Hierarchical Screen Space Indirect Illumination For Video Games

Cyril Soler, Olivier Hoel, Franck Rochet, Frederic Jay, Nicolas Holzschuch

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Hierarchical Screen-Space Indirect Illumination for Video Games

Cyril Soler — Olivier Hoel — Franck Rochet — Frédéric Jay — Nicolas Holzschuch

N° 7162

Decembre 2009

Thème COG

ISSN 0249-6399 ISRN INRIA/RR--7162--FR+ENG

Hierarchical Screen-Space Indirect Illumination for Video Games

Cyril Soler , Olivier Hoel *, Franck Rochet , Frédéric Jay †, Nicolas Holzschuch ‡* †

Thème COG — Systèmes cognitifs
Équipes-Projets Artis

Rapport de recherche n° 7162 — Decembre 2009 — 19 pages

Abstract: Indirect lighting accounts for subtle but essential effects in virtual scenes, and plays a great role in our perception of geometry. It is especially wanted in video games, on fully dynamic scenes, where it greatly enhances the perceived realism. In this paper, we present a screen-space hierarchical algorithm for computing indirect lighting for animated scenes. Our algorithm is fully compatible with deferred-shading rendering engines for video games, and computes indirect lighting in less than 10 ms, leaving enough computation time for other gaming tasks, such as interaction and animation. Our algorithm works in two steps: first, we compute indirect illumination in screen-space at all possible scales, then we filter and combine together the illumination received at the different scales. We describe both the algorithm and its practical integration inside a commercial video-game rendering engine.

Key-words: Global Illumination, Real Time, Video Games

* INRIA Rhône Alpes
† EDEN Games
‡ INRIA Rhône Alpes

Centre de recherche Inria de Grenoble – Rhône-Alpes
655, avenue de l'Europe, 38334 Montbonnot Saint Ismier (France)
Téléphone : +33 4 76 61 52 00 — Télécopie +33 4 76 61 52 52

Eclairage Indirect Hierarchique en Espace Image piour les Jeux Vidéos

Résumé : L'éclairage indirect prend en compte des effets lumineux subtils dans les scènes virtuelles. Ces effets sont néanmoins très utiles dans la perception de la géométrie, ce qui est particulièrement important dans les jeux vidéos où il augmente considérablement le niveau de réalisme. Dans ce papier nous présentons un methode de calcul en espace image de l'éclairage indirect pour des scènes animées. Notre méthode est compatible avec la technique du *deferred shading* utilisée dans les jeux vidéos, et calcule l'éclairage en moins de 10 ms. Ceci laisse le temps au moteur de jeux d'effectuer les taches incompressibles que sont l'intelligence artificielle, la communication réseau et l'interaction avec le joueur. Notre méthode est basée sur un calcul multi-echelles de l'éclairage dans un *mipmap*, qui est ensuite re-combiné et composé avec l'eclairage direct. Nous décrivons également comment utiliser cette méthode dans un pipeline réel de jeu vidéo.

Mots-clés : Eclairage Global, Temps Réel, Jeux Vidéos

Contents

1 Introduction

Computing the indirect lighting in video games adds a lot to both gameplay and scene realism. However, computing indirect lighting in real time is both challenging and computationally expensive, as it requires to solve an integral equation. There are several solutions to compute indirect lighting on the GPU (*e.g.* [2, 3, 13, 15, 8]). However, in order to be usable in commercial game rendering engines, an illumination algorithm should satisfy some fairly strict criteria:

- The whole rendering algorithm (display and illumination simulation) should run really fast. The game engine must display every frame within the allocated budget of 33 ms, to achieve real-time rendering, including physical simulation, artificial intelligence, geometric interactions and data streaming, none of which can be reduced. The specific budget for computing indirect illumination is at most 5 to 10 ms.
- The video game engine should run at a constant frame rate. The algorithm used for any computation must be independent from unbounded parameters, such as the amount of polygons to display. For this reason, image-space computations, such as deferred shading, are becoming a standard in video game engines.
- The algorithm used for computing indirect lighting must fit within the framework used in the rest of the video game engine, keeping the modifications to a minimum. Both for software stability and efficiency.
- The pictures must absolutely be noise-free and the animations must remain stable and without any temporal artifacts. On the other hand, full numerical accuracy is not necessary, and we can make drastic assumptions to reduce the computational load.

In this paper, we present a new algorithm for computing indirect illumination inside a deferred shading framework, that addresses these requirements. Our algorithm works in screen-space, hierarchically. First, we compute indirect illumination at several scales, using multiple resolutions of screen space. These contributions are then upsampled and combined together to result in indirect lighting between all visible parts of the scene. To enhance the stability of the algorithm, we show how to combine multiple indirect illumination components from several cameras (cooperative screen-space illumination), and how to combine indirect lighting across successive frames, with filtering to avoid noise.

Our paper is organised as follows: in the next section, we review existing algorithms for global illumination and indirect lighting in real-time. In section 3, we describe our algorithm for hierarchical screen-space indirect illumination. Then in section 4 we discuss practical implementation issues, especially integration inside a commercial video game engine. In section 5, we show examples validating our method both in terms of accuracy and applicability to video gaming. Finally, in section 6, we conclude and present new directions for future work.

2 Previous Work

Among the large number of research papers in global illumination, we will limit ourselves to works that are close to our topic: ambient occlusion, screen-space indirect illumination and illumination for video games.

2.1 Ambient Occlusion

Ambient Occlusion was an early attempt at simulating global illumination effects in real time. It was originally defined as *obscurance* by Zhukov *et al.* [25] to be the form factor between each point and the part of the environment that is not occluded by nearby geometry as viewed from this point. In all succeeding works, the main goal was real time rendering. In 2003, Iones proposes to pre-compute and store obscurance in light maps [25, 5]. This work was further extended to simulate color bleeding effects [12], using the obstacle's reflectance to simulate color bleeding.

Several papers present possible extensions of ambient occlusion to dynamic scenes at the cost of additional object-space pre-computation: In 2005 Kontkanen and Laine proposes to pre-compute and store per-object ambient occlusion fields [10]. This allowed at run time a fast computation of ambient occlusion with moving objects. This approach relies on approximations of the occluder objects and also needs a heuristic to combine different objects for shading the same pixel. Oat and Sander pre-compute a circular approximation of the unoccluded field around each point of a rigid scene [17]. This allows to generate ambient occlusion accounting for dynamic illumination. Ambient occlusion has also been experimented for animated characters [9] and trees [4].

Since a few years, it is possible to compute ambient occlusion in image space on GPU, thanks to the power of programmable graphics cards [6]. Ambient is always a very local computation, as one only needs to measure the effect of geometry in the near vicinity of each point. This allows efficient implementations such as the one proposed by Shanmugam and Arikan [21]. Epic Games pre-

sented at GDC 09 a fast and visually accurate screen-space ambient occlusion (SSAO) algorithm using down sampled resolution and time coherency [23], successfully used in their Unreal Engine 3 technology. Ritschel extends the SSAO by reading the long distance incoming light from an environment map, and the short distance light from nearby objects [20] hence simulating local effects such as color bleeding.

2.2 Interactive and real-time indirect illumination

Accurate global illumination in real time is still a challenge. Existing methods approach it in several ways by precomputing data, neglecting visibility, or limiting the number of bounces.

Coherent Surface Shadow Maps is a multi-bounces indirect illumination algorithm for dynamic scenes made of several rigid objects [19]. A hemicube is swept accross surfaces and visibility to the rest of the scene is computed. Because it is only suitable to rigid objects and is significantly memory consumming, this algorithm would not fit video games. Reflective Shadow Maps [1] uses the traditionnal shadow maps technique to encode which points in the scene actually contain direct illumination. The contribution from these points is then splatted in screen space using deferred shading. This technique only handles one bounce of indirect lighting and neglects visibility during this step.

Implicit visibility [3] is a solution to the inherent cost of accounting for occlusion: the space of directions is binned and links are established between bins of different points. Only the closest source point for each bin of a receiver point is considered, which implicitely accounts for visibility. A very similar solution is proposed by Dachsbacher in which links possibly transport a negative amount of energy to counterbalance normally occluded contributions [2]. These methods however are not suitable to dynamic environments, as they both rely on a precomputation of a hierarchical link structure, and scene geometry preprocessing.

Recently, true screen-space indirect illumination algorithms have been published. Nichols and Wyman propose a multiresolution spatting method based on careful selection and clustering of virtual indirect point light sources [13, 14]. Because screen-space methods only account for the geometry facing the user in the screen, more accurate results can be expected for height fields. Nowrouzezahrai proposes an extension of his screen-space soft self-shadowing technique [24] to handle indirect illumination [15]. For this, the visibility and radiance at each point are approximated by low order polynomials and spherical harmonic distributions. This algorithm is one of the few handling glossy reflexions, although it is limited to low frequency lighting effects.

2.3 Global illumination for video games

Although screen-space ambient occlusion is now widely used in video games (*e.g.* S.T.A.L.K.E.R Clear Sky [22], Gears of War 2, Crysis Warhead, Uncharted 2: Among theves,...), it does not approximate indirect illumination very well. Indirect illumination at arbitrary distances is still a rare asset. This is mostly because of the extreme CPU/GPU/Memory conditions to fill for fitting such a rendering algorithm into a complete game pipeline. For a long time, indirect

illumination has been pre-computed and stored into object textures (*e.g.* in the game Counter-Strike).

Irradiance volumes has been a first solution suitable to video games [16], based on precomputing irradiance at regularly spaced positions using in a static scene using spherical harmonics, and coupled with precomputed radiance transfer for moving objects. This technique however was not working with dynamic lighting and could not handle dynamic scene geometry.

In his Siggraph'09 course, Anton Kaplanyan presented a remarkable extension of this technique: the radiance volume [8], used in CryEngine 3. His method simulates volumetric light propagation, stored in a 4 channel 3D texture of spherical harmonic distributions fed with direct illumination. The radiance volume technique is completed by local screen-space indirect illumination for color bleeding, and screen-space ambient occlusion.

2.4 Contributions

We compute the indirect illumination in screen space only, using deferred shading. This makes our algorithm similar to screen-space ambient occlusion methods. However, to allow light exchanges at an arbitrary distance while keeping short range texture look-ups, we make this algorithm hierarchical. Because our primary application field is video games, we also give practical explanations and feedback on the integration of our technique into a real game engine, developed by EDEN Games (used in the game *Alone in the dark*).

3 Hierarchical screen-space indirect illumination

We describe in this section our algorithm for computing indirect illumination in screen-space space. We start by examining sampling issues so that the actual indirect illumination is properly computed whenever possible (Section 3.1), then discuss the approximations we make and the actual GPU implementation in Section 3.2.

3.1 Screen-space sampling of indirect illumination

In order to compute the indirect illumination at every point in the scene, one needs according to the light transport equation, to gather the direct illumination from all other points in the scene that are visible from that point [7]. Calling $\rho(x,\omega,\omega')$ and $L(x,\omega)$ the reflectance and direct radiance functions at x and directions ω and ω', the indirect illumination at a point x in a direction ω can be computed as:

$$L'(x,\omega) = \int_\Omega \rho(x,\omega,\omega') L(x',-\omega') \cos\theta d\omega$$

Point x' is the point seen from x in direction ω' and θ is the angle between the normal at x and direction $-\omega'$. This integral can be carried on all scene surfaces instead of directions, introducing a new visibility term:

$$L'(x,\omega) = \int_S \rho(x,\omega,\omega') L(x',\omega') \frac{\cos\theta\cos\theta'}{\|x-x'\|^2} v(x,x') ds$$

where $v(x, x')$ is defined by:

$$v(x, x') = \begin{cases} 1 & \text{if } x \text{ and } x' \text{ are mutually visible} \\ 0 & \text{otherwise} \end{cases}$$

For numerically computing this integral, one usually sums over a large number N of small areas ds across the scene, so that:

$$L'(x, \omega) = \sum_{i=1}^{N} \rho(x, \omega, \omega'_i) L(x_i, -\omega'_i) \frac{\cos\theta_i \cos\theta'_i}{\|x - x_i\|^2} v(x, x_i) ds_i \tag{1}$$

To perform the same computation in a deferred shading setup, we need to draw samples in image space. We thus have to re-write ds in terms of the elementary screen-space area dP covered by each of our screen-space samples. Assuming notations of Figure 1, we can write:

$$ds = \frac{z^2 4 \tan\frac{f_h}{2} \tan\frac{f_v}{2} dP}{WH} \quad \frac{\cos\alpha}{\cos\beta} \tag{2}$$

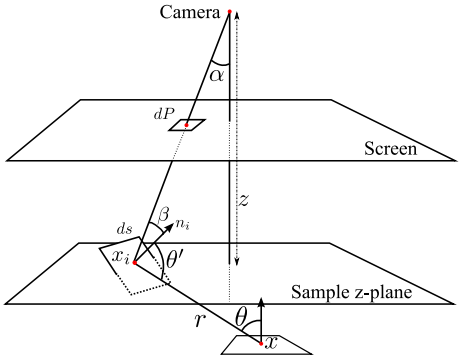


Figure 1: Notations for the computation of the indirect illumination in screen space. x is the point being computed, x_i is the current sample, dP the screen differential area and ds the scene differential area of current sample.

In this expression, f_h and f_v are the horizontal and vertical field of view, W and H the width and height of the screen in pixels, and z the depth of the projected pixel. The left term in Equation 2 is the area of one pixel projected at depth z on a surface parallel to the screen. The right term corrects this area when the receiving surface is not parallel to the screen. Reporting Equation 2 in Equation 1 gives us the indirect illumination of a point x when sampled in screen-space (we adopt notations ρ_i, v_i, L_i for the reflectance, visibility and illumination terms of equation 1):

$$L'(x, \omega) = \sum_{i=1}^{N} \rho_i L_i \frac{\cos\theta_i \cos\theta'_i}{r_i^2} v_i \frac{z_i^2 4 \tan f_h \tan f_v dP_i}{WH} \frac{\cos\alpha}{\cos\beta} \tag{3}$$

Equation 3 cannot however directly be used in the computation: It is well known in the global illumination community that the point-to-differential area

form-factor component of Equation 3 is a source of variance, since this term is not bounded as $r_i = \|x - x_i\|$ gets close to zero. This is indeed what we observed (See Figure 5) in our first experiments. We propose instead the following workaround: In Equation 3, we replace the point to differential area form factor by a closed formula that normally stands for discs [18]:

$$\frac{\cos\theta' ds}{r^2} \approx \frac{R^2}{r^2 + R^2} \qquad \text{with} \qquad R^2 = \frac{1}{\pi}\cos\theta' ds \tag{4}$$

In the discrete summation, this approximation is only valid for small values of area ds, but considerably reduces the variance due to the denominator in the leftwise expression. We validate in Figure 5 the gain in using this formulation (Observe that the giant spikes in the red curve (eq.3) are not present in the green (eq.4)). In practice, because it is applied in screen space, the resulting illumination will be heavily biased – *e.g.* when only small portions of the entire scene are visible. However, we found important to get the result as close a possible to the actual illumination when the geometric configuration allows it, and noise-free in any case.

3.2 GPU Hierarchical screen-space computation

Using Equation 3 in a shader would eventually result in an unacceptable compromise between too much locality and too large a cost. Indeed, long distance light interactions would need costly long distance texture lookups. We thus turn toward a hierarchical implementation of Equation 3: the depth, normals and materials buffers are mipmapped and successively sent to a unique local shader. The contribution from all mipmaps are then up-sampled and added together (see Figure 3 for a view of the full pipeline). Before that, we discuss the approximations we made in our GPU implementation.

Approximations

We assume diffuse reflectances everywhere. Applying Equation 3 in a deferred shader with glossy materials is not an intrinsic limitation of our method, although it certainly would need to modify the sampling in image space to importance-sample the BRDF. We assume in this paper, for the sake of simplicity that ρ is an RGB albedo.

Because not all surfaces contributing to the indirect illumination at point x appear on the screen, screen-space sampling the indirect illumination is necessarily approximate. This approximation can certainly be bad if only a small part of directly illuminated areas of the scene actually appear on screen. Similarly, properly accounting for the visibility between samples is not possible, since no parallax information is present in the screen. In such a case, completely omitting visibility appears to be an appropriate choice as compared to more accurate but costly estimates that would eventually produce inconsistencies across frames. We thus suppose that $v(x, x_i) = 1$ everywhere.

Indirect lighting shader

In each mipmap level, the shader draws samples in a constant crown defined by two radii r and $2r$ (See Figure 2). This choice ensures that by combining

contributions from all mipmap levels, the entire screen eventually gets sampled without cracks nor overlaps, provided that for the first mipmap level, the full disc is also sampled. The optimal value for r is the thus smallest possible value so that the entire screen is eventually sampled when combining contributions from all mipmap levels. Keeping a constant number of samples across mipmap levels also automatically adapts the sampling density in the resulting image so that farther regions gets sampled less. Because at step k the shader works on a mipmap at scale 2^k, the collected energy must be further scaled by a factor 2^{2k}.

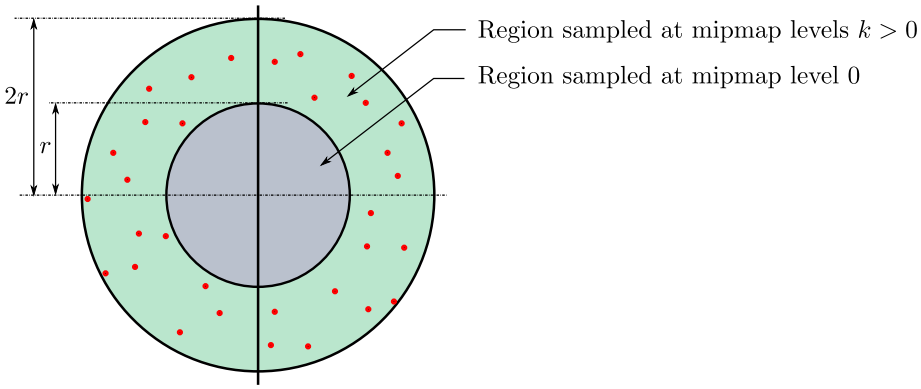


Figure 2: Placement of samples in the shader. In all mipmaps except the first (largest) one, the same region is sampled. This region is a crown delimited by two circles of radii r and $2r$. Applying this sampling to all mipmaps effectively samples the entire image while only performing short-distance lookups. In the first mipmap, we sample the remaining inside circle for completing the image sampling.

Continuous combination of all level contributions

We filter and blend illumination contributions from all mipmap levels using a joint bilateral up-sampling filter [11]: mipmaps levels of indirect illumination are computed starting from the smallest, and added into a buffer. At the end of each step, this buffer is up-sampled to the size of the next mipmap. This way, the illumination at level k is filtered several times in the final result. This produces smooth gradients in the final indirect lighting results, low noise and no edge artifact. In Figure 4 of the results section, we show an example where one can easily see the contribution of each mipmap level separately.

An important property is that we don't need to perform the computation starting at the full screen resolution, provided that the illumination mipmapped results are properly up-sampled to the final image resolution. This gives us an additional parameter to act on the speed of the algorithm (We use half and quarter start resolutions mainly), at the expense of image-space frequency content. Figure 8 shows the effect fo variing this parameter.

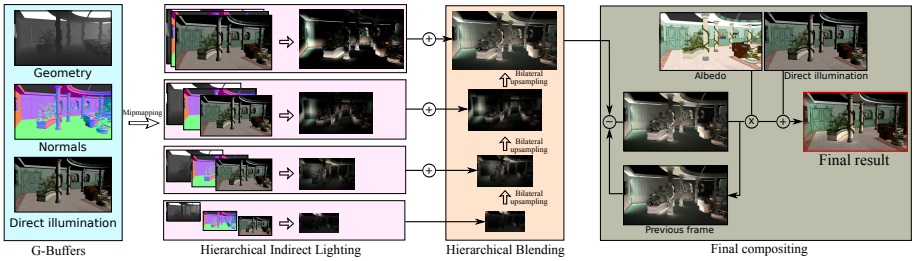


Figure 3: Complete pipeline for our screen-space indirect illumination algorithm. First, the direct illumination, normals and depth buffers coming from the rendering loop are mipmapped. Then each set of mipmap is sent to the indirect lighting shader, and the resulting indirect illumination contributions are up-sampled and added together. At each frame, the newly computed values are averaged with the values of the previous frame (for compatibility tested warped pixels only). The obtained illumination is multiplied by the albedo buffer and added to the direct illumination.

3.3 Temporal coherence

Because of the discrete nature of the computation, the indirect illumination computed using the previous method can only be computed very fast at the price of artefacts, mainly due to the variance of the Monte-Carlo integration when too few samples are used. Fortunately, it is possible to drastically reduce this variance, at no extra cost by blending the currently computed indirect illumination I_i with that of the previous frame I_{i-1}, after warping it to fit the current camera parameters.

$$F_i = a\mathcal{W}(I_{i-1}) + (1-a)I_i$$

In this expression $\mathcal{W}$ represents the function that reaches for current pixel the position of this pixel as seen with the camera settings of the previous frame. Once found, the depth and normal of this transformed position is compared to the depth and normal of the corresponding pixel in frame $i-1$, to lower the chances of blurring irrelevant pixels together. This, in essence, performs a convolution over the set of illumination values computed at a constant 3D location over time. The larger the parameter a, the farther in time is the influence of previous computations. This means that pixels laying on fast moving objects must use a smaller value of parameter a to avoid ghosting effects due to averaging irrelevant pixels. New pixels revealed by camera pannings and rotations, that weren't on the screen in the previous frame, also have an a depending on the camera speed, to smoothly blend illumination values.

4 Integration in a game rendering pipeline

In order to validate our technique in an industry constrained game engine, we have implemented our technique in the engine which is used in the game Alone In The Dark "Near Death Investigation". We believe this rendering engine is typical of modern game rendering pipelines based on deferred shading.

Integrating our indirect illumination algorithm in this engine therefore proves both its feasibility and usefulness.

Integration into the game lighting pipeline The indirect illumination being computed *after* the lighting pass, it has to be composed later with the resulting image. The deferred lighting pass computes the whole lighting of the scene, including specular reflections, that shouldn't be used when computing indirect illumination like we said in section 3.2. So we had to output an intermediate render target with lit scene without specular lighting in order to have a correct lighting source. To avoid splitting the deferred illumination shader in two parts, we used multiple render targets (a.k.a. MRT) to output the additional target. Then we compute the hierarchical indirect illumination and compose the final image, using this simple formula:

$$C_{final} = C_{direct\ lighting} + C_{indirect\ lighting} * Albedo$$

Where $C_{direct\ lighting}$ stands for the direct illumination, and $C_{indirect\ lighting}$ is the indirect illumination we compute. Albedo is the material color or texture where the indirect lighting is applied.

Hierarchical maps generation Because we use the deferred buffers at different resolutions, we had to generate the down-sampled versions of the full screen color, normal and position buffers (*i.e.* the z component encoded in view space). When down sampling, we keep the sample that matches closest of the four z positions in the parent level of the sampled point to keep details of nearest objects and make down sampling artefacts less visible. Generating these hierarchies have negligible cost compared to the GPU cost of the whole screen-space indirect illumination.

Memory cost Position and color render targets costs us up to 33% more memory than with a classic deferred renderer (depending on the number of mipmap levels we use, we currently keep 4 mips from full 720p resolution). We also had to allocate a full screen RGB render target for the source lighting of the bleeding color (the direct lighting without specular term). Intermediate render targets used during the indirect lighting system comes from a pool of render targets that are recycled when possible, so they usualy don't add any supplemental memory cost. Temporal coherence also costs a lot of memory, because we must keep the last position map and the last indirect lighting map. Fortunately, temporal coherence is very fast (negligible regarding the cost of our main shader) and leads us to very smooth results.

Shader compilation We tried to compile the shader both with static branching version and dynamic branching of the shader. The static (with unrolled loops) version is quite faster than the dynamic branching one, but generates a much bigger shader code. This becomes a limitation when pushing the number of samples to upper limits: when using 64 samples per level, the indirect illumination shader compiles with approximately 4318 instructions (194 texture, 4121 arithmetic) and we didn't manage to build a 80 samples version without running out of constants or making the shader compiler hanging up. At these resolutions we were forced to use the dynamic version. In such a case, the

shader is more expensive to execute, so we aggressively decrease pixel cost by using down-sampled resolutions, which also leads to better GPU texture cache coherence.

Dynamic objects and characters A basic implementation of temporal coherence assumes your world is static, which is obviously not the case in a real game. To correctly handle dynamic objects, we update all pixels as if they were static (like suggested in [23]), and have an additional pass when dynamic objects overwrite their indirect lighting with the good values, since the world matrix of the object has changed since last frame. Because we use down-sampled resolutions, we cannot use the hardware depth buffer and we must use the down-sampled position map to do depth testing to update good pixels.

For dynamic objects, we transform each vertex twice using current frame and last frame world space position matrices to be able to re-project it accurately in the previous frame.

When using skinned characters, the skinning should also be performed twice, first with the current frame's skinning, and then with the previous frame skinning matrices. This requires additional storage and may drastically increase the number of constants sent to the shader. This would force us making the skinning twice, leading to very bad performance, and may overflow the number of vertex constants available (*e.g.* 224 float constants in shader model 3.0). We instead update skinned objects as if they were simple dynamic solid objects, supposing that their pose hasn't changed since last frame. We also ensure that the characters reflectance is always zero, so they don't modify erroneously their environment indirect lighting, and just receive a coarser, yet artifact free, approximation of the indirect lighting they should really receive.

The Half pixel hell When using 3D API like Direct3D, there is a well-known issue when mapping directly pixels to texels, you have to offset your texture coordinates with a half pixel to fetch the center of of the texel. Because we're using down-sampled resolutions, we had to be particularly careful when computing the offset. We had to divide the offset size each time we divided the resolution. Indeed, a bad offset calculation leads to visible artefacts, especially when sampling the position map at low resolutions.

5 Results

Validation In order to validate the model we use for computing the indirect lighting, we present in Figure 5 a comparison between the first bounce of indirect lighting computed with path tracing and with our GPU technique. For this we fed our shader with a direct illumination image also computed with path tracing. This way we can measure the convergence of our method with respect to a reference solution. For this test, we used $r = 40$, 4 mipmap levels, 64 samples per mipmap, and no temporal filtering. Image resolution is 720×720. Note that in practice, because we neglect visibility and compute the indirect illumination using only the direct illumination displayed on the screen, the results may be heavily biased. This test shows the smooth blending of illumination mipmap levels and the advantage of Equation 4 over Equation 3.

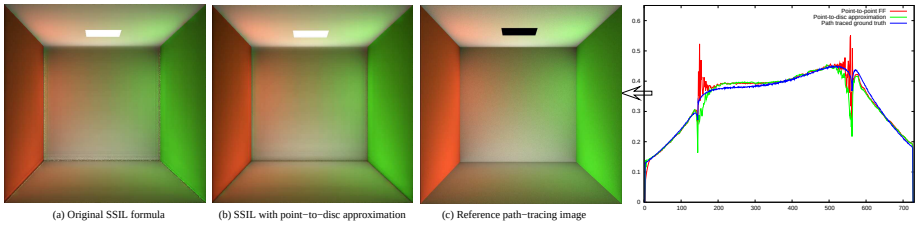
In Figure 4 we show the effect of the different mipmap levels that gather the indirect illumination, for a scene lit by a direct spotlight turned toward the ground. By successively allowing only the k^{th} mipmap level to contribute in the pipeline, we generated images that show which part of the indirect illumination each mipmap fills into the final image. This also illustrates that bilateral up-sampling does not introduce any bias and allows a smooth combination of the different illumination levels.

Figure 4: Using our technique, the indirect illumination is computed at various scales in screen space, to account for all-ranges indirect illumination. From left to right, top to bottom: the direct illumination component from a vertical spotlight, the indirect illumination computed by level 1 to 4, and the combined result of our shader. In levels 2 and 3, one can see the shape of the sampling area covered by the shader. Illumination from the different levels blend smoothly and continuously.

Computation times Figure 6 (*left*) shows GPU computation times for the differents steps of our method, with different quality settings. Each experiment was done on two different architectures: a nVidia GeForce 8800 GTS and a nVidia GTX260, rendering at a resolution of 1280×720. The sampling radius for the shader is $r = 20$ pixels, and the kernel of the up-sampling filter is 3×3 pixels. This table shows that we are able to reach performances that match video games constraints.

The varying parameters are the starting resolution (half or quarter resolution), the number of mipmap levels, and the number of samples used at each level. We observe that, for a same cost, we get a better quality by starting from a lower resolution but using more samples (entry 2 and 4). Local indirect illumination has less details but noise and variance is much more noticeable while animated, and better effect range (see Figure 8). Note that the temporal filtering cost is independent from these parameters (it only depends on the rendering resolution). The last entry shows that adding one indirect lighting level only adds a very small computation overhead (¡ 1%) whereas it greatly increases the effect range in the final result (see Figure 4).

On the right side of Figure 6, we show detailed GPU computation times for the different mipmap levels of indirect lighting. Note that more than 70% of computation time is spent in the first level, although it only contributes for the most local indirect illumination.



(a) Original SSIL formula (b) SSIL with point−to−disc approximation (c) Reference path−tracing image

Figure 5: Sanity check of formulas used in Equation 3 and 4 for computing the 1st bounce indirect illumination with our hierarchical deferred shading algorithm. While introducing a different kind of approximation, using the point-to-disc approximation of the point-to-differential area form factor drastically reduces the noise. On the right are slices of image intensity along the line indicated by the arrow for all three images.

Resolution	Quarter		Quarter		Half		Half		Half	
Mipmap levels	4		4		4		4		5	
Samples per level	64		256		256		64		64	
Architectures	(a)	(b)	(a)	(b)	(a)	(b)	(a)	(b)	(a)	(b)
HSSIL	7.1	2.3	26.8	8.9	102.4	29.9	26.6	7.8	26.8	7.85
Upsampling	5.7	1.7	5.7	1.7	7.1	2.1	7.1	2.1	7.2	2.1
Compositing	1.1	0.6	1.1	0.6	1.1	0.6	1.1	0.6	1.1	0.6
Total	13.9	**4.6**	33.6	**11.2**	110.6	32.6	34.8	**10.5**	35.1	**10.55**

Figure 6: Computation times in milliseconds for different parameter sets, measured using *(a)* NVidia 8800GTS and *(b)* NVidia GTX260 cards, for a display resolution of 1280×720 pixels. The sampling radius is $r = 20$ and the upsampling kernel size is 3×3.

Discussion A common problem which is inherent to screen space computation is that only visible geometry could participate to indirect lighting computation, and can sometimes result in noticeable popping or disappearing of indirect lighting when a bright object becomes visible. We implemented a partial solution to this problem: we extend our pipeline to compute the indirect illumination using additional cameras, and blend (with a max) the contributions of different cameras to each pixel that passes the depth and normal test we use in temporal coherence. However, unlike blending frames over time, this operation has a large cost because it needs to re-render the scene from as many different viewpoints. We don't think it can already be considered as a viable solution for existing gaming architectures. We therefore did not use it in any other screenshot than Figure 9 where we show an example of improving the indirect illumination using additional cameras.

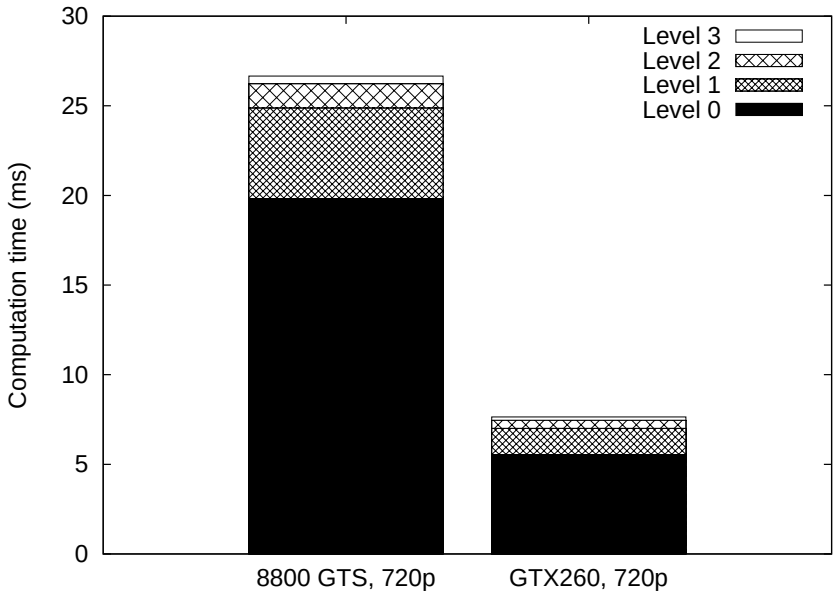


Figure 7: Computation times in milliseconds for different mipmap levels, for half resolution, 64 samples per level on the 8800 GTS card. Other parameters are those of Figure 6.

Depending on the architecture, a variable amount of GPU resources may be allocated to the computation of indirect lighting. Fortunately our algorithm allows a wide range compromise between accuracy and speed. One can easily increase the framerate by reducing the number of samples per level. Another efficient cost reduction technique is to apply the indirect illumination shader only starting from half or quarter resolution mipmaps.

6 Conclusion and future work

We have presented an algorithm for computing indirect illumination; our algorithm performs all computations in screen space, and computes indirect lighting hierarchically: illumination from distant points is computed with less precision then illumination from nearby points. In addition, we showed that temporal filtering drastically removes flickering, at a very low additional cost. Our algorithm is really fast: for typical settings, we are able to compute indirect illumination in approximately 10 ms. Because there are no precomputations and we rely only on screen-based information, our algorithm was easily combined with a deferred shading pipeline, in a typical industry-strength video-game engine.

Our tests show that our algorithm is both very fast and robust. It consumes little ressources, and can be used in production. In an industrial context, our algorithm can be used to replace light maps and other methods used to simulate indirect lighting, thus speeding up the process of developing a new game.

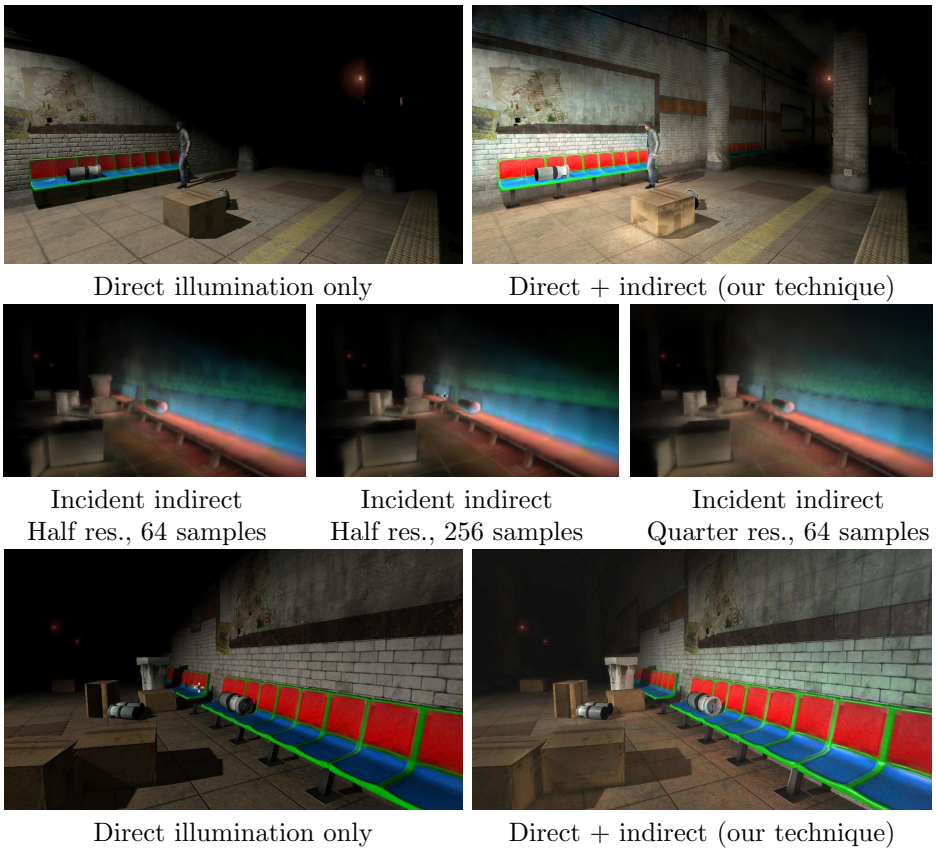


Figure 8: Screen-shots of the game Alone In the Dark, into which we have plugged our technique for computing the indirect illumination. The central row shows the incident indirect illumination computed with 3 different setups corresponding to the computation times of Figure 6. As illustrated by the bottom row, using 64 samples starting from the mipmap level of quarter resolution is sufficient to produce a satisfactory result, in 14 ms on a nVidia GForce 8800 GTS card. Note how much, on the top-right image, does the indirect illumination add realism and immersion, revealing regions of the scene not directly illuminated.

Single camera 2 additional side cameras

Figure 9: A typical situation where computing the indirect lighting using multiple cameras may prove useful. *At left*: with a single camera, the contribution from the illuminated side of the plant basket cannot be captured. *At right*: the illuminated side appears in one of the side views, which allows to compute its contribution and add it to the final image. However, we are not using this technique in the game engine because of its cost.

In future work, we want to address the case of non-diffuse BRDFs, as well as visibility issues in indirect lighting. Using arbitrary BRDFs would require combining importance sampling with screen-spaced sampling, a difficult step.

References

[1] Carsten Dachsbacher and Marc Stamminger. Reflective shadow maps. In *Proceedings of the 2005 ACM Symposium on Interactive 3D Graphics and Games*, pages 203–231. ACM SIGGRAPH, 2005.

[2] Carsten Dachsbacher, Marc Stamminger, George Drettakis, and Fredo Durand. Implicit visibility and antiradiance for interactive global illumination. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2007)*, 26(3), August 2007.

[3] Zhao Dong, Jan Kautz, Christian Theobalt, and Hans-Peter Seidel. Interactive global illumination using implicit visibility. In *Proceedings of the Fifteenth Pacific Conference on Computer Graphics and Applications*, pages 77–86. IEEE Computer Society, 2007.

[4] Kyle Hegeman, Simon Premože, Michael Ashikhmin, and George Drettakis. Approximate ambient occlusion for trees. In *I3D '06: Proceedings of the 2006 symposium on Interactive 3D graphics and games*, pages 87–92, New York, NY, USA, 2006. ACM.

[5] Andrey Iones, Anton Krupkin, Mateu Sbert, and Sergey Zhukov. Fast, realistic lighting for video games. *IEEE Computer Graphics and Applications*, 23(3):54–64, May/June 2003.

[6] V. Kajalin. Screen-space ambient occlusion. In *Shader X7*. Wolfgang Engel, Ed., Charles River Media, 2009.

[7] James T. Kajiya. The Rendering Equation. *Computer Graphics (ACM SIGGRAPH '86 Proceedings)*, 20(4):143–150, August 1986.

[8] Anton Kaplanyan. Light propagation volumes in cryengine 3. In *ACM SIGGRAPH 2009 Course Notes - Advances in Real-Time Rendering in 3D Graphics and Games*, 2009.

[9] Janne Kontkanen and Timo Aila. Ambient occlusion for animated characters. In Thomas Akenine-Möller Wolfgang Heidrich, editor, *Rendering Techniques 2006 (Eurographics Symposium on Rendering)*. Eurographics, jun 2006.

[10] Janne Kontkanen and Samuli Laine. Ambient occlusion fields. In *Proceedings of ACM SIGGRAPH 2005 Symposium on Interactive 3D Graphics and Games*, pages 41–48. ACM Press, 2005.

[11] Johannes Kopf, Michael Cohen, Dani Lischinski, and Matt Uyttendaele. Joint bilateral upsampling. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2007)*, 26(3), 2007.

[12] Alex Mendez, Mateu Sbert, and Jordi Cata. Real-time obscurances with color bleeding. In *Proceedings of Spring Conference on Computer Graphics (SCCG 2003)*, Bratislava, Slovakia, April 2003.

[13] Greg Nichols, Jeremy Shopf, and Chris Wyman. Hierarchical image-space radiosity for interactive global illumination. In Hendrik Lensch and Peter-Pike Sloan, editors, *Proceedings of Eurographics Symposium on Rendering 2009*, June 2009.

[14] Greg Nichols and Chris Wyman. Multiresolution splatting for indirect illumination. In *Proc. ACM Symposium on Interactive 3D Graphics and Games 2009 (I3D '09)*, 2009.

[15] Derek Nowrouzezahrai and John Snyder. Fast global illumination on dynamic height fields. In Hendrik Lensch and Peter-Pike Sloan, editors, *Proceedings of Eurographics Symposium on Rendering 2009*, June 2009.

[16] Chris Oat. Irradiance volumes for games. In *Game Developers Conference*, March 2005.

[17] Christopher Oat and Pedro V. Sander. Ambient aperture lighting. In *I3D '07: Proceedings of the 2007 symposium on Interactive 3D graphics and games*, pages 61–64, New York, NY, USA, 2007. ACM.

[18] O. S. Pianykh, J. M. Tyler, and Jr. W. N. Waggenspack. Improved monte carlo form factor integration. *Computers & Graphics*, 22(6):723–734, 1998.

[19] Tobias Ritschel, Thorsten Grosch, Jan Kautz, and Hans-Peter Seidel. Interactive global illumination based on coherent surface shadow maps. In *Proc. Graphics Interface 2008*, pages 185–192. ACM Press, May 2008.

[20] Tobias Ritschel, Thorsten Grosch, and Hans-Peter Seidel. Approximating dynamic global illumination in image space. In *Proc. ACM Symposium on Interactive 3D Graphics and Games 2009 (I3D '09)*, 2009.

[21] Perumaal Shanmugam and Okan Arikan. Hardware accelerated ambient occlusion techniques on GPUs. In *I3D '07: Proceedings of the 2007 symposium on Interactive 3D graphics and games*, pages 73–80, New York, NY, USA, 2007. ACM.

[22] Oles Shishkovtsov. Deferred shading in s.t.a.l.k.e.r. In *GPU Gems 3*, chapter 9. Addison-Wesley, 2005.

[23] Niklas Smedberg and Daniel Wright. Rendering techniques in Gears Of War 2, March 2009. GDC 2009.

[24] John Snyder and Derek Nowrouzezahrai. Fast soft self-shadowing on dynamic height fields. In *Proceedings of Eurographics Symposium on Rendering 2008*, June 2008.

[25] Sergei Zhukov, Andrej Iones, and Grigorij Kronin. An ambient light illumination model. In G. Drettakis and N. Max, editors, *Rendering Techniques '98 (Proceedings of Eurographics Rendering Workshop '98)*, pages 45–56. Springer Wien, 1998.

Centre de recherche INRIA Grenoble – Rhône-Alpes
655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier (France)

Centre de recherche INRIA Futurs : Parc Orsay Université - ZAC des Vignes
4, rue Jacques Monod - 91893 ORSAY Cedex
Centre de recherche INRIA Nancy – Grand Est : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex
Centre de recherche INRIA Rennes – Bretagne Atlantique : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex
Centre de recherche INRIA Paris – Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex
Centre de recherche INRIA Sophia Antipolis – Méditerranée : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399