# Dissecting The ATI RADEON 9700 PRO

Being aware of the costs of depth complexity, ATI has put great effort into using available resources as efficiently as possible. Though not as efficient in occlusion culling as deferred rendering, ATI has thus far developed the next best thing.

The improvements found in HyperZ III have surprisingly found very little discussion, but they are certainly worth noting. The first of these is in their occlusion culling. R200 based solutions used a hierarchical Z-buffer to determine occluded pixels. In doing this, they would test the visibility of an 8x8 pixel block based on what was already rendered. To determine this, farthest visible pixel was stored as a reference value on chip for the target location of the new set of pixels. The nearest vertex of the 8x8 pixel block was compared to the reference value to determine visibility. If the nearest vertex fell behind the reference value, it was determined that the block was occluded. However, if it did not the entire block was rendered.

With such an implementation the ability to determine occlusion is severely limited. Let us assume that a single pixel out of the 8x8 pixel block is nearer to the viewer than the reference value. The entire 8x8 block must then be rendered. This results in a lot of wasted rendering, so ATI has taken steps to avoid this.

With R300 the first thing to note is that ATI no longer uses the nearest vertex to test for visibility. Rather, the minimum and maximum Z values of the polygon (or portion of such) within the block are used. This allows for a more accurate testing as the nearest vertex might not accurately reflect the actual block being tested, resulting in pixels being rendered wastefully. At this point, the 8x8 block is tested. If it is determined to be visible, the block is culled and the next block is considered. However, unlike with R200, rendered does not take place at this point if the block is determined visible.

When blocks remain visible after a visibility check, the 8x8 block is sub-divided into two 4x4 blocks. These blocks are tested for visibility just as the 8x8 block was. If either one of these blocks is determined visible, occlusion tests go to the pixel level with early Z compares.

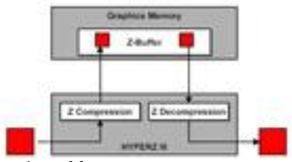
Early Z compares simply add an additional Z check just prior to the pixel pipeline. As with a final Z compare, the depth value of existing pixels are read and compared to this new pixel. If the pixel is determined to be visible, it is then rendered and displayed. If, however, it is occluded, it is culled.

One might question why ATI goes through the work of using a hierarchical Z-buffer when the last stage of HyperZ III is testing at the pixel level. Efficiency is the answer to this. Early Z checks must read each individual Z value from the Z buffer, then do a compare to determine visibility. This Z read requires bandwidth and bandwidth is what ATI is trying to conserve. A hierarchical Z can store the reference values on-chip, requiring no access to local memory (or, if stored in local memory, only a single read is required for the entire block). The result is little or no bandwidth being used while potentially culling an 8x8 or at least a 4x4 block.

## Compression

### HYPERZ III (Cont'd)

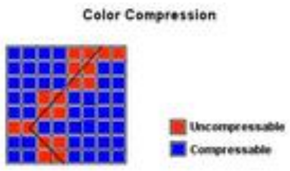
Z-buffer compression is another important aspect of HyperZ III, in that it can substantially reduce bandwidth requirements, especially in cases of anti-aliasing. The compression unit takes into consideration a pixel block, generating a plane equation by its slope. Instead of storing a Z value for each available pixel, the plane equation is stored and used for calculating the depth value of any pixel necessary within the block. Peaking at a compression ratio of 4:1, this is a best case that can only be met with one or two triangles within the block. This will often be achieved in compressing environments where larger triangles are used, but characters will not achieve this level of compression.



Z-buffer compression

Enabling anti-aliasing increases the theoretical compression ratio as more samples are used within each triangle. Theoretically, the compression ratio is increased to a peak of 24:1 in cases where 6x AA is being used. Being a peak, a 24:1 compression ratio should not be consistently expected. With that said, enabling anti-aliasing allows for the use of a new feature, namely color compression.

HyperZ III's color compression is really nothing of a surprised, as it is somewhat obvious in design. With multi-sampling anti-aliasing, each sub-pixel that does not fall along a triangle edge has a color identical to the original pixel. This same color value is then stored 2, 4 or 6 times, depending on the level of anti-aliasing. With the values being identical to the original value, storing more than one value is redundant. Examining a pixel block, the compression algorithm first determines if any of the pixels within the block fall along a triangle edge. If none of them do, a single value is stored for each sub-pixel, resulting in a compression ratio of 2x, 4x or 6x, depending on the respective anti-aliasing level. If a triangle edge is located along any of the pixels within the block, compression is either reduced or eliminated, depending on the specifics of the situation.



Color compression