Fast Hard and Soft Shadow Generation on Complex Models using Selective Ray Tracing

UNC CS Technical Report TR09-004, January 2009

Christian Lauterbach*
University of North Carolina at Chapel Hill

Qi Mo†
University of North Carolina at Chapel Hill

Dinesh Manocha‡
University of North Carolina at Chapel Hill

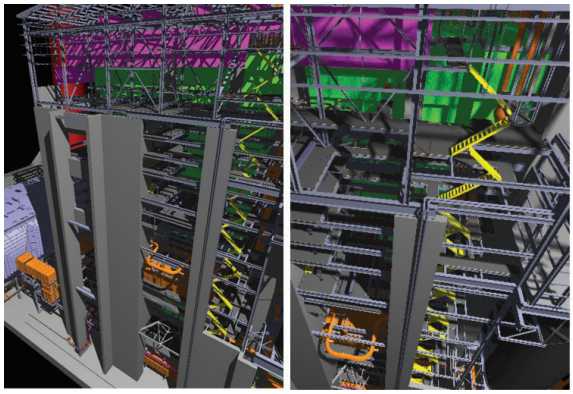


Figure 1: Soft shadows: *Our algorithm can render the 12.7M triangle Powerplant model at 16 fps with hard shadows (left) and over 2 fps with soft shadows with 16 light samples (right) running on a NVIDIA GTX 280 GPU.*

Abstract

We present fast algorithms to generate high-quality hard and soft shadows in complex models. Our method combines the efficiency of rasterization-based shadow mapping approaches with the accuracy of a ray tracer based on conservative image space bounds. The algorithm can handle moving light sources as well as dynamic scenes. In practice, our approach is able to generate shadows on CAD and scanned models composed of millions of triangles at close to interactive rates on current high-end GPUs.

1 Introduction

The state of the art in interactive rendering is constantly moving towards greater physical realism and detail. This is primarily driven by the rapid increase in performance provided by commodity GPUs which are able to utilize parallelism along with high coherence in memory accesses and computations to rasterize models with millions of triangles at interactive rates. However, one of the challenges is generating images with high-quality hard and soft shadows. Many rasterization-based approaches can be used to generate shadows, but when evaluating them on large, complex models they typically can only provide either high performance or accuracy, but not both. On the other hand, ray tracing provides a simple solution to generate accurate hard and soft shadows with good scalability for large in-core models [Foley and Sugerman 2005; Günther et al. 2007]. Despite recent advances and good use of parallelism, current ray tracing implementations are one or two orders of magnitude slower than rasterization approaches on GPUs.

Shadow volumes can be used to generate high-quality hard and soft shadows, but their complexity can increase considerably with model complexity and number of silhouette edges. The fastest techniques for hard shadows are based on regular shadow maps, which are well supported by current rasterization hardware. However, these algorithms can suffer from aliasing errors. It is possible to overcome these errors by using hybrid combinations of shadow maps and shadow volumes [Chan and Durand 2004] or using irregular shadow mapping [Aila and Laine 2004; Johnson et al. 2005]. However, these algorithms may not scale well to complex models composed of millions of triangles and are limited to hard shadow generation.

Main results: In this paper, we present a *selective ray tracing* algorithm to generate accurate hard and soft shadows on current many-core GPUs. Our approach is general and the overall image quality is comparable to that of a fully ray-traced shadow generation algorithm. Moreover, the algorithm can efficiently handle complex models, as long as the model and its bounding volume hierarchy can fit into the GPU memory.

Our hybrid formulation performs conservative rasterization based on shadow mapping techniques and identifies the regions or the pixels of the frame (called PIP) that are potentially inaccurate due to under-sampling in light space or missed primitives due to rasterization errors. We use selective ray tracing to compute correct shading information only for those pixels in PIP, as described in Section 3. We also present an efficient technique for coupling with a GPU ray tracer to provide the missing information for those pixels. In addition, our approach extends very easily to area light sources and can be used to generate high-quality stochastic soft shadows. In practice, the subset of inaccurate pixels is small and our approach shoots relatively few rays as compared to a full ray tracer, resulting in a significant speedup.

The notion of performing hybrid shadow generation based on conservative rasterization is not new. As compared to prior hybrid approaches, our algorithm offers the following advantages:

- High-quality shadows: In our complex benchmark scenes the hard and soft shadow generation algorithms result in almost no perspective or projective aliasing artifacts.
- Efficiency: We use compact hierarchical representations to accelerate ray intersections and allow ray tracing on GPUs with low memory overhead. As compared to pure shadow mapping, our hybrid algorithms are only 30-70% slower, but about $4-10$ times faster than full ray tracing.
- Hardware utilization: Our approach maps directly onto current GPUs and needs no special hardware support or changes to existing architectures. It uses less memory bandwidth than full ray tracing and its performance should increase with the growth rate of future GPUs, as described in Section 4.

We demonstrate our implementation on several models ranging from game-like scenes with dynamic objects to massive CAD models with millions of triangles at interactive rates on a high-end GPU, as described in Section 5.

*e-mail: cl@cs.unc.edu
†e-mail:qmo@cs.unc.edu
‡e-mail:dm@cs.unc.edu

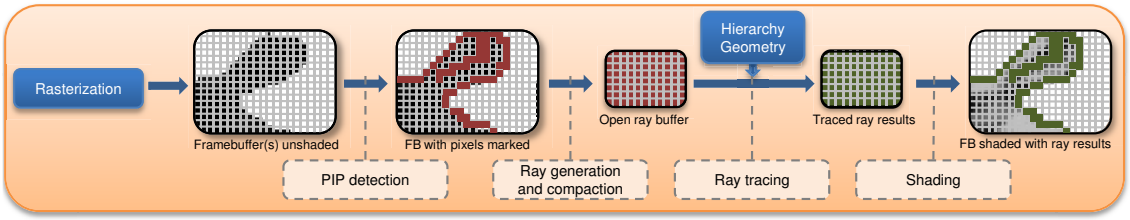


Figure 2: Overview: *Pipeline model of our hybrid rendering algorithm. After GPU-based rasterization is run, the PIP computation detects and marks pixels that need to be ray traced. The ray generation step generates a dense ray list from the sparse buffer of potentially incorrect pixels and then generates one or more rays per pixels as required. A selective ray tracer traces all the rays using the scene hierarchy and then applies the results to the original buffer. Finally, the pixels are shaded based on the ray results.*

2 Previous work

We give a brief overview of related work limited to interactive shadow generation and reducing aliasing errors and refer the readers to [Hasenfratz et al. 2003; Lloyd 2007; Laine 2006] for recent surveys on shadow algorithms.

Hard shadows: At a broad level, prior techniques to alleviate aliasing artifacts using rasterization methods are based on shadow maps [Williams 1978] and shadow volumes [Crow 1977]. Some hybrid approaches have been proposed that combine shadow mapping and volumes [McCool 2000; Chan and Durand 2004] that can improve shadow volume performance and allow interactive high-quality shadows on simple scenes. Most current interactive applications use variants of shadow mapping, but may suffer from aliasing problems. Many practical algorithms have been proposed to alleviate perspective aliasing [Stamminger and Drettakis 2002; Wimmer et al. 2004; Lloyd 2007] as well as projective aliasing [Lefohn et al. 2007]. Other shadow mapping algorithms can eliminate blocking artifacts [Aila and Laine 2004; Johnson et al. 2005] by implementing a rasterizer that can process arbitrary samples on the image plane.

Soft shadows: In general, soft shadows can be implemented by sample-based methods such as using averaging visibility from multiple shadow maps to calculate visibility or ray tracing. Both these methods can be slow, so many approaches have been developed to generate plausible soft shadows with methods such as post-filtering shadow maps or special camera models [Mo et al. 2007], which produce correct results only for simple scenes. More accurate approaches evaluate the light source visibility from the image samples by back-projection [Assarsson and Akenine-Möller 2003; Schwarz and Stamminger 2007; Sintorn et al. 2008; Bavoil et al. 2008] or by generating shadows from environment lighting [Annen et al. 2008]. Techniques using irregular z-buffering have also been extended for soft shadows on a proposed new architecture [Johnson et al. 2009].

3 Shadows using selective ray tracing

In this section, we present our algorithms for generating high quality shadows based on hybrid rendering and selective ray tracing. The main idea behind selective ray tracing is to only shoot rays corresponding to a small subset of the pixels in the final image in order to accelerate overall rendering. Our assumption is that we have underlying fast rasterization algorithms such as shadow mapping that compute the correct result for most of the frame, but may include localized error such as aliasing artifacts in parts of the image. We try to identify these regions of potentially incorrect pixels (PIP) in a conservative manner, since any additionally selected pixels will not change the image whereas missed ones may result in artifacts. As Fig. 2 illustrates, this is a multi-step process that starts with the results of a GPU shadow algorithm that provides a first approximation to the desired result. As a next step, we test the accuracy of each pixel and classify it accordingly, marking some pixels in the buffer as potentially incorrect. The buffer with all marked and unmarked pixels is then passed into the ray tracer where the first step filters out all non-marked pixels and keeps just the potentially incorrect pixels in a compact, non-sparse form. For each pixel in the PIP set, we shoot one or more rays to compute the correct visibility or shading information for the shadows. All rays are stored in a dense list that can be used as input for any data parallel, many-core GPU ray tracer. After the rays have been evaluated, the results are then written back to the original pixels in the PIP set. Back in the rasterizer, a shading kernel is used to compute colors for all the pixels.

3.1 Hard shadows

Shadow mapping is one of the most widely used algorithms for generating hard shadows for interactive applications. It works on general, complex 3D static and dynamic scenes and maps well onto current GPUs. However, the shadow maps may need high resolution to avoid aliasing artifacts. These errors can be classified into *perspective* and *projective* aliasing [Stamminger and Drettakis 2002]. Perspective error occurs due to the position of the surface with respect to the light and viewpoint and result in the 'blockiness' of shadows in the algorithm. Projective error stems from the orientation of the receiver to light and viewpoint. Geometric errors from under-sampling can also include missing contributions from objects that are too small or thin in light space, e.g. surfaces that are oriented close to coplanar with the view direction. Artifacts from this error result in missing and interrupted shadows for these objects. Finally, shadow map self-shadowing error occurs from inaccuracies in the depth values computed in the light view and the camera view. It stems both from depth buffer precision (numerical error) as well as orientation of the surface (geometric error). We consider this mainly an artifact that can be minimized by increasing depth precision and using slope-dependent bias and do not address it directly in our algorithm.

Pixel classification: We now discuss our method for estimating the set of pixels in the image that can potentially be incorrect (PIP) due to the error described above. We first note that most of the error in shadow mapping appears at shadow boundaries while the shadow interiors (as well as the interiors of lit regions) tend to be accurate. Thus, we can assume that when we look up the corresponding sample in the shadow map for a given image sample, it should not be considered accurate if it is adjacent to an edge in the shadow map. We therefore modify the standard depth buffer look-up to test the 8 texels around the sample value with depth s in the shadow map and find the maximum absolute difference $\Delta = \max(\|s - s'\|, s' \in \text{depth around S})$ in depth. If Δ exceeds a threshold value, e.g. a fraction of the possible scene depth as determined by z-buffer near and far planes, we assume that there is a shadow edge at this image pixel. In this case, we need to make sure

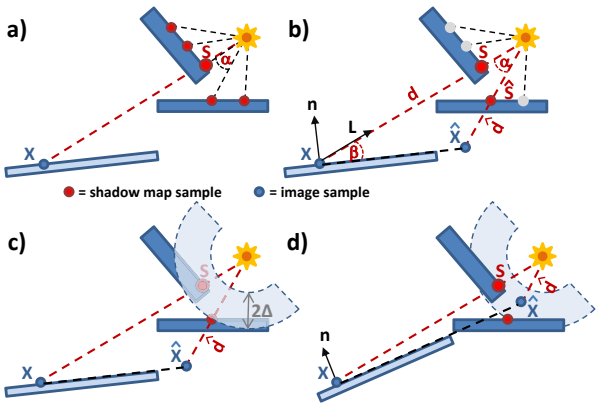


Figure 3: PIP computation: *a) For a given image sample **X** we project back to the shadow map and find the corresponding sample **S**. We then test the depths of the surrounding shadow map samples and select the one with the maximum difference Δ in depth value to **S** and label it as $\hat{\mathbf{S}}$. b) We now determine whether the surface at **X** can be affected by an edge at **S** and $\hat{\mathbf{S}}$ by finding the closest point $\hat{\mathbf{X}}$ on the surface within the angle α of one shadow map pixel and find its depth $\hat{d}$. c) If $\hat{d}$ is within the shaded region of depth Δ around **S** or on the other side of the region as seen from **X**, then it needs to be ray traced. In this case, the pixel can be classified as shadowed. d) Counter-example: $\hat{\mathbf{X}}$ is in the region and thus the pixel is ray traced.*

this edge can affect the shadow generation at the current shading sample **X** that is at distance d from the light source (see Fig. 3.1 for illustration.) This is to prevent that a relatively small shadow discontinuity at one side of the model does still affect pixels far away. To achieve this, we find the closest distance to the light $\hat{d}$ that the receiver surface can reach within the angle α of one texel of the shadow map. Intuitively, the closer to parallel the receiver surface is to the light direction, the larger the difference between $\hat{d}$ and d becomes. If $\hat{d}$ overlaps the interval $[s-\Delta, s+\Delta]$ or d and $\hat{d}$ are on different sides of **S** then we mark the pixel as part of the PIP set.

The actual computation of $\hat{d}$ for a local point light source follows from trigonometry such that $\hat{d} = d \sin\beta / \sin\alpha+\beta$. Given the normal vector **n** and normalized light direction **L**, then this is easily computed by using $\sin\beta = \mathbf{n}\cdot\mathbf{L}$. A similar calculation for directional lights is relatively straightforward. Note that is also possible to precompute Δ for the shadow map and store it for each pixel in addition to the depth value. This may be useful if the light source is mostly static since it reduces the memory bandwidth needed during the lighting pass of the rasterization algorithm.

One case that the method above does not detect is the geometric aliasing problem when a primitive is too small (e.g. a thin wire) as seen from the light view and thus not even drawn during scanline rendering. Some of the pixels that are actually covered by the object may not get rasterized and thus not detected during edge filtering. This can cause missing shadows regions in the actual image. Our solution is to identify these small objects during rasterization from the light view and employ conservative rendering techniques to assure that they are actually part of the PIP set. We use geometry shaders to implement the conservative triangle rendering method described in [Hasselgren et al. 2005] and modified by [Sintorn et al. 2008]. In essence, each triangle is transformed into a polygon with 6 corners by extruding at the original vertices. The new extruded primitive is then guaranteed to cover the center point of each pixel that it touches. Since extending all the triangles in the scene could be extremely costly, the shader also tests the area and aspect ratio of each triangle in image space and only uses conservative rendering for those that are thin and thus likely to be missed. At the same

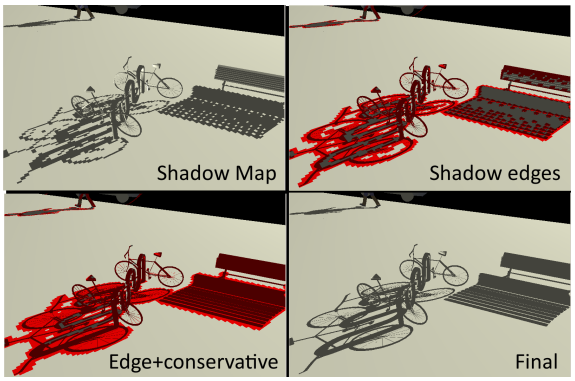


Figure 4: Detecting shadow artifacts: *Shadow on City model. Top left: shadows with shadow mapping at 2048^2 resolution. Top Right: pixels marked for ray tracing in red. Bottom left: pixels marked by conservative rendering. Bottom right: final result. The image is identical to the fully ray-traced result.*

time, this technique automatically avoids very small but regular triangles (e.g. as in scanned models) where conservative rendering is not needed. Note that an even more efficient culling method would detect whether the triangle also has a silhouette edge, but we found that this adds more constraints on the rendering pipeline by having to provide adjacency information. Figure 4 illustrates the effect of our conservative rendering approach.

3.2 Soft shadows

We now describe an extension of the approach presented above that can also be used to render high quality soft shadows. Ray tracing approaches commonly stochastically generate a number of samples on the area light source for each hit point, evaluate their visibility using shadow rays and then average the results to find an estimate of how much of the light source is visible from the hit point. However, a high number of samples and shadow rays are needed to avoid high frequency aliasing error. We observe that the only area that needs to be evaluated from multiple light samples is the penumbra region, because for umbra and fully lit regions the visibility of the light is binary. This enables us to handle those binary regions using rasterization-based techniques, e.g. shadow mapping, and to identify the penumbra as potentially incorrect pixels (PIP), and thereby perform selective ray tracing on these pixels. They only correspond to a portion of the final image, so significant amount of computation can be saved. More importantly, ray-based computation can be directed to regions that need it the most.

One standard shadow map taken from the center of the light suffices for our approach to estimate the penumbra region. For pixels in the image plane that do not fall in our estimated penumbra no shadow rays are needed and they will be shaded by shadow mapping. The penumbra is identified in the following manner. First we compute edge pixels in the shadow map in the same way we do for hard shadows. For hard shadows those edge pixels are the potentially inaccurate pixels, while for soft shadows they need to be expanded to account for the effect of the area light sources. Next we compute the projection of the area light onto the shadow map, centered at each of these silhouette pixels. This is equivalent to splatting each edge pixel using the shape of the light and a size based on the depth difference between the light and the edge. After splatting all the edge pixels we get a mask which is a union of all the splats in the map. Masked pixels are potentially in penumbra while unmasked pixels are either in the umbra or fully lit. We use this in a similar way as a shadow map. During rendering any points that are projected inside the masked area belong to the estimated penumbra

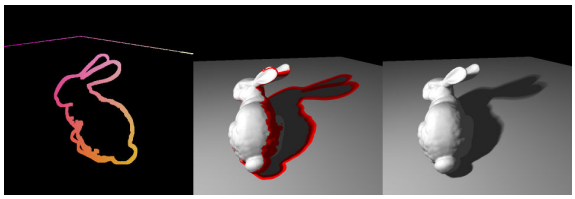


Figure 5: Penumbra classification: *Soft shadow on Bunny model. Left: mask in the shadow map formed by splatting edge pixels. Middle: penumbra pixels marked for ray tracing in red. Right: final result.*

and are ray traced using multiple shadow rays (e.g. 8×8 sampling), while points that are projected inside unmasked area are classified as umbra or fully lit based on the original shadow map. Figure 5 shows an example of this process.

The advantage of this approach is that it estimates the penumbra with the cost of little more than a standard shadow map, and consequently large parts of the image can be exempted from shadow ray tracing. Admittedly not all silhouette edges can be captured because the shadow map is generated only from the center of the light, but because such missed edges are often in the vicinity of the edges that are actually identified, it is very likely that most of the penumbra regions in the final image plane have been correctly identified. The accuracy can be further improved by computing shadow maps from multiple samples on the area light source, e.g. the corners, and then computing the union of masked pixels from each.

4 Analysis and Comparison

4.1 Comparison

A very important aspect for evaluation of our algorithm is the difference in image quality compared to the full ray tracing solution. We present a detailed comparison for several benchmark scenes in the appendix that show the original image generated with shadow mapping at 1024^2 resolution, selective ray tracing results and reference full ray tracing (with a difference image). In practice, for hard shadows we have observed that our algorithm computes images that are almost error free and virtually identical to fully ray traced results for all our benchmark scenes, with differences arising from biasing errors. Unfortunately, we cannot guarantee full correctness since some features such as very small holes inside a solid object could theoretically be missed due to the regular sampling in light space (i.e. geometric aliasing errors). However, these artifacts appear to be very rare.

It is hard to directly compare the quality of our results with only rasterization-based approaches. The fast shadow mapping methods based on warping and partitioning may not account for projective aliasing [Stamminger and Drettakis 2002; Wimmer et al. 2004; Lloyd 2007] and their accuracy can vary based on the relative position of the light source w.r.t. the viewpoint. Many techniques to handle projective aliasing [Lefohn et al. 2007] and alias-free shadow maps [Aila and Laine 2004; Johnson et al. 2005] can be implemented on current GPUs. These approaches can generate high quality shadows, but it is not clear whether they can scale well to massive models. For soft shadows, most of the accurate methods may not be able to handle complex models at interactive rates on current processors. Recently, Sintorn et al. [2008] presented a soft shadow algorithm that can handle models with at most tens of thousands of triangles at interactive rates. It uses a more accurate method for penumbra computation, but its scalability on large models with moving light sources is not clear.

Our hybrid approach shares the same theme as other hybrid shadow generation algorithms that use shadow polygons and LODs [Govindaraju et al. 2003] or shadow volumes [Chan and Durand 2004]. However, LODs can affect the accuracy of the shadow boundaries. Moreover, the complexity of shadow volumes tense to increase with the number of silhouette edges in complex models. Thus, the bottleneck becomes the fill rate needed for rendering all volumes, which becomes prohibitively large with high geometric complexity. The analysis in [McGuire 2004] shows that the overall expected number of silhouette edges for a model is proportional to the sum of the dihedral angles. As an example, the average dihedral angle per primitive on Powerplant model is about 6 times the average angle on the Buddha model. Based on our experiments with several viewpoints, over 4 million silhouette edges may need to be rendered per frame for shadow volumes on Powerplant. The hybrid method in [Chan and Durand 2004] uses shadow mapping and employs a simple discontinuity detection on the depth map, then performs selective rasterization for the marked pixels using hierarchical z-culling. However, hybrid shadow volumes still have significant drawbacks compared to our approach. First, generating and processing the volumes including silhouette computation can be expensive especially for large CAD models with complex topology such the models used here. Second, even though shadow volumes may need to be selectively rasterized for only a small set of pixels, all shadow volumes still have to be processed by the rendering pipeline in any case. Hierarchical culling in the hardware may cull areas of the image efficiently, but the hybrid approach will most likely decrease the efficiency of the GPU rasterizer since active pixels will be relatively sparse and the parallel rasterization units are not sufficiently utilized by rendering only a few pixels inside a block of pixels. Finally, our approach provides a much improved algorithm for detecting whether depth continuities actually affect each pixel, thus reducing the number of pixels ray traced on complex models, while our conservative rendering approach detects contributions from small objects that would be missed by the previously described approaches.

Model	Geometry	Full RT	SRT	SRT+SM	SRT % Rays
City	2 MB	1066 MB	113 MB	222 MB	5
Sibenik	2.2 MB	2280 MB	224 MB	859 MB	4
Buddha	27 MB	3148 MB	601 MB	1144 MB	12

Figure 6: Memory bandwidth: *Simulated memory bandwidth requirements for rendering one frame at 512^2 image resolution with selective (SRT) and full ray tracing (FRT) on several models (total storage for geometry and BVH is given in second column to show the total working set size.) The last column shows the time for SR plus a very conservative estimation of bandwidth needed by scanline rendering of the shadow map.*

4.2 Performance analysis

In this section, we show that our hybrid algorithm maps to current many-core GPUs and has lower memory bandwidth requirements as compared to full ray tracing. A key issue in designing GPGPU or related algorithms on current highly parallel architectures is to ensure that they are not limited by memory bandwidth. This is mainly because the growth rate for computational power far exceeds that of memory bandwidth. The streaming model of computation used in the rasterization pipeline has been shown to be very successful in this regard with memory bandwidth being mostly used for depth and frame buffer accesses.

We analyzed the memory bandwidth requirements when running both selective and full ray tracing for two of our benchmark models, in particular the memory bandwidth used by the actual ray tracing kernel. Since current GPU architectures have limited cache sizes, i.e. only a texture cache, such analysis is simpler than for CPUs. We implemented a simple software simulator that emulates the behavior of the memory unit for global memory accesses in CUDA in device emulation mode running on the host CPU. Care has to be

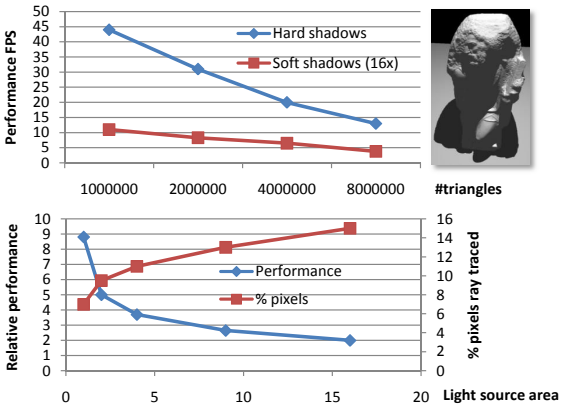


Figure 7: Scalability: *Top: Performance vs. model complexity on several simplification levels of the St. Matthew model, showing close to logarithmic scaling. Bottom: Performance vs. light source size on Buddha model.*

Benchmarks	Tris	Hard			Soft		
		SM	SRT	FRT	SM	SRT	FRT
City	58K	256	103 (3%)	21	200	19 (9.8%)	3.7
Sibenik	82K	150	66 (4%)	13	47	7.3 (6%)	0.64
Buddha	1M	42	29 (1.4%)	8	34	9 (7.7%)	2.5
Powerplant	12M	25	16 (7.1%)	4	4.4	2.1 (11%)	0.8

Figure 8: Performance: *Performance of our selective ray tracing (SRT) approach on our benchmark models, compared to the simple shadow mapping algorithm (SM) with one point light for hard and 4 point lights for hard shadows, as well as a fully ray traced solution (FRT). The percentages show the fraction of pixels marked for ray tracing. All numbers are frames per second (FPS) at* 1024^2 *screen resolution.*

taken mainly to correctly account for the behavior of the memory unit in combining data parallel accesses to contiguous memory. Our results (see Fig. 6) indicate that selective ray tracing consistently only uses about an order of magnitude less memory bandwidth per frame than full ray tracing. The bandwidth for selective ray tracing is still slightly higher than expected from the number of rays compared to full ray tracing. This is due to the fact that the ray groups in selective ray tracing are more incoherent and thus will access more memory locations. In order to perform a complete analysis, we also need to compute the memory bandwidth needed by the rasterizer for scan-line rendering of the shadow map. However, this is not possible for hardware accelerated rasterization (i.e. current GPUs) since the implementation details are not publicly available. However, we provide an estimate for the bandwidth used for rasterizing the shadow map by multiplying the peak bandwidth on the GPU by the time taken for rasterization, thus providing us a conservative upper bound. We list the summed memory bandwidth for shadow mapping plus selective ray tracing in the last column of Fig. 6. Note that the combined bandwidth is still significantly lower than full ray tracing.

4.3 Scalability analysis

We also demonstrate the scalability of our approach with model complexity. To eliminate bias introduced by different model characteristics, we use different simplification levels of the same model and then compare the time used for selective ray tracing for each of the levels. Our results in Fig. 7 show that we can achieve sub-linear scalability with model size due to the use of hierarchical structures and occlusion culling. We also look at the performance implications of increasing the area of the light source for soft shadow rendering (see Fig. 7 bottom). Similar to other approaches our performance decreases significantly with larger light sources mostly due to more of the pixels being in penumbra regions and subsequently being ray traced.

5 Implementation and Results

5.1 Implementation

We have implemented the algorithms described above on a NVIDIA GPU. We use OpenGL with Cg as the rendering interface and CUDA for general-purpose programming. Our ray tracer uses a BVH built on the GPU as the acceleration structure with a simple stack-based traversal similar to the one described in [Günther et al. 2007]. This also allows us to handle dynamic scenes by rebuilding the hierarchy each frame. A compaction step groups all the rays generated for marked pixels into a dense buffer that is then subdivided into small packets, each of which is handled independently. For rendering massive models, memory for storing the geometry and hierarchy on the GPU becomes an issue. We use a variant of the ReduceM representation [Lauterbach et al. 2008], but we modify the strip representation such that it is possible to also directly rasterize strips via OpenGL in order to use the same representation both for rasterization and ray tracing. In addition, we also use the top levels of the scene BVH for view frustum culling and occlusion culling based on occlusion queries both from the light as well as the camera view. These culling methods drastically accelerate the performance of the rasterization algorithm for large models. In our current implementation, view frustum culling is currently performed on the CPU, but could also be easily implemented in a CUDA algorithm. For soft shadows, we use a simple stratified sampling scheme for the shadow ray samples that is computed for each pixel with a simple random number generator inside the ray generation kernel. We base the random seed on sample location which allows us to compare our results for selective and full ray tracing without having to isolate variance in the estimate.

5.2 Results

We now present results from our implementation running on a Intel Core2 Duo system at 2.83 GHz using a NVIDIA GTX 280 GPU running Windows XP. Fig. 8 summarizes the timings for hard and soft shadows at 1024×1024 screen resolution, including comparison timings for GPU algorithm only, selective ray tracing and full ray tracing. The data for selective ray tracing also shows the percentage of pixels in the PIP set. We selected a wide range of benchmark models from relatively low-complexity game-like environments to high-complex scanned, CAD and architectural models to highlight the scalability of the algorithm. For shadows, all timings are for a moving light source, i.e. the shadow map is generated per frame, and soft shadows are generated by using 16 samples/pixel. Note that our algorithm is typically 5 times faster than full ray tracing.

We present a detailed analysis of the timing breakdown in selective ray tracing (see Fig. 9) for several of our benchmark scenes. There is a relatively constant overhead associated with PIP detection, ray compaction and parts of the implementation such as the overhead of CUDA/OpenGL communication in the current programming environment. Note that the actual tracing of the selected ray samples only makes up a fraction of the time spent. While we do not explicitly show the overhead in rasterization introduced by conservative rendering in the graph above, we have found that in practice it slows down the rasterization step by about 10% since only a relatively small set of primitives are rendered conservatively.

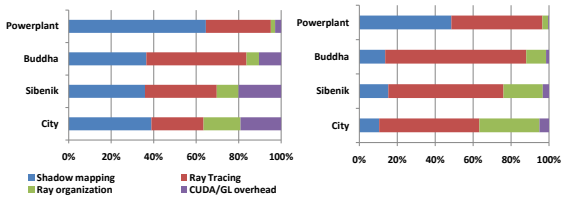


Figure 9: Timings: *Time spent in different parts of the algorithm for hard (left) and soft shadows (right). Rasterization includes both shadow map generation, frame buffer rendering and shading. Ray tracing includes time spent in the ray tracing kernel only, while ray generation is the time for generating the compact ray buffer from sparse frame buffer and writing back at the end. CUDA/GL times represent the cost for buffer transfers and similar as the overhead of switching between OpenGL and CUDA.*

Figure 10: Benchmarks: *Soft shadows using 16 light samples in Sibenik cathedral model, running at 7 fps at* 1024^2 *resolution.*

6 Limitations and future work

Overall, the main determining factor for our algorithm is the size of the PIP set. If the set is too large, then our algorithm cannot achieve a significant speedup over full ray tracing; however, at worst it can also only be slightly slower to the extent of shadow mapping overhead. In addition, the rays generated by selective ray tracing may exhibit less ray coherence than in full ray tracing which means that tracing these rays will be slightly more expensive on a per-ray metric. Since the rays still can access any part of the model, we also can only render models that fit into GPU memory and need to store an additional ray tracing hierarchy as well as update or rebuild it for deformable models. Our approach may also still carry over some of the geometric errors from rasterization such as depth buffer errors and resulting shadow map bias. The accuracy of our soft shadow algorithm is also governed by the underlying sampling algorithm.

We presented new algorithms for selective ray tracing that augment existing fast rasterization approaches for shadows. In practice, they can generate high-quality hard and soft shadows and are about 5 times faster than ray tracing on current GPUs. Our approach is robust and scalable with geometric complexity. We also analyzed its bandwidth requirements compared to full ray tracing and demonstrate that our approach maps well to current architectural trends.

There are many avenues for improvement. For one, the shadow accuracy detection could be made less conservative by using a better edge representation in shadow maps, such as silhouette maps [Sen et al. 2003], but as a trade-off the shadow mapping may be slower. Our approach should also be directly applicable to screen-space ambient occlusion approaches. An important aspect in the pipeline is the implementation of the actual ray tracing algorithm. The traditional ray tracing paradigm using accelerating structures means that all the geometry needs to be stored for random access during ray tracing, which may be incompatible with the GPU streaming model. One interesting solution here is to use ray hierarchies, i.e. a hierarchy that is built on top of the total set of rays and is intersected with the scene.

Acknowledgments

This work was supported in part by ARO Contracts DAAD19-02-1-0390 and W911NF-04-1-0088, NSF awards 0400134, 0429583 and 0404088, DARPA/RDECOM Contract N61339-04-C-0043, Disruptive Technology Office, Intel and NVIDIA.

References

AILA, T., AND LAINE, S. 2004. Alias-free shadow maps. In *Proceedings of Eurographics Symposium on Rendering 2004*, Eurographics Association, 161–166.

ANNEN, T., DONG, Z., MERTENS, T., BEKAERT, P., SEIDEL, H.-P., AND KAUTZ, J. 2008. Real-time, all-frequency shadows in dynamic scenes. *ACM Trans. Graph. 27*, 3, 1–8.

ASSARSSON, U., AND AKENINE-MÖLLER, T. 2003. A geometry-based soft shadow algorithm using graphics hardware. *ACM Transactions on Graphics 22*, 3, 511–520.

BAVOIL, L., CALLAHAN, S. P., AND SILVA, C. T. 2008. Robust soft shadow mapping with backprojection and depth peeling. *Journal of Graphics Tools 13(1)*.

CHAN, E., AND DURAND, F. 2004. An efficient hybrid shadow rendering algorithm. In *Proceedings of the Eurographics Symposium on Rendering*, Eurographics Association, 185–195.

CROW, F. C. 1977. Shadow algorithms for computer graphics. *ACM Computer Graphics 11*, 3, 242–248.

FOLEY, T., AND SUGERMAN, J. 2005. KD-tree acceleration structures for a GPU raytracer. In *Proc. ACM SIGGRAPH/EG Conf. on Graphics Hardware*, 15–22.

GOVINDARAJU, N., LLOYD, B., YOON, S., SUD, A., AND MANOCHA, D. 2003. Interactive shadow generation in complex environments. *Proc. of ACM SIGGRAPH/ACM Trans. on Graphics 22*, 3, 501–510.

GÜNTHER, J., POPOV, S., SEIDEL, H.-P., AND SLUSALLEK, P. 2007. Realtime Ray Tracing on GPU with BVH-based Packet Traversal. In *Proc. IEEE/EG Symposium on Interactive Ray Tracing*, 113–118.

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.

HASSELGREN, J., AKENINE-MÖLLER, T., AND OHLSSON, L. 2005. Conservative rasterization on the gpu. *GPU Gems 2*, 677–690.

JOHNSON, G. S., LEE, J., BURNS, C. A., AND MARK, W. R. 2005. The irregular z-buffer: Hardware acceleration for irregular data structures. *ACM Trans. Graph. 24*, 4, 1462–1482.

JOHNSON, G. S., HUNT, W. A., HUX, A., MARK, W. R., BURNS, C. A., AND JUNKINS, S. 2009. Soft irregular shadow mapping: fast, high-quality, and robust soft shadows. In *Proc. I3D '09*, 57–66.

LAINE, S. 2006. *Efficient Physically-Based Shadow Algorithms*. PhD thesis, Helsinki University of Technology.

LAUTERBACH, C., YOON, S.-E., TANG, M., AND MANOCHA, D. 2008. ReduceM: Interactive and memory efficient ray tracing of large models. *Computer Graphics Forum 27*, 4, 1313–1321.

LEFOHN, A. E., SENGUPTA, S., AND OWENS, J. D. 2007. Resolution-matched shadow maps. *ACM Trans. Graph. 26*, 4, 20.

LLOYD, B. 2007. *Logarithmic Perspective Shadow Maps*. PhD thesis, University of North Carolina at Chapel Hill.

MCCOOL, M. 2000. Shadow volume reconstruction from depth maps. *ACM Trans. on Graphics 19*, 1, 1–26.

MCGUIRE, M. 2004. Observations on silhouette sizes. *jgt 9*, 1, 1–12.

MO, Q., POPESCU, V., AND WYMAN, C. 2007. The soft shadow occlusion camera. *Proc. Pacific Graphics 2007*, 189–198.

SCHWARZ, M., AND STAMMINGER, M. 2007. Bitmask soft shadows. *Comput. Graph. Forum 26*, 3, 515–524.

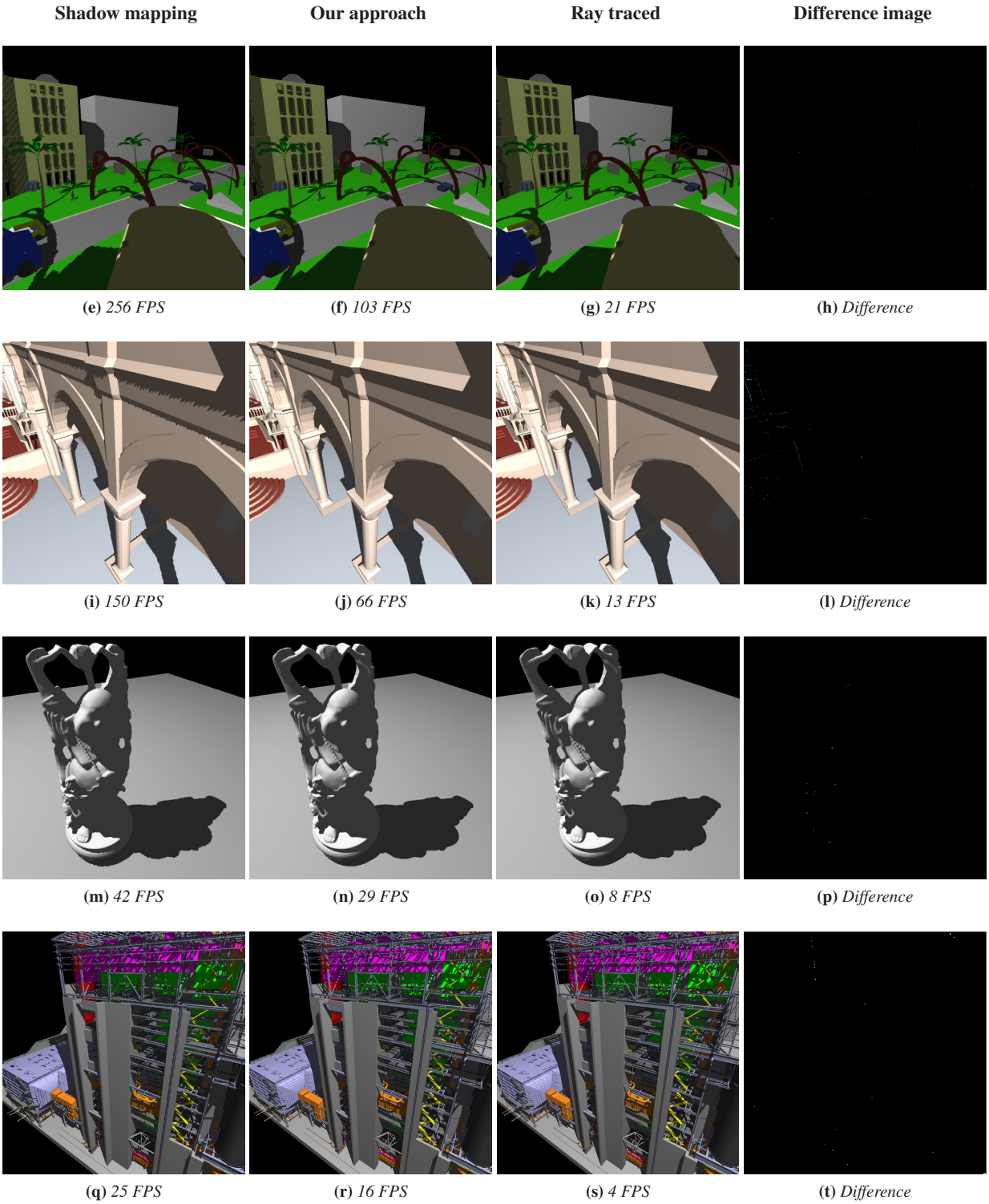
SEN, P., CAMMARANO, M., AND HANRAHAN, P. 2003. Shadow silhouette maps. *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH 2003) 22*, 3 (July), 521–526.

SINTORN, E., EISEMANN, E., AND ASSARSSON, U. 2008. Sample-based visibility for soft shadows using alias-free shadow maps. *Computer Graphics Forum (Proc. EGSR '07) 27*, 4, 1285–1292.

STAMMINGER, M., AND DRETTAKIS, G. 2002. Perspective shadow maps. In *Proc. SIGGRAPH '02*, 557–562.

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

WIMMER, M., SCHERZER, D., AND PURGATHOFER, W. 2004. Light space perspective shadow maps. In *Proc. of the Eurographics Symposium on Rendering*, 143–152.



(e) *256 FPS* **(f)** *103 FPS* **(g)** *21 FPS* **(h)** *Difference*

(i) *150 FPS* **(j)** *66 FPS* **(k)** *13 FPS* **(l)** *Difference*

(m) *42 FPS* **(n)** *29 FPS* **(o)** *8 FPS* **(p)** *Difference*

(q) *25 FPS* **(r)** *16 FPS* **(s)** *4 FPS* **(t)** *Difference*

Figure 11: Hard shadows: *We compare the image fidelity of our algorithm to a pure ray-traced reference solution. From left to right: shadow mapping at* 1024^2*, selective ray tracing, ray traced reference, difference selective to reference.*

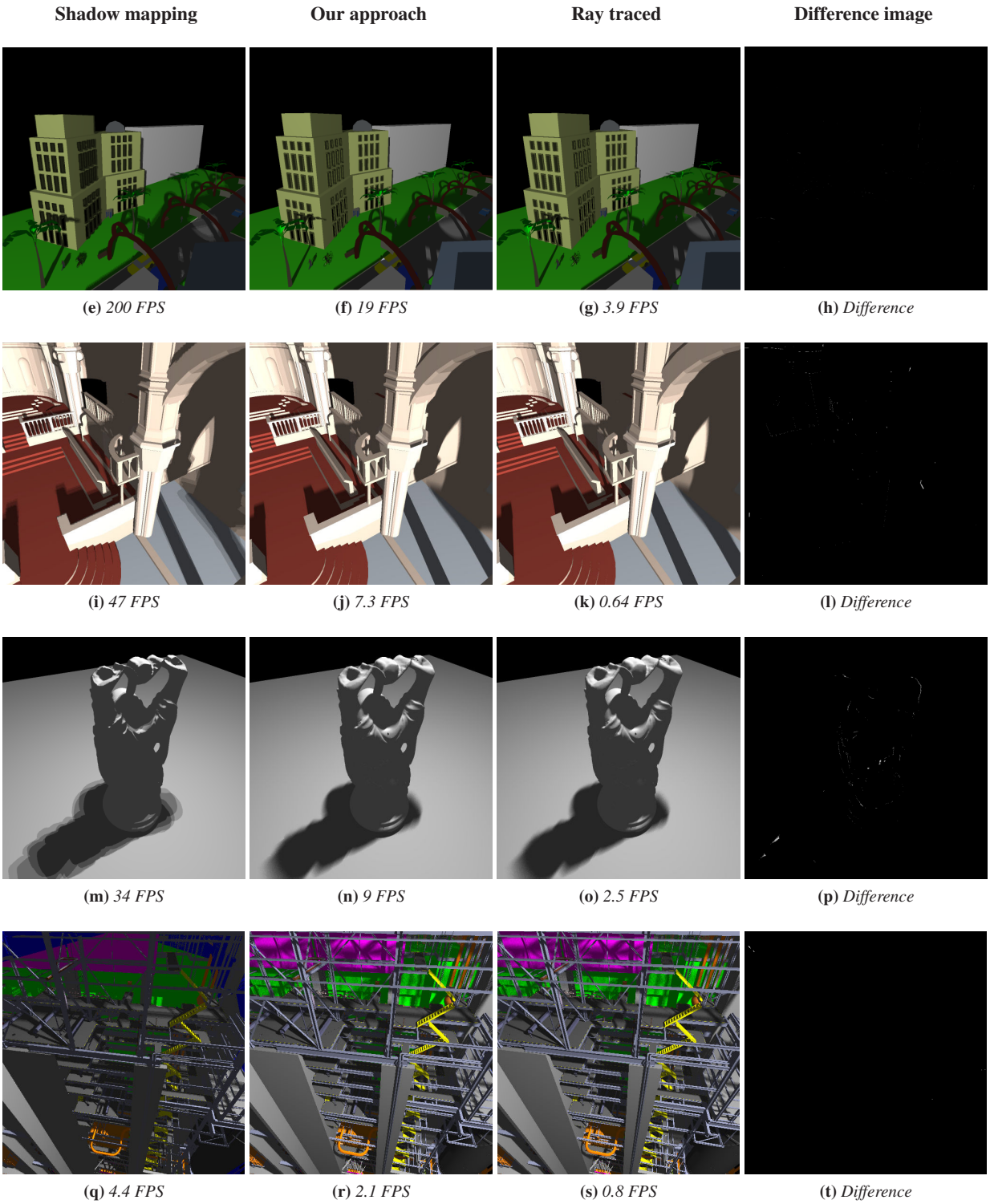


Figure 12: Soft shadows: *We compare the image fidelity of our algorithm to a pure ray-traced reference solution . From left to right: shadow mapping with 4 samples at* 1024^2*, selective ray tracing, ray traced reference (both at 16 samples/pixel), difference selective to reference.*