The Observer Algorithm for Visibility Approximation

Per-Magnus OLSSON [a] and Patrick DOHERTY [a]
[a] *Department of Computer and Information Science. Linköping University*
{perol, patdo}@ida.liu.se

Abstract. We present a novel algorithm for visibility approximation that is substantially faster than ray casting based algorithms. The algorithm does not require extensive preprocessing or specialized hardware as most other algorithms do. We test this algorithm in several settings: rural, mountainous and urban areas, with different view ranges and grid cell sizes. By changing the size of the grid cells that the algorithm uses, it is possible to tailor the algorithm between speed and accuracy.

Keywords. Visibility, occlusion calculation, unmanned aerial vehicles, constrained path planning.

Introduction

We wish to perform visibility calculations as a part of a larger framework used for planning and simulating missions for unmanned aerial vehicles (UAVs). One part of the planning process is path planning where we search for a path between two positions. As the search is done in a graph which can cover large areas with arbitrary granularity, a large number of nodes might be used. We wish to calculate whether the nodes are visible from some positions, which can be the locations of our own personnel that the UAV should be visible to at all times or the locations of potential adversaries that we wish to stay out of sight from. When the node visibility has been calculated, it can be used when doing the path planning. This makes it possible to force a UAV to use a path that fulfills certain visibility constraints. The requirements make it difficult to use existing algorithms to determine visibility and this is the reason for developing an algorithm for visibility approximation.

1. Previous Work

In computer graphics, visibility calculations are used to determine what objects are visible and should be rendered on the screen. Frustum culling is used to remove (cull) the objects that are outside the viewer's field of view (frustum), from the rendering queue. Occlusion culling is used to remove the objects that are occluded by other objects and thus not visible. An example is the occlusion culling of a small object that is occluded by a larger object. A lot of research in computer graphics has been based on hierarchical spatial partitioning methods such as quadtrees, as well as potentially visible sets (PVS) and portals [1]. It is also common to use specialized graphics hardware to improve the performance of culling algorithms. Earlier work has mainly dealt with indoor environments due to the computational complexity of outdoor environments. Occluders and occluder shadows has previously been used in an algorithm to determine visibility [2]. Good results were achieved for urban environments if the occluders could be determined in advance. The requirements on the occluder makes it is difficult to use the algorithm in a non urban environment. A survey of different visibility algorithms in computer graphics is provided in [3]. Most of these require either extensive preprocessing and/or specialized graphics hardware.

In the area of robotics, visibility is often used as a means to calculate areas that have low visibility. Several algorithms for visibility calculations in 2D environments have been devised [4]. These have been used in settings where a robot tries to avoid being seen by one or more sentries while moving to a goal position. Algorithms for finding a covert paths in the presence of stationary and moving sentries has been devised by [5] [6]. An approximate visibility algorithm was devised and used in a simulated environment [6]. That algorithm requires means of finding the visibility in some different directions, and uses that data to get an approximate visibility measure in all other directions.

A ray casting algorithm shoots an infinitely thin ray from the start position to the goal position. As the ray is moved towards the goal, it is checked for intersection with objects in the environment [1]. Ray casting algorithms are often used in Geographic Information Systems (GIS). GIS are performing visibility calculations to determine what is visible from a certain location. Few companies are willing to disclose information about how this is done, but it is believed that these calculations are performed with massive amounts of ray casting [7]. Often a ray for every 0.1 degrees is used, which often results in poor performance due to the amount of calculations that have to be performed. GIS systems normally calculate visibility to certain positions in a horizontal plane and positions outside the plane are omitted. Ray casting algorithms work directly on the polygons that model the world and the objects. In games and simulators, the amount of polygons used to model the environment has increased steadily for several years and the increase is expected to continue in the future. Consequently the time spent ray casting is also expected to increase as environments get more complex and detailed. As the ray casting algorithm is exact, it will give the correct answer as long as all objects are included in the intersection checking. An often neglected implication of using a fixed number of ray casts to calculate visibility in circles is that as the distance from the originating point increases, the distance between the individual rays also increase and the resemblance between the visibility calculated by the rays and the actual visibility decreases.

2. The Observer Visibility Algorithm

In this section we present a new approximating visibility algorithm called the Observer algorithm.

2.1. Motivation

In our research laboratory we have several applications that are used in our research in the area of unmanned aerial vehicles [8]. One of these applications is used for mission planning. In this application we visualize the environment including terrain, buildings and other objects. The terrain consists of a height field with superimposed meshes for buildings and other objects. A grid with square grid cells is placed in a top down view of the world. The grid cell is considered to have the same elevation value in the whole cell, and this value is determined by sampling the height field in the middle of the cell. As grid cells are considered atomic and are the smallest entities used in calculations, we consider grid cells as either visible or not visible.

In this setting we want to determine the visibility from some position in all directions on the ground as well as in the air. Due to the large amount of ray casts that are necessary to determine visibility in all directions, and the potentially very large number of polygons used in modeling the environment, it is not always feasible to use ray casting methods. The algorithm must be able to handle both rural and urban terrain in different scales without time consuming precalculations or dependence on specialized hardware. Even at low resolution, the amount of memory required to store preprocessed data would be prohibitively large, and thus we can not depend on preprocessing. As most existing algorithms can not be used and we can accept the use of an approximate algorithm, we set out to formulate a new algorithm.

2.2. Algorithm Formulation

To represent what is visible from a certain position, we introduce the concept of an *observer* . The observer is an abstract entity that can represent a human, or any other entity for which we can measure the field of view (FOV). The observer has a position, heading and pitch. Other parameters are maximum view range as well as horizontal and vertical FOV. The horizontal FOV is centered around the heading, and similarly the vertical FOV is centered around the pitch. If there are no intersecting objects and the horizontal field of view is 360 degrees and the vertical field of view is 180 degrees, a sphere is formed around the viewer's position. As the observer is positioned in the middle of the sphere, all calculations should start in the middle and continue outwards toward the circumference. As a sphere forms a circle when viewed in two dimensions, we can calculate the grid cells in a horizontal plane, and then calculate the values for top and bottom of the FOV each cell.

This also allows us to take advantage of the fact that the terrain generally is 2.5D, i.e. a height field where each (x, z) coordinate pair corresponds to a single y-value. We rely on the fact that buildings and other objects can be seen as being a part of the height field when using an orthonormal projection. We wish to use a higher level representation of the world, so we use grid cells and store the elevation for each cell in a tessellated height field.

With the intuition in mind about an observer's FOV being a sphere, an algorithm was devised to work accordingly. The algorithm consists of three parts. The first part calculates the set of grid cells that make up the circumference of the observer's view range, adjusted for the edges of the simulated world and the observer's horizontal FOV. This can be done for example using Bresenham's circle algorithm [9] which is used here. The circle algorithm calculates the coordinates for points along the circle's circumference. The points are checked to determine if they are inside the grid as well as inside the observer's horizontal FOV. If a point is not inside the observer's FOV, it is discarded. If a point is not inside the grid, it replaced by the closest point that is inside the grid in the direction towards the observer.

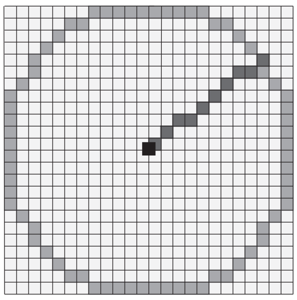

Figure 1. Schematic view of the circumference cells and traversal along a line, starting from the observer in the middle, marked as a black square.

The second part of the algorithm determines which grid cells are visible along a straight line from the observer to a grid cell located at the circumference of the view field. The grid cells from the first part of the algorithm are used here, as they represent the edge of view field. A general line drawing algorithm is used here, to determine which cells are covered, and we use a modified version of the Bresenham line algorithm [9]. The grid cells are traversed from the observer's position and outward towards the edge of the view field circumference along the grid cells determined by the line algorithm. As the line drawing algorithm is discrete, it will go through a number of grid cells along the traversal. These potentially visible cells are further analyzed in the final part of the algorithm. An overview of the first two parts of the algorithm is shown in figure 1.

The third part is performed for each potentially visible grid cell, where the values for the top and bottom of the view field are calculated for each cell. Depending on the terrain height and the values of the FOV, three possible cases exist:

- If the top of the FOV is less or equal to the grid cell's height, the grid cell is set to be blocked, as the object in the grid cell is higher than the FOV. Thus the grid cell is not visible and the analysis along this line is terminated.
- If the bottom of the FOV is above the grid cell height, the grid cell is marked as not visible, as the FOV is entirely above the height in the grid cell.
- If the bottom of the FOV is equal to or below the grid cell height, the grid cell is marked visible and the equation for the bottom of the FOV is set to match the height in the grid cell.

A schematic picture of the three cases is shown in figure 2. If the grid cell is either not visible as the height in the grid cell is below the view field or the grid cell is visible, the analysis continues along the line. When the visibility for all grid cells along a line has been calculated, the process is repeated along the next line. As this is done, the new line will sometimes go through grid cells that have their visibility already calculated. In that case, the old values are copied and the analysis continues. The output of the algorithm is a matrix showing the visibility for all grid cells within view range at ground level, as well as the maximum and minimum height values for the field of view in all non blocked grid cells.

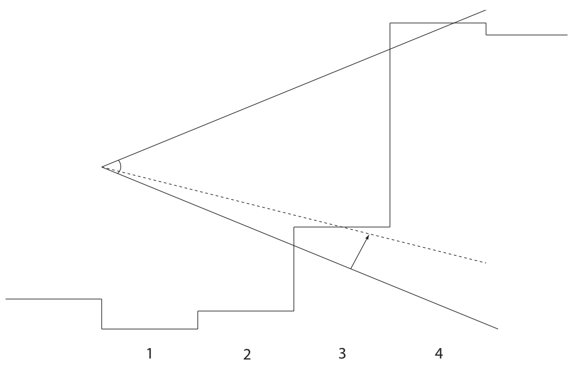

Figure 2. View of the third part of the visibility calculations, depicting the three cases. The first two cells are not visible, the third one is visible and the bottom of the field of view is adjusted to match the height of the third cell. The fourth cell is blocked as the object in the cell is higher than the top of the field of view.

2.3. Algorithm Pseudo Code

Line (1) makes up the first part and lines (2) - (3) are the second part. The third part consists of lines (4) - (14).

```
(1)  circumferenceCells = GetCircumferenceCells(observer pos, observer FOV);
(2)  for all circumferenceCells
(3)    lineCells = GetLineCells(observer pos, circumferenceCell);
(4)    for all lineCells
(5)      CalculateFOV(lineCell pos);
(6)      if FOV top <= lineCell height
(7)        lineCell visibility := blocked;
(8)        break for loop;
(9)      else if FOV bottom > lineCell height
(10)       lineCell visibility := not visible;
(11)     else if FOV bottom <= lineCell height
(12)       lineCell visibility := visible;
(13)       AdjustFOV(lineCell pos);
(14)     endif
(15)   endfor
(16) endfor
```

2.4. Theoretical Properties

For an observer with a 360 degree horizontal and 180 degree vertical FOV, where the FOV forms a sphere around the observer, the time complexity is O(n^2), where *n* is the number of grid cells along an observer's view range. At each step, positions for top and bottom of the view field are calculated, which is O(1). A ray casting algorithm needs to perform O(n^3) ray casts to determine the visibility of the volumes in a sphere. Also, the operations performed are different for the two algorithms, which make a large difference in practice.

3. Results

A series of tests were run to determine the performance of the algorithm. As the algorithm is approximate, we investigated how accurate it was compared to a ray casting algorithm and the time it took to perform the calculations.

3.1. Test Setup

For testing in a rural area, we used height data measuring 1100*900 meters in which the height data was used to create a height field. The elevations from buildings and other objects in the area were included in the height field. A tessellated height field was created, with square grid cells, whose cell sizes are shown in the tables 1 - 6. The grid cells form the base for cubic grid volumes, with all sides of equal length. A mountainous area was created by scaling the elevation of the rural terrain by a factor of 15, yielding elevations up to 600 m. For testing in an urban setting, we used a 1000*1000 m urban area with several large buildings, up to 300 m high. In all cases, the visibility was restricted to a maximum height of 500 m above the lowest point in the height field.

Within the volume 100 random positions were generated. All positions were required to be at least 40 m away from each other, and an observer was placed at each position. All observers had a 360 degree horizontal and 180 degree vertical FOV. The view range was changed between the tests, 100 m and 200 m were used. When all positions had been generated, the visibility algorithm was executed including creating all observers and calculating the visible grid cells.

For comparison, a ray casting method was used. The ray casting algorithm was taken from the Open Dynamics Engine [10] which is a high performance physics engine. From each position, the set of potentially visible grid volumes was determined, using position and view range. A ray cast was made from the observer's position to an end position in the middle of each grid volume. If no object was intersected between the starting position and the end position, the grid cell was judged to be visible. The set of visible volumes calculated by the ray casting algorithm was compared to the set of visible volumes calculated using the observer visibility algorithm. All tests were run on an Intel Core2Duo 2.4GHz processor running Debian Linux. The calculations were performed once and the results are shown in tables 1-6.

3.2. Quality Measure

As the observer algorithm is approximate, we compare the result of the calculated visibility with the exact ray casting method. For each position, the sets of visible grid volumes are calculated using both methods and compared. If a grid volume is visible to both methods, it is judged "Correct". If it is only visible to the observer algorithm, it is judged "False Positive" and if it is only visible to the ray casting algorithm it is "False Negative". The tables show the fractions of grid volumes in each category.

3.3. Test Results

The difference between doing intersection test on triangles versus the tessellated height field becomes more apparent with the longer view range and smaller grid size, and the difference in time is expected to continue to increase as view ranges are further increased

Table 1. Measures for algorithms in a rural setting using 100 m view range.

	Time [s]		Quality		
Cell size [m]	Observer	Ray casting	Correct	False Positive	False Negative
5	0.66	246.93	0.8915	1.8E-4	0.1084
10	0.09	32.84	0.8111	2.9E-4	0.1886
15	0.03	12.19	0.7439	5.3E-4	0.2555
20	0.02	5.78	0.7390	2.8E-4	0.2608
30	< 0.01	2.06	0.6897	0	0.3103
40	< 0.01	0.88	0.5922	0.0011	0.4067

Table 2. Measures for algorithms in a rural setting using 200 m view range.

	Time [s]		Quality		
Cell size [m]	Observer	Ray casting	Correct	False Positive	False Negative
5	6.43	7547	0.9290	4.1E-4	0.0706
10	0.58	953.4	0.8989	6.4E-4	0.1005
15	0.16	335.81	0.8702	0.0012	0.1286
20	0.07	151.15	0.8245	8.5E-4	0.1746
30	0.02	50.36	0.7629	2.3E-4	0.2369
40	0.02	18.43	0.7543	0.022	0.2435

Table 3. Measures for algorithms in a mountain setting using 100 m view range.

	Time [s]		Quality		
Cell size [m]	Observer	Ray casting	Correct	False Positive	False Negative
5	0.62	2859	0.8804	0.0165	0.1032
10	0.09	402.42	0.7991	0.0236	0.1773
15	0.03	132.73	0.7366	0.0241	0.2393
20	0.02	62.86	0.7251	0.0283	0.2466
30	< 0.01	21.94	0.6768	0.0335	0.2897
40	< 0.01	11.53	0.5898	0.0284	0.3817

and grid cell sizes are decreased. From tables 1, 3 and 5 we can see that execution time for the Observer algorithm is very similar despite very different kinds of terrain. The same applies for tables 2, 4 and 6. The correspondence with the ray casting algorithm improves as grid cell sizes decrease, which is to be expected as the grid cells used for storing the height values is smaller and more information is stored. Although the algorithm here is approximate, it yields good results quickly when the grid cells are small. The algorithm is approximately correct and is used to calculate visibility constrained paths, and we can rely on a priori collision detection calculations and on board proximity sensors to avoid any potential obstacles that might have been neglected by the algorithm.

Both the quality and the performance with respect to time is good when used with grid cells that are among the smaller of the grid cell sizes tested here, and it is feasible to use the observer algorithm for real time visibility calculations in our application area where the view ranges used sometimes are longer than tested here. We consider this algorithm an alternative to existing ray casting based algorithms, especially since it is possible for a user to choose between speed and accuracy by changing the cell size.

Table 4. Measures for algorithms in a mountain setting using 200 m view range.

	Time [s]		Quality		
Cell size [m]	Observer	Ray casting	Correct	False Positive	False Negative
5	5.77	71274	0.9053	0.0295	0.0653
10	0.52	9413	0.8617	0.0451	0.0932
15	0.14	2927	0.8296	0.0545	0.1159
20	0.06	1325.66	0.7838	0.0585	0.1577
30	0.02	427.27	0.7253	0.0695	0.2052
40	0.02	198.32	0.7132	0.0748	0.2120

Table 5. Measures for algorithms in an urban setting using 100 m view range.

	Time [s]		Quality		
Cell size [m]	Observer	Ray casting	Correct	False Positive	False Negative
5	0.66	1646.82	0.8434	9.3E-4	0.1556
10	0.09	235.19	0.7661	0.0018	0.2321
15	0.03	75.09	0.7001	0.0027	0.2972
20	0.02	35.85	0.6916	0.0035	0.3048
30	<0.01	12.83	0.6332	0.0073	0.3595
40	<0.01	7.22	0.5504	0.0089	0.4407

Table 6. Measures for algorithms in an urban setting using 200 m view range.

	Time [s]		Quality		
Cell size [m]	Observer	Ray casting	Correct	False Positive	False Negative
5	6.61	44377	0.8757	0.0016	0.1227
10	0.59	5960	0.8453	0.0034	0.1513
15	0.15	1826	0.8140	0.0054	0.1806
20	0.07	822	0.7714	0.0063	0.2223
30	0.02	271.59	0.7008	0.0091	0.2901
40	0.02	131.36	0.6960	0.0160	0.2880

4. Future Work

If the grid cells are too large to handle small objects, or these objects are not located where the ground is sampled, these are not taken into consideration by the algorithm. This is because they are not large enough to make an impression when sampling the height field, and they are thus not included in the visibility calculations. If the small objects are not included in the calculations this leads to overestimation of visibility and this can be decreased by sampling the height values in several positions in each grid cell, and using the highest found value for the whole cell. This will decrease the amount of "False Positive" but can also increase the "False Negatives".

Occasionally the visibilty for some cell may not be calculated because no line will go through that cell, depending on which line traversal algorithm is used. This will leave gaps in the calculations which can be remedied by adding extra cells along the circumference and calculate visibility along lines ending with these cells as normal.

As described here, the algorithm can not handle a terrain that is three dimensional. This can be improved by storing all occurring height values for the cells. When the observer's FOV intersects such cell with several height values, it would be split into several FOVs and each one would have to be handled separately.

5. Conclusion

We have presented a algorithm for visibility approximation that is a substantial speed up compared to ray casting based algorithms. It differs from many other visibility algorithms as it does not require extensive preprocessing or specialized hardware. As the algorithm does not depend on the polygons that model the world, the performance can be expected to be good even when used in complex and detailed environments. Despite the simplicity of the algorithm it yields good results in different terrains as well as with different grid resolutions. It differs from other algorithms as it is possible to choose between speed and accuracy. Testing has been performed in a framework for UAV mission planning, where it enables real time visibility calculations from several positions and with varying parameters. In our application, the resulting visibility data is used as input to a path planning algorithm, enabling real time calculation of paths that fulfill certain visibility constraints.

Acknowledgements

We wish to thank Kaj Holmberg, Department of Mathematics, Linköping University for the environment used for testing in the urban setting. This work is supported in part by the National Aeronautics Research Program NFFP04 S4203 and the Strategic Research Center MOVIII, founded by the Swedish Foundation for Strategic Research, SSF.

References

[1] D. Hearn and P. Baker, *Computer Graphics with OpenGL*, Prentice-Hall, 2003.

[2] P. Wonka, Occlusion Culling for Real-Time Rendering of Urban Environments, *Institute of Computer Graphics, Vienna University of Technology*, 2001.

[3] D. Cohen-Or and Y. Chrysanthou and C.T. Silva and F. Durand, A Survey of Visibility for Walkthrough Algorithms, *Transactions on Visualization and Computer Graphics* **9(3)** (2003), 412-431.

[4] M. Marzouqi and R. A. Jarvis, *Covert Robotics: Covert Path Planning for Autonomous Robot Navigation in Known Environments*, Proceedings of the Australasian Conference on Robotics and Automation, 2003.

[5] A.Y. Teng and D. DeMenthon and L.S. Davis, Stealth Terrain Navigation, *IEEE Transactions on Systems, Man and Cybernetics* **23(1)** (1993), 96-110.

[6] M. Marzouqi and R. A. Jarvis, *Fast Visibility Evaluation for Covert Robotics Path Planning*, Proceedings of the IEEE International Workshop in Safety, Security and Rescue Robotics, 2005.

[7] S. Rana, *Development of ray tracing algorithms in GIS for urban visibility analysis*, presented at the 2006 Meeting of the Association of American Geographers, 7-11 March 2006.

[8] P. Doherty and P. Haslum and F. Heintz and T. Merz and P. Nyblom and T. Persson and B. Wingman, *A Distributed Architecture for Autonomous Unmanned Aerial Vehicle Experimentation*, Proceedings of the 7th International Symposium on Distributed Autonomous Systems, 2004.

[9] J. Bresenham, Algorithm for computer control of a digital plotter, *IBM Systems journal* **4(1)** (1965), 25-30.

[10] Open Dynamics Engine, http://www.ode.org.