

Screen space light propagation volumes

Alen Leban¹, Uroš Šmajdek¹, Matija Marolt¹, Ciril Bohak¹, Žiga Lesar^{1,2}

¹Univerza v Ljubljani, Fakulteta za računalništvo in informatiko

²Univerza v Novi Gorici

E-mail: al2719@student.uni-lj.si, {uros.smajdek, matija.marolt, ciril.bohak}@fri.uni-lj.si, ziga.lesar@ung.si

Abstract

We present an adaptation of screen-space light propagation volumes (SSLPVs) method, for approximating indirect diffuse global illumination as a post-processing effect. The proposed method relies only on the G-buffer, which makes it easy to integrate into existing deferred rendering pipelines. The method works by treating visible pixels as virtual point light sources (VPLs) that accumulate their radiance in a cascaded light propagation volume (LPV) grid in the form of spherical harmonics (SHs). To avoid temporal flickering, we introduce temporally smooth LPV updates and align the grid globally to prevent invalidating the values by small camera movements. The method produces plausible indirect diffuse illumination while achieving interactive performance on consumer-grade graphics hardware. Code is available at: <https://github.com/AlenLeban/NRG-SSLP>

1 Introduction

Path Tracing (PT) is the state-of-the-art method for calculating realistic lighting in computer generated graphics by simulating millions of light rays as they bounce around the scene. The main downside to this method is the large number of samples required for the final image to converge, especially for the purposes of real-time graphics. If we can afford to approximate lighting, faster methods can be used to quickly simulate how light propagates throughout the scene using LPVs.

The main idea of LPVs is to extract lighting information from parts of the scene, simulate its propagation through space, and store its radiance in a dedicated data structure (such as a regular 3D grid) for fast lookup when we later want to use that information while shading a fragment. But since different game engines define light sources in their own way, it would require implementing this algorithm for each engine separately. To avoid this problem, we propose a screen-space implementation of LPVs – SSLPVs that serves as a post-processing step on top of any deferred rendering pipeline.

2 Related work

The main inspiration for this work was the contribution by Kaplanyan [1] about their LPV implementation in

CRYENGINE 3¹, where they sample from the main light’s point of view and stores their radiance in an LPV using SHs. They use ideas of an earlier reflective shadow maps technique by Dachsbacher and Stamminger [2], where pixels of shadow maps are treated as VPLs. Sloan *et al.* [3] similarly use SH to store precomputed radiance transfer functions around an object’s surface. Ray marching VPLs can also simulate light scattering effects in a participating medium [4] with results comparable to single scattering volumetric shadow maps. Similarly, regular grids of radiance probes storing low-order SHs have been applied to approximate indirect illumination and multiple scattering in participating media during volume rendering [5]. Mixed reality applications use LPVs blending virtual objects into the captured environment by estimating global lighting from the captured data to relight virtual objects [6]. Area lights can be simulated using LPVs by sampling the light’s surface to acquire VPLs which would subsequently be introduced into LPVs [7].

Aranha *et al.* [8] already implemented SSLPVs, where they also sample the scene from the main light’s point of view and generate two-bounce VPLs by efficiently sampling a paraboloid shadow map, combined with reflective shadow maps. In contrast, our work has access to lighting information only through the G-Buffer, which produces less accurate results, however it’s much simpler to implement.

3 Method

An LPV is represented as a grid of cells storing radiance values in parts of a scene. Once such a grid is precomputed, lighting can be sampled from it as one would sample a 3D texture, and then used while lighting a nearby pixel.

The first step requires adaptively sampling the scene by picking surfaces, that are directly illuminated by the main light source (according to its depth map). Think of the sampled points as places where light rays hit and scattered according to the bidirectional reflectance distribution function (BRDF) of that hit surface. The distribution of scattering (according to the BRDF) helps approximate average radiance in that area. A way to compute that radiance is to average the distributions of scattered light from all samples inside a grid cell in the form of SH coefficients.

¹<https://www.cryengine.com/>

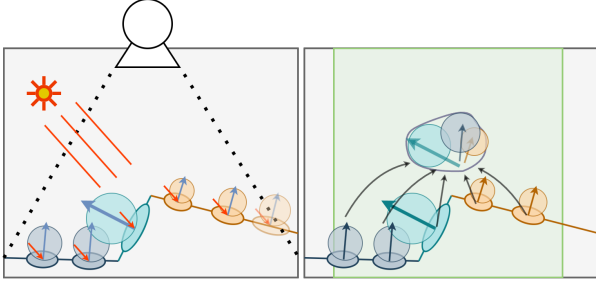


Figure 1: An illustration of VPLs being generated and their radiance accumulated inside of a cell. A camera is looking at the scene from the top (top image). Pixels are modeled as small surfaces (ellipses) with normals (blue arrows). Size of the pixel area depends on the viewing angle and the distance from the camera. All surface materials are Lambertian BRDFs so SHs approximate cosine weighted hemispheres (circles above ellipses). The direction of light rays also influence the orientation of calculated radiance. SH coefficients are then summed up inside the center of the cell (green box) to get the accumulated radiance (right image).

The second step requires propagating radiance from each grid cell to their 6 neighboring cells. The amount of radiance transferred between neighboring cells depends on the propagation direction. Since a regular grid has cube cells, radiance propagates inside of a cone, oriented from the center of the cell towards each neighbor through the face of the cell. To filter out all other radiance, we can calculate SH coefficients of a normalized, 90-degree cone, oriented around $\vec{n} = (x, y, z)$ as $(\frac{1}{2\sqrt{\pi}}, \frac{3}{4\sqrt{3\pi}}x, \frac{3}{4\sqrt{3\pi}}y, \frac{3}{4\sqrt{3\pi}}z)$, where y is the vertical axis, and multiply them with the propagated radiance’s SH coefficients. The result is then added to the neighboring cell’s radiance into a separate grid for storing results. Repeating this step a few times causes the light to smoothly propagate throughout the scene as shown in Figure 2.

Since diffuse lighting has primarily low frequency components, we can afford to leave out high frequency ones by encoding radiance (each RGB channel separately) using only 4 (L1) or 9 (L2) SH coefficients. To simplify the approach, we also assume that all surface materials are Lambertian BRDFs, which means specular lighting will not be propagated (it would also require using around SH 25 coefficients [3] for good results). L1 SH coefficients for a normalized cosine lobe, oriented around $\vec{n} = (x, y, z)$ were calculated as $(\frac{\sqrt{\pi}}{2}, \frac{\sqrt{3\pi}}{3}x, \frac{\sqrt{3\pi}}{3}y, \frac{\sqrt{3\pi}}{3}z)$.

3.1 Screen-space LPV

screen-space light propagation volume (SSLPV) differs from the main algorithm in how we sample the scene. Since we have to make use of screen-space information only (which includes accessing the G-Buffer for depth, normal, and color information) we use the already available pixels as our samples.

The final color buffer is used to extract pixel color information, represented as floating-point values to encode a wide range of intensities (from unlit black to emissive materials). The buffer is down-sampled 2 or 4 times for better performance at a slight cost in visual quality. Each pixel acts as a VPL and is weighted by its approximate

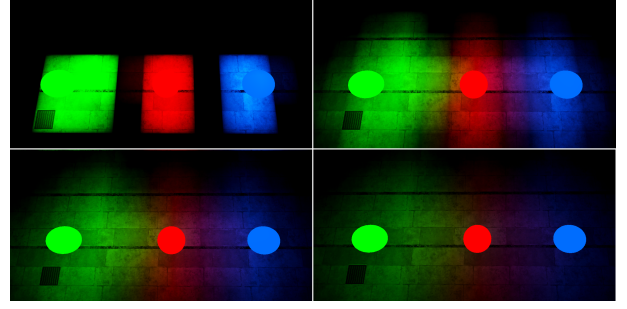


Figure 2: Effects of different propagation iterations at 0, 2, 4, and 6 (from top left to bottom right). Light diffuses more at higher iterations but also starts to lose intensity since the initial intensity gets spread out.

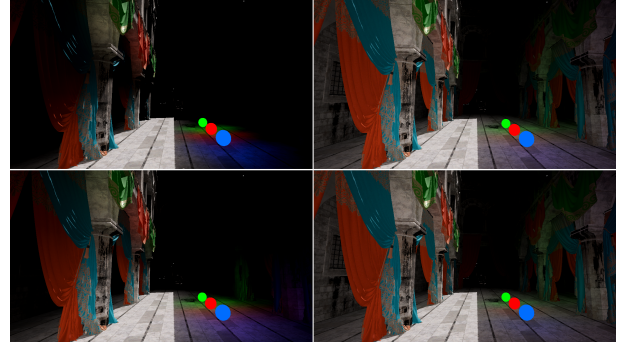


Figure 3: Comparison between 1, 2, 3 cascades with cascade scaling 2 (first row and bottom left), and 2 cascades with cascade scaling 4 (bottom right).

surface area $-\frac{D\|\vec{d}\|^2}{\vec{n}\cdot\vec{d}C^2}$ where $\vec{n}$ is the pixel normal vector, $\vec{d}$ vector from the camera to the pixel’s surface, D buffer down-sampling factor, and C is current cascade grid scale (more about that in 3.2). An illustration of VPLs and their contributions in the final radiance sum is shown in Figure 1.

SH coefficients for the radiance above the pixel’s surface are then calculated and stored as L1 SH coefficients, which are added to the total sum of the corresponding LPV grid cell (for each RGB channel separately). Since multiple pixels can occupy the same grid cell, addition had to be done atomically using an HLSL operation `InterlockedAdd`, which future versions of our method could try to avoid to improve performance.

3.2 Cascades

Finer grids allow capturing detailed lighting, but require many propagation iterations and do not propagate far enough, while on a coarser grid the propagation step can cover longer distances, but with less detailed lighting. We decided to combine grids at different scale factors C (number of grid cells is constant) to cover both weaknesses, but at a higher computational cost. Each cascade’s coefficients are sampled independently and then summed into the final radiance. Because increasing the number of cascades lowers performance and the lighting starts to leak more at larger scales, the maximum number of cascades was set to 4. Some results of using cascades are shown in Figure 3.

Larger cascades cover larger areas with lighting that does not change very often and is less detailed, so we can afford to update them less frequently. In our case, each

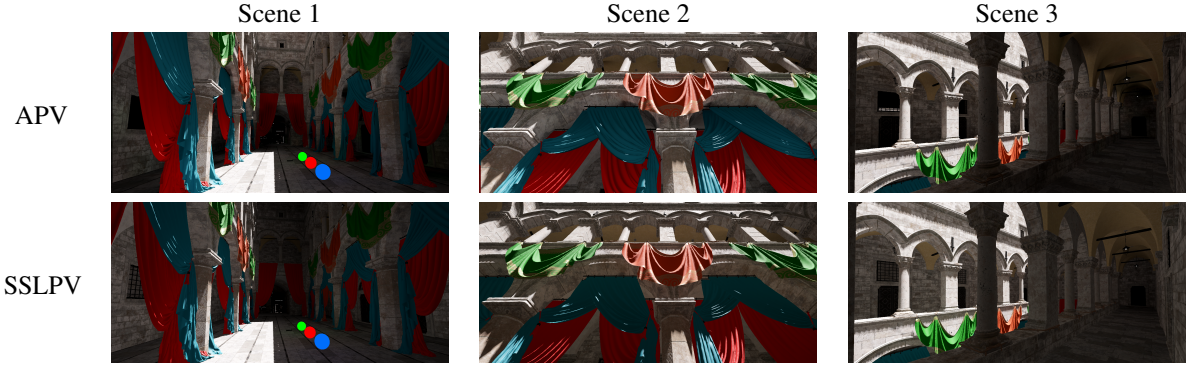


Figure 4: A visual comparison of adaptive probe volume (APV) and SSLPV results on 3 scenes of the Sponza with curtains benchmarking scene at 1720x820 screen resolution.

next, larger cascade was set up to update at half the frequency of the previous (the first every frame, second every second frame, third every four frames, and so on). This optimization reduces the number of updates performed each frame, but at a cost of global lighting being updated slightly slower on sudden lighting changes.

3.3 Rendering

A custom fragment shader first calculates the position of the pixel in world space, and then uses that position to sample from all cascades to obtain the final radiance. Each grid is sampled using trilinear interpolation, which amounts to sampling 8 grid cells. To then compute the final lit color of the pixel, the negative surface normal is extracted from the normal buffer and used to orient a cosine weighted hemisphere. Integrating the sampled radiance over all hemisphere directions is equivalent to multiplying radiance’s SH coefficients with the ones of the cosine weighted hemisphere.

3.4 Temporal smoothing

The amount of pixels each LPV grid cell receives can change rapidly when the camera moves, which can lead to flickering artifacts from a constantly changing gathered radiance, even if the lighting in a scene is static. To mitigate this effect, the lighting gathered in LPV grid cells is updated according to $L_t = \alpha L_{new} + (1 - \alpha)L_{t-1}$, where L_{new} is newly calculated radiance, L_t is radiance at frame t , and α set to 0.4. This reduces most of the flickering issues.

3.5 LPV grid movement

For simplicity, the LPV grid is centered around the camera’s origin. However, simply attaching the LPV grid to the camera causes the temporally smoothed lighting to incorrectly update since the temporal smoothing step assumes the previous lighting of a cell was calculated at the same position. Grid edges also become more apparent despite using interpolation techniques when sampling from it.

To prevent old radiance from shifting to different grid cells as the camera moves, the origin of each LPV grid cascade is snapped to the world grid. Specifically, the origin coordinates are rounded to the nearest multiple of the grid cell size. As a result, the grid can only occupy a discrete set of positions rather than moving continuously

Table 1: Image quality metrics for the evaluated scenes in Figure 4. SSIM is highest on Scene 2 probably because of an inherently more detailed structure, however LPIPS being lowest on Scene 1 suggests methods match there perceptually better than on Scene 2.

Metric	Scene 1	Scene 2	Scene 3
PSNR	22.30	21.85	24.60
SSIM	0.817	0.816	0.845
LPIPS	0.064	0.080	0.052

through space, which helps keep lighting stable between camera movements. Whenever a grid snaps to a new location, all stored radiance in its cells has to shift in the opposite direction of the location change. If an edge cell tries to shift its radiance outside the grid, shifting is ignored, and radiance in the new cells (covering new parts of the scene) are initialized to zero.

4 Results

4.1 Comparing with Adaptive Probe Volumes (APVs)

Ideally we would like this method to replicate realistic lighting as closely as possible, which can be simulated using PT. However, since it is too demanding for real-time use, we picked Unity’s APVs that can produce PT-like results, is real-time, and was the closest method to our SSLPV that we were able to find and run. Unfortunately, APV requires static lighting to be baked in advance into light probes, which means it cannot react to lighting changes.

Comparisons in Figure 4 expose several limitations of this method. Most notably in Scene 1, the first and second floor of the atrium do not receive any sky light in SSLPV since none of it is visible on the screen for the method to pick up. Colored light bleeding from the curtains in Scene 2 is appropriately subtle just like in the APV image. The lighting overall is also less detailed so it more easily leaks through or around obstacles as seen in Scene 3, where more light reaches areas behind the pillars which are supposed to receive less light according to the APV version. Due to time constraints, we weren’t able to add any temporal comparisons.

When comparing both methods using different image quality metrics (values in Table 1), we expected to see lower peak signal-to-noise ratio (PSNR) due to very dif-

Table 2: Performance measurements for different SSLPV configurations. Measurements were performed on a system with GTX 1660Ti (6GB VRAM), Ryzen 9 9900X (12 cores) @4.2Ghz, 32GB DDR5 RAM, at a screen resolution 2560x1440. Default values for each variable are 1 cascade, 6 propagation iterations, grid size $48 \times 48 \times 48$, and downsampling 4-times with performance of 23.2 ms.

Variable	Value	Time [ms]
# Cascades (with frame skip)	1	23.2 (23.2)
	2	33.3 (29.4)
	3	43.4 (35.0)
# Prop. Iter.	1	20.7
	2	21.0
	6	23.2
Downsample	1	54.4
	2	29.1
	4	23.2
	8	21.8
Grid size	$16 \times 16 \times 16$	15.4
	$32 \times 32 \times 32$	18.3
	$48 \times 48 \times 48$	23.2
	$64 \times 64 \times 64$	29.9

ferent resource budgets for each of the methods. SSLPV produces less detail in certain shadow areas, which makes structural similarity (SSIM) slightly lower. Both methods produce similar softer lighting, hence a low learned perceptual image patch similarity (LPIPS).

5 Performance

Each cascade independently adds a fixed amount of computation time since they run their radiance aggregation, propagation, and sampling steps independently. Larger grids can spend more time at the aggregation step due to containing more pixels and running more atomic operations (`InterlockedAdd`) when summing SH coefficients. The aggregation step also depends linearly on the number of pixels on the screen, so more aggressive downsampling of all buffers lowers the number of pixels quadratically. Trilinear sampling also takes some performance, where all 8 cells’ SH coefficients have to be linearly interpolated multiple times.

From the measurements, each cascade takes an additional 10.1 ms or around 6.4 ms with frame skipping, so optimizing the use of multiple cascades is important. Each propagation iteration takes 0.42 ms, and downsampling at 4 seems to be the sweet spot between the amount of flickering artifacts and performance. Downsampling set to 1 seems to run at a worse performance than expected, but the bottleneck seems to be the `InterlockedAdd` operation, which, if not used, improves performance from 54.4 ms to 21.7 ms at downsampling 1. Grid size increases the number of cells cubically which the performance measurements reflect.

6 Discussion and Conclusion

The major limitations with the implementation of SSLPV are:

- All radiance is currently summed into each cell using atomic `InterlockedAdd` operations, which is the biggest cause for bad performance at higher resolutions. This could be improved by implementing a version of a parallel prefix-sum algorithm for better GPU utilization.
- Cascades are calculated independently, so increasing the number of cascades linearly increases the total processing time. A possible optimization could be to reuse finer cascades and inject them into the larger cascades since all of them use the same set of VPLs.
- Cascades should not be centered around the camera but moved further in the viewing direction to cover less of the invisible parts of the scene behind the camera.
- A big limitation in this method is the lack of light direction information. This means we cannot model specular BRDFs on mirror surfaces. Implementing this would require having access to each light source, calculate the radiance it produces on affected pixels and accumulate it in SH coefficients of each pixel.
- Finally, there is no occlusion testing neither during radiance aggregation nor during propagation. Having precomputed signed distance fields for each mesh in the scene could allow us to perform occlusion tests in some useful way.

The SSLPV method runs overall sufficiently well to be used with 1 or 2 cascades for smaller area effects in video games.

7 Acknowledgment

The research was conducted within the projects Cell visualization of unified microscopic data and procedurally generated sub-cellular structures (J2-50221), funded by the Slovenian Research and Innovation Agency (ARIS).

References

- [1] A. Kaplanyan. “Light propagation volumes in cryengine 3”. In: *ACM SIGGRAPH Courses 7.2* (2009).
- [2] C. Dachsbacher and M. Stamminger. “Reflective shadow maps”. In: *Proceedings of the 2005 Symposium on Interactive 3D Graphics and Games*. I3D ’05. 2005, pp. 203–231.
- [3] P.-P. Sloan et al. “Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments”. In: *ACM Transactions on Graphics* 21.3 (2002), pp. 527–536.
- [4] M. Billeter et al. “Real-time multiple scattering using light propagation volumes”. In: *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*. I3D ’12. New York, NY, USA, 2012, pp. 119–126.
- [5] M. Drašček et al. “Radiance probes for global illumination in volume rendering”. In: *Proceedings of ERK 2025*. Ljubljana, 2025, str. 345–348.
- [6] T. A. Franke. “Delta Light Propagation Volumes for mixed reality”. In: *2013 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*. 2013, pp. 125–132.
- [7] C. Lambru et al. “Area Lights Voxelization for Light Propagation Volumes”. In: *VISIGRAPP 2022 (1: GRAPP)*. SciTePress, 2022, pp. 212–219.
- [8] V. M. Aranha et al. “Screen-space VPL propagation for real-time indirect lighting”. In: *2020 33rd SIBGRAPI Conference on Graphics, Patterns and Images (SIBGRAPI)*. 2020, pp. 46–53.