Profiling of Real-Time Translucency

Sam Oates Teesside University School of Computing

Overview

I began by running performance analysis upon my fully functioning translucent deferred renderer. The program was run in the default window size and the camera animated about a path around the scene. All results where collected and an average of the times were output. The test ran for 60 seconds. The shadow frame buffer is using 4 full RGBA 32 bit floating point textures and the geometry buffer uses RGBA 16 bit floating point textures. The following data was accumulated:

Timer	Average Time (ms)
GBuffer Render	4.00658
Directional Light	0.193933
Spot Light(0) First Pass	3.09662
Spot Light(0) Second Pass	1.15645
Spot Light(1) First Pass	3.09395
Spot Light(1) Second Pass	1.11099
Spot Light(2) First Pass	1.6175
Spot Light(2) Second Pass	5.40877
Spot Light(3) First Pass	0.965333
Spot Light(3) Second Pass	5.43625
Spot Light(4) First Pass	1.40222
Spot Light(4) Second Pass	5.33348
Post Processing	0.713384

The application averaged at 29.8 frames per second. As can be seen from the table of results, the slowest part of my application is the second pass of the spot lights.

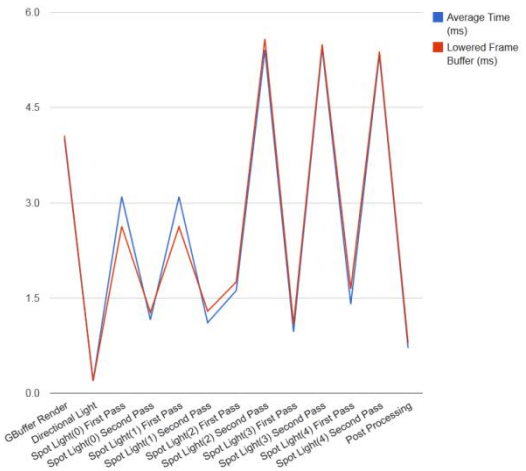
The second pass of the spot lights is where the translucency is calculated. However, translucency is only calculated on the spotlights 2, 3 and 4. Meaning the translucency itself is only costing around 2 milliseconds per spot light.

Texture Bandwidth

To test for issues within the texture bandwidth I changed my shadow map frame buffer from having 4 RGBA 32 bit floating point textures to having;

- One single channel 32 bit floating point texture
- Three RGBA 16 bit floating point textures

Timer	Average Time (ms)	Lowered Frame Buffer (ms)
GBuffer Render	4.00658	4.05736
Directional Light	0.193933	0.195054
Spot Light(0) First Pass	3.09662	2.62831
Spot Light(0) Second Pass	1.15645	1.27238
Spot Light(1) First Pass	3.09395	2.62253
Spot Light(1) Second Pass	1.11099	1.2924
Spot Light(2) First Pass	1.6175	1.75376
Spot Light(2) Second Pass	5.40877	5.58171
Spot Light(3) First Pass	0.965333	1.08855
Spot Light(3) Second Pass	5.43625	5.49658
Spot Light(4) First Pass	1.40222	1.64322
Spot Light(4) Second Pass	5.33348	5.38264
Post Processing	0.713384	0.790902

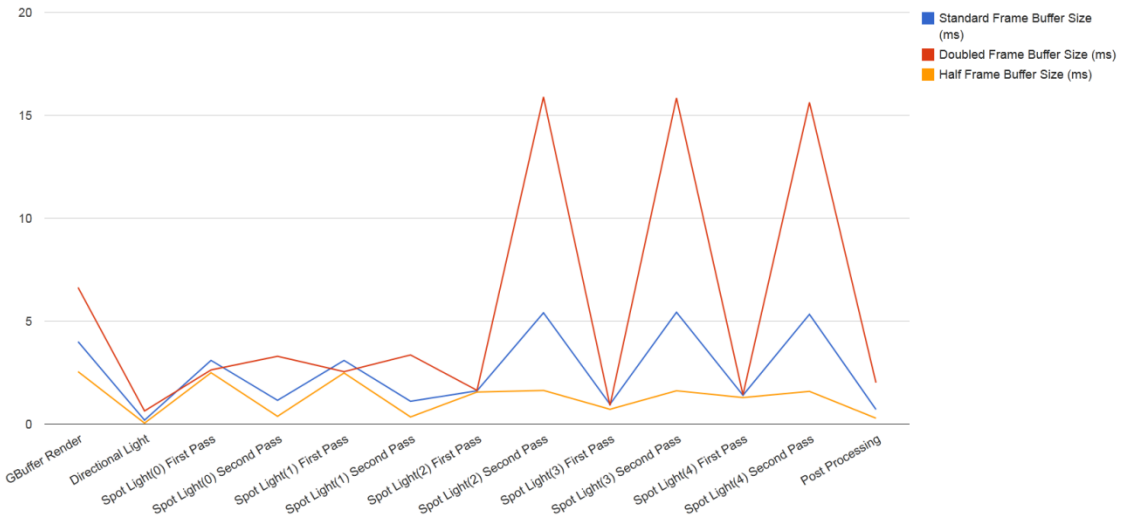


The results of the lowered texture format can be seen in the above table and visually compared in the adjacent line graph.

There is a marginal improvement using the lower quality textures as expected. However this improvement only results in an average frame rate improvement of two fps, resulting in an average fps of 32. Implying the larger buffer is having a negligible effect on performance.

Rasterization and Frame Buffer Bandwidth

Both the geometry buffer and light buffer sizes are equal to the window size. Below I test both halving and doubling the window size.



It is worth noting the fact that the spot lights which do not perform translucency, the ambiant light pass and the geomerty buffer render are all around the same value given the change in frame buffer size. However the lights that render translucency vary greatly whilst performing the lighting pass with different frame buffer sizes. This suggests that it is not the rasterization or frame buffer size which are having a detrimental effect on performance, instead the number of fragments to which lighting calculations are performed.

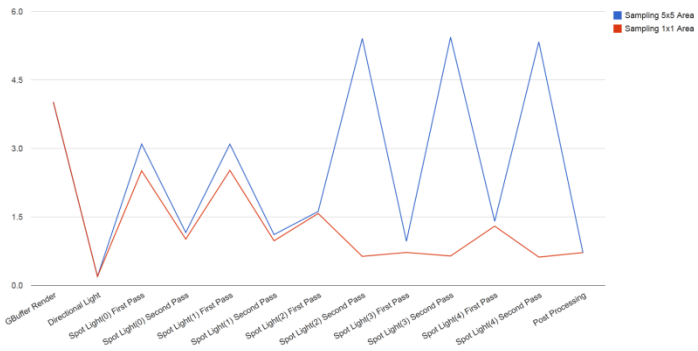
Fragment Operations

The second pass of the spot lights contains multiple sampling for the translucency calculations. Reducing the sampling from a 5x5 area to a 1x1 area had the effect shown to the right.

Reducing the sampling area makes the previously slowest section of the render (the second light pass of the spot lights which render translucency) one of the fastest.

This improves the performance of the application by around 200%, resulting in an average of 60 frames per second. This is most likely caused by the complexity of the transfer function required for translucency.

Timer	Sampling 5x5 Area	Sampling 1x1 Area
GBuffer Render	4.00658	4.02086
Directional Light	0.193933	0.193463
Spot Light(0) First Pass	3.09662	2.50939
Spot Light(0) Second Pass	1.15645	1.012
Spot Light(1) First Pass	3.09395	2.51871
Spot Light(1) Second Pass	1.11099	0.976009
Spot Light(2) First Pass	1.6175	1.5744
Spot Light(2) Second Pass	5.40877	0.634316
Spot Light(3) First Pass	0.965333	0.720829
Spot Light(3) Second Pass	5.43625	0.642681
Spot Light(4) First Pass	1.40222	1.29666
Spot Light(4) Second Pass	5.33348	0.618071
Post Processing	0.713384	0.717371



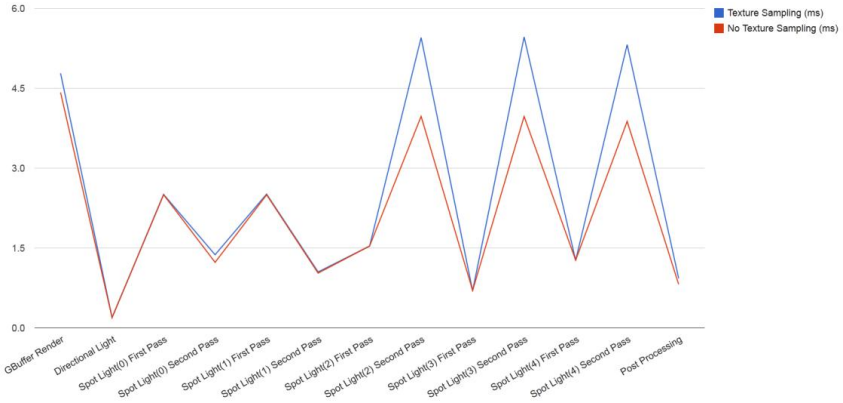
Texture Sampling within the Fragment Shader

Timer	Texture Sampling (ms)	No Texture Sampling (ms)
GBuffer Render	4.7815	4.42268
Directional Light	0.195271	0.195375
Spot Light(0) First Pass	2.50771	2.50221
Spot Light(0) Second Pass	1.37508	1.23267
Spot Light(1) First Pass	2.51384	2.50017
Spot Light(1) Second Pass	1.05195	1.03066
Spot Light(2) First Pass	1.53759	1.53882
Spot Light(2) Second Pass	5.45191	3.97309
Spot Light(3) First Pass	0.697576	0.703684
Spot Light(3) Second Pass	5.46359	3.97046
Spot Light(4) First Pass	1.27012	1.27304
Spot Light(4) Second Pass	5.31909	3.87837
Post Processing	0.93224	0.819925

To calculate translucency, not only do we need to sample from the geometry buffer but also from the shadow/irradiance buffer. The translucency samples from textures four times within the 5x5 sample area. Meaning 100 samples per fragment are need for translucency.

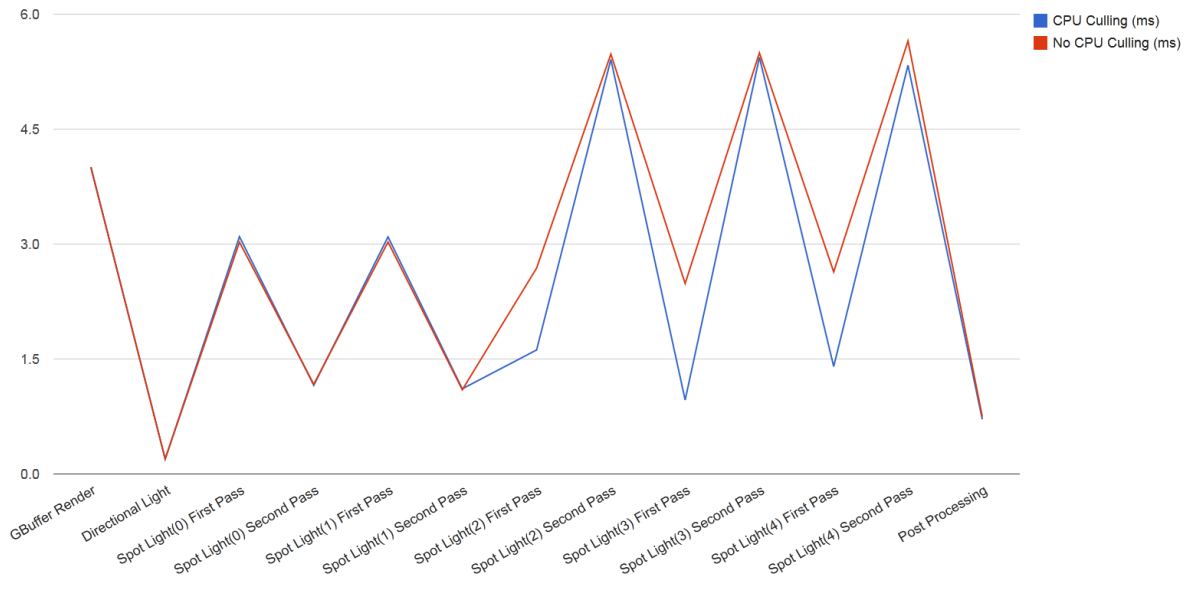
Removing the sampling from the translucency (meaning translucency no longer worked) had the following affect.

As can be seen there is a decrease in computational time for the translucency light pass. However the increase in performance is not as substantial as the fragment shader operations themselves.



Vertex Assembly

I implemented some crude CPU culling which meant the vertex intensive models within the scene are only drawn by lights which can see them. This results in a small performance gain whilst computing the shadow/irradiance maps for the three spot lights near the vertex intensive models. Giving a performance gain of 3 -4 frames per second. Suggesting the hardware is more than capable of working with the vertex intensity.



Conclusion

I believe the above implies that the current bottleneck within the applications render pipeline lies within the complexity of the fragment shaders (more so within the calculation of the sub surface scattering encountered whilst using the translucent shadow map algorithm). This could be resolved by pre-computing some values, such as the Schlick approximation along with this finding a way to sample less from textures would have a positive effect on performance. Sampling less could be done by packing surface normal and irradiance into one texture and packing light world position and the depth into one texture.