SDK White Paper

Soft Shadows

TB-01409-001_v01
July 2004

nVSDK

Abstract

This paper describes how Shader Model 3.0's conditional branching accelerates the computation of soft shadows. Note that you are not generating true soft-shadows, but rather approximating soft shadows by softening the boundaries of traditional hard shadows. The technique described in this paper performs a small amount of computation to estimate if the current pixel is in the approximated soft-shadow region. If the pixel is completely in shadow, or completely out of shadow, the shader exits early. Otherwise, additional work is performed to produce a more accurate estimate of the shadow's intensity at the current pixel. Using conditional branching to perform this additional work only when needed gains considerable performance over techniques that don't use branching.

This sample is written for DirectX 9.0c and requires PS/VS3.0 support.

Yury Uralsky, Anis Ahmad
sdkfeedback@nvidia.com
NVIDIA Corporation
2701 San Tomas Expressway
Santa Clara, CA 95050
July 15, 2004

Introduction to Shadows

Shadows are an important visual cue. Unfortunately, most real-time techniques for computing shadows generate hard and, in the case of shadow-maps, aliased shadows. We can soften the edges of the shadows by employing percentage-closer filtering. In fact, a sufficiently large percentage-closer filter approximates soft shadow boundaries well. High quality soft shadows, however, require a large number of samples.

In this paper we explain how to improve both the performance and the quality of shadow computations that use percentage-closer filtering. The technique we describe achieves higher quality by employing pseudo-random, jittered sampling over a large filter kernel, and it achieves higher performance by only multi-sampling pixels near the border of the original hard shadow. Figure 1 shows images generated by the technique. The provided code lets you experiment with these techniques to see how they enhance the appearance of shadows.

Figure 1. Real-time Softened Shadows

Percentage-Closer Filtering

Penumbras are formed when part of the light source illuminating a surface is visible, and part of is obscured. With shadow maps, because you are dealing with point light sources, this definition is meaningless. However, we can approximate soft shadows using a technique called *percentage-closer filtering* [1].

Researchers at Pixar developed percentage-closer filtering to help anti-alias shadow map boundaries. Beginning with the GeForce3, NVIDIA hardware performs percentage-closer filtering automatically using hardware accelerated shadow maps. Percentage-closer filtering works by comparing the current pixel's depth against its corresponding position in the shadow map, and against its immediate neighbors in the shadow map. The resulting shadow intensity is the percentage of shadow map tests determined to be in shadow.

While percentage-closer filtering does help eliminate shadow aliasing, it can not produce shadows with sufficiently large penumbras. Soft shadows can be produced by making a simple variation to this technique. Rather than testing only the immediate neighbors of the pixel, test an arbitrarily large set of samples near the pixel in question, as illustrated in Figure 2.

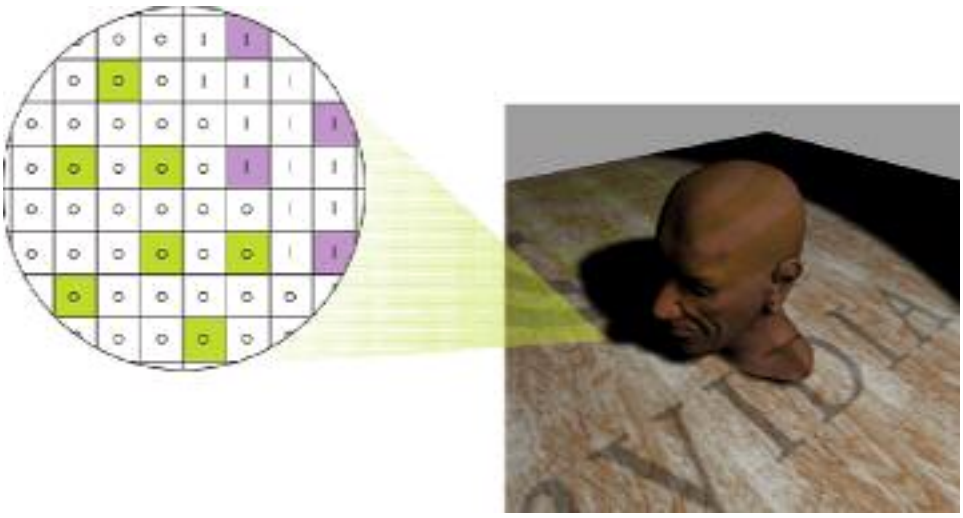


Figure 2. Sampling Around a Local Region of the Shadow Map Produces Approximated Soft Shadows

Blurring Hard Shadows

To make shadows appear *soft*, you need to blur hard shadows. Though it is not correct from a physics standpoint, results are acceptable. In signal processing terms, that means convolving shadow function with a box filter kernel. In other words, calculating the following integral over some local neighborhood of current shadow map sampling point.

$$\text{SoftShadow}(s) = \int_S \text{shadow}(x)\, ds$$

The bigger this region is, the softer the shadow. The size of this footprint is determined both by a user-provided value, which controls overall softness, and by the distance between the point in question and the light source, which in turn is determined from the w-component of the projected texture coordinate.

The discrete world deals with sampled representations of continuous functions, such as shadow maps. Therefore, the integral has to be approximated with a finite sum having sufficiently large number of terms. Getting higher quality approximation with regularly spaced sample locations is challenging because number of samples required can get really high and thus computationally expensive. Even with high number of samples, aliasing artifacts in the form of banding will still be visible.

To approximate this integral efficiently, we employ a well-known technique called Monte-Carlo sampling. According to this method, we sample shadow map at a number of randomly chosen locations within our sampling region, and average them. Essentially, the Monte-Carlo method hides the approximation error by replacing it with high-frequency noise. The human visual system is well-trained to deal with noise, so the error becomes much less objectionable than banding.

Jittered Sampling

Unfortunately, the naïve approach of averaging random set of shadow map samples at each screen pixel results in too much noise—which is still distracting to the eyes. If we look at the sample patterns generated from random numbers with uniform distribution, we see that sampling density is highly non-uniform across our region (some areas have too much sample locations concentrated; some other areas are under sampled). This increases noise variance and makes it much more visible than it should be.

Jittered sampling, shown in Figure 3, is an excellent way to reduce noise variance. Computing a jittered sample simply involves taking a sample point from a uniform grid and randomly offsetting it within its grid cell. This random offset is generally some fraction of the distance between the uniform grid points. Since we are generating samples for a disc-shaped filter, we take jittered samples from a [0,1]x[0,1] grid and map them onto a disc using polar coordinates, taking care to keep sampling density uniform across entire region.

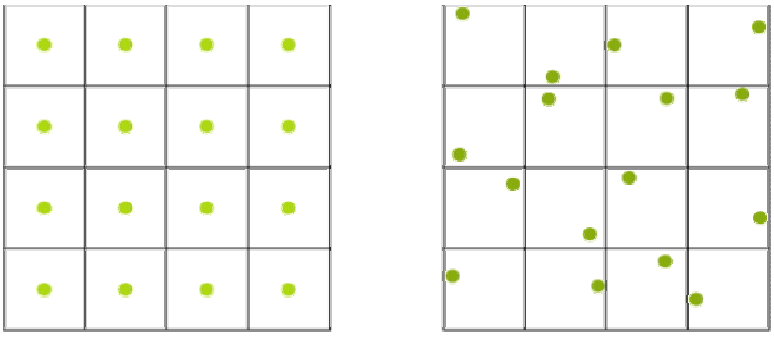


Figure 3. Comparison of Uniform and Jittered Samples

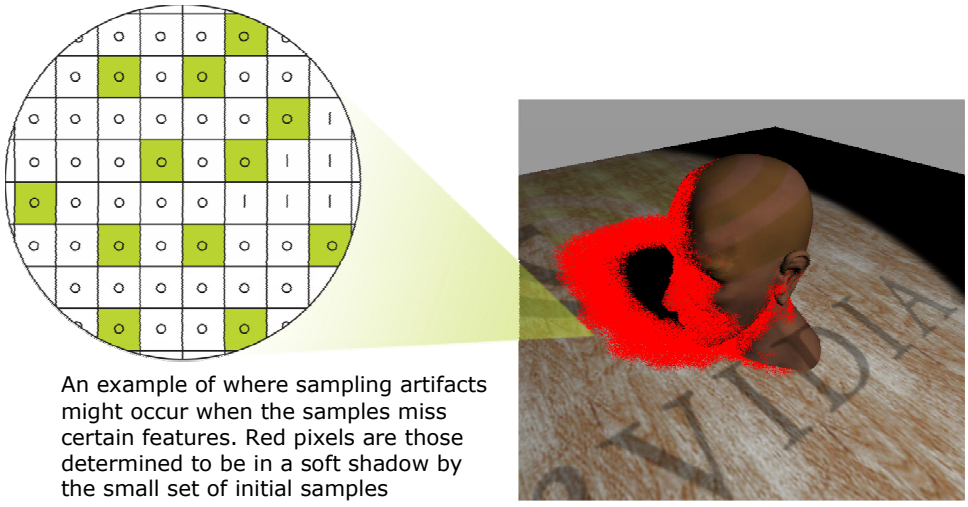
It is computationally expensive to compute a new set of randomly jittered samples at each pixel. Therefore, pre-compute a set of 64 jittered samples per pixel, for a small MxM region of screen pixels. The pre-computed samples are stored as 2D offsets in an MxMx32 3D texture, where the red/green channels store one offset and the blue/alpha channels store another offset.

Use each pixel's screen space coordinate, available through the VPOS register in PS3.0 (or fragment.position in OpenGL), to fetch this pixel's set of 64 jittered samples. By wrapping texture coordinates in x- and y- directions for this 3D texture, you effectively tile the precomputed samples over the entire screen. As a result, you get locally unique jittered samples that cost only a single 3D texture lookup to compute.

Adaptive Sampling

It is wasteful to compute many shadow-map tests for pixels that are either completely in shadow or completely visible to the light source. That wasteful computation is the motivation for this technique. We test a handful of samples first and determine whether or not these samples are completely in shadow or completely unobscured. If the small set of samples does not fall into either category, we can assume the pixel lies in the approximated soft shadow region and thus test additional samples to compute an estimate of the shadow's intensity.

In our implementation, the set of samples used to test whether or not the pixel is in the soft shadow region are distributed along the other rim of our filter. Although jittered, the positions of these samples are still fixed to small regions with relatively large gaps separating them. For large filters, these gaps could lead to sampling artifacts, as in Figure 4. These artifacts are particularly noticeable when the shadow caster has small features. One way to deal with these artifacts is to randomly rotate each set of jittered offsets. These rotations minimize the sampling artifacts by adjusting the location of the gaps between samples, from pixel to pixel.



An example of where sampling artifacts might occur when the samples miss certain features. Red pixels are those determined to be in a soft shadow by the small set of initial samples

Figure 4. Sampling Artifacts

Performance notes

Due to finite shadow map precision and disparity in sampling patterns in screen space versus shadow map space, shadows can exhibit aliasing in the form of *shadow acne* across object surfaces. This fragility of the shadow map test is a well-known problem common to all shadow map-based algorithms. Typically, this issue is hidden by rendering second-depth surfaces to shadow map, thus effectively moving the problem to the shadowed side of the object where it is not visible.

Note: The aliasing problem has certain performance implications with the technique described in this paper. Though not visible when second depth shadow mapping is used, this issue potentially makes more pixels to be estimated as being *penumbra* pixels, which wastes computation on those.

To improve performance of adaptive sampling techniques, take into consideration whether the current pixel belongs to the back side of the object with respect to the light source and never over sample those pixels, even if they are [incorrectly] estimated as being penumbra pixels. This is accomplished by comparing the scalar product of N and L to zero and taking the result into account when deciding whether extra shadow samples are needed (see Figure 5).

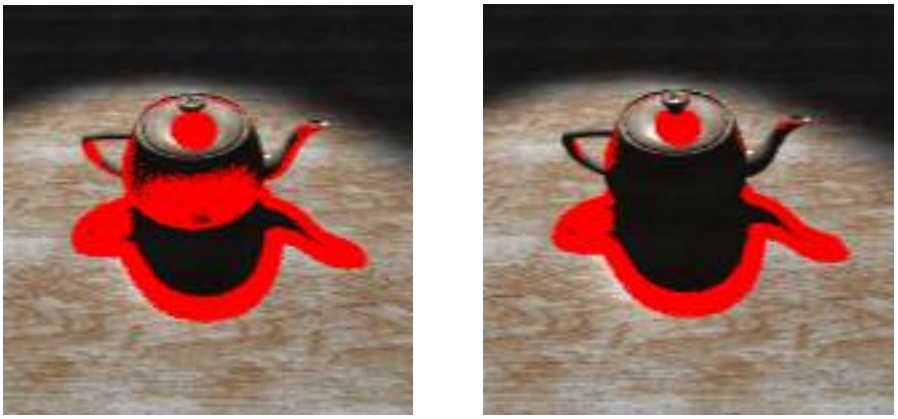


Figure 5. Penumbra Pixels With and Without Optimization

Another possible improvement is using mid-point shadow maps in concert with this technique. Mid-point shadow maps can improve the quality of shadows generated by the described method and eliminate light leakage artifacts visible in some places.

Conclusion

With the techniques described in this paper, we have shown how to compute approximated soft shadows efficiently using hardware conditional branching in the pixel shader. While these techniques are used to produce soft shadows, they can also be used to accelerate other algorithms where heavy sampling is required, such as motion blur or depth-of-field effects.

Explore ways to create or accelerate new effects with the techniques described in this paper.

Bibliography

1. Reevers, W. T., D. H. Salesin, and R. L. Cook. 1987. *Rendering Antialiased Shadows with Depth Maps*. Computer Graphics 21(4) (Proceedings of SIGGRAPH 87).

2. Robert L. Cook. *Stochastic sampling in computer graphics*, ACM Transactions on Graphics (TOG), v.5 n.1, p.51-72, Jan. 1986.

Notice

Trademarks

Copyright

NVIDIA Corporation
2701 San Tomas Expressway
Santa Clara, CA 95050
www.nvidia.com