

Online Occlusion Culling^{*}

Gereon Frahling Jens Krokowski

Heinz Nixdorf Institute, Computer Science Department, University of Paderborn,
D-33102 Paderborn, Germany
{frahling|kroko}@upb.de

Abstract. Modern computer graphics systems are able to render sophisticated 3D scenes consisting of millions of polygons. For most camera positions only a small collection of these polygons is visible. We address the problem of occlusion culling, i.e., determine hidden primitives. Aila, Miettinen, and Nordlund suggested to implement a FIFO buffer on graphics cards which is able to delay the polygons before drawing them [2]. When one of the polygons within the buffer is occluded or masked by another polygon arriving later from the application, the rendering engine can drop the occluded one without rendering, saving important rendering time.

We introduce a theoretical online model to analyse these problems in theory using competitive analysis. For different cost measures we invent the first competitive algorithms for online occlusion culling. Our implementation shows that these algorithms outperform the FIFO strategy for real 3D scenes as well.

1 Introduction

To visualize complex 3D scenes at interactive frame rates one needs efficient algorithms to determine the visible parts. Approximation and culling techniques are used to reduce the number of primitives that have to be rendered. Occlusion culling attempts to identify the parts of the scene hidden by objects in front of them. To identify all occlusions during the rendering it is necessary to render the primitives (or polygons) in front-to-back order in respect to the camera position. On the other hand, spatial sorting and the traversing of the used data structures are very expensive if done at all, especially for dynamic scenes. Therefore, most applications balance between updating, traversing, and rendering costs and perform only coarse spatial sorting. The rendering pipeline processes these polygons in causal order, i.e., in the order they arrive. Polygons occluded by polygons arriving later are unnecessarily rendered into the frame buffer and expensive pixel shader commands are executed. This increases the total rendering time.

Aila et al. [2] introduced the concept of a Delay Stream, i.e., a small cache, capable of storing (tiles of) polygons before rendering. If one of these polygons is culled by a polygon in the stream after it, it can be removed from the cache

^{*} Research is partially supported by DFG grant 872/8-2, by the DFG Research Training Group GK-693 of the Paderborn Institute for Scientific Computation (PaSCo).

without rendering. The authors implemented the cache as a First-In-First-Out (FIFO) buffer and showed by practical measurements that on real scenes such a buffer can reduce the number of rendered pixels by a factor of four.

This paper addresses the question if FIFO is the best cache management method. The rendering process consists of different stages. The most complex and costliest stages address the rendering of single pixels, such that the rendering time depends nearly linear on the number of pixels drawn. We address this by a theoretical model called the *size model*. Other stages perform actions for each polygon which do not depend on the size. To cover these cases we look at the *uniform model*. We analyse online algorithms for both models using competitive analysis. In the uniform cost model we propose the algorithm MARK, a variant of a paging algorithm given in [10]. In the size model we analyse the algorithm BALANCE, a variant of a weighted caching algorithm given in [7].

The paper is organized as follows: Section 2 introduces the occlusion culling models and online analysis methods. In sections 3 and 4 we show how to adapt the algorithms MARK and BALANCE to the occlusion culling scenario. Practical measurements in section 5 examine the behaviour of MARK, BALANCE, and other strategies on real test scenes. We conclude in section 6.

1.1 Related Work

Since the early nineties many occlusion culling algorithms have been presented, a recent survey is given in [8]. The methods compute the visible primitives either during preprocessing and store them in Potential Visible Sets (PVS) for regions of the camera position or determine them online. Online methods normally store the scene in a hierarchical data structure, e.g., octree[12], kd-tree [9], or bounding box hierarchy [18], to compute the point-based visibility. Usually, the sub-division is stopped if the leafs of the data structure contain less than a threshold of some hundred polygons. PVS methods usually overestimate the exact visible set to create larger regions for the camera. The system of [13] computes the PVS during the walkthrough in less than a second but the calculated sets are about two times larger than the exact visible sets. All these methods send a stream of primitives to the graphics card for rendering, which includes more or less many hidden primitives which cannot be determined by a causal visibility test. Therefore, online occlusion culling methods can be used to improve such rendering systems.

Aila et al. [2] noticed that buffering parts of the stream can be used to detect primitives that are hidden by other ones presented later. Their system consists of a FIFO buffer and the visibility of each primitive is checked two times: when it is inserted into the buffer and again when leaving the buffer.

From the theoretical point of view no results are known addressing the online occlusion culling problem using competitive analysis. Sleator and Tarjan [15] analysed the competitive performance of paging algorithms (closely related to online occlusion culling algorithms, see Section 2). They proved that LRU and FIFO are k -competitive for paging problems with a cache size of k . They also proved a matching deterministic lower bound. In [16] Young showed for several

deterministic caching strategies that they are *loosely* $\log k$ -competitive, which is a weaker model of competitiveness we don't discuss here.

A first algorithm for the weighted caching problem was given in [7]. It is k -competitive and will be discussed in detail in Section 4. Young discussed the greedy-dual algorithm, a generalization of many well known paging strategies which is as well k -competitive in the weighted caching model [16]. A general k -competitive algorithm LANDLORD for weighted caching was given in [6] and was analysed in more detail in [17]. Fiat et al. [10] showed that there are randomized online algorithms for paging which are $O(\log k)$ -competitive. The authors presented an algorithm MARK, which achieves a competitive ratio of $2H_k$, where H_k denotes the k -th harmonic number. MARK is not the best algorithm known so far, since in [11] and in [1] randomized algorithms for paging were introduced achieving a competitive ratio of H_k matching the lower bound [10].

1.2 Our Contribution

Observing similarities with online paging and caching problems we first prove lower bounds on the competitive ratio of online occlusion culling problems. We extend the algorithms BALANCE [7] and MARK [10] to online occlusion culling. Particularly, the extension of BALANCE is not trivial since one needs methods to handle the case of multi-occlusions (one polygon occluding several other polygons). We introduce new analysis methods for BALANCE and MARK and show that our variants achieve the same competitive ratio for online occlusion culling as for paging and weighted caching, resp. No competitive algorithms for online occlusion culling were known before. Our theoretical results are summarized in Table 1. In Section 5 we test BALANCE and MARK in real world scenarios. The results show that they belong to the best occlusion culling algorithms known so far, in many cases outperforming all other algorithms.

2 Occlusion Culling models and online analysis

In this section we introduce two models for occlusion culling having different cost measures. In each model our algorithm must handle a stream of polygons. Each polygon p has a screen size $w(p)$ denoting the number of screen pixels showing the polygon. A polygon can occlude other polygons on the screen. In our theoretical model we assume that a polygon is either completely occluded or not occluded at all by another polygon. By using a z-buffer [5], our algorithm can identify polygons occluded by polygons seen earlier within the stream (not paying any costs).

An online occlusion culling algorithm maintains a cache (or buffer) in which it can store k polygons. Each polygon p seen in the data stream has to enter the cache at a cache position chosen by the algorithm. If a polygon q in the cache is occluded by p the algorithm can drop q and replace it by p in the cache without paying any rendering costs. Notice that it is possible that p occludes several polygons within the cache simultaneously. If there are any free cache

positions, the algorithm can store p at a free cache position. In case the cache is full and no polygon within the cache is occluded the algorithm has to choose one polygon within the cache to be drawn on screen, paying the rendering costs for this polygon. After the drawing p can enter the free cache position. The online algorithms differ in the strategies of choosing the polygons to be drawn when the cache is full. They must decide *online*, i.e., without knowing future polygons.

We will analyse two different cost measures since rendering applications in practice can be limited by different bottlenecks [14]: in "vertex (or polygon) limited" situations all rendered polygons create uniform costs. This is addressed by the *uniform cost model*, in which an online algorithm has to pay one cost unit for each polygon it renders on the screen. Please note, if in practice the overall performance is bounded by geometry processing, our method can only speed up the rendering about a constant factor, since the time of the online occlusion culling step is dominated by the z-buffer test which is already linear in the number of tested polygons.

Many applications are "fill limited", i.e., the rendering time is dominated by processing all pixels. This is modeled in the more sophisticated *size model*, in which each polygon p contributes with its size $w(p)$ to the rendering costs.

We will analyse both models using *competitive analysis* [15]. We compare an online algorithm ALG (which must decide without knowing future polygons) with an optimal offline algorithm OPT which knows the whole instance in advance and computes the best strategy for the instance.

Definition 1. *Let c be a constant and ALG be a (randomized) online algorithm for occlusion culling. For an instance I let $C_{\text{ALG}}(I)$ denote the rendering costs of ALG on the instance and $C_{\text{OPT}}(I)$ the minimum rendering costs achievable by any algorithm on the instance. For a randomized algorithm $C_{\text{ALG}}(I)$ is a random variable dependent on the coin flips during the run of the algorithm. We call a deterministic online algorithm ALG c -competitive, if for each instance I : $C_{\text{ALG}}(I)/C_{\text{OPT}}(I) \leq c$. A randomized online algorithm ALG is c -competitive, if for each instance I : $\mathbf{E}[C_{\text{ALG}}(I)]/C_{\text{OPT}}(I) \leq c$.*

The uniform and size models have many similarities to paging [15] respectively weighted caching [7]. In the weighted caching online problem an algorithm has to maintain a cache of k pages. It sees a stream of requests of pages. When a requested page p is not in the cache, this is called a cache fault and the page must be brought to the cache by paying costs of $w(p)$. If the requested page is within the cache, an algorithm does not need to do anything, and pays no costs. In the paging online problem all page weights are one.

When we don't allow multi-occlusions (one polygon occluding several other polygons simultaneously) and we assume that each occluding polygon has the same size as the occluded polygon, the occlusion problem in the uniform model is equivalent to paging and the occlusion problem in the size model is equivalent to weighted caching. Each repeated page in a paging instance corresponds to an occluded polygon in a occlusion culling instance. Using this coherence we can transfer lower bounds from paging [10] and weighted caching [15] to online occlusion culling:

Problem	Lower bound	Upper bound
Deterministic algorithm / Uniform model	k	k (BALANCE)
Randomized algorithm / Uniform model	H_k	$2 + 2H_k$ (MARK)
Deterministic algorithm / Size model	k	k (BALANCE)
Randomized algorithm / Size model	H_k	k (BALANCE)

Table 1. Bounds on the competitive ratio shown in this paper for the analysed occlusion culling problems (see Section 2 for model descriptions). Notice $\ln k < H_k \leq 1 + \ln k$.

Theorem 1. *Let H_k be the k -th harmonic number. No deterministic (randomized) online algorithm for occlusion culling in the uniform model can be better than k -competitive (H_k -competitive). No deterministic online algorithm for occlusion culling in the size model can be better than k -competitive.*

3 The algorithm MARK

We alter the paging algorithm MARK and his analysis given by Fiat et al. [10] to handle the case of one polygon occluding several other polygons. MARK maintains for each cache position a marking bit.

Algorithm MARK
 Unmark all marking bits.
for each polygon p within the stream **do**
 If p is not occluded by a polygon already seen (test using z-buffer) **then**
 Drop all polygons within the cache occluded by p and unmark their positions (which are empty now)
 If there is an empty cache position **then**
 Bring polygon p into that position and mark the position.
 else
 If there is no unmarked cache position **then** unmark all positions
 Choose one of the unmarked positions uniformly at random
 Render the polygon of this position
 Bring polygon p into this position and mark the position
 end if
 end if
end for each

Theorem 2. *The algorithm MARK is $2 + 2 \cdot H_k$ -competitive for the uniform online occlusion culling problem, where H_k denotes the k -th harmonic number.*

The proof will be provided in the full version of the paper.

Since this occlusion culling algorithm MARK is equivalent with the paging algorithm MARK for paging on "paginglike instances" (see Section 2), we can apply the lower bound of Achlioptas et al. [1] for the paging case.

Lemma 1. *Let H_k denote the k -th harmonic number. The occlusion culling algorithm MARK is not c -competitive for $c < 2 \cdot H_k - 1$.*

4 The algorithm BALANCE

In this section we look at the more practical *size model*, in which each polygon contributes with its size on the screen to the drawing costs. From Section 2 we conclude that the weighted occlusion culling problem is at least as hard as weighted caching and no deterministic occlusion culling algorithm for the size model can have a better competitive ratio than k .

We first give evidence why all known algorithms for online occlusion culling are not competitive in the size model.

Lemma 2. *No (randomized) algorithm, which does not consider the sizes of the polygons can be c -competitive with a constant c .*

The Lemma shows that a competitive algorithm must prefer to render small polygons when the buffer is full. However, always preferring the smallest one within the buffer is also not a good strategy.

Lemma 3. *The algorithm LEASTVISIBLEFIRST which always renders the smallest polygon, is not c -competitive for any constant c .*

The proofs will be provided in the full version of the paper.

In [7] Chrobak, Karloff, Payne, and Vishwanathan showed that an algorithm called BALANCE is k -competitive for the weighted caching problem. We adapt BALANCE to the problem of weighted occlusion culling and show that it is still k -competitive. In contrast to Chrobak et al. we must deal with the difficult case of multi-occlusions.

The algorithm BALANCE maintains counters S_1, S_2, \dots, S_k for each of the k positions in the cache. Let w_i denote the size of the polygon currently within the i -th cache position.

Algorithm BALANCE

```
for each polygon  $p$  within the stream do
  If  $p$  is not occluded by a polygon already seen (test using z-buffer) then
    If  $p$  does not occlude any polygon within the cache then
      Find the index  $i \in \{1, \dots, k\}$  such that  $S_i = \min\{S_1, \dots, S_k\}$ 
      Render the polygon at the  $i$ -th cache position (if there is any)
      Store  $p$  at the  $i$ -th cache position
      Set  $S_i \leftarrow S_i + w(p)$ 
    end if
  If  $p$  occludes the polygons  $q_1, \dots, q_l$  at cache positions  $i_1, \dots, i_l$  then
    Replace the occluded polygon  $q_1$  by  $p$ 
    Set  $S_{i_1} \leftarrow S_{i_1} + w(p) - w(q_1)$ 
    For  $j = 2, \dots, l$  set  $S_{i_j} \leftarrow \max_{h \in \{1, \dots, l\}} \{S_h - w_h\}$ 
    Remove the occluded polygons  $q_2, \dots, q_l$  from their positions
  end if
end if
end for each
```

4.1 Analysis of algorithm BALANCE

We show that BALANCE is k -competitive for the weighted occlusion culling problem. Consider an optimal offline algorithm OPT. Consider a worst adversarial input sequence of polygons.

Lemma 4. *If BALANCE is not k -competitive, there is an input sequence such that*

1. BALANCE produces more than k times the costs of OPT.
2. No polygon is occluded by polygons already seen in the sequence.
3. At the end of the sequence OPT and BALANCE have the same set of polygons within their caches.
4. No polygon in the sequence occludes another one already rendered by OPT.
5. No polygon in the sequence occludes exactly one within the cache of BALANCE.
6. No counter S_i decreases during the run of BALANCE.

Proof : If BALANCE is not k -competitive, there is an input sequence, such that BALANCE produces more than k times the costs of OPT. We will alter this input sequence into an input sequence having all the remaining properties, such that the costs of BALANCE do not decrease and the costs of OPT do not increase.

A polygon which is occluded by polygons already seen in the sequence will be detected by the z-buffer and therefore has no impact on BALANCE or OPT. If at the end of the sequence OPT and BALANCE do not have the same set of polygons within their cache we alter the sequence by appending polygons occluding polygons within the cache of OPT. This does not increase the costs of OPT. At some point of time BALANCE will have rendered all polygons not in the cache of OPT and the caches have the same content.

Consider a polygon p which occludes polygons q_1, \dots, q_l already rendered by OPT. If no q_i is within the cache of BALANCE, p could be replaced by a polygon which does not occlude any polygon seen so far. This would not alter the costs of OPT or BALANCE. If q_1, \dots, q_m are within the cache of BALANCE p could be replaced by a polygon \tilde{p} having size $w(p) - w(q_1)$, and presented at a point of time, such that it will follow q_1 within the cache of BALANCE. Furthermore, we replace the polygons q_2, \dots, q_m by polygons not occluded by p , but of a smaller size, such that they remain in the buffer of BALANCE exactly until p is presented. Notice that the behavior of BALANCE is exactly the same as before. The cost of Balance therefore can only increase (by the rendering of q_2, \dots, q_m). The costs of OPT can only decrease because instead of q_2, \dots, q_m OPT now has to render smaller polygons.

We can furthermore avoid the case of a polygon p which occludes exactly one polygon q within the cache of BALANCE. If q is not within the cache of OPT, case 4 applies. If q is within the cache of OPT, we can replace q by p and delete the second occurrence of p without altering the costs of BALANCE or OPT.

We will now alter the sequence such that no counters of BALANCE decrease. A counter can only decrease if more than one polygon is occluded by a new polygon

p . W.l.o.g. this happens at cache positions $1, \dots, l$, such that $i_j = j$. Only for the occluded polygons q_2, \dots, q_l the counter values decrease, since p is stored within the first cache position. Let $\tilde{S}_j = S_j - w(q_j)$ denote the counter value of cache position j before the element q_j came into it. Let $\tilde{S} = \max_{h \in \{1, \dots, l\}} \{S_h - w_h\}$ denote the corresponding counter value after BALANCE dropped q_2, \dots, q_l . We alter the sequence to one without decreasing counters in the following way: We replace the polygons $q_j, j = 2, \dots, l$ by polygons \tilde{q}_j of size $w(\tilde{q}_j) = \tilde{S} - \tilde{S}_j$ and p by a polygon \tilde{p} of size $w(\tilde{p})$ occluding $q_1, \tilde{q}_2, \dots, \tilde{q}_l$. This way the counters S_2, \dots, S_l are exactly at value \tilde{S} when p is presented within the stream and do not decrease. Notice that all counters are at least of value $\min\{S_1, \dots, S_k\}$ until p is presented. This guarantees that no \tilde{q}_i is rendered before the occurrence of p . Therefore, BALANCE behaves the same way on the altered input sequence and the costs of BALANCE are the same as before the alteration (since q_2, \dots, q_l don't need to be rendered either way). The costs for OPT can only decrease since the sizes of the polygons are smaller after this instance change. \square

We will now concentrate on sequences fulfilling the requirements of Lemma 4. For these instances the proof follows the ideas of the proof from [7] for the paging case. We just have to deal with the fact that one page can occlude several pages in the cache. For instances fulfilling the requirements of Lemma 4 this can be analysed in the same way than single occlusions.

Theorem 3. *BALANCE is k -competitive for the weighted occlusion culling problem.*

The proof will be provided in the full version of the paper.

5 Experiments

We have implemented our proposed approach as a prototypical rendering system in C++ using OpenGL routines for the rendering. All experiments are made on a windows based system with a 1.6GHz Pentium M and a NVIDIA Quadro FX graphics card. For all benchmarks we choose a resolution of 1024×768 pixels.

We perform two common culling methods called view frustum culling and backface culling in front of the online occlusion culling step. Additionally, causal occluded polygons detected at the beginning of our online occlusion unit by the z-buffer test are dropped without counting. Therefore, remaining polygons after these standard tests will pass all stages of a common rendering pipeline, (temporally) change pixel values and cause rendering costs.

The z-buffer of the occlusion test was realized using an OpenGL feedback buffer to guarantee that the depth values used for the occlusion test are exactly the same as the depth values used later in the rendering process. Hence, for each rendered polygon we have to read out the z-buffer of the graphics card, which is a time consuming process. Therefore, we cannot state any realistic results for the improvement of the rendering time. Like Aila et al.[2], we assume that the savings in pixel processing should directly result in a increased frame rate.

Scene	Town			PowerPlant-Corridor			PowerPlant-Tubes		
# Polygons in view frustum	44563			380639			329168		
# Polygons after standard tests	14512			163053			53041		
buffer size k	40	200	1000	40	200	1000	40	200	1000
Reduction of polygons in per cent									
MARK	24.4%	33.4%	44.5%	25.4%	34.7%	46.1%	32.4%	38.8%	44.9%
BALANCE	15.8%	24.6%	38.5%	16.6%	25.8%	39.9%	21.7%	26.2%	35.5%
LRU	24.7%	34.1%	45.7%	25.3%	32.9%	45.3%	35.9%	40.5%	45.9%
NF	3.0%	19.5%	44.1%	3.4%	20.2%	30.3%	3.4%	9.4%	29.0%
MVF	4.1%	19.5%	42.2%	4.8%	20.7%	39.8%	2.3%	7.6%	28.9%
LVF	0.2%	2.5%	15.7%	0.3%	3.4%	16.4%	0.4%	2.1%	8.4%
Depth complexity	1.91			3.84			6.46		
reduction of drawn hidden pixels in per cent									
MARK	6.2%	13.7%	23.5%	8.8%	14.4%	24.6%	16.3%	21.9%	28.1%
BALANCE	7.9%	21.0%	34.9%	9.1%	23.9%	36.4%	22.7%	41.9%	60.2%
LRU	6.5%	14.7%	23.8%	7.4%	15.7%	25.3%	18.5%	21.5%	28.3%
NF	1.0%	7.8%	18.5%	1.3%	9.2%	20.1%	2.9%	6.0%	19.6%
MVF	0.7%	4.6%	12.2%	0.6%	5.3%	14.3%	2.2%	3.9%	13.1%
LVF	3.4%	15.8%	32.7%	4.1%	16.7%	33.3%	22.7%	38.9%	55.9%

Table 2. Statistical overview for different test scenes. Reduction of polygons and drawn hidden pixels for buffer size $k = 40, 200, 1000$. Best and close to the best (within 5%) results are highlighted in bold. *PowerPlant Corridor* & *Tubes* courtesy of the Walkthrough Group at the University of North Carolina at Chapel Hill (www.cs.unc.edu/walk).

5.1 Strategies

We investigated the empirical performance of the strategies MARK and BALANCE of Sections 3 and 4, respectively. Since our occlusion culling scenario is related to caching, we also consider the standard caching strategy *Least-Recently-Used* (LRU) as well as fairly natural strategies in this scenario, depending on the distance to the view point and the projected screen size:

Selection by Distance. If the polygons of a scene are processed strictly front-to-back, all completely occluded polygons will be determined by a causal occlusion test and the rendering costs will be minimized. In order to get a partially spatial sorting we have implemented the strategy *Nearest-First* (NF), which always renders the polygon nearest to the viewer among all polygons currently in the buffer.

Visible Pixels. The *Least-Visible-First* (LVF) strategy records the number of visible pixels of a polygon at the time it is inserted into the buffer. If a buffer overflow occurs, it selects the polygon with the smallest pixel counter. The opposite strategy *Most-Visible-First* (MVF) selects the polygon with the greatest pixel counter.

Depending on the cost model used for the investigation, there are arguments for both strategies: on the one hand, one expects that as the size of a polygon

increases, the "probability" for its occlusion will decrease. Additionally, it is more likely that small polygons will be far away from the view point, since the projected screen size of a polygon decreases approximately with the square of its distance to the view point [9]. Therefore, the selection strategy should prefer small polygons to be kept in the buffer, as MVF does. On the other hand, if the *size model* is observed, the rendering of a small polygon increase the total cost insignificantly. But, if it is kept in the buffer, it wastes valuable storage capacity that could be used for efficient buffering otherwise.

Note, the strategies First-In-First-Out (FIFO) and Least-Recently-Used (LRU) are identical in the online occlusion culling model because each polygon appears in the stream exactly once.

5.2 Results

Our test scenes and their characteristics are summarized in Table 2. We made experiments for each combination of strategy, scene and buffer size. The polygons are roughly spatial sorted by using an octree which is traversed in a front-to-back order. Since there is some redundancy in the results we do not give figures for all possible combinations. Instead we try to focus and explain the behavior on selected examples. We discuss the results for varying buffer sizes in detail for the *Town* scene, but the characteristics of the curves are similar for the other scenes. Therefore, we summarized the results for buffer sizes 40, 200 and 1000 in Table 2 and the results for the test scene *Town* are shown in detail in Figure 1 at the top row.

Uniform cost model. First we compare the strategies w.r.t. the uniform cost model. In the top-left diagram of Figure 1 we see the number of polygons recognized to be occluded for different buffer sizes and strategies. After all standard visibility tests 14512 polygons have to be rendered without our online occlusion culling unit, whereas just 48.7% of them are (partial) visible. The highest reduction is achieved by strategies MARK and LRU for all buffer sizes. With buffer size 1000 LRU recognizes 45.7% of the remaining polygons as occluded

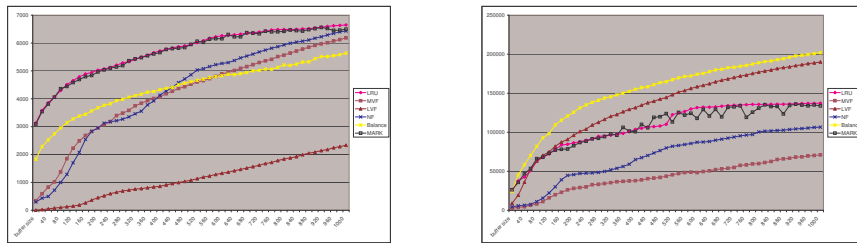


Fig. 1. Comparison of the strategies for the test scene *town*. On the left side the results for the uniform cost model are shown and the results for the size model are on the right.

(MARK: 44.5%). The second best group of strategies is composed of NF and MVF followed by BALANCE with 38.5% – 44.1% identified occlusions. Finally, the performance of LVF is very poor, because many small polygons are occluded by large polygons later arriving the cache, but the strategy LVF already selected these polygons for rendering.

Size model. After the standard visibility tests 1,212,416 pixels are changed during the rendering, but only 637,225 different pixels are visible (the remaining pixels are filled with the background color without creating any costs). Therefore, the depth of complexity of the *Town* scene is 1.91, i.e., 91% of the pixels of the screen are rendered twice on average.

The top-right diagram of Figure 1 illustrates the number of "economized" pixels for different cache sizes, i.e., the number of pixels that were saved due to the fact that the corresponding hidden polygon was not rendered. In this cost model the strategy BALANCE achieved best reduction results. With cache size 1000 it excludes 34.9% of the hidden pixels from rendering. The second best strategy is LVF (which is the worst strategy in the polygon cost model) reducing the number of changed pixel values about 32,7%. The strategies MARK and LRU behave nearly the same and exclude 23.5% and 23.8% of the hidden pixels from rendering, resp., followed by NF and MVF. The good performance of BALANCE can be explained the following way: Strategies which do not prefer rendering small polygons must have higher drawing costs. However, holding the biggest polygons within the cache forever occupies valuable cache positions. BALANCE is in theory *and* practice a good compromise.

Note, our online occlusion culling system is not able to reach the optimal depth complexity of 1.0 because many polygons are only partial visible and we do not split these polygons.

To summarize briefly, considering the uniform cost model MARK and LRU achieve the best reduction results for all of our tests and BALANCE and LVF do so for the size model.

6 Conclusion and Future Work

Interesting extensions of these results would be randomized algorithms which achieve a better competitive ratio than k for the size model. Since these algorithms would directly lead to improved weighted caching algorithms this is a challenging problem to look at.

It would be also interesting to translate other caching algorithms to online occlusion culling. Results about lookahead could probably be transferred to occlusion culling. Results about loose competitiveness would imply good results in practice.

Another challenge could be not to look at the worst case behavior of algorithms. A first step could be to develop reasonable models to represent input distributions of scenes. Different online occlusion culling algorithms could be analysed according to these input distributions and lead to even better algorithms in practice (compare [3] for average case paging analysis).

7 Acknowledgements

We would like to thank Christian Sohler for many fruitful discussions about this topic.

References

1. D. Achlioptas, M. Chrobak, and J. Noga. Competitive analysis of randomized paging algorithms. *Proc. 4th Annual European Symposium on Algorithms (ESA)*, pp. 419–430, 1996.
2. T. Aila, V. Miettinen, and P. Nordlund. Delay streams for graphics hardware. In *ACM Transactions on Graphics*, 22(3), pages 792–800. ACM, ACM Press, 2003.
3. L. Becchetti. *Modeling Locality: A Probabilistic Analysis of LRU and FWF*. Proc. 12th Annual European Symposium on Algorithms (ESA), pp. 98–109, 2004.
4. A. Borodin and R. El-Yaniv. *Online computation and competitive analysis* Cambridge University Press, 1998.
5. E. Catmull. *A Subdivision Algorithm for Computer Display of Curved Surfaces*. PhD thesis, University of Utah, 1974.
6. P. Cao and S. Irani. *Cost-aware WWW proxy caching algorithms*. USENIX Symposium on Internet Technologies and Systems, 1997.
7. M. Chrobak, H. Karloff, T. H. Payne, and S. Vishwanathan. *New results on server problems*. *SIAM Journal on Discrete Mathematics*, 4(2), pp. 172–181, 1991.
8. D. Cohen-Or, Y. Chrysanthou, C. Silva, and F. Durand. *A survey of visibility for walkthrough applications*. *Transactions on Visualization and Computer Graphics*, 9(3):412–431, 2003.
9. S. R. Coorg and S. J. Teller. *Real-time occlusion culling for models with large occluders*. In *Symposium on Interactive 3D Graphics*, pages 83–90, 189, 1997.
10. A. Fiat, R. M. Karp, M. Luby, L. A. McGeoch, D. D. Sleator, and N. E. Young. *On competitive paging algorithms*. *Journal of Algorithms*, 12, pp.685–699, 1991.
11. L. A. McGeoch and D. D. Sleator. *A strongly competitive randomized paging algorithm*. *Algorithmica* 6, pp.816–825, 1991.
12. N. Greene, M. Kass, and G. Miller. *Hierarchical z-buffer visibility*. In *Proc. of ACM SIGGRAPH 93*, pages 231–238, 1993.
13. T. Leyvand, O. Sorkine, and D. Cohen-Or. *Ray Space Factorization for From-Region Visibility*. In *ACM Transactions on Graphics*, 22(3), pages 595–604. ACM, ACM Press, 2003.
14. D. Shreiner. *Performance opengl: Platform independent techniques*. In *ACM SIGGRAPH 2001 course notes*, 2001.
15. D. D. Sleator and R. E. Tarjan. *Amortized efficiency of list update and paging rules*. *Communications of the ACM*, 28(2), pp.202–208, 1985.
16. N. E. Young. *The k-server dual and loose competitiveness for paging*. *Algorithmica* 11(6), pp.525–541, 1994.
17. N. E. Young. *Online file caching*. Proc. 9th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp.82–86, 1998.
18. H. Zhang, D. Manocha, T. Hudson, and K. Hoff. *Visibility culling using hierarchical occlusion maps*. In *Proc. of ACM SIGGRAPH 97*, pages 77–88, 1997.