HDAO10.1 Sample

Collapse All

Path

Source	SDK root\Samples\C++\Direct3D10\HDAO10.1
Executable	SDK root\Samples\C++\Direct3D10\Bin\x86 or x64\HDAO10.1.exe

Sample Overview

This sample, contributed by AMD, presents an innovative technique for achieving high-definition ambient occlusion (HDAO). It utilizes Direct3D 10.1 APIs and hardware, making use of the new shader model 4.1 gather4 instruction, to greatly accelerate the performance of this technique.

Valley Detection

How does HDAO actually work? Compared to most SSAO algorithms, HDAO is remarkably simple. At its core, it detects valleys in depth:

Essentially, HDAO performs a single full-screen pass that requires only a single sample depth buffer as input. The pixel shader samples many depth texels in a solid pattern around the pixel of interest. It actually collects twin pairs of depth texels that have a special relationship. These texels are then mirrored through the pixel of interest.

For each twin pair of depth samples, the shader calculates their camera space Z values. If both Z values are closer to the camera than the pixel of interest, then the shader has detected a valley in Z. The greater the number of valleys detected, the greater the occlusion factor is for a given pixel.

Increasing the number of depth samples leads to a higher quality result, although it should be said that the technique becomes very much a high end effect. HDAO differs from other SSAO techniques in that it captures high frequency detail. This has proven to make the technique popular with a growing number of developers.

In addition, the sample demonstrates that if camera space normals are available in the application, they can be used to generate ambient occlusion on surfaces that have rich normal maps. This is an optional extra, and for many deferred rendering engines is ideal.

Dealing with an MSAA Depth Buffer

As already mentioned, this technique requires only a single sample depth buffer. So how should you correctly handle the common case where you have an MSAA depth buffer?

Most game engines use single sample depth for post-processing effects, and they usually employ one or more of the following three methods for generating a single sample depth buffer:

1. Perform a depth-only pass, whereby single sample depth is written to an R32 color buffer. Naturally, this can be costly, as it adds a whole extra pass to the engine.
2. Write out depth to a secondary render target, as part of the main render pass. A multiple render target setup will always add a performance cost.
3. Perform a custom shader resolve of the MSAA depth buffer. This is possible only in Direct3D 10.1. It's possible to avoid methods 1 and 2 in this way, which could lead to a significant performance boost.

Shader Constants

The GUI exposes the various shader constants used to fine-tune the effect to a given style:

- **Reject Radius**: If either of a twin pair of sampled camera Z values is further away from the central pixel's camera Z value than **Reject Radius**, then the twin pair of samples will not contribute to the occlusion factor. This alleviates a common problem with SSAO, where distant objects can cause a halo of occlusion on the silhouette edges of nearby objects.
- **Accept Radius**: This works in a similar fashion to **Reject Radius**. Again, if either of a twin pair is closer to the central sample than **Accept Radius**, then the twin pair of samples will not contribute to the occlusion factor. This can be particularly useful for avoiding unwanted occlusion on low-density meshes.
- **Intensity**: This value simply scales the final occlusion factor.

Low-Density Meshes

HDAO can yield unwanted occlusion on some low-density meshes, this happens because there are no settings for the **Accept Radius** and **Reject Radius** parameters that yield the desired result. In the sample, the Desert Tank scene highlights this problem with the low-density terrain mesh. For this reason, it could be necessary to mask out HDAO for some meshes.

This sample implements a method for doing this using the stencil buffer. However, it could be more efficient to use the alpha channel of another full-screen surface, if available.

Performance

HDAO is an expensive effect, and in particular the texturing burden is very high. To alleviate this cost, the Direct3D 10.1 gather4 instruction has been used to accelerate the sampling. Instead of reading a single sample at a time, we collect four. This greatly reduces the texturing overhead of the technique. Another benefit to working on four samples at a time is that the calculations can also be vectorized in a very natural way.