Forward+: Bringing Deferred Lighting to the Next Level

Takahiro Harada, Jay McKee, and Jason C.Yang

Advanced Micro Devices, Inc.

Abstract

This paper presents Forward+, a method of rendering many lights by culling and storing only lights that contribute to the pixel. Forward+ is an extension to traditional forward rendering. Light culling, implemented using the compute capability of the GPU, is added to the pipeline to create lists of lights; that list is passed to the final rendering shader, which can access all information about the lights. Although Forward+ increases workload to the final shader, it theoretically requires less memory traffic compared to compute-based deferred lighting. Furthermore, it removes the major drawback of deferred techniques, which is a restriction of materials and lighting models. Experiments are performed to compare the performance of Forward+ and deferred lighting.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Color, shading, shadowing, and texture

1. Introduction

In recent years, deferred rendering has gained in popularity for rendering in real time, especially in games. The major advantages of deferred techniques are the ability to use many lights, decoupling of lighting from geometry complexity, and manageable shader combinations. However, deferred techniques have disadvantages such as limited material variety, higher memory and bandwidth requirements, handling of transparent objects, and lack of hardware anti-aliasing support [Kap10]. Material variety is critical to achieving realistic shading results, which is not a problem for forward rendering. However, forward rendering normally requires setting a small fixed number of lights to limit the potential explosion of shader permutations and needs CPU management of the lights and objects. Also, with expensive dynamic branching performance on current consoles (e.g., XBox 360) it is understandable why deferred rendering has become appealing.

The latest GPUs have improved performance, more ALU power and flexibility, and the ability to perform general computation – in contrast to current consoles. Thus, rendering with many lights with forward rendering could be a realistic option, however the naïve approach of iterating through every light in a per-pixel shading fashion is impractical.

We present Forward+: a method of rendering with many lights by culling and storing only lights that contribute to the pixel. The lights are evaluated one by one in the final shader. In this manner, we retain all the positive aspects of forward rendering and gain the ability to render with lots of lights.

Figure 1: *A screenshot from the AMD Leo demo using Forward+.*

This paper first presents the pipeline and gives a high level explanation of the implementation. The theoretical memory traffic of Forward+ is compared to deferred lighting.

2. Related Work

Forward rendering has practical limitations on the number of lights that can be used when shading [AMHH08]. Deferred techniques overcome this issue and have been gaining popularity, especially on consoles. Deferred renderers that exports G-buffers storing all geometry information for the screen-space lighting stresses the memory system. Deferred lighting decouples lighting calculation and material evaluation to reduce the memory traffic. Andersson [And11] implemented a deferred-lighting system using the compute capability of the GPU. Theoretically, the method can achieve the best performance among the existing deferred techniques by its reduction of memory traffic.

Light-indexed deferred lighting takes a slightly different approach by storing light indices instead of accumulating lighting components [Tre09]. It reduces the cost of the G-buffer export, but needs to read more data during the final shading than a deferred-lighting technique. This method has several restrictions: for example, the number of lights per pixel has a predefined maximum capacity. The proposed method, Forward+, shares this method's concept without its restriction and reduces further memory traffic by using the flexible compute capability of modern GPUs.

3. Method

Forward+ extends a forward-rendering pipeline by adding only a light-culling stage before final shading. The pipeline consists of three stages: depth prepass, light culling, and final shading. Another modification is for the data structure of lights, which has to be stored in a linear buffer accessible from shaders for light culling and final shading. Depth prepass is an option for forward rendering, but it is essential for Forward+ to reduce the pixel overdraws of the final shading step, which is especially expensive for Forward+. If depth prepass is compared to G-prepass used in deferred techniques, in which full-screen geometry information is exported, the forward-render depth prepass is cheaper because it populates only the depth buffer.

3.1. Light Culling

The light-culling stage is similar to the light-accumulation step of deferred lighting. Instead of calculating lighting components, light culling calculates a list of light indices overlapping a pixel. The list of lights can be calculated for each pixel, which is a better choice for final shading, but there are reasons it is not efficient if we look at the efficiency of the entire rendering pipeline. The most important issues are memory footprint and the efficiency of the computation at the light-culling stage. Therefore, the screen is split into tiles and light indices are calculated on a per-tile basis. Although tiling can add false positives to the list for pixels in a tile, it drastically reduces the memory footprint and computation. The memory size for the light-index buffer and efficiency of the final shader is a trade-off. By utilizing the compute capability of the modern GPUs, light culling can be implemented entirely on the GPU as detailed in Sec. 4. Thus, the whole lighting pipeline is executed entirely on the GPU.

3.2. Shading

Whereas light culling creates the list of lights overlapping each pixel, final shading goes through the list of lights and evaluates materials using information stored for each light – for example, light position and color which are not available for final shading when a deferred technique is used. Now, per-material instance information can be stored and accessed in linear structured buffers passed to the final shaders. Therefore, all constraints on pixel quality have been removed because light accumulation and shading happen simultaneously in one place with complete material and lighting information. Usage of complex materials and more accurate lighting models to improve visual quality is not constrained other than by the GPU computational cost which is largely determined by the number of overlapping lights hitting a single pixel $\times$ material cost $\times$ lighting model cost. With this method, high pixel overdraw can kill performance; therefore, the depth prepass is critical to minimize the cost of final shading.

4. Light Culling Implementation

The proposed rendering pipeline can reuse most of the code from an existing forward-rendering pipeline because it is primarily an extension of it. Compute-based light culling is the big addition to the technique. Thanks to the flexibility of current GPUs and graphics APIs, light culling can be implemented in several ways, each of which has pros and cons. This section describes two implementations that run entirely on the GPU. These approaches build lists of lights per tile. Although they could also build light lists per pixel by running additional kernel, this increases the computation time and the memory footprint.

4.1. Gather Approach

This method builds the list of lights in a single compute shader similar to [And11]. It executes a thread group per tile. A frustum of the tile is calculated using the range of the screen space of the tile and maximum and minimum depth values of the pixels. The kernel first uses all the threads in a thread group to read a light to the local register. Then the overlap of the lights to the frustum of the tile is checked in parallel. If the light overlaps, the thread accumulates the light to thread local storage (TLS) using local atomic operations. After all the lights overlapping a tile are collected in TLS, it flushes the lights to the global memory using all threads. This method is simple and effective if the number of lights is not too large. When there is a massive number of lights, one might consider using the scatter approach.

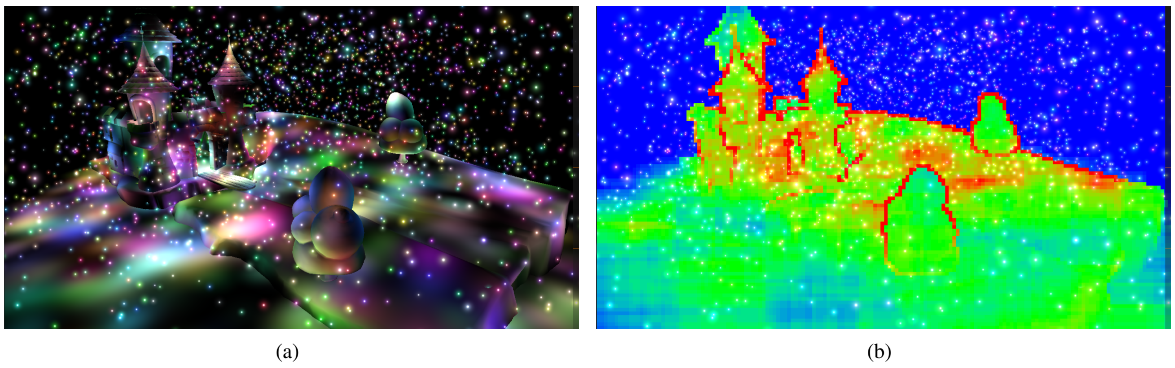

Figure 2: *A scene with dynamic 3,072 lights rendered in 1280x720 resolution. (a) Using diffuse lighting. (b) Visualization of number of lights overlapping each tile. Blue, green and red tiles have 0, 25, and 50 lights, respectively. The numbers in between are shown as interpolated colors. The maximum number is clamped to 50.*

4.2. Scatter Approach

This method creates lists in several steps. It first computes which tile a light overlaps and writes the light- and tile-index data to a buffer. This is done by executing a thread per light. The data of the buffer (ordered by light index at this point) needs to be sorted by tile index because we want a list of light indices per tile. We use a radix sort and then run kernels to find the start and end offsets of each tile in the buffer.

5. Results and Discussion

Forward+ was implemented using DirectX11©. Fig. 1 (a) is a screen shot of the AMD Leo demo using Forward+. Because the method allows the use of any surface shader, there is no limit to the lighting and surface materials. One example of materials used in the demo is the metal shader in which physically accurate effects are evaluated for all the lights contributing to the pixel.

However, nothing comes for free. What is the cost we are willing to pay for better pixel quality? In order to perform a detailed analysis of Forward+, for comparison, we implemented a compute-based deferred-lighting pipeline which does not write G-buffers [And11]. This implementation has a benefit that it saves memory bandwidth by reading and writing frame-buffer information once using the compute shader. Thus, it is theoretically the best in terms of memory bandwidth, which is a drawback of the standard deferred techniques. We compared the theoretical memory traffic of Forward+ and deferred for three steps of the pipeline. For light culling implementation, the gather approach is chosen because of the similarity of light culling to the deferred light accumulation. Table 1 compares the theoretical memory traffic of those methods. Deferred compacts all data to be written into a single float4 render target at G-prepass (normal.xyz, depth). Forward+ does not write a full-screen light accumulation buffer; instead, it writes only

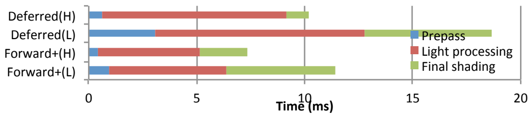

Figure 3: *Breakdown of the computation time for three stages on an AMD Radeon HD 6970 GPU. Deferred(L) and Forward+(L) are the times with lower GPU memory frequency.*

light indices overlapping each tile at light processing. In most cases, the data size of light indices is far less than full-size frame buffers. However, addition of light-culling stage increases memory reads at final shading: Forward+ has to read all the overlapping light indices and light information while the deferred technique reads only float4 from the light-accumulation buffer. If we subtract the total memory traffic of Forward+ from the total of deferred, we find $Total_{diff} = N \times F \times (15 - M \times (1/T + 1 + L))$ (see the caption of Table 1 for definitions of N, F, M, T, L). Solving $Total_{diff} > 0$ for M gives $M < 15 \times (1 + (1 + L) \times T)/T$. Thus, if lights are point lights ($L = 8$, position, radius and color) and the average number of lights per tile M is less than $15 \times (1 + (1 + L) \times T)/T > 15 \times 9$, Forward+ requires less memory traffic compared to the deferred ($Total_{diff} > 0$). Our comparison of Forward+ to compute-based deferred lighting technique that requires the least memory among deferred variants indicates that – in theory – no deferred variants can beat Forward+ in terms of memory traffic.

To confirm the analysis, we built a demo scene with 3,072 lights scattered in space as shown in Fig. 2(a) and compared the performance of these two methods. We used 8×8 for the tile size. Fig. 2(b) is a visualization of the number of lights per tile. The reason the tiles overlapping the edges of the

Table 1: *Memory traffic comparison for three steps. (A), (B), (C) for Forward+ are depth prepass, light culling, and final shading; for Deferred, G-prepass, light accumulation, and final shading. This table does not include common memory traffic such as depth buffer write at (A) and light read at (B).* N,F,M,T,L *are number of screen pixels, size of float, average number of lights overlapping a tile, tile size, and data size of a light, respectively. Forward+ reads depth at (B), writes light indices per tile at (B), and reads light indices and light at (C). Deferred writes normal vector at (A), reads depth and normal at (B), writes light accumulation at (B) and reads light accumulation at (C). Total is the summation of (A), (B), and (C).*

		(A) Prepass	(B) Light processing	(C) Final shading	Total
Forward+	Read	0	$N \times F$	$M \times N \times F \times (L+1)$	$N \times F + M \times N \times F \times (1/T + 1 + L)$
	Write	0	$M \times N/T \times F$	0	
Deferred	Read	0	$N \times F \times 4$	$N \times F \times 4$	$N \times F \times 16$
	Write	$N \times F \times 4$	$N \times F \times 4$	0	

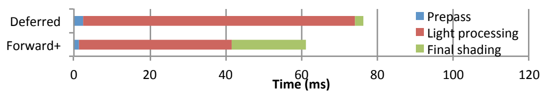

Figure 4: *Computation time on AMD A8-3510MX.*

models have more lights is the frustum tends to be long in the depth direction; thus, it reports lights overlapping to the entire frustum. To improve this issue, empty space in each frustum has to be detected. However, there can be any number of depth layers in a tile. A complicated empty-space detection does not pay off. Using a spatial subdivision instead of the two-dimensional subdivision might solve the issue but it requires more memory and ALU operations.

In Fig. 3, Forward+(H) and Deferred(H) are rendering times using an AMD Radeon$^{\mathrm{TM}}$HD 6970. We measured the computation time for prepass, light processing, and final shading of Table. 1 on the CPU, which includes kernel dispatch cost and GPU pipeline flush overhead. The measured timing is not pure GPU processing time, but the comparison is fair because the conditions are the same for both tests. The results confirm the analysis of theoretical memory bandwidth consumption: Forward+ is better for prepass and light processing, while deferred is better on the final shading. In total, Forward+ was faster. We also tested by slowing the memory clock of the GPU to emulate a GPU with lower memory bandwidth. As shown in the comparison of Forward+(L) and Deferred(L) in Fig. 3, the gap between Forward+ and the deferred widened because the deferred lighting requires more memory access. In the future, GPUs might increase the ALU/memory bandwidth ratio because adding more ALU units is easier than improving memory bandwidth. This experiment shows that Forward+ is promising on a GPU with that specification. We performed another test on an AMD A8-3510MX, which has an integrated AMD Radeon HD 6620G GPU. The results shown in Fig. 4, confirm the theoretical analysis of Forward+ on an integrated GPU, which has more limited memory bandwidth and fewer SIMDs compared to discrete GPUs. Some mobile or integrated GPUs employ tile-based rendering. Forward+ with the tiled light culling is well adapted to this rendering architecture because it can reduce the cost of reading a light list at final shading.

In this paper, we did not investigate how tile size affects performance, which depends on the tile size as well as many other factors such as the geometric complexity of the scene, number of lights, and influence maximum distance of lights. Automatic finding of the best tile size for a scene might be an interesting topic to explore in the future.

6. Conclusion

We presented Forward+, a rendering method that adds a GPU compute-based light-culling stage to the traditional forward-rendering pipeline to handle many lights. It unlocks us from the restrictions of deferred techniques and enables us to achieve better pixel quality by coupling the advantages of forward and deferred techniques. Forward+ theoretically requires less memory traffic than compute-based deferred lighting. We performed experiments that show Forward+ outperforms the least memory-intensive deferred-lighting technique.

As future work, many avenues can be explored because of the freedom at the final shading. An example is shadow computation from all the lights in the scene by local short ray casting to the lights.

References

[AMHH08] AKENINE-MOLLER T., HAINES E., HOFFMAN N.: *Real-Time Rendering*, 3 ed. AK Peters, July 2008. 2

[And11] ANDERSSON J.: DirectX 11 Rendering in Battlefield 3. *Game Developers Conference* (2011). 2, 3

[Kap10] KAPLANYAN A.: CryENGINE 3: Reaching the speed of light. *ACM SIGGRAPH 2010 Courses* (2010). 1

[Tre09] TREBILCO D.: Light indexed deferred rendering. In *ShaderX7* (2009), Charles River Media. 2