Vision, Modeling, and Visualization (2013)
Michael Bronstein, Jean Favre, and Kai Hormann (Eds.)

Progressive Visibility Caching for Fast Indirect Illumination

Justus Ulbrich, Jan Novák, Hauke Rehfeld and Carsten Dachsbacher

Karlsruhe Institute of Technology

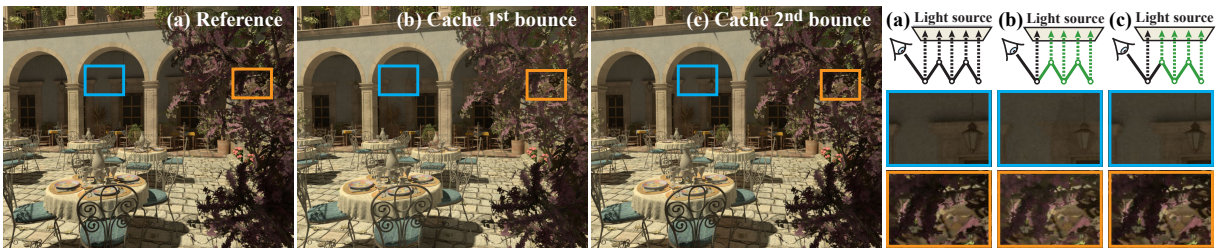

Figure 1: *Our visibility caching significantly speeds up tracing of secondary rays, here from 0.15 MRays/s (reference) to 0.70 MRays/s (with the visibility cache). The images were rendered using path tracing where some of the rays (highlighted by green in the illustrations; solid and dashed lines represent path segments and shadow rays, resp.) were traced using our visibility cache with 2000 shadow maps achieving 4.7× higher throughput of secondary rays per second.*

Abstract

Rendering realistic images requires exploring the vast space of all possible paths that light can take between emitters and receivers. Thanks to the advances in rendering we can tackle this problem using different algorithms; but in general, we will likely be evaluating many expensive visibility queries. In this paper, we leverage the observation that certain kinds of visibility calculations do not need to be resolved exactly and results can be shared efficiently among similar queries. We present a visibility caching algorithm that significantly accelerates computation of diffuse and glossy inter-reflections. By estimating the visibility correlation between surface points, the cache automatically adapts to the scene geometry, placing more cache records in areas with rapidly changing visibility. We demonstrate that our approach is most suitable for progressive algorithms delivering approximate but fast previews as well as high quality converged results.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Raytracing

1. Introduction

Visibility computation, one of the principal components of realistic image synthesis, is generally a costly operation that often represents the bottleneck of rendering algorithms. It is known that precise calculation of visibility is usually required only for certain types of rays, e.g. primary or secondary specular rays [YCK*09]. Since these rays are rather coherent, tracing them is fast and can be further accelerated, e.g. using packets or hardware rasterization. In contrast, secondary rays are much more incoherent, requiring costly random memory accesses. Fortunately, many applications tolerate a certain level of imprecision when tracing these rays and thus we can sacrifice a little amount of accuracy in favor of higher performance [CNS*11, ND12].

In this paper, we address the problem of expensive visibility computation by caching and subsequently reusing it. We build upon the work of Clarberg and Akenine-Möller [CAM08] who analyze correlation in visibility for different points in the scene. They demonstrate that since spatially and orientationally nearby surfaces tend to "see" similar parts of the scene, we can capture the visibility information only sparsely and reuse it for nearby queries. However, their analysis and the algorithm are tailored predominantly for computing direct illumination from environment maps. We show that visibility correlation can be exploited also for indirect illumination. For this, we use distance (shadow) maps and derive a new measure to assess the correlation that is not limited to binary visibility only [CAM08].

DOI: 10.2312/PE.VMV.VMV13.203-210

We primarily focus on progressive rendering algorithms that are becoming increasingly popular in lighting design and cinematic industry. These synthesize final results incrementally over time, providing fast previews as well as high quality results when given more time. Our technique fits into such scenario and accelerates visibility computation by constructing an initial *visibility cache*, which is then further refined during the progressive rendering by improving the placement of the cache records. The iterative nature of the rendering algorithm enables leveraging additional information, e.g. the utility of each cache record, which can be combined with the correlation measure providing an efficient refinement strategy. Our main contributions in this work are:

- New measure for assessing visibility correlation between distance-based shadow maps.
- Easily adjustable refinement heuristic trading the utility and local correlation of cache records.
- Progressive visibility caching algorithm with user-defined maximum memory footprint.

2. Previous Work

Caching Techniques. Caching specific terms of the rendering equations is a common optimization to many rendering algorithms. Ward et al. [WRC88] demonstrated that irradiance can be efficiently cached and reused for nearby surface points. This has been further extended to radiance caching [KGPB05], [SNRS12], and also to participating media [JDZJ08]. Walter et al. [WDP99] propose to cache and reproject results from the previous frame to quickly synthesize the current frame. Yao et al. [YWC*10] use a small number of well chosen shadow maps to cache visibility and trace photons only using the shadow maps. Since estimating the quantity directly from the cache will inevitably introduce bias, many approaches use it to "only" guide the sampling of incident light. This can be done by building a significance cache in world space [BRDC12], or by creating a sparse set of screen-space distributions for importance sampling the illumination from a large set of point lights [GKPS12].

The most relevant to ours is the work of Clarberg and Akenine-Möller [CAM08], who identify regions of points with highly correlated hemispherical visibility. They leverage this information to adaptively place a number of binary visibility maps that are used as a control variate for sampling direct illumination from environment maps. To reduce the noise in areas with partial occlusion, Ghosh and Heidrich [GH06] propose correlated visibility sampling. Popov et al. [PGSD13] quantize the visibility function into clusters to share visibility samples inside these clusters.

Shadow Maps. Shadow Maps (SM) [Wil78] store depth values to front-most surfaces as seen from a single point. The coherent nature of the SM allows for an efficient generation using hardware rasterization. Nevertheless, generation of shadow maps can still be the bottleneck of rendering algorithms. Ritschel et al. [RGK*08,REH*11] accelerate the creation of imperfect shadow maps using a coarse, adaptively placed set of point samples of the scene geometry. They also compress large sets of SMs into a data structure that allows for arbitrary visibility queries [RGKS08]. SMs can also be used to retrieve properties of the front-most surfaces: Dachsbacher and Stamminger [DS05] enhance SMs with color and normal information to efficiently generate VPLs. For a thorough overview of shadow mapping we refer the reader to the survey by Scherzer et al. [SWP11].

Progressive Algorithms. Most high-quality rendering algorithms are still prohibitively expensive for interactive applications. Making them progressive enables fast feedback [DKL10], interaction [LSK*07], and even convergent results [HOJ08]. Since we target these applications we refine the cache progressively maximizing its overall entropy.

3. Visibility Caching

In order to make our caching algorithm efficient, we need to store the visibility information in a form that allows inexpensive updates and fast queries. One option would be to store the mutual binary visibility of arbitrary pairs of points; however, such representation requires elaborate and possibly expensive look-ups. Thus, rather than storing one-to-one binary visibility, we opt to cache one-to-many *distance* information: for a given surface point we use a shadow map to store the distance to all visible surfaces. In contrast to a binary representation, the distance also allows for a fast retrieval of nearest surface points. We leverage this in the image-space path tracing described in Section 4.2.

Our visibility cache consists of a number of records (i.e. shadow maps) that are sparsely distributed over the surfaces of the scene. Visibility between any two points is resolved by finding a nearby cache record and then comparing the distance between the points to the corresponding value from the shadow map. The key component of our visibility cache is the refinement scheme, which places cache records only on *relevant* surfaces.

Surface points with similar orientation that are close to each other have similar hemispherical visibility [CAM08]. Figure 2 illustrates that the amount of correlation is dependent on the local curvature and the surrounding geometry. To maximize the amount of information captured by the cache, we need to place the cache records adaptively, i.e. more densely around areas with low correlation. We proceed in the following steps: first, we sample the surfaces sparsely and create an initial set of cache records, and link them to their spatially and orientationally nearby neighbors. Then, we assess the gradient of the *hemispherical visibility* and gather statistics from the rendering algorithm to estimate the *utility* of each record. Finally, we improve the cache by progressively replacing the most redundant and rarely used records. We detail each step in the following sections.

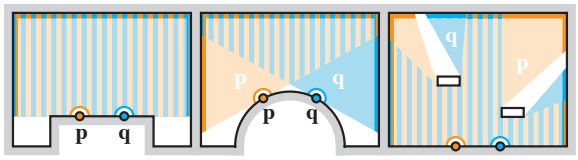

Figure 2: *Hemispherical visibility: on the left the visibility of point* $\mathbf{p}$ *strongly correlates with that of point* $\mathbf{q}$ *(striped area), in the middle and right examples the correlation is reduced due to geometric properties and nearby occluders.*

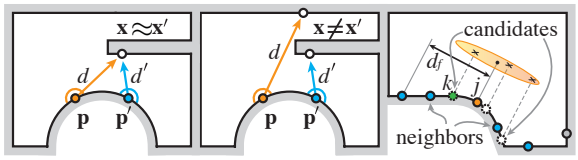

Figure 3: *To estimate the correlation in visibility between* $\mathbf{p}$ *and its neighbor* $\mathbf{p}'$ *we sample directions and evaluate if the respective distances project to the same (left) or different (middle) surface points. The right image shows the placement of a new cache record.*

3.1. Initial Distribution

We start by tracing paths from the camera creating a cache record at each bounce of the path. This ensures that the initial records are placed at locations where they are likely to be used. We improve the uniformity of the initial distribution by rejecting records that are close to the existing ones. Then we capture the hemispherical visibility of each record by rendering the scene into paraboloid shadow maps.

To analyze how the visibility of a cache record correlates to its local neighborhood, we connect each record to a number of neighbors that score the highest according to a weighting function w, which trades the spatial and angular distance. As the weighting function we use the one from [CAM08]; we provide the definition in Appendix A for completeness.

3.2. Importance of Records

Our main contribution is how we assess the importance of each cache record w.r.t: 1) capturing the global visibility, and 2) being useful for the rendering algorithm. To this end, we keep track of two quantities, *correlation* and *utility*, and then combine them into a single intuitive importance function, which defines whether the cache is important or can be removed. The following paragraphs detail how we estimate each component of the importance function.

Visibility Correlation. In order to estimate the visibility correlation between a record $\mathbf{p}$ and its neighbor $\mathbf{p}'$ (see Figure 3 left and center) we sample several directions within the upper hemisphere of $\mathbf{p}$. For each direction d we find out whether the surface point $\mathbf{x}$ seen along this direction is visible to both cache records. We first retrieve point $\mathbf{x}$ by fetching the distance from the paraboloid shadow map of $\mathbf{p}$. Then we compute direction $d' = \mathbf{x} - \mathbf{p}'$, and analogously obtain the closest point $\mathbf{x}'$ visible from $\mathbf{p}'$ along d'. By comparing the distance between $\mathbf{x}$ and $\mathbf{x}'$, we effectively assess whether both cache records see the same surface. To estimate the overall correlation of the two records, we count the relative number of directions for which $\|\mathbf{x} - \mathbf{x}'\|$ is smaller than a certain threshold. For each cache record i we store the average correlation ρ_i to all its neighbors and use it to identify regions with excessive or insufficient number of records.

Utility of Records. Additionally, we also estimate the relative utility $\mu_i = u_i / u_{max}$ of each cache record i, where u_i is the number of times the rendering algorithm actually used the cache record i, and u_{max} is the maximum of these values. Records with low utility should be removed and placed elsewhere to improve the local visibility representation. This enables adapting the cache to the specifics of the rendering algorithm, e.g. the number of indirect bounces in a path tracer, as well as to dynamic camera movements. As such, exact definition of u_i depends on the rendering algorithm; we provide two examples in Section 4.

Importance Function. Finally, we define the overall importance function γ, which is used during the progressive refinement to identify records with low correlation ρ_i and high utility μ_i as:

$$\gamma_i = \mathrm{lerp}(1 - \rho_i, \mu_i, \alpha). \tag{1}$$

The global parameter α balances the influence of ρ and μ on the progressive refinement and can be adjusted by the user. Higher values increase the impact of the correlation, lower values emphasize the utility of records. We discuss the range of meaningful values of α in Section 5.

3.3. Progressive Refinement

The key component of our visibility caching is a progressive refinement, during which we add new records to regions with low correlation. Once the cache reaches its capacity we continue refining by first removing record i with the *lowest* importance γ_i before placing a new record k. We simply delete i and mark its neighbors for updating. Then we search for a region with poorly sampled visibility by finding record j with the *highest* importance γ_j.

Our intention is to place the new record k on a surface in the vicinity of j, but not too close to any of the existing records as this would not reveal new information. For this, we generate a number of samples on a disk, which is parallel to the surface but slightly offset above j (see Figure 3, right). Then we project the samples onto surfaces around j using ray casting, creating a number of candidates. Candidates that are too far from the disk, e.g. due to discontinuous surfaces

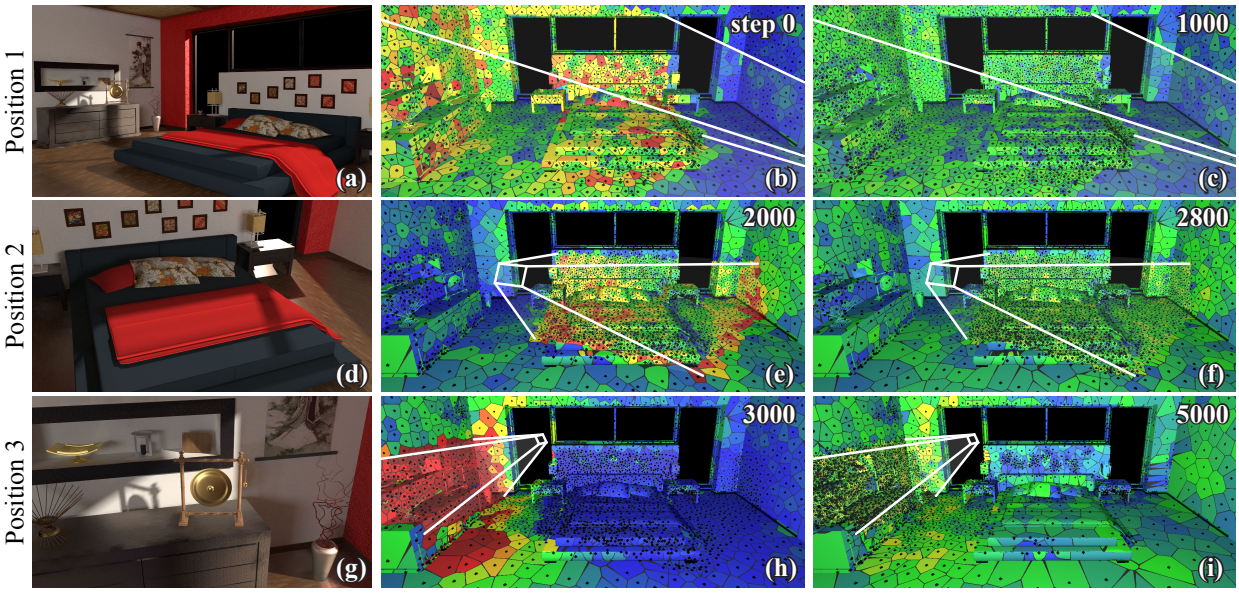

Figure 4: *Visualization of the progressive refinement. The left column shows renders from three different camera positions. The middle and right columns show Voronoi diagrams constructed from the cache records (represented by black dots) with individual cells colored w.r.t the importance function γ; blue is low, red is high. The camera stays in the initial position (a,b) for 1000 refinement steps (c), then gradually moves above the bed in 1000 steps (d,e) and stays there for 800 steps (f), then rapidly turns towards the cupboard in 200 steps (g,h), and stays still for another 2000 refinement steps (i). In the initial position (b) or after each movement (e, h) the cache oversamples some and undersamples other parts of the scene. The progressive refinement helps to equalize the importance function over the set of all records (right column).*

around j, are discarded to prevent placing the new record far away from j. From the remaining we select candidate k, which is farthest to existing records (w.r.t Equation 2) to avoid redundancy. Finally, we render the shadow map for k and update the neighbor links and the correlation values for k, its neighbors, and all previously marked records.

4. Results and Applications

The estimation of correlation between cache entries as well as the rendering of shadow maps is computed on the GPU. All timings were measured on an Intel Core i7-2600 with 12 GB of RAM and an NVIDIA GTX 560 GPU. We use between 2k and 4k cache records sampling the visibility using paraboloid shadow maps with 128×128 resolution.

Figure 4 visualizes the importance function in a sequence with moving camera (please see the accompanying video). The camera stops in three different positions (a, d, g) to emphasize the progressive refinement: images in the middle column show the importance function in the moment when the camera stops, the right column illustrates the adaptive refinement after a number of steps (the cumulative step count is shown in the top right corner). Notice how the initial distribution (b) with 4000 records is adaptively refined (c) by moving records with lower importance to areas with poorly captured visibility, and then further adjusted w.r.t the changing camera position (d-i).

Operation	Location	Time
Find records i and j	CPU	0.042 ms
Place record k	CPU	0.044 ms
Update kd-tree	CPU	0.711 ms
Mark old neighbors of i	GPU	0.089 ms
Find new neighbors	GPU	0.075 ms
Render paraboloid	GPU	19.843 ms
Compute correlation	GPU	0.035 ms
Copy back to CPU	GPU/CPU	0.758 ms

Table 1: *Performance breakdown of a single refinement step in the Bedroom scene (790k tris) with 4k cache records.*

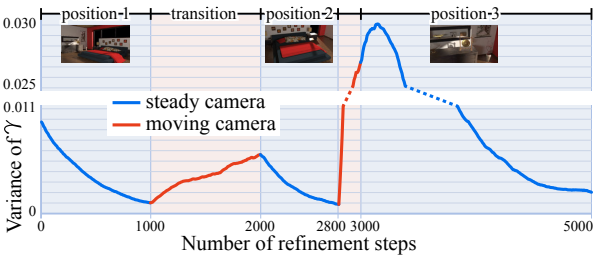

Figure 5: *Variance of the importance function γ for the sequence shown in Figure 4. The blue and red curves correspond to steady and moving camera, respectively.*

Figure 6: *GPU-based VPL rendering of indirect illumination in the San Miguel scene (24M triangles): our shadow map-based caching of visibility (left and right) renders 36× faster than the traditional approach (middle), where shadow maps are created for each VPL. The traditional VPL rendering shows artifacts (e.g. the bright splotch in the arcade) even after 10 minutes.*

The progressive refinement concentrates records inside the view frustum and in areas with difficult visibility while locally preserving the blue noise-like distribution. Table 1 shows the cost of individual operations of a single refinement step. Re-rendering the shadow map (19.8 ms) for the moved cache record clearly dominates the overall cost (21.9 ms).

In Figure 5, we plot the variance of the importance function over time. Notice how the variance decreases and the importance function becomes more and more equalized after the camera stops in a new position. When converged, all records of the cache are roughly equally important.

4.1. Instant Radiosity

Our visibility caching can be applied to existing many-light algorithms that synthesize indirect illumination progressively: in each frame the algorithm traces a number of light paths, deposits virtual point lights (VPLs), and uses them to illuminate the scene. We demonstrate the benefits of our caching on a GPU implementation of instant radiosity (IR), where we use the cache to resolve the visibility between surface points and VPLs. As a reference, we create one shadow map per VPL to resolve visibility. Note that both implementations use the exact same set of VPLs.

When using our visibility cache, we need to generate only a single shadow map per frame to progressively refine the cache distribution, and the generation is thus well amortized over several VPLs and successive frames. To resolve the visibility, we first assign three nearest cache records to each VPL. The occlusion of each VPL is then computed as a *weighted average* of visibility queries between the shading point and the three assigned records. Assigning more records did not improve the result; less records tend to produce visible artifacts. The normalized weights are obtained from the weighting function w (see Appendix A), which is evaluated for the VPL and the cache record. If the record does not have valid data for the visibility query (i.e. the direction is not captured by the hemispherical SM), we set the weight to zero. The utility of each cache record is computed by accumulating its weights over all queries within the current frame.

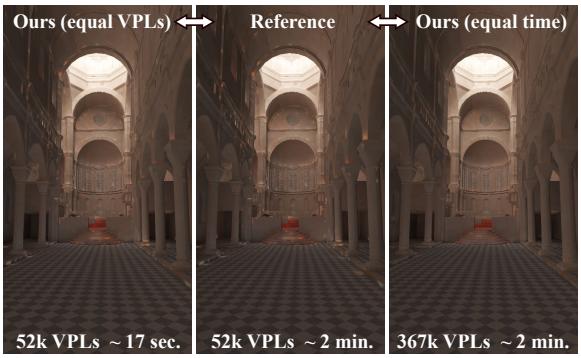

Figure 7: *GPU-based VPL rendering of the Sibenik cathedral (241k triangles): we achieve 7× higher performance.*

Figure 6 compares the reference and our visibility caching using equal number of VPLs and equal rendering time. While both approaches are approximate (even the reference suffers from artifacts due to limited resolution of shadow maps), the visibility caching performs significantly faster allowing for 36× more VPLs with the same rendering time.

Figure 7 shows a similar comparison for the Sibenik cathedral, where we achieve 7× higher performance. The main bottleneck of the reference is the expensive creation of shadow maps: we need to render all shadow maps for about 500 VPLs in each frame, which takes 93% of the frame time. With the visibility cache, which in this case contains 3000 (view-independent) records that are precomputed in 8.5 sec., only one shadow map needs to be rendered for each refinement step. In both cases, we accelerate the creation of shadow maps and lighting by VPLs using a GPU.

4.2. Image-Space Path Tracing

Our visibility caching can also be efficiently applied to path tracing. We first augment the cache with surface normals, material references, and texture coordinates; i.e. for each record we store a reflective shadow map (RSM) [DS05], obtaining an approximate representation of the entire scene.

We start tracing the paths with an accurate scene representation (e.g. BVH with triangles); however, once we discover the first (or second) bounce along the path we switch to the visibility cache. We search for three records in the vicinity of the path vertex and probabilistically select one w.r.t how they evaluate the weighting function w. Then we look up the distance along the sampled direction from the cache record; this defines the next vertex of the path. In order to extend the path further we fetch the local surface properties from the RSM, update the path throughput, sample the outgoing direction, and continue tracing solely within the augmented visibility cache until the path is terminated. As such, the cost of tracing a secondary ray amounts to finding the nearest cache record and fetching few values from the RSM.

In order to evaluate the visibility caching in a reasonably complex environment, we modified PBRT [PH10] to render progressively and extended its BVH accelerator to maintain our visibility cache. During rendering, the modified BVH collects utility statistics u for each record (we increase u_i by $1/d_p$ whenever i is used; d_p is the path depth), computes the correlation values ρ (on the GPU), progressively refines the cache, and decides whether or not to use the visibility cache.

Figure 1 compares three variants of path tracing in the San Miguel scene that differ in how much they utilize the cache, i.e. either from the first or second bounce (see the illustrations in Figure 1). This scene is particularly challenging since it contains a lot of high frequency geometry. Some differences can be seen in the top part of the foliage on the right, which is slightly brighter than in the reference solution. Using the cache immediately after the first bounce can result in structured artifacts. These can be largely avoided by using the cache only from the second bounce. Tracing secondary rays using the visibility cache is 4.7× faster than with a regular BVH. The overall rendering is 1.56× faster, which is less than one would expect. This is due to the small relative number of secondary rays since the scene is "open" and the paths are thus rather short.

Figure 8 shows another example of path tracing in the Bedroom scene. Compared to the rendering time of a single frame (6.1 sec.) the overhead of maintaining the cache with 4000 records is negligible (21.9 ms). The performance of tracing secondary rays increases by a factor of 2.

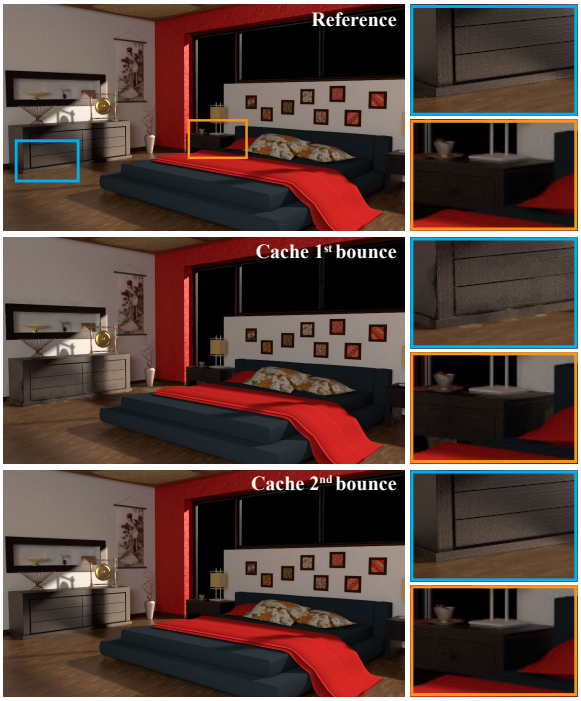

Figure 8: *Comparison of traditional path tracing (top) to image-space path tracing in the Bedroom scene (790k tris). The latter uses the visibility cache after the first (middle) or the second (bottom) bounce. The performance of tracing secondary rays using our visibility cache (0.67 MRays/s) is 2× higher than using a regular BVH (0.33 MRays/s).*

5. Discussion of Parameters

The accuracy of the cache is related to the area density of the visibility samples (and thus the number of cache records) and the resolution of the shadow maps, both of which define the memory footprint of the cache. To double the density we need to quadruple the number cache records. In our implementation, all cache records are placed in the GPU memory. Depending on the scene, we use between 2k and 4k cache records capturing the visibility using paraboloid shadow maps with 128×128 resolution.

For estimating the local visibility correlation, each cache record is linked to *four* neighbors that score the best w.r.t the weighting function w. Using fewer neighbors leads to poor estimation; more neighbors often create large neighborhoods and make the computation more expensive. To estimate the correlation between two records, we generate 1024 hemispherical directions. Increasing the number of directions did not improve the estimation. There may still be some variance in the importance function γ though, but this is due to the noisy utility statistics from the rendering algorithm. This can be seen in the accompanying video as flickering of the false colored Voronoi cells. We did not experience any noticeable impact of the variance on the refinement, but it can be further reduced e.g. by increasing the number of paths traced per one refinement step.

The user defined parameter α weights the impact of the correlation and the utility on the importance function γ. Setting α to an extreme value often leads to sub-optimal results, especially in complex scenes. If γ is solely based on the correlation, cache records will be placed around high frequency features, which may be hidden or unimportant for the current camera position. When γ depends on the utility only, the cache wastes a lot of records on regions with minor changes

in visibility. Figure 10 illustrates each of these cases. Keeping α between 0.3 and 0.7 produces desired results most of the time with 0.5 being a good initial value in general.

When placing the new cache record (Section 3.3), we compute the distance to its furthest neighbor d_f and set the radius of the tangent disk to $0.7 \times d_f$. We found that this value ensures that candidates are not placed too far from the record to be improved, but we still cover enough space to create at least one that is not too close to any of the existing records. To prevent creating a new record too far due to holes in the surface around j, we also discard candidates that are further than $1.25 \times d_f$ from the disk. Generating candidates that fulfill all criteria may become lengthy if the geometry in the target region happens to be high frequency (e.g. the foliage in Figure 1). Since this is undesired, we progressively double d_f, and half the number of required candidates (we start with 10) whenever 50 disk samples are not sufficient.

6. Limitations and Future Work

Caching the visibility in environments with high frequency geometry (e.g. trees) is generally challenging. Although our algorithm handles these cases, the discretized cached representation may be too coarse, leading to artifacts. This could be solved by tightly bounding all difficult geometry with proxies and using accurate accelerators for tracing rays *inside* the proxies.

In all our examples we use a straightforward approach to resolve the visibility using the cache: we simply "snap" the ray to the selected record and determine the visibility along the sampled direction. We compare this approach to ray marching in Figure 9, where the cache records are used also for direct illumination for better visualization. While marching along the ray and checking for an intersection with the height field defined by the shadow map improves the quality, it also comes at a certain cost, which in our examples did not pay off. An optimized implementation could yield better quality with performance only slightly worse than snapping.

While our approach is suitable for interactive previews with dynamic camera, the method is currently limited to scenes with static geometry only. Detecting (parts of) cache records that need to be updated due to dynamically changing geometry is an interesting future work.

7. Conclusion

The rationale behind using progressive algorithms is to provide low quality previews fast, and converge to high-quality results gradually over time. Our visibility caching naturally fits into these scenarios: we start with a coarse representation that is refined over time to minimize the artifacts. By reducing the cost of visibility queries we enable the rendering algorithm to draw more samples that are necessary to reduce the variance of indirect illumination estimators.

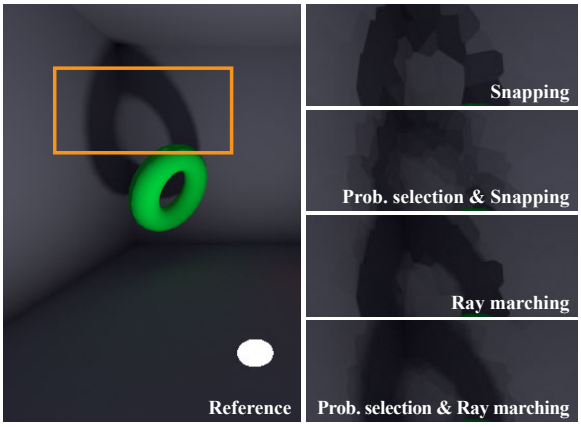

Figure 9: *Comparison of snapping and ray marching. The cache contains 1000 records and the refinement was disabled. First, we either use the nearest record or probabilistically select one of the nearest records. Then we either snap the ray to its center or march along the ray and check for an intersection with the shadow map.*

Acknowledgments. This work was supported by the DFG grant DA 1200/1-1.

References

[BRDC12] BASHFORD-ROGERS T., DEBATTISTA K., CHALMERS A.: A significance cache for accelerating global illumination. *Computer Graphics Forum 31*, 6 (2012). 2

[CAM08] CLARBERG P., AKENINE-MÖLLER T.: Exploiting visibility correlation in direct illumination. *Computer Graphics Forum (Proc. of EGSR 2008) 27*, 4 (2008). 1, 2, 3, 8

[CNS*11] CRASSIN C., NEYRET F., SAINZ M., GREEN S., EISEMANN E.: Interactive indirect illumination using voxel cone tracing. *Computer Graphics Forum (Proc. of Pacific Graphics 2011) 30*, 7 (2011). 1

[DKL10] DAMMERTZ H., KELLER A., LENSCH H. P. A.: Progressive point-light-based global illumination. *Computer Graphics Forum 29*, 8 (2010). 2

[DS05] DACHSBACHER C., STAMMINGER M.: Reflective shadow maps. *Proc. of the symposium on Interactive 3D graphics and games - I3D 2005* (2005), 203. 2, 5

[GH06] GHOSH A., HEIDRICH W.: Correlated visibility sampling for direct illumination. *The Visual Computer 22*, 9 (2006). 2

[GKPS12] GEORGIEV I., KŘIVÁNEK J., POPOV S., SLUSALLEK P.: Importance caching for complex illumination. *Computer Graphics Forum (Proc. of Eurographics 2012) 31*, 2 (2012). 2

[HOJ08] HACHISUKA T., OGAKI S., JENSEN H. W.: Progressive photon mapping. *ACM Transaction on Graphics 27*, 5 (2008). 2

[JDZJ08] JAROSZ W., DONNER C., ZWICKER M., JENSEN H. W.: Radiance caching for participating media. *ACM Transactions on Graphics (SIGGRAPH 2008) 27*, 1 (2008). 2

[KGPB05] KŘIVÁNEK J., GAUTRON P., PATTANAIK S., BOUATOUCH K.: Radiance caching for efficient global illumination computation. *IEEE Transactions on Visualization and Computer Graphics 11*, 5 (2005). 2

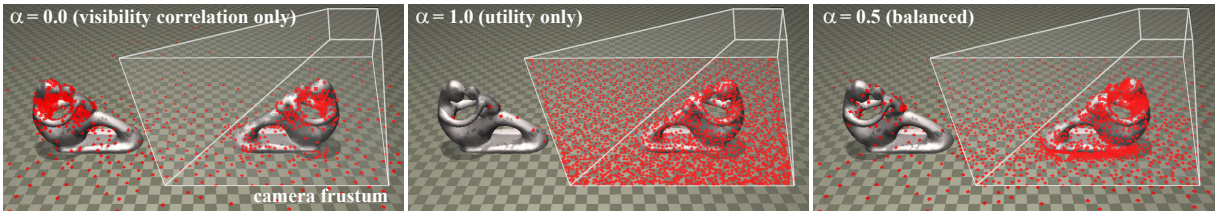

Figure 10: *The parameter α of the importance function γ trades between the visibility correlation ρ and utility statistics μ. If α equals 0 (left), the refinement of the cache is driven only by ρ placing records on surfaces with rapidly changing visibility. Setting α to 1 (center) concentrates most of the records in the camera frustum. Keeping α between 0.3 and 0.7 (right) provides a good balance between the two criteria.*

[LSK*07] LAINE S., SARANSAARI H., KONTKANEN J., LEHTINEN J., AILA T.: Incremental instant radiosity for real-time indirect illumination. In *Proc. Eurographics Symposium on Rendering 2007* (2007), Eurographics Association. 2

[ND12] NOVÁK J., DACHSBACHER C.: Rasterized bounding volume hierarchies. *Computer Graphics Forum (Proc. of Eurographics 2012) 31*, 2 (2012). 1

[PGSD13] POPOV S., GEORGIEV I., SLUSALLEK P., DACHSBACHER C.: Adaptive quantization visibility caching. *Computer Graphics Forum (Proc. of Eurographics 2013) 32*, 2 (2013). 2

[PH10] PHARR M., HUMPHREYS G.: *Physically based rendering, second edition: from theory to implementation*, 2nd ed. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2010. 6

[REH*11] RITSCHEL T., EISEMANN E., HA I., KIM J. D., SEIDEL H.-P.: Making imperfect shadow maps view-adaptive: high-quality global illumination in large dynamic scenes. *Computer Graphics Forum (Proc. of EGSR 2011)* (2011). 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. In *ACM Transactions on Graphics (TOG)* (2008), vol. 27, ACM, p. 129. 2

[RGKS08] RITSCHEL T., GROSCH T., KAUTZ J., SEIDEL H.-P.: Interactive global illumination based on coherent surface shadow maps. In *Proceedings of graphics interface 2008* (Toronto, Ont., Canada, Canada, 2008), GI '08, Canadian Information Processing Society, pp. 185–192. 2

[SNRS12] SCHERZER D., NGUYEN C. H., RITSCHEL T., SEIDEL H.-P.: Pre-convolved radiance caching. *Computer Graphics Forum (Proc. EGSR 2012) 4*, 31 (June 2012), 1391–1397. 2

[SWP11] SCHERZER D., WIMMER M., PURGATHOFER W.: A survey of real-time hard shadow mapping methods. *Computer Graphics Forum (Proc. of Eurographics 2010) 30*, 1 (Feb. 2011), 169–186. 2

[WDP99] WALTER B., DRETTAKIS G., PARKER S.: Interactive rendering using the render cache. In *Rendering Techniques (Proceedings of the Eurographics Workshop on Rendering)* (New York, NY, 1999), vol. 10, Springer-Verlag/Wien. 2

[Wil78] WILLIAMS L.: Casting curved shadows on curved surfaces. *ACM SIGGRAPH Computer Graphics* (1978), 270–274. 2

[WRC88] WARD G. J., RUBINSTEIN F. M., CLEAR R. D.: A ray tracing solution for diffuse interreflection. *ACM SIGGRAPH Computer Graphics 22*, 4 (1988), 85–92. 2

[YCK*09] YU I., COX A., KIM M. H., RITSCHEL T., GROSCH T., DACHSBACHER C., KAUTZ J.: Perceptual influence of approximate visibility in indirect illumination. *ACM Transaction on Applied Perception 6*, 4 (2009). 1

[YWC*10] YAO C., WANG B., CHAN B., YONG J., PAUL J.-C.: Multi-image based photon tracing for interactive global illumination of dynamic scenes. *Computer Graphics Forum (Proc. of EGSR 2010) 29*, 4 (2010). 2

Appendix A: Weighting Function *w*

To connect cache records with each other, their neighborhood needs to be defined. We connect each record to four nearby records that score the highest according to a weighting function *w*, which we adopt from [CAM08]:

$$w = \underbrace{\left(1 - \frac{\arccos|n \cdot n'|}{\pi}\right)}_{\text{angular difference}} \underbrace{\left(1 - \frac{d/d_{max}}{1+5d/d_{max}}\right)}_{\text{spatial difference}} \underbrace{\sqrt{1-|n \cdot v|}}_{\text{surface offset}}, \quad (2)$$

where n and n' are normals of the record and the neighbor, and v and d are direction and distance towards the neighbor. Figure 11 illustrates the individual terms.

Taking into account the *angular difference* between the respective surface normals prevents connecting surfaces with opposite orientation. The *spatial difference* mimics the observation that the falloff in correlation between cache records is large at first and then gradually becomes smaller. Finally, the *surface offset* penalizes candidates that may have similar orientation, but are offset mostly along the normal, i.e. the candidate sits on a surface that is above or below the cache record.

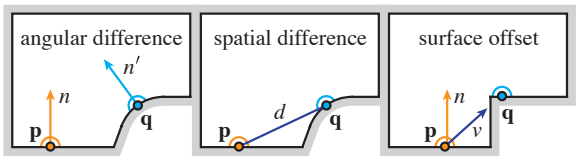

Figure 11: *The weighting function for selecting the nearest neighbors considers the relative orientation and offset of the corresponding surfaces (left and right), as well as the distance between the two points (middle).*