Variance Shadow Mapping

Kevin Myers
kmyers@nvidia.com

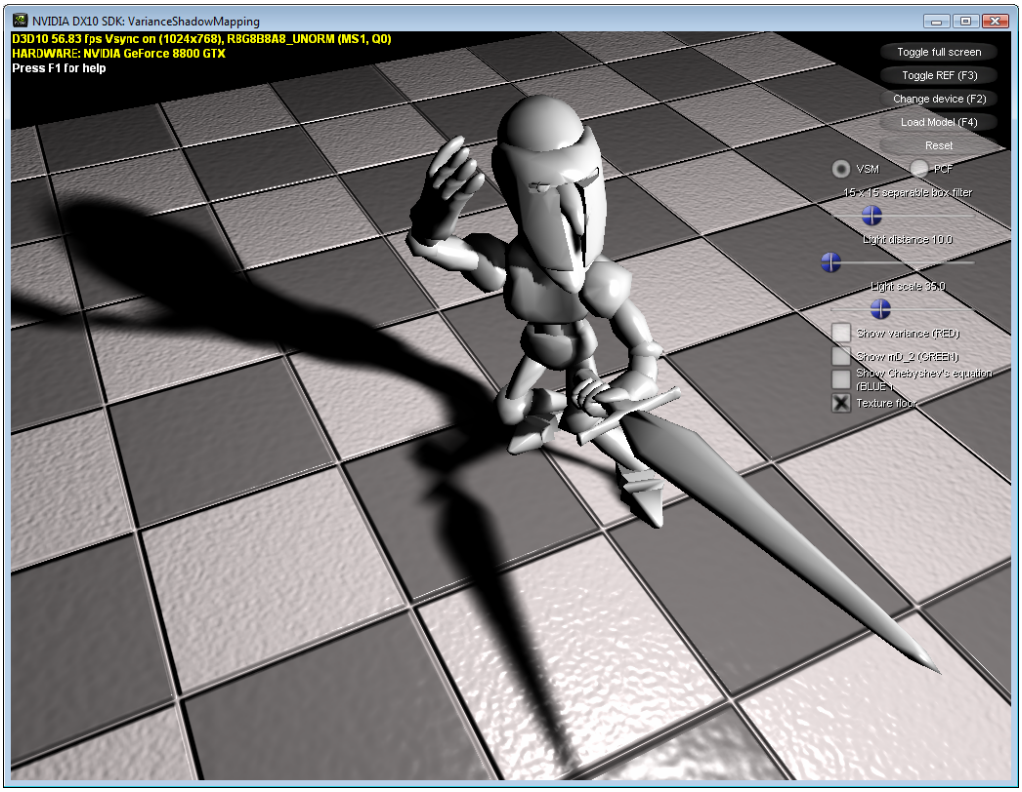
January 2007

Document Change History

Version	Date	Responsible	Reason for Change
			Initial release

Abstract

Variance shadow maps (VSM) replace the standard shadow map query with an analysis of the distribution of depth values. VSM employs variance and Chebyshev's inequality to determine the amount of shadowing over an arbitrary filter kernel. Since it works with the distribution and not individual occlusion queries the shadow maps themselves can be pre-filtered. This is novel, and allows for very fast soft shadowing with very large filter kernels.



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

Motivation

DX10-class hardware is perfect for VSM. Without fp32 filtering the algorithm must be done at fp16 precision which is not enough as the computation is numerically very unstable. In addition multisample anti-aliasing (MSAA) can be used to improve the quality of the shadow map by turning it on when rendering the shadow map.

How Does It Work?

Shadow maps work by first rendering depth from the point of view of the light to a texture. This texture now contains the closest fragments to the light. We can then query this shadow map when rendering our scene to determine if the fragment we're shading is occluded by a fragment in the shadow map. Percentage-closer filtering (PCF) achieves soft shadows by filtering an arbitrary number of these *queries*.

The problem with PCF is that there's no good way to pre-filter the depth values, or arbitrarily filter the shadow map with something like anisotropic filtering or mip-mapping. If you did, the depth value you arrive at is not a true occluder but the average of several possible occluders, which can produce artifacts.

Summary of the Algorithm

A detailed explanation of the algorithm can be found in the original Variance Shadow Map paper by William Donnelly and Andrew Lauritzen at http://www.punkuser.net/vsm/ .

In summary, the authors of VSM made the observation that what we're actually trying to do with PCF is obtain a bound on the percentage of pixels over the PCF filter region whose depth is greater than a single depth value, the value of the pixel we're shading. In other words we're comparing a single value to a particular distribution of values (the region we're sampling), and we want to know what percentage of those values are greater than the single value. We don't care about individual samples, just the percentage of the distribution. To that end Chebyshev's inequality gives us this bound given the average (or expected value $E(x)$) and variance of the distribution.

$$P(x \geq t) \leq \frac{\sigma^2}{\sigma^2 + (t - E(x))} \qquad (1)$$

$$when\ t > E(x)$$

The average is easy to get from a standard shadow map. We can filter in the shader, use mip-mapping or a separable blur. Variance σ^2 is also fairly trivial. Variance can be computed given the average value $E(x)$ the average squared value $E(x^2)$.

$$E(x) = average\ value\ over\ filter\ region$$

$$E(x^2) = average\ squared\ value\ over\ filter\ region$$

$$\sigma^2 = E(x^2) - E(x)^2$$

As we said before we can easily get the average value, to get the average squared we just need to change our shadow map to be a two component texture and render squared depth into the second channel. We can still filter arbitrarily before lighting our scene, then compute average and variance with a single texture lookup.

VSM Shader

Here is the VSM shader from DrawScene.fx:

```
float2 VSM_FILTER( float2 moments, float fragDepth )
{
      float2 lit = (float2)0.0f;
      float E_x2 = moments.y;
      float Ex_2 = moments.x * moments.x;
      float variance = E_x2 - Ex_2;
      float mD = moments.x - fragDepth;
      float mD_2 = mD * mD;
      float p = variance / (variance + mD_2);
      lit.x = max( p, fragDepth <= moments.x );

      …
      return lit;  //lit.x == VSM calculation
}
```

First of all note the final max. If the current fragment's depth is smaller than or equal to the average depth, then $t \leq E(x)$ and equation 1 does not apply. Figure 2 illustrates p or the result from Chebyshev's equation if we do not assume the bound $t > E(x)$. Notice how the blue area corresponds directly to the area of transition from light to dark in the shadowed scene. To get a feel for why this works, think about what causes variance to change. Variance increases whenever the distribution contains a wider range of values. Sharp changes in depth, such as a silhouette edge will produce higher values for variance. This is clearly shown in Figure 3. The areas of increased variance show up at the silhouette edges. Looking back at our equation these areas will drive the equation towards 1. On either side of the silhouettes variance quickly goes down allowing the equation to be influenced by the other variable $(t - E(x))$.

If we look at figure 4 we can see $(t - E(x))$ visualized in green. Notice how the opposite happens. As we approach the silhouette edge $(t - E(x))$ quickly decreases as the filtered depth values converge towards the fragment's depth value.

Figure 1 Shadow

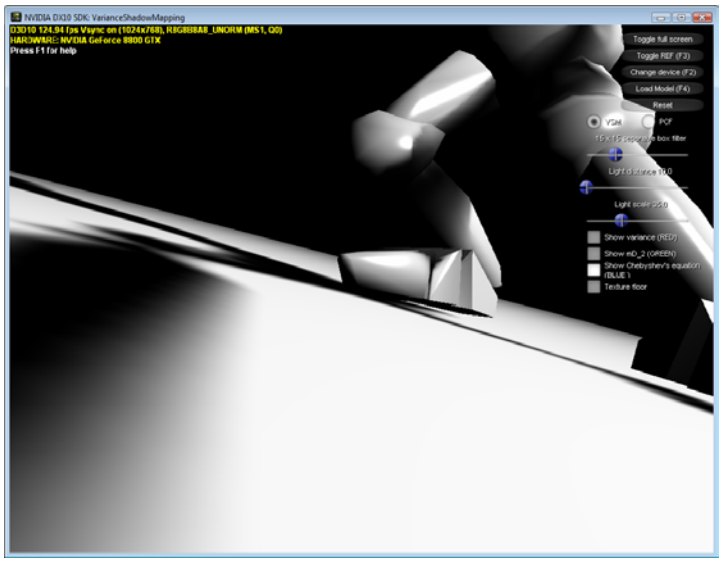


Figure 2 Chebyshev (unbounded)

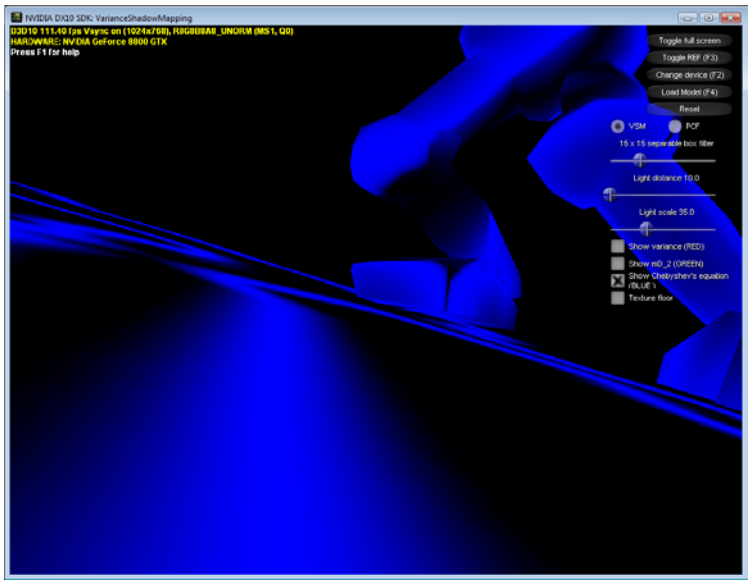


Figure 3 Variance

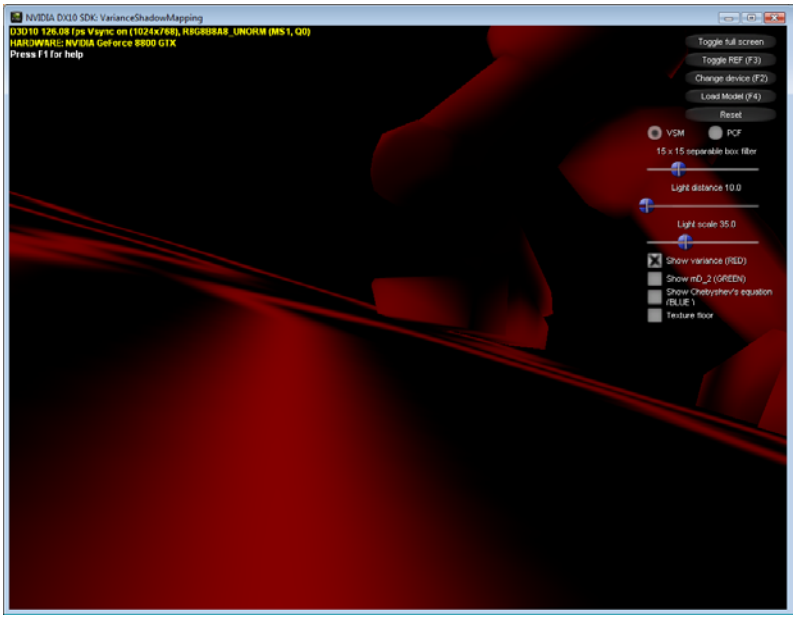
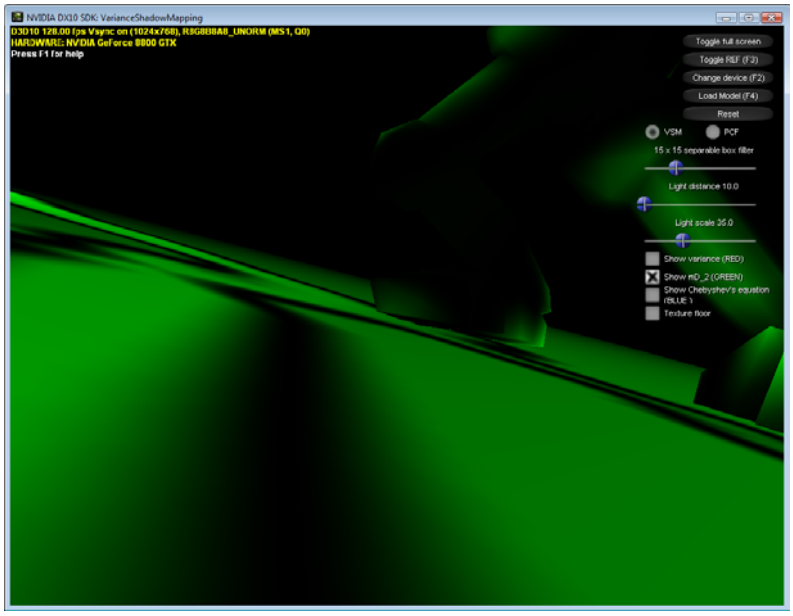


Figure 4



Implementation Details

- `VarianceShadowMapping.cpp`: The primary source file. This is where all resources are created and where the main render loops occur. There is a blur class `SeparableBlurEffectAndBuffer` that manages the separable blur shader and an extra render target to accomplish the blur.
- `VarianceShadowMap.h`: This file contains a class called `VarianceShadowMap` which contains an MSAA render target and Z buffer to render the shadow map, as well as an aliased buffer which the MSAA target is resolved to. It also manages the shader that generates the VSM.
- `GenerateVSM.fx:` This is the shader that constructs the shadow map.
- `Blur.fx:` This is the separable blur shader.
- `DrawScene.fx:` This is our scene shader.
- `VarianceShadowMapping.fxh:` A shader header file that all shaders pull in. We construct a poll from this file.
- `D3D10Quad.h:` Header file for a quad helper class:
- `D3D10Quad.cpp:` Contains a quad helper class. This creates an index buffer and a vertex buffer that form a quad tessellated to a desired level.
- `EffectWithOneTechnique.h:` Effect utility code.

Running the Sample

The sample has two sliders to control both the separable filter and the pixel shader filter. Playing with these two sliders quickly shows the performance benefit of being able to pre-filter the shadow map.

There are also two check boxes to help visualize the math. One shows variance in red the other shows mD_2 in green from our shader above. The section *VSM Shader* explains how to interpret these visualizations.

Performance

VSM really shines when a separable blur is used, which is something novel in shadow map filtering. Here are the results from using a 1024x1024 VSM running the sample at a resolution of 1024x768 on a GeForce 8800 GTX.

Filter size	Separable box filter FPS	PCF
7x7	~173	~128
15x15	~130	~48
49x49	~60	~6

Integration

VSM is fairly trivial to implement in an application that is already using shadow maps. The shadow map needs to be changed to an R32G32F texture to store depth and depth^2. When writing out depth one does not need to use projected depth, a better metric is linear distance from the light. Finally your lighting shader needs to handle the VSM math.

VSM does suffer from light bleeding when there is high depth complexity and high variance in depth values. The original paper goes through a proof that VSM computes exactly the correct result when assuming a planar occluder and a planar receiver. When this is not the case, sharp changes in variance will cause the equation to return a false positive, lit when it should not be.

Precision also has to be managed. Even with fp32 precision is an issue because we need to compute squared depth values. The problems can be seen by playing with the light distance slider. As the distance increases the depth values for our occluders get larger, and our squared depth values quickly start to suffer from precision issues. Moving the light scale slider changes what the depth values are scaled by helping mitigate the problem. In practice this can be managed in most scenes by keeping tight bounds on the light's viewing volume.

One benefit of VSM is that since it deals with a distribution of z values and not a simple depth test, biasing issues are no longer a problem. We do not need a z-offset to avoid precision issues.

Corporation products are not authorized for use as critical components in life support devices or systems without express written approval of NVIDIA Corporation.

Trademarks

NVIDIA, the NVIDIA logo, GeForce, and NVIDIA Quadro are trademarks or registered trademarks of NVIDIA Corporation in the United States and other countries. Other company and product names may be trademarks of the respective companies with which they are associated.

Copyright