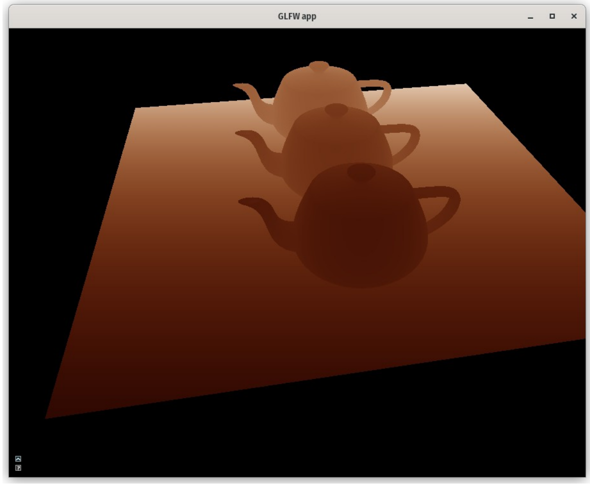
CS562 Project 2

https://momentsingraphics.de/Media/I3D2015/MomentShadowMapping.pdf

Synopsis

Implement the Moment Shadow Map (MSM) algorithm. Include a GPU filter pass (convolution blur or summed-area-table) between the generation of the shadow map, and it's use. Produce images with several different blur amounts, a small amount for anti-aliasing and large amount to simulate a diffuse lighting situation.



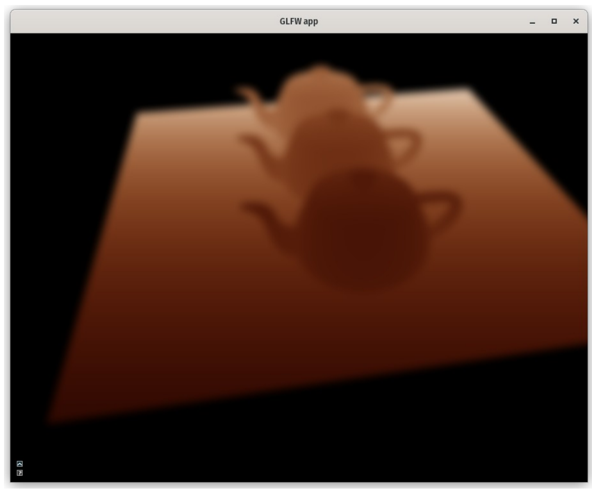
A Moment Shadow Map presented as a color image.

Instructions

Implement MSM on top of a traditional Shadow Map implementation with the following changes:

- Instead of storing depth z in a single channel shadow map, store (z, z^2, z^3, z^4) in a four channel shadow map.
- Filter (i.e., blur) the shadow map using compute shaders. Choose either a summed-area-table or convolution blur.
- For the shadow calculation during the lighting phase:
    - Calculate pixel depth, called z_f below. (As in normal Shadow Map).
    - Calculate light depth by projecting the pixel onto the shadow map and extracting the (now blurred) (z, z^2, z^3, z^4) values. (As in normal Shadow Map).
        - If using the summed-area-table, determine the amount of blur desired based on the pixel's environment, and retrieve the blurred values by combining 4 reads.
        - If using the convolution blur, the amount of blur was chosen at the time the blur was calculated, so just read a single pixel from the blurred shadow map.
    - Instead of comparing the two depths for a shadow amount of $s=0$ or $s=1$, run the z_f and (z, z^2, z^3, z^4) values through the MSM algorithm (reproduced below) to calculate a full range $s \in [0,1]$.
    - As a small step between the basic shadow map algorithm and the full MSM, consider variance shadow map: Use the first two values of the blurred (z, z^2, z^3, z^4) values, now called M_1, and M_2 to calculate

      $\mu = M_1$, $\sigma^2 = M_2 - M_1^2$ and $s = \dfrac{\sigma^2}{\sigma^2 + (t-\mu)^2}$



A Moment Shadow Map after a blur filter.

Relative depths

The MSM algorithm works best if the depths are adjusted to **relative** depths which range (roughly) over the interval [0,1]. To do this, use knowledge about the light's position and the scene's extent to estimate the smallest and largest depths from the light's point of view, say z_0

and z_1. Then the relative depth is $\bar{z} = (z - z_0)/(z_1 - z_0)$. Use this transformation for both the values going into the shadow map $(\bar{z}, \bar{z}^2, \bar{z}^3, \bar{z}^4)$ and the pixel depth $\bar{z}_f$.

Report

Submit a zip file containing the relevant code and a project report. Your project report should contain sufficient screen captures (and accompanying text) to demonstrate the correctness of your project, including:

- Images of the depth buffer both before and after the blur.
- Final images showing a range of shadows, from just enough blurring to be anti-aliased, to lots of blurring to produce very soft shadows simulating a wide light source.

Filter the shadow map

You have two choices here, both implemented in compute shaders. For efficiency sake, the computation will be split into separate horizontal and vertical stages.

- A fixed width blur using a convolution filter with a Gaussian weight kernel.
  - Advantage: This will give you a head start on a later project which needs a convolution filter.
  - Disadvantage: The blur width is fixed across the whole scene. Also, you'll miss an opportunity to learn the other technique.
- A summed-area-table.
  - Advantage: Each pixel can chose it's amount of blur according to it's situation, modeling real shadows. (See below.)
  - Disadvantage: ?

The details of implementing the filter step in a compute shader is in a separate document.

MSM details:

1. In the algorithm, b is the (z, z^2, z^3, z^4) vector retrieved from the blurred shadow map.
2. An α value of about $1\cdot 10^{-3}$ works well for me.
3. For solving the 3x3 system for $c = (c_1, c_2, c_3)$, I suggest two methods below. Cramer's rule is easy, but the suggested Cholesky decomposition is certainly more efficient and possibly more numerically stable.
4. The returned value G is reversed in sense from my usual s value. So the final lighting calculation is

   $\mathit{ambient} + (1-G)*[\mathit{diffuse} + \mathit{specular}]$
5. The algorithm from the paper:

Algorithm 3 Hamburger 4MSM (special case of Algorithm 2).
Input: Filtered sample from the moment shadow map $b \in \mathbb{R}^4$, fragment depth $z_f \in \mathbb{R}$, bias $\alpha > 0$ (e.g. $\alpha = 3 \cdot 10^{-5}$)
Output: Shadow intensity $G(b, z_f)$

1. $b' := (1-\alpha)\cdot b + \alpha \cdot (0.5, 0.5, 0.5, 0.5)^\mathsf{T}$
2. Use a Cholesky decomposition to solve for $c \in \mathbb{R}^3$:

$$\begin{pmatrix} 1 & b'_1 & b'_2 \\ b'_1 & b'_2 & b'_3 \\ b'_2 & b'_3 & b'_4 \end{pmatrix} \cdot c = \begin{pmatrix} 1 \\ z_f \\ z_f^2 \end{pmatrix}$$

3. Solve $c_3 \cdot z^2 + c_2 \cdot z + c_1 = 0$ for z using the quadratic formula and let $z_2, z_3 \in \mathbb{R}$ with $z_2 \leq z_3$ denote the solutions.
4. If $z_f \leq z_2$: Return $G := 0$.
5. Else if $z_f \leq z_3$: Return $G := \frac{z_f \cdot z_3 - b'_1 \cdot (z_f + z_3) + b'_2}{(z_3 - z_2)\cdot(z_f - z_2)}$.
6. Else: Return $G := 1 - \frac{z_2 \cdot z_3 - b'_1 \cdot (z_2 + z_3) + b'_2}{(z_f - z_2)\cdot(z_f - z_3)}$.

Cramer's rule

Cramer's rule for solving a 3x3 system involves the ratios of 3x3 determinants formed from the three columns of the matrix (here named A, B, C) and the right hand column (here named Z):

```
d  = det3(A,B,C)
c1 = det3(Z,B,C)/d
c2 = det3(A,Z,C)/d
c3 = det3(A,B,Z)/d
```

Where

```
float det3(vec3 a, vec3 b, vec3 c)   { return a.x*(b.y*c.z-b.z*c.y) + a.y*(b.z*c.x-b.x*c.z) + a.z*(b.x*c.y-b.y*c.x); }
```

Cholesky Decomposition

Cholesky is applicable to symmetric matrices (as ours is).

Decomposes a matrix into the product of a lower triangular matrix and it's (upper triangular) transpose. Solving a triangular system of equations is straightforward.

To solve this $\begin{bmatrix} m_{11} & m_{12} & m_{13} \\ m_{12} & m_{22} & m_{23} \\ m_{13} & m_{23} & m_{33} \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix} = \begin{bmatrix} z_1 \\ z_2 \\ z_3 \end{bmatrix}$ (note the symmetry):

Decompose as $\begin{bmatrix} a & 0 & 0 \\ b & d & 0 \\ c & e & f \end{bmatrix} \begin{bmatrix} a & b & c \\ 0 & d & e \\ 0 & 0 & f \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix} = \begin{bmatrix} z_1 \\ z_2 \\ z_3 \end{bmatrix}$ (*)

which is $\begin{bmatrix} a^2 & ab & ac \\ ab & b^2+d^2 & bc+de \\ ac & bc+de & b^2+d^2+f^2 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix} = \begin{bmatrix} z_1 \\ z_2 \\ z_3 \end{bmatrix}$.

With the substitution $\begin{bmatrix} a & b & c \\ 0 & d & e \\ 0 & 0 & f \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix} = \begin{bmatrix} \hat{c}_1 \\ \hat{c}_2 \\ \hat{c}_3 \end{bmatrix}$ (*) becomes $\begin{bmatrix} a & 0 & 0 \\ b & d & 0 \\ c & e & f \end{bmatrix} \begin{bmatrix} \hat{c}_1 \\ \hat{c}_2 \\ \hat{c}_3 \end{bmatrix} = \begin{bmatrix} z_1 \\ z_2 \\ z_3 \end{bmatrix}$.

Algorithm:

Warning: Because of round off errors, the three square roots below may be of negative values. In that case set the result, a, d, or f to a very small (but non-zero) value, say 10^{-4}.

$cholesky(m_{11}, m_{12}, m_{13},\ \ m_{22}, m_{23},\ \ m_{33},\ \ \ z_1, z_2, z_3)$:

$$a = \sqrt{m_{11}}$$
$$b = m_{12}/a$$
$$c = m_{13}/a$$
$$d = \sqrt{m_{22} - b^2}$$
$$e = (m_{23} - bc)/d$$
$$f = \sqrt{m_{33} - c^2 - e^2}$$

$$\hat{c}_1 = z_1/a$$
$$\hat{c}_2 = (z_2 - b\,\hat{c}_1)/d$$
$$\hat{c}_3 = (z_3 - c\,\hat{c}_1 - e\,\hat{c}_2)/f$$

$$c_3 = \hat{c}_3/f$$
$$c_2 = (\hat{c}_2 - e\,c_3)/d$$
$$c_1 = (\hat{c}_1 - b\,c_2 - c\,c_3)/a$$