Graphics Hardware (2006)
M. Olano, P. Slusallek (Editors)

Non-interleaved Deferred Shading of Interleaved Sample Patterns

B. Segovia [1,2], J. C. Iehl[2], R. Mitanchey[1] and B. Péroche[2]

[1] ENTPE: Ecole Nationale des Travaux Publics de l'Etat
[2] LIRIS: Lyon Research Center for Images and Intelligent Information Systems

Abstract

This paper presents a novel and fast technique to combine interleaved sampling and deferred shading on a GPU. The core idea of this paper is quite simple. Interleaved sample patterns are computed in a non-interleaved deferred shading process. The geometric buffer (G-buffer) which contains all of the pixel information is actually split into several separate and distinct sub-buffers. To achieve such a result in a fast way, a massive two-pass swizzling copy is used to convert between these two buffer organizations. Once split, the sub-buffers can then be accessed to perform any fragment operation as it is done with a standard deferred shading rendering pipeline. By combining interleaved sampling and deferred shading, real time rendering of global illumination effects can be therefore easily achieved. Instead of evaluating each light contribution on the whole geometric buffer, each shading computation is coherently restricted to a smaller subset a fragments using the sub-buffers. Therefore, each screen pixel in a regular $n \times m$ pattern will have its own small set of light contributions. Doing so, the consumed fillrate is considerably decreased and the provided rendering quality remains close to the quality obtained with a non-interleaved approach. The implementation of this rendering pipeline is finally straightforward and it can be easily integrated in any existing real-time rendering package already using deferred shading.

1. Introduction

Major improvements in graphics hardware have been recently achieved. A commodity GPU now offers programming capabilities, a very high fillrate and a large memory bandwidth. Unfortunately, the current architecture of the GPUs requires specific cares and many classical and efficient CPU techniques must often be completely re-designed to remain fast with a GPU. In this paper, we present a GPU-friendly method to perform interleaved sampling i.e. an efficient way to achieve uncorrelated computations on nearby pixels. The principle of our algorithm is quite simple. Instead of performing every computation for each pixel, we propose to perform all shading operations on interleaved and separate tiled sub-buffers. These sub-buffers are actually built by splitting an initial buffer. Formally, the texels inside a $n \times m$ regular pattern are dispatched over different regions of the screen, the separate sub-buffers. Hence, texel (x,y) will go to texel $(x/n, y/m)$ belonging to sub-buffer $(i,\ j)$ with $i = x \bmod n$ and $j = y \bmod m$. Combined with an extended deferred shading rendering pipeline, our interleaved sampling technique can easily be used to perform real time global illumination effects on a single commodity computer. In comparison with a brute force approach, the resulting performance is thus improved by more than an order of magnitude.

(a) (b)

Figure 1: ***The Indirect Lighting Impact*** *(a) presents a snapshot of Q3tourney2 (courtesy of Id Software) with direct lighting only (two point light sources). The screen resolution is* 1024×768*. (b) The same scene with 512 secondary point light sources. Using interleaved deferred shading, the scene is rendered at 17 f/s on a GeForce 6800 GT and 33 f/s on a NVidia GeForce 7800 GT.*

The remainder of the paper is organized as follows. Section 2 sums up the related work and our motivation. Section 3 presents our core technique i.e. the buffer splitting strategy used to perform interleaved sampling and all following shading operations. Section 4 shows how the standard deferred shading rendering pipeline can be greatly accelerated by combining it with the buffer splitting technique. Sections 5 and 6 are devoted to the results obtained for every step of the pipeline and two different kinds of applications. A conclusion and possible future work are finally given in Section 7.

2. Previous Work

Since Kajiya formalized the light transport problem with the rendering equation [Kaj86], many sampling methods have been proposed. Cook et al. first presented several Monte-Carlo strategies using camera path samples and incoherent sampling from one pixel to the other [CPC84] [Coo86]. More recently, Keller and Heidrich proposed a simple interleaved sampling strategy [KH01] quite suitable to graphics hardware. First, interleaved samples are taken from a number of independent regular grids which are then merged into a single sampling pattern. The samples of all regular grids are thus interleaved such that in the final high-quality image, adjacent pixels are not correlated.

This sampling technique was successfully used by Ingo Wald et al. to perform interactive global illumination [WKB*02]. First, the whole incoming radiance field is replaced and represented by a set of hemispherical virtual point lights (commonly called *VPLs*) computed with the Instant Radiosity algorithm [Kel97]. Then, all visibility requests are computed with an efficient and parallelized ray-tracer [WBWS01] [WSBW01]. To perform the integration with limited computation capabilities, the *VPL* contributions are interleaved. Therefore, not every pixel computes every lighting sample since each pixel in a regular pattern (3×3 for example) uses a different light sample set. The *VPL* contributions are then filtered using a discontinuity buffer. With a cluster of commodity computers, they achieved interactive frame rates on highly complex scenes. Unfortunately, ray-tracing is not currently supported by specific graphics hardware and even if significant advances were recently achieved in this domain [SWS02] [SWS05], commodity "RPUs" are not currently available.

This motivates a GPU approach. Indeed, major improvements in the domain of graphics hardware considerably accelerated many algorithms which used to be considered slow. Deferred shading first introduced by Deering et al. [DWS*88] then developed by Saito and Takahashi [ST90] is certainly one of them. It suggests storing the geometric and material information in buffers (generally called "G-buffer") and reusing them to perform the lighting computations. It greatly simplifies the rendering pipeline and it also prevents the geometry from being reprojected each time a shading pass is performed. Unfortunately, achieving real time global illumination without noticeable artifacts and therefore with a large set of virtual point light sources requires too large computation capabilities. That is why Dachsbacher and Stamminger recently presented several techniques [DS06] to limit shading computations while integrating the incoming radiance field. They precisely bound all computations by ellipsoids surrounding the light sources and fix their size proportionally to the light source power. Nevertheless, too tight bounding volumes can lead to noticeable artifacts while too large ones strongly limit the frame rate. Simply sub-sampling the screen space also leads to visible problems since filtering the irradiance as a post-process becomes quite difficult.

These issues motivated our approach. We tried to set up a GPU-friendly way to perform interleaved sampling and therefore to achieve real time rendering of global illumination effects by combining it with a deferred shading process.

3. GPU-friendly Interleaved Sampling

The goal of this section is to limit computations to separate and distinct fragment subsets. In a standard deferred shading application, since a light source will illuminate *all* screen pixels, a large fillrate will be consumed if many light sources are present in the scene. With interleaved sampling and a $n \times m$ interleaved pattern, each pixel inside a $n \times m$ rectangle will have its own set of light contributions and no correlation between adjacent pixels occur. By exploiting the coherence of neighbor pixels and blending their irradiance, the rendering quality can therefore remain close to the quality provided by non-interleaved methods with a performance equivalent to the performance obtained with a sub-sampling technique. This section introduces the shortcomings of several basic ideas to perform interleaved sampling and also presents a more GPU-friendly approach. To compare all the tested approaches, we will assume that a geometric buffer containing the normal, the position and the material information of each pixel has been first created and stored.

3.1. Basic Approaches

The obvious way to accomplish interleaved sampling is to render $n \times m$ passes by using a stencil buffer for each of them so that only one pixel in each $n \times m$ cell should be rendered. This approach was tested and as shown in Table 1, it gave very poor results. Indeed, even if early fragment culling is activated, the incoherence of the input data prevents the GPU from being efficient. Several variants were also explored. First, stencil tests were replaced by depth tests in order to exploit early-z culling capabilities. However, as the hierarchical Z-buffer algorithm [GKM93] commonly used by the GPUs also requires the coherence of the depth data, the method was inefficient too. We secondly tested dynamic branching but the incoherence of the data once again caused

	4×4	2×2
No Interleaving	75 ms	75 ms
Stencil Culling	73 ms	72 ms
Depth Culling	72 ms	74 ms
Dynamic Branching	78 ms	82 ms

Table 1: ***Time to accumulate 16 point light contributions*** *The G-buffer resolution is equal to* 1024×1024*. Stencil or depth culling and dynamic branching give very poor results. The incoherence of the data inside a* 2×2 *or* 4×4 *pattern strongly limits the performance.*

bad results. We finally tried to directly create the tiled and separate sub-buffers presented in Figure 3.c by accessing the G-buffer data with an indirection during *every* shading pass. Since the texture accesses are very incoherent, the application becomes utterly bound by the memory latency and considerably slows down. These failures motivated more sophisticated techniques.

3.2. One-pass G-Buffer Splitting

To provide the coherence needed by the GPU, explicitly splitting the G-buffer $\mathcal{G}$ into $n \times m$ smaller tiled sub-buffers $\mathcal{G}_{i,j}$ seems a good idea since operations for each light source can be performed on contiguous groups of pixels. To understand this simple concept, Figure 3.b gives an example of a split normal buffer. Hence, texel (x,y) from $\mathcal{G}$ goes to texel $(x/n, y/m)$ belonging to sub-buffer $\mathcal{G}_{i,\,j}$ with $i = x \bmod n$ and $j = y \bmod m$. The split can be done in a single pass with a specific fragment program. A look-up texture is first precomputed and used to move a texel of the initial G-buffer to the associated texel of the split G-buffer. Using one pass to split the buffer is unfortunately slow since memory accesses remain strongly incoherent during the splitting pass. Fortunately, the approach can be easily accelerated.

3.3. Two-pass G-Buffer Splitting

The memory organization on GPUs requires special care. Indeed, like CPUs, GPUs are all the more efficient that memory and cache accesses are coherent. To provide coherence during any mapping operation, the textures are 2D block allocated by the graphics card drivers. It motivated a coherent two-pass approach.

Block Splitting: The memory and cache access incoherence during the one-pass splitting is mainly caused by the size of the manipulated data. A simple idea is therefore to limit splitting operations to small 2D blocks to enhance data locality. The initial G-buffer $\mathcal{G}$ is thus subdivided into $p \times q$ blocks and each block is split into $n \times m$ separate sub-blocks. If the blocks are small enough, memory accesses remain coherent during the pass. After this pass, each sub-buffer is subdivided into $p \times q$ sub-blocks spread across the whole buffer as shown by Figure 2.c.

Block Translation: To rebuild each sub-buffer, another pass performs the translation of the interleaved sub-blocks (See Figure 2.c and 2.d). Once again, the memory accesses remain coherent since entire blocks are moved.

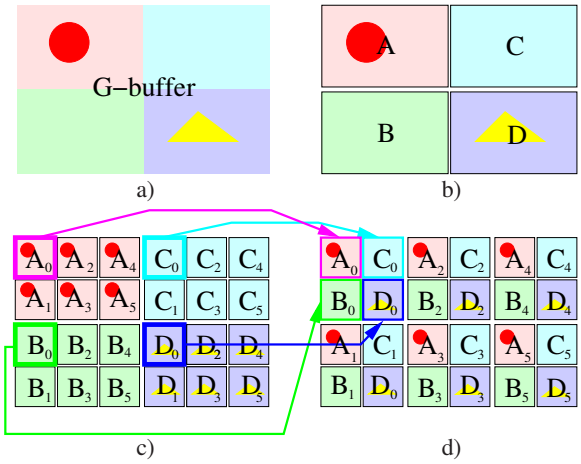


Figure 2: ***Two-pass G-buffer splitting*** *The desired interleaved pattern is* 3×2*. (a) presents the initial G-buffer. (b) The G-buffer is subdivided in 4* (2×2) *blocks. The interleaved pattern size is* 3×2*. (c) shows the G-buffer after the block splitting. As the interleaved pattern is* 3×2*, each block has been split in 6* (3×2) *sub-blocks. The wanted sub-buffers* $(A_iB_iC_iD_i)$ *are therefore spread across the whole buffer. (d) shows how the sub-buffers* $(A_iB_iC_iD_i)$ *are retrieved by the block translation.*

As shown in Section 5, this approach gives satisfactory results. In most of the cases, performing block manipulations is much more efficient than the one-pass naive approach.

3.4. Buffer Gathering

Once the shading operations have been achieved (See Section 4.3), the interleaved pattern is reconstructed by gathering the sub-buffers. This pass is the opposite of the buffer splitting one (see Figure 3.d) and for the same reasons, it is performed in two passes, the block translation pass and the block gathering one.

With the buffer splitting approach, uncorrelated light contributions or more generally uncorrelated computations can be made for nearby pixels in a way as generic as the one proposed by a standard deferred shading approach. In the next sections, we will show that computing and accumulating hundreds of light contributions is now possible with a decent framerate and a good rendering quality by combining buffer splitting with deferred shading and fast filtering techniques.

4. Non-interleaved Deferred Shading of Interleaved Sample Patterns

We now present an extension of deferred shading using the buffer splitting/gathering techniques. Instead of performing the shading operations on the whole G-buffer, they are restricted to low-resolution tiled sub-buffers. In comparison with standard deferred shading, three passes are added. The first one splits the G-buffer in several sub-buffers. The second one reconstructs the interleaved sampling pattern after the shading passes and the third one exploits the spatial coherence of neighbor pixels to blend uncorrelated lighting contributions. Therefore, the rendering pipeline is now decomposed into 5 steps all presented in Figure 3.

4.1. G-Buffer Creation (BC)

Before the shading operations, three float buffers (the G-buffer) which respectively contain positions, normals and colors are first created (see Figure **??**). The material information such as material identifiers is also packed in the remaining components. For bandwidth reasons, precision is limited to 16 bits and the scene is therefore bounded.

4.2. G-buffer Splitting (BS)

The initial G-buffer is split into separate sub-buffers. Two look-up textures are first computed. The first one stores the block splitting function presented in Section 3.3 while the second one stores the block translation function. Then, two fragment programs successively execute the two functions. Once it is done, the sub-buffers are tiled in a buffer with the same size than the initial one (see Figure **??**).

4.3. Shading Computations (SSM / SNSM)

Different lighting contributions are computed for each sub-buffer. Any operation possible with deferred shading is still available. Indeed, as the former G-buffer is explicitly split into smaller sub-buffers, any deferred shader can also be used by focusing the viewport or drawing a quad on a given sub-buffer. A tile of small irradiance sub-buffers is then obtained (see Figure 3.c). It may be noticed that the visibility of hemispherical (resp. spherical) point light sources is solved by unrolling the hemicube (resp. the cube) in a standard shadow map [Wil78] and reindexing it with a small cube map as described in [KN04]. Depending on the application (see Section 6), two shading techniques can finally be used: SSM (Shading with a Shadow Map) is a shading pass with a shadow map reprojection and SNSM (Shading with No Shadow Map) is a shading pass with no shadow map reprojection (visibility is ignored). In both cases, glossy and diffuse BRDFs are handled.

4.4. Buffer Gathering (BG)

Once all light contributions have been accumulated, the irradiance buffer is rebuilt by two fragment programs which successively perform the two passes of the buffer gathering technique described in Section 3.4.

4.5. Filtering (F)

To maintain interactive or real time rendering, few light contributions per pixel can be computed. If a filter is applied on continuous zones of the screen, the geometric coherence of the scene can be exploited to virtually compute many light contributions per pixel.

Discontinuity Buffering (DB): A discontinuity buffer is first computed. Two discontinuity thresholds (respectively on normals and positions) are initially fixed. Then, a fragment program reads the normal and the position buffers and decides if the current pixel is on a discontinuity or not by evaluating an arbitrary measure between the current fragment (x_0, y_0) and its three "positive" neighbors (x_0+1, y_0), (x_0, y_0+1) and (x_0+1, y_0+1). Similar techniques are described with more details by Simmons and Séquin in [SS00].

Gaussian Blurring (B): The discontinuity buffer is used to apply a two-pass separable Gaussian blur on continuous zones of the screen. Two cases can occur.

- The current fragment (x_0, y_0) is not on a discontinuity. The +x (resp. +y) and -x (resp -y) directions are explored one after the other. Neighbor pixels are blended to the current pixel until a discontinuity is encountered in the given direction or the filter kernel size is reached.
- The current fragment (x_0, y_0) is on a discontinuity. Since either (x_0+1, y_0) or (x_0, y_0+1) or (x_0+1, y_0+1) is not similar to (x_0, y_0), only the -x (resp. -y) direction is explored. A neighbor pixel is once again blended to the current pixel until a discontinuity is encountered.

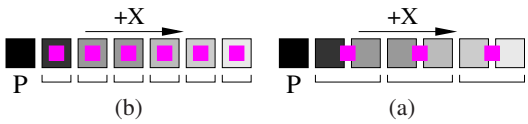


Figure 4: ***Gaussian Blur*** *(a) The standard approach. For a given direction, all texture accesses are made at texel center. (b) With hardware-supported bilinear filtering. All texture accesses are made between two texels. Bilinear filtering blends their contributions. One texture access is therefore sufficient for two texels.*

To speed up the filtering pass, we modify the standard Gaussian blur by using hardware-supported filtering (see Figure 4). Instead of fetching data at the center of each texel, a 0.5 bias is applied and combined with hardware-supported bilinear filtering. As extra artifacts can occur on discontinuities with the fast filtering approach, the accesses to the discontinuity buffer are also modified by activating the 0.5 bias and the bilinear filtering. The texel contributions are then ignored

(a) (b) (c) (d) (e) (f) (g)

Figure 3: ***Non-interleaved Deferred Shading of Interleaved Sample Patterns*** *(a) G-buffer Creation. (b) G-buffer Splitting. The G-buffer is subdivided into $n \times m$ separate smaller sub-buffers. Each sub-buffer contains a distinct texel subset. (c) Different shading computations are performed on every sub-buffer. (d) G-buffer Gathering. The irradiance buffer is computed by gathering all the irradiance sub-buffers. The resulting interleaved pattern may be noticed. (e) A discontinuity buffer is computed. (f) A Gaussian blur is applied on the irradiance buffer and the discontinuity buffer is used to prevent non-neighbor texels from being filtered. (g) The self-colors of the objects and the irradiance buffer are finally blended.*

as soon as the discontinuity texel value is greater than 0. In comparison with the first approach, this criteria is *sufficient*.

To prevent the object colors from being filtered, the self-colors of the objects are blended only after applying the filter. As shown in Section 5, combining discontinuity buffering and interleaved sampling provides high quality filters with very few visible artifacts.

4.6. Remarks

Two important remarks may be made. First, the discontinuity buffer is already computed by many deferred shading engines since it helps to perform antialiasing. Secondly, the technique presented in this section can also be considered as a Level-Of-Detail algorithm. If a $2^n \times 2^m$ (low details) interleaved pattern has been computed, $2^i \times 2^j$ interleaved patterns (higher details with $i \leq n$ and $j \leq m$) can also be performed by clustering several sub-buffers. Depending on the application, it is therefore easy to interleave few direct point lights and many secondary ones with only one interleaved pattern.

5. Results

We present in this section the results obtained for every rendering pass and the impact of all technical choices presented in the previous sections.

5.1. Buffer Splitting Results

We analyzed the performance of the buffer splitting approach with or without block manipulations. Several sizes of blocks were tested. Figure 5 presents some results obtained on a GeForce 6800 GT. The performance greatly varies with the size of the blocks and the interleaving sampling pattern chosen. For a 2×2 interleaved sampling, buffer splitting with 32×32 blocks is the most efficient method (only 5% faster than the naive approach). For a 4×4 interleaved sampling, buffer splitting with 16×16 blocks is 70% faster than

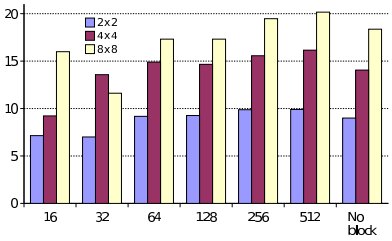


Figure 5: ***Buffer Splitting Results*** *The size of the G-buffer is equal to* 1024×1024*. The x axis gives the size of the blocks and the y axis gives the computation time in milliseconds. Different interleaved sampling patterns are presented (*2×2*,* 4×4 *and* 8×8*)*

the one-pass approach. Other tests were made. For larger resolutions such as 1280×1024 and large interleaved sampling patterns, a finely tuned two-pass approach is more than three times as fast as the naive one. Unfortunately, the choice of the block size strongly depends on the organization of the memory and caches on the card. That is why the optimal configuration must often be empirically chosen for each screen resolution and each regular pattern size.

5.2. Shading Results

The shading performance depends on the type of applications but it is identical to the one obtained with standard deferred shading methods. All details are given in Section 6.

5.3. Buffer Gathering Results

The performance is quite similar to the one presented in Section 5.1 but it may be noticed that only one three-component 16 bit float buffer has to be gathered since only the irradiance buffer is rebuilt. This pass is therefore about 40% faster than the splitting one.

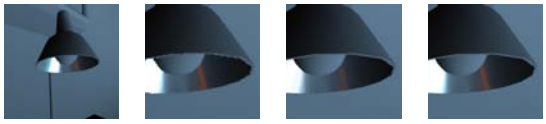
	Normal Gaussian Blur	Fast Gaussian Blur
7×7	16.9 ms	10.9 ms
11×11	29.7 ms	16.9 ms
17×17	51.8 ms	32.7 ms

Table 2: ***Filtering Results:*** *Times to perform a Gaussian blur on the picture. Two techniques are presented, with and without hardware supported bilinear filtering. The screen resolution is equal to* 1280×1024*. The GPU used for the tests is a NVidia GeForce 6800GT.*

5.4. Discontinuity Buffering and Filtering Results

Discontinuity Buffer: Computing discontinuities in a fragment program does not strongly limit our application since it is much less expensive than shading computations. Nevertheless, handling discontinuities during the Gaussian blurring is a bit more difficult and expensive.

Filtering: To stop blending texels beyond a discontinuity, accesses to the irradiance buffer are sequentially done along a direction. As soon as a discontinuity is encountered, a flag is set to 0 to ignore the next texel contributions. It can be noticed that dynamic branching can also be used to stop blending texels beyond a discontinuity. This approach was tested and does not provide any significant performance enhancement (because the blurring operation is most of the time performed on the whole kernel).

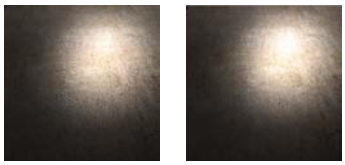


(a) (b) (c) (d)

Figure 6: ***Rendering Quality*** *(a) the* 1024×1024 *reference image computed with 512 point lights and no interleaved sampling. (b) close-up on a lamp with a simple sub-sampling approach. (c)* 4×4 *interleaved sampling with no filter. (d)* 4×4 *interleaved sampling + filtering*

Table 2 shows results obtained with both standard and hardware-supported approaches. Using float buffer filtering capabilities provides a speed-up equal to 60 % without visible differences.

Quality: Figure 6 presents some results obtained with sub-sampling and interleaved sampling methods (self-colors of objects are not blended). Even with high-variance estimators due to glossy reflections, interleaved sampling combined with a discontinuity buffer and our fast Gaussian blur provides good results. Sub-sampling does not handle high-frequencies details and it is much more difficult to deal with discontinuities. As shown in picture 7, details brought by a normal map or a finely tessellated model disappear with sub-sampling whereas interleaved sampling and fast Gaussian filtering still provide satisfactory results.



(a) (b)

Figure 7: *The surface details brought by a normal map are conserved with interleaved sampling and our fast Gaussian blur as shown in (a). They disappear with sub-sampling as shown in (b).*

6. Applications

After analyzing each computation pass of the algorithm, we now present the overall performance obtained with two different kinds of applications. For both applications, a set of virtual point light sources is first generated with the sampling strategy provided by Instant Radiosity [Kel97].

6.1. Implementation Details

To handle visibility problems, several classical algorithms were implemented. To speed up the visibility requests, the geometry is first segmented with a kd-tree. Then, frustum culling is performed to eliminate the unseen leaves of the tree. A *PVS* or Potential Visible Set is finally computed for the whole tree to cull leaves not seen by a given leaf. The rasterization code was written using the OpenGL 2.0 API and the ray tracing requests are done by using a finely tuned kd-tree.

6.2. Fully Interactive Applications

Using our extension of the deferred shading pipeline with real time applications (like video games) is straightforward but requires specific approximations. An interactive application needs a decent frame rate and computing all visibility requests for every point light is very expensive. The first approximation which has to be done is thus to ignore visibility for secondary light sources. Secondly, the variance is all the larger that the number of bounces increases. To efficiently handle this problem, only one-bounce indirect illumination will be taken into account. It is generally visually sufficient.

Table 3 sums up the computation times of all passes with two GPUs of different generations, a NVidia GeForce 6800 GT and a NVidia GeForce 7800 GT. With standard deferred shading, BS, BG and F passes are not executed. Only one or several shading passes are thus performed. It is therefore easy to analyze the extra costs due to interleaved sampling.

	BC	BS	SSM	SNSM	BG	B
6800GT	10.9	10.7	4.9	4.7	4.5	7.3
7800GT	5.2	5.8	2.7	2.6	2.3	3.9

Table 3: ***Overview of the performance*** *The results are given in milliseconds per million of fragments. Times for the G-buffer creation pass are given with a scene of Quake 3 (courtesy of Id Software) counting 250000 triangles. All acronyms are given in Section 4. The interleaved pattern size is* 4×4 *and the kernel size of the Gaussian filter is* 5×5.

	IS	no IS	speed-up
Q3dm11	36 f/s	1.2 f/s	$\times 31$
Dragon in Q3t1	29 f/s	1.2 f/s	$\times 25$
Buddha in Q3dm12	36 f/s	1.2 f/s	$\times 30$

Table 4: ***The performance with or without interleaved sampling on a 7800 GT*** *The screen resolution is* 1024×768. *The interleaved pattern size is* 8×6. *480 point light sources are accumulated. With standard deferred shading, the situation is intractable. If it is combined with interleaved sampling, real time frame rate is achieved without too noticeable artifacts.*

For a NVidia 6800GT, the extra time taken by the interleaved sampling extension for one million of pixels is equal to 22.5 ms. It is roughly equivalent to 4.5 shading passes. Therefore, if interleaved sampling saves more than 4.5 million shading computations, it becomes competitive. For example, with our implementation and a 4×4 interleaved pattern, computing the contributions of 16 or more lights is already more efficient with interleaved deferred shading. Furthermore, our approach seems to be adapted to current graphics hardware improvements. With roughly the same memory bandwidth, the GeForce 7800 GT is roughly twice faster than the GeForce 6800 GT. Therefore, even with large memory manipulations, the memory latency and bandwidth do not strongly limit our approach and block splitting remains efficient. Table 4 gives frame rates obtained with typical scenes used in video games. Without interleaved sampling, real time frame rate cannot be achieved. By extending the rendering pipeline with buffer splitting, the performance is improved by more than an order of magnitude without major visible artifacts as shown in Figure 9.

6.3. Physically Based Rendering

Physically based rendering requires unbiased approaches. The Instant Radiosity sampling strategy first gives a set of virtual point lights (*VPLs*). For every *VPL*, a high quality shadow map is computed. It is then reprojected into a given sub-buffer. As visibility is solved for every *VPL*, the bottleneck of the algorithm can be either the shading computations or the shadow map computations. Table 5 gives numerical results obtained for scenes with different geometric complexities. Hence, due to their linear algorithmic complexities, the shadow maps become very expensive for large scenes. With interleaved sampling, the shading problem can be however efficiently handled. If the point of view and the scene do not change, the G-buffer (split or not) does not have to be recomputed. Therefore, if many *VPL* contributions are accumulated, the G-buffer creation, the manipulation passes and the filters become free. That is why our approach remains most of the time much faster than deferred shading with a speed-up roughly varying from the interleaved pattern size like 2^2, 4^2 or 8^2 (for very simple scenes) to 1 (for very large scenes). Figure 8 gives several images obtained with this approach.

	Shadow Map	No IS	2 x 2 IS	Speed Up
Office	2.2 ms	2.4 s	0.7 s	$\times 3.4$
Theater	4 ms	2.8 s	1.0 s	$\times 2.8$
Conference	8 ms	3.8 s	2.2 s	$\times 1.7$
Cruiser	15 ms	5.9 s	4.0 s	$\times 1.5$

Table 5: *Times to obtain less than 1% RMS error with a NVidia GeForce 6800 GT. The office contains 31000 triangles, the Candlestick theater 100000 triangles, the conference room 228000 triangles and the cruiser one million triangles.*

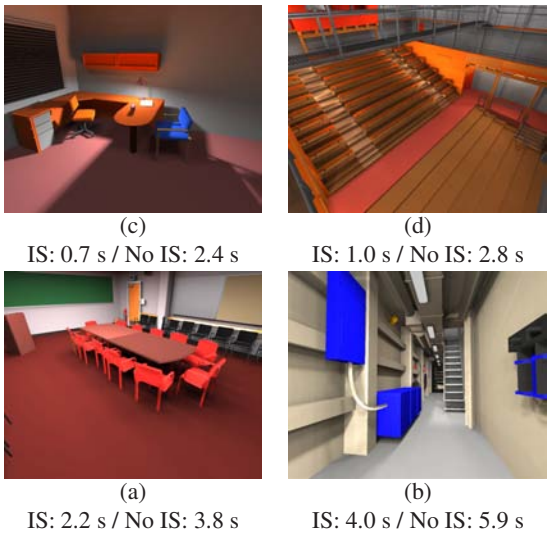
7. Conclusion and Future Work

In this paper, we presented an efficient and new way to perform interleaved sampling with today's graphics hardware and a novel and conservative extension of deferred shading. Instead of performing the shading computations with the whole geometric buffer, they are made with small, separate and interleaved sub-buffers. Doing so, the necessary fillrate is strongly limited and hundreds of light sources can be accumulated in real time. By exploiting the pixel coherence, the rendering quality remains very close to the quality obtained with a brute-force approach. Furthermore, all operations available with deferred shading remain available with our method and the algorithm can easily be integrated in any existing real-time rendering packages already using a deferred shading techniques.

The major step in improving the method is certainly to solve the visibility problems in a better way. Shadow maps like shadow volumes have a linear complexity in the relation of the number of triangles. An interesting method would be to perform the shadowing computations by ignoring the visibility during the shading passes but using a set of positive and *negative* virtual point lights.

8. Acknowledgments

We would like to thank the anonymous reviewers for their very valuable comments about the presentation issues and



(c) IS: 0.7 s / No IS: 2.4 s

(d) IS: 1.0 s / No IS: 2.8 s

(a) IS: 2.2 s / No IS: 3.8 s

(b) IS: 4.0 s / No IS: 5.9 s

Figure 8: ***Physically Based Rendering*** *Pictures and convergence times obtained with less than 1% RMS error. Results are given with or without interleaved sampling. The screen resolution is* 1280×1024 *and the interleaved sampling pattern is* 2×2*. (The theoretical maximum speed-up is therefore equal to 4) (a) The office. 35 000 triangles are rendered. (b) The Candlestick Theater with 100 000 triangles. (c) The conference room. 200 000 triangles are rendered. (d) The cruiser. One million triangles are displayed.*

pointing out the missing references. The Cruiser, the Office, the Conference Room and the Candlestick theater are courtesy of Greg Ward and can be found on the RADIANCE web site. Q3tourney1, Q3tourney2, Q3dm11 and Q3dm12 are courtesy of ID Software.

References

[Coo86] COOK R. L.: Stochastic sampling in computer graphics. In *ACM Transaction on Graphics* (1986), pp. 51–72.

[CPC84] COOK R. L., PORTER T., CARPENTER L.: Distributed ray tracing. In *SIGGRAPH '84 (Proceedings of the 11th annual conference on Computer graphics and interactive techniques)* (1984), pp. 137–145.

[DS06] DACHSBACHER C., STAMMINGER M.: Splatting indirect illumination. In *SI3D '06 (Proceedings of the 2006 symposium on Interactive 3D graphics and games)* (2006), pp. 93–100.

[DWS*88] DEERING M., WINNER S., SCHEDIWY B., DUFFY C., HUNT N.: The triangle processor and normal vector shader: a vlsi system for high performance graphics. In *SIGGRAPH '88 (Proceedings of the 15th annual conference on Computer graphics and interactive techniques)* (1988), pp. 21–30.

[GKM93] GREENE N., KASS M., MILLER G.: Hierarchical z-buffer visibility. In *SIGGRAPH '93 (Proceedings of the 20th annual conference on Computer graphics and interactive techniques)* (1993), pp. 231–238.

[Kaj86] KAJIYA J. T.: The rendering equation. In *SIGGRAPH '86 (Proceedings of the 13th annual conference on Computer graphics and interactive techniques)* (1986), pp. 143–150.

[Kel97] KELLER A.: Instant radiosity. In *SIGGRAPH '97 (Proceedings of the 24th annual conference on Computer graphics and interactive techniques)* (1997), pp. 49–56.

[KH01] KELLER A., HEIDRICH W.: Interleaved sampling. In *Proceedings of the 12th Eurographics Workshop on Rendering* (2001).

[KN04] KING G., NEWHALL W.: Efficient omnidirectional shadow maps. In *ShaderX3* (2004), pp. 435–448.

[SS00] SIMMONS M., SÉQUIN C. H.: Tapestry: A dynamic mesh-based display representation for interactive rendering. In *Proceedings of the 11th Eurographics Workshop on Rendering* (2000), pp. 329–340.

[ST90] SAITO T., TAKAHASHI T.: Comprehensible rendering of 3-d shapes. In *SIGGRAPH '90 (Proceedings of the 17th annual conference on Computer graphics and interactive techniques)* (1990), pp. 197–206.

[SWS02] SCHMITTLER J., WALD I., SLUSALLEK P.: Saarcor – a hardware architecture for ray tracing. In *Proceedings of the conference on Graphics Hardware 2002* (2002), pp. 27–36.

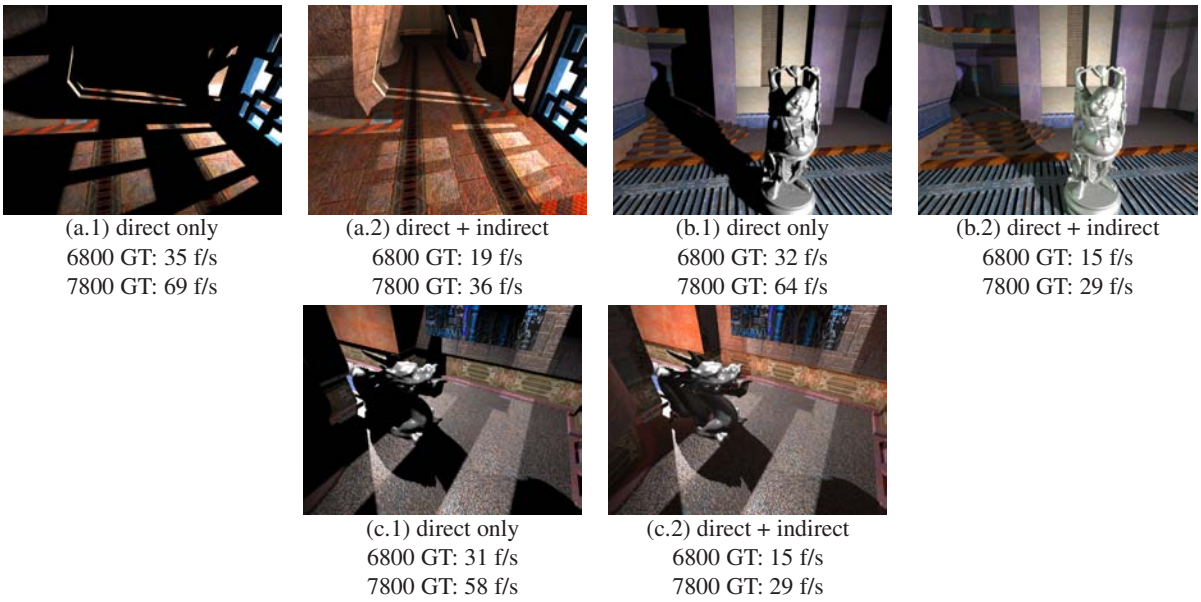
[SWS05] SVEN WOOP J. S., SLUSALLEK P.: Rpu: A programmable ray processing unit for realtime ray tracing. In *SIGGRAPH '05 (Proceedings of the 22nd annual conference on Computer graphics and interactive techniques)* (2005), pp. 434–444.

[WBWS01] WALD I., BENTHIN C., WAGNER M., SLUSALLEK P.: Interactive rendering with coherent ray tracing. *Computer Graphics Forum (Proceedings of Eurographics 2001)* (2001).

[Wil78] WILLIAMS L.: Casting curved shadows on curved surfaces. In *SIGGRAPH '78 (Proceedings of the 5th annual conference on Computer graphics and interactive techniques)* (1978), pp. 270–274.

[WKB*02] WALD I., KOLLIG T., BENTHIN C., KELLER A., SLUSALLEK P.: Interactive global illumination using fast ray tracing. In *Proceedings of the 13th Eurographics Workshop on Rendering* (2002).

[WSBW01] WALD I., SLUSALLEK P., BENTHIN C., WAGNER M.: Interactive distributed ray tracing of highly complex models. In *Proceedings of the 12th Eurographics Workshop on Rendering Techniques* (2001), pp. 277–288.



(a.1) direct only
6800 GT: 35 f/s
7800 GT: 69 f/s

(a.2) direct + indirect
6800 GT: 19 f/s
7800 GT: 36 f/s

(b.1) direct only
6800 GT: 32 f/s
7800 GT: 64 f/s

(b.2) direct + indirect
6800 GT: 15 f/s
7800 GT: 29 f/s

(c.1) direct only
6800 GT: 31 f/s
7800 GT: 58 f/s

(c.2) direct + indirect
6800 GT: 15 f/s
7800 GT: 29 f/s

Figure 9: ***Fully Interactive Applications*** *The screen resolution is* 1024×768*. The interleaved pattern size is* 8×6*. Visibility for secondary light sources is ignored. All scenes count 480 secondary VPLs. Frame rates obtained for direct contributions only (with a standard deferred shading method) are also given. (a) Q3dm11 (about 80 000 triangles) lit by one hemispherical point light source. (b) Buddha in Q3dm12 lit by one hemispherical point light source. About 200 000 triangles are rendered. (c) Dragon in Q3tourney1 lit by two hemispherical point light sources. About 150 000 triangles are rendered in this scene.*