this paper cannot be trivially extended to handle ballistics to this degree of accuracy. Note that this concern does not affect their application for the computation of concealment. Also, one might still consider using the algorithms presented here in a preprocessing step to determine promising candidates for a more accurate cover computation.

6.4 Extension: Multiple Postures

If the model of the entity is capable of adopting more than one posture (e.g. standing, crouching, prone), the most straightforward way to take this into account is to handle each posture separately, while keeping track of the posture that corresponds to each concealment opportunity. Further investigation may yield optimization opportunities.

6.5 Optimization: Use of Graphics Hardware

The raster/depth map approach requires computations very similar to those used to render a view of the environment for a human user. These computations are normally delegated to a specialized graphics board for normal rendering, and in fact it may be possible to do the same for the algorithms described here. Note however that bandwidth from current graphics boards back to the main CPU is quite limited, and this consideration may force the computations to be done by the CPU.

6.6 Optimization: Amortized Ray Casting

When multiple rays are cast to determine visibility, note that there is no absolute need to cast all these rays in one agent decision cycle. Instead, the most critical of the rays may be cast first, for example visibility of the top of the head, and an initial choice of concealed position may be made on that basis. In later decision cycles, additional rays can be cast, and the agent can re-plan its motion if it determines that its initial choice was bad.

7. Analysis and Conclusions

The visibility models presented can be used directly as a conservative (from the point of view of the hiding entity) acquisition model, i.e. if any part of the entity is visible, then the entity can be considered acquired as a target. Note that entertainment products and many simulations treat target acquisition as an atomic process, i.e. the target is acquired at all levels from detection to identification simultaneously. Alternatively, the algorithms could be easily modified to produce an estimate of the visible surface area of an entity that could be incorporated into the variant of Johnson's probabilistic detection model presented in [11].

In terms of performance, i.e. the accuracy with which the exposed surface of the entity model is computed, it should be clear that the object space approach is the best, followed by the raster approach, with the multiple ray casting approach generally being most prone to failure. All approaches can clearly exceed the performance of the single ray cast of the traditional line-of-sight. We call failures of the visibility model Type I if they label an entity is visible when it is not, and Type II if the reverse is true.

Visibility Model	Possible Errors	
	Type I	Type II
Object Space	no	no
Raster	no	yes
Multiple Ray	yes	yes

Improved performance generally comes with increased computational cost. In general, one would expect the object space approach to cost more than the raster

approach and the the raster approach to cost more than multiple ray casting. But clearly if as the raster becomes finer and finer (higher and higher resolution), eventually it must overtake the object space approach in computational cost. Likewise, the raster can be expected to cost more than one or two ray casts. However, as the number of ray casts grows, at some point a sufficiently coarse raster will be cheaper. After all, computation of the raster with the z-buffer algorithm involves finding the set of polygons that may affect visibility only once, whereas each ray cast must perform this task.

All of the algorithms we discuss have in common that they rely on "observer's eye views", i.e. on the computation of what can or cannot be seen by a particular observer. Obviously, this is not a realistic assumption, though clearly human beings are capable of estimating what another person is able to see to some extent [7]. Nonetheless, this assumption is typical of engineering approaches to concealment detection, and we make it as well.

More realistic models of concealment finding that avoid computing visibility from an observer's point of view are clearly desirable. For example, it may be possible to build general techniques for hiding based on a depth map computed from the agent's own point of view. The agent might detect the presence of object edges as discontinuities in depth and move so as to occlude the observers with the object. In environments containing familiar cover- and concealment-providing objects like boulders, trees, or foxholes, object recognition (perhaps simulated simply by object labeling) should be included.

Additional investigation should be done to determine if an additional layer of processing on top of the raster visibility algorithm described here could yield a new empirical predictive model of the target acquisition ability of the human eye that is superior to Johnson's [13]. Such an algorithm could handle cases when the entity's surface is visible to the observer, but is not detectable because of shadows or camouflage.

Finally, the asymmetric manner in which multiple observers are handled by the depth map algorithm may lead to failure to find existing concealment opportunities when the observers are widely dispersed. Further research is necessary to address this issue.

Computing visibility is already a major component of the computational budget devoted to AI for both military simulations and entertainment products. Nonetheless, we believe that further investment in perceptual modeling, together with careful algorithm optimization, can yield large benefits.

8. Acknowledgements

This work was supported by funds from the Naval Postgraduate School, the Navy Modeling and Simulation Management Office, and the U.S. Army Training and Doctrine Analysis Center, Monterey.

9. References

[1] Chin, N. and Feiner, S., "Fast Object-Precision Shadow Generation For Area Light Sources Using BSP Trees." Proceedings of the 1992 Symposium on Interactive 3D Graphics. March 1992.

[2] Chrysanthou, Y. and Slater, M., "Shadow Volume BSP Trees for Computation of Shadows in Dynamic Scenes." paper presented at the 1995 Symposium on Interactive 3D Graphics. Monterey, CA, 1995.

[3] Schneider, P. and Eberly, D., *Geometric Tools for Computer Graphics*, Morgan Kaufman, 2002.

[4] Liden, L., "Strategic and Tactical Reasoning with Waypoints." AI Game Programming Wisdom. pp 211-220. Charles River Media, 2002.

[5] Reece, D., "Movement Behavior for Soldier Agents on a Virtual Battlefield." *Presence*, Vol. 12, No. 4, pp. 387—410, August 2003.

[6] Reece, D., Dumanoir, P., "Tactical Movement Planning for Individual Combatants", Proceedings of the 9th Conference on Computer Generated Forces and Behavioral Representation, 2000. Available at http://www.sisostds.org.

[7] Kelly, J., Beall, A., and Loomis, J., "Perception of Shared Visual Space: Establishing Common Ground in Real and Virtual Environments", to appear in *Presence.*

[8] Richbourg, R., and Olson, W., "A Hybrid Expert System that Combines Technologies to Address the Problem of Military Terrain Analysis," *Expert Systems with Applications,* Vol. 11, No. 2, pp. 207—225, 1996.

[9] Latombe, J., *Robot Motion Planning,* Kluwer Academic, 1991.

[10] Morgan, D., "Algorithmic Approaches to Finding Cover in Three-Dimensional Virtual Environments", Master's Thesis, Naval Postgraduate School, 2003. Available at http://www.movesinstitute.org

[11] Champion, D., Fatale, L., and Krause, P., "Effects of Vegetation on Line-Of-Sight (LOS) for Dismounted Infantry Operations", TRAC White Sands Technical Report TRAC-WSMR-TR-99-001(R), June 1999.

[12] Epic Games, UnrealScript Source to "Unreal Tournament", 1999.
[13] Johnson, J. and Lawson, W., "Performance Modeling Methods and Problems", *Proc. of the IRIS Imaging Systems Group,* Jan. 1974.
[14] Berg, M., van Krevald, M., Overmars, M., and Schwarzkopf, O., *Computational Geometry: Algorithms and Applications (2nd Ed.),* Springer, 2000.
[15] Foley, J., van Dam, A., Feiner, S., and Hughes, J., *Computer Graphics: Principles and Practice in C (2nd Ed.),* Addison-Wesley, 1995.

Author Biography

CHRISTIAN DARKEN is an Associate Professor of Computer Science at the Naval Postgraduate School, where he also collaborates intensively with the MOVES Institute. Previously he was Project Manager of the Decision Support Systems project and Senior Member of Technical Staff at Siemens Corporate Research in Princeton, NJ, where he was variously associated with the Learning Systems, Adaptive Information and Signal Processing, and Software Engineering Departments. He was also a programmer of one of the first commercial first-person perspective massively-multiplayer games. He received his Ph.D. in Electrical Engineering from Yale University in 1993, and previously received the M.S. and M.Phil. in Physics from the same institution.