# Parallel-Split Variance Shadow Maps

Andrew Lauritzen
Computer Graphics Lab, David R. Cheriton School of Computer Science, University of Waterloo

## Abstract

Standard shadow maps suffer heavily from aliasing for several reasons, including excessive magnification and poor texture filtering. Common linear filtering algorithms such as mipmapping and summed-area tables are inapplicable to typical shadow maps which require a non-linear depth comparison per pixel.

Parallel-split shadow maps (PSSM) provide a simple solution to the magnification problem by partitioning the view frustum into different zones, and using a different shadow map for each zone. PSSMs, however, do nothing to address aliasing due to the lack of shadow map filtering.

Variance shadow maps (VSM) address the shadow filtering problem by representing the shadow map in a way that can be linearly filtered. This allows the use of hardware anti-aliasing and texture filtering. However VSMs do not address extreme magnification artifacts due to poorly distributed shadow map data.

We present an algorithm that combines the advantages of parallel-split shadow maps and variance shadow maps. The resulting algorithm has a good distribution of detail over the camera frustum and also produces properly filtered shadows with arbitrarily soft edges.

## Parallel-Split Shadow Maps

We split the view frustum using planes parallel to the camera projection plane, assigning each of these regions a unique shadow map. Any occluder that may cast a shadow into one of these regions is rendering into the associated shadow map.

In this way, fragments near to the camera will be shadowed using a shadow map that only covers a small region of the world near the camera. Thus, by using several lower resolution splits rather than a single high resolution shadow map, more precision is appropriately allocated to the regions that are greatly magnified from the perspective of the camera.

## Variance Shadow Maps

By rendering depth and squared depth into a shadow texture, we recover the moments $M_1$ and $M_2$ of the depth distribution over the texture filter region, from which we can compute:

$$\mu = E(x) = M_1 \tag{1}$$
$$\sigma^2 = E(x^2) - E(x)^2 = M_2 - M_1^2 \tag{2}$$

Finally we apply Chebyshev's Inequality to approximate the probability that a surface at depth $t$ is in shadow:

$$P(x \geq t) \approx p(t) = \frac{\sigma^2}{\sigma^2 + (t - \mu)^2} \tag{3}$$

Blurring the variance shadow map before shading has the effect of clamping the minimum filter size, and produces uniform soft shadow edges.

Shadow Map (1024×1024)

Variance Shadow Map (1024×1024)

Parallel-Split Shadow Map (4 splits, each 512×512)

Split Visualization (4 splits)

Parallel-Split Variance Shadow Map (4 splits, each 512×512)

## Parallel-Split Variance Shadow Maps

1. Compute split frustums using the "practical split scheme"
2. For each split:
   (a) Find the minimum bounding rectangle of the split frustum in projected light space
   (b) Derive a projection matrix by "zooming in" the standard light space projection on the bounding rectangle
   (c) Render a variance shadow map using this projection matrix
   (d) Optionally blur the VSM, scaling the blur kernel size by the zoom factors computed in step 2b
   (e) Generate mipmaps for the VSM
3. When shading the scene, determine the relevant split using the fragment's $z$ coordinate in camera space
4. Compute texture coordinates using the projection matrix for the proper split
5. Perform a filtered texture lookup into the corresponding shadow map to recover the moments $M_1$ and $M_2$
6. Compute the mean and variance using (1) and (2)
7. Evaluate (3) and multiply the light contribution by $p(t)$

## Texture Coordinate Derivatives

To compute the texture filter size, modern graphics hardware uses finite differencing to approximate the screen-space texture coordinate derivatives. These derivatives are computed in "quads" of four pixels with each pixel having the same derivative as one of its neighbours in each screen dimension.

For simplicity let us consider only a single screen-space dimension. A problem occurs when two pixels in the same quad compute different split indices in step 3. In this case, two different projection matrices will be used to compute two unrelated texture coordinates, and thus the derivatives obtained by differencing these values will be meaningless. Consequently the texture filter size will be wrong and visible artifacts will appear along split seams.

A simple way to avoid this problem is to choose a single split per quad. To make this decision for a given fragment, we make use of hardware derivative instructions. Unfortunately given two adjacent fragments in the same quad with values $x$ and $x + 1$, computing $\delta x = (x + 1) - x = 1$ gives us no information from which to deduce $x$, since we do not know which of the two fragments is currently being shaded (both fragments will compute the same $\delta x$). However if we compute $\delta(2^x) = 2^{x+1} - 2^x = 2^x$ we can recover $x$ consistently for both fragments by computing the base two logarithm.

Using this trick extended to two dimensions, we can make an arbitrary choice about which split to use in a quad, and thus ensure that hardware derivatives and texture filtering work properly.