Triple Depth Culling

Jean-Eudes Marvie Pascal Gautron Gaël Sourimant
{jean-eudes.marvie, pascal.gautron, gael.sourimant}@technicolor.com
Technicolor Research & Innovation

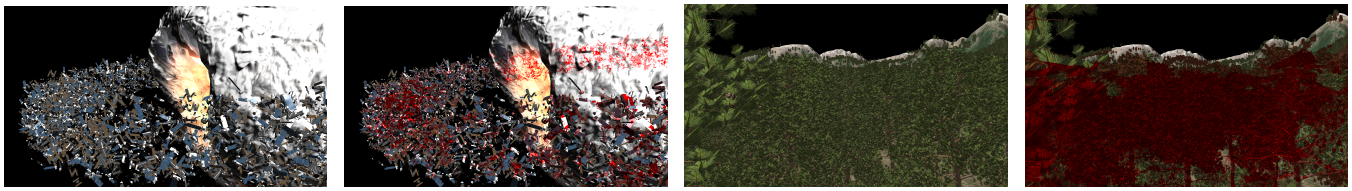

Figure 1: *Triple Depth Culling introduces controllable early depth testing for highly complex scenes with arbitrary shader operations.*

Virtual worlds feature increasing geometric and shading complexities, resulting in a constant need for effective solutions to avoid rendering objects invisible for the viewer. This observation is particularly true in the context of real-time rendering of highly occluded environments such as urban areas, landscapes or indoor scenes. This problem has been intensively researched in the past decades, resulting in numerous optimizations building upon the well-known Z-buffer technique. Among them, extensions of graphics hardware such as early Z-culling [Morein 2000] efficiently avoid shading most of invisible fragments. However, this technique is not applicable when the fragment shader discards fragments or modifies their depth value, or if alpha testing is enabled [nVidia 2008].

We introduce *Triple Depth culling* for fast and controllable per-pixel visibility at the fragment shading stage using multiple depth buffers. Based on alternate rendering of object batches, our method effectively avoids the shading of hidden fragments in a single pass, hence reducing the overall rendering costs. Our approach provides an effective control on how culling is performed prior to shading, regardless of potential discard or alpha testing operations. *Triple Depth culling* is also complementary with the existing culling stages of graphics hardware, making our method easily integrable as an additional stage of the graphics pipeline.

1 Triple Depth Culling

Our algorithm builds upon the Z-Buffer approach, in which a fragment is kept or discarded *after shading* by testing its depth against the current depth stored in the depth buffer. Early Z-culling performs this elimination beforehand, but involves restrictions on shading operations, or requires a depth-only prepass to populate the depth buffer [nVidia 2008]. Conversely, our method works in a single pass by introducing a depth culling step between the early Z-culling and the shading of fragments (Figure 2).

Starting with a roughly depth-sorted list of objects batches, a first batch is rendered into a classical RGBA buffer Z_1 and a depth buffer. For each fragment we store its depth into the alpha channel of Z_1 (or an additional render target). Then, upon rendering of a second batch of objects into Z_2, each fragment undergoes the following steps: its depth is first compared to the corresponding alpha value in Z_1. If the fragment is occluded by the first batch of objects, its output is simply discarded. Otherwise, the fragment is shaded and stored. The remainder of the object batches is then rendered using alternatively Z_1 and Z_2 as render targets.

Note that as the objects are sorted according to the viewing distance the batches tend to spread over the entire image space, increasing the screen coverage of each batch and hence the efficiency of our method.

An overhead of our method lies in the alternation of render buffers, which tends to generate pipeline stalls. We amortize the cost by adjusting the size of the object batches. This size can be adjusted

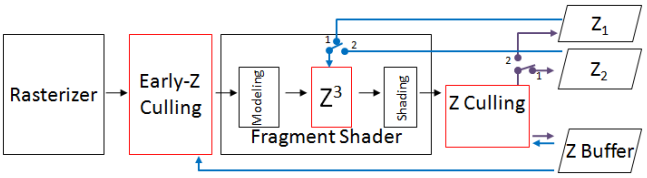

Figure 2: *Triple Depth culling is an additional, programmable depth culling step at the fragment shader stage. While rendering into Z_2, the culling is performed using the partial depth information available in Z_1 (step 1). Z_1 and Z_2 are then swapped, hence maintaining information in both buffers (step 2).*

either manually or automatically using a simple convergence based on the render time of the last frame. While the overhead does not completely vanish, the savings are significant especially in scenes containing many objects.

Our technique allows the shader to determine whether the shading must be carried out, depending on its depth output. The expensive shading is then performed only if the fragment is determined as visible. Note that this visibility determination is approximate: as each of the additional depth buffers holds only a part of the rendered fragments, some fragments may be erroneously considered as visible. While this results in unnecessary computations, this also ensures the conservativeness of our visibility algorithm and does not introduce artifacts as the corresponding fragments eventually get discarded after shading by classical depth testing.

2 Results

Our method has been implemented within fragment shaders on an nVidia GeForce GTX480. The presented scenes contains 15K and 50K objects with complex shader operations rendered at a resolution of 1280×720. Compared to classical Z-buffering, Triple Depth culling provides performance increases of 8 to 50% using batches of 50 objects, while remaining applicable in any context. We believe further performance could be achieved by implementing our approach in an additional stage in future graphics hardware for programmable fragment elimination, potentially taking advantage of the hierarchical representation of depth buffers.

Scene	# tri	# obj	Z	Early	Triple Depth
Forest	70M	50K	90 ms	N/A	44.5 ms
Asteroids	35M	15K	215 ms	N/A	198 ms

References

MOREIN, S. 2000. ATI Radeon Hyper-Z technology. In *ACM SIGGRAPH/Eurographics workshop on graphics hardware*.

NVIDIA. 2008. GPU programming guide version for GeForce 8 and later GPUs.