Octree Light Propagation Volumes

Olovsson, John David; Doggett, Michael

Published in:
[Host publication title missing]

2013

SIGRAD 2013, pp. 1–6
T. Ropinski and J. Unger (Editors)

Octree Light Propagation Volumes

John David Olovsson[1] and Michael Doggett[2]

[1]EA DICE, Sweden
[2]Lund University, Sweden

Abstract

This paper presents a new method for representing Light Propagation Volumes using an octree data structure, and for allowing light from regular point light sources to be injected into them. The resulting technique uses full octrees with the help of a separate data structure for representing the octree structure. The octree structure enables light propagation to be performed at a lower resolution than the base resolution, resulting in fast propagation and overall faster rendering times. The implementation of the technique is capable of rendering the Sponza scene at 9 *frames per second, which is 8% faster than the original LPV technique.*

1. Introduction

Realistic real-time illumination has been an important field of research in computer graphics and games for a long time. One large subset of real-time rendering techniques are dedicated to providing realistic local illumination. There are also techniques that try to model various aspects of global illumination, for example *diffuse indirect lighting*.

Unfortunately the nature of global illumination makes it computationally difficult to achieve in real-time. As such it is common for techniques targeted at real-time applications to use rather crude approximations or to rely on preprocessing. One problem with techniques relying on preprocessing is that they are usually static. The introduction of dynamic changes of the environment would require the preprocessing to be redone in order to retain correct illumination.

Light Propagation Volumes [Kap09] (LPV) is a technique which approximates global illumination, specifically diffuse indirect lighting, without any preprocessing stage, in real-time. While LPV runs in real-time, the uniform grid representation does limit the scalability. In order to achieve a more detailed and accurate result, the grid resolution would have to be increased. Unfortunately this also means that it would require more iterations to propagate the light throughout the scene. Such a high resolution representation would also be wasteful in any part of the scene which has no or relatively uniform geometry.

This paper proposes to use a full octree, where all nodes are allocated, instead of a uniform grid, to create a more efficient and scalable technique. We further investigate the possibility of using multiple fully dynamic point light sources instead of a single directional light source as used in the original LPV technique. Our approach incorporates omnidirectional point lights as the primary light sources and a dense octree implementation as the data structure. We present results that compare performance to an LPV implementation with a uniform grid and omnidirectional point lights.

2. Previous work

The LPV process is performed in four distinct steps. The scene is first rendered from each light source into *reflective shadow maps* [DS05]. The texels in these reflective shadow maps are then used to represent virtual point lights. After the reflective shadow maps have been created the contributions of the virtual point lights are injected into a uniform three dimensional grid. The grid encodes light color, intensity and direction as two-band spherical harmonics. After the injection stage these initial values in the grid are then propagated throughout the rest of the grid using an iterative process. In each iteration each grid element receives a contribution from each of the grid elements immediately surrounding it. The accumulated result of all the propagation iterations is then sampled during rendering to illuminate the final image.

LPV was extended with Cascaded LPV [KD10], which replaces the single uniform grid with multiple such grids with different density, positioned relative to the camera. These grids all have the same resolution, but because of the different densities they have different extents in the scene. A

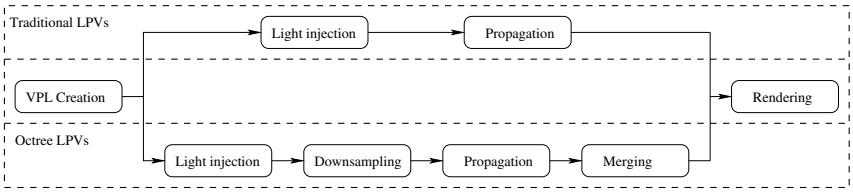

Figure 1: *An overview of the steps included in the octree based and the traditional light propagation volumes technique. The steps are divided into those that are the same for both techniques and those that need to be implemented specifically for one particular technique.*

high density grid is used close to the camera providing high detail close to the viewer while gradually lower density grids are used further away from the camera.

The data structure used in CLPV is in some ways very similar to that of the octree light propagation volumes which is presented in section 3. Both use multiple overlapping grids. The difference is that CLPV uses a small preset of grids which all have the same resolution. Octree LPV (OLPV) uses more volumes but with different resolutions. In CLPV the different grids also cover a differently sized area in world space, while OLPV grids all span the same volume, no matter their resolution. Also, CLPV does not need any separate grid to maintain the structure of the different grids. CLPV aims to prioritize spatial proximity to the viewer when it comes to the quality and accuracy of the technique. In this paper we focus instead on proximity to scene geometry and reflected indirect light instead.

Subsurface LPV [rea11] instead approximates the effects of subsurface light scattering by usilizing a structure identical to LPV. It also introduces a new method for injecting point lights into an LPV which is adapted by OLPV.

Sloan et al. [SKS02] introduced the concept of storing pre-computed indirect illumination using spherical harmonics. LPV uses this concept and in our implementation we use the first two bands of spherical harmonics to store light flux.

Crassin et al. [ea11] present a technique for real-time indirect illumination that uses octrees. Their technique uses a more complex sparse pointer based octree, resulting in more complex memory access patterns than our dense octree method.

3. Algorithm

There are essentially two important parts of the octree data structure. First and foremost are the octree levels themselves. These store the actual data, but have no information about the current structure of the octree. Then there is the index volume. It complements the octree levels by keeping track of how to index the octree levels and what structure the octree currently has. To create the initial light contribution, omni-directional light sources are rendered into six RSMs. These RSMs are converted to Virtual Point Lights which have their light propagated through the octree. The steps involved in this technique compared to the original technique are illustrated in Figure 1.

3.1. Octree Representation

We use a full octree which allocates memory for all nodes that could possibly exist down to a pre-defined level. By ensuring that this memory is laid out linearly, it is possible to compute the index or address for any node in the tree. Such a representation is highly inefficient with regard to memory usage if the represented data is sparse. It is however suitable in cases where the larger nodes represent a more coarse representation of the data.

Because of the ease of use and the very straightforward representation on the GPU, a full octree representation has been chosen to replace the light propagation volumes from the original technique. This imposes an additional cost in GPU memory but has the benefit that each individual level of the octree is just a uniform grid which is relatively easy to work with. There will essentially be several overlapping light propagation volumes of different resolutions covering the same scene.

In order to simplify the description of these octrees a basic notation is proposed. An octree is assigned a size, or resolution n, which should be a power of two $n = 2^1, 2^2, \ldots$. Given an octree that corresponds to a $n \times n \times n$ light propagation volume, the highest resolution level of that octree also has the size $n \times n \times n$. This level is the *first* level, or level $i = 0$. There are a total of $l = \log_2 n + 1$ levels in such an octree, $i = 0, \ldots, l-1$. Since every level of the octree is a three dimensional grid it can be indexed using three indices, $\hat{x}$, $\hat{y}$ and $\hat{z}$. Along with a level index i in which these indices are used they uniquely denote a single element in the octree. The indices are defined within the range $\hat{x}, \hat{y}, \hat{z} \in [0, \ldots, \frac{n}{2^i}]$. The notation $O_i(\hat{x}, \hat{y}, \hat{z})$ denotes the grid element on octree level i at $(\hat{x}, \hat{y}, \hat{z})$.

As a complement to the full octree representation a separate *index volume* can be introduced. The index volume would store integer indices instead of spherical harmonics.

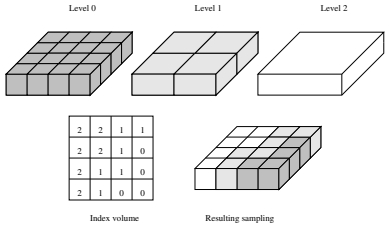

Figure 2: *A two dimensional slice of a flattened index volume, three levels of a full octree and the resulting sampling.*

This volume is denoted I. It could be represented as just another three dimensional structure of the same size n as O_0. Unfortunately, such a representation requires some quite inefficient operations to be added to the propagation step. In order to avoid this, a hierarchy of such structures is used instead. These are laid out in the same way as the data storage structures themselves and can be denoted I_i where $i = 0,\ldots,l-1$, just as for the data itself. It is also indexed using the same coordinates (x,y,z) as the data. Each element in the index volume is an integer index $I_i(x,y,z) = 0,\ldots,l-1$, originally initialized to $l-1$. This index is the number of the octree level to sample for a particular volume in world space. It will be populated with values during several of the steps performed throughout the technique. Details of this will follow later. During the sampling, the world position is then mapped to an element in the *index* volume. The value in the index volume is then used to do a second mapping into the correct octree level. The complication with this representation is that there are l index volume elements that could be picked for each world space position, one from each level i. To solve that, the element containing the minimum value is used.

An index volume I, with the levels I_i, $i = 0,\ldots,l-1$ where I_0 has size $n \times n \times n$, is considered to have size n. It can be indexed using the coordinates (x,y,z) where $x,y,z \in [0,\ldots,n-1]$ according to:

$$I(x,y,z) = \min_i I_i\left(\frac{x}{2^i}, \frac{y}{2^i}, \frac{z}{2^i}\right)$$

The resulting sampling in the octree corresponds to $O_{I(x,y,z)}(\frac{x}{2^{I(x,y,z)}}, \frac{y}{2^{I(x,y,z)}}, \frac{z}{2^{I(x,y,z)}})$. This is illustrated, using a flat visualization of the index volume, in Figure 2. Such a sampling is additionally given the simplified notation:

$$S(x,y,z) = O_{I(x,y,z)}(\frac{x}{2^{I(x,y,z)}}, \frac{y}{2^{I(x,y,z)}}, \frac{z}{2^{I(x,y,z)}}) \qquad (1)$$

There are several possible ways to use the proposed octree structure to replace an LPV. In this project it was chosen that the propagation of the injected light should be performed individually on each level of the octree. This allows the LPV propagation scheme to be used almost without any modifications in the octree representation. On first sight this may seem wasteful compared to just propagating on the highest resolution level, as in the original technique. However, the elements of the other levels in the octree cover a larger volume in world space. As a result the light on those levels will propagate further during each iteration. This allows the more detailed levels of the octree to be used to light the parts of the scene close to where the light was first injected. This will often be the areas where that light still has high enough intensity to be clearly visible. On the other hand, parts of the scene further away from this reflected light are unlikely to be hit by much indirect lighting. Because of that, those parts can be safely lit using a more coarse approximation of the indirect lighting.

3.2. Light Injection and Downsampling

Parts of the LPV technique were implemented using CUDA. CUDA was chosen to perform the light injection and light propagation in the place of shaders. Among other things this requires a different approach to light injection then the point based rendering used in the original papers [Kap09, KD10]. It also relies upon efficient sharing of memory between the GPGPU processing and the shaders.

Light is injected into the octree level O_0 in the exact same way as in the original technique. Each element $O_i(x,y,z)$, $i = 1,\ldots,l-1$ will cover exactly the same volume in world space as a specific set of eight elements from O_{i-1}. These elements are $O_{i-1}([2x,2x+1],[2y,2y+1],[2z,2z+1])$. In order to maintain a uniform range of light intensities in all octree levels, each element in O_i is populated by simply averaging the corresponding eight elements from O_{i-1}:

$$O_i(x,y,z) = \frac{1}{8}\sum_{\hat{x}=2x}^{2x+1}\sum_{\hat{y}=2y}^{2y+1}\sum_{\hat{z}=2z}^{2z+1} O_{i-1}(\hat{x},\hat{y},\hat{z}) \qquad (2)$$

The process of populating the layers $i \neq 0$ using the above averaging formula is called the *downsampling* step. It is considered as a separate step, since it is done after, and completely without any interaction with the injection step.

3.3. Propagation

After the injection and downsampling steps the octree contains an initial distribution of the diffuse reflected light in the scene. The index volume generally gives lower values close to geometry and higher values in empty areas. The next task is to allow this initial light distribution to propagate throughout the scene.

The propagation is done individually on each level of the octree. Each level is propagated in the same way as the single level was propagated in the original technique. But because of the hierarchy of the octree, light on lower levels of the octree is propagated further each iteration and can thus achieve coverage of the entire scene using much fewer iterations.

We use the light propagation scheme described by Kaplanyan and Dachsbacher [KD10], because it has better the-

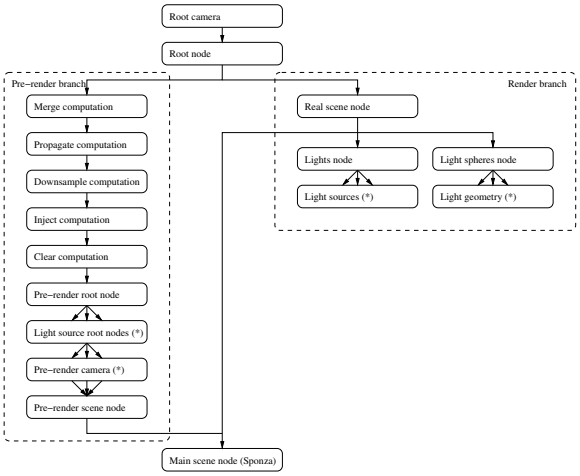

Figure 3: *Diagram showing the structure of the scene graph used in the real-time implementation. Non-essential branches have been omitted. Nodes marked with a star (⋆) and diverging incoming arrows, may exist in multiple copies for each parent node.*

oretical coverage. This includes maintaining a separate grid that contains the geometry of the scene.

3.4. Merging and Rendering

After the propagation, the scene can be rendered as in the original technique, with the exception that the sampling scheme from Equation 1 must be used. This is however not practical, since it would require the rendering pipeline to have access to, and be able to read both all the levels of the octree O_i, and all the levels of the index volume I_i. To avoid this an additional merge step is introduced. It stores all the sampled values using $S(x,y,z)$ into a single traditional $n \times n \times n$ LPV M where:

$$M(x,y,z) = S(x,y,z)$$

Once the merge step has been completed the result is just a single LPV that works in the exact same way as in the traditional technique. This also makes it possible to use the exact same rendering step as in the traditional technique.

3.5. Real-Time Implementation

A scene graph is used to put together the rendering passes and the setup is shown in Figure 3. The pre-render branch is responsible for performing all the steps of the technique from the creation of the RSMs to the merging step. Note that it does not perform the actual rendering. That is done in a separate branch, the rendering branch. Both these branches eventually end up in the main scene node which contains the loaded Sponza model.

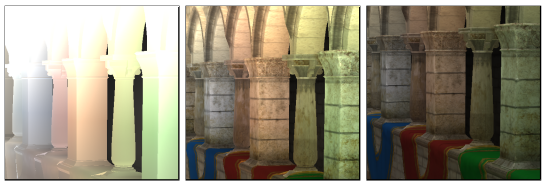

Figure 4: *Samples of rendered images using the real-time implementation. The leftmost image shows only indirect illumination without any textures. The center image shows indirect illumination but with textures. The rightmost image shows the final rendering with both direct light, indirect light and textures. Note that the effect of indirect lighting has been slightly exaggerated in these images.*

4. Results

We have implemented our algorithm using OpenSceneGraph with osgCompute and CUDA. The final version of the real-time implementation has been manually tuned both based on visuals, performance and simulated results. This version uses a total of $k = 4$ iterations, since further iterations no longer contributes visually to the resulting intensity levels. The octree has a size of 32^3 and the RSMs have resolution 1024×1024, while the effective resolution with regard to the LPV technique is 256×256. The volume is sampled only once in each 4×4 region. This allows for high quality shadows while maintaining reasonable performance during light injection. In our implementation the parameters for propagation are based on those used in NVIDIA's implementation of CLPV from [Cor0]. The propagation factor is set to $p = 3$. A rendering using our technique of the Sponza dataset is shown in Figure 4. Figures 5, 6 and 7 show the full version, and other examples.

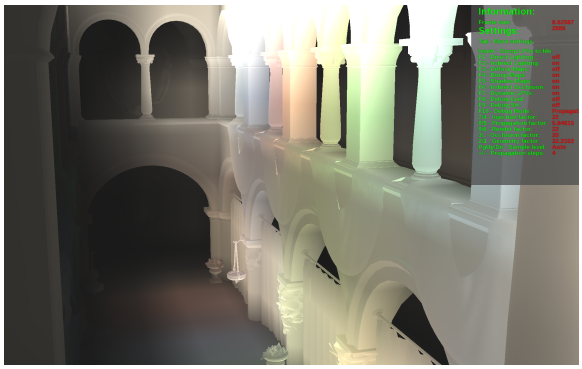

Figure 5: *The sponza scene rendered using only indirect illumination and without any diffuse texturing. Note that the indirect lighting has been slightly exaggerated in this image.*

There are small intensity variations between levels of the octree, but these have little visual impact on the final image. Also, light can propagate further in the lower levels of the

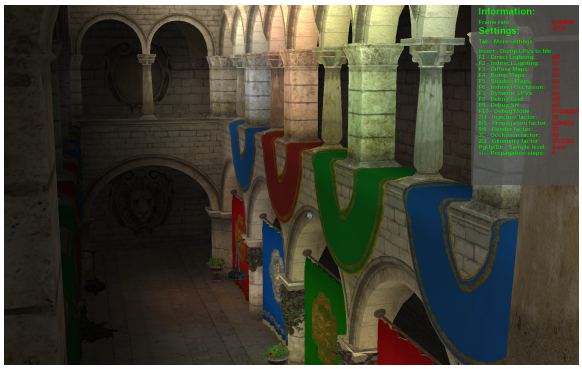

Figure 6: *The sponza scene rendered using only indirect illumination but with diffuse texturing. Note that the indirect lighting has been slightly exaggerated in this image.*

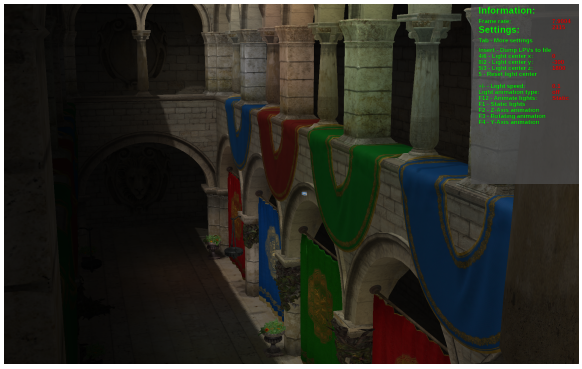

Figure 7: *The final rendering of the sponza scene with both direct and indirect illumination and diffuse texturing. Note that the indirect lighting has been slightly exaggerated in this image.*

octree leading to a slightly higher overall ambient lighting of the scene. Again this has minimal impact on the final image.

4.1. Error Measurement

There are a variety of ways to measure errors of the type of data contained in an LPV. Here we measure the average pairwise error between two spherical harmonics vectors a_e and b_e. The infinity norm $||v||_\infty = \max_i v_i$, or max norm, of the vectors is used to calculate the errors. Both the absolute error $||a_e - b_e||_\infty$ and the relative error $\frac{||a_e - b_e||_\infty}{||a_e||_\infty}$ is considered.

Each element in the octree can be denoted by four indices, the octree level i and the coordinates within that level (x,y,z). For each such element $i \neq 0$, it is possible to calculate an averaged value from the corresponding elements from level $i-1$ according to the downsampling in Equation 2. The downsampled value can be denoted $\hat{O}_i(x,y,z)$. There are then two error metrics, the average absolute error e_{abs}

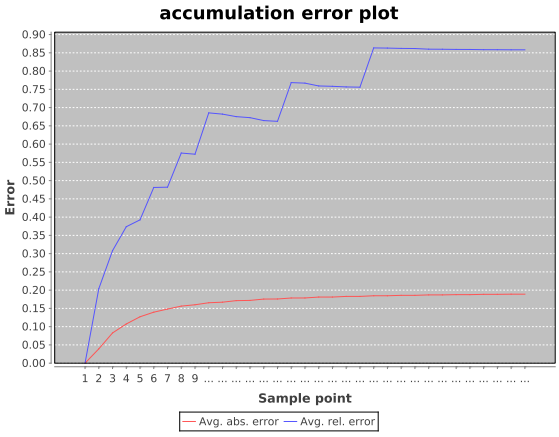

Figure 8: *The errors measured after each iteration during a simulation. Both the average absolute errors and the average relative errors are presented.*

and the average relative error e_{rel} of all elements in the octree $i \neq 0$. They are defined according to:

$$e_{\text{abs}} = \frac{7}{8^l - 8} \cdot \sum_{i=1}^{l-1} \sum_{p \in O_i} ||O_i(p) - \hat{O}_i(p)||_\infty$$

$$e_{\text{rel}} = \frac{7}{8^l - 8} \cdot \sum_{i=1}^{l-1} \sum_{p \in O_i} \frac{||O_i(p) - \hat{O}_i(p)||_\infty}{||O_i(p)||_\infty}$$

In the above expressions, $\frac{7}{8^l-8}$ is the inverse of the combined number of elements in $O_1, \ldots, O_{l-1}$, derived from the geometric sum, and $p = (x,y,z)$.

We have created a simulator that models just the light propagation. The two metrics e_{abs} and e_{rel} are the primary error measurements calculated by the simulator. It calculates the values for both of them after each iteration during the propagation. Figure 8 shows the errors for a simulation closely corresponding to the situation in the real-time implementation. This shows that the average relative error generally increases steadily up until approximately 20 iterations. After that it is essentially constant. This is simply because there is no noticeable light intensity left in the buffers at that point. This causes the accumulation buffer to remain the same after each iteration. The average absolute error exhibits a less erratic behaviour. It also stops increasing much sooner then the average relative error.

It makes sense for the errors to be increasing gradually each iteration. After all, during the downsampling, the values of O_i are chosen to be exactly that of $\hat{O}_i$. Each iteration then introduces a small error compared to what a newly downsampled value would give. This makes O diverge from $\hat{O}$ more and more until they stop changing. Using fewer iterations will result in less coverage by high resolution levels of the octree. At the same time it will also result in less diver-

gence from the ideal light distribution. Using many iterations can however void the problem with the higher divergence to some extent since the final solution would instead contain more of the higher resolution levels of the octree, and specifically O_0. Along with the related performance considerations it is still a trade off between speed, accuracy and visual quality.

4.2. Performance

The original LPV technique theoretically needs $k = 32$ iterations in order to propagate light throughout the entire volume, even though $k = 16$ iterations is often more then enough in practice. Using the OLPV technique, this number was lowered to $k = 4$. The intention is that this decrease should be able to more than make up for the added time to manage the more complex data structure.

All of the performance data in this section is generated on a computer with an NVIDIA GeForce GTX 480 graphics card. The focus has been on the newly added steps that are closely tied to the octree representation. This also includes modifications to previously existing steps from the original technique which were required for compatibility with the new octree structure. This includes the downsampling, propagation and merging steps, but generally not the creation of RSMs, light injection or the final rendering.

To ensure that timings were accurate, the CUDA kernels were run with `CUDA_LAUNCH_BLOCKING=1` to disable asynchronous kernel launches. This only ensures that CUDA kernels are not asynchronous, so other CUDA operations and OpenGL operations could run synchronously. We rendered 500 frames and timing results are shown in Table 1.

Table 1 shows that the OLPV technique outperforms traditional LPV by 8%. The propagation is the stage of the technique which experiences the highest increase in speed, almost 60%. This follows from the decrease in iterations from $k = 16$, when using the uniform implementation, to $k = 4$, when using the octree implementation. This speed increase is partially countered by the addition of the downsampling step, the merge step and managing the index volume.

The NVIDIA implementation of CLPV [Cor0] reaches a frame rate of between 60 and 80 frames per second with similar settings, but with a single directed light source. This equals a single RSM, rather than the 18 that are used in the implementation in this project.

5. Future Work

There are several possible improvements to the OLPV technique. Currently there is only one propagation factor for the entire octree. Analytically computing separate propagation factors for each level of the octree would give a more realistic propagation in the lower levels. Also propagating light on the lowest level of the octree instead on each level could improve performance and enable investigation of more sparse octree representations.

The biggest performance gain could be achieved by using a more efficient implementation of the light injection, such as point based light injection. Several optimiziations of the CUDA implementation are possible. Including making better use of local, shared and texture memory instead of global memory, ensuring memory access is optimized for cache access, and optimizing branching within warps to ensure threads are less divergent.

6. Conclusion

The paper presented a new technique for Light Propagation Volumes using octrees. It uses a representation based on full octrees and adds a so called index volume which keeps track of which level of the octree to use for each part of the scene. On top of the original technique and the new octree representation, the technique also adds new steps of downsampling the injected light and merging the octree into a traditional uniform LPV. Also light is injected from omnidirectional point light sources rather than just directional light sources.

With more restricted usage of RSMs and careful artist control of the scene, we believe that this technique could be used for real-time global illumination effects in games.

References

[Cor0] CORPORATION N.: Nvidia direct3d 11 sdk - diffuse global illumination demo. http://developer.nvidia.com/nvidia-graphics-sdk-11-direct3d, 0. 4, 6

[DS05] DACHSBACHER C., STAMMINGER M.: Reflective shadow maps. In *Proceedings of the 2005 symposium on Interactive 3D graphics and games* (New York, NY, USA, 2005), I3D '05, ACM, pp. 203–231. 1

[ea11] ET AL C. C.: Interactive indirect illumination using voxel cone tracing. http://research.nvidia.com/sites/default/files/publications/GIVoxels-pg2011-authors.pdf, 2011. 2

[Kap09] KAPLANYAN A.: Light propagation volumes in cryengine 3. http://www6.incrysis.com/Light_Propagation_Volumes.pdf, 2009. 1, 3

[KD10] KAPLANYAN A., DACHSBACHER C.: Cascaded light propagation volumes for real-time indirect illumination. http://dx.doi.org/10.1145/1730804.1730821, 2010. 1, 3

[rea11] RLUM ET AL J. B.: Sslpv subsurface light propagation volumes. http://cg.alexandra.dk/wp-content/uploads/2011/06/SSLPV_preprint.pdf, 2011. 2

[SKS02] SLOAN P.-P., KAUTZ J., SNYDER J.: Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 2002), SIGGRAPH '02, ACM, pp. 527–536. 2

		Octree	(ms)		Uniform	(ms)
Part	**Count**	**Average**	**Total**	**Diff**	**Average**	**Total**
Frame	500	106	53193	-4801	115	57994
RSMCreate	9000	0	3053	316	0	2737
GVClear	500	1	671	-50	1	721
GVInject	1500	10	15593	-110	10	15703
GVDownsample	500	0	55	55	-	-
IXClear	500	5	2923	2923	-	-
LPVClear	500	8	4461	270	8	4191
LPVInject	1500	13	19929	-2884	15	22813
LPVDownsample	500	0	142	142	-	-
LPVPropagate	500	7	3788	-5615	18	9403
LPVMerge	500	0	39	39	-	-
Total				-4914 8.28%		

Table 1: *Data specifying the amount of times each part of the technique was executed and how long these executions took, both in total and on average. The data is presented both for the octree based implementation and the implementation using uniform light propagation volumes.*