Variance shadowmaps, Highlighting, Rainmaps, Celshading and more

Niklas Krave* Johan Kllberg†

Lund University
Sweden

Abstract

Through the techniques used a reduction in artefacts in shadows has been managed. Most important and significant though is the effects of visual changes such as rain maps and celshading. The scene created simulates rain that seem to fall down around the viewer at different distances. The viewer has the option to seek shelter which will make the rain fall only at a distance. If a area is not roofed the area will be covered in continuous rain splashes and will have a look as though it is wet.

All objects in the scene will have a grayscale color or a red color.

The scene created has a gritty look but due to many checks and look ups in every pixel the scene is quite performance heavy. With more time and work the performance could probably be a lot more optimized though.

1 Introduction

The goal of this project was to create a scene with a similar look to the movie sin city. That is a scene with a non-realistic look, especially concerning colors and lighting.

A lot of different techniques can be used to achieve this goal and it gives a lot of flexibility to add things like rain and objects.

2 Algorithms and application

2.1 Grayscale, levels of color and a highlight color

Early on a few simple effects were created to get a different feel on the sponzaa scene. A grayscale was created by giving the color of the pixel the mean of the textures rgb values. A few levels of color was defined by only outputting a few set values to the color of each pixel depending on its gray scale value. If the grayscale value was within a certain range this defined it's value. The same thing was done to the lighting of the scene. These effects gave the scene a flat look. This technique is the original meaning of celshading which is nowadays mostly known for its hard outlines which wasn't implemented in this project. On top of the grayscale there is a highlight color added. By taking the dot-product of the pixel color and our highlight color and checking if this was within a certain range a decision to not change the color of this pixel and let the light affect this pixel less could be made.

2.2 Variance shadow map

The idea of the variance shadow map(VSM) is to treat shadows as a probability instead of a binary value. The technique is quite similar to regular shadow maps with Percentage-Close Filtering (PCI) and if no extra hardware filtering will be applied the same shadow map can be used for both techniques. The difference is the data stored in the sampling. With PCI only the depth is stored but when using VSM depth squared is also needed. Following is the chebychev formula which is used for VSM to calculate the probability of an area being lit.

$$M_1 = E(x) = \int_{-\infty}^{\infty} xp(x)dx$$

$$M_2 = E(x^2) = \int_{-\infty}^{\infty} x^2 p(x)dx.$$

$$\mu = E(x) = M_1$$

$$\sigma^2 = E(x^2) - E(x)^2 = M_2 - M_1^2.$$

$$P(x \geq t) \leq p_{\max}(t) = \frac{\sigma^2}{\sigma^2 + (t-\mu)^2}.$$ [Lauritzen 2007]

For VSM x is the depth and instead of an integral over the whole depth a simple boxfilter surrounding the current pixel was used. This makes it a necessity to clamp the variance sigma^2

to a small value above zero as a negative variance shouldn't be possible.

2.2.1 Light bleeding

A major issue with variance shadow maps is light bleeding. Light bleeding occurs when a point has two or more occluders between itself and the light source and the occluder closest to the light source only partially occludes the light. This gives the effect of a soft shadow on every point behind it, and not only the second closest like intended. This is a hard problem to solve. GPU gems 3 [Lauritzen 2007] advices a method were some of the soft shadows simply are removed. This is accomplished by not using the full range of the calculated probability from zero to one and instead only take an upper value from x to one. This range is then transformed to a range from zero to one. Another method implemented stores the largest depth in the shadow map while the first and second moment is calculated. This value is then compared to the depth of the current pixel. If the current pixel has a larger depth than the largest value interpolated over in the shadow map the pixel is set as 100 percent shadowed. This lets the edge cases were some of the pixels are not occluded get a soft shadow. One issue with this method is the cases were a piece of geometry cast a shadow on one of it's own surfaces. Since our shadow buffer takes uses back face culling the depth in the shadow buffer is very close to the depth of a pixel that should be shadowed in these cases. To solve this a small value epsilon can be withdrawn from the max depth value.

One problem when dealing with shadows is Shadow acne when a surface shadows itself due to small differences between the depth computation of the shader and the depth computation of the shadow buffer on the same surface. To mitigate this the second moment M2 can be given some information about the tilt of the surface of the current shadow buffer texel. The second moment still describes the depth squared and the mean value is still E(u^2).

The tilt of the pixel has the standard deviation of half a pixel in both the x and y direction and the partial derivatives can be calculated by the hardware. This gives us the second moment M2 as:

$$M_2 = \mu^2 + \frac{1}{4}\left(\left[\frac{\partial f}{\partial x}\right]^2 + \left[\frac{\partial f}{\partial y}\right]^2\right).$$

The added term is called a shadow biasing value.

*e-mail: niklaskrave@gmail.com
†johan.kallberg88@gmail.com

2.3 Rain effect

2.3.1 Rendering on cones

The rain effect close to the viewer is created with a technique similar to the one developed by [Wang and Wade, 2004]. This uses a geometric object in the form of a double cone with the camera in the center. On this double cone a texture is added and translated to simulate the falling rain. The texture is added three times with different translation speeds and the colors are then added together to simulate different distances of rain.

The double cones shape makes it look like the rain is falling from the sky when looking up and at the ground when looking down.

2.3.2 Rain map

A height map was created through the use of an added camera that looks down on the scene. The z-value for the camera is then encoded and stored in a texture. In the shader for the double cone the value in the height map is compared to coordinates of the cone translated to the height cameras matrix and if the z-value of the cone coordinate is bigger than that in the height map this means that pixel of the cone is under some kind of protection. In these cases the cone will be removed.

In cases where the cone is removed the rain is instead shown on four quads that are located at the start of the four possible covers in the scene.

2.3.3 Wet effect

The height map created for the rain effect is also used in the shader for the lighting in the scene. The same check is made as in the rain shader to see if the pixel being lighted is under cover, if the pixel isn't under cover the area will have a much higher specular and shininess value which will create a wet look in the area.

2.3.4 Splash effect

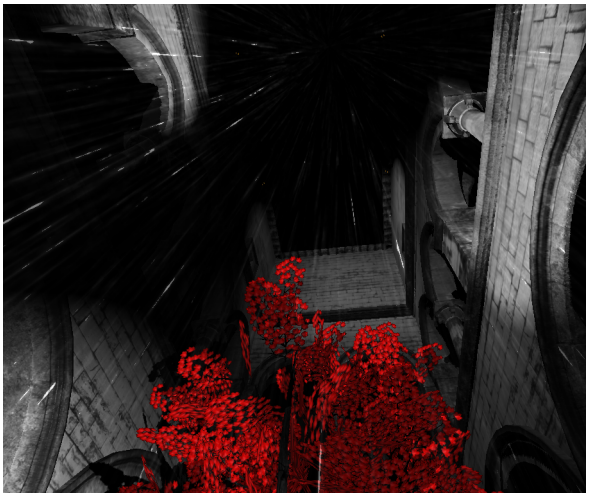
The rain splashes are created trough a particle simulation where objects will be quickly generated and then removed over the area that is not covered.

2.4 Discarding pixels

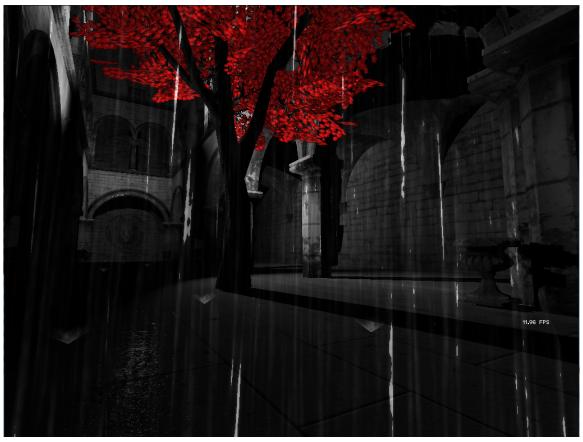
One object was brought into the sponzaa scene brought a small additional challenge with it. It was a tree that had flat planes with a texture on it for its twigs and leaves. Parts of each plane had wasn't covered by the twig or leaves and wasn't intended to be rendered. This method lowers the number of triangles needed for the object significantly. The texture used was completely black were it wasn't supposed to be rendered. In the fragment shader the openGL discard command was called if the pixel contained next to no color information. The discard command removes the pixel from the rendering pipeline. The check had to be done in not only the geometry buffer but also in the shadow buffer to allow the leaves to cast shadows correctly. This means that those shaders has to be sent a texture which wasn't previously needed.

3 Results

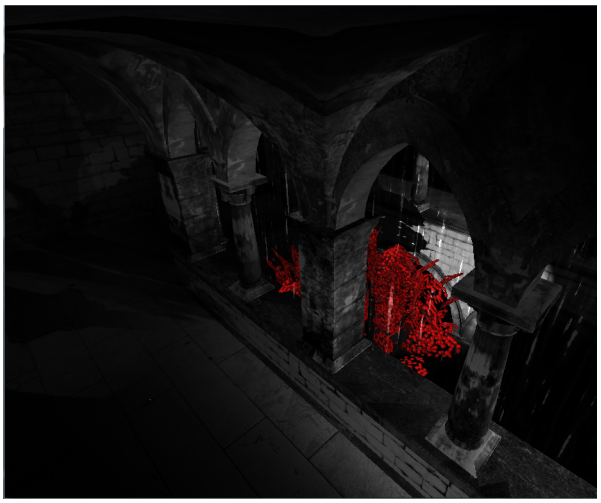
The result is visually pleasing with rain fall, wetness and splashes that looks quite realistic from most directions.



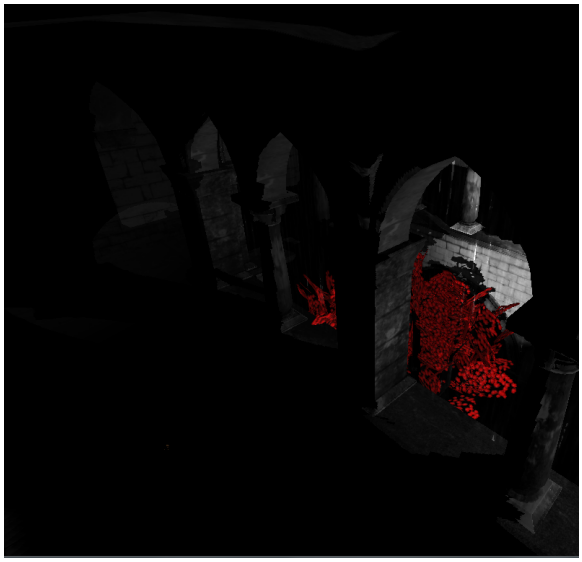
Looking up in the sky, you can see the rain falling down



Looking straight forward, notice the splashes on the ground



Scene with light bleeding



Scene without light bleeding

4 Discussion

4.1 Variance shadow map

The algorithm in itself was no harder than simple shadow mapping to implement and it comes with the added benefits of linear filtering at close to no additional performance cost. The method of removing softness proved insufficient in our scene and close to all shadow softness had to be removed to get rid of the light bleeding. Remove light bleeding by using the max depth with an epsilon offset worked very well and removed several ugly shadow artefacts in the scene. The algorithm we used to remove shadow acne did not have much of an effect on the scene however most of the shadow acne artefacts were removed when the epsilon value was added to the max depth check. this was probably due to many shadow acne artefacts are created due to small differences in the depth check which the epsilon value amended. The combined result is a scene with much nicer shadows in general. Some aliasing artefacts still exists in the scene and these bugs would be next in line to be addressed before looking into enhancing the shadows with different filtering techniques.

4.2 Rain with rain map

The rain effect is acceptable but could be improved by adding opacity to the edges of the cones to remove some artefacts, due to limited time frame and the artefacts not being major this was not done.

Some scaling could also be added to the different texture layers of the rain to simulate distance more but the current look of the rain has been considered good enough.

Concerning the quads used for when the cone is covered this method is not very flexible and the quads has to be added manually. For a more flexible approach one could consider using some kind of algorithm to find the areas where cover is reached or adding a second double cone that is larger and therefore shown further away from the camera and for a larger distance under cover.

4.3 Wet effect

The wet effect is costly because of many lookups to make the edges less noticeable, to increase performance one could consider using a more simulated approach closer to the quads in the rain effect where the area that is supposed to be wet is manually calculated.

4.4 Splash effect

As is now the particles in the particle system will only be shown on the ground, using the height map one could create an effect where the particles are shown on the first object they would reach when falling from the sky but this would be costly for performance.

4.5 Discarding pixels

It is a quite simple method to implement and it came with no noticeable performance hit. Compared to the number of polygons that would be needed for each twig and leave it rather saves performance and the resulting model looks quite good despite the cheat. The method of checking the color value could cause unwanted pixels to be discarded if those pixels were coloured black. It would be better if the texture was given an alpha value of zero for the parts to be discarded, alternatively a separate texture in black and white could be used as an alpha channel if the method of checking color value was producing problems.

5 References

Lauritzen, A, 2007,Summed-Area variance Shadow Maps http://http.developer.nvidia.com/GPUGems3/gpugems3$_ch08.html$

[Online; accessed 10-Dec-2013]

Wang, N and Wade, B., 2004. Rendering Falling Rain and Snow. http://www.cse.iitb.ac.in/graphics/ pisith/references/

[Online; accessed 10-Dec-2013]