Memory optimization for high-density assets in Unreal Engine

Research into computer graphics optimization

Sorin Stoleru
Technological University of the Shannon: Midlands Midwest
Athlone, Co.Westmeath, Ireland
stolerusordev@gmail.com

Guilherme Gomes
Technological University of the Shannon: Midlands Midwest
Athlone, Co.Westmeath, Ireland
Guilherme.gomes@tus.ie

ABSTRACT

This study investigates memory optimization techniques for high-density assets in game development using Unreal Engine. The research focused on a project containing over 400 meshes with 4K textures. The methodology employed comprehensive performance testing of four configurations: a baseline ground truth scene, texture compression (mipmapping) with texture streaming, Level of Detail (LOD) generation with mesh streaming (excluding Nanite meshes), and a combination of both techniques. Performance metrics, including frames per second (FPS), frame time, and RAM usage, were captured using Nvidia NSight Systems over a 54 second benchmark repeated 10 times for each technique. The results demonstrated poor performance for the unoptimized ground truth and LODs alone, while texture compression showed good overall performance and reduced RAM usage. The combination of LODs and texture compression showed the most performance gains, creating a stable scenario with a significant improvement in all the metrics. Video game optimization is a key step in development, ensuring a smooth and engaging player experience [1]. This study highlights the critical importance of memory optimization techniques for managing high-density assets in Unreal Engine to achieve better performance without compromising visual quality.

1 Introduction

The rapid advancement in computer graphics has allowed the creation of increasingly complex and visually rich digital environments in modern video games. These games often feature 3D models with high polygon counts and very high-resolution textures, promising realistic visual fidelity. However, this technological progress presents significant challenges in terms of resource management and performance optimization. For instance, a large number of polygons places a heavy computational load on the Graphics Processing Unit (GPU) during rendering, directly impacting the frames per second (FPS). Similarly, the use of very high-resolution textures consumes substantial video memory (VRAM) and bandwidth, potentially causing performance bottlenecks and longer loading times for each frame. Video game optimization is the fine-tuning process that pushes the boundaries of technology to deliver the best possible performance across a myriad of devices [1]. Effectively managing high-density assets without compromising visual integrity and improving performance is a key concern for game developers. This study aims to provide insights into the effectiveness of texture compression (mipmapping) with texture streaming and Level of Detail (LOD) generation with mesh streaming, both individually and combined, for optimizing memory usage and performance in Unreal Engine when dealing with a large number of high-density assets. The research focuses on a "Old West" project using over 400 meshes with 4K textures that was provided by Epic Online Learning Team to the public, this specific project has been chosen to evaluate the impact of these optimization strategies on key performance indicators such as FPS, frame time, and RAM usage. Performance plays a crucial role in video games as it directly impacts player experience [2]. Due to the fact that the project uses 4K textures and the research is being done on a 1080p monitor this has affected the visuals of the scene greatly as higher resolution textures are rasterized and create artefacts in some scenarios. Due to this, the research doesn't look into the interpretation of visual quality using Frechet Inception Distance to assess any changes to visual quality.

2 Related work

While the sources online do not contain a dedicated section reviewing work directly in the area of assessing the effectiveness of texture compression or level of detail in the same specific context as this report, however, they do give special importance to these optimization techniques.

2.1 Seminal work

The use of Level of Detail (LOD) optimization is described as a "tried and true way of improving performance in a game" [2]. The fundamental idea involves using multiple versions of a 3D asset with varying levels of detail based on the distance from the camera. Simpler models are used for objects farther away to reduce the computational load [2]. This early approach aimed to address the performance limitations of rendering high-complexity geometry that was not visually significant due to distance.

Similarly, texture optimization, including techniques like texture compression and the use of mipmaps, represents foundational work in improving game performance and reducing memory usage. Texture mipmaps, which are precomputed, scaled-down versions of textures, allow the game to use lower resolution textures for objects farther away and reducing GPU workload. Compressing textures and reducing their resolution without significant loss of visual quality are crucial for minimizing memory requirements and loading times. These techniques were essential for enabling playable frame rates on older hardware with limited resources.

2.2 Current work

Current work in the domain continues to recognize LOD and texture optimization as essential for achieving optimal performance across a variety of modern devices, including PCs, consoles, and mobile platforms [1]. Game engines are continuously being optimized to efficiently handle LOD and texture streaming [3]. Modern game development practices emphasize the early integration of these optimization considerations throughout the development cycle.

Specifically regarding LOD, current implementations focus on dynamically adjusting the level of detail based on the camera's distance to objects, ensuring seamless transitions between LODs to maintain visual continuity. For texture optimization, current efforts include using efficient texture compression formats tailored to specific platforms [2]. The concept of prioritizing critical textures and loading non-essential textures at lower resolutions to save memory is also a current practice [1].

Performance profiling tools are also critical in current work to identify bottlenecks related to texture and model complexity, allowing developers to fine-tune their implementation of LOD and texture optimization [3]. Nowadays popular game engines have their own profilers that have improved significantly over the past few years but they are focused only on what is going on inside the engine. Some external applications like Nvidia NSight Systems in the other hand are reporting all device activity, which includes all CPU and GPU activity on the entire system, this ensures that there is full understanding when background processes impact game performance.

2.3 Problem to be addressed

The research was guided by three core questions.

First, what are the impacts of various memory optimization techniques on key performance metrics such as frames per second, frame time, and RAM usage?

Second, how does texture compression alone compare to implementing LOD for meshes?

Third, can a combined approach, integrating both texture compression and LOD generation, provide even better performance than a single technique by itself?

These questions guided the research to achieve better performance at minimal cost of visual fidelity.

3 System Design / Methodology

The research has begun by finding an appropriate project that visually is as similar to a modern game, for that Old West by Epic Online Learning Team has been chosen as it features 4K textures, Lumen support for Unreal engine 5.0 and later, each asset was created with realistic AAA-quality visuals in mind. This perfectly fits the description of a modern demanding game, in fact the original project was too demanding for the PC that would be benchmarking this project. The project contained volumetric cloud system that was slowing down the performance significantly, so a decision to remove this feature has been made as this feature is not in scope of this research.

3.1 Configuration

The research methodology involves testing four primary configurations within Unreal Engine (version 5.4.4): Ground Truth, Texture Compression, Level of Detail (LOD) and Texture Compression combined with LOD.

- Ground Truth Scene: This serves as the baseline, all LODs on static objects were forced to 0, meaning only the original meshes would be rendered at all time with mesh streaming turned off and each texture's mipmap setting has been set to NoMipmaps individually and texture streaming has been turned off. This has affected the visuals of the scene greatly as higher resolution textures on a 1080p monitor are rasterized and create artefacts in some scenarios.
- Texture Compression Scene: all textures changed to their default mip gen settings (Sharpen9) and compression settings on DXT1/5. It also has texture streaming active, as without it on the mips would impact the performance negatively by loading the entire mipmap for each texture into VRAM.
- LOD Scene: it involves changing the forced LOD to -1 so the LODs would work as they have been configured. Majority of the models already had premade meshes for each level of detail but they have not been configured at which screen size each level should change. For every model with polygon count larger than 1000 a screen distance has been evaluated for each level, this action has not been done to Nanite models.
- Combined Optimization Scene: for this scene simply the all modified files from the texture compression scene have been transferred into LOD scene and texture streaming activated.

3.2 Benchmark

Creating a benchmark is crucial for this research, it ensures consistency in each report gathering metrics, it should be based on real time instead of in game ticks as they can vary from time to time. For the benchmark, a camera movement sequence has been created where the camera would be looking at the closest possible point of interest throughout the entire scene. The 54 second sequence has created a balanced amount of time spent in each section of the scene for gathering needed metrics. The benchmark has a manual start on the press of a button and an automated shutdown when the camera sequence has ended, this ensures consistency in the metrics gathered as the metrics will only start to be collected when the sequence has begun and automatically stop when it is finished.

3.3 Profiler

There has been a choice to be made when choosing a profiler as there is plenty of variations. Unreal Engine provides it's own profiler (Unreal insights) which is a great for looking into the problems in the scene and adjusting appropriately, but not as much for gathering data over a period of time and representing overall metrics, as well as not taking into consideration anything that is outside the engine program. Nvidia NSight Systems records each processes that are in CPU and GPU and creates a very simple table for collecting overall metrics of a benchmark, it proven to be ideal for this kind of research on an Nvidia GPU RTX 3060 Ti 8GB. The profiler allows for profiling the application without opening the editor, which is perfect for keeping the system usage at its minimum and ensuring only the system processes, profiler and the game build are the only processes on the system that are active during the benchmark. The profiler has been configured to open the game and wait for the button input to start recording metrics for its report, the button has been set to the same one as the benchmark to remove any possibility in human error.

3.4 Running the benchmark

For each of these four configurations, a benchmark was conducted 10 times to minimize error in the gathered metrics. The process of running each benchmark is:

1. Restart PC
2. Wait for background processes to fully load (~5 min)
3. Launch profiler
4. Launch game from profiler
5. Wait for all assets to load completely (~3 min)
6. Run benchmark (press of a button)
7. Wait for benchmark (~54 sec)
8. Wait for profiler report (~10 min)
9. Ensure there is no artefacts
10. Record metrics into Excel sheet

This process has been repeated 40 times in total, 10 times for each configuration. For each step where waiting is necessary it was decided to allow extra time to ensure that each benchmark is run at their full capacity and there is nothing that would affect the results.

3.5 Data processing

After each benchmark, the relevant metrics were placed into an Excel sheet where the average would be calculated and is easier to see any trends in results.

The following metrics were captured using the profiler:

- Total amount of frames
- Average FPS
- Minimum and maximum milliseconds per frame (min ms, max ms)
- Average milliseconds per frame (avg ms)
- 99th percentile of the benchmark (99% ms)
- RAM usage

Total amount of frames represents how many frames have been generated during the benchmark process.

Average FPS represents how many frames in average have been generated in a time frame of a second, typically for games a number of 60 FPS is considered a good result.

Milliseconds per frame are a better identifier in improvement of performance than average FPS due to the way it is being perceived. 30 FPS is 33.334 ms per frame and an improvement in 3 ms will change FPS to 32.967, which is not as noticeable amount but on 120 FPS, that is 8.334 ms per frame an improvement in 3 ms, makes average FPS 187.5, and this is a difference of 67.5 FPS in comparison to the 2.967 FPS on poorer performance shows how a metric can impact the way developers perceive performance.

99th percentile represents the worst 1% of the performance, it is typically the stutters and an improvement in this can affect greatly on how the game is perceived.

Random Access Memory (RAM) and Video Random Access Memory (VRAM) usage shows how much is memory is being used on the short term memory of the system and the memory of the GPU.

The benchmark has been conducted on a personal computer with these specifications:

Intel core i7-8700K, 32GB DDR4 3200MHz,

Nvidia RTX 3060 Ti, Kingston UV500 M.2 120 GB

4 Results / Testing and Evaluation

The results for each configuration are surprising in their own ways.

Table 1: Ground Truth results

Name	Frames (total)	avg (ms)	min (ms)	max (ms)	FPS (avg)	99% (ms)
GT1	627	85.75	33.9	229.07	11.66	204.03
GT2	594	90.22	32.89	242.01	11.08	219.77
GT3	425	126.03	25.98	277.84	7.93	249.83
GT4	1218	43.9	15.82	178.84	22.78	116.34
GT5	425	126.04	18.16	322.47	7.93	230.02
GT6	425	126.83	22.12	310.94	7.88	275.24
GT7	494	108.78	22.53	260.41	9.19	223.39
GT8	614	87.52	22.25	239.98	11.43	207.92
GT9	647	82.87	17.52	223.81	12.07	205.45
GT10	427	126.17	23.05	324.8	7.93	300.87
	Frames	avg	min	max	FPS	99%
Average	589.6	100.411	23.422	261.017	10.988	223.286
Min	425	43.9	15.82	178.84	7.88	116.34
Max	1218	126.83	33.9	324.8	22.78	300.87

The ground truth scene, without any optimizations, performed very poorly. As shown in the Table 1, the average metrics across multiple runs were 589.6 total frames, an average frame time of 100.411 ms, an average FPS of 10.988 and a 99th percentile frame time of 223.286 ms, with a peak RAM usage of 13543.5 MB. On report GT4 we can see why it is crucial to have a wide spread of results as it performed exceptionally well but it does not match with any of other results even closely.

Table 2: Texture Compression results

Name	Frames (total)	avg (ms)	min (ms)	max (ms)	FPS (avg)	99% (ms)
TC1	1828	29.26	15.83	101.16	34.17	65.33
TC2	1532	34.89	16.26	102.87	28.66	79.45
TC3	1904	28.09	16.07	74.77	35.6	63.31
TC4	1694	31.58	15.84	94.1	31.66	77.88
TC5	1503	35.57	16.32	94.25	28.11	72.94
TC6	1786	29.96	16.17	96.8	33.38	74.91
TC7	1704	31.37	15.92	82.9	31.88	70.75
TC8	1663	32.16	16.07	156.58	31.09	66.5
TC9	1547	34.55	16.63	97.07	28.94	77.19
TC10	1625	32.89	15.67	101.58	30.4	82.41
	Frames	avg	min	max	FPS	99%
Average	1678.6	32.032	16.078	100.208	31.389	73.067
Min	1503	28.09	15.67	74.77	28.11	63.31
Max	1904	35.57	16.63	156.58	35.6	82.41

Implementing texture compression showed very good performance overall and a significant reduction in RAM usage compared to the ground truth. The Table 2 indicates an average of 1678.6 total frames, an average frame time of 32.032 ms, an average FPS of 31.389 and a 99th percentile frame time of 73.067 ms, with a RAM usage of 8960.2 MB. The table in the Table 2 highlights a 185.67% increase in average FPS, a 67.28% reduction in stutters and a 33.84% reduction in RAM usage compared to the ground truth.

Table 3: Level of Detail results

Name	Frames (total)	avg (ms)	min (ms)	max (ms)	FPS (avg)	99% (ms)
LOD1	667	80.62	24.17	240.91	12.4	208.39
LOD2	665	80.59	26.57	201.83	12.41	181.96
LOD3	417	128.48	41.04	282.55	7.78	254.45
LOD4	468	114.59	37.78	237.74	8.73	215.4
LOD5	376	142.73	24.44	296.73	7.01	284.69
LOD6	494	108.46	34.26	243.09	9.22	208.18
LOD7	598	89.63	26.09	261.64	11.16	210.96
LOD8	497	108.08	32.92	260.74	9.25	211.97
LOD9	372	145	37.92	335.66	6.89	322.54
LOD10	495	108.85	17.3	317.58	9.19	272.73
	Frames	avg	min	max	FPS	99%
Average	504.9	110.703	30.249	267.847	9.404	237.127
Min	372	80.59	17.3	201.83	6.89	181.96
Max	667	145	41.04	335.66	12.41	322.54

Interestingly, the use of LODs alone showed poor performance, excluding a reduction in RAM usage. Table 3 shows an average of 504.9 total frames, an average frame time of 110.703 ms, an average FPS of 9.404 and a 99th percentile frame time of 237.127 ms, with a RAM usage of 12614.3 MB. Table 3 indicates a 14.42% decrease in average FPS and a 6.86% reduction in RAM usage relative to the ground truth. The poor performance might be due to an increase in rendering complexity in an already poorly performing scenario. It is difficult to record the average amount of polygons drawn in the scene but for comparison in a specific place on the map ground truth is drawing ~4,500,000 polygons where in the same place with LODs there is ~2,000,000 polygons drawn.

Table 4: Combined optimization results

Name	Frames (total)	avg (ms)	min (ms)	max (ms)	FPS (avg)	99% (ms)
All1	1824	29.31	15.95	91.69	34.12	67.72
All2	1690	31.64	15.99	96.05	31.61	63.28
All3	1859	28.76	16.52	88.5	34.77	52.07
All4	1865	28.68	15.63	87.76	34.87	59.35
All5	1947	27.47	16.17	78.24	36.4	54.61
All6	1735	30.83	16.25	103.67	32.44	80.82
All7	1906	28.05	15.72	93.31	35.65	69.17
All8	1944	27.52	16.26	126.72	36.34	66
All9	1829	29.25	16.42	72.99	34.19	55.08
All10	1677	31.88	15.74	96.85	31.37	66.15
	Frames	avg	min	max	FPS	99%
Average	1827.6	29.339	16.065	93.578	34.176	63.425
Min	1677	27.47	15.63	72.99	31.37	52.07
Max	1947	31.88	16.52	126.72	36.4	80.82

Integrating both LODs and texture compression demonstrated the advantages of LODs when video memory is not overwhelmed, resulting in a larger performance boost than texture compression alone. Table 4 shows an average of 1827.6 total frames, an average frame time of 29.339 ms, an average FPS of 34.176 and a 99th percentile frame time of 63.425 ms, with a RAM usage of 7998.4 MB . As presented in the Table 4, this combined approach created a 211.03% improvement in average FPS, a 71.59% reduction in the 99th percentile frame time indicating reduced stutters and a 40.94% reduction in RAM usage compared to the ground truth.

It is important to consider frame times because average FPS can sometimes obscure performance issues like stuttering. A lower average frame time and a lower 99th percentile frame time indicate a smoother and more consistent experience.

5 Conclusions / Recommendations

Table 5: Texture compression evaluation

Name	GT	TC	% Difference
Frames (total)	589.6	1678	184.60%
avg (ms)	100.411	32.032	-68.10%
min (ms)	23.422	16.078	-31.36%
max (ms)	261.017	100.208	-61.61%
FPS (avg)	10.988	31.389	185.67%
99% (ms)	233.286	73.067	-67.28%
RAM (MB)	13543.5	8960.2	-33.84%

Table 6: Level of Detail evaluation

Name	GT	LOD	% Difference
Frames (total)	589.6	504.9	-14.37%
avg (ms)	100.411	110.703	10.25%
min (ms)	23.422	30.249	29.15%
max (ms)	261.017	267.847	2.62%
FPS (avg)	10.988	9.404	-14.42%
99% (ms)	233.286	237.127	6.20%
RAM (MB)	13543.5	12614.3	-6.86%

Table 7: Combined Optimization evaluation

Name	GT	All	% Difference
Frames (total)	589.6	1827	209.87%
avg (ms)	100.411	29.339	-70.78%
min (ms)	23.422	16.065	-31.41%
max (ms)	261.017	93.578	-64.15%
FPS (avg)	10.988	34.176	211.03%
99% (ms)	233.286	63.425	-71.59%
RAM (MB)	13543.5	7998.4	-40.94%

The unoptimized Ground Truth scene has shown poor performance, characterized by low average FPS and high frame times, as well as significant RAM usage. This represents the necessity of employing optimization techniques when dealing with projects containing a large number of high-detail assets.

Texture compression (mipmapping with texture streaming) proved to be a highly effective optimization strategy. It resulted in a significant 185.67% increase in average FPS, a 67.28% reduction in the 99th percentile frame time (stutters), and a 33.84% reduction in RAM usage compared to the Ground Truth. This demonstrates the crucial role of texture optimization in improving performance and reducing memory footprint.

The implementation of LODs alone yielded surprisingly poor performance, with a 14.42% decrease in average FPS despite a 6.86% reduction in RAM usage. This suggests that in an already unoptimized scenario with high texture memory pressure, the benefits of reduced polygon count from LODs might be offset by other factors, possibly related to rendering overhead or mesh streaming inefficiencies. This highlights that LOD as a standalone optimization might not always guarantee performance improvements.

The combination of texture compression and LODs demonstrated the most significant performance gains, achieving a 211.03% improvement in average FPS, a 71.59% reduction in the 99th percentile frame time, and a 40.94% decrease in RAM usage compared to the Ground Truth. This indicates that these two optimization techniques are most effective when used together, where reduced texture memory pressure allows the benefits of LODs to be fully realized. The substantial reduction in the 99th percentile frame time in this combined configuration is particularly important for ensuring a smoother and more engaging player experience by minimizing stutters.

In conclusion, this study strongly supports the critical importance of memory optimization techniques, particularly texture compression and its effect when combined with LODs, for effectively managing high-density assets in Unreal Engine and achieving better performance without compromising visual quality. The findings in this research highlight that game developers should prioritize these optimization strategies early in the development cycle to ensure a smooth and engaging player experience.

REFERENCES

[1] polydin, "The Essentials of Video Game Optimization | Maximizing Performance," Polydin, 2023.

[2] A. Edesberg, " Optimizing your 3D Models for Better Video Game Performance," Sloyd, 2024.

[3] J. R. Lowrey, " Basic Game Optimization Techniques," Jarlowrey, 2017.

[4] S. Wasson, "Inside the second: a new look at game benchmarking," Techreport, 2011.

[5] J. Roach, " These are the rules I live by when optimizing PC game performance," Digitaltrends, 2023.

[6] S. D. ,. S. M. L. ,. a. D. P.-L. Raluca D. Gaina, " Rolling Horizon Evolutionary Algorithms for," IEEE, 2022.