The Ultimate Visual™ Experience for Gaming

Depth In-depth

Emil Persson
AMD, Inc.
emil.persson@amd.com

Introduction

With the introduction of the original Radeon the general rendering performance landscape changed significantly due to the revolutionary technology that is HyperZ. Today this technology has been upgraded, fine-tuned and improved in several generations and is now a general commodity that everyone expects to have. Yet it is sometimes not well understood or taken proper advantage of. Developing 3D applications with HyperZ in mind will often produce a significant performance increase. This paper aims to give developers a better understanding of how HyperZ operates under the hood.

Overview

HyperZ is the collective name for several depth and stencil optimizations that are incorporated into most ATI hardware for the last few generations. This includes Hierarchical Z, Early Z, Z compression and Fast Z clear. Understanding each component of HyperZ and their strengths and weaknesses is important to achieve the best performance using this technology.

Hierarchical Z

Hierarchical Z, or HiZ for short, allows tiles of pixels to be rejected in a hierarchical fashion. This allows for faster rejection of occluded pixels and offers some bandwidth saving by doing a rough depth test using lower resolution buffers first instead of reading individual depth samples. Tiles that can safely be discarded are eliminated and thus the fragment[1] shader will not be executed for those pixels. Tiles that cannot safely be discarded are passed on to the Early Z stage, which will be discussed later on.

On the Radeon HD 2000 series, HiZ can use video memory to store HiZ information. This makes it capable of storing more information than previous hardware which used an on-chip buffer to store the HiZ information. On pre-HD 2000 series hardware, this is a limited resource, so if the application is

[1] For clarity this document makes a distinction between a fragment and a pixel. Pixel refers to the data stored in a particular location in the framebuffer whereas fragment refers to the incoming data being processed. In D3D terminology the word pixel is used for both pixels and fragments.

© 2007, Advanced Micro Devices, Inc., AMD, and the AMD Arrow logo, and combinations thereof, ATI, the ATI logo, CrossFire, the CrossFire logo, Radeon and combinations thereof, and The Ultimate Visual Experience are trademarks of Advanced Micro Devices, Inc.

using many depth buffers there may not be enough space to allocate HiZ RAM for all of them, which means HiZ could get disabled for some depth buffers.

For each tile HiZ stores a low resolution minimum or maximum depth value depending on the depth comparison the application uses. For LESS and LESSEQUAL comparisons it stores the maximum value within the tile and for GREATER and GREATEREQUAL it stores the minimum. Since it only stores one of the values changing the comparison direction in the middle of a frame makes the existing values in the buffer unusable. Thus HiZ will have to be disabled until the buffer is either cleared or the application changes the comparison direction back to the original. Typically the driver will configure the buffer depending on the first depth test function used after the depth buffer is cleared.

Since the stored HiZ values are low resolution the effectiveness depends on the amount of range covered. Keeping a low far/near plane ratio can help HiZ to throw away more hidden tiles.

The Radeon HD 2000 series chips can use HiZ for both depth and stencil. However, for older hardware, HiZ only operates on depth values. For hardware prior to the Radeon HD 2000 series, all fragment rejection based on stencil values is on a per pixel granularity, and is performed in the Early Z stage which is later in the pipe. However, in many cases HiZ can still reject occluded tiles based on the depth test even when the stencil test is enabled, but there are cases where the stencil operation may interfere with HiZ causing it to be disabled. If either of z-fail or stencil-fail operations is not KEEP, HiZ gets disabled. This is because it can't reject tiles if the stencil is to be updated for any pixel in the rejected tiles. This is the case with z-fail, and it could be the case for stencil-fail (depending on the outcome of the stencil test). The stencil-pass operation doesn't interfere with HiZ though since stencil only passes if the depth test also passes, meaning that if a tile is rejected based on the depth test we're guaranteed that the stencil-pass operation won't get executed for any fragment in the tile.

A limitation that exists on previous generations of hardware (Radeon X850 and earlier) is that HiZ cannot operate with a depth test function of EQUAL. EQUAL is typically used in multipass rendering with the same geometry. In most cases using LESSEQUAL will work exactly the same and is thus recommended over EQUAL since it works nicely with HiZ. On the X1xxx generation hardware this limitation no longer exists and HiZ remains enabled even for EQUAL.

Early Z

The Early Z component operates on a pixel level and allows fragments to be rejected before executing the fragment shader. This means that if a certain fragment is found to be occluded by the current contents of the depth buffer, the fragment shader doesn't have to run for that pixel. Early Z can also reject fragments before shading based on the stencil test. On hardware prior to the Radeon HD 2000 series, early Z was a monolithic top-of-the-pipe operation, which means that the entire read-modify-

因为在D3D中，Alpha Test在Depth Test之前，如果一个Fragment的Alpha Test没有通过，则这个Fragment就会被Kill掉，但是Early-Z在Alpha Test之前就要更新Depth Buffer，所以如果开启了Alpha Test，按么Early-z就需要被关闭

write cycle is executed before the fragment shader. As a result this impacts other functionality that kills fragments such as alpha test and texkill (called "clip" in HLSL and "discard" in GLSL). If Early Z would be left on and the alpha test kills a fragment, the depth- and/or stencil-buffer would have been incorrectly updated for the killed fragments. Therefore, Early Z is disabled for these cases. However, if depth and stencil writes are disabled there are no updates to the depth-stencil buffer anyway, so in this case Early Z will be enabled. On the Radeon HD 2000 series, Early Z works in all cases.

Z compression

In order to save bandwidth reading and writing to the depth buffer a lossless tile-based compression scheme is applied. Each tile can be in various states such as cleared, compressed or uncompressed. When a tile is cleared the depth buffer need not be read to get the value of a sample, instead the cleared value is returned. This eliminates the first read from the depth buffer, which in low depth complexity scenes could be a significant chunk of the total depth reads. When the depth buffer gets updated during rendering most tiles will end up in a compressed state as depth data is typically very compressible. Writing out and reading in compressed data requires significantly less bandwidth than the raw samples, thus Z compression normally saves a sizable amount of depth buffer bandwidth. If a tile cannot be compressed, which may happen if a polygon edge crosses it or if the shader uses depth output, it will end up in an uncompressed state, which means all samples are read and written in full.

Note that Z compression does not reduce the amount of memory that is required to store the depth buffer in video memory. It only saves bandwidth. It will still need to allocate the full buffer to handle all potential uncompressed states. As with HiZ, the additional info needed to track tile states is stored in an on-chip buffer. This is a limited resource and if there isn't enough space available for some depth buffers they will not benefit from Z compression.

With the Radeon HD 2000 series, further bandwidth optimizations are made by compressing the depth and stencil buffers separately.

Also note that while technically not belonging to the HyperZ, a similar technique is used for color buffers as well to reduce bandwidth requirements.

Fast Z clear

The Fast Z clear feature is enabled by Z compression. Instead of writing out the depth clear value to the entire depth buffer the hardware can instead just set all tiles to the cleared state. This significantly reduces the burden and makes clearing the depth buffer a cheap operation.

对于一般的压缩算法来说，每个Tile只需要写一个Depth Value就好，这样就大大规约了存储器带宽的需求。也不需要像SGRAM/VRAM/WRAM那种特殊的Block Write了。

© 2007, Advanced Micro Devices, Inc., AMD, and the AMD Arrow logo, and combinations thereof, ATI, the ATI logo, CrossFire, the CrossFire logo, Radeon and combinations thereof, and The Ultimate Visual Experience are trademarks of Advanced Micro Devices, Inc.

High Precision Z

The HD 2000 series products all support high precision depth buffers which are capable of 32 bit floating point format..

Optimization hints

Render in front-to-back order and/or use Pre-Z pass

This cannot be repeated enough. This is the by far most important thing to keep in mind when optimizing for HyperZ. In order for HyperZ to be able to save any pixel processing or bandwidth you need to be able to reject at least some pixels early in the pipeline. While applications have other things to consider as well (such as the number of draw calls) there is usually room for at least a high-level depth-sort. If sorting by depth cannot easily be done without introducing a significant amount of overhead due to worse batching and more render state changes a Pre-Z pass is often a good option. A Pre-Z pass is essentially just rendering the scene with a minimal shader and color writes disabled. This will fill the depth buffer with the scene's final depth values. Using this method, the rest of the scene can be rendered with depth test enabled and depth writes disabled. Note that it's not necessary to render the entire scene in the Pre-Z pass as long as you keep track of what objects have been rendered to the depth buffer and set depth write accordingly in later passes. In some cases it may be better to limit yourself to just the objects that are expected to be significant occluders relative to their geometrical complexity. A rule of thumb when deciding what objects to include is that triangles less than 100 pixels are almost certainly not worth including in a Pre-Z pass and triangles more than 1,000 pixels should seriously be considered to be included, unless the fragment shader is trivial. This means that for instance a detailed game character in the distance with a relatively cheap pixel shader compared to the vertex shader will probably not see a gain from participating in the Pre-Z pass.

Even if the application can sort objects relatively easy there may still be a benefit of using a Pre-Z pass though. There's usually a fair amount of internal occlusion within an object (see the *Optimize models for HyperZ* section for a way of reducing this problem); and even though a Pre-Z pass takes away the need to do depth-sorting on the main rendering passes it's still beneficial to do a rough depth-sort at least on the Pre-Z pass to speed that pass up a bit (this can save depth buffer writes). Often the best approach is to do a rough depth-sort for the Pre-Z pass, and then sort the main rendering passes for other parameters such as render states, shaders or materials.

Only use alpha test, texkill and alpha to coverage when necessary

When any of these features are used, fragments or samples will potentially get killed. With depth writes enabled, Early Z will be disabled as a result. Always make sure to disable these features when they're no

© 2007, Advanced Micro Devices, Inc., AMD, and the AMD Arrow logo, and combinations thereof, ATI, the ATI logo, CrossFire, the CrossFire logo, Radeon and combinations thereof, and The Ultimate Visual Experience are trademarks of Advanced Micro Devices, Inc.

longer required. Although these features disable Early Z, in this case they are still compatible with HiZ. It is therefore recommended to render all alpha test, texkill or alpha to coverage primitives *after* opaque primitives in order to increase the chances of those being rejected by HiZ. This includes the rendering of a Pre-Z pass.

If you're using alpha test for foliage or similar and the shading is sufficiently advanced it may be beneficial to do a Pre-Z pass with the alpha test, then the main shading pass with depth writes disabled. On the X1xxx series the best method for the second pass would be to use a depth test of EQUAL, whereas on earlier hardware it would probably be better to repeat the alpha test in the second pass but stick to LESSEQUAL for the depth test.

这是因为ATI R500上对每个Tile只有一个 半开半闭区间Z Range，如果具有一个闭区间的Z Range，即一个Tile包含Nearest Fragment和Farest Fragment两个Z Value，则可以不受这个限制

Don't change depth comparison direction

Hierarchical Z can operate in two different modes: either it[1] stores the maximum Z value for each tile to work with LESS/LESSEQUAL depth test, or it[2] stores the minimum Z to work with GREATER/GREATEREQUAL. Which mode is selected typically depends on the first draw call using that depth buffer after a clear. If the comparison direction is changed the existing values in the buffer is unusable and thus HiZ will get disabled until either the buffer is cleared or depth comparison direction is restored. In order to maximally benefit from HiZ, refrain from using techniques that require the inversion of the depth compare mode.

Avoid shader depth output

Outputting depth in the fragment shader should be avoided if possible. If a fragment shader modifies the Z value to be used in the depth test, it is not available until after the fragment shader has executed, meaning that no Z optimizations are possible. As a result, Hierarchical Z and Early Z will both be disabled and the depth test will be done at the end of the pipe. Another effect of using depth output is that tiles written to through shader depth output will end up uncompressed, meaning that consecutive passes may also see a performance reduction. Instead of modifying depth to kill fragments it's generally better to use alpha test or texkill to achieve the same thing. That will keep HiZ and Z compression operating properly.

Clear the depth-stencil buffer

In order for Z compression to remain operational the depth buffers needs to get cleared regularly, typically once per frame or per pass for some rendering techniques. In the past some applications have tried to optimize by avoiding the depth clear through clever tricks. However, clearing a depth buffer is cheap since all it needs to do is reset the state of each tile to cleared, rather than writing out the clear

© 2007, Advanced Micro Devices, Inc., AMD, and the AMD Arrow logo, and combinations thereof, ATI, the ATI logo, CrossFire, the CrossFire logo, Radeon and combinations thereof, and The Ultimate Visual Experience are trademarks of Advanced Micro Devices, Inc.

value to the entire depth buffer. So old-school tricks to skip depth clears no longer apply and are in fact highly counterproductive. This is because tiles can end up in an uncompressed state thus consuming the full bandwidth of the individual samples they need. If the depth buffer is not cleared, more and more tiles will end up in an uncompressed state after each frame until essentially the entire depth buffer is uncompressed. At that point performance will be as if there was no Z compression hardware at all. While the X1x00 series hardware is less prone to tile decompression the recommendation to clear the depth buffer remains the same.

All ATI hardware stores depth and stencil information in the same buffer so for best performance it is essential that both depth and stencil be cleared together. Otherwise a slower read/modify/write operation will occur, like for instance only clearing the depth part of the buffer and leaving the stencil buffer untouched.

Carefully plan your stencil usage

The stencil operation is important to HiZ. The stencil pass operation doesn't matter, but if either the stencil fail or zfail operation is not KEEP HiZ gets disabled. Thus it's preferable to rely on stencil pass for stencil update over the other two. A typical example where this applies is stencil shadows. A commonly used technique is the so called Z-fail method, which unlike conventional stencil shadows (also called the Z-pass method) is much more robust and allows the viewer to be located within the shadow volume. The problem with the Z-fail method is that it – as implied by the name – updates stencil on z-fail, whereas the conventional method updates stencil on stencil pass. For this reason the performance will be worse than that of the conventional method. Unfortunately the conventional method can get artifacts if the shadow volume intersects the near clipping plane (if front plane capping is not used, that is), which has made the Z-fail method the algorithm of choice for most developers. In order to get the best of both worlds one could use Z-fail when necessary and Z-pass otherwise. There are also a number of methods to make the Z-pass method robust.

In many algorithms using stencil, the stencil buffer needs to be cleared multiple times during a frame, often separately from the depth buffer. While the stencil buffer should be cleared on a regular basis, typically once per frame at the same time as the depth buffer is cleared, it's worth noting that there's nothing wrong with doing a partial clear of the stencil buffer by using a stencil pass operation of ZERO. For instance in a multi-pass algorithm, the stencil buffer may initially be cleared to zero, and then the first pass may have laid out a mask in stencil, and the second pass renders to the pixels where stencil is 1. Using a stencil pass operation ZERO in the second pass to clear the stencil buffer at the same time can eliminate the need for a stencil clear and will work fine with HiZ. For this to work it's important that the first pass tags pixels to render, rather than pixels not to render, so that zeroing it can be done when the stencil test passes in the second pass.

© 2007, Advanced Micro Devices, Inc., AMD, and the AMD Arrow logo, and combinations thereof, ATI, the ATI logo, CrossFire, the CrossFire logo, Radeon and combinations thereof, and The Ultimate Visual Experience are trademarks of Advanced Micro Devices, Inc.

Draw the skybox last and the gun first

A common abuse of the depth buffer is using a skybox to clear the color and depth buffer. While this method looks intuitive and used to improve performance a few graphic hardware generations ago this is no longer the case. A less severe misuse is to clear the depth buffer appropriately but clear the color buffer by rendering the skybox with depth test off. This approach, which lets HyperZ operate properly while rendering the rest of the scene, unfortunately misses the opportunity to let HyperZ cull pixels from the skybox itself. Since the skybox is always behind the scene geometry it should *always* be drawn last.

The same principle applies to objects that are known to always be on top of everything else, including opaque GUI components or the gun in FPS style games: these objects should be drawn first.

One of the issues in changing existing code rendering the skybox first to rendering it last is that the box will usually get clipped in the corners by the far clipping plane and/or by previously-rendered geometry. This may happen because the skybox is modeled by an artist and has a fixed size, but even a programmer generated skybox may see issues like this. Instead of pushing the far clipping plane back a solution to these issues is to leave the far clipping plane untouched and instead simply peg the depth of the skybox to the far clipping plane. There are different methods to achieve this. The perspective divide produces a depth value of z / w, and the far clipping plane is always at depth 1.0, so one method to peg an object to the far plane would be to arrange for z to be equal to w. This can be done in a vertex shader by simply assigning w to z after transformation as the following code fragment shows:

```
//    Out.position = mul(mvp, vertex);
      Out.position = mul(mvp, vertex).xyww;
```

The only change necessary is to add the .xyww swizzle in the end. As a bonus, this compiles down to just three instructions instead of four. Another solution is to simply modify the transformation matrix by replicating the fourth row (which produces w) into the third so that the result in z and w becomes the same. This method also works with the fixed function pipeline. Another technique involves changing the depth range used in the graphics API (D3DVIEWPORT9 structure in DirectX9, D3D10_VIEWPORT structure in DirectX 10 and glDepthRange() in OpenGL) to [1, 1] when rendering the skybox. This will force all skybox pixels to occupy the maximum depth. Note that this technique works with the LESSEQUAL compare mode; a compare mode of e.g. LESS may require further depth partitioning on previously-rendered object to avoid clashes at the maximum Z value.

To peg Z to the near clipping plane for rendering the GUI components first the idea is equally trivial. The near clipping plane is at depth 0, so all we need to do is assign zero to z after transformation. This can be done directly in a vertex shader or by zeroing the third row in the transformation matrix.

© 2007, Advanced Micro Devices, Inc., AMD, and the AMD Arrow logo, and combinations thereof, ATI, the ATI logo, CrossFire, the CrossFire logo, Radeon and combinations thereof, and The Ultimate Visual Experience are trademarks of Advanced Micro Devices, Inc.

Rendering more complex 3D components first (e.g. a gun in a FPS-style game) may require more complex tricks to avoid interactions with geometry subsequently rendered, e.g. proper depth partitioning.

Optimize models for HyperZ

Sorting objects in front-to-back order is often relatively simple; however, sorting triangles within a model is usually not feasible in real-time. Most of the time models are optimized only for the vertex cache, for instance using `D3DXOptimizeFaces`() or other similar tools. Models aren't typically optimized for best HyperZ performance partly because there is no static triangle order that would result in front-to-back rendering from all possible view directions. However, it is true that some parts of a mesh are more likely to be occluders than others. In order to optimize for HyperZ without noticeably hurting vertex cache performance ATI has released a tool called Tootle for this task. This tool divides the model into a number of clusters and then determines what clusters are more likely to occlude other clusters when viewed from a number of directions around the sphere. Each cluster is then optimized for vertex cache locality. This results in meshes that are optimized for overdraw with only a negligible reduction in vertex cache effectiveness. The number of clusters can be adjusted to balance between optimizing for overdraw and vertex cache. The triangle reordering is static, thus no changes to any runtime components are necessary. The model just needs to be preprocessed once and this can be done offline. It's worth noting that even though Tootle assumes static models it's still useful to use it on animated or skinned objects. While the results will not be optimal for all cases the optimization will typically still be grossly valid.

Disable depth writes when possible

It easily happens that depth writes are left on for some rendering where this is not required, for instance in multi-pass rendering or after a Pre-Z pass. In many cases this has little impact on performance since the hardware optimizes away redundant depth writes anyway, but there are situations where this could still hurt performance, for instance updating the depth buffer when the resulting depths are never going to be read, like when rendering a transparent HUD at the end of the frame.

Tightly pack near and far clipping planes

HiZ stores only a low resolution Z value per tile. This means that some tiles that could theoretically be culled aren't culled in practice since the low precision meant it couldn't tell if it was safe. The lower the far / near clipping plane ratio is, the more likely is HiZ to identify that tiles are safe to cull when the depth difference is small. Keeping this ratio down is generally recommended anyway since depth

© 2007, Advanced Micro Devices, Inc., AMD, and the AMD Arrow logo, and combinations thereof, ATI, the ATI logo, CrossFire, the CrossFire logo, Radeon and combinations thereof, and The Ultimate Visual Experience are trademarks of Advanced Micro Devices, Inc.

precision in the depth buffer is also dependent on it, thus the lower the ratio the less likely you are to see z-fighting problems. As a general rule, always push the front clip plane to as high a value as possible.

Use as few depth buffers are possible

Something to keep in mind, especially on pre-HD 2000 series hardware, is that HiZ utilizes an on-chip buffer to store some of its info. This buffer is limited, so if you create a lot of depth buffers, some may not be able to use HiZ. It's recommended that you create your most important depth buffers first, and keep the number of buffers to a minimum. Reuse as much as you can. If you're doing most of your drawing to render targets (rather than the backbuffer) it may be better not to create the backbuffer depth buffer with the device. If you still need a backbuffer depth buffer, create it later to avoid having it take HiZ space from other more important depth buffers. With the Radeon HD 2000 series, HiZ can use off-chip memory, so more depth buffers can be used than before.

© 2007, Advanced Micro Devices, Inc., AMD, and the AMD Arrow logo, and combinations thereof, ATI, the ATI logo, CrossFire, the CrossFire logo, Radeon and combinations thereof, and The Ultimate Visual Experience are trademarks of Advanced Micro Devices, Inc.

Reference table

Below is a table for your reference to quickly check whether a particular operation has any impact on Early Z or Hierarchical Z. Note that in this table “disabled” means that it will be disabled for sure, whereas “enabled” means that this particular item in the list does not disable it, but it’s still possible that something else will.

	Early Z	Hierarchical Z
Shader depth output	Disabled	Disabled
Alpha test, alpha to coverage or texkill with depth or stencil writes on	Disabled	Enabled
Alpha test, alpha to coverage or texkill with depth and stencil writes off	Enabled	Enabled
Stencil op fail or zfail is not KEEP	Enabled	Disabled
Stencil op fail and zfail is KEEP, pass is any op	Enabled	Enabled
Depth test comparison is EQUAL	Enabled	X8xx: Disabled X1xxx: Enabled
Depth test comparison is NOTEQUAL	Enabled	Disabled
Reversed depth comparison direction	Enabled	Disabled
Reversed stencil comparison direction	Enabled	Enabled

前提是HZB自己不会更新HZB内的值，仅仅由后面的Depth Test来完成更新

硬件精度引起的问题

由非闭区间的Z-Range引起的问题

不相等测试对于HZB没有意义，因为就算HZB相等，也有可能在Fragment上不相等。所以为了节约HZB的bandwidth，尤其ATI 500这种off-chip HZB，就取消这种情况下的HZB测试

Conclusion

HyperZ is a powerful tool when used correctly. In this paper we have discussed this technology and shown how to utilize its power optimally and avoid common pitfalls. Programming with HyperZ in mind could enhance application performance significantly.

References

Nehab, D. Barczak, J. Sander, P.V. 2006. Triangle Order Optimization for graphics hardware computation culling. In Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games, pages 207-211.

Tootle download site:
http://www.ati.com/developer/tootle.html

© 2007, Advanced Micro Devices, Inc., AMD, and the AMD Arrow logo, and combinations thereof, ATI, the ATI logo, CrossFire, the CrossFire logo, Radeon and combinations thereof, and The Ultimate Visual Experience are trademarks of Advanced Micro Devices, Inc.