Many-Lights Real Time Global Illumination Using Sparse Voxel Octree

Che Sun and Emmanuel Agu(✉)

Computer Science Department, Worcester Polytechnic Institute, Worcester, USA
emmanuel@cs.wpi.edu

Abstract. The many-lights real time Global Illumination (GI) algorithm is promising but requires many shadow maps to be generated for Virtual Point Light (VPL) visibility tests, which reduces its efficiency. Prior solutions restrict either the number or accuracy of shadow map updates, which may lower the accuracy of indirect illumination or prevent the rendering of fully dynamic scenes. In this paper, we propose a hybrid real-time GI algorithm that utilizes an efficient Sparse Voxel Octree (SVO) ray marching algorithm for visibility tests instead of the shadow map generation step of the many-lights algorithm. Our technique achieves high rendering fidelity at about 50 FPS, is highly scalable and can support thousands of VPLs generated on the fly.

1 Introduction

Global illumination (GI) simulates the propagation of light through a 3D volume and its interaction with surfaces, dramatically increasing the fidelity of computer generated images. While offline GI algorithms such as ray tracing and radiosity can generate physically accurate images, their rendering speeds are too slow for real-time applications.

A new wave of GI algorithms targeting real-time applications such as video games have recently emerged. One class of these approaches is based on the many-lights method that is derived from Kellers instant radiosity technique [1]. The many-light method transforms solving the lighting transport equation into the calculation of direct illumination from many virtual light sources. This algorithm is hardware-friendly and can easily be implemented on modern GPUs. One shortcoming of the many-lights algorithm is that it requires many shadow maps to be generated for Virtual Point Light (VPL) visibility tests, which reduces the its efficiency. While imperfect shadow maps [2] can alleviate this issue, they are noticeably inaccurate for indirect shadows.

Another class of real-time GI methods discretize the original scene into voxels, which has several advantages: First, voxels are geometry-independent and many efficient scene voxelization methods have previously been proposed. Secondly, ray-geometry intersection and visibility tests on voxel data structures are very fast. Thirdly, high quality anti-aliasing techniques can also be implemented using voxel data [3].

© Springer International Publishing Switzerland 2015
G. Bebis et al. (Eds.): ISVC 2015, Part II, LNCS 9475, pp. 150–159, 2015.
DOI: 10.1007/978-3-319-27863-6_14

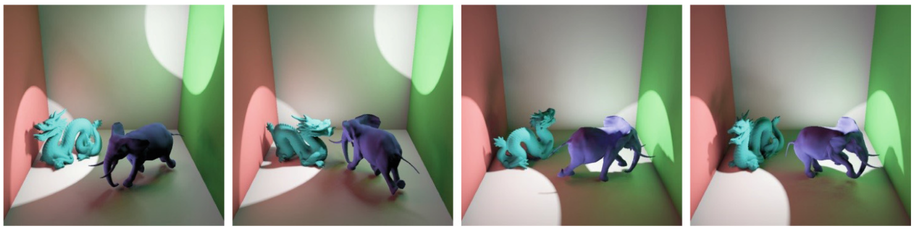

Fig. 1. Sample results using our technique: Dynamic Cornell box scene with dragon and running elephant rendered using two spot lights at 50 fps on an NVIDIA GeForce Titan X GPU. Indirect illumination is generated by 512 one-bounce virtual point lights

Inspired by discussions in Ritschel et al. [4], in this paper we introduce a hybrid real-time GI algorithm that combines the advantages of the many-lights method with those of voxelization. We use an efficient Sparse Voxel Octree (SVO) ray marching algorithm for visibility tests in place of the expensive shadow maps generation step of the many-light method. Our technique achieves high fidelity rendering quality at real time speeds (about 50FPS, shown in Fig. 1). Moreover, our technique is highly scalable and can support thousands of VPLs generated on the fly. Our main contributions are:

– An alternative real-time many-lights GI method that can support thousands of VPLs generated dynamically.
– Sparse Voxel Octree (SVO)-based VPL visibility tests at interactive rates without using computationally expensive shadow maps.

2 Related Work

Instant Radiosity Methods: Instant radiosity [1] uses many VPLs to approximate indirect illumination and several is the basis of many proposed real-time GI methods. To achieve high performance, some earlier methods only gather near-field VPLs, ignoring visibility tests [5]. Others such as Laine et al. [6] restrict movement in parts of the scenes in order to reuse VPL shadow maps over multiple frames. Ritschel et al. [2] efficiently generate inaccurate VPL shadow maps by using a point-based scene representation. Their method is very fast for one-bounce indirect illumination, but consumes lots of GPU memory bandwidth for second and third bounce VPLs. Knecht [7] exploits a temporal coherence technique to alleviate a flickering issue that occurs in real-time instant radiosity when there is an insufficient number of VPLs.

The distribution of VPLs within the 3D scene also affects the rendering quality of instant radiosity methods. Poor one-bounce VPL sampling leads to artifacts and unshaded scene surfaces [8]. To construct robust light transport paths, Tokuyoshi et al. [9] changed the VPL sampling strategy to bidirectional path tracing. Their algorithm uses global ray-bundles [10], which are groups of parallel rays sampling the scene geometry. Their technique is implemented via a

GPU-based per fragment concurrent link list [11]. However, global ray-bundles introduce additional burdens, since every time a VPL is used, scene polygons must be rasterized again in order to create a corresponding global ray-bundle. Consequently, while their technique produces photorealistic images for complex scenes, it does not run at high frame rates.

Voxel-based Methods: Voxel-based methods discretize the scene into 3D grid cells (voxels). An outstanding feature of scene voxelization is that it generates an approximation of scene geometric information extremely fast (usually less than several milliseconds on current commodity GPUs for a scene with over a hundred thousand triangles). Using this discretized scene representation, an iterative diffusion process can be performed to transfer light energy between neighboring grid cells [12]. Thiedemann et al. [13] introduces a regular grid-based voxelization technique that supports fast near-field VPL visibility tests. However, voxelization using regular grids consumes a lot of GPU memory. To solve this problem and its artifacts related to instant radiosity methods, Crassin [3] develops a fast dynamic GPU SVO generation method and uses it for high quality GI rendering: they utilize an injection and pre-filtering process to filter data related to indirect illumination in a bottom-up fashion. The resulting hierarchical octree structure is then used by their voxel cone tracing technique to produce high quality indirect illumination that supports both diffuse and high glossy indirect illumination.

3 Our Algorithm

Our lighting system is a deferred renderer, which supports illumination by multiple scene lights. First, we voxelize the scene and create an SVO to represent geometric occlusion. Similar to other deferred rendering systems, we then create a geometry buffer (G-buffer) [14] which stores position, normal and materials information for direct and indirect illumination. The G-buffer is then split for VPL interleaved sampling [15]. For each scene light we render the scene from the lights view and create a reflective shadow map [5] into which we store position, normal and flux information that is needed by VPL importance sampling [16].

Direct illumination is calculated using standard shadow maps and light contributions of all scenes lights are accumulated into a direct illumination buffer. Indirect illumination is calculated by accumulating contributions of VPLs into an indirect illumination buffer. To accelerate indirect illumination, we adopt a technique similar to Segovia [17]: each pixel being shaded is only assigned a subset of the VPLs using a pre-generated interleaved sampling pattern. A visibility test is performed by shooting shadow rays from the pixel toward the VPL subset. Here, instead of performing shadow map lookup and comparison, we perform SVO ray-marching to query if the shadow ray is occluded. To utilize cache coherence of modern GPUs, we group pixels that use the same VPL subset together by splitting the G-buffer into 4×4 tiles. On completion, the indirect illumination buffer is merged, filtered and combined with the direct illumination buffer to create the final image. Figure 2 shows our steps.

Voxel Data Representation: We use SVO as a spatial data structure for fast VPL visibility tests. Inspired by Crassin [3], we first build the SVO non-recursively and store it in GPU memory. Later the SVO structure is read by multiple GPU threads concurrently during VPL visibility tests. To exploit GPU thread-group caches, groups of eight tree nodes are placed together.

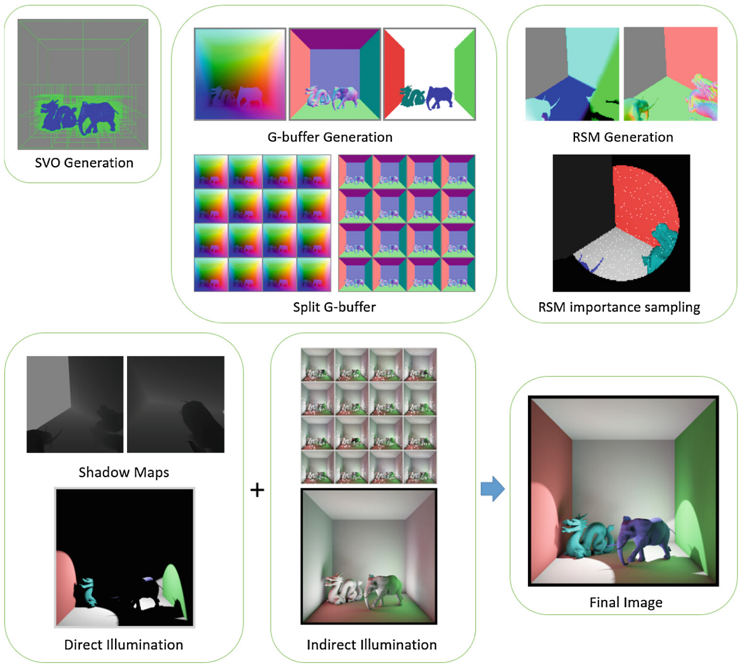

Fig. 2. Lighting pipeline steps. Top left: Scene voxelization and SVO generation. Top middle: G-buffer generation. World position, normal and diffuse materials rendered into floating-point textures. Position and normal buffers are split for calculating indirect illumination. Top right: RSM generated from scene lights view. First-bounce VPLs are created by sampling the RSM. Bottom left: Direct illumination using scene lights and corresponding shadow maps. Bottom middle: Indirect illumination calculated with split G-buffer and SVO ray-marching. Merging and filtering are performed to produce the final indirect illumination result. Bottom right: Adding direct and indirect illumination together. A simple HDR tone mapping is applied to produce the final image.

3.1 VPL Visibility Tests Using Sparse Voxel Octree

We use SVO to perform shadow ray marching between each surface point being shaded and the VPLs. To exploit data coherence, surface positions accessing the same VPL subset are grouped together. We also assume that the surface material only has a diffuse property for indirect illumination. This assumption leads to incoherent tree node access patterns due to the nature of diffuse reflection. Thus, packet traversal [18] cannot be applied easily as grouping individual shadow rays which have a uniform distribution over a hemisphere does not benefit much from fetching the same nodes during ray marching. However, dividing shadow rays into packets based on VPLs that reside in the same solid angle is a viable option to improve node access coherence, which we will explore in future work. Currently we only implement a per-ray traversal method. In Sect. 4, we will show that the algorithm has a good performance for diffuse indirect illumination.

The kd-restart traversal algorithm is an efficient method for ray marching thousands of rays concurrently on GPUs [19,20]. Consequently, we implement an SVO-restart traversal algorithm for VPL visibility tests. We also selected a kd-tree since it splits spatial regions in two as compared to the octree which splits the space into eight parts, making packet traversal more efficient on kd-trees. Our ray marching algorithm in pseudocode is shown in Fig. 3.

```
1 Transform world space ray start and end positions to SVO space
2 Offset SVO space ray start and end positions by epsilon
3 $rayDirSVO = rayendpositionraystartposition$
4 $sceneMaxT = length(rayDirSVO)$
5 normalize(rayDirSVO)
6 $hit = 0$
7 $minT = maxT = 0.0$
8 while( $maxT < sceneMaxT$ )
9     $curNode = root$
10    $mint = maxT$
11    $maxT = sceneMaxT$
12    $rayentryposition = raystartposition + raydirection * mint$
13    while( !isLeafNode(curNode) )
14        Figure out which child node of curNode the ray entry position is in
15        Update curNode to be the child node
16    if( Flaged(curNode) )
17        $hit = 1$ and break
18    Fetch AABB for curNode
19    Compute ray-AABB intersection positions and possibly return $hit = 0$
20    Update $maxT = t1 + SVO\_RAY\_T\_EPSILON$
21 return hit
```

Fig. 3. Pseudocode for our ray marching algorithm

The ray marching algorithm takes world space ray end positions and SVO root as input. It first transforms the ray end positions to SVO space. In line 2 the ray segment is shrunk slightly to avoid intersection issues because both the shading position and VPL position are on geometry surfaces that have been flagged as SVO leaf nodes. From line 3 to 7 we initialize some variables used by the subsequent while loop. Here, sceneMaxT is the distance between the shrink ray end positions. It is used as the ray marching termination condition when the ray goes through the leaf nodes. The hit variable indicates whether or not the ray hits a leaf node which has geometric information in it. Since we use SVO ray marching primarily for VPL visibility tests, whenever the ray hits geometry, the algorithm terminates and returns immediately. If we were to implement glossy reflection in future, this SVO-restart method could be extended easily to fetch hit position geometric data. Lines 8 to 21 lists the main while loop. The idea is similar to that of kd-restart algorithm: marching the ray through the leaf nodes iteratively and checking if shadow ray occlusion occurs. The differences here are (1) how we determine through which subdivided region we should keep marching the ray and (2) how we clip the ray each time it intersects a subdivided region. In our implementation we use the octree nodes AABB, which is easy and efficient to be implemented on a GPU. The implementation on line 19 is based on the ray-AABB intersection detection algorithm described in PBRT. In line 20 we update maxT using the result t1 from line 19, which is the maximum t value of the intersection position with the current nodes AABB. To make the algorithm numerically robust, we offset maxT using a user specified epsilon value. Otherwise the ray may stop marching forward in the next iteration due to inaccuracies caused by a floating point representation. Finally in line 21, the result is returned indicating whether the ray passed the VPL visibility test.

3.2 Rendering

Since we replace the time-consuming VPL shadow map generation with a scene voxelization pass, the GPUs main burden changes from performing hundreds or thousands of scene drawing and rasterization operations to a one-time scene voxelization and subsequent shadow ray tests for indirect illumination. One important advantage of our method is that implementing the quasi-random walk [1] for n-bounce VPL distribution is much more straightforward than shadow maps based real-time instant radiosity method. The reason is simple: since we already have a voxelized scene representation, against which shooting rays is extremely efficient. We just needed to extend our VPL sampling pass by adding second-bounce VPL sampling from first-bounce VPLs sampled with reflective shadow maps. In complex scenes such as the Crytek Sponza, second and third-bounce VPLs are useful for illuminating the scenes complex geometry.

In our system, standard deferred shading with scene light shadow maps is applied to produce a direct illumination buffer. For indirect illumination, we accumulate all the VPLs that have influence on a shading fragment and store the results in an indirect illumination buffer. Since we have split the G-buffer into 4×4 tiles with an interleaved sampling pattern, all the fragments inside the same tile use exactly the same subset of VPLs. In this way, data access to VPLs and SVO nodes exhibits good locality, which in turn improves the performance of shadow ray marching significantly. Similar to Dachsbacher et al. [5], given a VPLs flux Φ_{VPL}, world space position p_{VPL}, world space normal n_{VPL}, shading surface points position p and normal n, the indirect illumination at a surface position due to a VPL is formulated as follows:

$$E_{VPL}(p, n, p_{VPL}, n_{VPL}) = \Phi_{VPL} G(1 - V) \tag{1}$$

$$G = \frac{max(0, cos\theta_i) max(0, cos\theta_o)}{max(B, |p - p_{VPL}|^2)} \tag{2}$$

$$cos\theta_i = dot(n, -\frac{p - p_{VPL}}{|p - p_{VPL}|}) \tag{3}$$

$$cos\theta_o = dot(n_{VPL}, \frac{p - p_{VPL}}{|p - p_{VPL}|}) \tag{4}$$

We fetch the VPL related information from our VPL buffer, which has been generated by the VPL sampling stage of the system. V is the visibility term between the VPL and shading surface point. To evaluate V, we have to perform a shadow ray test using our SVO structure. Note that for shadow maps based instant radiosity method, evaluating V is just a simple shadow map lookup and comparison. In our case, since the evaluation of V is time-consuming, we could avoid a fruitless V term evaluation by first checking the luminance of the VPL and the G term. If either the luminance of the VPL is too small or the G term equals zero, we then skip accumulating this VPL to the indirect illumination buffer. B is the bouncing singularity constant factor used to clamp the distance between the VPL and the surface point. Since our technique currently only handles diffuse lighting, once the indirect illumination is complete, the buffer is merged and filtered for final combination with the direct illumination buffer.

4 Results

Our real-time global illumination method was rendered on an Intel i7 3930k CPU with an NVIDIA GeForce Titan X GPU. We tested using two scenes: A Cornell box with complex models (dragon and running elephant) and Crytek Sponza. All scenes are fully dynamic and no preprocessing methods are required.

The time spent on critical stages of our system are summarized in Table 1. The Cornell box scene is rendered at 768×768 (128^3 SVO) and Crytek Sponza scene is rendered at 1280×720 (256^3 SVO). The G-buffer, direct illumination buffer and indirect illumination buffer all have the same resolution as the back-buffer. For interleaved sampling of VPLs, the G-buffer is split into 4×4 tiles. Therefore, the VPL set is divided into 16 subsets, each of which is assigned to a G-buffer tile. The number of VPLs is varied and their influence on rendering speed and image quality is observed. Figure 4 shows the Cornell box scene rendered with 512 and 2048 VPLs respectively. Note that the visual difference between two settings is very small due to the accurate VPL visibility tests, which makes the integration of incoming radiance from VPLs converge very quickly. We also tested the Cornell box scene using a SVO resolution of 256^3, but the visual improvement is negligible. For Crytek Sponza scene, one-bounce indirect illumination is not enough to illuminate all the regions of the scene. Artifacts may appear in regions where insufficient number of VPLs are accumulated. Figure 5 shows the rendering result of the sponza scene using 1024 and 2048 first-bounce VPLs.

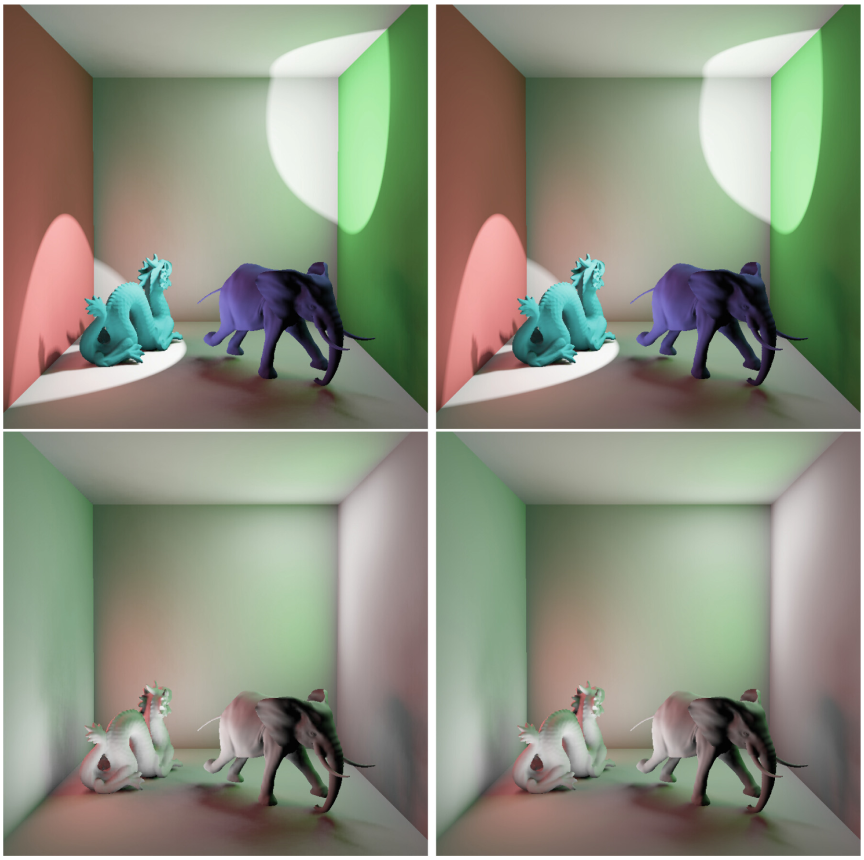

Fig. 4. Fully dynamic Cornell box scene with dragon and running elephant (150k triangles) rendered using two spot lights. SVO grid dimension is 128^3. Top left: Final image (512 first-bounce VPLs, 50 fps). Top right: Final image (2048 first-bounce VPLs, 15 fps). Bottom left: Indirect illumination (512 first-bounce VPLs, 50 fps). Bottom right: Indirect illumination (2048 first-bounce VPLs, 15 fps)

Table 1. Detailed timings for the Cornell box (128^3 SVO, 768×768 resolution, no MSAA) and Crytek Sponza scene (256^3 SVO, 1280×720 resolution, no MSAA) measured in milliseconds for critical stages of our lighting system. Three VPL number settings are used: 512, 1024 and 2048.

	Cornell box with dragon and running elephant	Crytek Sponza
Scene Voxelization and SVO Generation	0.57 / 0.57 / 0.58	4.65 / 4.65 / 4.65
RSM rendering	0.46 / 0.46 / 0.46	0.61 / 0.62 / 0.62
VPL Generation	0.02 / 0.02 / 0.02	0.02 / 0.02 / 0.02
Direct Illumination	0.17 / 0.17 / 0.18	0.31 / 0.31 / 0.31
Indirect Illumination	16.0 / 33.0 / 69.0	19.5 / 36.2 / 73.0

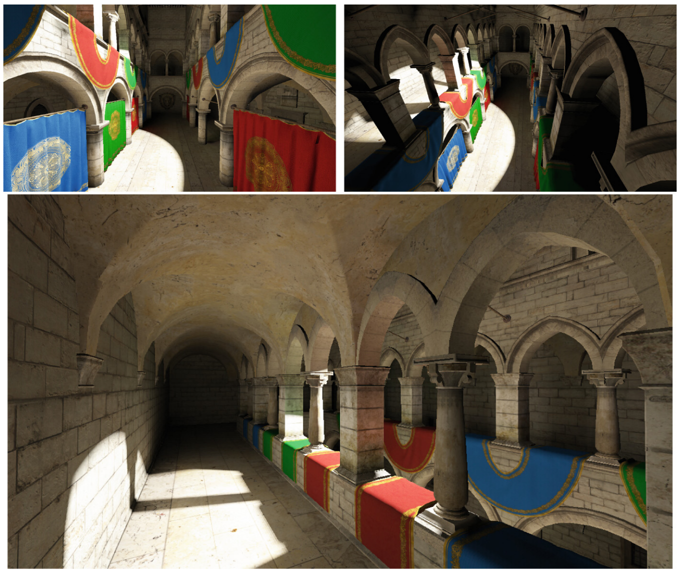

Fig. 5. Fully dynamic Crytek Sponza scene (180k triangles) rendered using one spot light and 1024 first-bounce VPLs. SVO grid dimension is 256^3. Top left: Final image from camera position 1 (19 FPS). Top right: Final image from camera position 2 (20 FPS). Bottom: Final image from camera position 3 (23 FPS).

5 Conclusion and Future Work

We have presented an alternate method to perform VPL visibility tests for instant radiosity-based real-time global illumination. While our method has a fixed voxelization cost and is not as efficient as fast shadow maps for computing one-bounce indirect illumination. However, for second- and third-bounce VPL sampling our technique is promising especially for complex scenes while maintaining real-time frame rates. Our method is automatic, supports fully dynamic scenes, and requires no scene preprocessing.

Similar to other real-time instant radiosity techniques, our method inherits limitations from instant radiosity. For instance, highly glossy surfaces cannot be reconstructed without having adequate VPLs evenly distributed in the entire scene. Meanwhile, applying interleaved sampling to scene surfaces not directly illuminated exhibits noticeable artifacts if a shading surface point fetches VPLs, most of which are blocked due to poor VPL distribution. To solve these issues, as mentioned before, our next step is implementing a robust VPL distribution and gathering method. We believe this can be done by taking advantage of the SVO data structure, using it as a scene geometric data representation for n-bounce VPL generation and rendering-time VPL acquisition. To support large scale scenes, a cascaded SVO grid management scheme could be implemented by following the idea of cascaded Light Propagation Volumes (LPV) [12] as well.

References

1. Keller, A.: Instant radiosity. In: Proceedings of the ACM SIGGRAPH, pp. 49–56 (1997)
2. 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. (TOG) **27**, 129 (2008)
3. Crassin, C., Neyret, F., Sainz, M., Green, S., Eisemann, E.: Interactive indirect illumination using voxel cone tracing. CG Forum **30**, 1921–1930 (2011)
4. Ritschel, T., Dachsbacher, C., Grosch, T., Kautz, J.: The state of the art in interactive global illumination. CG Forum **31**, 160–188 (2012)
5. Dachsbacher, C., Stamminger, M.: Reflective shadow maps. In: Proceedings of the ACM Symposium on Interactive 3D graphics and games, pp. 203–231 (2005)
6. Laine, S., Saransaari, H., Kontkanen, J., Lehtinen, J., Aila, T.: Incremental instant radiosity for real-time indirect illumination. In: Proceedings of the Eurographics Conference on Rendering Techniques, pp. 277–286 (2007)
7. Knecht, M.: Real-time global illumination using temporal coherence (2009)
8. Segovia, B., Iehl, J.C., Péroche, B.: Metropolis instant radiosity. CG Forum **26**, 425–434 (2007)
9. Tokuyoshi, Y., Ogaki, S.: Real-time bidirectional path tracing via rasterization. In: Proceedings of the ACM Symposium on Interactive 3D Graphics and Games, pp. 183–190 (2012)
10. Sbert, M., i Sàndez, X.P.: The Use of global random directions to compute radiosity: global Montecarlo techniques (1996)
11. Yang, J.C., Hensley, J., Grün, H., Thibieroz, N.: Real-time concurrent linked list construction on the GPU. CG Forum **29**, 1297–1304 (2010)
12. Kaplanyan, A., Dachsbacher, C.: Cascaded light propagation volumes for real-time indirect illumination. In: Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games, pp. 99–107 (2010)
13. Thiedemann, S., Henrich, N., Grosch, T., Müller, S.: Voxel-based global illumination. In: ACM Symposium on Interactive 3D Graphics and Games, pp. 103–110 (2011)
14. Saito, T., Takahashi, T.: Comprehensible rendering of 3-D shapes. ACM SIGGRAPH Comput. Graph. **24**, 197–206 (1990)
15. Keller, A., Heidrich, W.: Interleaved sampling. Springer (2001)
16. Clarberg, P., Jarosz, W., Akenine-Möller, T., Jensen, H.W.: Wavelet importance sampling: efficiently evaluating products of complex functions. ACM Trans. Graph. (TOG) **24**, 1166–1175 (2005)
17. Segovia, B., Iehl, J.C., Mitanchey, R., Péroche, B.: Non-interleaved deferred shading of interleaved sample patterns. In: Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Graphics hardware: Vienna, Austria, vol. 3, pp. 53–60 (2006)
18. Aila, T., Laine, S.: Understanding the efficiency of ray traversal on GPUs. In: Proceedings of the ACM Conf High Performance graphics, vol. 2009, pp. 145–149 (2009)
19. Foley, T., Sugerman, J.: KD-tree acceleration structures for a GPU raytracer. In: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics hardware, pp. 15–22 (2005)
20. Horn, D.R., Sugerman, J., Houston, M., Hanrahan, P.: Interactive Kd tree GPU raytracing. In: Proceedings of the ACM Symposium on Interactive 3D graphics and games, pp. 167–174 (2007)