Resolution-Matched Shadow Maps

AARON E. LEFOHN
University of California, Davis and Neoptica
and
SHUBHABRATA SENGUPTA and JOHN D. OWENS
University of California, Davis

This paper presents resolution-matched shadow maps (RMSM), a modified adaptive shadow map (ASM) algorithm, that is practical for interactive rendering of dynamic scenes. Adaptive shadow maps, which build a quadtree of shadow samples to match the projected resolution of each shadow texel in eye space, offer a robust solution to projective and perspective aliasing in shadow maps. However, their use for interactive dynamic scenes is plagued by an expensive iterative edge-finding algorithm that takes a highly variable amount of time per frame and is not guaranteed to converge to a correct solution. This paper introduces a simplified algorithm that is up to ten times faster than ASMs, has more predictable performance, and delivers more accurate shadows. Our main contribution is the observation that it is more efficient to forgo the iterative refinement analysis in favor of generating all shadow texels requested by the pixels in the eye-space image. The practicality of this approach is based on the insight that, for surfaces continuously visible from the eye, adjacent eye-space pixels map to adjacent shadow texels in quadtree shadow space. This means that the number of contiguous regions of shadow texels (which can be efficiently generated with a rasterizer) is proportional to the number of continuously visible surfaces in the scene. Moreover, these regions can be coalesced to further reduce the number of render passes required to shadow an image. The secondary contribution of this paper is demonstrating the design and use of data-parallel algorithms inseparably mixed with traditional graphics programming to implement a novel interactive rendering algorithm. For the scenes described in this paper, we achieve 60–80 frames per second on static scenes and 20–60 frames per second on dynamic scenes for 512^2 and 1024^2 images with a maximum effective shadow resolution of $32,768^2$ texels.

Categories and Subject Descriptors: I.3.7 [**Computer Graphics**]: Three-Dimensional Graphics and Realism—Color, shading, shadowing, and texture

Additional Key Words and Phrases: shadows, shadow maps, adaptive shadow maps, scan, GPU, graphics hardware, GPGPU

1. INTRODUCTION

In complex and realistic scenes, shadows provide important visual cues to viewers that both aid in image understanding and match the expected photorealistic results. Consequently, accurate calculation of shadows is an important component in today's rendering systems, from film-quality renderers that may take hours per frame to real-time, graphics processor (GPU) based renderers that must render plausible shadows many times per second.

Despite an expansive body of literature, efficient generation of high-quality shadows is still largely an unsolved problem. Ray tracing methods are very high quality but are used sparingly even in offline renderers due to their cost. Examples of real-time ray-traced shadows exist [Wald et al. 2003], but their widespread adoption is limited by the challenge of requiring random access to the scene database and efficient support for dynamic scenes. Shadow volume techniques [Crow 1977] are popular in real-time applications, but require object-based analysis that can be costly and are difficult to combine with displacement/vertex shaders that procedurally deform geometry. Shadow mapping techniques [Williams 1978] are commonly used in both

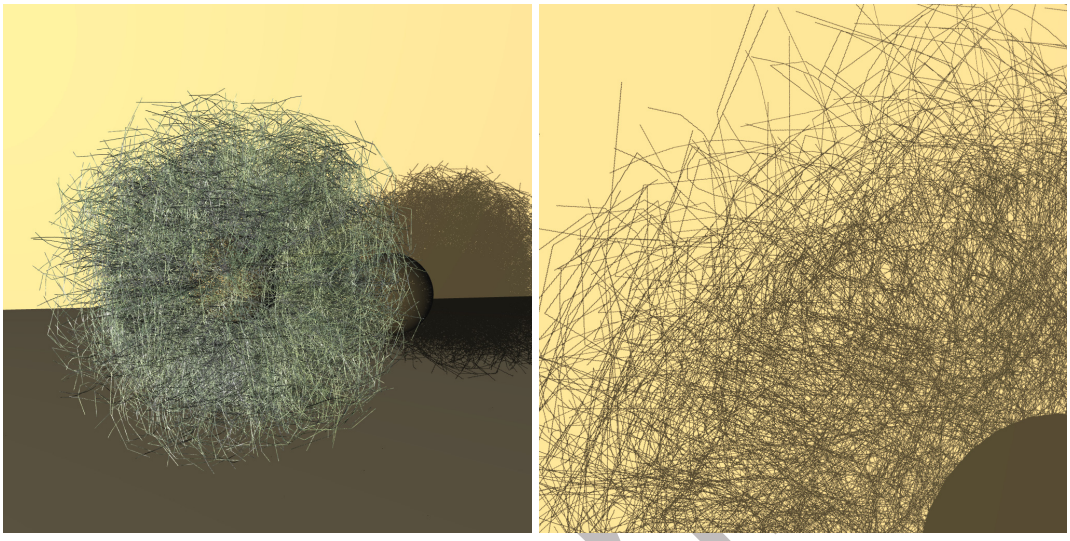


Fig. 1. A difficult scene for shadow map algorithms, this furball consists of 4,000 self-shadowing hairs of 12 line segments each and is shadowed with a $32{,}768^2$ effective resolution, resolution-matched shadow map. For a 1024^2 image, our implementation running on an NVIDIA GeForce 8800 GPU GTX renders the left image at 60–75 fps with a static light and 20–25 fps for a moving light. The right image is a close-up of the hair shadow on the wall, and renders at 70–75 frames per second (fps) with a static light and 60–65 fps with a moving light.

offline and interactive renderers due to their simplicity, support for procedural deformations, and native GPU support. However, shadow mapping techniques suffer from projective, perspective, and depth-precision aliasing [Stamminger and Drettakis 2002].

This paper presents resolution-matched shadow maps (RMSMs), an improved adaptive shadow map algorithm and novel implementation that both addresses the aliasing problems of shadow map algorithms and allows real-time performance for complex scenes on graphics hardware. Like adaptive shadow maps (ASMs) [Fernando et al. 2001], our technique generates shadow data at the resolution required by the current camera view and stores the data in a quadtree of small shadow map pages. Unlike ASMs, however, RMSMs leverages coherence between image and adaptive-shadow-map space to generate shadows in a single step rather than using iterative refinement. We achieve real-time performance by combining this coherency insight with data-parallel scene analysis algorithms implemented on the GPU. Our technique is up to ten times faster than previous GPU-based adaptive shadow map implementations, performs more predictably, and is guaranteed to produce correct shadows for a given camera view, within the limits of the maximum resolution supported by the quadtree.

2. BACKGROUND

Real-time shadow calculations generally use two main classes of techniques, shadow maps and shadow volumes. Woo et al. provide an early survey of shadow algorithms [1990], with Hasenfratz et al.'s more recent survey encompassing both an overview of these two algorithms as well as more recent work [2003]. Ray tracing can produce physically correct hard and soft shadows, and has recently been shown to be possible for some real-time scenes [Reshetov et al. 2005; Thrane and Simonsen 2005]. However, the requirement of having random access to the entire geometry database and the difficulty of maintaining spatial acceleration data structures for dynamic scenes limits the current practicality of ray tracing for interactive rendering.

The simplicity, efficiency, and hardware support for shadow maps makes them an increasingly common choice for real-time renderers [Forsyth 2004]. Briefly, the *shadow map*, first described by Lance Williams [1978], is an image-space technique that computes the scene from the light's point of view, storing the distance from the light to the scene geometry in a shadow map. The scene is then rendered from the point of view of the camera, using the information in the shadow map to decide if portions of the image are directly illuminated by the light or are in shadow.

The classic shadow map algorithm suffers from three kinds of aliasing: perspective, projective, and depth-precision aliasing [Stamminger and Drettakis 2002]. Perspective aliasing is caused by a mismatch between the sampling rate of screen-space pixels and shadow texels, projective aliasing occurs when the light is nearly parallel to an occluder (Figure 2), and depth-precision aliasing results in *shadow acne* due to false self-shadowing of surfaces. The algorithm described in this paper addresses both perspective and projective aliasing and uses standard bias techniques to avoid depth-precision aliasing.

2.1 Recent Work in Shadow Maps

Recent approaches address perspective aliasing by computing the shadow map in a perspective-warped space rather than light space [Stamminger and Drettakis 2002; Wimmer et al. 2004]. While these techniques remedy many aliasing artifacts, they do not remove all aliasing, and have special cases that make them difficult to use in practice [Forsyth 2004]. Recent work by Lloyd et al. helps reduce the special cases by using multiple parameterizations instead of a single, global one [2005]. However, the perspective-based techniques do not address projective aliasing errors.

Other recent efforts include the hybrid shadow map/volume rendering algorithm of Chan and Durand [2004], trapezoidal shadow maps [Martin and Tan 2004], and silhouette maps [Sen et al. 2003; Sen 2004], which apply a non-linear deformation to the shadow lookup based on an object-based edge analysis. Chong and Gortler's work [2004] is related to adaptive shadow maps, but is correct only for user-selected planes and requires a separate pass for each selected plane. The tiled shadow map work of Arvo is also related to adaptive shadow maps and subdivides shadow map space using a heuristic based on depth discontinuities, surface distance from the light and eye, and other factors [Arvo 2004]. Unfortunately, the approach suffers from aliasing artifacts caused by inconsistent shadow resolutions. Forsyth's shadow-map-based "shadow buffers" method renders several shadow buffers per light in order to better match the resolution of the shadow map to the screen [2004]. Variance shadow maps [Donnelly and Lauritzen 2006] introduce a shadow map representation that supports mipmapping to address minification aliasing and support large filter regions, but the technique does not address magnification aliasing where additional shadow resolution is required. Lloyd et al. [2006] give an in-depth analysis of the relative error resulting from combining light frustum partitioning, view frustum partitioning, and perspective reparameterizations. None of these methods achieve the combination of performance, aliasing artifact avoidance, and algorithmic elegance that we target with our work here.

Another set of approaches eliminates shadow map aliasing by proposing new graphics hardware that rasterizes shadow data at the exact location required by the shadow coordinates in the current camera view [Aila and Laine 2004; Johnson et al. 2005]. These approaches entirely eliminate aliasing and are equivalent to ray-traced hard shadows. Our technique closely approximates these methods with an algorithm that runs in real time on current graphics hardware.

Adaptive Shadow Maps. The adaptive shadow map (ASM) algorithm introduced by Fernando et al. [2001] addresses both projective and perspective aliasing problems by adaptively sampling the shadow map and matching its resolution to the pixels in the current camera view. The approach stores shadow texels in a quadtree of small shadow map *pages* rather than a uniform grid. The quadtree is built via an iterative refinement algorithm that refines along shadow boundaries found in the current camera view (see Algorithm 1). Shadow map resolutions are estimated based on derivatives of the shadow map coordinates in

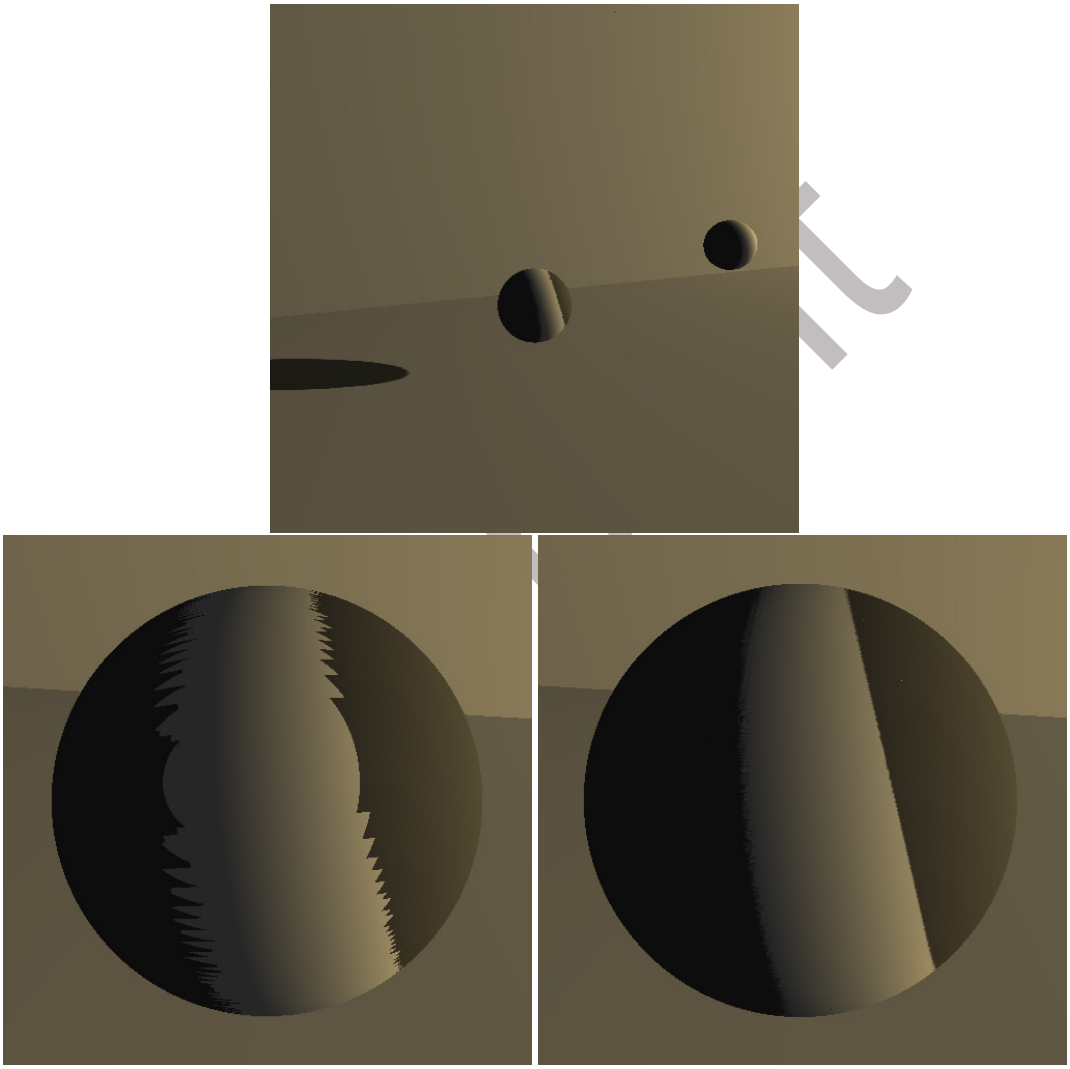


Fig. 2. A difficult case for standard and perspective-shadow-map-based algorithms is projective aliasing, which occurs when the light is parallel to the occluder. In the top picture, the grazing angles of the light that cause the two shadow boundaries on the middle sphere result in severe projective aliasing artifacts with a standard shadow map (bottom-left). By matching shadow resolution to the current camera view, adaptive shadow map techniques (including our resolution-matched shadow maps) give a more accurate shadow boundary (bottom-right), although close inspection reveals that even the finest $32{,}768^2$ resolution level we show here is insufficient; in fact, in the limiting case, infinite resolution would be required.

screen space, much in the same way that texture mapping computes mipmap levels [Segal and Akeley 2004]. The refinement algorithm begins with a low-resolution 'seed' shadow map. During each iteration, the current quadtree is analyzed for shadow edges. Edge texels that are not at the correct resolution are refined to the correct resolution by generating new shadow pages and updating the data structure. Queried virtual shadow maps [Giegl and Wimmer 2007] have similar goals as adaptive shadow maps and also use recursive subdivision to iterate to convergence, but use a heuristic to determine shadow refinement rather than computing the exact projected area.

ASMs can generate alias-free, hard shadows if the ASM refinement algorithm finds all shadow edges in the image and the required resolution does not exceed the maximum depth of the quadtree. The technique is widely regarded as a robust solution to the shadow mapping problem but impractical for interactive, GPU-based rendering solutions. For example, Chan and Durand indicate that "the required data structures and host-based calculations preclude real-time performance for dynamic scenes" [2004]. Fernando's original implementation was a hybrid CPU-GPU solution that performed all shadow lookups on the CPU. The work we describe in this article builds on work we first presented as a SIGGRAPH sketch [Lefohn et al. 2005], based on our adaptive multiresolution quadtree GPU data structure [Lefohn et al. 2006].

This paper addresses three problems with the original ASM algorithm: it is not guaranteed to find all shadow edges (see Figure 5 for an example of incorrect ASM refinement), it cannot easily be constrained to converge in a fixed amount of time, and the costly scene analysis prevents interactive rendering of dynamic scenes. Section 3 describes our new non-iterative approach, resolution-matched shadow maps, for building a quadtree of shadow map pages that is guaranteed to find all shadow edges in the current view, within the limits of the maximum resolution of the quadtree. We also present a data-parallel scene analysis algorithm that is amenable to efficient GPU implementation. Section 4 provides detailed analysis showing the practicality of our technique, and compares our approach to iterative ASMs and standard shadow maps.

3. ALGORITHM AND IMPLEMENTATION

This section describes the design and implementation of an interactive, non-iterative, adaptive shadow map algorithm. We refer to this algorithm and its implementation as resolution-matched shadow maps or RMSMs. Our goal is to transform the original, iterative ASM algorithm of Fernando et al. [2001] that we previously mapped to the GPU [Lefohn et al. 2006] (Algorithm 1) into a real-time rendering solution suitable for dynamic scenes.

3.1 Locality in Adaptive Shadow Space

We begin our discussion of design decisions with an observation about the relationship between shadow space and image space. In an adaptive shadow map, each image pixel is mapped to a shadow texel of the correct shadow resolution, where the size of the texel is as close as possible to the size of the pixel. Consider two adjacent image pixels that are also adjacent samples of an object, and a shadow that is cast onto that surface. Because the pixels are adjacent in image and object space, they will likely be mapped to identical shadow resolutions. And because of the 1:1 mapping between image space and adaptive-shadow-map space, neighboring pixels in image space map to neighboring texels in the adaptive shadow map. We thus draw the conclusion that *continuously visible surfaces in image space result in coherent accesses in adaptive-shadow-map space*. This observation is vital to the success of our algorithm because we take advantage of this coherency in several ways to improve upon the original ASM algorithm.

Do all scenes exhibit this coherency property? No; a scene with a *different object at each pixel*, a light perpendicular to the eye, and depths distributed across the entire light frustum will exhibit no locality. We submit, however, that interesting scenes are ones that do have continuously visible surfaces and will exhibit this property. For humans to recognize an object, we must see enough of that object to draw a conclusion

```
1: render low-resolution seed into quadtree memory
2: repeat
3:   for all pixels in image rendered from camera do
4:     calculate (s, t, z_l, ℓ) shadowmap coords. and LOD
5:     lookup in quadtree memory
6:     if pixel on shadow edge and page not in ASM then
7:       convert (s, t, z_l, ℓ) to shadow page request
8:     transfer page requests to CPU
9:     remove invalid page requests
10:    generate unique page requests
11:    allocate new page in quadtree
12:    bin requests into superpages
13:    render shadow data into superpages
14:    copy shadow data from superpage to quadtree
15: until page requests == 0
```

Algorithm 1: The original, iterative ASM algorithm. The algorithm generates a quadtree of small shadow map pages by rendering a low-resolution shadow map, then renders from the eye to perform shadow lookups, requests shadow sample refinement on shadow boundaries whose resolution does not yet match the required resolution, renders the requested shadow data into the quadtree shadow map, and iterates until no more shadow samples are requested. The original ASM algorithm implemented all steps on the CPU except for 4 and 13. In contrast, the optimized ASM implementation used in this paper to compare against resolution-matched shadow maps, performs all computations on the GPU except for steps 8, 9, 10, and 12. The shadow map coordinates ($\in [0, 1]$) are denoted (s, t, z_l) with resolution (analogous to a mipmap resolution) denoted ℓ.

about it, which implies many pixels will be visible. Conversely, a scene with a different object at each pixel would be quite difficult to understand. In Section 4.3.1, we show that even highly complex, incoherent shadow receivers, such as the furball of Figure 1, exhibit substantial locality in shadow space. We take advantage of this locality in three ways: we organize our adaptive shadow map into shadow pages, use a single-step, non-iterative algorithm to request shadow pages, and introduce a connected-component step to optimize the scene analysis stage.

3.2 Optimizing the ASM Algorithm

As we describe above, ASM's iterative edge-finding algorithm imposes a high cost, high performance variability, and is error-prone (Figure 5); however, the advantage of the edge-finding step is that it only refines on shadow boundaries, which reduces the number of shadow pages that need to be processed and allocated to render an image (see Figure 4). Fortunately, the image-shadow coherency property described in Section 3.1 enables us to eliminate the iteration by requesting a shadow page for *every* pixel in the image. Because of the coherency, many of these requests will be redundant, and the number of unique shadow pages is only slightly more than are required by the iterative method. In order to make this single-step method practical, we use data-parallel, GPU-based algorithms to create a set of unique page requests before sending the request to the CPU to initiate shadow data generation. This section describes our new algorithm (summarized in Algorithm 2) and the design process. In the following discussion, we use the example "robot" scene (Figure 3) to show how each of our optimizations impacts performance for both our new, non-iterative RMSM implementation and the previous iterative ASM implementation. Note that all performance results are computed on an NVIDIA GeForce 7800 GTX GPU.

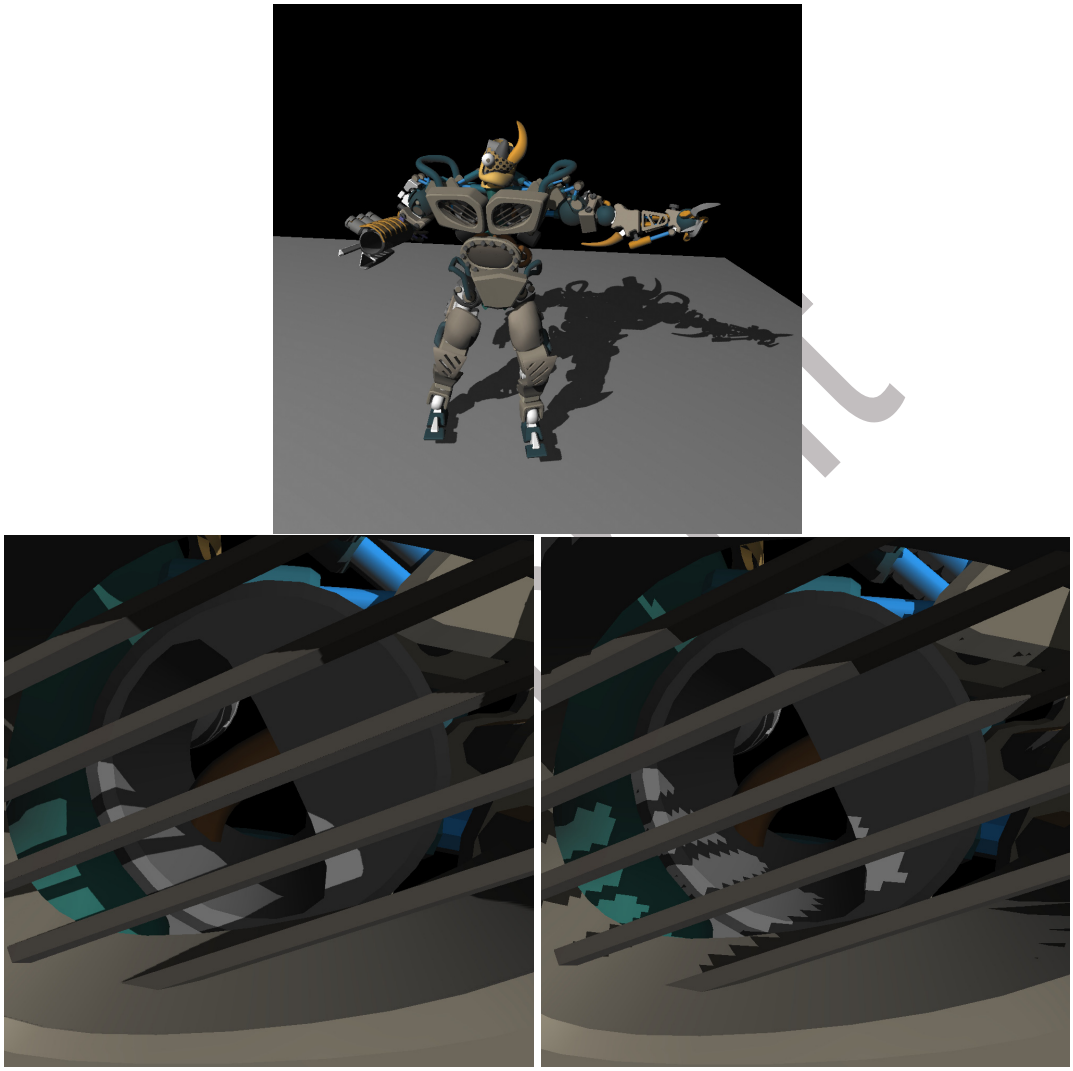


Fig. 3. We use this robot scene (top) with 66,000 polygons as an example to explain our design decisions. The bottom-left image shows a shadow close-up using our resolution-matched shadow map algorithm, and the bottom-right image shows the same view with a standard, 2048^2 shadow map.

To begin, if we simply read back all shadow requests to the CPU without any simplification, we achieve 0.5–1 frame per second due to the cost of the CPU processing all of the requests and the time required to transfer data from the GPU to CPU. Fortunately, we can take advantage of the image-shadow coherency by performing a GPU connected-components analysis before transferring the data to the CPU. We eliminate redundant page requests between neighboring pixels by marking only requests whose immediate neighbor pixels below and to the left of the current pixel request a different page (in the best case, we only mark one request per page). This optimization improves the robot performance by 10–20 times and achieves an

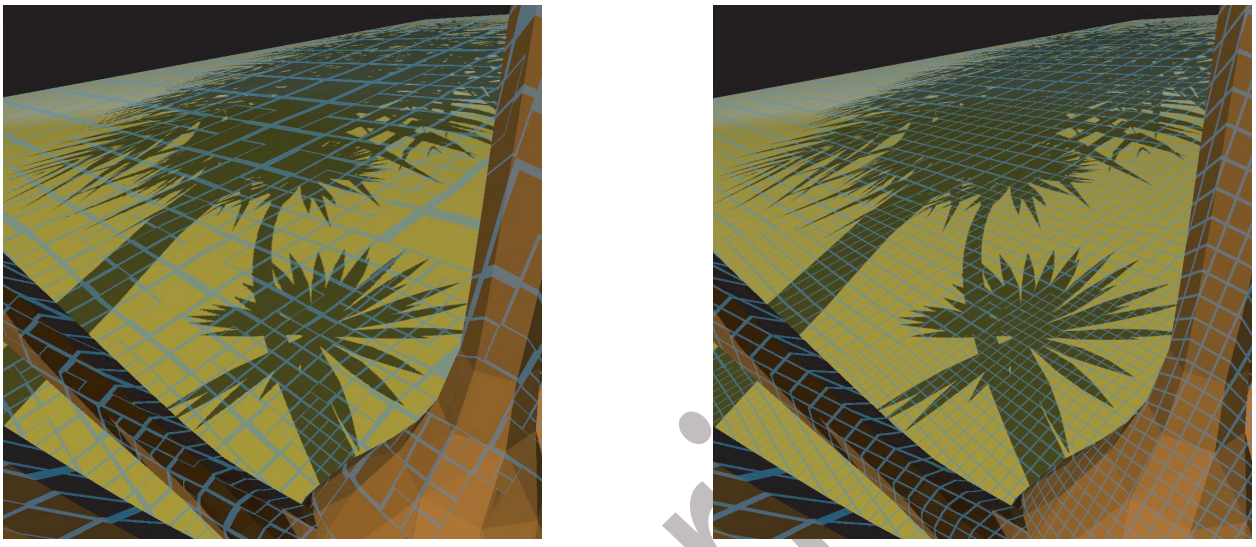


Fig. 4. Visualization of shadow pages in a shadowed image, where each blue square represents 32×32 shadow texels. The left figure shows traditional adaptive shadow mapping (ASM) and the right image shows resolution-matched shadow maps (RMSM). Note that ASMs refine only near shadow boundaries, whereas RMSMs refine based solely on the projected area of a fragment into shadow map space. Although this results in RMSMs using slightly more memory, the shadow data can be generated 2–10 times faster with less variation in performance than the iterative refinement required for ASMs.

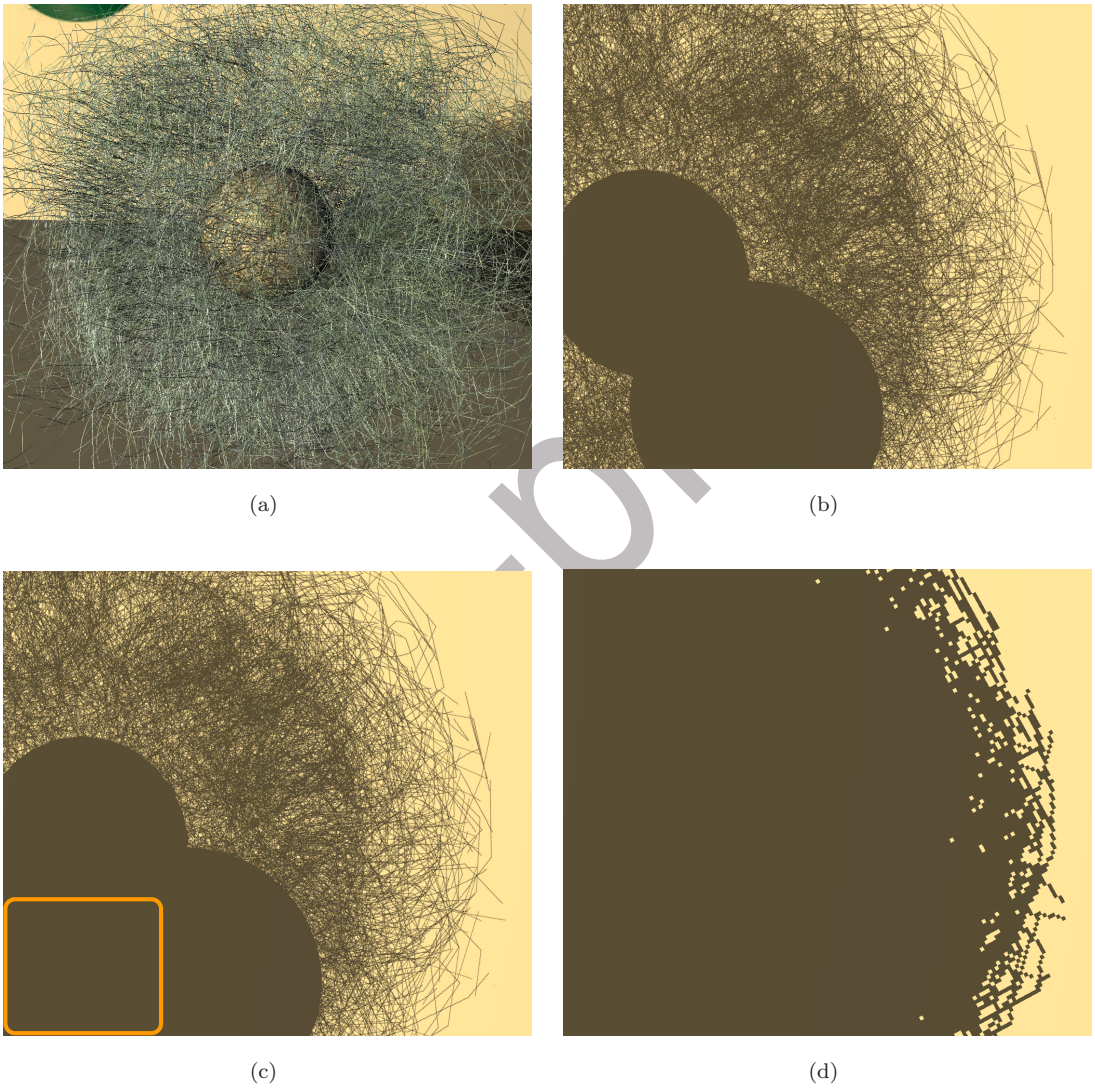
average frame rate of 10 fps.

Our next optimization step eliminates invalid page requests before transferring page requests to the CPU. Invalid requests arise from pixels that do not request a shadow page due to the connected-components pass, a scene that does not completely cover the viewport, or, for the iterative ASM algorithm, a shadow page that is already allocated from a previous iteration. Eliminating invalid page requests on the GPU takes the average frame rate from 10 fps to 15 fps.

Our final optimization step is to eliminate all remaining redundant page requests on the GPU before transferring them back to the CPU. This redundancy may be missed by the connected-components step if occlusion breaks up otherwise continuously visible surfaces. We perform this *uniquify* operation by first sorting the page requests by page address using GPUSort [Govindaraju et al. 2006], then marking unique elements by comparing each element with its immediate predecessor. Finally, we compact the sorted list to remove all non-unique elements. For the robot scene, this step results in a slight performance gain, increasing the average frame rate from 15 fps to 17 fps, but performance improves up to 2 times for scenes with significant occlusion, such as the furball scene in Figure 1.

For comparison, we implemented the same three optimizations for the iterative ASM algorithm. The first optimization takes the average frame rate for the robot scene from 4 fps to 5 fps. The second optimization increases the average frame rate to 7 fps. The third optimization has no discernible effect on the average frame rate.

The final resolution-matched shadow map algorithm is summarized as Algorithm 2, and complete results and analysis results for these optimizations are presented in Section 4.2 (especially see Figure 8).



(a) (b)

(c) (d)

Fig. 5. We use a scene (a) with incoherent, fine geometry (4,000 hairs each consisting of 12 line segments) as a stress-test for our resolution-matched shadow map (RMSM) algorithm (b) to compare against traditional, iterative ASMs (c) and standard shadow maps (d). Note the refinement error in the lower-left corner of the iterative ASM refinement image (missed hair shadows highlighted by the orange box)(c).

1: **for** all pixels in image rendered from camera **do**
2: calculate (s, t, z_l, ℓ) shadowmap coords. and LOD
3: convert (s, t, z_l, ℓ) to shadow page request
4: eliminate redundant requests via connected-components
5: eliminate invalid requests (compaction)
6: sort page requests
7: compact again to generate unique page requests
8: transfer unique page requests to CPU
9: allocate new page in quadtree
10: bin requests into superpages
11: render shadow data into superpages
12: copy shadow data from superpage to quadtree memory

Algorithm 2: The resolution-matched shadow map (RMSM) algorithm. The algorithm generates a quadtree of small shadow map pages by rendering from the eye to generate shadow requests for all pixels, renders all requested shadow pages into a quadtree shadow map, then renders again from the eye to perform the shadow lookup. Unlike the original ASM algorithm, the RMSM algorithm generates all shadow data in a single step rather than iterating. All computations except for steps 8 and 10 are performed on the GPU, and the shadow map coordinates ($\in [0, 1]$) are denoted (s, t, z_l) with resolution (analogous to a mipmap resolution) denoted ℓ.

3.3 Implementation

Here we describe our implementation of the resolution-matched shadow map algorithm (Algorithm 2).

3.3.1 *The Quadtree Shadow Data Structure.* We store the quadtree of shadow data in the adaptive, multiresolution GPU data structure described in Lefohn et al. [2006]. The structure uses a mipmap hierarchy of page tables and stores all pages in a single physical memory data texture. The structure is addressed by s and t coordinates in shadow space as well as the level-of-detail ℓ. The (s, t) coordinates are identical to the coordinates used in a standard shadow map. Reading from it returns a scalar that indicates the fraction of shadow coverage at the shadow map coordinate s, t; this value is computed by performing two percentage-closest-filtered lookups [Reeves et al. 1987] of 4 shadow texels each at two adjacent levels of detail and then blending the results together. We insert pages by rendering to the page table hierarchy, and we write to quadtree nodes by either rendering depth data directly into the pages stored in the physical memory texture or copying data from a larger scratch space.

3.3.2 *Algorithm Phase 1: Generating Requests.* Our first phase (Steps 1–3 of Algorithm 2) begins with a geometry pass that generates the shadow coordinates. At each pixel, we calculate the s, t, and z_l coordinates for the light-space shadow map, where s and t are standard shadow map coordinates and z_l is the depth of the current pixel transformed into light space. In order to better handle anisotropy, we have found it necessary to compute the shadow level-of-detail, ℓ, more accurately than OpenGL's mipmapping calculation. We compute ℓ by:

$$\begin{aligned} dX &= \left(\frac{\partial s}{\partial x}, \frac{\partial t}{\partial x}\right) \\ dY &= \left(\frac{\partial s}{\partial y}, \frac{\partial t}{\partial y}\right) \\ A &= |dX \times dY| \\ \ell &= \log_2(\sqrt{A}). \end{aligned} \tag{1}$$

This computation computes the area of the parallelogram formed by dX and dY rather than the OpenGL computation that computes the area of a bounding square [Segal and Akeley 2004]. Note that the derivative

vectors are not the actual axes of the filter ellipse, and an interesting direction of future work would be to investigate computing the level-of-detail based on the actual ellipse axes [Greene and Heckbert 1986; McCormack et al. 1999].

We create a set of unique shadow page requests (Steps 4–8 of Algorithm 2) using several data-parallel algorithm primitives: sort, scan, and gather. The uniquify algorithm has four stages. First, the connected-components step eliminates redundant requests between neighbors by marking only requests whose immediate neighbor pixels below and to the left of the current pixel request a different page. Next, we *compact* the list to remove all unmarked page requests using a combination of parallel-prefix scan and gather. Lefohn et al.'s original GPU ASM implementation uses Horn's $O(n \log n)$ compaction implementation [Horn 2005] for this task, reporting that this single kernel alone took 85% of the runtime of the entire shadow-mapping pipeline [Lefohn et al. 2006]. We developed an alternate, more efficient $O(n)$ implementation of parallel compaction that is significantly faster. Sengupta et al. describe this new scan implementation in detail [2006]. After compaction, we then *sort* the resulting stream by request page address using GPUSort [Govindaraju et al. 2006], then mark unique elements by comparing each element with its immediate predecessor. Finally, we compact the sorted list to remove all non-unique elements, and read the compacted stream back to the CPU for the next step.

3.3.3 *Algorithm Phase 2: Generating a quadtree of shadow maps.* We generate a quadtree of shadow map pages by rendering into both the quadtree's page tables and physical memory texture (Steps 9–12 of Algorithm 2). We insert new shadow pages into the GPU quadtree via a parallel memory allocation routine. We draw a quad per page request into each page table resolution level that is at least as fine as the page request. Each quad has a depth proportional to the page's resolution level, allowing any conflicts to be resolved by the depth buffer. Such conflicts arise when pages at different resolutions overlap. In such cases, each page table resolution should use the highest resolution page that is less than or equal to that page table's resolution. Because all of the request quads can be rendered in parallel, this operation is quite fast.

Next, we write shadow data into the newly allocated pages. In theory, we could render each page as a separate pass, but this leads to a large number of geometry passes and requires a high-resolution acceleration structure to achieve good performance. Instead, we again leverage the coherency of shadow requests and bin pages into 1024×1024 *superpages*. Each superpage is rendered into a temporary buffer, and the requested shadow pages are copied into the quadtree physical memory by drawing one quad for each page. All valid pages from a given superpage are processed in parallel. We discuss performance implications of the superpage approach in Section 4.2.1.

3.3.4 *Optimizing for Static Scenes.* The above algorithm recomputes shadows on every frame and is hence applicable to scenes with dynamic geometry or lighting. Some scenes, such as architectural walkthroughs, may be wholly static, where the only difference from frame to frame is the position of the camera. With a static scene, the data in the quadtree remains valid between frames; our algorithm can also be used for these scenes in what we call *cached mode* by retaining the data structure and incrementally updating it with any new requested shadow pixels. Because cached mode only adds information to a stored quadtree, resulting in large memory usage, it may be necessary to periodically flush the structure and rebuild it.

3.3.5 *Quality vs. Runtime/Memory Tradeoffs.* If the runtime for our shadow algorithm is unacceptably large, application developers may choose to reduce the quality of the shadows in exchange for higher shadow performance. We provide two orthogonal methods for doing so.

First, Phase 1 of our algorithm analyzes the shadow coordinates for every pixel and thus guarantees shadow correctness. Developers may choose to reduce the resolution of this analysis step. This reduces the cost of the sort and improves runtime while possibly missing some shadow pages. Note that in this case, an additional 'backup' standard shadow map is necessary to handle lookup requests not found in the quadtree.

In practice, we find that downsampling 2x, 4x, and even 8x is possible with little or no loss of shadow quality. Interestingly, the most likely viewing scenario that will result in missed shadows are incoherent receivers on which the error may not be perceptible.

Second, developers may also choose to reduce the resolution of the finest level of the shadow map, which reduces the number of shadow page requests as well as the amount of memory required for the data structure (we also automatically apply this technique if the amount of required shadow data exceeds the amount of physical memory allocated for shadow pages). The result of this operation is a uniform loss of resolution for detailed shadows. As the resolution is reduced, the resulting shadows degrade to those no less objectionable than the perspective and projective aliasing artifacts found in standard shadow map methods. The two constraint techniques can be applied either separately or together.

4. RESULTS AND DISCUSSION

We evaluate the performance, memory usage, and quality of four variants of GPU adaptive shadow maps (including RMSMs) and standard shadow maps for five scenes. Overall, our results show that RMSMs are 2–10 times faster than our optimized implementation of traditional, iterative ASMs, use 1–3 times more memory, and increase in performance 2–5 times with each generation of GPU (measured over three generations). The results also show that, due to current PC's limited CPU-GPU bandwidth and CPU compute power, the algorithm is only practical if the scene analysis is performed using GPU-based data-parallel algorithms. Lastly, we show the method can be easily scaled to tradeoff quality versus performance.

The four ASM variants tested include iterative versus non-iterative and CPU-based versus GPU-based scene analysis as well as the design tradeoffs discussed in Section 3. The resolution-matched shadow map algorithm refers to the non-iterative, GPU-based solution. Unless otherwise noted, all quadtree shadow maps use a maximum effective shadow resolution of $32,768^2$ and a page size of 32^2 texels. The standard shadow maps use a 2048^2 OpenGL hardware shadow map and a fixed light frustum.

The five scenes included in our evaluation are:

Scene	Number of Primitives	Image
Robot	61,126	Figure 3
Skeletons	80,000	Figure 6
Trees	48,000	Figure 16
City	58,000	Figure 7
Furball	45,000	Figures 1 and 5

The robot, skeletons, and city scenes are representative of scenes used in recent shadow papers and represent a single character, a collection of detailed characters, and an outdoor scene with relatively simple models of various sizes. The trees scene is slightly more difficult, having a mixture of fine, incoherent geometry and coarse objects. The furball is the most difficult scene, with almost all of the geometry being small hairs represented as individual lines. The many incoherent receivers result in many thin occluded surfaces and are a tough case for nearly all shadow map algorithms.

4.1 Experimental Setup

To account for asynchronous GPU execution and the dynamic, data-dependent GPU computation, performance measurements were obtained by measuring the total frame time as time between swap-buffers. This unfortunately includes time spent by the application processing events such as mouse input, etc. While this approach adds noise and an effective upper-bound to our frame-rate measurements, it was the most accurate approach available short of being able to insert time-stamp labels into the GPU command stream. Unlike

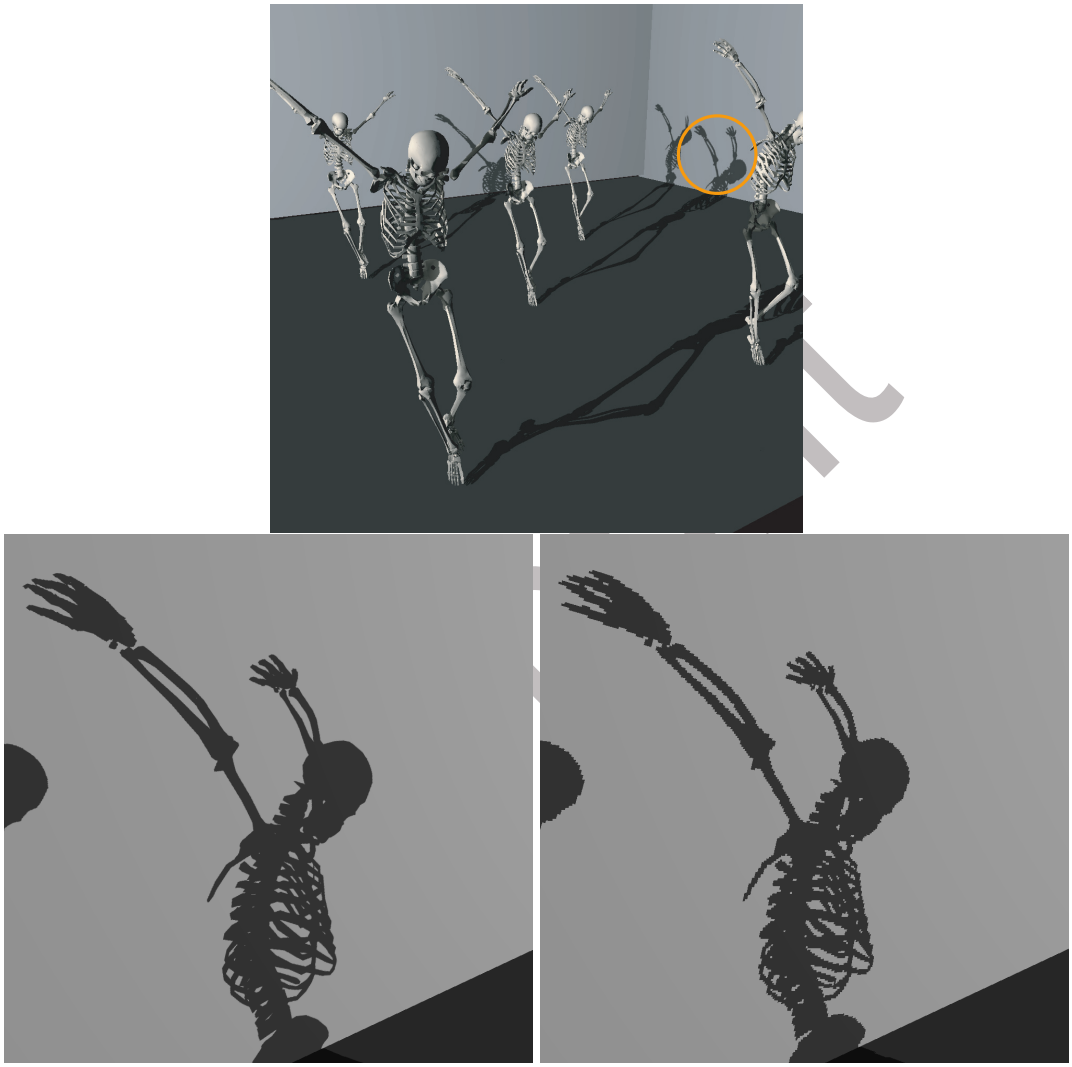


Fig. 6. Top shows the skeleton scene shown shadowed with a $32{,}072^2$ maximum effective resolution, resolution-matched shadow map (RMSM). The scene consists of 80,000 primitives and, at 1024^2 image resolution, renders at 16–20 frames per second for a dynamic light and 34–38 frames per second for a static light with an NVIDIA GeForce 7800 GTX GPU. Bottom-left shows a shadow closeup with the RMSM, and bottom-right shows a 2048^2 standard shadow map (80–100 fps).

conventional graphics algorithms, most of our computation stages perform a varying amount of computation dependent on the results from the previous stage. As a result, we cannot employ the standard GPU measurement technique of isolating a single pass, repeatedly looping over it, etc.

All results were collected on a 2.4 GHz AMD Athlon system with 1 GB of RAM running Microsoft Windows XP. Unless otherwise specified, all experiments use an NVIDIA GeForce 7800 GTX running NVIDIA version 83.21 drivers. The implementation was optimized for the NVIDIA GeForce 7800 GTX GPU. We also show several results for the latest generation generation, the GeForce 8800 GTX (see Figure 10).

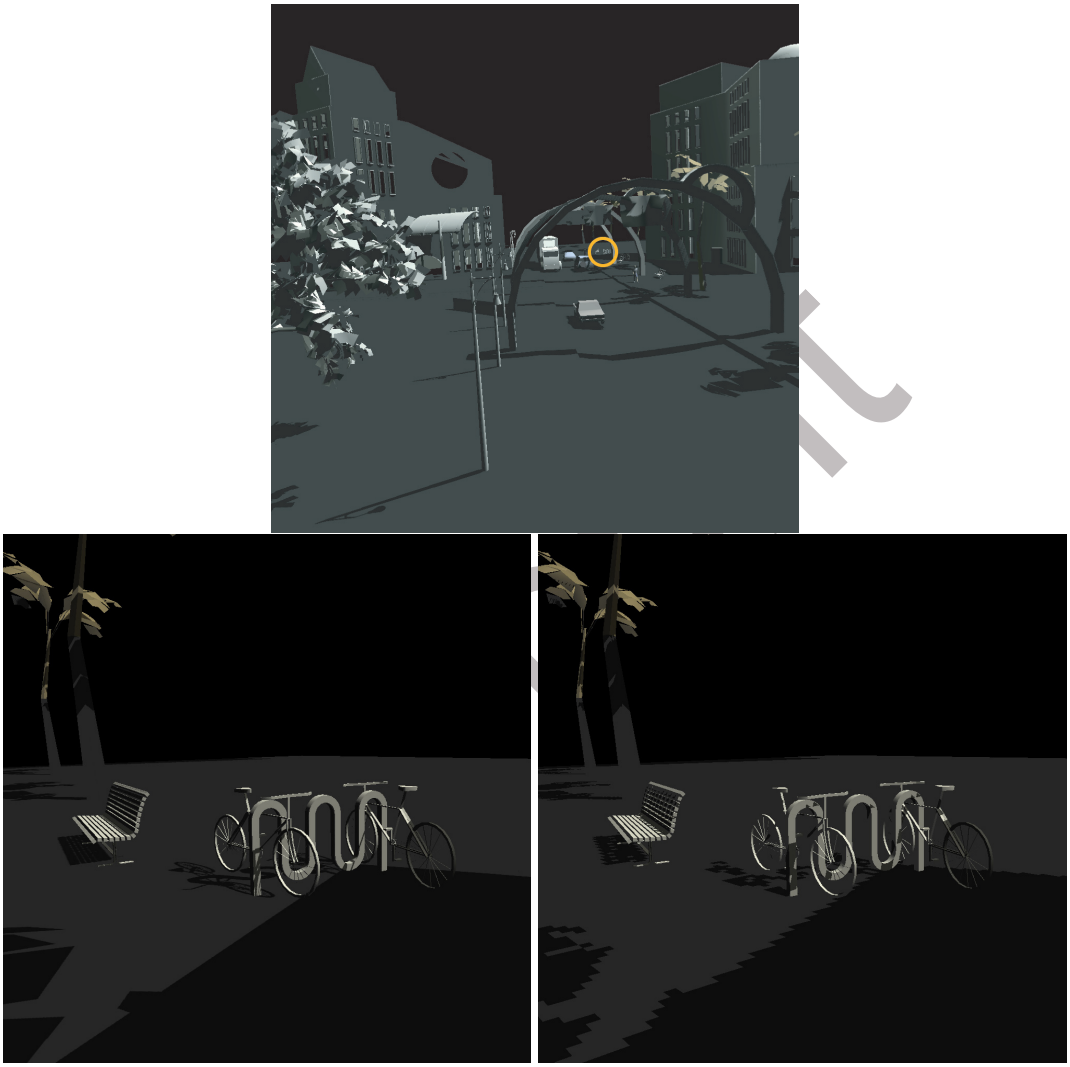


Fig. 7. Top shows the city scene shown shadowed with a $32{,}072^2$ maximum effective resolution, resolution-matched shadow map (RMSM). The scene consists of 58,000 primitives and, at 1024^2 image resolution, renders at 16–28 frames per second for a dynamic light and 20–25 frames per second for a static light with an NVIDIA GeForce 7800 GTX GPU. Bottom-Left shows a closeup of the bike rack and bench circled in the left image using RMSMs, and bottom-right shows the same closeup with a 2048^2 standard shadow map (80–100 fps).

4.2 Performance Analysis

We evaluate the performance of our algorithm in three ways: across various scenes, across image sizes, and across multiple generations of graphics processors.

Figures 8 shows the performance results for all five scenes for standard and adaptive shadow map algorithms at an image size of 1024^2. We achieve 30–38 frames per second (fps) for most static scenes and 13–28 fps for most dynamic scenes. The exception is the view of the furball scene where the furball covers the entire

Performance for static scene 1024^2 image, frames-per-second						**Performance for dynamic scene** 1024^2 image, frames-per-second				
Scene	SM	A/I/C	A/N/C	A/I/G	A/N/G	SM	A/I/C	A/N/C	A/I/G	A/N/G
Robot	80–100	6–7	0.1–1	11–15	30–32	80–100	3–4	0.5–1	4–7	13–24
Skeletons	80–100	7–9	0.5–1	16–20	34–38	80–100	5–7	0.5–1	7–8	16–20
Trees	80–100	8–9	0.5–1	12–18	30–35	80–100	3–4	0.5–1	6–7	13–14
City	80–100	6–8	0.5–1	11–13	20–25	80–100	2–3	0.5–1	6–8	16–28
Furball-W	80–100	8–9	0.5–1	20–30	30–35	80–100	1–2	0.5–1	2–3	20–25
Furball-F	80–100	5–6	0.5–1	7–10	15–17	80–100	0.5–1	0.5–1	4–5	6–7

Fig. 8. Performance comparison for both a static light and a dynamic light for a 2048^2 standard shadow map (SM) and four variants of $32{,}768^2$ GPU adaptive shadow maps (A). The ASM variants include Iterative (I) versus Non-Iterative (N) and CPU-based (C) versus GPU-based (G) scene analysis (resolution-matched shadow maps (RMSMs) are A/N/G). Furball-W is a view of the furball scene looking at the wall (Figure 1, right), and Furball-F is the same scene but looking at the furball (Figure 1, left). The right-most column shows the performance for our non-iterative, GPU-based ASM (RMSM). It achieves highly interactive frame rates for all static scenes and is 2–3 times faster than other ASM methods for dynamic scenes. Note that only the combination of *both* the non-iterative algorithm and GPU-based scene analysis results in high performance. All results are measured with an NVIDIA GeForce 7800 GTX GPU.

Performance with image size
Dynamic furball scene, frames-per-second

Image size	View	SM	A/I/C	A/N/C	A/I/G	A/N/G
512^2 image	Look at wall	70	4–7	3–4	4–10	20–30
	Look at furball	70	4–5	3–4	4–10	12–15
1024^2 image	Look at wall	40	1–2	0.5–1	2–3	20–25
	Look at furball	40	0.5–1	0.5–1	4–5	6–7

Fig. 9. Performance scaling with varying image resolution for a 2048^2 standard shadow map (SM) and four adaptive shadow map (A) variants: iterative (I), non-iterative (N), CPU-based analysis (C), and GPU-based analysis (G). Our resolution-matched shadow map (RMSM) algorithm (denoted A/N/G on far-right) is nearly the same speed for both image sizes with large, continuous visible surfaces. For the incoherent, furball view, our method requires only twice as much time to shadow 4 times as many texels. Also note that the method is up to 10 times faster than the iterative algorithm.

viewport (such as Figure 5 (left)). This difficult scene has a large number of visible surfaces and therefore generates many more page requests than the other scenes. Note that only the combination of *both* the non-iterative ASM algorithm and GPU-based data-parallel scene analysis result in real-time performance.

For most scenes, the non-iterative ASM algorithm with GPU-based scene analysis is 2–3 times faster than the iterative method, and up to 10 times faster in some cases. In no cases is it slower than the iterative method. The RMSM algorithm is 3–4 times slower than standard shadow maps for static scenes and 3–10 times slower for dynamic scenes.

The results also show that the data-parallel GPU scene analysis algorithms are required in order to take advantage of the image-shadow coherence described in Section 3.1. Using the non-iterative method with CPU-based analysis yields only 0.5–1 frame per second.

The performance of the RMSM ASM algorithm scales much better with image size than the iterative or CPU-based ASM methods. Figure 9 shows the results of the ASM variants running on two views of the furball scene at 512^2 and 1024^2 image resolution. The CPU-based methods scale linearly with the number of pixels; however, the GPU-based methods are only 1–2 times slower to shadow 4 times more pixels.

Resolution-matched shadow maps are arguably not fast enough for games released today; however, we show that the approach scales well with GPU hardware improvements and will therefore likely be viable in the near future. Figure 10 shows the performance of resolution-matched shadow maps running on three successive

Performance with GPU generations for three views
1024^2 image, dynamic tree scene, frames-per-second

GPU	GeForce 6800 GT	GeForce 7800 GTX	GeForce 8800 GTX
Zoomed out	5–6	10–12	35–40
Zoomed in	10–12	20–23	35–40
In tree	2–4	10–11	25–30

Fig. 10. Performance scaling across three generations of graphics processing units (GPUs) for our resolution-matched shadow map algorithm with dynamic geometry (recomputing entire quadtree shadow map each frame). Note that the performance improves 2–5 times with each generation of GPU.

Rendered Superpages: iterative vs. non-iterative
1024^2 image, dynamic light, number of superpages

Scene	Iterative	Non-Iterative
Robot	35–61	31–32
Skeletons	30–63	35–64
Trees	40–104	32–77
City	53–154	59–149
Furball	45–155	35–39

(a)

Number of rendered superpages as a function of image size
Dynamic furball scene, number of rendered superpages

Image size	View	Iterative	Non-Iterative
512^2 image	Look at wall	79	15
	Look at furball	42	34
1024^2 image	Look at wall	155	35
	Look at furball	49	36

(b)

Fig. 11. Comparison of the number of rendered 1024^2 superpages for dynamic scenes for the iterative and non-iterative adaptive shadow map algorithms. The iterative edge-finding algorithm requires 1–4 times more superpages (i.e., culled geometry passes) to build the quadtree shadow map than the non-iterative solution (Figure 11(a)). Figure 11(b) shows that both methods scale similarly with image resolution, and, due to the increased coherency in the larger image, they require less than two times as many passes to shadow four times as many pixels. Note that, for static scenes where the quadtree shadow map is valid for multiple frames, a very small number of superpages are required per frame.

generations of GPUs. We observe a 2–5 times speedup with each generation. If this trend continues, the technique will be practical for games released for the generation of GPU beyond the NVIDIA GeForce 8800 GTX.

However, our shadow algorithm implementations are not optimized for GeForce 8800's architecture and further improvements may make the technique practical for games on this current hardware generation. For example, Sengupta et al. [2007] describe a GPU scan implementation that takes advantage of the read-write, on-chip memory capabilities of the GeForce 8800 GTX to achieve an approximately four times speedup over the OpenGL version used in our implementation. Their paper also describes alternate sorting techniques such as a data-parallel quicksort.

Memory consumption: iterative vs. non-iterative
Number of 32^2 pages for 1024^2 image

Scene	Iterative	Non-Iterative
Robot	737–2467	3294–4666
Skeletons	1300–2200	2900–5600
Trees	2300–5500	3800–5900
City	1300–2900	3800–7100
Furball	1300–5000	4800–5900

(a)

Memory consumption with image size
Furball scene, number of 32^2 pages

Image size	View	Iterative	Non-Iterative
512^2 image	Look at wall	539	1376
	Look at furball	3506	3948
1024^2 image	Look at wall	1446	4886
	Look at furball	5056	5472

(b)

Fig. 12. Comparison of memory consumption for iterative and non-iterative (RMSM) $32{,}768^2$ adaptive shadow maps. The iterative version refines only on shadow edges, thereby using less memory but performing more slowly. The faster, non-iterative method uses approximately 1–3 times more memory (Figure 12(a)). Figure 12(b) shows the scaling of memory usage with image size. To shadow four times more pixels in the wall view, the iterative and non-iterative methods require 2.7 and 3.5 times more memory, respectively. For the incoherent, furball view, both methods require ∼1.4 times more memory. Note that static scenes, where the quadtree shadow map is valid for multiple frames, generate a very small number of page requests per frame (1–50).

4.2.1 *Geometry Performance.* Generating shadow data for the quadtree requires rendering portions of the scene at various resolutions. As described in Section 3.3, we generate shadow data by coalescing page requests into *superpages*. Each rendered superpage is one geometry pass over a subset of the scene. Figure 11(a) shows the number of superpages rendered to shadow various views of the five scenes. We see that the iterative method requires 1–4 times more render passes than the non-iterative method.

Similarly, Figure 11(b) shows how the number of geometry passes scales with image size. Due to the increased coherency in the larger image, both techniques require less than two times as many passes to shadow four times as many pixels.

4.3 Memory Analysis

Removing the iterative edge-finding step from the ASM algorithm represents a space-time tradeoff. While the previous section describes the performance benefits of the approach, this section describes the memory impact of the design. We analyze the memory consumption, utilization, and coherency for both iterative and non-iterative adaptive shadow maps.

Figure 12(a) shows the number of unique pages requested for various views of the five scenes with a dynamic light. The non-iterative method requires 1–3 times more shadow pages than the edge-finding iterative method. Figure 12(b) shows how the memory requirements scale with image size.

RMSM Memory Summary. At a 512^2 image size, with effective resolution up to $32,768^2$, a 2048^2 physical memory buffer (16 MB) is typically sufficient to store the shadow pages. With this resolution and a 32^2 page size, the page table occupies 5.3 MB, for a total of 21.3 MB. Increasing the image resolution to 1024^2, with the same shadow resolution, often requires a larger physical memory buffer for shadow pages. We tested up to the maximum size of 4096^2. A 2048^2 shadow buffer is sufficient for a 1024^2 image size if the maximum shadow resolution is limited to 8192^2.

4.3.1 *Memory Usage Efficiency.* In Section 3.1, we asserted that shadow references in real scenes exhibit substantial locality within a page. We expect that any shadow page we render will result in a significant number of references into that page from the scene.

Figure 13 shows that this assertion is correct for the majority of the camera path through our tests scenes.

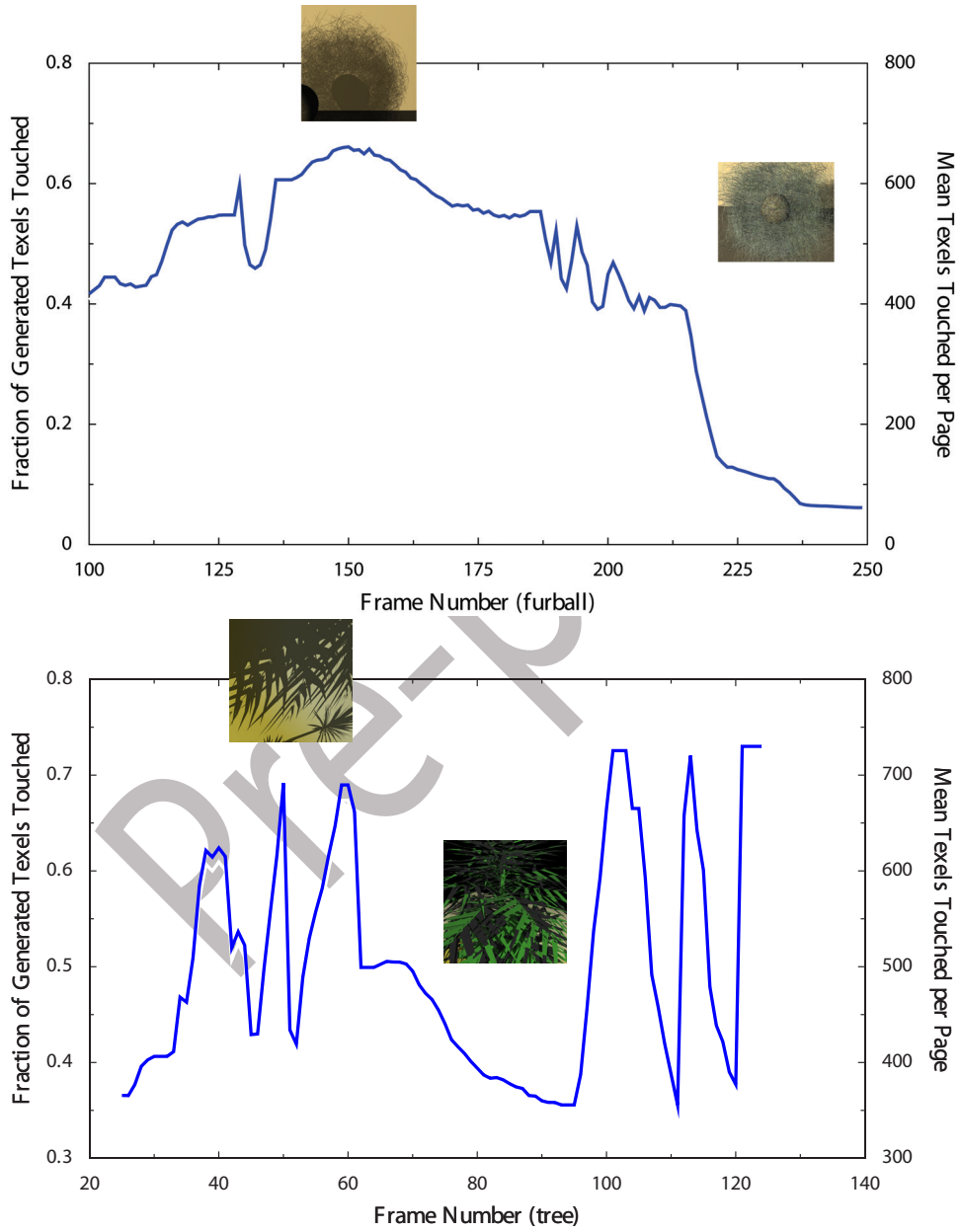


Fig. 13. These graphs show the fraction of the generated shadow data actually used to shadow the image throughout camera paths through two different scenes. In these scenes, shadow pages each had 32 × 32 shadow entries. Over these scenes, we observe substantial locality of reference within a page. Coherent shadow receivers (e.g., planar objects) deliver the best locality, while incoherent receivers (e.g., hair or leaves) result in less coherence between shadow accesses and therefore less efficient use of shadow memory.

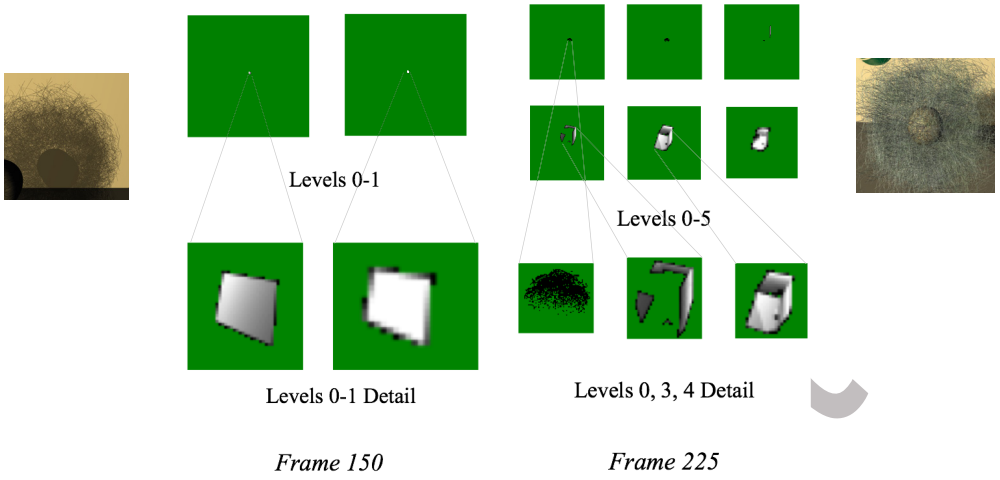


Fig. 14. Visualization of pages allocated in the quadtree data structure we use to store shadow pages. We show levels-of-detail (LOD) for two frames of the furball flythrough. The left images are from frame 150, which had a fairly coherent shadow receiver. The right images, from frame 225, had a highly incoherent shadow receiver. The top images are visualizations of the entire LOD; the bottom images are closeups of their interesting regions. In the visualizations, pages that were never requested and are thus never allocated are colored green. Pages that were requested are colored from black (few texels in the page were accessed) to white (all texels in the page were accessed) according to the number of unique accesses made to that page. Note that regardless of if the shadow receiver is coherent or incoherent, most potential pages are neither touched nor allocated. Also note that even with incoherent receivers, most pixels in accessed pages are white or light gray, showing that accesses into the quadtree data structure generally exhibit substantial locality.

Despite a large variation in the shadow complexity over these frames, all frames have substantial locality within a page, ranging from dozens to hundreds of unique shadow texel accesses per page. Because of this locality, a significant fraction (on average, about half) of all generated shadow texels are actually used in the scene. Note that coherent shadow receivers such as walls achieve the highest locality and most efficient use of shadow memory (50%–70%), while incoherent receivers such as hair and leaves result in lower reuse (10%–30%).

Another measure of coherence is the ratio of unique page requests to rendered superpages. As described in Sections 3.3.3 and 4.2.1, we cluster the requested shadow pages into superpages and render all data for the superpage with a single geometry pass. For the uncached, non-iterative ASM algorithm, superpages reduce the number of required geometry passes by a factor of 50 to 150 (see Figures 11(a) and 12(a)). In contrast, the cached ASM algorithm benefits less from superpaging (achieving 2–20 times compression) and would benefit more from drawing individual pages using aggressive frustum culling.

Figure 14 takes a closer look at two representative frames of our furball flythrough. Note that very few of the possible quadtree pages are allocated or accessed. Those pages that are allocated, even with incoherent shadow receivers, exhibit substantial locality.

4.4 Limitations of Performance

Our implementation is limited in performance by two factors. The first is the sort step. Despite the initial compaction step that eliminates a large fraction of page requests, sort is an inherently expensive algorithm—up to 20 times as expensive as compaction—that takes a significant amount of the overall runtime. It is also a step that takes a variable amount of time; incoherent scenes will have a larger input to sort than coherent

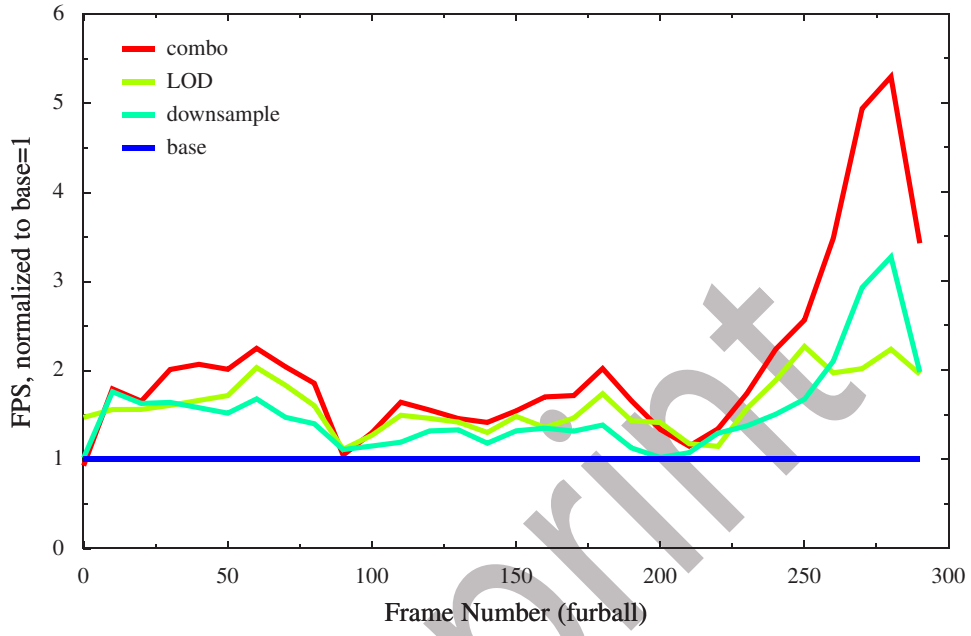


Fig. 15. The performance of our quality-runtime tradeoffs are normalized to the performance of "base", which performs full-resolution shadow coordinate analysis and supports shadow LODs down to a fine level of detail (32,768 × 32,768). Larger FPS is faster. Reducing shadow quality has the largest impact for the most complex (incoherent) frames.

scenes. We analyzed the number of elements sorted in two camera paths for both cached and non-cached RMSMs. For the incoherent furball scene, the average number of non-unique page requests for the coherent receiver views are about 3000 pages per frame. Sorting these elements takes less than 0.5 ms. When viewing the furball, however, the number of requests is ∼140,000, taking about 30 ms to sort.

Thus, for scenes with simple geometry, sort is both our largest cost and our largest source of variance. Reducing this cost requires reducing the number of initial (non-unique) page requests, which we can control in two ways: avoiding views of highly incoherent shadow receivers or reducing the resolution of the shadow analysis (Section 3.3.5). We note that users can animate these parameters of our algorithm based on camera paths if a performance guarantee is required for known paths. Also, unlike ASM's iterative refinement, the upper bound for our sort step is easily determined a priori.

In addition to sorting shadow page requests, the other dominant cost of our algorithm is generating shadow data. The cost of this stage depends on the number of unique shadow page requests, the coherence of those pages in shadow space, and a geometry engine with efficient frustum culling support. Our current implementation does not use frustum culling, but as noted in Lloyd et al. [2005], geometry engines with very efficient culling are available and would improve the performance of our shadow data generation stage.

4.4.1 *Quality-Runtime Tradeoffs.* In section 3.3.5, we describe two methods of trading off performance for quality, reducing the resolution of either the shadow coordinate analysis computation or the reducing the finest LOD in our quadtree data structure. Figure 15 shows that using either technique improves performance, and using both techniques together further improve performance. Figure 16 shows the results of various shadow resolutions on a close-up of the tree scene. In general, reducing the resolution of the analysis gives a solid gain in performance for very little loss in visual detail (Figure 16), and reducing

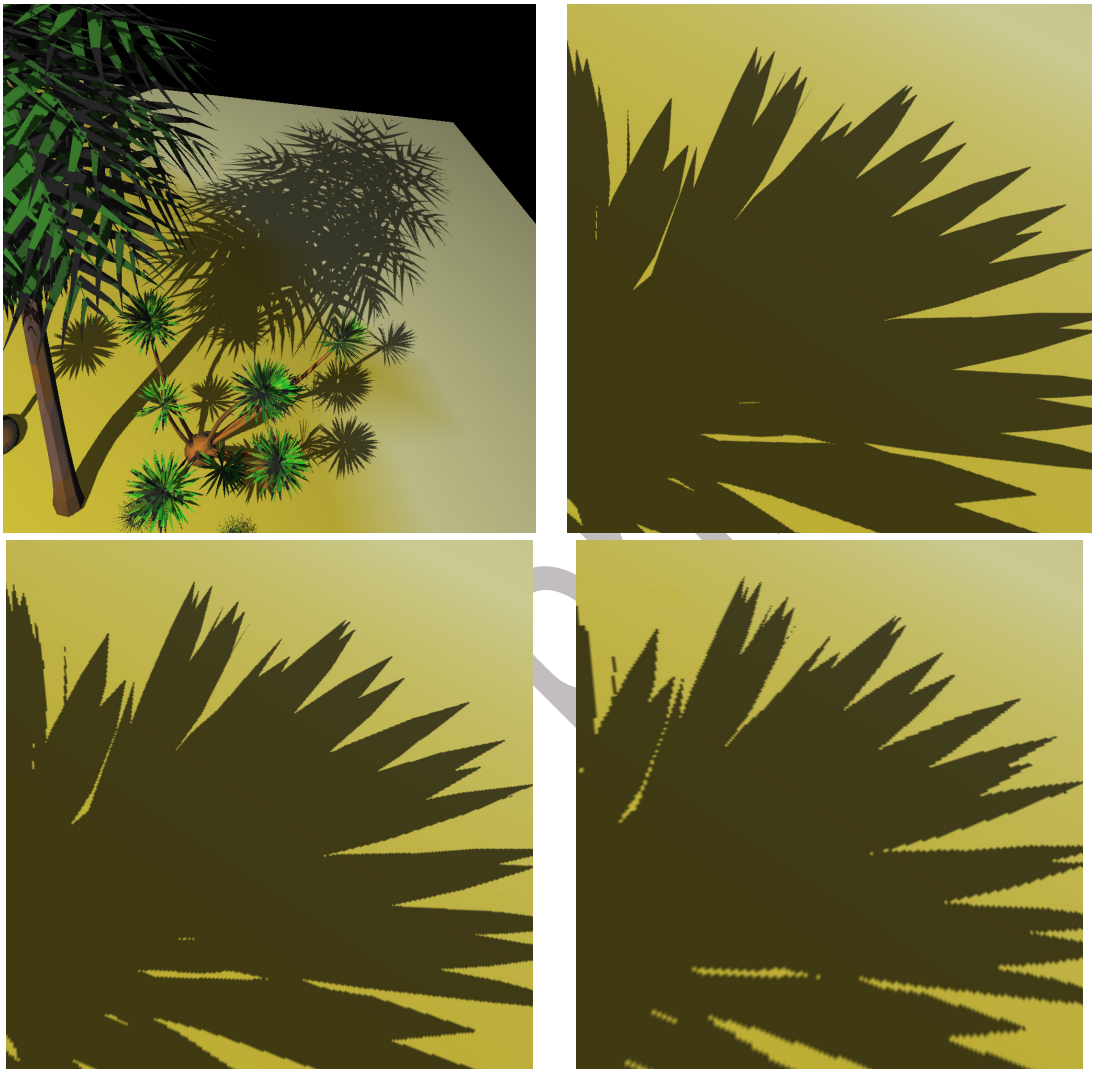


Fig. 16. Quality comparison for a range of RMSM maximum effective resolutions. Top-left shows a wide-angle view of the tree scene. We show three closeups with a $32{,}768^2$ ASM, requiring 21.3 MB of GPU memory (top-right); $16{,}384^2$ ASM, requiring 17.3 MB of GPU memory (bottom-left); and 8192^2 ASM, requiring 16.3 MB of GPU memory (bottom-right).

the resolution of the finest LOD is recommended for particularly incoherent receivers that would otherwise exhibit poor locality in the quadtree data structure.

4.5 Limitations of the Algorithm

The largest limitation of our method is the additional memory and time cost over other shadow map approaches. We've shown the technique is capable of interactively rendering complex scenes at 1024^2 image resolution on current hardware. The method achieves a 2–5 times speedup with each generation of GPUs for the last three generations, but will likely take one more generation before being appropriate for applications

such as games. However, the technique is immediately useful for applications such as interactive film preview rendering [Pellacini et al. 2005].

Another limitation is that, while RMSMs closely approximate the goal of generating alias-free hard shadows by rendering depth samples at the position requested by the shadow coordinates from the current image, the discretization of the levels-of-detail (LOD) means that the sample positions are approximate and not exact. The errors arising from the discrete LODs are not usually visible, but we have seen artifacts in shadows of very thin geometry.

5. CONCLUSION

We present an image-based shadow algorithm, resolution-matched shadow maps (RMSMs), that improves upon adaptive shadow maps (ASMs) [Fernando et al. 2001] to deliver correctly-sampled hard shadows for dynamic scenes at interactive rates on current graphics hardware. The performance of our algorithm is based on the insight that the resolution-matching property of ASMs means that, for surfaces continuously visible from the eye, adjacent eye-space pixels map to adjacent ASM shadow texels. As such, our algorithm scales with the number of continuously visible surfaces in the eye image. Like ASMs, our technique stores shadow data in a quadtree of small shadow maps; however, our algorithm is not iterative and performs 2–10 times faster than an optimized implementation of traditional iterative ASMs. In addition, our technique is guaranteed to produce correct shadows, up to the maximum resolution of the quadtree.

The second critical component of our performance is the use of data-parallel GPU algorithms for the scene analysis. Our results shows that the simplified, non-iterative ASM algorithm is completely impractical without using the compute power of the GPU to accelerate the analysis. We expect that future real-time rendering algorithms will also benefit from combining traditional graphics programming with general, data-parallel algorithms.

REFERENCES

Aila, T. and Laine, S. 2004. Alias-free shadow maps. In *Rendering Techniques 2004: 15th Eurographics Workshop on Rendering*. 161–166.

Arvo, J. 2004. Tiled shadow maps. In *Computer Graphics International*. 240–247.

Chan, E. and Durand, F. 2004. An efficient hybrid shadow rendering algorithm. In *Rendering Techniques 2004: 15th Eurographics Workshop on Rendering*. 185–196.

Chong, H. Y. and Gortler, S. J. 2004. A lixel for every pixel. In *Rendering Techniques 2004: 15th Eurographics Workshop on Rendering*. 167–172.

Crow, F. C. 1977. Shadow algorithms for computer graphics. In *Computer Graphics (Proceedings of SIGGRAPH 77)*. Vol. 11. 242–248.

Donnelly, W. and Lauritzen, A. 2006. Variance shadow maps. In *Proceedings of ACM SIGGRAPH 2006 Symposium on Interactive 3D Graphics and Games*. ACM Press, New York, NY, USA, 161–165.

Fernando, R., Fernandez, S., Bala, K., and Greenberg, D. P. 2001. Adaptive shadow maps. In *Proceedings of ACM SIGGRAPH 2001*. Computer Graphics Proceedings, Annual Conference Series. 387–390.

Forsyth, T. 2004. Practical shadows. In *Game Developers Conference 2004*. `http://www.eelpi.gotdns.org/papers/papers.html`.

Giegl, M. and Wimmer, M. 2007. Queried virtual shadow maps. In *Proceedings of ACM SIGGRAPH 2007 Symposium on Interactive 3D Graphics and Games*. ACM Press, New York, NY, USA, 65–72.

Govindaraju, N. K., Gray, J., Kumar, R., and Manocha, D. 2006. GPUTeraSort: High performance graphics coprocessor sorting for large database management. In *Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data*. 325–336.

Greene, N. and Heckbert, P. S. 1986. Creating raster omnimax images from multiple perspective views using the elliptical weighted average filter. *IEEE Comput. Graph. Appl. 6,* 6, 21–27.

Hasenfratz, J.-M., Lapierre, M., Holzschuch, N., and Sillion, F. 2003. A survey of real-time soft shadows algorithms. *Computer Graphics Forum 22,* 4 (Dec.), 753–774.

Horn, D. 2005. Stream reduction operations for GPGPU applications. In *GPU Gems 2*, M. Pharr, Ed. Addison Wesley, Chapter 36, 573–589.

Johnson, G. S., Lee, J., Burns, C. A., and Mark, W. R. 2005. The irregular Z-buffer: Hardware acceleration for irregular data structures. *ACM Transactions on Graphics 24,* 4, 1462–1482.

Lefohn, A., Sengupta, S., Kniss, J., Strzodka, R., and Owens, J. D. 2005. Dynamic adaptive shadow maps on graphics hardware. In *ACM SIGGRAPH 2005 Conference Abstracts and Applications*.

Lefohn, A. E., Kniss, J., Strzodka, R., Sengupta, S., and Owens, J. D. 2006. Glift: Generic, efficient, random-access GPU data structures. *ACM Transactions on Graphics 26,* 1 (Jan.), 60–99.

Lloyd, B., Tuft, D., Yoon, S., and Manocha, D. 2006. Warping and partitioning for low error shadow maps. In *Proceedings of the Eurographics Symposium on Rendering 2006*. Eurographics Association, 215–226.

Lloyd, B., Yoon, S., Tuft, D., and Manocha, D. 2005. Subdivided shadow maps. Tech. Rep. TR05-024, University of North Carolina at Chapel Hill.

Martin, T. and Tan, T.-S. 2004. Anti-aliasing and continuity with trapezoidal shadow maps. In *Eurographics Symposium on Rendering*. 153–160.

McCormack, J., Perry, R., Farkas, K. I., and Jouppi, N. P. 1999. Feline: Fast elliptical lines for anisotropic texture mapping. In *Proceedings of SIGGRAPH 99*. Computer Graphics Proceedings, Annual Conference Series. 243–250.

Pellacini, F., Vidimče, K., Lefohn, A., Mohr, A., Leone, M., and Warren, J. 2005. Lpics: a hybrid hardware-accelerated relighting engine for computer cinematography. *ACM Transactions on Graphics 24,* 3 (Aug.), 464–470.

Reeves, W. T., Salesin, D. H., and Cook, R. L. 1987. Rendering antialiased shadows with depth maps. In *Computer Graphics (Proceedings of SIGGRAPH 87)*. Vol. 21. 283–291.

Reshetov, A., Soupikov, A., and Hurley, J. 2005. Multi-level ray tracing algorithm. *ACM Transactions on Graphics 24,* 3 (Aug.), 1176–1185.

Segal, M. and Akeley, K. 2004. *The OpenGL Graphics System: A Specification (Version 2.0 - October 22, 2004)*.

Sen, P. 2004. Silhouette maps for improved texture magnification. In *Graphics Hardware 2004*. 65–74.

Sen, P., Cammarano, M., and Hanrahan, P. 2003. Shadow silhouette maps. *ACM Transactions on Graphics 22,* 3 (July), 521–526.

Sengupta, S., Harris, M., Zhang, Y., and Owens, J. D. 2007. Scan primitives for gpu computing. ACM.

Sengupta, S., Lefohn, A. E., and Owens, J. D. 2006. A work-efficient step-efficient prefix sum algorithm. In *Proceedings of the Workshop on Edge Computing Using New Commodity Architectures*. D–26–27.

Stamminger, M. and Drettakis, G. 2002. Perspective shadow maps. *ACM Transactions on Graphics 21,* 3 (July), 557–562.

Thrane, N. and Simonsen, L. O. 2005. A comparison of acceleration structures for GPU assisted ray tracing. M.S. thesis, University of Aarhus.

Wald, I., Purcell, T. J., Schmittler, J., Benthin, C., and Slusallek, P. 2003. Realtime ray tracing and its use for interactive global illumination. In *Eurographics 2003, State of the Art Reports*. 85–122.

Williams, L. 1978. Casting curved shadows on curved surfaces. In *Computer Graphics (Proceedings of SIGGRAPH 78)*. Vol. 12. 270–274.

Wimmer, M., Scherzer, D., and Purgathofer, W. 2004. Light space perspective shadow maps. In *Eurographics Symposium on Rendering*. 143–151.

Woo, A., Poulin, P., and Fournier, A. 1990. A survey of shadow algorithms. *IEEE Computer Graphics & Applications 10,* 6 (Nov.), 13–32.

Received January 2007.