IFT6095 - Final Project

Olivier Mercier

This project is based on the paper *Matrix Row-Column Sampling for the Many-Light Problem* [Hašan et al. 2007]. The paper shows a way of approximating the contributions of many lights (e.g. 100,000 lights) by using a sampling approach to cluster the lights.

This report follows the structure of the project, which contains two main parts. First, we describe the Virtual Point Light (VPL) solution we use to compute global illumination in the scenes. Then, we explain how those lights are processed with the method presented in the paper so that only a reduced set of scaled VPLs is used to render the final image. We then show results produced by our method by comparing them to the accumulation of all VPLs, and also present timing results. We conclude with future work and some thoughts on the difficulties encountered during the project.

1 Virtual Point Lights

We take as input the scene objects and the position of a few light sources. It is easy to compute the direct illumination from those light sources. But to compute the contributions coming from global illumination, we place secondary light sources in the scene, called VPLs. A VPL placed on a surface approximates the illumination coming from one of the light sources that is reflected in the scene from that surface. We assume here that all surfaces are diffuse, so the VPLs always have a constant BRDF and always light the scene in the same way. We also assume that the original light sources have a radiance distribution equivalent to any other VPL, so that we don't have to treat them as special cases.

For each light source, we distribute VPLs by doing a rendering of the scene from the point of view of the light. To account for every direction, we use a hemicube map around each VPL. This hemicube map stores the depth, normal and color of the scene. For each pixel of this hemicube map, we place a VPL in the direction of this pixel using the depth at this pixel, which effectively deprojects the pixel into the scene. Figure 1 shows a 2D version of this process. After the VPLs are distributed in the scene for each initial light source, there is no additional difficulty in projecting more VPL for each secondary VPL. This is a simple iterative process of throwing VPLs from the initial light sources (level 0) to obtain the level 1 VPLs, and then throwing VPLs from the level 1 VPLs to get the level 2 VPLs, and so on. The bottom row of figure 3 show the VPLs for a different maximum VPL level, i.e. 0, 1 and 2 maximum VPL levels. We use v_i to represent a VPL, and note v_i^o for the i-th VPL thrown from VPL o.

There are multiple factors to take into account to determine the color and intensity we assign to a VPL deposited in the scene. Since we sample the VPLs on the hemicube instead of sampling the directions on the spheres, we need to adjust the VPL power accordingly. For instance, the directions sampled by the corners of the hemicube will be more densely sampled, so the VPLs in those directions should have a smaller weights than the VPL thrown in the center of a hemicube face. By assuming the hemicube faces are at a distance 1 from the VPL v_o, and using simple trigonometry, we find the sampling density in the direction of a given VPLs v_i^o to be

$$\sqrt{1 + d_i^2}$$

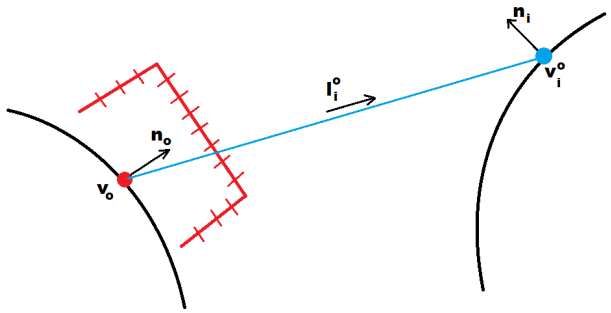


Figure 1: *Throwing secondary VPLs v_i^o from a given VPL v_o. The scene is rendered from the point of view of v_o and stored on a hemicube. A VPL v_i^o is thrown through each texel of the hemicube.*

where d_i is the distance between v_o and the pixel on the hemicube in the direction of which v_i^o is thrown. For a given VPL v_o, define

$$S_o = \sum_{i \in \mathcal{P}_o} \frac{1}{\sqrt{1 + d_i^2}}$$

where $\mathcal{P}_o$ is the set of all VPLs thrown from v_o. The weight assigned to VPL v_i^o is then

$$\frac{1}{S_o} \frac{1}{\sqrt{1 + d_i^2}}.$$

The deposited VPLs also have to be adjusted with respect to the normal of the surface they are deposited onto, as well as the color of this surface. Note that all this information is easy to obtain since we store it on the hemicube. Let n_i be the normal at the point where v_i^o is deposited, n_o the normal on the surface where v_o is located, l_i^o the normalized vector from v_o to v_i^o, and d_i^o the distance from v_o to v_i^o (See figure 1). The VPL radiance is weighted by

$$\frac{\lfloor \cos(n_0, l_i^o) \rfloor \cdot \lfloor \cos(n_i, -l_i^o) \rfloor}{(d_i^o)^2}$$

to take into account the relative angles of the surfaces. To summarize, the radiance of v_i^o is

$$(\text{radiance of } v_o) \cdot (\text{color of receiving surface}) \cdot \frac{\lfloor \cos(n_0, l_i^o) \rfloor \cdot \lfloor \cos(n_i, -l_i^o) \rfloor}{(d_i^o)^2}.$$

The final image is computed as the sum of the direct illumination from all VPLs. This is easily done by rendering every image lit by only one VPL to a texture, and accumulate the results for all VPLs. Note that a high enough precision must be used for the textures, since the individual contributions of some VPLs is very small. In this project, we use 32 bits per color channel. Figures 2 and 3 show global illumination results from accumulating the contributions of many VPLs.

2 Matrix Row-Column Sampling

The goal of sampling the VPLs is to get a reduced set of VPLs so that the accumulation of all light contributions is faster to compute.

The approach used in the paper [Hašan et al. 2007] is to select a few surface elements in the scene, look at the contribution of each light with respect to only those selected elements, and then select a reduced set of VPLs based on the obtained information. We split the details of our implementation here in two main steps : sampling and clustering.

2.1 Sampling

To sample the image, we render the scene with all VPLs on a very small texture (16x12 pixels instead of 1024x768 pixels for the full image). This is different than sampling pixels in the image, as it instead averages the contribution of lights over multiple pixels. This averaging approach has some advantages. First, it is trivial to implement, as the only thing to do is to reduce the rendered texture sizes. In the paper, they select surface elements with ray tracing, and compute the VPL contributions with a hemicube shadow map over each surface sample. In our case, we reuse the hemicube shadow maps centered at the VPL positions that were used to throw VPLs in the scene, so no additional code is required.

A second advantage is that if we randomly pick surface samples, we might pick very bad samples. Consider a scene like the one in figure 5 with the camera view as in subfigure e). We are interested in rendering the global illumination behind the holed wall, but if the surface samples we pick happen to all fall on the holed wall, the contributions behind it will be ignored. By averaging the contributions over multiple pixels, we make sure to take all parts of the scene into account.

Using this averaging approach, the definitions of the *reduced estimators* derived in the paper (section 3.4) cannot a priori be used directly for our implementation. However, the reduced estimator they propose is only an unbiased estimators for their reduced matrix R, which is only an approximation of the full matrix A. Therefore, using their estimator with our averaged samples, we will still get an unbiased estimator for some averaged matrix, say Q, even if this matrix is not the same as their matrix R. Whether their matrix R or our matrix Q is better approximation of A is unclear, since in both cases, increasing the number of samples will make Q and R converge to A. This justifies using the estimator presented in the paper for our situation.

2.2 Clustering

Once the contribution of each VPL is estimated from the averaged surface samples, a reduced VPL set is selected to represent all VPLs. We use the same idea as in the paper, i.e., clustering the VPLs and then selecting a representative for each cluster. As in the paper (section 3.5), we split the clustering into two phases. The first one (phase 1) uses clustering by sampling, and the second (phase 2) uses clustering by splitting. As in the paper, a portion of clusters are computed with phase 1, and the remaining clusters are computed with phase 2.

Our phase 2 implementation follows the paper exactly. However, our phase 1 implementation uses a slightly more aggressive approach. As in the paper, we select cluster centers, and then regroup VPLs together by assigning them to their closest cluster center. But instead of selecting centers at random and risking to get a very bad clustering, we proceed as follows. We begin by picking the first cluster center at random, and for each VPL we compute its distance[1] to the cluster center. We then pick the second center to be the one that is farthest from the first center, and for each VPL we compute the sum of the distances to both centers. The third center is the one whose total distance from the previous centers is the greatest, and so on. We repeat the process until a given number of cluster centers is reached.

Since this approach is slower than what is presented in the paper, we also implemented a simplified phase 1, where the cluster centers are picked uniformly at random. This doesn't yield very good cluster centers, but it allows to compare the time taken by our method with something that is closer to what is used in the paper.

3 Results

We present here results for two different scenes, along with timing results.

3.1 Cornell Box

We test our implementation on a simple Cornell box scene. Figure 2 shows the global illumination computed with an increasing number of VPLs. This uses all VPLs and doesn't use the sampling algorithm. The results confirm that our VPL implementation is satisfying, as increasing the number of VPLs increases the quality of the illumination, and the images converge. Note that using 85 VPLs (subfigure a)) creates some bright and dark spots in the image, while using more VPLs yields smoother results. Note also that the illumination from VPLs has to be clamped to avoid problems caused by the division by the squared distance when a VPL falls close to a corner. Still, some artifacts remain visible when using few VPLs, because those VPLs are scaled by a large value in order to represent the illumination of their whole cluster. When using more VPLs, the individual contribution of each VPL is smaller, and those artifacts are less noticeable.

Figure 3 shows the effect of increasing the maximum level of VPLs (i.e. the number of light bounces). Those results use few VPLs so we can easily see the VPL distribution after 2 bounces, so the quality of the results is not excellent.

Figure 4 compares our sampling method to the accumulation of all VPL contributions. We see that the sampling does a good job of selecting meaningful VPLs. For instance, it selects more VPLs on the colored walls than on the white wall, as the colored walls are closer to the sphere and the monkey head, and therefore contribute more in the illumination of the scene. Also, the results from using 150 sampled VPLs or all 1327 VPLs are similar.

3.2 Holed Wall

We tested our implementation on a scene where part of a room is only lit by indirect illumination. The scene can be seen in figure 5a). Figure 5b) shows the position of the initial light[2] and the VPLs cast in the scene. Note that we use a very strong light so that indirect illumination effects are more clearly visible.

Subfigures c) and d) show the scene from the same camera view. Subfigures c) shows all VPLs, while subfigure d) shows the VPLs

[1] By *distance* we mean the distance used in the paper.

[2] The initial light can be seen at the bottom of the figure. It is the only one not on a surface.

selected by the sampling algorithm. Since the left part of the room is not visible from this point of view, the VPLs falling on the red wall are not judged important. Only a few of them are selected near the bottom of the wall, where their contribution to the visible part of the floor is at its highest.

Subfigures e) to h) show another point of camera view. This time, the left part of the room is visible, so the distribution of VPLs selected by the sampling is different. Subfigure e) shows all VPLs and subfigure f) shows the VPLs selected by our sampling. We see that more VPLs are selected on the red wall, since they greatly affect the part of the scene seen through the holed wall. Subfigures g) and h) use the VPLs shown in subfigures e) and f) respectively, but do not display the VPLs. We see that both results are similar.

3.3 Computational time

We compare the time taken by our algorithm for different settings. Table 1 contains all timing results. It indicates the total number of VPLs thrown in the scene (*total nb VPLs*). It also indicates whether we use all VPLs (*render all VPLs*) or only the sampled VPLs (*sample VPLs*). If sampling is used, we indicate the number of sample VPLs used (*nb sample VPLs*) and the number of clusters computed by the phase 1 of the clustering algorithm (*nb phase 1*). We also indicate whether or not we use the simplified version of phase 1 clustering (*simplified phase 1*). The time appears in the last column, in milliseconds. The Cornell box scene is used for all tests.

When 331 total VPLs are used, we see that the sampling approach can reduce the computational time in some cases. However, when using a total of 1327 VPLs, the sampling increases computation times. The problem with our implementation is that all lighting evaluations are done on GPU, while the clustering is done on CPU. The clustering is therefore very slow to compute, which explains the observed performance reduction.

One constant observation is that computing more clusters with phase 1 reduces computational time, suggesting that phase 1 is slower than phase 2. This observation is reinforced by observing that using the simplified version of phase 1 significantly reduces computational time in some cases. This is due to the fact that regrouping VPLs according to their closest cluster center is a costly operation. This issue is noted in the paper, where some accelerations are suggested (section 4), but we did not implement them.

Unfortunately, a GPU implementation of the sampling and clustering would be necessary to significantly test the performance of the sampling approach.

4 Conclusion

We have shown our implementation based on *Matrix Row-Column Sampling for the Many-Light Problem* [Hašan et al. 2007] gives satisfying results. Even if we cannot conclude that the sampling reduces computational time, we did show that the method selects a meaningful VPL subset, and that the results rendered from this subset are comparable the results using all VPLs.

Our implementation could be improved in many ways. First, even if using a hemicube rendering to distribute VPLs in the scene is fast, it gives a VPL distribution that is too regular. A random sampling would give a more organic distribution and would avoid grid-related artefacts, which caused problems that had to be controled in some of our tests (e.g. a line of VPLs falling near a corner, creating a clear line of artefacts). Also, using a more intelligent sampling for the VPLs would greatly increase the efficiency of the method. The hemicube approach throws VPLs in every direction, even though the directions nearly orthogonal to the normal of v_o generate secondary VPLs v_i^o with very small contribution to the scene.

Some inherent problems of VPLs were encountered during the project, mostly artefacts for VPLs falling close to a corner that have a very high contribution because of the $frac1d^2$ term. Those VPLs needed to be clamped, and the clamping factor had to be changed for different scenes. Using a more robust VPL model would directly benefit the implementation and results.

As mentioned in the paper, shadow maps are not ideal for computing visibility between lights and surface elements. First, the finite resolution of shadow maps is sometimes visible in the illumination. Also, finding an offset bias to control shadow acne that works for all VPLs in the scene in not easy. This is a parameter that had to be tightly controlled in our tests.

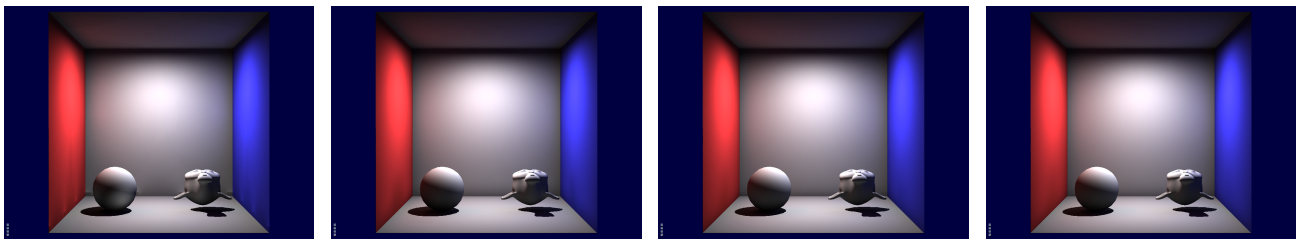
Overall, the fundamental ideas behind the method are easy to understand and implement. However, some optimization is required in order to observe the performances described in the paper. Still, an unoptimized implementation could be used to precompute a VPL subset that can then be used in a dynamic scene, as long as the scene is not too different from the one used for precomputations.

References

HAŠAN, M., PELLACINI, F., AND BALA, K. 2007. Matrix row-column sampling for the many-light problem. In *ACM Transactions on Graphics (TOG)*, vol. 26, ACM, 26.

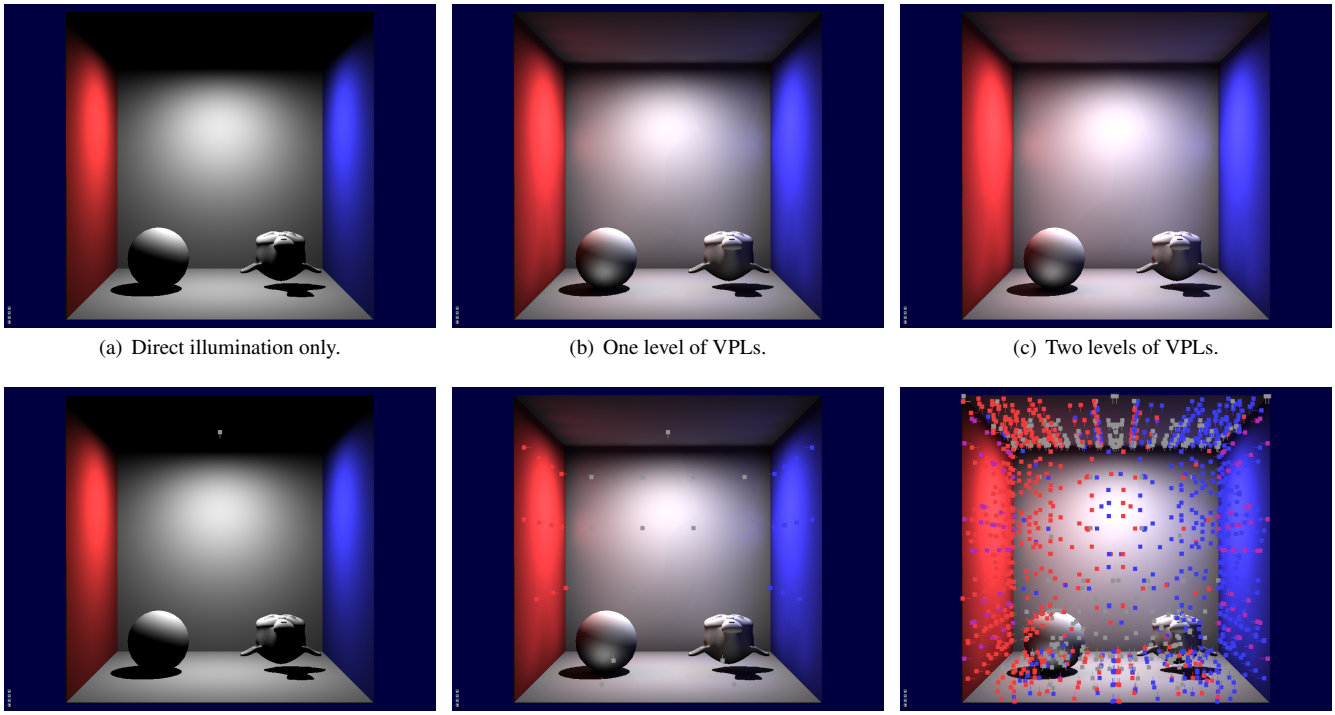
total nb VPLs	render all VPLs	sample VPLs	nb sample VPLs	nb phase 1	simplified phase 1	time (ms)
331	x					0.29
331		x	50	15		0.24
331		x	50	45		0.22
331		x	50	45	x	0.21
331		x	150	35		0.32
331		x	150	135		0.28
331		x	150	135	x	0.28
1327	x					1.17
1327		x	50	15		5.84
1327		x	50	45		4.82
1327		x	50	45	x	2.67
1327		x	150	35		5.32
1327		x	150	135		3.68
1327		x	150	135	x	2.17
1327		x	600	140		3.93
1327		x	600	540		1.46
1327		x	600	540	x	1.50

Table 1: *Computational times for different settings of the method.*



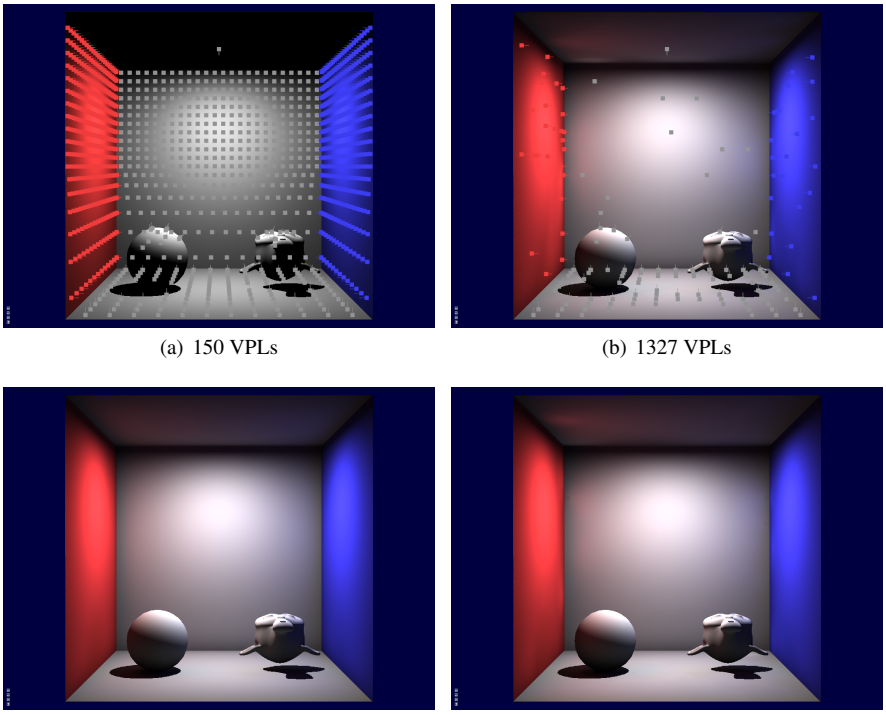
(a) 85 VPLs (b) 331 VPLs (c) 751 VPLs (d) 1327 VPLs

Figure 2: *Global illumination computed from multiple VPLs. The initial light source is located on the ceiling of the box, and VPLs are thrown from it. No level 2 VPL is thrown.*



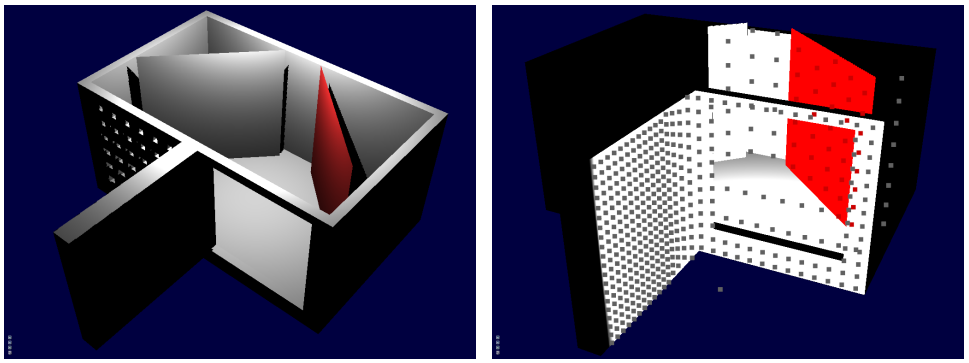
(a) Direct illumination only. (b) One level of VPLs. (c) Two levels of VPLs.

Figure 3: *Global illumination from different levels of VPLs. The top and bottom figures are the same, except the bottom ones include the VPL positions. Note that the quality of illumination is not very high here since few VPLs are used, in order to easily distinguish the different VPLs in figure c). Note that VPLs are displayed even if they are occluded.*



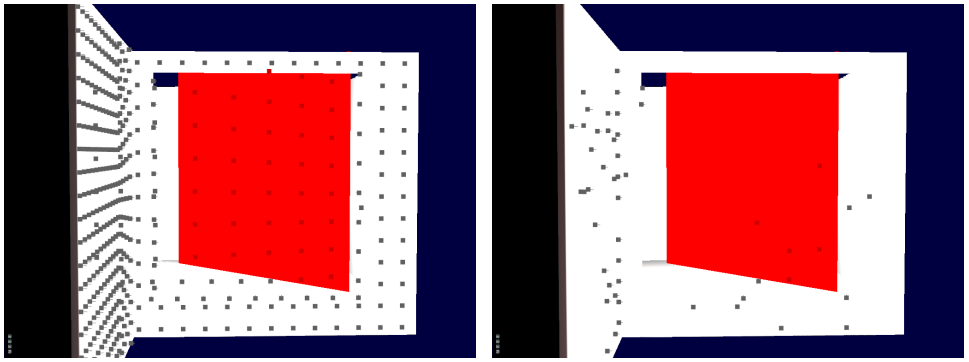
(a) 150 VPLs (b) 1327 VPLs

Figure 4: *Comparison between the sampling method and the accumulation of all VPLs. The top and bottom rows are the same, but the top row shows the position of the VPLs. Note that VPLs are displayed even if they are occluded.*



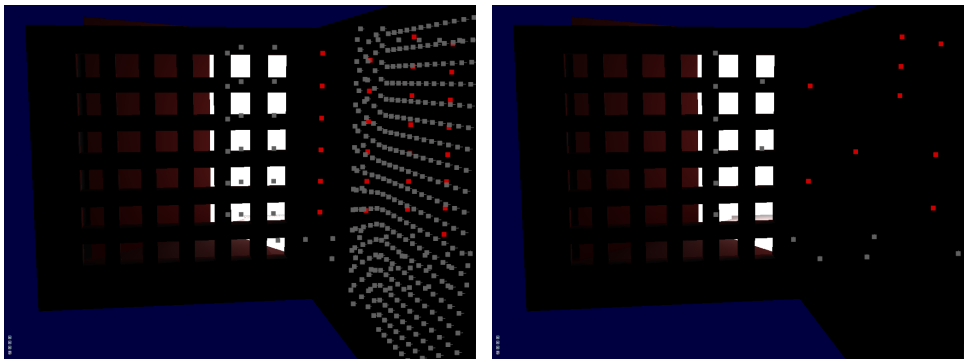
(a) Overview of the scene.

(b) All VPLs of the scene lit by direct illumination.



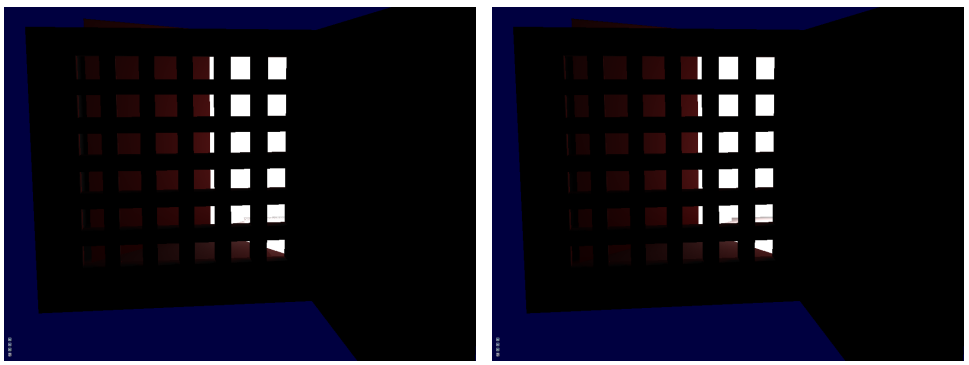
(c) All VPLs.

(d) Sampled VPLs. Notice how few VPLs appear on the red wall, which has a small impact on the illumination of the exterior of the room.



(e) All VPls.

(f) Sampled VPLS. More VPLs are sampled on the red wall as it is an important contribution to the indirect illumination for the visible part of the scene.



(g) Same as e) but VPLs are not shown.

(h) Same as f) but VPLs are not shown.

Figure 5: *Results for the holed wall scene. The left part of the room is only lit by indirect illumination coming mainly from the large red plane. The left column always uses all VPLs, while the right column uses the sampled VPLs. Note that VPLs are displayed even if they are occluded.*