A Dynamically-Updating Hierarchical Stopping Condition for Monte Carlo Illumination

Keith Jeffery
DreamWorks Animation
keith.jeffery@dreamworks.com

ABSTRACT

An image-space hierarchy is introduced to reduce artifacts from prematurely stopping in adaptive sampling in a Monte Carlo ray tracing context, while maintaining good performance and fitting into an existing sampling architecture.

CCS CONCEPTS

• **Computing methodologies** → **Rendering**; **Ray tracing**;

KEYWORDS

ray tracing, mcrt, global illumination

ACM Reference Format:
Keith Jeffery. 2019. A Dynamically-Updating Hierarchical Stopping Condition for Monte Carlo Illumination. In *Proceedings of ACM SIGGRAPH*. ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/8888888.7777777

1 INTRODUCTION

The adaptive sampling framework of Dammertz et al. [2010] is expanded to account for low-probability visual effects while both fitting into the existing sampling framework for DreamWorks' MoonRay Monte Carlo renderer and fulfilling MoonRay's goal: Keep all lanes of all cores of all machines busy all the time with meaningful work.

Dammertz et al. [2010] offers an adaptive sampling framework, in which the authors specify both an error calculation and an implicit, dynamically-created hierarchy of sampling regions that gets refined as samples are added to the render buffer. These regions are added to a queue which specifies which pixels need further refinement.

The previous generation of adaptive sampling in MoonRay was influenced by the error calculation in Dammertz et al. [2010], but their hierarchy was discarded in favor of eliminating thread contention. The new adaptive sampling framework extends their work to not only fit into MoonRay's rendering framework efficiently, but to allow the hierarchy to respond to newly discovered scene information.

2 A FRAMEWORK FOR PREDEFINED TILES

MoonRay, a unidirectional path tracer, creates its primary rays for rendering through tiles of 8 by 8 pixels over the image plane. Each tile is rendered by a single thread, keeping thread contention to a minimum. While MoonRay queues and sorts many of the secondary rays, there is neither queuing nor sorting of the primary rays. As such, maintaining this tiling architecture is important for primary cache coherency. In order to maintain the tiling framework, instead of adding regions to a render queue, an explicit binary tree is maintained where the root node represents the total area of the final image (as the initial region of the Dammertz et al. does). As a region is split, two nodes are added as children. Instead of removing completed nodes, as in Dammertz et al., internal and leaf nodes can be marked as complete in the tree.

During rendering, tile scheduling is performed as usual, but each tile is checked against the binary tree. If it is completely within a completed region, the tile is skipped. If it overlaps any region that is not complete, the intersection of the tile and the tree partition is rendered as normal. The intersection is cheap to compute, and it reduces the number of pixels rendered as the image converges.

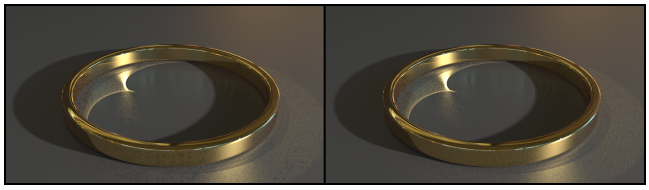

Figure 1: Left: Previous adaptive sampling with box artifacts. Right: New adaptive sampling.

3 ADAPTING TO NEW INFORMATION

The previous version of MoonRay's adaptive sampling eschewed the hierarchy of Dammertz et al. [2010] in order to minimize thread contention, having each tile check only against the error in its region. This, however, led to artifacts under some circumstances: a tile could prematurely conclude that it had converged while it may be that low-probability effects, such as caustics, had not yet been discovered. This loss of information is visually jarring as the artifacts result in tiles that do not match the features of their neighbors as opposed to additional noise (see Figure 1). It was assumed that the hierarchical method of Dammertz et al. could reduce these artifacts with more global information. However, in implementing the tree as described above, the artifacts, while slightly different in nature, persisted.

This leads to conflicting desires: early stopping and the adaption to newly-discovered image features. If the tree can adapt to new information found in neighboring tiles, previously completed tiles can be revisited. The work of Dammertz et al. already accounts for error varying over an image by non-uniformly splitting the image regions such that the error is nearly equally divided between the newly-formed halves. However, by queuing and discarding regions,

this division is never revisited. In order to adapt to new error, the algorithm presented here rebuilds the entire tree each pass through the image plane. If the error increases in a node from a newly-found feature (e.g. aliased geometry, caustics, or fireflies), the node is implicitly re-split (through rebuilding the tree), giving less area to the node with increased error and giving the neighboring node more opportunity to revisit those pixels.

3.1 Floating-point image-space division

It is natural to split the image plane on integer boundaries, as is done in [Dammertz et al. 2010]. However, in the desire to re-balance the tree, this does not often work advantageously: the image-space split location is a step function that can require a significant change in error to move, meaning newly found data may not affect the integer boundaries of the tree. With that in mind, and not being constrained to queues of image regions, the tree splitting was modified to work on floating-point values, making the split location much more sensitive to changes in calculated error (and more balanced). The pixel boundaries that mark the rendering region are rounded in such a way as to increase the bounds to the nearest integer.

3.2 Sample consistency

As pixels may be interrupted in their sampling, a buffer is used to keep count of the primary samples for each pixel. As rendering progresses, the samples are generated using this value instead of an overall sample count for the image, allowing each pixel to retain its sampling distribution.

4 THREADING

The largest challenge with this adaptive sampling technique is thread contention. As each tile is independent, but the tree is global, care must be taken in reading and updating the tree. This section details efforts in minimizing thread-contention during rendering.

4.1 Barrier-free tree updates

MoonRay progressively renders by adding samples to the tiles discussed in Section 2. The tree is updated every pass over the image plane, where the number of samples per pass increases as rendering progresses. Updating the tree on every pass makes it tempting to use a thread-barrier to update when the pass is complete. This, however, would leave a segment of time where threads are idle and tiles are not being updated.

By default, MoonRay renders tiles in an order defined by a Morton curve, which helps provide cache coherency for primary rays. In general, following a Morton curve, quadrants of an image are completed (with some overlap) in sequence. This is taken advantage of by using four trees, one for each quadrant of the image. To minimize locking, the tree is updated on a "last one out turn off the lights" basis, allowing one thread to perform an update while others continue to render in other quadrants. For error calculation, quadrants overlap each other by the size of a tile, eliminating artifacts at the seams. As the tiles are checked exclusively to the non-overlapping quadrant trees, this adds inconsequential computation time.

An atomic counter is kept for each quadrant, initially set to the number of tiles. Each tile, upon being visited, decrements this atomic counter. When the counter reaches zero, the active thread locks the tree and performs the update. For efficiency, there is no synchronization between the atomic variable and the lock for updates. Once the lock is taken, the atomic variable must be read again and incremented in such a way as to account for threads having possibly modified the atomic variable again. However, the update of this atomic variable can be made slightly more efficient because the memory fences on the atomic read and write at this point do not have to be ordered, as the secured lock will ensure proper ordering.

4.2 Reader-writer locks

In order to check a tile against the tree, a lock must be taken. However, in a film resolution image, a quadrant is read nearly 6,500 times more than it is updated. Therefore, reader and writer locks are used: shared reader locks for the common case of checking a tile, and a writer lock for the updating, which is only used once all of the threads are done rendering in that region.

4.3 Reducing mutex contention

Ideally, shared reader locks would offer no contention when locked by multiple threads. In practice, this is rarely the case. In order to further minimize thread contention, an array of shared mutexes is used, and accessed in such a way that no neighbors in 2D space access the same mutex. A lock is accessed as $y(p/2+1)+x$, where x and y are the 2D grid coordinates and p is the power-of-two length of the array. Dividing by two means each row accesses the array offset as much as possible, adding one to make it odd, ensuring that it is co-prime with the power-of-two size. Each mutex is cache-line aligned to eliminate false sharing.

5 PERFORMANCE

Rebuilding the tree from scratch each pass takes a single thread less than 20ms per quadrant. While the algorithm for checking a tile for completeness compromises no discernible time, the wait time for the locks required for checking if a tile is complete comprises 0.376% of the render time on a moderately complex production scene. Even though they require exclusive access, the locks for updating the tree show no discernible wait time due to the atomic counter and the quadrant partitioning.

6 CONCLUSIONS

The new framework for adaptive sampling performs much better on the worst-case scenarios, while improving under-sampling issues in production scenes.

REFERENCES

Holger Dammertz, Johannes Hanika, Alexander Keller, and Hendrik Lensch. 2010. A Hierarchical Automatic Stopping Condition for Monte Carlo Global Illumination. In *Proc. of the WSCG 2010.* 159–164.