Institute for Visualization and Interactive Systems

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Bachelorarbeit

Energy-Efficient Visualization using MegaMol

Tim Schmidt

Course of Study:	Softwaretechnik
Examiner:	Prof. Dr. Daniel Weiskopf
Supervisor:	M. Sc. Patrick Gralka, M. Sc. Moritz Heinemann, Dr. Guido Reina, Dipl.-Inf. Christoph Müller
Commenced:	April 18, 2023
Completed:	October 18, 2023

Zusammenfassung

Aufgrund von steigendem Umweltbewusstsein und höheren Lebenshaltungskosten wird die Energieeffizienz in den meisten technologischen Sektoren, einschließlich bei Computern, zu einem immer entscheidenderen Faktor. Bei Grafikprozessoren (GPUs), die ohnehin bereits zu den stromhungrigsten Komponenten in Computern gehören, geht der Trend in letzter Zeit jedoch dahin, eine höhere maximale Leistungsaufnahme zuzulassen, als je zuvor . Es gibt zwar Forschungsarbeiten zur Energieeffizienz von GPGPU-Anwendungen, die sogar teils so weit gehen, den Stromverbrauch bis hinunter auf die Befehlsebene zu modellieren, aber es wird so gut wie keine Forschung betrieben, die sich speziell mit Visualisierung wissenschaftlicher Datensätze befasst und damit, wie Faktoren wie die Größe der Datensätze oder der Rendering-Ansatz den Stromverbrauch und die Effizienz beeinflussen. Darüber hinaus konzentrieren sich die meisten Forschungsarbeiten ausschließlich auf GPUs des Herstellers Nvidia, obwohl es auch andere Hersteller gibt, die sich ebenfalls um die Optimierung von Rechenleistung und Effizienz bemühen. Diese Arbeit analysiert die Auswirkungen verschiedener Faktoren auf den Stromverbrauch und die Effizienz verschiedener wissenschaftlicher Rendering-Methoden (Sphere Splatting und Raycast Volume Rendering) auf einer Auswahl von leistungsstarken Normalverbraucher-GPUs der drei gängigen Hersteller (AMD, Intel und Nvidia). Die Leistungsaufnahme wird mittels der in den meisten modernen GPUs integrierten Sensoren und eines externen Hardware-Messaufbaus gemessen. MegaMol, eine Plattform zur Visualisierung wissenschaftlicher Daten mit Automatisierungsfunktionen, dient als Framework für diese Benchmarks.

Abstract

Because of rising ecological awareness and higher cost of living, power efficiency is becoming an increasingly deciding factor in most technology, including computers. However, a recent trend with GPUs, already one of the most power-hungry computer components, is to allow higher maximum power draw than ever before. While there is research on the power efficiency of GPGPU applications, even going as far as modelling power consumption at the instruction level, there is next to none specifically on scientific visualization and how factors like dataset size or rendering approach affect power consumption and efficiency. Additionally, most research focuses solely on GPUs by manufacturer Nvidia, although other manufacturers exist and also strive to improve computing power and efficiency. This work analyses the effects of various factors on power consumption and efficiency of different scientific rendering approaches (sphere splatting and raycast volume rendering) across a selection of high-end consumer GPUs by the three common manufacturers of (AMD, Intel and Nvidia). Power consumption is measured with the integrated power sensors integrated into most modern GPUs and an external hardware measuring setup. MegaMol, a scientific visualization platform with automation capabilities, is the framework for the benchmarks.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Motivation

In 2019, Sun et al. [SADK19] compared more than 4,000 different CPU and GPU models dating back to 2000. They concluded that both Moore's Law[1] and Dennard Scaling[2], despite numerous claims through the years of them not holding up for much longer, are still in effect today. While the minimum possible FET size (currently at 3nm via TSMC's Nano-sheet FET [ST21]) is still shrinking, it continues getting closer to the physical limit of "keeping electrons in place" and will reach this point eventually. To increase computing power despite these physical constraints, Sun et al. [SADK19] also predicted that GPU manufacturers would increase both power limit and die size, as the power density per area will likely not decrease (Dennard Scaling). As larger monolithic chips are more prone to defects, Sun et al. [SADK19] also predicted the use of MCM packaging for upcoming GPUs, meaning that the chips are assembled from multiple individual chiplets instead of a larger monolithic chips, leading to smaller per-unit die size. Both predictions held up well, as at least AMD utilize multiple dies per package for their current RX 7000 series GPUs, and both AMD and Nvidia, the arguably most relevant GPU manufacturers, are now beyond 300 W TDPs for their flagship consumer GPUs. What was not directly predicted is the order of magnitude of the TDP increases: While Sun et al. [SADK19] stated that, over the last ten years, maximum TDPs slowly increased up to 300 W, recent flagship GPUs, especially on the Nvidia side, where the flagships' TDP stayed consistently around 250 W for multiple generations, the flagship's TDP suddenly increased to up to 450 W (GeForce RTX 2080 Ti -> 3090 Ti). While these newer, overall more power-hungry GPUs are proven to offer noticeably higher computing power than their predecessors [TPGPU], the question remains if the higher power draw is justified by this increase in speed.
More than ten years ago, Johnsson et al. [JGDA12] speculated that reporting energy consumption per pixel would be as common as reporting frame times in a research context. In the meantime, there was research on accurately measuring the power consumption of GPUs [JA14] mainly focusing on general-purpose computation on GPUs (GPGPU), even down to the instruction level [AEE+20] and also modelling approaches [LHE+13] [KPK+21] for predicting a compute kernel's power consumption. However, a field that, up until now, was mostly neglected in terms of analyzing power consumption and efficiency is scientific visualization. As complex simulations running continuously on high-performance computing clusters often have a visualization component, optimizing this aspect could also play a part in reducing global energy consumption. But, without insight into

[1] Not much of a law, rather an observation by Gordon Moore in the 1960s. The exact wording varies by source, but it states in some form that every n years (usually with $n = 2$), the possible complexity of an integrated circuit (or rather its transistor density) doubles. This metric alone does not imply that its speed or computing power doubles, but it is connected to increased computing power over time.

[2] A model that states that the power density of transistors stays constant despite their shrinking size, making power proportional to the area rather than transistor count. It was initially proposed by Dennard et al. [DGY+74] is related to Moore's Law, as the shrinkage of transistors is an unavoidable side effect of increasing transistor density.

how different parameters affect a system's power consumption, it is hard to tell where potential improvements can be made. Also, as time is not only money but longer runtimes of workloads inevitably increase the resulting power consumption, how efficiently the power is used also plays a role. It is more desirable if a workload is processed quicker by more power-hungry hardware than slower and still results in the same overall power consumption.

1.2 Goals

To gain intel on which factors influence the power consumption and efficiency of different scientific visualization techniques (sphere splatting and raycast volume rendering), this work will analyze an extensive set of combinations of different parameters. These factors include variations in hardware (GPU choice), frame rate, render method, test data, resolution and more case-dependant factors. As the GPGPU-focused research nearly exclusively focuses on GPUs manufactured by Nvidia, even less is known about the influence of these factors on GPUs by other manufacturers (AMD, Intel). Those other manufacturers may be often overlooked, primarily because their GPUs do not support CUDA[3], but they also strive to improve computing power and efficiency and offer comparable performance on paper, often at a more affordable price range. To accommodate this, our GPU selection will incorporate GPUs by all manufacturers and from multiple generations to capture potential improvements. A dedicated hardware power measuring setup allows to accurately measure the momentary power consumption of different components in the system at a high temporal resolution. As the complexity overhead of creating such a setup may play a role in why not more GPU research tracks power consumption and efficiency in addition to performance, we also compare the measurements acquired by this setup with measurements taken from the power sensors integrated into most modern GPUs by default. Knowing what exactly to expect from the integrated power sensors would allow us to use them as a convenient substitute for the complex hardware setup and would make including power and efficiency measurements in other work more easily realizable.

[3]CUDA is Nvidia's proprietary API for parallel computing on their GPUs. Although most prevalent in GPGPU applications, there are open alternatives like the Open Computing Language (OpenCL)

2 Related Work

Among other things, Sun et al. [SADK19] observed GPU manufacturers settling for TDPs of none higher than 300 W and predicted further increases in GPU power limits and die size. However, while power limits of recent GPUs have grown faster and higher than predicted, manufacturer-specified maximum power draw is a lacklustre metric for measuring real-world power efficiency. Over the almost 20 years of hardware they analyzed, architectures improved and changed drastically with a rising focus on power efficiency (as indicated by emerging techniques like power gating and dynamic voltage and frequency scaling (DVFS)).
To accurately determine a GPU's influence on the system's total power draw, a measurement setup that measures power directly at the GPU (or instead ignores all power consumed by other components) is necessary. One such setup, which is already quite similar to the setup used for this work, was proposed by Johnsson et al. [JGDA12]. It is a hardware setup for discrete (PCIe) GPUs and integrated/mobile devices. Included are a PCIe expander card (Ultraview PCIeEXT-16HOT) with shunt resistors on the 3.3 V and 12V lanes for measuring current, a custom board with four ACS710 Hall effect current sensors (up to 12A) and two shunt resistors (up to 1A, intended for smaller mobile device currents). The sensors are connected to an ARM Cortex-M3, which acts as an analog-to-digital converter (samples at 40 kHz) and sends the measured values to a PC via ethernet. The high sampling frequency also allows for analyzing individual frames. They examined one GPU each of every major manufacturer and an iPhone 4S (modified to allow power measurements directly at the battery). For the iPhone and the Intel iGPU, where only full-system power measurements were an option, idle power was analyzed and subtracted to approximate GPU power draw. Various primary and shadow rendering approaches were used as power benchmarks, and the results were compared within each category (discrete and integrated GPUs). They found that efficiency (as in performance per W) is not always directly proportional to performance. Also, for some algorithms, efficiency (and its influencing factors relative frame times and relative power consumption) vary noticeably across different platforms. It also should be mentioned here that this exact setup may cause issues with modern PCIe 4.0 GPUs (judging by experience with AMD 7000 cards) because of the increased length of the PCIe lanes on the riser card.
As a later addition, Johnsson and Akenine-Möller [JA14] proclaim that measurements should be done from the beginning of one frame to the beginning of the next. This change allows power consumption of idle time to be tracked and inter-frame coherency effects to better show up in the results. To enable this approach without intrusion into the source code (this was necessary for the earlier approach and caused power measurements to terminate before the start of the next frame), they propose a semi-automatic detection approach for finding "frame starts" by analyzing the shape of the recorded power curve of a single frame and pattern-matching it to the rest of the curve. Once a "potential frame start" is found, the surrounding samples are further analyzed to confirm that the beginning of a frame was found. This approach works best when V-Sync is enabled and, in some edge cases, may not identify all frames correctly with a variable frame rate. They also warn about measurements taken within the time right after a launch because energy consumption will be much higher on most architectures and, hence, less representative of performance within a continuous

load.
Müller et al. [MHWE22] used a similar hardware measuring setup to Johnsson et al. [JGDA12]. They primarily proposed Power Overwhelming, a C++ library measuring power from different sensors with a generalized API. Additionally, they ran a suite of benchmarks spanning different scientific visualization techniques (namely sphere and volume rendering) on systems with both Nvidia (RTX 3090) and AMD (Radeon Pro W6800) GPUs and compared their hardware measurements against the GPUs' integrated power sensors. They found that limiting the frame rate raises the energy consumption per frame and that the readings from the software sensors undershoot the hardware measurements (3-10% for Nvidia's NVML and 14-20% for AMD's ADL). They also recorded the course of power consumption of browser-based visualization and found that the AMD W6800 had more variance in power draw than the Nvidia 3090, which stayed relatively constant for a short time.
Another take on analyzing the power consumption of scientific visualization was done by Heinemann et al. [HBFE17]. They analyzed volume rendering on a mobile device utilizing an IntrinsycOpen-Q820 DevelopmentKit and an ARM Energy Probe and observed an approximately linear correlation between the Qualcomm Snapdragon 820 SoC's utilization and power consumption.
While a hardware setup like the aforementioned can produce very accurate measurements with high temporal resolution, it also requires dedicated and not as readily available hardware (at least not in an assembled and ready-to-use state). When the research goal is not primarily evaluating power efficiency, such a setup is likely too complicated and causes too much overhead for "additionally recording the GPU power consumption". Fortunately, most modern GPUs ship with onboard power sensors, which are accessible via API calls and, while overall less accurate, offer far more convenience and a significantly smaller barrier of entry. Burtscher et al. [BZZ14] found shortcomings in the integrated power sensor on the Nvidia Tesla K20 (Kepler Architecture) and proposed an approach for improving the quality/accuracy of the sensor's measurements. They noticed a non-linear correlation between kernel runtime and total energy consumption and that power consumption is still higher than expected after a kernel has succeeded. The first phenomenon resulted from the energy consumption while kernel runtime resembled limited growth (even though the kernel ran at full speed from the beginning). The second resulted from the tailing limited decay of power right after kernel succession and a kernel-independent constant amount of energy before returning to idle power. As these findings mostly match the behaviour of a capacitor (de)charging, they can be modelled this way, and the corresponding formulas can be applied to get closer to the actual power consumption. However, it should be noted that this was evaluated on the first iteration of Nvidia's onboard power sensor, and it needs to be clarified if those findings apply to the more recent GPUs.
Instead of benchmarking whole compute kernels, Arafa et al. [AEE+20] focused on a more generalized approach by analyzing the power consumption of individual PTX instructions[1]. Measurements were done on multiple generations of Nvidia GPUs (Maxwell, Pascal, Volta, Turing) utilizing their onboard power sensors and three different software approaches for querying the sensor (NVML with polling at constant intervals, NVML with multi-threaded synchronization and via CUDA/PAPI) were compared. The results were verified against a hardware setup (also utilizing a PCIe riser with shunt resistors and an oscilloscope with hall-effect sensors), and NVML with multi-threaded synchronization (where the start and end timestamps of the benchmarks were most accurate) produced the closest results.

[1] PTX is an assembly language for CUDA programming and the finest granularity of publically available instructions for Nvidia GPUs. The machine-independent PTX gets compiled to the machine-dependent SASS instructions with variations across different GPUs/GPU generations.

On the other side, there is also the modelling approach for determining, or rather "predicting", power consumption instead of measuring it. Hong and Kim [HK10] proposed an "integrated power and performance prediction model" to predict the optimal number of active processor cores (active meaning "used when running the program") for a given application for a specified GPU architecture. Peak memory bandwidth serves as the primary metric for determining the point where more cores are unlikely to improve performance further, and it would be more beneficial for overall efficiency to disable them (they measured an improvement of 11%). For just running the model, no measurements of execution time, hardware performance counters or architecture simulation are required. It takes the code of a GPGPU kernel as input and outputs predictions for both performance per watt and the optimal thread/block configuration. Predictions are based on GPU instructions, while each instruction is factored by the logical components (ALU, FPU, Cache, etc.) it utilizes. Power consumption resulting from VRAM transfers/usage, cooling fans and temperature increases are also factored in. To create the model, factors like "maximum power" for various parts of the GPU and architecture-dependant scaling factors must be determined empirically for each architecture by synthetic micro-benchmarks. The model's limitations are inherited from the underlying power and timing models and include control flow intensive, asymmetric, and texture cache intensive applications. This model was initially established with a GTX 280, built on an obsolete, over ten years old architecture. This difference in architectural fidelity may render the model less applicable to modern GPUs, where many more power-saving techniques are present.
Another now obsolete model is GPUWatch by Leng et al. [LHE+13]. They claimed that performance per watt is more crucial than peak performance and provided an improved cycle-level model for predicting GPU power consumption down to microarchitecture components. Eighty micro-benchmarks are utilized for setting initial parameters, and the model was validated for Fermi GPUs. They show how DVFS can reduce energy consumption by 14.4% in general and 66.6-13.6% for clustered execution and that clock gating reduces dynamic power, one component of total power, by 11.2%. These techniques are staples in modern GPU architectures. Abe et al. [ASP+12] have proven earlier that, in general, tighter control of GPU voltage and frequency can reduce the whole system's power by 28% while only losing less than 1% of performance.
While Leng et al. [LHE+13] showed how DVFS can be beneficial, GPUWatch does not account for the benefits of its implementation in newer architectures (Nvidia Pascal, Volta, Turing and onward) and will falsely predict power consumption of more than 2000% of the measured value, as shown by Kandiah et al. [KPK+21] mostly caused by a lack of support for both the DVFS implementation and the "radical" power gating on newer architectures. They also claim that while GPU performance modelling has improved over the years, power modelling still needs to catch up. Based on a heavily modified GPUWatch [LHE+13], they propose Accelwatch, a new detailed and accurate cycle power model capable of capturing constant and static power. It supports emulation, trace profiling and hardware counters, also simultaneously and in combination, and accounts for power gating, control-flow divergence and DVFS. The latter two are not accounted for in Hong and Kim [HK10]'s model. The modular construction of the model is tailored to study discrete hardware components without developing new models for all of the architecture's components by allowing the use of real-world data where available and filling the rest with simulations. Once a good model for one architecture is compiled (absolute percentage error of 7.5–9.2±2.1–3.1% on their model for the Volta architecture), it is not necessarily required to be "retuned" for another GPU architecture to deliver accurate power models (unchanged Volta model delivers error of 11±3.8% (Pascal) or 13 ± 4.7% (Turing)), making design space exploration possible. What should be kept in mind with these percentage values is that the Volta model was only verified against NVML data, which is, as shown by Müller et al. [MHWE22], itself off by 3-11% and mostly lower than the value measured by an

external hardware setup.
Bridges et al. [BIM16] gave a further overview and a comprehensive picture of various modelling/profiling/simulation approaches. Their main claim is that the comparably advanced CPU modelling and simulation provide a basis or a jumping-off point for further GPU research. Future work should focus on more generalization to allow experimenting with combining parts of different architectures effectively, which was not as possible with rather architecture-dependant models of the time they gave this overview, but the newer Accelwatch already ticks some of the boxes.

3 Background

3.1 MegaMol

MegaMol [mm] [GBB+19] is an open-source and cross-platform tool for scientific visualization. It was initially intended for visualizing point-based molecular datasets. It also supports volume ray cast rendering for visualizing volumetric data[1] and various 2D render modes for analyzing statistical data (scatterplot matrix, parallel coordinates and table histogram). OpenGL is used for rendering, and the various render modes are all implemented in GLSL[2].
A sphere renderer is used to visualize particle datasets as a cloud of spheres. Different types of particles can be highlighted. There also is the option of a raytracing mode, which is not utilized for the benchmarks in this work. Different render methods are available, which allow the utilization of different data structures (standard OpenGL buffer arrays and Shader Storage Buffer Arrays) for uploading the particle data to the GPU. The SSBO render method also allows the data to be kept in the GPU's VRAM, removing the need to upload it for every rendered frame repeatedly.
In contrast to the sphere renderer, where the render methods (at least the ones we use in the benchmarks) implement different procedures but produce visually indistinguishable results, Integration and Isosurface follow the same steps up to a point until diverting to vastly different results (more on the visual implications of different config parameters in 4.4.1). Integration applies a transfer function to the voxels, which assigns different voxel ranges to different RGBA colours. Conversely, Isosurface extracts a connected shape from the volume by filtering for only a specific grey value (iso value).
Scenes in Megamol can be constructed with the module graph, where various modules can be connected by different calls to exchange and process data. The parameters of the modules can be edited via a hierarchical GUI and console commands.
In this work, MegaMol mainly serves as the framework for running various sphere and volume renderer test cases, and its source code is modified to help realize this (see 4.2 for details). Additionally, MegaMol's 2D visualization modes proved themselves helpful in analyzing the benchmark results.

3.1.1 LUA Scripting Interface

Megamol projects containing the entire state of MegaMol's graph and current GUI state are stored as Lua scripts.
Lua [Lua] is a comparably minimalistic programming/scripting language, which is not only designed

[1] Volumetric datasets consist of layers of voxels (usually evenly-spaced on each layer), and each voxel has a grey value (often representing density or resistance to specific radiation). Volumetric data is commonly produced by different types of 3D scanners (computer tomography, magnetic resonance, etc.).
[2] OpenGL's shader language.

to be fast and robust but with a small footprint (< 2 MB for the standard library, the interpreter and documentation) in mind. The small size encourages embedding Lua into more extensive projects, and MegaMol is one of them.
Lua is used within MegaMol as a scripting interface, which can also be actively used at runtime via the console. However, the more relevant use case is to automate MegaMol with consecutively executed Lua instructions.
With the project files being functional Lua scripts, loading them does not simply load data, but actively instructs MegaMol to set the graph and the GUI to a specific state. However, it does not have to end at reconstructing a specific state, as the whole of Lua as a fully-featured modern programming language allows control flow to automate arbitrarily complex sequences of events.
To be accessible via the Lua interface, "Lua callbacks" must be registered within MegaMol. A Lua callback is created by defining a Lua function and the parameters it takes within MegaMol's source code, which is written in C++, not Lua. The actual implementation of the function is also done in C++ but will be called whenever the defined Lua function is called. As it is also possible to provide a return value, communication between the Lua interface and MegaMol can be bidirectional to control and retrieve data from MegaMol during runtime.
Examples of Lua callbacks include

- **mmCreateView(), mmCreateModule()** and **mmCreateCall()** for generating the graph,
- **mmSetParamValue()** for setting GUI parameters to a given value (parameters are addressed by following the hierarchy in the graph),
- **mmRenderNextFrame()** for rendering frames on command
- or **mmQuit()** for exiting MegaMol.

3.2 Power Overwhelming Library

Power Overwhelming [pov] [MHWE22] is a C++ library for retrieving data from different types of power sensors with strictly separate APIs in a manner that is as generic as possible across various integrated sensors and external devices. Currently supported are

- the integrated power sensor(s) on AMD GPUs via the **AMD Display Library (ADL)**,
- the integrated power sensor on Nvidia GPUs via the **NVIDIA Management Library (NVML)**,
- Running Average Power Limit (RAPL), which is used for accessing the CPUs integrated power sensor,
- **Tinkerforge** Voltage/Current Bricklets 2.0,
- Rohde & Schwarz oscilloscopes of the RTB 2000 family,
- and Rohde & Schwarz HMC8015 power analysers.

The library allows all sensors to be accessed synchronously, where samples are acquired on demand, or asynchronously, where the sensor is configured to provide examples at a fixed time interval. To allow asynchronous sampling on sensors that operate strictly synchronous (like NVML), Power Overwhelming creates a sampling thread as an abstraction, which handles the repeated queries at fixed intervals and can be configured to process the samples as desired. The main advantage to asynchronous sampling, at least for the sensors that support it out of the box (like Tinkerforge), is that once configured, the communication with the sensor is unidirectional, as only the sensor has to send data. If synchronous sampling is used, the bidirectional communication of the queries and the samples would require twice the bandwidth, and the sensor could be under a higher load for processing the queries. Both factors could negatively influence the maximum possible sampling interval, decreasing temporal accuracy. The library provides a Windows file time[3] timestamp in milliseconds along with every sensor sample. For all sensors but ADL, where the samples already come with a timestamp, the timestamps are generated when the sample is acquired. The timestamps are helpful when assigning sensor samples to benchmark test cases in retrospect.

3.3 Power Measurements with Tinkerforge Bricks and Bricklets

As all of the PC's components run on direct current (DC), so power draw P [W] can be computed by simply multiplying measurements of voltage V [V] and current I [A]:

$$P = U \cdot I \tag{3.1}$$

Voltage is measured parallel to the carrier by referencing the desired point's electrical potential to a common ground. Assuming that the PC's PSU outputs a stable voltage (of either 3.3, 5 or 12 V), we could theoretically assume this as constant. However, as we want to strive for high accuracy and there are always small voltage fluctuations in the mV range, and because we get this measurement more or less for free in our measuring setup, we do include it.
Measuring current is generally more complex, as it has to be measured in series. There are two common approaches to accomplish this. The first inserts a shunt resistor (usually in the mΩ range) and measures the voltage drop across it. Ohm's law can be used to derive the current from these factors:

$$I = \frac{U_{drop}}{R_{shunt}} \tag{3.2}$$

This approach is intrusive as it is necessary to route all of the current contributing to the total power across the shunt resistor.
The second approach uses a hall-effect sensor to detect the magnetic field induced by the current externally. The sensor surrounds a wire section instead of intercepting it, making it less intrusive. However, according to Maniar [Man18], hall-based solutions are more susceptible to noise and temperature changes, less accurate and overall less flexible.
For our purposes, the current measuring is handled by a Tinkerforge Voltage/Current Bricklet 2.0,

[3]Time in nanoseconds since January 1, 1601, 12:00 AM as a 64-bit unsigned integer

internally using a Texas Instruments INA226 power monitor. It is intrusive as it is shunt-based, but it also measures voltage and directly computes power.
Tinkerforge is an open-source modular platform with various building blocks conveniently accessible via USB or (W)LAN. There are two types of building blocks: *Bricklets*, like the Voltage/Current Bricklet 2.0, are connected to and controlled by *Bricks*, which include a microcontroller (32bit ARM) and are stackable for more connectivity. We can save bandwidth by only retrieving the already computed values for power. This optimization is convenient and critical as retrieving two values (voltage and current) could be problematic at the high temporal resolution (5 ms between samples) we want to measure.

4 Experiment

With this work being a continuation of [MHWE22] and being carried out at the exact location, hardware and proposed library for measuring power were reused wherever possible and adapted when necessary and viable. While the hardware mainly remained unchanged (except for an overhaul of the Tinkerforge setup and the use of a different selection of GPUs), it was necessary to extend MegaMol's source code [mm] to enable its use as a benchmarking platform for power benchmarks.

4.1 Hardware and Software Setup

4.1.1 Testbench

The testbench is an open PC case, to allow for easy component access and quick GPU changes, with the modified power cables and the tinkerforge setup next to it. It is an x86_64 PC configuration based on an AMD Ryzen 5900X Processor (12 cores, 24 threads at 3.7-4.8 GHz rated at 105 W) on an ASUS ROG Strix X570-E Gaming motherboard. It has 64 GB of DDR5 memory and multiple solid-state drives: a SATA SSD for the operating system and an NVME SSD for all relevant data/software. A be quiet Dark Power Pro 1200 W is the power supply unit.
While the rest of the system's hardware remains unchanged for all test cases, seven different GPUs featuring all major GPU manufacturers (AMD, Intel, Nvidia) are used in succession. All selected GPU models are targeted at consumers (no professional/workstation GPUs) and, with one exception, include each manufacturer's top and penultimate models from their current generation (AMD: Radeon RX 7900 XTX and 7900 XT; Nvidia: GeForce RTX 4090 and 4080) and the top models from their last (AMD: Radeon RX 6900 XT; Nvidia: GeForce RTX 3090 Ti). As Intel does not have a comparable offering to the other manufacturers' highest-end consumer cards, neither in their current nor their last generation, only one Intel GPU (ARC A770) is included. Table 4.1 provides an overview of selected technical specifications of the chosen GPU models (see Chapter A for further details). Except for the GeForce RTX 4090 and ARC A770, all GPUs are factory overclocked, raising out-of-the-box GPU and memory clocks compared to their base models. For the benchmarks, no further overclocking was attempted.

Chip M.	GPU	VRAM Size	VRAM Clk. (eff.)	VRAM Bandw.	FP32 (float)	Pixel Rate	TDP	MSRP (2023)
AMD	Radeon RX 6900XT	16 GB	16 Gbps	512 GB/s	23.65 TFLOPS	295.7 GPixel/s	300 W	700 USD
AMD	Radeon RX 7900XT	20 GB	20 Gbps	800 GB/s	55.05 TFLOPS	491.5 GPixel/s	300 W	899 USD
AMD	Radeon RX 7900XTX	24 GB	20 Gbps	960 GB/s	63.04 TFLOPS	492.5 GPixel/s	355 W	999 USD
Intel	ARC A770	16 GB	17.5 Gbps	559.9 GB/s	19.66 TFLOPS	307.2 GPixel/s	225 W	350 USD
Nvidia	GeForce RTX 3090 Ti	24 GB	21 Gbps	1008 GB/s	41.29 TFLOPS	215.0 GPixel/s	450 W	1499 USD
Nvidia	GeForce RTX 4080	16 GB	22.4 Gbps	716.8 GB/s	50.49 TFLOPS	290.6 GPixel/s	320 W	1199 USD
Nvidia	GeForce RTX 4090	24 GB	21 Gbps	1008 GB/s	82.58 TFLOPS	443.5 GPixel/s	450 W	1599 USD

Table 4.1: Specifications of the selected GPUs (source: TechPowerUp GPU Database [TPGPU]).

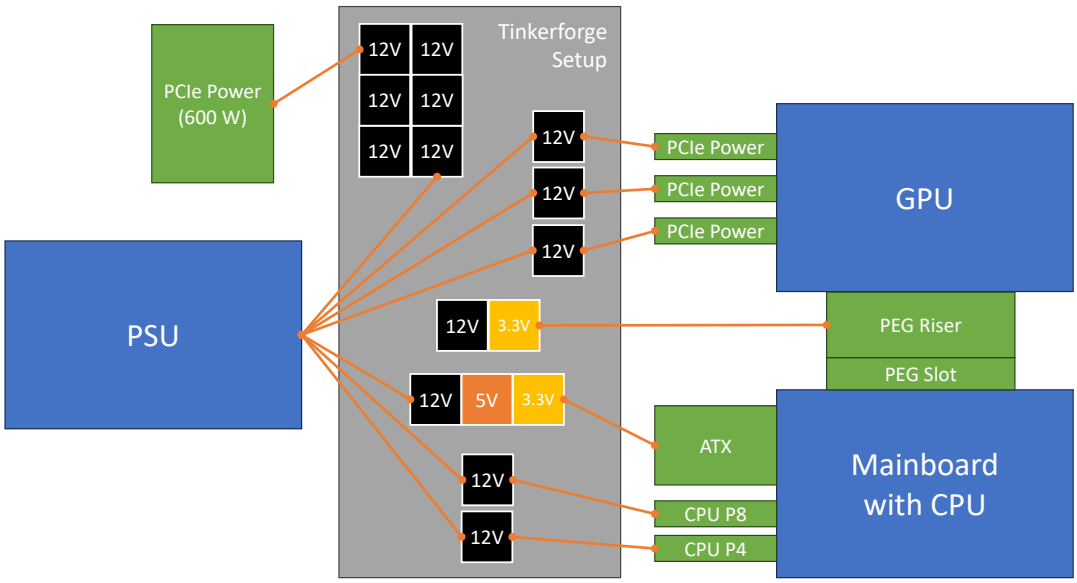


Figure 4.1: Schematic overview of all relevant power connections (green) and how each set of power lines (orange) is connected to the Voltage/Current Bricklets (based on Fig. 3 of [MHWE22]). The specific setup shown (with the 600 W connector disconnected) would be applicable for e. g. the AMD Radeon 7900 XTX.

4.1.2 Tinkerforge and Hardware Modifications

Like the testbench, the power measuring setup used by Müller et al. [MHWE22] was also adopted for this work. To accurately measure each component's power consumption (or at least differentiate between the power draw of the GPU and the rest of the System) all of the PSU's outgoing power lines are cut and intercepted by Voltage/Current 2.0 Bricklets (featuring Texas Instruments INA226 power monitors). Additionally, because the GPU draws a part of its power from the motherboard's power via the PEG slot, a special PCIe riser card with voltage/current probe points is required. This riser card is installed in the PEG slot instead of the GPU, and the GPU is placed on top of it. Because the CPU (similar to the GPU) also draws power from the motherboard, its isolated power can also not be measured by only including the readings from the 8-pin CPU power connectors. However, in contrast to the GPU, we do not know of a "CPU riser" which would allow to intercept the CPU socket's power rails to allow for measurements. Originally, another PCIe riser card without probe points was installed between the first riser and the GPU to decrease wear on the riser caused by the numerous GPU swaps. However, the increase in PCIe lane length (or some other related factor) caused by the second riser made the AMD 7000-series GPUs stop working correctly [1], so it was omitted.
The riser's probe points are also connected to Voltage/Current Bricklets and later allow us to

[1]The GPU was unable to even produce a stable 60 FPS on an idling Windows 10 and dragging around console or explorer windows caused noticeable stutters. Also, GPU utilization, as displayed by the Windows Task Manager, was inconsistent and did not seem to match the actual utilization but was generally too high for what it should be (seemingly random spikes to 100% while not actively interacting with anything or otherwise causing GPU load).

determine how much of the motherboard's power is used by the GPU. As shown in 4.1, we need 16 Voltage/Current Bricklets for the following power lines:

- Motherboard and periphery (ATX) 3.3 V, 5 V and 12 V
- dedicated CPU (150 W 8-pin connectors) 2 x 12 V
- dedicated GPU (3 x 150 W 8-pin PCIe connector) 3 x 12 V
- dedicated GPU (1 x 600 W 12+4-pin PCIe connector) 6 x 12 V
- GPU via motherboard (probe points on riser card) 3.3V and 12 V

Wherever possible (without exceeding the Bricklet's current limit), multiple power lines within the same connector were connected to a single Bricklet. Although none of the selected GPUs requires all of the power connectors simultaneously because rewiring the Bricklets would stretch out the GPU changing process significantly, and the Dark Power Pro 1200 W has enough PCIe power connectors, all Bricklets stay connected all the time.
Although technically possible, the power connector for the SATA SSD has no dedicated Bricklet to save bandwidth, as the power consumption of a mostly idling SSD is insignificant compared to the CPU and GPUs with three-digit power ratings.
All Bricklets are connected to the Brick, which is connected to the testbench via USB-C. Although available both physically and within Power Overwhelming (the library for interfacing with the different types of power sensors), the Rohde & Schwarz HMC8015 power analyzer and the Rohde & Schwarz RTB2004 Digital Oscilloscopes were not used in this particular test setup. The oscilloscope is out of the question because it is tailored to high-resolution (per frame) momentary readings, while this test is highly automated, has a long runtime to cover all relevant combinations of test parameters, and measurements over 5 seconds have to be combined. The tinkerforge sensors' resolution is sufficient and the setup does not produce as much data this way. Also, even with the two oscilloscopes, there would not be enough sensors to cover all power sources separately. However, the oscilloscopes were used for a debugging step in preparation for this test run. The power analyzer was omitted to streamline the benchmark process further and because the whole system's power does not help analyze the power draw of individual components. However, the measurements could have helped measure the PSU's influence on efficiency and more accurately represent the whole system's efficiency (PSU losses included).

4.2 Implementation of Powerlogging Service

The current master of MegaMol [mm] already includes a recent version of Power Overwhelming, but the integration to configure and actually use the power measurements was part of this work. The result can be found on a forked Git repository [mm-f] on GitHub.
The desired procedure is to start and configure MegaMol to cycle through various pre-defined test cases via a Lua file. It is not sufficient to simply record all possible power sensor data in the background, we also need a way to match the sensor data to the test cases accurately. To accomplish this functionality, we settled on a MegaMol service, a structure within MegaMol that uses polymorphism to inherit from "AbstractFrontendService" (based on a template found in frontend/services/service_template). A service is initialized as a service object and a corresponding

config object, which are added to the FrontendServicesCollection ”services” in MegaMol’s main.cpp. It provides various abstract methods that get called at specific points during the rendering process and allows for conveniently sharing resources with other services (like the GUI, for example). For our purposes, we do not care about most resources, as we neither require our service to be controllable via MegaMol’s GUI nor to interfere much with MegaMol, except for a few specific cases. Of the abstract methods for sharing resources, we only really need setRequestedResources() to enable us to create custom Lua callbacks, which help us control the Power Library and extract specific data from our current MegaMol session. The other abstract function we make use of is postGraphRender(), but only for synchronous sampling (or rather sample_sensor()-calls, counting frames and flushing the sample buffer accordingly (note that buffers will also be flushed when full in asynchronous mode). Synchronous sampling and automatic flushing are technically not required for the benchmarks, as buffers are manually flushed after completing a test case via a Lua callback. However, they were implemented as an intermediate step and are still present. As there was no intention of dynamically controlling this ”Powerlogging_Service” via MegaMol’s GUI, it is either enabled when MegaMol_USE_POWER is set and always running in the background or not enabled at all (this should, however, be achievable without completely rewriting the service).
Like common for classes in modern C++, a MegaMol service consists of a header (.hpp) and a body (.cpp) file. As intended for services, the header defines a ’Config’ structure, which contains variables for setting properties specific to the service’s implementation:

- **powerlog_file:** path for logging file (only for synchronous, all synchronous sensors are logged to this one file (for asynchronous, every sensor has its file))
- **frames_per_flush:** how many frames to render before flushing buffer
- **frames_per_request:** how many frames to render until sampling sensor
- **asynchronous_logging:** can be en/disabled. Synchronous logging writes samples directly to the log, asynchronous uses a separate thread that is notified whenever a sample buffer is full
- **asynchronous_sampling:** can be en/disabled. Synchronous sampling samples in between frames and utilizes a shared buffer, asynchronous in a specified interval on a separate thread with separate buffers for each sensor
- **sample_timeout:** time between samples when using asynchronous sampling
- **sample_buffer_size:** maximum number of samples (per sensor) that can be stored before automatically flushing the sample buffer. Should be chosen so that one of the buffers can be flushed in time before the other buffer is full (asynchronous sampling uses double buffering)
- **sensors:** which sensor types to sample (currently supported: ADL, NVML, Tinkerforge). If one sensor type is enabled, all sensors of this type will be sampled.

In addition to most of what was already defined in the template service, the Powerlogging_Service’s header also contains private structs for internally handling sensor samples as timestamp/value pairs with a constructor (compact_sample) and for keeping track of sampling threads (sampling_container). There are also collections for tracking the individual sensors in use and their corresponding logging threads (in case of asynchronous sampling).
For all collected services, init() will eventually be called. In the case of the Powerlogging_Service, it does the following in this order:

- The internal config parameters are set to the values given by the external Config object (the one that was added with the service object to services collection)
- A unique ID for this MegaMol instance is derived from the current timestamp. This ID is used for generating different names for asynchronous log files of different test runs (for example, running the same test cases once with and once without V-Sync, where MegaMol has to be restarted) so they do not overwrite each other by sharing the same file name. The ID will also be accessible via a Lua callback for being usable for the exact purpose outside of the Powerlogging_Service.
- Each sensor type is checked for being enabled, and all available sensors of the enabled types are collected (NVML is usually only one for single Nvidia GPUs, ADL are likely numerous, even with a single GPU, and Tinkerforge will be as many as connected). This collection step is more elaborate for Tinkerforge sensors, as they get configured to internally average 4 samples and their conversion time is set to 588 μs (this should result in a sampling interval of approximately 5ms and got Müller et al. [MHWE22] the best results).
- for ansynchronous sampling only: bind_sensor() is called for all collected sensors

Synchronous sampling happens within postGraphRender(), where a sensor sample is captured, reformatted to the desired CSV format and written to a shared buffer. On the other hand, asynchronous sampling is enabled via bind_sensor(), which does the following:

- A unique file path for the sensor called on is generated by combining the unique session ID and the sensor's ID
- A new sampling_container for this sensor's sampling thread is created. It consists of the following:
  - the name,
  - double buffers for sensor samples and the maximum buffer size
  - mutex locks and a signalling variable for controlled and thread-safe buffer swapping and flushing
  - a time-to-die flag

  Pointers to all sampling_containers are stored within the Powerlogging_Service.
- Power Overwhelming's asynchronous sampling is configured:
  - sampling timeout and the pointer to this sensor's sampling container are provided
  - method to execute each time when sampling is provided: "store_sample_and_flush_if_necessary"
    - flush is considered "necessary"when the active buffer is full or the time-to-die flag is set
    - blocks the thread until buffers are successfully swapped (should be almost instantaneous, as only pointers are swapped using std::vector<>::swap())
    - the corresponding logging thread is notified via the signalling variable
- the corresponding logging thread is also created

  - also provided with the corresponding sampling container
  - executes "flush_powerlog_buffer":
    * waits for a signal on the sampling container's signalling variable (intended to originate from the sampling thread, when the buffers were swapped after the active buffer has been full)
    * formats and writes the contents of the inactive buffer to a CSV file and empties the buffer (the writing will likely take multiple milliseconds, but the clearing of the buffer should happen almost instantaneously via std::vector<>::clear())
    * terminates if the time-to-die flag is set (this usually happens when the service is terminated by exiting MegaMol)

At least when MegaMol is exited, the Powerlogging_Service will be terminated alongside all the other services and close() will be called. It is implemented as follows: close():

- The time-to-die flag gets set for all sampling containers. This step will eventually cause the sampling threads to notify the logging threads to do their final flush, regardless of how filled the buffers are.
- The service has to wait for the logging threads to empty the remaining samples from the buffers.
- All sensors get unbound, so the threads for asynchronous sampling threads within Power Overwhelming also terminate.

In addition to all of this, fill_lua_callbacks() is utilized to create the following Lua callbacks, which can later be used via the Lua scripting interface:

- **mmFlushPowerlog():** manually flush the powerlog buffers of all sensors
- **mmSwapPowerlogBuffers():** swaps active and inactive powerlog buffers of all sensors
- **mmSwapSwapAndClearPowerlogBuffers(bool clear_active_buffer, bool clear_logging_buffer):** swaps active and inactive powerlog buffers of all sensors and clears them if specified so
- **mmGetPowerTimeStamp():** Returns a numeric timestamp (in ms) matching the format used within the power overwhelming library (this includes a shift from the UNIX epoch (1/1/1970) to the Windows epoch (1/1/1601))
- **mmGetInstanceName():** Returns a string that changes on each execution of Powerlogging_Service (the same ïnstance IDäs generated within init())

This sums up all required changes to MegaMol's source code. Any further configuration of the test cases is done externally via MegaMol's Lua scripting interface.

4.3 Software Environment

The benchmarks were run under Windows 10 Pro Version 22H2. The following versions of the GPU drivers were installed:

- AMD: 31.0.21001.45002 (Adrenalin 23.7.1)
- Intel: 31.0.101.3959
- Nvidia: 31.0.15.3640 (NVIDIA 536.40)

The driver version as an influencing factor in performance and power consumption should not be overlooked as some of the GPUs are, when this is written, approximately or even less than a year old and especially the Intel A770 is known to have significantly better performance with more recent drivers.
The version of MegaMol is the commit with id f6f022e0dd from August 8 2023 form the custom fork [mm-f], and was compiled with Microsoft Visual Studio 2022 with all default settings, except for enabling MEGAMOL_USE_POWER, to enable the Powerlogging_Service.

4.4 Benchmark Process

4.4.1 Test Parameters

As our overarching goal is to determine how different factors in scientific visualization influence power draw and efficiency, we first need to establish a parameter space with the exact parameters and their corresponding values we want to analyze. To accurately determine the influence of each parameter on all other parameters, we have to run at least one test case for all reasonable combinations of parameters. ”Unreasonable” combinations of parameters include all test cases, where a test parameter is used, that does not apply to the test case in use. The render methods and test data values are only intended for either the sphere or volume renderer. The thinning factor and sphere radius only apply to the sphere renderer, and the step ratio only applies to the volume renderer. Additionally, there are all cases of the type (Intel ARC A770, ..., SSBO stream with static data, ...) because this combination does not work correctly on the version of MegaMol used for the benchmarks. All of these test cases would either be duplicates of other ones because they do not change anything or would not run at all. Because of time constraints, we settled on running each test case exactly once and for 5 seconds. As further elaborated in 4.4.2, if MegaMol is still in the middle of rendering a frame when the 5-second time limit is reached, the rendering will not be force stopped, but it will be the last frame rendered for this test case. This behaviour may cause the total runtime to be off by a few milliseconds (typically < 10). Because the exact runtime and the number of actually rendered frames are tracked, the average FPS of a test case can be determined accurately. Ideally, we would want to use a single MegaMol project file, but as shown in Table 4.2, certain constraints to some of the test parameters render such a setup impossible.
By design, our outermost test parameter will be the selection of the **GPU**, as changing this parameter requires physically changing the GPU installed to the testbench. As the system needs to be shut down for this, we can not automate this step.
The following two parameters are also not viable to change within one MegaMol session:
The **test case**, or more precisely using either the sphere or volume renderer, dictates the type of usable

test parameter	applicable test case	possible values
GPU	all*	AMD Radeon RX 6900 XT, AMD Radeon RX 7900 XT, AMD Radeon RX 7900 XTX, Intel ARC A770, Nvidia GeForce RTX 3090 Ti, Nvidia GeForce RTX 4080, Nvidia GeForce RTX 4090
test case type	all**	Sphere Renderer, Volume Renderer
V-sync state	all**	true, false
render method	Spheres	simple, SSBO stream with static data
	Volume	Integration, Isosurface
test data	Spheres	ace_drop_1m, expl30m
	Volume	bonsai, bunny, chameleon
camera angle	all	1-8
resolution	all	720p, 1080p, 1440p, 2160p
thinning factor	Spheres	1, 4, 16
sphere radius	Spheres	1, 2
step ratio	Volume	0.5, 1, 2

* requires hardware modification

** requires MegaMol restart

Table 4.2: Overview of all test parameters, the test cases where they are applicable and all potential values they can be at

dataset (particle or volumetric), the available render methods and some other test-case-exclusive parameters. Loading another MegaMol project while one is already open is possible. However, instead of overwriting the current project, the new project's contents are added, making this approach unsuitable for our purpose.

The **V-Sync state** determines whether the FPS are capped to the display's refresh rate (60 Hz in our case) or unlocked to render as many frames as quickly as possible. Turning V-Sync on or off is done before MegaMol is interactable by either changing a setting in the "MegaMol_config.lua" or by using the CLI option "–vsync" when launching MegaMol. We can use a batch script to launch multiple sessions of MegaMol with the correct project file and CLI options.

The remaining test parameters are all changeable via the Lua scripting interface and thus can be handled within an ongoing MegaMol session and automated within the same MegaMol project file. Two of these parameters apply to both test cases:

Resolution includes a selection of four common high-definition resolutions in 16:9 format. One expected outcome would be that higher resolutions produce lower framerates and may or may not be more expensive and less efficient. **Camera angle** comprises eight possible angles, as shown in Fig. 4.2. They were generated via the Lua callback "mmGenerateCameraScenes()"to stay consistent across different resolutions, render modes and datasets. In our case, the mode was set to "orbit". The main purpose of the different angles is to minimize the influence of asymmetry in the datasets

(all suffer from this to a degree).
The parameters specific to the sphere renderer test cases include the two possible **render methods** (simple renderer and SSBO stream with static data). The **datasets** consist of the smaller and cubic ace_drop_1m (1 million particles) and the larger and more elongated expl30m (30 million particles). There are also two parameters exclusive to spheres:
The **thinning factor** reduces the particle density (thus reducing the particle count). A thinning factor of *n* results in only every *n*th particle from the dataset being included in the render (and skipping the rest). For the simple renderer, this is done by setting the stride when uploading the data to the GPU (handled by the GPU driver's OpenGL implementation). Stride is a byte offset that determines, after how many values in an array a new element begins. The SSBO renderer uploads

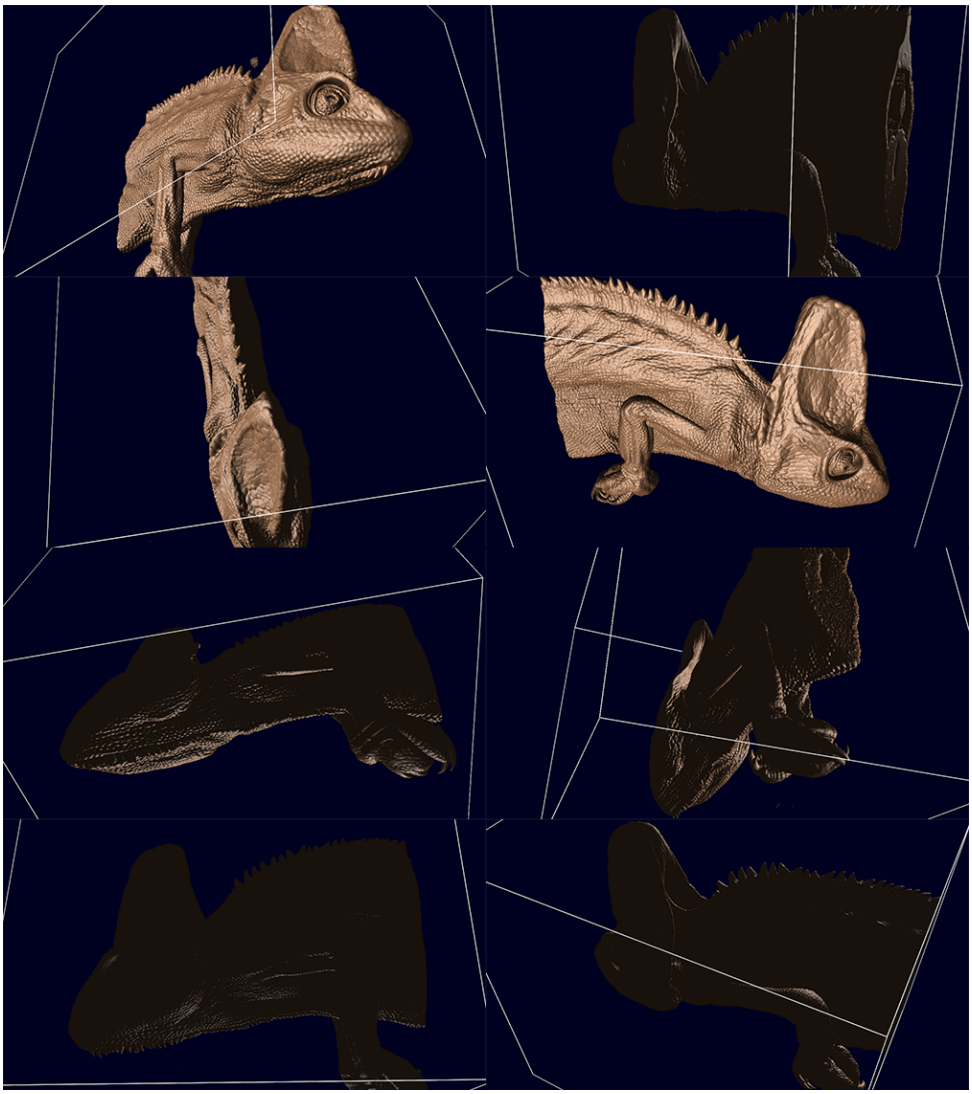
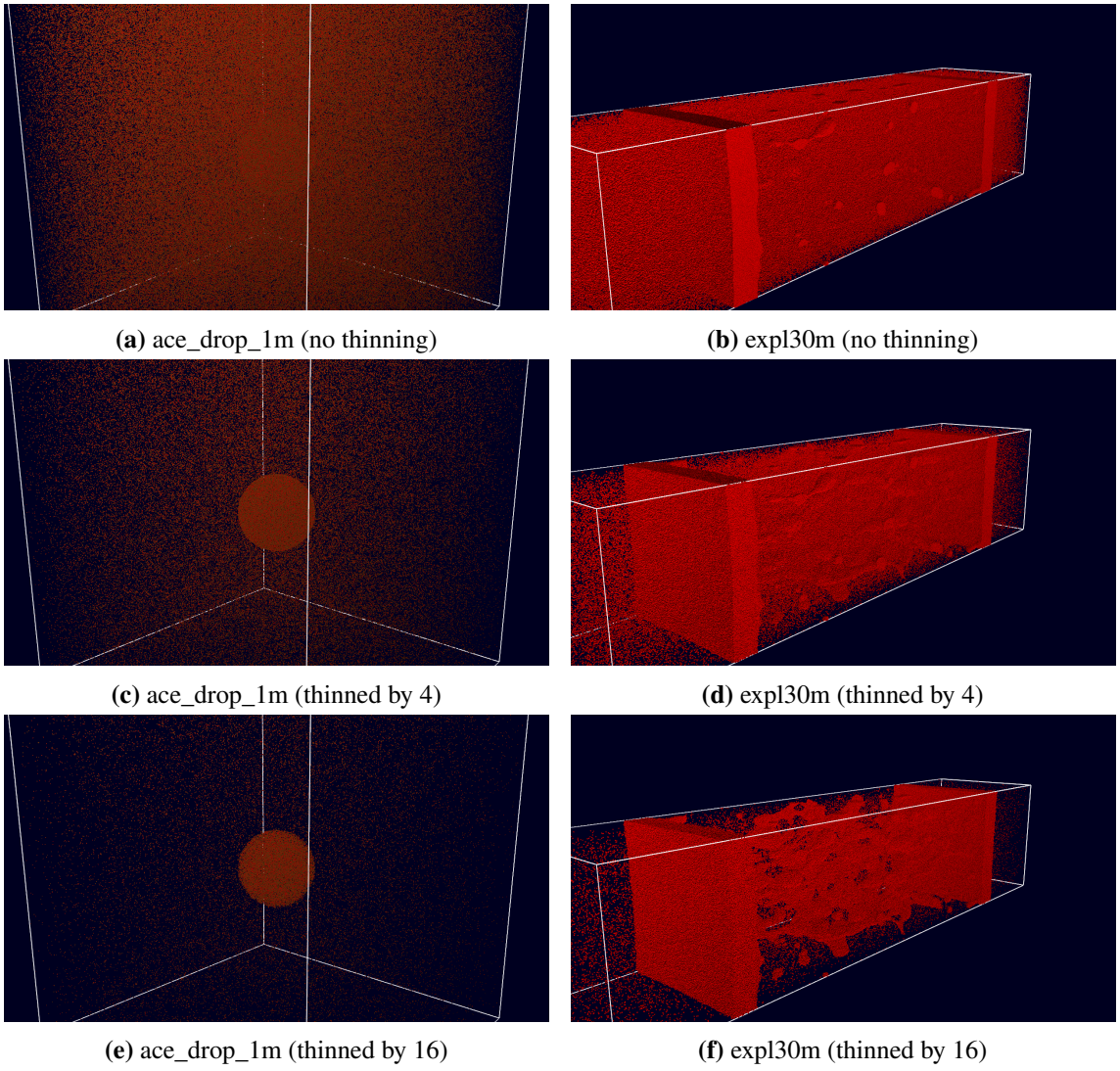


Figure 4.2: All eight camera angles were used as one of the test parameters. This example shows the chameleon dataset, but the same angles were used for all other datasets and render modes. All Screenshots were taken by MegaMol (volume renderer set to Isosurface) right before starting a test case.



(a) ace_drop_1m (no thinning)

(b) expl30m (no thinning)

(c) ace_drop_1m (thinned by 4)

(d) expl30m (thinned by 4)

(e) ace_drop_1m (thinned by 16)

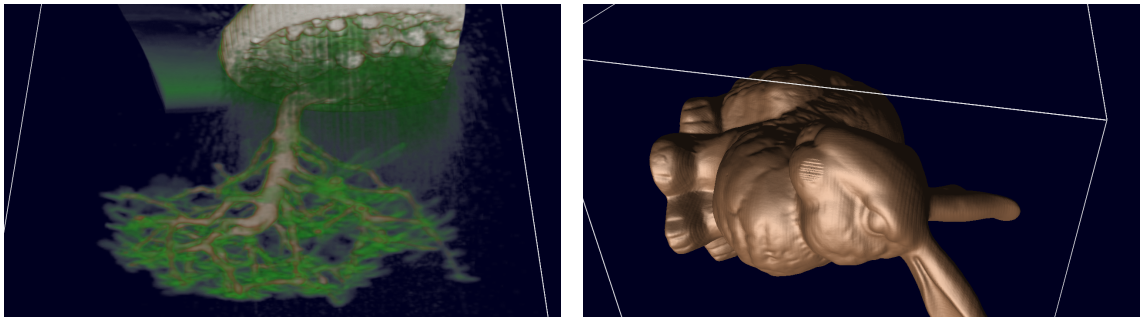
(f) expl30m (thinned by 16)

Figure 4.3: Screenshots of both spheres datasets used as one of the test parameters and visualized by MegaMol's sphere renderer. Each row shows them at a different thinning factor (which is another test parameter set to either 1 (top), 4 (middle) or 16 (bottom)). All screenshots were taken by MegaMol right before starting a test case.

the whole data, resulting in technically unused data in the VRAM. Fig. 4.3 shows both datasets on various thinning factors visualized in MegaMol's sphere renderer. **Sphere radius** determines the radius of the rendered spheres. Larger spheres will likely lead to more overdraw in already dense areas.

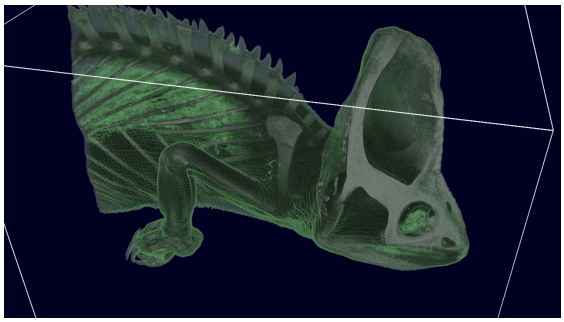
For the volume test cases, there are also exclusive render methods, datasets and one additional test parameter:

The selection of volumetric data consists of three cubic **datasets** with increasing size: bonsai (256^3), bunny (512^3) and chameleon (1024^3). The two **render methods** Integration and Isosurface cause the renders to be visually distinct, as seen in Fig. 4.4. On top of these differences, shadows are turned off for Integration (variance in colour is only a result of the transfer function and potential

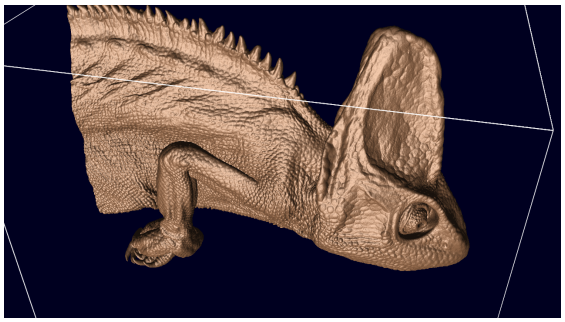


(a) Bonsai (Integration)

(b) Bunny (Isosurface)



(c) Chameleon (Integration)



(d) Chameleon (Isosurface)

Figure 4.4: Screenshots of all volume datasets used as one of the test parameters visualized by MegaMol's volume renderer (set to either Integration (left) or Isosurface (right)). All screenshots were taken by MegaMol right before starting a test case.

transparency).
Similar to the thinning factor in the spheres test cases, there is **step ratio** for volume rendering, which determines how many rays are cast per voxel. A step ratio of n results in only every nth voxel sampled.

4.4.2 Operation

As already discussed in 4.4.1, the choice of GPU has to be our outermost test parameter, as swapping GPUs is not (realistically) automizable, so the benchmarking process has to start with physically installing the correct GPU. After the GPU is installed and secured (with a screw and support for its "tail end"), the power and display output (2160p@60Hz) plugs can be connected. Then, the PSU's power plug can be plugged in, and the system can be started to boot into Windows. On the desktop, specific explorer windows and the Task Manager were opened (and positioned consistently across multiple runs) to confirm that everything was running correctly regularly. Fortunately, this ends the manual part of the benchmark, as the only step left is to launch the batch script that handles the rest automatically.
The batch script launches megamol.exe four times (the spheres and volume project files with V-Sync enabled and disabled). Both project files are of similar structure but differ significantly in the sections where the MegaMol graph is set up, and unapplicable test parameters are handled.

As it is not required for automated test cases and could get in the way, the GUI is hidden after the project is fully set up, and a variable for storing the output file is created. Next is a nested for loop spanning all the relevant test parameters in the following order:

- Spheres: dataset - thinning factor - render method - sphere radius - resolution - camera angles
- dataset - render method - step ratio - resolution - camera angles

The datasets are first because loading them has the potential to take longer than changing other parameters, and not having a dataset selected in the beginning causes problems when changing renderer-specific parameters (sphere radius, step ratio). In our case, the thinning factors are already encoded within the datasets' files. When changing the render method within the volume project, the correct transfer function is applied for Integration and the correct iso value for Isosurface. Both are predefined, the bunny dataset reuses the chameleon's transfer function and suiting iso values for each dataset were selected empirically (testing different values until only the desired parts of the volume were rendered).
The following happens within the innermost loop: First, a name for identifying the current test cast is generated by concatenating the various test parameter values (excluding GPU and V-Sync state). After slightly modifying this name to be a viable Windows file name, a screenshot of MegaMol rendering the current test case is taken for later verification. Next, ten frames are rendered to eliminate power fluctuations observed when starting kernels/engaging loads. After this step, the actual benchmark begins. It consists of four steps:

- A line consisting of the test case name, "start" and the current timestamp (as provided by mmGetPowerTimeStamp()) is appended to the output file.
- The function render() is called. This function was defined elsewhere in the script and will render (via mmRenderNextFrame()) and count a new frame until the given time (5 seconds in our case) is over. It uses the system clock via Luas os library, which is accurate to the millisecond. However, the exact runtime might not be accurate to the millisecond, as the rendering of the current frame is not terminated immediately after reaching 5000 ms (the frame is always rendered to completion, and the time is only checked to determine if another frame should be rendered). When the rendering is done, the number of rendered frames is returned.
- Another line is appended to the output file. This time, with a new timestamp, "end" instead of "start" and the number of rendered frames. These two lines allow us to accurately determine the test case's runtime and, as the total number of frames is also known, to compute the average FPS of the test case.
- mmFlushPowerlog() is called to write all values our Powerlogging_Service sampled in the background while running the test case to their respective files. We can later use the start and end timestamps to select the relevant sensor samples for this test case.

After the outermost loop and all test cases are completed, the output file is written to disk (the test bench has more than enough memory to not require buffered writing for <1 MB of text). Finally, MegaMol is closed (mmQuit()), and the next line in the batch script is executed.
The only manual step after the batch file has finished is to move all data to another folder and continue with the next GPU.

4.4.3 Collected Data

Although all sensors are configured within the Powerlogging_Service to be polled asynchronously every 5 ms, the exact interval for each sensor is neither consistently in this range nor consistent across multiple sensors. Tinkerforge samples were acquired every 5ms on average. However, less than 0.005% of the samples' timestamps were off by approximately 5000ms, which is one test case. The exact cause of this phenomenon is unknown, but as they were always singular stray samples, they should not have interfered too much with the outcome of a test case, as, on average, there were 999 valid samples to compensate for them. Ignoring these stray samples results in a standard deviation of approximately 2 ms and 99.99% below 30 ms. ADL and NVML performed similarly with a sample every 16 ms on average, a standard deviation of approximately 1.5 ms and 99.99% below 40 ms. This lack of consistency makes it hard to directly compare samples from one sensor to another because the other sensor might not have a sample at the given timestamp. Instead of significantly reducing the temporal resolution (potentially until no data is left) by omitting all samples at timestamps not covered by every sensor, all samples are quantized at an interval of 1 ms (1000 Hz). No data is lost because this is above twice the maximum possible sample frequency

$$5\,ms \implies 200\,Hz < 500\,Hz = 1000\,Hz * 0.5) \tag{4.1}$$

No dedicated interpolation is performed before or after the quantization step. The quantization iterates over every timestamp t and checks if its value P_t exists. If it does, the last known value P_{last} is updated; if not, the current timestamp is assigned the last known value. There always is a P_{last}, as the first timestamp, by definition, always has a value.

$$P_t, P_{last} = \begin{cases} P_t, & \text{if } P_t \neq \text{null} \\ P_{last} & \text{otherwise} \end{cases} \tag{4.2}$$

After quantization, samples from multiple sensors can be (re)combined to produce more meaningful power readings.
As a first step, some power connectors consisting of multiple power lines (with potentially different voltages) can be recombined for some basic building blocks:

$$\begin{aligned} P_{CPUP} &= P_{P4} + P_{P8} \\ P_{ATX} &= P_{ATX3.3V} + P_{ATX5V} + P_{ATX12V} \\ P_{PEG} &= P_{PEG3.3V} + P_{PEG12V} \end{aligned} \tag{4.3}$$

These building blocks already cover all power sources for every component on the motherboard (except the GPU). P_{PEG} is a reading of how much of P_{ATX} is used by the GPU. As from a "system total perspective", P_{PEG} is already included in P_{ATX} and must be subtracted here:

$$P_{System\backslash GPU} = P_{CPUP} + P_{ATX} - P_{PEG} \tag{4.4}$$

It could be interesting to split up P_{System} further into hypothetical values for $P_{CPU}, P_{RAM}, P_{Chipset}$ and more. However, as our setup has no way of determining how P_{ATX} is divided among different components (except for the GPU via P_{PEG}), we have to live with only isolating the GPU's power draw from the rest of the system. To get P_{GPU}, we first have to add up all PCIe power sensors:

$$P_{PCIe} = \sum_{i=1}^{9} \begin{cases} P_{PCIe_i}, & \text{if } x < \infty \\ 0 & \text{otherwise} \end{cases} \tag{4.5}$$

Note that while there are nine separate power bricklets/sensors dedicated to PCIe power, there are currently no cases where all of them are simultaneously used for a single GPU because the first three bricklets cover one 150 W connector each (most GPUs do not utilize them all) while the rest covers the single 600 W connector (see Fig. 4.1). If a sensor is not used, every sample will have a value of ∞, which is used as a criterion to filter out the sensor entirely (one ∞-sample is enough; this can be done before quantization to save time).
Both of the GPU's power sources are covered now:

$$P_{GPU} = P_{PEG} + P_{PCIe} \tag{4.6}$$

With both $P_{System\backslash GPU}$ and P_{GPU} known, it is possible also to estimate the system's total power draw:

$$P_{System} = P_{System\backslash GPU} + P_{GPU} \tag{4.7}$$

Note that P_{System} is not equivalent to the power draw of the testbench when measured from the wall outlet, as the PSU generally does not operate without losses. Additionally, there is a mostly idling SSDs to consider, which has its own power connector. However, it is omitted in this setup to remove clutter and because of its lack of significance to the total power draw (likely <1W). The NVMe-SSD where MegaMol is running from and where the benchmark data is written should be comparably insignificant, but it is conveniently covered by P_{ATX} and hence included in $P_{System\backslash GPU}$.

For some later analysis steps, the P_{GPU} was normalized to the GPU TDP to give an approximation of the GPU's utilization and allow for a better comparison between GPUs with different TDPs:

$$util_{GPU_i} = \frac{P_{GPU_i}}{P_{TDP_{GPU_i}}} \tag{4.8}$$

As alternatives to P_{GPU}, the power samples from the GPUs' internal power sensor are also recorded, which gives us P_{ADL} on AMD GPUs and P_{NVML} on Nvidia GPUs (there is currently nothing comparable for Intel GPUs). These values offer the benefit of being measurable "out of the box", i.e. without physical modifications to the hardware while having lower temporal resolution and

less accuracy than the tinkerforge setup. Our primary use for them is evaluating their accuracy compared to the tinkerforge samples while also serving as an overall ”sanity check”, if available. The NVML samples can be directly interpreted as P_{NVML}, and no post-processing to improve the values further, as suggested by Burtscher et al. [BZZ14] was attempted. The reasons for not trying to compensate for these artefacts resembling capacitor (de)charging include a lack of clarity about the applicability of these compensation suggestions, initially formulated for Kepler GPUs, to the more modern architectures of the tested Nvidia GPUs (we do not even have proof[2] of them still existing). As there was no suggestion for improving ADL sensor values, the compensation would also make the GPU sensor results across different GPU manufacturers less comparable. On the other hand, the ADL samples require minor post-processing to get a P_{ADL} that can be compared to the other sensors' values without problems:
Different ASICs within the AMD GPUs are recognized as different ”software sensors”, and each one provides its values paired with a timestamp (timestamps for other sensors are generated on acquisition within the power library). However, the values they provide appear to originate from the same physical sensor because only their timestamps vary slightly (sampling the physical sensor at different times), while the overall values match across all ”software sensors”. For the sake of simplicity, the results from all ”software sensors” but those originating from the ASIC recognized as ”0” are discarded.

$$P_{ADL} = P_{ADL_0} \tag{4.9}$$

Additionally, the ADL timestamps occur approximately 2 hours later than other sensors. When running the benchmark, the system clock was set to Central European Summer Time, which is UTC+2. The ADL timestamps are seemingly UTC±0, so they must be shifted by 2 hours to align with the rest.

$$t_{ADL} = t_{ADL} - 72000000 \text{ ms} \tag{4.10}$$

On top of the power samples (with their timestamps) from the various sensors (ADL, NVML, Tinkerforge), the start and end timestamps of each test case and the number of rendered frames within these timestamps are recorded as well. Because MegaMol does not get restarted after every test case (only to change between sphere/volume rendering and to turn vertical synchronization on/off), the logging produces a single stream of samples, necessitating these start/end timestamps (t_{start}, t_{end}) to assign samples to test cases correctly. If a sample s_t has a timestamp t between t_{start} and t_{end}, it is assigned to the test case T:

$$t_{start} \leq t < t_{end} \implies s_t \in T \tag{4.11}$$

[2] Müller et al. [MHWE22] (Fig. 9) showed that the NVML sensor on the Nvidia 3090 still has some vaguely similar behaviour, but this was on the browser rendering tests and the sections in question looked more like linear than exponential growth/decay.

Not every sample is necessarily assigned to a test case because, as the logging is active while MegaMol is running, samples still get corrected in between test cases (when MegaMol parameters are changed, dummy frames are rendered, screenshots are taken or powerlog buffers are flushed). However, no sample can be part of two different test cases.

The start/end timestamps, in combination with the number of rendered frames per test case, also allow us to calculate the average FPS throughout the test case, which is a more tangible metric (especially regarding VSync):

$$\frac{n_{\text{frames}}}{t_{\text{end}} - t_{\text{start}}}\ [\text{s}^{-}1] = \text{avg. FPS}\ [\text{s}^{-}1] \tag{4.12}$$

The abovementioned metrics can also be used to calculate efficiency in J/frame. It is especially useful when comparing GPUs on different ends of the computing power spectrum because it estimates how effectively the consumed power is used. To save time, $avg.\ FPS$ can be used alternatively:

$$\frac{\text{P}_{\text{mean}} \cdot (t_{\text{end}} - t_{\text{start}})}{n_{\text{frames}}}\ [\text{W s}] = P_{\text{mean}} \cdot (\text{avg. FPS})^{-1}\ [\text{W s}] = \text{J/frame}\ [\text{J}] \tag{4.13}$$

5 Results

For the sake of simplicity, any GPU will typically be referred to by a combination of only manufacturer and specific model name (omitting "model line"names like "Radeon RX", ÄRC", and "GeForce RTX"). In some cases, all GPUs by the same manufacturer or GPU generation behave similarly. However, this does not imply that all GPUs by this manufacturer or within this generation behave this way, only the ones examined within this work (see 4.1).
While the same metrics are recorded for both spheres and volume test cases, their vastly different implementations and differences in test parameters (see 4.4.1) render them unsuitable for equal comparisons. Hence, they will be analyzed separately, except for some general observations which do not rely on the individual influences of specific test parameters. However, the analysis itself will be similar for both types of test cases, as it will first look into the causes of different FPS and power draw readings (because both are required to compute efficiency) and then analyze efficiency in J/ per frame.
In certain sections, scatterplots with highlighting for various test parameters are used. These are excerpts from a more extensive table of scatterplots, which can be found in the appendix (B).

5.1 Spheres

When using the Intel A770 and set to SSBO stream, MegaMol's sphere renderer crashed because the driver did not accept the set buffer size. Intel's OpenGL implementation does not follow the standard specifications at specific points (most notably for our case: default maximum SSBO buffer size). Earlier, there were problems with the volume renderer always showing an empty volume when using the Intel A770. However, in contrast to these problems, the issue with the SSBO renderer could not be fixed without significantly altering the renderer. In MegaMol's current state, it is possible at least to run the SSBO renderer with the Intel A770, but thinning does not work correctly yet (it is uneven and whole "blocks" are removed instead of every *n*th particle) and enabling static data breaks it even further. Because of this, only test cases of the Intel A770 with the simple renderer are included, and the Intel A770 will be ignored when discussing SSBO with static data.

5.1.1 Performance

First, we compare the average performance (measured by the average number of FPS) of the different render modes (Simple and SSBO stream with static data) across the different GPUs and resolutions. For this purpose, all spheres test cases on the same GPU with equal resolution, render mode and V-Sync state are averaged (see Fig. 5.1 and 5.2). Averaging over the different camera angles and especially the different datasets in their various stages of thinning aims to paint a general picture of the GPUs' performance (detailed analysis of their influences follows in the next subsection).
Comparing the values for the different render methods in Fig. 5.1 and 5.2 reveals that for every case,

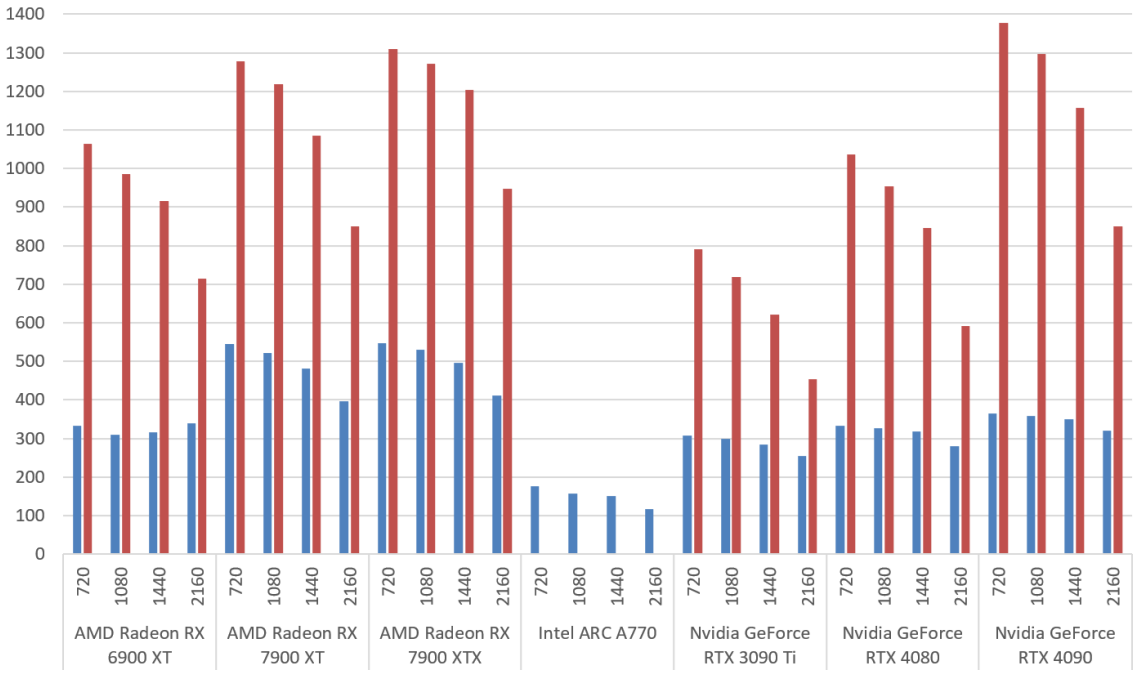


Figure 5.1: Mean FPS by GPU and resolution. **left/blue** is **simple** renderer, **right/red** is **SSBO stream** renderer with static data enabled. V-Sync is **disabled**. Each column is composed of the mean of the FPS of all spheres test cases that match these parameters.

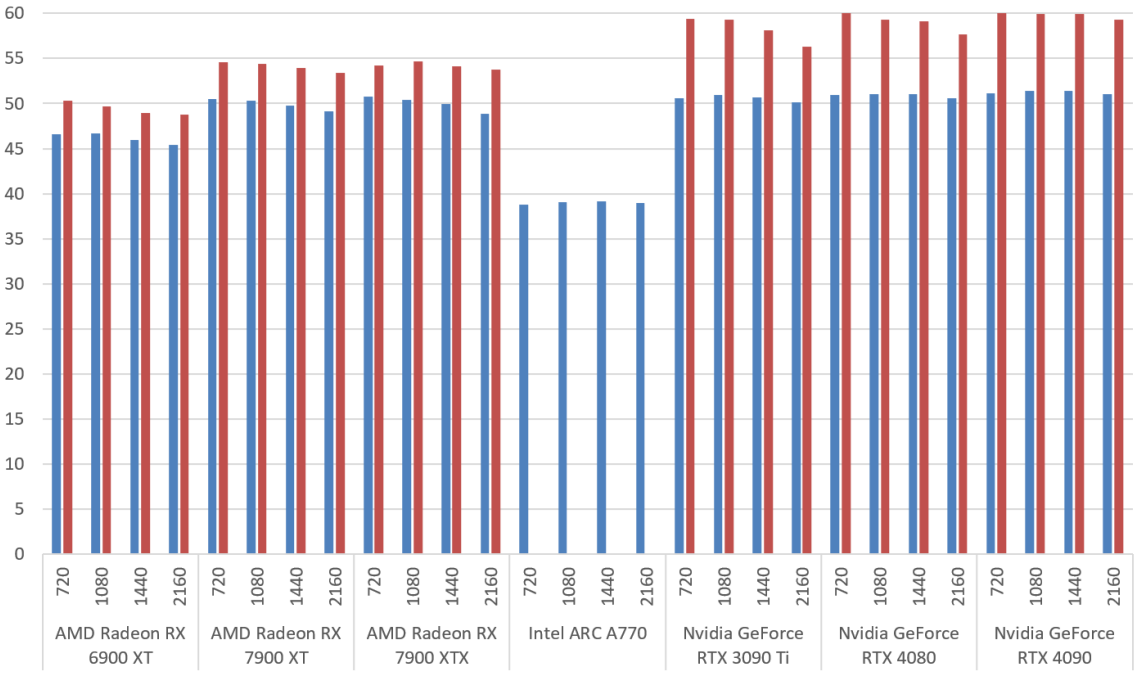


Figure 5.2: Mean FPS by GPU and resolution. **left/blue** is **simple** renderer, **right/red** is **SSBO stream** renderer with static data enabled. V-Sync is **enabled**. Each column is composed of the mean of the FPS of all spheres test cases that match these parameters.

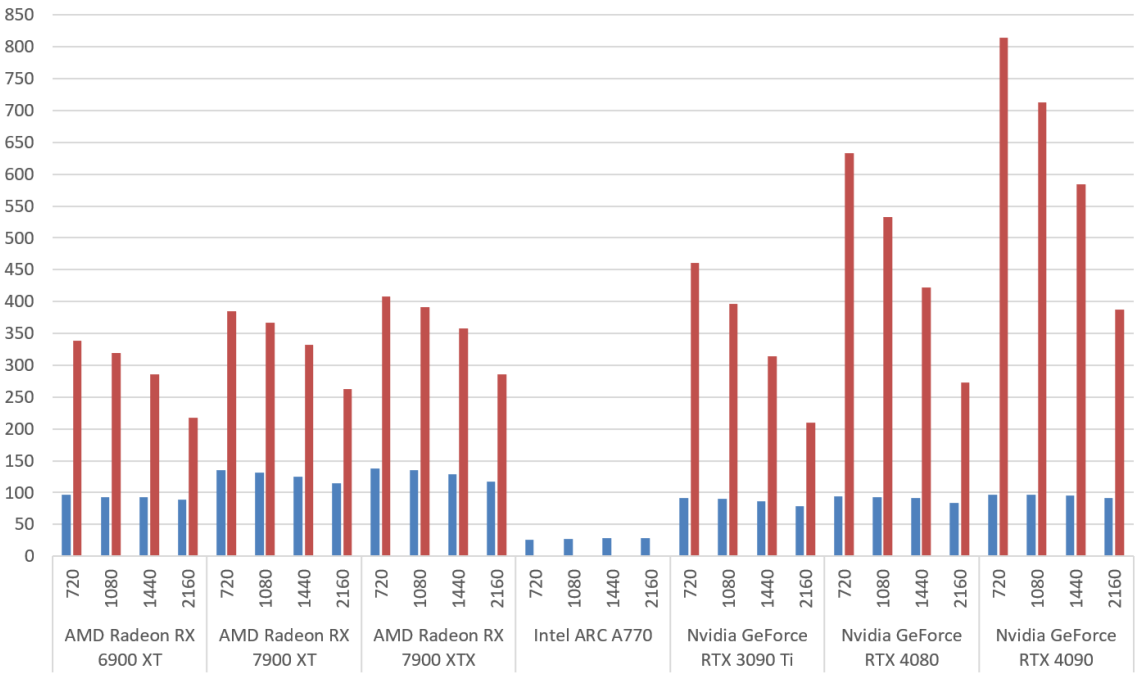


Figure 5.3: Mean FPS by GPU and resolution. **left/blue** is **simple** renderer, **right/red** is **SSBO stream** renderer with static data enabled. V-Sync is **disabled**. **Only cases with the larger expl30m dataset** are included. Each column is composed of the mean of the FPS of all spheres test cases that match these parameters.

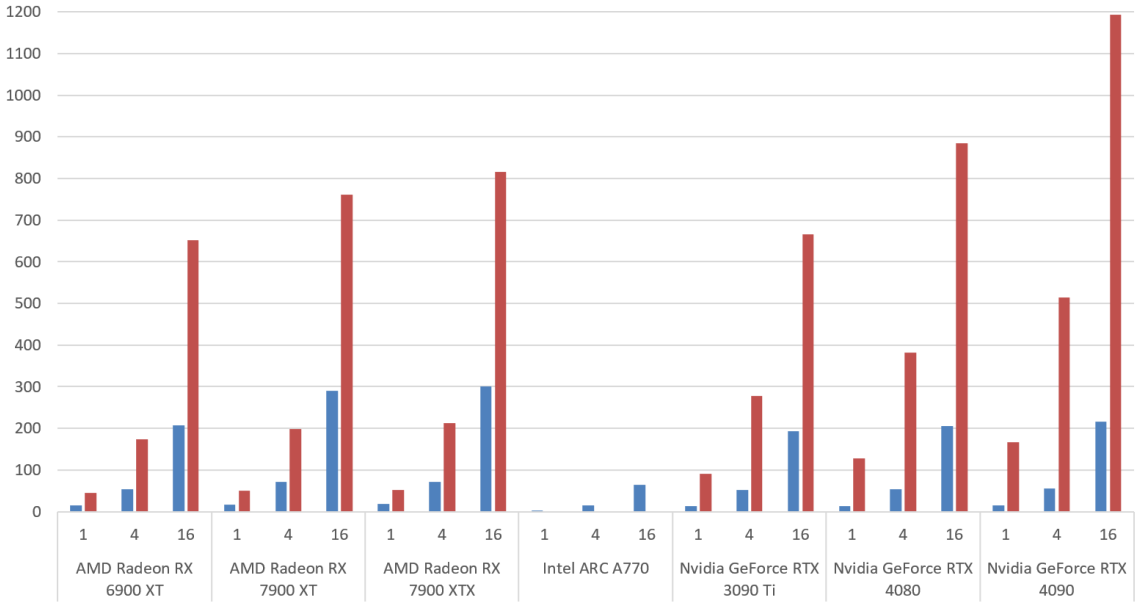


Figure 5.4: Mean FPS by GPU and thinning factor. **left/blue** is **simple** renderer, **right/red** is **SSBO stream** renderer with static data enabled. V-Sync is **disabled**. Only cases with the **larger expl30m dataset** are included. Each column is composed of the mean of the FPS of all spheres test cases that match these parameters.

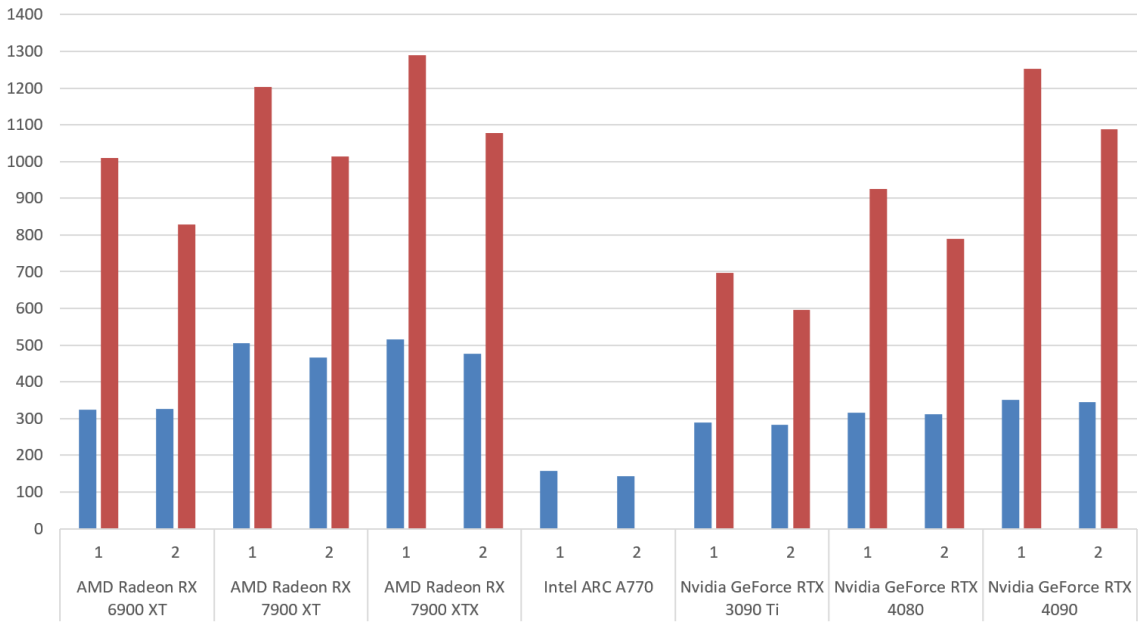


Figure 5.5: Mean FPS by GPU and sphere radius. **left/blue** is **simple** renderer, **right/red** is **SSBO stream** renderer with static data enabled. V-Sync is **disabled**. Each column is composed of the mean of the FPS of all spheres test cases that match these parameters.

FPS with SSBO static data are consistently and significantly higher than with the simple renderer. The unlimited FPS (Fig. 5.1), in particular, are always more than twice as high. With V-Sync (Fig. 5.2), SSBO static data delivers respectively 10% (AMD) and 20% (Nvidia) more FPS compared to simple.
For the simple renderer, the GPUs rank in the following order: AMD 7000-series > Nvidia GPUs and AMD 6900 XT > Intel A770. Despite their comparably lower performance on paper, the AMD GPUs outperform their Nvidia counterparts when using the simple renderer (even when exclusively looking at the larger expl. dataset 5.3).
Changing the renderer yields the following order: Nvidia 4090 > AMD 7000-series > Nvidia GPUs and AMD 6900 XT. This order also does not match the theoretical raw power of the GPUs, especially with the Nvidia 3090 Ti performing noticeably worse than the AMD 6900 XT. Eliminating the influence of the smaller ace drop dataset, Fig 5.3 reveals that the theoretically more powerful Nvidia GPUs mostly take the lead for the larger and denser data. However, the Nvidia GPUs seem to have more pronounced scaling with the resolution and especially at 2160p, their AMD counterparts hold up reasonably well. The cause of this is likely not the AMD GPUs being underutilized at low resolutions but rather a case of the Nvidia GPUs being underutilized for smaller and less dense datasets. Fig. 5.4 provides evidence of this, as, even with only the expl. dataset, the AMD GPUs show higher relative performance increases with higher thinning factors (more robust evidence provided in the scatterplots).
Filtering for sphere radius will not change any of the orders but does still have a significant impact on performance: Fig. 5.5 consistently shows a general decrease in performance for the larger sphere radius, although less noticeable with the simple renderer (except AMD 7000-series GPUs).

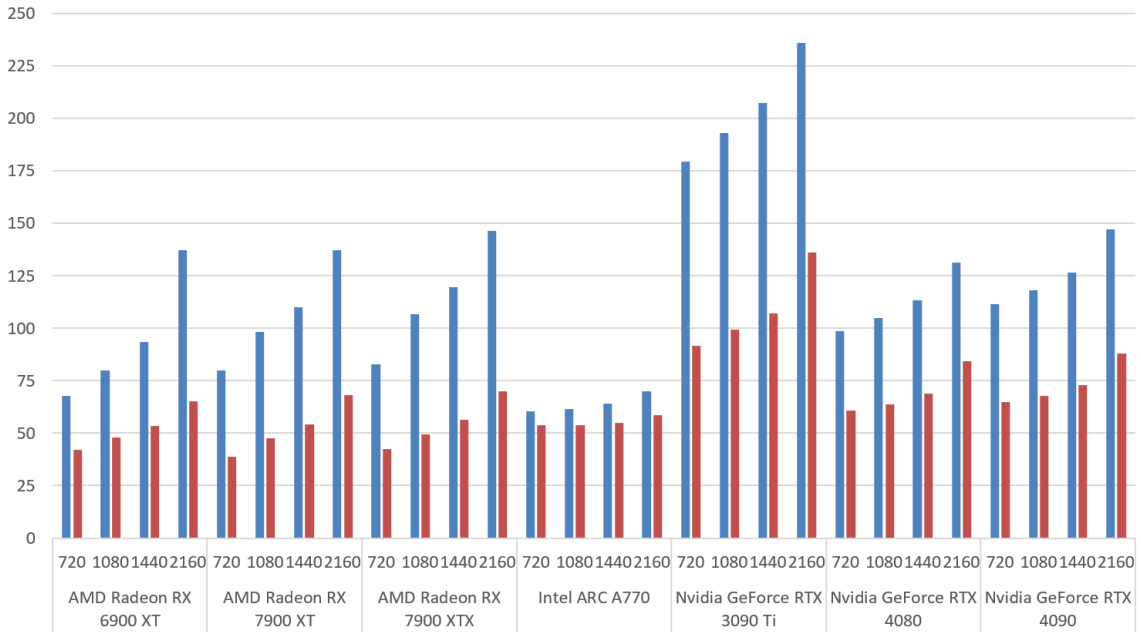


Figure 5.6: Mean GPU Power (W) by GPU and resolution. **left/blue** is with **V-Sync disabled** and **right/red** with **V-Sync enabled**. Only test cases utilizing the **simple renderer** are included. Each column is composed of the mean of the mean power values of all spheres test cases that match these parameters.

Additionally, Nvidia GPUs seem less impacted by sphere radius but not by orders of magnitude. Also, the margin of the sphere radius for all GPUs seems to stay relatively equal across different resolutions and thinning factors.

5.1.2 Power Consumption

Similar to the performance analysis, we start with looking at the general influences of the different parameters, but on the mean power consumption of the GPU (as measured via the tinkerforge setup) this time. Fig. 5.6 shows the GPUs' mean power consumption when using the simple renderer. Most GPUs score within a similar range, with the only apparent outliers being the Nvidia 3090 Ti as the highest and the Intel A770 as the lowest. It should be noted that, although taking similar power at higher resolutions, the AMD GPUs take 20 W less at 720p than the Nvidia 4000 GPUs. Independent of V-Sync being enabled or disabled, power consumption increases with higher resolutions. This effect seems less pronounced on the Intel A770, making it more expensive at lower resolutions with enabled V-Sync than the AMD GPUs.
Regarding the other render mode, Fig. 5.6 looks similar for enabled V-Sync, with only the Nvidia 3090 Ti sticking out as the highest. However, looking at the values for disabled V-Sync paints a different picture, as the order now changes to the following: AMD 7900 XTX > Nvidia 3090 Ti > AMD 7900 XT > AMD 6900 XT + Nvidia 4090 > Nvidia 4080. Neighbouring GPUs in this order are reasonably close in value and significantly overlap.
As seen in Fig. 5.8, the relative power step from V-Sync disabled to enabled seems to increase for cases that produce higher FPS. A possible interpretation would be that the GPUs generally save

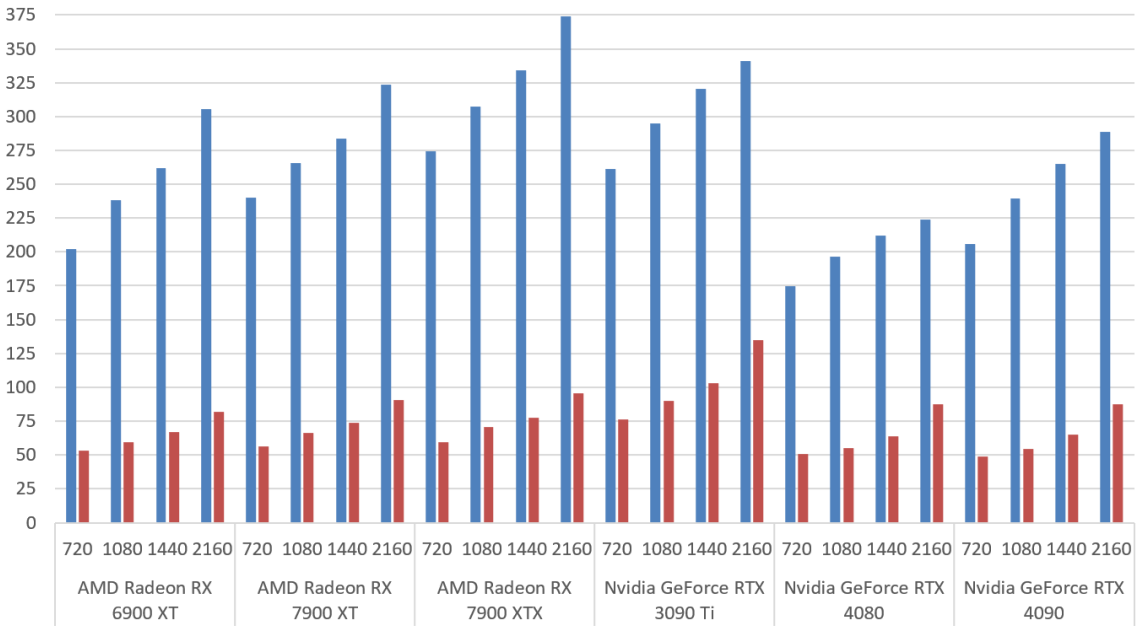


Figure 5.7: Mean GPU Power (W) by GPU and resolution. **left/blue** is with **V-Sync disabled** and **right/red** with **V-Sync enabled**. Only test cases utilizing the **SSBO stream renderer** (with static data enabled) are included. Each column is composed of the mean of the mean power values of all spheres test cases that match these parameters.

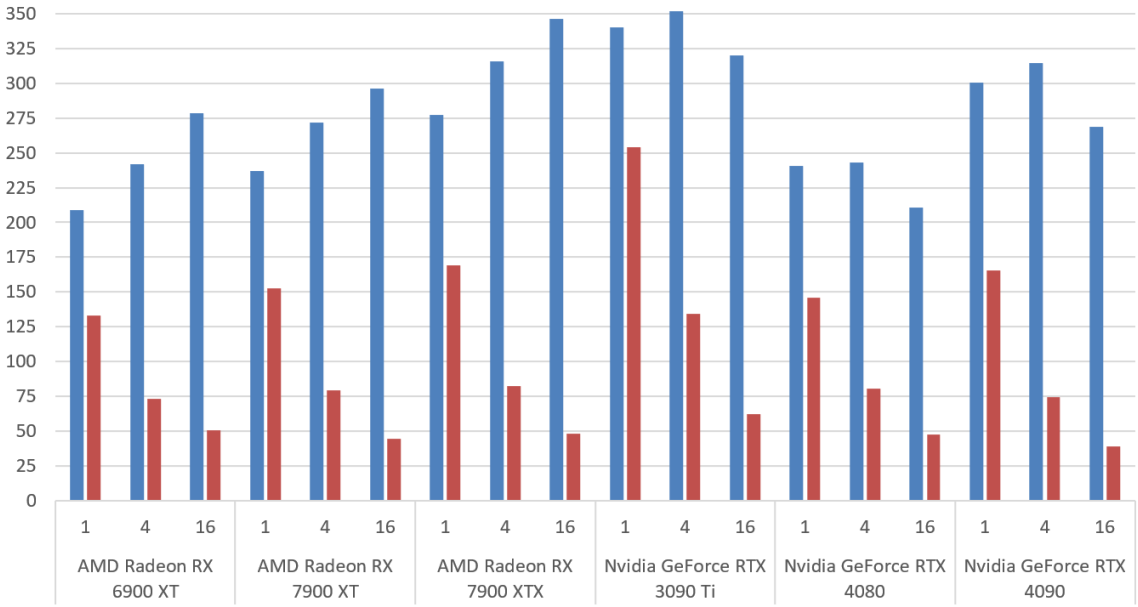


Figure 5.8: Mean GPU Power (W) by GPU and thinning factor. **left/blue** is with **V-Sync disabled** and **right/red** with **V-Sync enabled**. Only test cases with the **larger expl30m dataset** and utilizing the **simple renderer** are included. Each column is composed of the mean of the mean power values of all spheres test cases that match these parameters.

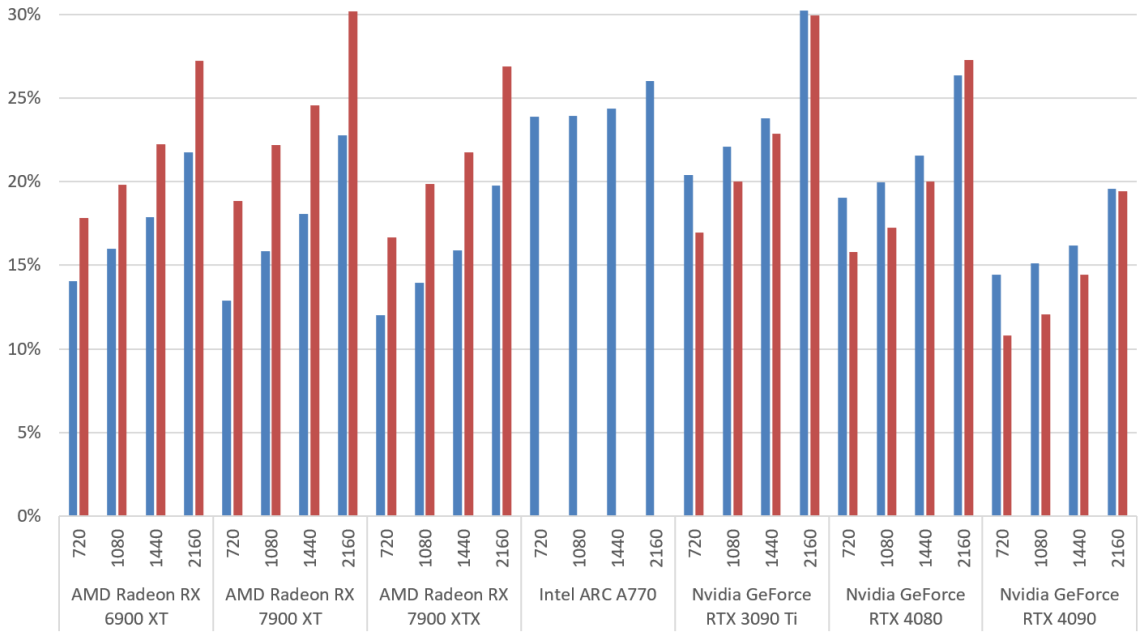


Figure 5.9: GPU Mean Power (normalized to their respective TDP) by GPU and resolution. **left/blue** is **simple** renderer, **right/red** is **SSBO stream** renderer with static data enabled. V-Sync is **enabled**. Each column comprises the mean of the mean utilization of all spheres test cases that match these parameters.

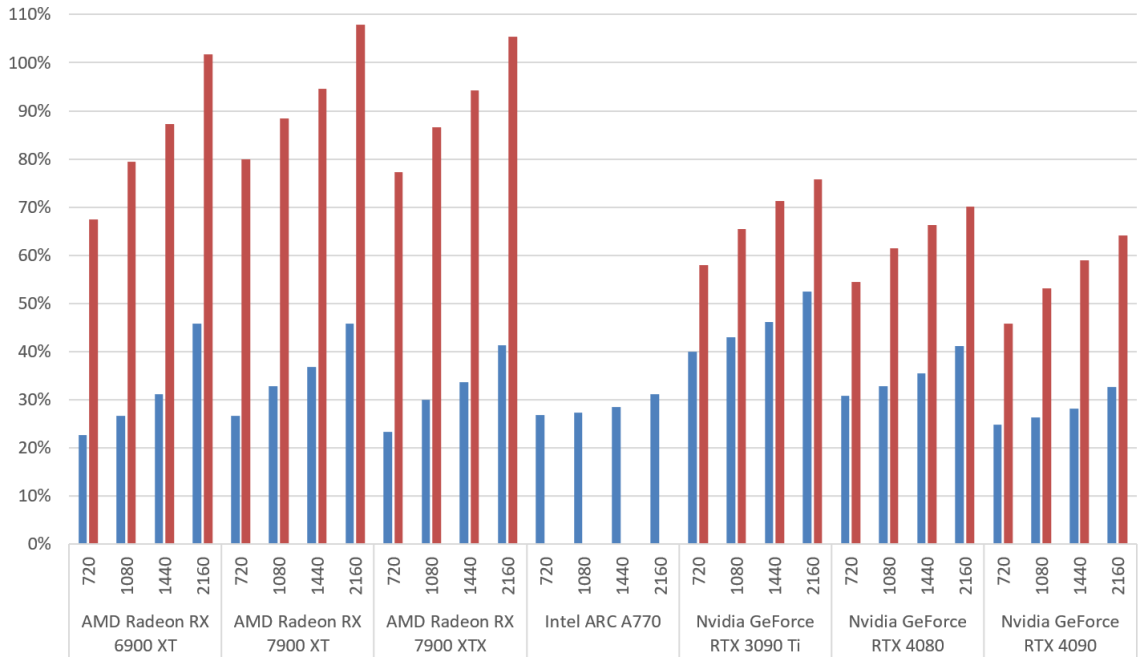


Figure 5.10: GPU Mean Power (normalized to their respective TDP) by GPU and resolution. **left/blue** is **simple** renderer, **right/red** is **SSBO stream** renderer with static data enabled. V-Sync is **disabled**. Each column comprises the mean of the mean utilization of all spheres test cases that match these parameters.

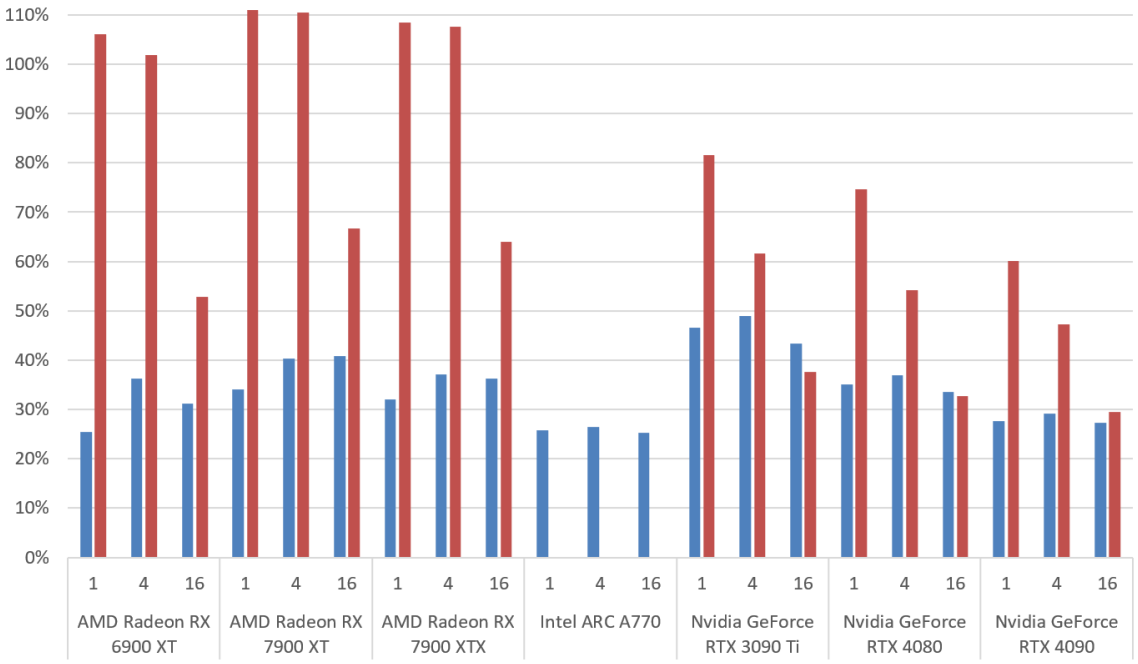


Figure 5.11: GPU Mean Power (normalized to their respective TDP) by GPU and thinning factor. **left/blue** is **simple** renderer, **right/red** is **SSBO stream** renderer with static data enabled. V-Sync is **disabled**. Only test cases with the **smaller ace_drop_1m dataset** are included. Each column comprises the mean of the mean utilization of all spheres test cases that match these parameters.

more power by the more frames they do not have to render. However, while it holds for the small dataset or increasing the thinning factor, lowering the resolution does not increase the gap.
The graphs of both render modes show that the GPUs' ranking order is off from their order by TDP, implying that their utilization must be below 100% and varies by GPU. To approximate the GPU utilization within a test case, the test case's mean power is divided by the respective GPU's TDP. This approach is not without flaws because different manufacturers interpret the TDP differently, but it still allows for a more evenly matched comparison across different GPUs.
Fig. 5.9 shows how the average utilization of all GPUs when using V-Sync is below 30% and, for some, drops below 15% at lower resolutions. For AMD GPUs, SSBO with static data is consistently more expensive. At the same time, the simple renderer is more expensive for Nvidia GPUs (except for 2160p, where all GPUs notably struggle to maintain a stable 60 FPS in every test case). However, it should be noted that the differences on AMD GPUs disappear when only looking at the smaller ace_drop350k dataset, and the differences on Nvidia GPUs are less pronounced on the larger expl30m dataset.
When V-Sync is disabled, SSBO with static data is, on average, more expensive for all GPUs, as shown by 5.10. Their utilization with the simple renderer is mostly below 50%, being the highest on the Nvidia 3090 Ti and the lowest on both the Nvidia 4090 and the Intel A770, with mostly less than 33%. This low utilization indicates a bottleneck caused by having to retransfer the data to the GPU on every frame.
When using SSBO with static data, AMD GPUs generally get more than 70% and even reach full utilization at higher resolutions, while Nvidia GPUs reach less than 60-75% (even less the stronger

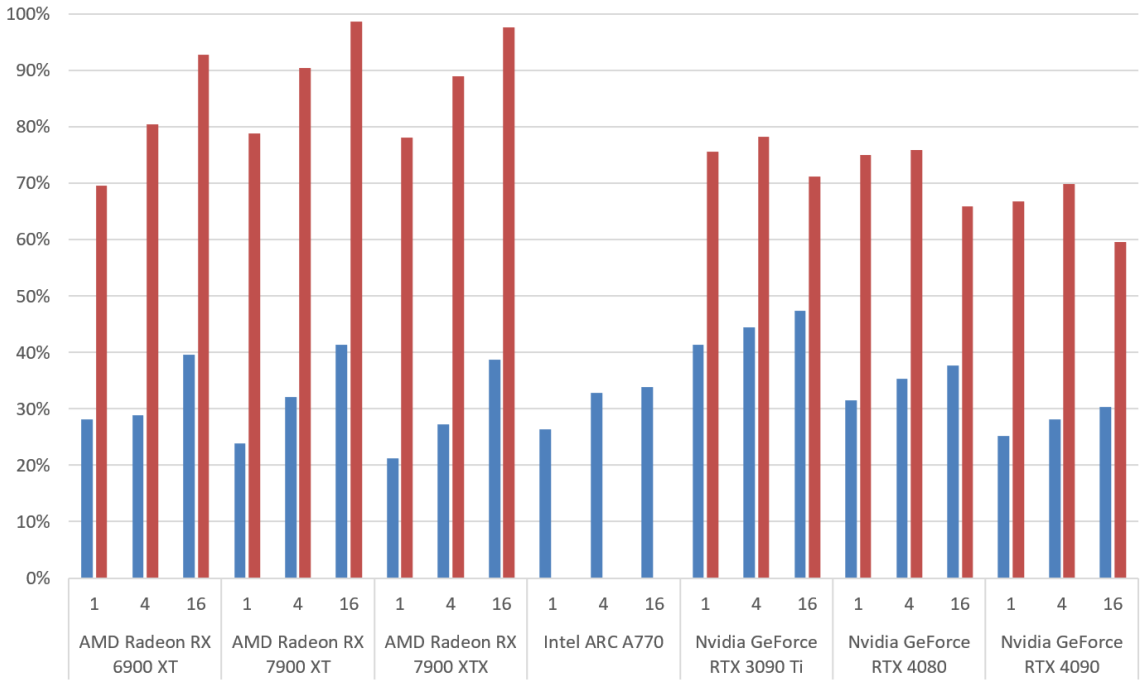


Figure 5.12: GPU Mean Power (normalized to their respective TDP) by GPU and thinning factor. **left/blue** is **simple** renderer, **right/red** is **SSBO stream** renderer with static data enabled. V-Sync is **disabled**. Only test cases with the **larger expl30m dataset** are included. Each column is composed of the mean of the FPS of all spheres test cases that match these parameters.

the GPU gets). Utilization generally increases with higher resolutions, independent of the render mode. The utilization gap between the render modes is more pronounced on AMD GPUs.
To get a good picture of the influence of thinning factors on the utilization, the two datasets are better viewed separately: Fig. 5.11 shows that when the dataset gets too small, another bottleneck appears (even though these are the cases where the highest FPS are produced). On the other hand, as seen in Fig. 5.12, the utilization for larger datasets generally benefits from higher thinning factors. Only Nvidia GPUs seem to stagnate across different thinning factors when using SSBO with static data. One possibility is that, even without thinning, the dataset is still too small to cause a difference on Nvidia GPUs.
As for the influences of the sphere radius, Fig. 5.13 shows approximately 10% more utilization on AMD GPUs with the larger spheres when using the simple renderer. This increase is also visible for all other GPUs but is less pronounced. For SSBO with static data, there is no significant difference caused by the sphere radius (if at all, a slight increase on Nvidia GPUs).

After establishing these general influences of the different test parameters, we shift the focus to analyzing the influence of the parameters on the outcomes of the individual test cases and their spatial relation to each other. For this purpose, we use plots where the axes are the earlier discussed mean FPS and GPU power (each on its axis). Additionally, as dividing mean power by FPS yields performance per Watt, all points on a line intersecting with the origin imply equal efficiency. Because the GPUs of the same manufacturer (quite literally) paint a similar picture, we will primarily

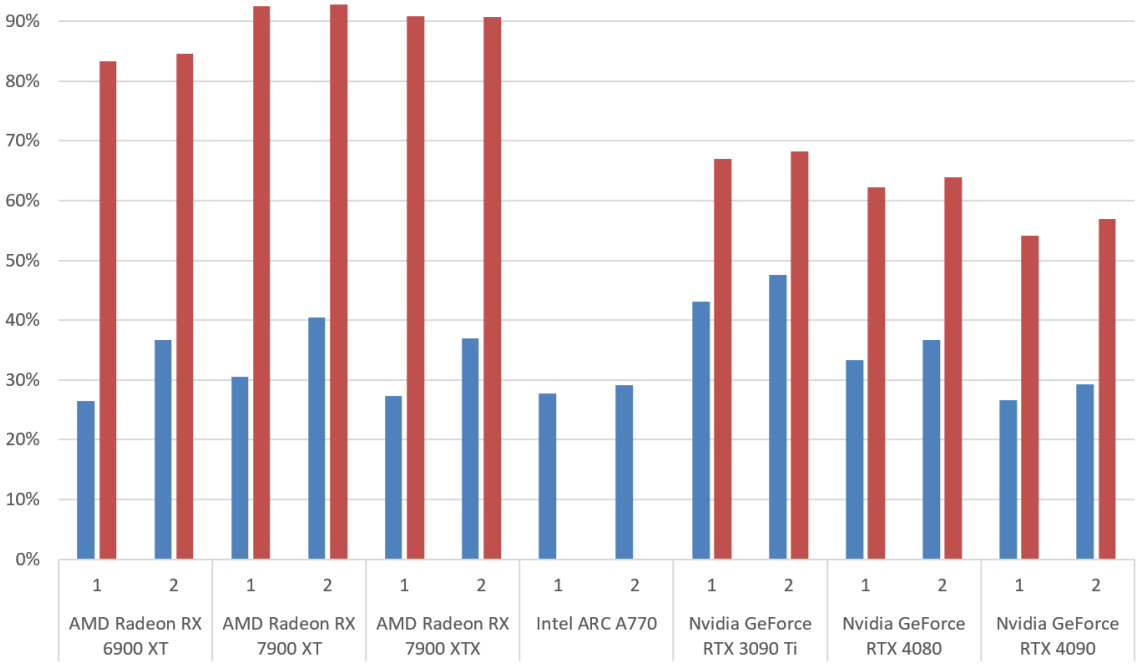
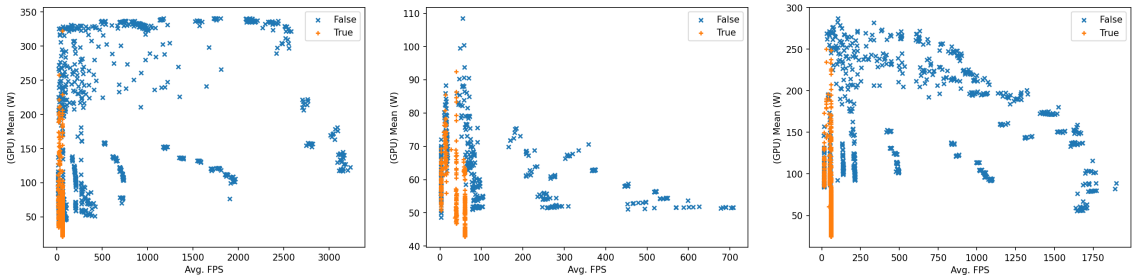
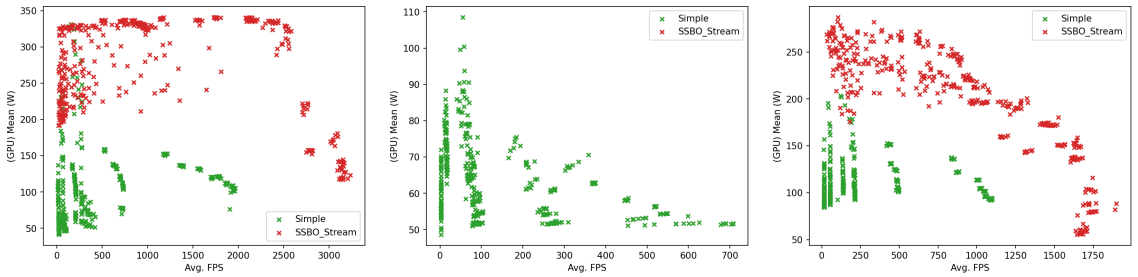


Figure 5.13: GPU Mean Power (normalized to their respective TDP) by GPU and sphere radius. **left/blue** is **simple** renderer, **right/red** is **SSBO stream** renderer with static data enabled. V-Sync is **disabled**. Each column comprises the mean of the mean utilization of all spheres test cases that match these parameters.



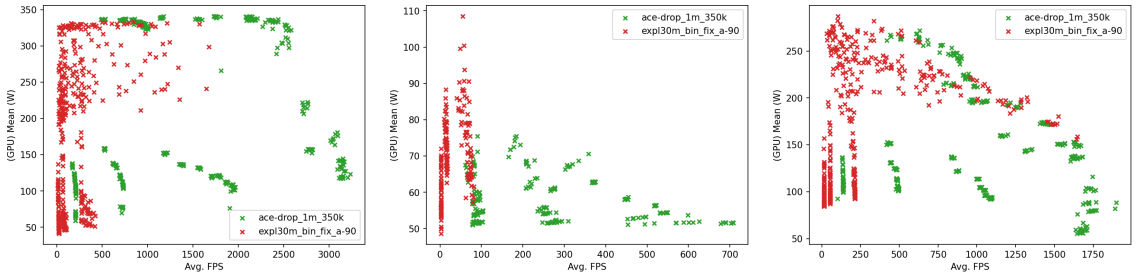
(a) AMD 7900 XT **(b)** Intel A770 **(c)** Nvidia 4080

Figure 5.14: Scatterplot of Mean GPU Power (W) and Mean FPS. Each point is one spheres test case with different parameters. Different **states of V-Sync** are highlighted.



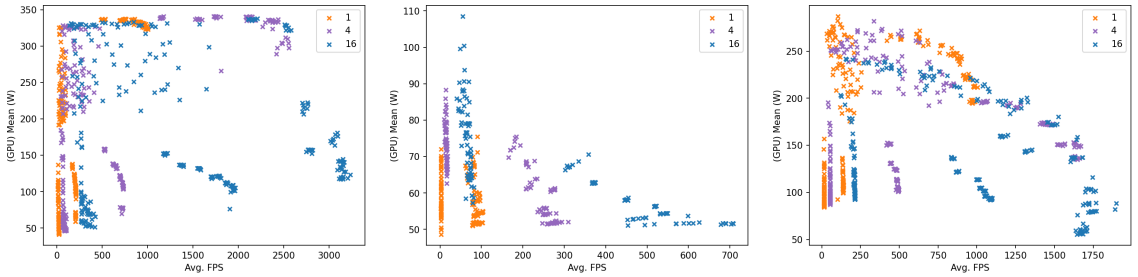
(a) AMD 7900 XT **(b)** Intel A770 **(c)** Nvidia 4080

Figure 5.15: Scatterplot of Mean GPU Power (W) and Mean FPS. Each point is one spheres test case with different parameters. Different **render methods** are highlighted.



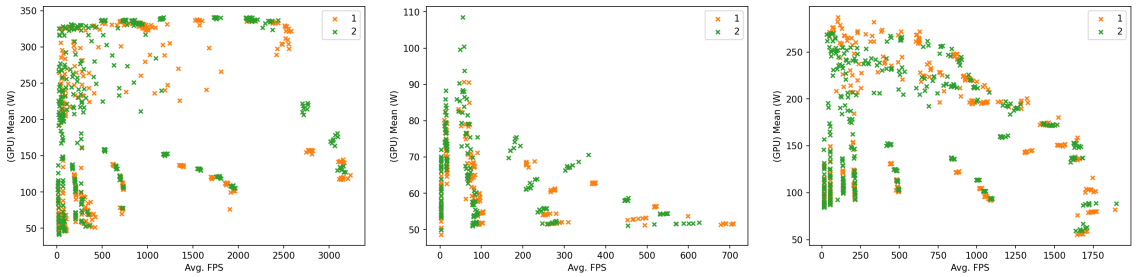
(a) AMD 7900 XT **(b)** Intel A770 **(c)** Nvidia 4080

Figure 5.16: Scatterplot of Mean GPU Power (W) and Mean FPS. Each point is one spheres test case with different parameters. Different **datasets** are highlighted.



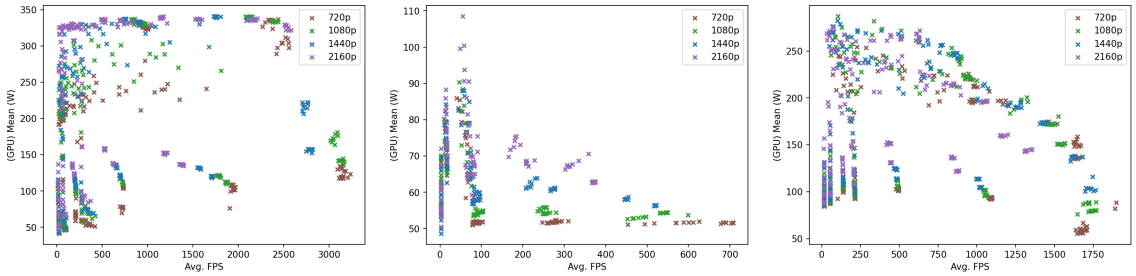
(a) AMD 7900 XT **(b)** Intel A770 **(c)** Nvidia 4080

Figure 5.17: Scatterplot of Mean GPU Power (W) and Mean FPS. Each point is one spheres test case with different parameters. Different **thinning factors** are highlighted.



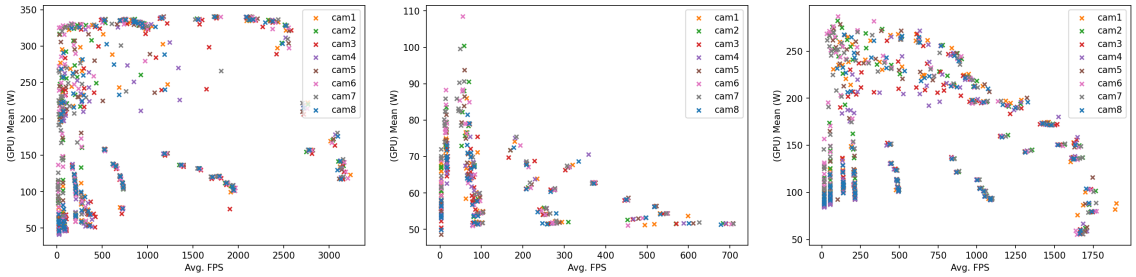
(a) AMD 7900 XT **(b)** Intel A770 **(c)** Nvidia 4080

Figure 5.18: Scatterplot of Mean GPU Power (W) and Mean FPS. Each point is one spheres test case with different parameters. Different **sphere radii** are highlighted.



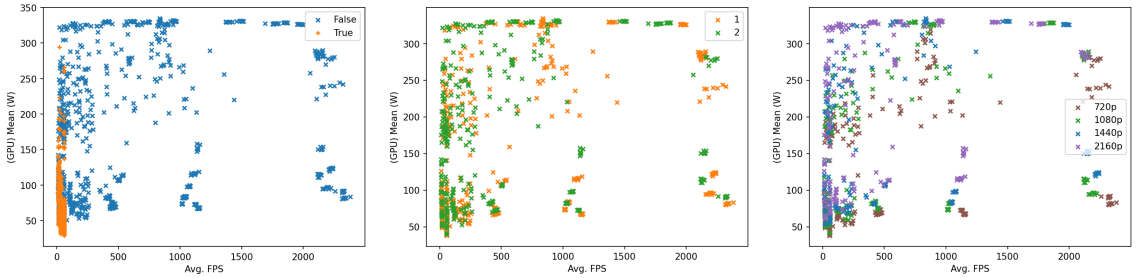
(a) AMD 7900 XT **(b)** Intel A770 **(c)** Nvidia 4080

Figure 5.19: Scatterplot of Mean GPU Power (W) and Mean FPS. Each point is one spheres test case with different parameters. Different **resolutions** are highlighted.



(a) AMD 7900 XT **(b)** Intel A770 **(c)** Nvidia 4080

Figure 5.20: Scatterplot of Mean GPU Power (W) and Mean FPS. Each point is one spheres test case with different parameters. Different **camera angles** are highlighted.

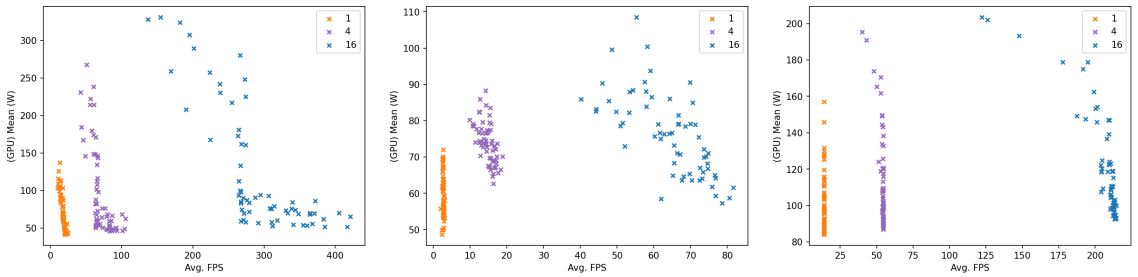


(a) V-Sync **(b)** Sphere Radius **(c)** Resolution

Figure 5.21: Scatterplot of Mean GPU Power (W) and Mean FPS. Each point is one spheres test case **running on an AMD 6900 XT** GPU with different parameters. Different factors are highlighted.

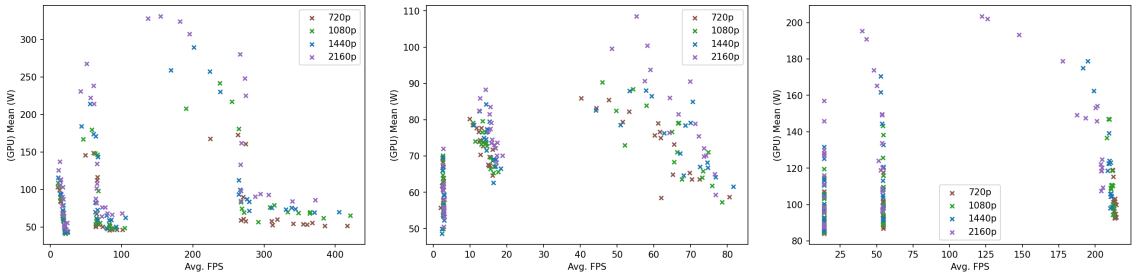
focus on the AMD 7900 XT, the Intel A770 and the Nvidia 4080. This selection is mostly arbitrary, but these specific GPUs are closest regarding TDP, recency and general performance.
Fig. 5.14 shows the general ”shapes” of the GPUs’ plots and highlights the test cases with enabled V-Sync, which appear as vertical lines at integer divisions of 60. Being confined to only very few positions on the lower end of the FPS axis, these test cases do not allow many clear spatial relations and mostly get in the way, so they are omitted from the following graphs.
Looking at the cases with disabled V-Sync, the AMD and Nvidia GPUs both show a more spread-out upper section and a lower section consisting of diagonal descending lines of clusters (note: this direction implies lower power but more FPS). The upper section for the AMD GPUs is primarily horizontal, with concentration points at the top (except for the area with the highest FPS), while it is approximately diagonally descending for the Nvidia GPUs and completely missing for the Intel A770. Fig 5.15 explains that, as it shows that, the upper section, except for minor overlap, mainly consists of test cases for SSBO stream with static data, which the Intel A770 has no test cases for. On top of that, the simple renderer test cases for the Intel A770 are more spread out, which could be a side effect of the lower FPS on this GPU (more evidence for this later when analyzing the leftmost conglomeration of values). The shape of the data points for the SSBO stream with static data on the AMD GPUs matches the previous observations of the AMD GPUs being fully utilized in most cases and running into bottlenecks in cases with high FPS.
Fig. 5.16 highlights the generally more clustered data points to the right belonging to the ace_drop_1m dataset and the more spread-out data points to the left belonging to the expl30m dataset. While the lines earlier established as belonging to the simple render remain concise for the expl30m dataset on the Nvidia GPUs and the Intel A770, they are not as clearly separable on the AMD GPUs. Also, one of the ”simple renderer lines” belonging to the ace_drop_1m consistently appears to the left of the rightmost line belonging to the expl30m dataset. This behaviour applies to all GPUs (except the Intel A770, where it just overlaps instead of being separate on the left) and implies that there is a factor that causes the larger dataset to produce higher FPS than the smaller consistently. Fig. 5.17 makes it clear that this ”factor” is a thinning factor of 16, which seems odd, as the expl30m dataset should, even this thinned out, still contain more data points. In addition to this observation, all ”simple renderer lines” can now be classified as the influence of different thinning factors (even observable for the expl30m dataset on the AMD 7900 XT). On top of all these influences on the simple renderer test cases, distinguishable areas of equal thinning factors have appeared even in the more spread-out area of the SSBO with static data test cases. Again, the

areas of equal thinning factor are more concise within the values for the ace_drop_1m dataset. It is also clearly visible now that the high FPS areas with non-GPU bottlenecks primarily consist of ace_drop_1m data at a high thinning factor (on both AMD and Nvidia GPUs).
Now that the factors causing various areas are more transparent, Fig. 5.18 provides more details on the composition of the individual small clusters within these areas. Focusing once again on the ”simple renderer lines”, we can see that every cluster of test cases with sphere radius 1 has a corresponding cluster of sphere radius 2 to its upper left (implying more power while producing less FPS). While not as clearly distinguishable for the SSBO with static data test cases, at least within the area with high FPS, these pairs of clusters are also observable, and it can be assumed that the sphere radius has a similar influence on test cases with lower FPS outcomes.
Fig. 5.19 explains the relationship of the cluster pairs within the ”simple renderer lines”, as it depicts each line consisting of one cluster pair for every resolution. At least on the AMD GPUs, for the SSBO stream with static data, a pattern of higher resolutions being more common in the upper left, where FPS are lower and power consumption is higher, is visible. At this point, it should be mentioned that the general trend of simple renderer test cases requiring less power for lower resolutions/sphere radii while producing more FPS does not apply to the AMD 6900 XT. Fig. 5.21 shows that The ”simple renderer lines” are not diagonal here, but rather approximately vertical. This shape implies that lower resolutions/sphere radii require less power but generally do not directly increase FPS on a per-case basis.
The only missing detail for the simple renderer test cases is how the clusters are composed. Incidentally, the only test parameter not discussed yet should play a role here, and Fig. 5.20 confirms it. Although the distribution of the camera angles across the whole plot seems chaotic, it is relatively straightforward that each cluster contains all camera angles. It should be noted that because of the proximity of the test cases that are only differentiated by the camera angle, a significant amount of overdraw causes the camera angles with higher indices to hide the angles with lower indices. Other than this observation, the different camera angles seem to be evenly distributed across the plots' more spread-out areas, but other than that, no clear pattern is observable.
Because of the compressed nature of the data points in the lower FPS range, it is harder to make out if ”simple renderer lines” also exist for the expl30m dataset, so eliminating the test cases for the ace_drop_1m and all SSBO cases gives more insight:
Fig. 5.22 shows line-like structures with a line representing each thinning factor for all GPUs, although they are more spread out on the AMD GPUs and the Intel A770. Compared to the lines corresponding to the ace_drop_1m test cases, these lines are generally more straight downward (meaning there are more test cases with different power draws that produce similar FPS) and no defined clusters within the lines (only hints of clusters at higher thinning factors). The absence of clusters also makes the distribution of different resolutions/sphere radii less organized, and relations between different test cases are unclear. However, there is still a tendency for higher resolutions/sphere radii to appear more to the upper left (higher power draw, less FPS) (see Fig. 5.23 and Fig. 5.24). As the expl30m test cases for SSBO stream with static data also show the same tendency of being less organized than their ace_drop_1m counterparts, it can be assumed that larger datasets (especially on higher thinning factors) have higher variance in power draw and FPS across similar cases (especially different camera angles).



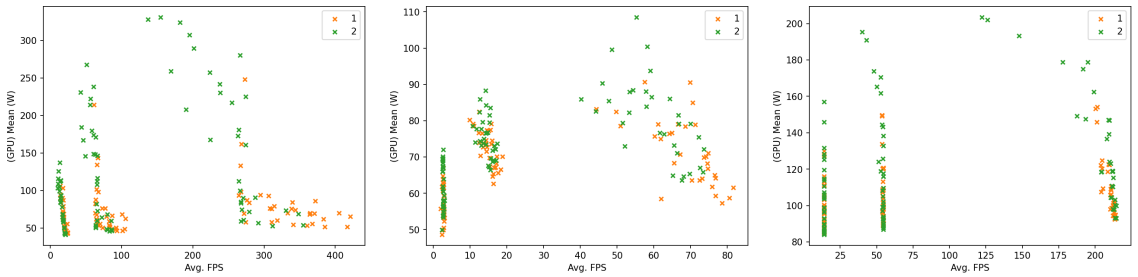
(a) AMD 7900 XT **(b)** Intel A770 **(c)** Nvidia 4080

Figure 5.22: Scatterplot of Mean GPU Power (W) and Mean FPS (only the **expl30m dataset with simple renderer**). Each point is one spheres test case with different parameters. Different **thinning factors** are highlighted.



(a) AMD 7900 XT **(b)** Intel A770 **(c)** Nvidia 4080

Figure 5.23: Scatterplot of Mean GPU Power (W) and Mean FPS (only the **expl30m dataset with simple renderer**). Each point is one spheres test case with different parameters. Different **resolutions** are highlighted.



(a) AMD 7900 XT **(b)** Intel A770 **(c)** Nvidia 4080

Figure 5.24: Scatterplot of Mean GPU Power (W) and Mean FPS (only the **expl30m dataset with simple renderer**). Each point is one spheres test case with different parameters. Different **spheres** are highlighted.

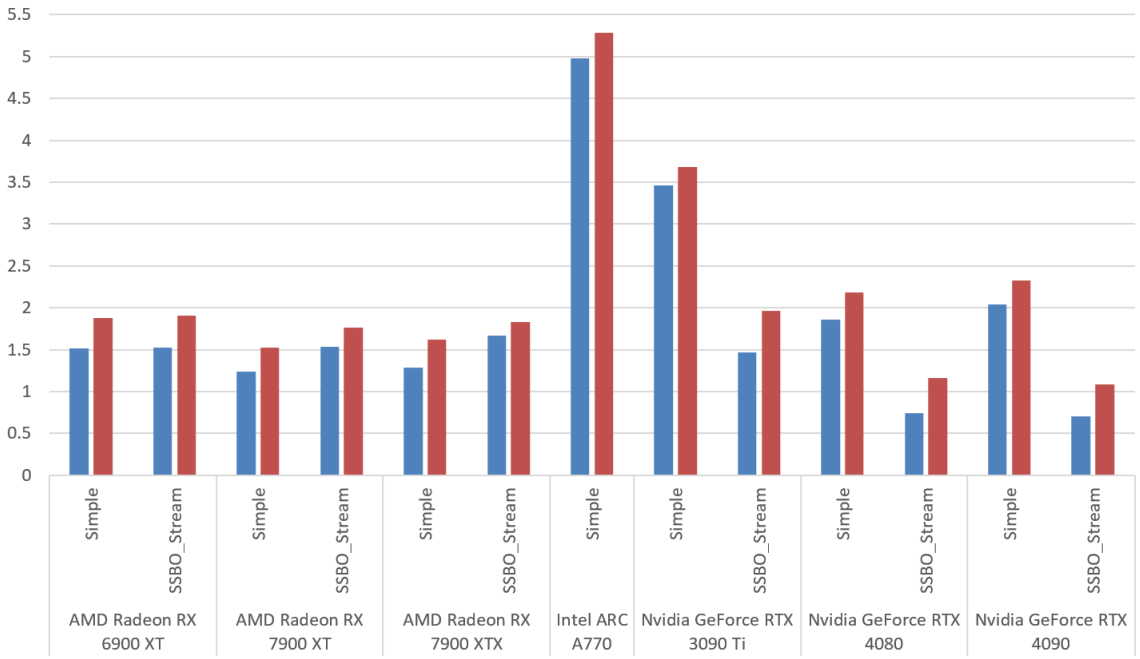


Figure 5.25: GPU efficiency (measured in J per Frame) by **render method**. **left/blue** is with **V-Sync disabled** and **right/red** with **V-Sync enabled**. Each column comprises the mean of the mean efficiency of all spheres test cases that match these parameters.

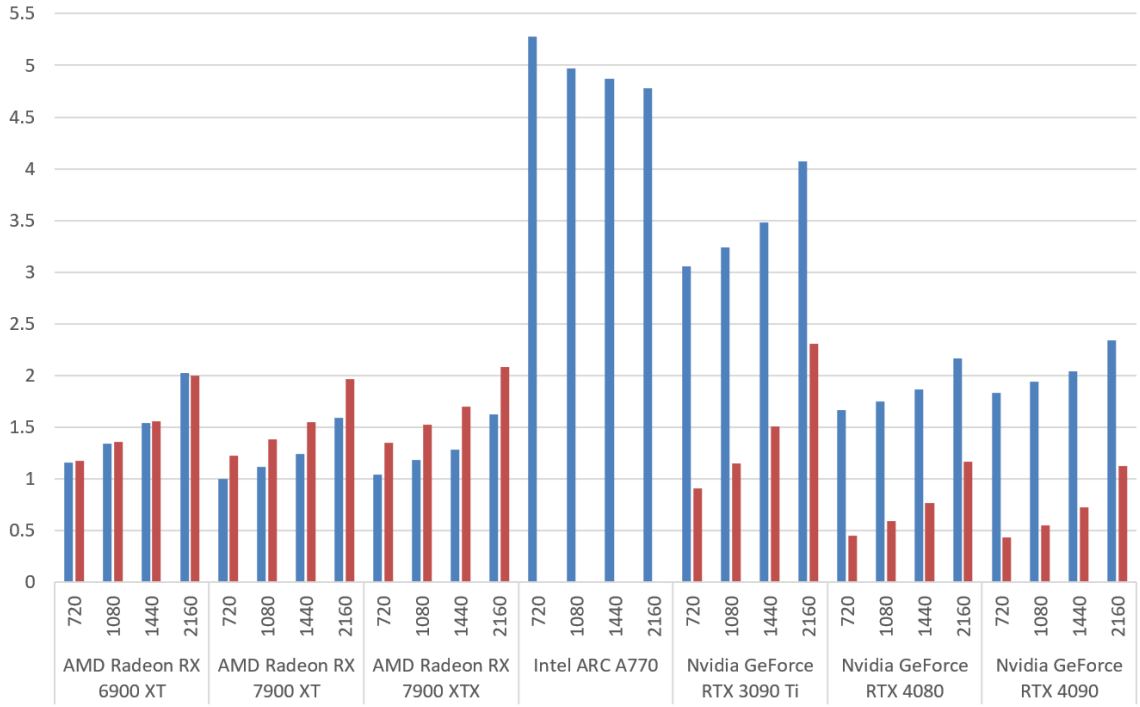


Figure 5.26: GPU efficiency (measured in J per Frame) by **resolution**. **left/blue** is with **V-Sync disabled** and **right/red** with **V-Sync enabled**. Each column comprises the mean of the mean efficiency of all spheres test cases that match these parameters.

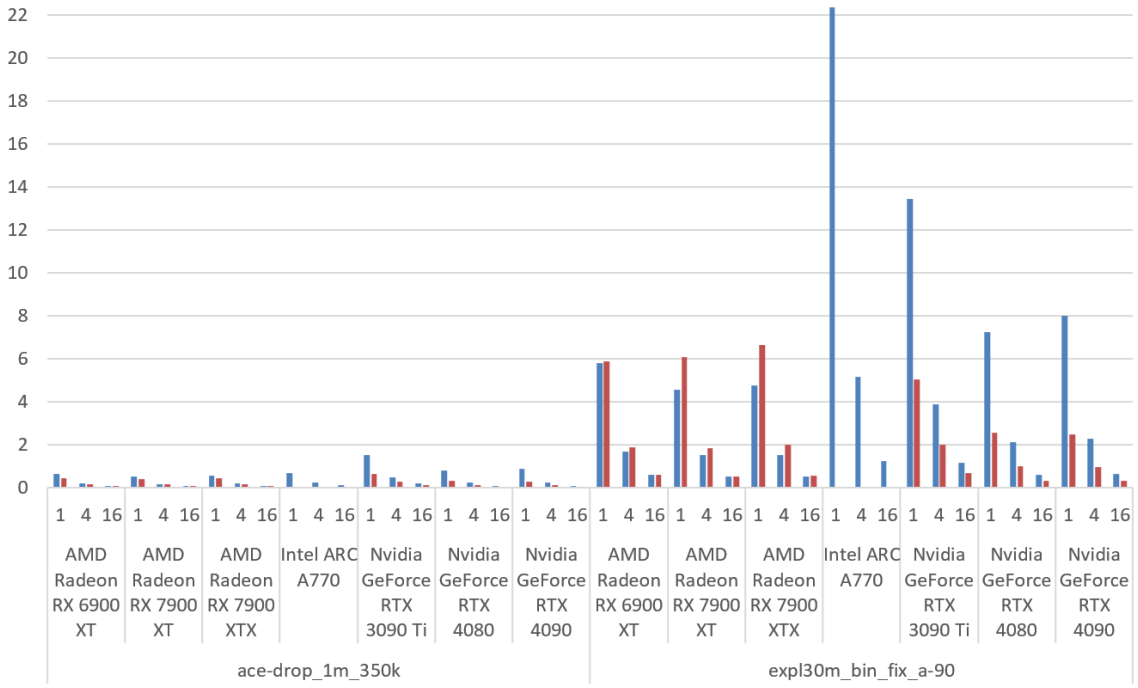


Figure 5.27: GPU efficiency (measured in J per Frame) by **dataset and thinning factor**. **left/blue** is with **V-Sync disabled** and **right/red** with **V-Sync enabled**. Each column comprises the mean of the mean efficiency of all spheres test cases that match these parameters.

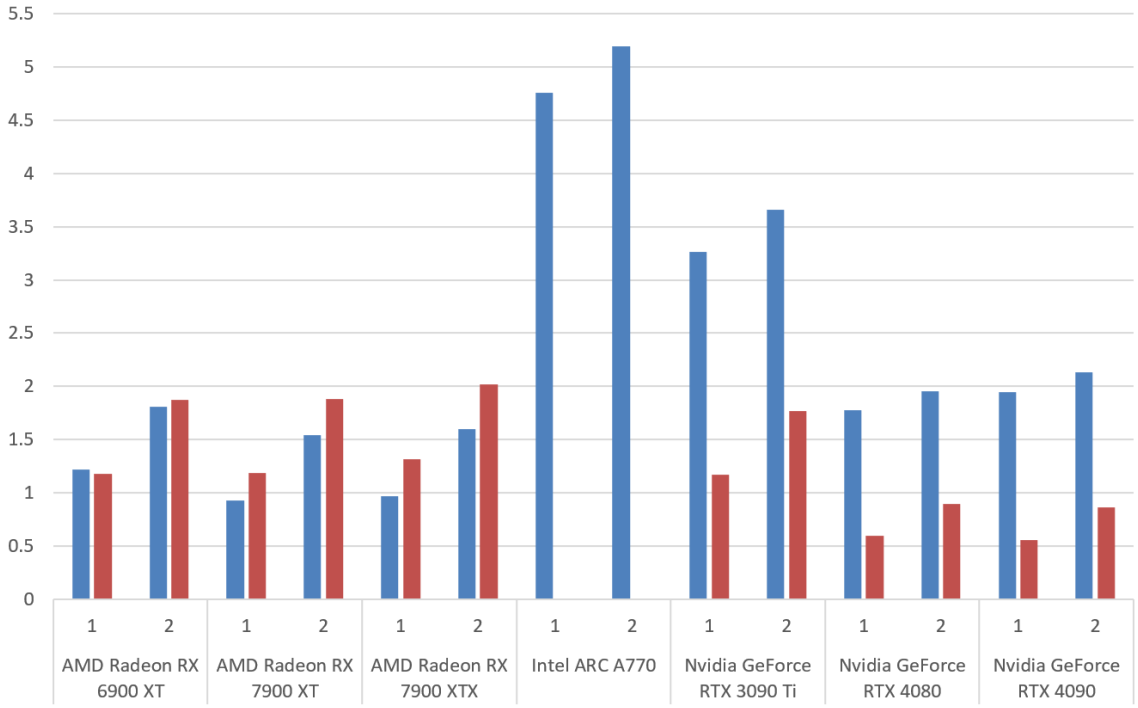
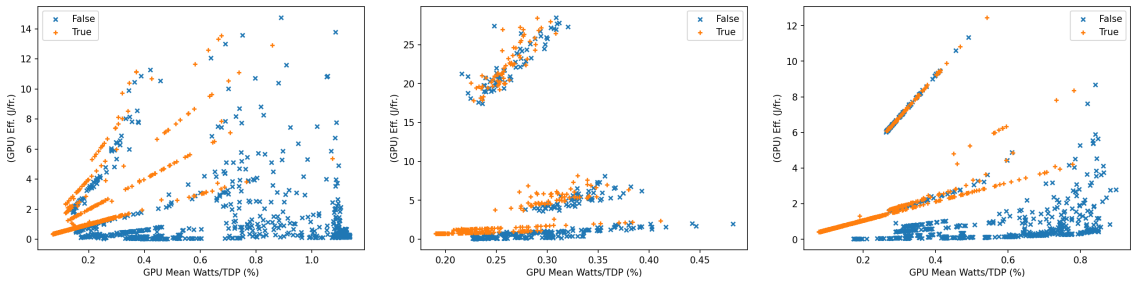


Figure 5.28: GPU efficiency (measured in J per Frame) by **sphere radius**. **left/blue** is with **V-Sync disabled** and **right/red** with **V-Sync enabled**. Each column comprises the mean of the mean efficiency of all spheres test cases that match these parameters.

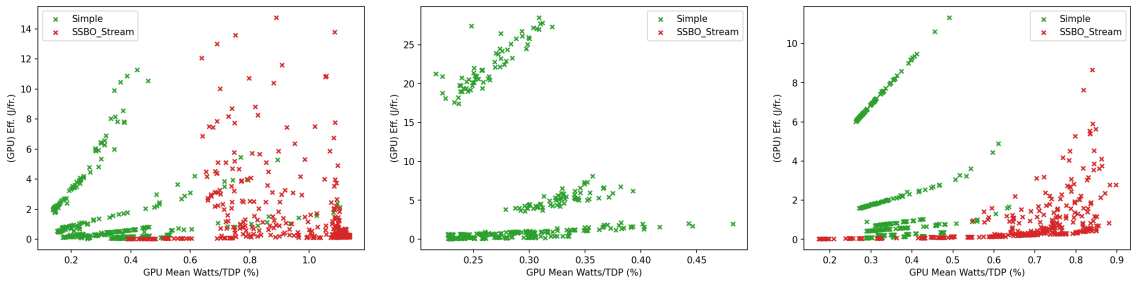


(a) AMD 7900 XT **(b)** Intel A770 **(c)** Nvidia 4080

Figure 5.29: Scatterplot of GPU Efficiency (J per frame) and GPU Mean Power (normalized to their respective TDP). Each point is one spheres test case with different parameters. Different **states of V-Sync** are highlighted.

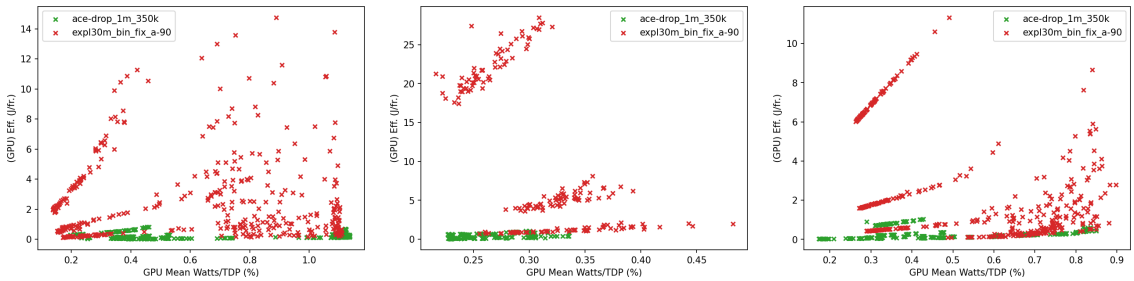
5.1.3 Efficiency

While the scatterplots of Mean GPU Power (W) and mean FPS offer some insight into performance per watt, they lend themselves poorly to broad, generalized statements. Fig 5.25 shows that, regardless of render mode, enabling V-Sync causes an average overhead of less than 0.5 J. When factoring in the (although relatively constant) power consumption of the rest of the system, this is raised to ̃1-2 J, although Fig. 5.51 shows that enabled V-Sync decreases power draw by 5-10 W on average. Fig 5.25 also shows that the simple renderer on Nvidia GPUs is around twice as expensive as the SSBO stream with static data. On the AMD 6900 XT, there is no distinguishable difference between the render modes, while on the AMD 7000-series GPUs, the simple renderer is less than 0.5 J more efficient. This behaviour leads to the following order of efficiency from least to most efficient: Intel > Nvidia 3090 Ti Simple > Nvidia 4000-series Simple > Nvidia 3090 Ti SSBO, AMD 6900 and AMD 7000-series SSBO > AMD 7000-series Simple > Nvidia 4000-series SSBO. Although it has the lowest average power draw, the Intel A770 is the least efficient because of the generally low FPS it produces. Despite its 450 W TDP, the Nvidia 4090 is the most efficient because it produces reasonably high FPS while not always drawing as much power as it technically could. The observations hold when looking at Fig. 5.26, but can be expanded by frames at higher resolutions generally being more expensive (except for the Intel A770. Additionally, the gap between the render modes on Nvidia GPUs decreases at higher resolutions. This decrease implies that the FPS generally increase by a smaller factor than the power draw. What seems odd is that the efficiency of the Intel A770 increases at higher resolutions. The best explanation may be the wider FPS spread across the test cases for the exact resolution (see Fig. 5.19) combined with the generally lower FPS. On the other hand, what holds for every tested GPU is a substantial increase in efficiency for smaller/thinner datasets, as shown in Fig. 5.27. While dataset size alone does not seem to scale linearly with performance per watt (potentially a result of the bottlenecks encountered at the frequent high-FPS test cases of the smaller ace_drop_1m dataset), thinning out a dataset in some cases, yields an increase in efficiency that almost matches the thinning factor.
The only highly influential test parameter left to discuss is the sphere radius, which, as shown by Fig. 5.28, increases efficiency when lower. However, not by much, as it results in 1 J less per frame on average for AMD GPUs and less than 0.5 J for the other manufacturers.



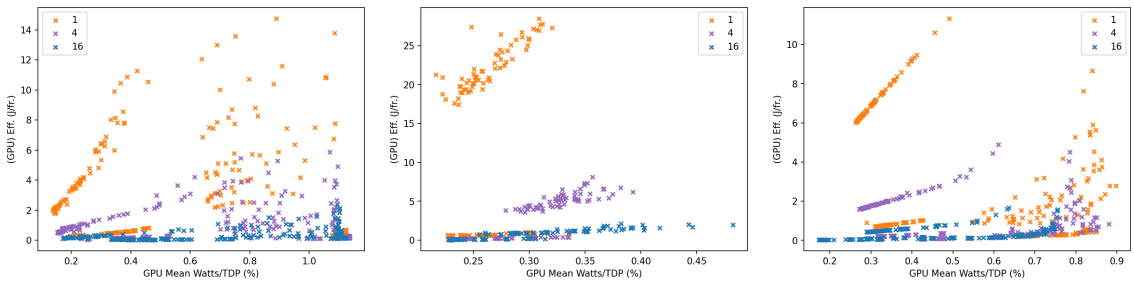
(a) AMD 7900 XT **(b)** Intel A770 **(c)** Nvidia 4080

Figure 5.30: Scatterplot of GPU Efficiency (J per frame) and GPU Mean Power (normalized to their respective TDP). Each point is one spheres test case with different parameters. Different **render methods** are highlighted.



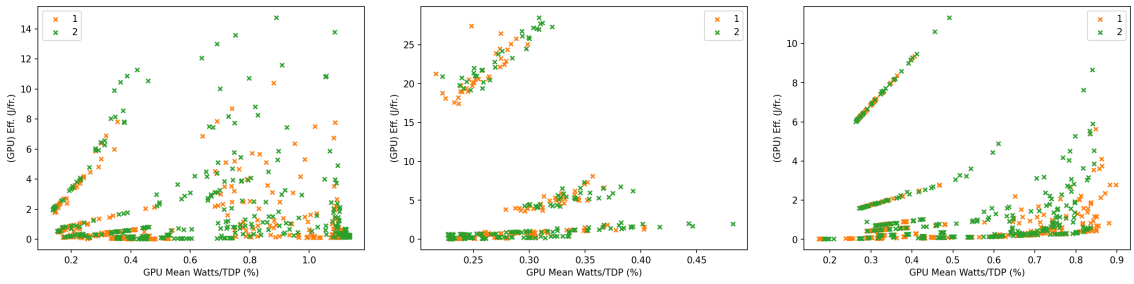
(a) AMD 7900 XT **(b)** Intel A770 **(c)** Nvidia 4080

Figure 5.31: Scatterplot of GPU Efficiency (J per frame) and GPU Mean Power (normalized to their respective TDP). Each point is one spheres test case with different parameters. Different **datasets** are highlighted.



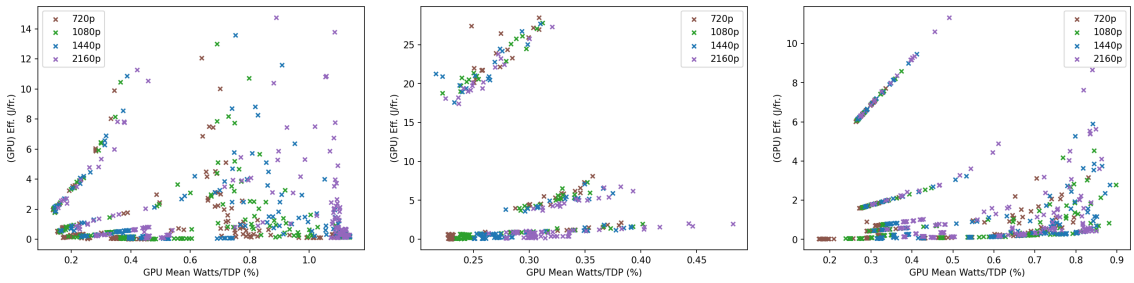
(a) AMD 7900 XT **(b)** Intel A770 **(c)** Nvidia 4080

Figure 5.32: Scatterplot of GPU Efficiency (J per frame) and GPU Mean Power (normalized to their respective TDP). Each point is one spheres test case with different parameters. Different **thinning factors** are highlighted.



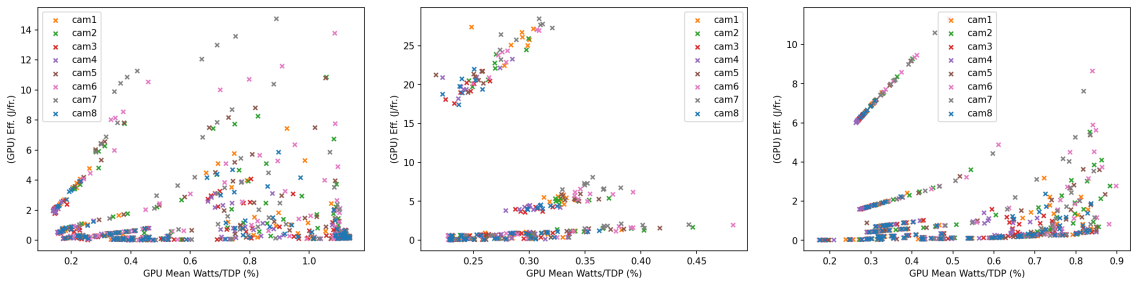
(a) AMD 7900 XT **(b)** Intel A770 **(c)** Nvidia 4080

Figure 5.33: Scatterplot of GPU Efficiency (J per frame) and GPU Mean Power (normalized to their respective TDP). Each point is one spheres test case with different parameters. Different **sphere radii** are highlighted.



(a) AMD 7900 XT **(b)** Intel A770 **(c)** Nvidia 4080

Figure 5.34: Scatterplot of GPU Efficiency (J per frame) and GPU Mean Power (normalized to their respective TDP). Each point is one spheres test case with different parameters. Different **resolutions** are highlighted.



(a) AMD 7900 XT **(b)** Intel A770 **(c)** Nvidia 4080

Figure 5.35: Scatterplot of GPU Efficiency (J per frame) and GPU Mean Power (normalized to their respective TDP). Each point is one spheres test case with different parameters. Different **camera angles** are highlighted.

Again, to get more intel on the spatial relations between the test cases, we plot the efficiency values against the power consumption (this time normalized by each GPU's respective TDP, the metric we use to approximate GPU utilization).
First, Fig. 5.29 shows many diagonal lines, implying less efficiency at higher utilization. The lines are generally less concise on the Intel A770. The most concise lines stem from the test cases with enabled V-Sync; each line represents a fraction of 60 (the more potent the slope, the higher the divisor). But even the test cases with V-Sync disabled show similar diagonal lines but with more vertical spread. Fig. 5.30 shows that these lines result from the simple renderer test cases and appear mostly below 50% approximate utilization (matching the earlier observations), while the more spread out section at a higher utilization level is from the SSBO stream with static data test cases. Fig. 5.31 reveals that most of the data visible is from the expl30m dataset because the significantly higher FPS of most ace_drop_1m test cases causes the resulting efficiency generally better than 1 J per frame. For all GPUs, the spread in the expl30m simple renderer lines matches the spread previously visible in Fig. 5.22m, so there is less (in this case vertical) spread in these lines. The lines for simple renderer with ace_drop_1m are also visible enough to confirm their existence (see B for more details). Despite the AMD GPUs' tendency to be at high utilization when FPS are unlocked when using SSBO stream with static data, not only the ace_drop_1m cases, but even most of the expl30m cases are below 2 J per frame. However, the same is true for Nvidia GPUs, which do not show as wide of a spread on "low-efficiency outlier cases" as AMD GPUs do (even at lower utilization levels). Fig. 5.32 improves the visibility of the simple renderer lines of the ace_drop_1m cases, but, more importantly, reveals that most of the outliers on the AMD GPUs were from the unthinned dataset, as its test cases make io the majority of the upper left area of SSBO test cases. The lower thinning factors on the AMD GPUs are primarily located within a cluster at full utilization with <2 J per frame. As for Nvidia GPUs, test cases with higher thinning factors get rarer at higher utilization levels.
As for the influences of the sphere radius, Fig. 5.33 shows that, for the simple renderer test cases, AMD GPUs are less efficient with the larger sphere radius at higher thinning factors. At the same time, there is no noticeable difference for the other GPUs. What is noticeable about the test cases for the SSBO stream with static data is that Nvidia GPUs are generally less efficient at the larger sphere radius, and the highest outliers on the AMD GPUs are also test cases with the larger sphere radius. Fig. 5.34 shows how the ace_drop_1m cases are somewhat sorted by resolution, as higher resolutions come with higher utilization. On the other hand, the SSBO stream with static data cases with significantly lower efficiency are mostly higher resolutions on Nvidia GPUs. At the same time, there are almost vertical lines of matching resolutions on the AMD GPUs. When looking at the topmost outliers, there are also clearly visible horizontal (or somewhat slightly diagonal) lines that show how, in these cases, higher resolution causes higher utilization and worse efficiency. To get an explanation for these outlier lines, this time, the camera angles, which still are primarily chaotic, reveal an important detail: Fig. 5.35 shows that every line represents a camera angle, and the standing out ones are 7, 6, 5 and 2. Orientation-wise, no apparent factor sets them apart from the other angles. However, the orientation of the expl30m dataset causes these angles to feature the data fully lit, while the other angles show the dataset's "dark side". When directly lit, the increased complexity of the shadows seems to significantly affect the AMD GPUs' performance, causing the efficiency to plummet.

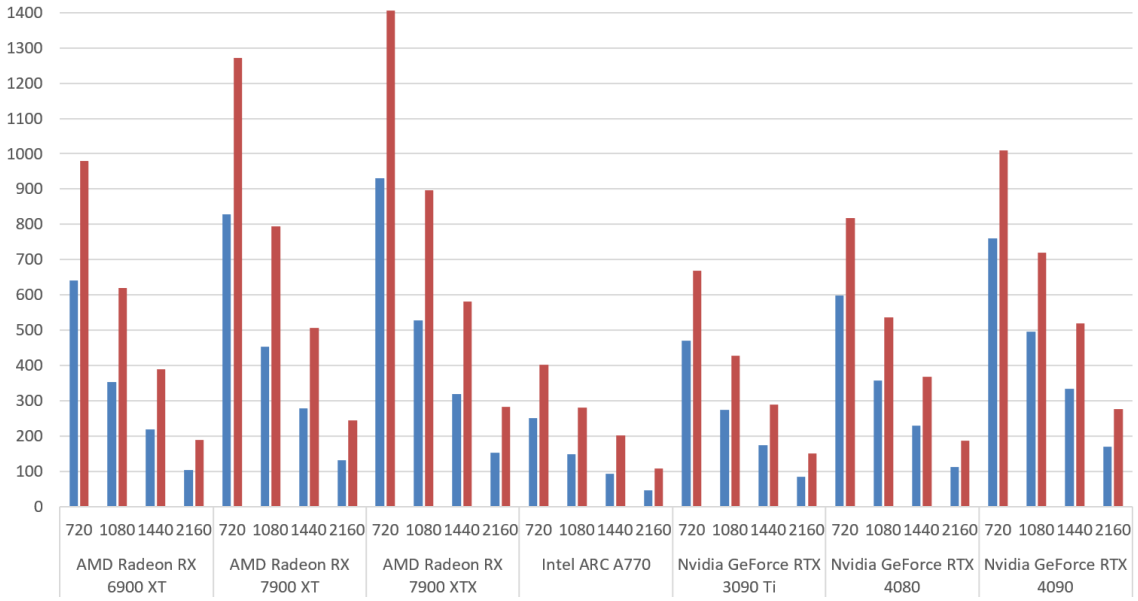


Figure 5.36: Mean FPS by GPU and **resolution**. **left/blue** is **Integration** renderer, **right/red** is **Isosurface**. V-Sync is **disabled**. Each column is composed of the mean of the FPS of all volume test cases that match these parameters.

5.2 Volume

5.2.1 Performance

As shown in Fig. 5.36, the GPUs perform in the following order: AMD 7000-series > Nvidia 4090 + AMD 6900 > Nvidia 4080 > Nvidia 3090 Ti > Intel A770. Isosurface always performs significantly better (likely because it is a process with fewer steps), and the increase over Integration looks like a fixed percentage at first glance but increases significantly at higher resolutions for all GPUs. The difference between render modes is the highest on the Intel A770 (60-130%) and the lowest on the Nvidia GPUs (35-65%). Doubling the resolution comes with a decrease in FPS of 30-50%. The difference in render modes and resolution is also visible in the test cases with enabled V-Sync, as seen in Fig. 5.37. It also shows that only the Nvidia 4090 could maintain a stable 60 FPS at 2160p in most cases.
Fig. 5.38 shows that the step ”bonsai to bunny” is generally less pronounced than ”bunny to chameleon” when using Integration. This observation also somewhat applies to Isosurface, but here, it is more apparent that each ”step” (8x increase in dataset dimensions, 2x per dimension) approximately halves the FPS.
Doubling the step ratio, as shown in Fig. 5.39, has a similar effect of increasing FPS by 50-100%. The scaling with higher resolution is generally more pronounced at higher step ratios for all GPUs. The absolute FPS overhead of the Isosurface render method stays relatively consistent across different step ratios on the same GPU. The relative FPS overhead is the most pronounced on the Intel A770 and the least on the Nvidia GPUs.

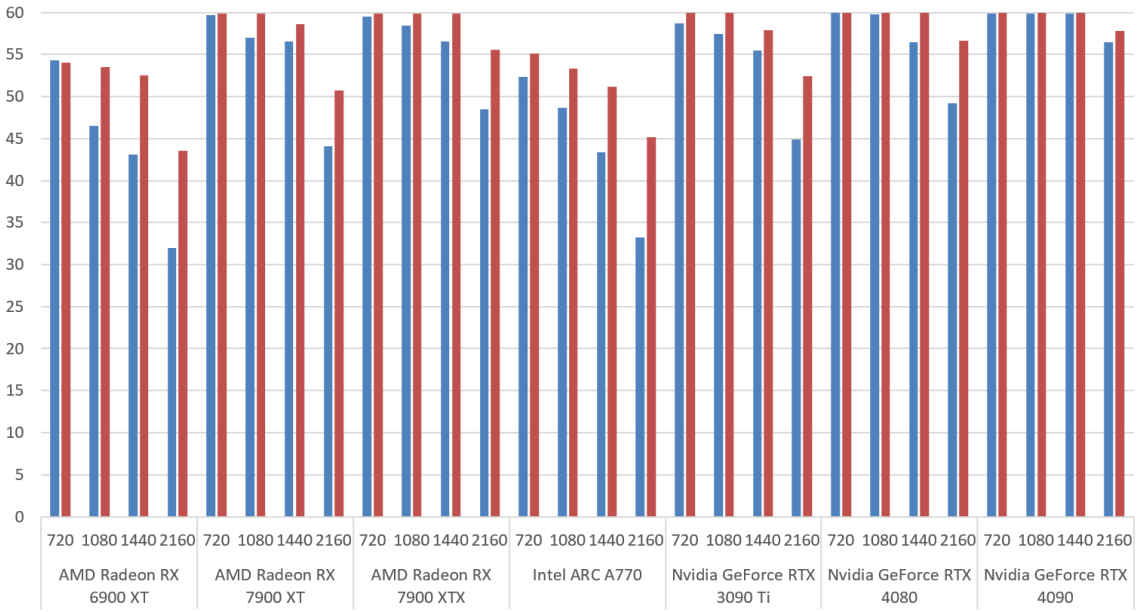


Figure 5.37: Mean FPS by GPU and **resolution**. **left/blue** is **Integration** renderer, **right/red** is **Isosurface**. V-Sync is **enabled**. Each column is composed of the mean of the FPS of all volume test cases that match these parameters.

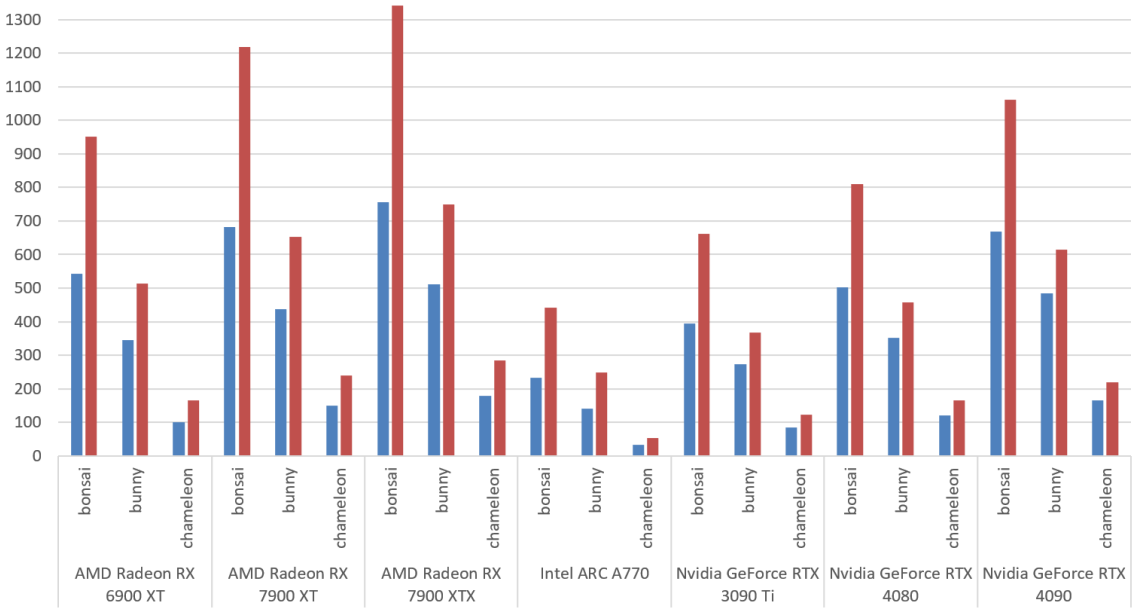


Figure 5.38: Mean FPS by GPU and **dataset**. **left/blue** is **Integration** renderer, **right/red** is **Isosurface**. V-Sync is **disabled**. Each column is composed of the mean of the FPS of all volume test cases that match these parameters.

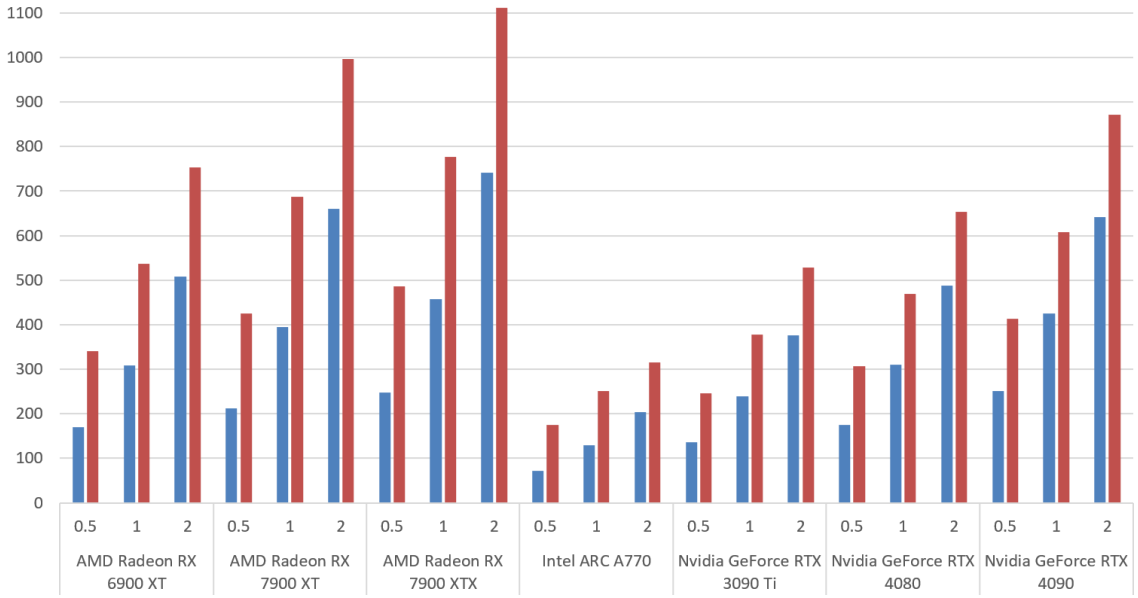


Figure 5.39: Mean FPS by GPU and **step ratio**. **left/blue** is **Integration** renderer, **right/red** is **Isosurface**. V-Sync is **disabled**. Each column is composed of the mean of the FPS of all volume test cases that match these parameters.

5.2.2 Power Consumption

Fig. 5.40 shows that the power consumption of all GPUs scales with the resolution at least to some degree. The order from highest to lowest power consumption goes as follows: Nvidia 3090 Ti > AMD 7900 XTX and Nvidia 4090 > AMD 6900 XT and 7900 XT > Nvidia 4080 > Intel A770. Isosurface not only generally produces higher FPS than Integration but also requires less power to do so (although the difference shrinks at higher resolutions). Overall, the scaling is insignificant on the AMD GPUs, as they, on average, only require less power for Isosurface on the lowest resolution. As Fig. 5.41 shows, all GPUs operate close to their TDP in these cases, so their absolute power consumption is in the earlier discussed order. The scaling on the AMD GPUs is overshadowed by them operating at full utilization. The approximate utilization above 100% is a byproduct of our TDP-based approximation, which depends on how a manufacturer measures their TDP.
Fig. 5.42 shows that when V-Sync is enabled, the scaling with resolution is much more pronounced, and all GPUs, except the Nvidia 3090 Ti, which is higher than the rest, have comparable absolute power consumption. Fig. 5.43 shows that their utilization varies, as now, from a relative viewpoint, the order of utilization is now as follows: Intel A770 > Nvidia 3090 Ti > the other GPUs > Nvidia 4090. The order is not as clear, as there is much overlap.
The results for different datasets and step ratios are mainly similar to the resolution results, as larger datasets and lower step ratios cause higher power consumption, and the scaling is more pronounced when V-Sync is enabled. Of these factors., the step ratio has the most minor effect on the power consumption.

Jumping to the spatial relations of the volume test cases, Fig. 5.44 shows horizontal lines at the expected FPS values (fractions of 60) for the V-Sync cases. The AMD GPUs show a horizontal line that kinks and scatters at higher FPS, while the Nvidia GPUs show a diagonally descending line

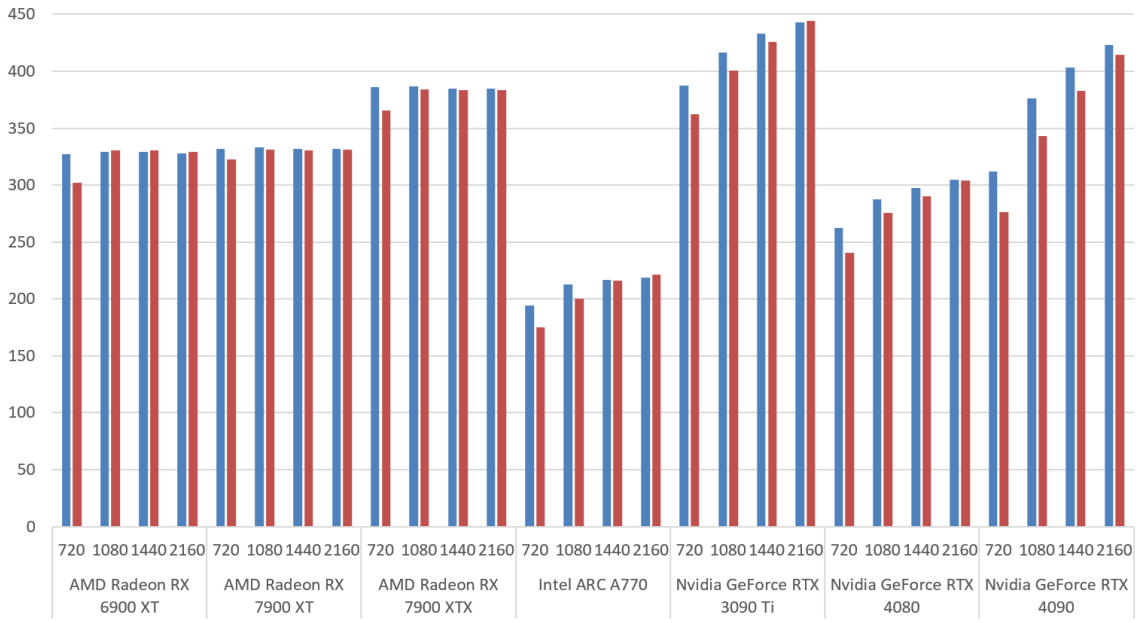


Figure 5.40: Mean GPU Power (W) by GPU and **resolution**. **left/blue** is **Integration** renderer, **right/red** is **Isosurface**. V-Sync is **disabled**. Each column is composed of the mean of the FPS of all volume test cases that match these parameters.

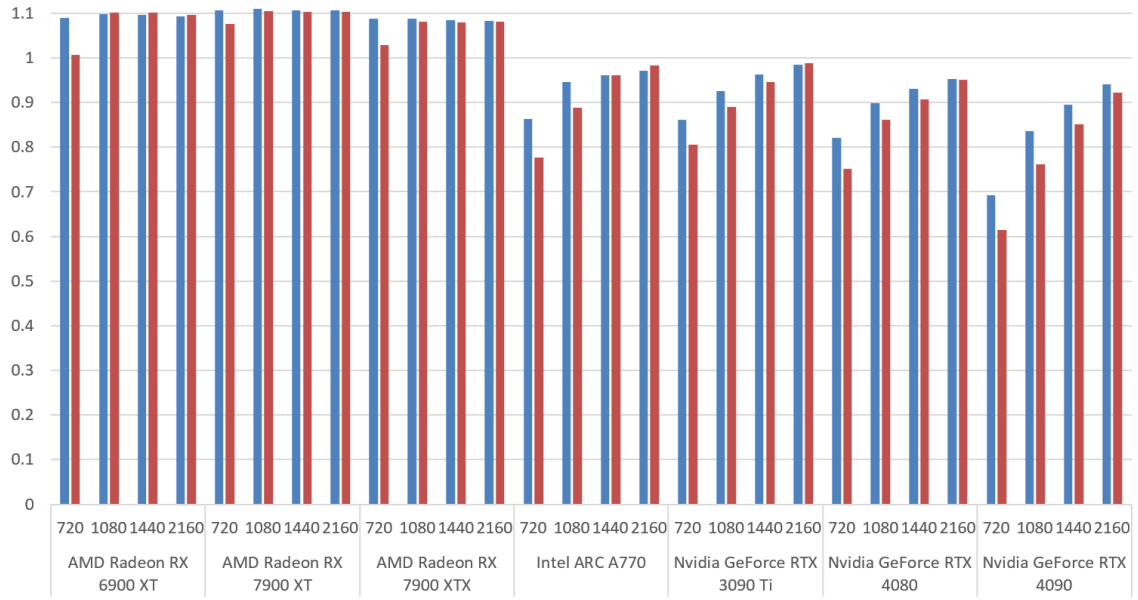


Figure 5.41: GPU Mean Power (normalized to their respective TDP) by GPU and resolution. **left/blue** is **Integration** renderer, **right/red** is **Isosurface**. V-Sync is **disabled**. Each column is composed of the mean of the FPS of all volume test cases that match these parameters.

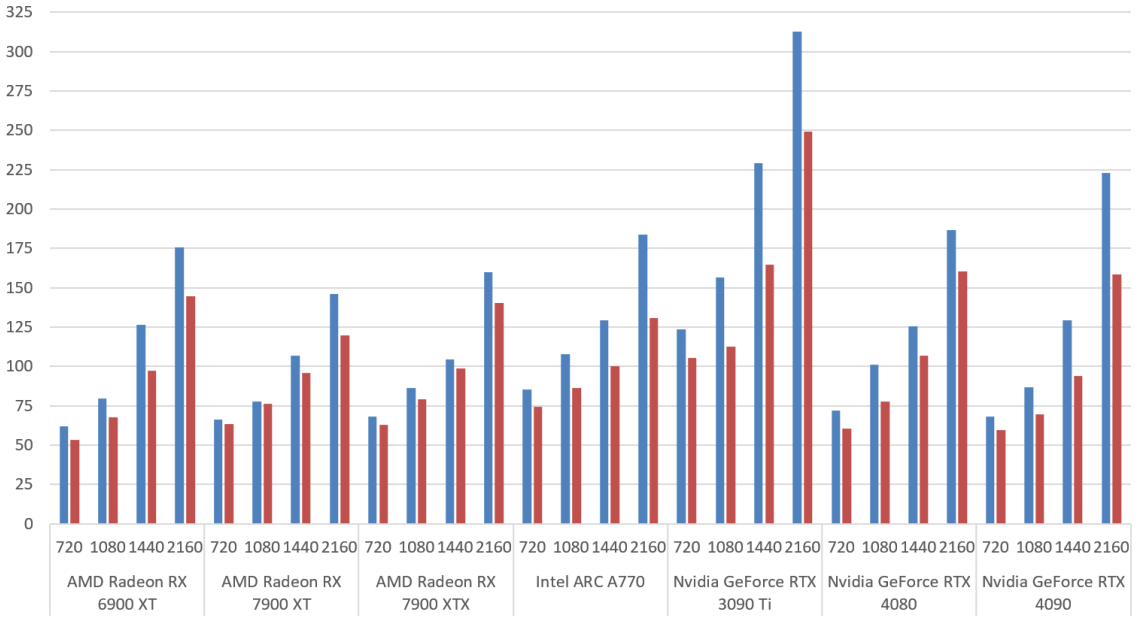


Figure 5.42: Mean GPU Power (W) by GPU and **resolution**. **left/blue** is **Integration** renderer, **right/red** is **Isosurface**. V-Sync is **enabled**. Each column is composed of the mean of the FPS of all volume test cases that match these parameters.

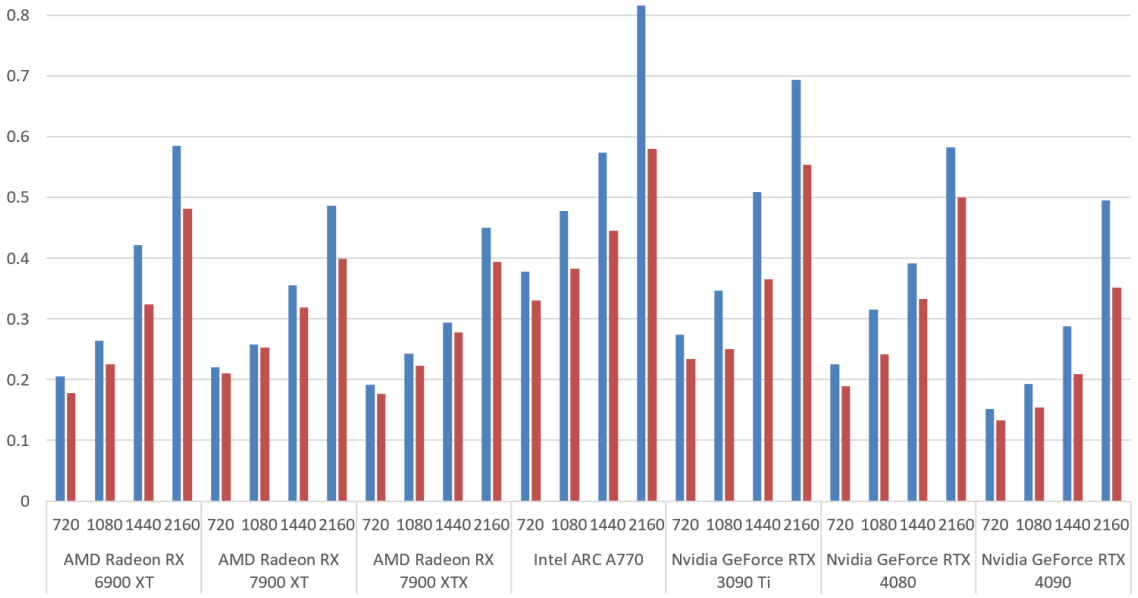
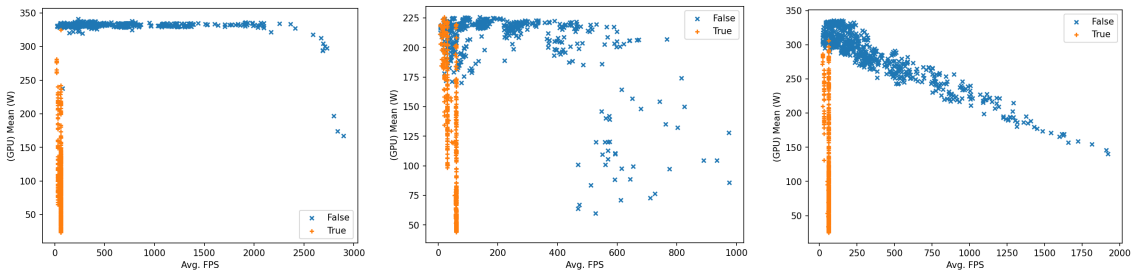
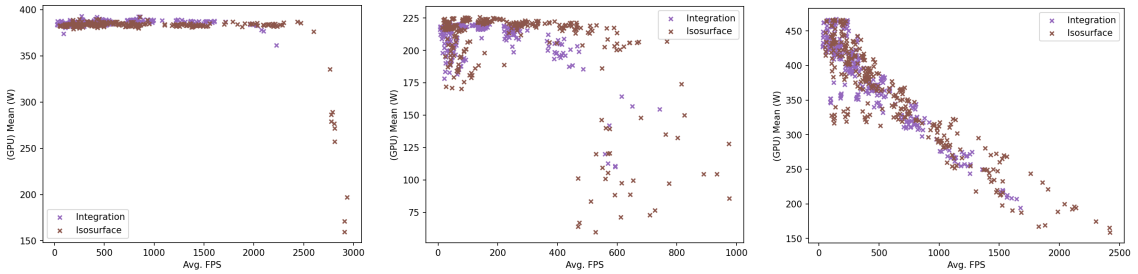


Figure 5.43: GPU Mean Power (normalized to their respective TDP) by GPU and resolution. **left/blue** is **Integration** renderer, **right/red** is **Isosurface**. V-Sync is **enabled**. Each column is composed of the mean of the FPS of all volume test cases that match these parameters.



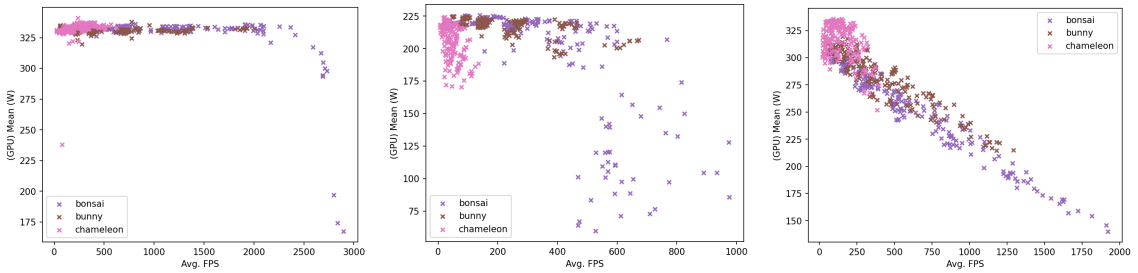
(a) AMD 7900 XT **(b)** Intel A770 **(c)** Nvidia 4080

Figure 5.44: Scatterplot of Mean GPU Power (W) and Mean FPS. Each point is one volume test case with different parameters. Different **states of V-Sync** are highlighted.



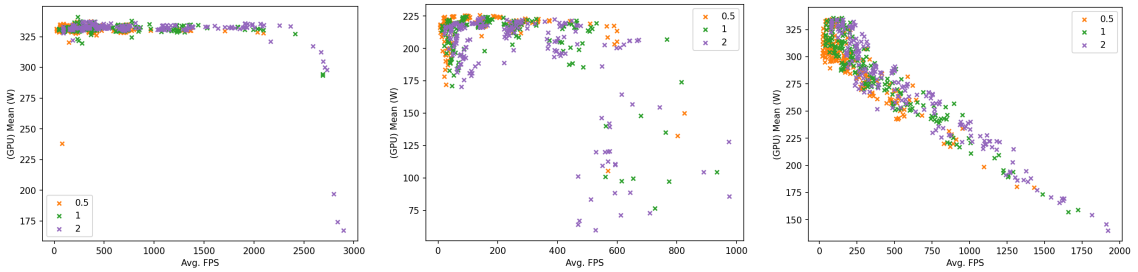
(a) AMD 7900 XTX **(b)** Intel A770 **(c)** Nvidia 4090

Figure 5.45: Scatterplot of Mean GPU Power (W) and Mean FPS. Each point is one volume test case with different parameters. Different **render methods** are highlighted.



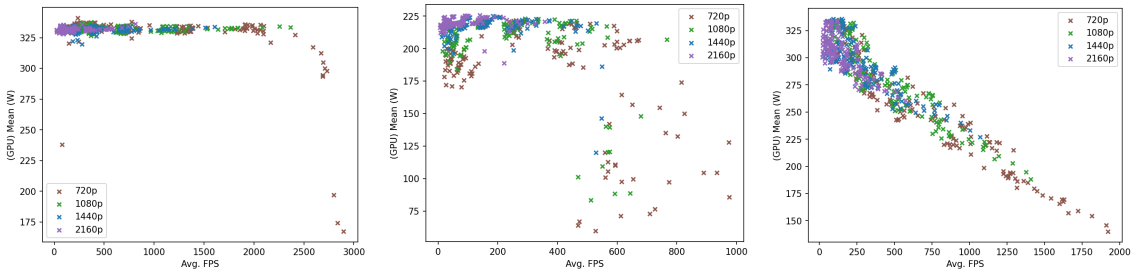
(a) AMD 7900 XT **(b)** Intel A770 **(c)** Nvidia 4080

Figure 5.46: Scatterplot of Mean GPU Power (W) and Mean FPS. Each point is one volume test case with different parameters. Different **datasets** are highlighted.



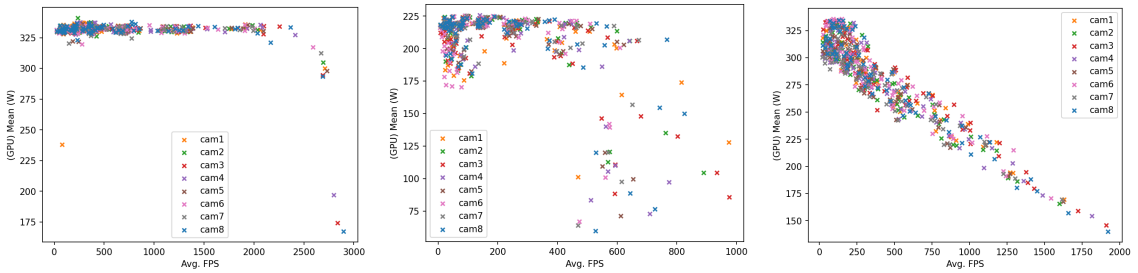
(a) AMD 7900 XT **(b)** Intel A770 **(c)** Nvidia 4080

Figure 5.47: Scatterplot of Mean GPU Power (W) and Mean FPS. Each point is one volume test case with different parameters. Different **step ratios** are highlighted.



(a) AMD 7900 XT **(b)** Intel A770 **(c)** Nvidia 4080

Figure 5.48: Scatterplot of Mean GPU Power (W) and Mean FPS. Each point is one volume test case with different parameters. Different **resolutions** are highlighted.



(a) AMD 7900 XT **(b)** Intel A770 **(c)** Nvidia 4080

Figure 5.49: Scatterplot of Mean GPU Power (W) and Mean FPS. Each point is one volume test case with different parameters. Different **camera angles** are highlighted.

that gets thinner at higher FPS. The Intel A770 is somewhere in between, with a slight diagonal that scatters a lot at higher FPS. The diagonal orientation implies lower power draw at higher FPS, and the scattering only at high FPS indicates a bottleneck that prevents even higher numbers.
In contrast to the sphere renderer, where the two render methods appeared in different plot areas, Fig. 5.45 shows Integration and Isosurface test cases intertwined. The high-FPS area for all GPUs is mainly populated by Isosurface test cases, making them the majority of cases where the bottlenecks occur. It looks like there is a slight bias on AMD 7000-series GPUs to Isosurface cases being less power-hungry, while they seem slightly more power-hungry on the Intel A770 and Nvidia GPUs. To confirm if there is a significant difference between the areas of Integration and Isosurface, a t-test was conducted, which led to mixed results: p for most GPUs was orders of magnitude below 0.05, but the AMD 7900 XTX was hitting right on the spot with $p = 0.496$, and the Nvidia 4090 was even higher with $p = 0.0856$. The results indicate evidence for the render methods creating different areas, but there is more evidence against it.
Fig. 5.46 shows that the datasets are less intertwined but still not separable. There is, as expected, a strong tendency for the smaller datasets to be more to the lower right. The bunny test cases seem to cluster at various medium-FPS areas (at least on the AMD GPUs and the Intel A770), while all chameleon test cases are located in the upper left corner.
Fig. 5.47 shows different behaviour for each manufacturer: There are clusters on the AMD GPUs, but they overlap strongly. However, the lowest step ratio cases are more present in the area of lower FPS and are entirely missing from the bottleneck area. For the Intel A770, the scattered area does not seem to follow a pattern, but the step ratios in the ”regular” area are staggered as rotated L-shapes, with the lowest factor being the top-left-most. The spread also increases at higher step ratios. As for the Nvidia GPUs, the lowest step ratio makes up the underside and the highest the top side of the diagonal. So, for Nvidia GPUs, higher step ratios generally increase FPS and lower power draw, while for the Intel A770, higher step ratios often decrease power draw. The increase in power draw on the Nvidia GPUs for individual test cases is in contrast to the average, which seems to be lowered significantly by the test cases in the high-FPS area.
Fig. 5.48 shows that the resolutions paint a more precise picture, with higher resolutions aligned to the top left and lower resolutions more to the bottom right. However, the lower the resolution, the more the test cases are spread along the FPS axis. Last, Fig. 5.49 reveals that the single low-FPS outlier all AMD GPUs share is the chameleon viewed from camera angle nr. 1 at the lowest step ratio and 720p. This being the first test case rendered after loading the chameleon dataset, this outlier is likely just a measuring artefact.

5.2.3 Efficiency

According to Fig. 5.50, the efficiency in order of worst to best order goes as follows: Intel A770 + Nvidia 3090 Ti > AMD 6900 XT > AMD 7000-series and Nvidia 4000-series. Despite having the widest TDP gap, the Intel A770 and Nvidia 3090 Ti are comparably (in)efficient. As expected from Integration consuming more power than Isosurface, it is, on average, always less efficient. Enabling V-Sync causes a similar absolute overhead between 0.25 and 0.5 J per frame on all GPUs. The overhead is the highest on the AMD 6900 XT (both render methods) and the AMD 7000-series GPUs (Isosurface only). The only exceptions are AMD 7000-series GPUs with Integration, where the overhead is <0.2J per frame. The influences of other test parameters are significant, but not surprising: Higher resolutions and larger datasets decrease while higher step ratios increase efficiency.

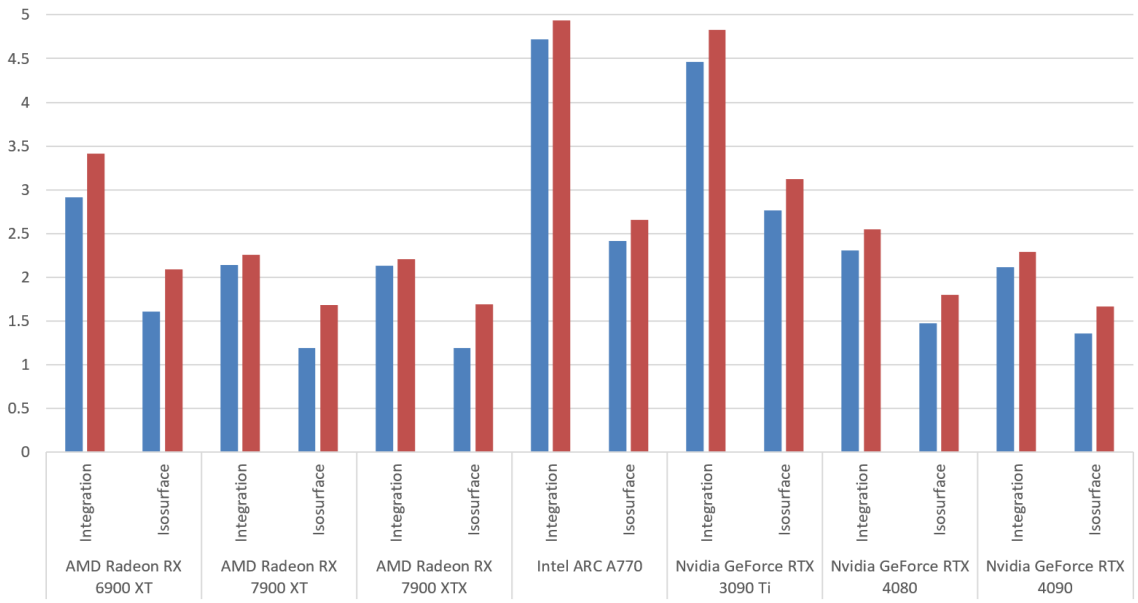


Figure 5.50: GPU efficiency (measured in J per Frame) by **render method**. **left/blue** is with **V-Sync disabled** and **right/red** with **V-Sync enabled**. Each column comprises the mean of the mean efficiency of all volume test cases that match these parameters.

5.3 General observations

5.3.1 Power Consumption of the Rest of the System

Up until now, we exclusively looked at the GPUs' power consumption. This perspective leaves the question of how much different render methods on different GPUs might affect the power consumption of the rest of the system (mainly CPU and memory). Fig. 5.51 shows a general trend across all GPUs and render modes (even across sphere and volume renderer test cases) of power consumption of 130 W without and 120 W with V-Sync. This relatively consistent power consumption (with a standard deviation of <10 W for all render methods) also suggests a relatively consistent CPU load. Minor exceptions to this are the Intel A770, which causes the rest of the system to draw equal (spheres simple) or even >5 W less (volume) power with disabled V-Sync (which could mostly be an artefact of the Intel A770 struggling to produce a stable 60 FPS in many test cases even without V-Sync) and the Nvidia 4090, which also makes the rest of the system sit at 120 W.
A minor (<5 W) decrease in power draw can generally be observed from the simple renderer to the SSBO stream with static data. Although this effect is explainable by enabling static data, liberating the CPU and system memory from repeatedly copying the data to the VRAM, it is an insignificant change to the overall power consumption compared to how much the different test parameters influence the GPUs' power consumption alone.
It should also be mentioned here that the effects of the test parameters on the rest of the system's power consumption are generally even more negligible. Changes in resolution, dataset size or thinning

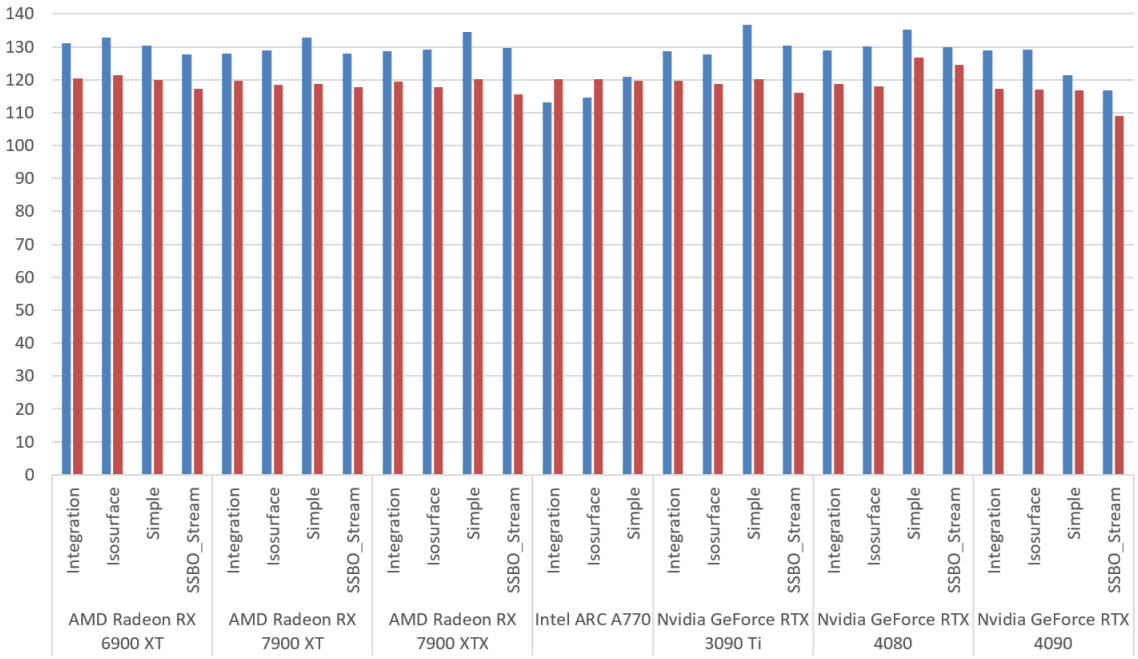


Figure 5.51: Mean System Power (GPU excluded) by **render method**. **left/blue** is with **V-Sync disabled** and **right/red** with **V-Sync enabled**. Each column comprises the mean of the mean efficiency of all test cases that match these parameters.

factor, which, as shown previously, have a significant effect on the GPUs' power consumption, influence the rest of the system's power consumption similarly, but only in a general range of <5 W.

5.3.2 Manufacturer-Dependant TDP and Software Sensor Observations

When discussing certain test cases with high utilization on AMD GPUs earlier, there were multiple cases where the tinkerforge power measurements exceeded the TDP on AMD GPUs by more than 10%. Fig. 5.52 highlights this pattern even more by only including test cases where high GPU utilization was consistently recorded previously (unlocked FPS, the largest available datasets (expl30m for spheres and chameleon for volume) and render methods other than the simple spheres renderer). It is apparent that the Intel and Nvidia GPUs mostly meet but do not exceed their TDPs, while the AMD GPUs consistently overshoot them by 10%. Judging by the findings of Müller et al. [MHWE22], a possible explanation would be that the ADL sensor tends to be off around 10% more than the Nvidia sensor. This divergence would imply that AMD based its TDP ratings on the internal sensor's measurements. However, Fig 5.53 shows that a percentage differing by the manufacturer can not simplify the difference between the external and integrated power sensors. The last-generation GPUs (AMD 6900 XT and Nvidia 3090 Ti) show percentages that match the observations by Müller et al. [MHWE22], which were recorded on GPUs with the exact architectures (AMD W6800 (RDNA 2) and Nvdia 3090 (Ampere)). However, the current-generation AMD GPUs' ADL sensors

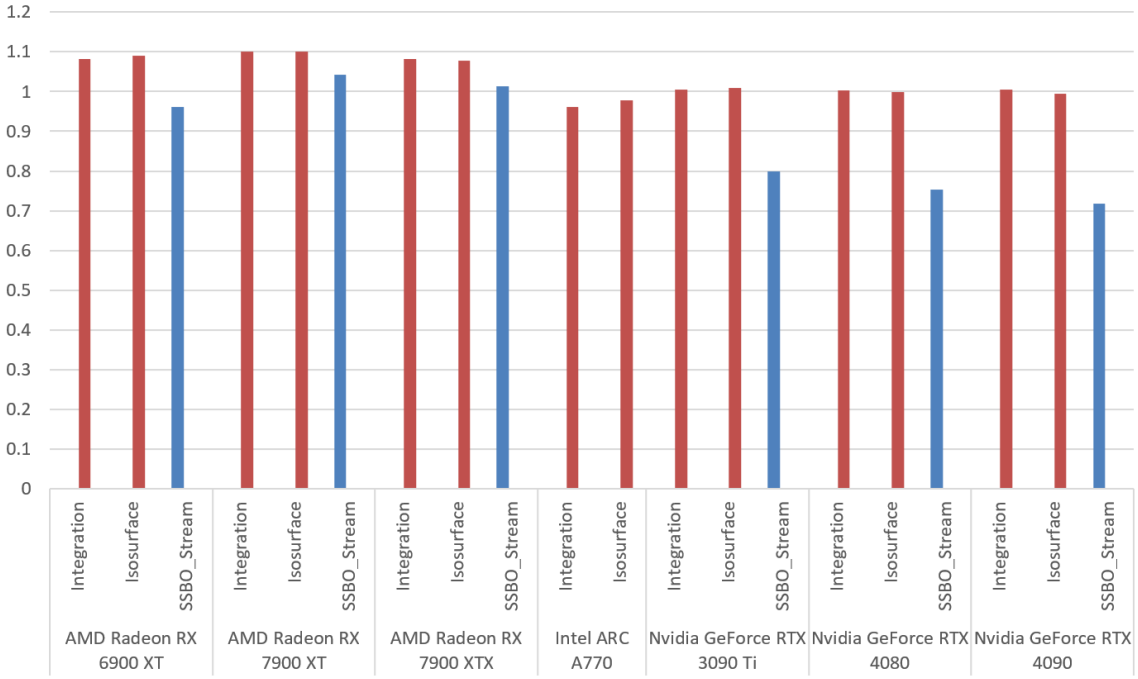
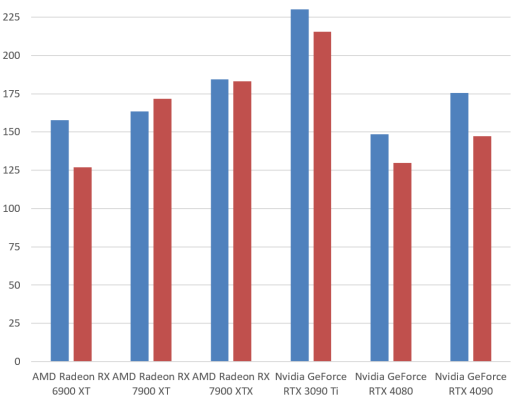
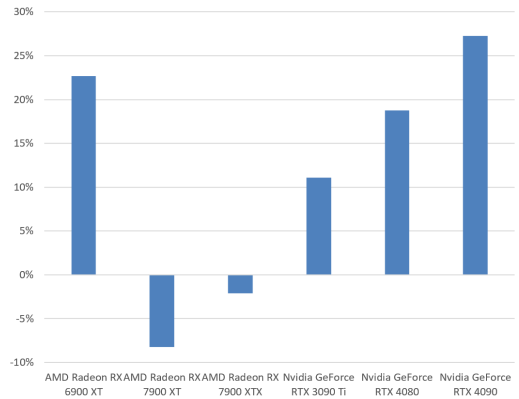


Figure 5.52: GPU Mean Power (normalized to their respective TDP) by render method. Volume render methods are highlighted**red**, the spheres render method **blue**. V-Sync is **disabled**. Only cases with the **largest dataset** (expl30m for spheres and chameleon for volume) are included. The filters were applied specifically to only include **test cases with generally high GPU utilization**. Each column comprises the mean of the mean utilization of all test cases that match these parameters.



(a) Absulte power (W) measured by Tinkerforge (blue/left) and ADL/NVML (red/right)



(b) Relative additional power measured by the Tinkerforge sensors

Figure 5.53: Comparison of average power measurements of tinkerforge and integrated GPU sensors (ADL/NVML). Each column comprises the mean of the mean utilization of all test cases.

tend to report more power than the tinkerforge sensors measure. The current-generation Nvidia GPUs are missing more power than the previous generation. Additionally, the Nvidia 4090 is off by more than the 4080, so this specific Nvidia 4000-series issue could be connected to the TDP.

6 Conclusion and Outlook

In this work, we compared how various combinations of parameters influence the power consumption and efficiency of different visualization methods. Larger and denser datasets, higher resolution, and, for the sphere renderer, larger spheres lead to lower FPS and higher power draw, resulting in worse efficiency. When using the simple sphere renderer, test cases with different datasets and dataset thinning stages appear to be separable into areas. For the SSBO stream with static data and the volume test cases, this is also true to an extent. However, while test cases with equal datasets and thinning stages/sampling intervals behave the most similarly, they are not as clearly separated. Also, all simple sphere renderer test cases show signs of a bottleneck, as FPS and GPU utilization are generally low. However, this only negatively affects the efficiency of the Nvidia GPUs. In volume rendering, Isosurface consistently performed better and more efficiently than Integration. The choice of the render method and practically all other parameters showed little to no influence on the power consumption of the rest of the system. Enabling V-Sync significantly decreases power draw, as long as the GPU can produce much higher FPS when not limited to 60 Hz, but generally not enough to be more efficient on average. Following this observation, limiting FPS for interactive workloads, which are not constrained by having to render as many frames as quickly as possible, benefits power consumption.

We also found that multiple GPU models by the same manufacturer tend to show similar behaviour. When V-Sync is disabled, AMD GPUs always tend to use as much power as they can to produce the highest possible FPS, except for when they run into a bottleneck somewhere above 2000 FPS, while Nvidia GPUs get either steadily more bottlenecked or steadily reduce power draw on purpose, the higher the FPS are. This behaviour also causes AMD GPUs to squeeze more FPS out of smaller and more diluted datasets. The Intel A770 is, behaviour-wise, somewhere in between but leaning more toward the AMD side. Especially at lower FPS levels, with large particle datasets, Nvidia GPUs tend to perform more consistently across different resolutions, sphere radii and camera angles. Despite generally drawing the least power, efficiency-wise, the Intel A770 is mostly outclassed by the other, more power-hungry GPUs because they can generate significantly higher FPS. This advantage is especially true for the Nvidia 4090, which, despite its massive 450 W TDP, shows massive improvements compared to its predecessor (3090 Ti) and often comes out as the most efficient. For a fairer comparison, it would be insightful to compare the Intel A770 to similarly priced and specced offerings from AMD and Nvidia to see if the comparably low efficiency is an architectural issue on Intel's side or simply a shortcoming of lower-powered GPUs. Adding more GPUs of older generations to the comparison could also be insightful to analyze generational efficiency improvements and predict how this trend will continue (assuming GPU manufacturers successfully maintain Dennard Scaling).

Comparing the power consumption measured by our hardware setup and the GPUs' integrated sensors showed significant differences, where, in most cases, the GPUs' integrated sensors reported too little power. There were also significant differences in how much the values were off, even within the same generation of GPUs, so it could be insightful to compare this aspect across multiple GPUs of the same model, although the results from the older GPU generation matched previous

research. If the exact shortcomings of the integrated sensors were known, and preferably also their cause, accurate power measurements would be possible with low overhead.
Nevertheless, despite this work analyzing multiple factors and drawing some conclusions, there is still more than enough that still needs to be explained and a potentially nearly infinite number of parameters that could influence power consumption and efficiency in ways yet to be analyzed and discovered.

Bibliography

[AEE+20] Y. Arafa, A. ElWazir, A. ElKanishy, Y. Aly, A. Elsayed, A.-H. Badawy, G. Chennupati, S. Eidenbenz, N. Santhi. "Verified instruction-level energy consumption measurement for nvidia gpus". In: *Proceedings of the 17th ACM International Conference on Computing Frontiers*. 2020, pp. 60–70 (cit. on pp. 15, 18).

[ASP+12] Y. Abe, H. Sasaki, M. Peres, K. Inoue, K. Murakami, S. Kato. "Power and performance analysis of {GPU-Accelerated} systems". In: *2012 Workshop on Power-Aware Computing and Systems (HotPower 12)*. 2012 (cit. on p. 19).

[BIM16] R. A. Bridges, N. Imam, T. M. Mintz. "Understanding GPU power: A survey of profiling, modeling, and simulation methods". In: *ACM Computing Surveys (CSUR)* 49.3 (2016), pp. 1–27 (cit. on p. 20).

[BZZ14] M. Burtscher, I. Zecena, Z. Zong. "Measuring GPU power with the K20 built-in sensor". In: *Proceedings of Workshop on General Purpose Processing Using GPUs*. 2014, pp. 28–36 (cit. on pp. 18, 39).

[DGY+74] R. Dennard, F. Gaensslen, H.-N. Yu, V. Rideout, E. Bassous, A. LeBlanc. "Design of ion-implanted MOSFET's with very small physical dimensions". In: *IEEE Journal of Solid-State Circuits* 9.5 (1974), pp. 256–268. DOI: 10.1109/JSSC.1974.1050511 (cit. on p. 15).

[GBB+19] P. Gralka, M. Becher, M. Braun, F. Frieß, C. Müller, T. Rau, K. Schatz, C. Schulz, M. Krone, G. Reina, T. Ertl. "MegaMol – a comprehensive prototyping framework for visualizations". In: *The European Physical Journal Special Topics* 227.14 (Mar. 2019), pp. 1817–1829. ISSN: 1951-6401. DOI: 10.1140/epjst/e2019-800167-5. URL: https://doi.org/10.1140/epjst/e2019-800167-5 (cit. on p. 21).

[HBFE17] M. Heinemann, V. Bruder, S. Frey, T. Ertl. "Power efficiency of volume raycasting on mobile devices". In: *Energy (J/60s)* 20 (2017), p. 30 (cit. on p. 18).

[HK10] S. Hong, H. Kim. "An integrated GPU power and performance model". In: *Proceedings of the 37th annual international symposium on Computer architecture*. 2010, pp. 280–289 (cit. on p. 19).

[JA14] B. Johnsson, T. Akenine-Möller. "Measuring per-frame energy consumption of real-time graphics applications". In: *Journal of Computer Graphics Techniques* 3.1 (2014) (cit. on pp. 15, 17).

[JGDA12] B. Johnsson, P. Ganestam, M. C. Doggett, T. Akenine-Möller. "Power Efficiency for Software Algorithms Running on Graphics Processors." In: *High Performance Graphics*. 2012, pp. 67–75 (cit. on pp. 15, 17, 18).

[KPK+21] V. Kandiah, S. Peverelle, M. Khairy, J. Pan, A. Manjunath, T. G. Rogers, T. M. Aamodt, N. Hardavellas. “AccelWattch: A power modeling framework for modern GPUs”. In: *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*. 2021, pp. 738–753 (cit. on pp. 15, 19).

[LHE+13] J. Leng, T. Hetherington, A. ElTantawy, S. Gilani, N. S. Kim, T. M. Aamodt, V. J. Reddi. “GPUWattch: Enabling energy optimizations in GPGPUs”. In: *ACM SIGARCH Computer Architecture News* 41.3 (2013), pp. 487–498 (cit. on pp. 15, 19).

[Lua] *Lua scripting language*. https://www.lua.org/about.html. Accessed: 2023-10-15 (cit. on p. 21).

[Man18] K. Maniar. “Comparing shunt-and hall-based isolated current-sensing solutions in HEV/EV”. In: (2018) (cit. on p. 23).

[MHWE22] C. Müller, M. Heinemann, D. Weiskopf, T. Ertl. “Power Overwhelming: Quantifying the Energy Cost of Visualisation”. In: *2022 IEEE Evaluation and Beyond - Methodological Approaches for Visualization (BELIV)*. 2022, pp. 38–46. DOI: 10.1109/BELIV57783.2022.00009 (cit. on pp. 18, 19, 22, 25, 26, 29, 39, 71).

[mm] *Megamol*. https://github.com/UniStuttgart-VISUS/megamol (cit. on pp. 21, 25, 27).

[mm-f] *Megamol*. https://github.com/chelast55/megamol (cit. on pp. 27, 31).

[pov] *Power Overwhelming*. https://github.com/UniStuttgart-VISUS/power-overwhelming (cit. on p. 22).

[SADK19] Y. Sun, N. B. Agostini, S. Dong, D. Kaeli. “Summarizing CPU and GPU design trends with product data”. In: *arXiv preprint arXiv:1911.11313* (2019) (cit. on pp. 15, 17).

[ST21] E. Sicard, L. Trojman. *Introducing 3-nm Nano-Sheet FET technology in Microwind*. Research report INSA Toulouse France on the Nano-Sheet 3NM CMOS technology implemented in Microwind. Oct. 2021. URL: https://hal.science/hal-03377556 (cit. on p. 15).

[TPGPU] *TechPowerUp GPU Database*. https://www.techpowerup.com/gpu-specs/. Accessed: 2023-10-10 (cit. on pp. 15, 25, 79, 80).

A Extended GPU Specs

Chip M.	GPU	Process Size	Transistor Count	Transistor Density	Die Size	TDP	Min. PSU	Release Date	MSRP (launch)	MSRP (2023)	PCIe Version
AMD	Radeon RX 6900XT	7 nm	26,800M	51.5M / mm²	520 mm²	300 W	700 W	Oct 28th, 2020	999 USD	700 USD	PCIe 4.0 x16
AMD	Radeon RX 7900XT	5 nm	57,700M	109.1M / mm²*	529 mm²	300 W	700 W	Nov 3rd, 2022	899 USD	899 USD	PCIe 4.0 x16
AMD	Radeon RX 7900XTX	5 nm	57,700M	109.1M / mm²*	529 mm²	355 W	750 W	Nov 3rd, 2022	999 USD	999 USD	PCIe 4.0 x16
Intel	ARC A770	6 nm	21,700M	53.4M / mm²	406 mm²	225 W	550 W	Oct 12th, 2022	350 USD	350 USD	PCIe 4.0 x16
Nvidia	GeForce RTX 3090 Ti	8 nm	28,300M	45.1M / mm²	628 mm²	450 W	850 W	Jan 27th, 2022	1999 USD	1499 USD	PCIe 4.0 x16
Nvidia	GeForce RTX 4080	5 nm	45,900M	121.1M / mm²	379 mm²	320 W	700 W	Sep 20th, 2022	1199 USD	1199 USD	PCIe 4.0 x16
Nvidia	GeForce RTX 4090	5 nm	76,300M	125.3M / mm²	609 mm²	450 W	850 W	Sep 20th, 2022	1598 USD	1599 USD	PCIe 4.0 x16

* Transistor density varies between GCD and MCD transistors

Table A.1: Specifications of the selected GPUs regarding various general information (source: TechPowerUp GPU Database [TPGPU]).

Chip M.	GPU	Architecture	Variant	Base Clock	Boost Clock	VRAM Clock	VRAM Clock (effective)
AMD	Radeon RX 6900XT	RDNA 2.0	ASUS TUF Gaming Radeon™ RX 6900 XT TOP Edition 16GB GDDR6	1900 MHz	2310 MHz	2000 MHz	16 Gbps
AMD	Radeon RX 7900XT	RDNA 3.0	XFX Speedster MERC310 RX 7900 XT	1810 MHz	2560 MHz	2500 MHz	20 Gbps
AMD	Radeon RX 7900XTX	RDNA 3.0	ASUS TUF Gaming Radeon RX 7900 XTX OC Edition 24GB GDDR6	1895 MHz	2565 MHz	2500 MHz	20 Gbps
Intel	ARC A770	Alchemist	Intel Arc A770 Limited Edition 16GB	2100 MHz	2400 MHz	2187 MHz	17.5 Gbps
Nvidia	GeForce RTX 3090 Ti	Ampere	ASUS TUF RTX 3090 Ti GAMING OC	1560 MHz	1920 MHz	1313 MHz	21 Gbps
Nvidia	GeForce RTX 4080	Ada Lovelace	ASUS TUF Gaming GeForce RTX 4080 16GB GDDR6X OC Edition	2205 MHz	2595 MHz	1400 MHz	22.4 Gbps
Nvidia	GeForce RTX 4090	Ada Lovelace	NVIDIA GeForce RTX 4090	2235 MHz	2520 MHz	1313 MHz	21 Gbps

Table A.2: Specifications of the selected GPUs regarding GPU variants and their variant-specific clocks (source: TechPowerUp GPU Database [TPGPU]).

Chip M.	GPU	VRAM Capacity	VRAM Type	VRAM Bus Width	VRAM Bandwidth	Pixel Rate	Texture Rate	FP16 (half)	FP32 (float)	FP64 (double)
AMD	Radeon RX 6900XT	16 GB	GDDR6	256 bit	512 GB/s	295.7 GPixel/s	739.2 GTexel/s	47.31 TFLOPS (2:1)	23.65 TFLOPS	1.478 TFLOPS (1:16)
AMD	Radeon RX 7900XT	20 GB	GDDR6	320 bit	800 GB/s	491.5 GPixel/s	860.2 GTexel/s	110.1 TFLOPS (2:1)	55.05 TFLOPS	1.720 TFLOPS (1:32)
AMD	Radeon RX 7900XTX	24 GB	GDDR6	384 bit	960 GB/s	492.5 GPixel/s	985.0 GTexel/s	126.1 TFLOPS (2:1)	63.04 TFLOPS	1.970 TFLOPS (1:32)
Intel	ARC A770	16 GB	GDDR6	256 bit	559.9 GB/s	307.2 GPixel/s	614.4 GTexel/s	39.32 TFLOPS (2:1)	19.66 TFLOPS	N/A
Nvidia	GeForce RTX 3090 Ti	24 GB	GDDR6X	384 bit	1008 GB/s	215.0 GPixel/s	645.1 GTexel/s	41.29 TFLOPS (1:1)	41.29 TFLOPS	645.1 GFLOPS (1:64)
Nvidia	GeForce RTX 4080	16 GB	GDDR6X	256 bit	716.8 GB/s	290.6 GPixel/s	788.9 GTexel/s	50.49 TFLOPS (1:1)	50.49 TFLOPS	788.9 GFLOPS (1:64)
Nvidia	GeForce RTX 4090	24 GB	GDDR6X	384 bit	1008 GB/s	443.5 GPixel/s	1,290 GTexel/s	82.58 TFLOPS (1:1)	82.58 TFLOPS	1.290 TFLOPS (1:64)

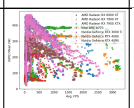
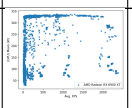
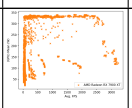
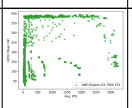
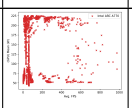
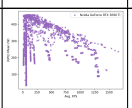
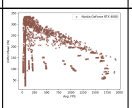
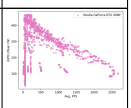
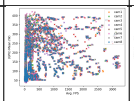
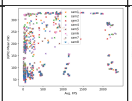
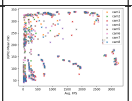
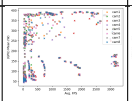
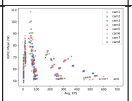
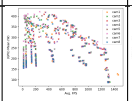
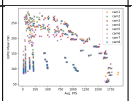
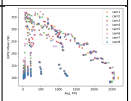
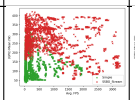
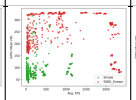
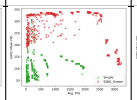
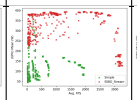
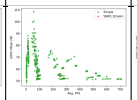
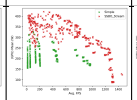
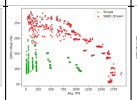
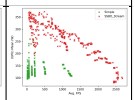
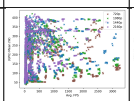
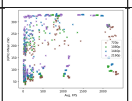
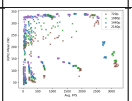
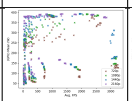
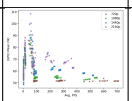
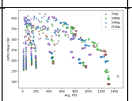
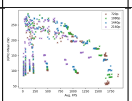
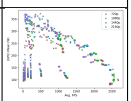
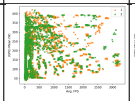
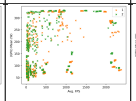
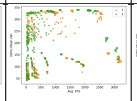
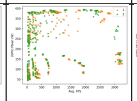
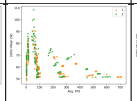
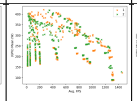
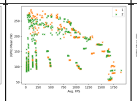
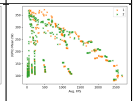
Table A.3: Specifications of the selected GPUs regarding VRAM and theoretical performance (source: TechPowerUp GPU Database [TPGPU]).

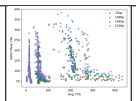
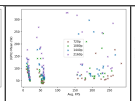
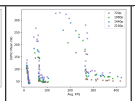
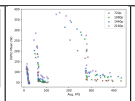
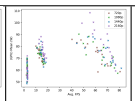
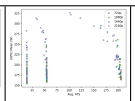
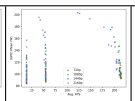
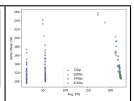
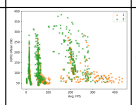
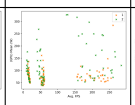
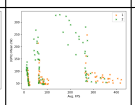
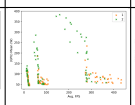
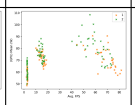
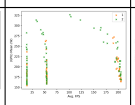
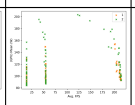
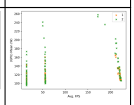
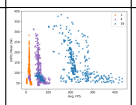
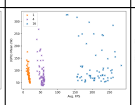
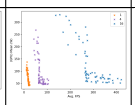
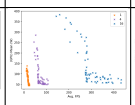
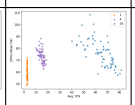
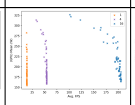
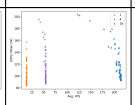
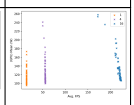
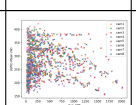
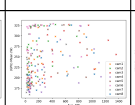
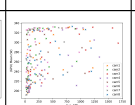
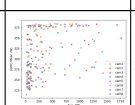
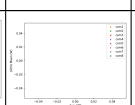
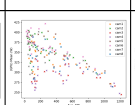
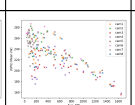
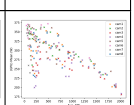
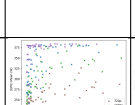
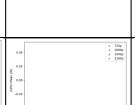
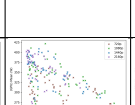
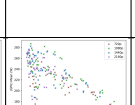
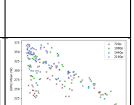
Chip M.	GPU	Shader Units	TMUs	ROPs	Compute Cores*	Tensor Cores	RT Cores	L1 Cache	L2 Cache	Direct X	OpenGL	OpenCL	Vulkan	CUDA	Shader Model
AMD	Radeon RX 6900XT	5120	320	128	80	N/A	80	128 KB (/Array)	4MB**	12_2	4.6	2.1	1.3	N/A	6.7
AMD	Radeon RX 7900XT	5376	223	192	84	N/A	84	256 KB (/Array)	6 MB***	12_2	4.6	2.2	1.3	N/A	6.7
AMD	Radeon RX 7900XTX	6144	384	192	96	N/A	96	256 KB (/Array)	6 MB****	12_2	4.6	2.2	1.3	N/A	6.7
Intel	ARC A770	4096	256	128	512	512	32	N/A	16 MB	12_2	4.6	3.0	1.3	N/A	6.6
Nvidia	GeForce RTX 3090 Ti	10752	336	112	84	336	84	128 KB (/SM)	6 MB	12_2	4.6	3.0	1.3	8.6	6.7
Nvidia	GeForce RTX 4080	9728	304	112	76	304	76	128 KB (/SM)	64 MB	12_2	4.6	3.0	1.3	8.9	6.7
Nvidia	GeForce RTX 4090	16384	512	176	128	512	128	128 KB (/SM)	72 MB	12_2	4.6	3.0	1.3	8.9	6.7

*This includes Compute Units (AMD), Execution Units (Intel) and Streaming Multiprocessors (Nvidia)
**additionally 128 MB of L3 Cache and 32KB of L0 Cache (per WGP)
***additionally 80 MB of L3 Cache and 64KB of L0 Cache (per WGP)
****additionally 96 MB of L3 Cache and 64KB of L0 Cache (per WGP)

Table A.4: Specifications of the selected GPUs regarding their render configuration and feature levels (source: TechPowerUp GPU Database [TPGPU]).

B Additional Scatterplots

GPU M. Power (W) / Mean FPS	all GPU	AMD 6900 XT	AMD 7900 XT	AMD 7900 XTX	Intel A770	Nvidia 3090 Ti	Nvidia 4080	Nvidia 4090
All Cases								
Spheres (no V-Sync) by Camera Angle								
Spheres (no V-Sync) by Render Method								
Spheres (no V-Sync) by Resolution								
Spheres (no V-Sync) by Sphere Radius								
Spheres (no V-Sync) by Test Data								
Spheres (no V-Sync) by Thinning Factor								
Spheres (simple, expl, no V-Sync) by Camera Angle								
Spheres (simple, expl, no V-Sync) by Render Method								

Spheres (simple, expl, no V-Sync) by Resolution								
Spheres (simple, expl, no V-Sync) by Sphere Radius								
Spheres (simple, expl, no V-Sync) by Thinning Factor								
Spheres (SSBO static data, expl, no V-Sync) by Camera Angle								
Spheres (SSBO static data, expl, no V-Sync) by Resolution								
Spheres (SSBO static data, expl, no V-Sync) by Sphere Radius								
Spheres (SSBO static data, expl, no V-Sync) by Thinning Factor								
Spheres by V-Sync								
Test Case								
Volume (chameleon, no V-Sync) by Camera Angle								

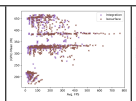
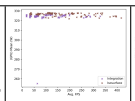
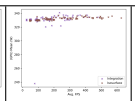
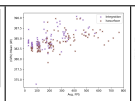
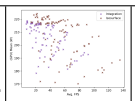
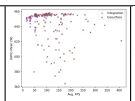
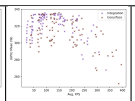
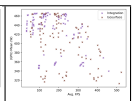
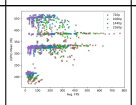
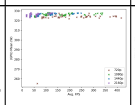
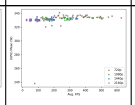
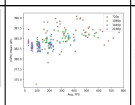
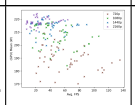
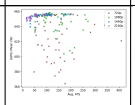
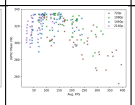
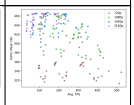
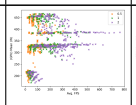
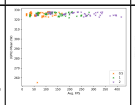
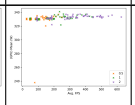
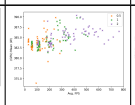
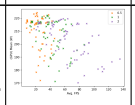
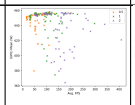
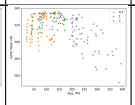
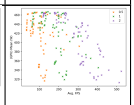
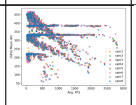
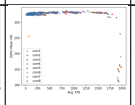
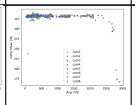
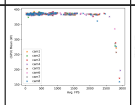
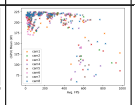
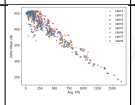
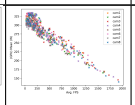
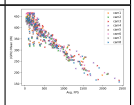
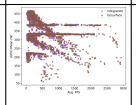
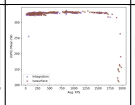
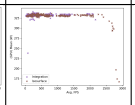
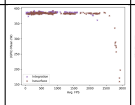
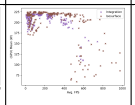
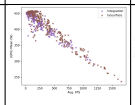
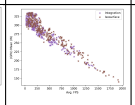
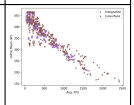
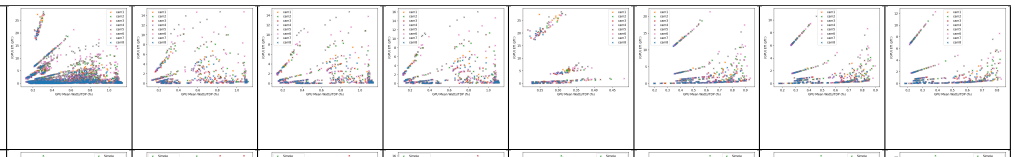
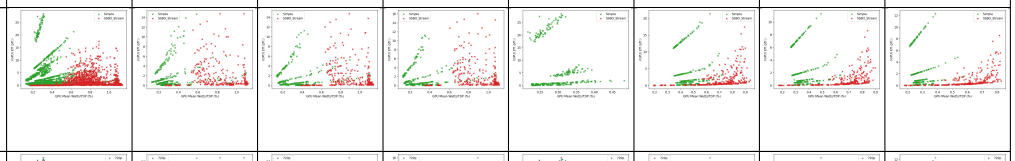
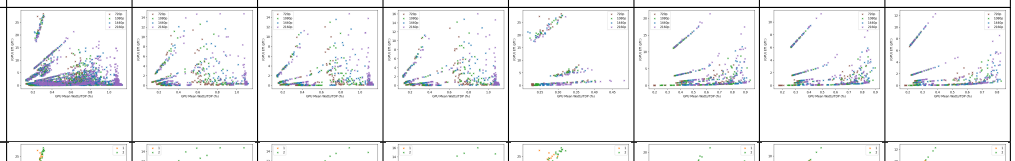
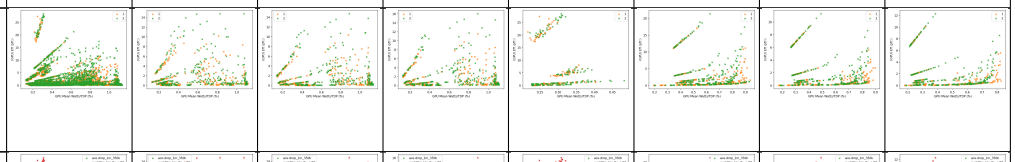
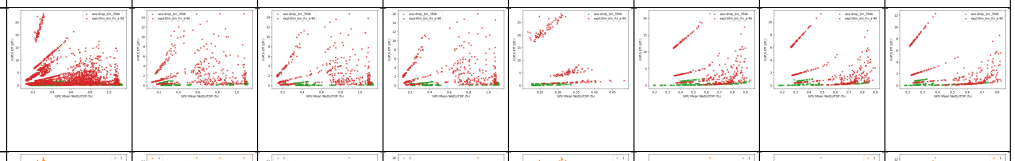
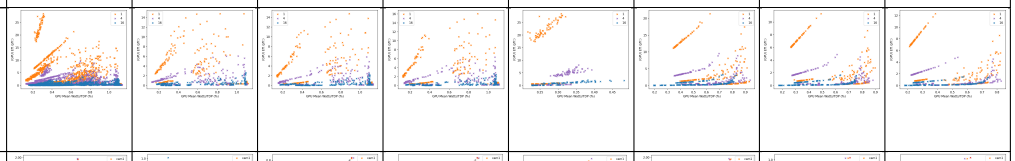
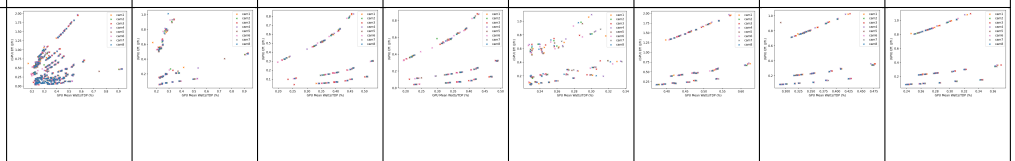
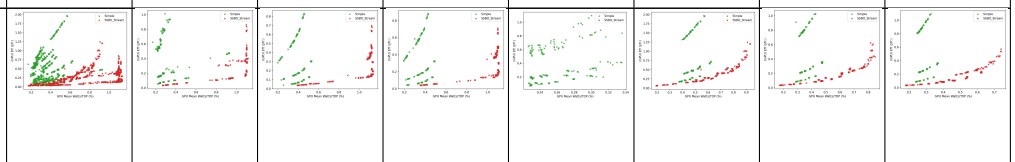
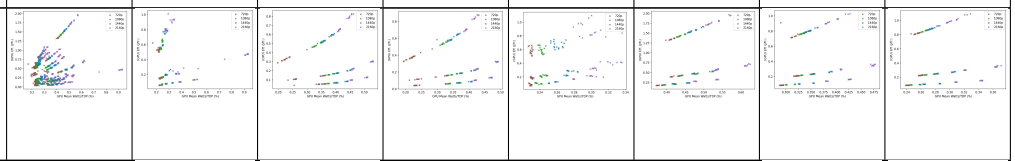
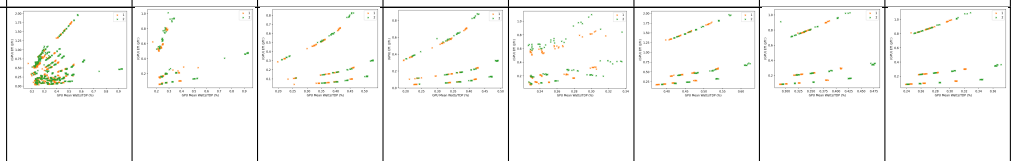
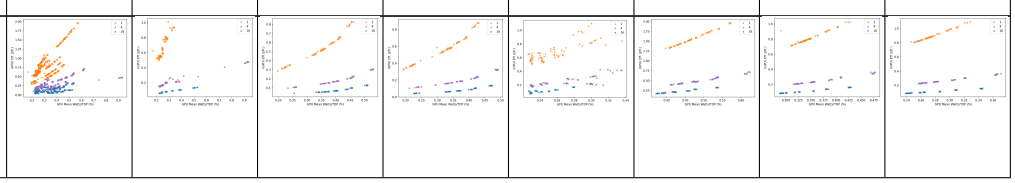
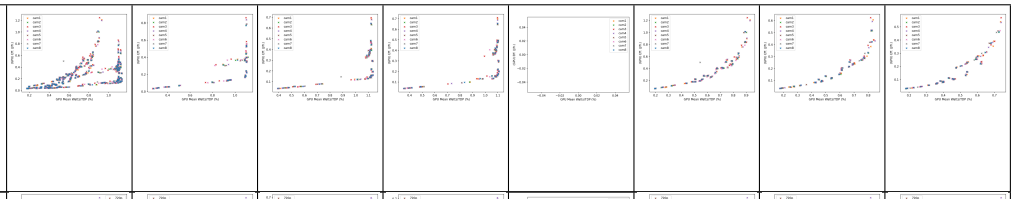
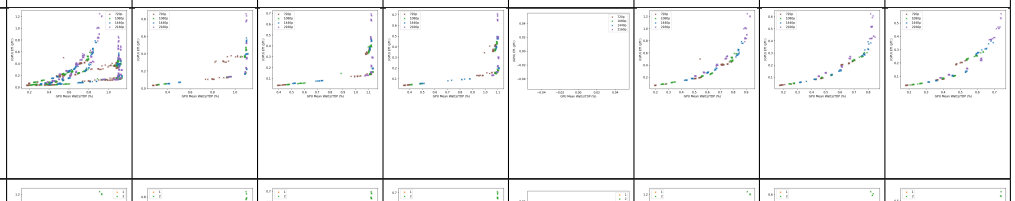
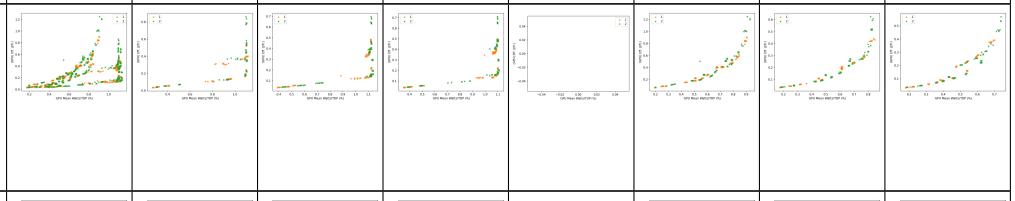
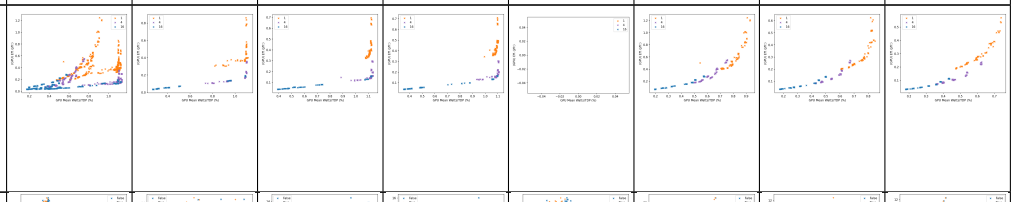
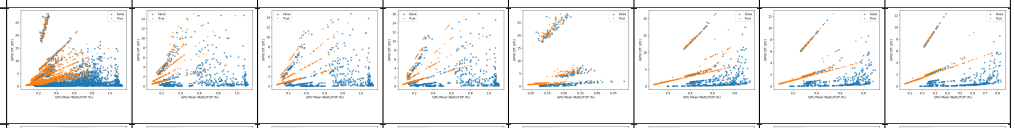
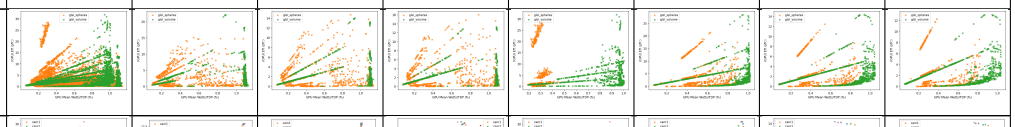
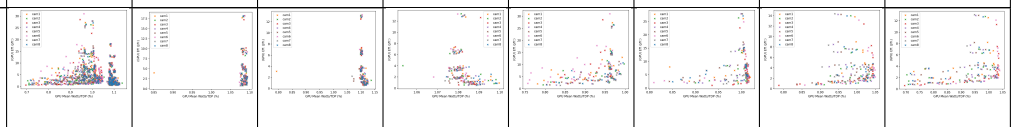
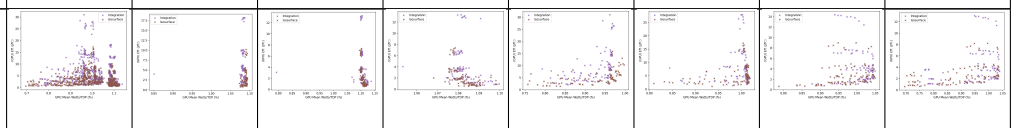
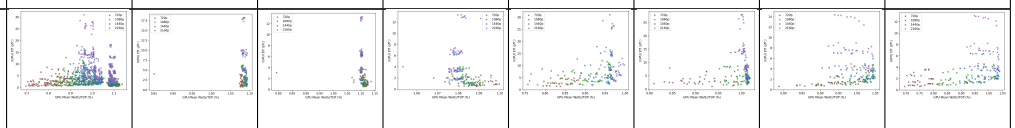
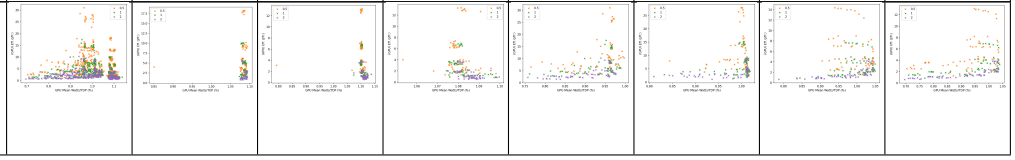
Volume (chameleon, no V-Sync) by Render Method								
Volume (chameleon, no V-Sync) by Resolution								
Volume (chameleon, no V-Sync) by Step Ratio								
Volume (no V-Sync) by Camera Angle								
Volume (no V-Sync) by Render Method								
Volume (no V-Sync) by Resolution								
Volume (no V-Sync) by Step Ratio								
Volume (no V-Sync) by Test Data								
Volume by V-Sync								

Table B.1: More scatterplots of Mean GPU Power and Mean FPS. Each point is one test case with different parameters. Different test parameters are highlighted and test cases.

GPU Eff. (J/fr.) / Appr. Util. (%)	all GPU	AMD 6900 XT	AMD 7900 XT	AMD 7900 XTX	Intel A770	Nvidia 3090 Ti	Nvidia 4080	Nvidia 4090
All Cases								

Spheres (no V-Sync) by Camera Angle								
Spheres (no V-Sync) by Render Method								
Spheres (no V-Sync) by Resolution								
Spheres (no V-Sync) by Sphere Radius								
Spheres (no V-Sync) by Test Data								
Spheres (no V-Sync) by Thinning Factor								
Spheres (simple, ace, no V-Sync) by Camera Angle								
Spheres (simple, ace, no V-Sync) by Render Method								
Spheres (simple, ace, no V-Sync) by Resolution								
Spheres (simple, ace, no V-Sync) by Sphere Radius								
Spheres (simple, ace, no V-Sync) by Thinning Factor								

Spheres (SSBO static data, expl, no V-Sync) by Camera Angle								
Spheres (SSBO static data, expl, no V-Sync) by Resolution								
Spheres (SSBO static data, expl, no V-Sync) by Sphere Radius								
Spheres (SSBO static data, expl, no V-Sync) by Thinning Factor								
Spheres by V-Sync								
Test Case								
Volume (chameleon, no V-Sync) by Camera Angle								
Volume (chameleon, no V-Sync) by Render Method								
Volume (chameleon, no V-Sync) by Resolution								
Volume (chameleon, no V-Sync) by Step Ratio								

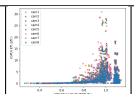
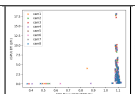
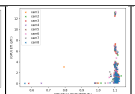
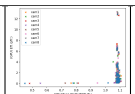
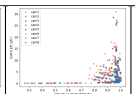
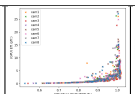
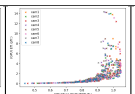
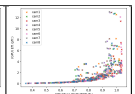
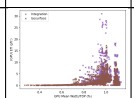
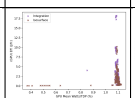
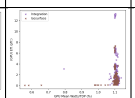
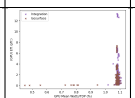
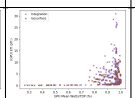
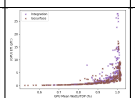
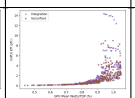
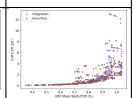
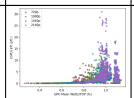
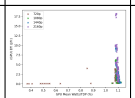
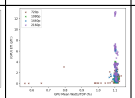
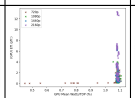
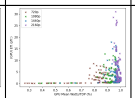
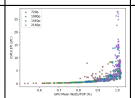
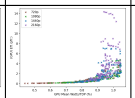
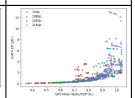
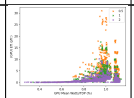
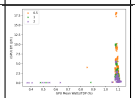
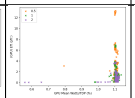
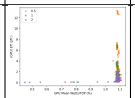
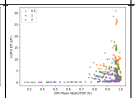
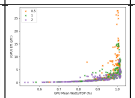
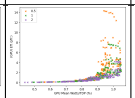
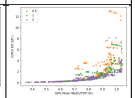
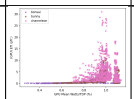
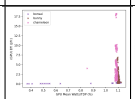
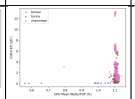
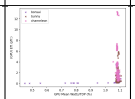
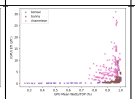
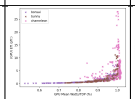
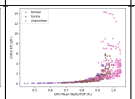
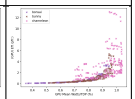
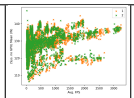
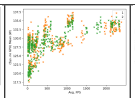
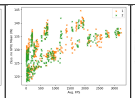
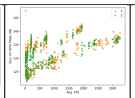
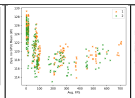
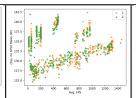
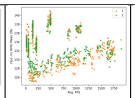
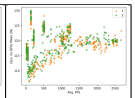
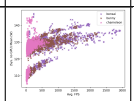
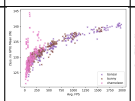
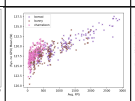
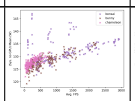
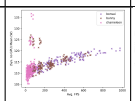
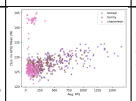
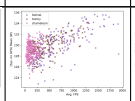
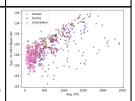
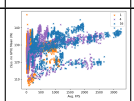
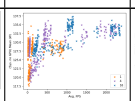
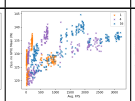
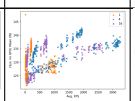
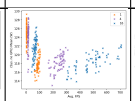
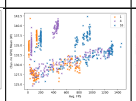
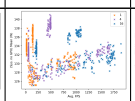
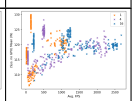
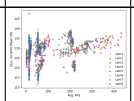
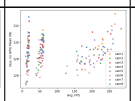
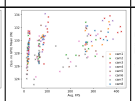
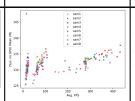
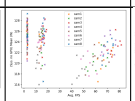
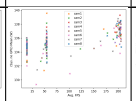
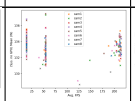
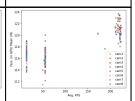
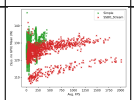
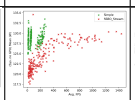
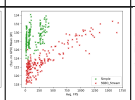
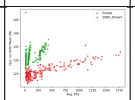
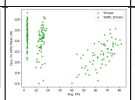
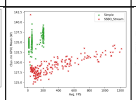
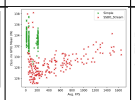
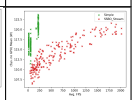
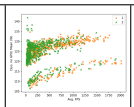
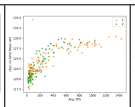
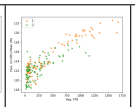
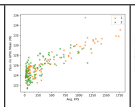
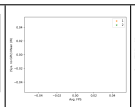
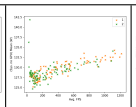
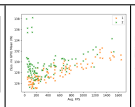
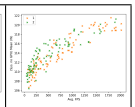
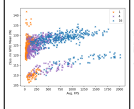
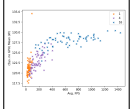
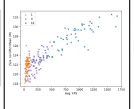
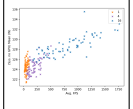
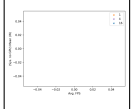
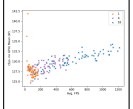
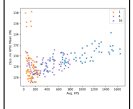
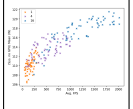
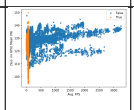
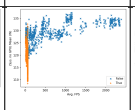
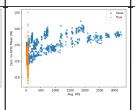
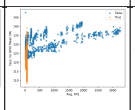
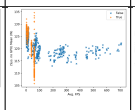
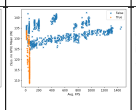
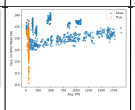
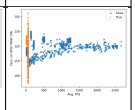
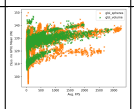
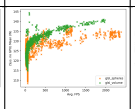
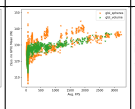
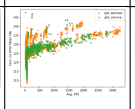
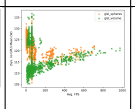
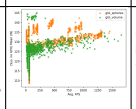
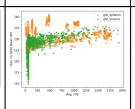
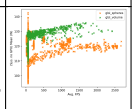
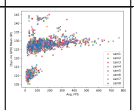
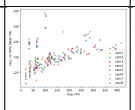
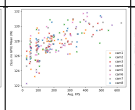
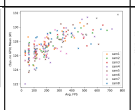
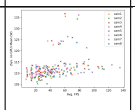
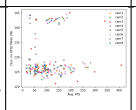
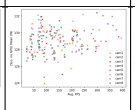
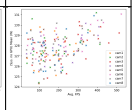
Volume (no V-Sync) by Camera Angle								
Volume (no V-Sync) by Render Method								
Volume (no V-Sync) by Resolution								
Volume (no V-Sync) by Step Ratio								
Volume (no V-Sync) by Test Data								
Volume by V-Sync								

Table B.2: More scatterplots of GPU Efficiency (J per frame) and GPU Mean Power (normalized to their respective TDP). Each point is one test case with different parameters. Different test parameters are highlighted and test cases.

Sys. M. Power (W) / Mean FPS	all GPU	AMD 6900 XT	AMD 7900 XT	AMD 7900 XTX	Intel A770	Nvidia 3090 Ti	Nvidia 4080	Nvidia 4090
All Cases								
Spheres (no V-Sync) by Camera Angle								
Spheres (no V-Sync) by Render Method								
Spheres (no V-Sync) by Resolution								

Spheres (no V-Sync) by Sphere Radius								
Spheres (no V-Sync) by Test Data								
Spheres (no V-Sync) by Thinning Factor								
Spheres (simple, expl, no V-Sync) by Camera Angle								
Spheres (simple, expl, no V-Sync) by Render Method								
Spheres (simple, expl, no V-Sync) by Resolution								
Spheres (simple, expl, no V-Sync) by Sphere Radius								
Spheres (simple, expl, no V-Sync) by Thinning Factor								
Spheres (SSBO static data, expl, no V-Sync) by Camera Angle								
Spheres (SSBO static data, expl, no V-Sync) by Resolution								

Spheres (SSBO static data, expl, no V-Sync) by Sphere Radius								
Spheres (SSBO static data, expl, no V-Sync) by Thinning Factor								
Spheres by V-Sync								
Test Case								
Volume (chameleon, no V-Sync) by Camera Angle								
Volume (chameleon, no V-Sync) by Render Method								
Volume (chameleon, no V-Sync) by Resolution								
Volume (chameleon, no V-Sync) by Step Ratio								
Volume (no V-Sync) by Camera Angle								
Volume (no V-Sync) by Render Method								
Volume (no V-Sync) by Resolution								

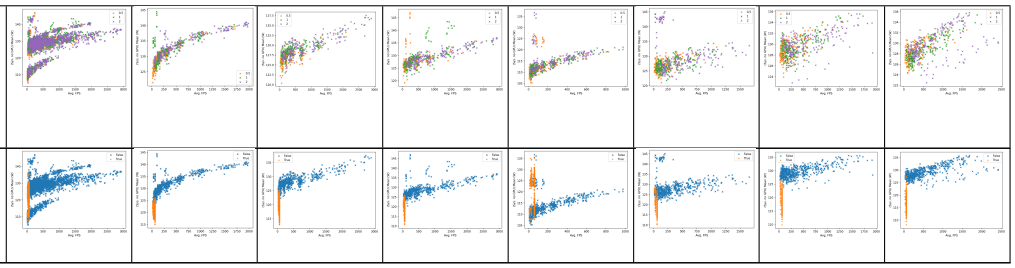
Volume (no V-Sync) by Step Ratio	
Volume by V-Sync	

Table B.3: More scatterplots of Mean Power of the System (excluding GPU power) and Mean FPS. Each point is one test case with different parameters. Different test parameters are highlighted and test cases.

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

---

place, date, signature