ForwardPlus11

AMD Developer Relations

Overview

This sample provides an example implementation of the Forward+ algorithm, which extends traditional forward rendering to support high numbers of dynamic lights while maintaining performance. It utilizes a Direct3D 11 compute shader (DirectCompute 5.0) to divide the screen into tiles and quickly cull lights against those tiles, resulting in per-tile light lists for the forward pixel shader.

While deferred rendering can also be used to support high numbers of dynamic lights, achieving high performance with MSAA enabled can be difficult in a deferred renderer. In contrast, MSAA is easy with the Forward+ algorithm, which utilizes the built-in MSAA capabilities of the hardware as intended.

Implementation

The Forward+ algorithm consists of three steps.

1. Depth pre-pass
2. Tiled light culling
3. Forward shading, using the per-tile light lists from Step 2

Depth Pre-Pass

A depth pre-pass is commonly used in forward rendering to avoid executing the pixel shader on a pixel that is later overwritten. This is especially important for Forward+, because the shader may be looping over many lights. For performance reasons, it is important to avoid paying that cost for pixels that will be invisible in the final image.

For Forward+, the depth pre-pass also serves to produce the data needed to calculate min and max depth per tile during light culling. We will discuss the use of the min and max depth in more detail in the Tiled Light Culling section.

When performing a depth pre-pass, be sure to enable depth-only rendering by setting a null color buffer with `OMSetRenderTargets`. Depth-only rendering is much faster than standard rendering. Also, set a null pixel shader for opaque geometry and a simple discard pixel shader for alpha test geometry.

Tiled Light Culling

The key to the Forward+ algorithm is the tiled light culling step. During this step, the screen is divided into fixed-size tiles, and a DirectCompute 5.0 compute shader is used to calculate a list of lights for each tile.

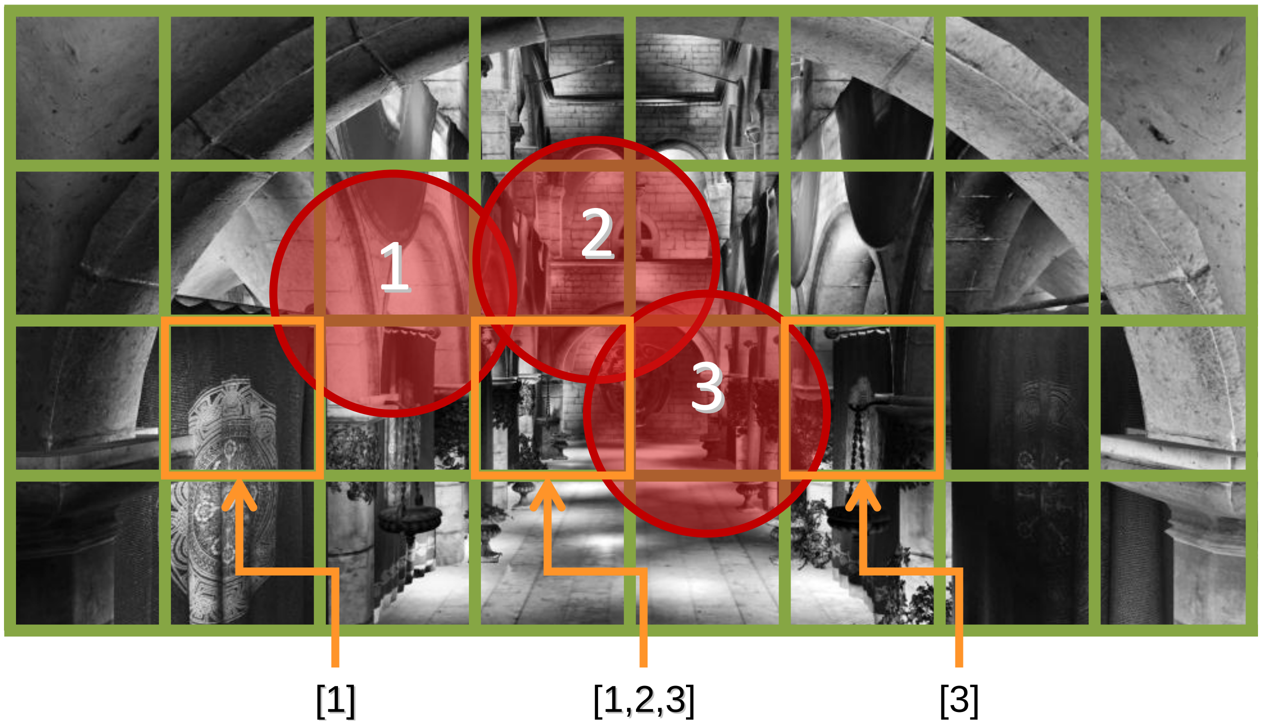

Figure 1: Simplified illustration of screen-space tiles

Figure 1 gives a simplified illustration of this process. To perform the culling, a compute shader thread group is executed per tile. For example, a 16x16 thread group configuration corresponds to a 16x16-pixel tile size. Each tile (i.e. each thread group) builds a per-tile list of lights that intersect that tile. The per tile lists store light indices into the global light list.

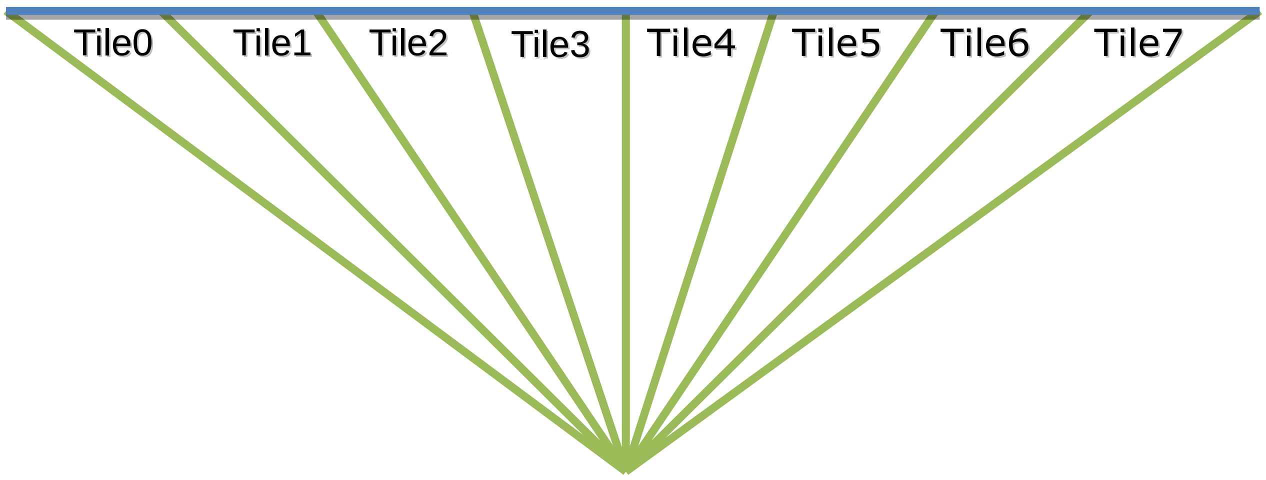

Figure 2: Top-down 2D illustration of screen-space tiles, showing how the view frustum is divided

Figure 2 shows how the view frustum is divided to perform the per-tile culling. Each compute shader thread group calculates an asymmetric frustum in view space that fits around the tile.

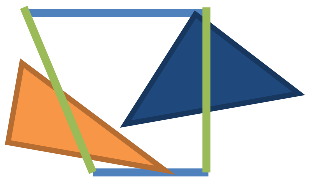
Figure 3: Bound the frustum with min and max depth

Then, to make the frustum fit more tightly around the scene geometry for the current frame, the min and max depth per tile is calculated using the depth buffer from the depth pre-pass. Figure 3 gives a simplified illustration of this process. This smaller frustum is used to build the per-tile light list by testing each light's bounding sphere against the frustum.

For each light that intersects the frustum, the compute shader performs a thread-safe write (using atomics) of the index of that light to the per-tile list in thread group shared memory. Once all threads in the thread group have finished culling lights, the list in thread group shared memory is written out to an Unordered Access View (UAV). The end of the list is marked with a sentinel to avoid the need to keep a separate list of per-tile light counts.

Forward Shading

Forward shading for Forward+ is the same as classic forward rendering, except that the light list is now per tile from the compute shader, instead of per object in shader constants. Ease of implementation is one of the attractive qualities of the Forward+ algorithm.

Performance

The performance of the Forward+ algorithm is quite good on modern graphics hardward. For example, culling 1024 lights takes 0.46 milliseconds for 1x MSAA at 1080p fullscreen on a Radeon HD 7970 GHz Edition. For 4x MSAA, culling 1024 lights takes only 0.59 milliseconds. “Cheap and easy” MSAA is one of the advantages of the Forward+ algorithm.

Total frame time will depend on the expense of your light loop during forward shading. For the sample, it takes 2.28 milliseconds total frame time for 1024 lights, 4x MSAA. This total frame time includes the depth pre-pass, tiled light culling, and forward shading, but does not include other rendering such as drawing the HUD and text.

Performance Considerations

While the illustrations in the Tiled Light Culling section show very large tile sizes (for simplicity), the actual tile size in the sample is 16x16 pixels. This tile size strikes a balance between compute shader performance and culling efficiency. Recall that the compute shader is organized such that a thread group executes per tile. A larger number of threads per thread group will often lead to better compute shader performance. Therefore, the compute shader prefers larger tile sizes. However, a larger tile size means coarser culling, which results in an increased likelihood that a light that intersects a particular tile will not actually affect many of the pixels within that tile. This means that the forward pixel shader will be processing more false positive light intersections. Refering to Figure 4, note that 16x16 provides the best performance.

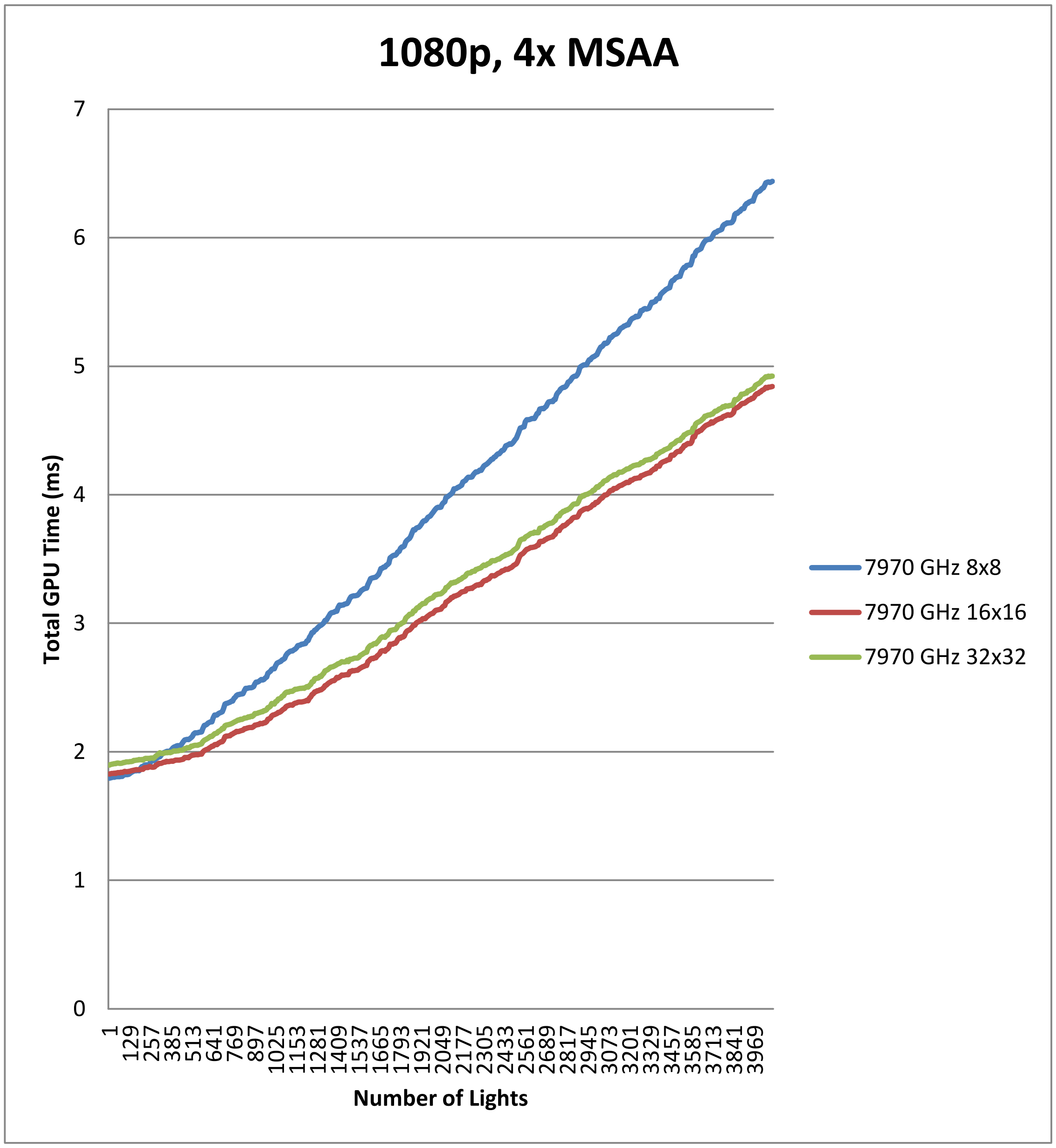

Figure 4: Tile size comparison

Feedback

If you have any feedback on this sample please send it to: jason.stewart@amd.com

Advanced Micro Devices
One AMD Place
P.O. Box 3453
Sunnyvale, CA 94088-3453

www.amd.com
http://ati.amd.com/developer