DepthBoundsTest11

AMD Developer Relations

Overview

This sample shows how to use the Depth Bounds Test with AMD graphics hardware. You can also use this sample as an example of how to use other driver extensions, since the Depth Bounds Test uses the AMD driver extension interface.

Depth Bounds Test is a hardware feature that causes the GPU to cull pixels if the depth buffer value is outside the specified range. This is different than the way we normally think of the depth buffer working, where the depth value of the pixel is compared to the corresponding value in the depth buffer. The sample uses this behavior to optimize deferred shading. By setting the depth range to the depth range of the light, the sample is able to cull pixels that are either too near or too far to be affected by the light.

Implementation

The deferred rendering in this sample has 3 main passes:

1. Render the scene into the G-Buffer
2. Shade in screen space with the list of lights
3. Post-processing in screen space

The g-buffer pass and the post process step are fairly standard so there's no need to go into much detail. Post-processing is where screen facing quads are rendered to represent the glowing lights. The shading pass is where the depth bounds test is used.

Shading Pass

The shading pass is used for rendering the lighting and shading in the scene. With deferred rendering, the shading happens in screen space and it uses the g-buffer render targets as textures. The position of the pixel in world space is known from its (x,y) screen coordinate and the depth value. The light is rendered as a quad placed behind the light's radius, with width and height equal to the diameter of the light range, and with the depth test set to "greater than". By using a quad that has the same size as the light range, you avoid shading pixels that are not affected by the light. By using the "greater than" depth test, you only render pixels that are in front of the quad. Therefore everything outside of the light's range behind the light will not be drawn. However, pixels in front of the quad but outside the range of the light will not be culled. Fortunately, this can be done with the hardware Depth Bounds Test.

Depth Bounds Test

The depth bounds test is a hardware feature that limits rendering to a specified depth range. In figure 1, the depth range is shown in relation to the viewing frustum from a top down point of view. It's important to understand the distinction between "pixel depth" and "depth buffer value". Pixel depth is the z value of the pixel being rendered in a pixel shader. Depth buffer value is the value already in the depth buffer corresponding to the location where the pixel is being rendered. It was set by earlier rendering, not by the pixel currently being rendered. The Depth Bounds Test *only uses the depth buffer value* to determine if the current pixel should be rendered. In Figure 1, the green object is drawn because the depth buffer contains only the depths of the blue objects, and only one of the blue objects is within the depth bounds range. The screen space location of that blue object overlaps only the green object's screen space location. The red objects will not be drawn. The locations in the depth buffer corresponding to the red objects are not within the depth bounds range, even though the red objects themselves are within the depth bounds range. One way to think of the Depth Bounds Test is that it's like scissoring, only with depth.

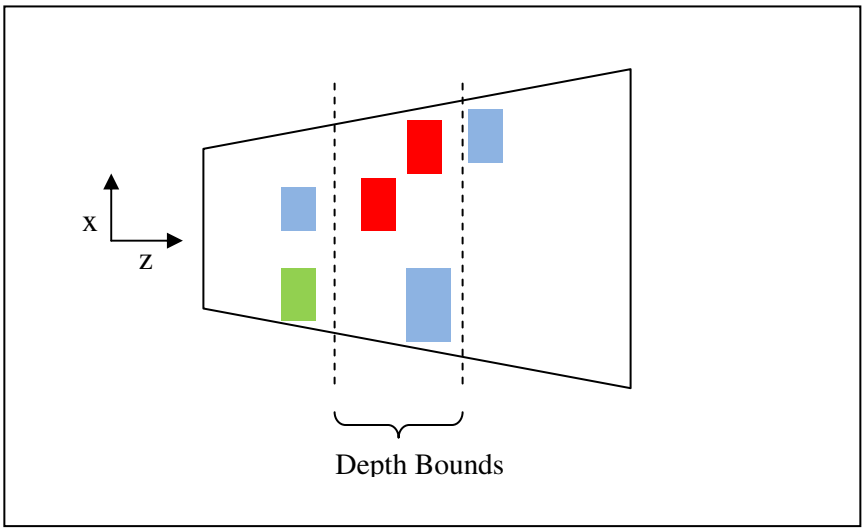

Figure 1: Depth Bounds Test in relation to the viewing frustum. The blue objects were drawn prior to the test and have set the values in the depth buffer. The red and green objects are being drawn with the depth bounds test enabled. Only the green object will be visible, because the corresponding pixels in the depth buffer (from one of the blue objects) pass the depth bounds test.

On AMD Radeon HD7000 series graphics cards and later, the depth bounds test is accessed through the AMD driver extension API. To use the driver extensions the application needs to include the AMD extension header files. There are 3 header files required for all of the extensions:

`AMDDxExt.h`, `AMDDxExtApi.h`, and `AMDDxExtIface.h`

In addition to these headers each extension has its own set of include files. For the depth bounds extension, the files are:

`AMDDxExtDepthBounds.h` and `AMDDXExtDepthBoundsApi.h`

This sample has a function called `OpenAMDExtensionInterface()` which is used for initializing the AMD extension and getting a pointer to the depth bounds extension interface.
Setting the depth bounds is like setting a state, in that once you set the depth bounds test, it stays in effect until you change it or disable it. Because of this, you can't change the depth bounds inside of a draw call. This means that each light has to be rendered with a separate draw call instead of batching the lights into one draw call. However, culling unnecessary pixels with depth bounds test can be worth the extra draw call overhead, as demonstrated in this sample. Alternatively, it may be possible to group the lights by depth range, and set the depth bounds test for the range of multiple lights. That way you could render more than one light in a draw call, and reduce the overhead. In the sample, the function `SetDepthBoundsFromLightRadius()` is called right before the draw call for the light. A view vector is created from the look-at point (set to the depth of the light) to the camera. Note that a vector from the camera to the light center would cause the depth range to be too small. The function scales the view vector by the light radius, and then adds that vector to the center of the light to get the near bound of the light in 3D world space. It then uses the view-projection matrix to transform that coordinate into screen space so it can use the z value (divided by the homogenous coordinate) as the near depth value to pass to the depth bounds test. It then repeats this, except subtracting the view vector from the center of the light to get the far depth bound. Here is the call in SetDepthBoundsFromLightRadius() which sets the hardware depth bounds:

```
g_pDepthBoundsTest->SetDepthBounds(true, nearBound, farBound);
```

where `g_pDepthBoundsTest` is a pointer to the depth bounds extension. The first parameter which is set to `true`, enables the depth bounds test. When all of the light rendering is finished, the sample calls:

```
g_pDepthBoundsTest->SetDepthBounds(false, 0.0f, 1.0f);
```

to disable the depth bounds test.

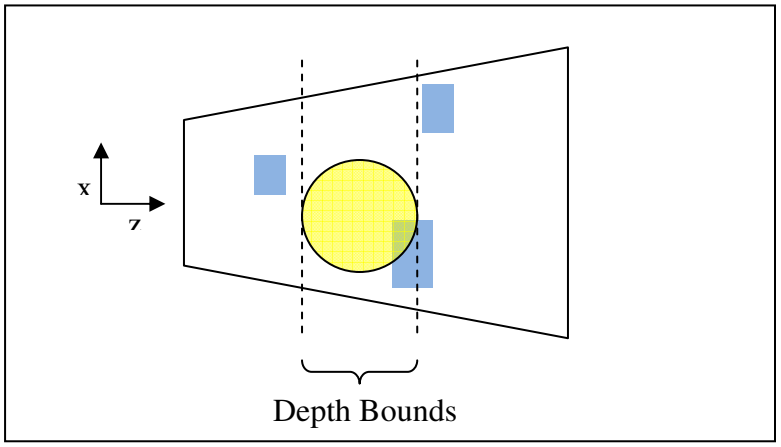

Figure 2: Depth Bounds Test used for lighting. When rendering the light, the depth bounds range is set to the range of the light (yellow circle). Only the pixels where the light overlaps the blue rectangle will be shaded since other pixels will be culled because the depth values will not pass the depth bounds test.

Performance

The performance gains from using Depth Bounds Test with deferred shading comes from not rendering pixels that would normally need be rendered with deferred shading that doesn't use depth bounds test. More precisely, the performance gains come from not having to execute the lighting pixel shader as many times. The reason for this distinction is that deferred lighting pixel shaders can use the discard instruction to not render pixels that are outside of the light's range. However, some portion of the pixel shader still needs to get executed even if the pixel gets discarded. The sample uses discard in the lighting pixel shader to improve overall performance, but even with this optimization, depth bounds test is still able to further improve performance. The following table shows performance differences when depth bounds test is enabled and when it's not enabled for various number of point lights. These tests were done on a Radeon HD 7970 at 2560x1600 resolution with a 3.21 GHz Phenom II X6 CPU.

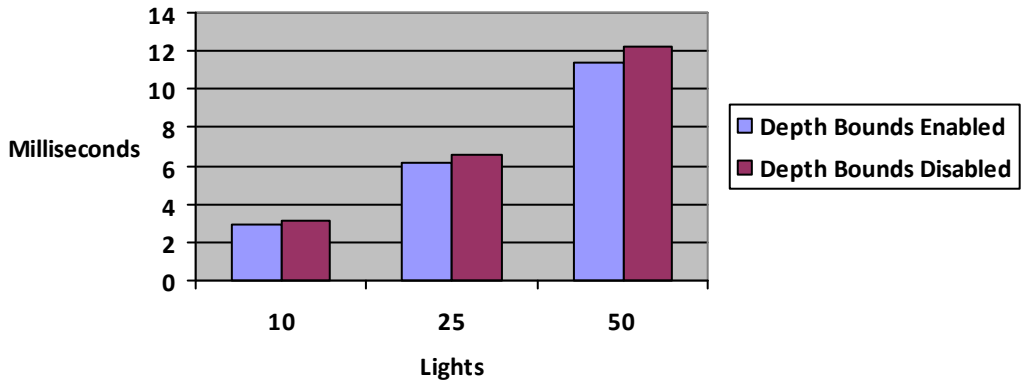

Table 1: Overall frame time comparison between using Depth Bounds Test and not using Depth Bounds Test with deferred lighting.

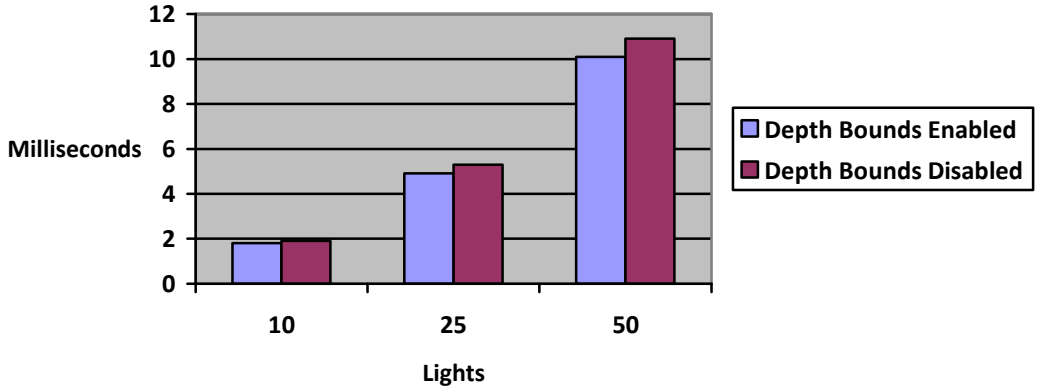

Table 2: Timing values for just the shading pass.

To visualize the difference between using Depth Bounds Test and not using it, the sample has a checkbox to show the discarded pixels. Figure 3 shows the difference in rendering with and without Depth Bounds Test. Although discarding pixels is an optimization, this visualization shows that the Depth Bounds Tests culls pixels before the discard needs to be used in the pixel shader, which is why it improves performance.

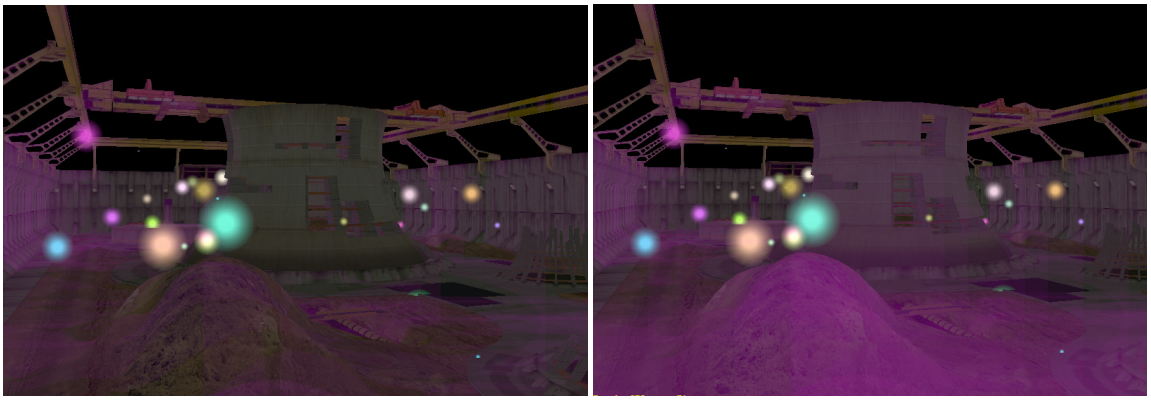

Figure 3: Screen shots from sample showing discarded pixels. The image on the left is with Depth Bounds Test enabled, and the image on the right is with it disabled. The more times a pixel is discarded for a given location, the higher the intensity of the magenta color. Notice that with Depth Bounds Test enabled (left) there are fewer pixels discarded. This is because the unnecessary pixels are culled by the Depth Bounds Test before they can be discarded in the pixel shader. Since the shader never gets executed for the culled pixels, performance is faster with Depth Bounds Test enabled.

Requirements

The Depth Bounds Test extension is only supported on Radeon HD 7000 series graphics cards and later, with Catalyst 13.1 or newer drivers.

AMD Smarter Choice

Advanced Micro Devices
One AMD Place
P.O. Box 3453
Sunnyvale, CA 94088-3453

www.amd.com
http://ati.amd.com/developer