Partial Visibility for Virtual Reality Applications

D. Pearce
School of Computing Sciences, UEA,
Norwich,
NR4 7TJ.
danny.pearce@uea.ac.uk

A. M. Day
School of Computing Sciences, UEA,
Norwich,
NR4 7TJ.
amd@cmp.uea.ac.uk

ABSTRACT

This paper presents new approximated visibility algorithms. The aim is to develop output sensitive algorithms for virtual environment walk-through applications. We aim to achieve efficient rendering of complex computer models containing partially occluded areas. Emphasis is paid to the rendering of natural environment models. The algorithms presented in this paper perform visibility calculations in a plane, these algorithms have a more general 3D analogy which has many practical applications within the field of virtual environments. The algorithms presented here have been used with minor extensions to be applied to 2.5D ground based virtual environment walk-throughs.

Keywords
Virtual environments, real-time rendering, occlusion culling, level of detail.

1. INTRODUCTION

Visibility determination has been an area of interest since the start of the computer graphics field, initially to solve the hidden surface removal (HSR) problem [App68] [Sut74]. Visibility determination is now an area of research focused on the efficient rendering of 3D datasets, efficient network communication and lighting calculations. Visibility culling of densely occluded scenes has been the focus of much research, initially presented in [Air91] and then advanced in [Tel92]. In this paper we are concerned with visibility culling for the efficient rendering of large densely occluded scenes populated with many small rendering primitives such as forest scenes. We focus on an approach which efficiently rejects invisible geometry before submission to the graphics pipeline, where HSR is performed by the z-buffer.

During the rendering of a densely occluded scene populated with small rendering primitives, from many observer viewpoints much of the scene will be partially visible (or not visible at all) due to being viewed through a cluster of primitives causing restricted or partial visibility. Partially visible objects are often less important than completely visible objects as the details of the object are likely to be less noticeable to an observer. Therefore, such objects can be displayed in reduced detail [And00] [Pea03].

In scenes with many small rendering primitives, typically no geometry exists which can be used as effective occluders. We can attempt to create virtual occluders [Kol00], which can be used in visibility culling algorithms. However, this is an exact approach and there is no way of ensuring that sufficient virtual occluders can be created to allow efficient rendering. Additionally, this approach does not naturally allow for level of detail switching based on visibility. We can also attempt to use occluder fusion techniques [Won00], which combine several disjoint occluders into a single occluder. While this has similar problems to the virtual occluder approach it is also computationally expensive. Particularly in a natural environment, where there are likely to be many small rendering primitives to combine.

We present algorithms for computing "from-region" visibility information at a pre-processing stage, the visibility information is stored and recalled during run-time. The advantages of this approach are that it has negligible run-time CPU costs and certain predictive capabilities which allow loading of the dataset into memory from disk or alternative source (such as a network) as and when it is needed. The main problem for the pre-computed approach is memory requirements to store the visibility information, since the memory consumptions are likely to be high in large scenes, but can be reduced using existing techniques to compress the visibility data or intelligent selection of cell sizes [Nad99]

[Got99] [Pan99]. Additionally, visibility information can be stored in a central location and streamed from a network [Coh98a] [Coh98b] if it is too costly to store entirely on a standard PC.

We present run-time approximations of visibility information similar to that of [And00] [Sao99] [Coh98a]. The main differences of our approach are the data structures used to achieve efficient approximation of visibility information, and we focus on using spatial cells which more compactly store visibility information. We present two variations of visibility computation, the first computes visibility information for the entire dataset at run-time and has no memory overheads other than those required for the data-structures used by the algorithm (which is minimal). The second approach is performed on stored visibility information computed by a pre-computed approach, where visibility information is pre-computed and stored for each spatial cell. A smaller subset of the stored visibility information is produced at run-time by using a more accurate culling when specific viewing variables are available.

The main drawback of approximated visibility techniques is their tendency to allow for errors. This is due to objects appearing to flicker where they are incorrectly approximated as invisible when they are not. In [Pea03] we discussed why this is less noticeable during the rendering of natural environments using the Partial Visibility technique. When objects typically switch to lower levels of detail or culled objects when they are partially visible, this masks the popping effects. Additionally, image based techniques can be used to eliminate this problem, where a simple backdrop is used to replace areas where much culling is performed.

An area of recent research has been to identify more effective occluders to improve the performance of visibility culling algorithms [Ber00] [Law99] [Bru01] [Kol00] [Zha98]. We describe an approach to automatically synthesise partial occluders to improve the performance of the Partial Visibility algorithms.

This paper presents the problems and concepts involved in the new partial visibility algorithms, our ideas are presented in a 2D model domain. This model has a 3D analogy which can be applied to general 3D computer models. We discuss the potential applications of these algorithms in the field of virtual environments.

2. DATA STRUCTURES AND CONCEPTS

A partial occluder occupies a volume of space which restricts visibility through the volume by some amount, known as the solidity value (SV). The solidity value is the percentage of the partial occluders' volume that obstructs light from passing the volume. This information is used during the visibility approximation algorithms to approximate a visibility value (VV) of each object or subspace in the scene. Additional visibility information (such as hole information) may be stored with the partial occluder if necessary, to allow more reliable visibility approximations.

A shadow volume (or area in 2D) is computed for each partial occluder given an observer position. The portion of the scene which lies inside a partial occluders' shadow volume is said to be partially occluded by the partial occluder. For a partially occluded object (or subspace of the scene) a visibility value (VV) is approximated. The visibility value of an object is the percentage of the object that is visible from a given observer position. A visibility value is approximated using the set of the shadow volumes that partially occludes the object.

The ideas presented in this paper are based around a number of separate problems. The problems that we are posed with are as follows: given a set of rendering primitives (or computer model) such as that in Figure 1. We attempt to find the areas of the scene which act as good partial occluders, then identify the areas that are useful free cells. Free cells are non-partially occluding areas from which visibility calculations are performed. Figure 2 illustrates partial occluders as shaded rectangles and free spaces as clear rectangles synthesised for the park scene in Figure 1.

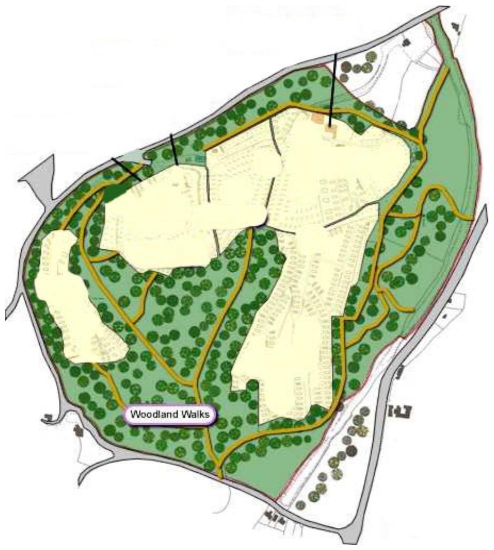

Figure 1: Natural scene simplified to a planar model

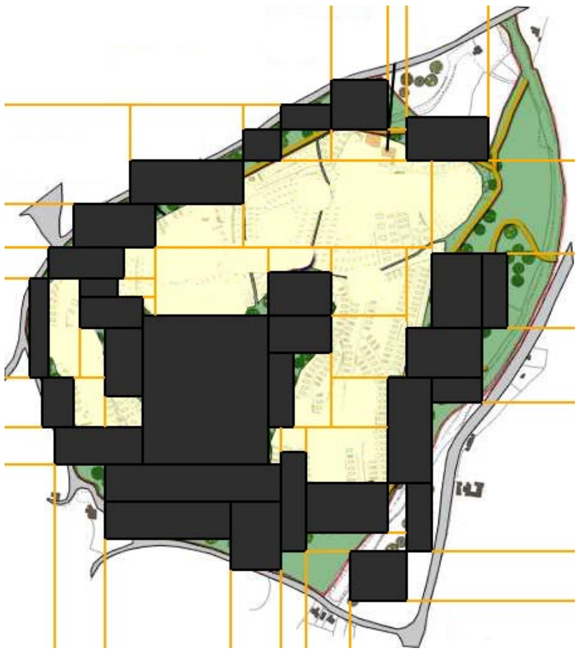
Figure 2: Spatial subdivision

Data Structure Construction Phase

Given a computer model, the data structures are constructed at a pre-processing phase, before any visibility computations are performed. The following sections discuss the underlying data structure construction phase.

2.1.1 Occluder Synthesis

This phase analyses the scene and synthesises suitable partial occluders. The input of this algorithm is a computer model, points in a plane are used as rendering primitives in this model problem. However, the extended version of these algorithms uses typical 3D computer models. The scene is spatially subdivided into cells, the set of cells that have desirable visibility properties, such as solidity value and hole information are identified and used as partial occluders. Figure 2 illustrates a typical set of partial occluders (the shaded rectangles) synthesised for the scene in Figure 1. One implementation of this phase uses a quad-tree data structure to recursively divide the scene into progressively smaller cells until a desired cell size or number of objects per cell is achieved. The contents of the cell are then projected onto its boundary, and a solidity value is calculated for the cell as the percentage of the boundary that is occupied by these projections.

2.1.2 Adjacency Graph

The scene is organised into a spatially subdividing data structure which is similar to the work by Teller [Tel92] where spatial cells are connected to neighbouring cells by portals to create a structure known as an adjacency graph. In our structure we distinguish between two different spatial cells, a cell is either a free cell (containing no useful partially occluding geometry), or a partially occluding cell (containing sufficient useful partially occluding geometry). An adjacency graph connects all neighbouring cells (Figure 3 shows the data structure, including cell connectivity information). This graph is computed once as a pre-process and allows efficient traversal of the subspaces of the scene during run-time. Using this traversal of the scene allows more efficient visibility approximations than the brute force approach presented in [Pea03], which required the partial occluders to be sorted in front to back order before visibility approximations we performed.

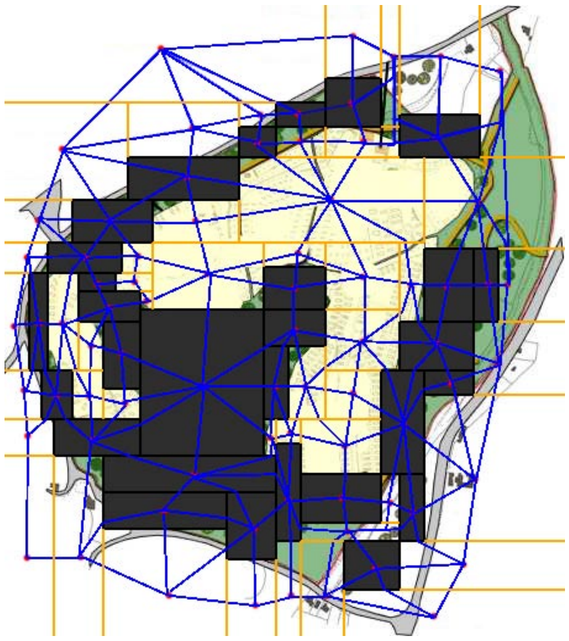
Figure 3: Spatial subdivision with adjacency graph

3. VISIBILITY COMPUTATION

The purpose of the previously described data structures is to drive visibility queries and eliminate the distance sort presented in [Pea03]. We have developed two methods of performing visibility approximations, a pre-computed and a run-time approach. The pre-computed approach minimises the amount of computational effort required during run-time at the expense of additional memory resources. The run-time approach has negligible memory requirements and performs visibility calculations during run-time. In this section we describe both approaches to approximating visibility information.

Run-Time Visibility Approximation

This approach approximates the visibility information for a given observer position. The visibility information is updated each time the observers' position is changed beyond a given threshold. We use a breadth first search of the adjacency graph data structure to encounter spatial cells in a front to back order. As a partially occluding cell is encountered, it is tested with the view frustum and a new shadow volume is calculated and added to the back of a queue of shadow volumes (EPO) if the partial occluder lies inside the view frustum. As a new *target* cell is encountered the set of shadow volumes is used to approximate the degree of visibility of the target as follows: the visibility of a target cell is calculated considering all shadowing volumes (considering the solidity of the shadow casting partial occluder). The percentage of the target cell that lies inside a shadow volume is multiplied by the solidity value of the partial occluder that casts the shadow, and the visibility of a target is calculated in this way considering all shadow volumes encountered on the traversal before reaching the target object.

```
ComputeVisRT(Queue EPO, view)
{
 while(!EPO.IsEmpty())
 {
  Cell C = EPO.front
  ApproximateVV(C)

  if(C.IsVisible())
  {
   for(all adjacent cells, adc)
   {
    if(adc.IsInside(view))
    {
     EPO.AddToBack(adc)
    }
   }
  }
  EPO.remove(C)
 }
}
```

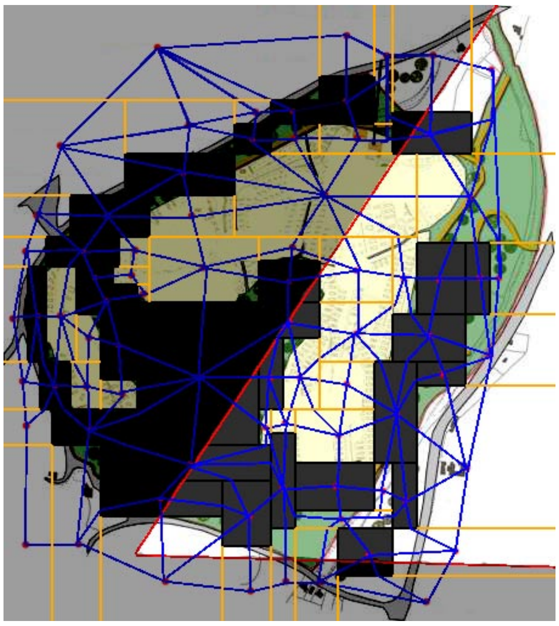
Figure 4: Run-time visibility information computation

Pre-Computed Visibility Approximation

This approach capitalises from a pre-processing phase to compute and store 'from region' visibility information. This phase stores visibility information with each cell of the spatial subdivision, which can be efficiently recalled during run-time. The visibility information for each cell is valid for any observer position inside the space it occupies. During the pre-processing phase, visibility queries are preformed from each of the free cells using the partial occluders to cast shadow volumes in the scene. The visibility value of the remaining cells in the scene is approximated using the set of shadow volumes which fully or partially contain each cell. The adjacency graph data structure is exploited for efficiency much in the same way as the run-time approach.

```
PreComputeVis(Queue EPO)
{
 while(!EPO.IsEmpty()) {
  Cell C = EPO.front
  ApproximateVV(C)

  if(C.IsVisible()) {
   for(all adjacent cells, adc) {
     EPO.AddToBack(adc)
   }
  }
  EPO.remove(C)
 }
}
```

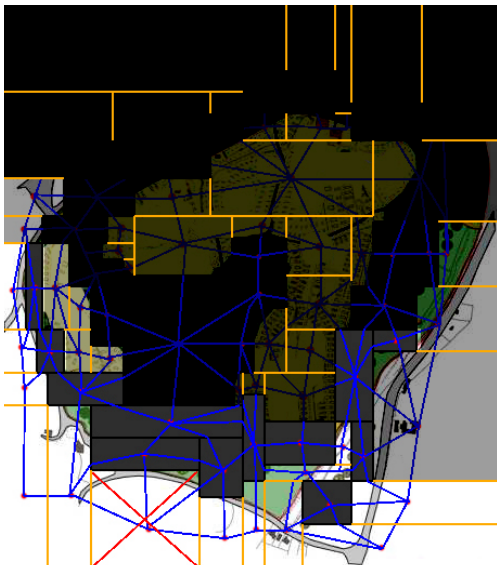

Figure 5: Pre-computed visibility information computation

4. RENDERING PHASE

Two approaches for calculating a visibility value for objects in a scene have been described. In [Pea03] we described how a visibility value can be used to speed up the rendering of a natural environment. This involves reducing the amount of rendering primitives sent to the graphics pipeline based on the visibility of each object. The objects that are completely visible are sent to the graphics hardware in full detail, while those that are less visible use a coarser representation (typically consisting of fewer rendering primitives). Additionally, objects which have a VV that is sufficiently low (and thus make an insignificant contribution to the final rendered image) are culled from the rendering process completely. Heuristic threshold values are used to determine when level of detail switches are performed and objects are culled from the rendering phase, adjusting these heuristics allows a trade off between image quality and rendering performance.

More specifically, before rendering the scene database, each object in the database is placed into a set that determines how it will be rendered based on the VV of the object from the observer's current location. The rendering of each object is determined by a number of heuristics: a number of LOD thresholds, and a cull threshold. The objects in a scene are classified into visibility sets as follows:

Visible Set, V

Contains the objects in the scene with a VV of 1. Which are completely visible, these are displayed at full resolution.

Partially Visible Set, PV_i

There can be several partially visible sets. Each partially visible set is associated with a LOD which is used to display the objects in the set. The LOD thresholds are used to classify objects into the appropriate partially visible set.

Cull Set, C

Objects in the cull set are not displayed during the display phase. Objects are assigned to the cull set when they have a VV that is lower than the cull threshold.

The objects in set V are displayed in full detail, the objects in the PV_i set are displayed at LOD i, and the objects in set C are not displayed at all. This reduces the number of polygons sent to the graphics hardware, improving the display rate of the application.

We currently have no scientific method of assigning the thresholds, they are used as adjustable heuristics to match users requirements. Assignment of these thresholds is a tradeoff between effective cull performance allowing increased rendering speeds (greater thresholds) and minimal visual artifacts (lower thresholds).

5. PRACTICAL APPLICATIONS

The partial visibility algorithms can be exploited in a number of different areas within virtual reality. These include:

Real-Time Complex Environment Rendering

The algorithms could be used to achieve output sensitive rendering in complex virtual environment models. Such as models representing forests, park lands, crop fields, and underwater, foggy, or crowded scenes.

Exploiting Partial Visibility to Assist in Scene Budgeting

The scene structure could be exploited to intelligently budget the system resources to achieve maximum quality or efficiently renderable scenes. When visibility information is pre-computed this allows the identification of exactly which rendering primitives are available from any single viewing position. This can be exploited to distribute the rendering budget amongst the visible portion of the scene.

Instancing Similar Partial Occluders To Save Memory

We could potentially use instanced simplifications of similar cells to save memory. Where partially occluded cells have similar visibility attributes, a single copy of the cell could be instanced throughout the scene to save memory requirements. Image error is a major concern during render time, where the instanced cell has differences from the original model.

Approximated Lighting Calculations

Partial Visibility algorithms could be exploited to achieve efficient approximate lighting and shadow calculations. The amount of light that illuminates areas of a model can be approximated in a similar way as that in [Ree85]. The difference in using the methods described in this paper is that we estimate the percentage of light that passes through areas to achieve a more accurate approximation.

Efficient Network Transmission

Partial Visibility algorithms could achieve an output sensitive network protocol. During the transmission of avatar (or object) positions in a multi-user virtual environment, avatar positions can be transmitted less frequently when they are located in less visible areas. Prediction techniques can be applied minimise any potential error.

6. RESULTS

The rendering process has been simulated to illustrate the potential savings when using these algorithms. A number of scenes were constructed with a varying number of rendering primitives. Partial occluders were synthesised in the scene containing a number of rendering primitives. During rendering, partial occluders which are fully visible are displayed using all rendering primitives, and those that are less visible are rendered with fewer primitives. Objects are culled when they are visible by a very small amount. Figure 6 illustrates the average number of rendering primitives displayed at a number of sample points inside the scene (plotted along the y-axis), in increasingly large scene sizes consisting of an increasing number of rendering primitives (plotted along the x-axis). Figure 7 illustrates the number of objects that are culled from the rendering process.

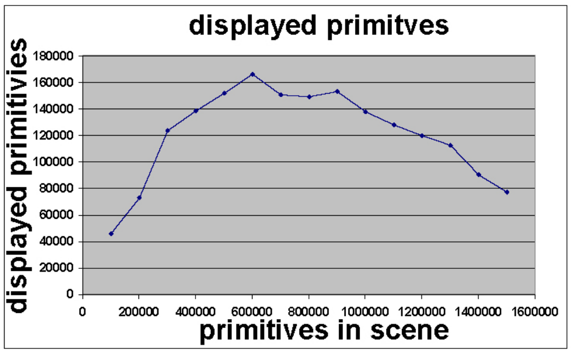

Figure 6: displayed graph

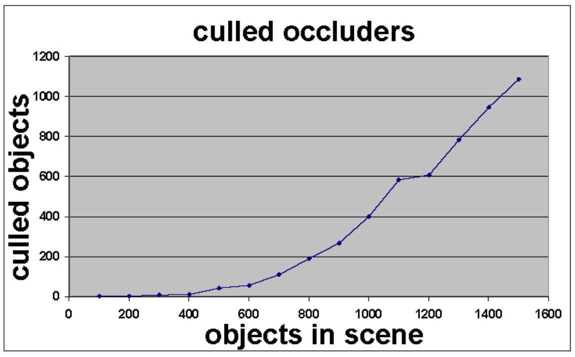

Figure 7: culled graph

Figures 8 and 9 illustrate these visibility approximations in a scene consisting of five hundred partial occluders to highlight the potential savings when using the partial visibility rendering technique. The observer is positioned at X, partial occluders are represented as boxes, with their shadow volumes extended from the box. The partial occluders with dark shading (close to X) are highly visible while the boxes with lighter shading are less visible, and the dark shaded objects that are distant from X are culled completely.

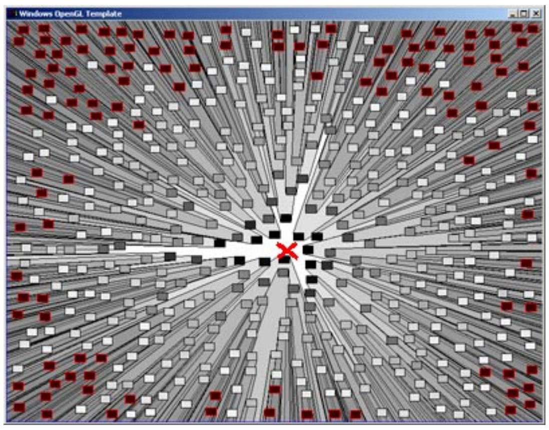

Figure 8: sample scene after visibility approximations

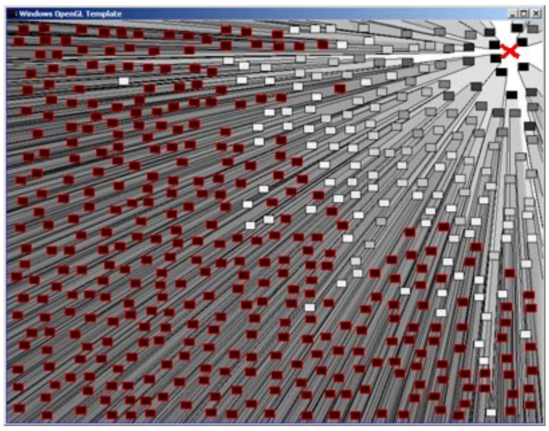

Figure 9: sample scene after visibility approximations

A walkthrough has been implemented which uses these techniques in 2.5 dimensions. Rendering speeds have been tested with this visibility approximation technique during a run-time walkthrough (with level of detail switching and culling based on visibility as previously described), this is compared to brute force rendering (where all objects are displayed in full detail). The scene consisted of 2,000,000 texture mapped quads. Average rendering times during a walkthrough of the scene are compared in Figure 10, and image quality can be compared in Figures 11 (full detail) and 12 (optimised).

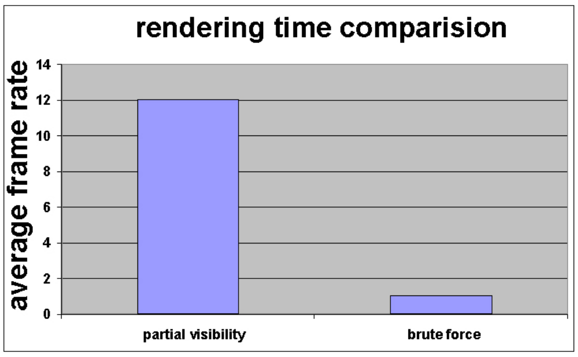

Figure 10: rendering times

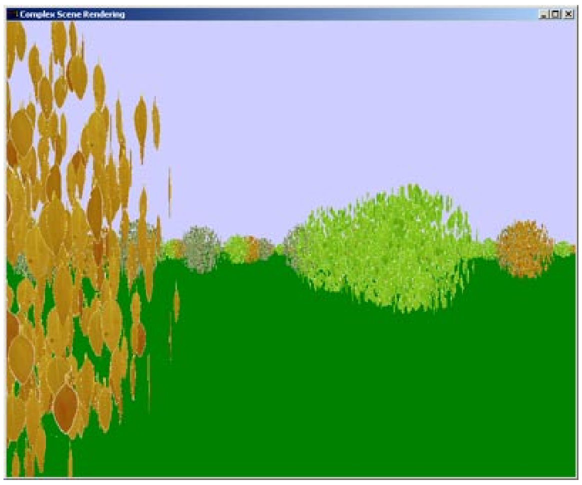

Figure 11: scene rendered using full detail

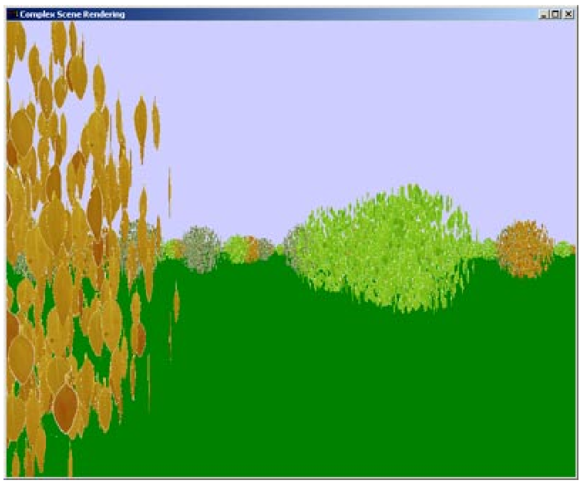

Figure 12: scene rendering using optimisations

7. CONCLUSIONS

In this paper we present the developing partial visibility algorithms, illustrated using a 2D model of the problem. We use a 2.5D implementation to perform experiments to highlight the savings during the rendering of some complex scenes.

The main contributions of this paper are advances to the brute force algorithms presented in [Pea03], where we take advantage of a data structure which allows efficient traversal of the scenes sub-spaces to perform visibility computations in an efficient manner, eliminating the need for a distance sort.

Another addition to the previous version of this algorithm is the introduction of the partial occluder synthesis phase. The previous publication on this work [Pea03] constructed a partial occluder for each tree in a natural scene. The introduction of the synthesis phase increases the potential for much larger (and fewer) partial occluders to be constructed, this improves the speed of the visibility computation phases.

8. FURTHER WORK

A full 3D version of the algorithms will be developed. Rendering times will be measured during walk-throughs of practical models and visual fidelity issues will be addressed.

Further optimisations to the algorithms are planned: we aim to reduce memory overheads, computation times, and improve the partial occluder synthesis phase. To improve computation time we plan to more intelligently synthesis partial occluders, much work must be performed in this area to identify the most effective way to subdivide the scene to identify suitable partial occluders. The spatial subdivision method also effects the memory requirements for storing the visibility information in the pre-computed approach, as each cell stores visibility information it is desirable to use a minimal number of cells in the scene, this cannot be done in a naive approach, as we do not wish for the visibility set to be too conservative. Thus we aim to achieve an optimal number of cells where neighbouring cells store minimal duplicate visibility information and each cell groups similar visibility properties so that the visibility set is not far different from all observer positions in the cell.

9. REFERENCES

[Air91] J, Airey. Increasing Update Rates in the Building Walk-through System with Automatic Model-Space Subdivision and Potentially Visible Set Calculations. PhD Thesis, University of North Carolina. 1991.

[And00] Andúlar, C., Saona-Vázquez, C., Navazo, I., and Brunet, P. Integrating Occlusion Culling and Levels of Detail Through Hardly-Visible Sets. Computer Graphics Forum. p. 499-506. 2000.

[App68] Appel, A. Some techniques for shading machine renderings of solids. AFIPS 1968 Spring Joint Computer Conf 32. p. 37-45. 1968.

[Ber00] Bernardini, F., Klosowski, J T., and El-Sana, J. Directional discretized occluders for accelerated occlusion culling. Computer Graphics Forum. 2000.

[Bru01] Brunet, P., Navazo, I., Rossignac, J., and Saona-Vázquez, C. Hoops: 3d curves as conservative occluders for cell visibility. Computer Graphics Forum 20. No. 3. 2001.

[Coh98a] Cohen-Or, D., Fibich, G., Halperin, D., and Zadicario, E., Conservative visibility and strong occlusion for viewspace partitioning of densely occluded scenes. Computer Graphics Forum. p. 243-254. 1998.

[Coh98b] Cohen-Or, D., and Zadicario, E. Visibility streaming for network-based walkthroughs. Graphics Interface '98. p. 1-7. 1998.

[Got99] Gotsman, C., Sudarsky, O., and Fayman, J. Optimized occlusion culling, Computing and Graphics. p. 645-654. 1999

[Kol00] Koltun, V. Chrysanthou, Y., and Cohen-Or, D. Virtual occluders: An efficient intermediate pvs representation. Rendering techniques 2000: 11th Eurographics Workshop on Rendering. p. 59-70. 2000.

[Law99] Law, F., Tan, T. Preprocessing Occlusion for Real-Time Selective Refinement. Symposium on Interactive 3D Graphics, ACM SIGGRAPH. p. 47-54. 1999.

[Nad99] Nadler, B., Fibish, G., Lev-Yehudi, S., and Cohen-Or, D. A qualitative and quantitative visibility analysis in urban scenes, Computing and Graphics 23. No. 5. p. 655-666. 1999.

[Pan99] Panne, M van de., and Stewart, J. Efficient compression techniques for precomputed visibility. Proceedings of Eurographics Workshop on Rendering. 1999.

[Pea03] Pearce, D., and Day, A M. Exploiting Partial Visibility Approximation in the Real-Time Display of Natural Environments. Proceedings of Theory and Practice of Computer Graphics. p. 66-73. 2003.

[Ree85] Reeves, W., and Blau, R. Approximate and Probabilistic Algorithms for Shading and Rendering Structured Particle Systems. SIGGRAPH '85. p. 313-322. 1985

[Sao99] Saona-Vazquez, C., Navazo., and Brunet, P. The visibility Octree: A datastructure for 3D navigation. Computing and Graphics. p. 635-644. 1999.

[Sut74] Sutherland, I E., Sproul, R F., and Schumaker, R A. A characterization of ten hidden surface algorithms. ACM Computing Surveys. p. 1-55. 6(1). 1974.

[Tel92] Teller, S J. Visibility Computations in Densely Occluded polyhedral Environments. University of California at Berkeley. 1992

[Won00] Wonka, P., Wimmer, M., Schmalstieg, D. Visibility Preprocessing with Occluder Fusion for Urban Walkthroughs. Technical Report, Institute of Computer Graphics, Vienna University of Technology. 2000.

[Zha98] Zhang, H. Effective Occlusion Culling for the Interactive Display of Arbitrary Models. Department of Computer Science. University of North Carolina at Chapel Hill. 1998.