Variance Shadow Maps

William Donnelly* Andrew Lauritzen†

Computer Graphics Lab, School of Computer Science, University of Waterloo

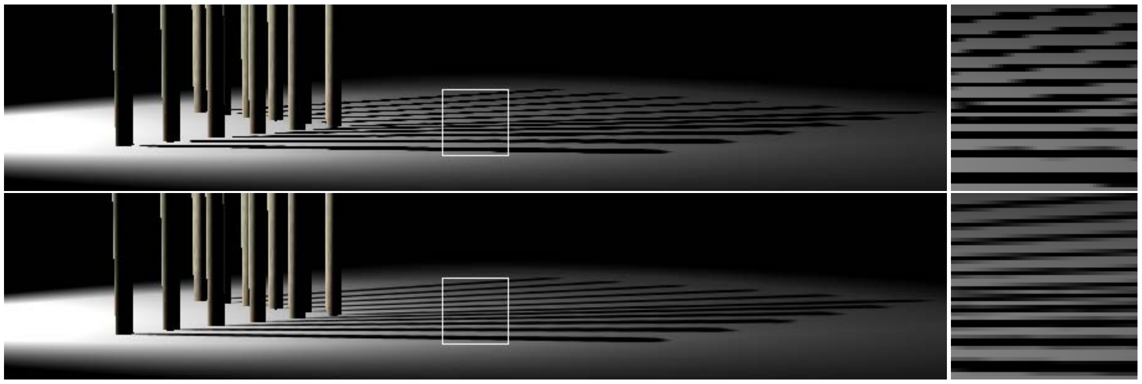


Figure 1: Comparison of anisotropic filtering vs. no anisotropic filtering. Top: Regular shadow map with bilinear percentage closer filtering. Bottom: Variance shadow map with mipmapping and 16x anisotropic filtering.

Abstract

Shadow maps are a widely used shadowing technique in real time graphics. One major drawback of their use is that they cannot be filtered in the same way as color textures, typically leading to severe aliasing. This paper introduces *variance shadow maps*, a new real time shadowing algorithm. Instead of storing a single depth value, we store the mean and mean squared of a distribution of depths, from which we can efficiently compute the variance over any filter region. Using the variance, we derive an upper bound on the fraction of a shaded fragment that is occluded. We show that this bound often provides a good approximation to the true occlusion, and can be used as an approximate value for rendering. Our algorithm is simple to implement on current graphics processors and solves the problem of shadow map aliasing with minimal additional storage and computation.

CR Categories: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Color, shading, shadowing, and texture

Keywords: real-time rendering, shadow maps, shader programming, graphics hardware

*e-mail: wdonnelly@math.uwaterloo.ca

†e-mail: atlaurit@cgl.uwaterloo.ca

1 Introduction

Shadow maps [Williams 1978] and shadow volumes [Crow 1977] are the two most common shadowing algorithms used in real time applications. Shadow maps have many advantages compared to shadow volumes; for example they are easy to implement, their cost is less sensitive to geometric complexity and they can be queried at arbitrary locations.

Unfortunately, like most textures, shadow maps suffer from aliasing if not filtered properly. Modern graphics hardware provides built-in methods to reduce texture aliasing on color textures: namely mipmapping and anisotropic filtering. These techniques are inapplicable to standard shadow maps, since they will simply interpolate the depths of neighboring pixels. Typically real time implementations use nearest-neighbor sampling of shadow maps, or take several samples and average the results. This method is expensive, causes aliasing and does not take full advantage of graphics hardware's aforementioned fast, built-in filtering capabilities.

To address the problem of efficiently filtering shadow maps, we note that each texel of a standard shadow map can only represent the depth of a single point. Variance shadow maps improve on this scheme by representing a *distribution* of depths at each texel. To approximate such a distribution using a small amount of data, we store the first and second moments: the mean depth and mean squared depth. One major advantage of this representation is that we can approximate the average of two distributions by averaging the moments.

When querying the variance shadow map, we use the moments to compute a bound on the fraction of the distribution that is more distant than the surface being shaded. We show that this bound provides a good approximation for the amount of light reaching any given surface, and therefore can be used for rendering correctly anti-aliased shadows.

Because the moments can be interpolated, we can make use of the wide range of filtering techniques that are available for color textures, effectively eliminating aliasing.

- Reduces aliasing on shadow maps by enabling the use of filtering techniques such as mipmapping and anisotropic filtering,
- Allows shadow maps to be pre-filtered for percentage closer filtering, and
- Can be implemented on current graphics hardware at a cost comparable to that of ordinary shadow maps.

2 Related Work

Williams introduced shadow maps [Williams 1978] as an efficient algorithm for computing shadows in general scenes. However, he points out that the usual filtering techniques for color textures cannot be applied to depth values.

Percentage closer filtering [Reeves et al. 1987] provides a solution to the problem of shadow map aliasing. The key insight is that a correct filtering algorithm needs to filter the *results* of the depth comparisons, instead of filtering the depths. This is accomplished by randomly sampling the shadow map, so many samples are required to eliminate noise.

Deep shadow maps [Lokovic and Veach 2000] store a distribution of depths instead of a single depth at each pixel. As a result, percentage closer filtering can be done as a pre-process. Subsequently, each query requires a constant amount of work independent of the filter size. However, deep shadow maps are not particularly amenable to implementation on graphics hardware for two reasons: each pixel must encode a piecewise linear function using a large amount of data, and the procedure for averaging two distributions is non-trivial.

Opacity shadow maps [Kim and Neumann 2001] encode a distribution of depths, but use a fixed amount of storage per pixel. Each pixel stores a function just as in deep shadow maps, but instead the function is constructed from its values at a fixed set of points. Thus the algorithm can take advantage of the speed of graphics hardware. The price to pay for this flexibility is extreme quantization of depths, so the technique is mostly suitable for rendering of dense volumetric objects such as hair and fur.

An alternative approach to the problem of shadow map aliasing is to alter the shadow map projection. This avenue was pursued in adaptive shadow maps [Fernando et al. 2001], perspective shadow maps [Stamminger and Drettakis 2002], light space perspective shadow maps [Wimmer et al. 2004] and trapezoidal shadow maps [Martin and Tan 2004]. Because we make no assumptions about the shadow projection, these approaches are compatible with variance shadow maps; in fact, the two approaches are complementary.

Some graphics hardware has native support for shadow maps, for example through the OpenGL "GL_ARB_shadow" extension. However, the extension does not specify behaviour with respect to interpolation or mipmapping. Some NVIDIA graphics processors support bilinear filtering [Everitt et al. 2000], but to our knowledge, no graphics hardware supports trilinear or anisotropic filtering on shadow maps.

3 Algorithm Overview

As with conventional shadow mapping, we first render the scene from the light's point of view. For shadow mapping, we would render the depth as seen from the light; for variance shadow maps we render into a two-channel buffer, rendering both the depth and the square of the depth. Although in regular shadow mapping we would not want to use any type of anti-aliasing when rendering from the light's point of view, anti-aliasing will actually be of benefit when rendering variance shadow maps.

Once we have created the shadow map, we can do preprocessing on the texture to facilitate filtering. This can include generating mipmaps [Williams 1983] or computing summed area tables [Crow 1984]. To further reduce aliasing and soften shadow edges we can also blur the variance shadow map.

Since we have rendered depth and squared depth in the texture, the result of filtering our texture will be to recover the moments M_1 and M_2 over that filter region, defined as follows:

$$M_1 = E(x) = \int_{-\infty}^{\infty} x p(x) dx \tag{1}$$

$$M_2 = E(x^2) = \int_{-\infty}^{\infty} x^2 p(x) dx \tag{2}$$

From these we compute the mean μ and variance σ^2:

$$\mu = E(x) = M_1 \tag{3}$$

$$\sigma^2 = E(x^2) - E(x)^2 = M_2 - M_1^2 \tag{4}$$

The variance can be interpreted as a quantitative measure of the width of a distribution. As a result, it should place a bound on how much of the distribution can be concentrated far away from the mean. This bound is stated precisely in Chebyshev's inequality:

Theorem 1 (Chebychev's inequality, one-tailed version) *Let x be a random variable drawn from a distribution with mean μ and variance σ^2. Then for $t > \mu$*

$$P(x \geq t) \leq p_{max}(t) \equiv \frac{\sigma^2}{\sigma^2 + (t-\mu)^2} \tag{5}$$

The quantity $P(x \geq t)$ in equation 5 is exactly the quantity we wish to compute in order to perform percentage closer filtering, since it represents the fraction of pixels over a filter region that will fail the depth comparison with a fixed depth t.

However, equation 5 is only an upper bound; a priori there is no reason to assume it will allow us to compute the true value $P(x \geq t)$. It can, however, provide a good approximation, as we show in the following example.

3.1 Planar Occluders and Receivers

Consider the case of a single planar occluder at depth d_1, casting a shadow onto a planar surface at depth d_2. Suppose we have a fixed filter, where p is the percentage of the filter that is unoccluded. Then we have:

$$\begin{aligned}
\mu = E(x) &= pd_2 + (1-p)d_1 \\
E(x^2) &= pd_2^2 + (1-p)d_1^2 \\
\sigma^2 &= pd_2^2 + (1-p)d_1^2 - (pd_2 + (1-p)d_1)^2 \\
&= (p - p^2)(d_2 - d_1)^2
\end{aligned}$$

Using these values, we can compute $p_{\max}$ according to equation 5:

$$\begin{aligned} p_{\max}(d_2) &= \frac{\sigma^2}{\sigma^2 + (\mu - d_2)^2} \\ &= \frac{(p-p^2)(d_2-d_1)^2}{(p-p^2)(d_2-d_1)^2 + (pd_2 + (1-p)d_1 - d_2)^2} \\ &= \frac{(p-p^2)(d_2-d_1)^2}{(p-p^2)(d_2-d_1)^2 + (1-p)^2(d_2-d_1)^2} \\ &= \frac{p-p^2}{1-p} \\ &= p \end{aligned}$$

Thus in this simple situation, we see that Chebyshev's inequality is an equality, and gives the *exact* result of percentage closer filtering.

Although this is a very particular situation, it actually provides a reasonable approximation to a common situation in many real scenes. In the case of a single occluder and single receiver, we can take a small neighbourhood in which the depth of the occluder and receiver are approximately constant. In this case, equation 5 will not provide an exact value, but a close approximation. Thus we use $p_{\max}$ in rendering as an approximation to the true value p.

4 Implementation

We implemented variance shadow maps on a GeForce 6800GT using Sh [McCool and Du Toit 2004] and OpenGL. Sh allows us to build the variance shadow mapping shader on top of existing light shaders, and automatically combine it with any of the surface shaders in the scene.

The GeForce 6 series supports filtering of 16-bit per component floating point textures (fp16), so we use hardware mipmapping and anisotropic filtering. Our implementation is roughly as follows:

1. Render to a four component fp16 framebuffer object from the light's point of view[1]. In the fragment program, output the depth and squared depth of the current fragment to the framebuffer. We scale the depths to be in the range [0,1] to avoid overflowing the fp16 numeric boundaries.

2. Optionally prefilter the shadow map using a two-pass separable gaussian blur.

3. Have OpenGL automatically generate mipmaps.

4. Render the scene normally from the camera's point of view. In the fragment shader, read the shadow map to get the moments M_1 and M_2. If the current fragment has depth $< \mu$, then the surface is unshadowed. Otherwise, compute the variance from the first two moments (equation 4) and scale the light intensity by $p_{\max}$ (equation 5).

In addition, we implemented a version using a 32-bit per component shadow map. Since our hardware does not support filtering at that precision, we implemented bilinear filtering in the fragment shader. In principle we could also emulate mipmapping and anisotropic filtering with shader code; in practice, this would run too slowly on current hardware.

[1]We only need two components, but current hardware does not support rendering to two component fp16 textures. Optionally one could use multiple render targets instead of multiple components.

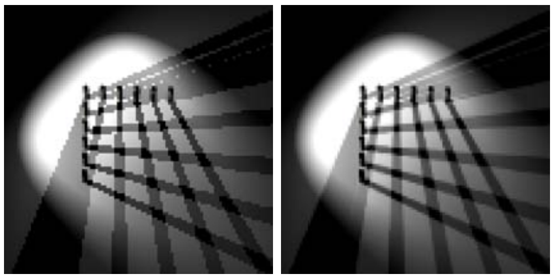


Figure 2: Comparison of variance shadow mapping without mipmapping (left) and with mipmapping (right).

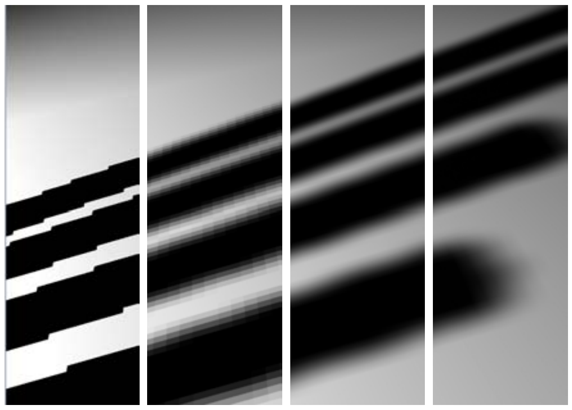


Figure 3: Left to right: 1) standard shadow mapping, 2) 5x5 percentage closer filtering, 3) 5x5 bilinear percentage closer filtering, 4) variance shadow maps with 5x5 separable gaussian blur.

We also implemented the following alternate shadowing methods for comparison. Each of them uses a single unfiltered fp16 component to store the occluder depth.

1. Simple shadow mapping using a single nearest neighbor depth comparison.

2. Bilinear filtering of the neighboring four depth comparisons.

3. Percentage closer filtering with gaussian weights, each sample being a nearest neighbor depth comparison.

4. Percentage closer filtering with gaussian weights, each sample being the bilinear filtered result of the neighboring four depth comparisons.

5 Results

Figure 2 shows the result of using mipmapping with variance shadow maps. As with color textures, mipmapping effectively reduces the aliasing of the shadow map when viewed from a distance. Anisotropic filtering can also be used with variance shadow maps, eliminating aliasing that occurs when viewing surfaces at shallow angles (see Figure 1).

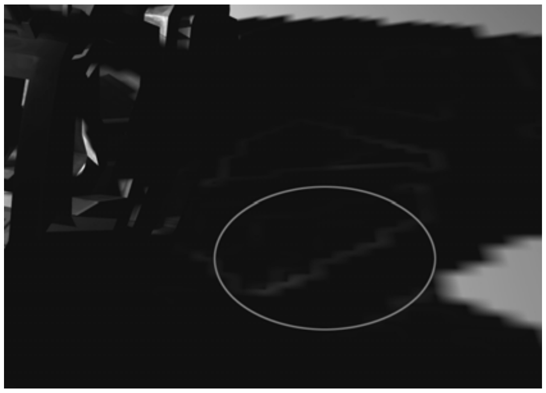


Figure 4: When the variance over a filter region is high, light bleeding artifacts can occur. The circled region of the car's shadow should be solid black. Note that the contrast has been increased here so that the artifact can be seen more easily.

Figure 3 shows a side-by-side comparison of the results of using a prefiltered variance shadow map, and an equivalent percentage closer filter. As the figure demonstrates, the output of these two methods is almost identical, and vastly superier to simple shadow mapping and standard nearest neighbor percentage closer filtering.

5.1 Light Bleeding

Our formula for $p_{\max}$ was derived as a lower bound on the brightness, and although it works well in many situations, it is not guaranteed to be an accurate approximation. We can see from equation 5 that whenever the variance σ^2 is nonzero, $p_{\max}(d) > 0$ for all d. When σ^2 is small, $p_{\max}$ goes to zero quickly, and so the effect is not very noticeable. However, when σ^2 is large the results can be noticeable as seen in figure 4.

In practice, this is a only a problem for scenes with a high depth complexity relative to the shadowed light source. For scenes with a low depth complexity (eg. many scenes lit by the sun), the artifacts are non-existent or negligible.

5.2 Performance

To compare the performance of several shadow mapping implementations we chose a very simple scene with a single spotlight, shown in Figure 1. Because all of the methods examined operate in image space, their performance is independent of geometric complexity.

The variance shadow map results used hardware mipmapping, trilinear and 16x anisotropic filtering. Hardware texture filtering is inapplicable to the other algorithms[2], and thus any required filtering was implemented in the shader.

All measurements are in frames per second, taken on a GeForce 6800GT. Rendering at a resolution of 1024x768 and varying the shadow map size, the results are as follows:

[2]NVIDIA provides an extension to perform post-depth-comparison bilinear filtering of shadow map lookups [Everitt et al. 2000]. We have not used this extension in our implementation, but it should be noted that its usage may improve performance on some platforms.

	128x128	256x256	512x512	1024x1024
Shadow Map	340	334	314	243
Bil. PCF 1x1	224	219	209	175
VSM	265	255	228	154
PCF 3x3	153	151	149	130
Bil. PCF 3x3	22	22	22	21
VSM 3x3	254	225	154	63

We now fix the shadow map size at 512x512 and vary the framebuffer resolution:

	640x480	800x600	1024x768	1280x960
Shadow Map	470	470	314	210
Bil. PCF 1x1	457	309	209	133
VSM	450	334	228	159
PCF 3x3	336	229	149	92
Bil. PCF 3x3	53	33	22	15
VSM 3x3	230	195	154	119

Notice that the PCF results do not vary greatly with shadow map size, but rather by the number of onscreen pixels. By contrast, the VSM results scale with the framebuffer resolution at the same rate as standard shadow mapping. However, when using a prefilter such as in the VSM 3x3 case, the variance shadow mapping performance scales with the shadow map size.

5.3 Numerical Stability

Equation 4 for computing the variance is known to be numerically unstable when using floating point arithmetic. This is because when the variance is small compared to the average depths we have $E(x^2) \approx E(x)^2$ and so we are subtracting two approximately equal large numbers, potentially resulting in loss of precision. In practice, we found that artifacts were sometimes visible when using a 16-bit floating point shadow map, but never with 32-bit floating point values. The fp16 artifacts can be largely eliminated by splitting each 32-bit float into two 16-bit floats for storage (since we are currently forced by the hardware to use a four component texture anyways) and recombining them after the texture lookup. Moreover, we expect future hardware to support mipmapping and anisotropic filtering of 32-bit floating point textures and thus resolve this issue.

6 Conclusions

We have introduced variance shadow maps as a simple and effective solution to the problem of aliasing in shadow maps. Our results are based on an upper bound on the result of percentage closer filtering based on the mean and variance of a distribution of depths, which we showed provides a good approximation to percentage closer filtering. Finally, we have shown that variance shadow maps can be implemented easily on modern graphics hardware and compare favorably in both performance and quality to existing real time techniques for shadow map filtering.

Acknowledgements

We would like to thank Michael McCool and Stefanus Du Toit for their helpful suggestions, and Chris Iacobucci for the car model.

References

CROW, F. 1977. Shadow algorithms for computer graphics. In *Computer Graphics (Proc. SIGGRAPH)*, vol. 11, 242–248.

CROW, F. C. 1984. Summed-area tables for texture mapping. H. Christiansen, Ed., vol. 18, 207–212. Held in Minneapolis, Minnesota.

EVERITT, C., REGE, A., AND CEBENOYAN, C. 2000. Hardware shadow mapping. Tech. rep., NVIDIA Corp., Feb. Available at http://www.nvidia.com/.

FERNANDO, R., FERNANDEZ, S., BALA, K., AND GREENBERG, D. P. 2001. Adaptive shadow maps. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, ACM Press, New York, NY, USA, 387–390.

KIM, T.-Y., AND NEUMANN, U. 2001. Opacity shadow maps. In *Proceedings of the 12th Eurographics Workshop on Rendering Techniques*, Springer-Verlag, London, UK, 177–182.

LOKOVIC, T., AND VEACH, E. 2000. Deep shadow maps. In *Computer Graphics (Proc. SIGGRAPH)*, 385–392.

MARTIN, T., AND TAN, T.-S. 2004. Anti-aliasing and continuity with trapezoidal shadow maps. In *Proceedings of the 2nd EG Symposium on Rendering*, Eurographics Association, Springer Computer Science, Eurographics.

MCCOOL, M., AND DU TOIT, S. 2004. *Metaprogramming GPUs with Sh*. AK Peters.

REEVES, W., SALESIN, D., AND COOK, R. 1987. Rendering antialiased shadows with depth maps. In *Proc. SIGGRAPH*, vol. 21, 283–291.

STAMMINGER, M., AND DRETTAKIS, G. 2002. Perspective shadow maps. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, ACM Press, New York, NY, USA, 557–562.

WILLIAMS, L. 1978. Casting curved shadows on curved surfaces. In *Proc. SIGGRAPH*, vol. 12, 270–274.

WILLIAMS, L. 1983. Pyramidal parametrics. In *Computer Graphics (SIGGRAPH '83 Proceedings)*, 1–11.

WIMMER, M., SCHERZER, D., AND PURGATHOFER, W. 2004. Light space perspective shadow maps. In *Proceedings of the 2nd EG Symposium on Rendering*, Eurographics Association, Springer Computer Science, Eurographics.

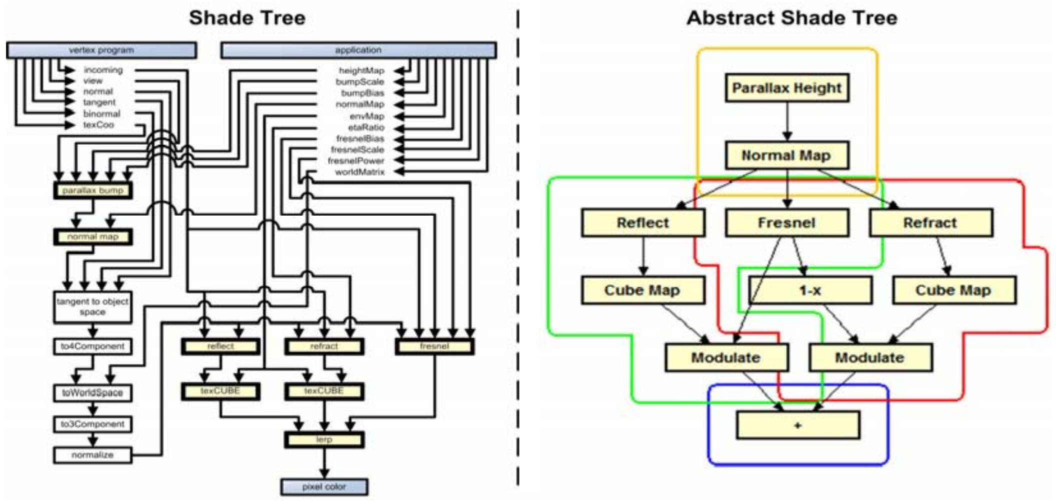


Figure 1 Revisited: *A conventional Shade Tree (left) for a "Bumpy Glass" shader, mocked up in the Visio drawing program. The equivalent Abstract Shade Tree on the right is an actual screenshot from our shader authoring plugin to the Eclipse IDE.*

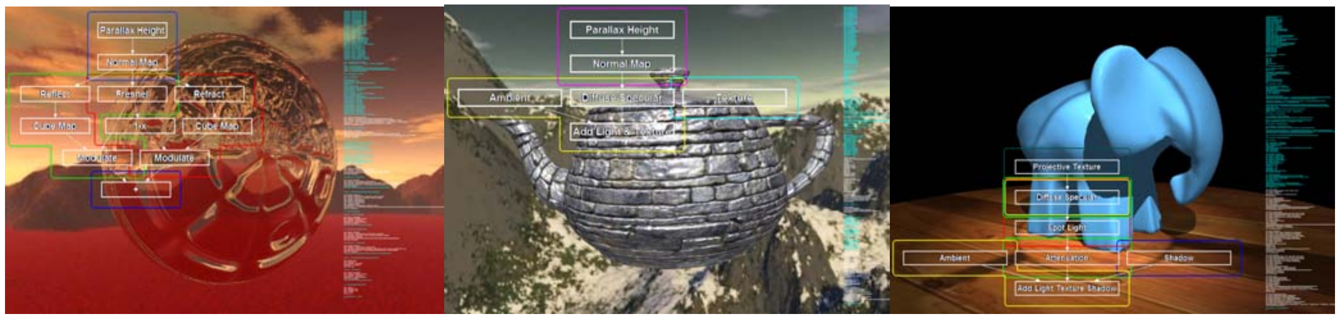
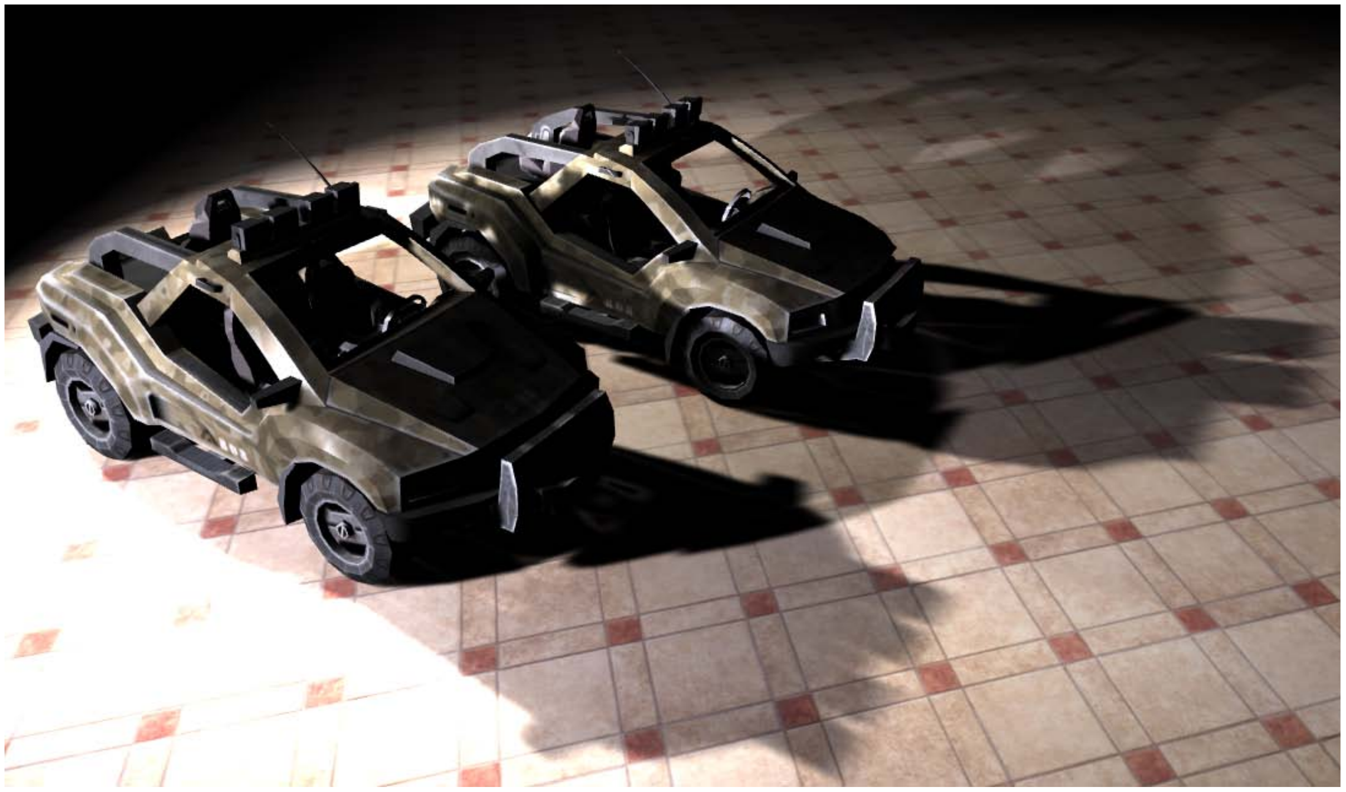


Figure 8. *Shade trees and the equivalent code composited over the effects they produce. (Color reproductions of Figures 4-7).*



Two cars and their shadows rendered with variance shadow maps. The resulting shadows have smooth penumbras, and are rendered without aliasing in real time.