Efficient multi-view ray tracing using edge detection and shader reuse

Andersson, Magnus; Johnsson, Björn M; Munkberg, Jacob; Clarberg, Petrik; Hasselgren, Jon; Akenine-Möller, Tomas

Published in:
The Visual Computer

DOI:
10.1007/s00371-011-0560-4

2011

Efficient Multi-View Ray Tracing using Edge Detection and Shader Reuse

Magnus Andersson · Björn Johnsson · Jacob Munkberg · Petrik Clarberg · Jon Hasselgren · Tomas Akenine-Möller

Abstract Stereoscopic rendering and 3D stereo displays are quickly becoming mainstream. The natural extension is autostereoscopic multi-view displays, which by the use of parallax barriers or lenticular lenses, can accommodate many simultaneous viewers without the need for active or passive glasses. As these displays, for the foreseeable future, will support only a rather limited number of views, there is a need for high-quality interperspective antialiasing. We present a specialized algorithm for efficient multi-view image generation from a camera line using ray tracing, which builds on previous methods for multi-dimensional adaptive sampling and reconstruction of light fields. We introduce multi-view silhouette edges to detect sharp geometrical discontinuities in the radiance function. These are used to significantly improve the quality of the reconstruction. In addition, we exploit shader coherence by computing analytical visibility between shading points and the camera line, and by sharing shading computations over the camera line.

Keywords multi-view, ray tracing, adaptive sampling, edge detection

1 Introduction

Already around the beginning of the 20th century, inventions such as parallax barriers, lenticular lenses, and integral photography saw the light of day [19]. These form the basic foundations for stereoscopic and multi-view displays available today about a hundred years later. However, it is only during the past couple of years that this technology has seen more widespread use. For example, many new feature films are being produced with 3D in mind, and if this development continues, 3D cinema viewing may be more common than old-fashioned "normal" (2D) viewing. Recently, stereo rendering has also been re-introduced to real-time graphics with products targeting game players, and in South Korea, broadcasts of public television for stereo started in January 2010. In addition, automultiscopic displays, with many more than two views (e.g., 8 or 64) have become publicly available, and there is ongoing standardization work to augment MPEG to handle multiple views (see the previous work section and proposed algorithm by Ramachandra et al. [25]). All this implies that stereo and multi-view display technology is going mainstream, and may soon be ubiquitous.

Hence, it is expected that rendering for such displays will be increasingly important in the near future. For ray tracing-based algorithms, techniques such as multi-dimensional adaptive sampling (MDAS) [12] can easily render images for multi-view displays by sampling in the image plane, (x, y), and along the camera axis, v. While MDAS can be used for rendering any multi-dimensional space, we focus on a specialized algorithm for high-quality rendering using ray tracing with complex shaders for multi-view images.

We present two core contributions. First, we detect multi-view silhouette edges, which encode a subset of the geometric edges in the scene. We use these multi-view silhouette edges during reconstruction of the final image to improve quality. Second, since shading is expensive, we exploit the special case of multi-view image rendering by reusing the shading computations between views when possible. This is done efficiently using analytical visibility. Apart from these techniques, our algorithm uses adaptive sampling as described by Hachisuka et al. [12]. We render images with substantially higher

M. Andersson
Lund University/Intel Corporation,
E-mail: magnusa@cs.lth.se

image quality in equal amounts of time as state-of-the-art adaptive sampling techniques.

2 Previous Work

The most relevant work is reviewed here. Techniques and algorithms for multi-view rasterization are not discussed as they have little overlap with our work. We note that sampling the camera line can be seen as a temporal movement of the camera, and therefore, we also include work done in temporal sampling and ray tracing.

A four-dimensional light field [21,11] is described by the *plenoptic* function. Isaksen et al. [18] introduce dynamically reparameterized light fields, which is a technique that enables real-time focus and depth-of-field adjustments from a full four-dimensional light field. Any depth in the scene can appear in focus, by using a *wide-aperture* reconstruction filter. Light field generation can be done using both texture mapping-based algorithms or using ray tracing. In the latter case, they simply iterate over the cameras. Our work focuses on a reduced light field with three dimensions, where two dimensions form the screen plane, and the third dimension is a line where the camera can be positioned. We also use adaptive sampling, back tracing, and multi-view silhouette edges, in contrast to previous algorithms.

A suitable kernel for recovering a continuous light field signal from (sparse) samples is proposed by Stewart et al. [27]. They discuss how the reconstruction filter should be modified to handle non-Lambertian BRDFs. In addition, low-pass filtering is done to avoid ghosting, and then they use the wide-aperture approach [18] to isolate features around the focal plane. A high-pass filter is used to extract these features and those details are finally added back to the low-pass filtered version.

Determining the minimal sampling rate needed for alias-free light field rendering is discussed by Chai et al. [5]. They formalize previous results concerning light field sampling to a mathematical framework. Given the minimum and maximum depth of the scene, a reconstruction filter with an optimal constant depth can be designed. Recently, Egan et al. [9] presented a similar analysis for motion blur. Zhang and Cheng [29,30] present non-rectangular sampling patterns and extend previous work to non-Lambertian surfaces.

Halle [13] introduces the notion of *interperspective aliasing* and presents early work on band-limiting signals for holographic stereograms. Both image-space filters and 3D filters with depth information are discussed. By using a camera with a "wide" aperture when capturing the light field, it will be perspectively filtered. His PhD thesis presents a thorough overview of rendering for multi-view displays [14], mostly focused on rasterization-like algorithms. Contemporary displays using parallax barriers or lenticular lens arrays have a limited number of views, and are prone to interperspective aliasing. Zwicker et al. [31] propose a pre-filtering step to avoid high-frequency information that the display cannot handle. A general framework for automultiscopic display anti-aliasing is presented. They combine the reconstruction filters by Stewart et al. and display pre-filtering. Furthermore, by remapping/compressing the depth range, more details can be preserved.

Kartch [20] discusses efficient rendering and compression techniques for holographic stereograms. In order to achieve this goal, scene polygons are mapped into a four-dimensional space. He presents an extensive overview of current autostereoscopic and holographic techniques and their respective advantages and drawbacks. He also presents an overview of compact lumigraph representations, comparing DCT and wavelet-schemes for compression. Rendering is also discussed in detail. If the cameras are placed along a coordinate axis, v, and the image plane is parametrized with coordinates (x, y), the triangle will sweep out a volume in (x, y, v) space. He proposes a coherent rendering algorithm based on (clipped) simplex primitives in four dimensions.

View interpolation techniques [6,22] can be used to approximate a large number of views by reprojecting pixels from one or a few images with depth information. The main problems are missing data due to occlusion, and incorrect handling of view-dependent effects and transparency. This can be improved by using deferred shading and multi-layer representations [23, 26], and interpolation-based techniques have gained interest for multi-view video compression [32]. In contrast, our goal is high-quality multi-view image generation by directly sampling and reconstructing the three-dimensional radiance function.

A lot of research has already been done in the field of ray tracing stereoscopic images. In most cases, however, the existing stereoscopic ray tracing algorithms reproject samples from the left eye to the right eye, and detect whether there were any occlusions [1,3,10]. These algorithms only use two discrete camera locations (one per eye location) instead of a full camera line. Havran et al. [16] divide shaders into a view-dependent and a view-independent part, and reuse the latter of these between frames in animation. This technique resembles our shader reuse over the camera line. Finally, we refer to the article on spatio-temporal anti-aliasing by Sung et al. [28], which includes a thorough survey of the field.

To reduce the number of samples needed, we use a variant of multi-dimensional adaptive sampling (MDAS)

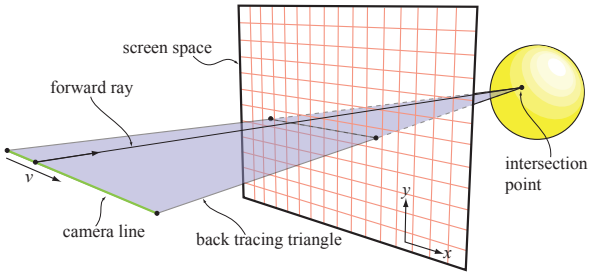

Fig. 1 The three dimensions that we sample in our system consists of the image space, (x, y), and a continuous camera line, v. In our algorithm, we trace forward rays from a specific point on the camera line, i.e., for a specific v-value. From the intersection point on the hit surface, we create a back tracing triangle, which originates from the intersection point and connects with the entire camera line. This triangle is used for both analytical visibility and for detecting multi-view silhouette edges (Section 4).

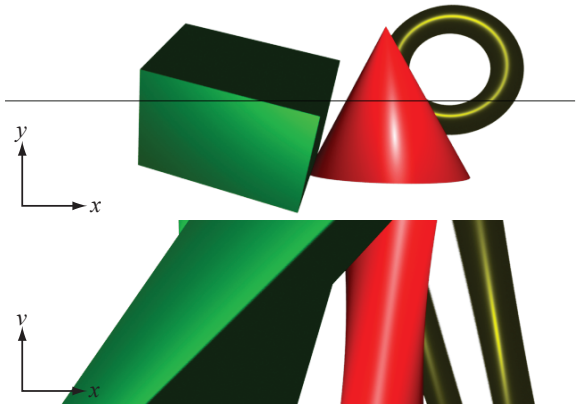

Fig. 2 Top: a single image as seen from one point, v, on the camera line, viewed in the xy-plane. Bottom: epipolar image in the xv-plane for the black line (top) with a particular y-coordinate. We used extreme disparity of the camera to illustrate a wide range of events in the epipolar plane.

[12]. This is a two-pass algorithm where in the first pass, samples are generated in the multi-dimensional space, focusing on regions where the local contrast between samples is high. In the second pass, the image is reconstructed by integrating the multi-dimensional function along all dimensions but the image dimension. They also compute structure tensors from gradients in order to better filter edges. MDAS is used to render motion blur, soft shadows, and depth of field.

Bala et al. [4] use edges projected to the image plane to prevent sample exchange over edge discontinuities during reconstruction. We extend this technique to three-dimensional patches in xyv-space. Similar to our reconstruction, Bala et al. also uses the projected edges to divide pixels into regions to produce anti-aliasing.

3 Overview

In this section, we will briefly present an overview of our rendering system. The space we want to sample has three dimensions, denoted (x, y, v), where v is the *view* parameter along the camera axis, and (x, y) are the image space parameters. This is illustrated in Figure 1. In our subsequent sections, where our algorithms are described in detail, we often refer to the epipolar plane, which is visualized in Figure 2.

Our goal is to sample and reconstruct the light field, $L(x, y, v)$, in order to display it in high quality on an automultiscopic display. These displays conceptually have a grid of pixels, where each pixel can simultaneously display n distinct radiance values projected towards different positions, v_i, along the camera axis. Each such *view* is visible within a small range, and there is usually some overlap between views based on the optical properties of the display. As the number of views is limited (e.g., 8–64), the display is severely bandwidth-limited along v. To avoid interperspective aliasing, L is integrated against a view-dependent filter, g_i, in the reconstruction step to compute n distinct images, L_i:

$$L_i(x, y) = \int L(x, y, v)\, g_i(v) dv, \tag{1}$$

This effectively replaces the strobing effect seen when the human viewer moves, by blurring of all objects in front of and behind the focus plane. To determine final pixel values, L_i is integrated against a spatial anti-aliasing filter as usual.

Random sampling of L is expensive. For each sample, a ray has to be traced from v through (x, y), in order to find the intersection with the scene, and the shading has to be evaluated. We call these *forward rays*. There has, however, been little work done on efficient multi-view ray tracing. One approach that can be applied is multi-dimensional adaptive sampling (MDAS) [12], which focuses samples to regions of rapid change, and thus drastically reduces the total number of samples required. This is a general method, which can be seen as performing edge detection in the multi-dimensional space. Discontinuities in L have two main causes; abrupt changes in the shading, and geometrical edges in the scene. As each sample is shaded individually, the method does not exploit the shading coherence that is inherent in multi-view settings; the shading of a point often varies very slowly along the camera axis, and for the view-independent part, not at all.

We propose a specialized algorithm that extends upon MDAS for multi-view ray tracing, which addresses this inefficiency. Our adaptation still supports higher dimensional sampling (such as area lights) to be handled as usual by MDAS, as long as they do not affect vertex positions (such as motion blur). In combination with sampling of L using adaptive forward rays, as described in MDAS, we also analytically detect geometric silhouette edges by tracing a triangle *backwards* from an

intersection point towards the camera axis. This will be described in Section 4.1. In Section 4.2, we introduce a concept called *multi-view silhouette edges* to encode this data. For shading discontinuities, we rely on adaptive sampling.

The analytical detection of geometric edges provides the exact extent of the visibility between a shading point and the camera axis. Hence, we can insert any number of additional samples along the segments known to be visible, without any need for further ray tracing. The shading of these extra *back tracing samples* has a relatively low cost, as all view-independent computations can be re-used, e.g., the sampling of incident radiance. See Section 4.3 for details. In the reconstruction step, described in Section 5, a continuous function is created based on the stored samples and edge information. Whereas MDAS reconstructs edges by computing per-sample gradients, we know the exact locations of geometric silhouette edges. This significantly improves the quality of the reconstruction, and hence of the final integrated result.

4 Sample Generation

When a forward ray, as seen in Figure 1, hits a surface, shading is computed and stored as a sample in the (x, y, v)-set in a *sample kD-tree*. We note that for the parts of the camera line, i.e., for the values of v, from which this intersection point can be seen, all view-independent parts of the shading can be reused. Our goal is to exploit this to add additional, low-cost samples to the sample set. However, as shown in Figure 3, this may cause sharp variations in the sample density of different areas. Unless care is taken during reconstruction, this may cause geometric edges to translate slightly. These sharp variations mainly occur around the geometric silhouette edges. We solve this using an edge-aware reconstruction filter (Section 5).

4.1 Analytical Back Tracing

The back tracing part of our algorithm commences after a forward ray has hit an object at an intersection point. The goal is to compute the parts of the camera line that are visible from the intersection point. This visibility is defined by the *silhouette edges*, and we find these analytically using an approach where the last part resembles shadow volume rendering [7] in the plane.

First, a *back tracing* triangle is created from the intersection point and the camera line, as seen in Figure 1. The back tracing triangle is intersection tested against the scene's *back facing* triangles. When an intersecting triangle is found, both of the triangle edges that intersects the back tracing *plane* are inserted into

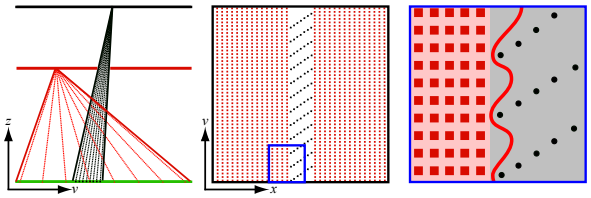

Fig. 3 Non-uniform sample densities due to back tracing. The leftmost figure shows a scanline in the xz plane of a scene with a red horizontal line with a gap in focus, and a black horizontal line in the background. For each intersection point, a number of back tracing samples are distributed along the visible part of the camera line. In the middle figure, the epipolar plane is shown after a moderate number of forward rays has been traced and corresponding back tracing samples has been distributed. Note that the sample density is both significantly lower and less evenly distributed in the gray region. Finally, a close-up shows the incorrectly reconstructed edge as a red line, compared to the correct regions (light red and gray background). Using our approach, the correct result is obtained.

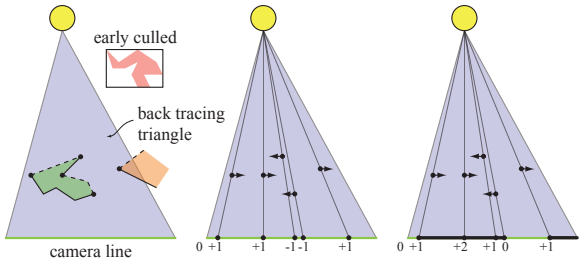

Fig. 4 Illustration of how analytical visibility is computed from an intersection point (on the yellow circle) back to the camera line. The back facing lines, as seen from the intersection point, are solid. Left: the triangles of objects that cannot be early culled against the back tracing triangle are processed. A hash table is used to quickly find the silhouette points (black dots) on the remaining lines. Middle: the winding of the silhouette points are used to give them either a weight $+1$ or -1. Right: When the weights are summed from left to right, the occluded parts are the segments with summed weight greater than zero (black regions). The inner silhouettes are discarded from further processing.

a hash table. If the edge is already in the hash table, this means that it is shared by two back facing triangles, and is removed from the table. For an edge to be a silhouette, as seen from the intersection point, it must be shared by one back facing and one forward facing triangle. This means that the edges that remain in the hash table after traversal are the potential silhouettes. This is illustrated to the left in Figure 4. Using this approach, closed 2-manifold surfaces can be handled, but by adding another hash table for front facing triangles, open 2-manifold surfaces can also be handled.

When the potential silhouette edges have been found, their intersection points with the plane of the back tracing triangle are projected to the camera line and are then processed from left to right. A counter is initialized to the number of silhouette edges whose triangles originates outside and to the left of the backtracing tri-

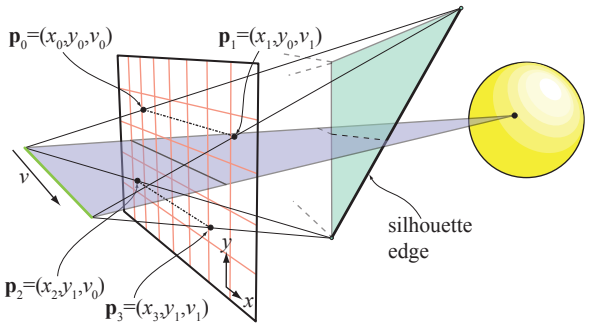

Fig. 5 Multi-view silhouette edge generation. When the end points of a silhouette edge are projected towards the end points of the camera line, four points, $\mathbf{p_0}$, $\mathbf{p_1}$, $\mathbf{p_2}$, $\mathbf{p_3}$, are obtained. These define the multi-view silhouette edge, which is a bilinear patch. The multi-view silhouette edges are used in our reconstruction algorithm to better preserve geometric edges. Note that the focus plane is identical to the screen-space plane in this figure.

angle. Each point has either weight $+1$ or -1 according to their *winding* order, (indicated with small arrows in Figure 4). Similar to shadow volume rendering [7], we compute the sum of the weights along the camera line, and when the sum is greater than zero, the camera line is occluded—otherwise, it is visible from the intersection point. In the end, we only keep the *outer* silhouette points, i.e., the end points of the black segments in Figure 4. Note that since the intersection point is visible from at least one point on the camera line, situations where the entire camera line is occluded cannot occur. The silhouette tests are executed on the edges of the primitives with no regards to the surface shader, so transparent or alpha-textured geometry will be handled as solid occluders, and are thus not supported. We leave this for future work.

4.2 Multi-View Silhouette Edges

In this section, we describe how we generate silhouettes in a multi-view setting. These *multi-view silhouettes* encode a subset of the geometric edges in the scene, and are used in Section 5 to substantially improve the final image quality in the reconstruction.

In the back tracing step described above, we have identified a number of *outer* silhouette points. Each such point is generated from a triangle edge, which is a silhouette for at least some v on the camera line and y in the image plane. Seen in the three-dimensional (x, y, v)-space, these lines will trace out a surface which is a bilinear patch. We call these patches *multi-view silhouette edges*, but we will use the term patch interchangeably. The geometrical situation is illustrated in Figure 5, where the multi-view silhouette edge is defined by the four points, $\mathbf{p_i}$, $i \in \{0, 1, 2, 3\}$, obtained by projecting the triangle edge end points towards the

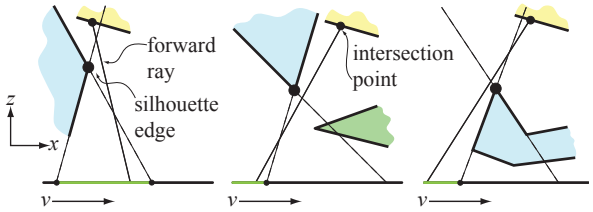

Fig. 6 A multi-view silhouette is clipped in the v-dimension. We use the planes incident to the silhouette edge to limit the extents of the multi-view silhouette edge along the v-axis. The multi-view silhouette edges are clipped to the green intervals on the v-axis. In all three cases, we can only keep the part of the camera line where the edge is a silhouette, and where the intersection point is visible.

end points of the camera line. Note that reducing the dimensionality by fixating y or v to a specific value, the patch will be reduced to a line segment in the xv- and xy-plane, respectively.

Each multi-view silhouette edge locally partitions the sampling domain into disjoint regions, which represent geometrical discontinuities in the scene, and this is what we intend to exploit in the reconstruction step. We note that it is only silhouettes that are visible from the camera line that are of interest. We know that at least one point on each patch is visible (the outer silhouette point found in the back tracing step), but other parts may very well be occluded by other patches. In general, only a limited region of each patch will be visible from the camera line. The ideal solution would be to clip each patch against all others, in order to find the correct partitioning. We note that our multi-view silhouette edges resemble the EEE-events used when computing discontinuity meshes for soft shadows [8], however, our surfaces are more restricted and thus simpler, since there are only two different y-values and two different v-values. Despite this, full patch clipping is still much too expensive for our purposes.

To reduce the complexity of clipping, we lower the dimensionality of the problem by discretizing the y-dimension into *y-buckets*. Hence, in each y-bucket, the patch is reduced to a line segment, which lies in the epipolar plane, so this gives us a much more tractable 2D clipping problem. A silhouette edge in the (x, v) plane for some y can easily be retrieved by interpolating between the points $\mathbf{p_0}, \mathbf{p_2}$ and $\mathbf{p_1}, \mathbf{p_3}$ respectively. The depth of the silhouette can also be acquired by the inverse of the interpolation of $\frac{1}{z}$ for these end points. The retrieved silhouette edge is overly conservative since the edge is not necessarily an outer silhouette as seen from the entire camera line. By clipping the camera line with the two planes of the triangles that share the silhouette edge we can determine the range in v over which the edge is a true outer silhouette. The common case, and the two special cases are shown in Figure 6. Note that

we only keep the portion of the multi-view silhouette in which there is a point known to be visible from the camera line (i.e., the part where the forward ray originated from). The silhouette is then inserted into an *edge kD-tree*, held by the y-bucket, and is clipped against the existing edges in that kD-tree.

When an existing edge and the new edge intersects, the edge with the largest depth is clipped against the other. The winding associated with an edge corresponds to which side of it is occluded. Hence, the part of the clipped edge that is in the non-occluded region of the other edge is kept. However, the hit point where the back facing triangle intersected the incoming silhouette edge is known to be a part of the new edge, so even if it is in the occluded region of an existing edge, we keep it and rely on future back tracing to find the missing edge information. A typical edge clipping scenario is shown in Figure 7.

Instead of inserting the multi-view silhouette edge in to all buckets the patch overlaps, we insert only in the current y-bucket. We then rely on the back tracing from other intersection points to repeatedly find the same multi-view silhouette edge and insert it into the other buckets. This may seem counterintuitive, but otherwise occluded parts of the multi-view silhouette edge could be inserted, resulting in incorrect discontinuities in the light field. All per-patch setup (i.e., its corner points $\mathbf{p}_i$, depth, etc) is computed only once and stored in a hash table, keyed on the vertex id:s that make up the edge.

Finally, it should be noted that to find multi-view silhouettes, we only shoot forward rays through the *centers* of y-buckets in the current implementation. This avoids the problem of getting patches that are only visible along the forward ray's y-coordinate, but not from the y-bucket's center. In our renderings, we used four buckets per pixel, which did not give any apparent visual artifacts.

4.3 Shading Reuse

We define a *back tracing sample* as the radiance originating from the intersection point hit by a forward ray, as seen from a point, v, on the camera line. Once the analytical back tracing has identified a set of *visible* segments on the camera line, a set of these back tracing samples are generated very inexpensively in those segments, and these samples are inserted into the sample kD-tree holding all the samples. We exploit shader coherence by reusing shading computations for back tracing samples generated from the same intersection point. In addition, we can further reduce the shader evaluations using careful interpolation, since some parts of the shading equation vary slowly. Our technique is somewhat similar to approximate multi-view rasterization [15].

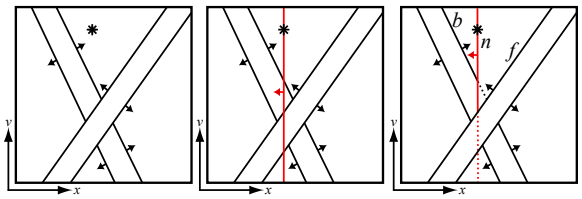

Fig. 7 Insertion and clipping in a y-bucket. Left: the existing multi-view silhouette edges and the intersection (the star) between a new silhouette edge and the backtracing triangle. Middle: the 2D edge, which is in perfect focus (vertical), representing the new silhouette edge in this y-bucket, is added. Right: the clipped result, where the new edge, n, is clipped against the edges in the foreground, f, and clips one of the edges in the background, b. The winding (arrows) of the clipping edge determines which part to keep of the clipped edge.

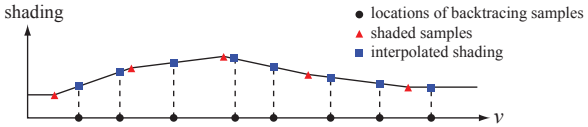

Fig. 8 Illustration of our interpolation technique for shading. The locations of $n_{\mathrm{bt}} = 8$ back tracing samples (circles) and $n_{\mathrm{vd}} = 5$ view-dependent shading samples (triangles) are shown. The shading is evaluated (squares) at the back tracing samples using interpolation of the shading samples. Note that interpolation is replaced by clamping outside the shaded samples' range.

Each forward ray will hit an intersection point that needs to be shaded. In the simplest implementation, we divide the shader into a view-dependent (V_D) and a view-independent (V_I) part [16,20]. For *every* back tracing sample, the V_D part of the BRDF is evaluated. This includes, for example, specular highlights, reflections, and refractions. The V_I part of the BRDF is only evaluated *once* for *all* the back tracing samples. Examples of this are irradiance gathering, visibility computations, and diffuse lighting. This alone results in a significant speedup. We also note that colors derived from anisotropic or mip mapped texture filtering belong to V_D. When looking towards a point on a surface, the texture lookups will get different footprints depending on the viewpoint.

Now, assume we have n_{bt} back tracing samples, where we want to evaluate *approximate* shading. Furthermore, assume that the V_D shading is computed at n_{vd} locations on the camera line, and texturing is done at n_{t} locations. We have observed that high-quality approximations of shading and texturing can be obtained using interpolation even if $n_{\mathrm{bt}} > n_{\mathrm{vd}} > n_{\mathrm{t}}$. This is illustrated in Figure 8. The type of quality that can be achieved using our technique is shown in Figure 9. Note that our approximation resembles that of RenderMan-like motion blur rendering, where shading is evaluated less often compared to visibility [2]. For texturing, we

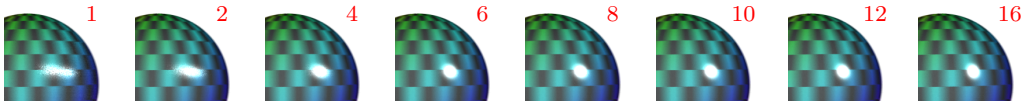

Fig. 9 A specular sphere rendered with $n_{\mathrm{bt}} = 16$ back tracing samples (visibility) and a varying number of view-dependent shading samples (n_{vd}). Since the material used is highly view-dependent, the shading needs to be evaluated rather often for faithful reconstruction.

have found that setting $n_{\mathrm{t}} = 1$ is sufficient. This introduces a small error, but this has been insignificant for all our images. In extreme cases, more texture samples may be required.

The number of view-dependent (n_{vd}) and texture (n_{t}) samples per forward ray is a variable parameter, and can be adjusted depending on the shader. The back tracing samples are jittered on the visible part of the camera line, and for the sub-segment where the forward ray originated in, the corresponding back tracing sample is removed. Instead the forward ray is used there, since it was generated by MDAS exactly where the error measure was largest. It is important to note that whereas MDAS always generates the next sample in the region with the highest error, the back tracing samples will, in general, be of lower importance since they are located in regions with lower local error estimates. Hence, there is no point in adding an excessive number of back tracing samples, as this will make sample kD-tree lookups substantially slower. However, for expensive shaders or when there are many views, the benefit of reusing shader computations and evaluating parts of the shaders at lower frequencies is still significant. Currently, we let the user set the n_{bt}, n_{vd}, and n_{t} constants. See Section 6.

5 Multi-View Image Reconstruction

In this section, we describe our reconstruction algorithm, which is, in many respects, similar to that of multi-dimensional adaptive sampling (MDAS) [12]. However, there is a fundamental difference, which is the use of multi-view silhouettes (Section 4.2) for more accurate reconstruction around geometric edges. Recall that all the samples are three-dimensional, i.e., have coordinates (x, y, v). In addition, each sample has a color. From this sample set, n different images will be reconstructed, where n depends on the target display. For example, for a stereo display, $n = 2$. See Figure 10 for an illustration.

The first steps of our reconstruction follow those of MDAS closely. First, the sample kD-tree is subdivided until it contains only one sample per node. Then, as each node is processed, the k nearest neighbor samples in Euclidean space are found and stored in the node. The samples found are representatives of the local radiance that may ultimately affect the color in the kD-tree

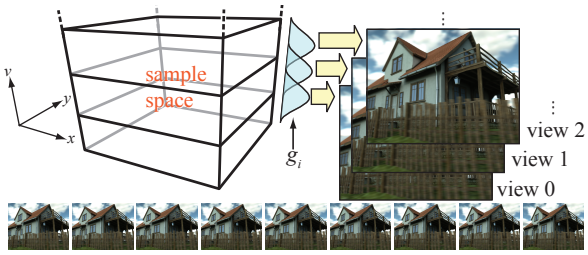

Fig. 10 After the sample generation step, we have a three-dimensional sample set, (x, y, v), where v is the view dimension, i.e., the parameter along the camera line. From this sample set, n images are reconstructed. In this case, $n = 9$, where all nine images are shown at the bottom. An arbitrary filter kernel, g_i, along the v-axis can be used.

node region. Next, gradients are computed and used to calculate the structure tensors for each node.

After this initial reconstruction setup, the final color of a pixel is found by integrating along a line in the v-dimension, with (x, y)-coordinates fixed to the pixel center. In a multi-view setting, only a certain range of v needs to be processed (see Figure 10) for a particular view. The intersection of the line with the kD-tree nodes generate disjoint line segments. The clipped line segment's extent in the v-dimension determines the color weight of the node. The node color is computed by placing an integration point in the middle of the line segment $(x_p, y_p, v_{\mathrm{mid}}^{\mathrm{node}})$ and using an anisotropic nearest neighbor filter. This filter uses the node's structure tensor to find the sample, out of the k nearest neighbor representatives, with the shortest Mahalanobis distance to the integration point [12].

MDAS reconstructs high-quality edges when the sample kD-tree has a faithful representation of the true edge in itself, and MDAS sampling creates precisely that [12]. We modify MDAS reconstruction to exploit the multi-view silhouette edges. This gives an edge-aware reconstruction that is better suited the sample distribution generated by our algorithm. Our sample distributions will be less (locally) uniform (see Figure 3) due to insertion of many inexpensive back tracing samples (Section 4).

Therefore, we have developed an edge-aware reconstruction technique that uses the multi-view silhouettes to remedy this, and improves the reconstruction quality for the type of sample set that our algorithm generates.

Instead of directly using the Mahalanobis nearest neighbor, we now describe how to use the silhouette edges obtained in Section 4.2. Conceptually, we search

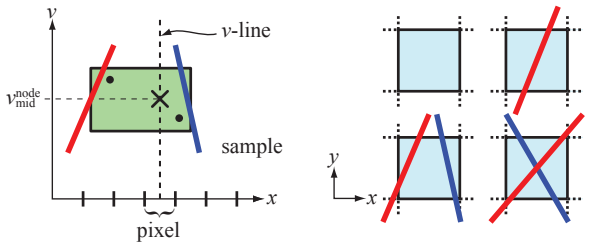

Fig. 11 Left: a sample kD-tree node (green box) viewed in the epipolar plane, xv. The plus is the integration point on the integration line chosen for this kD-tree node, and black circles are samples. At $v = v_{\text{mid}}^{\text{node}}$, our algorithm searches in x for the two closest multi-view silhouettes (red and blue lines). Right: the closest multi-view silhouettes are evaluated for the y-value of the pixel center, and hence the edges are projected to screen space, xy. With two edges, four different cases can occur, where the pixel area is split into 1–4 regions. The color of each region is then computed using the closest samples in each region.

for the two closest edges to the integration point in the closest y-bucket edge kD-tree. Recall that the intersection of a multi-view silhouette edge, which is a bilinear patch (Figure 5), with a plane $y = y_p$ is a line. This line is used to compute the x-coordinates at $v = v_{\text{mid}}^{\text{node}}$ for all the nearest multi-view silhouettes, and the two closest edges (if any) with $x < x_p$ and $x \geq x_p$ are kept. This situation is illustrated to the left in Figure 11.

Each patch, which is found in this process, is extrapolated in all directions to partition the (x, y, v) space into two disjoint sub-spaces. Each sample is then classified into one of these sub-spaces for each patch, giving four possible combinations. Samples that are separated by geometric edges in the scene are now classified into different sets, and the node color can be calculated by only considering the samples that are in the same subspace as the integration point.

As a further improvement, when there are 1–2 edges overlapping a pixel for a sample kD-tree node we can produce high-quality anti-aliased edges in the images. Consider the cross section of the two patches at $v = v_{\text{mid}}^{\text{node}}$, which are lines in the xy plane. Clipped against the pixel extents in xy, this can result in the four different configurations as shown to the right in Figure 11, where the pixel contains 1–4 of the regions. The areas, a_i, of these clipped regions are computed, where the sum is equal to the area of a pixel. Most of the time, all but one $a_i = 0$, since the common case is that the patches do not intersect the pixel extents at $v_{\text{mid}}^{\text{node}}$. When this does occur, though, we perform separate anisotropic nearest neighbor filtering for each region with $a_i > 0$. The final color contribution of a sample kD-tree node is then $\sum_{i=1}^{4} a_i \mathbf{c_i}$, where $\mathbf{c_i}$ is the color of region i.

In some rare cases, there will not be any samples in one or more regions overlapping a pixel. In this case, the reconstructed color cannot be calculated, and our remedy to this is to trace a new ray at a random location within that region, and use the color of that sample in the reconstruction. In order to avoid rebuilding the sample kD-tree, recomputing structure tensors etc., we simply discard the new sample after it has been used. This was found to be faster.

It should also be noted that while the silhouette patches greatly help by categorizing samples into different partitions, we only use a small subset of the local patches for performance reasons. Furthermore, there may even be patch information missing due to undersampling. This, however, has not proven to be a problem in our test scenes.

As described in Section 4.2, the y-dimension was discretized to a small number of y-buckets per scanline. As a consequence, this approach is prone to aliasing in the y-direction. To counter this, we jitter the y-value of the integration point within the pixel, for each node and scanline during reconstruction, and use the y-bucket containing jittered integration point as a representative of the local geometry. This replaces aliasing with noise, given a sufficient number of y-buckets.

6 Results

We have implemented our algorithms as well as MDAS in our own ray tracer, which is similar to `pbrt` [24]. Our version of MDAS is at least as fast as Hachisuka et al's implementation. To be able to use a large number of samples, we split the image into rectangular tiles. Each tile is then sampled individually. The difference compared to Hachisuka et al. [12] is that our tiles are only split in the y-direction. Any split in the x- or v-directions would limit the extent of the epipolar plane, and thus reduce the advantage of our algorithm. Since all steps are independent for every tile, we can easily process the tiles in parallel using many threads. We have observed close to linear speedups with both our algorithm and MDAS using this approach.

Only Lambert and Phong shading were used in our test scenes. For irradiance gathering, we used 128 direct illumination rays and 16 indirect illumination rays with 8 secondary illumination rays per hit point (except for the dragon scene, where 64 were used), which makes the shading moderately expensive. All scenes were rendered with high-quality texture filtering [17], and using an overlapping box filter along the v-axis when filtering out the different images. Note that our target display is a Philips WoW 3D display with nine views, where each image has 533×400 pixels. In all our resulting images, only one out of these nine images is chosen. However, note that there is nothing in our algorithms that limits our results to this particular display.

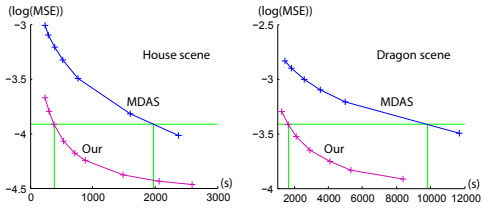

Fig. 12 Error measurements for our algorithm (purple) and MDAS (blue) of the house and dragon scenes. The rendering time is on the x-axis and the log(MSE) over all nine views is on the y-axis.

All images were rendered on a Mac Pro with two 6-core Intel Xeon at 2.93 GHz with 8 GB RAM. To make it simpler to time different parts of the algorithms, we always use single-threaded algorithms. We split the y-dimension into 10 tiles to keep the memory usage down. Note that we find $k = 15$ nearest neighbors during reconstruction and allow six samples per node during sampling, which is a reasonable value shown previously [12]. The MDAS-scaling factor for the entire view axis, v, is set to 1.5. We use four y-buckets for all our images.

For all our test scenes, we used $n_{\text{bt}} = 12$ back tracing samples. For the house and fairy (top and middle in Figure 13) scenes, we found that using a single view-dependent shading sample ($n_{\text{vd}} = 1$) was enough to produce high quality images. All the materials used in these scenes (except for the waterspout in the house scene) have subtle or no specular components, and thus sparse shading works well. In contrast, the dragon in the bottom part of Figure 13 has a very prominent specular component. Here, we used $n_{\text{vd}} = 6$ view-dependent shading samples to better capture this effect. In Figure 12, we plot the image quality as a function of rendering time for two of our test scenes, using the logarithm of mean-square error (MSE) :

$$\log(\text{MSE}) = \log\left(\frac{1}{3pv}\sum_{v}^{views}\sum_{p}^{pixels}\Delta R_{pv}^2 + \Delta G_{pv}^2 + \Delta B_{pv}^2\right),$$

where each Δ is the difference in pixel color compared to the reference image. As can be seen, our algorithm renders images with the same quality as MDAS in about 10-20% of the time (for that particular quality).

Our algorithm and MDAS spend different amounts of time in the different parts of the algorithms. For our test scenes, MDAS typically spends most of its time in the sampling and shading step. Our algorithm is able to quickly produce more samples due to shader reuse and analytical back tracing, but on the other hand, our filter cost per sample is higher. As a consequence, more time is spent in the reconstruction step in our algorithm. Below is a table comparing the average times spent in each step for our algorithm and MDAS. The reconstruction setup involves the kD-tree subdivision, nearest neighbor (NN) lookup and tensor calculations, where the NN setup alone amounts to roughly 90% of the setup time.

	House		Fairy		Dragon	
Algorithm step	MDAS	Our	MDAS	Our	MDAS	Our
Sampling & shading	95%	66%	95%	65%	>98%	60%
Reconstruction setup	4%	22%	4%	22%	1%	8%
Filtering	1%	12%	1%	13%	<1%	32%

While the house and fairy scenes have relatively few silhouette edges, the grass in the dragon scene produces significantly more silhouettes. Consequently, our algorithm has to spend more time in the filtering step, since the cost of updating the active patches increases. There are also more pixels that use the improved filtering described in Section 5, and more regions lack samples and hence require retracing with forward rays.

We found that, besides silhouette patches, our algorithm benefits from the reused irradiance gathering. For the dragon scene, we only trace about half the number of forward rays compared to MDAS, but due to back tracing, we have roughly five times more samples than MDAS in the filtering step. As a consequence, we shoot only half the number of irradiance gathering rays.

7 Conclusions and Future Work

We have presented a novel ray tracing algorithm, where we have exploited the particular setting of multi-view image generation. This was done by computing analytical visibility back from the intersection point of a forward ray, and by quickly generating back trace samples, which reuse common calculations for shading. In addition, we detected multi-view silhouette edges, and used these during final reconstruction for improved image quality around geometrical edges. Our results indicate that we can render images with the same quality as generated by MDAS up to 13 times faster. An interesting result of our research is that it opens up new avenues for future work. We have shown that we can exploit geometrical information about edges to drastically reduce the sample rate needed to generate an image, and we believe that this can be explored also in other contexts, such as depth of field and motion blur.

Acknowledgements We acknowledge support from the Swedish Foundation for Strategic Research, and Tomas is a *Royal Swedish Academy of Sciences Research Fellow* supported by a grant from the Knut and Alice Wallenberg Foundation.

References

1. Adelson, S., Hodges, L.F.: Stereoscopic Ray-tracing. The Visual Computer, **10**(3), 127–144 (1993)
2. Apodaca, A., Gritz, L.: Advanced RenderMan: Creating CGI for Motion Pictures. MKP (2000)
3. Badt, S.J.: Two Algorithms for Taking Advantage of Temporal Coherence in Ray Tracing. The Visual Computer, **4**(3), 123–132 (1988)

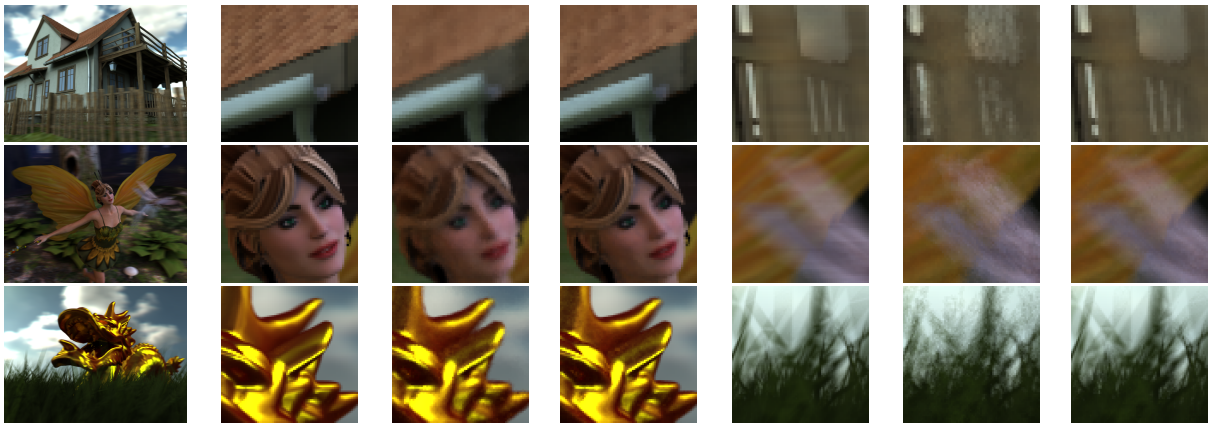

Fig. 13 Image quality comparison for equal time renderings. The noise in our images is greatly reduced compared to MDAS. The images are rendered with 16k initial random samples, after which adaptive sampling kicks in. **Top:** House scene (93.6k tris) after 676s rendering with our algorithm and 712s with MDAS. **Middle:** Fairy scene (194k tris) after 504s with our algorithm and 519s with MDAS. **Bottom:** Dragon scene (301k tris) after 4114s with our algorithm and 4973s with MDAS.

4. Bala, K., Walter, B., Greenberg, D.P.: Combining Edges and Points for Interactive High-Quality Rendering. ACM Transactions on Graphics **22**, 631–640 (2003)
5. Chai, J.X., Tong, X., Chan, S.C., Shum, H.Y.: Plenoptic Sampling. In: Proceedings of ACM SIGGRAPH, pp. 307–318 (2000)
6. Chen, S.E., Williams, L.: View Interpolation for Image Synthesis. In: Proceedings of ACM SIGGRAPH, pp. 279–288 (1993)
7. Crow, F.: Shadow Algorithms for Computer Graphics. In: Computer Graphics (Proceedings of ACM SIGGRAPH), pp. 242–248 (1977)
8. Drettakis, G., Fiume, E.: A Fast Shadow Algorithm for Area Light Sources using Backprojection. In: Proceedings of ACM SIGGRAPH, pp. 223–230 (1994)
9. Egan, K., Tseng, Y.T., Holzschuch, N., Durand, F., Ramamoorthi, R.: Frequency Analysis and Sheared Reconstruction for Rendering Motion Blur. ACM Transactions on Graphics, **28**(3), article no 93 (2009)
10. Ezell, J.D., Hodges, L.F.: Some Preliminary Results on Using Spatial Locality to Speed Up Ray Tracing of Stereoscopic Images. In: Stereoscopic Displays and Applications (Proceedings of SPIE), vol. 1256, pp. 298–306 (1990)
11. Gortler, S.J., Grzeszczuk, R., Szeliski, R., Cohen, M.F.: The Lumigraph. In: Proceedings of ACM SIGGRAPH, pp. 43–54 (1996)
12. Hachisuka, T., Jarosz, W., Weistroffer, R., K. Dale, G.H., Zwicker, M., Jensen, H.: Multidimensional Adaptive Sampling and Reconstruction for Ray Tracing. ACM Transactions on Graphics, **27**(3), 33.1–33.10 (2008)
13. Halle, M.: Holographic Stereograms as Discrete Imaging Systems. In: Practical Holography VIII (Proceedings of SPIE), vol. 2176, pp. 73–84 (1994)
14. Halle, M.W.: Multiple Viewpoint Rendering for Three-Dimensional Displays. Ph.D. thesis, MIT (1997)
15. Hasselgren, J., Akenine-Möller, T.: An Efficient Multi-View Rasterization Architecture. In: Eurographics Symposium on Rendering, pp. 61–72 (2006)
16. Havran, V., Damez, C., Myszkowski, K., Seidel, H.P.: An Efficient Spatio-Temporal Architecture for Animation Rendering. In: ACM SIGGRAPH Sketches & Applications (2003)
17. Igehy, H.: Tracing Ray Differentials. In: Proceedings of ACM SIGGRAPH, pp. 179–186 (1999)
18. Isaksen, A., McMillan, L., Gortler, S.: Dynamically Reparameterized Light Fields. In: Proceedings of ACM SIGGRAPH, pp. 297–306 (2000)
19. Javidi, B., Okano, F.: Three-Dimensional Television, Video, and Display Technologies. Springer-Verlag (2002)
20. Kartch, D.: Efficient Rendering and Compression for Full-Parallax Computer-Generated Holographic Stereograms. Ph.D. thesis, Cornell University (2000)
21. Levoy, M., Hanrahan, P.: Light Field Rendering. In: Proceedings of ACM SIGGRAPH, pp. 13–42 (1996)
22. Mark, W.R., McMillan, L., Bishop, G.: Post-Rendering 3D Warping. In: Symposium on Interactive 3D Graphics, pp. 7–16 (1997)
23. Max, N., Ohsaki, K.: Rendering Trees from Precomputed Z-Buffer Views. In: Eurographics Rendering Workshop, pp. 45–54 (1995)
24. Pharr, M., Humphreys, G.: Physically Based Rendering: From Theory to Implementation. MKP (2004)
25. Ramachandra, V., Zwicker, M., Nguyen, T.: Display Dependent Coding For 3D Video on Automultiscopic Displays. In: IEEE International Conference on Image Processing, pp. 2436–2439 (2008)
26. Shade, J., Gortler, S., He, L.w., Szeliski, R.: Layered Depth Images. In: Proceedings of ACM SIGGRAPH, pp. 231–242 (1998)
27. Stewart, J., Yu, J., Gortler, S., McMillan, L.: A New Reconstruction Filter for Undersampled Light Fields. In: Eurographics Symposium on Rendering, pp. 150–156 (2003)
28. Sung, K., Pearce, A., Wang, C.: Spatial-Temporal Antialiasing. IEEE Transactions on Visualization and Computer Graphics, **8**(2), 144–153 (2002)
29. Zhang, C., Chen, T.: Generalized Plenoptic Sampling. Tech. Rep. AMP01-06, Carnegie Mellon (2001)
30. Zhang, C., Chen, T.: Spectral Analysis for Sampling Image-Based Rendering Data. IEEE Transactions on Circuits and Systems for Video Technology, **13**(11), 1038–1050 (2003)
31. Zwicker, M., Matusik, W., Durand, F., Pfister, H.: Antialiasing for Automultiscopic 3D Displays. In: Eurographics Symposium on Rendering, pp. 73–82 (2006)
32. Zwicker, M., Yea, S., Vetro, A., Forlines, C., Matusik, W., Pfister, H.: Display Pre-filtering for Multi-view Video Compression. In: International Conference on Multimedia (ACM Multimedia), pp. 1046–1053 (2007)

Magnus Andersson received his M.Sc. in Computer Science and Engineering from Lund University 2008 and has returned to pursue his Ph.D., funded by the Intel Corporation. His main research focus has been in the field of multi-view rendering techniques, but recent interests include many-core rendering and compression.

Björn Johnsson studied Computer Science at Lund University in 2002-2007. After that he worked at Massive Entertainment for two years as a game engine programmer before returning to Lund University to recieve his M.Sc. in Computer Science in 2010. He is currently at Intel Corporation as a Ph.D. student, enrolled at Lund University. Research interests include multi-view rendering, real-time global illumination and tree rendering.

Jacob Munkberg received his M.Sc. in Engineering Physics from Chalmers University of Technology in 2002 and is currently a Ph.D. student in the graphics group at Lund University. He has worked on texture compression and culling algorithms for the real-time graphics pipeline. He is currently working at Intel Corporation. Current research interests are in graphics hardware and high-quality rendering techniques.

Petrik Clarberg received his M.Sc. in Computer Science and Engineering from Lund University in 2005. He is currently a Ph.D. student in the graphics group at Lund University, while working at Intel Corporation. Petrik has worked on photo-realistic rendering, importance sampling, shader analysis, and texture compression, and has published two papers at the ACM SIGGRAPH conference. His research interests include high-quality rendering and graphics hardware.

Jon Hasselgren received his M.Sc. in Computer Science and Engineering from Lund University in 2004. He finished his Ph.D. in computer graphics at Lund University graphics group in 2009, and is currently working at Intel Corporation. Jon has worked on rasterization, compression and culling algorithms with focus on graphics hardware, as well as GPU accelerated rendering algorithms. Currently his focus lies on graphics hardware and high-quality rendering techniques.

Tomas Akenine-Möller received his M.Sc. in Computer Science and Engineering from Lund University in 1995, and a Ph.D. in graphics from Chalmers University of Technology in 1998. He has worked on shadow generation, mobile graphics, wavelets, high-quality rendering, collision detection, and more. Tomas has several papers published at the ACM SIGGRAPH conference, and his first SIGGRAPH paper was on the pioneering topic of mobile graphics together with Jacob Ström in 2003. He co-authored the Real-Time Rendering book with Eric Haines and Naty Hoffman, and received the best paper award at Graphics Hardware 2005 with Jacob Ström for the ETC texture compression scheme, which is now part of the OpenGL ES API and Android. Current research interests are in graphics hardware, new computing architectures, novel visibility techniques, high-quality rapid rendering techniques, and many-core rendering.