02561 - Computer Graphics

Improving shadows

Yevgen Zainchkovskyy (s062870)[*]

Due date: 12.01.2017, 23:59

Abstract Shadows are very important in computer graphics, providing visual cues about the shape and geometry of objects in the scene, as well as improving overall realism.

This project relates to Worksheet 8 of the course, exploring and implementing two additional improvements for Shadow Mapping: *Percentage Closer Filtering* (PCF) and *Variance Shadow Mapping* (VSM) resulting in higher quality of shadows. Motivation being that default Shadow Mapping results in poor quality of shadows with aliasing and absence of penumbra.

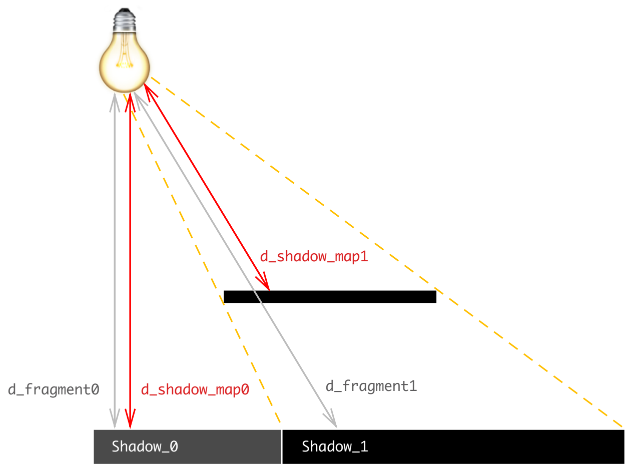
Final result of this project is a WebGL implementation of the above mentioned techniques (plus default Shadow Mapping for comparison) and can be found at the following url `http://www.student.dtu.dk/~s062870/02561/project/`

[*]Department of Applied Mathematics and Computer Science, Technical University of Denmark

Background

Recall general Shadow Mapping algorithm, which both the PCF and VSM are based upon, is a two step process where we first render the scene from the point of view of the light source and store the result (being depth) in a temporary texture.

Second, we render the scene as usual, but for every fragment we check if it is in the shadow. The way this test is done is by comparing if the current fragment is further away from the light than the distance stored in the shadowmap for that exact fragment. A figure to the right illustrates the general idea, and the calculations are along the lines of:



$$\begin{aligned}\textbf{Shadow}_\textbf{0} &= d_{\text{fragment0}} > d_{\text{shadow\_map0}} \Rightarrow \quad \textbf{NO}\\ \textbf{Shadow}_\textbf{1} &= d_{\text{fragment1}} > d_{\text{shadow\_map1}} \Rightarrow \quad \textbf{YES}\end{aligned}$$

Now the problem with a standard Shadow Mapping algorithm (given we solved both *Shadow Acne* and *Peter Panining*) is that the shadow does not look satisfactory. Focus being the edge of the shadow, where a harsh transition is being made: one pixel is in shadow and the next is not. Following is a screenshot from this project demonstrating the problem with the basic Shadow Mapping:

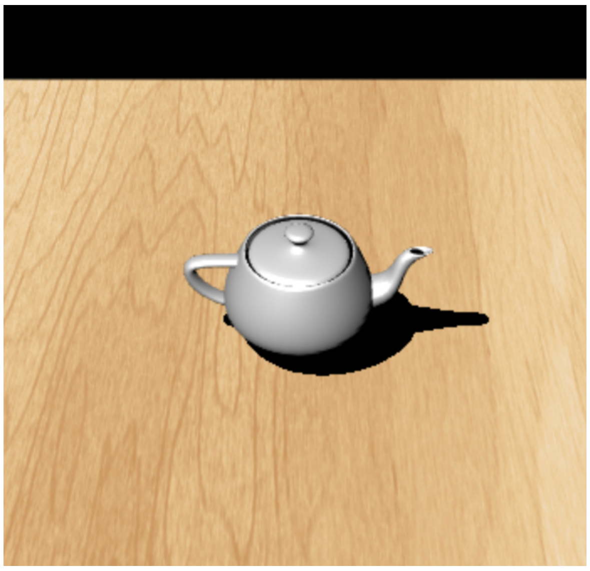


Figure 1: Basic Shadow Mapping.
`use_filter = off, ortho_size = 5`

While one could decrease the intensity of the whole shadow to get a little better result, the shadow looses it character. A solution we are looking for is a one that will give us a smoother transition (as seen in real life) but only around the edge of the shadow. Additionally, we wish for a variance in the softening, as shadows are harder near the object.

Percentage Closer Filtering

PCF is a step towards soft shadow edges, where the idea is to take several samples around the fragment in order to determine if the fragment is in shadow or not. It results in a so-called penumbra as an effect of attenuation of aliasing. The implementation resides in fragment shader (`index.html`, lines 133 - 146):

```
133 float variance = depth_2 - pow(mean, 2.00);
134 variance = max(variance, 0.005);
135
136 float p = smoothstep(distance - 0.02, distance, mean);
137 float d = distance - mean;
138
139 float p_max = linstep(0.2, 1.0, variance / (variance + d*d));
140 shadowCoeff = clamp(max(p, p_max), 0.0, 1.0);
```

Here we average across a 16 samples kernel and setting the `shadowCoeff`, which is later used as a multiplier for final `gl_FragColor` assignment. Visually the result is much better than the standard Shadow Mapping:

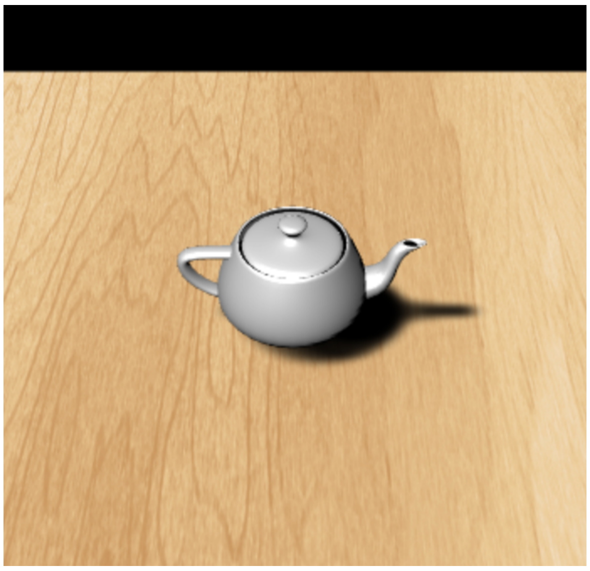


Figure 2: Shadow Mapping + PCF.
`use_filter = on, ortho_size = 18`

Notice the use of additional filtering (`use_filter = on`) and downsampling (`ortho_size = 18`) of the shadowmap which further increases the softening of the shadow.

Variance Shadow Mapping

While the result is quite satisfactory, it can still be improved. A technique called **Variance Shadow Mapping** (VSM) can be used to both get a more realistic penumbra, but also solve the performance regression introduced by the PCF as VSM takes advantage of hardware optimizations (via hardware implemented filters and OES standard derivatives extension[2]). Filtering a normal shadowmap the same way as ordinary color textures results in severe aliasing.

Briefly explained[3], Variance Shadow Mapping relies on a probabilistic prediction for the transition between shadow and no-shadow area. First we extend our ordinary shadow map by storing an additional value in the green channel: squared depth adjusted for biasing using partial derivatives (shadow fragment shader `index.html`, lines 256 - 271):

```
256 #extension GL_OES_standard_derivatives: enable
257
258 precision mediump float;
259 varying float vDepth;
260
261 void main(void) {
262   float depth2 = pow(vDepth, 2.0);
263
264   // approximate the spatial average of vDepth^2
265   // to avoid self shadowing.
266   float dx = dFdx(vDepth);
267   float dy = dFdy(vDepth);
268   float depth2Avg = depth2 + 0.50 * (dx*dx + dy*dy);
269
270   // depth saved in red channel while average depth^2 is
271   // stored in the green channel
272   gl_FragColor = vec4(vDepth, depth2Avg, 0., 1.);
273 }
```

Later, when rendering the scene, mean (μ) and variance (σ^2) of the distribution can be computed:

$$\mu = \texttt{vDepth}$$
$$\sigma^2 = \texttt{depth2Avg} - \texttt{vDepth}^2$$

Now for every distribution in which the mean and variance are defined, one can apply Chebyshev's inequality, which states that (generally) there are no more than $1/k^2$ of the distribution's values can be more than k standard deviations away from the mean.

Because variance can be interpreted as a width of a distribution, we can place a bound on how much of the distribution is "far away" from our mean μ. This bound is stated precisely in Chebyshev's inequality (one-tailed version): *Let x be a random variable*

[2] `OES_standard_derivatives` availability `http://webglstats.com/webgl/extension/standard_derivatives`

[3] Full VSM paper can be found here: `http://www.punkuser.net/vsm/vsm_paper.pdf`

drawn from a distribution with mean μ and variance σ^2. Then for $t > \mu$

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

Giving us the probability (in our case the fraction of samples) that will fail the distance test at a fixed distance t. The actual implementation in the fragment shader (`index.html`, lines 155 - 162) is as follows:

```
155 float variance = depth_2 - pow(mean, 2.00);
156 variance = max(variance, 0.005);
157
158 float p = smoothstep(distance - 0.02, distance, mean);
159 float d = distance - mean;
160
161 float p_max = linstep(0.2, 1.0, variance / (variance + d*d));
162 shadowCoeff = clamp(max(p, p_max), 0.0, 1.0);
```

While a perfect testbench for VSM would be an implementation that includes a large complex scene and some sort of filtering on the VSM texture, testing has been made with a special debug mode `debug_vsm_shadow` (available in the sidebar, along with two other debug modes). When enabled it shows the areas with high variance hinting at a correct VSM implementation.

Following is a screenshot showing the a scene in VSM debug-mode:

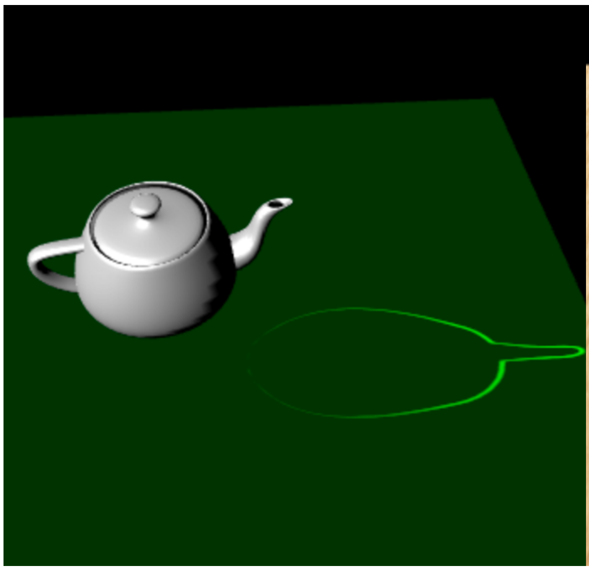


Figure 3: Debugging VSM.

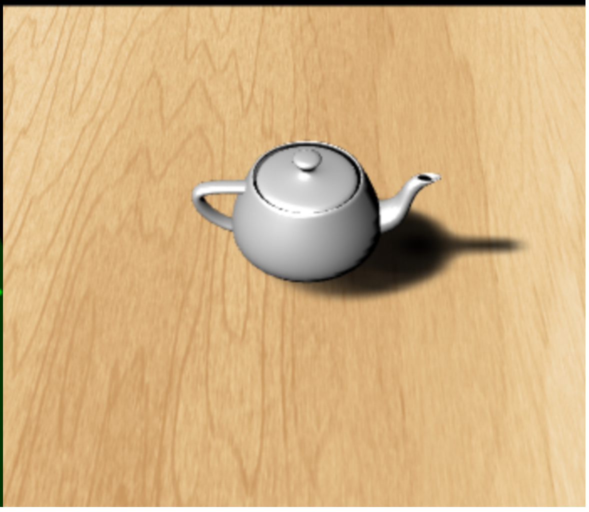


Figure 4: Shadow Mapping + VSM.

Bright green areas are areas with high variance. Notice lowered variance when the shadow is close to the object — this should result in a realistic penumbra, but unfortunetely the effect is hard to spot on such a small and simple scene.

Last, we present our final screenshot, a VSM with a slight 6×6 blur filter and slight downsampling applied.