Analyzing Deferred Rendering Techniques

Alexandru-Lucian Petrescu, Florica Moldoveanu, Victor Asavei, Alin Moldoveanu

Computer Science and Engineering Department, University POLITEHNICA of Bucharest, Bucharest, 0600042, Romania
(e-mail: alexandru.petrescu@cti.pub.ro, {florica.moldoveanu, victor.asavei, alin.moldoveanu}@cs.pub.ro)

Abstract: In this article various deferred rendering algorithms are investigated and a classification that formalizes the comparison between these popular rendering techniques is introduced. This classification consists of measuring functions that can be used to determine the expected algorithm performance in various situations. Multiple analysis spaces are defined that better express the strengths and weaknesses of each algorithm. Given the abundance of deferred rendering methods and the performance tradeoffs implied by different hardware targets, rendered objects complexity and light setup complexity, our framework makes choosing or modifying an algorithm out of this collection a simpler process. The following spaces are used for algorithm examination and comparison: GPU commands, processing cost, allocated memory and expected bandwidth consumption. Furthermore, the analysis spaces are independent of the illumination model and are suitable for a decoupled examination, in which each stage of rendering process is usually executed at a different sampling rate.

Keywords: computer graphics, real-time rendering, deferred, decoupled, analysis

1. INTRODUCTION

The light-object intersection problem is one with a high complexity for real time rendering, because it is traditionally computed in $O(lights \times objects)$ complexity (Kircher et al., 2009).

The deferred algorithms split the rendering equation used to evaluate each light-object interaction into multiple evaluation stages that are both cache friendly due to data locality and equivalent to the initial rendering equation from the standpoint of the final result. Through this division, the family of deferred algorithms solves the complexity problem by lowering it to $O(lights + objects)$, with the notable mention that it is only applicable when shadows are not computed for all the lights. Nonetheless, there are workarounds for solving shadows for many lights (Olsson et al., 2014; Ritschel et al., 2008).

The equation separation is performed through the use of explicit or implicit intersection acceleration structures. In the case of the explicit intersection acceleration structures, one-level quad trees, buckets and tridimensional grids have been used. From an implicit intersection acceleration structure standpoint the raster works just as a bi-dimensional grid.

The order of intersection differs between algorithms. Some intersect the set of objects with the set of lights, while others intersect the set of lights with the set of objects. Even if this operation is mathematically commutative each approach has different optimization structures and strategies. Independently of the direction of the intersection between the scene lights set and the scene objects set the first operand is used to fill the acceleration structure and the second operand just queries the first operand through the acceleration structure and then stores the results of the evaluated equation part in the correct pixel. At the end of the process, the final illumination result is stored in each pixel and the pixels are sent to be displayed.

Furthermore, decoupled (Ragan-Kelley et al., 2011, Liktor et al., 2012) algorithms can also be considered deferred algorithms. In decoupled algorithms the principle of computation separation is taken to its logical extreme where each rendering sub-process is sampled and evaluated at a separate frequency. This is extremely effective in rendering processes where there are large frequency differences between stages, like the REYES (Cook et al., 1988) pipeline.

The decoupled rendering algorithm class becomes even more important when techniques such as per frame visibility caches or illumination caches are used. On the other hand, decoupled techniques require state of the art hardware for real time rendering and this impediment makes their usage in a real world real-time scenario rare for the moment.

Additionally, some algorithms that evaluate the scattering of light through transparent media are also exhibiting deferred and decoupled principles, the largest category being Order Independent Transparency algorithms.

The deferred algorithms also offer other advantages such as the separation of scene objects and scene lights management and increased memory access coherency. They also provide excellent modularity for software architectures such as rendering engines where they can easily be used together with different global illumination (GI) algorithms (Tokuyoshi et al., 2012; Davidovic et al., 2012). While deferred algorithms do not offer the same visual quality as the global illumination algorithms, they are much faster, because global illumination algorithms and their acceleration structures are optimized for much more complicated light processes and interactions. Global illumination usually employs a form of hierarchical traversal such as a BVH tree, a Kd-tree or any

other variant of acceleration structure, while deferred algorithms use the raster which acts as a hierarchical coherent hash tree, binning objects through a spatial hash function.

Thus, compared to the global illumination family of algorithms the deferred family is much faster due to much lower processing costs and due to superior memory coherency, even if many coherency enhancing methods were introduced for global illumination algorithms such as path tracing or photon mapping. Moreover, the complexity of deferred methods is lighter, because the hierarchical early Z rejection algorithm leads to a complexity of $O(constant)$ with inexact Z sorting, compared to the natural complexity $O(\log n)$ of GI methods. In practical applications the constant is decidedly small and the difference of performance only increases when using multiple samples per pixel.

Because of all these reasons, the deferred and decoupled algorithms will continue to see plenty of use and choosing the correct algorithm for a specific problem will lead to some difficult choices. In order to ease such choices and to provide a way of estimating expected algorithm performance, a measuring and analysis method is proposed, that formalizes the comparison between deferred algorithms. Different analysis spaces are introduced, such as the GPU commands, processing cost, allocated memory and expected bandwidth consumption, which greatly help in establishing which algorithm would be better suited for a specific task.

2. STATE OF THE ART

The deferred rendering technique was introduced by (Deering et al., 1988), although back then the authors didn't use the term "deferred" to describe their method. The use of the Geometry Buffer (G - buffer) was as the support for an intermediate processing stage in a rendering pipeline which used 2D operations to handle discontinuities and to enhance the final image's edges and contour lines. Because of the improved complexity, the decoupling of scene and light management and the elegance of implementation, the deferred techniques have evolved into some of the most used algorithms in real-time rendering (Shishkovtsov, 2005; Koonce, 2007).

The deferred algorithms decouple the processes of illumination evaluation and object visibility determination through the use of an implicit or an explicit acceleration structure. Thus, these processes can run at different sampling frequencies, although the shading samples and the visibility samples still run at coupled frequencies. If hardware multisampling (MSAA) (Jimenez et al., 2011) is used to improve the sampling frequency of the visibility determination process, in order to maintain equal sampling frequencies, the intermediate geometry buffers will need to have sufficient space to store all the results from the visibility determination stage. Therefore, the decoupling of visibility determination and light evaluation leads to an increase in memory use which is proportional to the increase in sampling frequency of the visibility determination stage (Mara et al., 2013; Thibieroz, 2009) .

Other more intricate deferred rendering pipelines such as decoupled sampling (Liktor et al., 2012), further complicate the management of samples. Without decoupling sampling rates the memory requirements would be unachievable in real-time, on current consumer hardware.

Because of the different sampling frequencies used in the visibility determination and in the light equation evaluation stages, results have to be reconstructed sometimes. Therefore, all the potential contributors per pixel must be accounted for. This leads to a lot of erroneous intersections between lights and objects which have to be stored and computed, but which contribute nothing to the final image. Thus, aliasing and noise become major efficiency problems with deferred algorithms, not only from the traditional perspective of visibility determination. Consequently, classic solutions such as MSAA are not sufficient for this type of problems and many explicit space partitioning schemes such as 2.5D culling, tiles and clusters have been introduce to reduce these effects.

The problem of transparency is even more complicated for deferred algorithms because correctly sorting objects independently of viewpoint is impossible (Salvi et al. 2011), thus space isn't sampled only in two dimensions but in three, each pixel or pixel sample requiring many depth samples for correct rendering. The memory requirements are in general prohibitive and several algorithms exist that provide solutions through intermediate stages or approximations (Pangerl, 2009; Kircher et al., 2009; Mara et al., 2013).

Deferred algorithms have been successfully combined with global illumination methods (Tokuyoshi et al., 2012) with the expected results being identical to those obtained by using only global illumination techniques. Thus, because of the modularity of the deferred algorithms, it is preferable to use them in combination with global illumination methods instead of only using the latter. The multitude of deferred algorithms can be classified into three major categories:

- implicitly accelerated intersection of lights and objects through the raster grid structure, which acts as an implicit bidimensional associative array, in which the objects are binned, and in which the objects intersecting the lights are queried during the rasterization of lights, during the lightpass stage.
- explicitly accelerated intersection of lights and objects through clusters, tiles, lists, bvh trees, kd trees, etc.
- decoupled rendering with many stages each running at distinct sampling rates, where the samples are linked through many-to-one or many-to-many mappings in addition to other acceleration structures.

2.1 Implicitly Accelerated

The first technique from this category is depth pre pass rendering, in which the objects are drawn twice. In the first stage the objects are rendered only for visibility determination, thus materials and textures are disabled and therefore, this rendering pass is very fast. In the second pass the objects are rendered with the full material setups but the early Z rejection test will discard all the resulting object fragments that are not visible on the screen, thus evaluating the costly illumination equation only for the shading samples which affect the final visual result.

In deferred rendering (Shishkovtsov, 2005) there are three rendering passes. In the first one the objects are rendered into the geometry buffer, which holds positions, normals, albedo and other reflectance factors. In the second rendering pass the lights are rendered and at each fragment generated by rasterizing a light, the illumination equation is evaluated and accumulated. In the final pass the accumulated illumination is composited with the albedo. Stencil optimized deferred rendering (Olsson et al., 2011) is an optimization of the classic deferred rendering algorithm, specialized for imperfect bounding light geometries. It uses shadow volumes principles to evaluate the illumination equation only where light support geometry affects the result.

Light pre-pass rendering (Lee, 2009), also known as deferred lighting is a three pass technique that uses a much smaller geometry buffer, with only depth and normal entries. In the first pass, the light geometry buffer is filled, in the second pass the illumination is evaluated and accumulated and in the final pass the objects are rendered with full materials and textures and the resulting image is composited with the lighting from the second pass.

Transparency for a single layer can be handled with a variant of deferred rendering that uses a screen door transparency scheme (Pangerl, 2009). First, the opaque objects are rendered at full resolution and then, only one out of every four pixels is written for the transparent objects in the upper corner of a 2x2 pixel vicinity. In the composition pass the pixel's transparency is evaluated by querying the entire 2x2 pixel vicinity. The previous idea is taken further by inferred rendering (Kirscher et al., 2009), which handles transparency by employing a complex stippling pattern and a secondary geometry buffer. This method interlaces transparency samples in a small number of layers, and then uses a bilateral filter, named discontinuity sensitive filter in (Kirscher et al., 2009), to reconstruct the final result.

Techniques that use multiple samples without any explicit acceleration structure such as deep deferred shading (Mara et al., 2013) or multisample anti-aliasing deferred rendering (Thibieroz, 2009) should also be considered implicitly accelerated. In deep deferred shading a small number of layers is kept in many geometry buffers. The memory costs for this method are extremely large. In multisampled anti-aliased deferred rendering more than one sample is allocated per pixel, but the illumination is evaluated at sample level – not pixel level- only for the pixels where the sample coverage mask is not complete. Thus, even if the memory is allocated for all possible samples the bandwidth is consumed only where sample level evaluation is necessary.

2.2 Explicitly Accelerated

Light indexed deferred rendering (Treblico, 2009) uses depth sorted lights to obtain the closest lights for each pixel. This is done through four binary per pixel lists implemented through the rasterization blending mechanism. After the light index lists are built the objects are rendered normally and, for each pixel, the indexed lists are queried and only the closest lights are accessed in order to evaluate the illumination equation. This artificial mechanism is required because this algorithm doesn't utilize the Shader Model 5 (SM5) instruction set.

List based light indexed deferred rendering (Lauritzen, 2012) uses the same concept as the previous algorithm but implements the lists directly, at a per pixel level. In the geometry rendering pass the lists are queried and the illumination equation is evaluated with the lights which had their indices stored in that pixel's list.

Tiled forward rendering (Olsson et al., 2011) uses a low resolution bidimensional grid to bin the lights in tiles. A list is kept for each tile and if a light is rasterized over that tile, its index is added to that tile's list. In the object rendering pass the computed pixels query the parent tile list in order to obtain the light ids needed to evaluate the illumination equation. Tiled deferred rendering (Olsson et al., 2011) uses a geometry pass and a final pass that interpose the tiled lighting pass in a manner similar to that of the classic deferred rendering. While lower memory consumption and a significantly smaller number of lights per list are the advantages of tiled forward, tiled deferred has only one geometry pass. Both algorithms are prone to storing non-intersecting lights in the light index lists.

Tiled deferred with 2.5D culling (Harada, 2012), known as Forward+, is an improvement of tiled forward rendering. By using a depth occupancy mask per tile this algorithm culls a large part of the non-intersecting lights, offering significant performance improvements in setups with high range low variety depth distributions. Clustered forward (Olsson et al., 2012) evolves the bidimensional grid used as an acceleration structure to a tridimensional grid. Thus, there are many more lists and the intersection between lights and objects is evaluated at a considerably larger spatial sampling rate, giving superior results and less erroneous light-object intersections. Clustered deferred (Olsson et al., 2012) rendering is similar to clustered forward but instead uses the available visibility information to have a better spatial distribution for the clusters.

2.3 Decoupled

Since deferred rendering is prone to aliasing, some sort of anti-aliasing solution is required. The trivial solution of multisampling the Geometry Buffer leads to an enormous amount of allocated memory. The preferred solutions are to run the deferred algorithm at per pixel sampling resolution and then do antialiasing work in a post processing framework. The main algorithms used in the post processing stage are FXAA, MLAA and SRAA (Jimenez et al., 2011). But even with these anti aliasing methods there are situations where without programmable multi sampling an extremely large number of samples would be required. Geometric setups like those that led to the development of WireAA (Jimenez et al., 2011), and effects such as motion blur and depth of field all make choosing the Geometry Buffer samples a complicated and frame varying choice.

The concept of decoupling sampling rates in real time rasterization was inspired from the REYES (Cook et al.,

1988) rendering pipeline. In (Ragan-Kelley et al., 2011) the authors separate the visibility and the shading samples and create a many-to-one dependency between the former and the latter. Decoupled sampling was adopted for deferred rendering methods in (Liktor et al., 2012) with the introduction of the Compact Geometry Buffer (CG - buffer). This method is the first one in which the coupling of frequencies between the visibility sampling rate and the shading sampling rate is handled rigorously in a deferred context. Instead of using a lot of visibility and shading samples to implement stochastic rasterization, decoupled sampling is used to provide extra samples in the deferred buffer in the places where it is under sampled. By using this method, motion blur and depth of field effects are attainable. The implementation of decoupled sampling is complicated and requires SM5 hardware.

Sort based deferred rendering (Clarberg et al., 2013) improves the Compact Geometry Buffer from (Liktor et al., 2012) by sorting and then culling occluded primitives in worktiles, and shading the fragments in an order that improves texture access coherency.

The Visibility Buffer algorithm (Burns et al., 2013), also named Deferred++, applies the idea of decoupled sampling to bandwidth starved hardware. Instead of storing many different data in the G-Buffer this method only stores the ids of the closest visible primitives and then computes the final visual result per tile. For each tile it loads all the required information such as geometry, lights and textures into the fastest available memory. Because the algorithm requires a limited amount of memory it has low energy consumption.

Furthermore, techniques in which deferred rendering is the foundation for global illumination methods such as Reflective Shadow Maps (Dachsbacher, 2005), SSDO (Ritschel et al., 2009) or in which it acts as the first bounce in a path or cone tracing type of algorithm such as in Voxel Cone Tracing (Crassin, 2011) or Real-time Birdirection Path Tracing via Rasterization (Tokuyoshi et al., 2012) can also be considered decoupled sampling. The primary segments in the path are sampled at the multisampled screen resolution frequency, while the rest of the segments in the path are obtained by sampling the scene a different frequency, often much lower than the one used for the first segments.

3. ANALYZING DEFERRED

In this section we define the symbols that are used in algorithm comparison and analysis. The four spaces of analysis are GPU commands, processing cost, allocated memory and expected bandwidth consumption. Each of these can be a performance bottleneck. The GPU commands are used to weigh the algorithm's cost in transferring commands through the memory bus, it includes state changes, drawing commands and pipeline stalls. On some hardware architectures with a lot of processing power and large bandwidth capabilities, the number of GPU commands can become the most difficult to solve bottleneck.

The processing cost measures the computational weight of the algorithm. The allocated memory and the bandwidth consumption are used to differentiate between the predetermined memory costs of the algorithm (e.g. G-buffer) and the expected memory accesses. The difference between these becomes more pronounced when using superior sampling methods. Also, bandwidth requires a lot of energy consumption, so a lot of attention has to be given to it in mobile oriented rendering.

This is the list of symbols used in our algorithm analysis framework. Bandwidth and memory are measured in floats. Where it is applicable, the bandwidth includes attributes and interpolators, for example in the vertex bandwidth cost.

o	number of scene objects.
l	number of scene lights.
o_{cmd}	cost of a single GPU object draw command
l_{cmd}	cost of a single GPU light draw command
ov	number of object vertices.
ov_{proc}	processing cost for an object vertex.
ov_{band}	bandwidth cost for an object vertex
lv	number of light vertices.
lv_{proc}	processing cost for a light vertex.
lv_{band}	bandwidth cost for a light vertex
of	number of fragments from objects rasterization.
of_{band}	bandwidth cost of a single object fragment sample.
lf	total number of fragments from lights rasterization.
lf_{band}	bandwidth cost of a single light fragment.
eq_{proc}	processing cost of the illumination model
p, c, t	total number of pixels, tiles, respectively clusters.
$vspp, vsppe$	maximum, expected visibility pixel samples.
$sspp, ssppe$	maximum, expected shading samples per pixel.
$memll$	maximum allocated memory for light lists.
$memol$	maximum allocated memory for object data lists.
lcp, vcp	light, vertex culling probability.
K_{cmd}	total cost of the GPU commands.
K_{mem}	total cost of allocated memory, in floats.
K_{proc}	total cost of processing.
K_{band}	total bandwidth cost, in floats read or written.

4. ALGORITHM DISCUSSION

In the following algorithm examinations we assume that the output framebuffer is of the 8 bit RGBA type, which can be written in a single memory instruction, and that the depth buffer has the size of a 32 bit float. Where the visibility and shading samples are coupled we always use $sspp$ instead of $vspp$. In both the memory and bandwidth costs we include everything besides the default framebuffer. We do not add constant costs such as the commands cost for the final pass in deferred rendering. The equations will be written with readability not compactness in mind, following the steps of each algorithm. Special attention is given to the equations layout, where each rendering pass is written between curly braces, with the number of the pass added as a subscript.

4.1 Forward Rendering

Forward rendering (***F+***) is included for comparison reasons.

$$K_{cmd} = \{o \times l \times o_{cmd}\}_1$$
$$K_{proc} = \{ov \times ov_{proc} + of \times l \times eq_{proc}\}_1$$
$$K_{mem} = 0$$
$$K_{band} = \{ov \times ov_{band} + of \times (of_{band} + 2)\}_1$$

4.2 Depth Pre-Pass

Depth pre-pass rendering (***DPP***) is the first to separate visibility determination and shading. While it does not decouple light and object intersection, the authors believe that because it does actual separation it is correct to consider it an early deferred algorithm. It is also commonly used and thus its analysis is practical. It consists of two passes, in the first one the scene objects are rendered without materials, textures or special effects in a process called depth only rendering. After the first pass, the depth buffer stores for each pixel the depths of the closest objects. In the second pass the scene objects are drawn with forward rendering, but the early Z rejection destroys all the fragments that are not visible before the shading stage, therefore the shading is evaluated only for the visible fragments and it is executed at a different frequency than that used for visibility determination. The performance metrics are:

$$K_{cmd} = \{o \times o_{cmd}\}_1 + \{o \times l \times o_{cmd}\}_2$$
$$K_{proc} = \{ov \times ov_{proc}\}_1 + \{ov \times ov_{proc} + p \times eq_{proc}\}_2$$
$$K_{mem} = 0$$
$$K_{band} = \{ov \times ov_{band} + p \times (of_{band} + 2)\}_2 +$$
$$+\{ov \times ov_{band} + of\}_1$$

While benefiting from material flexibility and being easily multisampled with hardware MSAA, the GPU code path combination explosion and the inability to handle many lights makes rendering dense scenes a difficult task with the depth pre-pass method. Still, on some memory constrained hardware platforms depth pre pass can prove to be a useful technique.

4.3 Deferred Rendering

Also known as deferred shading (***D***), this is a three phase algorithm and it is the first that can handle a large number of lights because it is the first that decouples light scene management and object scene management. In the first pass the objects are rendered and saved into a geometry buffer, similar to the memory structure depicted in Figure 1.

DT	Depth			
RT1	Accumulation R	Accumulation G	Accumulation B	extra 1
RT2	Normal X	Normal Y	Normal Z	extra 2
RT3	Albedo Red	Albedo Green	Albedo Blue	extra 3

Fig. 1. The Geometry Buffer. Normals, albedo, illumination and other details of the visible opaque objects of the scene is saved in a multi-rendertarget buffer.

Besides the lighting accumulation buffer and the normal buffer the other information can vary depending on rendering architecture, but the principle is identical. In the second pass the lights are rendered as geometry, being rasterized over the geometry buffer, which contains information about all the on-screen visible opaque objects. For each light the G-Buffer is queried for information, the initial object position is reconstructed from depth and then, the lighting equation is evaluated, its result is then added in the accumulation buffer.

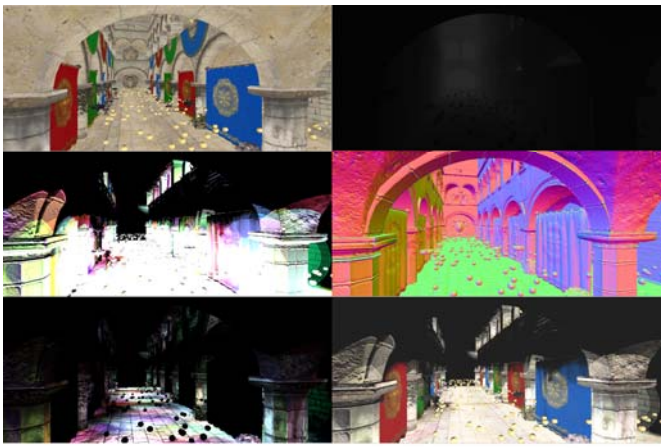

Fig. 2. Render targets in deferred rendering: albedo, depth, diffuse radiance, normals, specular radiance and the final composited image.

In the final pass, called composition pass or shading pass, the albedo information is combined with the accumulated lighting from the second pass to obtain the final radiance per pixel. This is done in a full screen pass. A visual outline of the render targets is given in Figure 2.The performance metrics for deferred rendering are:

$$K_{cmd} = \{o \times o_{cmd}\}_1 + \{l \times l_{cmd}\}_2$$
$$K_{proc} = \{ov \times ov_{proc}\}_1 + \{lv \times lv_{proc} + lf \times eq_{proc}\}_2$$
$$K_{mem} = p \times 4$$
$$K_{band} = \{ov \times ov_{band} + of \times (of_{band} + 3)\}_1$$
$$+ \{lv \times lv_{band} + lf \times (lf_{band} + 3)\}_2 + \{p \times 3\}_3$$

In the bandwidth cost the $lf_{band} + 3$ term refers to the bandwidth cost of the light fragment, plus reading the depth and normals and writing to the light accumulation buffer. The $of_{band} + 3$ term contains the reading from the albedo and light accumulation render targets and the final color output. The deferred algorithm lowers the command costs significantly and eliminates the processing cost from the pixels where lights don't intersect objects, therefore it can easily handle a much larger number of lights. On the other hand, the allocated memory is considerable. The most important problem with deferred rendering is the impossibility to handle correct transparency, because the algorithm only keeps information about the closest objects for each pixel. A variant of this technique, deep deferred rendering (***DD***), employs a series of layers that partition the scene objects into multiple layers ordered by depth. The layers can be implemented through multiple shading samples, each sample representing a layer. The disadvantage is that with deep deferred rendering each layer adds memory and bandwidth costs equal to those of an additional G-buffer, which quickly add up to a prohibitive expenditure. The performance metrics for deep deferred rendering are:

$$K_{cmd} = \{o \times o_{cmd}\}_1 + \{l \times l_{cmd}\}_3$$
$$K_{proc} = \{lv \times lv_{proc} + lf \times sspp \times eq_{proc}\}_2 +$$
$$\{ov \times ov_{proc}\}_1$$
$$K_{mem} = p \times 4 \times sspp$$
$$K_{band} = \{ov \times ov_{band} + of \times (of_{band} + 3) \times sspp\}_1$$
$$+\{lv \times lv_{band} + lf \times (lf_{band} + 3)\}_2 + \{p \times 3 \times sspp\}_3$$

Moreover, the final shading pass resolution is limited to match that of the first visibility determination pass, therefore MSAA is very expensive from a memory and bandwidth standpoint because it has to be applied to the entire G-buffer, thus effectively becoming SSAA (Jimenez et al., 2011) and also being visually identical to deep deferred rendering. The results can be improved with full screen geometric anti aliasing methods such as MLAA, FXAA, SRAA (Jimenez et al., 2011), TXAA or RSAA (Reshetov, 2012).

4.4 Deferred with MSAA

Hardware MSAA is an important and efficient means to lowering geometric aliasing in rasterization. It is efficient because it computes samples only when they are needed, in contrast with SSAA. As the bandwidth and not the allocated memory is usually the major bottleneck with deferred rendering variants, this algorithm focuses on lowering bandwidth by increasing sampling rate only for the pixels on which a geometric edge is rasterized.

Deferred MSAA (***DAA***) has three stages. In the first stage the objects are rendered into a multisampled G-buffer. The existence of edges is determined using a centroid interpolation scheme (Thibieroz, 2009). If the pixel contains an edge then the stencil will be set to non-negative, otherwise it is set to zero. The second stage is composed of two different shading passes. The first pass is run at pixel frequency and it evaluates the illumination equation and then writes its result to sample zero in a multisampled light accumulation buffer. The second pass is run at sample frequency, with the stencil set to pass only for the pixels that contain an edge, as it was previously determined in the first stage.In the final stage the multisampled G-buffer and accumulation buffer are composited and then final result is outputted. The performance metrics for MSAA deferred rendering are:

$$K_{cmd} = \{o \times o_{cmd}\}_1 + \{l \times l_{cmd}\}_2$$
$$K_{proc} = \{lv \times lv_{proc} + lf \times sspp e \times eq_{proc}\}_2 + \{ov \times ov_{proc}\}_1$$
$$K_{mem} = p \times 4 \times sspp$$
$$K_{band} = \{ov \times ov_{band} + of \times (of_{band} + 3) \times sspp\}_1 + \{lv \times lv_{band} + lf \times sspp e \times (lf_{band} + 3)\}_2 + \{p \times 3 \times sspp\}_3$$

This method offers an allocated memory size comparable to that of deep deferred rendering but with a bandwidth closer to classic deferred rendering. Even with the increased processing cost, it is practical.

4.5 Light Pre Pass Rendering

Light pre-pass rendering (***LPP***), also known as deferred lighting, is a three pass algorithm that combines the light object decoupling introduced by deferred rendering with the ease of material management and hardware multisampling from forward rendering. It does so by further decoupling visibility and shading. Compared to deferred rendering, it does not need to load and compute albedo or material factors in the first geometry pass. While deferred rendering effectively mixes shading and visibility determination work and thus wastes processing and bandwidth on occluded fragments, light pre pass rendering does shading work only for the visible shading samples, because this work is computed after completing the visibility determination stage.

In the first pass the objects are rendered into a smaller G-buffer, which holds only depth and normals. In the second pass the lighting equation is evaluated like in the classic deferred lighting pass, with position reconstructed from depth, and the result accumulated in the lighting buffer. In the final pass the objects are rendered with full materials and textures. Because of the depth buffer from the first pass only the closest object fragments will be shaded in this pass. The first pass is through effect similar to depth pre-pass. The performance metrics for light pre pass rendering are:

$$K_{cmd} = \{o \times o_{cmd}\}_1 + \{l \times l_{cmd}\}_2 + \{o \times o_{cmd}\}_3$$
$$K_{proc} = \{ov \times ov_{proc}\}_1 + \{lv \times lv_{proc} + lf \times eq_{proc}\}_2 + \{ov \times ov_{proc}\}_3$$
$$K_{mem} = p \times 3$$
$$K_{band} = \{ov \times ov_{band} + of \times 2\}_1 + \{lv \times lv_{band} + lf \times (lf_{band} + 3)\}_2 + \{ov \times ov_{band} + p \times (of_{band} + 2)\}_3$$

In the bandwidth cost is only $of_{band} + 2$ since in the final pass the albedo isn't read from a previous render target. Because this algorithm does not write more than twice per fragment, one write being depth, it can be implemented on hardware that does not have multiple render targets. Furthermore, with this technique hardware MSAA can be used in the final pass and the lighting information can be used for transparent objects. On the other hand the large number of GPU commands can become a bottleneck for large scenes. Since the bottleneck in deferred rendering is generally overdraw in the geometry pass, the bottleneck in light pre pass rendering is usually either the GPU commands cost or the vertex processing cost. This performance varies depending on scene configuration. Hybrid deferred rendering (Hoef, 2013) is an algorithm that combines deferred rendering and light pre pass rendering, dynamically deciding the shading path per object using a path selection method. It does so by predicting the overdraw cost per object and then it decides which path is more efficient, at a per object level.

4.6 Deferred with Transparency

Correct transparency is impossible to obtain in rasterization with deferred methods, since they store only a fraction of the required information. Nevertheless, methods such as deferred rendering transparency (Pangerl, 2009) and Inferred rendering (Kirscher et al., 2009) render inexact transparency through the use of multiple passes which decouple the opaque and transparent objects and combine their results through stippling patterns. Deferred Transparency's (***DT***) first pass the opaque objects are drawn into a G-buffer, like in all deferred algorithms. We make use of the stippling vicinity, a kernel of $k \times k$ size used for transparency operations. Let $mask$ be the number of elements in the stippling vicinity which will be considered for transparent storage and let τ be the percentage of the transparent objects out of all the scene objects. Common stippling patterns are shown in Figure 3.

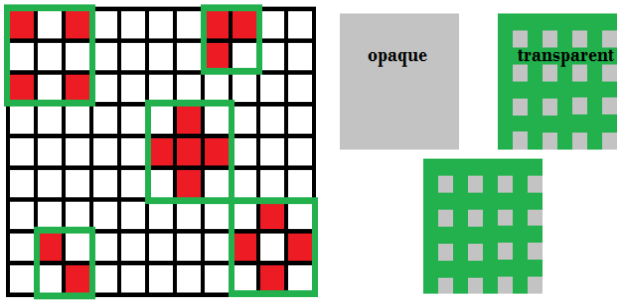

Fig. 3. Stippling deferred transparency. Stippling patterns can be used to store sparse transparency information in a Gbuffer.

In the second pass, the transparent objects are rasterized with a stippling pattern, writing only where the stippling mask permits it, as shown in the right side of Figure 3. The information from the transparent objects is written into another G-buffer that stores information only where the stippling mask permits it, being of maskp/k^2 resolution.

In the third pass the lighting is computed relative to both G-buffers and then accumulated. In the final pass the lighting is applied to the objects. A simple approximation filter (Pangerl, 2009) or a bilateral filter, named discontinuity sensitive filter in (Kirscher and Lawrence, 2009), can be used to reconstruct transparency information for the kernel entries that were masked out, but the entire vicinity has to be read. The filter can use the distance between the pixel and each masked entry in the stippled vicinity as a weight function.

Stippling masks can act as more than a set of mask entries. Each single mask entry can store transparency information for the entire vicinity, making this method act as deep deferred rendering for transparent objects, but at a $1/k^2$ resolution. In the composition pass, for each pixel the entire vicinity is read, sorted by depth and then composited. Even with this method correct complex transparency effects are impossible to obtain.

The performance metrics are:

$$K_{cmd} = \{(1-\tau)\times o \times o_{cmd}\}_1 + \{\tau \times o \times o_{cmd}\}_2 + \{l \times l_{cmd}\}_3$$

$$K_{proc} = \{(1-\tau)\times o \times o_{proc}\}_1 + \{\tau \times o \times o_{proc}\}_2 + \left\{lv \times lv_{proc} + lf \times (1+\frac{mask}{k^2}) \times eq_{proc}\right\}_3$$

$$K_{mem} = p \times 4 + p \times 3 \times \frac{maskp}{k^2}$$

$$K_{band} = \left\{of \times (1-\tau+\tau\times\frac{mask}{k^2}) \times (of_{band}+3)\right\}_{12} + \left\{lv \times lv_{band} + lf \times (1+\frac{mask}{k^2}) \times (lf_{band}+3)\right\}_3 + \left\{p \times (of_{band}+2+\frac{mask}{k^2})\right\}_4 + \{ov \times ov_{band}\}_{12}$$

4.7 Light Indexed Deferred

In general, the greatest processing cost when doing deferred rendering is in evaluating the illumination equation. Since all the lights are intersected with the nearest objects per pixel, this leads to a lot of computations that have no impact in the final rendering because the lights are too far away from the object surface stored in that pixel. Another problem is the fact that some lights have a dominant contribution in the accumulation buffer and this makes other lights to be insignificant by comparison. Probably the most important problem originates from a quality constraint. In order to correctly accumulate the light contributions the accumulation process has to be evaluated in a high floating point resolution buffer which largely increases the bandwidth of the algorithm. Normally, deferred rendering accumulates light in an 8 bit per channel setup, which can easily lead to overflows and therefore has to be clamped.

Light indexed rendering is a type of deferred rendering where the lighting pass is used exclusively to gather all the potentially contributing lights and, with them, to compute the lighting equation during the final pass. The lights are gathered through index lists, minimizing bandwidth. There are two variants for this algorithm. The first one runs on SM4, while the second one requires SM5. The first one, called light indexed deferred rendering (***LiD***), starts with a geometry stage that fills the G-buffer. Then, in the lighting stage, it divides the light index into n parts, each representing b bits. For the current formulas we will use $n = 4$ and $b = 2$ as they have the most practical value for this algorithm. A first in first out list which keeps only the most recent n indices per pixel can be implemented with the following blending equation:

$$Result = NewFragment + 0.25 \times OldFragment$$

The lights need to be coarsely sorted for this algorithm to work. After the lighting pass has ended the last 4 indices are from the closest lights. Then, in the final composition pass, for each pixel, the four closest lights' indices are reconstructed and illumination is evaluated. The performance metrics for light indexed deferred are:

$$K_{cmd} = \{o \times o_{cmd}\}_1 + \{l \times l_{cmd}\}_2$$

$$K_{proc} = \{ov \times ov_{proc}\}_1 + \{lv \times lv_{proc}\}_2 + \{p \times 4 \times eq_{proc}\}_3$$

$$K_{mem} = p \times 4$$

$$K_{band} = \{ov \times ov_{band} + of \times (of_{band}+3)\}_1 + \{lv \times lv_{band} + lf\}_2 + \{p \times (lf_{band} \times 4 + 4)\}_3$$

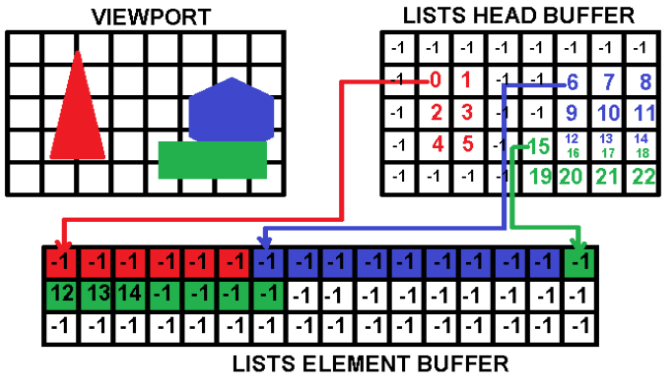

Fig. 4. Linked lists on the GPU. Linked lists are used to store lights that intersect pixels, tiles or clusters.

List based light indexed deferred rendering (***LLiD***) is the second variant of the light indexed variants. The algorithm is

identical to light indexed deferred rendering, with the only difference being in how the lists are handled. In the list based version full lists are implemented for each pixel, keeping the index of every light that was rasterized over it. In the final pass, the entire list is loaded and the light equation is evaluated for each light index. The process of adding elements to a list on the GPU is described in Figure 4. The performance metrics for list based light indexed deferred:

$$K_{cmd} = \{o \times o_{cmd}\}_1 + \{l \times l_{cmd}\}_2$$
$$K_{proc} = \{ov \times ov_{proc}\}_1 + \{lv \times lv_{proc}\}_2 + \{lf \times eq_{proc}\}_3$$
$$K_{mem} = p \times 3 + memll$$
$$K_{band} = \{ov \times ov_{band} + of \times (of_{band} + 3)\}_1 + \{lv \times lv_{band} + lf\}_2 + \{lf \times lf_{band} + p \times 4\}_3$$

Both light indexed algorithms can be written from a light pre pass perspective, with two geometry passes, named light indexed forward (***LiF***) and list based light indexed forward (***LLiF***). An interesting property of light index based solutions is that they trade the sequential light data memory access pattern found in the lighting pass of deferred methods for a sequential G-buffer data memory access pattern in the final pass. Since light data is usually much smaller in terms of bandwidth compared to G-buffer data, this trade-off becomes a big performance gain when using many lights.

4.8 Tiled Shading

Tiled rendering emphasizes coherent memory accesses and local optimizations, both of which are extremely common in rendering. Tiled shading (Olsson, 2011) combines the deferred rendering principles with the tiled rendering benefits. It has two variants: tiled deferred shading (***TD***) and tiled forward shading (***TF***). Tiled deferred shading is a three pass algorithm that is similar to deferred rendering and tiled forward shading is a three pass algorithm similar to light pre pass rendering, but instead of the normal lighting pass both algorithms use a tiled lighting pass. What makes tiled shading unique is that it has an explicit light intersection acceleration structure, a uniform grid, which is fully programmable compared to the raster acting as an implicit uniform grid.

Because tiled rendering explicitly groups lights by post projection positions in its lighting pass, it acts as a clustering algorithm in screen space. Thus, pixels in a neighbourhood which are in the same tile extruded frustum and which have a high probability to be lit by the same lights are now sharing a light index list instead of each pixel owning one. This leads to a large reduction in bandwidth in the lighting pass but also increases the synchronization needs, which are matched by expensive atomic operations.

In the tiled composition final pass the parent tile of the evaluated pixel is queried for the list of potential contributor lights indices, which are then read and with which the illumination equation is evaluated. By doing so the tiled lighting pass is equivalent to doing a list based light indexed deferred rendering light pass but at 1/t resolution. Also, similar to the light indexed variants, the tiled shading variants trade the sequential light data access pattern for a sequential G-buffer access pattern, which leads to improved bandwidth.

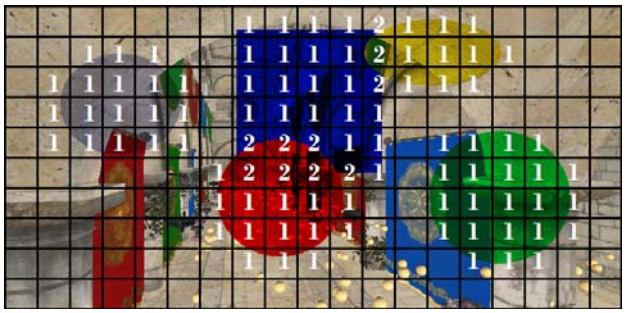

Fig. 5. Tiled light rasterization. The screen is divided into tiles, each tile holds a list of light indices that are potential contributors. The numbers in the image represent the number of lights for each tile list, zero where not shown.

The final pass is evaluated in a compute shader, thus the bandwidth for reading lights for the entire tile is consumed only once, by using local memory in a compute shader. The performance metrics for tiled deferred rendering are:

$$K_{cmd} = \{o \times o_{cmd}\}_1 + \{l \times l_{cmd}\}_2$$
$$K_{proc} = \{ov \times ov_{proc}\}_1 + \{lv \times lv_{proc}\}_2 + \{lf \times eq_{proc}\}_3$$
$$K_{mem} = p \times 3 + memll$$
$$K_{band} = \{ov \times ov_{band} + of \times (of_{band} + 3)\}_1 + \left\{lv \times lv_{band} + \frac{lf}{t}\right\}_2 + \left\{\frac{lf}{t} \times lf_{band} + p \times 4\right\}_3$$

The performance metrics for tiled forward rendering are:

$$K_{cmd} = \{o \times o_{cmd}\}_1 + \{l \times l_{cmd}\}_2 + \{o \times o_{cmd}\}_3$$
$$K_{proc} = \{ov \times ov_{proc}\}_1 + \{lv \times lv_{proc}\}_2 + \{ov \times ov_{proc} + lf \times eq_{proc}\}_3$$
$$K_{mem} = p \times 2 + mmll$$
$$K_{band} = \{ov \times ov_{band} + of \times 2\}_1 + \left\{lv \times lv_{band} + \frac{lf}{t}\right\}_2 + \left\{ov \times ov_{band} + p \times of_{band} + lf_{band} \times \frac{lf}{t} + p \times 2)\right\}_3$$

4.9 Forward+

Even with the usage of tiles there are many situations where the illumination equation is evaluated for false positives. The tiles do not offer to possibility of detailed light-object intersection on the depth axis, therefore even if a light could be easily determined to be non-intersecting with any object rasterized on that tile, it would still be added to the tile light list. Forward+ (***F+***) is an advanced variant of tile based forward rendering where a depth mask is kept for each tile. The algorithm, known as 2.5D culling is depicted in Figure 6.

Fig. 6. 2.5D culling. Low resolution depth occupancy maps are used to quickly solve intersection tests. Lights not intersecting tile geometry are not used in lighting operations.

The depth mask bits are set for the conservatively approximated depth intervals where an object is rasterized over the tile. When the light pass is executed, a depth mask is

created for each light and it is compared to the objects depth mask of the tile. If the masks do not intersect than the light is culled, if they intersect the light is added to tile light list. In order to establish the best depth interval for the object depth mask the objects can be rendered twice: once they write only depth, to establish the minimum and maximum bounds per tile. The second time the objects are rendered to determine the exact mask. The performance metrics for Forward+ are:

$$K_{cmd} = \{o \times o_{cmd}\}_1 + \{l \times l_{cmd}\}_2 + \{o \times o_{cmd}\}_3$$
$$K_{proc} = \{ov \times ov_{proc}\}_1 + \{lv \times lv_{proc}\}_2 +$$
$$+ \{ov \times ov_{proc} + lf \times (1 - lcp) \times eq_{proc}\}_3$$
$$K_{mem} = p \times 3 + memll$$
$$K_{band} = \left\{lv \times lv_{band} + \frac{lf}{t} \times (1 - lcp)\right\}_2 +$$
$$\left\{ov \times ov_{band} + p \times of_{band} + lf_{band} \times \frac{lf}{t} \times (1 - lcp)\right\}_3$$
$$+ p_3 + \{ov \times ov_{band} + of\}_1$$

As scenes contain more lights the lcp factor grows and the algorithm obtains superior performance compared to normal tiled variants. This algorithm is especially optimized for scenes where the ratio between light complexity and geometric complexity is high. Furthermore the algorithm can be adjusted towards a light pre-pass stage structure in order to work with hardware MSAA and to handle transparency.

Also, if the vertex processing cost is deemed too high the depth masks can be programmed to use the entire depth resolution, therefore a single object rendering pass is necessary. On the other hand, as expected, this will decrease light culling efficiency.

4.10 Clustered Deferred

Clustered rendering follows the evolution path of tiled rendering and forward+ by improving the light object explicit acceleration structure. Instead of using tiles and depth masks per tile, the acceleration structure for light object intersection is a tridimensional grid, where each cluster will manage a list of lights. Basically, each tile from tiled deferred is divided into multiple entries and instead of using a single bit per depth partition an entire cluster is used.

For clustered rendering the same interval computation method is used as the one in Forward+. The scene objects are rendered twice: once to determine the depth interval per screen space tile and once to populate the clusters.

In the lighting stage instead of adding all the lights to a single list per tile, they are added to the parent cluster. If a cluster is guaranteed to not intersect any objects then the entire cluster is discarded. If no depth distribution information is available, for example when only an object rendering pass is used, then the clusters can be constructed for the entire depth resolution, with suboptimal results but with cheaper vertex processing and GPU command costs.

The light clustering technique can be used for both deferred rendering and for light pre pass rendering resulting in clustered deferred (***CD***), respectively clustered forward (***CF***). The performance metrics for clustered deferred rendering:

$$K_{cmd} = \{o \times o_{cmd}\}_1 + \{l \times l_{cmd}\}_2$$
$$K_{proc} = \{ov \times ov_{proc}\}_1 + \{lv \times lv_{proc}\}_2$$
$$+\{lf \times (1 - lcp) \times eq_{proc}\}_3$$
$$K_{mem} = p \times 3 + memll$$
$$K_{band} = \{ov \times ov_{band} + of \times (of_{band} + 3)\}_1$$
$$+ \left\{lv \times lv_{band} + \frac{lf}{c} \times (1 - lcp)\right\}_2$$
$$+ \left\{ lf_{band} \times \frac{lf}{c} \times (1 - lcp) + p \times 4\right\}_3$$

The performance metrics for clustered forward rendering:

$$K_{cmd} = \{o \times o_{cmd}\}_1 + \{l \times l_{cmd}\}_2 + \{o \times o_{cmd}\}_3$$
$$K_{proc} = \{ov \times ov_{proc}\}_1 + \{lv \times lv_{proc}\}_2 +$$
$$+ \{ov \times ov_{proc} + lf \times (1 - lcp) \times eq_{proc}\}_3$$
$$K_{mem} = p \times 2 + memll$$
$$K_{band} = \{ov \times ov_{band} + of \times 3\}_1 +$$
$$\left\{lv \times lv_{band} + \frac{lf}{c} \times (1 - lcp)\right\}_2 + p_3$$
$$\left\{ov \times ov_{band} + p \times of_{band} + lf_{band} \times \frac{lf}{c} \times (1 - lcp)\right\}_3$$

The depth distribution of the clusters can also be different from uniform, for example they can be logarithmic.

4.11 Deferred++

Deferred++ (***D++***), also known as the Visibility Buffer algorithm (Burns et al., 2013), is a deferred technique that has 4 stages. Its main focus is to reduce both memory and bandwidth requirements. The process is shown in Figure 7.

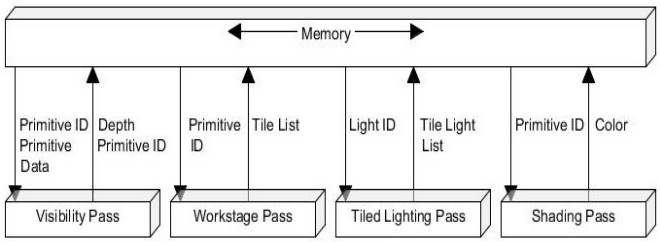

Fig. 7. The Visibility Buffer and its four stages: visibility determination, workpass and tile list construction, a tiled light pass, and a shading pass.

In the first visibility determination stage, instead of using a normal G-buffer like structure, it stores only the primitive indices of the intersecting objects and the depth on multiple visibility samples per pixel. In the second stage, named workstage pass, the visibility entries from the previous pass are read and pairs of tile index and shader index are created. These pairs are sorted by the shader index. The third stage is a tiled light pass, identical to the one used in tiled deferred rendering. In the final stage each shader is executed over its list of tiles, offering improved cache coherency for shading. In the compute shader based shading pass the entire rasterization process is executed programmatically. The primitive indices from the tile are read and the vertex shader function inside the compute shader is executed for each of them. The vertices are then rasterized programmatically and their attributes are interpolated. The resulting in-tile

fragments are processed with the fragment function inside the same compute shader.

On a local level, the intersection acceleration structure acts as a many-to-many object-light intersection instead of the one-to-many type of intersection found in the explicitly or implicitly accelerated deferred algorithms. Because of this increased information at the tile level, multiple local samples can be easily obtained and they can also be sorted, therefore motion blur, depth of field and transparency can be implemented without extra impediments. Let $tile_{cmd}$ be GPU command cost of issuing one workpass tile execution, $sort_{cost}$ be the cost of sorting the workpass tiles. The performance metrics for Deferred++ are:

$$K_{cmd} = \{o \times o_{cmd}\}_1 + \{l \times l_{cmd}\}_3 + \{t \times tile_{cmd}\}_4$$
$$K_{proc} = \{ov \times ov_{proc}\}_1 + \{sort_{cost}\}_2 + \{lv \times lv_{proc}\}_3 +$$
$$+ \left\{\frac{ov}{t} \times ov_{proc} + lf \times (1 - lcp) \times eq_{proc}\right\}_3$$
$$K_{mem} = p \times 3 + memll$$
$$K_{band} = \{ov \times ov_{band} + of \times 2\}_1 + \left\{\frac{p}{t} \times 2\right\}_2$$
$$+ \left\{lv \times lv_{band} + \frac{lf}{t} \times (1 - lcp)\right\}_3 + p_4$$
$$+ \left\{\frac{ov}{t} \times ov_{band} + p \times of_{band} + lf_{band} \times \frac{lf}{t} \times (1 - lcp)\right\}_4$$

4.12 Decoupled Deferred Rendering

Decoupled Sampling for rendering was introduced in (Ragan-Kelley et al., 2011) and it was inspired by the REYES rendering pipeline (Cook et al., 1988). The algorithm acts as a generalization of MSAA and it completely decouples the visibility samples and shading samples. It creates a many-to-one relationship between visibility samples and shading samples which dramatically improves efficiency in evaluating effects that require multiple shading samples per fragment such as depth of field or motion blur. For example a moving surface from a primitive might be rasterized over a different number of pixels in a single time frame. This is very common since no frame is instantaneous and objects will move during the frame time, creating the visual effect of motion blur. Normal rasterization rendering requires evaluating and shading this surface more than once for the correct computation of motion blur, but with decoupled sampling all the visibility samples generated by the moving surface will point to the same shading sample, therefore the surface would be only once shaded, and a substantial amount of computation and bandwidth is saved.

In order to implement decoupled sampling a mapping mechanism is required, to map the visibility samples to the shading samples. This structure is named a memoization cache. It acts as a first in first out queue, which holds the most recent mappings between visibility samples and shading samples. It is implemented as queue and not as a map in order to fit into the small caches available to GPUs. Furthermore, geometry exhibits data access patterns of high locality, thus a large cache would mostly not improve performance. The memorization cache is inefficient on GPUs, as it needs synchronization. The concept is described in Figure 8.

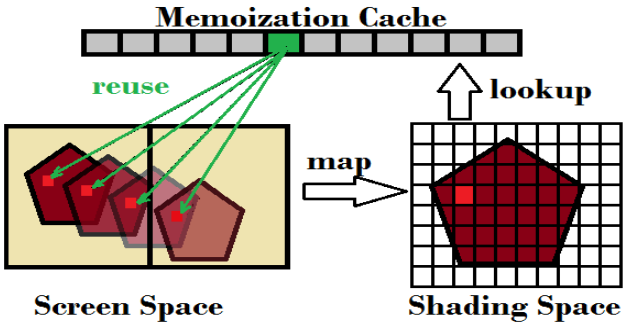

Fig. 8. Decoupled sampling. Shading samples are reused instead of being recomputed for each visibility determination sample.

The concept of decoupled sampling was applied to deferred rendering in (Liktor et al., 2012). The structure of the geometry buffer was altered and the new memory structure is named Compact Geometry Buffer (CG-buffer). In Deferred Decoupled Sampling (***DDS***) instead of keeping a structure similar to that displayed in Figure 1, the CG-buffer is more similar to the structures used in List Based Light Indexed Deferred Rendering. The CG-buffer stores the shading sample (position, normal, albedo, reflectance, etc) data. Each entry in the CG-buffer is referenced by visibility samples. For example the 4 visibility samples from Figure 8 would all reference the same CG-buffer entry obtaining a significant bandwidth consumption reduction. The CG-buffer also contains a screen sized multisampled memory structure for visibility samples and depths.

The algorithm works in 3 stages, similar to normal deferred rendering. In the geometry stage the CG-buffer is filled as following: prior to rendering each object primitive is assigned a unique surface shading id range, named $ssid\ range$. With this id and a hash function each fragment can uniquely identify which is the shading sample $ssid$ that points to the real shading data that would normally be stored in a G-buffer. If the indicated shading sample data was previously read and stored, the current visibility sample just references it. Otherwise, the current visibility sample reads, evaluates and stores the shading sample. The lighting stage is identical to that in tiled deferred rendering, keeping only the indices of the potential contributor lights. In the final stage, the radiance is evaluated by intersecting the lights stored in the lighting stage with the shading samples referenced by the visibility samples from the CG-buffer. Let ch_{band}, ch_{proc} and $cache_{\%}$ be the bandwidth cost, processing cost and hit percentage of searching the cache for a reference to a shading sample. The performance metrics are:

$$K_{cmd} = \{o \times o_{cmd}\}_1 + \{l \times l_{cmd}\}_2$$
$$K_{proc} = \{ov \times ov_{proc} + vsppe \times of \times ch_{proc}\}_1 +$$
$$\{lv \times lv_{proc}\}_2 + \{p \times sppe \times eq_{proc}\}_3$$
$$K_{mem} = vspp \times p \times 2 + sspp \times p \times 3 + memll$$
$$K_{band} = \{ov \times ov_{band} + vsppe \times of \times 2\}_1 +$$
$$\{(1 - ch_{\%}) \times sppe \times ch_{band} \times of \times (of_{band} + 3)\}_1 +$$
$$\{lv \times lv_{band} + lf\}_2 + \{lf \times (lf_{band} + 3) + p \times 3 \times sppe\}_3$$

Because of the bottleneck created through the high cost of atomic operations in the memoization cache an alternative version of the algorithm is provided in (Liktor et al., 2012). The geometry stage is solved through three passes. The first pass uses rasterization to store the $ssid$ for each visibility sample, without storing the shading sample data. The second pass uses compute shader tiles to map the visibility samples to shading samples, making memory accesses more coherent. The last pass uses rasterization to render the objects and store the shading samples. Therefore the algorithm has five stages. Let $tile_{cmd}$, $tile_{proc}$ and $tile_{band}$ be the command, processing and bandwidth costs of a tile. The algorithm has the following performance metrics:

$$K_{cmd} = \{o \times o_{cmd}\}_1 + \{t \times tile_{cmd}\}_2 + \{o \times o_{cmd}\}_3 + \{l \times l_{cmd}\}_4$$
$$K_{proc} = \{ov \times ov_{proc}\}_1 + \{tile_{proc}\}_2 + \{ov \times ov_{proc}\}_3 + \{lv \times lv_{proc}\}_4 + \{p \times ssppe \times eq_{proc}\}_5$$
$$K_{mem} = vspp \times p \times 2 + sspp \times p \times 3 + memll$$
$$K_{band} = \{ov \times ov_{band} + vsppe \times aofp \times p \times 2\}_1 + \{t \times tile_{band}\}_2 + \{lv \times lv_{band} + alfp \times p\}_4 + \{ov \times ov_{band} + ssppe \times p \times (of_{band} + 3)\}_3 + \{alfp \times p \times (lf_{band} + 3) + p \times 3 \times ssppe\}_5$$

4.13 Sort Based Deferred For Decoupled Sampling

Sort based deferred for decoupled sampling (***SbDDS***) was introduced in (Clarberg et al., 2013). Similar to the $ssid$ shading sample identifiers, they define shading point identifiers, $spid$. These shading point identifiers differ from $ssid$ in that they also contain a Morton encoded Z-order that works with 2x2 quads in order to provide implicit derivative information during rasterization. Compared to decoupled deferred rendering it does not need to use a memoization cache as the $spid$ includes both primitive ID, the shading space 2x2 quad and the exact tile in the quad.

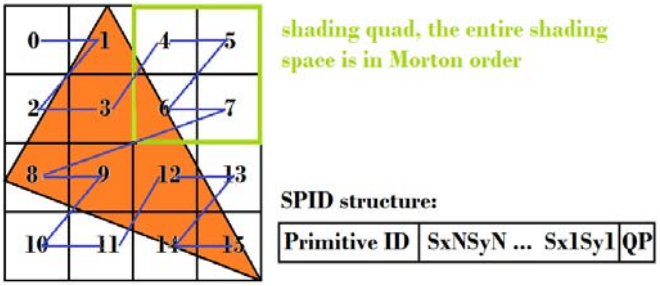

Fig. 9. SPID in sort based decoupled sampling. Data from the rasterized primitive is stored and shaded in Morton order.

The algorithm runs in four passes. In the first pass the objects are rasterized and for each visibility sample $spid$ and depth are stored. In the second pass the $spids$ are sorted in a compute shader, obtaining an order that maximizes coherent memory accesses. In the third pass a tiled lighting stage identical to the one in tiled deferred rendering is executed. In the fourth and final pass a compute shader loads each tile, sorts the potentially contributing primitives that had their ids stored in the $spids$ and then runs vertex shading, vertex interpolation and fragment shading functions. The performance metrics are:

$$K_{cmd} = \{o \times o_{cmd}\}_1 + \{t \times tile_{cmd}\}_2 + \{l \times l_{cmd}\}_3 + \{t \times tile_{cmd}\}_4$$
$$K_{proc} = \{ov \times ov_{proc}\}_1 + \{tile_{proc}\}_2 + \{lv \times lv_{proc}\}_3 + \{ov \times ov_{proc} + p \times ssppe \times eq_{proc}\}_4$$
$$K_{mem} = vspp \times p \times 2 + sspp \times p \times 3 + memll$$
$$K_{band} = \{lf \times (lf_{band} + 3) + p \times 3 \times ssppe\}_4 + \{ov \times ov_{band} + ssppe \times of \times 3\}_1 + \{t \times tile_{band}\}_2 + \{lv \times lv_{band} + lf\}_3 + \{(1 - vcull) \times ov \times ov_{band}\}_4$$

Besides using decoupled sampling and saving the bandwidth for multiply referenced shading samples, this algorithm also saves vertex bandwidth through the sorting and culling of occluded primitives, at tile level. Therefore the bandwidth for vertex attributes and for their interpolation is used only when it will affect shading. Furthermore, this algorithm does not require a complex cache mechanism.

5. CONCLUSIONS

We have introduced a set of measuring functions with which deferred rendering algorithms can be analyzed and compared. The introduced performance metrics have both practical and academic value since they depict in a clear and comparable way the strengths and weaknesses of the deferred rendering algorithms.

Because of the large variety of rendering architectures and hardware platforms no single algorithm can always represent the best choice, but with the introduced functions a quick assessment can easily be obtained. A high level comparison of the algorithms is given in Appendix A and Appendix B.

For the future we are interested in extending our work to better measure rendering algorithms that heavily rely on synchronization functions, such as those used in decoupled rendering. We are also interested in researching how our newly introduced measuring functions could be used to swap deferred algorithms dynamically during rendering.

5. ACKNOWLEDGEMENTS

Part of the research presented in this paper was supported by the Sectoral Operational Program of Human Resources Development, 2014-2015, of the Ministry of Labor, Family and Social Protection through the Financial Agreement POSDRU/159/1.5/S/134398 between University POLITEHNICA of Bucharest and AM POS DRU Romania.

REFERENCES

Burns C., Hunt W. (2013), The visibility buffer: a cache friendly approach to deferred shading, *Journal of Computer Graphics Techniques*, vol. 2, no. 2.

Clarberg P., Toth R., Munkberg J. (2013), A sort based deferred shading architecture for decoupled sampling, in *ACM Transactions of Graphics (Proceedings of SIGGRAPH 2013)*, vol 32(4), pp 141:1-141:10, July.

Cook H., Carpenter L., Catmull E. (1988), The Reyes image rendering architecture, in *Tutorial: computer graphics, image synthesis*, pp 28-35.

Crassin C., Neyret F., Sainz M., Green S., Eisemann E. (2011), in *ACM SIGGRAPH 2011 Talks*, Article no. 20.

Dachsbacher C., Stamminger M. (2005), Reflective Shadow Maps, in *Proceedings of the 2005 Symposium on Interactive 3D Graphics and Games*, pp. 203-231.

Davidovic T., Georgiev I., Slusallek P. (2012), Progressive Light Cuts for GPU, *ACM SIGGRAPH* 2012 Talks.

Deering M., Winner S., Schediwy B., Duffy C., Hunt N. (1988), The triangle processor and normal vector shader: A VLSI system for high performance graphics , in *Proceedings of the 15th annual conference on computer graphics and interactive techniques*, pp 21-30.

Harada T. (2012), A 2.5D culling for forward+, GDC, 2012

Hoef M. (2013), Hybrid Deferred Rendering. unpublished.

Jimenez J., Gutierrez D., Yang J., Reshetov A., Demoreuille P., Berghoff T., Perthuis C., Yu H., Mcguire M., Lottes T., Malan H., Persson E., Andreev D., Sousa T. (2011), Filtering approaches for real-time anti aliasing, *ACM SIGGRAPH Courses*.

Kirscher S., Lawrence A. (2009), Inferred Lighting: fast dynamic lighting and shadows for opaque and translucent objects, in *Proceedings of the 2009 ACM SIGGRAPH Symposion of Video Games*, pp. 39-45..

Koonce R. (2007), Deferred Shading in Tabula Rasa, *GPU Gems 3*, Chapter 19.

Lauritzen A. (2012), Intersecting Lights, in the *39th International conference and Exhibition on Computer Graphics and Interactive Techniques*, SIGGRAPH.

Lee M. (2009), Pre-Lighting in Resistance 2, *GDC*.

Liktor G., Dachsbacher C. (2012), Decoupled deferred shading for hardware rasterization, in *Proceedings of the ACM SIGGRAPH Symposion on Interactive 3D Graphics and Games*, pp. 143-150.

Mara M., Mcguire M., Luebke D. (2013), Lighting Deep G-Buffers: A single-pass, layered depth images with minimum separation applied to indirect illumination, *NVIDIA Corporation*, December.

Olsson O., Sintorn E., Kämpe V., Billeter M., Assarson U. (2014), Implementing Efficient Virtual Shadow Maps for Many Light, in *SIGGRAPH '14: ACM SIGGRAPH* 2014.

Olsson O., Assarsson U. (2011), Tiled Shading, *Journal of Graphics GPU and Game Tools*, vol 15, nr 4, pp 235-251.

Olsson O., Billeter M., Assarsson U. (2012), Clustered Deferred and Forward Shading, in *Proceedings of the Conference on High Performance Graphics*.

Pangerl D. (2009), Deferred rendering transparency, *ShaderX7*

Ragan-Kelley J., Lehtinen J., Chen J., Doggett M., Durand F. (2011), Decoupled Sampling for Graphics Pipelines, *ACM Transactions of Graphics*, Vol. 30, Iss. 3, May.

Reshetov A. (2012), Reducing Aliasing Artifacts Through Resampling, in *High Performance Graphics*.

Ritschel T., Grosch T., Kim M., Seidel H., Dachsbacher C., Kautz J. (2008), Imperfect Shadow Maps, in *ACM Transaction of Graphics*, vol. 27, nr. 5.

Ritschel T., Grosch T., Seidel H. (2009), Approximating dynamic global illumination in image space, in *Proceedings of the 2009 symposium on the interactive 3D graphics and games*, pp 75-82.

Shishkovtsov O. (2005), Deferred Shading in *S.T.A.L.K.E.R., GPU Gems 2: Programming and Techniques for High Performance Graphics and General Purpose Computation*, Chapter 9.

Thibieroz N. (2009), Deferred Shading with Multisampling Anti-Aliasing in DirectX 10, *ShaderX 7*.

Tokuyoshi Y., Ogaki S. (2012), Real-time bidirection path tracing via rasterization, in *Proceedings of the ACM Symposium on Interactive 3D Graphics and Games*.

Treblico D. (2009), Light Indexed Deferred Rendering, *ShaderX 7*.

Appendix A. FIRST APPENDIX

Algorithm	Light-Object ∩Acc.	Trans-parency support	HW MSAA support	Light access pattern	Material access pattern
F	imp	y	y	rand	rand
DPP	imp	n	y	rand	rand
D	imp	n	n	seq	rand
DD	imp	p	n	seq	rand
LPP	imp	y	y	seq	rand
DT	imp	p	n	seq	rand
LiD	exp	n	n	rand	seq
LiF	exp	y	y	rand	seq
LLiD	exp	n	n	rand	seq
LLiF	exp	y	y	rand	seq
TD	exp	n	n	rand	seq
TF	exp	y	y	rand	seq
F+	exp	y	y	rand	seq
CD	exp	n	n	rand	seq
CF	exp	y	y	rand	seq
D++	dec	y	y	seq	seq
DDS	dec	y	n	seq	seq
SbDDS	dec	y	y	seq	seq

imp=implicit, exp=explicit, dec=decoupled, rand=random, seq=sequential, p=partial

Appendix B. SECOND APPENDIX

Algorithm	Decouple rates	GPU Cmd cost	Proc Cost	Alloc Mem	Band-width
F	n , n	++	+++	---	+++
DPP	n , n	++	++	-	-
D	n , n	--	-	++	++
DD	n , n	-	+	+++	+++
LPP	n , n	++	+	+	+
DT	n , n	-	++	+++	+++
LiD	n , n	-	-	+	++
LiF	n , n	++	+	+	+
LLiD	n , n	-	-	+	++
LLiF	n , n	+	+	+	+
TD	n , n	-	-	++	++
TF	n , n	+	+	+	+
F+	n , n	+	+	+	+
CD	n , n	-	-	++	++
CF	n , n	+	+	+	+
D++	y , n	+	+++	+	-
DDS	y , n	+	+++	++	--
SbDDS	y , y	-	+++	++	--

The metrics are given for large numbers of lights and objects.