Vision, Modeling, and Visualization (2015)
D. Bommes, T. Ritschel and T. Schultz (Eds.)

The Bounced Z-buffer for Indirect Visibility

O. Nalbach[1], T. Ritschel[1,2] and H.-P. Seidel[1]

[1]MPI Informatik, Germany [2]MMCI / Saarland University, Germany

Figure 1: *Screen space methods for indirect illumination [NRS14a]* (left) *not only underestimate effects of invisible geometry* (left inset)*, but also routinely ignore visibility* (right inset)*, which is added by our approach* (middle) *to produce results similar to a path-tracing reference* (right)*. Our approach renders this scene with 202 k triangles in 65 ms (768×512 px.).*

Abstract

Synthesizing images of animated scenes with indirect illumination and glossy materials at interactive frame rates commonly ignores indirect shadows. In this work we extend a class of indirect lighting algorithms that splat shading to a framebuffer – we demonstrate deep screen space and ambient occlusion volumes – to include indirect visibility. To this end we propose the bounced z-buffer: While a common z-buffered framebuffer, at each pixel, maintains the distance from the closest surface and its radiance along a direction from the camera to that pixel, our new representation contains the distance from the closest surface and its radiance after one indirect bounce into a certain other direction. Consequently, with bounced z-buffering, only the splat from the nearest emitter in one direction contributes to each pixel. Importance-sampling the bounced directions according to the product of cosine term and BRDF allows to approximate full shading by a simple sum of neighboring framebuffer pixels.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Color, shading, shadowing, and texture

1. Introduction

A wide range of methods is available to render global illumination in animated scenes at interactive rates [RDGK12]. However, several important constraints remain: Scenes of moderate geometric complexity allow for Instant Radiosity-type [Kel97, LSK*07, RGK*08] or voxel-based solutions [KD10, CNS*11] that include indirect visibility, but fail to produce detailed shading for massive tessellated or skinned geometry which might not even fit into memory at once. For such scenes, screen-space approaches [BSD08, RGS09, JSG09, Tim13, NRS14a] show their strength in producing fine spatial details of indirect light and shadow due to their output-sensitive nature, at the price of ignoring indirect visibility.

Ignoring occlusions between senders responsible for a shading effect and the corresponding receiving points leads to bias. In the case of global illumination, the senders reflect indirect light into the direction of the receivers and lack of indirect visibility becomes visible as light bleeding through surfaces. In the case of ambient occlusion, senders contribute to darkening at nearby receiving points as they occlude parts of the hemisphere over the receivers. Incorrect treatment of visibility here leads to summing up multiple occlusion events

DOI: 10.2312/vmv.20151261

from the same direction and the resulting indirect shadows consequently appear too dark compared to a reference.

Unfortunately, fast visibility queries between arbitrary points in a scene need (hierarchical) spatial data structures such as they are common in ray-tracing. For dynamic scenes, their memory cost and building time precludes their use in modern interactive applications such as computer games. Our approach to indirect visibility is fire-and-forget as it does not require any hierarchical data structures and – in the spirit of screen space – does not even need to be aware of the entire scene geometry at any point in time.

Differently from a common z-buffered framebuffer, which at each pixel maintains the distance from the closest surface and its radiance, the bounced z-buffer contains the distance from the closest surface and its radiance after one bounce from a certain other direction. Splatting onto the framebuffer uses the bounced z-buffer and consequently, only the splat from the nearest emitter in the respective direction contributes to each pixel. Importance-sampling the bounced directions according to the product of cosine term and BRDF allows to approximate full shading by a simple sum of neighboring framebuffer pixels. The idea is simple to implement and can be used for different splatting-based approaches that lack visibility, such as ambient occlusion volumes (AOV, Fig. 2) [McG10] or deep screen space (DSS, Fig. 1) [NRS14a]. We furthermore propose modifications to those target algorithms to improve their splatting quality.

After reviewing previous work in Sec. 2, we introduce the idea of a bounced z-buffer in Sec. 3. As an example, in Sec. 4, we present an adapted deep screen space using a (hierarchical) bounced z-buffer and demonstrate its effectiveness in Sec. 5.

2. Previous Work

Screen space shading was first used for ambient occlusion (AO) [Mit07, SA07, BSD08] and later extended to subsurface scattering [JSG09], diffuse bounces [RGS09, BBH13, MMNL14] and surface-to-volume shading [NRS14b]. Different ideas were proposed to overcome its limitations such as multi-resolution [NW09], layering [VPG13], sweeps [Tim13] and on-the-fly reconstruction of screen space-like surfel geometry [NRS14a].

Screen space methods typically do not account for visibility, with the exception of image-space blocker accumulation [SGNS07]: Their approach tracks a spherical harmonics approximation of visibility in screen space that is updated with spherical harmonics (SH) exponentiated values. It is limited to diffuse or moderately glossy scenes and requires to store a number of SH coefficients depending on the lighting and shadow frequencies. Our approach adapts to all ranges of materials from diffuse to mirror-like.

Instant Radosity [Kel97] accounts for visibility but does not scale well to large scenes as VPLs are also placed where they do not contribute to the framebuffer. Similarly, shadow maps for each VPL, even when accelerated [RGK*08], process geometry that does not contribute. At the same time, the level of detail resolved is limited by the shadow maps' resolution and low compared to screen space shading.

Some point-based global illumination (PBGI) methods splat approximations of scene geometry into buffers to resolve indirect visibility [Chr08, REG*09], too. Differently, we do not resolve visibility of all directions from a scattered set of sparse points, but from sparse scattered positions into sparse scattered directions. We also avoid finding a surfel or light cuts and use approximate level-of-detail in the 2D framebuffer instead. Our approach can be seen as an irregular decoupling of a hemicube [CG85] or micro-buffers [REG*09] into a map of depth-resolved samples of indirect radiance. This light field is the same as the one used by Lehtinen et al. [LALD12] where sophisticated reconstruction is applied to a low number of indirect light samples from an initial path-tracing pass. In contrast, our method produces such samples by means of splatting. We then use simple cross-bilateral blur without re-projection for reconstructions in milliseconds. Executing their improved reconstruction on our result would produce even better results, but at much higher reconstruction times (seconds to minutes).

Mattausch et al. [MBV*15] exploit coherence in ray position and direction to efficiently cull groups of rays and primitives in a hierarchy. To this end, the distance from the the nearest primitive is stored along each ray, which is the same data structure we use. Our work avoids creating any hierarchy, both for occluders and occluding primitives, and does not even require the geometry to fit into memory.

3. Method

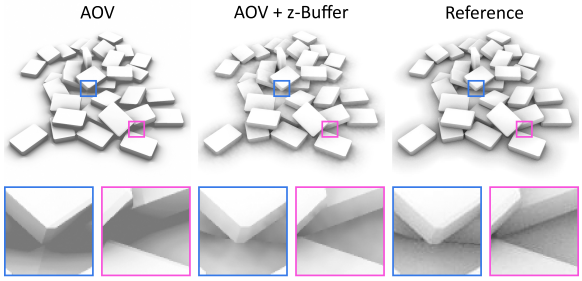

Figure 2: *Many splatting-based methods like AOV over-occlude dense geometry, which is then compensated for by tone-mapping* (left). *Augmenting a bounced z-buffer* (middle) *and using reconstruction retrieves more detailed shading in problematic areas and appears closer to the reference* (right).

We propose to add visibility to approaches that use splatting to transfer information from geometry, e.g., represented as triangles or approximate cloud of disks to pixels on which it (potentially) has an effect [DS06, SGNS07, NW09, McG10,

NRS14a]. Splatting nicely fits the parallel processing model of GPUs and is output-sensitive because we can cull primitives too far from view frustum bounds to have a noticeable effect on visible primtives.

Classical rasterization can be seen as splatting of the scenes' primitives onto rays from the camera while tracking in each pixel the one closest to the camera by a z-buffer. We transfer the idea to splatting-based GI and AO approaches. The first difference is that we are interested in not only one direction per pixel following a simple pinhole projection but in multiple ones depending on the surface BRDF. The second difference is that the splats do not simply correspond to projections of the primitives anymore but to projected regions of influence with respect to the effect being computed.

Adding indirect visibility requires three steps: modifications to the framebuffer setup and the splatting (Sec. 3.1 and Sec. 3.2) as well as a final reconstruction step (Sec. 3.3).

3.1. Direction Setup

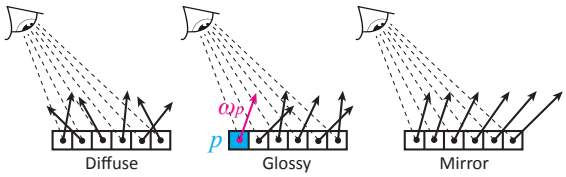

Figure 3: *Directions in a bounced z-buffer are chosen according to view, surface normal and BRDF at each pixel. For simplicity the camera is shown looking at a flat surface, which is however not a requirement.*

Input is a deferred shading buffer, rasterized from the view of the camera, providing for each pixel p the position $\mathbf{x}_p \in \mathbb{R}^3$, normal $\mathbf{n}_p \in \mathbb{R}^3$ and material information ρ_p, in the form of parameters to an analytical BRDF model, of the primitive visible at p. Based on this information, each pixel is associated with a direction ω_p (Fig. 3). For AO, uniform sampling of the hemisphere centered around the surface normal is used, for GI importance sampling according to the product of BRDF and cosine term, to also deal with glossy receivers. This direction is stored in an additional buffer.

3.2. Splatting

While splatting-based approaches typically use blending [DS06, NW09, McG10, NRS14a] to sum up contributions from multiple primitives, we only want to keep the information from the closest one. Therefore any blending is disabled and replaced by depth-buffering. When a splat, e.g., in the form of a point [NRS14a] or a polygon [McG10], is drawn for a primitive P, at each covered pixel p, first an intersection test of the ray from $\mathbf{x}_p$ into direction ω_p and P is performed. Note, that this is just a single-primitive intersection that does not require an acceleration structure. If there is no intersection, the fragment is discarded. However, if the primitive is hit at position $\mathbf{y}$, the incident indirect illumination L_p^{in} is set to the contribution of primitive P in direction $-\omega_p$. This is either radiance from a Lambertian or glossy sender or occlusion for AO shading. The squared distance between $\mathbf{x}_p$ and $\mathbf{y}$, scaled by a maximal distance $d_{\max}$ to keep it in the range $[0,1]$, is used as the fragment's depth value z_p. Due to the depth buffering, at each pixel p only the result from the primitive closest to $\mathbf{x}_p$ in direction ω_p among all those which were splatted onto p is kept. Fig. 4 gives an example.

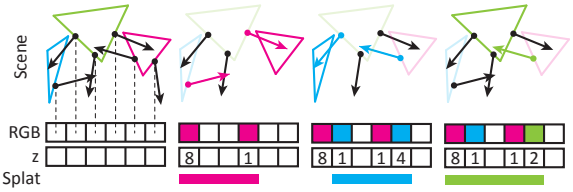

Figure 4: First: *An initially empty framebuffer. For each pixel, a ray* (arrows) *starting at the first visible surface point has been chosen.* Second: *The splat caused by the red primitive covers the four leftmost pixels in the framebuffer, but only two of the associated rays* (colored) *actually intersect the primitive.* Last: *Thanks to depth buffering, finally, each pixel p only contains the result from the closest primitive in direction ω_p. Note, that the location and size of the splats depend on the respective algorithm. Unlike standard rasterization, the original primitives are not simply projected to the screen.*

An important observation is that the fact that the ray associated with some pixel p hits a primitive P does not imply that a splat caused by P will also cover p. This is because there usually is a fall-off of the shading effect limiting the splat size. For example, for AO only primitives within a certain radius are considered as occluders.

3.3. Reconstruction

After all primitives were splatted, the buffer represents an unstructured sampling of the field of incident indirect illumination. From this, exitant indirect illumination is computed in two steps: Importance weighting and filtering.

Weighting Recall that the sample directions were importance-sampled. Therefore, we first need to divide the incident indirect illumination by the probability density $\alpha_p = \alpha(\omega_p|\mathbf{x}_p, \mathbf{n}_p)$ for picking direction ω_p at position $\mathbf{x}_p$ with normal $\mathbf{n}_p$. This can either be implemented by storing α_p when ω_p is generated as described in Sec. 3.1 into an additional buffer, or by re-computing it (deterministically) on-the-fly for a second time.

Filtering To reconstruct exitant indirect illumination L_p^{out} at pixel p into the direction ω_{out} of the camera, a weighted sum

over the incoming indirect radiance L_q^{in} of all pixels q in a neighborhood $\mathcal{N}(p)$ is used:

$$L_p^{\text{out}} = \sum_{q \in \mathcal{N}_p} w(p,q) f_{\text{r}}(\mathbf{x}_p, \omega_{\text{out}}, \omega_q) L_q^{\text{in}} \langle \mathbf{n}_p, \omega_q \rangle_+ / \sum_{q \in \mathcal{N}_p} w(p,q).$$

The weight $w(p,q)$ depends on the differences in position, normal and material for pixels p and q. In other words, in one step, the incident indirect illumination at q is used to reconstruct the incident indirect illumination at p and at the same time convolved with the BRDF f_{r} and the geometric term $\langle \mathbf{n}_p, \omega_q \rangle_+$.

Thanks to the simplicity of the approach, scenes can be rendered in high resolution, resulting in very similar values of position, normal and material. A filter with large support in the spatial, angular and material domain results in less variance but has a stronger bias, compared to a less-biased sharper filter yielding a noisier result. Our reconstruction defers the BRDF and geometry term to be computed exactly once per directional sample per pixel so that BRDF and geometry details in the framebuffer (e.g., bump maps) are preserved.

4. Z-buffered Deep Screen Space

In this section, after briefly reviewing the deep screen space pipeline [NRS14a] (Sec. 4.1), we describe how it can be adapted to a bounced z-buffer (Sec. 4.2) and suggest optimizations which are enabled or motivated by this adaptation (Sec. 4.3). For a more thorough description of the method, we refer to the original paper [NRS14a].

4.1. The Deep Screen Space Pipeline

The basic idea of deep screen space is to tessellate the scene into a view-dependent level-of-detail surfel representation ("surfelization") which is then scattered to a multi-resolution deferred shading buffer by means of splatting to compute the effect of the surfels on their surroundings.

Surfelization In the surfelization step, the triangles of the original scene are dissolved into a cloud of surfels, i.e., small disks, represented by their position, normal and radius in world space. The cloud is designed such that all surfels have roughly the same projected radius in screen space, maintaining the same total surface area. This step is implemented using the hardware tessellation provided by modern GPUs, which not only allows to output triangle meshes, but also only the set of generated vertices of the tessellated mesh, each of which becomes a surfel by associating it with a normal and radius. In the case of indirect lighting, additional information about each surfel's material is necessary to be kept.

Splatting Into a Hierarchical Framebuffer To compute shading from the surfel cloud, a splatting approach is used. A hierarchical framebuffer with $l_{\max}$ levels is constructed. As depicted in Fig. 5, pixels are partitioned into 4^l sub-images

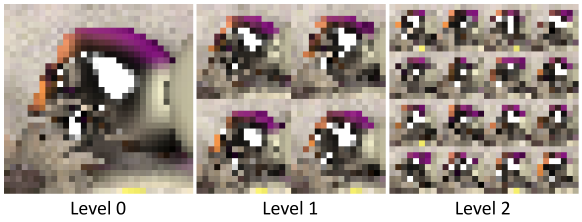

Figure 5: *A possible layout for the hierarchical framebuffer used in the splatting stage of the DSS pipeline.*

for level l by distributing neighborhoods of $2^l \times 2^l$ pixels in the original image. Each sub-image receives one of the pixels at random, where the pixel takes the same relative position that the neighborhood had with respect to the full-size image.

Opposed to naïve splatting, using one final-resolution framebuffer, in the deep screen space pipeline, for each surfel $l_{\max}$, same-sized (in screen space) splats are drawn, namely one into each level of the hierarchical framebuffer. The sub-image receiving the splat is selected at random per level. As the extent of the sub-images is halved with each level-step, the distance covered by the splats in world space is doubled accordingly. At the same time, the effect is sampled at a decreasing fraction of all the pixels that a corresponding (larger) splat in a full-resolution buffer would cover.

On each level, the effect of the surfel on a different (world space) shell around the surfel's center is computed. The shell for level l hereby has an inner radius of 0 on level 0 and of $d_{\max} \times 2^{l-1}$ units on the other levels, and an outer radius of $d_{\max} \times 2^l$ units, where $d_{\max}$ is a distance depending on the estimated influence of the surfel. This partitioning of the effect over several shells corresponding to splats on different framebuffer levels gives rise to the name *shell splatting*.

4.2. Shell Splatting with Visibility

To add visibility, for every pixel p, one unique direction ω_p is produced as described above (Sec. 3.1), but for each pixel independent z values are maintained on each level of the hierarchy. The buffer is splat with shells as explained in the original paper but now visibility is tracked for each pixel on each level as explained in Sec. 3.2. After splatting, but before reconstruction, the levels have to be combined. As only the contribution of the closest surfel in direction ω_p is relevant, the result from the finest level containing a surfel-hit is selected. Since the levels correspond to disjoint distance ranges and are ordered with respect to increasing distance, this is the most "precise" result among all levels. The final image is then reconstructed as explained in Sec. 3.3. Note, that when splatting, different from the original approach, it is not advised to reduce the size of splats from darker surfels, as they now contribute valuable information as blockers of indirect light; in other words: $d_{\max}$ is a constant and not longer a function of the surfel.

4.3. Additional Improvements

Surfelization Different limitations of the original deep screen space tessellation lead to sub-optimal surfel clouds, resulting in underestimation of indirect light. With added indirect visibility, not only light but also shadowing of indirect light is underestimated and a higher-quality surfel cloud with properly sized and placed surfels is even more desirable.

The surfelization described in [NRS14a] can only amplify geometry, not reduce it. A typical scene also contains triangles which are too small to directly tessellate them into surfels of the desired size. In fact, the target surfel size might be much larger than the initial triangle itself and several small triangles should be grouped to one surfel. Another problem are thin triangles, for which a placement of the surfels according to the barycentric coordinates generated by the tessellation hardware does not lead to even coverage of the triangle.

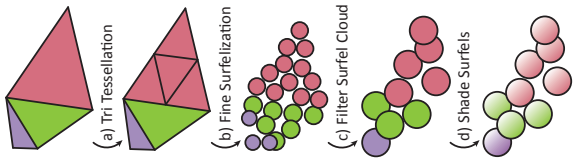

Figure 6: *We handle extremely large or small triangles by pre-tessellation* (a) *and post-filtering* (c) *passes, respectively.*

We first compute an initial surfel cloud with fine surfels, smaller than the target size, and store them using the transform feedback mechanism (Fig. 6, a+b). Instead of simply placing a surfel at the barycentric coordinates determined by the hardware tessellation pattern, we compute the world space position corresponding to the coordinates but then hash this position into an offset in a lookup texture containing pre-computed pseudo-random numbers. Using the respective random number, the surfel is placed uniformly at random on the initial triangle. This results in more even placement of surfels, also for thin triangles.

In a second pass, we compare for each surfel P its area A_P resulting from the first tessellation pass with the actually desired ("ideal") area A_P^* (cf. [NRS14a]). We discard the surfel with probability $\max(1 - A_P^*/A_P, 0)$ by either emitting it from the geometry shader or not (Fig. 6, c). The surfels which are not discarded are then assigned their ideal radius. In the case of indirect lighting, we additionally shade the surfels in this step (Fig. 6, d).

This second pass is only conceptually separate from the splatting. In fact, it can be integrated in the splatting pipeline to save a second pass over the surfels. While our two passes add computational cost during tessellation, they pay off in speed by a faster splatting stage due to the lower number of surfels and in quality of the constructed surfel cloud (Fig. 7).

More efficient use of fill rate In shell splatting, point primitives are used to cover what is actually the projection of a 3D-shell on the framebuffer. Fragments corresponding to the inner part of the shell have to be detected and discarded using a conditional because the effect of a surfel is otherwise not distributed disjointly among the different levels. Not only are these fragments "wasted" but also does the check whether a fragment belongs to the inner part of the shell slow down computations for all of the fragments.

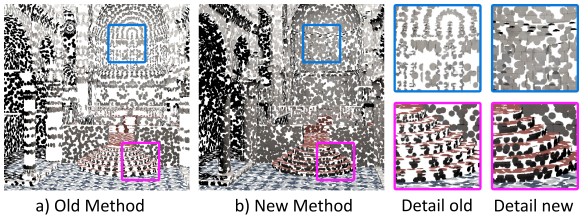

Figure 7: *Surfels produced using the new approach* (b, 109 k. surfels, 7.4 ms) *and the old one* (a, 120 k.surfels, 4.1 ms).

We modify this behavior as follows: Instead of discarding a fragment at a pixel p corresponding to a world space position $\mathbf{x}_p$, we compute which shell around the surfel which $\mathbf{x}_p$ belongs to and then perform the same computations as if the fragment was actually placed on the corresponding level. Note, that in the end only the highest-precision hit for each pixel / direction is kept, so doing this, we never override hits found on finer levels. We may override "non-hits" on finer levels by actual surfel-hits but we never replace information computed with one precision (i.e., subsampling rate) by information computed with a lower precision. In practice this approach is not only faster but also produces cleaner results.

Min / Max MIP-map for splat culling Finally, an even higher level of output-sensitivity is achieved by culling splats against a min / max MIP map of the deferred position texture before the splatting begins. During splatting, the smallest mip level l_{MIP} where all pixels covered by the splat in the original image have been reduced to the same pixel p_{MIP} is computed. The values at p_{MIP} of level l_{MIP} in the min and max MIP maps define a local axis-aligned bounding box (AABB) of all receiver points in the framebuffer. Splats not intersecting this AABB are culled.

5. Results and Discussion

In the following, we demonstrate the z-buffered DSS approach presented in the previous section applied to AO and one-bounce indirect illumination with diffuse as well as glossy receivers. Results for combining our approach with ambient occlusion volumes [McG10] are seen in Fig. 2.

Comparison with previous methods Fig. 8 provides a qualitative evaluation of our adapted method (third column) versus the original deep screen space without indirect visibility (second column), a corresponding screen space method (first

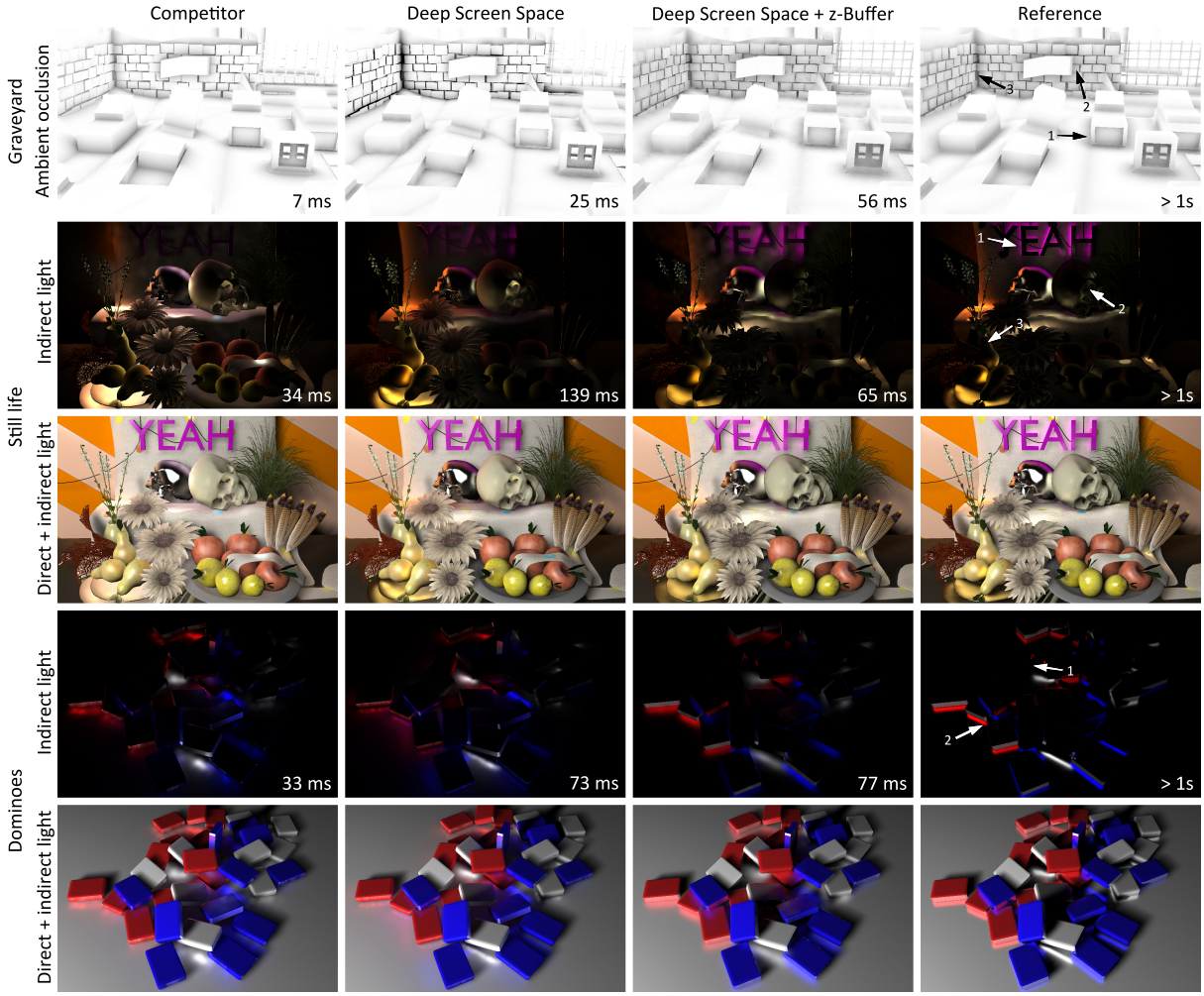

Figure 8: *Comparison of our method* (3rd col.) *to screen space competitors* (1st col.)*, the original shell splatting* (2nd col.) *and a reference* (4th col.)*. The competing AO method used is horizon-based AO (HBAO) [BSD08], for the indirect light it is screen space GI [RGS09]. See the accompanying video for animated versions of the scenes and the text for a discussion.*

Table 1: *Computation time for different stages and effects.*

Stage	Graveyard	Still Life	Dominoes
Tessellation	8.3 ms	8.8 ms	6.5 ms
Shuffling	4.0 ms	4.5 ms	4.5 ms
Splatting	38.5 ms	46.8 ms	61.2 ms
Unshuffling	1.3 ms	1.5 ms	1.5 ms
Reduction	0.3 ms	0.3 ms	0.3 ms
Blurring	3.5 ms	3.2 ms	3.0 ms
Total	55.8 ms	65.1 ms	77 ms

column) as well as a reference based on ray-tracing (last column). For our approach, timings of the individual stages are given in Tbl. 1.

For AO, we chose horizon-based AO (HBAO) [BSD08] as the screen space method, which is state of the art in quality. For GI, we used screen space GI [RGS09]. We applied the same geometry-aware blur to both, our method and the screen space reference. (The original DSS result is blurred per level as in [NRS14a].) All results were obtained for a resolution of 768×512 pixels on a Nvidia GTX 770 GPU. We always used a 4 samples per pixel for our new method, where each sample corresponds to a unique direction.

The first row (Fig. 8) shows AO for a graveyard scene (122 k tris, 375 k surfels). Albeit being the fastest of the four approaches we compare, HBAO fails to reproduce proper

darkening due to triangles which are not visible or seen under grazing angles. This leads to a quite different 3D perception of the scene (e.g., arrow 1). Also the subtle shadowing of the brick wall due to differently protruding bricks (2), which is visible in the reference, is missing. DSS reproduces darkening from invisible geometry but over-occlusion such as in the corner of the brick wall (3) is a problem. This makes scaling of the effect necessary which in turn reduces more subtle shadowing too much. Our method most resembles the reference, suffering the least from under- and over-occlusion. The most noticeable difference to the reference is noise due to the low number of sampled directions per pixel. Yet, unlike the reference, our method runs at interactive speed.

The second and third row (Fig. 8) show indirect light and the combination with direct light, respectively, for a still life scene with glossy as well as diffuse objects (202 k tris, 167 k surfels). Screen space GI misses both, indirect light, e.g., from the letters in the background (1), and indirect occluders, e.g., inside of the skull (2) which is mistakenly lit by the cloth below. Again, the screen space method is fastest in comparison. Deep screen space deals with the light missing for the screen space method but the lack of indirect visibility is even worse, as can be seen inside the skull (2) or on the pears (3) to the left. Our adapted deep screen space again is closest to the reference with the same artifacts as for AO.

In the last two rows of Fig. 8 we show a scene containing many colorful dominoes on a highly specular surface (1.4 k tris, 237 k surfels). Since many faces of the dominoes are not visible or cover only little screen area, the amount of available indirect light samples in screen space varies strongly, resulting in perceivable noise. Due to the lack of indirect visibility, the deep screen space result overestimates indirect light (e.g., 1) or mixes light from differently colored emitters in some places, leading to purple areas not found in the reference (2). The z-buffered deep screen space is again closest to the reference, even though we cannot achieve the same level of specularity due to the blurring which is necessary to reduce noise from sub-sampling.

Table 2: *Amount of data after different tessellation stages (Fig. 6) at the example of the Graveyard test scene (Fig. 8).*

Stage	Time	Data size after
Initial full scene		122 k tris
Tri tessellation	1.2 ms	19 k tris
Fine surfelization	4.1 ms	797 k surfels
Filtering surfels	3.0 ms	375 k surfels

Time complexity The total running time for our method consists of the time for tessellation on one hand and time for splatting on the other. For tessellation, we again have to consider several stages with different complexity (Tbl. 2).

During the initial tessellation step which is necessary to handle very large triangles [NRS14a], we have to process all of the scenes' triangles. However, we can cull the triangles against the view frustum (plus an additional safety margin depending on the effect quality settings). The most costly step in the surfelization is usually the second in which we produce the fine surfel cloud, from which the final surfels are selected afterwards, because it writes many small surfels. The filtering step again has to consider all of the small surfels, but in practice, it can be integrated into the splatting pass and run concurrently (cf. Sec. 4.3). Splatting is the most expensive step of the pipeline. Since all splats have the same size on screen [NRS14a], the runtime depends on the number of visible splats in the first place, i.e., the number of surfels that cannot be culled times the number of levels in the framebuffer. Again, since the surfels are constructed to have the same projected size, the number of surfels basically only depends on the depth complexity of the visible part of the scene.

Table 3: *Memory consumption of our method, for different shading effects and an output resolution of* 768×512 *px.*

Data	Memory requirement
Min / max MIP map	6 MB
Framebuffer layout	24 MB
Shuffled positions	18 MB
Shuffled bounce directions	18 MB
Layered framebuffer	6 MB (AO) / 18 MB (GI)
Total	72 MB (AO) / 84 MB (GI)

Memory requirement Tbl. 3 summarizes the memory consumption of our method. Most of the memory is necessary to store the framebuffer layout, i.e., the lookup textures used for shuffling and unshuffling. Next is the hierarchical deferred shading information itself. The only difference between AO and GI is that for AO the effect can be computed using float textures while GI needs 3D vectors.

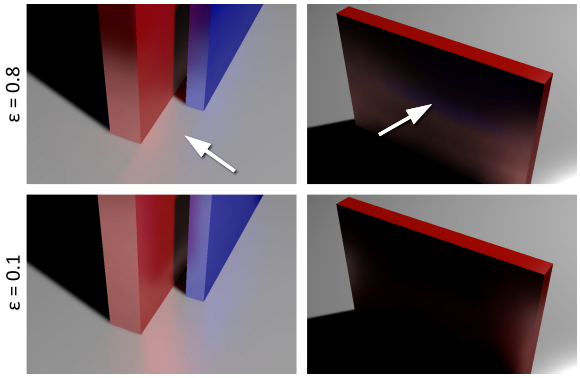

Figure 9: *Effect of* ε *on precision for far-range shading.*

Problems and limitations In shell splatting, many small surfels are approximated by fewer enlarged ones for the far

range, which implicitly assumes that the enlarged version of the picked surfel is representative for surrounding surfels. This assumption breaks when the surface normal changes rapidly in the neighborhood of the selected surfel, for example at convex or concave corners and becomes observable for large values of ε [NRS14a] (which regulates the trade-off between the amount of sub-sampling and runtime) as in Fig. 9 (top left). Similarly, we may observe artifacts resulting from enlarged surfels protruding through other geometry and becoming wrongly visible to a pixel (Fig. 9, top right).

Another (typical) problem is temporal coherency. Not only are the sampling directions randomly selected for each frame but also does the subset of the surfel cloud which is sampled at each pixel change, leading to flickering in animated scenes. We currently apply simple re-projection [SYM*12] for animated scenes as shown in the video (but not in any figure in this article). Further research could address spatio-temporal filtering of the bounced z buffer itself.

6. Conclusion

We have devised a conceptually simple and computationally efficient method to add visibility to splatting-based global illumination rendering. In future work, we would like to address more advanced visibility such as for transparent or volumetric occluders or receivers. Our filtering currently does not yet make use of known distance information for advanced filtering such as adaptive kernel sizes or re-projection. Finally, an interesting algorithmic extension could associate every pixel with an opening angle depending on the BRDF for improved pre-filtering using finite sized solid angles.

Acknowledgements We thank Morgan McGuire for providing the source code of his AOV implementation and the Sibenik cathedral mesh by Marko Dabrovic.

References

[BBH13] BARAK T., BITTNER J., HAVRAN V.: Temporally coherent adaptive sampling for imperfect shadow maps. *Comp. Graph. Forum (Proc. EGSR) 32*, 4 (2013), 87–96. 2

[BSD08] BAVOIL L., SAINZ M., DIMITROV R.: Image-space horizon-based ambient occlusion. In *SIGGRAPH 2008 talks* (2008). 1, 2, 6

[CG85] COHEN M. F., GREENBERG D. P.: The hemi-cube: A radiosity solution for complex environments. In *ACM SIGGRAPH Computer Graphics* (1985), vol. 19, ACM, pp. 31–40. 2

[Chr08] CHRISTENSEN P. H.: Point-based approximate color bleeding. Pixar Technical Memo, 2008. 2

[CNS*11] CRASSIN C., NEYRET F., SAINZ M., GREEN S., EISEMANN E.: Interactive indirect illumination using voxel cone tracing. *Comp. Graph. Forum 30*, 7 (2011), 1921–30. 1

[DS06] DACHSBACHER C., STAMMINGER M.: Splatting indirect illumination. In *Proc. I3D* (2006), pp. 93–100. 2, 3

[JSG09] JIMENEZ J., SUNDSTEDT V., GUTIERREZ D.: Screen-space perceptual rendering of human skin. *ACM Trans. Appl. Percept. 6*, 4 (2009), 23:1–23:15. 1, 2

[KD10] KAPLANYAN A., DACHSBACHER C.: Cascaded light propagation volumes for real-time indirect illumination. In *Proc. I3D* (2010), pp. 99–107. 1

[Kel97] KELLER A.: Instant radiosity. In *Proc. SIGGRAPH* (1997), pp. 49–56. 1, 2

[LALD12] LEHTINEN J., AILA T., LAINE S., DURAND F.: Reconstructing the indirect light field for global illumination. *ACM Trans. Graph. 31*, 4 (2012), 51:1–51:10. 2

[LSK*07] LAINE S., SARANSAARI H., KONTKANEN J., LEHTINEN J., AILA T.: Incremental instant radiosity for real-time indirect illumination. In *Proc. EGSR* (2007), pp. 277–86. 1

[MBV*15] MATTAUSCH O., BITTNER J., VILLANUEVA A. J., GOBBETTI E., WIMMER M., PAJAROLA R.: CHC+RT: Coherent hierarchical culling for ray tracing. *Comp. Graph. Forum (Proc. EG)*, 2 (2015). 2

[McG10] MCGUIRE M.: Ambient occlusion volumes. In *Proc. HPG* (2010). 2, 3, 5

[Mit07] MITTRING M.: Finding next gen: CryEngine 2. In *SIGGRAPH courses* (2007), pp. 97–121. 2

[MMNL14] MARA M., MCGUIRE M., NOWROUZEZAHRAI D., LUEBKE D.: *Fast global illumination approximations on deep G-buffers*. Tech. rep., NVIDIA Corp., 2014. 2

[NRS14a] NALBACH O., RITSCHEL T., SEIDEL H.-P.: Deep screen space. In *Proc. I3D* (2014). 1, 2, 3, 4, 5, 6, 7, 8

[NRS14b] NALBACH O., RITSCHEL T., SEIDEL H.-P.: Shell splatting for indirect lighting of volumes. In *Proc. VMV* (2014). 2

[NW09] NICHOLS G., WYMAN C.: Multiresolution splatting for indirect illumination. In *Proc. I3D* (2009), pp. 83–90. 2, 3

[RDGK12] RITSCHEL T., DACHSBACHER C., GROSCH T., KAUTZ J.: The state of the art in interactive global illumination. *Comput. Graph. Forum 31*, 1 (2012), 160–88. 1

[REG*09] RITSCHEL T., ENGELHARDT T., GROSCH T., SEIDEL H.-P., KAUTZ J., DACHSBACHER C.: Micro-rendering for scalable, parallel final gathering. *ACM Trans. Graph. (Proc. SIGGRAPH Asia) 28*, 5 (2009). 2

[RGK*08] RITSCHEL T., GROSCH T., KIM M. H., SEIDEL H.-P., DACHSBACHER C., KAUTZ J.: Imperfect shadow maps for efficient computation of indirect illumination. *ACM Trans. Graph. (Proc. SIGGRAPH Asia) 27*, 5 (2008). 1, 2

[RGS09] RITSCHEL T., GROSCH T., SEIDEL H.-P.: Approximating dynamic global illumination in image space. In *Proc. i3D* (2009), pp. 75–82. 1, 2, 6

[SA07] SHANMUGAM P., ARIKAN O.: Hardware accelerated ambient occlusion techniques on GPUs. In *Proc. I3D* (2007), pp. 73–80. 2

[SGNS07] SLOAN P.-P., GOVINDARAJU N. K., NOWROUZEZAHRAI D., SNYDER J.: Image-based proxy accumulation for real-time soft global illumination. In *Proc. Pacific Graph.* (2007), pp. 97–105. 2

[SYM*12] SCHERZER D., YANG L., MATTAUSCH O., NEHAB D., SANDER P. V., WIMMER M., EISEMANN E.: Temporal coherence methods in real-time rendering. *Comp. Graph. Forum 31*, 8 (2012), 2378–408. 8

[Tim13] TIMONEN V.: Line-sweep ambient obscurance. In *Comp. Graph. Forum* (2013), vol. 32, pp. 97–105. 1, 2

[VPG13] VARDIS K., PAPAIOANNOU G., GAITATZES A.: Multi-view ambient occlusion with importance sampling. In *Proc. I3D* (2013), pp. 111–18. 2