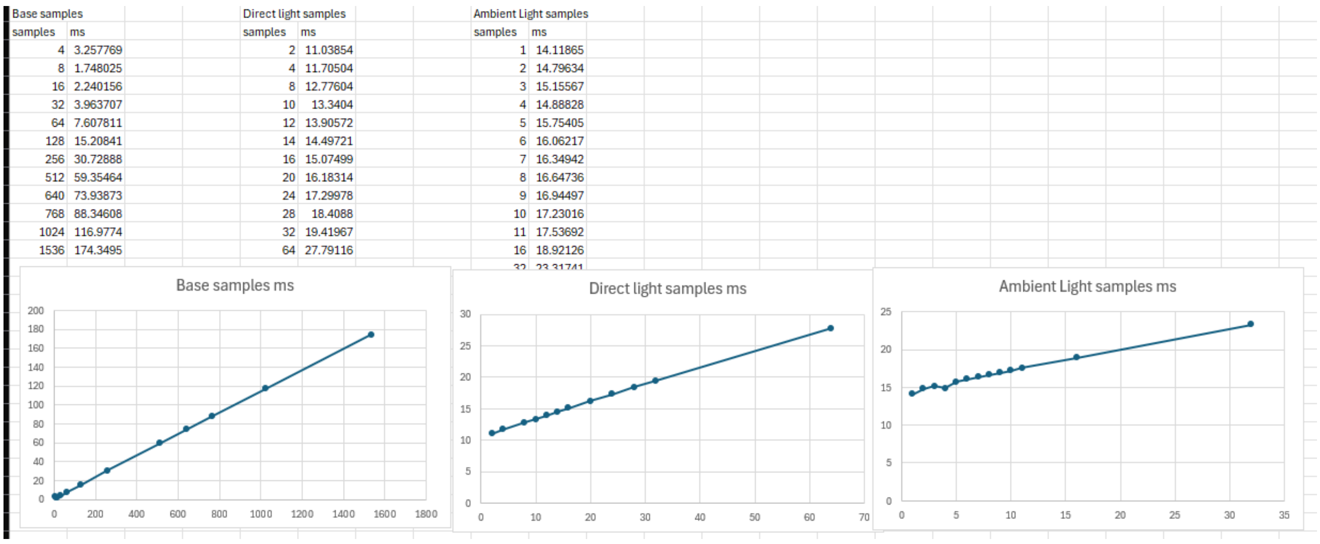
Scalability / Tests

I created a testing project that runs the project with different settings to quickly see how changes impact base performance and scalability based on settings like amount of samples. This measures performance by measuring frame time over X number of frames, to get a broad idea of performance quickly.

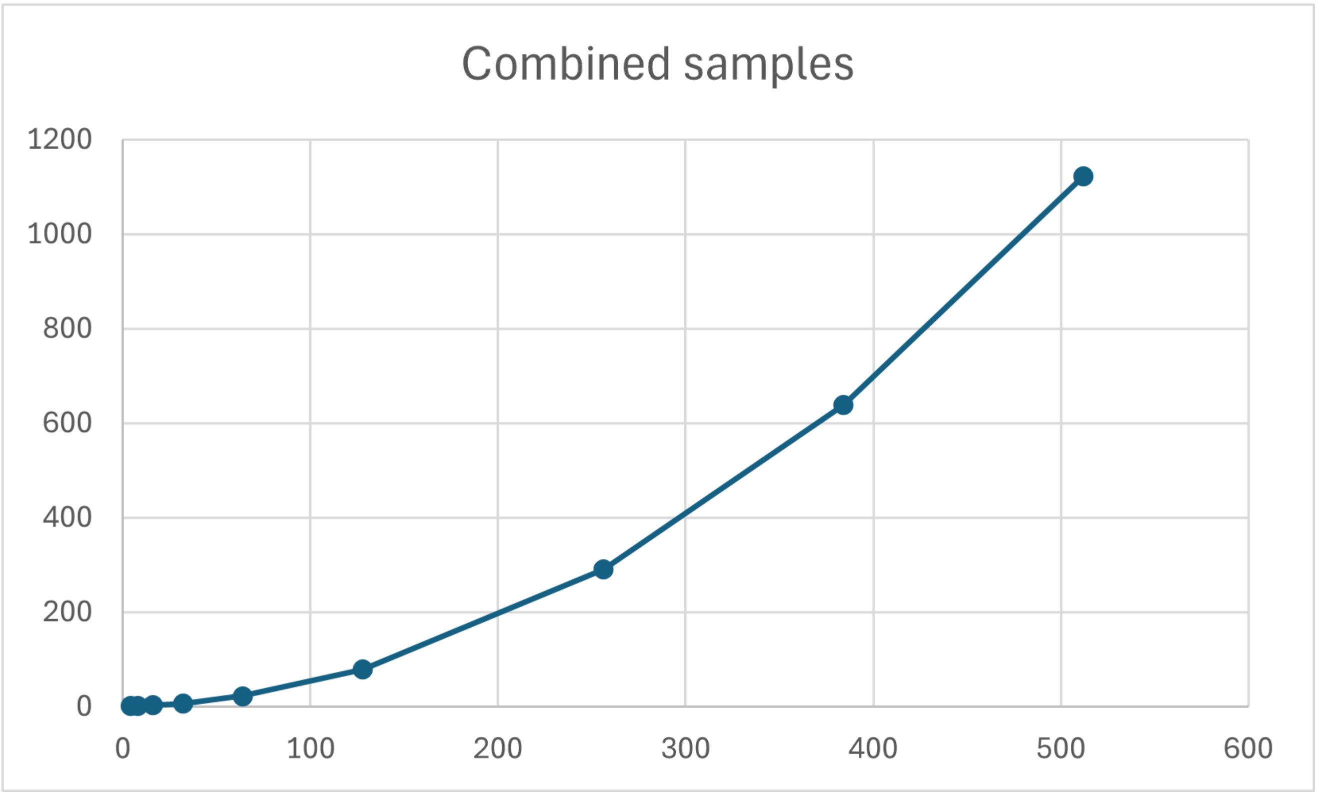
using this project i measured the scalability of differing sample counts before doing any big optimisations,



Base samples		Direct light samples		Ambient Light samples	
samples	ms	samples	ms	samples	ms
4	3.257769	2	11.03854	1	14.11865
8	1.748025	4	11.70504	2	14.79634
16	2.240156	8	12.77604	3	15.15567
32	3.963707	10	13.3404	4	14.88828
64	7.607811	12	13.90572	5	15.75405
128	15.20841	14	14.49721	6	16.06217
256	30.72888	16	15.07499	7	16.34942
512	59.35464	20	16.18314	8	16.64736
640	73.93873	24	17.29978	9	16.94497
768	88.34608	28	18.4088	10	17.23016
1024	116.9774	32	19.41967	11	17.53692
1536	174.3495	64	27.79116	16	18.92126
				32	23.31741

This shows that changing base samples, direct light samples and ambient samples all increase in cost linearly with their sample counts,

Since these samples combine in the march their cost affects each other massively as shown when increasing them together.



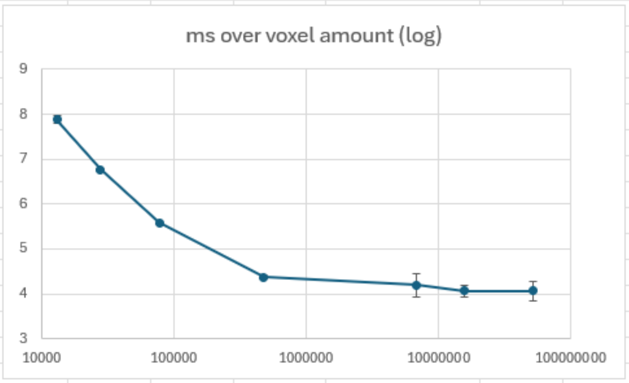
which scales significantly worse, due to for example direct light samples happening per base sample
Because of this improving the performance of any of these samples or making it so I have to take less of them has huge benefits on overall performance.

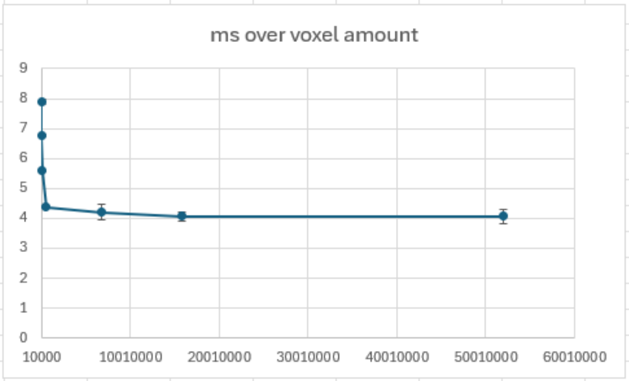
Resolution

Note: these results are from after optimizing texture sizes/precision already, and captured using nsight multi pass.

Testing differing cloud voxel resolutions, gave an interesting result

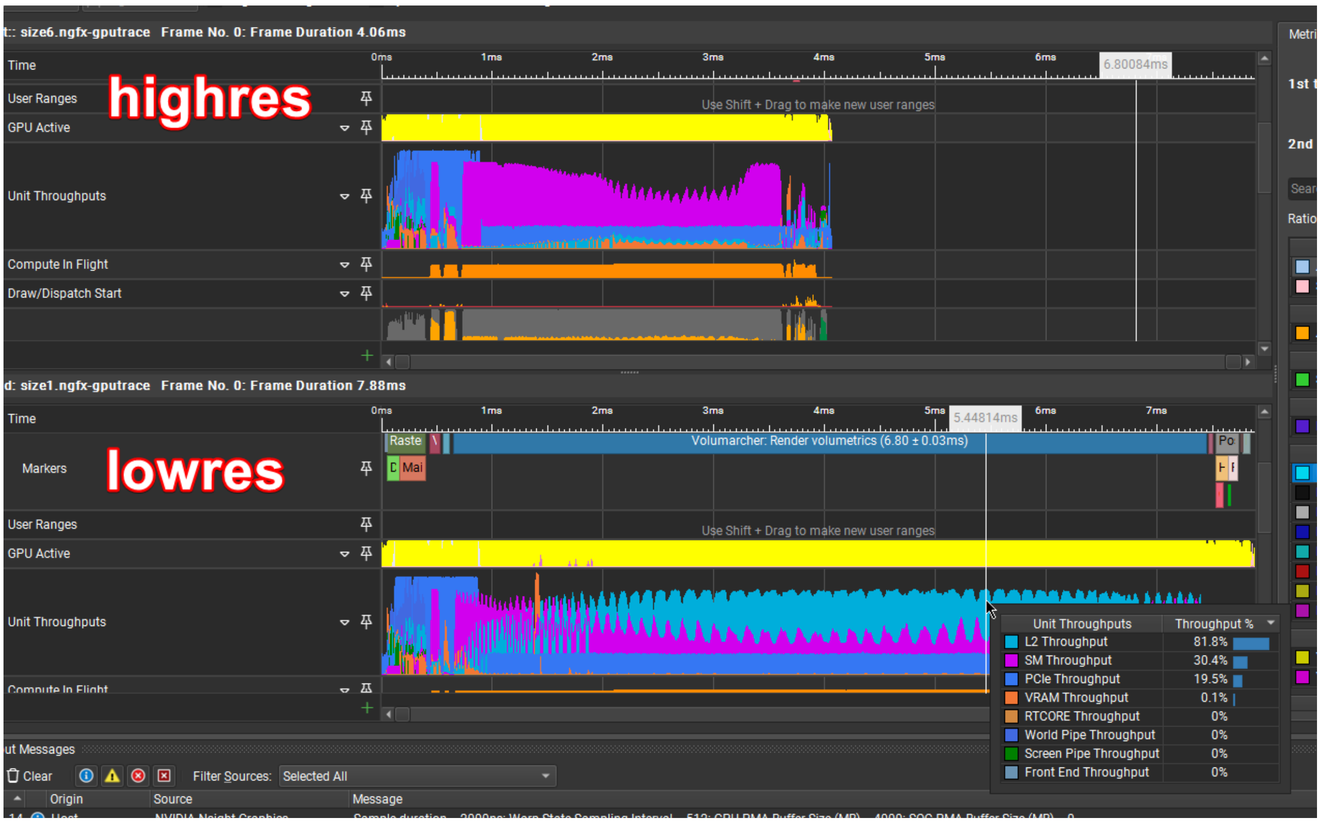
Voxels	ms	inaccuracy (ms)
13041	7.88	0.09
27750	6.77	0.03
77910	5.58	0.01
478016	4.37	0.05
6773310	4.19	0.26
15762800	4.06	0.14
51999296	4.06	0.23





Frame times are actually lower with higher resolution clouds
Checking with NSight this is caused because the caches are overused, since at low

resolution a lot of threads use the same voxels it gets bottlenecked by the cache,
While higher resolutions do not have to fight over that so wait less causing way higher SM usage


size6.ngfx-gputrace Frame No. 0: Frame Duration 4.06ms
highres
Time
User Ranges
GPU Active
Unit Throughputs
Compute In Flight
Draw/Dispatch Start
6.80084ms
Use Shift + Drag to make new user ranges
size1.ngfx-gputrace Frame No. 0: Frame Duration 7.88ms
lowres
Markers
Volumarcher: Render volumetrics (6.80 ± 0.03ms)
5.44814ms
ut Messages
Clear
Filter Sources: Selected All
Origin
Source
Message

Unit Throughputs	Throughput %
🟦 L2 Throughput	81.8%
🟪 SM Throughput	30.4%
🟦 PCIe Throughput	19.5%
🟧 VRAM Throughput	0.1%
🟧 RTCORE Throughput	0%
🟦 World Pipe Throughput	0%
🟩 Screen Pipe Throughput	0%
🟦 Front End Throughput	0%

Coverage

I also tested how cloud screen coverage affects performance, due to my masking based on bounds and density and the sphere tracing screen coverage has a big affect becoming way cheaper when looking at less clouds or away from the AABB,
Due to the big branching this causes it has a limit and does not scale linearly with amount rendered. This could be improved by doing indirect compute masking passes to reduce branching a lot.

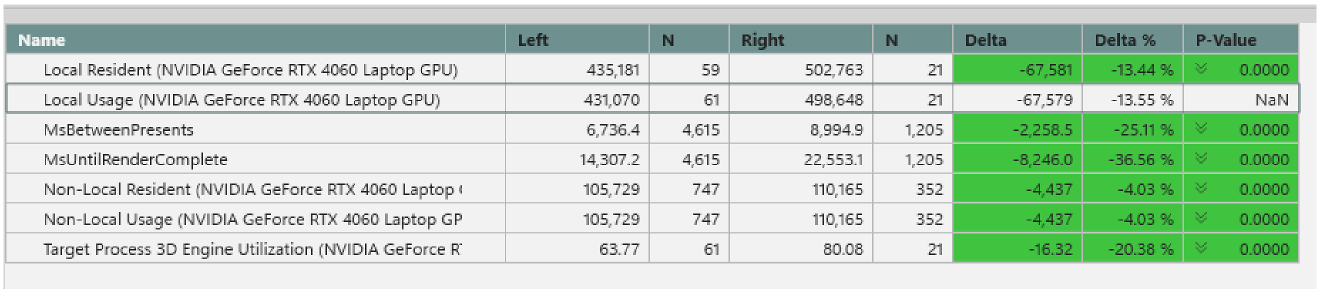
I didn't graph this since I don't have a solid metric for coverage.

Optimizations

GPU memory usage

After implementing adaptive sampling by sampling a distance field I noticed my performance going down a lot.
I had added the distance field as a extra channel to my cloud density texture, so this massively increased memory usage.
But this was not fully needed since it samples outside of a cloud only the distance data is used.
To improve this I split it into 2 textures instead by splitting them it avoids loading useless data.
I profiled the impact of this change in pix and noticed this gave a big improvement

Left Seperate textures (2x R32_Float) | Right Combined texture (1x R32B32_Float)
This means delta is change from new to old.



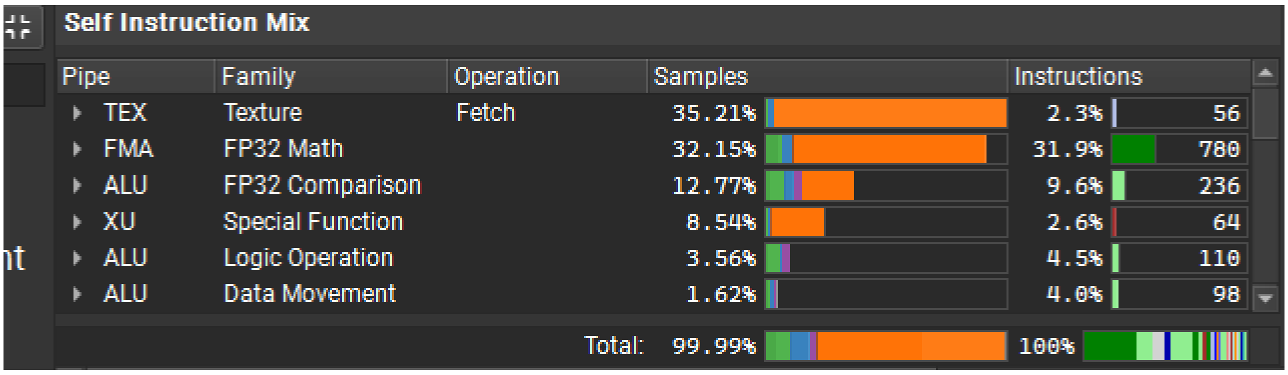
Name	Left	N	Right	N	Delta	Delta %	P-Value
Local Resident (NVIDIA GeForce RTX 4060 Laptop GPU)	435,181	59	502,763	21	-67,581	-13.44 %	0.0000
Local Usage (NVIDIA GeForce RTX 4060 Laptop GPU)	431,070	61	498,648	21	-67,579	-13.55 %	NaN
MsBetweenPresents	6,736.4	4,615	8,994.9	1,205	-2,258.5	-25.11 %	0.0000
MsUntilRenderComplete	14,307.2	4,615	22,553.1	1,205	-8,246.0	-36.56 %	0.0000
Non-Local Resident (NVIDIA GeForce RTX 4060 Laptop (	105,729	747	110,165	352	-4,437	-4.03 %	0.0000
Non-Local Usage (NVIDIA GeForce RTX 4060 Laptop GP	105,729	747	110,165	352	-4,437	-4.03 %	0.0000
Target Process 3D Engine Utilization (NVIDIA GeForce R	63.77	61	80.08	21	-16.32	-20.38 %	0.0000

This shows that there is a huge improvement in local memory resident and usage, which means that the samples are cache hits more often giving a 36% render time reduction.

Ambient sampling,

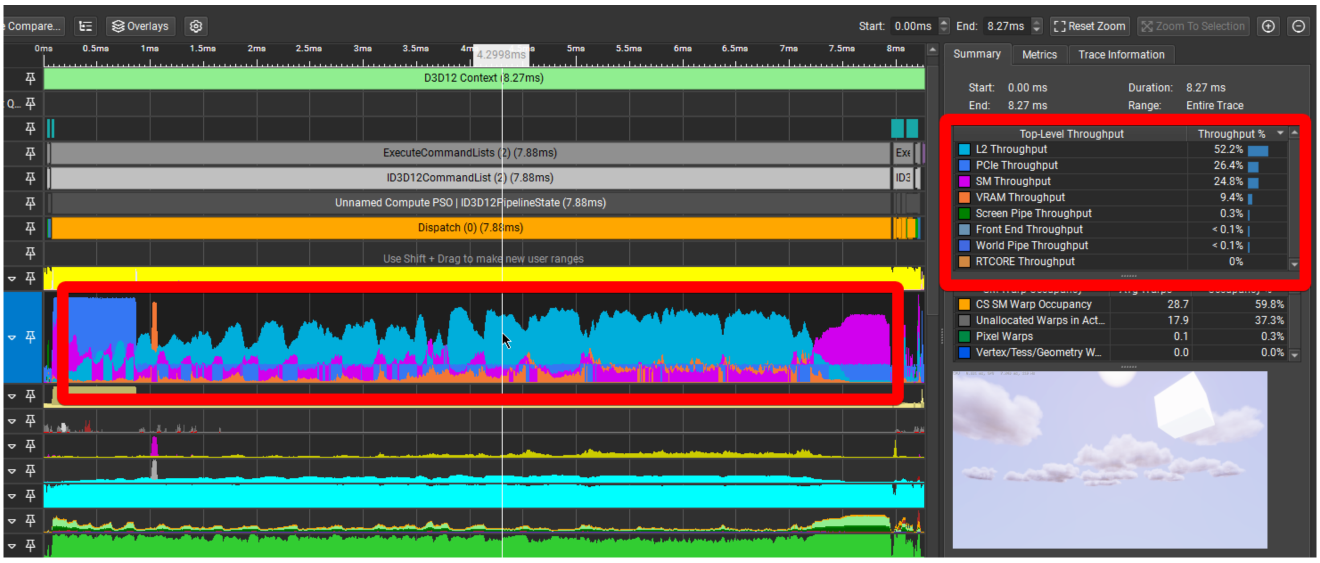
When profiling i noticed that my marcher was almost fully memory bound, which makes sense since it has to fetch the voxels per step, and per step in a cloud integrate densities into the light direction and upwards to get a approximation for ambient light, Which is a lot more texture
samples.

Nsight showing that there is a lot of memory throughput and texture fetches compared to the amount of SM throughput:



Self Instruction Mix

Pipe	Family	Operation	Samples	Instructions	
TEX	Texture	Fetch	35.21%	2.3%	56
FMA	FP32 Math		32.15%	31.9%	780
ALU	FP32 Comparison		12.77%	9.6%	236
XU	Special Function		8.54%	2.6%	64
ALU	Logic Operation		3.56%	4.5%	110
ALU	Data Movement		1.62%	4.0%	98
		Total:	99.99%	100%	


Compare...
Overlays
Start: 0.00ms
End: 8.27ms
Reset Zoom
Zoom To Selection
D3D12 Context (8.27ms)
ExecuteCommandLists (2) (7.88ms)
ID3D12CommandList (2) (7.88ms)
Unnamed Compute PSO | ID3D12PipelineState (7.88ms)
Dispatch (0) (7.88ms)
Use Shift + Drag to make new user ranges
Summary
Metrics
Trace Information

Start: 0.00 ms
End: 8.27 ms
Duration: 8.27 ms
Range: Entire Trace

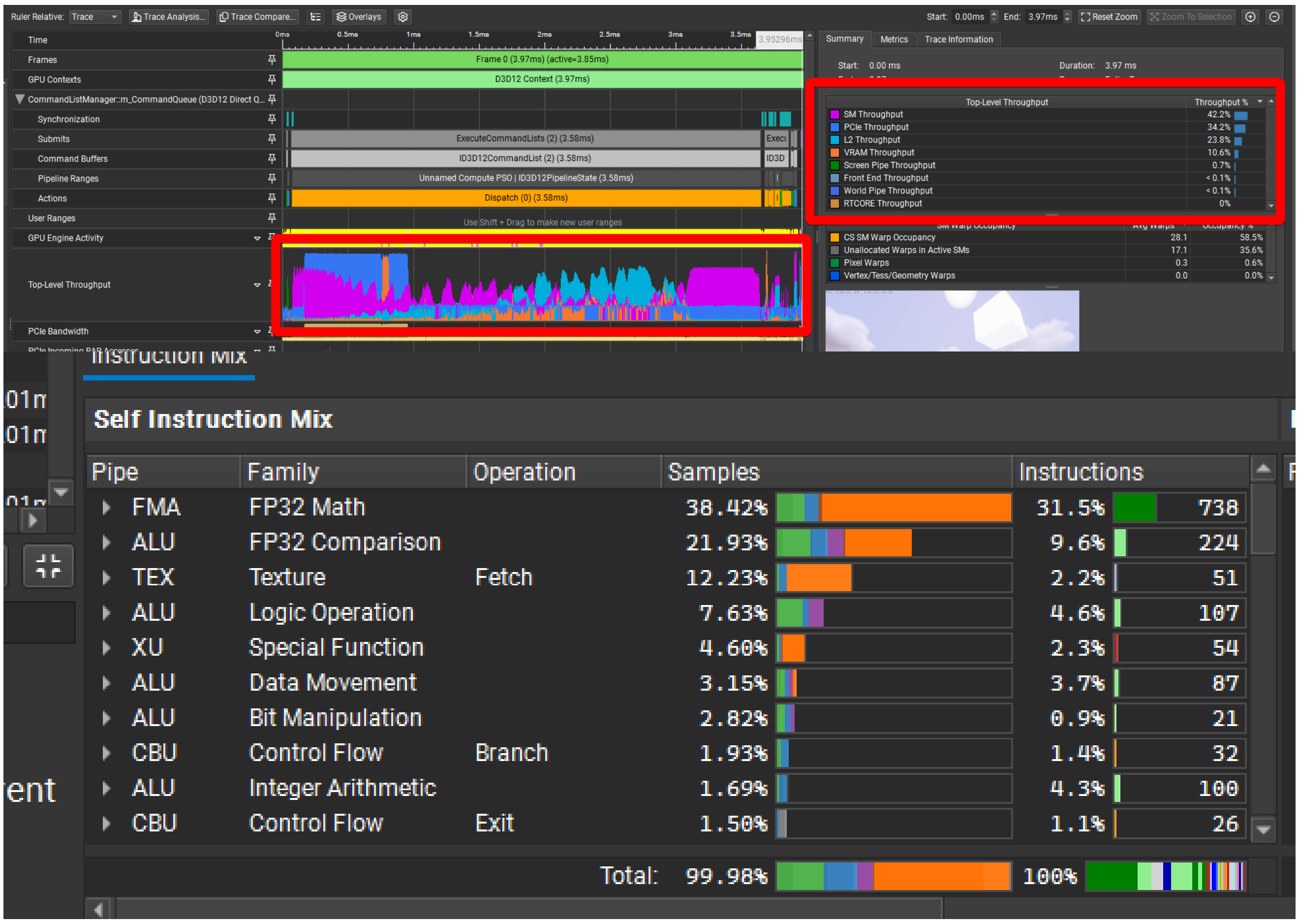
Top-Level Throughput	Throughput %
L2 Throughput	52.2%
PCIe Throughput	26.4%
SM Throughput	24.8%
VRAM Throughput	9.4%
Screen Pipe Throughput	0.3%
Front End Throughput	< 0.1%
World Pipe Throughput	< 0.1%
RTCORE Throughput	0%

SM Warp Occupancy	Avg Warps	Occupancy %
CS SM Warp Occupancy	28.7	59.8%
Unallocated Warps in Act...	17.9	37.3%
Pixel Warps	0.1	0.3%
Vertex/Tess/Geometry W...	0.0	0.0%

Running at 8.27ms in total

Since I assume clouds don't change and the ambient integration is always up, I can fully precalculate this causing it to go from taking SampleCount number of texture fetches per cloudsample, to taking only 1.

This massively reduced frame time and caused texture fetched and math time to be more balanced


Frame 0 (3.97ms) (active=3.85ms)
D3D12 Context (3.97ms)
ExecuteCommandLists (2) (3.58ms)
ID3D12CommandList (2) (3.58ms)
Unnamed Compute PSO | ID3D12PipelineState (3.58ms)
Dispatch (0) (3.58ms)
Top-Level Throughput
Start: 0.00ms End: 3.97ms
Summary
Metrics
Trace Information

Start: 0.00 ms Duration: 3.97 ms

Top-Level Throughput	Throughput %
SM Throughput	42.2%
PCIe Throughput	34.2%
L2 Throughput	23.8%
VRAM Throughput	10.6%
Screen Pipe Throughput	0.7%
Front End Throughput	< 0.1%
World Pipe Throughput	< 0.1%
RTCORE Throughput	0%

SM Warp Occupancy	Avg Warps	Occupancy %
CS SM Warp Occupancy	28.1	58.5%
Unallocated Warps in Active SMs	17.1	35.6%
Pixel Warps	0.3	0.6%
Vertex/Tess/Geometry Warps	0.0	0.0%

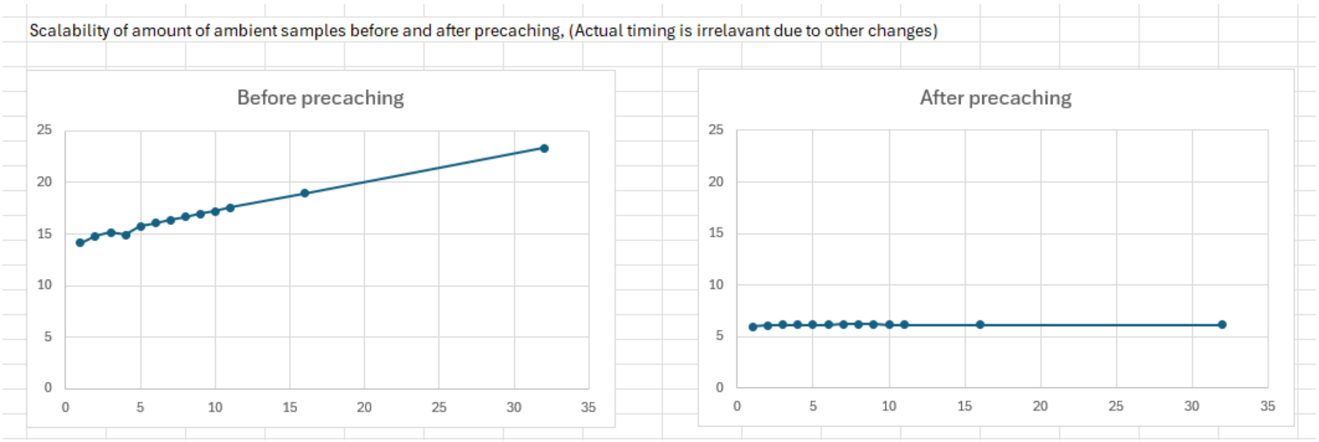
Instruction Mix

Self Instruction Mix

Pipe	Family	Operation	Samples	Instructions	
FMA	FP32 Math		38.42%	31.5%	738
ALU	FP32 Comparison		21.93%	9.6%	224
TEX	Texture	Fetch	12.23%	2.2%	51
ALU	Logic Operation		7.63%	4.6%	107
XU	Special Function		4.60%	2.3%	54
ALU	Data Movement		3.15%	3.7%	87
ALU	Bit Manipulation		2.82%	0.9%	21
CBU	Control Flow	Branch	1.93%	1.4%	32
ALU	Integer Arithmetic		1.69%	4.3%	100
CBU	Control Flow	Exit	1.50%	1.1%	26
		Total:	99.98%	100%	

Running at 3.97ms in total

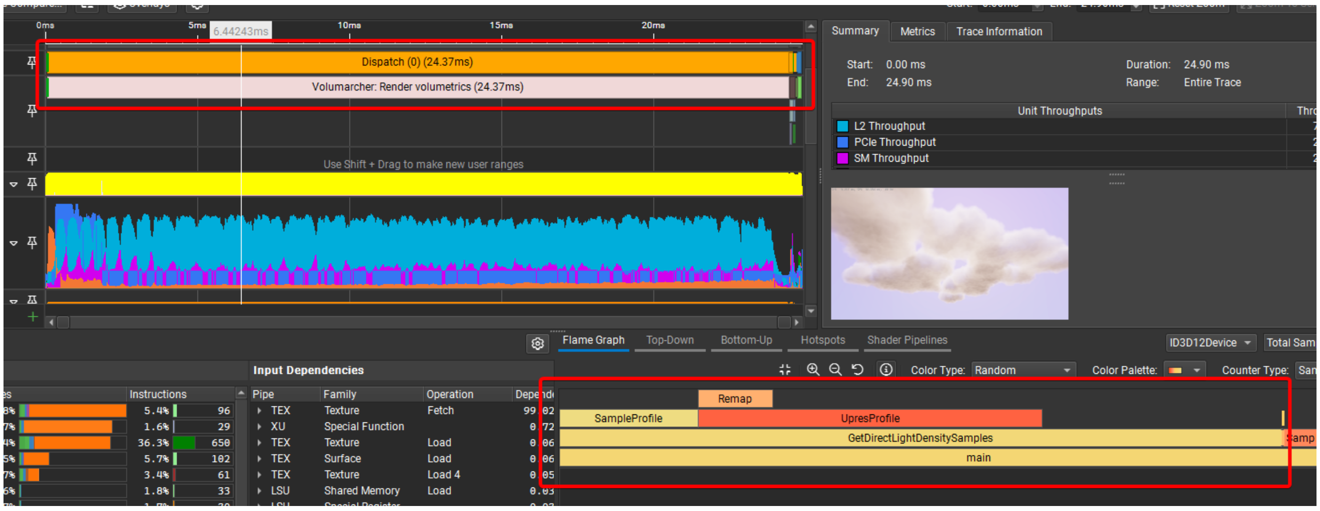
And this also causes the scalability to go from O(N) to O(1), for runtime sampling, since the computation gets moved to bake time.



Direct sampling,

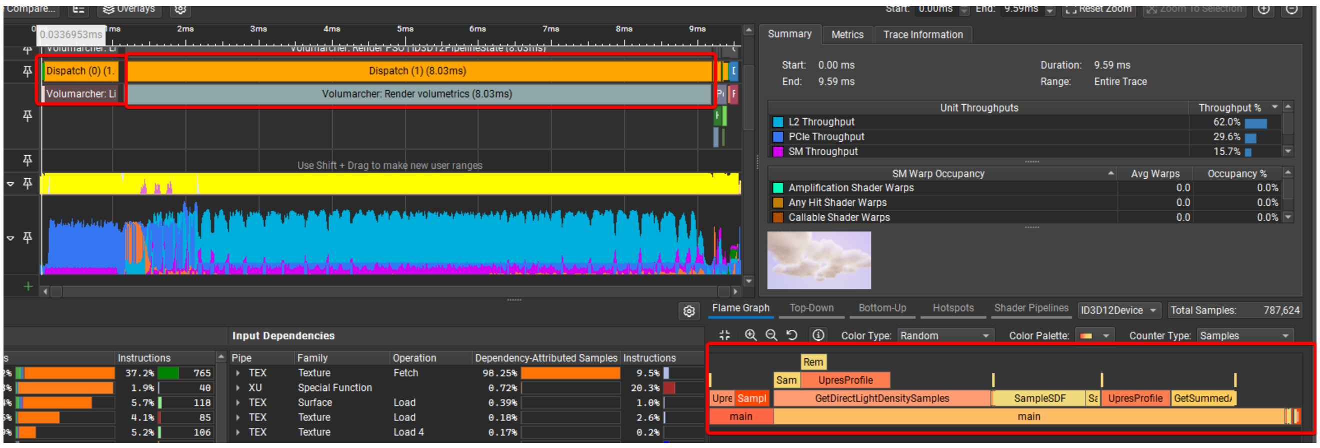
The same could be done for direct sampling somewhat,
By calculating the lighting per voxel in a different pass instead of per pixel in the main pass, it massively reduces the cost of calculating lighting. and it can also be cached to not calculate all the lighting if it doesn't change.
Since the cache is lower detail, I first do the first 2ish steps normally then sample the cache.

Without cache everything is done in 1 pass, DirectLightSamples is taking up almost all of the gpu time. (32 per sample(per pixel) light samples)



24,9ms total

When cached, all voxels are calculated in the first pass. This makes gpu time way more balanced . (2 per pixel(per sample) light samples and 30 cached samples)



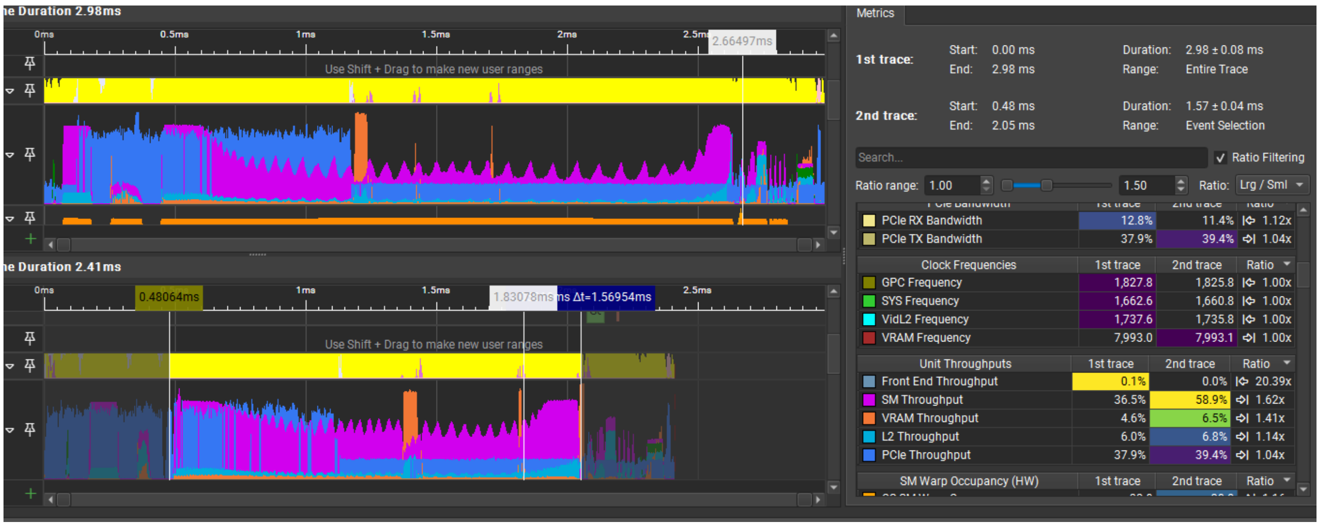
9,6ms total

This makes it so I have to do way less samples in total since I can calculate the cached lighting in chunks over multiple frames, and in most cases even without calculating in chunks, there are more pixels than there are cache voxels.

Precision

A lot of the (3d)textures I'm using also don't require that much precision, so storing them as 32 bit float textures is a waste. So instead I changed my textures to being 16 bit floats which improved memory throughput in my shaders a lot.

Here profiling before(top) and after(bottom) of changing the high frequency noise texture to 16bit 2 channels instead of 32 bit 4 channels.



1st trace:	Start: 0.00 ms	End: 2.98 ms	Duration: 2.98 ± 0.08 ms	Range: Entire Trace
2nd trace:	Start: 0.48 ms	End: 2.05 ms	Duration: 1.57 ± 0.04 ms	Range: Event Selection

✓ Ratio Filtering
Ratio range: 1.00 – 1.50 Ratio: Lrg / Sml

PCIe Bandwidth	1st trace	2nd trace	Ratio
PCIe RX Bandwidth	12.8%	11.4%	1.12x
PCIe TX Bandwidth	37.9%	39.4%	1.04x

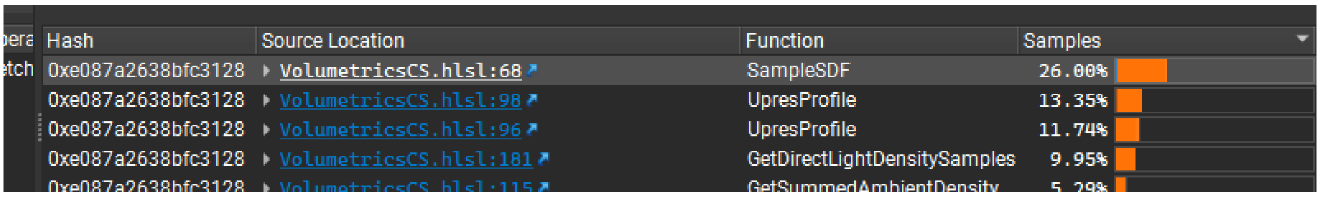
Clock Frequencies	1st trace	2nd trace	Ratio
GPC Frequency	1,827.8	1,825.8	1.00x
SYS Frequency	1,662.6	1,660.8	1.00x
VidL2 Frequency	1,737.6	1,735.8	1.00x
VRAM Frequency	7,993.0	7,993.1	1.00x

Unit Throughputs	1st trace	2nd trace	Ratio
Front End Throughput	0.1%	0.0%	20.39x
SM Throughput	36.5%	58.9%	1.62x
VRAM Throughput	4.6%	6.5%	1.41x
L2 Throughput	6.0%	6.8%	1.14x
PCIe Throughput	37.9%	39.4%	1.04x

SM Warp Occupancy (HW)	1st trace	2nd trace	Ratio

Block compression

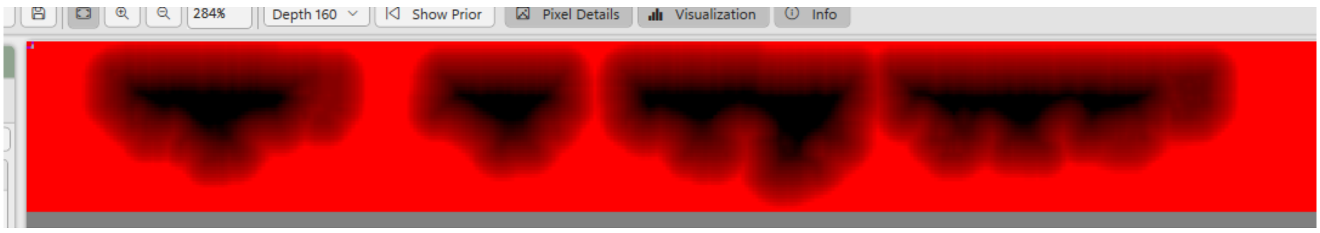
My biggest hotspot is fetching the SDF texture



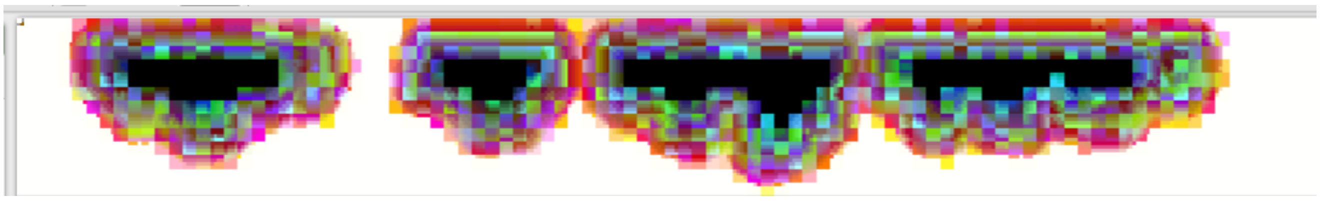
Hash	Source Location	Function	Samples
0xe087a2638bfc3128	VolumetricsCS.hlsl:68	SampleSDF	26.00%
0xe087a2638bfc3128	VolumetricsCS.hlsl:98	UpresProfile	13.35%
0xe087a2638bfc3128	VolumetricsCS.hlsl:96	UpresProfile	11.74%
0xe087a2638bfc3128	VolumetricsCS.hlsl:181	GetDirectLightDensitySamples	9.95%
0xe087a2638bfc3128	VolumetricsCS.hlsl:115	GetSummedAmbientDensity	5.29%

At GPC there was a talk by guerrilla about compressing the SDF for clouds and it giving them a 30% speedup.
As long as the precision loss always causes values lower it only causes it to have to take 1 extra sample sometimes,
if the precission results in higher values it creates artifacts, so they have a custom bc1 compressor to avoid that.
I implemented their custom BC1 compression in a compute shader making my texture 4x smaller, but even though my program is limited by texture fetches this gave no improvement for me, no matter the resolution of the clouds or how spaced they are. This is probably because my sdf texture was already small enough to fit into cache.

Old SDF Texture, 378x49x382 16Bits pixels



New BC1 Texture, 95x13x382 64bits per block (~4x smaller)



Future optimization opportunities.

A possible improvement would be to cache where/if rays hit in screen space, allowing to skip empty parts of the sky.

Another good improvement would probably be to split the renderer into more indirect passes, masking based on depth or bounds hit. Which avoids a huge uneven branch

Apart from these optimization I think most higher level optimizations have been done, the rest of future optimizations would have to focus on lower level improvements like reducing the amount of registers being used, which is currently the main bottleneck.