# Antialiased Shadows

**Dharshan Vishwanatha**

November 6, 2020

# Contents

**1 Introduction** **2**

**2 Percentage-closer filtering (PCF)** **3**

**3 Variance shadow maps** **3**

**4 Moment shadow mapping** **5**

**5 Conclusion** **7**

**6 Reference** **8**

# Antialiased Shadows

Dharshan Vishwanatha

November 6, 2020

## 1 Introduction

Shadows are an important aspect of rendering as they give a sense of realism and gives context to the rest of the objects. So, it's important to consider adding shadows into our rendered scenes and getting the look of the shadows (i.e. anti-aliasing) just right. In this paper, I introduce multiple solutions to shadow mapping techniques and highlight their importance and flaws.

The basic principle of rendering shadows was introduced by Lance Williams in 1978. It consisted of rendering the scene both from the viewer's eye (camera) and the view direction of the light. More specifically, we render the depth values from the viewpoint of the light. This way, it tells us if a given object is in shadow or not.

Consider the scene:

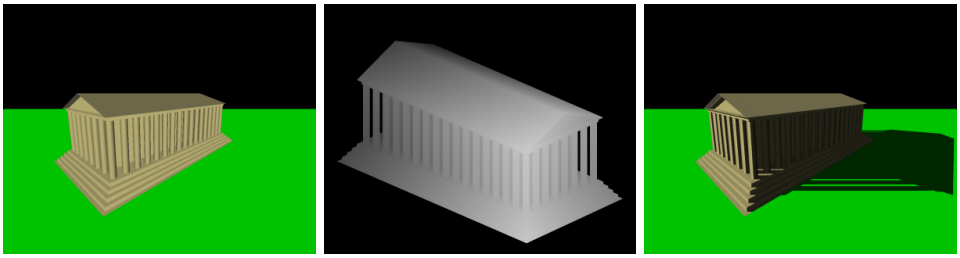

Figure 1: Going from left to right: Rendered from camera viewpoint. Depth image from light viewpoint. Shadowed render image. By Praetor Alpha.[2]

So, we check the distance from our 3d point from our camera viewpoint to the light. If this distance $d$ is bigger than the depth value on the right image, then we are in shadow. So we are sampling the depth value from the depth image texture. This results in very pixelated shadows on our surfaces and not very smooth shadows. Because, we are doing a binary check (i.e. are we in shadow or not), and pixels are discrete. Hence, we have pixelated shadows along the edges. Here is an example:

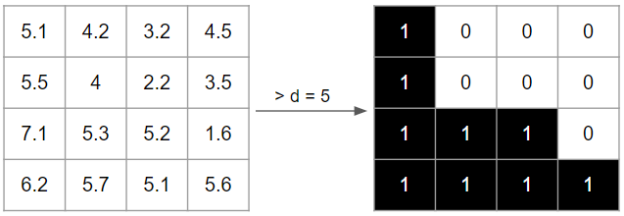


Figure 2: Zoomed and close up of the shadow map

From here on, I present multiple options to combat this: Percentage-closer filtering (PCF), Variance shadow maps, and Moment shadow mapping.

## 2 Percentage-closer filtering (PCF)

This is a simple way of smoothing the hard shadow edges. The idea is to sample nearby shadow map values (like an image kernel) and average out to get the average instead of binary shadow value. So from our previous binary example, applying PCF would give a filtered shadow:

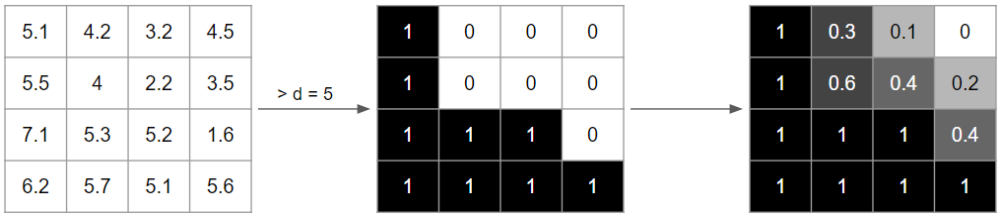


Figure 3: Apply the PCF with 3x3 kernel to get filtered shadows

1. Pros:
   - Easy and fast to implement to get soft shadows
   - We can get further apply "smoothing" if we take bigger $n \times n$ kernels
2. Cons:
   - Very costly to even do $4 \times 4$ or $5 \times 5$ per pixel. Falls short if we had multiple light sources.
   - Shadow Acne: floating point errors which causes depth test to fail. (Fixed by applying a small bias)
   - Multiple texture reads from the texture file per pixel

## 3 Variance shadow maps

When we are doing PCF, we are essentially taking the mean (average) of the neighboring shadow values. We are checking how much does a point pass or fails the shadow test. The Variance shadow maps (VSM) takes this idea further and offers a new way of sampling average shadows.

Instead of storing just the single depth value $z$ per pixel from the viewpoint of light, we also store the square of the distance ($z^2$) per pixel. This gives us a distribution of values stored in our texture. From the paper by William Donnelly and Andrew Lauritzen[4], they note that "we can approximate the average of two distributions by averaging the moments", where the two moments are $M_1 = z$ and $M_2 = z^2$.

To illustrate this, consider the diagram:

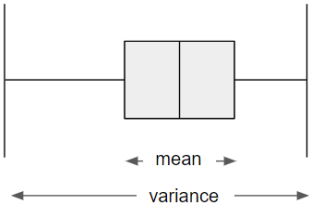


Figure 4: A Box Plot example with mean and variance

From looking at this, we see that if our new point we want to test is completely out of the variance, then it's completely in shadow or not. But anywhere in between, we take a percentage based on this distribution.

Moving on, we formulate these concepts in a mathematical foundation to find these percentages. Define the moments $M_1$ and $M_2$ as such, where $X$ is a random variable on that pixel:

1. $M_1 = z = E[X] = \int_{-\infty}^{\infty} x * p(X = x)\, dx$
2. $M_2 = z^2 = E[X^2] = \int_{-\infty}^{\infty} x^2 * p(X = x)\, dx$

Using these moments, we can design a distribution:

$$\mu = M_1$$
$$\sigma^2 = E[X^2] - E[X]^2 = M_2 - M_1^2$$

Since now we have this, we want to check how much this new point $X'$ with distance $t$ to the light is in shadow. We only care about the distance $t$ since that determines the shadow value.

So, define this value (or probability) be $P(X = x \geq t)$. Which tells us how much does the random variable depth occludes the given depth $t$. There is an upper bound on this probability, which is given by Chebyshev's inequality. This states that:

$$P(X = x \geq t) \leq \sigma^2/(\sigma^2 + (t - \mu)^2) = p_{shadow}$$

Finally, $p_{shadow}$ tells us how much is the point $X'$ with the distance t is in shadow. This is a numerical float value instead of a binary value, it acts just like PCF.

```
float vsmShadow(float2 moments, float depth){
    float pShadow = 0;
    float EX = moments.x;
    float EX2 = moment.y;
    float variance = EX2 - (EX * EX);
    float denom = (depth - EX) * (depth - EX);
    float shadowValue = variance / (variance + denom);
    //Clamp the percentage so it does not exceed 1.
    pShadow = min(max(shadowValue, depth <= moments.x),1.0);
    return pShadow;
}
```

1. Pros:
   - Removes unnecessary texture look up, just once's per pixel.
   - Already gets the averages per pixel with out looking at its neighbors.
   - Minimal to none additional changes and computations.
   - To make further soft edges, we apply a Gaussian blur on the shadow map values, and then sample the resulted blur values. This also allows to apply many different filters to get better results.
2. Cons:
   - Since we are using percentages, we need to figure out the right clamp values so that our values don't result in any artifacts. Such as Shadow Acne when the light view direction is almost parallel.
   - Peter Panning Shadows: This is when the shadows tend to not be attached to the model. Like its floating. This is fixed by clamping our shadowValue appropriately.

# 4 Moment shadow mapping

Moment shadow mapping takes the idea of the two moments from Variance Shadow Mapping but considers two more additional moments. This idea came to form the authors Christoph Peters and Reinhard Klein[1]. They note that their technique is "[l]ike variance shadow mapping it allows for the application of all kinds of efficient texture filtering and antialiasing to its moment shadow map...provide a substantially higher quality".

Just like in Variance Shadow Mapping, the moments are $M_1 = z$ and $M_2 = z^2$, but here the authors consider moments $M_3 = z^3$ and $M_4 = z^4$. They did consider other options: fewer moments and trigonometry functions. But, found that 4th moments give the best performance vs quality. Here are there results:

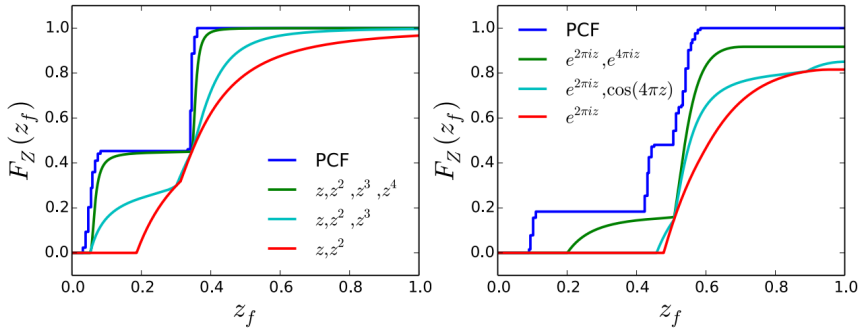


Figure 5: Shadow intensity with varying moments. By Christoph P. and Reinhard K. [1]

So the problem they are trying to solve is curve fitting with some depth values. They considered the polynomial basis: $\{z, z^2, z^3, z^4\}$ (Taylor Series) and trigonometry basis: $\{sin(2\pi z), cos(2\pi z), sin(4\pi z), cos(4\pi z)\}$ (Fourier Series). They note that the polynomial basis gives an error rate of 2.19%, while the trigonometry basis gives a 1.83% error rate. Since their difference is quite small, they considered the polynomials.

At first, they consider a general and well-defined moment problem with arbitrary moment size $m$. The problem boils down to:

**Problem:** (General Moment Problem). Given $I \subseteq \mathbb{R}, b \in \mathbb{R}^m, z_f \in I$, $\mathbf{b} : I \to \mathbb{R}^m$, and a search space defined:

$$\mathbf{G}(b) = \{S \in \mathbf{P}(I) | \mathbb{E}_s(\mathbf{b}) = b\}$$

where S is a probability distribution on interval $I$. And we want to find an infimum (greatest lower bound) on:

$$G(b, z_f) = \inf_{S \in \mathbf{G}(b)} F_S(z_f)$$

Breaking this down:

- $z_f$ : is the depth value we want to check how much is it in shadow.
- $b : I \to \mathbb{R}^m$ : is just the moments $b(z) = (z, z^2, ..., z^m)$. So in VSM $m = 2$ and $b(z) = (z, z^2)$.
- $\mathbb{E}_S(\mathbf{b}) = b$ : is referring to the expected value of some some distribution $S$. Like in VSM, the moment $M_1$ was the expected value.
- $\mathbf{G}(b)$ : is a set of distribution's such that the expected value is $b$.
- $F_S(z_f) = S(s < z_f)$ : is the cumulative distribution function. This is the same as $P(x \geq t)$ in VSM. And taking the minimum of these values will result in shadow factor.

- $G(b, z_f)$ : takes set of all the distribution, it returns the minimum cumulative distribution function. This cumulative is the shadow factor based on $z_f$.

They provide multiple general solutions to this problem, but the one at the end is the most relevant towards shadow mapping.

**Algorithm:** Hamburger 4MSM

---

**Input:**

1. $b \in \mathbb{R}^4$. Shadow map value with 4 moments per pixel.
2. $z_f \in \mathbb{R}$. Depth value to check how much is in shadow.
3. $\alpha > 0$, bias factor to avoid artifacts.

**Output:** $G(b, z_f)$ shadow intensity at the point $b$ with depth $z_f$.

---

1. $b' = (1 - \alpha) * b + \alpha * (0.5, 0.5, 0.5, 0.5)$
2. Let matrix A = $\begin{pmatrix} 1 & b'_1 & b'_2 \\ b'_1 & b'_2 & b'_3 \\ b'_2 & b'_3 & b'_4 \end{pmatrix}$, $x \in \mathbb{R}^3$, $z' = (1, z_f, z_f^2)$.
3. Then we have to solve for $x$, using $Ax = z'$.
4. Apply Cholesky decomposition to $A = LDL^T = L'L^T$. Then, we do forward substitution $L'y = z'$ (solve for y), and then backwards substitution $L^T x = y$ to finally solve $x$.
5. Solve for quadratic $x_3 * z^2 + x_2 * z + x_1 = 0$. Since its quadratic, we get two soln's. Let those soln's be $z_2, z_3 \in \mathbb{R}$, with $z_2 \leq z_3$.
6. If $z_f \leq z_2$ return $G = 0$.
7. Else if $z_f \leq z_3$, return $G = \frac{z_f * z_3 - b'_1 * (z_f + z_3) + b'_2}{(z_3 - z_2) * (z_f - z_2)}$
8. Else, return $G = 1 - \frac{z_2 * z_3 - b'_1 * (z_2 + z_3) + b'_2}{(z_f - z_2) * (z_f - z_3)}$

---

1. Pros:
   - Provides a higher quality shadow mapping with little to no artifacts.
   - Modified to also provide numerical stability
   - Other filters can be applied like Gaussian Blur in the shadow map.
2. Cons:
   - Since we want better quality, we have to use more information(data). Hence, we consider 2 more additional moments.
   - Although it is stable, it is prone to noise, such as Cholesky decomposition. Maybe other decomposition's like SVD will be more numerically stable.

# 5 Conclusion

This concludes with three various options to obtain smooth shadows. The main idea with all of them was to use moments and cumulative distribution function to obtain the average (expected) shadow intensity for each fragment depth. The most interesting part is how each idea is a linear progression from PCF to Moment Shadows, where later methods rely upon the previous. Also, the rise of quality and stability with the linear progression.

The final thing to note, both in VSM and MSM, we can apply Gaussian Blur to further make our shadows smoother. But, another way of improving the quality without applying any filter is to use the idea from the paper "Efficient High-Quality Shadow Mappin" by Sagar Bhandare[3]. In this paper, he provides a way of restricting the view frustum of the light in such a way to get higher quality shadow maps. We can reduce the frustum to just what the camera (eye) see's or just the shadowed region by using a bounding box. This way, the smaller the view frustum the higher quality shadow map texture we get, without any filtering.

# 6 Reference

1. Christoph P. and Reinhard K., *Moment Shadow Mapping*. University of Bonn, Germany. `http://momentsingraphics.de/Media/I3D2015/MomentShadowMapping.pdf`

2. Praetor A., *Shadow Mapping*. Wikipedia. `https://en.wikipedia.org/wiki/Shadow_mapping`

3. Sagar B., *Efficient High-Quality Shadow Mapping*. University of Florida. `https://ufdcimages.uflib.ufl.edu/UF/E0/04/50/80/00001/BHANDARE_S.pdf`

4. William D. and Andrew L., *Variance Shadow Mapping*. Computer Graphics Lab, School of Computer Science, University of Waterloo. `https://www.punkuser.net/vsm/vsm_paper.pdf`