Enhancing GPU Rendering Efficiency through Scene Optimizations: A Case Study using Octane for Cinema 4D

1. Examiner: Rico Dober
2. Examiner: Jennifer Meier

Bachelor Thesis
Lucas de Melo Bernardino
Submitted to the Department of Media Production
University of Applied Sciences Ostwestfalen-Lippe, 32756 Detmold
January, 2024

Layout by:
Sergej Gavrilov

Proofread:
Katharina Cardoso Diefenbach
Kliment Gavrilov
Sergej Gavrilov

Thanks to:
Bruno Flávio Carneiro Malaco & Safari Post-Production for funding the assets used on this paper.

Abstract

This thesis examines the impacts of scene and rendering optimizations techniques aimed for GPU rendering within the realm of animated scene for offline rendering. Focusing on the unique challenges posed by dynamic animations, the research explores how these rendering approaches impact rendering speed, memory efficiency, and the attainment of lifelike visual quality. Using Octane render as ground for analysis, this study aims to uncover how scene optimizations intersect with the demands of animated content. Through meticulous comparison, this thesis endeavours to provide insights that empower rendering practitioners and animators to go through optimization strategies, bridging the gap between efficient resource allocation and the pursuit of captivating visual.

Keywords: GPU Rendering, Scene Optimization, 3D Rendering, Cinema 4D, Octane Render

Table of Contents

List of Tables

List of Figures

Enhancing GPU Rendering Efficiency through Scene Optimizations: A Case Study using Octane for Cinema 4D

1. Introduction

The realm of computer-generated imagery (CGI) is characterized by an unrelenting pursuit of realism and visual fidelity, with artists and content creators continuously pushing the boundaries of what is achievable in the digital domain. While the limits of creative potential continue to expand, there is a simultaneous contraction in production timelines. Due to those dynamics, GPU rendering (the process of rendering with graphic cards) has emerged as an efficient and cost-effective approach to achieving high-quality visuals in real-time and offline scenarios, speeding up the rendering process and, in return, providing the artists with more time to work on the project.

With the advent of powerful Graphics Processing Units (GPUs) and sophisticated rendering engines, artists and designers have been empowered with remarkable rendering speed and stunning visual quality through GPU renderers. One such rendering engine, Octane for Cinema 4D, stands out as a pioneering tool in this domain, offering real-time, physically-based rendering that has reshaped the landscape of 3D animation and visualization.

However, GPU rendering comes with its limitations: in the rendering process, GPUs are constrained by the available graphic memory, the VRAM (Video Random Access Memory), and everything that is to be rendered in a scene is solely loaded on the graphic card. Thus, efficient utilization of GPU resources has become of paramount concern.

The fundamental challenge in GPU rendering lies in the computational intensity required to simulate and render complex scenes, which often involve highly detailed 3D models, realistic materials consisting of high-resolution textures and complex shaders, multiple light sources, volumetrics, and animations, pushing hardware to its limits. To address this challenge, scene optimizations are a crucial aspect of the rendering pipeline and encompass a wide array of techniques aimed at smoothing the rendering process, reducing computational usage, improving overall efficiency when GPU rendering, and taking advantage of the speed offered by this technique.

This thesis embarks on a comprehensive exploration of the practical analysis of scene optimizations for offline rendering scenarios, presenting their profound impacts on GPU rendering while using the render engine OctaneRender® for Cinema 4D (C4D) as reference point. Octane‘s reputation as a high-performance offline renderer, combined with its integration into the Cinema 4D software suite, positions it as an ideal candidate for this study.

In the following chapters, information on the latest advancements in scene optimization techniques is provided, delving into the technical intricacies of offline GPU

rendering with Octane for C4D, and case studies are presented to illustrate the real-world impact of these optimization strategies. This study takes into consideration the pros and cons of GPU rendering, offering insights into maximizing the potential of the hardware available in the artist‘s hands. Furthermore, the process of constructing a functional frustum culling system for Octane scatter with native C4D tools is meticulously described.

By the end of this research, readers will possess the knowledge to excel their skills in offline rendering scenarios and understand how to maximize the use of the hardware at their disposal, showcasing best practices in different aspects of scene optimization, and making informed decisions when creating visually stunning works in offline GPU rendering scenarios.

2. Literature Review

Despite all the advantages of GPU rendering, knowing how to handle its limitations is of the utmost importance and learning how to take advantage of the available hardware is crucial. Regardless of the abundance of videos on rendering optimizations, there is a shortage of content on practical scene analyses comparing the resource usage of specific techniques used in 3D scenes. However, in a time of fast development of GPU technology, GPU rendering is becoming increasingly common in the market and it is important for artists to have such content available.

Scott Benson (2023), a C4D artist, has been developing highly explanatory guides for Octane C4D, sharing them on his Behance page. The ongoing series Resource Management presents detailed information about how Octane for C4D works when rendering a scene; furthermore, it explains which resources are tracked and how this process must be done to acquire an accurate result. This series is being featured as an official guide on OTOY‘s website, the developers of OctaneRender.

> *The goal of this series is to explore what C4D and Octane are doing under the hood, and how to tune our system resources and habits to make our workflow as zen and frustration-free as possible. (Benson, 2023, part 1)*

Of equal importance, the Raphael Rau (aka Silverwing) Youtube channel also provides much information on rendering techniques and best practices when working with Octane and C4D, many of which are unconsciously applied on workflow used for this paper.

Florean Renaux (aka Florenaux) Youtube channel was a major reference for environment concept art and on my personal workflow with Octane’s scatter systems applied in the produced scenes for this paper. Furthermore, Renaux provides a free asset of high-resolution water heightmaps that were used as displacement textures on the water present on the reference scenes of this paper.

The official documentation of Octane Render and Cinema 4D is also invaluable for this paper, bringing core information about both software, such as explanations of expression types and information about each tool available and how to use those.

3. Methodology

The methodology section of this thesis outlines the systematic approach used to investigate how each of the presented optimization techniques impacts GPU usage and the final visual outcome within a quantitative research framework. This section delineates the hardware and settings used, research design, data collection methods, and data analysis techniques employed to address the research questions and objectives.

Every scene done for this paper was rendered in 1920x1080, 24 frames per second (FPS) and using path trace kernel from Octane Render. The following hardware settings were used for the tests:

- Intel i7-6800K CPU
- 48GB RAM
- RTX 4090 - 24GB VRAM
- Windows 10
- Cinema 4D 2024
- Octane Render 2023.1

To have trustworthy parameters for analysis of hardware usage, C4D logs, Octane Render logs, and Octane Device Setting were of prime importance. Those tools provided solid data to compare the results of each presented situation. To collect solid data, equal performance across the different renders, and optimal computer performance, the following steps were followed before starting rendering:

1. Restart the computer.
2. Close all non-necessary software.
3. Terminate every non-essential process in Windows Task Manager.
4. Not use the computer for any task during the rendering process.

The software MSI Afterburner was used to acquire a real-time analysis of the hardware usage, helping to comprehend how the usage of each technique impacted the hardware, but it was not used as a data collection tool for this paper. Understanding how each optimization affects resources usage is fundamental when managing resources in a 3D scene; therefore, presenting the necessary tools to equip the readers with tools for an in-depth and trustworthy analysis of their 3D scenes was considered when writing this paper.

Every scene was modeled in C4D and rendered with Octane Render, being saved in EXR (Octane) format with DWAB compression and compression level of 45. When the resolution of textures had to be lowered, the bulk resize function of Adobe Bridge 2024 was used, and every resized texture was renamed with the Bulk Rename Utility for Windows.

The development of the frustum culling was documented in every step for this paper, presenting the pros and cons of both developed versions while analyzing its performance and effectiveness.

For this quantitative study, the crucial analysis criteria for optimizations were VRAM usage, render time, total render time, and RAM usage. Parameters such as visible triangles or displayed meshes were mentioned but not used as decisive factors. When analyzing the hardware usage, the data collected from the render logs was transcribed into charts to present a graphical comparison of the influential factors for each optimization. The original logs were also attached in the Appendix section of the paper, presenting all the information acquired from the software. As for the visual analysis, visual congruency was used as the main criteria.

4. Results

4.1 Research Objectives

For time-sensitive productions, a balance between production and rendering time is necessary. It is crucial to be aware of the most effective optimizations for each situation and applying those since the beginning of the project ensures optimal system performance, mitigating the risks of software crashes and potential bottlenecks.

A project that runs smoothly on a system empowers the artist to work more efficiently, providing more time for refinements, thereby opening possibilities for more creativity and higher-quality results.

GPU rendering has its limitations, of which the most important is making large scenes with large numbers of assets, complex shaders, high-resolution textures, and multiple light sources fit in the VRAM (Video Random Access Memory) available in the Graphic Card. On the other side, the biggest advantage of rendering on a GPU is its speed, so it is of major importance to take advantage of it.

Render engines have settings that enable speed improvements at cost of VRAM; therefore, optimizing the memory usage allows the artist to benefit from those features. Knowing how each task impacts the hardware can bring different perspectives to the artist while developing something aimed at being rendered in GPU.

This section provides a brief introduction to GPU technology and techniques, laying the foundation for subsequent discussions on advanced resource management in a 3D scene. It introduces essential knowledge to leverage current GPU technologies, software features, and rendering techniques. Further in this chapter, specific techniques for enhanced resource management are explored, with an analysis of their potential impact on the visual outcome of the 3D scene, all exemplified using Octane Render for C4D. Properly optimizing 3D scenes not only results in performance enhancements but also allows the project to run on a wider range of systems with different hardware settings and elevates the visual quality of the output image. If the visuals do not meet the needs of the project, an optimization becomes unnecessary.

4.2 Relation of GPU Renderers to Time Sensitive Productions

GPU renderers play a crucial role in time-sensitive productions, such as animation or visual effects for advertisements, where the production cycles are shorter. Meeting the deadline is a top priority, and reducing rendering time minimizes the risk of delays.

The speed and efficiency of GPUs for rendering are well known, and they are invaluable for productions where rapid iterations and sudden turnarounds are required. The scalability of the hardware can provide even higher speeds without requiring great changes to the system. Furthermore, it is cost-effective, as it can significantly reduce rendering times, with associated hardware costs lower compared to CPU-based solutions.

4.2.1 Impact on Artists Workflow

Rendering is the last step of the 3D pipeline. Prior to this process, the scene must be complete, with models and their animations arranged, shaders (which control the appearance of 3D objects, e.g., textures and material properties) and virtual light sources configured, and 3D cameras properly tuned, aiming to make the final visual look appealing and realistic.

For that, the 3D artist requires the interactive feedback of the render engine, a preview accurate in detail but not in full resolution of the scene, with a relatively rough and fastest possible light calculation. It is an essential feature when making quick decisions and adjustments to the visual. Thus, the speed of this interaction has a direct impact on the overall time needed for a 3D production.

GPUs and their high-speed rendering provide faster feedback, enabling artists to accelerate their work while producing something. The Live Viewer is the interactive feedback tool from Octane, and it includes multiple features to speed up the previsualization, such as render region and clay modes. The C4D Octane User Documentations present concise explanations for those:

> *CLAY MODES: A toggle that will render your scene in grayscale without textures (the result looks like Clay). This is useful to check the overall light distribution in your scene. You can also use it to review Shadows and Ambient Occlusion.*
>
> *RENDER REGION & FILM REGION: These commands are used to render just a portion of the scene in Live Viewer, allowing multiple adjustments to an object or material in your scene. Select Render Region from Live Viewer and see only that part of the scene, without waiting for the rest of the image to render. This is useful to dial in material settings without waiting for the full frame to render. (C4D Octane User Documentation, n.d)*

4.2.2 Impact of Faster Interactive Preview

In the NVIDIA Success Story, *GPU Rendering with Octane Lets Elastic Spend More Time Creating (n. D.)*, published by NVIDIA, illustrates how the fast interactive feedback of GPU rendering is a great advantage for 3D production:

> *With the demand for more complex imagery done at a high standard in less time, GPU rendering allows Elastic to keep pushing its creative and visual boundaries. The interactive feedback has given artists the power to take shots further, irrespective of deadlines. Time efficiencies also put more jobs in play, giving the firm the ability to engage in work that wasn't always possible before, while providing the time necessary for creative development (S. 4).*

Chalet (2016) points out the relevance of time optimization in rendering for advertising productions in her article *CG in Ads: The Evolution of Advertising Part 1 [The Graphic Masters Series]*:

> *But while the creative opportunities are seemingly limitless, project deadlines aren't. Shorter production cycles and higher client expectations are two of the biggest challenges facing studios today. You might have a team of CG artists who are proficient in the latest software, but if you can't render your state-of-the-art imagery in the tight timescales allocated, you'll lose the contract (Chaleat, 2016)*

4.3 Scope and Limitation

This research focuses on optimizing hardware usage for GPU rendering in offline rendering scenarios, aimed at providing a thorough analysis of resource consumption and improvement strategies, all while considering the impact of each presented technique on the visual outcome of the rendered image.

This study contains a comprehensive guide on how to build a frustum culling for Octane‘s scatter system with native C4D features, analyzing its impact on the GPU, render time, and final look of the scene.

Further, provide useful information on how to quickly resize the multiple textures from an external asset while using Adobe Bridge and its image bulk resizing feature, aiming to reduce VRAM consumption on a 3D scene. In the sequence, it was explained how to quickly replace the original texture with its optimized version using the Octane Texture Manager.

An analysis of optimization techniques for VDB meshed (aimed at the native volume builder system from C4D) is presented, while illustrating examples of common failures that can come along the way and exemplifying how to avoid them.

Furthermore, it provides an in-depth explanation of the instance types, comparing the performance of those inside C4D, and presents a factual analysis of the resource consumption between them.

At last, features from the render engine aimed at expediting the rendering process are presented, exemplifying how those can be of use to speed up the interactive preview and reduce rendering time while maintaining image quality.

4.4 Aiming for the Hardware

In the process of rendering, graphics cards are limited by the VRAM (graphics memory). The VRAM, a version of the RAM (main memory) specialized for graphics processing, runs at a higher frequency and is a volatile memory like the RAM. The function of both is to temporarily store data and calculations that are currently being executed. The VRAM is connected to the GPU, and the RAM to the CPU.

If a 3D scene includes high-poly models (models with a high number of polygons, resulting in a high level of detail and complexity), high-resolution textures, complicated shaders, animations, and lighting, then it requires more calculation power to compute the rendering, for which a GPU‘s VRAM may not be enough since the GPU‘s computing power is tied to the VRAM‘s cache memory. The CPU, on the other hand, is slower but has no limitations in its cache memory and better final rendering results. Brian Caulfield (2009) elaborates on another difference between graphics cards and processors in his article *What‘s the Difference Between a CPU and a GPU?*

> *Architecturally, the CPU is composed of just a few cores with lots of cache memory that can handle a few software threads at a time. In contrast, a GPU is composed of hundreds of cores that can handle thousands of threads simultaneously (Caulfield, 2009).*

Meanwhile, GPUs, with their thousands of cores, are thus better at processing millions of identical operations since these operations can be processed in parallel by the multitude of cores. CPUs, however, process these operations sequentially with their few but more powerful cores.

4.4.1 Hardware Limitations

- **Memory Constraints:**
  GPUs have limited memory compared to CPUs, which can restrict the complexity of scenes that can be rendered. It is not possible to add more VRAM to a graphic card, contrary to the RAM. Large and intricate scenes may exceed GPU memory capacity, leading to performance degradation or rendering failure.

- **Real-Time Rendering vs. Offline Rendering:**
  While GPUs are excellent for real-time rendering in applications like video games, they might struggle with the demands of high-quality, photorealistic offline rendering. Offline rendering often requires more computational resources and advanced algorithms, which can strain GPUs.

- **Algorithmic Limitations:**
  Some rendering algorithms are better suited for GPUs than others. While GPUs excel at parallelizable tasks, algorithms that are inherently serial in nature might not benefit as much from GPU acceleration.

- **Precision Issues:**
  GPUs operate with limited precision due to the finite number of bits available for representing numbers. This can lead to issues with accuracy and artifacts in rendering, especially for certain types of simulations or physically based rendering (PBR). For this reason, GPU renderers provides a less precise image then CPU renderers.

- **Specialized Hardware:**
  While GPUs are versatile, they are not designed for all types of computations. Certain rendering tasks, such as ray tracing or global illumination, might require specialized hardware or hybrid rendering approaches to achieve optimal results.

  Current Graphic Cards, such as NVIDIA RTX cards with RT Cores, are now able to better approach ray tracing. Through an AI analysis of the scene to be rendered, NVIDIA provides the user with the ability to implement the calculation of light rays in real time with their graphics cards.

- **Energy Consumption:**
  GPUs are known for their high-power consumption, which can be a concern for energy efficiency, especially in scenarios where rendering is performed on a large scale, such as data centres.

- **Interactivity and Responsiveness:**
  While GPUs can provide real-time rendering for interactive applications, there can still be latency and responsiveness issues, especially when dealing with complex scenes or when multiple computations are running in parallel.

- **Data Transfer Bottlenecks:**
  Transferring data between the CPU and GPU can cause bottlenecks, particularly in situations where frequent data exchange is required. This can impact performance and efficiency. There are functions that allows a GPU renderer to use of the RAM memory while strongly affecting the render speed.

- **Limited Software Support:**
  While GPU rendering has gained significant traction, not all software applications fully support GPU acceleration. Compatibility and optimization can vary between different software platforms.

- **Maintenance and Upgrades:**
  Keeping up with the rapid pace of GPU hardware advancements can be challenging. Older GPUs might quickly become obsolete, requiring constant upgrades to maintain optimal rendering performance.

Despite these limitations, GPUs remain a cornerstone of modern computer graphics and rendering, while ongoing research and technological advancements continue to address many of these challenges. Hybrid approaches that combine GPU rendering with other techniques, such as CPU rendering or cloud-based solutions, are often used to mitigate some of these limitations and achieve a balance between performance and quality.

4.4.2 Hardware Strengths

- **Speed and Efficiency:**
  One of the most significant advantages of GPU rendering is its speed. GPUs are designed to perform parallel computations efficiently, making them well-suited for rendering tasks. In offline scenarios, where time is not as constrained as in real-time applications, GPUs can significantly reduce rendering times compared to CPU rendering.

- **Scalability:**
  GPUs can be easily scaled by adding more GPUs to a rendering setup. This scalability is valuable for handling complex scenes or rendering tasks that require substantial computational power. It is important to notice that while rendering, stacked GPUs do not add up their VRAM, but just their available cores. The graphic memory is limited to the VRAM of the main GPU on the hardware, however the rendering speed grow accordingly to the number of cores available.

- **Realistic Graphics:**
  Modern GPUs support advanced rendering techniques, including ray tracing and global illumination, which can produce highly realistic and visually stunning images. This is especially beneficial for offline rendering scenarios where quality is a top priority.

- **Parallel Processing:**
  GPUs are capable of processing thousands of threads simultaneously, allowing them to efficiently handle tasks like ray tracing, path tracing, simulating complex materials and lighting.

- **AI Acceleration:**
  GPUs are well-suited for AI-based rendering techniques, such as denoising or upscaling, which can improve the efficiency and quality of offline rendering.

- **Interactive Preview:**
  GPU rendering can provide near-real-time feedback and interactive previews of scenes, enabling artists and designers to make quick iterations and adjustments to achieve the desired result.

- **Community and Support:**
  GPU rendering benefits from a large and active user community, resulting in a wealth of online resources, tutorials, and support. Due to the accessibility to GPUs, the user community and the availability of related content has expanded.

- **Multi-GPU Support:**
  Many rendering software packages support multi-GPU setups, which can further accelerate rendering times in offline scenarios.

Overall, GPU rendering offers a powerful and efficient solution for offline rendering scenarios, particularly when high-quality, photorealistic results are desired, or when time and cost efficiency are crucial considerations. It has become a standard in the film, animation, and visual effects industries for its ability to deliver stunning visuals in a timely and cost-effective manner.

4.5 Accepting the Pain: Dealing with Hardware Limitations

Scene optimization techniques are essential for improving rendering efficiency and achieving optimal performance in any render engine, whether CPU- or GPU-based. However, when rendering on a GPU, the VRAM usage needs to be taken into consideration. To make the most of what GPU rendering offers, lowering memory usage is always an advantage. Therefore, proper handling of assets, textures, and lights is of invaluable importance.

Optimization techniques should be present since the beginning of the production process, and they must be a natural part of the artist's workflow. In this section, key concepts of multiple optimization techniques are briefly explained.

4.5.1 Level of Detail

The level of detail (LOD) involves creating multiple versions of an object or model at different levels of complexity. When distant or in less visible parts of a scene, lower-detail versions can be used, reducing the computational load and improving rendering speed. Both biased and unbiased render engines can benefit from LOD to enhance efficiency.

Dynamic LOD systems are not commonly used in production. The level of detail of each model is chosen according to the needs for detail in the respective scene. However, most of the 3D animation software have dynamic LOD tools available.

4.5.2 Culling Techniques

Various culling techniques, such as frustum culling and occlusion culling, help determine which objects or parts of a scene are not visible to the camera and can be excluded from rendering calculations. This optimization reduces unnecessary computations and accelerates rendering.

4.5.3 Baking Lights into HDRI

Multiple light sources consume resources and increase rendering time. Baking Lights into HDRI images maintain quality and drastically reduces calculation time.

4.5.4 Instances and Batch

Instancing involves rendering multiple instances of the same object using a single set of geometry data. The batching process group similar objects together and render those in a single draw call. These techniques reduce CPU-GPU communication overhead and enhance rendering speed.

4.5.5 Geometry Simplifications

Simplifying complex geometry by reducing polygon count or using simplified mesh representations optimizes rendering efficiency. This technique can be applied to objects that are less visible or have a minor impact on the final image quality.

4.5.6 Texture Atlasing

Combining multiple textures into a single texture atlas reduces memory consumption and improves cache efficiency. This technique is useful for optimizing texture loading and rendering performance.

4.5.7 Texture Resolution

8K textures have better quality but are not always necessary and can result in useless data being loaded to the memory. Higher resolution textures have higher impact on VRAM usage and render time.

4.5.8 Procedural Textures

Procedural textures and effects are generated on-the-fly rather than stored as pre-made textures. This can significantly reduce the memory footprint of the scene since it does not need to be loaded into GPU memory.

4.5.9 UV vs. Triplanar Texturing

UV Mapping takes time, so triplanar texturing can be the solution for a faster workflow when texturing.

4.5.10 Caching

Caching involves storing precomputed data for reuse, reducing CPU-GPU communication. Caching enhances rendering speed by avoiding redundant calculations in subsequent frames or scenes, cutting out the calculation time of each frame, consequentially reducing render time of animations and reducing computational power needed. A common method for caching animated scenes is with Alembic (ABC) files.

4.6 Optimizing the 3D Scene: Collecting Data through Experimentation

4.6.1 Gathering Render Information

4.6.1.1 C4D Console

Direct in C4D console (Shortcut: shift + F10), it is possible to acquire detailed information about the render, as shown in Figure 1:

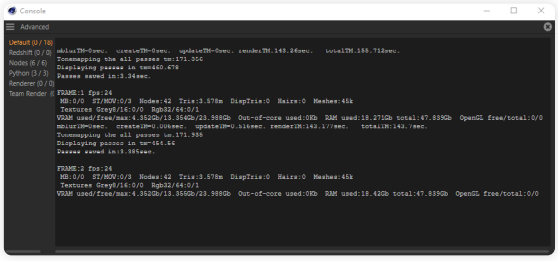


Figure 1. *C4D Console view*

4.6.1.2 Octane Device Setting

The Device Setting window is found under the Octane settings and provides an accurate account of how much VRAM is used in a scene and how it is being allocated.

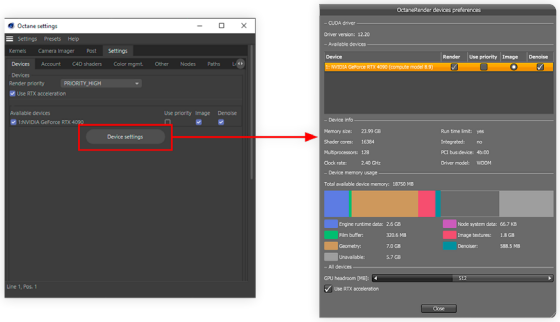


Figure 2. *Octane Device Setting*

4.6.1.3 Hardware Monitoring: MSI Afterburner

When troubleshooting, optimizing, or analyzing a 3D scene, it is important to follow in real-time the impacts each modification has on the hardware, such as variation peaks on VRAM, RAM, or CPU usage. For that, MSI Afterburner provides all this information in real-time, in a clean interface with separated graphs for each resource.

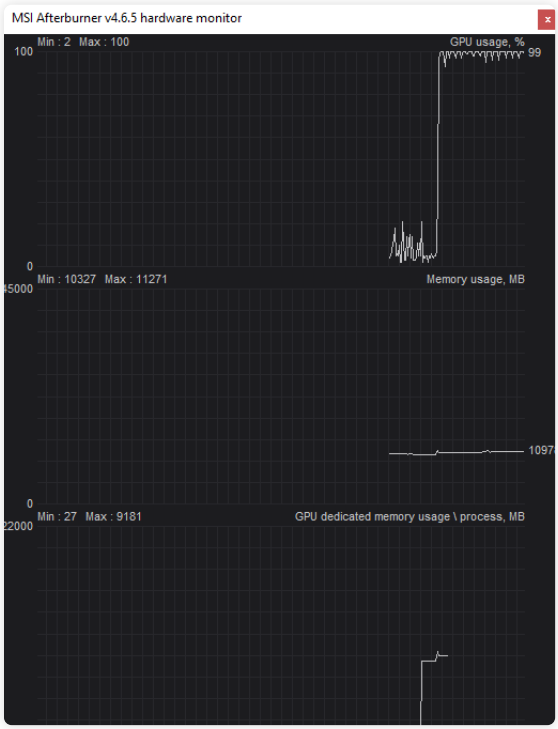


Figure 3. *Hardware Monitoring in MSI Afterburner*

4.6.1.4 C4D Render Log:

After rendering, C4D also saves a log in the scene file directory. This log provides mostly render time and file/environment information; therefore, it is not of much use for this paper and will not be taken into consideration.

However, it is good to know that this file provides the following information about the project:

- Used hardware for rendering (processor type, operational system, GPU)
- Scene information used for the render (render settings, active camera, active take, resolution, FPS, and frame range)
- Path of the rendered images
- Render time per frame and total render time

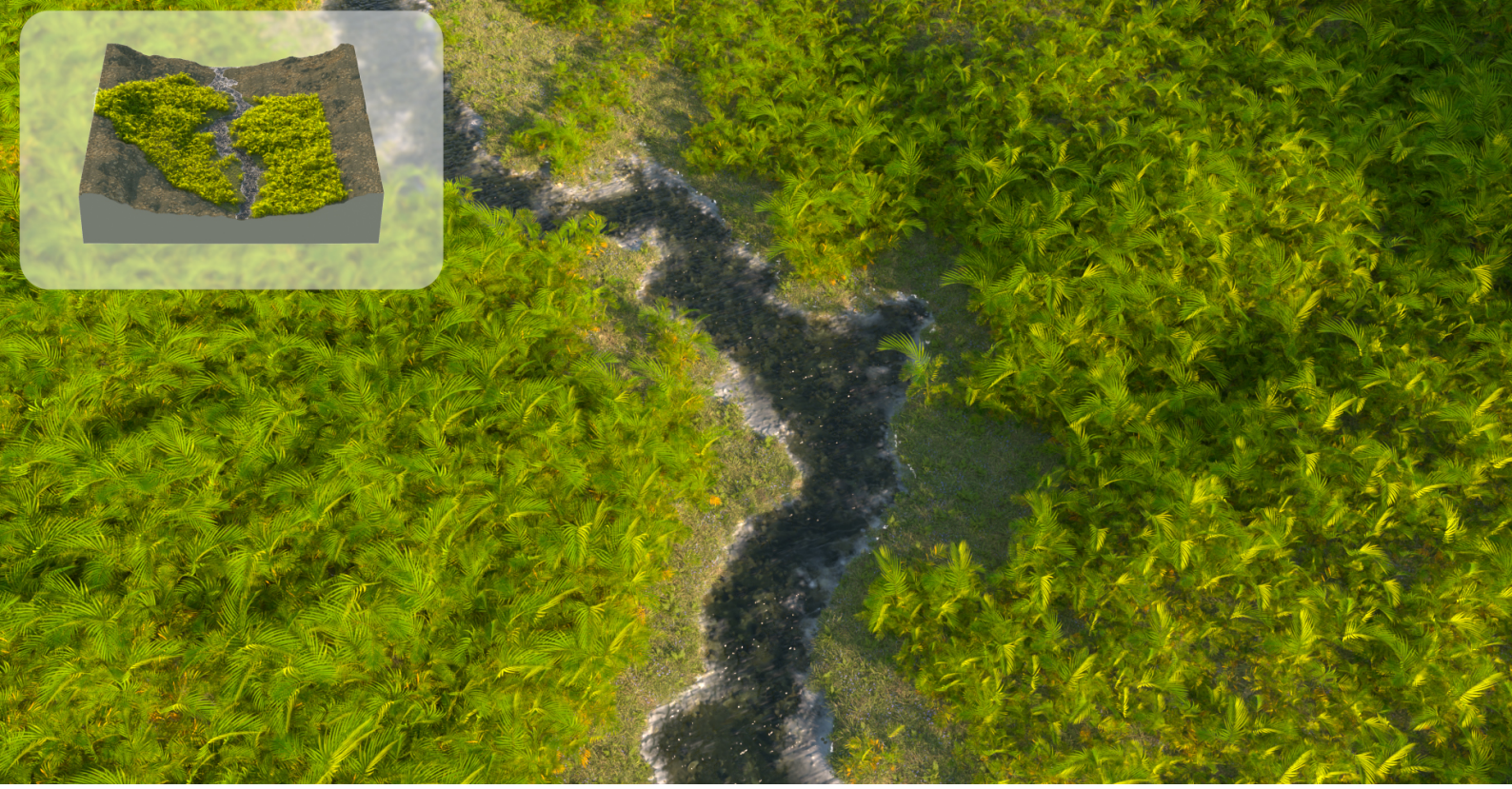


Figure 4. *Frustum Culling Octane Scatter*

4.6.2 Frustum Culling for Octane Scatter

In this section, the development of a frustum culling setup for octane scatter was documented and carefully described. The only tools necessary for that are the ones available in native C4D. An important aspect to be considered for this topic is that the scattering system inside Octane is already highly optimized, which results in low VRAM consumption. However, culling the scatter helps reduce its impact even further.

> ***Observation note:*** *When working with smaller scattered areas containing fewer objects, the viewport FPS drastically improves while scatter is active.*

Cinema 4D provides its users with the field system, an invaluable feature for the frustum culling developed in this paper. Fields can be mixed with multiple tools inside C4D, bringing a wide range of possibilities for the artist.

The use of linear fields, along with the principle of object parenting (when the child object inherits the coordinates of its parent), was the basis for this idea.

- **“Backfield Culling” – The start of the idea.**

In this first experiment, a single linear field was parented to the camera, inheriting its coordinates (position and rotation) and subtracting everything behind the camera‘s position. The Figure 5 illustrates how it affects the vertex map:

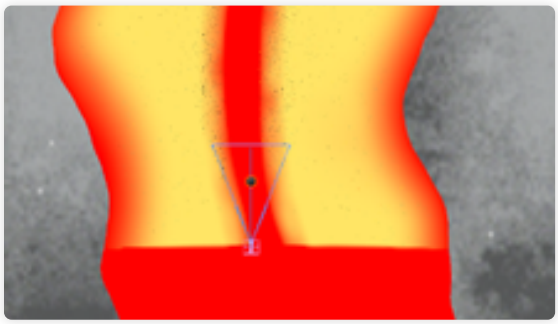


Figure 5. *First concept - Preview culling through Vertex Map*

Despite being a functional idea and providing positive results, it is not the optimal solution. There is still a large area being calculated that is not visible through the camera‘s field of view (FoV). The proper frustum culling area should exclude everything that is outside the FoV.

4.6.2.1 Building the Frustum Culling Area

The goal of frustum culling is to remove everything outside the FoV. To achieve this goal, it is necessary to combine four linear fields, one for each side of the view plane, and the length of each field needs to be set to zero.

It is crucial to note that manually positioning the linear fields would be an outdated solution. The goal is to build a system that self-adapts according to the camera settings, and Xpresso is the best choice to automate this process using native C4D tools.

The FoV from the 3D camera was utilized as a controlling parameter for establishing the precise horizontal and vertical rotation of each field. This value can be obtained from the camera settings.

The mathematical expressions for calculating the horizontal and vertical angle of view (AoV) of a camera, according to its focal length and dimension, are presented and explained next. These calculations are crucial when setting the rotations of the planar fields in Xpresso.

The Calculator Academy explains on its website that the AoV "(...) is the total extent of a scene that a camera can see. This term is also sometimes referred to as field of view."

- **Horizontal AoV**
  The horizontal rotation of the fields responsible for the left and right borders of the camera view can be calculated simply by dividing the horizontal FoV by 2. However, if the horizontal FoV is not provided by the software being used, the mathematical expression needed to reach the same result is explained below.

$$AOV = 2\arctan(d \div 2f)$$

  - Where AOV is the angle of view
  - d is the chosen dimension (often film or sensor size) (width)
  - f is the effective focal length.

  "To calculate the angle of view, divide the sensor size by 2 times the effective focal length, take the inverse tangent of this result, then multiply by 2." (Calculator Academy, n.d.)

- **Vertical AoV**
  The vertical rotation of the fields responsible for the top and bottom borders of the camera view.

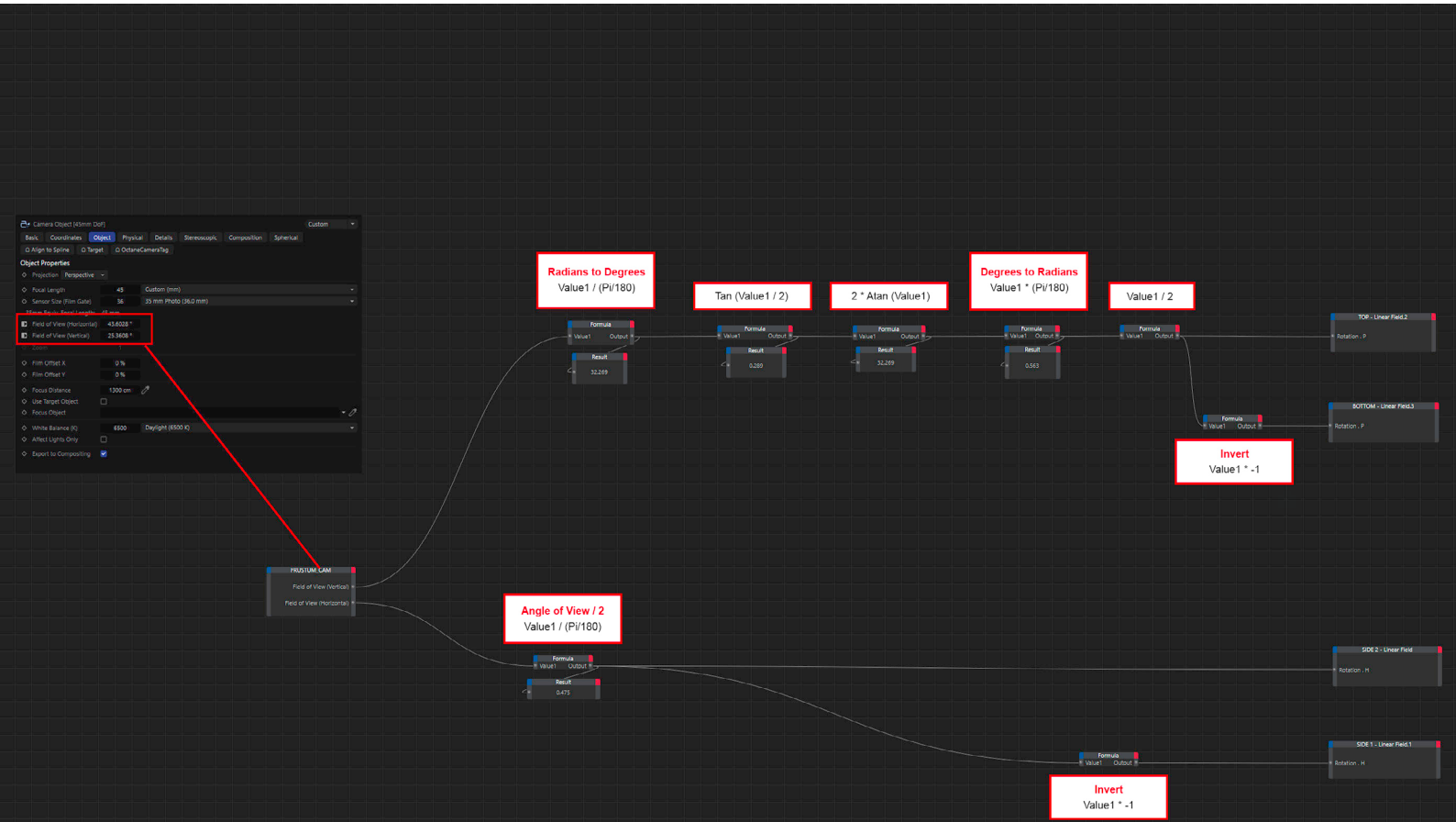
$$VFOV = 2 \times \arctan(\tan(h \div 2) \times AR)$$

  - Where VFOV is the vertical field of view
  - h is the horizontal field of view
  - AR is the aspect ratio (i.e. 16:9 = 16/9 = 1.7777)

  "To calculate the vertical field of view first take the tan of the horizontal field of view divided by two, multiply the result by the aspect ratio, take the arctan of that result, and then multiply by 2." (Calculator Academy, n.d.)

Converting the mathematical expressions into the Xpresso language results in the node graphic presented in Figure 6.

Figure 6. *Xpresso graph - Frustum culling basic setup*



4.6.2.2 Troubleshooting

When the culling area covers the exact size of the FoV, issues arise with the culled scatter: the geometry to be culled is still inside the FoV, therefore it is possible to see it disappearing.

The reason for this is simple: the culling process begins precisely at the pivot position of the object, which is a point holding only position information without any area or volume. In contrast, the geometry contains a bounding box, a box that surrounds the 3D object, which occupies an area and volume. Despite the pivot going out of the view sight, the same cannot be applied to the geometry itself.

There are other situations in C4D that relate to this fact. A simple example to illustrate this behavior is presented in Figure 7. A cloner without “reset cordites” is affected by an effector with a linear field. The pivot of the cloned object is not centered. As soon as the effect takes place, the objects must gradually change color

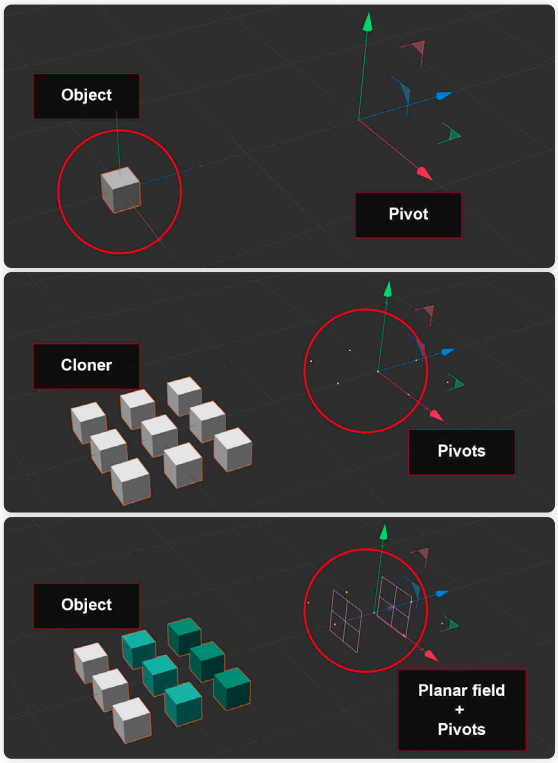


Figure 7. *Previewing influence of pivot position on effectors*

- **Conclusion:**

As explained, the effect just takes place after crossing the pivot position, resulting in the gradual color change.

There are three different ways to solve the problem on the culling system:

- Instead of letting the field object zeroed into the camera position, manually moving them left or right, above or below the respective culling direction.
- Adding an extra rotation to the planar field. This can be done in Xpresso – Each situation needs its own variation in the rotation angle; therefore, providing a new node to control this parameter is an optimal solution for it.
- Moving the group containing all four fields further behind the camera, thus providing enough space for the objects closer to the camera to get out of the view sight.

The best solution found was mixing two of the given options: bringing the group containing the fields further behind the camera and adding the extra rotation angle for each field, as illustrated in Figure 8:

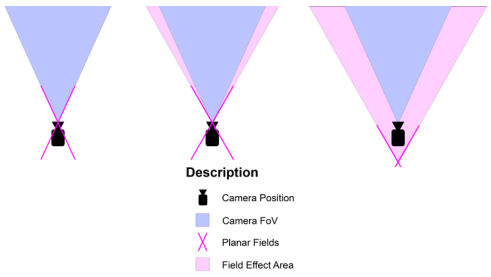
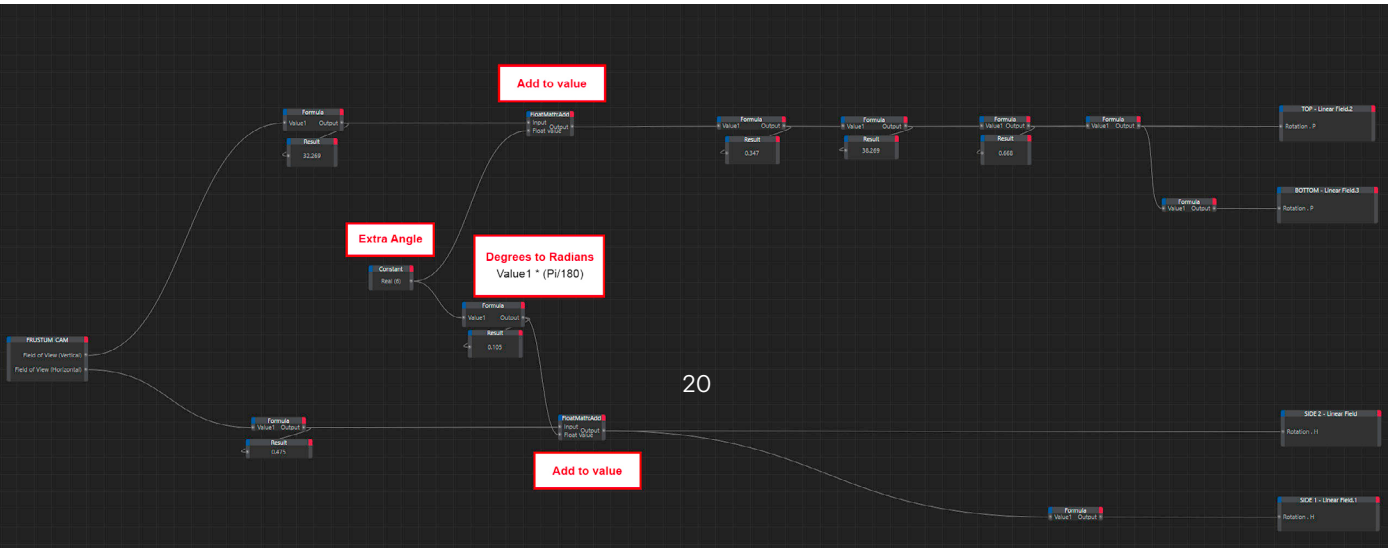


Figure 8. *Representation of the final positioning of the fields*

The corrected Xpresso graph with the new mathematical functions stays as shown in Figure 9:

Figure 9. *Final Xpresso graph - Frustum culling setup*



4.6.2.3 Bringing the Frustum to Life

There are two viable ways to apply the frustum in the scatter systems: one is fully calculated through vertex maps, while the other is calculated through a plain effector on the scattered objects. The biggest difference between those options is their overall control over the final look.

01. Calculating a vertex map, which sets the area where the scattered should be applied.

02. Scattering objects on the entire surface and subsequently deactivating all objects outside the FoV using a plain effector.

The biggest difference between those options is the overall control over the final look.

Now that the planar fields are properly positioned, they need to be blended and structured correctly. In order to achieve the desired result, all the planar fields should aim inside the FoV. This way, blending them in multiply mode will result in the exact FoV.

4.6.2.4 Method one: Frustum Culling the Vertex Map

The planar fields used to build the culling area are applied in multiple vertex maps; therefore, the most logical way is to group them in a group field.

> ***Observation note:*** *This group field still accepts other different fields for further culling operations, which can be independent of the camera movement.*

When all four linear fields are in the correct direction and blend mode, the group field itself can be applied on top of the field's hierarchy of any vertex map. There are three possible blend modes to use: *min*, *multiply*, or *clip*. The choice is made according to the needs of each situation.

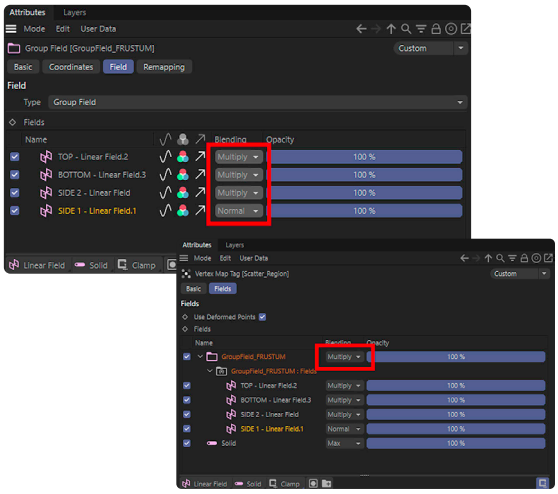


Figure 10. *Setting up and blending the planar fields*

Figure 11 serves the purpose of previewing the frustum area:

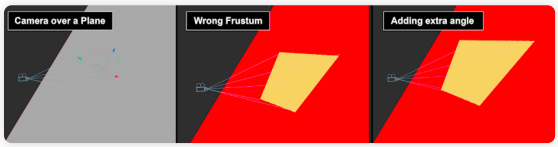


Figure 11. *Preview culled vertex map - Blended planar fields*

The last step is applying the corresponding vertex map to each Octane scatter system.

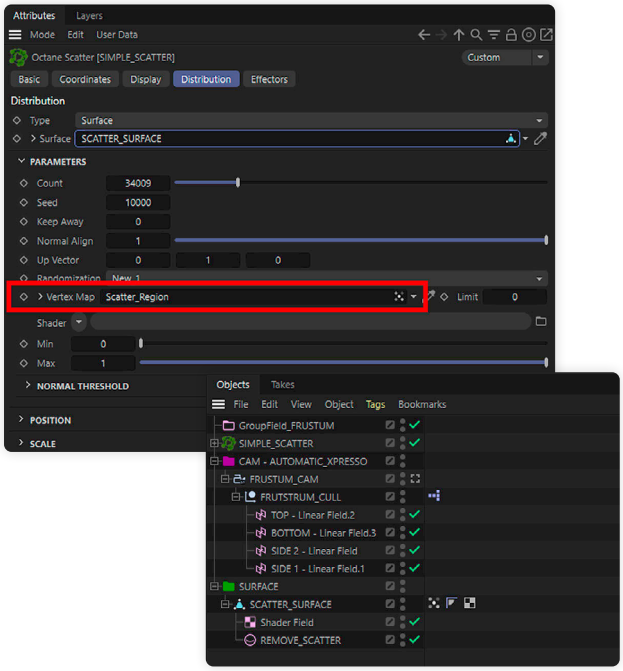


Figure 12. *Applying vertex map*

> ***Observation note:*** *As a personal setting, I like setting the group field on top of my scene, while the linear fields are inside a null parented to the camera. However, parenting the group field and zeroing out the coordinates to get the camera position and rotations also work.*

To add variation to the scatter distribution (scale, position, and rotation), use procedural shaders on the Distribution tab inside the scatter object; those must be CPU-based shaders, meaning that Octane procedural textures will not work.

Figure 13 illustrate the result, presenting how the objects are scattered accordingly to the different camera positions. For demonstration purposes, the display mode of the scatter is set to „geometry". The results prove that both the vertex map and scattered objects update as expected.

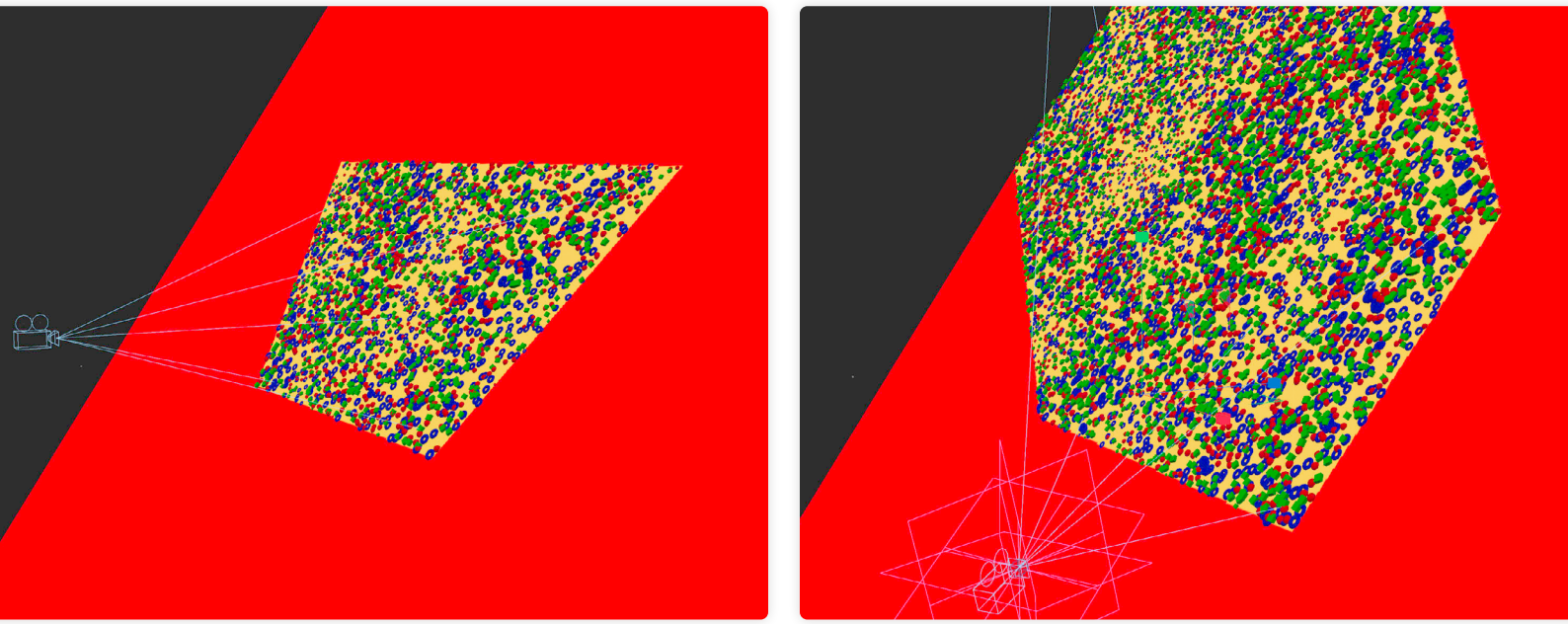


Figure 13. *Final result - Frustum culling the vertex map*

- **Considerations**
  - The distribution modes are only texture-based and are found inside the distribution tab of the scatter system.
  - The procedural textures for the distribution must be CPU-based. Octane textures do not work.
  - Gray-scale images can be used to restrict the scattering area.
  - Gray-scale textures or images are ideal when controlling larger areas of the scattering system distribution.
  - Using shaders to control the scale provides areas that are harder to fully cover, requiring more instances to be scattered around.
  - It is harder to achieve realistic randomization per instance (in scale and rotation) since the distribution mode must be texture-based.
  - When culling the vertex map, the scatter system requires a denser mesh for the distribution surface; otherwise, a “tilling effect” might happen.
  - Effectors cannot be used to work on the distribution. Those are based on the number of instances being scattered. If this number varies, the effect applied to each instance will also change, bringing glitches to the final animation – Since the visible area of an animated camera is constantly changing, using a vertex map based on the FoV limits the active scattering area, therefore changing the number of active instances per frame.

4.6.2.5 Method two: Frustum Culling with Plain Effectors

Effectors are easy to comprehend and provide great control over multiple objects.

The logic used in this method is the opposite of the logic used on method one: instead of controlling where the instances should be placed, it sets where they should be deactivated.

The vertex maps delimitating the scatter region should remain unchanged, delimitating where the objects need to be placed over the entire surface. A plain effector will be responsible for culling the geometries outside of the FoV while using the group field.

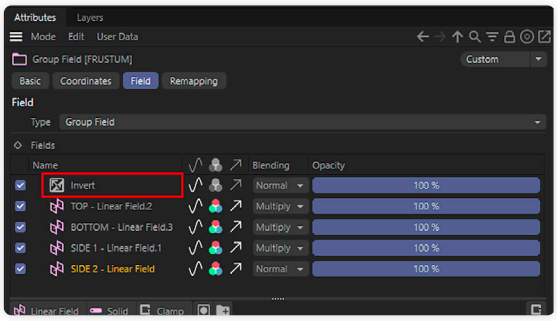


Figure 14. *Invert modifier applied on top of the planar fields*

It is possible to set an invert modifier on top of the group field or simply invert the direction of each planar field. However, inverting the active area works well.

> ***Observation note:*** *Using a group field containing all linear fields remains a good choice, but it is not essential this time. The fields are applied only to the plain effector responsible to cull the geometry, while this effector is applied over the multiple scatter systems.*

In this method, the culling is done through a single plain effector, which contains the group field applied in the Fields tab. To set the frustum culling, first activate scale in the Parameter tab of the plain effector, set it to be uniform and the value to -1 (everything that has a value of -1 is turned off from calculations), then apply the effector on each scatter system, as shown in Figure 15

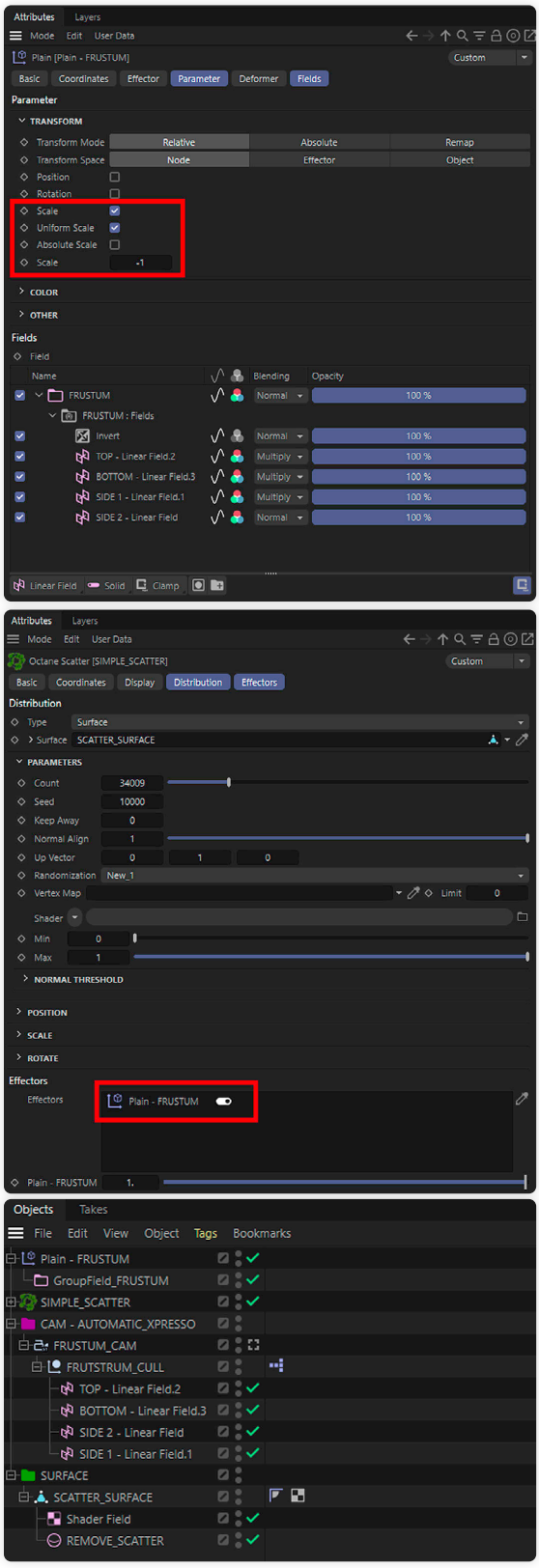


Figure 15. *Preparing the culling using plain effector*

Now other C4D effectors can also be applied and stacked in the Effectors tab of the scatter system, and the plain effector acts as the frustum culling.

In this method, Octane scatters the objects on the entire distribution surface and the plain effector deactivates it in the non-visible areas. Therefore, the number of instances is always constant, allowing effectors to take place without producing glitches.

- **Considerations**
  - Using effectors is the most common workflow when scattering objects with Octane.
  - Texture distribution methods are still available inside each scatter system, which provides more control and freedom for the artist.
  - Gray-scale images can be used to restrict the scattering area of each scatter system.
  - Effectors have a per-instance effect, providing a more organic visual.
  - The size of each instance is better randomized, providing a more realistic feel of nature.

4.6.2.6 What are the Possible Problems when Frustum Culling?

- **Shadows Glitches**

When light is coming from behind the camera, casted shadows might be visible through the camera. Therefore, culling geometries can yield problems on the final visual. As soon as geometry disappears, the same will happen with its shadows.

The example presented in Figure 16 illustrates the described situation. Both images are extracted from the same frame of the same animation. On the left, all the scattered objects are visible, while on the right, the scatter systems are being culled.

It is important to acknowledge that culling every scatter system is not always the best solution. Since each scatter system has its own control, there is the option to select only the scatter systems that influence the shadow casting, without activating the culling system on those.

> ***Observation note:*** *culling only the smaller objects scattered through the scene presented in Figure 16 (grass, fallen leaves, and stones) would not affect the final visual congruency.*

Figure 16. *Example of shadow failures when culling geometries. On the left, all scattered objects are visible and on the right the scatter systems are being culled*

- **Reflections**

When reflective surfaces are present in the scene, the culled area will be visible through them. Therefore, it's expected to have incongruencies in the rendered result. If preserving the details in the reflections is not important, culling is a good idea and will save VRAM.

The results are presented in Figure 17. On the left, the frustum culling is active for every scatter system, and on the right, the frustum culling is off. On the reflexes of the metallic sphere, it is possible to see the results of the culling.

Figure 17. *Visualizing the culling through a reflective sphere*

4.6.2.7 Frustum Culling: Performance Analysis

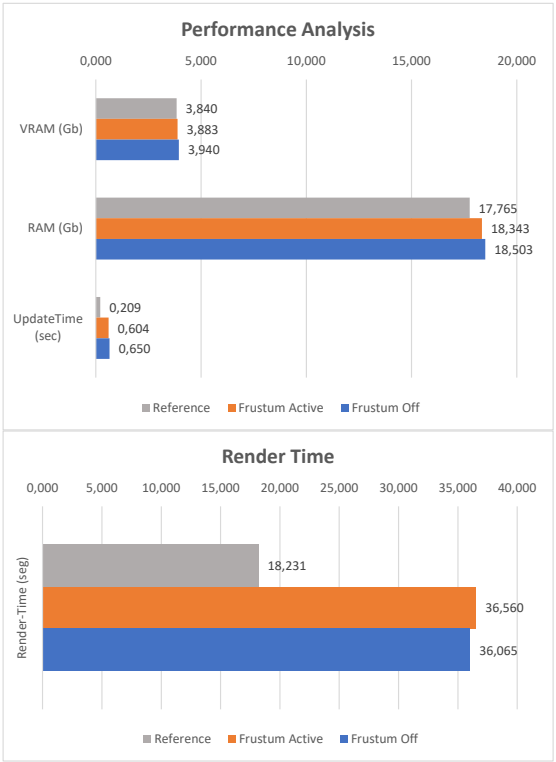
In the next two examples, five scatter systems were used to compose the scene:

- **Small-scale Scene**

Figure 18. *Small-scale scene with a reflex ball to preview the scatter systems being culled*

To set a reference parameter for resource usage, the same frame in Figure 18 was rendered without scatter systems. The rendered images and render logs (see Appendix A, Table 1) were taken into consideration for the following analysis. The information presented on the logs was transcribed into charts, presenting a graphical comparison between the performance results of each considered case:

Table 1. *Performance analysis of frustum culling system in small scale scene*



When using the frustum culling system in a small-scale environment, the considerations are the following:

- As more objects are culled, VRAM usage reduces slightly. The opposite happens when more objects get back into the FoV.
- There are no significant changes in VRAM usage.
- There are no significant changes in the update time.
- Render time has no significant changes.

- **Conclusion:**

When working on small-scale setups, culling does not provide much improvement and therefore, should not be considered a top-priority optimization technique. Furthermore, the impact on performance does not justify removing the details on reflexes or light bounces.

Figure 19. *Large scale scene with scattering systems*

- **Large-scale Scene**

To set a reference parameter for resource usage, the same frame in Figure 19 was rendered without scatter systems. The rendered images and render logs (see Appendix A, Table 2) were taken into consideration for the following analysis. The information presented on the logs was transcribed into charts, presenting a graphical comparison between the performance results of each considered case.

When using the frustum culling system in large-scale scatter systems, the considerations are the following:

- As more objects are culled, VRAM usage reduces slightly. The opposite happens when more objects get back into the FoV.
- The frustum culling provides a small increase in render time.
- The frustum culling reduces VRAM usage.
- RAM usage increases significantly with frustum culling.
- Update time has made a significant improvement, being enough to compensate for the difference in render time.

- **Conclusion:**

If enough RAM is available, frustum culling larger scenes brings improvements that should be taken into consideration; it brings a considerable reduction in update time per frame and reduces the VRAM usage.

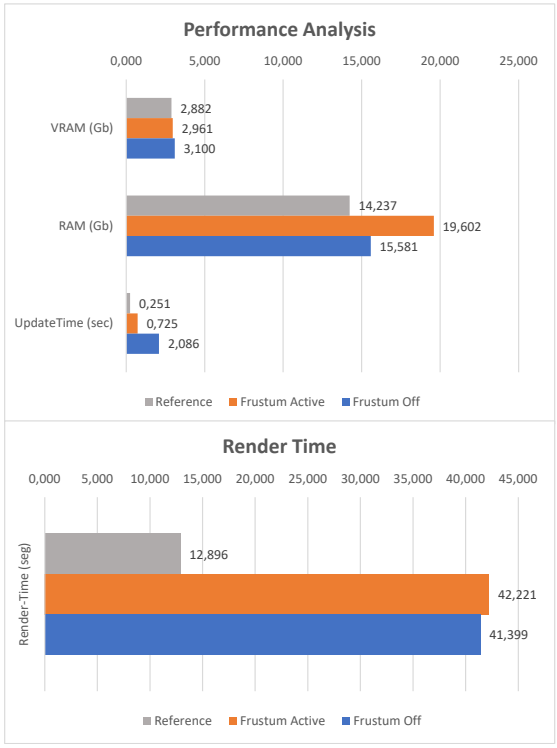


Table 2. *Performance analysis of frustum culling system in large scale scene*

4.6.2.8 Comparing Performance and Visual Outcomes of Both Methods

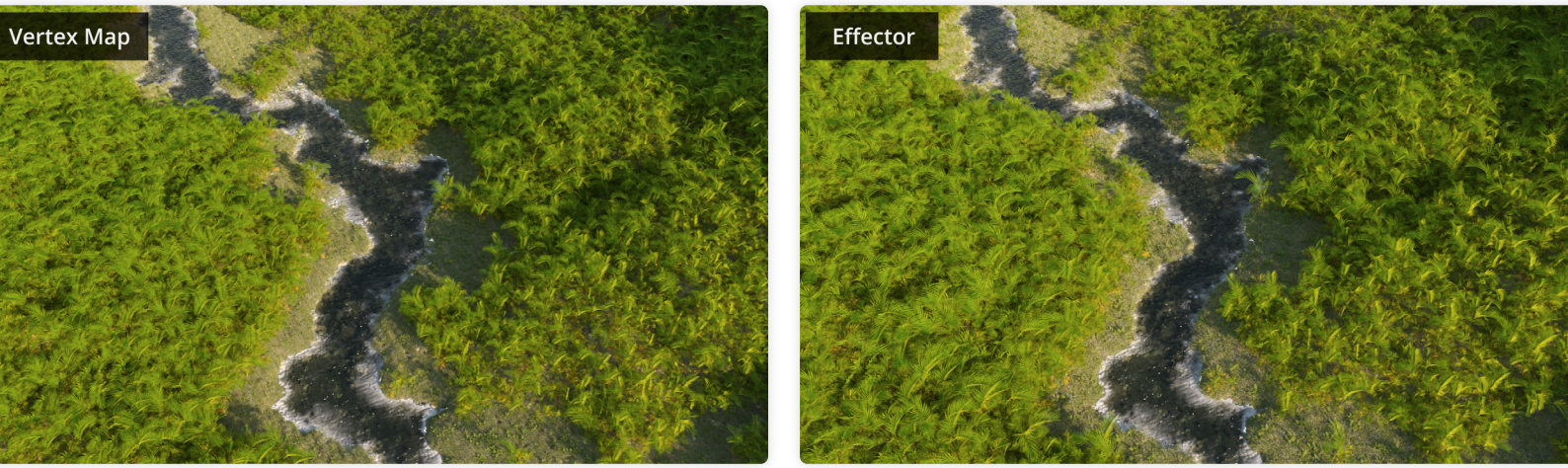


Figure 20. *Comparing both culling methods - Culling the vertex map and culling with plain effector*

The rendered images and render logs (see Appendix A, Table 3) were taken into consideration for the following analysis. The information presented on the logs was transcribed into charts, presenting a graphical comparison between the performance results of each considered case:

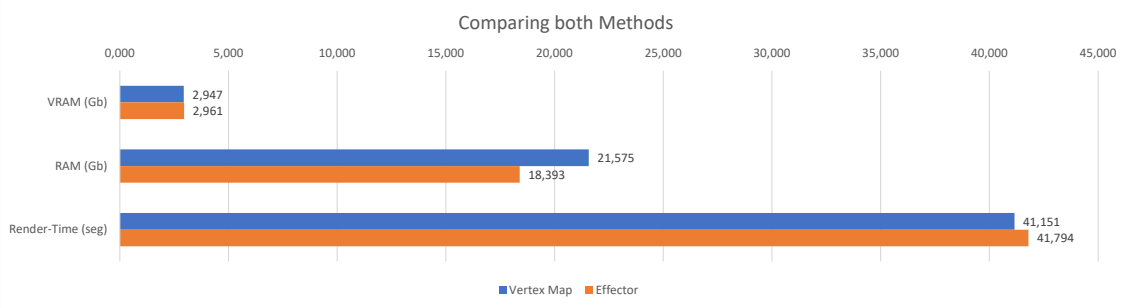


Table 3. *Performance analysis of both developed frustum culling methods*

There are no significant changes in GPU usage between both methods; however, the RAM usage reduces when culling the scatter with method two, culling with the plain effector. Furthermore, it also allows the number of scattered objects to be reduced by almost half, while covering the same area.

Since effectors have a random per-instance effect, objects close to each other can have totally different scales, resulting in a more homogenic outcome. Meanwhile, texture distribution methods provide a generic scale for the overall scatter system: objects positioned on darker areas of the shader are smaller, and those gradually increase in size when getting closer to brighter areas of the shader. The darker areas are harder to fully cover and, therefore, require a higher number of scattered objects.

Using shaders to control the distribution can provide optimal results when working on larger chunks of objects, while randomizing the distribution with effectors provides a more organic and realistic randomization per object, not being biased by its position on a grayscale image. Therefore, mixing both methods provides great results and gives the artists more control over the look-development (look-dev).

4.6.3 Bulk Resizing Textures

Textures play an important role in a 3D scene, as they contribute with details, realism, and visual information to 3D models and scenes. However, not every model requires high texture resolutions, as sometimes they are placed far from the camera and the fine details are not even possible to be noticed on the rendering.

Nonetheless, when working with external assets, the choice of image resolution and file type is limited and can be above the needs of the project, resulting in useless memory consumption.

E.g., assets from Quixel Megascans (Quixel Megascans, n.d.) are offered with textures in 2K resolution or higher. Often, small elements scattered throughout the scene or those at greater distances do not require such high resolution. A resolution of 1K or even 512x512 pixels is sufficient in many cases.

There are straightforward options to quickly convert these textures into an optimal format. This process is commonly used by photographers when generating previews of their picture in RAW format.

Adobe Photoshop, Lightroom, and Bridge can bulk resize images (resize multiple images at once), and during the process it is also possible to change file type, consequentially changing compression methods or bit-depth, thus resulting in an even smaller file size.

4.6.3.1 Setting Up Presets on Adobe Bridge

Inside Adobe Bridge add a preset on the export tab for the desired file resolution. If needed, multiple presets can be added.

01. In the preset settings window, the option to save the converted images in the original file location must be selected.

> ***Observation note:*** *saving it to a subfolder helps keeping everything more organized. This subfolder can be named as pleased, but as good practice, the name should clearly indicate the resolution of the textures stored there.*

02. Next comes the image format tab, where the file type, bit-depth and color space can be set.

03. At last, is the image resize tab, where the settings for image resolution are found.

Now draggin and dropping images on the presets will create a queue, allowing to export everything at once.

> ***Observation note:*** *For organizational clarity, rename the converted images by appending the respective texture resolutions to the names. This makes it easier to locate the appropriate texture. The* ***Bulk Rename Utility*** *software is a fitting tool for executing this task.*

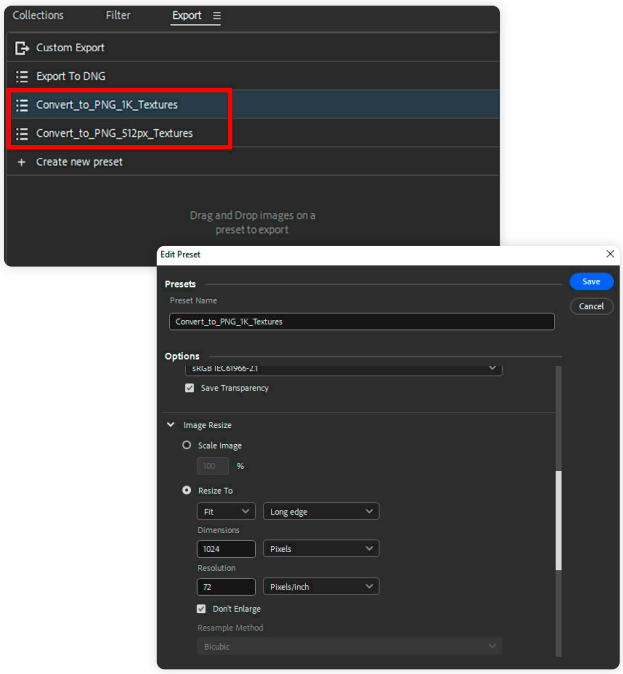


Figure 21. *Export presets in Adobe Bridge*

4.6.3.2 Changing Multiple Textures at Once with Octane Texture Manager

The Octane Texture Manager presents every image being loaded in Octane, also showing its name and file path. Furthermore, it provides some features that will help with this process, such as the "*replace… with*".

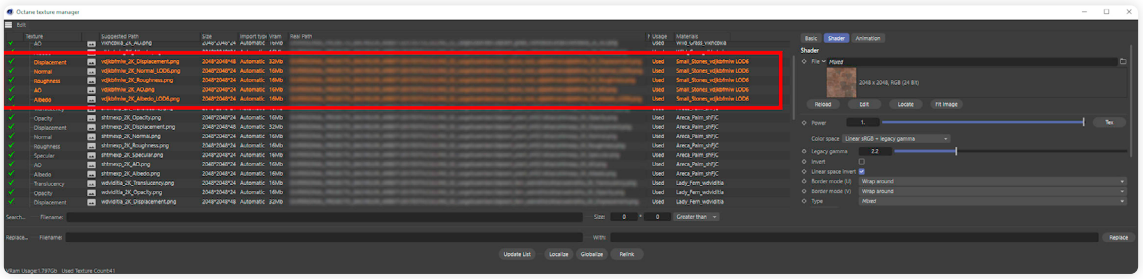


Figure 22. *Finding the image textures in Octane Texture Manager*

In this example, only the highlighted files in Figure 22 are being replaced. For that, it is necessary to have a unique way to point out which specific texture needs to be changed (in the presented case, the identification code of the asset from Quixel Megascans will be used).

Right after the code is the texture resolution, which currently is 2K and must be replaced with the 1K version, as shown in Figure 23. Pressing *replace* will make Octane search for the textures matching the name settings and automatically locate and replace them in their directory.

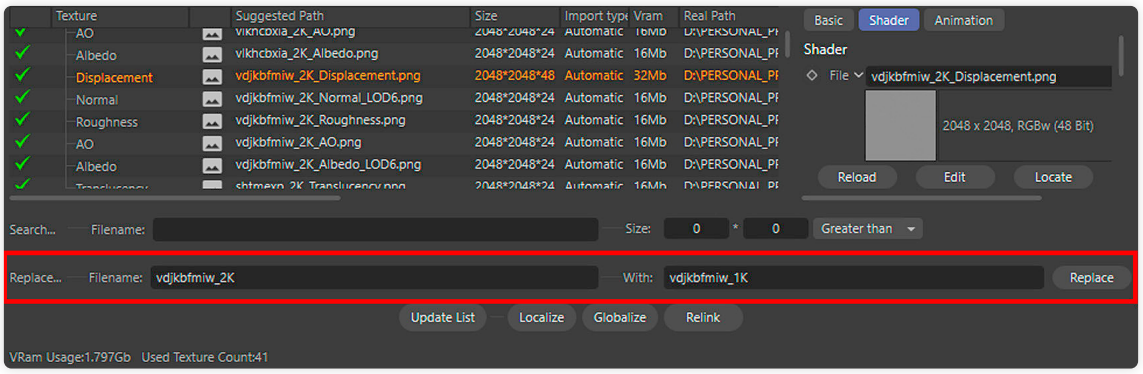


Figure 23. *Replace function in Octane Texture Manager*

Updating the shader will make the changes to be visible in the viewport and render. This process must be repeated for all textures that must be resized.

4.6.3.3 Analyzing the Results

The same frame from Figure 20 was taken as an example for the next analysis. All the texture images of the four shaders applied to the scattered objects were replaced with their 1K version image, resulting in a reduction of 237MB of VRAM usage while maintaining the same quality on the rendered image.

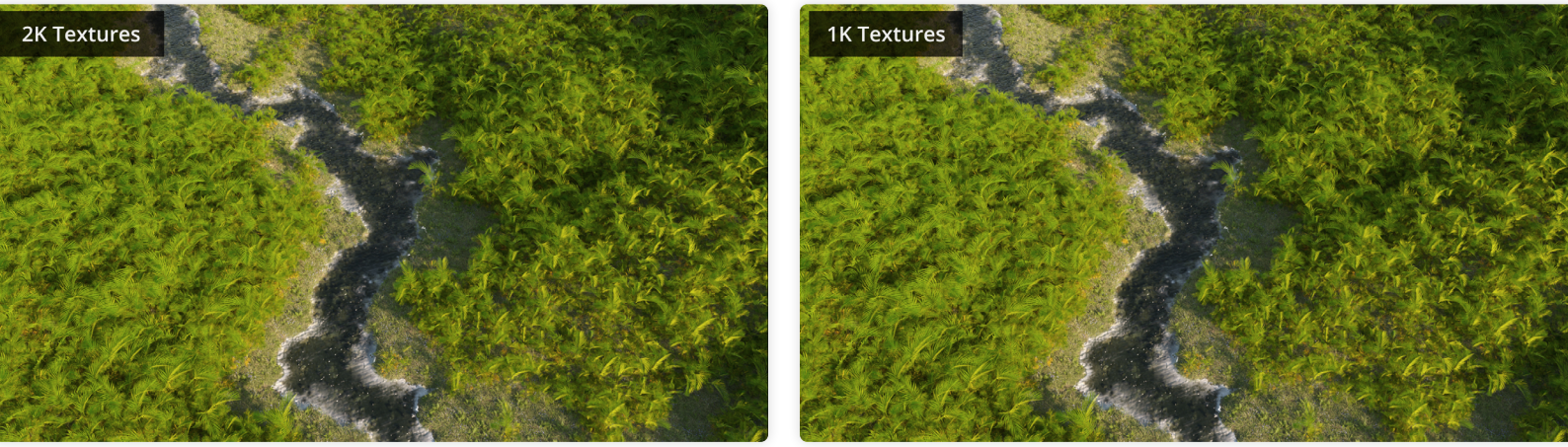


Figure 24. *Comparing texture resolution*

Octane Device Setting accurately presented how much memory was used for textures in each situation of Figure 24, as can be seen in Figure 25.

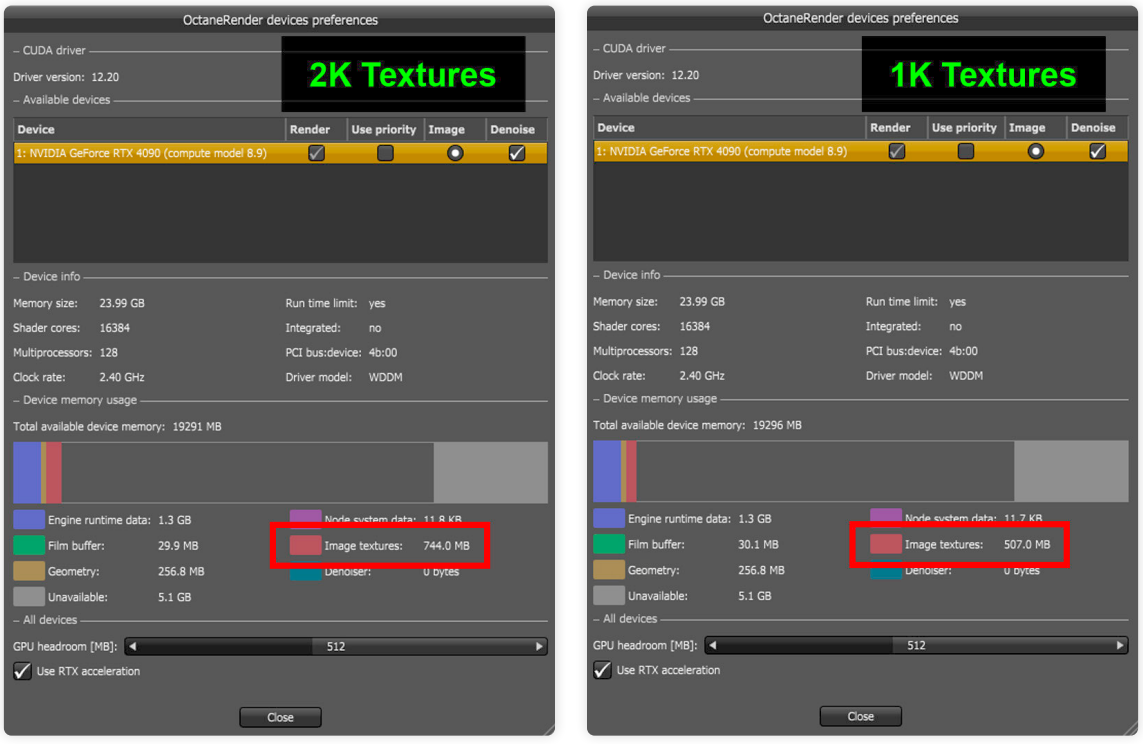


Figure 25. *Memory consumption with 2K and 1K textures*

Figure 26. *Frustum Culling Octane Scatter*

4.6.4 Model Optimization: Volume Builder / VDB Geometries

Volume builder has become increasingly popular among Cinema 4D users, whether for creating static geometries or animations. It is an OpenVDB-based tool that allows the artist to generate complex geometries very quickly, providing much control of the result. VDB meshes are generated by exploiting the properties of voxel grids, which can result in highly dense meshes.

When working with refractive and reflective materials, a dense mesh or a proper topology is necessary to achieve an accurate response to light during rendering. Therefore, optimizing meshes from volume builder requires careful attention; otherwise, the results may not meet the expected visual quality.

In this section, possible optimization processes for meshes from volume builder will be presented. To set a reference parameter for resource usage, the same scene was rendered without the geometry being analyzed. Certain information from the render logs (see Appendix A, Table 4) was selected and transcribed into charts, presenting a graphical comparison between the performance results of each considered case.

4.6.4.1 Optimizing Static Geometries

The first thing to take into consideration when working with a dense mesh is reducing its polygon count (poly-count). Nowadays, the 3D software has great tools to do this automatically, such as remesh and poly-reduction tools. Furthermore, when working with VDB, it is also possible to use the adaptivity feature when meshing the volume. However, those features can also generate problems in the process.

Examples of the static mesh and possible failures that can happen during the optimization process of VDB meshes are presented next, along with the differences in VRAM usage, triangle amount, and render time among those.

As a reference for calculation, a render of the scene without the main geometry was made, as can be seen in Figure 27.

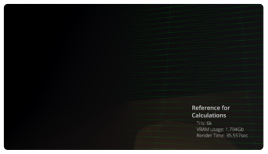


Figure 27. *VDB Mesh: Reference for calculations*

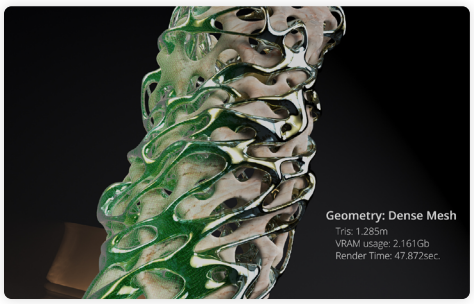


Figure 28. *VDB Mesh - Original (Dense Mesh)*

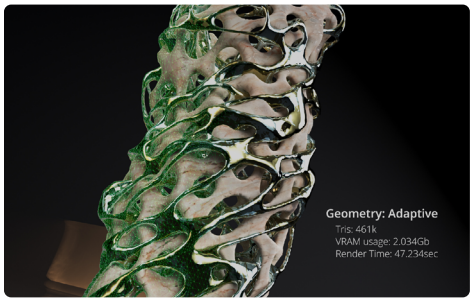


Figure 29. *VDB Mesh - Adaptive mesh active*

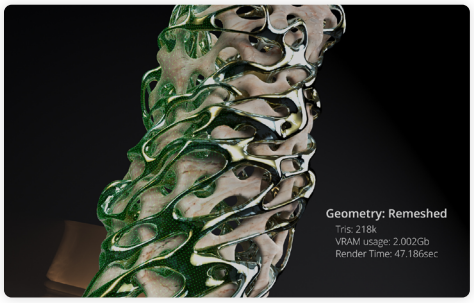


Figure 30. *VDB Mesh - Remeshed geometry*

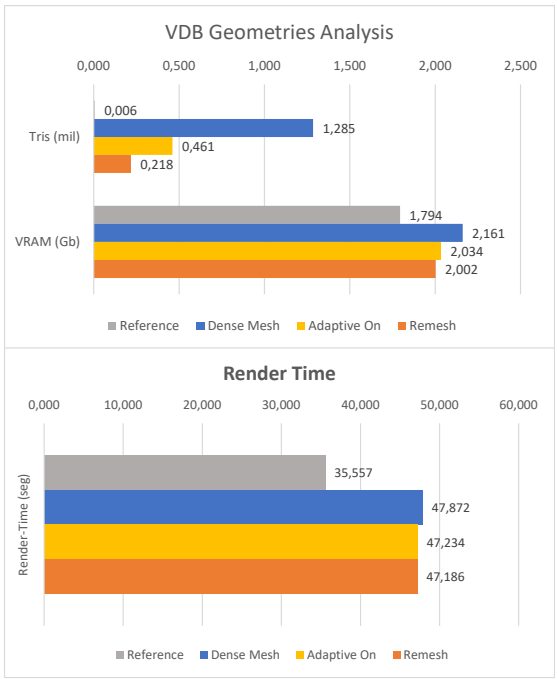


Table 4. *Performance analysis of the optimizations on a volume builder geometry*

- **Static Geometry - Results Analysis:**
  - Dense mesh uses significantly more VRAM than the retopologized mesh (retopo-mesh).
  - The render time of the different methods is almost the same.
  - The reflections on the retopo-mesh are better.
  - The final shape of the retopo-mesh is smoother.
  - The lowest poly-count is the remeshed version.
  - Adaptive mesh produces irregularities on the mesh, which strongly affect the reflections.
  - When “adaptive” is set up, there is a significant polycount reduction, but at the cost of the final shape quality.

4.6.4.2 Special Situations – High Detailed Geometry

For the example presented next, the remeshed geometry presented before was used in every situation. The only geometry that changes is the fluid geometry.

It is important to acknowledge that highly detailed VDB meshes or simulations are commonly set in high importance on a 3D scene; otherwise, the VDB simulation itself would be in lower resolution, consequently resulting in a mesh with lower polycount and fewer details.

It is important to mention here that VDB meshes are commonly cached, reducing the long calculation times and also avoiding software crashes. However, this is not the focus of this topic.

Figure 31. *VDB fluid geometry - Original (Dense Geometry)*

Figure 32. *VDB fluid geometry - Adaptive mesh active*

Figure 33. *VDB fluid geometry - Remesh*

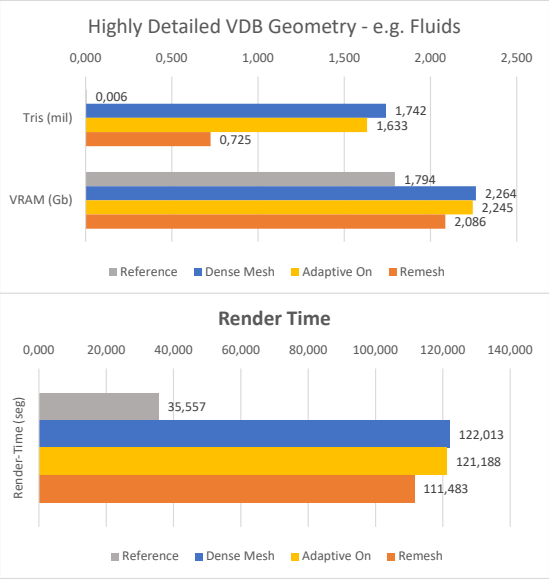


Table 5. *Performance analysis of the optimizations on a high detailed VDB geometry*

- **Fluid - Results Analysis:**
  - Dense mesh preserves the fine details.
  - Dense mesh has accurate reflections and refractions.
  - When “adaptive” is set, even with a small reduction in the polycount, the reflections and refractions become inaccurate.
  - Retopo-mesh has good refractions and reflections at the cost of the fine details in its shape.
  - Retopology is not recommended for animated meshes and would not be an option when working with fluid animations.

4.6.4.3 Compairing the Final Scene

Figure 34. *On the left, the optimized scene and the right, the scene with the original VDB geometries*

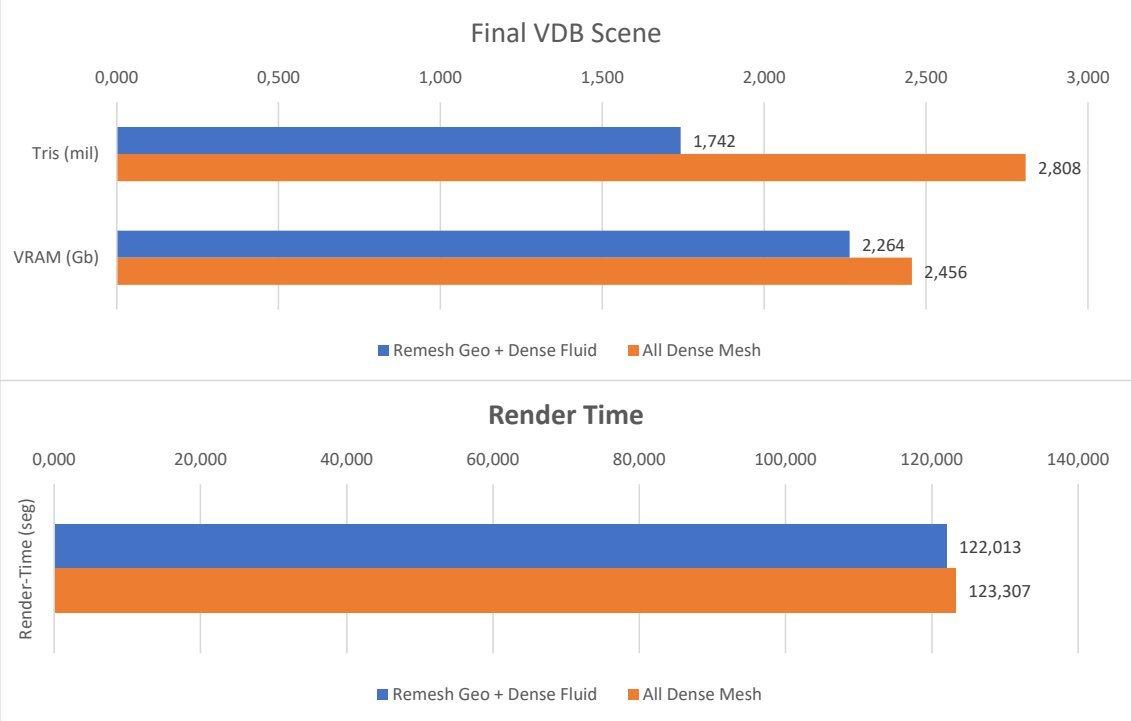


Table 6. *Graphic camparison of both optimized and non-optimized scenes*

4.6.4.4 Final Considerations

GPU renderers can rapidly process polygons; hence, reducing the polycount does not always decrease render time, but it does reduce VRAM consumption.

In the presented example, the reduction in render time when remeshing the liquid was related to the amount of information generated through the reflections and refractions. Due to the significant loss of detail in the fluid, the number of light bounces needed within the geometry has consequently decreased. A highly detailed refractive surface allows more light rays to bounce around, resulting in an increase in render time.

On the voronoi geometry, the remeshed process generated an optimal topology with accurate reflections and a sixth of the polycount, resulting in a reduction of almost 200 MB of VRAM usage. Since its material was not refractive, there was no significant improvement in render time.

When a highly detailed mesh derived from a VDB object is included in the scene, it is commonly set as a high-importance element, demanding a substantial level of detail. Consequently, it is advisable to focus on optimizing other elements within the 3D scene. Alternatively, a second option involves diminishing the quality of the VDB simulation, leading to a lower poly mesh. The optimal render quality is achieved with the original VDB mesh, the dense mesh.

4.6.5 Instance vs. Multi-Instance

4.6.5.1 What are Instances?

Instances are procedural copies of a geometry. It is useful to replicate or duplicate objects in a 3D scene. Instead of generating the same geometry set twice, those are referenced or pointed to the original object data, allowing the creation of multiple copies of this object without significantly increasing the resources required. Furthermore, when modifying the original geometry, all its instances will be modified accordingly, avoiding the need to edit each copied element.

In C4D, instances are commonly known as Clones because, most of the time, they are used in Cloner Objects. The concept of instances is commonly applied in the 3D world and is present in most of the 3D software.

There are three types of instances: instance, render-instance, and multi-instance. As Benson (2023) mentions his guide *Resource Management 04: Instances*, Octane does not deal well with render-instances; therefore, those will not be mentioned in this paper.

- **Instances: Instance Mode**

When an object is instanced in instance mode, it has small limitations on its controls, but those are exact copies of the original object, i.e., when the original object is modified, its instances are also modified accordingly. Benson (2023) provides the following description in the part four of his guide *Resource Management 04: Instances:*

> *Creating an Instance Object set to Instance mode is very similar to just making a copy of the object […], but it has the added benefit of being able to modify or swap the source geometry once and having all the linked Instances update to match. It's great for creating, manually placing, and deforming a few variations of an object in a scene. We can't adjust the parameters (fillet, segments, size) of the instance, but we can directly adjust anything in the Coordinates tab (position/scale/rotation). (Benson, 2023)*

- **Instances: Multi-Instance Mode**

Multi-instances provide major benefits in terms of performance and efficiency. Every object is considered one and is loaded only once on VRAM and RAM. However, this efficiency comes at the cost of versatility when it comes to individual animation, deformation, and texturing.

> *[…] C4D isn't great at managing a whole lot of single objects. The Instance mode of the Cloner really highlights this limitation when we go over a few thousand instances. Multi-instance removes this issue by treating the entire system as one single object. Suddenly a few hundred thousand, or even a million clones is no big thing. Since the source geometry only gets loaded into RAM and VRAM once (instead of for each clone like in Instance mode), this means we can potentially have billions or even trillions of polygons in our renders. (Benson, 2023)*

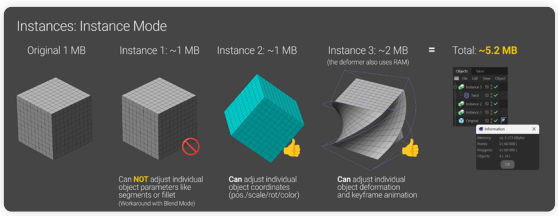


Figure 35. *(Benson, S. (2023). Memory usage and limitations of instances [image]. OTOY. https://help.otoy.com/hc/en-us/articles/13900681276571-Resource-Management-Instances)*

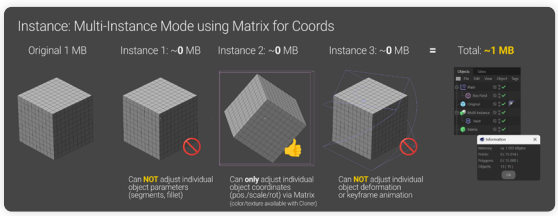


Figure 36. *(Benson, S. (2023). Memory usage and limitations of multi-instancing [image]. OTOY. https://help.otoy.com/hc/en-us/articles/13900681276571-Resource-Management-Instances)*

Furthermore, multi-instances do not support nested instancing (i.e., a multi-instance cannot instance a group that already contains a multi-instanced object inside). The following example serves to illustrate the concept of nested instances.

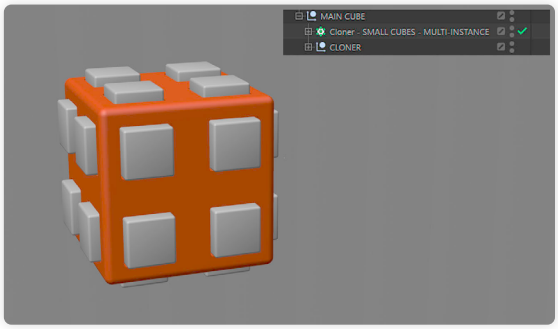


Figure 37. *Main cube*

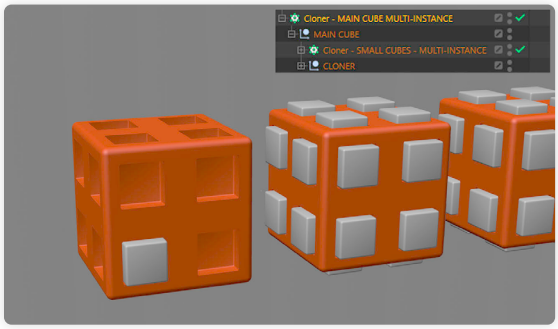


Figure 38. *Nested multi-instances*

Main cube:

Inside the group "MAIN CUBE" is a cloner instancing the smaller cubes in multi-instance mode, as presented in Figure 37.

Cloning the original cube with nested multi-instances:

When cloning the group "MAIN CUBE" in multi-instances mode (nested multi-instances), it generates problems on the original object, as presented in Figure 38.

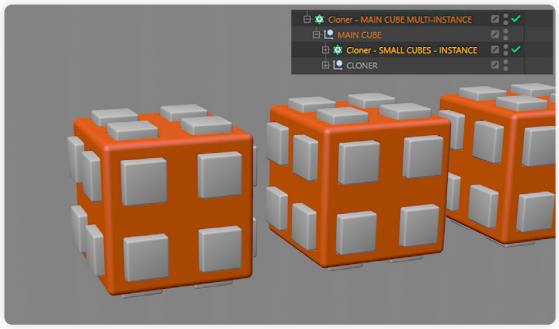


Figure 39. *Remove nested multi-instances*

Removing the nested multi-instances and cloning the cube:

Once the cloner instancing the smaller cubes is set to instance mode, the cloner multi-instancing the "MAIN CUBE" starts working properly, as presented in Figure 39.

Figure 40. *Analysed scene to compare performance of instance modes*

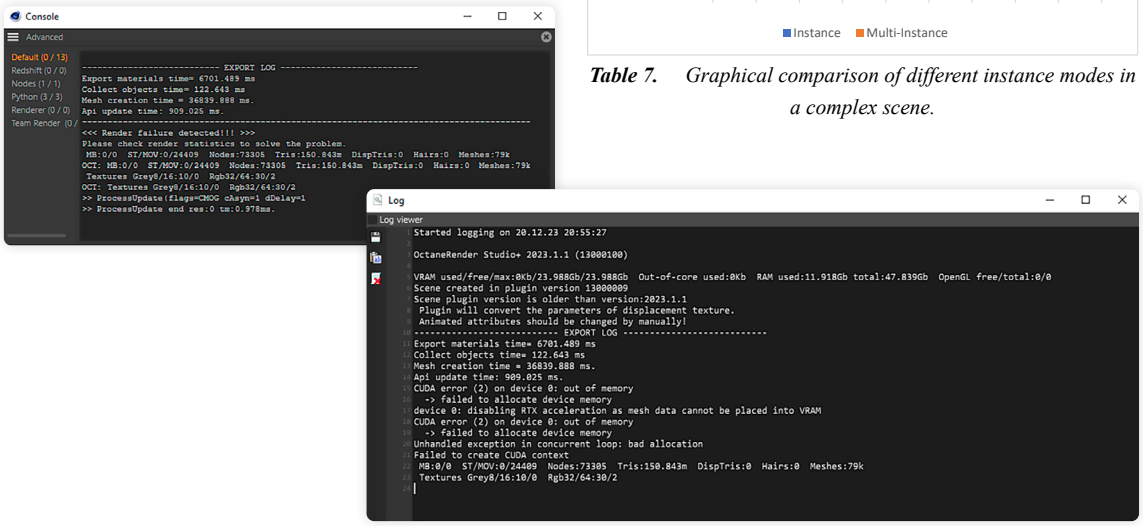
4.6.5.2 Performance Test with Instance Modes

The scene above will serve as an example for this topic. There are five different car models and one null object being cloned on the scene.

Since each normal instance has an impact on the VRAM usage, having a great number of detailed models copied through the scene will have a greater impact on memory.

When rendering the Parking Lot scene with the cloner object in Instance Mode, the VRAM usage went over the maximum available memory, making it impossible to render the scene and, as result, generating the crash report presented on Figure 41.

Figure 41. *Crash report - system out of memory*



However, when the cloner is set to multi-instance mode, there is a discernible difference in resource consumption, which enables the scene to be rendered. See Appendix A, Table 5 for the full logs related to this topic.

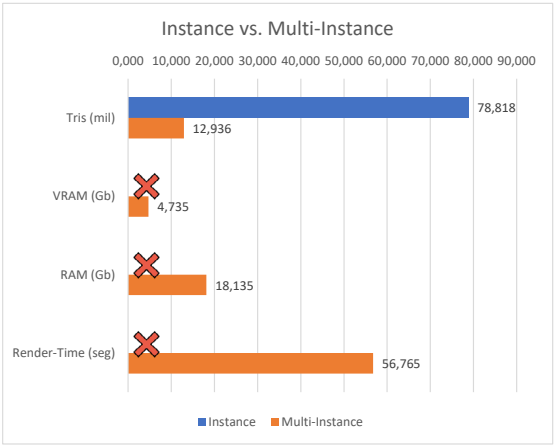


Table 7. *Graphical comparison of different instance modes in a complex scene.*

4.6.5.3 Adapted Scene for Analysis:

For the purpose of analysis, an adaptation of the original parking lot scene was prepared to compare the resource usage between instances and multi-instances in practice. The analysis draws on the render logs (refer to Appendix A, Table 6), and the necessary information was selected and transcribed into charts for a graphical comparison of performance results in each considered case.

- **Base for calculation: rendering without clone/ instances:**

Figure 42. *Instancing - Reference for calculations*

- **Cars in Instance Mode:**

Figure 43. *Cloner in instance-mode*

- **Cars in Multi-Instance Mode:**

Figure 44. *Cloner in multi-instance mode*

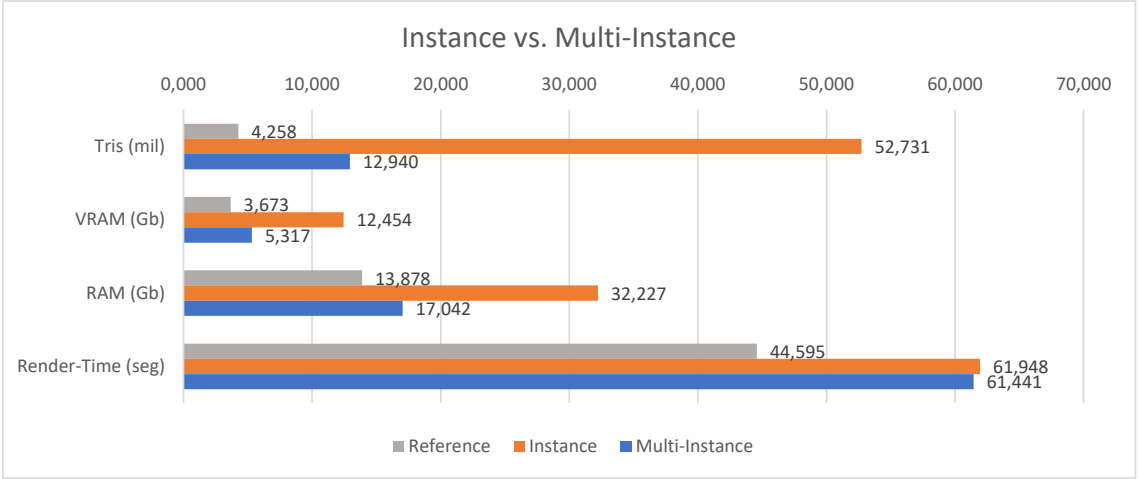


Table 8. *Graphical comparison of scene using different instance modes*

Rendering with normal instances had a major impact on the results, leading to a substantial increase in VRAM and RAM usage, with no significant changes in rendering time.

> ***Observation note:*** *The images rendered also present differences on the result of the shading between the different instance-mode types, but it does not come into question for this topic.*

- **Conclusion:**

Knowing the appropriate instance mode for each situation has a significant impact on scene performance, making it one of the most important techniques presented in this paper. Duplicating multiple high-quality models across a scene can rapidly increase memory consumption, leading to poor project performance, crashes, and bugs.

4.7 Aiming for Speed: Engine Settings

4.7.1 Adaptive Sampling

As an unbiased render engine, Octane samples every area of the image without bias. I.e. even if there are already noise free areas, the engine will continue sampling it to achieve a realist and physically accurate result.

Adaptive sampling in unbiased render engines expedites the rendering process by constraining light calculations based on the level of noise present in specific areas. This feature has become powerful when improving rendering time.

A noise threshold is established, and once this threshold is crossed in part of the images (controlled by the group pixel parameter: 2x2 or 4x4 pixels of area), the engine understands that this section is noise-free. Consequently, it ceases further sampling in that area, directing the available resources towards unfinished regions, thus accelerating the calculations. A lower threshold means a lower tolerance for noise in the analyzed area, resulting in less noise.

Not every scene requires a high number of samples to achieve a noise-free result. In such cases, consider adjusting the minimum samples (min. samples) in the adaptive sampling settings to a lower value. Otherwise, there might not be a significant impact on the final render time. The minimum sample setting dictates when the engine should begin assessing if the image already has noise-free areas based on the established threshold.

If areas of the image are already noise-free with just 16 casted samples, consider setting this value in the min. samples setting.

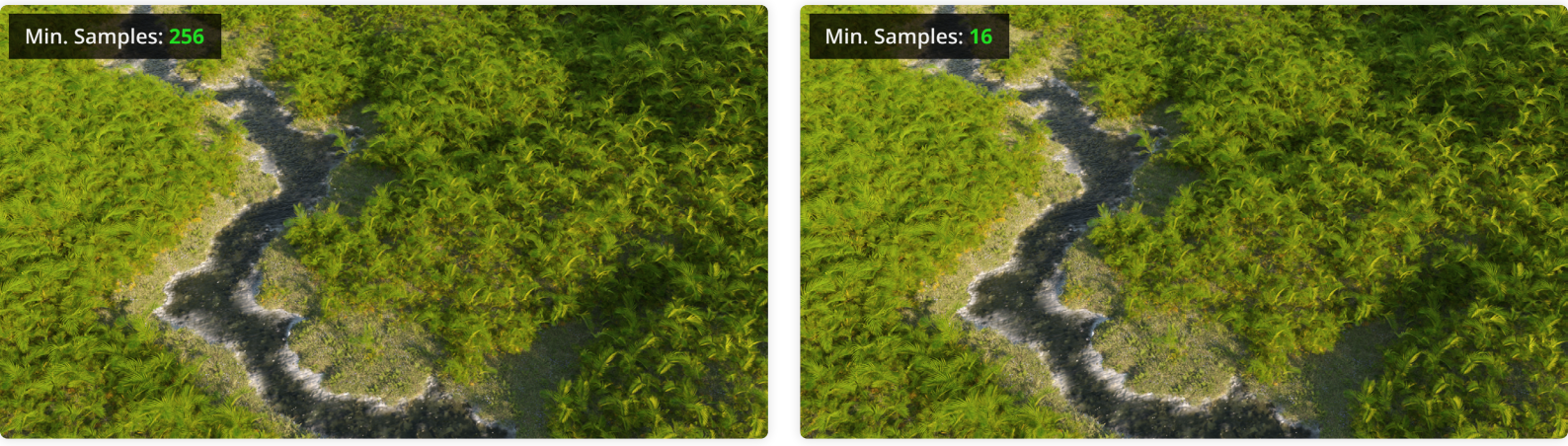


Figure 45. *Result comparison between different min. sample amount*

The Figure 45 was used as reference for the following analysis. The settings used were:

- 350 Samples
- Diffuse Depth: 4
- Specular Depth: 4
- GI Clamp: 1

Situation 01:	**Situation 02:**
Adaptive Sampling Active: • *Noise Threshold: 0.06* • *Min. Samples: 256 (standard value)* • *Expected Exposure: 1* • *Group Pixels: 2x2*	***Adaptive Sampling Active:*** • *Noise Threshold: 0.06* • *Min. Samples: 16* • *Expected Exposure: 1* • *Group Pixels: 2x2*

The standard value for the min. samples in the adaptive sampling setting is 256. However, in the scene presented in Figure 45, there were noise-free areas with 16 casted samples. This means that until casting 256 samples, every area of the image would be rendered without bias, thus casting constant samples on every area of the image, consequently spending more time rendering.

When adapting the setting to fit the correct value, a great reduction in render time was provided while remaining similar in terms of image noise. Furthermore, as presented on the render logs (see Appendix A, Table 7), there are no significant changes in hardware usage, only in render time.

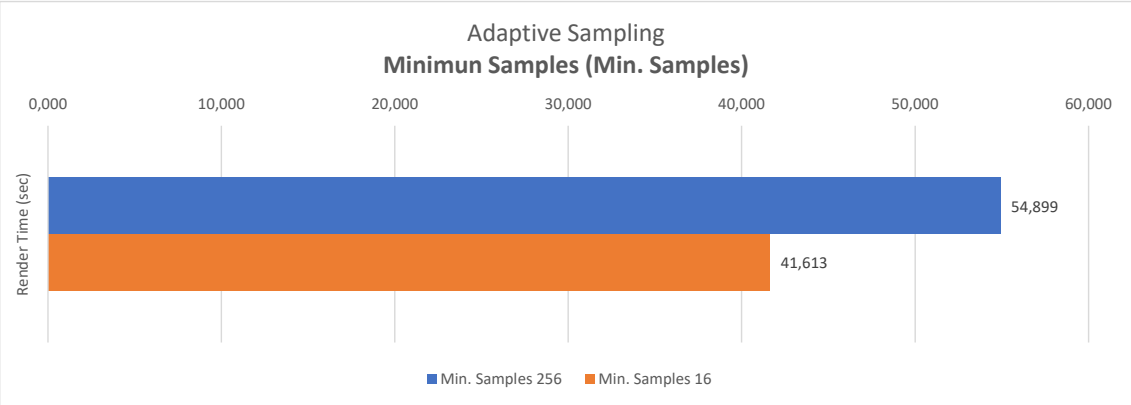


Table 9. *Graphical comparison of render time with different setting for min. samples (adaptive sampling)*

Figure 46. *Analysed scene to compare the performance with different setting of parallel samples*

4.7.2 Parallel Samples

Parallel samples can be configured to speed up rendering at the cost of VRAM usage or to reduce memory usage at the cost of render time. If memory is not the problem, consider increasing the parallel sample to get the best rendering speed.

Figure 46 was rendered with 32, 16, and 8 parallel samples, all having the exact same visual outcome but, according to the render logs (see Appendix A, Table 8), with a significant difference in render time and in VRAM and RAM consumption. Those changes were transcribed into charts, presenting a graphic visualization of the rendering performance:

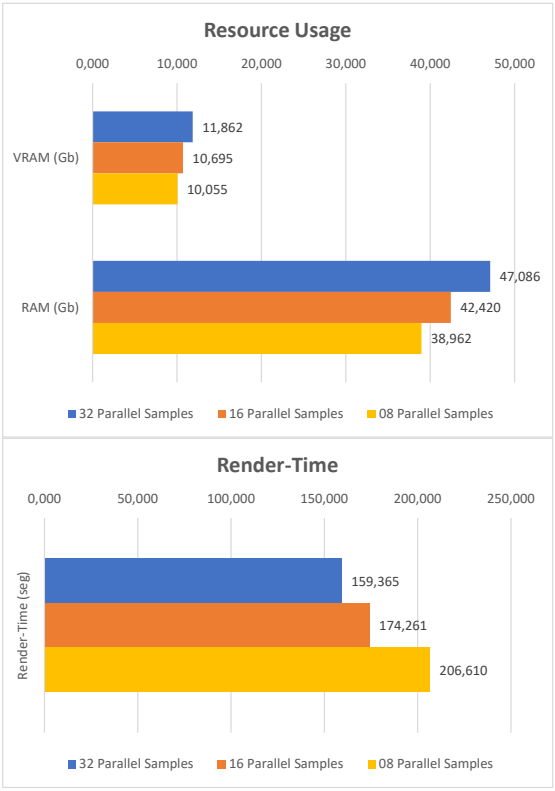


Table 10. *Graphical comparison of rendering performance with different setting of parallel samples*

5. Discussion

While GPU rendering is significantly faster than CPU rendering, it comes with its own limitations. The most crucial consideration is that the entire scene must be loaded into the available memory of the graphics card. Therefore, exceeding the VRAM on the GPU makes it impossible to render the scene using all the potential of the GPU, unless the hardware is upgraded with a graphic card with enough video memory to run the scene.

Taking into consideration the rapid pace of GPU advancements, constantly upgrading the hardware is cost-intensive and not always viable. As a result, not every artist involved in a project operates with identical system specifications, and GPUs with enormous amounts of VRAM are not always available. Therefore, effective resource management allows the project to run on a wider range of systems, cutting expenses while avoiding the need for constant upgrades. Additionally, stacking up lower-grade GPUs can yield impressive performance at a reduced cost, albeit constrained by their graphic memory.

The results presented in this study indicate that reducing memory consumption leads to improved software performance, as it reduces the need for data transfer between GPU and system memory, possibly resulting in faster rendering. In addition, it also avoids the risks of software delays and crashes, performance bottlenecks, low-speed processing or hardware overloads, thus speeding up the production process and equipping the artists with more time to further develop the project.

The process of optimizing a scene must become natural in the workflow of a 3D artist, who, after reading this paper, should have a better comprehension of which techniques to use or avoid during the multiple steps of production, endowing maximum performance on the project.

This study demonstrates a correlation between scene optimizations and the visual outcome. Being able to foresee the possible visual advantages or problems each technique can bring to the output image empowers the artist with faster and more concise work, reducing the need for constant trial-and-error or pointless renderings.

Each technique presented here is explained and assessed within distinct scenarios. Yet, complex scenes require all those techniques to be applied several times. Adding up the results of each optimization in a complex scene can lead to substantial improvements.

Multiple tools aimed at analyzing hardware consumption were introduced in this paper. Those must be taken into account during the optimization and troubleshooting processes since they provide the artist with invaluable insights on resource management, occasionally making it possible to avoid the massive work of going through every element of the scene searching for unclear ways to improve the performance of the project.

- **Frustum Culling**

This paper introduced a frustum culling system for Octane scatter implemented exclusively with native C4D tools, which unlocks potential for future advancements, including geometry culling and occlusion culling, to be explored in subsequent studies.

The performance test results proved that the native scatter system from Octane is highly optimized. Culling the scattered geometry did not provide a great reduction in VRAM usage; however, it provided improvements on scene update time per frame when rendering and on interface performance while the scatter system is active. A further advantage of this technique is the gradual reduction of VRAM usage according to the amount of scatter culled from the scene. That said, when a resource-intensive scatter system gets out of the FoV, it is deactivated from the calculations, further decreasing the VRAM consumption and improving scene performance.

The developed frustum culling system can be independently activated on each scatter system, providing the user with more control over the scene while allowing them to choose what to cull and what to keep in the render. Thus, when a single scatter system must remain active to avoid visual failures, it is possible to deactivate the culling effect only for this specific system.

Furthermore, the possibility of adding other static or animated fields that are not influenced by the camera movements makes it possible to cull further geometries from the scene, increasing the overall control of the system.

Frustum culling completely removes the scattered geometry from the scene and, therefore, can have major impacts on the final visuals. Analyzing if the final visual is still appropriate to the needs of the project is of major importance.

- **Bulk Texture Resize**

Bulk resizing images is commonly used by photographers when generating previews of their picture shots in RAW format. However, it is not commonly discussed in the 3D artist community, and it can bring great results in terms of memory savings for a project. Texture resolution and its compression method impact the file size,

consequentially impacting the VRAM consumption.

Using external assets is a common practice on 3D projects; however, the available textures in those assets are not always optimized for the project under development. High-quality assets offer high-quality textures; however, if the asset is not close to the camera, high resolution might not be necessary, resulting in useless information being loaded onto the graphic card. Therefore, being able to quickly adapt image settings to fit best for the project‘s needs is of major importance when optimizing memory consumption of a 3D scene, resulting in optimized resource usage.

- **VDB Topology Optimization**

VDB geometry requires special attention from the artist most of the time. Comprehending how those geometries impact hardware usage and how possible optimizations can affect the final visuals is of great importance. Volume builder, the VDB tool in C4D, has become increasingly popular among Cinema 4D users, whether for static or animated geometries.

When working with static geometry, it is easier to find ways to optimize it, and reducing the polycount would be a first appropriate step. For that, available features such as adaptive mesh, remesh, and polygon reduction are the logical choice.

However, GPU renderers can process polygons very quickly. Therefore, reducing the polycount of a mesh does not necessarily reduce the render time, but it does reduce VRAM consumption. However, it is important to acknowledge that doing it with the automated features of the software can introduce problems in the mesh being post-worked, resulting in loss of details, drastic topology changes, and, if not done properly, mesh failures, bringing undesirable changes to the overall shape of the model and affecting its visual on the output image. When the goal is to keep the most details in a VDB model, using the dense mesh is often the best way to go.

Yet there are major differences in the workflow when working with animated VDB meshes:

- Retopology is not an optimal solution.
- Changing the dense mesh often introduces glitches.

An animated and highly detailed VDB simulation is often considered to be one of the main elements in a scene; otherwise, the simulated geometry itself would be in lower resolution, which consequently results in a less dense mesh. Helpful for that is having a cached animation.

The results presented in this study indicate that trying to reduce memory consumption while modifying the topology of highly detailed VDB meshes can introduce failures in important visual characteristics of that element. Reflections, refractions, and fine details of the geometry will be affected, producing glitches, wrong light reactions, and diminishing the quality of the specific element in the final image. If there is still a need to reduce the polycount, consider reducing the resolution of the VDB simulation itself.

The best choice when working with highly detailed VDB meshes is to use the original mesh while coping with its cons, therefore optimizing other elements of the scene.

- **Instances**

The proper use of instances in a scene is one of the most important techniques presented in this paper. It is also important to apply this technique from the beginning of the project, during the construction steps. This will provide the artist with higher software performance and more control over multiple elements copied through the scene. Choosing the correct instance mode can also avoid overloads on hardware, system crashes, or slow processing of the 3D scene.

The analysis of the instance modes proved to be of great importance. Even with 24GB of VRAM available, it was not possible to render the example scene when using the wrong instance type in the cloner object. However, when properly set up, it proved that the resources were just being falsely allocated and that not much memory was needed to achieve the desired result.

Since instances can have a major impact on VRAM, learning how to cope with the limitations of each instance mode is an advantage for a 3D artist.

- **Adaptive Sampling**

The results indicate that adaptive sampling has no impact on hardware usage, only on rendering time. This setting is aimed at the rendering process and not the 3D scene itself; however, it speeds up the interactive preview, allowing the artists to make faster decisions when working on lights, shaders, and render.

- **Parallel Samples**

Parallel samples proved to be a powerful setting of the render engine, but since it comes at the cost of VRAM and RAM usage, it is directly related to the amount of memory available; therefore, better-performing GPUs have better chances of taking advantage of this feature to speed up the rendering process.

Besides potentially accelerating the rendering process, it can also serve as a lifesaver when the available memory is insufficient for rendering the scene. Decreasing the number of parallel samples may extend the rendering time, but it could decrease memory consumption to a level supported by the graphic card. Having enough VRAM to take advantage of this setting brings a great reduction in the rendering time per frame.

Better-performing GPUs come equipped with a larger amount of memory; consequently, in addition to the higher speed, they are better suited to take advantage of this feature, reducing the render time even more.

Due to the lack of data from GPUs with less memory, the results cannot confirm if this setting can provide system instabilities. While running the tests on the RTX 4090, there were no crashes or problems, even when increasing the parallel samples to maximum. According to the render logs, almost all the available RAM was used; therefore, the rendering speed might also have been limited. Nonetheless providing a great improvement in render time and no software crashes.

6. Conclusion

This quantitative study sheds light on optimization techniques for 3D scenes aimed at better allocating GPU resources for faster rendering and workflow, a smoother software experience, and better performance. Furthermore, analyzes the possible impacts each technique can have on the final visual outcome of the project.

The significance of the findings presented here lies not only in the specific context of Octane for Cinema 4D but also in a broader applicability to scene optimizations and GPU rendering. Techniques not commonly disseminated through the artist community, such as bulk resizing textures and frustum culling Octane scatter systems with native C4D tools were presented, analyzed, and explained.

By a meticulous examination of the rendering pipeline while using Octane Render for Cinema 4D, we identified strategies and techniques to address challenges related to memory constraints, rendering speed, and overall system performance. Advancements in technology inherently tie the pursuit of efficiency in GPU rendering, and this work contributes to shaping this evolving landscape. The findings here presented underscore the importance of continuous development in rendering algorithms, GPU architectures, and software optimizations to meet the evolving demands of the creative and technical industries.

As moving forward, the lessons learned from this case study serves as a catalyst for further exploration, innovation, and the continual improvement of GPU rendering methodologies.

In the ever-evolving world of rendering, render engine developers are constantly coming up with new features to improve results and speed up the rendering process. Understanding the theoretical concepts of rendering helps the creators and artists keep up to date with the technology, finding faster solutions for 3D scene problems, and making informed decisions to achieve better results in a more efficient way.

Understanding resource management is a crucial skill for 3D artists. Having an optimized scene since the beginning of a project ensures software stability and optimal hardware performance, resulting in a smoother software experience and avoiding crashes and system bottlenecks. When the objective is time efficiency, these enhancements accelerate the workflow, affording artists more time to refine their work.

The exploration of enhancing rendering performance has become increasingly pertinent in the context of ever-expanding demands for high-quality visuals, especially in industries such as film, advertisement, animation, and architectural visualization.

It became evident during the development of the paper that a nuanced approach to scene optimization is crucial for balancing the trade-offs between speed and image quality. The results of this research underscore the importance of continuous innovation in rendering algorithms and software optimizations to meet the evolving demands of the creative and technical industries.

7. Asset List

Cinema 4D Asset Browser

- Hot Hatch - **Model by Dosch Design: Part of the collection Dosch 3D: Concept Cars 2011**
- Sedan - **Model by Dosch Design: Part of the collection Dosch 3D: Concept Cars 2011**
- SUV Small - **Model by Dosch Design: Part of the collection Dosch 3D: Concept Cars 2011**
- Van - **Original C4D Asset**

Quixel Megascans: Models

- Areca Palm – **shFjC**
- Birch Tree Trunk - **thxvdfjfa**
- Brick Debri on Ground – **vbnjcbdfw**
- Common Fern - **rhDso**
- Green Herb - **tbbpaier**
- Huge Mossy Forest Cliff - **vktudjsaw**
- Lady Fern - **wdvlditia**
- Mossy Embankment - **titfbczfa**
- Mossy Forest Bouder - **wfstcdnaw**
- Mossy Wooden Log - **rhfdj**
- Nature Rock M - **wdbncbl**:
- Nordic Forest Rock Moss - **xetlaff**
- Rotten Tree Stump - **wjzueccs**
- Rusty Metal Barrel - **tewscfuda**
- Rusty Metal Barrel - **vijnbjz**
- Sidewalk Vegetation - **rmskb**
- Small Stones Pack - **vdjkbfmiw**
- Wild Grass – **vlkhcbxia**
- Wooden Pallet - **vh1icei**
- Yellow Archangel – **wcwmcfvia**

Quixel Megascans: Surfaces

- Mossy Ground – **ukimchjew**
- Mossy Ground - **vjoefgo**
- Mossy Wild Grass - **vemnae1s**

Poly Haven:

- Alps Field - **HDRI by Mischok, Andreas -** *https://polyhaven.com/a/alps_field*
- Evening Road 01 - **HDRI by Guest, Jarod & Majboroda, Sergej -***https://polyhaven.com/a/evening_road_01_puresky*
- Kiara 6 Afternoon - **HDRI by Zaal, Greg** - *https://polyhaven.com/a/kiara_6_afternoon*
- Kloofendal Misty Morning Puresky - **HDRI by Zaal, Greg -***https://polyhaven.com/a/kloofendal_misty_morning_puresky*

Florian Renaux:

Water Heightmap - heightmap2_water - *https://www.artstation.com/marketplace/p/voRqV/water-height-map*

BBC Sound Library:

Death‘s Head Hawk Moth (Acherontia Atropos) - **NHU05078054**
Water - **NHU05013029**
Wind Atmosphere - **NHU10217728**

8. References

Angle of View Calculator. (n.D). Calculator Academy. https://calculator.academy/angle-of-view-calculator/

Benson, S. (2023, March 16). *Resource Management—Instances*. Help | OTOY. https://help.otoy.com/hc/en-us/articles/13900681276571-Resource-Management-Instances

Florenaux. (n.d.). *Home* [Youtube Channel]. Youtube. https://www.youtube.com/@florenaux/

Mathematical Functions Manual (Classic): Cinema 4D C++ SDK. (n.d.). Maxon Developers. https://developers.maxon.net/docs/Cinema4DCPPSDK/html/page_manual_maths.html

Poly Haven. (n.d). Poly Haven. https://polyhaven.com/

Quixel Megascans. (n.d.). Quixel. https://quixel.com/megascans

Renaux, F. (n.d.). *Water HeightMap | Artworks*. ArtStation. https://www.artstation.com/marketplace/p/voRqV/water-heightmap

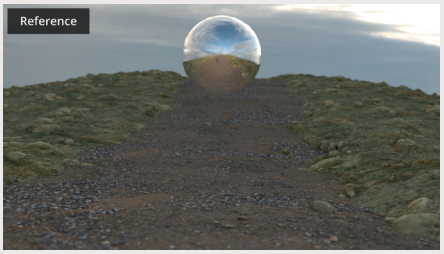
SilverwingVFX. (n.d.) *Home* [Youtube Channel]. Youtube. https://www.youtube.com/@SilverwingVFX

Vertical FOV Calculator. (n.d). Calculator Academy. https://calculator.academy/vertical-fov-calculator/

9. Appendix A. Render Logs

This appendix consists of the full Octane render logs for each rendered image or frame, without filtering for analysis.

Table 1. *Full render logs of the analysed situations on a small-scale environment*



Reference Frame

```
FRAME:120 fps:24
 MB:0/0  ST/MOV:0/35  Nodes:91  Tris:123k  Disp-
Tris:825k  Hairs:0  Meshes:35
 Textures Grey8/16:17/0  Rgb32/64:21/1
VRAM used/free/max:3.84Gb/13.559Gb/23.988Gb  Out-
of-core used:0Kb  RAM used:17.765Gb total:47.839Gb
OpenGL free/total:0/0
mblurTM=0sec.  createTM=0.002sec.  updateTM=0.209sec.
renderTM:18.231sec.   totalTM:18.443sec.
Tonemapping the all passes tm:41.638
Displaying passes in tm=28.434
Passes saved in:0.752sec.
```

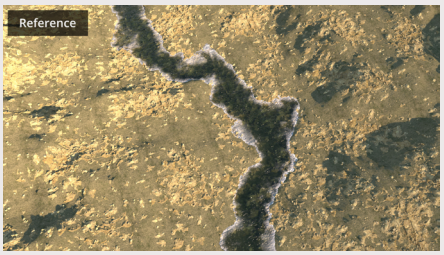
Frustum Culling Active

```
FRAME:120 fps:24
 MB:0/0  ST/MOV:0/35  Nodes:96  Tris:245k  Disp-
Tris:825k  Hairs:0  Meshes:44k
 Textures Grey8/16:17/0  Rgb32/64:21/1
VRAM used/free/max:3.883Gb/13.513Gb/23.988Gb  Out-
of-core used:0Kb  RAM used:18.343Gb total:47.839Gb
OpenGL free/total:0/0
mblurTM=0sec.  createTM=0.007sec.  updateTM=0.604sec.
renderTM:36.56sec.   totalTM:37.172sec.
Tonemapping the all passes tm:39.765
Displaying passes in tm=29.505
Passes saved in:0.841sec.
```

Frustum Culling Off

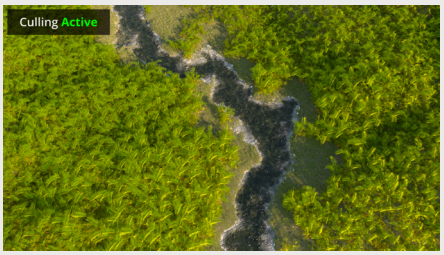
```
FRAME:120 fps:24
 MB:0/0  ST/MOV:0/35  Nodes:96  Tris:245k  Disp-
Tris:825k  Hairs:0  Meshes:165k
 Textures Grey8/16:17/0  Rgb32/64:21/1
VRAM used/free/max:3.94Gb/13.456Gb/23.988Gb  Out-
of-core used:0Kb  RAM used:18.503Gb total:47.839Gb
OpenGL free/total:0/0
mblurTM=0sec.  createTM=0.004sec.  updateTM=0.65sec.
renderTM:36.065sec.   totalTM:36.72sec.
Tonemapping the all passes tm:40.036
Displaying passes in tm=29.814
Passes saved in:0.835sec.
```

Table 2. *C4D render logs of the analysed situations on a large-scale environment*



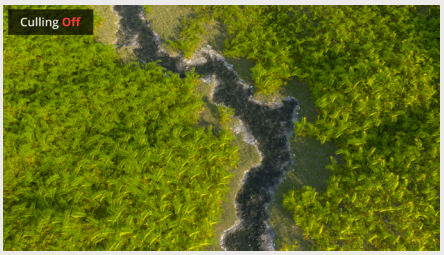
Reference Frame

```
FRAME:100 fps:24
 MB:0/0  ST/MOV:0/36  Nodes:91  Tris:132k  Disp-
Tris:160k  Hairs:0  Meshes:37
 Textures Grey8/16:18/0  Rgb32/64:21/0
VRAM used/free/max:2.882Gb/14.521Gb/23.988Gb  Out-
of-core used:0Kb  RAM used:14.237Gb total:47.839Gb
OpenGL free/total:0/0
mblurTM=0sec.  createTM=0.003sec.  updateTM=0.251sec.
renderTM:12.896sec.   totalTM:13.152sec.
Tonemapping the all passes tm:41.348
Displaying passes in tm=0.018
Passes saved in:0.925sec.
```



Frustum Culling Active

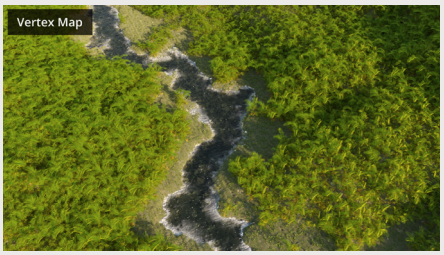
```
FRAME:100 fps:24
 MB:0/0  ST/MOV:0/36  Nodes:97  Tris:265k  Disp-
Tris:160k  Hairs:0  Meshes:118k
 Textures Grey8/16:18/0  Rgb32/64:21/0
VRAM used/free/max:2.961Gb/14.287Gb/23.988Gb  Out-
of-core used:0Kb  RAM used:19.602Gb total:47.839Gb
OpenGL free/total:0/0
mblurTM=0sec.  createTM=0.014sec.  updateTM=0.725sec.
renderTM:42.221sec.   totalTM:42.961sec.
Tonemapping the all passes tm:41.629
Displaying passes in tm=29.539
Passes saved in:0.992sec.
```



Frustum Culling Off

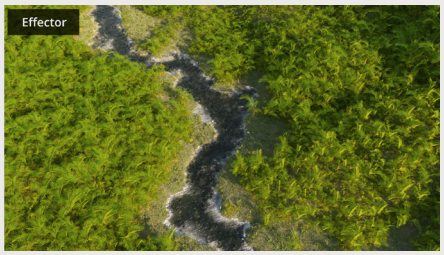
```
FRAME:100 fps:24
 MB:0/0  ST/MOV:0/36  Nodes:97  Tris:265k  Disp-
Tris:160k  Hairs:0  Meshes:469k
 Textures Grey8/16:18/0  Rgb32/64:21/0
VRAM used/free/max:3.1Gb/13.904Gb/23.988Gb  Out-
of-core used:0Kb  RAM used:15.581Gb total:47.839Gb
OpenGL free/total:0/0
mblurTM=0sec.  createTM=0.02sec.  updateTM=2.086sec.
renderTM:41.399sec.   totalTM:43.508sec.
Tonemapping the all passes tm:39.81
Displaying passes in tm=30.147
Passes saved in:1.036sec.
```

Table 3. *C4D render logs: comparing both frustum culling methods*



Culling the Vertex Map

```
FRAME:100 fps:24
 MB:0/0  ST/MOV:0/36  Nodes:91  Tris:132k  Disp-
Tris:160k  Hairs:0  Meshes:37
 Textures Grey8/16:18/0  Rgb32/64:21/0
VRAM used/free/max:2.882Gb/14.521Gb/23.988Gb  Out-
of-core used:0Kb  RAM used:14.237Gb total:47.839Gb
OpenGL free/total:0/0
mblurTM=0sec.  createTM=0.003sec.  updateTM=0.251sec.
renderTM:12.896sec.   totalTM:13.152sec.
Tonemapping the all passes tm:41.348
Displaying passes in tm=0.018
Passes saved in:0.925sec.
```



Culling with Plain Effector

```
FRAME:100 fps:24
 MB:0/0  ST/MOV:0/36  Nodes:97  Tris:265k  Disp-
Tris:160k  Hairs:0  Meshes:118k
 Textures Grey8/16:18/0  Rgb32/64:21/0
VRAM used/free/max:2.961Gb/14.287Gb/23.988Gb  Out-
of-core used:0Kb  RAM used:19.602Gb total:47.839Gb
OpenGL free/total:0/0
mblurTM=0sec.  createTM=0.014sec.  updateTM=0.725sec.
renderTM:42.221sec.   totalTM:42.961sec.
Tonemapping the all passes tm:41.629
Displaying passes in tm=29.539
Passes saved in:0.992sec.
```

Table 4. *C4D render logs: Comparing rendering performance for each VDB geometry optimization*

Original Dense VDB Geometry

```
Export materials time= 146.555 ms
Collect objects time= 3.318 ms
Mesh creation time = 646.32 ms.
 MB:0/0  ST/MOV:0/3  Nodes:43  Tris:1.285m  DispTris:0  Hairs:0  Meshes:6
 Textures Grey8/16:0/0  Rgb32/64:4/4
VRAM used/free/max:2.161Gb/15.987Gb/23.988Gb  Out-of-core used:0Kb  RAM used:15.357Gb to-
tal:47.839Gb  OpenGL free/total:0/0
mblurTM=0sec.  createTM=0sec.  updateTM=0sec. renderTM:47.872sec.   totalTM:52.62sec.
Tonemapping the all passes tm:299.8
Displaying passes in tm=447.667
Passes saved in:8.799sec.
```

Adapative Geometry

```
Export materials time= 201.783 ms
Collect objects time= 3.504 ms
Mesh creation time = 565.262 ms.
 MB:0/0  ST/MOV:0/3  Nodes:43  Tris:461k  DispTris:0  Hairs:0  Meshes:6
 Textures Grey8/16:0/0  Rgb32/64:4/4
VRAM used/free/max:2.034Gb/16.141Gb/23.988Gb  Out-of-core used:0Kb  RAM used:15.288Gb to-
tal:47.839Gb  OpenGL free/total:0/0
mblurTM=0sec.  createTM=0sec.  updateTM=0sec. renderTM:47.234sec.   totalTM:51.402sec.
Tonemapping the all passes tm:305.167
Displaying passes in tm=449.032
Passes saved in:8.898sec.
```

Remeshed Geometry

```
Export materials time= 181.454 ms
Collect objects time= 3.467 ms
Mesh creation time = 599.819 ms.
 MB:0/0  ST/MOV:0/3  Nodes:43  Tris:218k  DispTris:0  Hairs:0  Meshes:6
 Textures Grey8/16:0/0  Rgb32/64:4/4
VRAM used/free/max:2.002Gb/16.159Gb/23.988Gb  Out-of-core used:0Kb  RAM used:15.244Gb to-
tal:47.839Gb  OpenGL free/total:0/0
mblurTM=0sec.  createTM=0sec.  updateTM=0sec. renderTM:47.186sec.   totalTM:51.12sec.
Tonemapping the all passes tm:294.1
Displaying passes in tm=448.498
Passes saved in:8.971sec.
```

Remeshed Geometry + Original Dense VDB Fluid

```
Export materials time= 173.207 ms
Collect objects time= 4.106 ms
Mesh creation time = 713.042 ms.
 MB:0/0  ST/MOV:0/5  Nodes:49  Tris:1.742m  DispTris:0  Hairs:0  Meshes:8
 Textures Grey8/16:0/0  Rgb32/64:4/4
VRAM used/free/max:2.264Gb/15.891Gb/23.988Gb  Out-of-core used:0Kb  RAM used:17.585Gb to-
tal:47.839Gb  OpenGL free/total:0/0
mblurTM=0sec.  createTM=0sec.  updateTM=0sec. renderTM:122.013sec.   totalTM:127.198sec.
Tonemapping the all passes tm:292.81
Displaying passes in tm=451.127
Passes saved in:10.49sec.
```

Remeshed Geometry + Adaptive Fluid

```
Export materials time= 188.882 ms
Collect objects time= 4.067 ms
Mesh creation time = 730.967 ms.
 MB:0/0  ST/MOV:0/5  Nodes:49  Tris:1.633m  DispTris:0  Hairs:0  Meshes:8
 Textures Grey8/16:0/0  Rgb32/64:4/4
VRAM used/free/max:2.245Gb/15.936Gb/23.988Gb  Out-of-core used:0Kb  RAM used:16.839Gb to-
tal:47.839Gb  OpenGL free/total:0/0
mblurTM=0sec.  createTM=0sec.  updateTM=0sec. renderTM:121.188sec.   totalTM:126.283sec.
Tonemapping the all passes tm:354.364
Displaying passes in tm=449.506
Passes saved in:10.518sec.
```

Remeshed Geometry + Remeshed Fluid

```
Export materials time= 187.215 ms
Collect objects time= 4.594 ms
Mesh creation time = 628.254 ms.
 MB:0/0  ST/MOV:0/5  Nodes:49  Tris:725k  DispTris:0  Hairs:0  Meshes:8
 Textures Grey8/16:0/0  Rgb32/64:4/4
VRAM used/free/max:2.086Gb/16.095Gb/23.988Gb  Out-of-core used:0Kb  RAM used:16.806Gb to-
tal:47.839Gb  OpenGL free/total:0/0
mblurTM=0sec.  createTM=0sec.  updateTM=0sec. renderTM:111.483sec.   totalTM:115.585sec.
Tonemapping the all passes tm:298.325
Displaying passes in tm=448.697
Passes saved in:10.629sec.
```

Optimal Choise: Remeshed Geometry + Original Dense VDB Fluid

```
Export materials time= 173.207 ms
Collect objects time= 4.106 ms
Mesh creation time = 713.042 ms.
 MB:0/0  ST/MOV:0/5  Nodes:49  Tris:1.742m  DispTris:0  Hairs:0  Meshes:8
 Textures Grey8/16:0/0  Rgb32/64:4/4
VRAM used/free/max:2.264Gb/15.891Gb/23.988Gb  Out-of-core used:0Kb  RAM used:17.585Gb to-
tal:47.839Gb  OpenGL free/total:0/0
mblurTM=0sec.  createTM=0sec.  updateTM=0sec. renderTM:122.013sec.   totalTM:127.198sec.
Tonemapping the all passes tm:292.81
Displaying passes in tm=451.127
Passes saved in:10.49sec.
```

Original Dense VDB Meshes

```
Export materials time= 146.314 ms
Collect objects time= 3.305 ms
Mesh creation time = 870.614 ms.
 MB:0/0  ST/MOV:0/5  Nodes:49  Tris:2.808m  DispTris:0  Hairs:0  Meshes:8
 Textures Grey8/16:0/0  Rgb32/64:4/4
VRAM used/free/max:2.456Gb/15.727Gb/23.988Gb  Out-of-core used:0Kb  RAM used:19.137Gb to-
tal:47.839Gb  OpenGL free/total:0/0
mblurTM=0sec.  createTM=0sec.  updateTM=0sec. renderTM:123.307sec.   totalTM:128.754sec.
Tonemapping the all passes tm:298.391
Displaying passes in tm=448.936
Passes saved in:10.381sec.
```

Table 5. *C4D render logs: Comparing rendering performance of instances and multi-instance in the original parking lot scene*

Original Parking Lot: Cloner in Instance Mode

```
Please check render statistics to solve the problem.
 MB:0/0  ST/MOV:0/12389  Nodes:37246  Tris:78.818m  DispTris:0  Hairs:0  Meshes:79k
OCT: MB:0/0  ST/MOV:0/12389  Nodes:37246  Tris:78.818m  DispTris:0  Hairs:0  Meshes:79k
 Textures Grey8/16:10/0  Rgb32/64:30/2
OCT: Textures Grey8/16:10/0  Rgb32/64:30/2
VRAM used/free/max:16.549Gb/0Kb/23.988Gb
OCT:VRAM used/free/max:16.549Gb/0Kb/23.988Gb
Free VRAM is too low!
Try to decrease polygon counts and use out-of-core for textures.
Use ‚render instances‘ when it‘s possible.‘
OCT:Free VRAM is too low!
Try to decrease polygon counts and use out-of-core for textures.
Use ‚render instances‘ when it‘s possible.‘
```

Original Parking Lot: Cloner in Multi-Instance Mode

```
Export materials time= 2875.509 ms
Collect objects time= 21.419 ms
Mesh creation time = 2494.3 ms.
 MB:0/0  ST/MOV:0/1109  Nodes:3403  Tris:12.936m  DispTris:0  Hairs:0  Meshes:79k
 Textures Grey8/16:10/0  Rgb32/64:30/2
VRAM used/free/max:4.735Gb/12.825Gb/23.988Gb  Out-of-core used:0Kb  RAM used:18.135Gb to-
tal:47.839Gb  OpenGL free/total:0/0
mblurTM=0sec.  createTM=0sec.  updateTM=0sec. renderTM:56.765sec.   totalTM:83.999sec.
Tonemapping the all passes tm:286.555
Displaying passes in tm=298.347
Passes saved in:7.068sec.
```

Table 6. *C4D render logs: Comparing rendering performance of instances and multi-instance in the adapted parking lot scene*

Adapted Parking Lot: Base for calculation

```
FRAME:1 fps:24
 MB:0/0  ST/MOV:0/196  Nodes:674  Tris:4.258m  DispTris:0  Hairs:0  Meshes:55k
 Textures Grey8/16:9/0  Rgb32/64:12/1
VRAM used/free/max:3.673Gb/13.887Gb/23.988Gb  Out-of-core used:0Kb  RAM used:13.878Gb to-
tal:47.839Gb  OpenGL free/total:0/0
mblurTM=0sec.  createTM=0.092sec.  updateTM=0.421sec. renderTM:44.595sec.   totalTM:45.108sec.
Tonemapping the all passes tm:384.94
Displaying passes in tm=442.605
Passes saved in:5.813sec.
```

Adapted Parking Lot: Cloner in Instance Mode

```
FRAME:1 fps:24
 MB:0/0  ST/MOV:0/8296  Nodes:24974  Tris:52.731m  DispTris:0  Hairs:0  Meshes:63k
 Textures Grey8/16:9/0  Rgb32/64:29/2
VRAM used/free/max:12.454Gb/3.578Gb/23.988Gb  Out-of-core used:0Kb  RAM used:32.227Gb to-
tal:47.839Gb  OpenGL free/total:0/0
mblurTM=0sec.  createTM=0.303sec.  updateTM=0.624sec. renderTM:61.948sec.   totalTM:60.262sec.
Tonemapping the all passes tm:381.46
Displaying passes in tm=449.206
Passes saved in:6.116sec.
```

Adapted Parking Lot: Cloner in Multi-Instance Mode

```
FRAME:1 fps:24
 MB:0/0  ST/MOV:0/899  Nodes:2785  Tris:12.94m  DispTris:0  Hairs:0  Meshes:63k
 Textures Grey8/16:9/0  Rgb32/64:29/2
VRAM used/free/max:5.317Gb/12.004Gb/23.988Gb  Out-of-core used:0Kb  RAM used:17.042Gb to-
tal:47.839Gb  OpenGL free/total:0/0
mblurTM=0sec.  createTM=0.122sec.  updateTM=0.667sec. renderTM:61.441sec.   totalTM:62.231sec.
Tonemapping the all passes tm:382.708
Displaying passes in tm=446.399
Passes saved in:5.983sec.
```

Table 7. *C4D render logs: Impact of min. samples of the adaptive sampling on render time*

Adaptive Sampling: 256 Min. Samples

```
FRAME:45 fps:24
 MB:0/0  ST/MOV:0/36  Nodes:97  Tris:265k  DispTris:160k  Hairs:0  Meshes:164k
 Textures Grey8/16:15/0  Rgb32/64:18/0
VRAM used/free/max:2.925Gb/14.815Gb/23.988Gb  Out-of-core used:0Kb  RAM used:17.306Gb to-
tal:47.839Gb  OpenGL free/total:0/0
mblurTM=0sec.  createTM=0.015sec.  updateTM=1.199sec. renderTM:54.899sec.   totalTM:56.115sec.
Tonemapping the all passes tm:41.718
Displaying passes in tm=28.528
Passes saved in:1.002sec.
```

Adaptive Sampling: 16 Min. Samples

```
FRAME:45 fps:24
 MB:0/0  ST/MOV:0/36  Nodes:97  Tris:265k  DispTris:160k  Hairs:0  Meshes:164k
 Textures Grey8/16:15/0  Rgb32/64:18/0
VRAM used/free/max:2.925Gb/14.812Gb/23.988Gb  Out-of-core used:0Kb  RAM used:17.347Gb to-
tal:47.839Gb  OpenGL free/total:0/0
mblurTM=0sec.  createTM=0.014sec.  updateTM=1.131sec. renderTM:41.613sec.   totalTM:42.76sec.
Tonemapping the all passes tm:42.524
Displaying passes in tm=29.439
Passes saved in:1.003sec.
```

Table 8. *C4D render logs: Impact of Parallel Samples setting on render time*

32 Parallel Samples

```
Export materials time= 29647.054 ms
Collect objects time= 8.38 ms
Mesh creation time = 4839.058 ms.
 MB:0/0  ST/MOV:0/619  Nodes:1374  Tris:31.642m  DispTris:3.62m  Hairs:0  Meshes:271k
 Textures Grey8/16:43/0  Rgb32/64:76/2
VRAM used/free/max:11.862Gb/4.369Gb/23.988Gb  Out-of-core used:0Kb  RAM used:47.086Gb to-
tal:47.839Gb  OpenGL free/total:0/0
mblurTM=0sec.  createTM=0sec.  updateTM=0sec. renderTM:159.365sec.   totalTM:242.075sec.
Tonemapping the all passes tm:423.68
Displaying passes in tm=542.833
Passes saved in:12.459sec.
```

16 Parallel Samples

```
Export materials time= 29957.576 ms
Collect objects time= 8.989 ms
Mesh creation time = 4844.885 ms.
 MB:0/0  ST/MOV:0/619  Nodes:1374  Tris:31.642m  DispTris:3.62m  Hairs:0  Meshes:271k
 Textures Grey8/16:43/0  Rgb32/64:76/2
VRAM used/free/max:10.695Gb/6.453Gb/23.988Gb  Out-of-core used:0Kb  RAM used:42.42Gb to-
tal:47.839Gb  OpenGL free/total:0/0
mblurTM=0sec.  createTM=0sec.  updateTM=0sec. renderTM:174.261sec.   totalTM:241.64sec.
Tonemapping the all passes tm:417.085
Displaying passes in tm=537.048
Passes saved in:12.277sec.
```

08 Parallel Samples

```
Export materials time= 29874.547 ms
Collect objects time= 7.407 ms
Mesh creation time = 4960.322 ms.
 MB:0/0  ST/MOV:0/619  Nodes:1374  Tris:31.642m  DispTris:3.62m  Hairs:0  Meshes:271k
 Textures Grey8/16:43/0  Rgb32/64:76/2
VRAM used/free/max:10.055Gb/7.162Gb/23.988Gb  Out-of-core used:0Kb  RAM used:38.962Gb to-
tal:47.839Gb  OpenGL free/total:0/0
mblurTM=0sec.  createTM=0sec.  updateTM=0sec. renderTM:206.61sec.   totalTM:266.141sec.
Tonemapping the all passes tm:431.52
Displaying passes in tm=539.25
Passes saved in:12.389sec.
```

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit mit dem Titel:

Enhancing GPU Rendering Efficiency through Scene Optimizations, a Case Study using Octane for Cinema 4D

Selbstständig, ohne unerlaubte fremde Hilfe verfasst und keine anderen als die angegebenen Hilfsmittel benutzt habe. Die Stellen der Arbeit, die dem Wortlaut oder dem Sinn nach anderen Werken (dazu zählen auch Internetquellen) entnommen sind, wurden unter Angabe der Quelle kenntlich gemacht.

Lemgo, 05.01.2024
Ort, Datum


Unterschrift