Exploiting Temporal and Spatial Coherence in Hierarchical Visibility Algorithms

Jiří Bittner *†
Vlastimil Havran ‡§

Czech Technical University in Prague

Abstract

We present a series of simple improvements that make use of temporal and spatial coherence in the scope of hierarchical visibility algorithms. The *hierarchy updating* avoids visibility tests of certain interior nodes of the hierarchy. The *visibility propagation* algorithm reuses information about visibility of neighbouring spatial regions. Finally, the *conservative hierarchy updating* avoids visibility tests of the hierarchy nodes that are expected to remain visible. We evaluate the presented methods in the context of hierarchical visibility culling using *occlusion trees*.

Keywords: Coherence, hierarchical algorithms, visibility, kD–tree.

1 Introduction

Exploiting various types of coherence during image synthesis is one of the main goals of modern computer graphics. In the scope of visibility algorithms at least three types of coherence can be used: object space, image space, and temporal. Hierarchical visibility algorithms make some use of the spatial coherence inherently by utilising a spatial hierarchy. For densely occluded scenes they may achieve a great benefit by quickly identifying groups of invisible objects that need not be considered for rendering.

A typical hierarchical visibility algorithm uses a *visibility test*, that classifies a node of the spatial hierarchy as completely visible, partially visible or invisible depending on the visibility of the spatial region corresponding to that node. The visibility test is applied recursively starting at the root node. As soon as a node is found completely visible or invisible, the current branch of the traversal can be terminated, since visibility of all nodes in the current subtree is imposed by the visibility of the current node. In this paper we do not focus on the amount of image space or temporal coherence, that may be exploited by the visibility test itself. Instead we suggest a more general framework that is rather independent of the particular visibility algorithm.

Traditional hierarchical visibility algorithms traverse the spatial hierarchy starting at the root node. Firstly, we propose a method, that saves up to half of the visibility tests by skipping certain interior nodes of the hierarchy (assuming the spatial hierarchy corresponds to a binary tree). The skipping is guided by visibility classifications obtained during the previous invokation of the visibility algorithm. Secondly, we describe an algorithm that increases the amount of spatial coherence exploited. It reuses visibility classifications of hierarchy nodes already processed in the current pass of the algorithm. The nodes are processed in front–to–back order and the algorithm tries to determine visibility of the region corresponding to the current node by combining visibility states of neighbouring regions. If it fails, the usual visibility test is applied. Finally, we propose a conservative method, that aims to avoid repeated visibility tests of nodes that probably remain visible.

2 Related Work

Some visibility algorithms exploit temporal coherence in a specialised way, that reflects the principles of each such algorithm. Greene et al. [5] uses the set of visible objects from one frame to initialise the *z–pyramid* in the next frame and so reduces "overdraw" of the *hierarchical z–buffer*. Coorg and Teller [2] present a visibility algorithm that uses *relevant planes* which form a subset of visual events. They restrict the hierarchy traversal to nodes corresponding to planes that were crossed between successive viewpoint positions. Another method [3] of Coorg and Teller exploits temporal coherence by caching occlusion relationships.

Chrysanthou and Slater have proposed a probabilistic scheme for view–frustum culling [11]. They partition objects into groups, which are sampled according to their distance from the view–frustum. It is difficult to generalise this method for visibility algorithms, since the "visible volumes" can be very complex, and usually they are not explicitly reconstructed. Moreover, this method is not conservative unless changes in viewing direction and position of the viewpoint are restricted.

The methods proposed in this paper can be used to make use of temporal and spatial coherence in the scope of ex-

* bittner@fel.cvut.cz
† Centre of Applied Cybernetics
‡ havran@fel.cvut.cz
§ Department of Computer Science and Engineering

isting visibility algorithms, that utilise a spatial hierarchy. Examples of these are algorithms based on hierarchical occlusion maps [12], coverage masks [6], shadow frusta [8], and occlusion trees [1].

3 Overview

In order to describe modifications of the visibility algorithm we first restrict our discussion to one particular approach – the *conservative hierarchical visibility culling*. Below, we give a short overview of the data structures and algorithms that are used in the scope of the proposed methods.

3.1 Spatial Hierarchy

The hierarchical visibility culling utilises a spatial hierarchy, that is built over all objects of the scene. We have focused on kD–trees [10] because of their high flexibility and simplicity of building and traversal. A node of the kD–tree corresponds to an axis–aligned bounding box. Each leaf of the tree contains a list of references to objects that intersect the corresponding box.

3.2 The Node Visibility Test

The elementary step of the hierarchical visibility culling is the *node visibility test*, i.e., visibility classification of a single node of the hierarchy using certain occlusion map. We assume that given a viewpoint and a viewing direction the visibility algorithm classifies visibility of the node as *completely visible*, *partially visible* or *invisible*. Although in this paper we do not focus on the visibility determination step itself, we give a brief description of one such algorithm (see [1] for further details).

For each position of the viewpoint several large polygonal occluders are identified. These are used to build an *occlusion tree*, that results from merging "shadow" frusta of each individual occluder. Briefly, the occlusion tree is a Binary Space Partitioning (BSP) tree [4], that has its leaves classified as *in* or *out*, if they are occluded or unoccluded, respectively. The node visibility test is performed using constrained depth first search (DFS) on the occlusion tree. The final visibility classification is obtained by hierarchical combination of visibility states of nodes reached by the DFS.

4 Classical Approach

The classical hierarchical visibility culling proceeds as follows: Starting from the root node of the hierarchy, the view–frustum culling is applied on the current node [9]. If the node is outside the view–frustum it is classified invisible. Otherwise, the node visibility test is performed. If the node is found visible all its descendants are visible. Similarly, if a node is invisible all its children are invisible. Descendants of nodes classified as partially visible are tested further to refine their visibility (see Figure 1). When the visibility of all leaves is known, objects from fully visible and partially visible leaves can be gathered and rendered using a low–level exact visibility solver (such as depth–buffer).

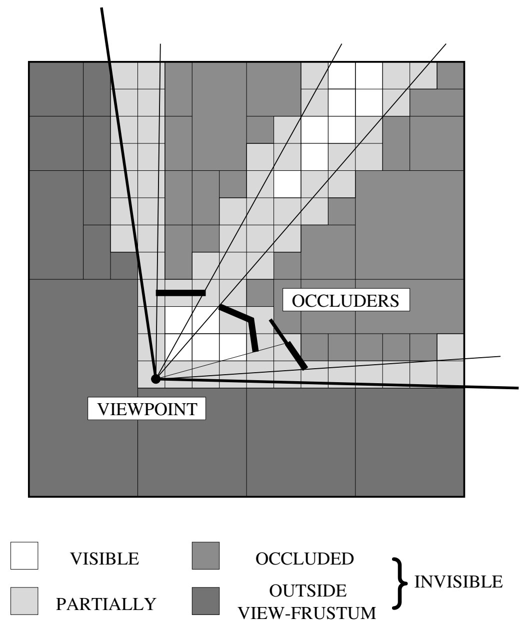


Figure 1: An example of the hierarchical visibility culling. The node visibility test uses merged occlusion volumes of four occluders.

A simple improvement can be used to avoid visibility tests of hierarchy nodes that contain only few objects and so the estimated cost of rendering the objects is lower than the cost of the visibility determination. In such a case the node can be simply classified as visible.

4.1 Modifications Overview

In order to give an overview of the proposed modifications we first show how they are exploited in the scope of the hierarchical visibility algorithm (see Figure 2). The *hierarchy updating* test is applied first. This test eventually decides to skip all the remaining steps and to continue determining visibility of descendants of the current node. The *view–frustum* culling can report the node as invisible if it is outside the view–frustum. Otherwise, the *visibility propagation* is applied, that can succeed classifying the node as visible or invisible. The *conservative hierarchy updating* classifies some nodes as visible with certain probability. If all previous steps failed in determining node's visibility, the node visibility test is applied. Note, that the steps are applied in order of increasing computational cost, what reflects the main idea of culling: use more complicated test only when the simple one fails to find a solution.

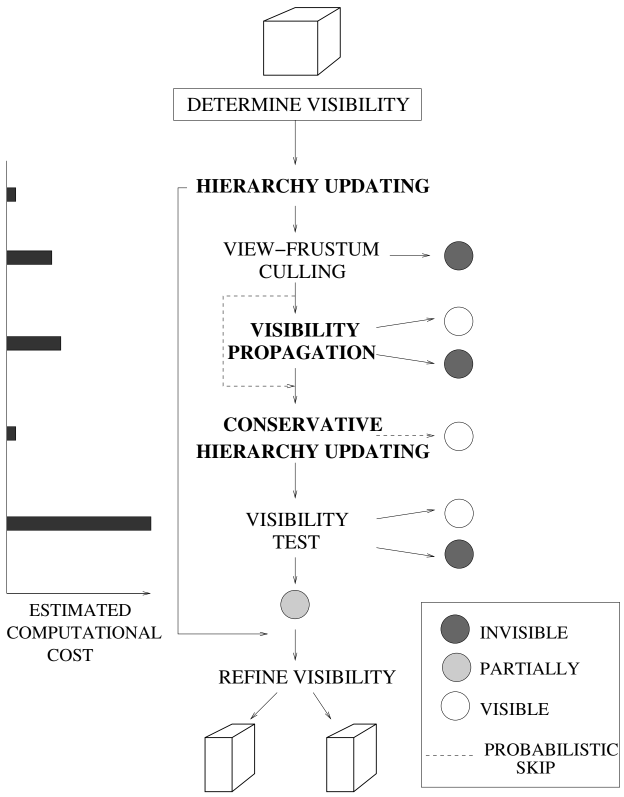


Figure 2: Series of steps determining visibility of a node of the hierarchy. The novel methods are highlighted.

The rest of the paper is organised as follows: In Section 5 the hierarchy updating method is introduced. The visibility propagation is presented in Section 6. In Section 7 a conservative modification of the hierarchy updating algorithm is outlined. Results and comparisons are presented in Section 8. Finally, Section 9 discusses some topics for future work and concludes.

5 Hierarchy Updating

The hierarchical visibility algorithm can be seen as a traversal of the hierarchy, that is terminated either at leaves or nodes classified either as visible or invisible. Let us call such nodes the *termination nodes* and nodes that have been classified partially visible the *opened nodes*. Denote sets of termination and opened nodes in the i-th frame $\mathcal{T}_i$ and $\mathcal{O}_i$, respectively. In the classical approach $\mathcal{T}_i \cup \mathcal{O}_i = \mathcal{V}_i$, where $\mathcal{V}_i$ is the set of all nodes visited in the i-th rendering frame.

Imagine the viewpoint is fixed. Visibility of all nodes of the hierarchy does not change and the sets $\mathcal{T}_i$, $\mathcal{O}_i$, and $\mathcal{V}_i$ are fixed as well. Nevertheless, the classical algorithm repeatedly tests visibility of all nodes $\mathcal{V}_i$.

The hierarchy updating is a modification that aims to eliminate the repeated visibility tests of the set of opened nodes from the previous frame. It skips all nodes of $\mathcal{O}_{i-1}$ and applies node visibility tests only on nodes of $\mathcal{T}_{i-1}$. In order to propagate eventual changes in visibility up into the hierarchy the visibility states determined at the termination nodes are pulled up according to the following rule: The visibility state of the node is updated as visible or invisible, if all its children have been classified as visible or invisible, respectively. Otherwise, it remains partially visible and thus opened. The pseudo–code of the hierarchical visibility algorithm with hierarchy updating is outlined in Figure 3.

```
Algorithm HierarchicalVisibility( NODE )
begin
  if NODE is leaf or NODE.visibility ≠ PARTIALLY
    (* termination nodes *)
    or NODE.frame < frame-1 then
  begin
    NODE.visibility ← TestVisibility( NODE );
    NODE.frame ← frame;
  end
  case NODE.visibility of
    VISIBLE   : Render subtree of NODE;
    PARTIALLY :
      if NODE is leaf then Render NODE;
      else
        for all children C of NODE do
          HierarchicalVisibility( C );
      (* pull-up *)
      if visibility of all children equals v then
        begin
          NODE.visibility ← v;
          NODE.frame ← frame;
        end
    INVISIBLE : (* terminate the DFS *)
  end
end
```

Figure 3: Pseudo–code of the hierarchical visibility culling with hierarchy updating. Note, that the set of termination nodes is not maintained explicitly. Instead, each node contains its previous visibility classification. The *frame* variable asociated with a node is used to identify nodes "below" the current termination nodes.

Consequently, the modification does not change the final visibility classification, that is the same as the one obtained using the classical approach. The behaviour of the modified hierarchical visibility algorithm is illustrated in Figure 4. Note, that if the pull up did not take place the algorithm could end up with the termination nodes being all leaves of the hierarchy. Hence, it would loose advantages of the hierarchical algorithm.

For kD–trees $|\mathcal{O}_i| = |\mathcal{T}_i| - 1$. Thus the hierarchy updating can save almost a half of the visibility tests, that would be applied on the interior nodes of the hierarchy.

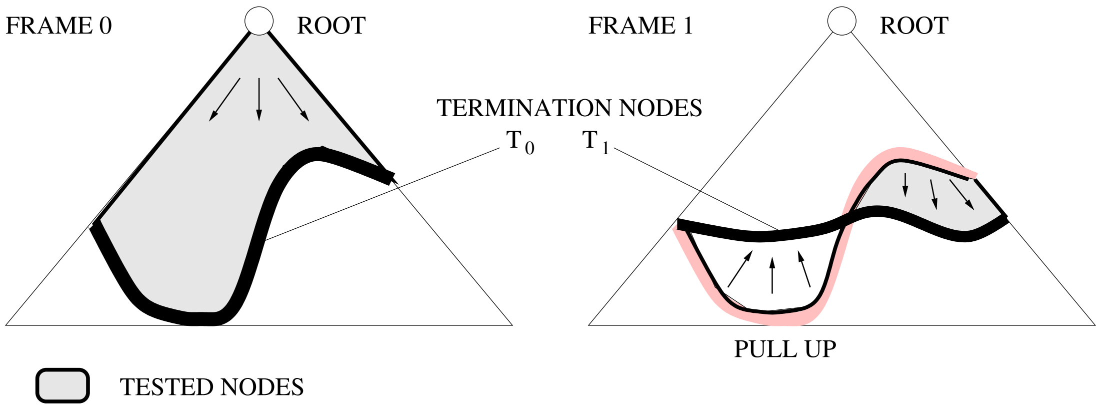


Figure 4: Illustration of the hierarchy updating. Initially the algorithm proceeds starting at the root of the hierarchy (left). In the second frame the opened nodes $\mathcal{O}_0$ are skipped and the visibility tests are applied on the termination nodes $\mathcal{T}_0$ (and eventually "below"). Visibility changes are propagated up to the hierarchy and the new set of termination nodes $\mathcal{T}_1$ is established.

6 Visibility Propagation

The hierarchical visibility culling already makes use of spatial coherence by utilising a spatial hierarchy (kD–tree). However, we can further increase the amount of coherence exploited by reusing visibility information computed for neighbouring regions.

Suppose that the nodes of the spatial hierarchy are processed in front–to–back order with respect to the viewpoint. Using kD–tree this ordering is determined in a simple way [4]. First, we try to determine visibility of the currently processed node by combining visibility classifications of its relevant neighbours. If the combination fails we revert to the node visibility test.

Let us denote the box corresponding to node N as B_N. The visibility of N can be determined combining visibility of potentially visible faces $\mathcal{F}_{B_N}$ of B_N ($|\mathcal{F}_{B_N}| \leq 3$). Consequently, visibility of a face $F \in \mathcal{F}_{B_N}$ can be determined combining visibility of appropriate *neighbour nodes*. If all faces of $\mathcal{F}_{B_N}$ are invisible the node N is invisible. Similarly, if all faces of $\mathcal{F}_{B_N}$ are visible and there is no occluder intersecting B_N, N can be classified as completely visible. Otherwise, the visibility propagation fails and the usual node visibility test must be applied. An example of a node that can be classified as invisible is depicted in Figure 5.

A neighbour node of N on a face F is a node U of the kD–tree with B_U laying in the opposite halfspace (induced by F) than B_N and having non–empty intersection with F. Naturally, we could keep a list of neighbour nodes for each face. Instead, we have used neighbour links (ropes) for kD–trees [7] that have low memory requirements and allow hierarchical visibility propagation.

Within each face F we associate a link to a neighbour node U that has a smallest box containing the face completely ($F \cap B_U = F$). When determining visibility of a face F there are three possible cases:

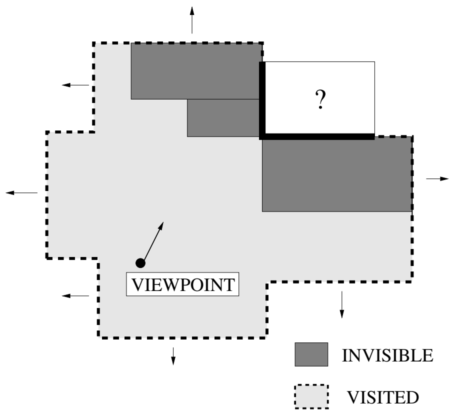


Figure 5: An example of a node that can be classified invisible since all its appropriate neighbours are invisible.

1. the link points to a node that is visible/invisible,
2. the link points to a node that is partially visible,
3. the link points to a node that has not been visited in the current frame.

The first case is trivial; the visibility of the face can be set immediately. In the second case we perform a constrained DFS and combine visibility of reached nodes. The search is constrained to nodes having non–empty intersection with the face F and terminates at the termination nodes $\mathcal{T}_i$. This process is illustrated in Figure 6. The visibility combination is performed using the same rule as in the pull up pass of the hierarchy updating (Section 5). Nevertheless, we can terminate the DFS whenever the combination results in partial visibility.

The third case is solved as follows: If the link is pointing to a node that has not been visited in the current frame, there must be some termination node on the path to the root. This path is followed until the termination node is

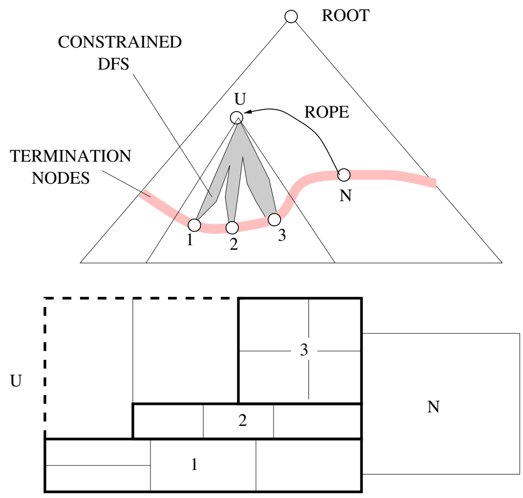


Figure 6: Illustration of the hierarchical visibility propagation using ropes.

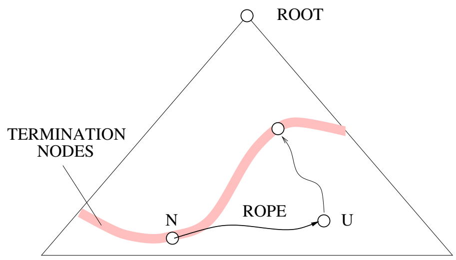


Figure 7: Lazy propagation of the visibility classification. Node U was not visited in the current frame. The algorithm must follow the path to the root to determine visibility of U.

reached (see Figure 7). Note, that if the visibility states were propagated into subtrees of the termination nodes, the third case would never occur.

6.1 Temporal coherence

The visibility propagation does not always succeed to determine visibility of the processed node. In such a case it introduces an additional overhead into the visibility determination. However, we can use information obtained in the previous frame to guide the algorithm in the current frame.

Firstly, we avoid visibility propagation on nodes that we expect to remain partially visible and thus the visibility propagation would probably fail. To achieve this we apply the visibility propagation only on nodes that have not been classified as partially visible in the previous frame. Secondly, if for a given node the visibility propagation succeeded in the previous frame, it is applied in the current frame as well. Otherwise, it is applied with certain probability $p_{vp} < 1$.

7 Conservative Hierarchy Updating

The hierarchy updating method ensures that on each path to a leaf node of the hierarchy at least one node is tested for visibility. We can further reduce the expected number of node visibility tests at the cost of the conservative behaviour of the modified algorithm. The conservative hierarchy updating produces a superset of visible nodes determined by the previously described algorithms (although they are generally conservative as well, depending on the properties of the node visibility test).

Due to the complexity of the occlusion volume it is difficult to predict changes in visibility unless a specialised visibility algorithm is involved [2]. To keep the conservative behaviour of the algorithm we cannot classify a node as invisible without really determining its visibility. Nevertheless, assuming visibility does not change significantly over successive frames, visibility states of visible and partially visible nodes do not have to be updated in each frame. Instead, we skip the visibility determination of these nodes with certain specified probability p_{skip} and mark them as visible. With $1 - p_{skip}$ probability the algorithm updates visibility of the node invoking the node visibility test.

8 Results and Discussion

The algorithms mentioned in the paper were tested using a walk through the model of the fifth floor of the Soda–Hall [1]. The measurements were conducted using SGI O2 workstation with 64MB memory. The constructed kD–tree consisted of 1187 nodes. In all measurements we used the visibility culling algorithm based on occlusion trees [1]. For each position of the viewpoint 16 occluders were identified and used to build the occlusion tree. In the scope of one walk the path depicted in Figure 10 was followed. If not stated differently all presented values are averaged per one frame of the walkthrough. The following methods were evaluated:

A — the classical approach,

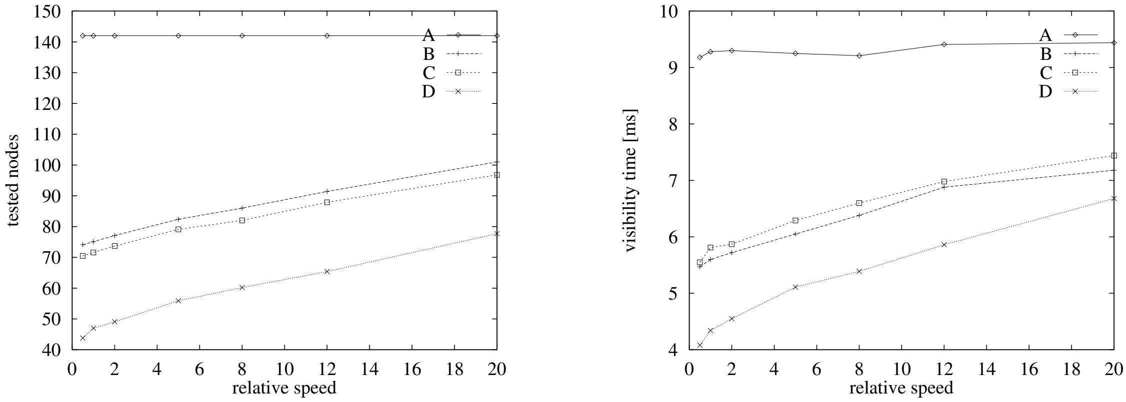
B — hierarchy updating applied,

C — hierarchy updating + visibility propagation with probability $p_{vp} = 0.5$,

D — as **C** + conservative hierarchy updating with probability $p_{skip} = 0.5$.

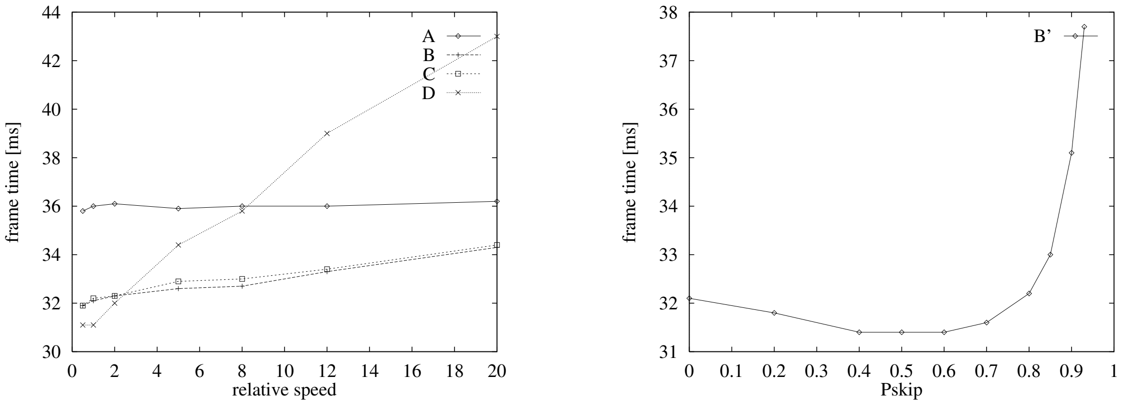
The first three plots illustrate the dependence of the algorithms on the relative speed of the walk (Figures 8–a, 8–b, and 9–a). A unit relative speed roughly corresponds to the usual walking speed. We have measured the number of node visibility tests, the time spent by the hierarchical visibility determination, and the total frame time.

[1] http://graphics.lcs.mit.edu/~becca/research/SodaHall



(a) (b)

Figure 8: (a) Dependence of the number of node visibility tests on the relative speed of the walk. (b) Average time spent by the hierarchical visibility algorithm.



(a) (b)

Figure 9: (a) Average frame time depending on the relative speed of the walk. (b) Dependence of the average frame time on the probability p_{skip} using the conservative hierarchy updating.

All evaluated methods exhibit a very slow growth of the number of necessary node visibility tests. For a walk of relative speed 1.0 the following savings in number of node visibility tests were achieved (compared to **A**): method **B** – 47%, method **C** – 50%, and method **D** – 67%. The hierarchy updating (method **B**) saves almost half of the node visibility tests as expected. We have observed that the visibility propagation (method **C**) succeeds in determining visibility of only few nodes that usually correspond to rather large regions. The **D** method significantly decreases the number of node visibility tests. This is paid by a higher number of nodes classified as partially visible or visible (details follow further in the text).

Figure 8-b shows that the time spent by the hierarchical visibility culling was roughly proportional to the number of node visibility tests. Nevertheless, we can observe that the time spent by the visibility propagation (method **C**) is not recovered by the savings in number of node visibility tests. In particular, this follows from the fact that the node visibility test using the occlusion tree is almost as fast as the visibility propagation.

In Figure 9-a we can observe the conservative behaviour of method **D**. When the viewpoint moves slowly, the method achieves better frame times than the other ones. As the relative speed of the walk increases the visibility states of many nodes change quickly. Hence "reusing" some previously visible nodes leads to a larger set of nodes to render and the frame time is increased.

Finally, the behaviour of the conservative hierarchy updating algorithm in dependence on the probability p_{skip} was measured (Figure 9-b). We can observe a local minimum in the average frame time at $p_{skip} = 0.5$. For probabilities greater than this minimum savings in visibility classification do not recover the time necessary for rendering otherwise invisible objects.

It is worth mentioning that our aim was not to evaluate the visibility algorithm itself, but rather to document the impact of the proposed methods. It is obvious that if the visibility algorithm was more demanding, the proposed methods would decrease the total frame time more significantly.

9 Conclusion and Future Work

In this paper we have introduced a series of modifications of the classical hierarchical visibility culling. The hierarchy updating proved to perform well in practice as it saves almost half of the visibility tests that would have to be applied using the classical approach. The savings would be less remarkable for hierarchies with higher branching factors, but our preliminary results indicate that kD–trees with arbitratry positioned partitioning planes are much more effective for visibility culling that octrees or bounding volume hierarchies.

Suprisingly, we have observed that the visibility propagation saves only few visibility tests. This documents that the spatial coherence is already exploited well in the classical approach. Finally, we have shown that the conservative hierarchy updating can improve the overall frame time for certain settings.

We have experimented with fixed probabilities used in both the conservative hierarchy updating and the probabilistic modification of the visibility propagation algorithm. Definitely, more elaborate methods should be used. For example an average "survival" time of visible nodes could be predicted and used in a sampling scheme ensuring that the visibility of a node is correctly updated within certain number of frames (such as Russian roulette). This approach would eventually adapt to properties of environment surrounding the viewpoint and the efficiency of the visibility classification algorithm. If the visibility algorithm was unsuccessful most regions would be visible and their survival time would increase. These nodes would be tested with lower probability hence the wasted effort for their visibility classification would be reduced.

Currently the visibility propagation method determines only if a node is visible or invisible. Nevertheless, it could be extended to determine that a node is partially visible with high probability and hence to avoid the visibility test even on nodes expected to be partially visible. This modification would benefit in sparsely occluded environments where many small regions (leaves of the hierarchy) are classified as partially visible.

Nowadays, the speed of the rendering subsystem is commonly the bottleneck of the total rendering time. Nevertheless, as specialize multiprocessor rendering architectures appear it is important to optimize all components of the rendering process. The proposed methods aim to make a step further in this direction.

Acknowledgements

This research was supported by the Czech Ministry of Education under Project LN00B096 and the Aktion Kontakt OE/CZ grant number 1999/17.

References

[1] J. Bittner, V. Havran, and P. Slavik. Hierarchical visibility culling with occlusion trees. In *Proceedings of Computer Graphics International '98 (CGI'98)*, pages 207–219, Hannover, Germany, June 1998. IEEE, New York. Also available as http://www.cgg.cvut.cz/~bittner/paper-copy.ps.gz.

[2] Satyan Coorg and Seth Teller. Temporally coherent conservative visibility. In *Proceedings of the Twelfth Annual ACM Symposium on Computational Geometry*, Philadelphia, PA, May 1996.

[3] Satyan Coorg and Seth Teller. Real-time occlusion culling for models with large occluders. In *Proceedings of the Symposium on Interactive 3D Graphics*, pages 83–90, New York, April27–30 1997. ACM Press.

[4] Henry Fuchs, Zvi M. Kedem, and Bruce F. Naylor. On visible surface generation by a priori tree structures. In *Proceedings of SIGGRAPH '80*, pages 124–133, July 1980.

[5] N. Greene, M. Kass, and G. Miller. Hierarchical Z-buffer visibility. In *Proceedings of SIGGRAPH '93*, pages 231–238. ACM Press, August 1993.

[6] Ned Greene. Hierarchical polygon tiling with coverage masks. In *Proceedings of SIGGRAPH '96*, pages 65–74, August 1996.

[7] V. Havran, J. Bittner, and J. Žára. Ray tracing with rope trees. In *Proceedings of 13th Spring Conference on Computer Graphics*, pages 130–139, Budmeřice, 1998. Also available as http://www.cgg.cvut.cz/~bittner/sccg98.ps.gz.

[8] T. Hudson, D. Manocha, J.Cohen, M.Lin, K.Hoff, and H.Zhang. Accelerated occlusion culling using shadow frusta. In *Proceedings of the Thirteenth ACM Symposium on Computational Geometry, June 1997, Nice, France*, 1997.

[9] John Rohlf and James Helman. IRIS performer: A high performance multiprocessing toolkit for real–Time 3D graphics. In *Proceedings of SIGGRAPH '94*, pages 381–395, July 1994.

[10] H. J. Samet. *Design and analysis of Spatial Data Structures: Quadtrees, Octrees, and other Hierarchical Methods*. Addison–Wesley, Redding, MA, 1989.

[11] M. Slater and Y. Chrysanthou. View volume culling using a probabilistic caching scheme. In *Proceedings of the ACM Symposium on Virtual Reality Software and Technology (VRST-97)*, pages 71–78, New York, September15–17 1997. ACM Press.

[12] Hansong Zhang, Dinesh Manocha, Thomas Hudson, and Kenneth E. Hoff III. Visibility culling using hierarchical occlusion maps. In *SIGGRAPH 97 Conference Proceedings*, Annual Conference Series, pages 77–88. ACM SIGGRAPH, August 1997.

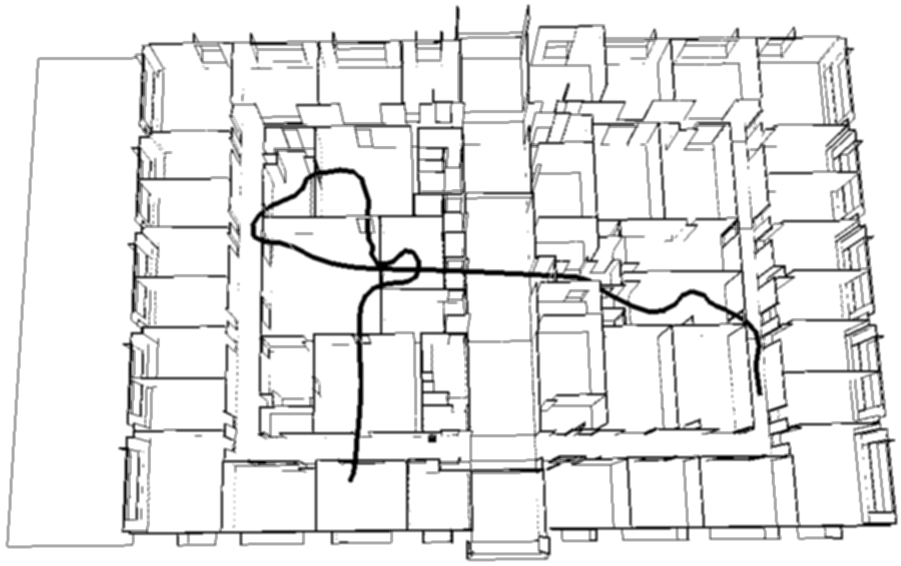


Figure 10: The path used for a walk through the model of the Soda–Hall. For relative speed of the walk equal to 1.0 the walk consists of 980 steps.

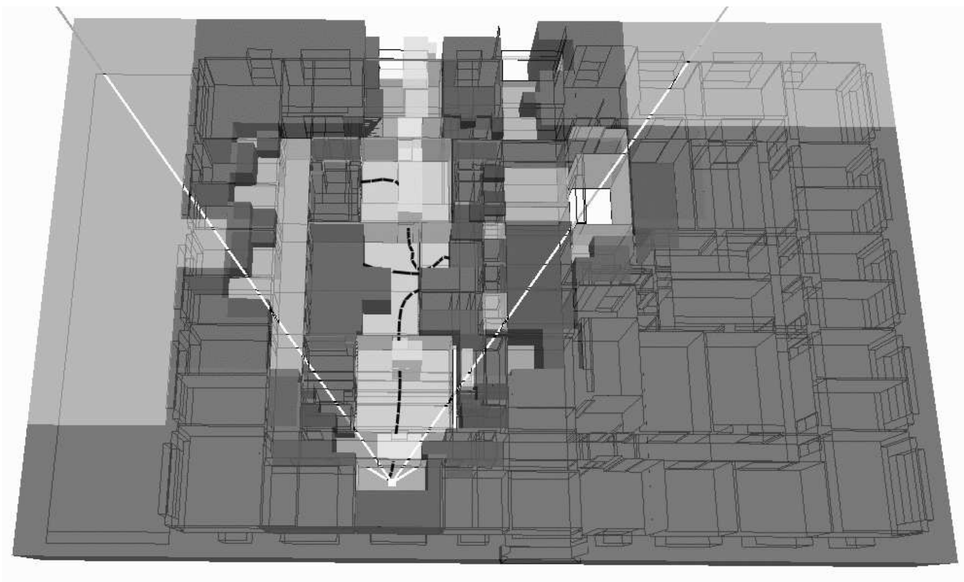


Figure 11: An example of the hierarchical visibility culling. The largest gray regions are outside of the view–frustum. Few lighter regions in the viewing direction are completely visible. Invisible regions are shown in dark gray. Lighter gray regions were found invisible by the visibility propagation algorithm. Partially visible regions are transparent.