Shadow Caster Culling for Efficient Shadow Mapping

Jiří Bittner* Oliver Mattausch† Ari Silvennoinen‡ Michael Wimmer†

*Czech Technical University in Prague§ † Vienna University of Technology ‡Umbra Software

Figure 1: *(left) A view of a game scene rendered with shadows. (right) Shadow map with shadow casters rendered using a naive application of occlusion culling in the light view (gray), and shadow casters rendered using our new method (orange). The view frustum corresponding to the left image is shown in yellow. Note that for this view our shadow caster culling provided a 3x reduction in the number of rendered shadow casters, leading to a 1.5x increase in total frame rate. The scene is a part of the Left 4 Dead 2 game (courtesy of Valve Corp.).*

Abstract

We propose a novel method for efficient construction of shadow maps by culling shadow casters which do not contribute to visible shadows. The method uses a mask of potential shadow receivers to cull shadow casters using a hierarchical occlusion culling algorithm. We propose several variants of the receiver mask implementations with different culling efficiency and computational costs. For scenes with statically focused shadow maps we designed an efficient strategy to incrementally update the shadow map, which comes close to the rendering performance for unshadowed scenes. We show that our method achieves 3x-10x speedup for rendering large city like scenes and 1.5x-2x speedup for rendering an actual game scene.

CR Categories: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism— [I.3.5]: Computer Graphics—Computational Geometry and Object Modeling

Keywords: shadow maps, occlusion culling, real time rendering

1 Introduction

Shadow mapping is a well-known technique for rendering shadows in 3D scenes [Williams 1978]. With the rapid development of graphics hardware, shadow maps became an increasingly popular technique for real-time rendering. A variety of techniques have been proposed, most of which focus on increasing the quality of the rendered image given a maximum resolution of the shadow map [Stamminger and Drettakis 2002; Wimmer et al. 2004; Lloyd et al. 2008]. Given sufficiently high resolution, these methods achieve shadows of very high quality.

For highly complex scenes, however, shadow mapping can become very slow. One problem is the rendering of the main camera view. This problem has been addressed by occlusion culling, which aims to quickly cull geometry which does not contribute to the image [Cohen-Or et al. 2003]. However, just rendering the camera view quickly does not guarantee high frame rates when shadow mapping is used. In particular, the overhead of creating the shadow map is not reduced. Thus, rendering the shadow map can easily become the bottleneck of the whole rendering pipeline as it may require rendering a huge amount of shadow casters at a very high resolution.

A naive solution to this problem would again use occlusion culling to reduce the amount of rendered shadow casters when rendering the shadow map. However it turns out that for common complex scenes like terrains or cities, the lights are set up so that they have a global influence on the whole scene (e.g., sun shining over the city). For such scenes, occlusion culling from the light view will not solve the problem, as the depth complexity of the light view is rather low and thus not much geometry will be culled. Thus even if occlusion culling is used for both the camera view and the light view, we might end up rendering practically all geometry contained in the intersection of the view frustum and the light frustum, and many rendered shadow casters will not contribute to the shadows in the final image.

In this paper, we propose a method for solving this problem by using the knowledge of visibility from the camera for culling the shadow casters. First, we use occlusion culling for the camera view to identify visible shadow receivers. Second, when rendering into the shadow map, we use only those shadow casters which cast shadows on visible shadow receivers. All other shadow casters are culled. For both the camera and the light views, we use an improved version of the coherent hierarchical culling algorithm (CHC++) [Mattausch et al. 2008], which provides efficient schedul-

*e-mail:bittner@fel.cvut.cz
†e-mail:{matt|wimmer}@cg.tuwien.ac.at
‡e-mail:ari@umbrasoftware.com
§Faculty of Electrical Engineering

ing of occlusion queries based on temporal coherence. We show that this method brings up to an order of magnitude speedup compared to naive application of occlusion culling to shadow mapping (see Figure 1). As a minor contribution we propose a simple method for incrementally updating a statically focused shadow map for scenes with moving objects.

2 Related Work

Shadow mapping was originally introduced by Williams [1978], and has meanwhile become the de-facto standard shadow algorithm in real-time applications such as computer games, due to its speed and simplicity. For a discussion of different methods to increase the quality of shadow mapping, we refer to a recent survey [Scherzer et al. 2010]. In our approach in particular, we make use of light space perspective shadow mapping (LiSPSM) [Wimmer et al. 2004] in order to warp the shadow map to provide higher resolution near the viewpoint, and cascaded shadow maps [Lloyd et al. 2006], which partition the shadow map according to the view frustum with the same aim.

There has been surprisingly little work on shadow mapping for large scenes. One obvious optimization which also greatly aids shadow quality is to focus the light frustum on the receiver frustum [Brabec et al. 2002]. As a result, many objects which do not cast shadows on objects in the view frustum are culled by frustum culling in the shadow map rendering step. Lauritzen et al. [2010] have reduced the focus region to the subset of visible view samples in the context of optimizing the splitting planes for cascaded shadow maps.

Govindaraju et al. [2003] were the first to use occlusion culling to reduce the amount of shadow computations in large scenes. Their algorithm computes object-precision shadows mostly on the CPU and used occlusion queries on a depth map to provide the tightest possible bound on the shadow polygons that have to be rendered. While their algorithm also renders shadow receivers to the stencil buffer to identify potential shadow casters, this was done *after* a depth map had already been created, and thus this approach is unsuitable for shadow mapping acceleration. In our algorithm, on the other hand, the creation of the depth map itself is accelerated, which is a significant benefit for many large scenes. Lloyd et al. [2004] later extended shadow caster culling by shadow volume clamping, which significantly reduced the fillrate when rendering shadow volumes. Their paper additionally describes a method for narrowing the stencil mask to shadow receiver fragments that are detected to lie in shadow. Décoret [2005] proposed a different method for shadow volume clamping as one of the applications for N-buffers. Unlike Lloyd et al. [2004] Décoret enhances the stencil mask by detecting receiver fragments which are visible from the camera. The methods for shadow volume culling and clamping have been extended by techniques allowing for more efficient GPU implementations[Eisemann and Décoret 2006; Engelhardt and Dachsbacher 2009].

While our method shares the idea of using visible receivers to cull shadow casters proposed in the above mentioned techniques, in these methods the shadow map served only as a proxy to facilitate shadow volume culling and clamping, and the efficient construction of the shadow map itself has not been addressed. When shadow mapping is used instead of more computationally demanding shadow volumes, the bottleneck of the computation moves to the shadow map construction itself. This bottleneck, which we address in the paper, was previously left intact.

3 Algorithm Outline

The proposed algorithm consists of four main steps:

(1) Determine shadow receivers

(2) *Create a mask of shadow receivers*

(3) *Render shadow casters using the mask for culling*

(4) Compute shading

The main contribution of our paper lies in steps (2) and (3). To give a complete picture of the method we briefly outline all four steps of the method.

Determine shadow receivers The potential shadow receivers for the current frame consist of all objects visible from the camera view, since shadows on invisible objects do not contribute to the final image. Thus we first *render* the scene from the point of view of the camera and determine the *visibility* status of the scene objects in this view.

To do both of these things efficiently, we employ hierarchical occlusion culling [Bittner et al. 2004; Guthe et al. 2006; Mattausch et al. 2008]. In particular we selected the recent CHC++ algorithm [Mattausch et al. 2008] as it is simple to implement and provides a good basis for optimizing further steps of our shadow caster culling method. CHC++ provides us with a visibility classification of all nodes of a spatial hierarchy. The potential shadow receivers correspond to visible leaves of this hierarchy.

Note that in contrast to simple shadow mapping, our method requires a render pass of the camera view before rendering the shadow map. However, such a rendering pass is implemented in many rendering frameworks anyway. For example, deferred shading, which has become popular due to recent advances in screen-space shading effects, provides such a pass, and even standard forward-renderers often utilize a depth-prepass to improve pixel-level culling.

Create a mask of shadow receivers The crucial step of our algorithm is the creation of a mask in the light view which represents shadow receivers. For a crude approximation, this mask can be formed by rendering bounding boxes of visible shadow receivers in the *stencil buffer* attached to the shadow map. In the next section, we propose several increasingly sophisticated methods for building this mask, which provide different accuracy vs. complexity tradeoffs.

Render shadow casters We use hierarchical visibility culling using hardware occlusion queries to speed up the shadow map rendering pass. In addition to depth-based culling, we use the shadow receiver mask to cull shadow casters which do not contribute to visible shadows. More precisely, we set up the *stencil test* to discard fragments outside the receiver mask. Thus, the method will only render shadow casters that are visible from the light source *and* whose projected bounding box at least partly overlaps the receiver mask.

Compute shading The shadow map is used in the final rendering pass to determine light source visibility for each shaded fragment.

4 Shadow Caster Culling

The general idea of our method is to create a mask of visible shadow receivers, i.e., those objects determined as visible in the first camera rendering pass. This mask is used in the shadow map rendering pass

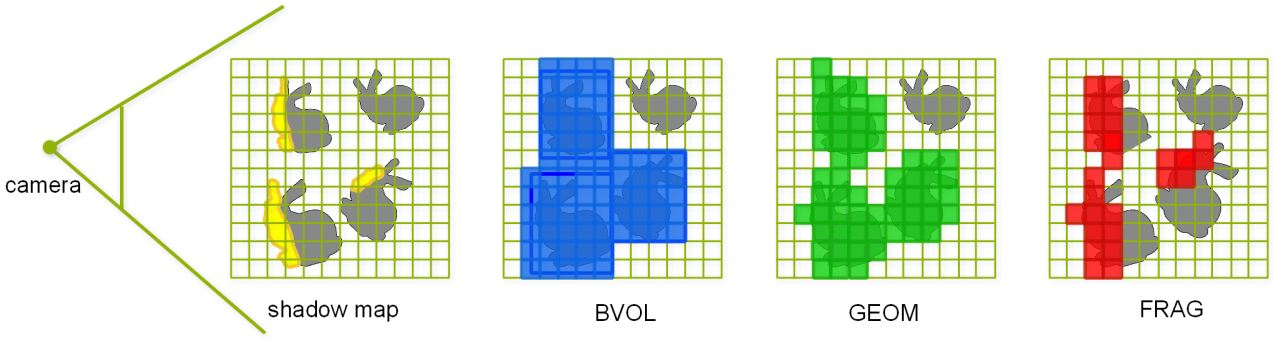


Figure 2: *Visualization of different receiver masks. On the left there is a light space view of the scene showing surfaces visible from the camera in yellow. On the right three different receiver masks are shown. Note that the invisible object on the top right does not contribute to any of the receiver masks.*

together with the depth information to cull shadow casters which do not contribute to the visible shadows. Culling becomes particularly efficient if it employs a spatial hierarchy to quickly cull large groups of shadow casters, which is the case for example in the CHC++ algorithm.

There are various options on how the receiver mask can be implemented. We describe four noteworthy variants which differ in implementation complexity as well as in culling efficiency. The creation of the mask happens in a separate pass before rendering the shadow map, but may already generate parts of the shadow map itself as well, simplifying the subsequent shadow map rendering pass.

We start our description with a simple version of the mask, which is gradually extended with more advanced culling. As we shall see later in the results section, in most cases the more accurate mask provides better results, although there might be exceptions for particular scene and hardware configurations.

4.1 Bounding Volume Mask

The most straightforward way to create the mask is to rasterize *bounding volumes* of all visible shadow receivers into the stencil buffer attached to the shadow map.

Such a mask will generally lead to more shadow casters being rendered than actually necessary. The amount of overestimation depends mostly on how fine the subdivision of scene meshes into individual objects is, and how tightly their bounding volumes fit. If an individual scene object has too large spatial extent, its projection might cover a large portion in the shadow mask even if only a small part of the mesh is actually visible. An illustration of the bounding volume mask is shown in Figure 2 (BVOL).

4.2 Geometry Mask

In order to create a tighter shadow receiver mask, we can rasterize the *actual geometry* of the visible shadow receivers instead of their bounding volumes into the stencil buffer.

While rendering the visible shadow receivers, we also write the depth values of the shadow receivers into the shadow map. This has the advantage that these objects have thus already been rendered into the shadow map and can be skipped during the subsequent shadow map rendering pass, which uses occlusion queries just for the remaining objects. An illustration of the bounding volume mask is shown in Figure 2 (GEOM).

In some scenes many shadow receivers also act as shadow casters and thus this method will provide a more accurate mask at almost no cost. However if most shadow receivers do not act as shadow casters (i.e., most of the scene is shadowed by objects outside of the camera view), the shadow mask creation can become rather expensive, since most parts of the shadow map will be replaced by other objects which act as real shadow casters. Consequently, the resources for rendering these shadow receivers into the shadow map were wasted, as they would have been culled by the occlusion culling algorithm otherwise.

4.3 Combined Geometry and Bounding Volume Mask

In order to combine the positive aspects of the two previously described methods, we propose a technique which decides whether a shadow receiver should fill the mask using its bounding box or its geometry. The decision is based on the estimation of whether the visible shadow receiver will simultaneously act as a shadow caster. Such objects are rendered using geometry (with both stencil and depth, as such objects must be rendered in the shadow map anyway), while all other visible receivers are rendered using bounding boxes (with only stencil write).

The estimation uses temporal coherence: if a shadow receiver was visible in the shadow map in the previous frame, it is likely that this receiver will stay visible in the shadow map and therefore also act as a shadow caster in the current frame.

Again, all objects that have already been rendered into the shadow map using depth writes can be skipped in the shadow map rendering pass. However, in order to estimate the receiver visibility in the subsequent frame, visibility needs to be determined even for these skipped objects. Therefore we include them in the hierarchical occlusion traversal and thus they may have their bounding volumes rasterized during the occlusion queries.

4.4 Fragment Mask

Even the most accurate of the masks described above, the geometry mask, can lead to overly conservative receiver surface approximations. This happens when a receiver object spans a large region of the shadow map even though most of the object is hidden in the camera view. An example would be a single ground plane used for a whole city, which would always fill the whole shadow receiver mask and thus prevent any culling. While such extreme cases can be avoided by subdividing large receiver meshes into smaller objects, it is still beneficial to have a tighter mask which is completely independent of the actual object subdivision.

Method	accuracy	fill rate	transform rate
BVOL	low	high	low
GEOM	medium	medium	medium
GEOM+BVOL	medium	low–medium	low
FRAG	high	medium	medium

Table 1: *Summary of the masking techniques and indicators of their main attributes. BVOL – bounding box, GEOM – geometry mask, GEOM+BVOL – combined geometry and bounding volume mask, FRAG – fragment mask. Note that the FRAG method either relies on availability of direct stencil writes, or requires slightly more effort in mask creation and culling phases as described in Section 4.4.*

Fortunately, we can further refine the mask by using a fragment level receiver visibility tests for all shadow receivers where the full geometry is used for the mask. While creating the mask by rasterizing the geometry in light space, we project each fragment back to view space and test the projected fragment visibility against the view space depth buffer. If the fragment is invisible, it is on a part of the shadow receiver that is hidden, and is thus not written to the mask, otherwise, the mask entry for this fragment is updated. The shadow map depth needs to be written in *both* cases, since even if the fragment is not visible in the camera view, it might still be a shadow caster that shadows another visible shadow receiver.

A similar mask construction approach was proposed by Décoret [2005] for culling and clamping shadow volumes of shadow casters. Décoret used a pair of *litmaps* to obtain minimum and maximum depths of visible receivers per texel in order to clamp shadow volumes. In his method all scene objects are processed twice in the mask construction phase and the method does not use the knowledge of visible shadow receivers. In contrast, we use a single binary mask of visible receiver fragments and combine the construction of the mask with simultaneous rendering of depths of objects which act both as shadow receivers and shadow casters, which is very important to reduce the bottleneck created by the shadow map construction.

The mask constructed using the fragment level visibility tests is pixel accurate, i.e., only locations that can receive a visible shadow are marked in the mask (up to the bias used for the fragment depth comparison). Note that the fragment visibility test corresponds to a "reversed" shadow test, i.e., the roles of the camera and the light are swapped: instead of testing the visibility of the camera view fragment with respect to the shadow map, we test visibility of the fragment rendered into the shadow map with respect to the camera using the camera view depth buffer. An illustration of the bounding volume mask is shown in Figure 2 (FRAG).

The implementation of this approach faces the problem that the stencil mask needs to be updated based on the outcome of the shader. This functionality is currently not widely supported in hardware. Therefore, we implement the fragment receiver mask as an additional texture render target instead of using the stencil buffer. This requires an additional texture fetch and a conditional fragment discard based on the texture fetch while rendering the occlusion queries in the shadow map rendering pass, which incurs a small performance penalty compared to a pure stencil test.

The summary of different receiver masking methods is given in Table 1. The illustration of the culling efficiency of different masks is shown in Figure 3. Note that apart from the BVOL method, all other methods initiate the depth buffer values for the light view with depths of visible receivers.

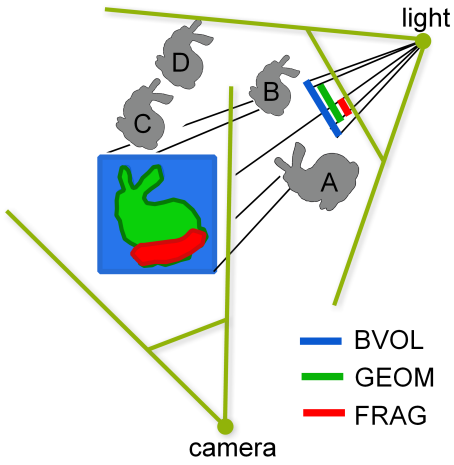


Figure 3: *2D illustration of the culling efficiency of different masking strategies. The figure shows an example of the scene with a shadow receiver object and different types of shadow receiver masks. Object A is always rendered as it intersects all masks, object B is culled only by the FRAG mask, C is culled by depth test (occluded by B) and by the FRAG and GEOM masks, and finally D is culled by all types of masks.*

5 Further Optimizations

This section contains several optimization techniques which can optionally support the proposed receiver masking in further enhancing the performance of the complete rendering algorithm.

5.1 Incremental Shadow Map Updates

If the shadow map is statically focused, we can extend the approach by restricting the shadow receiver mask only to places where a potential change in shadow can happen. We can do so by building the shadow receiver mask only from objects involved in dynamic changes. In particular we render all moving objects into the shadow receiver mask in their previous and current positions. This pass is restricted only to dynamic objects which are either visible in this or the previous frame.

If only a fraction of the scene objects is transformed, the shadow receiver mask becomes very tight and the overhead for shadow map rendering becomes practically negligible. However, this optimization is only useful if a static shadow map brings sufficient shadow quality compared to a shadow map focused on the view frustum.

Note that in order to create the stencil mask and simultaneously clear the depth buffer in a selective fashion, we reverse the depth test (only fragments with greater depth pass) and set the fragment depth to 1 in the shader while rendering the bounding boxes of moving objects.

5.2 Visibility-Aware Shadow Map Focusing

Shadow maps are usually "focused" on the view frustum in order to increase the available shadow map resolution. If visibility from the camera view is available, focusing can be improved further [Lauritzen et al. 2010]: The light frustum can then be computed by calculating the convex hull of the light source and the set of bounding boxes of the *visible* shadow receivers. Note that unlike the method of Lauritzen et al. [2010], our focusing approach does not require a view sample analysis on the GPU, but uses the readily available visibility classification from the camera rendering pass.

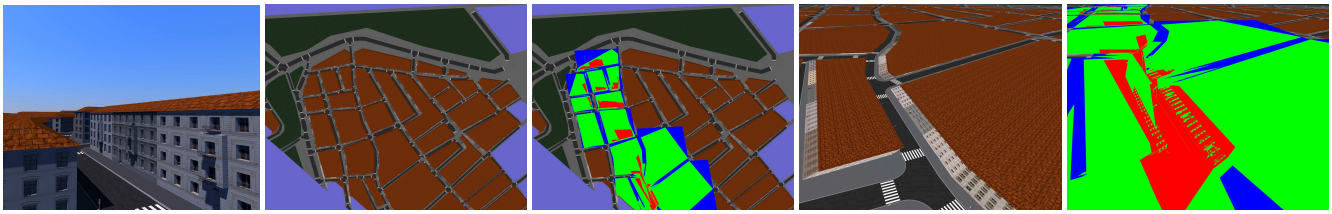


Figure 4: *This figure shows a viewpoint in the Vienna scene, the corresponding light view, and a visualization of the different versions of receiver masks where blue indicates the bounding volume mask, green the geometry mask, and red the fragment mask.*

This simple method effectively increases the resolution of the shadow map, as it is focused only on the visible portion of the scene. However this technique also has a drawback: if a large visibility change occurs in the camera view, the focus of the light view changes dramatically from one frame to the next, which can be perceived as temporal alias or popping of the shadow. This can easily happen in scenes in which a distant object suddenly pops up on the horizon such as a tall building, flying aircraft or a mountain.

5.3 CHC++ Optimizations

Our new shadow culling method works with any occlusion culling algorithm that allows taking the stencil mask into account. We use coherent hierarchical culling (CHC++ [Mattausch et al. 2008]) to implement hierarchical occlusion queries both for rendering the camera view and for rendering the shadow map. For the shadow map, occlusion queries can be set up so that they automatically take into account the shadow receiver mask stored in the stencil buffer. If the receiver mask is stored in a separate texture (as for the fragment mask), a texture lookup and conditional fragment discard is necessary in the occlusion culling shader.

We also propose a few modifications to the CHC++ algorithm, which should be useful in most applications using CHC++. Our recent experiments indicated that for highly complex views the main performance hit of CHC++ comes from the idle time of the CPU, when the algorithm has to wait for query results of previously invisible nodes. The reason is that previously invisible nodes, which sometimes generate further queries leading to further wait time, can be delayed due to batching, whereas queries for previously visible nodes, whose result is only of interest in the next frame, are issued immediately whenever the algorithm is stalled waiting for a query result.

Therefore, it turns out to be more beneficial to use waiting time to issue queries for *previously invisible* nodes instead, starting these queries as early as possible. Queries for the previously visible nodes, on the other hand, are issued *after* the hierarchy traversal is finished, and their results are fetched just before the traversal in the *next* frame. Between these two stages we apply the shading pass with shadow map lookups, which constitutes a significant amount of work to avoid any stalls.

Another very simple CHC++ modification concerns the handling of objects that enter the view frustum. If such an object has been in the view frustum before, instead of setting it to invisible, it inherits its previous classification. This brings benefits especially for the camera view for frequent rotational movements.

6 Results and Discussion

6.1 Test Setup

We implemented our algorithms in OpenGL and C++ and evaluated them using a GeForce 480 GTX GPU and a single Intel Core i7 CPU 920 with 2.67 GHz. For the camera view render pass we used a 800×600 32-bit RGBA render target (to store color and depth) and a 16-bit RGB render target (to store the normals), respectively. For the light view render pass we used a 32-bit depth-stencil texture and an 8-bit RGB color buffer with varying sizes. The application uses a deferred shading pipeline for shadowing.

City scenes exhibit the scene characteristics which our algorithm is targeted at – large, open scenes with high cost for shadow mapping. Hence we tested our algorithm in three city environments: a model of Manhattan, the ancient town of Pompeii, and the town of Vienna (refer to Table 3 for detailed information). All scenes were populated with various scene objects: Manhattan was populated with 1,600 cars, Pompeii with 200 animated characters, and Vienna with several street objects and trees. For Manhattan (Pompeii) we created 30^2 (40^2) floor tiles in order to have sufficient granularity for the object-based receiver mask methods. We measured our timings with a number of predefined walkthroughs. Figure 4 shows the different receiver masks in a real scene.

Abbreviation	Method
VFC	view frustum culling
REF	CHC++ occlusion culling (main reference)
FOCUS	REF + visibility-aware focusing
CHC++-OPT	optimized CHC++
INCR	incremental shadow map updates
UNSHAD	main render pass without shadow mapping

Table 2: *Abbreviations used in the plots (also refer to Table 1 for the variants of receiver masks).*

We compare our methods mainly against a reference method (REF) that uses the original CHC++ algorithm for *both* the camera view and the light view. We also show simple view frustum culling (VFC) and a minor modification of the REF method which uses visibility-aware focusing (FOCUS). Note that apart from the FOCUS method, all other tested methods do not use visibility-aware focusing. The tested methods are summarized in Table 2.

6.2 Receiver Mask Performance and Shadow Map Resolution

Table 4 shows average frame times for the tested scenes and different shadow map resolutions. As can be seen, receiver masking is faster than REF and FOCUS for all scenes and parameter combinations. There is a tendency that our algorithm brings slightly higher speedup for uniform shadow mapping than for LiSPSM. This has

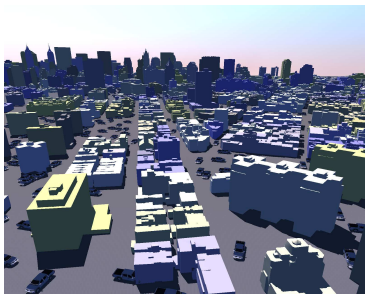
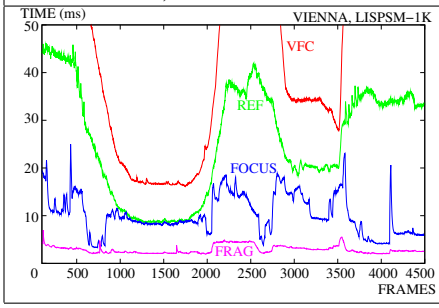
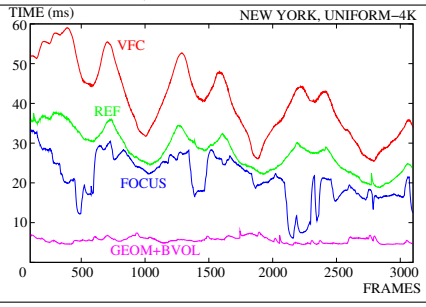
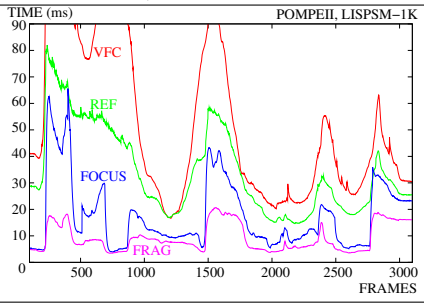
Vienna	Manhattan	Pompeii
		
12,685,693 vertices	104,960,856 vertices	63,141,390 vertices
22,464 objects	12,836 objects	11,097 objects
20,240 BVH nodes	6,471 BVH nodes	14,945 BVH nodes
		

Table 3: *Statistics for the scenes and characteristic walkthroughs.*

SM type	LISPSM			UNIFORM		
Shadow size	1K	2K	4K	1K	2K	4K
Scene	Vienna					
REF	21.6	22.3	22.4	28.7	28.7	29.2
FOCUS	9.1	9.3	9.3	10.8	11.0	11.4
GEOM+BVOL	4.9	4.9	**5.0**	4.9	5.0	**5.1**
FRAG	**2.9**	**3.5**	6.1	**2.9**	**3.4**	5.9
Scene	Manhattan					
REF	36.6	35.9	35.1	44.2	43.9	41.8
FOCUS	28.9	28.1	27.9	31.9	30.7	30.0
GEOM+BVOL	5.6	5.7	**6.6**	5.6	5.7	**6.5**
FRAG	**4.5**	**5.4**	9.0	**4.5**	**5.3**	8.6
Scene	Pompeii					
REF	34.8	34.8	39.2	40.5	40.2	44.8
FOCUS	17.4	17.4	20.6	18.2	18.3	21.2
GEOM+BVOL	11.2	11.2	13.4	11.4	11.3	13.0
FRAG	**9.7**	**10.0**	**13.2**	**9.8**	**10.2**	**12.5**

Table 4: *Average frame times for the tested scenes (in ms). The time for the best method for the given scene and shadow map resolution is marked in bold.*

two reasons: First, LiSPSM focuses much better on the visible parts of the view frustum. Second, LiSPSM trades the quality increase in the near field with quality loss in the far field regions (depending on the settings of the n parameter). Therefore it can happen that small objects in the background are simply not represented in the shadow map and will therefore be culled. Surprisingly, the frame times of the REF and FOCUS methods slightly decrease for increasing shadow map sizes in Manhattan. This unintuitive behavior might be connected with the heavy transform limitation of these algorithms for this particular scene configuration (i.e., due to many car instances).

Note that for shadow maps smaller or equal than $2K^2$, the fragment mask is consistently the fastest method in our tests. For larger shadow maps of $4K^2$ or more, it becomes highly scene dependent whether the benefit outweighs the cost of the additional texture lookups during mask creation and occlusion culling. In this case, the GEOM+BVOL method can be used, which has a smaller overhead caused only by the stencil buffer writes and the rasterization of the additional bounding boxes. On the other hand, we expect a higher speedup for the reverse shadow testing once it is possible to write the stencil buffer in the shader. There is already a specification available called *GL_ARB_STENCIL_EXPORT* [Khronos Group 2010], and we hope that it will be fully supported in hardware soon.

6.3 Walkthrough Timings

The plots in Table 3 show timings for walkthroughs selected scene and parameter combinations. For Manhattan, we applied uniform shadow mapping with $4K^2$ resolution and use the combined geometry and bounding volume receiver mask (GEOM+BVOL). Note that in this scene, the FOCUS method is often useless because skyscrapers are often visible in the distance, while most of the geometry in between can actually be culled by receiver masking. In Vienna and Pompeii, we applied LiSPSM shadow mapping with $1K^2$ resolution, and the fragment mask (FRAG). The plots show that receiver masking is the only method that provides stable frame times for the presented walkthroughs, which is an important property for a real-time rendering algorithm. The walkthroughs corresponding to these plots along with visualizations of the culled objects can be watched in the accompanying video for further analysis.

Table 5 shows the statistics and the timings for a scene taken from an actual game title (Left 4 Dead 2) using cascaded shadow mapping. This game scene was rendered using a different occlusion culling algorithm than CHC++, implemented in a different rendering engine. The receiver masking is however implemented exactly

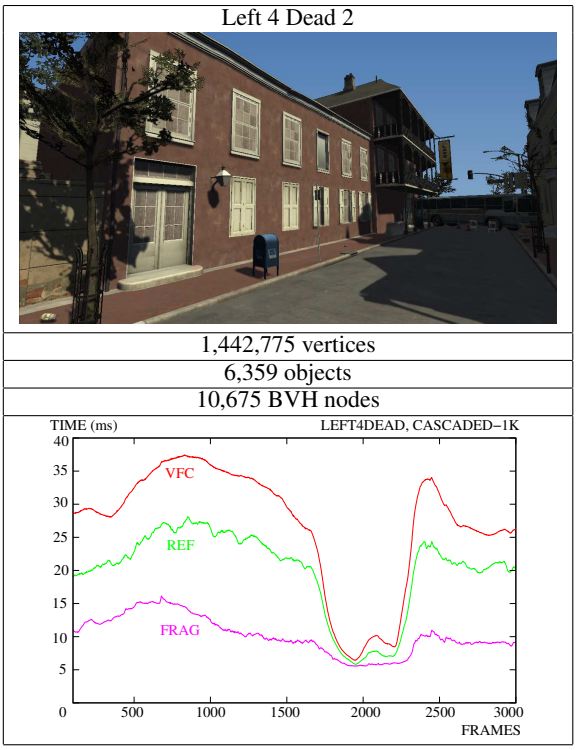


Table 5: *Statistics for the Left 4 Dead 2 scene and timings for a walkthrough using cascaded shadow mapping with four $1K^2$ shadow maps.*

as described in our paper. The fragment receiver mask and subsequent culling is applied separately for each of the four cascaded shadow maps using $1K^2$ resolution. Note that in this scene many objects are actually inside the houses and hence culled already by the REF method (depth-based culling without receiver masking). Even though this might imply a limited additional gain of our algorithm compared to REF, Table 5 shows that fragment masking still provides a significant speedup. Note that during frames 1,800-2,200, only a small fraction of objects is visible in the view frustum and almost all casters in the shadow frustum throw a visible shadow in the main view and thus there is very small potential for performance gain using any shadow caster culling method.

6.4 Geometry Analysis and CHC++ Optimization

Figure 5 analyses the actual amount of geometry rendered for different methods and contrasts that with frame times. In particular, here we include a comparison of the original CHC++ algorithm used in REF with our optimized version CHC++-OPT. The different receiver masking versions all use optimized CHC++. For all algorithms we used a $2K^2$ shadow map and LiSPSM.

Interestingly, the optimized CHC++ algorithm renders more vertices, but provides a constant speedup over the original version because of reduced CPU stalls when waiting for an occlusion query result. As can be observed from the plots, the speedup of the receiver masks corresponds closely to their level of tightness hence the number of rendered vertices.

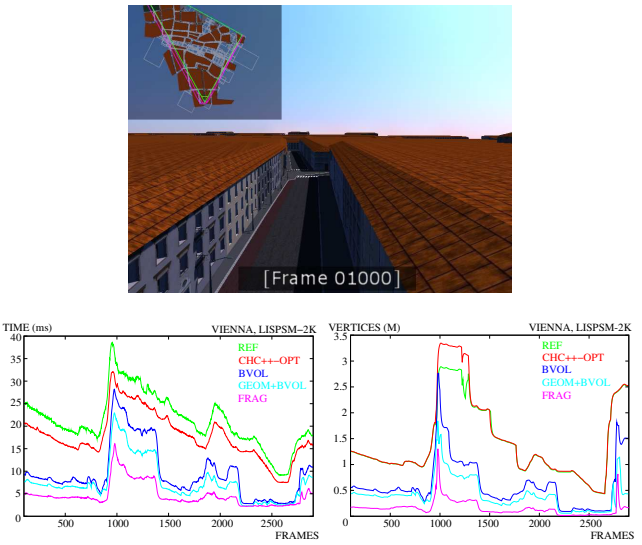


Figure 5: *Dependence of the frame time (bottom left) and number of vertices (bottom right) for a walkthrough of the Vienna scene and different shadow computation methods. Note that the walkthrouh also includes viewpoints located above the roofs. Such viewpoints see far over the city roofs (top). In this comparison the FRAG method achieves consistently the best performance followed by the the GEOM+BVOL method.*

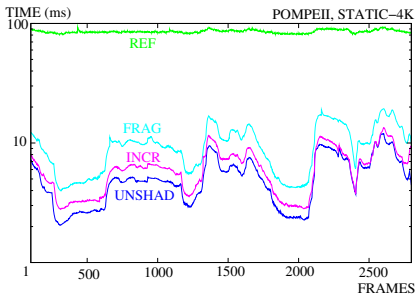


Figure 6: *Effect of incremental mask updates for a statically focused shadow map.*

6.5 Incremental Shadow Map Updates

In Figure 6, we use a static $4K^2$ shadow map for the whole Pompeii scene. Using this particular setup, REF performs over ten times worse than FRAG. Conversely, a static shadow map allows us to use the incremental shadow map updates optimization (INCR), which restricts shadow map updates to the parts of the shadow map that change, i.e., to the projections of the bounding boxes of dynamic objects. As can be seen, this technique can approach the performance of unshadowed scene rendering (UNSHAD). Note that we use a log-scale for this plot.

6.6 Dependence on Elevation Angle

In Figure 7, we show the dependence of the light source elevation angle on the (average) frame time for shadow mapping with resolution $1K^2$ in a Vienna walkthrough. For uniform shadow mapping, there is a noticeable correlation between the angle and the render times for REF and FOCUS. Interestingly, there seems to be a much weaker correlation for LiSPSM shadow mapping. The influence of the elevation angle on the performance of receiver masking is mini-

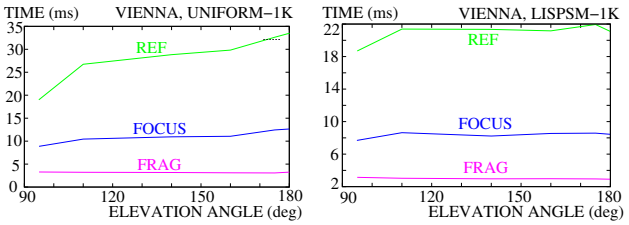


Figure 7: *Dependence of the (average) render time on the light elevation angle.*

mal in either case and our method delivers a consistent performance increase.

7 Conclusion

We proposed a method for efficient shadow caster culling for shadow mapping. In particular, our method aims at culling all shadow casters which do not contribute to visible shadow. We described several methods to create suitable receiver masks, and in particular one method which creates a practically pixel-exact mask through reverse shadow lookups.

Our method addresses the significant open problem of efficient construction of shadow maps especially in large outdoor scenes, which tend to appear more and more often in recent entertainment applications. We demonstrate speedups of 3x-10x for large city-like scenes and about 1.5x-2x for a scene taken from a recent game title. The method is also easy to implement and integrates nicely with common rendering pipelines that already have a depth-prepass.

Furthermore, we proposed a method for incremental shadow map updates in the case of statically focused shadow maps, as well as optimizations to the CHC++ occlusion culling algorithm, which are also applicable in other contexts.

In the future we want to focus on using the method in the context of different shadow mapping techniques and to study the behavior of the method in scenes with many lights.

Acknowledgements

We would like to thank Jiří Dušek for an early implementation of shadow map culling ideas; Jason Mitchell for the Left 4 Dead 2 model; Stephen Hill and Petri Häkkinen for feedback. This work has been supported by the Austrian Science Fund (FWF) contract no. P21130-N13; the Ministry of Education, Youth and Sports of the Czech Republic under research programs LC-06008 (Center for Computer Graphics) and MEB-060906 (Kontakt OE/CZ); and the Grant Agency of the Czech Republic under research program P202/11/1883.

References

BITTNER, J., WIMMER, M., PIRINGER, H., AND PURGATHOFER, W. 2004. Coherent hierarchical culling: Hardware occlusion queries made useful. *Computer Graphics Forum 23*, 3 (Sept.), 615–624. Proceedings EUROGRAPHICS 2004.

BRABEC, S., ANNEN, T., AND SEIDEL, H.-P. 2002. Practical shadow mapping. *Journal of Graphics Tools: JGT 7*, 4, 9–18.

COHEN-OR, D., CHRYSANTHOU, Y., SILVA, C., AND DURAND, F. 2003. A survey of visibility for walkthrough applications. *IEEE Transactions on Visualization and Computer Graphics. 9*, 3, 412–431.

DÉCORET, X. 2005. N-buffers for efficient depth map query. *Computer Graphics Forum 24*, 3.

EISEMANN, E., AND DÉCORET, X. 2006. Fast scene voxelization and applications. In *ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, ACM SIGGRAPH, 71–78.

ENGELHARDT, T., AND DACHSBACHER, C. 2009. Granular visibility queries on the gpu. In *I3D '09: Proceedings of the 2009 symposium on Interactive 3D graphics and games*, ACM, New York, NY, USA, 161–167.

GOVINDARAJU, N. K., LLOYD, B., YOON, S.-E., SUD, A., AND MANOCHA, D. 2003. Interactive shadow generation in complex environments. *ACM Trans. Graph. 22*, 3, 501–510.

GUTHE, M., BALÁZS, A., AND KLEIN, R. 2006. Near optimal hierarchical culling: Performance driven use of hardware occlusion queries. In *Eurographics Symposium on Rendering 2006*, The Eurographics Association, T. Akenine-Möller and W. Heidrich, Eds.

KHRONOS GROUP, 2010. Opengl extension registry. http://www.opengl.org/registry, October.

LAURITZEN, A., SALVI, M., AND LEFOHN, A. 2010. Sample distribution shadow maps. In *Advances in Real-Time Rendering in 3D Graphics and Games II, SIGGRAPH 2010 Courses*.

LLOYD, B., WENDT, J., GOVINDARAJU, N., AND MANOCHA, D. 2004. Cc shadow volumes. In *Proceedings of EUROGRAPHICS Symposium on Rendering 2004*.

LLOYD, D. B., TUFT, D., YOON, S.-E., AND MANOCHA, D. 2006. Warping and partitioning for low error shadow maps. In *Proceedings of the Eurographics Workshop/Symposium on Rendering, EGSR*, Eurographics Association, Aire-la-Ville, Switzerland, T. Akenine-Möller and W. Heidrich, Eds., 215–226.

LLOYD, D. B., GOVINDARAJU, N. K., QUAMMEN, C., MOLNAR, S. E., AND MANOCHA, D. 2008. Logarithmic perspective shadow maps. *ACM Trans. Graph. 27*, 4, 1–32.

MATTAUSCH, O., BITTNER, J., AND WIMMER, M. 2008. Chc++: Coherent hierarchical culling revisited. *Computer Graphics Forum (Proceedings of Eurographics 2008) 27*, 3 (Apr.), 221–230.

SCHERZER, D., WIMMER, M., AND PURGATHOFER, W. 2010. A survey of real-time hard shadow mapping methods. In *State of the Art Reports Eurographics*,

STAMMINGER, M., AND DRETTAKIS, G. 2002. Perspective shadow maps. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, ACM, New York, NY, USA, 557–562.

WILLIAMS, L. 1978. Casting curved shadows on curved surfaces. *Computer Graphics (SIGGRAPH '78 Proceedings) 12*, 3 (Aug.), 270–274.

WIMMER, M., SCHERZER, D., AND PURGATHOFER, W. 2004. Light space perspective shadow maps. In *Rendering Techniques 2004 (Proceedings Eurographics Symposium on Rendering)*, Eurographics Association, A. Keller and H. W. Jensen, Eds., Eurographics, 143–151.