A Texture Streaming Pipeline for Real-Time GPU Ray Tracing

Mark Lee
Walt Disney Animation Studios
Burbank, USA
mark.st.lee@gmail.com

Nathan Zeichner
Walt Disney Animation Studios
New York, USA
nathan.zeichner@disneyanimation.com

Yining Karl Li
Walt Disney Animation Studios
Burbank, USA
karl.li@disneyanimation.com

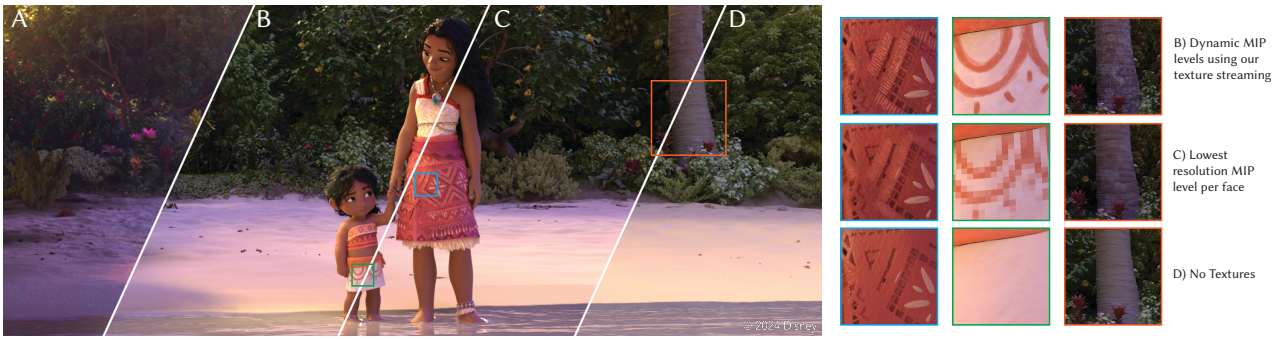

Figure 1: A scene from *Moana 2*, rendered using our real-time GPU ray tracer (B, C, D) and compared with the final frame (A) from Disney's Hyperion Renderer. Rendering without textures (D) is not a useful previsualization for (A). Without streaming, only the lowest resolution MIP tile per Ptex face can fit on the GPU (C). With our texture streaming, we handle 1.5 TB of Ptex files on disk using only 2 GB of GPU VRAM to achieve a result (B) that matches the texture detail of (A) while maintaining >95% of the average performance of (D), without stalls.

Abstract

Disney Animation makes heavy use of Ptex [Burley and Lacewell 2008] across our assets [Burley et al. 2018], which required a new texture streaming pipeline for our new real-time ray tracer. The goal from the start was to create a scalable system which could provide a real-time, zero-stall experience to users at all times, even as the number of Ptex files expand into the tens of thousands. We cap the maximum size of the GPU cache to a relatively small footprint, and employ a fast LRU eviction scheme when we hit the limit.

ACM Reference Format:
Mark Lee, Nathan Zeichner, and Yining Karl Li. 2025. A Texture Streaming Pipeline for Real-Time GPU Ray Tracing. In *SIGGRAPH '25: Aug 10 – Aug 14, 2025*. ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3721239.3734098

1 Related Work

While virtual texturing is a well established technique in both offline rendering [Peachey 1990] and in games [van Wavern 2009], our new system had the novel core requirement of supporting Ptex on the GPU. Our approach to GPU texture sampling is inspired by Lee et al. [2017]. A more rigorous analysis of this "filter after shading" approach can be found in Pharr et al. [2024]. The OptiX Toolkit provides a general streaming mechanism [Nvidia 2024] which can be used to page texture or geometry assets on demand to the GPU; however, Ptex brings its own set of requirements, such as need for supporting widely varying face dimensions within a single Ptex file, meaning the system needs to efficiently handle allocations from 8 bytes up to 64k and everything in between.

SIGGRAPH '25, Vancouver, BC, Canada

ACM ISBN 978-1-4503-XXXX-X/2025/06
https://doi.org/10.1145/3721239.3734098

2 System Overview

The LRUCache: The LRUCache is used to map a large address space to a much smaller physical pool of memory. We use a slab allocator [Bonwick 1994] as the underlying memory allocation mechanism. This scheme is well suited to the allocation patterns required by Ptex tiles/MIPs. Each slab always has a fixed size of 512kb, but the number of slots each slab gets divided into is not decided until runtime and is scene dependent. The eviction scheme is discussed later in the "Eviction Mechanism" section.

Tiles and TileKeys: Ptex is a hierarchical format, where the top level has one or more faces. Within each face there is a full MIP-chain, which contains progressively lower resolution versions of the highest resolution face. Furthermore, higher resolution MIP-levels in the chain may be tiled if the resolution exceeds a threshold. These texture tiles are the granularity we chose to cache at.

When a shader wants to sample a Ptex face, it does so by generating a 64-bit TileKey (see Table 1) and queries a hash table to retrieve a concrete GPU memory address where the texels live. Of course it is possible these texels were not uploaded to the GPU yet, so if the TileKey query does not find an entry then the requested TileKey is added to a tile request buffer. A fallback 1x1 tile is used instead, which is always resident for each tile of each face.

Uploading Tiles onto the GPU: An updated tile request buffer is asynchronously transferred back to the CPU after each render kernel execution. When the conditions are right to process the

Table 1: Ptex TileKey Definition

Bits	Index	Usage
50-63	tile	Used when a face has been split into tiles
46-49	MIP	Lower indices represent higher res MIPs
24-45	face	Up to 4,194,304 faces supported
23	flag	Set to 1 to indicate valid key
0-22	map	A maximum of 8,388,607 maps supported

latest tile request buffer (see Section 4), we deduce the memory size needed for each tile and batch allocate all of them at once from the LRUCache. This call may trigger a cascade of LRU evictions to satisfy the request but the request will always be satisfied. At that point we hand off the tile request list with the newly allocated GPU destination memory addresses for each tile to a set of CPU worker threads, whose job it is to place the TileKey texels at the associated GPU memory address. This happens asynchronously in the background and subsequent render kernel launches will simply use the most recent version of the TileKey to GPU address hash map in the meantime. These worker threads are allowed to finish at their own pace and the latest hashmap is only made visible to the GPU once they complete. This is a recurring pattern; trade off potential stalls against potentially increased latency. The amount of latency can be controlled by setting the size of the tile request queue which feeds the worker threads.

Eviction Mechanism: To ensure the LRUCache is evicting the least recently used tiles, it needs to be given information about when each tile was last touched. To accomplish this the hashmap index of each tile that was touched is recorded during the current render kernel invocation. This is stored compactly in a single bit-field whose size is equal to the number of entries in the hashmap. This bit-field contains sufficient information such that we can correlate each on-bit with the corresponding allocation in the LRUCache. The result is perfect LRU eviction with a per frame time granularity. It is worth noting that the work of filling in the tiles touched bit-field is only done when the cache detects it is close to filling up.

The act of evicting a tile in practice means that an existing resident tile will give up its memory to another tile. From the GPU's point of view, the eviction itself is not visible immediately, and will not be until one of the tile fetch worker threads transfers the texels from a requested tile into that memory address.

Probabilistic Tile Rejection: Consider what happens during the initial frames at renderer start up. Here the GPU texture cache is empty so there will be zero tile cache hits, which will cause the tile request buffer to quickly fill up. We write to the framebuffer in blocks of 8x4 pixels, starting at screen top left. Since every texture fetch will result in a new tile fetch request, the buffer will end up with many duplicate requests accounting for only the top portion of the framebuffer. Instead, ideally tile requests would be spread out evenly evenly over the framebuffer such that the total number would exactly fill the tile request buffer on average. To achieve this, shaders reject incoming requests with a probability *RP*, defined as:

$$RP = 1 - \frac{\text{unique tile requests from last frame}}{\text{total tile requests from last frame}} \quad (1)$$

As the tile cache fills up and the number of tile requests lessens, the rejection probability eventually drops to zero as the number of unique tile requests gets closer to the actual number of tile requests.

3 Texture Sampling

A custom texture sampler was developed for the GPU to support Ptex lookups. Primary rays are generated using filter importance sampling [Ernst et al. 2006], which we take advantage of to filter textures as well as anti-alias hard edges. Each time a shader samples a texture, internally we point sample a single texel from the MIP-level at each side of the idealized MIP-level, as determined by the ray differentials. This results in two texel fetches, which are then blended together. If a particular texel is not immediately available, we substitute the 1x1 fallback in its place. This is very similar to the scheme outlined in Section 6.2 of Lee et al. [2017]. For the typical case, this scheme means we can avoid having to upload Ptex face adjacency tables onto GPU. The downside of this becomes apparent when texture magnification is required however.

4 Stall Prevention

As mentioned, we want zero stalls during rendering as the number of textures increases. Section 2.3 discusses how the worker threads responsible for uploading tiles to the GPU never cause stalls.

Data transfers from the CPU to the GPU are generally not problematic since they can be queued on the same CUDA stream with their dependent render kernels. These will not cause stalls as long as the CPU memory is pinned. Note that we don't need to allocate the host memory used for the texture tile transfers as pinned. These transfers are performed synchronously on non-rendering CUDA streams by worker threads. These threads do not mark themselves as done until all of these synchronous transfers complete.

Potential stalls from data transfers from the GPU to the CPU are handled by double buffering all the data, in conjunction with pinned CPU memory. Since the texture tile transfers do not require pinned memory, the main consumer of pinned memory ends up being the double buffered hashtables, which can grow to accommodate the number of active texture tiles in the world.

5 Conclusions and Future Work

The result of integrating this cache into our real-time ray tracer was a big increase in render fidelity with little disruption to workflows. Our artists preferred greater texture latency over a reduced frame rate. In our progressive ray tracer the early samples that retrieve the fallback or lower resolution tiles lose influence over time as higher detailed textures are loaded in. The result is an artist will get a faster time to the first pixel and can make a faster judgement, without changing convergence time. We are currently working on porting this technique over to our real time rasterizer. This port provides additional challenges because of the need to randomly access the entire GPU cache from within OpenGL and Vulkan. We look forward to presenting this work in the future.

References

Jeff Bonwick. 1994. The Slab Allocator: An Object-Caching Kernel. In *USENIX Summer 1994 Technical Conference (USENIX Summer 1994 Technical Conference)*. USENIX Association, Boston, MA. https://www.usenix.org/conference/usenix-summer-1994-technical-conference/slab-allocator-object-caching-kernel

Brent Burley, David Adler, Matt Jen-Yuan Chiang, Hank Driskill, Ralf Habel, Patrick Kelly, Peter Kutz, Yining Karl Li, and Dan Teece. 2018. The Design and Evolution of Disney's Hyperion Renderer. *ACM Transactions on Graphics* 37, 3, Article 33 (Aug. 2018). https://doi.org/10.1145/3182159

Brent Burley and Dylan Lacewell. 2008. Ptex: Per-Face Texture Mapping for Production Rendering. *Computer Graphics Forum* 27, 4 (2008), 1155–1164. https://doi.org/10.1111/j.1467-8659.2008.01253.x

Manfred Ernst, Marc Stamminger, and Günther Greiner. 2006. Filter Importance Sampling. In *Proc. of IEEE Symposium on Interactive Ray Tracing*. 125–132. https://doi.org/10.1109/RT.2006.280223

Mark Lee, Brian Green, Feng Xie, and Eric Tabellion. 2017. Vectorized production path tracing. In *Proceedings of High Performance Graphics* (Los Angeles, California) *(HPG '17)*. Association for Computing Machinery, New York, NY, USA, Article 10, 11 pages. https://doi.org/10.1145/3105762.3105768

Nvidia. 2024. Optix Demand Loading Library. https://github.com/NVIDIA/optix-toolkit/tree/master/DemandLoading.

Darwyn Peachey. 1990. Texture on Demand. *Pixar Technical Memos*, Article 217 (1990).

Matt Pharr, Bartlomiej Wronski, Marco Salvi, and Marcos Fajardo. 2024. Filtering After Shading With Stochastic Texture Filtering. *Proc. ACM Comput. Graph. Interact. Tech.* 7, 1, Article 14 (May 2024), 20 pages. https://doi.org/10.1145/3651293

Jan Paul van Wavern. 2009. id Tech 5 Challenges: From Texture Virtualization to Massive Parallelization. *SIGGRAPH 2009: Beyond Programmably Shading Course Notes*, Article 7 (Aug. 2009). https://doi.org/10.1145/1667239.1667246