Analyzing Visibility Configurations

Carsten Dachsbacher

Abstract—Many algorithms, such as level of detail rendering and occlusion culling methods, make decisions based on the degree of visibility of an object, but do not analyze the distribution, or structure, of the visible and occluded regions across surfaces. We present an efficient method to classify different *visibility configurations* and show how this can be used on top of existing methods based on visibility determination. We adapt co-occurrence matrices for visibility analysis and generalize them to operate on clusters of triangular surfaces instead of pixels. We employ machine learning techniques to reliably classify the thus extracted feature vectors. Our method allows perceptually motivated level of detail methods for real-time rendering applications by detecting configurations with expected visual masking. We exemplify the versatility of our method with an analysis of area light visibility configurations in ray tracing and an area-to-area visibility analysis suitable for hierarchical radiosity refinement. Initial results demonstrate the robustness, simplicity, and performance of our method in synthetic scenes, as well as real applications.

Index Terms—Real-time rendering, visibility, GPUs, artificial intelligence.

1 Introduction

Level of detail and occlusion culling methods pursue the goal of reducing the rendering workload by detecting which parts of a scene require detailed representations, which can be sufficiently well rendered with less detail, and which are not visible at all. These decisions are most often made on the object level or for nodes in a scene hierarchy: either they are not visible at all and excluded from rendering, or they are visible and rendered after the required level of detail has been determined. In the simplest case, this is solely based on a distance metric, and possibly on the degree of visibility. Fig. 1 illustrates the quandary of this strategy: the information that an object's surface is partly visible does not reveal if the object is blocked by a solid occluder, or if it is located behind a "see-through" blocker. See-through blockers consist of structures that are fine compared to the analyzed object, e.g., the foliage in front of the "Polygirl" model in Fig. 1. Methods taking perceptual criteria into account, such as Drettakis et al. [1], determine the visual masking from such blockers and the optimal level of detail by quantifying the visible differences between the renderings with full and reduced detail; however, they introduce significant computational overhead. Effectively, visual masking is mainly due to strong variations in lighting or shading, or from foreground see-through blockers, such as foliage or fences.

In this paper, we present an efficient and robust method to classify the different *visibility configurations* shown in Fig. 1. The possibility to identify and differentiate between these configurations provides valuable information for many algorithms relying on visibility determination. For this analysis, our method determines the distribution, or structure, of visibility and occlusion across surfaces. This is achieved by partitioning a triangle mesh into clusters, and determining the visibility of each cluster separately. The clusters' visibility and location on the image plane constitute the information used for the visibility analysis. We generalize co-occurrence matrices [2] from pixel images to scattered data to generate classification features thereof. Commonly used classifiers, such as neural networks or support vector machines, reliably classify visibility configurations even with small synthetic training sets that are general and applicable to a wide range of applications. Representatively, we discuss three algorithms for which the visibility analysis has potential use: perceptually motivated level of detail methods for real-time rendering, and two applications for ray tracing and radiosity, as examples for offline rendering methods. Fig. 2 shows the visibility analysis of an area light source with respect to surface points, which can be used to control sampling with ray tracing. For radiosity methods, we demonstrate how the basic from-point visibility analysis can be extended to a surface-to-surface visibility classification.

2 Related Work

Visibility determination has been a central problem in computer graphics since the very beginning of the field. It is required for many aspects such as the detection of visible and occluded surfaces to speed up image generation, computing shading and shadowing from light sources, the simulation of global illumination, and even in acoustics simulation. Not surprisingly an enormous amount of research has been conducted of which we only mention the most related work: Law and Tan [3] describe a framework for deducing virtual occluders, Funkhouser et al. [4] aim for interactive walk-throughs of complex environments using level of detail (LOD) rendering and precomputed visibility, while Baxter et al. [5] and Yoon et al. [6] use cluster hierarchies and occlusion culling. Exploiting coherence in hardware-assisted visibility determination greatly improves performance [7], [8]. Eisemann and Décoret [9] use GPUs for area-to-area visibility determination and point to further work on visibility and shadows. Related to our work, Andújar et al. [10] identify partially occluded geometry for occlusion culling and precompute the size of contiguous

• *The author is with the Computer Graphics Group, Karlsruhe Institute of Technology, Am Fasanengarten 5, 76228 Karlsruhe, Germany. E-Mail: dachsbacher@kit.edu*

Manuscript received 18 June 2009; revised 23 Dec. 2009; accepted 11 Apr. 2010; published online 20 May 2010.
Recommended for acceptance by K. Bala.
For information on obtaining reprints of this article, please send e-mail to: tvcg@computer.org, and reference IEEECS Log Number TVCG-2009-06-0123.
Digital Object Identifier no. 10.1109/TVCG.2010.77.

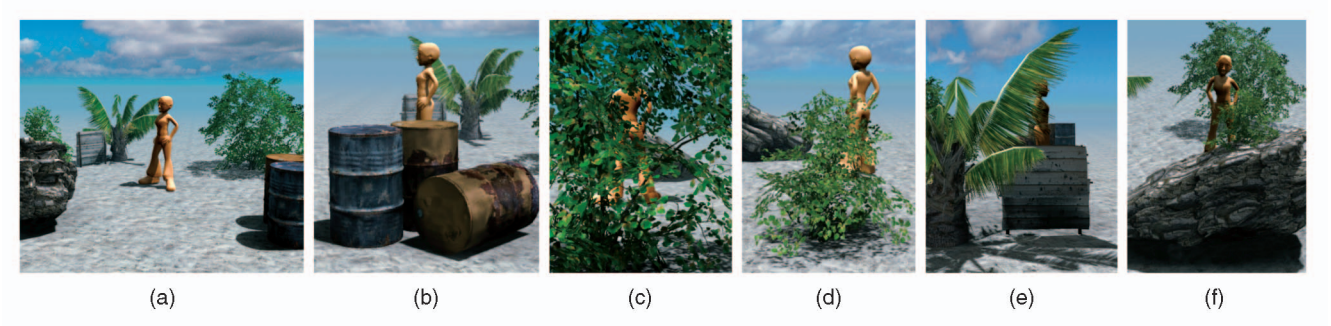


Fig. 1. Our method classifies the various visibility configurations for the model shown in orange (full invisibility/occlusion is not shown here). Although approximately 50 percent of the object's surface are visible in (b) and (c), the situation is different. We split the object into few clusters and combine the visibility and the image space locations of the clusters to extract features for an efficient classification with neural networks. (a) Fully visible. (b) Partial coverage solid blocker. (c) Full coverage see-through blocker. (d) Partial coverage see-through blocker. (e) Full coverage solid and see-through. (f) Partial coverage solid and see-through.

visible regions; however, they do not analyze their structure as we do. Charalambos et al. [11] presented a hierarchical LOD and occlusion culling method that uses visibility information obtained from hardware occlusion queries. This work is orthogonal to our method and could directly benefit from the visibility analysis. Wonka et al. [12] determine from-region visibility using stochastic ray shooting and path mutation, while Zhang and Turk [13] use visibility as a criterion for surface simplifications. Bittner and Wonka [14] and Cohen-Or et al. [15] provide comprehensive overviews over this research area.

Rendering algorithms have always been high consumers of computational resources and LOD methods aim at reducing the surface detail (with regard to geometry or shading), and thus, the rendering cost. Luebke et al.'s excellent textbook [16] gives an introduction to this topic, and the work of Drettakis et al. [1] provides an overview over perceptually based rendering, which has been the focus of much research over recent years.

Our method is also based on work from other research areas. We adapt co-occurrence matrices (CMs) [2], known from the vast field of image processing, for visibility analysis and use machine learning techniques [17] for classification. Previously, CMs have also been used for mesh segmentation [18], which can be considered as the counterpart to texture-based segmentation in images. To our knowledge, extracting features with CMs for analyzing visibility configurations is new in graphics, and may have many applications.

3 Classifying Textures and Visibility

Co-occurrence matrices are a well-known statistical technique for feature extraction in image and texture analysis. They gather information on the spatial distribution of colors or gray levels in a pixel neighborhood.

The key observation is that we can interpret the distribution of visibility across an object as a texture. As illustrative example, imagine that we replace the Polygirl model in Fig. 3 by a rectangular panel facing the viewer. Next, we virtually subdivide this panel into a grid of 6×6 subrectangles and determine the visibility for each of them (Fig. 3). This results in an image with 36 pixels with gray levels (quantized to G discrete levels) ranging from black for invisible subrectangles to white for fully visible subrectangles. The resulting "visibility texture" looks similar to the synthetically generated patterns shown in the second row of

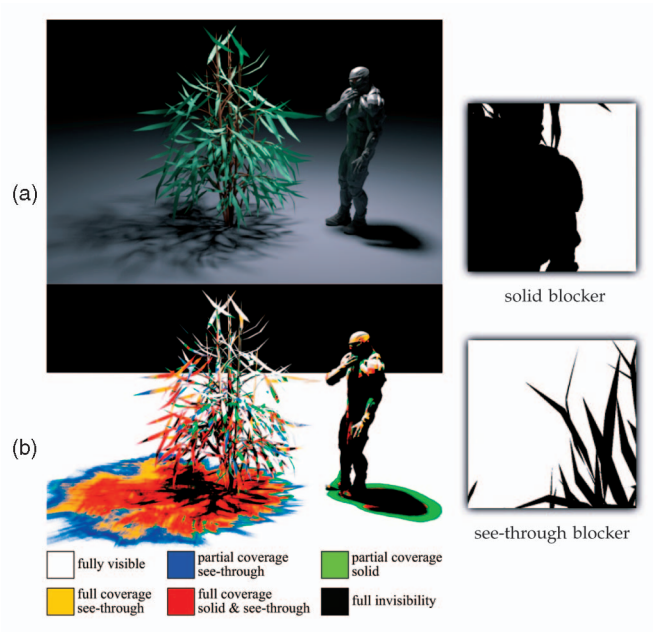


Fig. 2. (a) Our ray tracing test scene with solid and see-through blockers lit by an area light source. (b) The classified visibility configurations (color-coded) of the light source with respect to surface points can be used to control sampling.

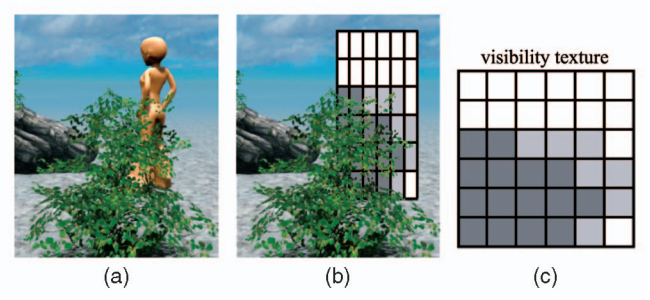


Fig. 3. The visibility of the model can be interpreted as a gray-scale texture. The characteristics of such textures (shown isolated on the right) allow the classification of the visibility configurations.

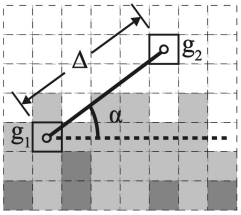


Fig. 4. Co-occurrence matrices count how often every pair of gray values (g_1, g_2) appears at a distance Δ and under an angle α off the x-axis.

Fig. 5, and in the following sections, we detail how to use CMs to describe the characteristics of such images. We will ultimately use the objects' surface divided into clusters to determine the visibility and adapt co-occurrence matrices to extract features directly.

3.1 Feature Extraction

In this section, we will first introduce the CMs used for our feature extraction, and next describe the classification and training for the analysis. In general, a CM $P_{\alpha,\Delta}$ characterizes a gray-scale pattern within a pixel region R of an image f. Intuitively, a CM counts how often a pair of gray values (g_1, g_2) appears at a distance Δ and under an angle α off the x-axis (Fig. 4):

$$P_{\Delta,\alpha}(g_1, g_2) = \sum_{\vec{x} \in R} \delta(f(\vec{x}) - g_1)\delta(f(\vec{x} + \vec{d}) - g_2), \qquad (1)$$

where δ is the discrete delta function with $\delta(0) = 1$, and $\delta(x) = 0$ for $x \neq 0$, and $\vec{d} = \Delta \cdot (\cos\alpha, \sin\alpha)^T$.

We will use a variant of CMs that completely ignores orientation and considers the bidirectional relationship of gray levels only. This yields symmetric matrices, which are invariant to rotation and mirroring of the input:

$$P_{\Delta}(g_1, g_2) = \sum_{\substack{\vec{x},\vec{y} \in R \\ \|\vec{x}-\vec{y}\| = \Delta}} \delta(f(\vec{x}) - g_1)\delta(f(\vec{y}) - g_2). \qquad (2)$$

CMs in image processing are typically large (for G gray levels, the matrix has size G^2) and sparse, and thus, various metrics of the matrix are taken to get a more useful set of features, e.g., the well-known Haralick [2] features which measure among others contrast, entropy, and correlation. As we will see, a small number of gray levels are sufficient for the visibility analysis to distinguish the configurations shown in Fig. 1 and we will directly use the CMs as input to classifiers.

A first indication about the visibility configuration can be obtained from the histogram of the fractional visibilities, which is contained in the diagonal elements of the P_0 matrix (the CM for $\Delta = 0$). Obviously, the lack of spatial relationship in the histogram prevents the differentiation between fundamentally different configurations, such as those shown in Figs. 5f and 5g. Information to overcome this problem is present in the P_{Δ_1} matrix, where Δ_1 is the size of a pixel of the visibility texture, i.e., the distance of the centers of two adjacent pixels. From this matrix, we extract the diagonal elements $P_{\Delta_1}(i, i)$ and the symmetric elements $P_{\Delta_1}(i, j) = P_{\Delta_1}(j, i)$, $i < j$. Together with the histogram, we get a feature vector with $G + (G^2 + G)/2$ elements for the classification. Using CMs with distances of a multiple of Δ_1 does not improve the classification of the visibility configurations shown in Fig. 1. Since we aim to use few visibility tests only, i.e., low-resolution visibility textures, the $P_{k\cdot\Delta_1}$, $k > 1$, matrices do not contain reliable information. Classifying more complex configurations, e.g., stripe patterns, is possible with higher resolution visibility and direction-dependent CMs, but of no interest for our visibility analysis.

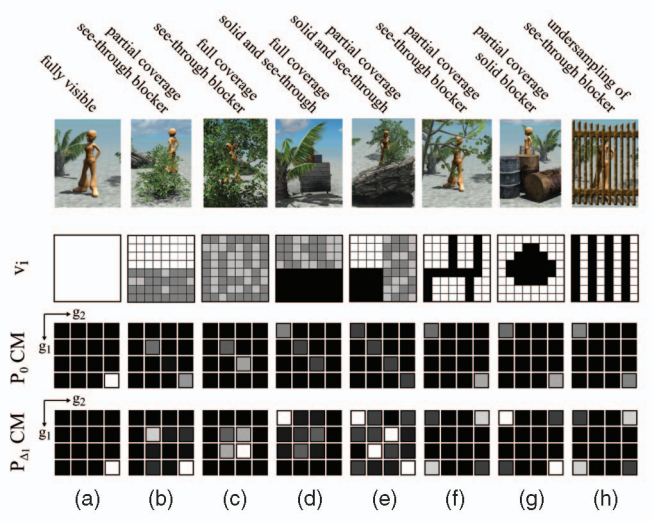


Fig. 5. Top rows: exemplary visibility configurations and corresponding synthetic visibility patterns of 8×8 clusters with $G = 4$ quantization levels from black (full occlusion) to white (full visibility). The corresponding P_0 and P_{Δ_1} CMs are shown below: brighter elements denote a larger number of co-occurring fractional visibility pairs. Note that the co-occurrence matrix P_{Δ_1} is required to distinguish the two patterns (f) and (g). The visibility pattern shown in (h) is undersampled: a slight camera movement can result in a visibility pattern similar to (c). This is the borderline case where the structures of the occluder are too small to be detected as solid blockers and treated as see-through blockers instead. Note that such a perfect alignment of clusters and occluders as in (h) is extremely rare in practice.

3.2 Classification and Training

After little training, a human observer can easily differentiate the various visibility textures or configurations shown in Fig. 5 by looking at the CMs. Deriving a fixed decision scheme based on these synthetic configurations is feasible; however, it requires handcraft (the feature vector has $G + (G^2 + G)/2$ elements), it is hard to fine-tune and error-prone. Instead, we model the observer with widely used classification tools: multilayer perceptrons (MLPs, an artificial neural network with a simple acyclic topology [19]) or support vector machines (SVMs) [20]. Both are simple to implement, quickly trained, and well suited for simple classification tasks as in this case, while SVMs can be more reliable in more complex applications [21], [22]. In addition to increased flexibility and tolerance to noisy data, both are able to model the observer just from example data, i.e., a given training set of visibility configurations. One rule of thumb for classification is that the quality of the extracted features is of utmost importance and the choice of the classifier is secondary. Our experiments attest this and both MLPs and SVMs performed equally well. In the remainder of the paper, we will focus our description on MLPs as they are simple to use and robust libraries for classification and back-propagation learning exist [17].

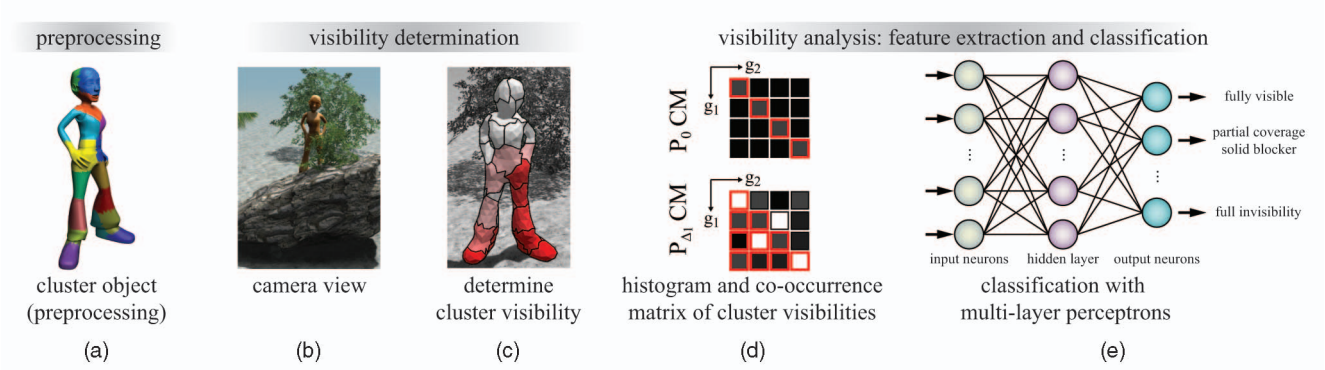


Fig. 6. Overview of analyzing the from-point visibility configuration of an object: in a preprocessing step, we cluster the object, and at runtime, determine the visibility of each cluster (typically after rendering the image). The fractional visibilities and the locations of the clusters in image space determine the histogram and the co-occurrence matrix. The visibility configuration is classified from the feature vector (all red-framed entries of the matrices) using a multilayer perceptron. The result can then be used for controlling the object's level of detail in the subsequent frame.

In total, we use the $G + (G^2 + G)/2$ inputs stemming from P_0 and P_{Δ_1}. In addition to the corresponding input neurons, we create a hidden layer with the same number of neurons, and as typical for neural networks, we create one output for each visibility configuration. As neural networks require the input values to be bounded, we scale the values obtained from P_0 and P_{Δ_1} prior to classification and training to the interval $[0; 1]$. For our examples, we use an MLP that has been trained using synthetic configurations as shown in Fig. 5. During training, we set all output values to zero, except for the output corresponding to the current visibility configuration which is set to one. When using the MLP for classification, the largest output indicates the classification result. Without the hidden layer, the classification becomes less robust—however, we did not observe better classification using more neurons. Since the evaluation of small MLPs is not time-critical, we did not investigate further optimization techniques such as pruning.

In our implementation, we used a freely available and lightweight library for MLPs [23]. For the training of the MLP (for $G = 4$, i.e., 14 inputs), we used automatically generated variations of the different synthetic patterns shown in Figs. 5a, 5b, 5c 5d, and 5e, and one for the case of full invisibility. The variations are created by slightly varying the size of the visible, partially visible, and occluded regions, and by filling partially visible regions with random mid-gray values. By this, we create a great variety of patterns covering all visibility configurations that we want to classify. Note that no rotations or mirrorings of the patterns need to be generated as our co-occurrence matrices are invariant to these transformations. The training is required only once and takes 390 milliseconds with 2,500 training rounds, which provide converged MLP weights. Using [23], the MLP can then be evaluated 180,000 times per second on a single core of an Intel Core2Quad CPU at 2.4 GHz.

4 Analyzing Visibility Configurations

Unlike previous methods, we do not only determine the degree of visibility of objects, but we also provide information about the *distribution* or *homogeneity* of the visibility across an object's surface using CMs to extract this information. This enables more intelligent decisions for LOD selection and culling algorithms, or more generally, for algorithms which make decisions based on visibility tests.

For the explanation of the basic algorithm, we first consider the case of a from-point visibility determination, i.e., we classify the visibility configuration for an object in a scene rendered from a standard camera as shown in Fig. 1. We distinguish the configurations depicted there, and of course, the case of full invisibility.

First of all, we need to determine the visibility distribution across each object to extract the classification features for the visibility analysis as described above. Obviously, replacing objects by planar panels to obtain visibility textures is no general solution and may yield wrong visibility, particularly in scenes with dense geometry. A simple solution is to virtually subdivide the screen space bounding rectangle of an object into a grid, and next determine for every grid cell how many pixels the object covers without occluders and how many pixels there are actually visible. Counting the pixels can be efficiently done using hierarchical item buffers [24]; however, this approach requires rendering each object twice: first, each object alone to determine the screen coverage without occlusion, and then, all objects together to resolve the final visibility.

Instead of determining the visibility on a regular grid (i.e., computing a visibility texture as in Fig. 3), we propose to use a combined object-space and screen-space approach: we partition the object's surface into clusters (of roughly equal surface area), and determine their fractional visibility (the ratio of visible to projected area for each cluster) at render time. For the visibility analysis, we then adapt CMs to work with irregularly sampled visibility information, in this case, the cluster visibility and the cluster centroid locations in image space. The actual visible surface area of each cluster is determined by sampling, e.g., with hardware occlusion queries, and the projected visible area without occlusion from other objects is computed analytically. However, efficiently computing the latter requires an estimation of intraobject self-occlusion to extract meaningful features for the visibility analysis, which we describe in Section 4.2. Fig. 6 gives an overview of our method. The steps of our algorithm are depicted in Figs. 7, 8, and 9.

4.1 Clustering

In order to classify the different visibility configurations, we need to determine the visibility distribution on an object's

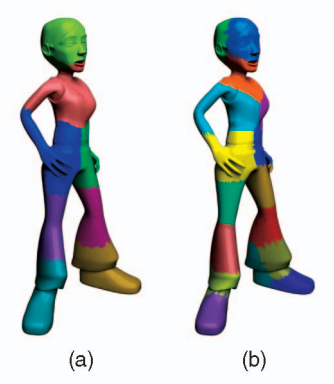


Fig. 7. The Polygirl model with 8 (a) and 16 (b) clusters created using the simple method described in Section 4.1.

surface. For this, we partition the surface into disjoint clusters, and at render time, we determine the visibility of all clusters. We decided to use a clustering algorithm similar to the method used by Sander et al. [25]. This method alternates between two phases (starting with an initial cluster seed): one which assigns triangles to clusters (starting the search from the centroids) and one which computes a new centroid for each cluster (searching from the cluster boundaries inward).

We modify this approach making it simpler and better suited for our requirements (Fig. 7 shows results of our clustering algorithm). The first modification is that cluster centroids are not required to reside on the surface. Second, we replace the search in phase 1 by simply assigning triangles to the cluster with the closest centroid. This effectively removes the normal variation penalty of the original method applied when growing clusters, and lifts all constraints on cluster topology. Both effects are intended as we do not aim for planar clusters or well-behaved chart parameterization as in [25]. In phase 2 of the clustering algorithm, we place the new cluster centroids at the center of the minimal bounding sphere containing all triangles of a cluster. Note that nonplanar clusters result in more uniformly sized, isotropic cluster shapes when projected in screen space. Enforcing planar clusters, however, can result in highly anisotropic cluster shapes, e.g., for Neptune's trident (Fig. 11) or the arms of the Polygirl model (Fig. 7). The visibility analysis with rotation invariant

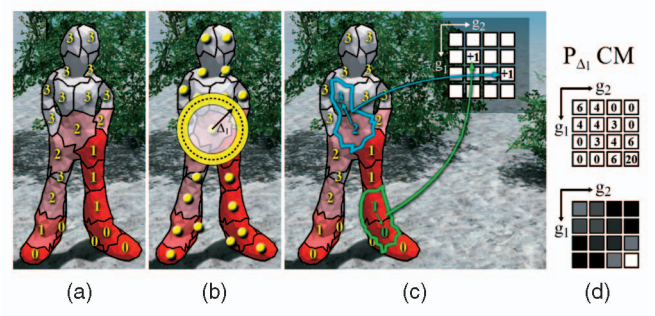


Fig. 8. For each cluster, we determine the quantized fractional visibility (a), and estimate the distance Δ_1, characterizing the direct neighborhood of two clusters in image space, from the 2D centroid locations (b). Counting the occurrences of pairs of gray values of neighboring clusters (c) yields the CMs used for classification (d).

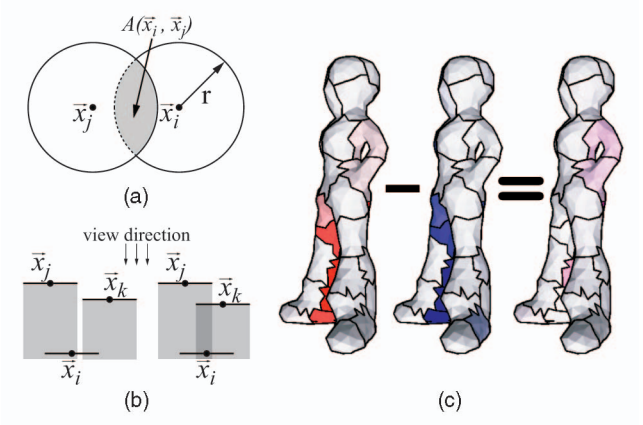


Fig. 9. We assume that clusters cover a circular area on the image plane (a). Multiple occlusions (b) are resolved by iteration. (c) Fractional visibility v_i (red), the estimated self-occlusion V_i^2 (blue), and the corrected term v_i' (purple, saturation doubled for illustration).

co-occurrence matrices works best with isotropic clusters (see also Section 5.1.1).

4.2 Fractional Visibility

For the view-dependent analysis of the visibility distribution across an object's surface, we determine the fractional visibility of each cluster in post-transformed view space. The fractional visibility v_i, of the ith cluster, is the ratio of actually visible pixels a_i to the total projected screen size s_i, of front-facing triangles *without blocking from other objects*. For the feature extraction, we quantize the fractional visibility to $G = 4$ levels (see Section 6 for a discussion). Clusters containing back-facing triangles only or located outside the view frustum do not provide any information to the visibility distribution and we cull them prior to the analysis. We also cull clusters which cover only few pixels on the screen, as their fractional visibility might be error-prone and unstable. As no practical analytic solutions to the general visibility determination exist, we sample the actual visibility a_i with hardware occlusion queries (in interactive applications), or with ray casting (in offline rendering). The occlusion query mechanism is natively supported by graphics hardware. It detects the visibility of a rasterized surface against the contents of the depth buffer and returns the number of pixels passing the depth test. The projected size of a cluster can be computed pixel-accurate given the transformation and projection matrices and the screen resolution; however, this computation cannot consider self-occlusion (from other triangles or clusters of the object itself) or occlusion from other objects. If there is intraobject self-occlusion, then it will reduce the number of actually visible pixels a_i, which is detected by the occlusion queries, and consequently, we also need to account for it when computing s_i.

Concave objects generate this self-occlusion: clusters in the front can occlude clusters behind them and this can be misinterpreted as occlusion from other objects in the scene. Thus, we need to determine the self-occlusion for each cluster in order to correct the determined fractional visibility. When using occlusion queries for this as well, then two additional render passes per object would be required: rendering all clusters without any occluders,

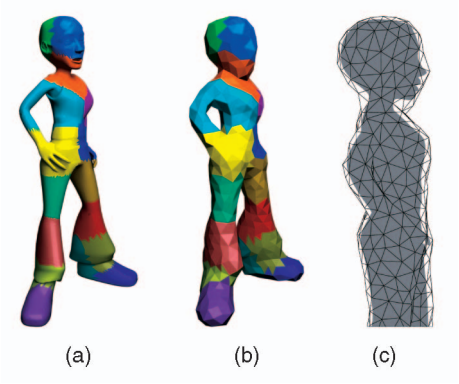


Fig. 10. (a) The Polygirl model and (b) its envelope with 16 clusters (see Section 5.1). (c) The superimposed meshes.

followed by rerendering with occlusion queries. That would increase the computation cost significantly, but we found that a fast approximation is sufficiently accurate to make the classification reliable. For this, we assume that all clusters cover a circular area around their centroid location in 2D image space coordinates $\vec{x}_i$ with a diameter r (the derivation of this parameter is described in the next section). The distance of a cluster to the viewer is d_i, and the size of the overlap region of two clusters is denoted by $A(\vec{x}_i, \vec{x}_j)$ (see Fig. 9). We then compute the self-occlusion for each cluster by determining the fraction of its disc area that is visible and not occluded by the other discs. However, instead of an accurate computation, which is costly with many clusters, we use an iterative approximation in the spirit of [26]:

$$V_i^m = V_i^{m-1} - \sum_{j=0, d_j<d_i}^{N-1} V_j^{m-1} \frac{A(\vec{x}_i, \vec{x}_j)}{r^2\pi}, \tag{3}$$

with iteration m and $V_i^0 = 1$. This typically yields results within 5 percent of the accurate disc-based overlap with $m = 1$ or $m = 2$ iterations (see below for a detailed analysis). We substitute the fractional visibility v_i (accounting for self-occlusion and occlusion from other surfaces) for the corrected term $v_i' = \min(1, v_i/V_i^m)$, if $V_i^m > 0$. This effectively removes self-occlusion to an extent, which proved sufficient in all our tests.

Note that this approximation does not correct for self-occlusion of triangles inside one cluster. However, for many models, and in particular, for the envelope meshes (introduced in Section 5.1 and shown in Fig. 10), clusters tend to be convex without pronounced concavities. As we cull back faces, the impact of intracluster self-occlusion is often negligible. Of course, if concavities dominate, either an accurate determination of self-occlusion or creating flatter clusters (either by normal-aware clustering or simply more clusters) to reduce intracluster occlusion is necessary. Enforcing planar clusters, however, does not avoid the intraobject self-occlusion itself.

Discussion. We evaluated the accuracy of the iterative disc-based overlap by comparing it to the analytically computed visibility of the clusters' discs. For a camera round trip in the scene shown in Fig. 1 (and in the accompanying video which can be found at http://doi.ieeecomputersociety.org/10.1109/tvcg.2010.77), the average error of approximation was 3.9 percent, the maximum error was 36.0 percent; a change in the quantized fractional visibility was very rare. Note that the classification is tolerable against few variations and we did not detect any different classifications hereby. In principle, it would be possible to compute the occlusion among all objects in the scene based on disc approximations instead of using visibility queries. However, for reasonably sized scenes, the number of clusters becomes large and reducing the computation cost requires hierarchical data structures [26]. Note that we also would need a disc representation for all objects (no matter if a visibility analysis for them is intended), and see-through blockers require a nonstraightforward mapping to semitransparent discs.

4.3 Co-Occurrence Matrices for Clustered Geometry

CMs have previously been used with images, but the above definition does not preclude us from using them for the fractional visibility distributions, as soon as we find a distance characterizing the "direct neighborhood" of two clusters on the image plane (analogous to the distance of the centers of two adjacent pixels in an image). A meaningful value is the mean shortest distance between two clusters. Let $\vec{x}_i$, $0 \le i < N$, be the centroid location in 2D image space coordinates (i.e., in post-transformed view space) of the ith cluster, then we get

$$\Delta_1 = \frac{1}{N} \sum_{i=0}^{N-1} \min_{\substack{0 \le j < N \\ j \ne i}} \|\vec{x}_i - \vec{x}_j\|. \tag{4}$$

Δ_1 is roughly equal to the mean cluster diameter projected onto the image plane due to the uniformity of the clustering. Thus, we can compute a CM for a set of clusters and their fractional visibilities v_i replacing the set of pixels R and the image f. Of course, it is unlikely that the distance between two centroids is exactly Δ_1, and thus, we replace the distance test in (2) by $|\|\vec{x} - \vec{y}\| - \Delta_1| < \epsilon$. The value chosen for ϵ proved to be uncritical and we chose 20 percent of the cluster image space diameter in all our examples.

Discussion. According to (2), all pairs of *not entirely backfacing* clusters (with distance Δ_1) contribute equally to the CM. We experimented with weighting the influence of clusters according to their projected surface area, which strengthens the influence of predominantly front-facing clusters on the classification result. Although weighting seems reasonable at first sight, we omitted it for two reasons. First, for thin object parts such as the Neptune's trident in Figs. 11 and 12, or parts seen from grazing angles (i.e., typically peripheral object parts), the influence of the respective contribution to the co-occurrence matrix diminishes. Note that we want to determine the visibility configuration across the entire object, not only for large parts. Nevertheless, clusters that cover very few pixels only are removed prior to classification to prevent unstable results. Second, for rather bulky objects without such thin parts, the impact of weighting was negligible.

5 Applications

We believe that our visibility analysis is an interesting tool to be used together with various existing algorithms that make decisions based on visibility, and will stimulate

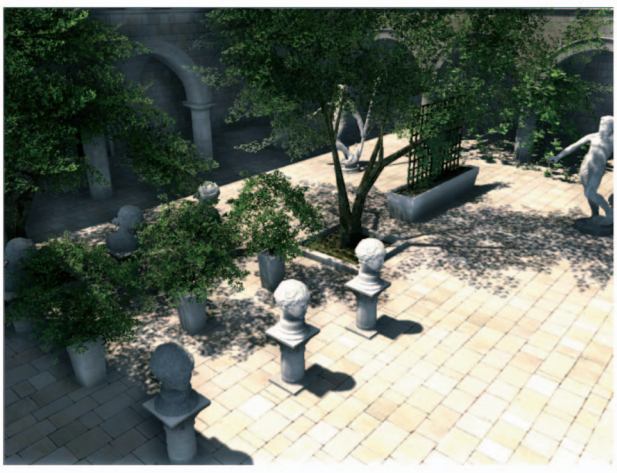


Fig. 11. We use the visibility analysis for adjusting the LODs for the statues and busts in this scene. The static part (Sponza Atrium and vegetation) consists of 199,000 triangles.

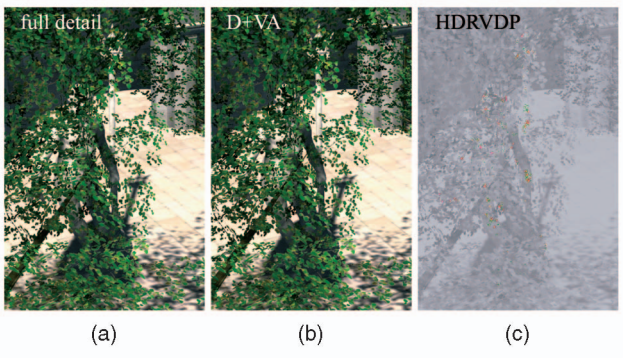


Fig. 12. A close-up comparison of rendering with full-resolution models (a), and LOD controlled by distance and visibility analysis (b) as described in Section 5.1. The visible differences prediction (c) was computed using HDRVDP [30] for $1{,}920 \times 1{,}080$ resolution on a $250\ cd/m^2$ display: green and red represent regions with little and strong visible differences; gray denotes no difference.

further research. Methods that possibly benefit from this method are, among others, perceptually motivated LOD methods, image-space occlusion culling techniques, e.g., [7], [8], [11], and offline rendering algorithms such as area light sampling with ray tracing or hierarchical radiosity. In this section, we outline initial ideas for both real-time and offline rendering for which the visibility analysis has potential use.

5.1 Perceptually Motivated Level of Detail Control

Real-time LOD control methods typically estimate the LOD for an object in a given view based on viewing distance (or closely related the screen size), the degree of occlusion from other surfaces [27], and sometimes additional perceptually motivated criteria, such as eccentricity or velocity. In general, LOD methods consist of three parts: the generation of simplified versions of an object, the selection of the detail levels, and switching or blending between them. Our visibility analysis can be used to improve on the selection of detail levels by considering the distribution of visibility across an object, and thus, providing an efficient way to incorporate findings from recent perceptual rendering methods.

Drettakis et al. [1] take visual masking due to contrast and spatial frequency for LOD selection into account. For this, they compute *threshold maps* [28] storing the predicted visibility threshold for each pixel. This perceptual threshold predicts the maximum (luminance) error that can be tolerated at every location over the image. At regular intervals, the currently chosen LOD for an object is compared against the highest quality representation, by rendering both and counting the number of pixels exceeding the visibility threshold. The decision for increasing, decreasing, or maintaining the LOD is then based on this result. Detecting the interobject visual masking is realized by splitting the scene into depth layers. Creating depth layers and reference images, computing threshold maps, and comparing introduce a significant rendering cost. This overhead only amortizes in scenes with complex geometry and widespread visual masking.

Intuitively, there are two cases causing significant visual masking for an object: masking from foreground occluders with fine structures, such as trees or fences, and second, strong variations in lighting or shading, for example, from hard shadows. We can directly map the first observation to our visibility analysis: whenever an object is partially or completely visible, we cannot expect pronounced masking effects everywhere on the surface and we base the LOD selection on conventional criteria (distance, eccentricity, etc.). However, if we detect that the object is entirely behind a see-through blocker, we reduce the detail (on an per-object basis) as we expect that the visual difference is less apparent to the human observer.

Of course, occlusion and visual masking are two different things: masking stems from spatial frequency content, orientation, and local contrast, while costly computations are required to detect when and where this phenomenon appears. However, for interactive applications, our visibility analysis—which is very cheap to compute, compared to perceptual models—might be a practical alternative when used together with conventional LOD criteria: we identify configurations where masking is very likely and reduce the level of detail accordingly. This does not require any intricate integration into rendering pipelines compared to [1], and allows to optimally exploit the considerable processing power of modern graphics hardware.

5.1.1 Masking and Perceptually Motivated Definition of See-Through Blockers

Obviously, neither the histogram nor a CM captures "perceptual factors," but we can infer visibility configurations from the contained statistical data, and thus, detect when masking can be expected. Consequently, the definition of see-through blockers has to be linked to the spatial frequencies that cause masking effects. This can be best explained with the help of the illustrative example of the visibility texture on the rectangular panel (Fig. 3). Each subrectangle of the panel measures visibility, and the size of the subrectangles determines which feature sizes of occluders can be resolved: larger features are detected as contiguous solid parts, whereas roughly equal-sized or

smaller features yield partly covered subrectangles, and thus, are detected as see-through blocking. Thus, for detecting cases of visual masking, the subrectangles should ideally be as small as the patterns causing masking. The assumption in our method is that all see-through blockings cause a certain degree of masking, which then can be detected using larger subrectangles (whose size is chosen manually). Note that the actual masking effects do not only depend on spatial frequencies but also on shading; however, we make another simplifying assumption here and rely on visibility only. This observation holds analogously for the visibility determination using clusters. When the camera moves closer to (or further away from) an object, we use more (or less) clusters to keep their size in image space—and thus, the relation to see-through blockers—roughly constant. Note that the relation between cluster and feature sizes is another reason for enforcing isotropic clusters: anisotropic clusters would distinguish solid and see-through blockers differently depending on the cluster and feature orientation.

Analogous to the identification of masking from occluders, we can also detect shadow configurations by replacing the visibility test with a shadow test. Our experiments showed that this works well with point light sources as they cause hard shadows, but evaluating masking from shading, in general, requires perceptual models, such as threshold maps [28] or visible differences predictors [29]. Note that in contrast to these metrics, our definition of see-through blockers completely ignores color and shading.

5.1.2 Level of Detail Control in Real-Time Applications

We implemented our approach in an interactive rendering system using discrete LODs and cluster each LOD of an object such that we can maintain roughly equal cluster sizes in screen space across objects and view distances. In practice, we used 8, 16, and 32 clusters, fewer for coarse LODs and more for detailed representations. Note that the method described in this section is meant for objects that are not made of subparts that, in turn, provide their own levels of detail. However, the visibility analysis can be combined with hierarchical LOD methods and, for example, carried out only for partially visible subparts. For each frame, we first render the whole scene and thereafter determine the cluster visibilities to perform the analysis. With distance-based metrics, the artist assigns LODs to distance ranges, so we let the user define how strongly the LOD selection is affected: in our example, we reduce the detail for objects behind see-through blockers (as in Figs. 1c and 1e) by 1 or 2 levels depending on the amount of see-through blocking indicated in P_0. Extracting more quantitative information from the co-occurrence matrices proved to be unreliable, mainly due to the noisy and little data (see the video for P_0 and P_1 under camera and object movement). For distant objects, i.e., LODs which are cheap to render, a visibility analysis is unprofitable and we rely on conservative LOD criteria instead.

For complex triangle meshes, it becomes too costly to render the full geometry for the occlusion queries and to compute the projected area on the CPU. And we also cannot use coarse LODs for the occlusion queries together with fine LODs for display: their surfaces potentially intersect and the visibility determination is void. Note that we cannot use bounding volumes in many scenes as they might intersect with surrounding objects. Instead, we use an *envelope mesh*, a coarsely triangulated hull, tightly enclosing the complex object (Fig. 10) for the visibility analysis, and consequently, compute the clusterings for the envelope. Note that envelope meshes are used for the visibility determination only *after* the visible geometry has been rendered, i.e., they are rasterized and tested against the depth buffer that contains the visible geometry, but they never modify its content. This ensures that no potential occluder is enlarged.

For the envelope construction, we compute a distance field to the input mesh and use the Computational Geometry Algorithms Library [31] to triangulate an isosurface with a positive distance to the object's surface. We only need to verify that the envelope and the input mesh do not intersect. In this case, we increase the isovalue and redo the surface reconstruction. Progressive hulls [32] or offset surfaces [33] can also be used to create envelopes; however, we decided to use the CGAL surface mesh generator which yields more equilateral triangles, and thus, well-shaped clusters. Using envelopes is more involved with animated objects as the transformation of an object needs to be applied to the envelope as well. For this, we specify the envelope vertices in the local tangent spaces of the input mesh. This works well as long as animated object parts are not welded in the envelope, like the hand and the hip of the Polygirl in Fig. 10. Clearly, offset surfaces or a preparation of the model and manual intervention by an artist might be the best options to avoid these problems. Envelopes potentially self-intersect under strong deformation (similar problems appear with any displacement mapping method), but the expected impact on the visibility determination from this is minor. In initial tests, we experimented with even simpler bounding geometry, in particular, axis-aligned boxes from a standard bounding volume hierarchy or octree subdivision. The functionality and reliability of the visibility analysis was not affected thereby. However, avoiding intersections of the bounding geometry and the surrounding geometry (in dense scenes) requires tighter bounding volumes for the actual occlusion queries that have to be created for this purpose. Although this is practicable for rigid objects, we opt for envelope meshes that can be easily used with animated objects as well.

At rendering time, the feature extraction and classification are executed on the CPU. Only the fractional visibility determination is realized using one hardware occlusion query for each cluster of an object. Of course, this is only done for objects intersecting or inside the view frustum and not being already rendered at the lowest LOD due to the distance criterion. Whenever we detect a sudden increase in visible pixels, we switch back to a conservative LOD scheme: in this case, a close-by object might appear from behind a blocker and we want to make sure to render it with sufficient detail. Fig. 11 shows our example scene rendered with techniques typical for current video games, such as high dynamic range lighting and soft shadows. In order to hide latencies from the occlusion queries, we allow the analysis to lag one frame behind the rendering, which is acceptable if high frame rates can be maintained. Latencies are usually more crucial in "stop-and-wait" algorithms when the application waits for an occlusion query result. Exploiting temporal coherence

TABLE 1
Average Rendering Speed (Measured on an NVIDIA 8800GTX) for a Flyby through the Scene Shown in Fig. 11 and in the Accompanying Video

rendering resolution and technique	LOD	frames per second	avg. tris per frame
1920×1200	D	95.5	1670582
single pass lighting	D+VA	123.2	1085491
1920×1200	D	66.3	1670582
deferred shading	D+VA	78.2	1085491
1024×768	D	164.3	671375
single pass lighting	D+VA	189.1	417251
1024×768	D	150.4	671375
deferred shading	D+VA	172.8	417251

Timings are given for standard and deferred rendering with LOD selection based purely on a distance criterion (D), of course, taking screen resolution into account, and distance plus visibility analysis (D+VA). Average triangles per frame comprise the statues and busts only.

similar to [7], [8] reduces this overhead and might be applicable to our method as well.

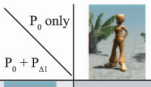
The preprocessing required for the LOD models took approximately 1 to 2 minutes depending on the model complexity. The most time-consuming step was the distance field computation, followed by the mesh decimation to obtain the levels of detail (using the quadric edge collapse decimation of MeshLab [34]); tessellation and clustering only takes a few seconds. We prepared eight meshes per object with varying detail (ranging from approximately 6,000 to 1,000,000 triangles); the envelope meshes consist of about 2,000 triangles each. Table 1 details the rendering performance of our method for the scene shown in Fig. 11. Note that aside from pure geometric LOD, other costly rendering techniques, such as normal map filtering [35] or displacement mapping [36], feature adjustable rendering quality and speed, thus, also providing a great potential for LOD rendering when used with our visibility analysis.

5.1.3 Evaluation of the Level of Detail Control

Rendering quality. Fig. 12 shows a qualitative analysis of the visible differences between rendering with full-resolution models and geometric LOD. Differences are primarily visible at object silhouettes and due to shading artifacts. Note that the latter can be reduced with commonly used, and inexpensive, normal mapping techniques. We also computed the PSNR to compare rendering with visibility analysis to rendering with distance-based LOD only. We consider the latter to be the "reference" instead of using full-resolution models that might cause aliasing. In a 90-second flyby of the Sponza scene (see Fig. 11 and the accompanying video), the LOD for at least one object was reduced because of detected potential masking in 75 percent of all frames. The PSNR was between 40.1 dB and 72.3 dB with an average of 56.5 dB.

Classification. In order to verify whether co-occurrence matrices provide additional information for the classification over the histogram alone, we run the flyby using P_0 only for classification. Instead of an MLP with 14 input and 14 hidden neurons, we trained a smaller network with only 2×4 neurons plus 7 output neurons; apart from that, the method remained unchanged. Table 2 illustrates the inferior

TABLE 2
Misclassification When Using the Histogram Only (as Opposed to P_0 and P_{Δ_1}); Only Occurred Configurations Are Shown

P_0 only / $P_0 + P_{\Delta 1}$						
	84	2	14	0	0	0
	1	76	12	0	11	0
	2	7	46	45	0	0
	0	0	7	71	22	0
	0	0	0	8	92	0
	0	0	0	0	0	100

Each row of the table lists the percentage of the respective configuration that is correctly or wrongly classified. For example, the second row shows that 76 percent of all configurations that have been classified as partial solid coverage using P_0 and P_{D_1} are correctly classified when using the histogram only (second row, second column); 12 percent are wrongly classified to partial see-through blocking (second row, third column), and 11 percent to full solid and see-through coverage (second row, fourth column). Note that a perfect classification would have resulted in 100 percent on the diagonal elements with all other entries being 0 percent.

classification resulting in wrongly detected masking. The results indicate that the "full see-through blocking" is classified quite reliable, which is to be expected as it can be easily detected using the histogram only (see Fig. 5c). The misclassification of this configuration to "full blocking with solid and see-through" is also tolerable in the LOD application. However, other configurations are wrongly classified to "full see-through blocking" although they do not (potentially) cause masking. Overall, the classification is also more vulnerable to inaccurate fractional visibilities (e.g., due to the disc-based intraobject occlusion approximation). Note that this test assumes that the analysis using P_0 and P_{Δ_1} is "correct." Obviously, there is no ground truth as the definition of the configurations is inherently fuzzy. However, we carefully inspected the classification manually in many synthetic and real-world scenes, where the visibility analysis yielded the expected results.

Limitations. We also examined more difficult cases for our method. To test highly concave objects, we treated a grid of 3×3 nearby Polygirl models as a single object to generate high intraobject self-occlusion. The disc approximation (Section 4.2) converges after 3 to 4 iterations in this case, and compares well to the exact visible disc areas. The classification was as reliable as for objects with less self-occlusion. Naturally, the capability to differentiate configurations depends on the cluster count and size. Changing the number of clusters may yield different classification results, e.g., because small visible (or occluded) object parts can be missed in the fractional visibility computation. In particular, the configuration shown in Fig. 1f is delicate with few clusters and might be classified as Figs. 1b or 1e.

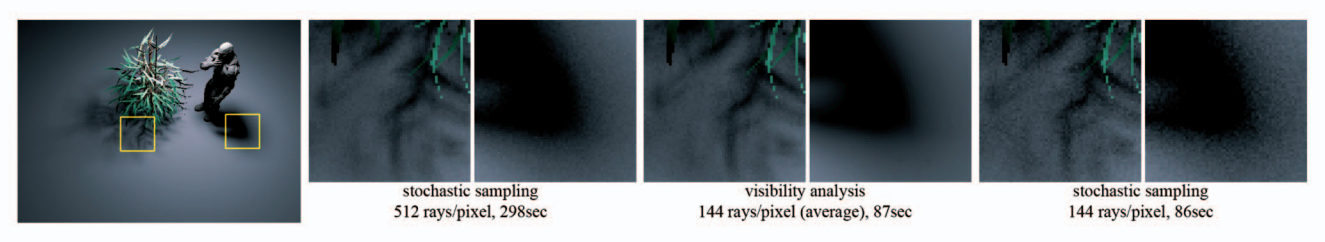


Fig. 13. The visibility analysis is used to decide after 64 samples whether to continue sampling adaptively or stochastically with up to 512 rays per pixel. The render time (image resolution 560×400 pixels) decreases significantly while the quality benefits from adaptive sampling in the penumbra region of the character model.

5.2 Applications in Offline Rendering

We also examined the use of the visibility analysis in the context of offline rendering, which we outline in this section.

Ray tracing. Strategies for area light source sampling in ray tracing aim to reduce variance, and thus, noise in images, and often use importance sampling to achieve this objective. However, if a priori information about the light source occlusion is available, then correlated [37] or adaptive [38] sampling can be selectively used to improve the result. We integrated the visibility analysis into PBRT [39] to analyze the visibility of the light source as seen from a surface point. For this, we split an area light source into 4×4 clusters and determine their fractional visibility with four shadow rays each. We experimentally used the classification results, shown in Fig. 2, to control further sampling: we used adaptive sampling for surfaces with partial coverage from solid blockers, and importance sampling otherwise. We applied the analysis to large or close area-light sources, where the cost for the visibility analysis is expected to amortize (see Fig. 13 for an example). Samples used for the classification can of course be reused for the lighting computation. In Fig. 13, the benefit of the visibility analysis for area light sources was a $3.4\times$ speedup over pure stochastic sampling with even higher quality. Obviously, this speedup is only possible when restricting the visibility analysis to situations where the lighting computation requires more than 64 rays to compute a noise-free image.

Area-to-area visibility. Many problems in computer graphics require the determination or classification of the visibility between surfaces. Fig. 14 shows a simple scene with two pairs of surfaces that exhibit different visibility configurations. In general, however, the mutual visibility of two polygons A and B may feature multiple visibility configurations according to our definition (see Fig. 15) at the same time. One possibility to extract the information necessary for decision making is to carry out multiple from-point classifications, which we describe for hierarchical radiosity (HR) refinement as an example. In HR, an oracle evaluates if a link between two surface patches which exchange energy has to be refined, and one or both patches have to be subdivided. Due to memory and time constraints, the tessellation cannot be arbitrarily fine and an important criterion for subdivision is mutual visibility, which is typically evaluated using ray casting (see [40] for an overview). A standard oracle [41] terminates refinement if two patches become totally invisible relative to each other; if they become totally visible, there is no need for further visibility tests, although further refinement may still need to occur. The visibility analysis can be used to decide upon refinement in case of partial coverage, which might be due to solid blockers (then a subdivision is necessary to capture shadow boundaries), or due to fine details, i.e., see-through blockers, which possibly cannot be captured even by the finest permitted tessellation. Note that this differentiation is related to the feature-based control of the visibility error in radiosity [42]. We integrated an experimental oracle into an (unoptimized) HR solver and perform 1 to 8 analyses (depending on the patch sizes) for each link. Each analysis is the same as a point-to-area analysis as described in the previous paragraph and computed for randomly chosen points located on both patches. If the analyses do not detect the same configuration, we subdivide the respective larger patch and repeat the visibility analysis for the newly created links until unambiguous configurations are detected, or until the maximum subdivision has been reached. We terminate refinement if see-through blocking is detected and details are too fine to be resolved within the tessellation budget, but we enforce it if partial solid blocking is detected. Besides direct illumination, the analysis is also very important for indirect light, which is usually transferred between larger patches. Note that the visibility tests can again be reused for form factor computation. The number of patches in Fig. 16 decreased by

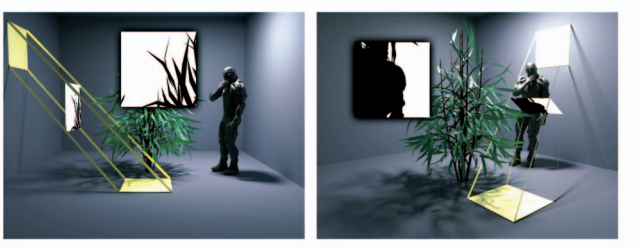


Fig. 14. Two pairs of surfaces (yellow polygons) that exhibit different visibility configurations: partial coverage from see-through blockers (left) and partial coverage from solid blockers (right).

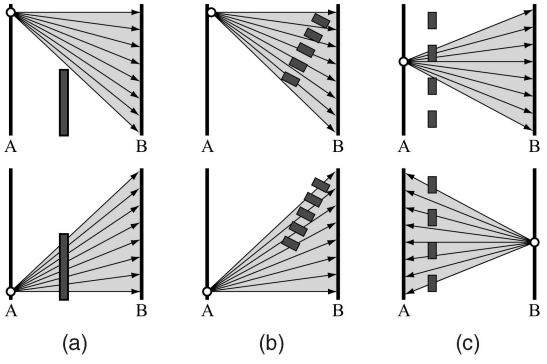


Fig. 15. Two polygons A and B may feature multiple visibility configurations simultaneously depending on the regarded point, e.g., full visibility and occlusion (a), or see-through and solid blocking (b) and (c).

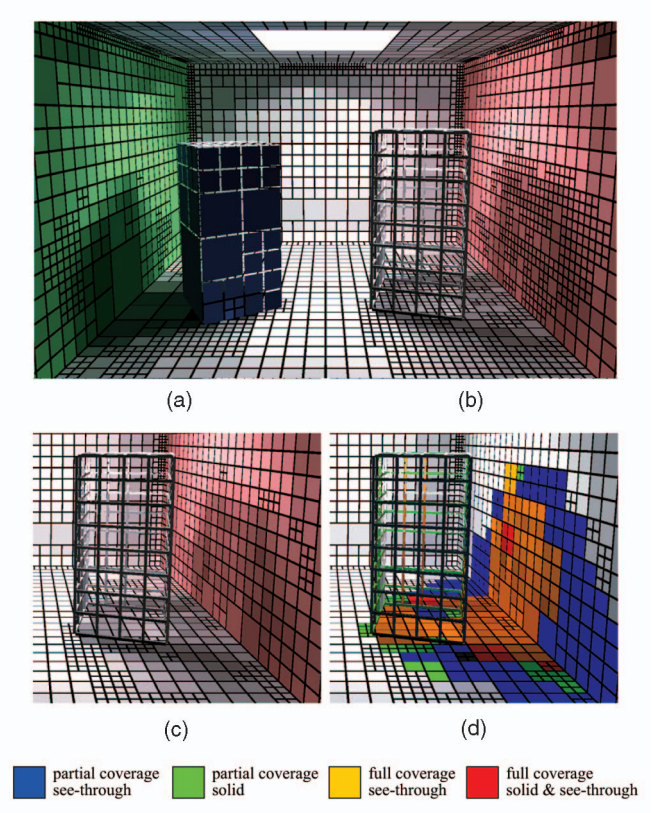


Fig. 16. Top: a typical HR refinement for our example scene with a blue solid box (a) and a box made of gratings (b). Bottom: the visibility analysis prevents refinement due to small features which cannot be resolved within a given tessellation budget (c); the predominant visibility configurations (with respect to the light source, i.e., for the direct lighting) are shown color-coded (d).

approximately 10 percent (from 7,717 to 6,971), the links by 22 percent, while the lighting solution looks very similar. Note that this moderate benefit is due to the simple geometry in this test scene. In a practical implementation, clustering and volumetric approximations [43] can be used to further accelerate the visibility analysis by reducing the number of cast rays. Fig. 16 shows the predominant visibility configurations of the scene surfaces with respect to the area light source.

6 Discussion and Future Work

Previous real-time techniques pursue the goal of efficiently determining if an object is visible or occluded (e.g., using coherent visibility queries or intelligently placed virtual occluders), while our method differentiates visibility configurations. The closest works in this spirit are hardly visible sets [10] focusing on occlusion culling only, and the perceptual rendering pipeline [1] which inspired our LOD algorithm. We believe that these and other recent approaches, e.g., [7], [8], are orthogonal and can be combined with our method.

During the analysis, the objects' clusters serve as probes for visibility determination. The differentiation between solid and see-through blockers, thus, depends on their image space size. To ensure a consistent analysis, we adapt the cluster size to the view (or light) distance (Section 5.1). Our examples indicate that the most important visibility configurations for our examples (Figs. 1a, 1b, 1c, and 1d) can be reliably classified with few clusters. However, an adaptive refinement using more clusters can be used to reveal further information at any time and enables a quantitative analysis of classification errors. Note that the classifiers and the LOD selection can be forced to a conservative behavior, for example, in a way such that the LOD is only reduced if the fractional visibility of all clusters is less than 30 percent.

Our experiments showed that neither the classification nor the differentiation of the visibility configurations according to our definition (shown in Fig. 1) improves when using more than 4 or 5 quantization levels for the fractional visibilities (less do not permit a clean differentiation of visibility, see-through, and occlusion). Furthermore, the CM size grows quadratically with the number of levels, thus CMs tend to become sparse when using few clusters, and of course, the classification also becomes more costly.

Generally speaking, our analysis determines visibility at a finer granularity, but it does extract more information than solely the degree of visibility. Methods for hierarchical occlusion culling, e.g., [7], [8], [11], determine visibility for hundreds to thousands of scene nodes and it could be beneficial to base culling strategies on identified visibility configurations, and thus, use the already present information on a higher level.

This work also raises further questions. We would like to use perceptual models to derive and validate metrics for LOD selection and ray-tracing-based sampling guided by the visibility configurations. The envelope generation for animated objects will be important for many real-time applications and is certainly a challenge on its own. We also plan to explore further applications for our visibility analysis within existing methods.

7 Conclusions

We presented a novel and efficient method for the classification of different visibility configurations and demonstrated its versatility and potential use in real-time and offline rendering methods for the analysis of from-point and area-to-area configurations. Based on a simple clustering approach, the feature extraction is derived from the established algorithms. The classification proved reliable in our test scenes and requires small training sets only. In conclusion, our visibility analysis is very simple to implement and can be easily integrated into, and combined with, existing methods. We believe that it has potential use in many algorithms that rely on visibility determination of some kind.

Acknowledgments

The author would like to thank Pierre Alliez for help with CGAL, and Christian Roessl for initial discussions. The Polygirl, bust, and statue models are provided courtesy of INRIA by the AIM@SHAPE Shape Repository. The Sponza model is courtesy of Marko Dabrovic, other models used in the example scenes and the accompanying video are courtesy of Crytek GmbH.

References

[1] G. Drettakis, N. Bonneel, C. Dachsbacher, S. Lefebvre, M. Schwarz, and I. Viaud-Delmon, "An Interactive Perceptual Rendering Pipeline Using Contrast and Spatial Masking," *Proc. Eurographics Symp. Rendering*, 2007.

[2] R.M. Haralick, K. Shanmugam, and I. Dinstein, "Textural Features for Image Classification," *IEEE Trans. Systems, Man, and Cybernetics,* vol. SMC-3, pp. 610-621, Nov. 1973.
[3] F.-A. Law and T.-S. Tan, "Preprocessing Occlusion for Real-Time Selective Refinement," *Proc. Symp. Interactive 3D Graphics (I3D '99),* pp. 47-53, 1999.
[4] T. Funkhouser, S. Teller, and D. Khorramabadi, "The UC Berkeley System for Interactive Visualization of Large Architectural Models," *Presence,* vol. 5, pp. 13-44, 1996.
[5] W.V. Baxter, A. Sud, N.K. Govindaraju, and D. Manocha, "GigaWalk: Interactive Walkthrough of Complex Environments," *Proc. 13th Eurographics Workshop Rendering (EGRW '02),* pp. 203-214, 2002.
[6] S.-E. Yoon, B. Salomon, and D. Manocha, "Interactive View-Dependent Rendering with Conservative Occlusion Culling in Complex Environments," *Proc. 14th IEEE Visualization (VIS '03),* pp. 163-170, 2003.
[7] J. Bittner, M. Wimmer, H. Piringer, and W. Purgathofer, "Coherent Hierarchical Culling: Hardware Occlusion Queries Made Useful," *Computer Graphics Forum,* vol. 23, no. 3, pp. 615-624, 2004.
[8] O. Mattausch, J. Bittner, and M. Wimmer, "CHC++: Coherent Hierarchical Culling Revisited," *Computer Graphics Forum,* vol. 27, no. 3, pp. 221-230, 2008.
[9] E. Eisemann and X. Décoret, "Visibility Sampling on GPU and Applications," *Computer Graphics Forum,* vol. 26, no. 3, pp. 535-544, 2007.
[10] C. Andújar, C. Saona-Vázquez, I. Navazo, and P. Brunet, "Integrating Occlusion Culling and Levels of Detail through Hardly-Visible Sets," *Computer Graphics Forum,* vol. 19, no. 3, pp. 499-506, 2000.
[11] J.P. Charalambos, J. Bittner, M. Wimmer, and E. Romero, "Optimized Hlod Refinement Driven by Hardware Occlusion Queries," *Proc. Third Int'l Symp. Visual Computing (ISVC '07),* pp. 106-117, 2007.
[12] P. Wonka, M. Wimmer, K. Zhou, S. Maierhofer, G. Hesina, and A. Reshetov, "Guided Visibility Sampling," *ACM Trans. Graphics,* vol. 25, no. 3, pp. 494-502, 2006.
[13] E. Zhang and G. Turk, "Visibility-Guided Simplification," *Proc. Conf. Visualization 2002 (VIS '02),* pp. 267-274, 2002.
[14] J. Bittner and P. Wonka, "Visibility in Computer Graphics," *Environment and Planning B: Planning and Design,* vol. 30, no. 5, pp. 729-756, 2003.
[15] D. Cohen-Or, Y.L. Chrysanthou, C.T. Silva, and F. Durand, "A Survey of Visibility for Walkthrough Applications," *IEEE Trans. Visualization and Computer Graphics,* vol. 9, no. 3, pp. 412-431, July-Sept. 2003.
[16] D. Luebke, B. Watson, J.D. Cohen, M. Reddy, and A. Varshney, *Level of Detail for 3D Graphics.* Elsevier Science, 2002.
[17] V.N. Vapnik, *The Nature of Statistical Learning Theory.* Springer-Verlag New York, Inc., 2001.
[18] S. Liu, R.R. Martin, F.C. Langbein, and P.L. Rosin, "Segmenting Periodic Reliefs on Triangle Meshes," *Proc. IMA Conf. Math. Surfaces,* pp. 290-306, 2007.
[19] C.M. Bishop, *Neural Networks for Pattern Recognition,* first ed. Oxford Univ. Press, Jan. 1996.
[20] S.R. Gunn, "Support Vector Machines for Classification and Regression," technical report, Univ. of Southhampton, 1998.
[21] N. Barabino, M. Pallavicini, A. Petrolini, M. Pontil, and A. Verri, "Support Vector Machines vs Multi-Layer Perceptrons in Particle Identification," *Proc. European Symp. Artificial Neural Networks 1999,* pp. 257-262, 1999.
[22] K.P. Bennett and C. Campbell, "Support Vector Machines: Hype or Hallelujah?" *SIGKDD Explorations Newsletter,* vol. 2, no. 2, pp. 1-13, 2000.
[23] Sebastian Nowozin "Libperceptronnetwork—Neural Network Library," http://user.cs.tu-berlin.de/nowozin/libpn/, 2009.
[24] T. Engelhardt and C. Dachsbacher, "Granular Visibility Queries on the GPU," *Proc. 2009 Symp. Interactive 3D Graphics and Games (I3D '09),* pp. 161-167, 2009.
[25] P.V. Sander, Z.J. Wood, S.J. Gortler, J. Snyder, and H. Hoppe, "Multi-Chart Geometry Images," *Proc. Symp. Geometry Processing,* pp. 146-155, 2003.
[26] M. Bunnell, "Dynamic Ambient Occlusion and Indirect Lighting," *GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation,* pp. 636-648, Addison-Wesley Professional, 2005.
[27] J. El-Sana, N. Sokolovsky, and C.T. Silva, "Integrating Occlusion Culling with View-Dependent Rendering," *Proc. Conf. Visualization (VIS '01),* pp. 371-378, 2001.
[28] M. Ramasubramanian, S.N. Pattanaik, and D.P. Greenberg, "A Perceptually-Based Physical Error Metric for Realistic Image Synthesis," *Proc. ACM SIGGRAPH,* pp. 73-82, 1999.
[29] S. Daly, "The Visible Differences Predictor: An Algorithm for the Assessment of Image Fidelity," *Digital Images and Human Vision,* pp. 179-206, The MIT Press, 1993.
[30] R. Mantiuk, S. Daly, K. Myszkowski, and H.-P. Seidel, "Predicting Visible Differences in High Dynamic Range Images—Model and Its Calibration," *Proc. Human Vision and Electronic Imaging X, IS&T/SPIE's 17th Ann. Symp. Electronic Imaging,* pp. 204-214, 2005.
[31] CGAL "Computational Geometry Algorithms Library," http://www.cgal.org, 2010.
[32] P.V. Sander, X. Gu, S.J. Gortler, H. Hoppe, and J. Snyder, "Silhouette Clipping," *Proc. SIGGRAPH,* pp. 327-334, 2000.
[33] D. Pavić and L. Kobbelt, "High-Resolution Volumetric Computation of Offset Surfaces with Feature Preservation," *Computer Graphics Forum,* vol. 27, no. 2, pp. 165-174, 2008.
[34] MeshLab, http://meshlab.sourceforge.net, 2010.
[35] C. Han, B. Sun, R. Ramamoorthi, and E. Grinspun, "Frequency Domain Normal Map Filtering," *ACM Trans. Graphics,* vol. 26, no. 3, p. 28, 2007.
[36] X. Wang, X. Tong, S. Lin, S.-M. Hu, B. Guo, and H.-Y. Shum, "Generalized Displacement Maps," *Proc. Eurographics Symp. Rendering Techniques,* pp. 227-234, 2004.
[37] L. Szécsi, M. Sbert, and L. Szirmay-Kalos, "Combined Correlated and Importance Sampling in Direct Light Source Computation and Environment Mapping," *Computer Graphics Forum,* vol. 23, no. 3, pp. 585-594, 2004.
[38] A.J.F. Kok and F.W. Jansen, "Adaptive Sampling of Area Light Sources in Ray Tracing Including Diffuse Interreflection," *Computer Graphics Forum,* vol. 11, no. 3, pp. 289-298, 1992.
[39] M. Pharr and G. Humphreys, *Physically Based Rendering: From Theory to Implementation.* Morgan Kaufmann, 2004.
[40] M. Feixas, J. Rigau, P. Bekaert, and M. Sbert, "Information-Theoretic Oracle Based on Kernel Smoothness for Hierarchical Radiosity," *Proc. Eurographics Short Presentations,* pp. 325-333, 2002.
[41] P. Hanrahan, D. Salzman, and L. Aupperle, "A Rapid Hierarchical Radiosity Algorithm," *Proc. ACM SIGGRAPH,* vol. 25, no. 4, pp. 197-206, 1991.
[42] F. Sillion and G. Drettakis, "Feature-Based Control of Visibility Error: A Multi-Resolution Clustering Algorithm for Global Illumination," *Proc. ACM SIGGRAPH,* pp. 145-152, 1995.
[43] F.X. Sillion, "A Unified Hierarchical Algorithm for Global Illumination with Scattering Volumes and Object Clusters," *IEEE Trans. Visualization and Computer Graphics,* vol. 1, no. 3 pp. 240-254, Sept. 1995.

Carsten Dachsbacher received the diploma in computer science and the PhD degree in computer graphics from the University of Erlangen-Nuremberg. He is a full professor at the Karlsruhe Institute of Technology. Prior to joining KIT, he was an assistant professor at the Visualization Research Center, University Stuttgart, Germany, a researcher at INRIA, Sophia Antipolis, France, within a Marie-Curie Fellowship, and a visiting professor at Konstanz University. His research focuses on real-time computer graphics, interactive global illumination, GPGPU, and perceptual rendering.