Journal of Real-Time Image Processing
https://doi.org/10.1007/s11554-019-00919-1

ORIGINAL RESEARCH PAPER

Voxel transformation: scalable scene geometry discretization for global illumination

Bora Yalçıner[1] · **Yusuf Sahillioğlu**[1]

Received: 23 March 2019 / Accepted: 25 September 2019

Abstract
In real-time computer graphics, efficient discretization of scenes is required in order to accelerate graphics related algorithms such as realistic rendering with indirect illumination and visibility checking. Sparse voxel octree (SVO) is a popular data structure for such a discretization task. Populating an SVO with data is challenging when dynamic object count is high, especially when data per spatial location is large. Problem of populating such trees is adressed with our Voxel Transformation method, where pre-generated voxel data is transformed from model space to world space on demand, in contrast to the common way of voxelizing each dynamic object over each frame. Additionally, an accompanying filtering technique for voxel transformation is also proposed. This technique serves proposed system in two ways: (1) resolves issues introduced by the proposed fast and scalable voxel transformation method, and (2) enables smooth transitions between frames and handles the aliasing problem naturally as shown in the supplementary video. As an application use case, the proposed Voxel Transformation method is demonstrated in order to achieve indirect illumination using the well-known voxel cone tracing method. Results, which is compared with the standard voxelization method and ground-truth, are visually appealing and also scalable over large number of dynamic objects as shown in the supplementary video.

Keywords Voxelization · Cone-tracing · Indirect Illumination

1 Introduction

Modern graphics hardware is capable of rendering scenes that have high complexity on a high pixel density canvas. Even with fast hardware however, numerical calculation of realistic scene rendering with indirect illumination still is a challenging task for real-time applications. Many methods that can provide indirect illumination are not scalable for scenes with many dynamic objects. With this important indirect illumination application in mind, a real-time capable scene discretization method that can handle complex scenes with many dynamic objects is developed. Proposed method utilizes the idea of space transformations.

Scalability is our main motivation. This involves dealing with scenes consist of large number of dynamic objects. Contribution of dynamic objects, especially highly glossy objects, to indirect illumination increases believability of the scene. Temporal caching such as pre-computation of illumination are not applicable to such scenes since dynamic objects' temporal state changes over time. Scalability is made possible with proposed voxel transformation based data generation. The artifacts brought by this speed up are handled by the acompanying filtering technique.

After the related work section, proposed scalable discretization method and then the accompanying filtering technique are explained. Next, the usage of proposed method on the indirect illumination case is shown. Results are compared with the standard voxelization and ground-truth (please also see the accompanying video). Finally limitations, conclusions and future work recommendations are explained.

It should be noted that the source code and the executable for the method that are presented in this paper are publicly available.

✉ Yusuf Sahillioğlu
ys@ceng.metu.edu.tr

Bora Yalçıner
yalciner.bora@metu.edu.tr

[1] Middle East Technical University, 06800 Ankara, Turkey

Published online: 09 October 2019

2 Related work

2.1 Voxelization

Voxel based models have many usages like intersection testing, visibility testing and efficient spatial scene traversal. Such instances only require binary voxelization [1]. GPU implementations of such solid and sparse-solutions voxelization are also available [2]. Although these implementations utilize GPU, graphics pipeline (hardware rasterizer) are not utilized.

Hardware rasterizer can also be utilized in order to generate voxels [3, 4]. In order to fully voxelize surfaces without any gaps, conservative rasterization can be applied [5]. Instead of conservative rasterization, hardware multisample anti-aliasing (MSAA) pipeline can be employed to over-generate fragments per pixel in order to fill the potential gaps during voxelization [6]. Such methods either rasterize over a dense 3D structure or rasterize over a linear array. When algorithm rasterizes the voxels over a linear array, this array is sorted and voxels that occupy the same space are blended. Hardware voxelization methods have high throughput unless the required data is high which depends on the application.

2.2 Global illumination

Global illumination (GI), or equivalently indirect illumination, is a heavily-studied research area. We refer to the survey of Ritschel et al. [7] for a comprehensive research compilation about GI methodologies. In addition, Davidovič et al. [8] recently published a survey paper about light transport techniques. In this section, we will cover the prominent indirect illumination and real-time capable methodologies.

Rendering photo-realistic images mostly requires high quality approximation of the rendering equation [9]. Brute force approaches like path tracing and its improvements such as bi-directional path tracing [10] and metropolis light transport [11] give good results but these methods tend to converge slowly due to their unbiased nature. Another unbiased method, Instant Radiosity [12], uses virtual point lights (VPL) to simulate bounces of light from the light source. Quick determination of VPL visibility is tackled by Hasan et al. [13] and Sun et al. [14]. Although it is natively acceleratable by GPU hardware [15–17], the method has problems approximating high-frequency (specular) indirect illumination.

Biased approaches are introduced in order to approximate the indirect illumination faster. Most common biased technique is the photon mapping technique [18] which has hardware accelerated implementations [19–21]. Because of the biased nature, photon mapping has predictable calculation time. Recently, a hybrid solution is proposed by Georgiev et al. [22] which simulate the light transport with a biased start and end with an unbiased algorithm.

All of the methods described above have usage for generating illumination data for real-time applications. Real-time applications uses these generated illumination data by storing them on low resolution data structures in order to meet the computation time budget.

In real-time each rendering phenomenon is tackled individually. Specular indirect illumination is approximated by image-based lighting (IBL) [23, 24] or screen-space methods [25, 26]. Ambient light occlusion [27] is mostly approximated by screen-space methods [28, 29]. Screen space methods fail to incorporate out of screen geometry. Image based lightning solutions have hard time incorporating dynamic objects to global illumination.

Moreover, probe based [30], field based [31] or image based [24] and other light discretization based [32, 33] solutions are available for approximating global illumination. Pre-baked light maps which is a image based solution, were most common on real-time applications. However; these maps have long generation times and dynamic objects could not be able to contribute to the overall illumination of the scene. Light or scene discretization methods [24, 30–33]; like any other discretization method, is prone to information loss of rapid changes of occlusion or radiance over space because of the low resolution.

Another approximation for real-time global illumination is light propagation volumes (LPV) [34], which approximates the indirect illumination by using the reflective shadow maps [15] and a dense volumetric structure. It is designed to work in previous generation hardware (Playstation® 3 and Xbox® 360). LPV is mostly used for low-frequency (diffuse) illumination because of the low resolution data structure. It can, however, be adapted to high-frequency (specular) indirect illumination by increasing the volume density and sample count.

Another related approach is the voxel cone tracing method [35]. Cone tracing method relies on multi-resolution data structure for volume sampling in order to approximate the rendering equation [9]. Rendering equation's hemispherical integral is approximated by only few amount of cones compared to hundreds or even thousands of path tracing rays. Voxel cone tracing method is a combination of volumetric and a ray based approaches. Proposed method is applied to voxel cone tracing method in order to show its use.

3 Voxel transformation

The main idea comes from one of the core purposes of the graphics hardware, space transformation. Calculating high amounts of small linear transformations over vertices is one

of the fundamental capabilities of the traditional graphics acceleration devices. Just like vertices, voxels can also fit into this transformation logic quite nicely.

3.1 Pre-voxelization

In order to transform voxels, they are needed to be created. For voxel generation, only triangle surfaces are considered since it is the most common surface representation for real-time graphics. Each surface in the scene are voxelized and stored on a cache. For demonstration (Sect. 4), Crassin's voxelization method for this pre-voxelization task [3] is utilized. This is actually may not be desirable for certain tasks, because resulting voxels only covers the surface of the provided object. For global illumination use-case, this representation is more prone to light leaks compared to sparse solid or dense voxelization. However, it is one of the faster algorithms for generating voxels. The reason of this choice is to do a fair comparison between voxel transformation and voxelization both computationally and visually (Sect. 5). Unlike voxelization, voxel transformation does not suffer from additional run-time costs when choosing another computationally expensive voxelization method because all of the voxelization cost of objects are moved to initialization-time of computation.

Data that is required to be voxelized can vary from problem to problem. For indirect illumination case, diffuse albedo, specularity and normal information are sufficient to achieve pleasant results. For more sophisticated (for example physically-based) illumination, more data may be required.

3.2 Transformation types

In this section, two fundamental transformation cases which should cover most of the real-time animated objects are explained: basic linear transformations and weighted linear transformations.

Basic linear transformations

Basic linear transformations are applied to each vertex independently. This is the most basic case and do not have any caveats.

Weighted linear transformations

The most common method for animating objects is to bind a skeleton to an object and animate that skeleton instead of each vertex. For these kind of animated objects, pre-voxelization data is required to hold the weight index and the weights just like the vertex weights and the weight indices. For rasterizer based pre-voxelization, those weights are interpolated from each vertex and re-normalized into the voxel and stored. For large voxel sizes, even selecting weights of random vertices of that triangle gives pleasant results.

Weighted linear transformations can introduce minimal stretches on locations that are close to joints. One may argue that these stretches should create artifacts since the voxel size is fixed and cannot stretch like a triangle. For very small voxel sizes this is true, however current generation hardware does not have enough power to derive such high resolution of voxels in real-time. Additionally, accompanying filtering method further minimizes these artifacts, and they are not visually apparent on final image.

Other transformation methods For the use-case scenario provided in Sect. 4, most of the other animation methods have too subtle of an impact over the scene. For example, morph target animations which are used in order to animate humanoid face expressions would not be captured by the current voxel resolution limit because these animations have real-time computation requirements. Additionally since such animations deform the mesh, transformation over voxels could introduce gaps due to fixed voxel size. Because of that proposed method cannot cover morph targets.

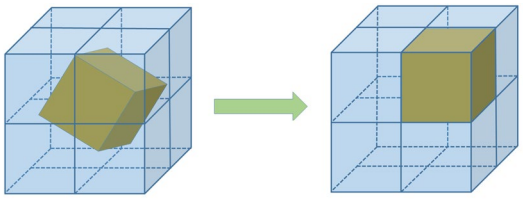
Traditional particle systems actually fit into this type of methods quite nicely. Similar to morph targets, voxels are too large to be as same size as a single particle. Unfortunately, particles that derive a sprite cannot be supported by voxel transformation algorithm since the surface information of such particles changes over time. Further limitations are explained in Sect. 6 in detail.

3.3 Filtering

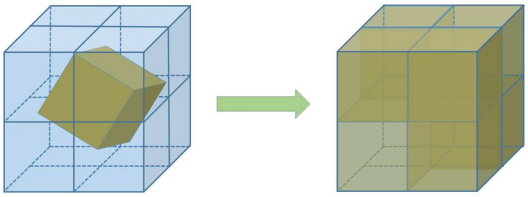
Fundamentally, pre-voxelized geometry is transformed over a static, world-space and axis-aligned grid. Since this grid represents discrete locations in space and transformation results are real numbers, the results are needed to be discretized in order to fit them into the grid. Most basic solution is to choose the nearest grid location (Fig. 1a). This is considered as nearest filtering.

Such naive voxel filtering does not have desirable results (Fig. 2, top). Nearest filtering allows multiple adjacent voxels to be on the same grid, thus creating voxel holes over the object. Results are required to be filtered into not a single grid location but all the neighboring grid locations depending on how much of the transformed voxels' volume resides on that gird location (Fig. 1b). This is called this 8-neighbor split filtering. 8-neighbor split filtering is temporally less jittery since it minimizes sudden jumps of transformed voxels.

However 8-neighbor filtering has higher computational cost, since each voxel splits into eight neighboring location. At each location, algorithm is required to blend voxel fragments that resides on the same space.



(a) Nearest Filtering



(b) 8-Neighbour Split Filtering

Fig. 1 After the transformation (which includes a rotation in this case), voxels are needed to be aligned with the world space voxel grid which is not always aligned with the world space axes. **a** Shows the nearest filtering method which snaps the transformed voxel into closest grid position. **b** Shows 8-neighbor filtering method which blends voxels into eight nearest neighbor grid slots and adjusts occlusion value of that grid according to space covered by voxels in that space

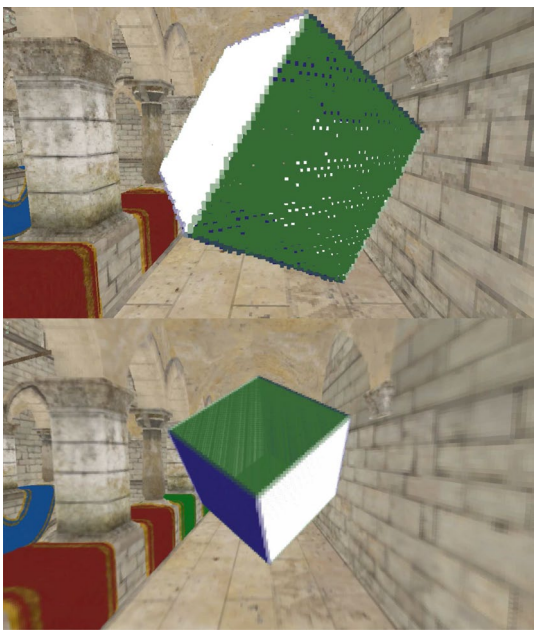


Fig. 2 Resulting images between two filtering methods. 8-neighbor filtering (bottom) method is computationally expensive. However it fixes the hole problem which is visible when nearest filtering (top) method is used. This visual discrepancy should not be that noticeable in an indirect illumination sampling unless the sampled surface is highly specular

4 Use case: cone tracing

Crassin's voxel cone tracing method [36] is chosen to incorporate proposed method in order to demonstrate method. It should be noted that voxel transformation method can be applied for other voxel based techniques when a high throughput scene geometry discretization is required.

In this section,how to utilize the generated world space voxels for indirect illumination using the voxel cone tracing method [35] is explained. Cone tracing method requires a sparse voxel octree (SVO) data structure. In Crassin's paper, the proposed SVO is split into two portions: a node hierarchy and a brick map. Node hierarchy defines the general tree structure and brick map, which is in a texture memory, holds the actual data. The reason of texture memory usage is to perform a hardware interpolation when you sample the actual data. Cones are sampled from higher levels of the structure depending on their current opening size which depends on the distance from the cone tip. This means that the proposed structure should hold data for every level, not only at the tree leaves.

There are couple of limitations of the proposed structure. First of all dynamic generations cannot overwrite static nodes of the SVO [36]. This is not a game-breaking limitation since most of the scene objects are rigid and rigid objects should not overlap with each other anyway. Another issue is the brick map set-up. Brick maps are designed to have fast sampling while having high construction time. This is purposely designed to be that way since most of the objects (i.e. static objects) are written on to the brick maps at initialization time.

Because our motivation is to design a highly scalable data structure in the presence of many dynamic objects, these limitations are not acceptable for our needs. We can overcome this problem by incorporating our high-throughput Voxel Transformation method to the Crassin's Cone Tracing algorithm. To this effect, a new SVO structure has to be designed.

4.1 Algorithm overview

Generic structure of the algorithm is comparable to the Crassin's Method. Figure 3 shows the overall voxel generation and usage. "SVO Average", "SVO Light Injection" and "Cone Tracing" parts are the same.

Apart from Crassin, a simple brute force Sparse Voxel Octree (SVO) generation and storage sceheme is incorporated instead of holding SVO as a Brick Map/Node Tree pair.

Unlike Crassin, propsed method first transforms the voxels from a pre-generated voxel cache. Then, it generates the SVO from scratch every frame. This is required because of our dynamic object scalability claim. Even though some

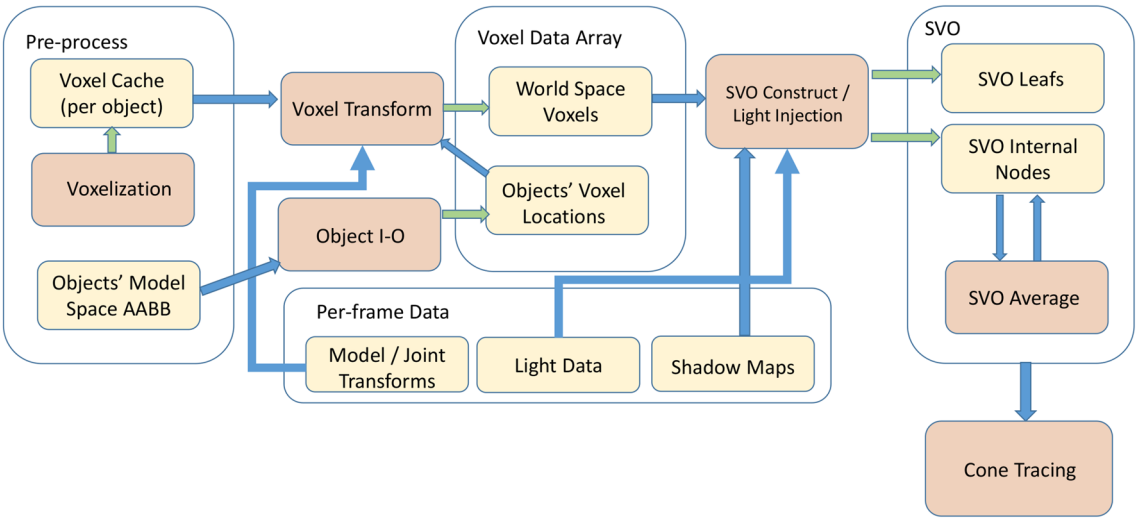


Fig. 3 Entire pipeline of the voxel cone tracing algorithm using voxel transformation. Orange boxes show processes and yellow boxes show data. Green arrows show that the process generated those data. Blue arrows show the used data by the process. Since the implementation is on the GPU, each orange box is a GPU Kernel

of the scenes in Sect. 5 section do not have many dynamic objects, pipeline considers them as dynamic in order to show the method's scalability. In a practical sense, this is not necessary unless most of the scene has dynamic voxels. Same argument can be done for light injection. Again, light is not required to be injected every frame if there are no spatial changes considering light and objects.

Additionally, proposed method requires to consider how to resolve overlapping voxels between dynamic and static objects if it wants to utilize portion of the previous frame's SVO. This makes the method unable to blend dynamic voxels over static voxels that resides on the same space since un-blending those voxels are impractical. Just like Crassin, dynamic voxels can be omitted if those are overlapped with static voxels. Thus making static object portion tree structure remains intact. In our situation, overlapping voxels are simply blended between dynamic and static objects since SVO is going to be generated from scracth on the next frame anyway.

Basic steps of the algorithm is as follows:

- Transform the voxels.
- Construct the SVO nodes.
- Inject the light over the SVO.
- Generate data for non-leaf nodes of the SVO.
- Do cone-tracing over the SVO.

4.2 Transformation

Up until this point technical details of the algorithm are not specified since actual method could be applied to many scenarios. In this section an example design of voxel transformation method is discussed over the use-case of indirect illumination calculation.

Pre-voxelized geometry is stored on voxel cache structure in which each voxel is stored linearly. Each voxel holds position that is stored as a 30-bit integer coordinates (10-bit for each dimension) relative to the corner of the axis aligned bounding box (AABB) of the object. Such AABB is on object-space.

Voxel Transformation is a two phase algorithm: allocation space determination and the actual voxel transformation.

GPU optimized algorithms favor simple data structures and contagious data access over these data structures. Furthermore dynamic space allocation on the GPU while GPU is doing some work has a performance penalty. With these facts in mind, voxel data array (Fig. 3) is a simple chain of the same sized arrays and these arrays are allocated in bulk. First phase of the algorithm, which is called "Object I-O", determines spaces on this chain for newly introduced objects. Such a case may happen when a moving object enters the scene, or at the initialization time. If enough space is not available, more chunks are appended to the chain. "Object I-O" also stores transformation scheme and transformation data indices for the voxel transformation algorithm. When an object leaves the scene, "Object I-O" marks the freed spaces available for future objects.

After location of every object is determined on this data structure, transformation algorithm starts. Main purpose of the transformation algorithm is to transform voxels according to the transformation scheme of the objects. Transformation algorithm converts each object-space voxel position from integer to world-space floating point by applying

Eq. 1. Then transformation matrix (or multiple matrices with weights in joint-transform case) are applied. After that world-space coordinates are discretized using Eq. 2.

$$p_{\mathrm{o}} = p_{\mathrm{AABB}} + i_{\mathrm{o}}\delta_x \tag{1}$$

$$i_{\mathrm{w}} = (p_{\mathrm{w}} - p_{\mathrm{grid}})/\delta_x \tag{2}$$

On Eqs. 1 and 2 , δ_x is voxel size, i_{o} and i_{w} are integer locations of voxels on object grid and world grid respectively, p_{w}, p_{o}, p_{grid} and p_{AABB} are positions of world-space, object-space, grid corner and AABB corner respectively. p_{o} to p_{w} conversion are done by transformation.

In order to support 8-neighbouring filtering, remainder of division operation on Eq. 2 is stored as well. This remainder will be used to split the transformed voxels into its 8-neighbours. On the other hand, quotient is converted into integer and stored as voxel position.

4.3 Sparse voxel octree (SVO) generation

Even though this papers aim is to explain voxel transformation method, an SVO structure should be provided in order to show proposed method over voxel cone tracing. Voxel cone tracing method requires quad-linear interpolation supporting SVO data structure, which means samples should be interpolated spatially and additionally between levels of the tree. This requirement mandates an efficiently accessible data structure with respect to spatial adjacency.

Algorithm 1: Atomic allocation pseudocode. Algorithm uses hardware compare and swap and atomic add operations. "VolatileCast" forces a global memory write on the GPU instead of a potential cache only write which guarantees that new node location is visible for all other threads.

```
1  function AtomicAllocate (node, allocator)
             Input  : node pointer to location that will
             point 8 children
             allocator holds index of the next available
   node position of one lower level
   Output: allocated location
2  if node < ALLOCATING then
3  |  return node;
4  end
5  old = ALLOCATING;
6  while old = ALLOCATING do
7  |  old = atomicCAS(node, EMPTY,
   |    ALLOCATING);
8  |  if old = EMPTY then
9  |  |  location = atomicAdd(allocator, 8);
10 |  |  VolatileCast(node) = location;
11 |  |  old = location;
12 |  end
13 |  threadfence();
14 end
15 return old;
```

SVO generation is as follows. A top-down atomic node generation scheme over a pre-allocated array of nodes is employed. Each tread is responsible for a leaf node and those threads tries to allocate required intermediate nodes for that leaf in a top down fashion. Since each thread runs on parallel using GPU, a spin-lock mechanism is utilized. Until allocation happens all other threads busy-waits. Allocation is marked as completed when the allocating node writes the index of the allocated node.

Even though SVO construction algorithm is parallel, at higher levels of the tree most of the work is done by small amount of threads while all other threads wait busily. This atomic congestion creates a quite a bit of performance penalty. In our tests, with high amount of voxels, SVO generation uses all of the real-time computation budget by itself. In order reduce computational budget, generation of the higher levels of the tree is eliminated completely. Which means that up until a certain level, tree is stored densely. With this method, both cone tracing over higher-levels will be faster since there will not be any tree traversal. Moreover neighbour data acquisition for interpolation will be simpler. The illustration of such partially dense tree can be found in Fig. 4. In this demonstration, Storing the nodes densely until level 6 ($64 \times 64 \times 64$) is a good trade-off between memory and performance.

In order to apply 8-neighbour filtering over the tree, each voxel will actually try to allocate 8-neighbouring leaves for each partial data of the tree. This is where most of performance cost of 8-neighbour filtering method comes from. Furthermore, overlapping voxels should be resolved between voxels since 8-neighbour split filtering will introduce data of multiple voxels will be filtered over same leaf node. In order to support parallel average operation of overlapping voxels, "moving weighted average" method is used similar to [3] using hardware atomics.

4.4 Neighbor pointers generation

In order to construct the SVO in real-time and support fast interpolation, neighboring pointer hierarchy is directly stored instead of generating brick maps according to the layout of the world space voxels. Although sampling from this structure is comparably less efficient, construction of such a tree is more scalable when employing GPU parallelism.

Until this point, only the leaf nodes have appropriate data for the solution and top down (parent to child) chain of pointers are generated. Additionally, all of the neighbouring pointer structures are not constructed yet.

In addition to the SVO node data, each node also holds three pointers for their forward axial neighbors. This can be considered as a three dimensional singly linked list. Only axial pointers are provided in order to save on memory. Axial pointers are enough to traverse over all 8-neighboring

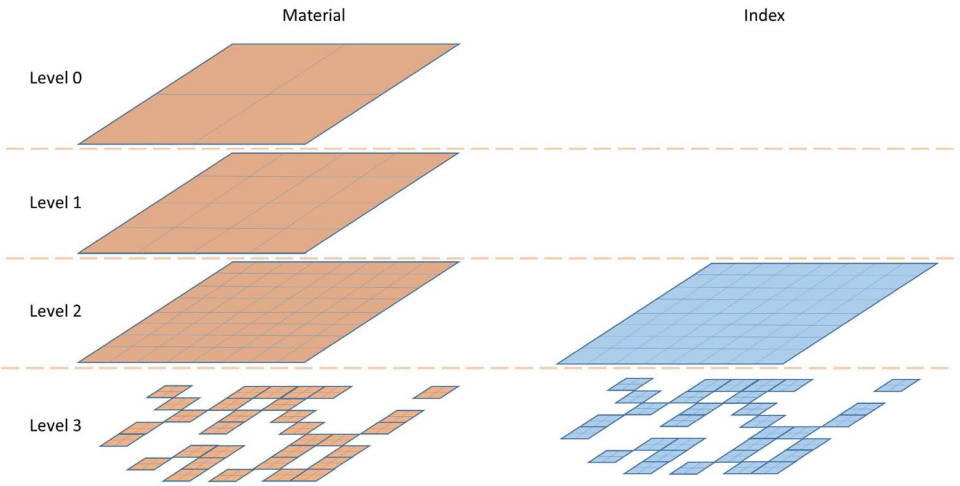


Fig. 4 Four level quadtree representation. First three levels are densely stored and last level is stored sparsely. Orange portion illustrates the data that will be sampled for indirect illumination calculation. All of the dense layers are required to be available for cone traversal. Blue portion illustrates the pointer hierarchy for traversing the sparse layers of the tree. Only the last dense layer is required to be allocated for storing initial pointers

voxel nodes. In order to rely on only forward pointers, sampling scheme should be defined. Each world-space position that is going to be sampled are rounded down to nearest node location. This guarantees all other interpolation locations to be accessible with forward pointers. In contrast, only backward pointers could be stored and sampled point would be rounded up the world-space positions instead.

In order to populate these pointers, each node on a level of the tree will try to allocate (or access) its backward neighbouring locations using (Algorithm 1) and sets proper pointers (Fig. 5). All of the 8 backward neighbouring locations are found and created this way. Constructing unavailable nodes are important since it guarantees that, on edge cases, each interpolation operation would be able to access a valid data. For example, a single lone node would create all its own 8-backward neighbours and sets pointer chain accordingly so that any sampled position would be able to access that data by first traversing to that level of the tree and then accessing data for interpolation by using neighbour pointer scheme.

Holding three pointers for each axial neighbors means that only non-diagonal neighbors can be accessed with a single pointer dereference. In order to access diagonal neighbors, multiple hops from the starting node is required. This pointer structure purposely designed in order to reduce the memory footprint of each node and the construction cost of the entire SVO tree. Additionally, we are going to hop every neighbouring location anyway since interpolation would require all 8-neighbouring nodes. Two tri-linear interpolations are done on neighbouring levels of the tree. Then these results are interpolated in order to create the final result.

4.5 Cone tracing

After the SVO is fully generated, cone tracing is done over the generated data structure. Sampling is comparable to the Crassin's method with slight deviations. Like Crassin, this implementation launches single specular cone and multiple diffuse cones in order to simulate the Phong Shading Model. Specularity of the object determines the specular cone aperture and diffuse cones are launched over different directions with a static aperture. Unlike Crassin's method, diffuse cones are launched at half resolution since high-frequency illumination would not require extra resolutions. Resulting diffuse sampled image is then filtered by a Gaussian filter. A sample resulting image can be seen in Fig. 6.

5 Results

Multiple scenes are utilized as shown through the figures and the supplementary video. Firstly, the run-time cost comparison between the real-time 6-separating thin voxelizer and voxel transformation is conducted. After that, total SVO generation timings and memory costs are provided.

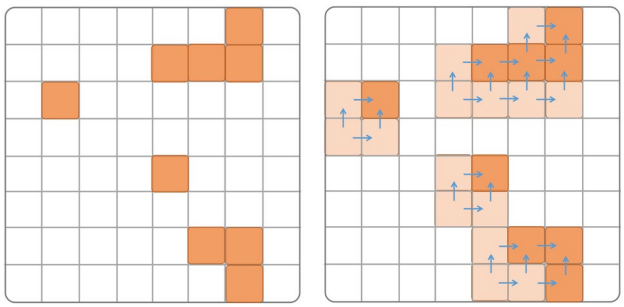


Fig. 5 2D grid representation of a quad-tree leaf. Orange boxes show the occupied nodes. The configurations before (left grid) and after (right grid) pointer and dummy node generation are shown. When algorithm samples a grid location, it can find all the required data by using pointers

Finally a comparison is shown of the application of the Voxel Transformation method over Voxel Cone Tracing and a ground-truth.

Scene information can be seen on Table 1. Throughout the scenes, skeletally animated Nyra character is used which has 54 individual parts all of which are driven by 22 joints. First scene is the modified Crytek Sponza Atrium scene with a single Nyra character patrolling the scene. Second scene only consists of single Nyra character, which is extensively used in the accompanying video in order to compare with the standard voxelization method and proposed voxel transformation method with different filtering schemes. Final scene consists of up to 256 Nyra characters with many other rotating tori and cubes. This scene is specifically constructed to show the scalability of the algorithm with many dynamic objects. On Table 2, 256 Nyra character version of the dynamic scene is used for time measurements.

Timings (in milliseconds) that are reported in this paper are obtained with Nvidia® GeForce GTX 980ti GPU and indirect illumination render resolution is chosen as 1280×720. On the other hand, direct illumination is rendered over an 1920×1080 canvas.

Fig. 6 Comparison between direct illumination (top) and indirect illumination sampled using transformed voxels (bottom). Scene is the modified Crytek Sponza Atrium scene with a single skeletally animated mesh. Sampling is done using the sparse voxel octree

5.1 Voxel transformation comparisons

Scaling comparison between real-time voxelization and voxel transformation can be seen on Fig. 7. Voxelization time converges toward 5.8 ms when object count goes to zero. This is because of the initial texture clear cost before voxelization starts. On the other hand, voxel transformation's scaling is directly proportional to the number of voxels that are required to be transformed. Around 256 complex dynamic objects, rasterizer based voxelizer completely saturates real-time frame budget (60 FPS, 16 ms). On the other hand Voxel Transformation barely passes 2ms mark at that same object count.

This comparison is done by swapping "Voxel Transfom" portion of the algorithm (Fig. 3) by a 6-separating thin rasterizer based voxelizer. Timings are only for those kernels, not the entire pipeline.

Direct renderings of SVO structures can be seen on Fig. 8. The images are generated using ray-tracing where ray blends traversed voxels considering their occupancy value as alpha.

Accompanying video has more extensive comparisons between different voxel resolutions and filtering methods in addition to voxelization method. 8-neighbour filtering represents the scene spatially more coherently. Additionally temporal transitions are much smoother as demonstrated on the supplementary video.

Timings of voxelization and voxel transformations can be seen on Table 2. Coverage values represent how much area of the underlying tree structure covers in the scene. First value represents the leaf voxel size. Second value represents the voxel count for each dimension. For example (0.6, 512) means that entire scene fits into 512^3 sized grid with leaf voxel size of 0.6 in scene distance units. Entire tree will cover $512 \times 0.6 = 307.2$ units. Dashed lines show that the scene is not constructed by that coverage value mostly because coverage being too small to hold entire scene inside.

6-separating rasterizer based voxelization scheme over a dense 3D canvas is used which is similar to [3]. A dense voxelizer is specifically chosen since it is the computationally most efficient algorithm. However memory cost is quite high because of that, 1024^3 sized voxel structures, GPU did not have enough memory for the data required. Same technique is used to pre-generate voxels for voxel transformation in

Table 1 Scene information, peak voxel counts and memory usage for each scene

	Sponza	Nyra	Dynamic
Triangles	280.7 K	25.7 K	6.6 M
Rigid dynamic	9	0	256
Joint dynamic	54	54	13824
Total objects	444	54	7170

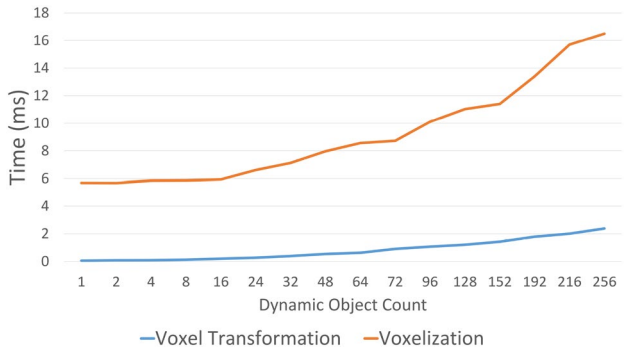


Fig. 7 Comparison between voxel transformation method and 6-separating thin voxelization with respect to different number of Nyra characters. Voxelization is done over 512^3 dense grid

Fig. 8 Comparison of nearest filtering (top) and 8-neighbor filtering (bottom). Images are generated using ray-tracing over the respective sparse voxel octrees

order to have fair visual comparison between methods. As it can be seen from the Table 2, our voxel transformation is faster on all cases.

Timings for other portions of the proposed algorithm (initialization time voxelization, SVO generation, down to top averaging and neighbour pointers generation) can be seen on Table 3. Pre-voxelization time specifically takes a long time. The reason for that is, each object on the scene should be voxelized individually since every object will be exposed to different transformations dictated by their model.

5.2 Indirect illumination comparison

In addition to the indirect illumination example given in Figs. 6, 9 contains the indirect illumination comparisons between the ground-truth, which is rendered using Arnold® Renderer, and Voxel Cone Tracing method. Figure 9 also shows the ambient occlusion comparisons.

Both specular sampling and diffuse sampling are queried with interpolation. Cone aperture is chosen as 30° for diffuse cones and for specular cones, cone aperture varies between 1° and 30° depending on specularity of surface material. Each pixel launches diffuse cones in circular pattern and then resulting frame buffer is blurred in order to simulate multiple diffuse cone per pixel. All of scenes are illuminated by a single directional light.

Cone-tracing computation costs are mostly bottlenecked by changes between camera angles. For example, if camera stares into a highly glossy material, computation cost of specular cone increases since specular objects require a narrower cone to be traced, which in turn increases the required SVO depth to be traversed. Computation cost On average total cost of cone-tracing over 1280×720 pixels varies between 5 ms and 25 ms. 25 ms is when entire camera stares on a mirror-like object which can be considered as a worst-case scenario. On scene by scene basis; Sponza scenes on average has 15 ms, dynamic scene has 12 ms, and single Nyra scene has 2 ms of overall cone tracing cost.

Images of proposed method are close to the ground-truth overall, however in detail, cone tracing method leaks light over dark places such as balcony (Figs. 6, 9a).

Ambient occlusion results can be seen in Fig. 9c and f. In some portions of the scene, like under the curtains, ambient occlusion is over-estimated and on flat surfaces it is underestimated.

6 Limitations

Since geometry is pre-voxelized, voxels' material information cannot change during run-time. Consequently, animated surface albedo, normal and specular properties cannot be supported by the proposed method. A solution for that problem can be achieved by holding multiple voxelized geometry for each different temporal state of that material and then material information can be interpolated depending on the time frame. However this will introduce additional memory cost for objects that have material animation. Overall animation over materials is a rare case.

Table 2 Timings (in milliseconds) of different coverage values for both voxel transformation and 6-separating thin voxelizer

SVO coverage	Voxel transformation			6-separating thin voxelizer		
	Sponza	Nyra	Dynamic	Sponza	Nyra	Dynamic
$(0.25, 1024^3)$	–	0.06	–	Out of Memory		
$(0.50, 1024^3)$	–	0.06	4.29	Out of Memory		
$(0.50, 512^3)$	–	0.05	2.45	–	0.11	–
$(1.00, 512^3)$	1.04	0.06	1.46	5.41	0.13	14.11
$(2.00, 512^3)$	0.32	0.05	1.25	3.11	0.11	10.13
$(4.00, 256^3)$	0.12	0.04	1.16	1.84	0.11	6.20

Coverage resolution can be interpreted as ("voxel span" and "grid count"). Dashed lines are present where grid size cannot enclose the entire scene

Table 3 Timings (in milliseconds) of different portions of the algorithm from different scenes and coverage values

Timings (ms)	Scenes		
Pre-voxelization	Sponza	Nyra	Dynamic
SVO generation			
Neighbor Ptr generation			
Down to top averaging			
$(0.25, 1024^3)$	–	692	–
	–	0.53	–
	–	0.15	–
	–	0.11	–
$(0.50, 1024^3)$	–	540	3408
	–	0.32	28.7
	–	0.15	3.07
	–	0.08	3.43
$(1.00, 1024^3)$	9766	588	4241
	16.12	0.32	13.67
	2.23	0.11	1.03
	2.67	0.08	1.07
$(0.50, 512^3)$	–	540	3408
	–	0.32	16.06
	–	0.13	1.71
	–	0.07	2.13
$(1.00, 512^3)$	9766	540	4241
	11.67	0.32	11.43
	1.49	0.13	0.86
	2.02	0.07	1.01
$(2.00, 512^3)$	6641	610	4396
	3.43	0.26	6.67
	0.50	0.12	0.31
	0.64	0.07	0.32
$(4.00, 256^3)$	6144	628	4695
	1.12	0.24	5.13
	0.17	0.09	0.15
	0.16	0.07	0.12

Coverage resolution can be interpreted as ("voxel span" and "grid count"). Again, dashed lines shows that the scene cannot be enclosed by that coverage values

Another limitation of this method is that it cannot support scale and shear transformations. Since voxels have static sizes, during run-time, object scale transformation will require us to introduce additional voxels in order to compensate the increased size of the object. However, this is not possible by the proposed system because voxel sizes and counts are pre-determined at the initialization time.

Pre-voxelization step only supports dynamic objects with either rigid or skinned transform, and cannot be extended for any heavily deformed objects with shape changes. We should, however, note that rigid or skinned transforms are the most common ones in computer graphics.

7 Conclusions and future work

Alternative to the voxelization technique, a novel method that is based on transforming pre-computed voxels is presented. With the proposed Voxel Transformation method, data generation for sparse voxel octrees (SVO) is shown to be faster than the version built on the classical voxelization method. It also produces temporally coherent data with the accompanying 8-neighbour filtering technique. Additionally, filtering technique improved the rendering by resolving the issue of hole problem, which is raised by the proposed voxel transformation method. The generated SVO is utilized in voxel cone tracing method to achieve realistic renderings with indirect illumination and ambient occlusion, which are comparable to the ground-truth counterparts. Finally, proposed generation scheme is scalable over high amount of dynamic objects.

In future, this method can be improved in two directions. Firstly, instead of averaging in the filtering, the pre-voxelized low-resolution voxels can be employed in order to resolve the non-leaf node data. Those voxels can again be transformed and stored in the appropriate locations. Secondly, applying N^3-tree [37] with values different than $N = 2$ should decrease the SVO reconstruction time by reducing the amount of pointers that are required to be generated while slightly increasing number of dummy nodes, which in

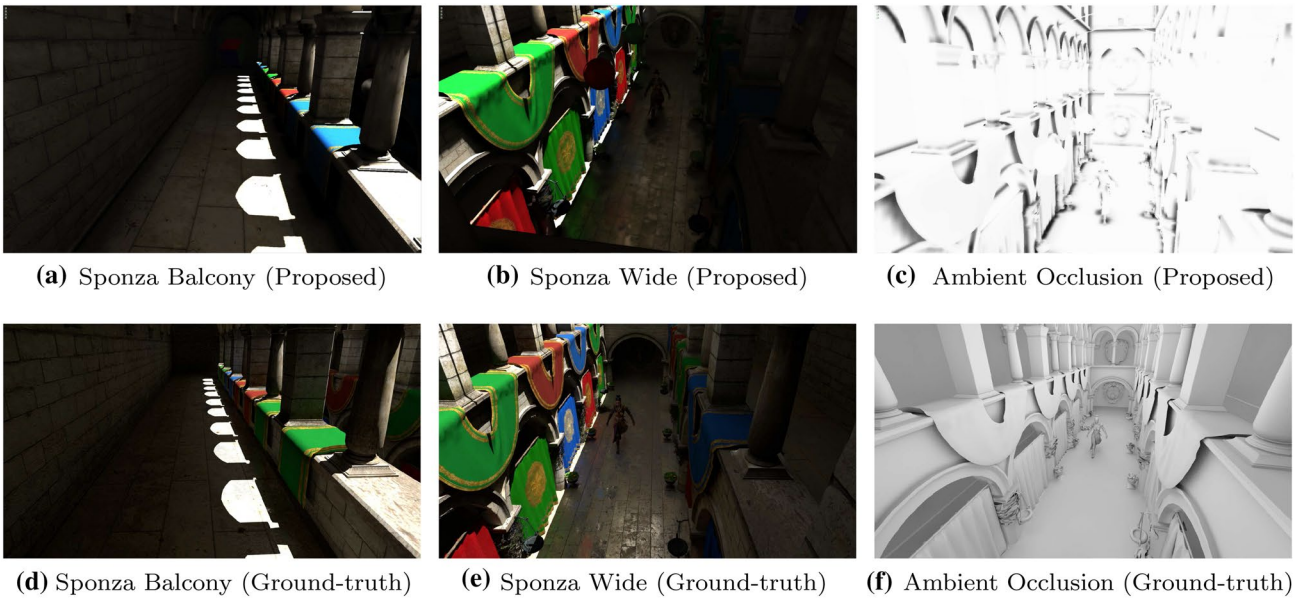


Fig. 9 Various comparisons between ground-truth and proposed method. Images on the left are rendered using proposed method. Images on the right are rendered using Arnold Renderer in order to simulate a ground-truth. Average rendering speed (excluding ambient occlusion which took a minute) for Arnold is around 10 minutes. All of the images on the right had real-time performance between 20 and 50 fps

turn should increase the overall memory cost. This trade-off may be desired for real-time scenes since the time constraint; most of the time, has a priority over the memory constraint.

Acknowledgements This work has been supported by the Scientific and Technological Research Council of Turkey (TUBITAK) under the Project EEEAG-115E471.

References

1. Eisemann, E., Décoret, X.: Single-pass gpu solid voxelization for real-time applications. In: Proceedings of graphics interface 2008, GI '08, pp. 73–80. Canadian Information Processing Society, Toronto, Ont., Canada, Canada (2008). http://dl.acm.org/citation.cfm?id=1375714.1375728
2. Schwarz, M., Seidel, H.P.: Fast parallel surface and solid voxelization on gpus. ACM Trans. Graph. **29**(6), 179:1–179:10 (2010). https://doi.org/10.1145/1882261.1866201
3. Crassin, C., Green, S.: Octree-based sparse voxelization using the gpu hardware rasterizer. In: OpenGL Insights. CRC Press, Patrick Cozzi and Christophe Riccio (2012)
4. Rauwendaal, R., Bailey, M.: Hybrid computational voxelization using the graphics pipeline. J. Comput. Graph. Tech. (JCGT) **2**(1), 15–37 (2013). http://jcgt.org/published/0002/01/02/
5. Hasselgren, J., Akenine-Möller, T., Ohlsson, L.: Conservative rasterization. In: M. Pharr (ed.) GPU Gems 2, pp. 677–690. Addison-Wesley (2005)
6. Takeshige, M.: The basics of gpu voxelization. https://developer.nvidia.com/content/basics-gpu-voxelization (2015). Accessed 3 Sep 2016
7. Ritschel, T., Dachsbacher, C., Grosch, T., Kautz, J.: The state of the art in interactive global illumination. Comput. Graph. Forum **31**(1), 160–188 (2012). https://doi.org/10.1111/j.1467-8659.2012.02093.x
8. Davidovič, T., Křivánek, J., Hašan, M., Slusallek, P.: Progressive light transport simulation on the GPU: survey and improvements. ACM Trans. Graph. **33**(3), 29:1–29:19 (2014). https://doi.org/10.1145/2602144
9. Kajiya, J.T.: The rendering equation. SIGGRAPH. Comput. Graph. **20**(4), 143–150 (1986)
10. Lafortune, E.P., Willems, Y.: Bi-directional path tracing. In: Compugraphics '93, pp. 145–153 (1993)
11. Veach, E., Guibas, L.J.: Metropolis light transport. In: Proceedings of the 24th annual conference on computer graphics and interactive techniques, SIGGRAPH '97, pp. 65–76. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA (1997). https://doi.org/10.1145/258734.258775
12. Keller, A.: Instant radiosity. In: Proceedings of the 24th annual conference on computer graphics and interactive techniques, SIGGRAPH '97, pp. 49–56. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA (1997). https://doi.org/10.1145/258734.258769
13. Hašan, M., Pellacini, F., Bala, K.: Matrix row-column sampling for the many-light problem. ACM Trans. Graph. **26**, 3 (2007). https://doi.org/10.1145/1276377.1276410
14. Sun, C., Agu, E.: Many-lights real time global illumination using sparse voxel octree. In: Bebis, G., Boyle, R., Parvin, B., Koracin, D., Pavlidis, I., Feris, R., McGraw, T., Elendt, M., Kopper, R., Ragan, E., Ye, Z., Weber, G. (eds.) Advances in Visual Computing, pp. 150–159. Springer International Publishing', Cham (2015)
15. Dachsbacher, C., Stamminger, M.: Reflective shadow maps. In: Proceedings of the 2005 symposium on interactive 3D graphics and games, I3D '05, pp. 203–231. ACM, New York, NY, USA (2005). https://doi.org/10.1145/1053427.1053460
16. 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. **27**, 5 (2008). (Proc. of SIGGRAPH ASIA 2008)
17. Ritschel, T., Eisemann, E., Ha, I., Kim, J.D.K., Seidel, H.P.: Making imperfect shadow maps view-adaptive: high-quality global illumination in large dynamic scenes. Comput. Graph. Forum **30**(8), 2258–2269 (2011). https://doi.org/10.1111/j.1467-8659.2011.01998.x
18. Jensen, H.W.: Realistic Image Synthesis Using Photon Mapping. A. K. Peters Ltd, Natick (2001)
19. McGuire, M., Luebke, D.: Hardware-accelerated global illumination by image space photon mapping. In: Proceedings of the conference on high performance graphics 2009, HPG '09, pp. 77–89. ACM, New York, NY, USA (2009). https://doi.org/10.1145/1572769.1572783
20. Yao, C., Wang, B., Chan, B., Yong, J., Paul, J.C.: Multi-image based photon tracing for interactive global illumination of dynamic scenes. In: Proceedings of the 21st eurographics conference on rendering, EGSR'10, pp. 1315–1324. Eurographics Association, Aire-la-Ville, Switzerland (2010)
21. Ritschel, T., Engelhardt, T., Grosch, T., Seidel, H.P., Kautz, J., Dachsbacher, C.: Micro-rendering for scalable, parallel final gathering. ACM Trans. Graph. **28**(5), 132:1–132:8 (2009). https://doi.org/10.1145/1618452.1618478
22. Georgiev, I., Křivánek, J., Davidovič, T., Slusallek, P.: Light transport simulation with vertex connection and merging. ACM Trans. Graph. **31**, XXX:1–XXX:10 (2012). SIGGRAPH Asia (2012)
23. Debevec, P.: Rendering synthetic objects into real scenes: Bridging traditional and image-based graphics with global illumination and high dynamic range photography. In: Proceedings of the 25th annual conference on computer graphics and interactive techniques, SIGGRAPH '98, pp. 189–198. ACM, New York, NY, USA (1998). https://doi.org/10.1145/280814.280864
24. Ritschel, T., Grosch, T., Seidel, H.P.: Approximating dynamic global illumination in image space. In: Proceedings of the 2009 Symposium on Interactive 3D Graphics and Games, I3D '09, pp. 75–82. ACM, New York, NY, USA (2009). https://doi.org/10.1145/1507149.1507161
25. Uludağ, Y.: Hi-z screen-space cone-traced reflections. In: Engel, W. (ed.) GPU Pro 5, pp. 149–192. CRC Press, Boca Raton (2014)
26. Hermanns, L., Franke, T.A.: Screen space cone tracing for glossy reflections. In: ACM SIGGRAPH 2014 Posters, SIGGRAPH '14, pp. 102:1–102:1. ACM, New York, NY, USA (2014). https://doi.org/10.1145/2614217.2614274
27. Zhukov, S., Iones, A., Kronin, G.: An ambient light illumination model. In: Drettakis, G., Max, N., (eds.) Rendering techniques '98: proceedings of the eurographics workshop, pp. 45–55. Springer, Vienna (1998). https://doi.org/10.1007/978-3-7091-6453-2_5
28. Shanmugam, P., Arikan, O.: Hardware accelerated ambient occlusion techniques on gpus. In: Proceedings of the 2007 Symposium on Interactive 3D Graphics and Games, I3D '07, pp. 73–80. ACM, New York, NY, USA (2007). https://doi.org/10.1145/1230100.1230113
29. Bavoil, L., Sainz, M., Dimitrov, R.: Image-space horizon-based ambient occlusion. In: ACM SIGGRAPH 2008 Talks, SIGGRAPH '08, pp. 22:1–22:1. ACM, New York, NY, USA (2008). https://doi.org/10.1145/1401032.1401061
30. McGuire, M., Mara, M., Nowrouzezahrai, D., Luebke, D.: Real-time global illumination using precomputed light field probes. In: Proceedings of the 21st ACM SIGGRAPH symposium on interactive 3D graphics and games, I3D '17, pp. 2:1–2:11. ACM, New York, NY, USA (2017). https://doi.org/10.1145/3023368.3023378
31. Greger, G., Shirley, P., Hubbard, P.M., Greenberg, D.P.: The irradiance volume. IEEE Comput. Graph. Appl. **18**(2), 32–43 (1998). https://doi.org/10.1109/38.656788
32. Vardis, K., Papaioannou, G., Gkaravelis, A.: Real-time radiance caching using chrominance compression. J. Comput. Graph. Tech. **3**(4), 111–131 (2014)
33. Jendersie, J., Kuri, D., Grosch, T.: Real-time global illumination using precomputed illuminance composition with chrominance compression. J. Comput. Graph. Tech. (JCGT) **5**(4), 8–35 (2016)
34. Kaplanyan, A., Dachsbacher, C.: Cascaded light propagation volumes for real-time indirect illumination. In: Proceedings of the 2010 ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games, I3D '10, pp. 99–107. ACM, New York, NY, USA (2010). https://doi.org/10.1145/1730804.1730821
35. Crassin, C.: GigaVoxels: a voxel-based rendering pipeline for efficient exploration of large and detailed scenes. Ph.D. thesis, Université de Grenoble (2011)
36. Crassin, C., Neyret, F., Sainz, M., Green, S., Eisemann, E.: Interactive indirect illumination using voxel-based cone tracing: an insight. In: ACM SIGGRAPH 2011 Talks, SIGGRAPH '11, pp. 20:1–20:1. ACM, New York, NY, USA (2011). https://doi.org/10.1145/2037826.2037853
37. Lefebvre, S., Hornus, S., Neyret, F.: GPU Gems 2 - Programming techniques for high-performance graphics and general-purpose computation, chap. Octree Textures on the GPU, pp. 595–613. Addison Wesley (2005)

B. Yalçıner received computer engineering BS degree from Bilkent University and MS degree from Middle East Technical University. He is currently pursuing PhD degree in Middle East Technical University. His research interest is mainly on real-time rendering of complex 3D scenes.

Y. Sahillioğlu received computer engineering BS degree from Bilkent University, MS degrees from Koc University and University of Florida, and PhD degree from Koc University. He worked as a post-doc in University of Pennsylvania. He is currently an Assoc. Prof. of computer engineering in Middle East Technical University. His research interests include computer graphics and vision. More details at: http://ceng.metu.edu.tr/~ys.