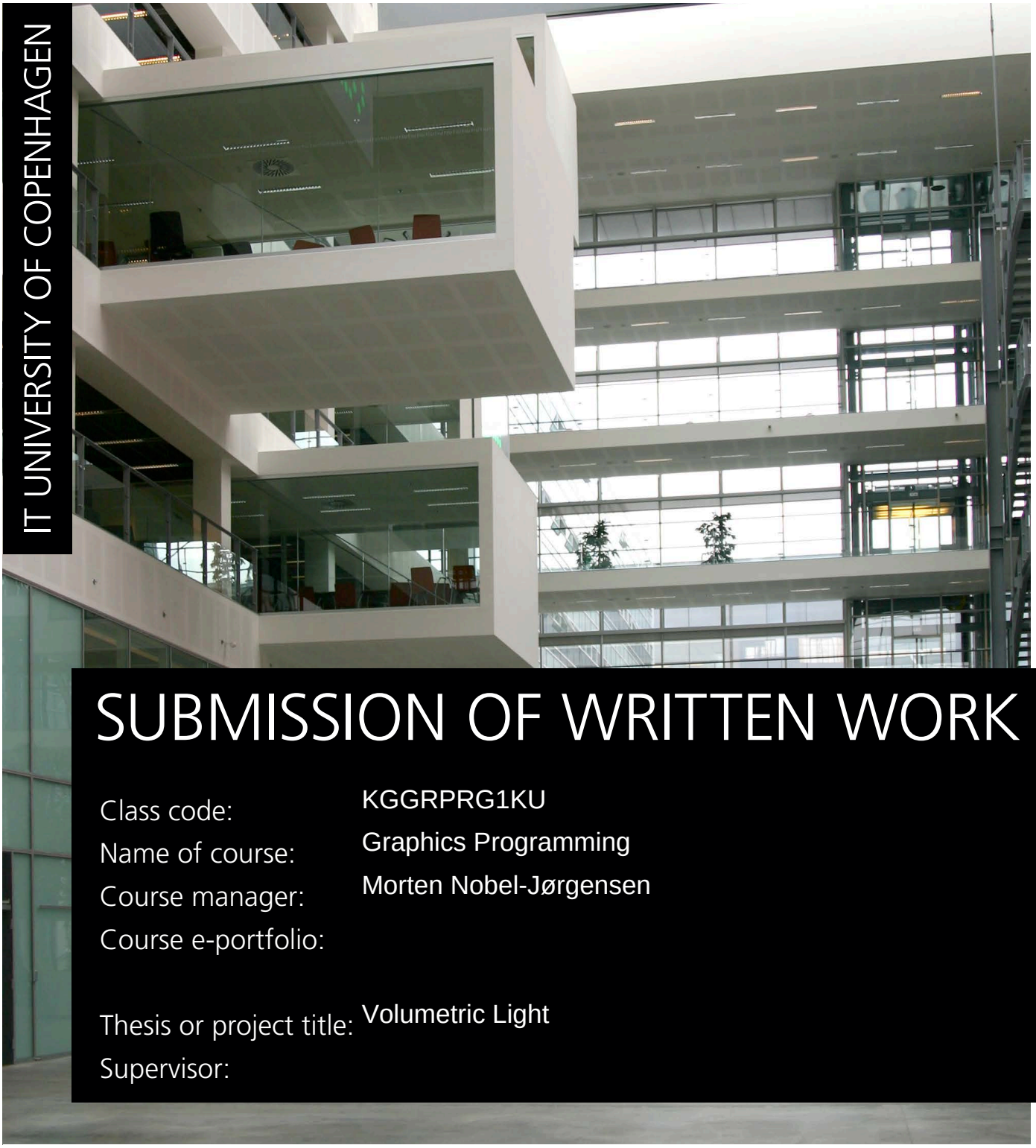
IT UNIVERSITY OF COPENHAGEN



SUBMISSION OF WRITTEN WORK

Class code: KGGRPRG1KU

Name of course: Graphics Programming

Course manager: Morten Nobel-Jørgensen

Course e-portfolio:

Thesis or project title: Volumetric Light

Supervisor:

	Full Name:	Birthdate (dd/mm-yyyy):	E-mail:
1.	Kasper Honnens de Lichtenberg	11/01-1993	khon@itu.dk
2.			@itu.dk
3.			@itu.dk
4.			@itu.dk
5.			@itu.dk
6.			@itu.dk
7.			@itu.dk

Volumetric Light - Graphics Programming

Kasper Honnens de Lichtenberg

May 2017

1 Introduction

In this report I will present the techniques used to create my demo, which entails deferred rendering, shadow mapping, volumetric light and post-processing. In figure 1 the final result can be seen with voluemtric light creating caustic-like effects. Volumetric light was implemented in a bruteforce-like fashion but the sampling used the light attenuation formula to distribute samples in such a way that most samples was close the camera and because the distance between samples calculated made for the perfect dithering amount to remove artifacts.

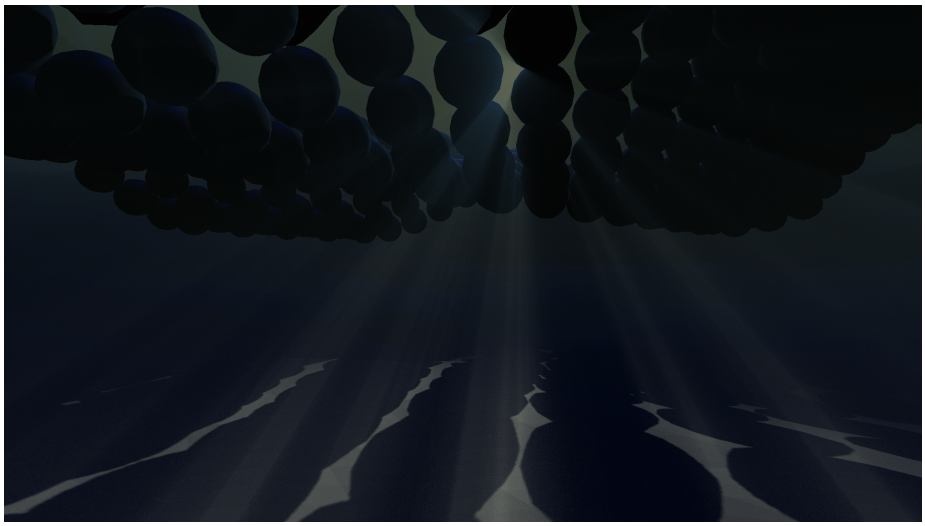


Figure 1: The final result.

2 Deferred Rendering

The technique Deferred Rendering is used in many games, both because it allows for a lot more lights than forward rendering and because it allows for easy and cheap post-processing. For this project I used 3 framebuffers one for geometric

properties, one for shadow maps, one for the light. In figure 2 the textures saved for deferred shading are showed, the position, normals, color and specular are all geometric properties saved into four textures, the shadow map is calculated before the geometric pass and the light is calculated from the geometric properties and the shadow map, it is then saved into yet another texture for post-processing.

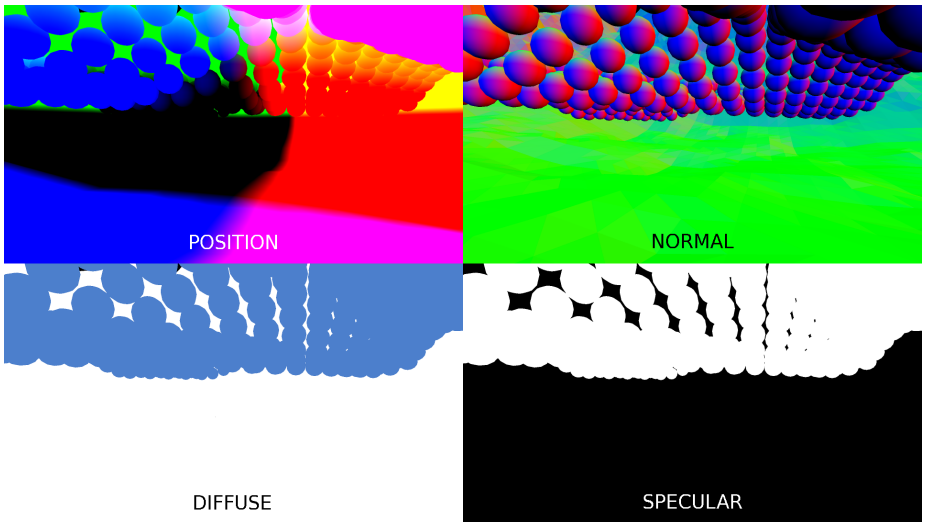


Figure 2: Position, Normals, Color and Specular is all saved to four textures in one pass, Light is calculated after that using the values and a shadow map.

After the light pass their is one final pass for post-processing, in this case just depth fog. The result can be seen in figure 3. There is also another pass which renders the light as primitives for debugging reasons, finally ImGui is drawn.

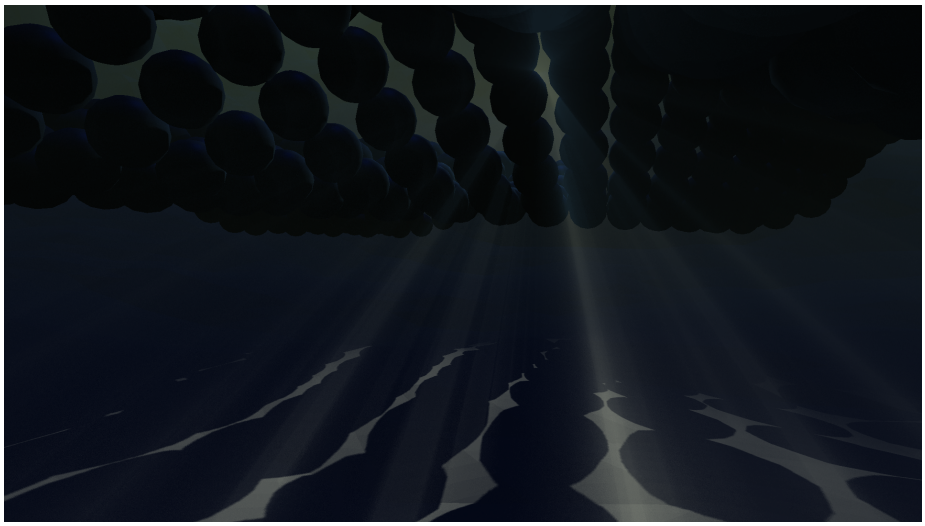


Figure 3: Result after Post-Processing, which includes depth fog and some blurring.

2.1 Light Influence Mesh

One ability deferred rendering is praised for is the ability to render a lot of lights, but it requires additional techniques to accomplish. The technique I chose to implement was calculating the influence of each light source and shaping a mesh from it, I chose this because the ability to find the influenced volume, by back-face and front-face comparison, would improve performance. Additionally to the light mesh another problem can occur, if the camera is within the light mesh no light calculations will occur. This happens because by default back faces are culled, a fix for this is using a stencil operation to increment and decrement the stencil buffer based on whether it is a front or back facing fragment. Then using the stencil buffer to pass the fragments that the light should render.

I ended up using a very crude approximation and an sphere mesh for point lights as well as spot lights. This makes the technique a lot less useful, since I only calculate volumetric light for spot lights. The technique only gave the demo the ability to have a lot of lights, but did not increase performance of volumetric light calculations.

3 Shadow Mapping

Shadow mapping was quite important for volumetric light, since the blocking of light creates volumetric shadows which enables the caustic-like effects. In figure 4 a linearized shadow map can be seen from the spot lights perspective. The shadow maps supports both directional lights and spot lights but do not

support point lights and the engine only supports one shadow map, because I chose to focus on other areas.

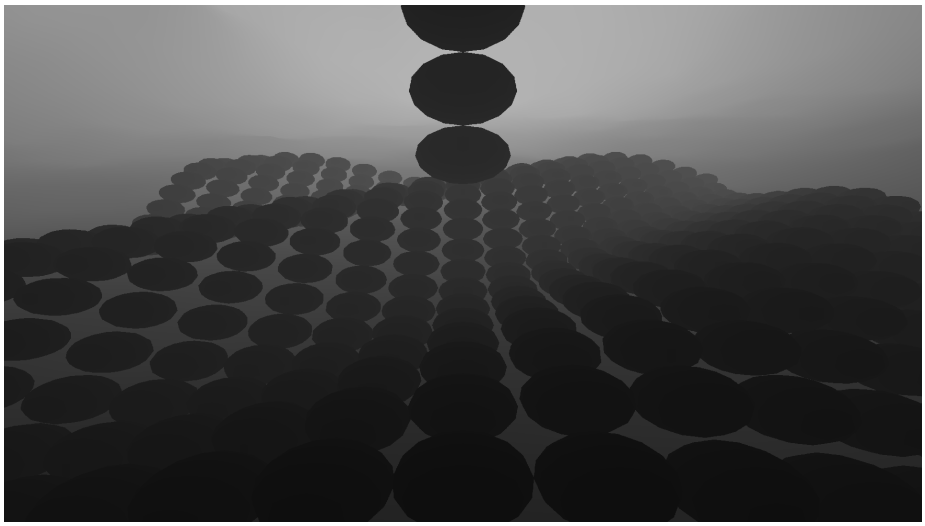


Figure 4: Linearized depth map from the spotlight's point of view.

4 Volumetric Light

In order to create volumetric light each pixel within the light influence is calculated, by raymarching from the camera and repeatably moving a given length until the depth buffer. For each sample the light is calculated which includes calculating the spot penumbra, attenuation and shadow for the position of the sample and adds it to the pixel. When it has hit the depth buffer it will divide the pixel with the number of samples.

The raymarching distance between samples is calculated with the same formula as light attenuation but to calculate the distance moved between samples it can be expressed like the following:

$$Number\ of\ samples\ per\ meter = \frac{1}{att.x + att.y * dist + att.z * dist * dist}$$

$$Distance\ to\ move\ per\ sample = \frac{1}{\frac{1}{att.x+att.y*dist+att.z*dist*dist}}$$

$$= att.x + att.y * dist + att.z * dist * dist$$

The formula can be seen plotted in figure 5 and 6 for the specific values i used, which was $att.x = 0.2, att.y = 0.01, att.z = 0.001$ but can be set within the demo, see "Raymarching Attenuation". The attenuation ensures that the majority of samples is closest to the camera and less samples are wasted further

from the camera, while keeping the delta time within a somewhat acceptable range.

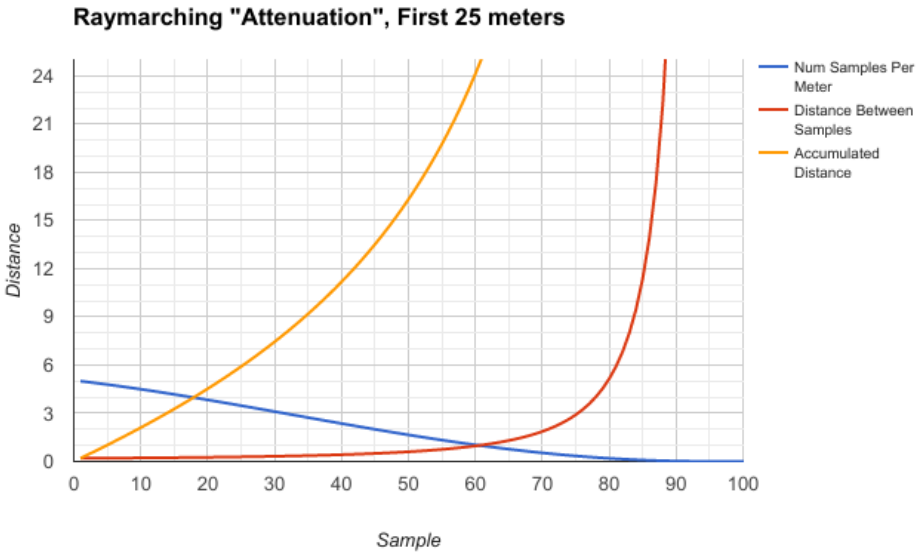


Figure 5: A plot of the raymarching attenuation with only the first 25 meters on the y-axis. Note where the Num Samples Per Meter crosses Distance Between Samples, at this point the sampling goes from sampling below a meter to above a meter, i.e. the first 60 samples is within the first 25 meters.

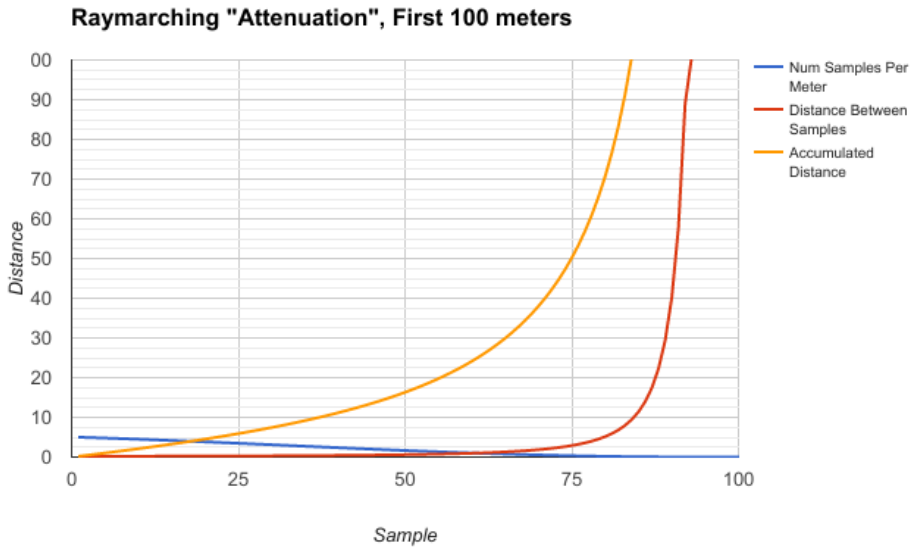


Figure 6: A plot of the raymarching attenuation with only the first 100 meters on the y-axis. Note that the first 50 samples is within the first 15 meters and the next 25 samples is within the next 35 meters.

By using the raymarching attenuation distance between samples to dither the sample position, it removed a lot of artifacts that the raymarching produces. The sample position is offset by noise multiplied with the sample distance, the effectiveness can be seen in figure 7. This value can also be tweaked in the demo see "Raymarching Rand Scalar".

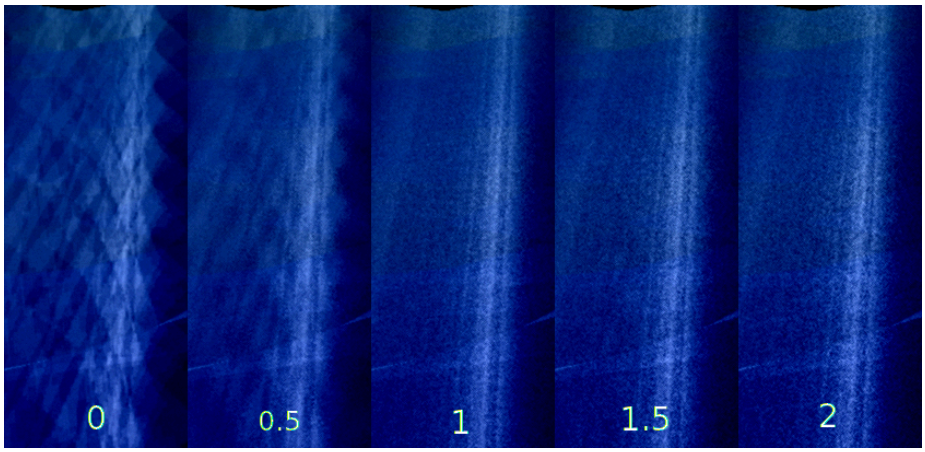


Figure 7: Shows the artifacts produced by the raymarching sampling, post-processed to clearly show the effect of dithering the raymarching. The numbers of each image is the multiplier to the raymarching dithering, which is a random number scaled between $-.5$ and $.5$ then multiplied with the distance between each sample

5 Post Processing

Figure 8: The depth fog soloed

I chose to implement depth fog since volumetric light makes most sense when fog is present, the fog contribution alone can be seen in figure 8. Once the fog

was implemented banding was very evident, but ultimately solved by dithering the position used to calculate depth from the camera. In figure 9 the banding artifact can be seen and the number represents that multiplier to the added noise, the value can be tweaked within the demo see "Position Rand Scalar" under Fog in the menu.

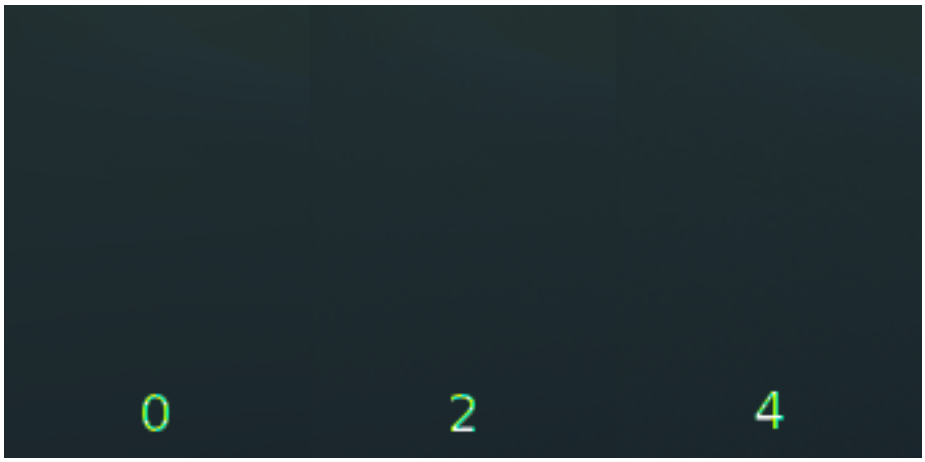


Figure 9: Position dithering of Depth Fog, the values represent the randomness multiplier, it can clearly be seen how the banding effect is less and less visible

Applying a blur effect was investigated to remove banding created by the volumetric light by ultimately decided against, because of the extensive blurring required to remove the banding. For that reason in the Debug menu a kernel can be found to apply effect with the color. In section 7 a better solution is presented.

6 Performance

On my laptop (specs below) the demo runs fairly steadily at 30ms delta time, which is not great but acceptable as a demo. The technique is definitely heavy to run, but i have already discussed various ways to improve performance and in next section Temporal Anti Aliasing(TAA) will quickly be discussed. Performance could be improved could especially be improved by proper light mesh influence calculations for spotlights which would allow for only sampling within the actual volume resulting in no wasted samples, currently the raymarching is initiated on the entire screen.

- CPU: i7-6700HQ @ 8x 3.5GHz
- GPU: GeForce GTX 960M
- RAM: 16Gb DDR4

7 Future Work

As Mentioned in the previous section performance might have been improved from having TAA, discussing Anti Aliasing as a performance increase is unusual in my experience, normally one would aim for better quality at the cost of performance. But I argue, based on Playdead's presentations (Pedersen, 2016) and (Gjoel & Svendsen, 2016), that it would make it possible to use fewer raymarching samples while keeping the quality. Mutlisample Anti Aliasing(MSAA) would improve the quality greatly but, if to be used with the volumetric light, decrease the performance dramatically simply because of the added sampling. I would have liked to implement TAA but did seem possible within the given time frame.

While the above AA technique would be great, changing the type of noise would improve the quality of dithering. While the currently added noise helps with the banding it creates it own noisy banding because of the poor quality of noise. Currently noise is generated by a function which gives poor results, preferably noise could be defined with a texture. This would enable me to apply Blue Noise which is great for dithering (Gjoel, 2014).

8 Acknowledgements

In order to create Deferred Rendering I used the following resource from LearnOpenGL (de Vries, 2015a), to create Light Influence Mesh and the stencil trick I used (Meiri, n.d.). To create the Volumetric Light technique I used the talk from Playdead (Gjoel & Svendsen, 2016) and for the dithering (Gjoel, 2014). To create shadow mapping I used another resource from LearnOpenGL (de Vries, 2015b).

Project was created on a Linux machine it has not been tested on any other OS's.

References

de Vries, J. (2015a). *Deferred shading.* Retrieved 22-05-2017, from https://learnopengl.com/#!Advanced-Lighting/Deferred-Shading

de Vries, J. (2015b). *Shadow mapping.* Retrieved 22-05-2017, from https://learnopengl.com/#!Advanced-Lighting/Shadows/Shadow-Mapping

Gjoel, M. (2014). *Banding in games.* Playdead. Retrieved 22-05-2017, from "http://loopit.dk/banding_in_games.pdf"

Gjoel, M., & Svendsen, M. (2016). *Low complexity, high fidelity.* Playdead. Retrieved 22-05-2017, from "http://loopit.dk/rendering_inside.pdf" ,"https://www.youtube.com/watch?v=RdN06E6Xn9E"

Meiri, E. (n.d.). *Deferred shading.* Retrieved 22-05-2017, from http://ogldev.atspace.co.uk/www/tutorial37/tutorial37.html

Pedersen, L. J. F. (2016). *Temporal reprojection anti-aliasing in inside.* Playdead. Retrieved 22-05-2017, from "http://gdcvault.com/play/1022970/Temporal-Reprojection-Anti-Aliasing-in"