Variations of Knapsack Problem Relevant to Interactive Rendering

Ali Lakhia
lakhia@eecs.berkeley.edu

Department of Electrical Engineering
and Computer Sciences
University of California at Berkeley

April 27, 2004

Abstract

We explain how the Knapsack problem and interactive rendering are related. We also discuss several additional concerns regarding interactive rendering and how these affect the solution to the original Knapsack problem. Lastly, we present two approximate algorithms and compare how they meet the needs of interactive rendering.

1 Background

Knapsack Problem

We are given a set of n items, $S = x_1, x_2, ..., x_n$. Each item has a weight, $w(x_1), w(x_2), ..., w(x_n)$ and value, $v(x_1), v(x_2), ..., v(x_n)$. The Knapsack problem is to find S', a subset of S, such that:

$$\sum_{y \in S'} v(y) \text{ is maximized over } S'$$

and

$$\sum_{y \in S'} w(y) \leq c$$

where c is the capacity of the knapsack.

This problem is NP-Complete [8] because a polynomial time checker can verify a solution but the best algorithm for determining the solution is $\mathcal{O}(n2^k)$, where k is the number of bits used to represent the capacity of the knapsack and the weights of each item [4].

Interactive Rendering

Rendering is the process of creating a 2-dimensional image from a 3-dimensional scene consisting of objects, lights, cameras and other entities. Interactive rendering is doing this conversion fast enough that a user can provide input and get feedback almost instantaneously. Interactive rendering is useful in different contexts such as architectural evaluation of buildings, CAD applications, game playing and simulations.

Interactive rendering is made possible by advances in graphics card hardware. However, the time required to render a scene is input-dependent. That is, as more objects are added to the scene, the render time increases even if the final 2D image is of the same size. This makes interactive rendering of large scenes challenging.

2 Similarities and Variations

The challenges of interactive rendering are similar to the Knapsack problem. The knapsack is analogous to the target render time or the time available for one frame. Each object in the scene is analogous to an item. The value of object is approximated[1] by measuring the screen space that it occupies. The weight of an object is the amount of time needed to render it. This is roughly proportional to the number of polygons in the object.

There are optimization techniques and other requirements for interactive rendering that modify this original Knapsack problem. We will describe these next.

Levels of Detail

To improve interactivity, levels of detail (LODs) were introduced by Clark in 1974 [2]. As seen in Fig. 1 (a) and (b), an object is simplified by various degrees. More detailed LODs are chosen when the detail can be observed. Otherwise, a lower detailed LOD is chosen.

Interactive rendering with LODs is known as the Multiple Choice Knapsack Problem (MCKP). That is, we are provided with m candidate sets, $s_1, s_2, ..., s_m$ where all sets are disjoint:

$$s_i \cap s_j = \emptyset \qquad \forall i, j \text{ where } i \neq j$$

Here, each set, $s_{1..m}$ contains one or more items $x_{1..n}$ and $m \leq n$:

$$\bigcup_{i=1}^{m} s_i = S$$

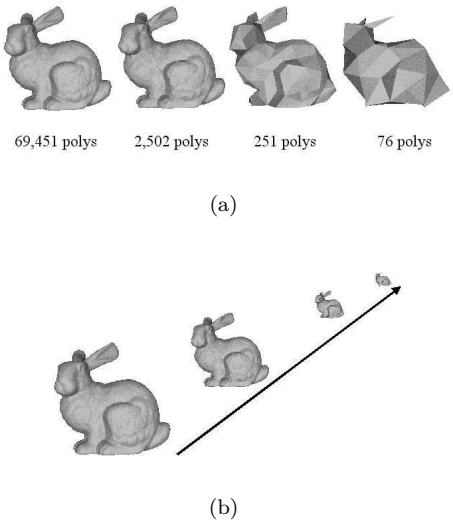


Figure 1: (a) The rabbit is simplified at different levels to create the LODs. (b) We switch from one LOD to a different one based on distance.

The set S' has an additional constraint that only one item from each candidate set must be picked:

$$x_i \in s_k \text{ and } x_j \in s_l \quad \Rightarrow \quad s_k \neq s_l \qquad \forall i, j \text{ where } i \neq j$$

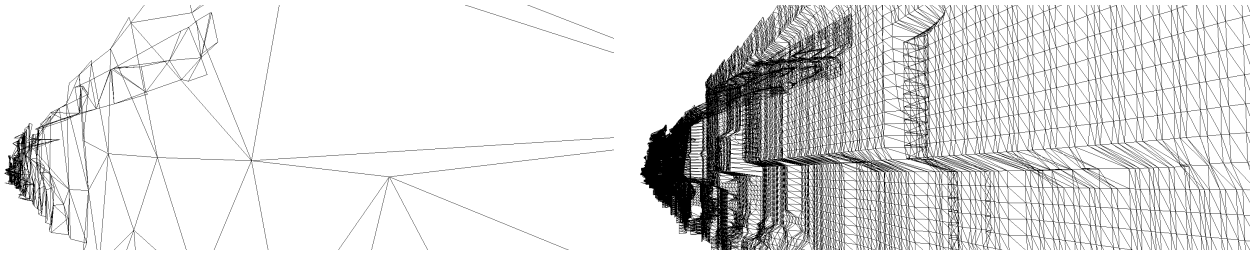
It is easy to see that the MCKP is a generalization of the Knapsack problem. For example, given the original Knapsack problem, place each item, x_i into a set s_i with no other elements in that set. The MCKP solution will yield a valid Knapsack problem solution.

Hierarchical Levels of Detail

LODs of large and detailed objects are not optimal. For example, consider a slanted view of a building facade in Fig. 2 (a). Note that a coarse representation of the object is ideal for the portion of the object that is far away from the camera but is too coarse near the camera. Similarly, a highly detailed LOD provides good detail near the camera but wastes too many triangles for detail that is not perceptible from that position.

One possibility to overcome this is to break large objects into smaller pieces. However, smaller pieces restrict simplification locally to that piece and yield substantially sub-optimal LODs at the coarsest levels. The use of a hierarchy of LODs, or HLODs, was proposed to overcome suboptimal use of LODs [5].

[1] Ideally, value should measure semantic and contextual information of the object as well.



(a) (b)

Figure 2: (a) LOD of large object is too coarse near the camera; (b) Using a higher detailed LOD wastes detail that is far away from the camera.

HLODs of an object are created by segmenting it into pieces that are further broken up into smaller pieces as more detail is added. This yields a hierarchy as shown in Fig. 3. For example, at the coarsest level, we could have a car that is represented by a very simple box. In the next level, we can break up the car into a front, middle and back section. Here, each HLOD node has more detail. And so on for subsequent levels.

The problem of selecting HLOD nodes is analogous to a front[2] selection. This problem is referred to as Hierarchical MCKP and is similar to MCKP except that candidate sets may contain replacement sets. A replacement set requires that either none or all of the elements in it are selected. Furthermore, replacement sets may contain candidate sets that may contain replacement sets, and so on.

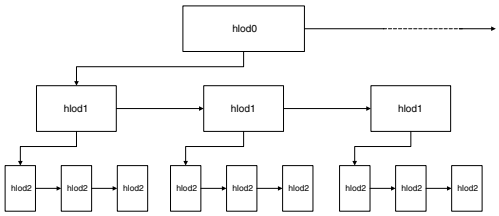


Figure 3: HLOD of an object. The node $hlod_0$ is most coarse whereas $hlod_2$ is the most detailed.

Note that Hierarchical MCKP is a generalization of MCKP. For example, if each node in the HLOD hierarchy has only one child, then the problem is reduced to the MCKP. That is, if all replacement sets contain exactly one item, then the solution to the Hierarchical MCKP solves MCKP.

Culling

Visibility culling can dramatically reduce render time by discarding nodes that do not contribute any pixels to the screen [1, 9, 3]. In case of LODs, culling must reject an entire object. For HLODs, a partly visible object can also be culled away. That is, if any of the nodes in the hierarchy is not visible, one may not consider the node and all its children for rendering.

Again, it is easy to see that this constraint is a variation of the MCKP where some nodes are temporarily removed from the list of items (for LODs) or the replacement sets (for HLODs).

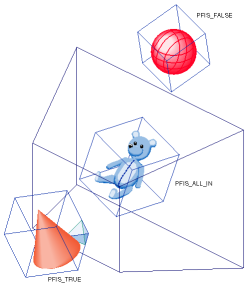


Figure 4: Frustum culling eliminates the ball and cone.

Out-of-core Rendering

Some scenes are too large to fit in memory and can lead to swapping. This implies that dynamic loading and unloading of different representations of each object must be performed and that some representations

[2]A front is the set of nodes such that all traversals from the root to a leaf node go through exactly one front node.

may not be available in memory.

The LOD selection algorithm may choose to substitute a representation that is not available with one that is available or choose to discard the object entirely.

On the other hand, if an HLOD node is not available, the front selection algorithm has four choices: 1) Render all the children of the unavailable node, if available. 2) Render the immediate parent and skip the unavailable nodes and its siblings. 3) Skip the object all together. 4) Render sibling nodes without the unavailable node, causing a hole to appear in the object.

Any of these strategies can be achieved by temporarily removing the unavailable node from the candidate set, replacement set or removing the entire object and then solving for the Knapsack problem.

Coherency

Ideally, as the user interacts and moves about in the virtual scene, only the value of each item should change and its cost should remain the same. Similarly, the capacity of the knapsack should remain constant.

In practical terms, the cost approximation for each node is not precise and depends on several intrinsic parameters of the graphics card. Since it is impossible to accurately model the cost, a feedback loop is employed that adjusts the cost of the nodes. Unfortunately, feedback loops are inherently subject to oscillations. These oscillations lead to objects switching in detail rapidly, distracting and disturbing the user.

One way to maintain frame-to-frame coherency is to bias the value of each item to discourage switching and then solving for the Knapsack problem. Another option is to seek a non-optimal solution to maintain coherency. This policy not only deviates from the original Knapsack problem, but contradicts it. Consequently, it is not possible to express this as a variation of the Knapsack problem.

Avoiding Missing Objects

The Knapsack problem will include all items in the knapsack only in two degenerate cases: 1) the knapsack capacity is very large, or 2) the number of items is very small. In all other cases, the Knapsack solution will exclude the less ideal items, resulting in missing objects.

When using LODs and HLODs, it may be preferable that all objects be included first and then some of these may be replaced with other representations from the candidate sets to optimally use the render time. This will ensure that all visible objects are always seen and never skipped.

One can adjust the value of the items to encourage all items to be included. However, this behavior cannot be guaranteed. Therefore, one must artificially impose this constraint. Unfortunately, this constraint will decrease the total value of the items in the knapsack and yield a sub-optimal solution. Therefore, it is not possible to express this constraint as a variation to the Knapsack problem.

3 Approximate Algorithms

Interactive algorithms typically render a frame once every 10-100 milliseconds. Therefore, even with a small collection of objects, an optimal solution to the Knapsack problem is not feasible. Below, we will describe and evaluate some approximate algorithms.

Adaptive Display Algorithm

Funkhouser and Séquin were the first to realize that levels of detail can be used not only to reduce the complexity of the scene but also to limit it. They called their approach the Adaptive Display algorithm.

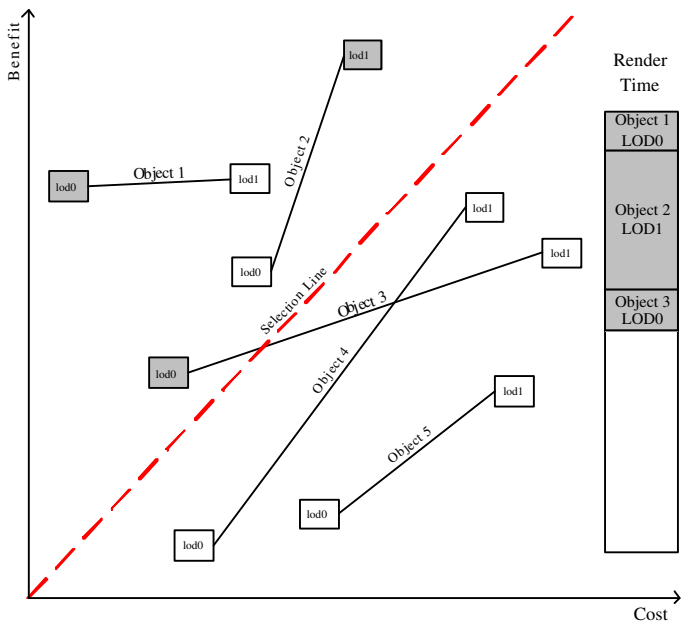


Figure 5: The Adaptive Display algorithm selects nodes by sweeping a selection line clockwise. The render time is shown on the right with selected nodes in grey.

They assign each node a "value" that is the ratio of the benefit and cost [7]. The selection of LODs to render is based on picking those nodes with the largest "value" that do not cause total estimated render time to exceed the target render time.

An analogous way to visualize this greedy algorithm is illustrated in Fig. 5 where each representation of all the objects, which are not culled away, is plotted in two dimensions based on its benefit and cost. The LODs that represent the same object are linked with a line.

The selection process starts with a vertical line through the origin. This line is rotated clockwise and any node that goes through the line is selected as long as the total cost does not exceed the target render time. If a node is selected, other nodes connected to it are not considered for selection. The line continues to be swept until there are no more objects left to be selected.

Our Algorithm

This algorithm, developed by the author, consists of traversing the HLOD hierarchy in breadth-first order and selecting a front to render each frame. On traversal, each node's priority is calculated and maintained in a priority queue. The loading thread queries the priority queue and asynchronously pre-fetches those nodes with the highest priority and unloads the nodes deemed least important.

- The **priority** reflects how important the node is in the current state of the scene. We calculate priority based on a benefit and cost ratio. The benefit is the geometric average of the screenspace and accuracy. The cost is the time needed to render the node. It is approximated by the number of triangles in the node.
- A **time slice** is assigned by splitting the available render time among the node and its siblings based on the ratio of their priorities and their cumulative priority. This time slice is then recursively divided among each node's children.
- We implement a **hysteresis** counter to limit excessive switching of HLODs. With each traversal, we increment or decrement the counter based on an urgency factor. The urgency is the normalized difference between the time slice and estimated render time. That is, if the time slice and render time are about the same, the counter will be approximately zero over time.
- Finally, a **cumulative hysteresis value** is calculated for a set of siblings by adding up each hysteresis counter and doing a truncated division by a constant. If the cumulative value is larger or equal to zero,

we recursively try to render the children. When the value is less than zero, the immediate parent is chosen to be rendered.

As illustrated in Fig. 6, the render time distribution is synonymous to drawing a curve across the hierarchy. Approximately those nodes are selected that are closest to the curve. Subsequent selection curves are drawn by moving the previous frame's selection curve up and down along each node. The amount of movement depends on how quickly the node changes in importance.

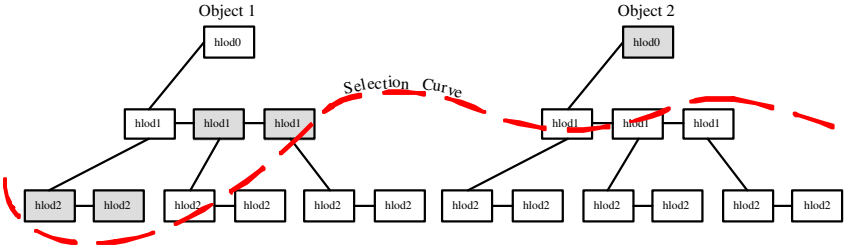


Figure 6: Our algorithm selects nodes by tracing a curve across the hierarchy. Nodes selected are shown in grey.

4 Analysis of the Adaptive Display Algorithm

Under-utilization of render time: Less detailed objects can be selected because they usually have very low cost. As shown in Fig. 7 (a), these nodes will be selected and some of the more detailed objects will not be selected in spite of having render time available for them.

The Adaptive Display algorithm compensates for this by iteratively replacing high "value" nodes with a node of higher accuracy [7]. Similarly, nodes of low "value" are replaced with a node of lower accuracy. When the same node is upgraded and downgraded, the algorithm terminates.

Over-utilization of render time: The Adaptive Display algorithm never chooses an LOD that would cause the total render time to exceed the target render time [7]. Consequently, if the render time is small, or if very few nodes can be culled away, or some nodes use up a large portion of the render time, then many objects will be skipped. This is illustrated in Fig. 7 (b).

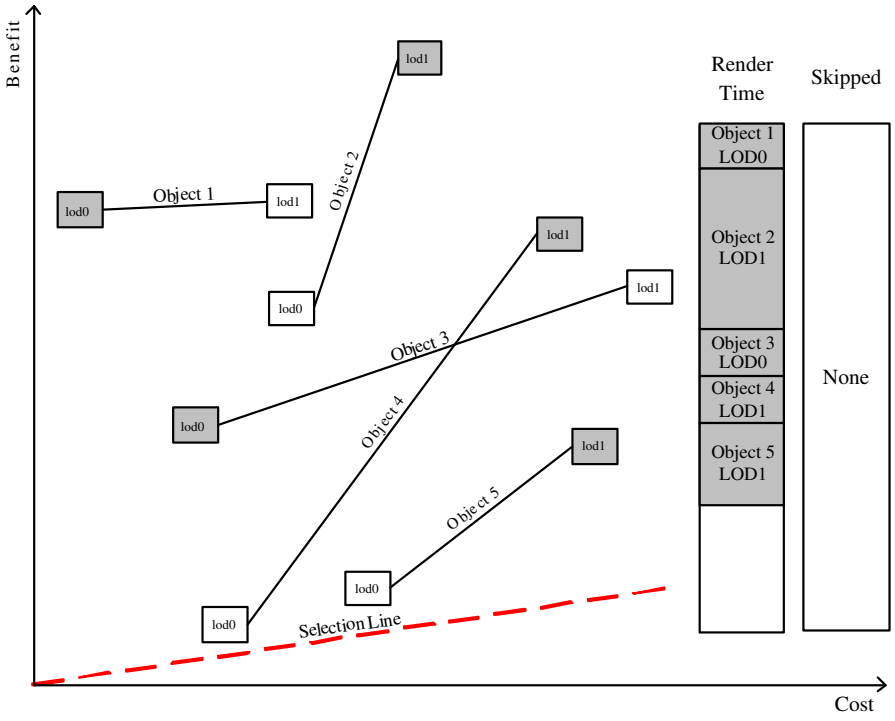
Flickering from changing benefit: As the user moves around, some objects become more beneficial as they approach the camera. Other objects become less beneficial. One can usually find certain positions where a small movement will cause flickering of the LODs.

The Adaptive Display algorithm incorporates hysteresis value into the benefit heuristic for each object. This value is proportional to the difference in the level of detail of a node from the one selected in the previous frame [7]. This scheme offers spatial hysteresis. That is, a node will toggle only when the user moves to compensate for the toggle penalty.

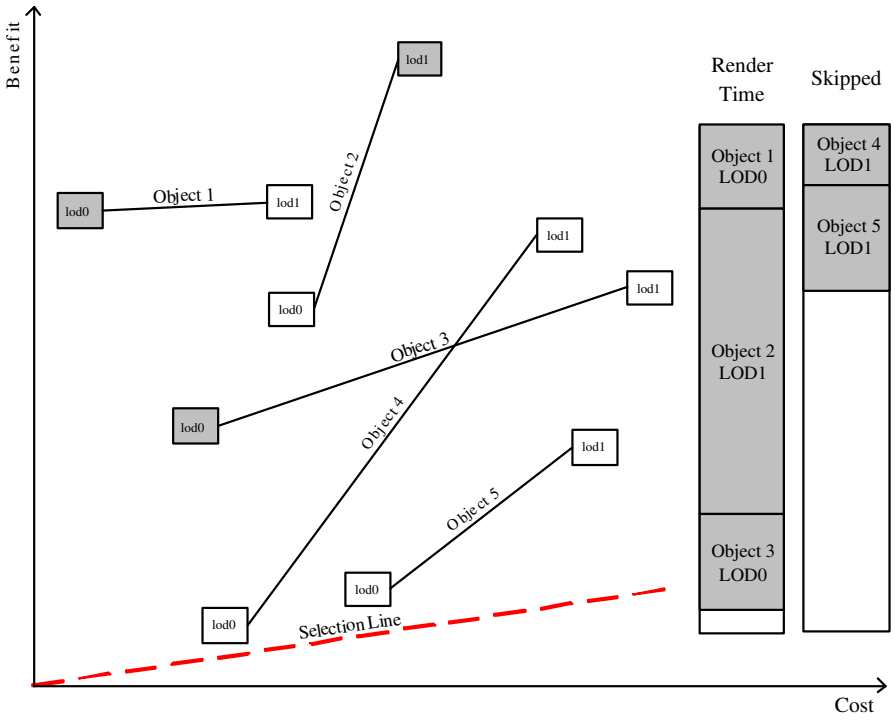
Flickering from changes in visibility: Again, one can find certain viewpoints where a slight movement or rotation can cause a substantial change in the number of objects in the view frustum. This affects how the render time is distributed and introduces flickering.

The Adaptive Display algorithm tries to use as much of the render time without exceeding it. This implies that the change in visibility of even a single object can cause flickering [6]. The Adaptive Display algorithm does not offer a solution to minimize this flickering.

Avoiding Missing Objects: Similar to the Knapsack problem, the Adaptive Display algorithm discards objects that decrease the total value of the knapsack. However, this leads to popping as the object suddenly appears when the user approaches it.

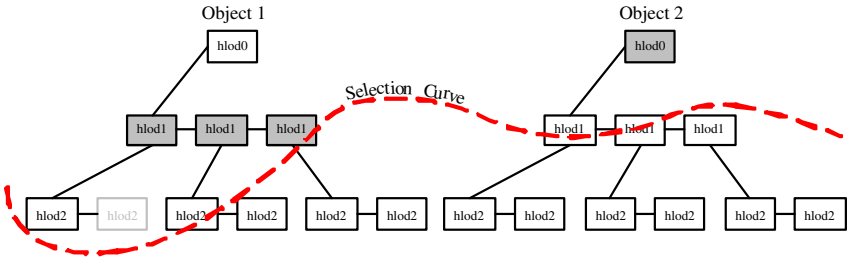


(a)

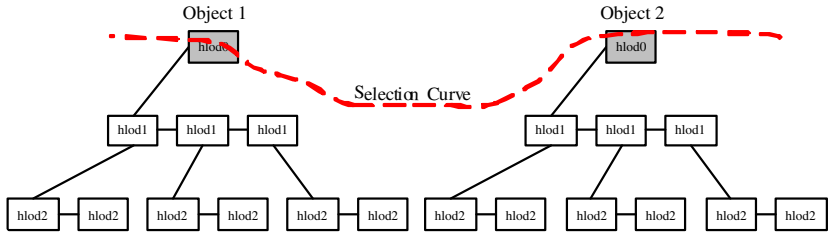


(b)

Figure 7: (a) Render time is under-utilized because objects 1 and 3 could have been rendered with more detail; (b) Over-utilization of render time is avoided by skipping objects 4 and 5. However, rendering object 2 in less detail could have prevented this.



(a)



(b)

Figure 8: (a) Render time is under-utilized because one of objects 1's $hlod_2$ is not available in memory. Also, there is enough time to render object 2's first $hlod_1$, but not enough to render all siblings; (b) We over-utilize render time by not skipping Object 1 and 2.

Analysis of Our Algorithm

Under-utilization of render time: Although the curve can be drawn arbitrarily, discrete nodes must be chosen for rendering. Also, one node cannot be chosen unless the siblings are selected as well. Lastly, if a node is not loaded in memory, the parent must be selected instead. These three issues lead to some wastage of the render time. An example is shown in Fig. 8 (a).

We compensate for this a feedback loop that adjusts the target render time passed to the root of the HLOD hierarchy for subsequent frames if the actual render time is unused and some loaded nodes were rejected.

Over-utilization of render time: The curve can be drawn such that root nodes for some objects would be skipped. Since we do not wish to skip these nodes, the target render time will be exceeded. This is demonstrated in Fig. 8 (b).

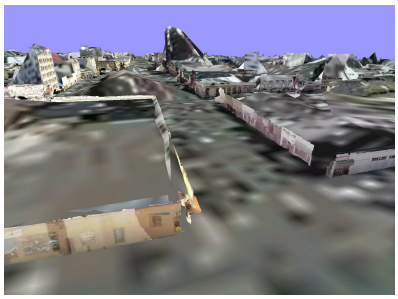
The feedback loop discussed above also decreases the target render time given to root nodes if the actual render time is larger than the target. This ensures that the target time is still met. See Fig. 9 for an example.

Flickering from changing benefit: Since we move the selection curve up and down from the previous location, our algorithm is not subject to immediate flickering over slight movements back and forth. Furthermore, the amount of time between each switch can be controlled by the hysteresis parameters. Contrary to the spatial hysteresis implementation of the Adaptive Display Algorithm, our hysteresis is temporal in nature.

Flickering from changes in visibility: Our hysteresis solution limits immediate switching when the number of objects in the view frustum change. The delay in switching depends on the number of objects that are added or removed by culling. The delay is controlled by the hysteresis parameters.

Avoiding Missing Objects: Our algorithm never discards any object. At worst, the object and surrounding high priority objects are displayed in low detail.

(a)



(b)

Figure 9: (a) When rendering at 15 frames/second, many highly detailed object representations are selected. (b) Rendering at 70 frames/second does not skip objects in the background, but lowers the detail of the foreground objects.

5 Conclusion

We have shown several considerations for interactive rendering and explained how each one can either be expressed as a variation of the Knapsack problem or shown to directly contradict it by constraining the optimal solution. We also saw two different greedy algorithms that address interactive rendering in different capacities.

The Adaptive Display algorithm uses LODs and is willing to skip objects to provide the most optimal solution. Our solution, however, is generalized for HLODs. It does not skip objects and prefers to remove detail from high priority nodes first. We change the capacity of the knapsack via the feedback loop to compensate for unavailability of objects in memory. Lastly, we allow the knapsack capacity to be marginally exceeded for better temporal coherency.

References

[1] U. Assarsson and T. Möller. Optimized view frustum culling algorithms for bounding boxes. *Journal of Graphics Tools: JGT*, 5(1):9–22, 2000.

[2] J. H. Clark. Hierarchical geometric models for visible surface algorithms. *Communications of the ACM*, 19(10):547–554, 1976.

[3] D. Cohen-Or, Y. Chrysanthou, and C. T. Silva. A survey of visibility for walkthrough applications. *Proc. of EUROGRAPHICS course notes*, 2000.

[4] G. B. Dantzig. Discrete-variatble extremum problems. *Operations Res. 5*, pages 266–277, 1957.

[5] C. Erikson, D. Manocha, and W. V. Baxter, III. HLODs for faster display of large static and dynamic environments. In *Proceedings of the 2001 symposium on Interactive 3D graphics*, pages 111–120. ACM Press, 2001.

[6] T. A. Funkhouser. *Database and Display Algorithms for Interactive Visualization of Architectural Models*. PhD thesis, The University of California at Berkeley, 1993.

[7] T. A. Funkhouser and C. H. Séquin. Adaptive display algorithm for interactive frame rates during visualization of complex virtual environments. *Computer Graphics*, 27(Annual Conference Series):247–254, 1993.

[8] M. R. Garey and D. S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., 1990.

[9] S. F. J. Foley, A. van Dam and J. Hughes. *Computer Graphics: Principles and Practice*. Addison Wesley, 1993.