Eurographics Symposium on Rendering (2007)
Jan Kautz and Sumanta Pattanaik (Editors)

Incremental Instant Radiosity for Real-Time Indirect Illumination

Samuli Laine[1,3] Hannu Saransaari[3] Janne Kontkanen[2,3] Jaakko Lehtinen[3,4] Timo Aila[1,3]

[1]NVIDIA Research [2]PDI/DreamWorks
[3]Helsinki University of Technology [4]Remedy Entertainment

Abstract

We present a method for rendering single-bounce indirect illumination in real time on currently available graphics hardware. The method is based on the instant radiosity algorithm, where virtual point lights (VPLs) are generated by casting rays from the primary light source. Hardware shadow maps are then employed for determining the indirect illumination from the VPLs. Our main contribution is an algorithm for reusing the VPLs and incrementally maintaining their good distribution. As a result, only a few shadow maps need to be rendered per frame as long as the motion of the primary light source is reasonably smooth. This yields real-time frame rates even when hundreds of VPLs are used.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Bitmap and framebuffer operations I.3.7 [Computer Graphics]: Shadowing, Radiosity

1. Introduction

This paper describes a method for rendering single-bounce indirect illumination at real-time frame rates using current graphics hardware (Figure 1). Our method requires no precomputation and is able to handle dynamic lighting conditions, given that the motion of the light sources is reasonably smooth. We require that the geometry used for bouncing off indirect illumination remains static. Dynamic objects cannot thus generate indirect illumination effects such as color bleeding, but they can receive indirect illumination from other parts of the scene. Shadowing of direct illumination is of course still possible.

Our method is a variant of the instant radiosity algorithm [Kel97]. In instant radiosity, light paths are traced from the light source, and *virtual point lights* (VPLs) are generated at the vertices of these paths (Figure 2a). These VPLs are then used for illuminating the scene, in addition to the primary light source (Figure 2b). In our method, we employ hardware shadow mapping for determining the visibility between the VPLs and the visible surfaces of the scene. Interleaved sampling [KH01] is used for reducing the number of shadow map lookups without introducing upsampling artifacts.

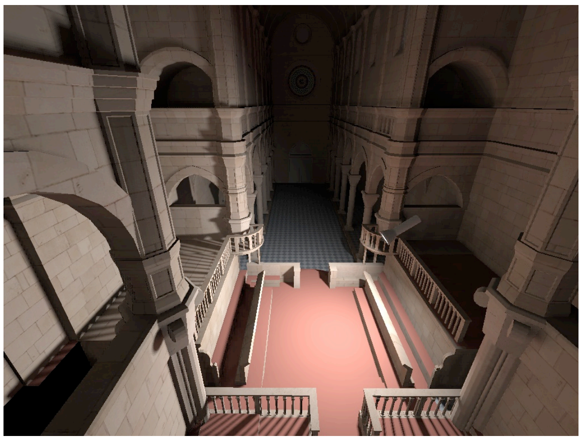

Figure 1: *An example of direct and single-bounce indirect illumination combined. Here, indirect illumination is computed using 256 virtual point lights, 4–8 of which are updated every frame. This scene with $\sim$80K triangles runs at 49 fps in* 1024×768 *resolution using a single off-the-shelf GPU. Computations related to indirect illumination consume approximately 71% of total rendering time.*

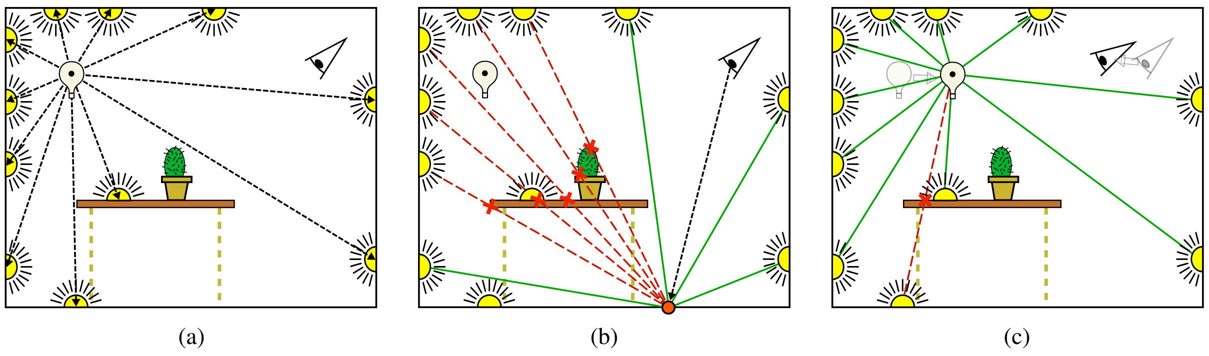

Figure 2: *Principle of instant radiosity and VPL reuse. (a) In instant radiosity, paths are traced from the primary light source, and virtual point lights (VPLs) are placed in each reflection/absorption point. Since we are limited to single-bounce indirect illumination, we do not continue the paths beyond the first hit. (b) When rendering a point seen by camera (orange dot on the floor), the visibility from each VPL is determined. Incident irradiance is accumulated from the visible VPLs. Direct illumination is computed separately (not illustrated here). (c) When the light source moves, the standard method is to recast rays and construct new VPLs. Instead of doing this, we examine which VPLs from previous frame are still valid, i.e. can be seen from the new light source position. In the case depicted, only one VPL below the table is invalid and cannot be reused. Potential camera movement does not affect this step.*

Limiting the light paths to a single bounce makes the method particularly straightforward to implement. Previously, Tabellion and Lamorlette [TL04] have demonstrated that a single bounce of indirect illumination is sufficient in many cases even in high-quality offline rendering. We feel that this limitation is not critical to interactive applications such as computer games, either.

Our key contribution is a method for reusing the VPLs so that their shadow maps need not be computed every frame (Figure 2c). By carefully recomputing the intensities of the reused VPLs, and making sure only valid VPLs get reused, no temporal lag is introduced. In practice, it is usually enough to compute less than ten shadow maps each frame in order to take the movement of the primary light source into account. Extreme motions of the light source may force us to discard more VPLs than we can afford to recompute, leading to temporary degradation of shadow quality.

The method for choosing which VPLs are reused is based on the *dispersion* measure [Nie92, LaV06] of a point set. Niederreiter characterizes dispersion as a metric for deviation from denseness, and it is closely related to the more common discrepancy metric. The dispersion of a point set is much easier to compute than its discrepancy, and generating new points so that dispersion is reduced is straightforward.

2. Previous Work

Interactive global illumination. The simplest way of interactively visualizing scenes including global illumination effects is to use precomputed static lighting solutions, e.g., radiosity [CW93]. Such techniques not only form a significant body of graphics literature, but have also found their way in practical applications in computer games, where the solutions are usually stored in "light maps". However, both the scene and the lighting have to remain static.

Ambient occlusion [ZIK98] is a heuristic approximation to global illumination now in common practical use [Lan02]. For a point being shaded, it is defined as the hemispherical integral of either the visibility function or some suitable function of the distance to the nearest surface in each direction. The model nicely reproduces soft dark corners, an important feature of "real" global illumination solutions. Kontkanen and Laine [KL05] and Malmer et al. [MMAH] describe techniques for rendering ambient occlusion to the surroundings of moving, rigid objects.

Dachsbacher and Stamminger [DS05, DS06] describe approximate image-space techniques for the interactive rendering of indirect illumination. Unlike our method, their techniques do not consider visibility, i.e., indirect shadows.

Non-heuristic methods for interactive or real-time rendering of indirect illumination fall roughly in two categories. Instant radiosity [Kel97] and its variants form one category. These methods construct (quasi-)random paths from the light sources and deposit *virtual point lights* (VPLs) at the vertices of the paths. When used together with a suitable shadow solver, usually shadow maps, these point lights represent the full indirect illumination in the scene. Instant radiosity techniques are GPU-friendly due to the fact that most of the computation is spent on per-pixel lighting. However, variants based on ray tracing on a cluster of PCs have also been described; see Wald's thesis [Wal04] for an overview. Several of these techniques employ interleaved sampling [KH01] for smooth reconstruction of indirect illumination [Wal04, SIMP06b]. Both Wald and coworkers and

Segovia et al. [SIMP06a] describe bidirectional algorithms for guiding the generation of VPLs according to view importance, i.e., their contribution to the final image. Temporal coherence in non-static indirect illumination, or rather the lack of it, remains a major problem for all these algorithms.

Precomputed light transport techniques form the second category of methods for interactive rendering of physically-based indirect illumination. These methods precompute transport operators that are used at runtime for computing a global illumination solution that corresponds to the current direct illumination conditions in a static scene. However, most of these techniques work with distant illumination only (e.g. [SKS02]). Kristensen et al. [KAMW05] describe a technique for rendering indirect illumination from omnidirectional, local, moving light sources. The results are impressive, but dynamic objects are not supported, and preprocessing costs are substantial. Kontkanen et al. [KTHS06] extend wavelet radiosity for computing a full hierarchical direct-to-indirect transport operator for a static scene. The technique supports all types of light sources, and, in principle, glossy BRDFs. However, dynamic objects cannot be easily supported. Furthermore, precomputation still takes tens of minutes.

Quality of sample distributions. In the context of integration, the quality of sample distributions is commonly measured using discrepancy, which is the ability of a sampling pattern to estimate areas of subregions inside a domain. If a subregion covers N% of the domain, ideally the subregion would also cover N% of the sampling points. Discrepancy is the the maximum difference between the two for any subregion. Numerous variants of discrepancy exist, and we refer an interested reader to the book by Matousek [Mat99]. Often the set of subregions is limited to axis-aligned rectangles; unfortunately the resulting measure is not rotation-invariant, which would be a problem in our application. Discrepancy is also fairly expensive to compute and does not directly indicate which sample is the most redundant or where exactly should a new sample be inserted during incremental updates.

Niederreiter [Nie92] presents dispersion as another metric for measuring the quality of point sets. Typically dispersion is computed as the largest empty circle in the domain, and it is thus rotation-invariant. As will be discussed in Section 4, dispersion is fast to compute [SH75, AK00] and directly indicates where new samples should be inserted during incremental updates. Discrepancy and dispersion are related so that low discrepancy implies low dispersion, while the converse is not true [LaV06]. For example, the Sukharev grids (regular grids) have optimal dispersion but bad discrepancy. In our application, however, the incremental updates do not seem to result in any visible regularity. Incremental dispersion reduction has been previously used at least in the context rapidly exploring random trees for motion planning [LL04]. While not intimately related to computer graphics, their problem has some similarities with our setting: inserting new samples is relatively expensive and existing samples cannot be moved. Obnoxious facility location [Cap99] is another field of study in which a point maximally far from everything else is desired.

Agarwal et al. [ARBJ03] describe an algorithm for importance sampling of environment maps. The Hochbaum-Shmoys algorithm they use for adding sampling points results in a similar minimization of dispersion as our method. However, they operate on a discrete set of possible locations for sampling points, in contrast to our continuous domain.

On conceptual level the incremental dispersion reduction resembles sequential Monte Carlo methods, e.g. [DDJ06, DFG01], that maintain and improve sampling distributions over time. Most such methods move the samples between frames, whereas in our context it is crucial to keep most of the samples stationary between frames.

3. Algorithm Outline

For each frame, our method performs the following steps for computing the indirect illumination:

1. Determine the validity of each VPL (Sec. 4.4).
2. Remove all invalid VPLs and possibly a number of valid ones to improvement the distribution (Sec. 4.5).
3. Create new VPLs according to allotted budget. Render paraboloid shadow maps for them (Sec. 4.6).
4. Compute intensities for VPLs (Sec. 4.7).
5. Render the positions, normals, and colors as seen from camera into a G-buffer.
6. Split the G-buffer into a number of tiles.
7. Loop over tiles and accumulate illumination from a subset of VPLs in each of them.
8. Combine the tiles back into a single image.
9. Smooth the accumulated illumination using a spatially-varying filter kernel.

After the indirect illumination is computed, it can be combined with direct illumination and precomputed illumination from stationary light sources (light maps). In practice, it is advisable to calculate direct illumination in parallel with CPU-intensive VPL management tasks (Steps 1–4). Also, G-buffer rendering and splitting (Steps 5 and 6) can be performed in parallel with VPL management.

VPL management (Steps 1–4), which is our main contribution, is discussed in detail in Section 4. Accumulating incident illumination and performing the spatially-varying illumination filtering (Steps 5–9) are presented in Section 5.

4. VPL Management

The directional distribution of the VPLs as seen from the point light source should ideally follow the directional intensity distribution of the light source. When this is the case, each VPL represents a similar fraction of the total light source power. The goal of *VPL management* is to keep the distribution of VPLs as close to the ideal as possible by removing and adding new VPLs within an allotted budget. Since we keep most of the VPLs fixed between frames, the movement of the light source skews the distribution away from the ideal.

This section describes our method for reusing VPLs and incrementally maintaining their good distribution. First, we take a look at the domains in which the calculations related to VPL positioning are performed (Section 4.1), and explain how rendering quality is controlled (Section 4.2). We then describe the metric we use for deciding which VPLs we delete and where new VPLs are located (Section 4.3). After this, we examine step by step our method for removing old VPLs and generating new ones (Sections 4.4–4.7).

4.1. Domains of Distributions

Because we only consider point-like light sources, the directional distribution of VPLs as seen from the light source can be mapped into a 2D domain. We can then perform all computations related to the VPL distribution in this 2D domain, instead of using the actual 3D positions of the VPLs. Despite the potential confusion, we refer to the points obtained by mapping the VPL directions into the 2D domain as *sampling points*, even though we are not actually sampling any function.

Our implementation supports two types of primary light sources: 180° spot lights with cosine falloff, and omnidirectional point lights. With spot lights, the most suitable 2D domain is the unit disc that is obtained by flattening the z coordinate of the hemisphere pointing at the spot direction (Figure 3a). According to the so-called Nusselt analog [CW93], the differential areas on the disc then correspond exactly to cosine-weighted differential solid angles, meaning that uniform distribution on unit disc corresponds to the desired cosine-weighted distribution on hemisphere (Figure 3b). Note that spot light sources with different falloff angles can be easily mapped to the unit disc by a radial stretch. With omnidirectional primary light sources, we operate on the surface of the unit sphere (Figure 3c). These 2D domains are different in the sense that the unit disc has a boundary, while the unit sphere has not.

Most of the VPL management operations require the Delaunay triangulation and the associated Voronoi diagram computed for the entire set of sampling points. These are computed in the same domain where we carry out the rest of the geometric operations, i.e. unit disc or unit sphere. For unit sphere, we compute the Delaunay triangulation by constructing a 3D tetrahedralization of the sampling points on the sphere. The surface faces of this tetrahedralization are then conceptually lifted onto the surface of the sphere, yielding spherical triangles. In our implementation, we use CGAL computational geometry library [CGA06] for performing all non-trivial geometrical operations such as triangulations and tetrahedralizations.

4.2. Quality Control

The quality of the VPL distribution is controlled by application-defined budget for constructing new VPLs. Two values are specified: minimum and maximum number of new VPLs created each frame. We denote these values by *recalcMin* and *recalcMax*, respectively. By using a nonzero *recalcMin*, we ensure that the quality of the VPL distribution is improved even when all VPLs are valid. On the other hand, by specifying *recalcMax*, we can guarantee real-time frame rate even in more demanding situations, at the expense of temporal degradation of quality. The maximum number of VPLs is denoted by *maxLights*.

4.3. Dispersion Metric

The dispersion of a point set P in metric space (X,d) is defined by

$$\delta(P) = \sup_{x \in X} \min_{p \in P} d(x,p) \tag{1}$$

In our case, space X is either the unit disc or unit sphere, and d is the Euclidean distance. To be pedantic, when X is the unit sphere, d should measure the distance along the spherical surface. However, since we are not interested in the value of the dispersion but want to know the point $x \in X$ that gives it, we may equally well use Euclidean distance in $\mathbb{R}^3$.

Simply put, Equation 1 states that for unit disc, dispersion δ is the radius of the largest empty circle, i.e. containing no sampling points, whose center is located inside the disc.

4.4. Validity Testing

When the primary light source is a spot light, a VPL becomes invalid if either of the following conditions are met: 1) it is occluded from the new position of the primary light source, or 2) it falls outside the illumination region of the primary light source. With omnidirectional light sources, condition 2 cannot occur, and occlusion is the only reason that forces invalidating VPLs.

We check the occlusion by casting a shadow ray from the primary light source to VPLs using a conventional CPU ray tracer. The ray tracer only considers the static part of the scene, as occlusion from dynamic objects is not to be taken into account. It might also be possible to use the shadow map computed for the primary light source for determining the validity of VPLs, but we did not investigate this option. Determining which VPLs are outside the illumination region of a spot light source is trivially accomplished by calculating the dot products between primary light source direction and the vectors to the VPLs.

4.5. Deleting VPLs

It often happens that even after deleting every invalid VPL, we need to delete valid ones to make room for *recalcMin*

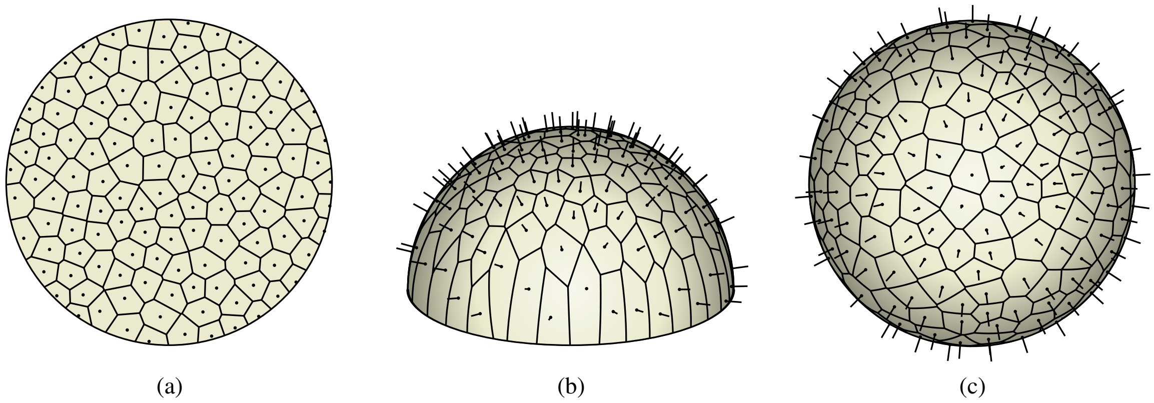
Figure 3: *Domains for sampling point distributions. In every frame, we first map the directions of VPLs into sampling points in a 2D domain, then perform VPL computations in this domain, and finally, if a new VPL are created, map the new sampling point back to directional distribution to cast the ray that determines the actual spatial location of the new VPL. (a) With spot lights, we perform sampling point computations on unit disc. The sampling points are obtained by constructing a basis where the z axis points to the spot direction, transforming the normalized direction vectors to VPLs into this basis, and dropping the z coordinate. (b) According to the Nusselt analog, the areas on the unit disc correspond to integrals of the emission function in spot light with 180° falloff. Therefore, uniform distribution on the unit disc yields the correct directional distribution of the VPLs. (c) With omnidirectional light sources, the sampling points are located on the surface of unit sphere, and the mapping is performed simply by normalizing the direction vectors to VPLs.*

new VPLs. In this case, we need to decide which VPLs to delete. We use a simple heuristic based on the Delaunay triangulation of the sampling points. Notice that the sampling points of the invalid VPLs have already been removed and the triangulation updated accordingly.

First, we find the shortest Delaunay edge in the triangulation. Second, we examine the neighbors of the two vertices connected to the edge, and compute the distances to the *second closest* neighbors of them. The vertex that has smaller distance to second-closest neighbor is then deleted. After deleting the sampling point, we update the triangulation. This operation is repeated until there are *recalcMin* free VPLs.

This deletion method always increases the smallest distance between two points, which represents the greatest local concentration of sampling points. We have found this simple heuristic to consistently perform well and predictably. It is certainly possible to develop more sophisticated heuristics by e.g. considering multiple removals at once, but we find it quite unlikely that significant improvements can be made.

4.6. Creating New VPLs

The most critical part of the reuse algorithm is the creation of new VPLs, i.e. sampling points in the 2D domain. Our approach is based on systematically reducing the dispersion of the set of sampling points.

Our strategy is to place the new sampling point at the center of the largest empty circle containing no sampling points. It is easy to see that this guarantees the reduction of dispersion except in the special case where there are more than one maximal empty circles with equal radii. Even then, the strategy guarantees that dispersion is reduced whenever possible.

Unlike discrepancy, dispersion is quite easy to calculate in Euclidean spaces. It has been shown [SH75, AK00] that for a planar set of points in a domain bounded by convex polygon, the largest empty circle is always centered at:

1) vertex of the Voronoi diagram, touching three points,
2) point where an infinite Voronoi edge intersects the boundary of the domain, touching two points, or
3) vertex of the bounding polygon, touching one point.

With the unit disc, we can regard its bounding circle as an infinitely tessellated polygon, which allows us to immediately see that cases 1 and 2 hold for the disc as well. However, case 3 may then occur at any point of the bounding circle. Fortunately the only way this can happen is when a Voronoi cell contains the entire line segment from its Delaunay point to the opposite side of the bounding circle through the origin. In this rare case the center of the largest empty circle is at the point where the line segment intersects the bounding circle.

Finding the largest empty circle and consequently the dispersion therefore requires no more than enumerating Voronoi vertices and edges, performing elementary intersection operations, and computing distances. In the spherical domain, the lack of boundary makes the computation of dispersion even simpler. From the previous list of cases, only case 1 is possible, and therefore the largest empty circle is always found at a Voronoi vertex.

After finding the location for the new sampling point, we add it to the distribution and update the Delaunay triangulation. Then, a ray is cast from the primary light source to the direction corresponding to the sampling point. If the ray exits the scene, no VPL is created. Otherwise, a new VPL is placed at the intersection point, and a paraboloid shadow map [HS98, BAS02, OBM06] is rendered for it. The color of the new light source is defined by the color of the surface at the hit point. With textured surfaces, we employ a heavily blurred version of the texture for determining the color, in order to avoid sporadic illumination effects due to small texture details.

This process is repeated until we have created *recalcMax* new sampling points, or when we hit *maxLights*, the given maximum number of VPLs.

4.7. Computing Intensities for VPLs

Because of the movement of the light source and limited number of sampling point repositionings, it is unlikely that the sampling points will be evenly spread across the domain. Therefore, assigning equal intensity to all VPLs would produce incorrect results. To remedy this, we need to recompute the intensity of each VPL per frame.

The intensity of a VPL should equal the emission function of the primary light source integrated over the solid angle it represents. Due to our choice of 2D domain for the sampling points, this is proportional to the area that the corresponding sampling point represents in the domain. Following Yakowitz et al. [YKS78], we assign the area of the Voronoi region around the corresponding sampling point as the intensity for the VPL. Computing the areas of the Voronoi regions can be done using elementary geometry both on unit disc and unit sphere. The intensities of the VPLs are finally scaled so that their sum equals the total power of the light source.

5. Illumination Accumulation and Filtering

When the VPLs have been updated, we may start rendering the final image as seen by the camera. All of these operations (Steps 5–9 of the algorithm outline) are conceptually identical previously presented methods for interleaved sampling and filtering [KH01, Wal04, SIMP06b], and we will therefore cover them quite briefly.

5.1. Indirect Light Accumulation

Rendering the G-buffer. We begin by rendering the world-space positions, normals, and colors of visible pixels into an off-screen frame buffer. Following the terminology of Segovia et al., we call this the *initial G-buffer*.

Splitting the G-buffer. To facilitate low-resolution accumulation of indirect illumination, we split the contents of the G-buffer into $n \times m$ tiles. The split is done so that each tile represents the entire image sub-sampled with a different offset. In other words, the contents of pixel (a,b) in the tile with index (i,j), where $i \in [0,\ldots,n-1]$ and $j \in [0,\ldots,m-1]$, are read from pixel $(an+i, bm+j)$ of the original image.

Segovia et al. observed that splitting should be done using multiple passes over the image to enhance cache coherency. However, we found in our tests that current hardware performs the split fastest in a single pass. The result of the split is stored into *split G-buffer*. Only positions and normals are needed in this buffer.

Accumulating incident illumination. We assign each VPL to one of the tiles in the split G-buffer so that each tile has approximately the same number of affecting lights. We then process each tile using a fragment shader that fetches the world-space position of the pixel, performs the shadow lookups from the paraboloid maps of the assigned VPLs, and computes the incident illumination based on position and normal of the pixel and positions and intensities of the light sources. Note that the color of the receiving surface is not taken into account at this point.

As noted by Keller [Kel97], the singularity in the $1/r^2$ falloff of VPLs may cause objectionable artifacts when receiving surface is very close to a VPL. To work around this problem, we clamp the maximum irradiance that a single VPL can contribute.

Combining tiles. After the incident illumination has been accumulated in the tiles, we combine the results back into a single full-sized illumination buffer, performing the exact opposite of the splitting step. Because each of the $n \times m$ tiles used a different set of VPLs, the resulting illumination buffer has a structured noise pattern that repeats in $n \times m$ pixel blocks.

5.2. Illumination Filtering

Our approach to filtering the incident illumination is a synthesis of previously used techniques. Segovia et al. [SIMP06b] compute a discontinuity buffer and performed a Gaussian blur constrained by the detected discontinuities. Wald [Wal04] also uses a discontinuity buffer, but performs the filtering using a box kernel whose size exactly matches the tiling of the noise pattern, i.e. $n \times m$ pixels. This kind of filter has the desirable property of being able to remove the structured noise completely from continuous surfaces (Figure 4).

In our method, we also use $n \times m$ wide box filter, but instead of computing a discontinuity buffer, we choose the support of the filter dynamically on a per-pixel basis. Unfortunately, it is impossible to use the physically motivated thresholding criterion of irradiance caching [WRC88], since

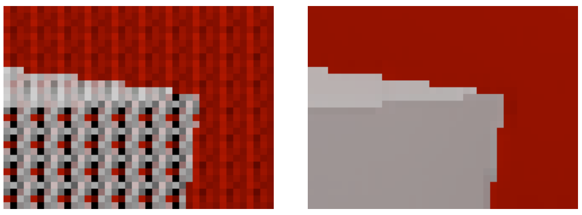

Figure 4: *After the G-buffer is merged, a structured noise pattern, repeating every $n \times m$ pixels, emerges due to the use of a different set of VPLs in each tile. (a) Noise pattern before filtering when 4×4 tiles were used. (b) The result after applying a spatially-varying box filter.*

```
ILLUMINATION-FILTER(int $px$, int $py$)
int $count = 0$
Vec3 $accum = \{0,0,0\}$
Vec3 $pos$ = TEX(texPositions, $px$, $py$)
Vec3 $normal$ = TEX(texNormals, $px$, $py$)
for $y$ in $[0,\dots,m]$ do
  for $x$ in $[0,\dots,n]$ do
    int $tx = px + x - \lfloor n/2 \rfloor$
    int $ty = py + y - \lfloor m/2 \rfloor$
    Vec3 $pos2$ = TEX(texPositions, $tx$, $ty$)
    Vec3 $normal2$ = TEX(texNormals, $tx$, $ty$)
    if $(|pos2 - pos| < \alpha)$ and $(normal2 \cdot normal > \beta)$ then
      $accum = accum +$ TEX(texIllumination, $tx$, $ty$)
      $count = count + 1$
    end if
  end for
end for
$result = accum/count$
```

Figure 5: *Pseudocode of the geometry-sensitive illumination filter. Constants α and β determine the thresholds for spatial distance and difference between normals, respectively.*

we have no knowledge of the mean distance from the receiving point to nearby surfaces.

The pseudocode for the box filter is given in Figure 5. We simply loop over the pixels under the box kernel, and compute geometric distance and normal difference between the pixel being processed and the pixels under the kernel. Two threshold values α and β are used for thresholding the differences in positions and normals, respectively. The same thresholding criteria were also used by Segovia et al. [SIMP06b] for constructing the discontinuity buffer, where only neighboring pixels are considered. The value for α obviously depends on the scale of the scene, while setting $\beta = 0.8$ seems to work reasonably well in all situations.

We note that the support of the filter contains always at least one pixel, namely the one that is directly under the center. When n or m is even, the filter cannot be exactly centered, but since we are dealing with smoothly varying indirect illumination, the resulting half-pixel shift is impossible to notice in practice. We also experimented with $2n \times 2m$ wide box filters, but the increase in quality was marginal, while the cost of filtering was roughly doubled.

When motion blur is used, the performance could be further optimized by using only a subset of the VPLs for each sub-frame, as proposed by Keller [Kel97].

6. Results

We studied the effect of VPL reuse on the performance by comparing against a method that generates all VPLs anew each frame. All other parts of the algorithm are the same for both our algorithm and the comparison method. We note that the comparison method strongly resembles the state-of-the-art algorithm of Segovia et al. [SIMP06b].

All performance measurements were run on a dual-core 2.2GHz AMD Athlon 64 with 1GB of memory and an NVIDIA GeForce 8800 GTX with 768MB of memory. Only one of the CPU cores was used, i.e. our program was single-threaded.

Three test scenes were used. CORNELL is the classical Cornell box, SIBENIK is an architectural model, and MAZE features a simple surrounding scene with a detailed statue. In each scene, we measured the efficiency of our method and the comparison method with a single light source. Both spot and omnidirectional light sources were tested. Because we use paraboloid mapping for the shadow maps of the VPLs, the scenes need to be tessellated so that linear triangle boundaries do not cause objectionable artifacts in the paraboloid maps. The scenes and triangle counts before and after tessellation are shown in Figure 6.

The number of VPLs was fixed to 256, which gave the best output quality considering both the visibility of artifacts and the speed of rendering. The recomputation budget was set to 4–8 VPLs per frame for all test cases. The resolution of the shadow maps for VPLs was 256×256, and with 16-bit depth values, the total video memory consumed by the 256 shadow maps was 32MB. For direct illumination we compute a shadow cube map with 1024*x*1024 resolution on each side. The direct shadow lookup is smoothed using a 16-tap percentage-closer filter [RSC87]. Aside from shadow maps, all buffers are in 16-bit floating-point format.

We chose to always split the G-buffer into 4×4 tiles, since this resulted in best rendering speed with 256 VPLs. With more tiles, the filtering step becomes slow enough to outweigh the gains from a faster accumulation step, and with fewer tiles, the accumulation starts to dominate.

As was mentioned in Section 3, the CPU calculations related to VPL management are always executed in parallel

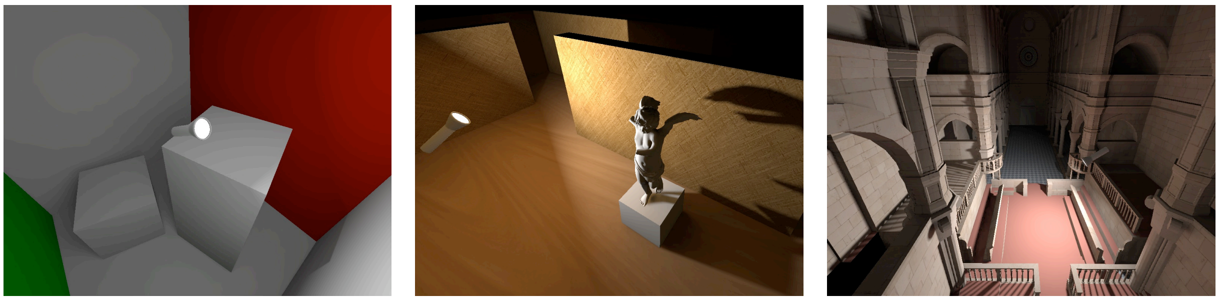
CORNELL: 32 / 4.4K triangles
MAZE: 55K / 63K triangles
SIBENIK: 80K / 109K triangles

Figure 6: *The test scenes used. For each scene, both the original and tessellated triangle counts are given. The tessellated version of the scene is used only for rendering the paraboloid shadow maps for the VPLs.*

with GPU tasks by allowing one-frame delay in the positions and intensities of the VPLs. Since the CPU processing power is not the limiting factor in our test cases, the frame rates are exactly the same with spot and omnidirectional primary light sources. The time taken by VPL management depends somewhat on the complexity of the scene, mainly because of the ray casts involved. With spot light sources, the VPL management took 3.6–5.0 milliseconds in our test scenes, while omnidirectional light sources required 16.0–16.2 milliseconds. The huge difference is explained by the need for 3D Delaunay tetrahedralization with omnidirectional lights, whereas 2D triangulation is sufficient for spot lights.

Table 1 summarizes the timings from our test runs. The timings for each individual GPU task were obtained by flushing the GPU pipeline before and after each task, whereas FPS measurements were made without any additional synchronization. This explains the slight discrepancy between the reported FPSs and the timing breakdowns. As can be observed, our algorithm was up to 7 times faster than the comparison method.

Relative to the comparison method, our algorithm scales favorably with number of VPLs and scene complexity because our technique updates fewer shadow maps per frame. These are important characteristics due to the increasing trend in scene complexities, and because the number of VPLs is an important factor in the perceived quality of the indirect illumination. Also, the results produced by our algorithm seem to exhibit less flickering artifacts than the comparison method using the same number of VPLs. Image resolution affects all frame buffer operations approximately linearly, and therefore the relative speedup decreases in higher resolutions.

In an additional test with 1024 VPLs, we obtained a frame rate of 31 FPS in Sibenik in 1024×768 resolution, while the comparison method performed at only 2 FPS. Unfortunately we ran into performance problems with the CGAL library when using this many VPLs, especially with omnidirectional light sources. This resulted in unpredictable stutters in the frame times. Thus, a faster implementation for performing Delaunay triangulations and tetrahedralizations would be needed. Ideally, the tetrahedralization should be replaced with a triangulation on the surface of the sphere.

7. Discussion and Future Work

Our implementation supports diffuse surfaces only, but handling glossy BRDFs would be quite simple in theory. The only modification would be in the accumulation step that would need to consider the BRDFs of the surfaces at the VPL and at the receiving point. In particular, VPL management would not be affected at all. Unfortunately, with highly specular BRDFs, the contributions of individual VPLs would become more conspicuous, necessitating the use of significantly more VPLs than for diffuse or only mildly glossy receivers.

Our primary light sources are point-like, and extending our algorithm to true area light sources is a potential avenue for future work. It seems that the VPL distribution calculations would then need to be done in 4D space. However, the number of VPLs needed for achieving a given quality should remain approximately the same as in the case of point light sources. A crude approximation for area light sources could be made by computing the intensities of VPLs as if the primary light source were an area light, even if the VPLs are in fact created by casting rays from a single point.

Perhaps the biggest limitation of our approach is that indirect illumination can bounce from static geometry only. We feel, however, that the ability of dynamic objects to receive indirect illumination largely alleviates this limitation. This support for dynamic objects is a major improvement over previously presented precomputation-based approaches [KAMW05, KTHS06], which are unable to illuminate dynamic objects. The lack of indirect illumination effects caused by dynamic objects seems to be a relatively minor issue, since shadowing of direct illumination—which of course works in conjunction with our method—is a much more noticeable dynamic illumination effect. Fur-

Scene	Resolution	Direct illum.	Indirect illumination				FPS		Speedup factor	
		SM+PCF	G-buffer rendering	Σ Common (Table 2)	Shadow map rendering: Our method	Shadow map rendering: Comp. method	Comp. method	Our method	Indirect	Frame
CORNELL	1024 × 768	4.4	1.1	12.1	0.7	30.8	35.1	65.1	**2.2**	**1.9**
	1600 × 1200	6.7	2.0	24.1			21.2	29.7	**1.5**	**1.4**
MAZE	1024 × 768	5.6	1.5	12.1	2.0	82.3	12.5	49.2	**5.1**	**3.9**
	1600 × 1200	8.8	2.5	24.1			10.2	28.5	**3.4**	**2.8**
SIBENIK	1024 × 768	6.9	1.8	12.1	3.1	140.3	7.1	48.6	**9.8**	**6.8**
	1600 × 1200	10.2	2.9	24.1			6.3	25.9	**5.2**	**4.1**

Table 1: *Timings from our test scenes. The number of VPLs is 256 in each case. All timings are in milliseconds. The columns from left to right: name of the test scene; resolution; time taken by direct illumination including shadow map rendering and percentage-closer filtering; rendering the G-buffer; common scene-independent operations (see Table 2 for breakdown); rendering shadow maps for new VPLs in our method; rendering shadow maps for all VPLs in the comparison method; frames per second for comparison method; frames per second for our method; speedup in indirect illumination calculations; speedup in total frame time. All results are averaged over 1000 frames where both the light source and the camera are moving. Note that the total frame time is the same for spot and omnidirectional lights for our method, since the VPL management is done on CPU in parallel with GPU operations.*

thermore, shadowing of indirect illumination could probably be approximated using ambient occlusion techniques [KL05, MMAH].

Limiting the illumination path length to single bounce makes the reuse of VPLs simple and efficient, but it limits the extent to which our method can approximate true global illumination. However, extending the algorithm to handle multiple bounces seems relatively straightforward. Instead of reusing first-hit VPLs, we could reuse entire paths, and validating the first-hit VPLs would also validate the path continuing from it, since the scene and the path remain static.

Another major limitation is that the method is not view-dependent. In a realistic application with large scenes the distribution of VPLs should be driven by view-importance, i.e., computations should be concentrated on effects visible to the camera. A simple distance-based heuristic augmented with precomputed visibility data might already work satisfactorily, but more elaborate methods could be developed, e.g. along the lines of bidirectional instant radiosity [SIMP06a].

As was mentioned in Section 5.2, we currently assign VPLs to G-buffer tiles in the order they are generated. It could be very useful to use some kind of heuristics for assigning the VPLs to tiles so that each tile has a representative set of VPLs. This could avoid worst-case scenarios where all brightly illuminating VPLs get assigned to a single tile. This kind of uneven VPL assignment yields high variance in the structured illumination pattern, which in turn results in filtering artifacts at sharp boundaries.

Considering our method's performance, scalability, simplicity, potential extensions, and unintrusiveness to the rest of the rendering pipeline, we believe that it has the potential to become the algorithm of choice for rendering global illumination effects in future applications.

Resolution	Split	Accum.	Combine	Filter	Σ **Common**
1024 × 768	2.1	5.7	0.7	3.6	**12.1**
1600 × 1200	4.1	11.7	0.7	7.6	**24.1**

Table 2: *Scene-independent common steps in indirect illumination rendering. Timings are in milliseconds. These steps are performed both in our method and in the comparison method. Columns from left to right: G-buffer splitting; indirect illumination accumulation; G-buffer combining; incident illumination filtering; total time taken by these common operations.*

Acknowledgments

Sibenik cathedral model courtesy of Marko Dabrovic, RNA studio, www.rna.hr. The graphics group of Helsinki University of Technology was funded by the National Technology Agency of Finland, Anima Vitae, ATI Research Finland, Hybrid Graphics/NVIDIA Helsinki, and Remedy Entertainment. In addition, Jaakko Lehtinen was supported by an NVIDIA Fellowship, and Timo Aila was supported by the Academy of Finland (Grant# 108 222).

References

[AK00] AURENHAMMER F., KLEIN R.: Voronoi diagrams. In J. Sack and G. Urrutia, editors, *Handbook of Computational Geometry, Chapter V* (2000), Elsevier Science Publishing, pp. 201–290.

[ARBJ03] AGARWAL S., RAMAMOORTHI R., BELONGIE S., JENSEN H. W.: Structured importance sampling of environment maps. *ACM Trans. Graph. 22*, 3 (2003), 605–612.

[BAS02] BRABEC S., ANNEN T., SEIDEL H.-P.: Shadow mapping for hemispherical and omnidirectional light sources. In *Proc. Computer Graphics International* (2002).

[Cap99] CAPPANERA P.: *A Survey on Obnoxious Facility Location Problems*. Tech. Rep. TR-99-11, Università di Pisa, Dipartimento di Informatica, 1999.

[CGA06] CGAL EDITORIAL BOARD: *CGAL-3.2 User and Reference Manual*, 2006.

[CW93] COHEN M. F., WALLACE J. R.: *Radiosity and Realistic Image Synthesis*. Morgan Kaufmann, 1993.

[DDJ06] DEL MORAL P., DOUCET A., JASRA A.: Sequential Monte Carlo samplers. *J. Royal Statist. Soc. B 68*, 3 (2006), 1–26.

[DFG01] DOUCET A., FREITAS N. D., GORDON N. (Eds.): *Sequential Monte Carlo methods in practice*. Springer, 2001.

[DS05] DACHSBACHER C., STAMMINGER M.: Reflective shadow maps. In *Proc. Symposium on Interactive 3D Graphics and Games* (2005), pp. 203–231.

[DS06] DACHSBACHER C., STAMMINGER M.: Splatting indirect illumination. In *Proc. Symposium on Interactive 3D Graphics and Games* (2006), pp. 93–100.

[HS98] HEIDRICH W., SEIDEL H.-P.: View-independent environment maps. In *Proc. Eurographics/SIGGRAPH Workshop on Graphics Hardware* (1998).

[KAMW05] KRISTENSEN A. W., AKENINE-MÖLLER T., WANN JENSEN H.: Precomputed local radiance transfer for real-time lighting design. *ACM Trans. Graph. 24*, 3 (2005), 1208–1215.

[Kel97] KELLER A.: Instant radiosity. In *Proc. ACM SIGGRAPH 97* (1997), pp. 49–56.

[KH01] KELLER A., HEIDRICH W.: Interleaved sampling. In *Proc. Eurographics Workshop on Rendering* (2001), pp. 269–276.

[KL05] KONTKANEN J., LAINE S.: Ambient occlusion fields. In *Proc. Symposium on Interactive 3D Graphics and Games* (2005), pp. 41–48.

[KTHS06] KONTKANEN J., TURQUIN E., HOLZSCHUCH N., SILLION F.: Wavelet radiance transport for interactive indirect lighting. In *Proc. Eurographics Symposium on Rendering* (2006), pp. 161–171.

[Lan02] LANDIS H.: Global illumination in production. ACM SIGGRAPH 2002 Course #16 Notes, 2002.

[LaV06] LAVALLE S. M.: *Planning Algorithms*. Cambridge University Press, Cambridge, U.K., 2006. Chapter 5: Sampling-Based Motion Planning.

[LL04] LINDEMANN S. R., LAVALLE S. M.: Incrementally reducing dispersion by increasing Voronoi bias in rrts. In *Proc. IEEE International Conference on Robotics and Automation* (2004), vol. 4, pp. 3251–3257.

[Mat99] MATOUSEK J. (Ed.): *Geometric Discrepancy: An Illustrated Guide*. Vol. 18 of Algorithms and Combinatorics. Springer, 1999.

[MMAH] MALMER M., MALMER F., ASSARSON U., HOLZSCHUCH N.: Fast precomputed ambient occlusion for proximity shadows. *Journal of Graphics Tools, to appear*.

[Nie92] NIEDERREITER H.: *Random number generation and quasi-Monte Carlo methods*. Society for Industrial and Applied Mathematics, 1992.

[OBM06] OSMAN B., BUKOWSKI M., MCEVOY C.: Practical implementation of dual paraboloid shadow maps. In *Sandbox '06: Proc. ACM SIGGRAPH Symposium on Videogames* (2006), pp. 103–106.

[RSC87] REEVES W. T., SALESIN D. H., COOK R. L.: Rendering Antialiased Shadows with Depth Maps. In *Computer Graphics (Proceedings of ACM SIGGRAPH 87)* (1987), pp. 283–291.

[SH75] SHAMOS M. I., HOEY D.: Closest-point problems. In *Proc. 16th Annu. IEEE Sympos. Found. Comput. Sci* (1975), pp. 151–152.

[SIMP06a] SEGOVIA B., IEHL J.-C., MITANCHEY R., PÉROCHE B.: Bidirectional instant radiosity. In *Proc. Eurographics Symposium on Rendering* (2006), pp. 389–398.

[SIMP06b] SEGOVIA B., IEHL J.-C., MITANCHEY R., PÉROCHE B.: Non-interleaved deferred shading of interleaved sample patterns. In *Proc. Graphics Hardware* (2006), pp. 53–60.

[SKS02] SLOAN P.-P., KAUTZ J., SNYDER J.: Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments. *ACM Trans. Graph. 21*, 3 (2002), 527–536.

[TL04] TABELLION E., LAMORLETTE A.: An approximate global illumination system for computer generated films. *ACM Trans. Graph. 23*, 3 (2004), 469–476.

[Wal04] WALD I.: Realtime Ray Tracing and Interactive Global Illumination. *PhD thesis, Saarland University* (2004).

[WRC88] WARD G. J., RUBINSTEIN F. M., CLEAR R. D.: A ray tracing solution for diffuse interreflection. *Computer Graphics (Proc. ACM SIGGRAPH 88) 22* (1988), 85–92.

[YKS78] YAKOWITZ S., KRIMMEL J. E., SZIDAROVSZKY F.: Weighted Monte Carlo integration. *SIAM Journal on Numerical Analysis 15*, 6 (1978), 1289–1300.

[ZIK98] ZHUKOV S., IONES A., KRONIN G.: An ambient light illumination model. In *Proc. Eurographics Workshop on Rendering* (1998), pp. 45–55.