Hochschule Karlsruhe
University of
Applied Sciences

Fakultät für
Informatik und
Wirtschaftsinformatik

Seminar

Name:	Hazard
Vorname:	Nathanael
Matr.-Nr.:	74247
E-Mail:	hana1020@h-ka.de
Tel.:	017644280333
Im Semester:	WS 2024
Studiengang:	INFB (PO 7)

Thema

Hardwareanforderungen für Neural Radiance Field (NeRF) Inferenz

Topic

Hardware requirements for Neural Radiance Field (NeRF) Inference

Bearbeitungsvermerke

Referent:	Prof. Dr.-Ing. Laubenheimer
Anmeldestatus:	Angemeldet
Start der Arbeit:	02.10.2024

Declaration of Authorship

I hereby declare that this thesis is my own work and that I have not used any sources or aids except the ones declared or referenced. Text passages quoted verbatim or near-verbatim from external sources, as well as figures taken from or based on external sources, are marked as such.
I further declare that this thesis has not been submitted to any other examination board in the same or similar form.

Karlsruhe, den 20.11.2024 ..... 

Place, Date Nathanael Hazard

List of Figures

Abstract

This paper explores the potential of Neural Radiance Fields (NeRF) in 3D scene reconstruction, focusing on new methods allowing for real-time edge-device inference. The goal is to find out the obstacles that hinder professional-grade performance to be accessible by everyday users. The research begins with a theoretical overview of NeRFs and their computational challenges, followed by a detailed exploration of hardware accelerators, mainly advanced GPUs. This helps understand a case-study which revolves around training and deploying a real-time Neural Light Field (NeLF) model using a teacher-student approach. The results demonstrate that NeLF models can achieve real-time inference with acceptable visual quality on consumer hardware, significantly outperforming traditional NeRF implementations in speed and efficiency. However, training remains resource-intensive, requiring powerful hardware and large datasets, which limits accessibility for casual users. There is a great number of new technologies in this field. This research is limited to the discussed technologies but could miss out on other successful alternatives. Future research should focus on reducing input-data requirements as this is the main obstacle that will keep hindering broad consumer adoption.

Keywords

Neural Radiance Fields, Neural Light Fields, 3D Scene Reconstruction, Real-Time Rendering, Hardware Accelerators, Machine Learning, Consumer Applications.

Table of Contents

1 Introduction

1.1 Motivation

Neural Radiance Fields (NeRF) offer a transformative approach to creating 3D models from collections of 2D images, with broad applicability across numerous industries. While it is a relatively young field with the first publication by Mildenhall et al. [1] appearing in 2020, it has since gained a vast number of papers that have been published in addition to fast advancements. This is easily measurable, as more than 620 preprints and publications have been released since 2020, with more than 250 new code repositories on GitHub. [2]

Creating a 3D model has always been necessary in many industries. In game design, for instance the preferred method usually is to recreate any real-life object with 3D modelling software like Blender. This ensures the lowest number of Polygons (low file size) and the freedom to add any texture to the Model. If the game designer needs to use a real-life object while maintaining the original texture, a digital recreation can be very demanding. 3D scanning can be used, for smaller objects at least, but the original texture can't be retained. The preferred method is then often photogrammetry or photogrammetry combined with 3D scanning or 3D modelling.

This technique consists of taking a huge number of overlapping photographs of a static object with a high-resolution camera, as can be seen in Figure 1.

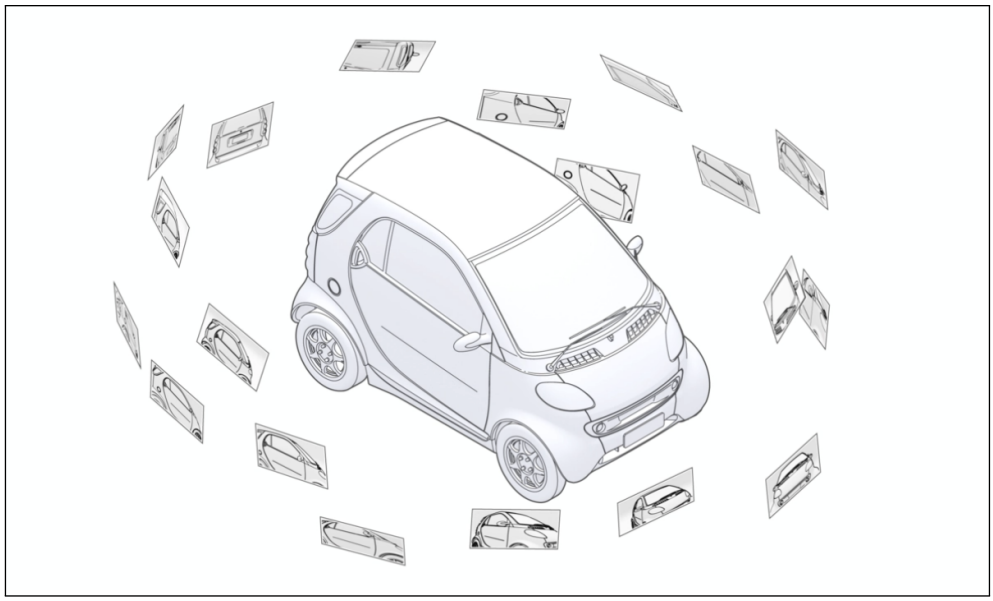

Figure 1: Overlapping Pictures from Different Angles for Photogrammetry. [3]

Those many Gigabytes of data only contain the texture initially. Then, an algorithm is used, recognizing the same visual points on multiple pictures and performs triangulation to recreate the object in a 3D space. See Figure 2.

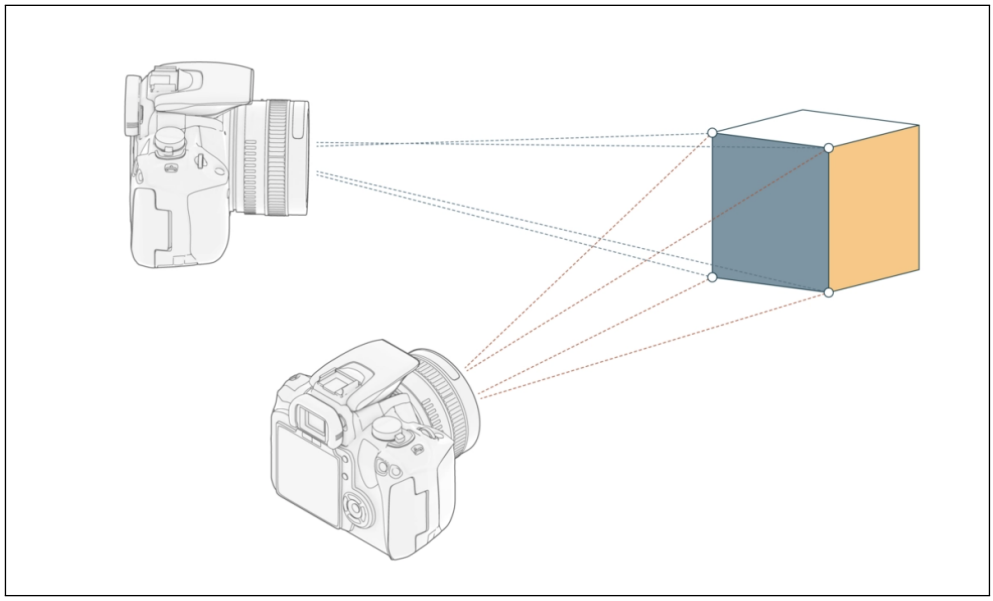

Figure 2: Visualization of Common Point Indentification for Triangulation[4]

The depth of each point can be computed since pictures from different angles will

show those data points being bigger or smaller depending on their relative distance. [5]

This method can be highly useful reproducing the textures accurately but demands a very expensive studio setup, an expensive camera or multiple cameras. Only with 100% photograph consistency, it is possible to have a complete 3D model, and any blind spot will be missing in the result. Additionally, the triangulation work is very compute intensive. For these reasons alone the newer approach of using NeRFs is much more promising. NeRFs can not only generate a complete scene in 3D instead of just an object, but can do so even with slight inconsistencies in images, actually tolerating some blind spots.

1.2 Problem

While NeRFs offers exceptional capabilities, its computational demands pose significant challenges. Training and inference require intensive computations, as the underlying neural networks process large datasets to construct accurate 3D scenes.[6] Tasks such as sampling numerous points in 3D space to estimate radiance and density (of an object's surface) necessitate high processing power, which is not always feasible for consumer-grade or edge devices.

Edge devices, such as smartphones, laptops, or embedded systems, typically have limited computational resources, including constrained memory and processing capacity. [7] This makes deploying NeRFs in real-time applications or on portable devices difficult. For example, creating 3D scenes from personal photographs for entertainment or creative purposes requires fast and efficient processing. However, the computational intensity of NeRFs often limits its performance on such devices. [8] Independent game studios or sole developers usually don't have access to high budget setups needed for photogrammetry. NeRFs, while cutting the need for high quality imagery drastically introduces a new cost problem. It is rare for the average consumer to have a home server with specialized hardware for accelerating neural network inference.

Overcoming these challenges requires significant optimization in both software

and hardware, ensuring that NeRFs can deliver high-quality results even in resource-constrained environments. [9]

1.3 Approach and Work Environment

This paper has the goal to study the advances in both consumer hardware and NeRF techniques that accelerate the ultimate goal of enabling an everyday consumer to have access to cheap and highly accurate 3D modeling software on their personal consumer hardware. By reviewing the common literature on the functioning of NeRFs, the necessary knowledge basis will be presented. This will be used as a foundation to then explore alternative methods that achieve better results on relatively underpowered edge-devices. An introduction in specialized hardware for neural network inference will follow. This is needed to understand what hardware blocks are used for those advance models and where there is room for improvement for future hardware generations.

With this established theoretical background knowledge, the work shifts from a theoretical paper to a more practical one. The work environment allows for access to modern consumer hardware that will be used to run some advanced models for 3D scene reconstruction, when possible. Those results will be documented and used to present the path of progress we are on and then enable us to make predictions, about the state of NeRF technology in consumer hands in the coming years.

1.4 Outline

This paper is structured as follows:

The first chapter establishes a necessary knowledge basis an both the software and the hardware side of the problem. First, it provides an overview on NeRFs while also exploring two new advance models that fit the restrictions of consumer hardware better. Then it provides a foundation on hardware that accelerates neural network inference. The main chapter explores actual open-source models that utilze the techniques previously discussed. Those are then run on local hardware while documenting various metrics

resulting from them. The subsequent chapter outlines the results of these metrics, while focusing on performance and power consumption, as well as performance estimate on advanced and future hardware. The last chapter provides a conlusion on the adaptability of NeRF technology in consumer spaces. It also touches on its weaknesses and puts out recommendations as an outlook.

Through this structured approach, the paper will provide a comprehensive evaluation of 3D scene reconstruction using NeRFs, in a consumer setting, helping readers understand in which direction this technology is heading.

2 Theoretical Background

This section dwelves into the necessery theoretical background needed to understand the software and hardware problem. It begins with an overview on Neural Radiance fields, explaining the key concepts while also giving some application context to facilitate it.

Then an introduction into hardware accelerators is provided. This provides the background information required to understand why progress is mainly obtained on certain hardware architectures.

2.1 Overview on Neural Radiance Fields

Neural Radiance Fields (NeRF) are a method in computer vision that reconstructs 3D scenes from 2D images. Unlike photogrammetry it does not use triangulation. Although it needs positional on every picture (x-y-z coordinates) of the viewfinder the technique itself does not involve explicitly calculating the 3D point correspondences by triangulating. This is because the 3D geometry of the scene is never explicitly computed.

NeRF implicitly encodes 3D geometry by learning how light interacts with objects in a scene, based on the input images provided from different viewpoints. The network learns to predict scene properties, like density and emitted color at any given point in 3D space—by minimizing photometric error across multiple views. [2]

At its core, NeRFs model a scene as a continuous volumetric function that predicts the color and density at each point in 3D space. This function is implemented using a multilayer perceptron (MLP), which takes spatial coordinates and viewing direction as inputs and outputs the corresponding radiance and density. [6]

This approach will be explained in more details by leaning on the work of Mildenhall et al. (2020) especially for their great visual examples. [1]

Architecture and Workflow

NeRF begins with a collection of 2D images of a scene, paired with their corresponding camera positions (3D - coordinates). To represent the 3D scene as a continuous function, we map a position (3D), and a viewing direction (2D), to a color (3D) and volume density (1D). This architecture is visualized in Figure 3.

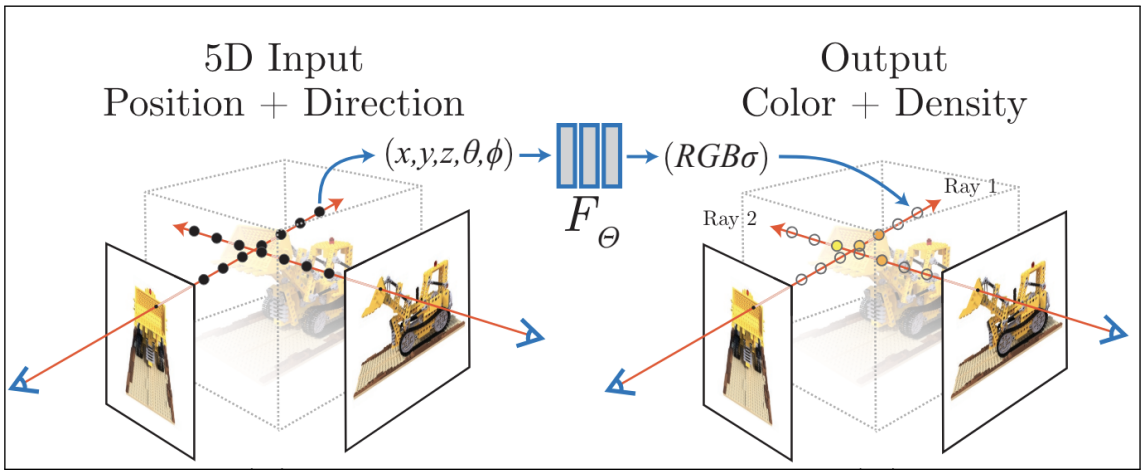

Figure 3: Representing Scenes as Neural Radiance Fields [1]

The 5D inputs are used as variables of a function because neural networks are universal function approximators, meaning they can learn to map these inputs to the corresponding color and density of the scene, like a function does. Now, we can approximate this continuous function that represents the scene we want with a simple Multi-Layer Perceptron. With enough data, we enable the neural network to learn the spatial structure of the scene. [7]

When looking at Figure 3 we can see that the left ray, that will be encoded in a 5D Vector goes straight into the yellow bucket of the excavator. When looking at the volume rendering of only this one ray (the way every ray will be stored in the Neural Network) we notice that there will be only one specific coordinate on that ray's path where the value (1D) that describes the volumes density should be high. This is why the Ray 1 in Figure 4 has one hump. The evaluated color (in this case yellow) is also fed as part

of that ray into the model and will only affect the "pixels" on that rays path that have an elevated volume density value. Incorporating the ray data, the 4D-output consisting of color and density is then done applying it to the neural network through the loss function. The model already provides an output for the 5D-input of Ray 1, although at first a random one. Since the output is now "known", this error, here called rendering loss can now be minimized with a simple calculation known as gradient descent.

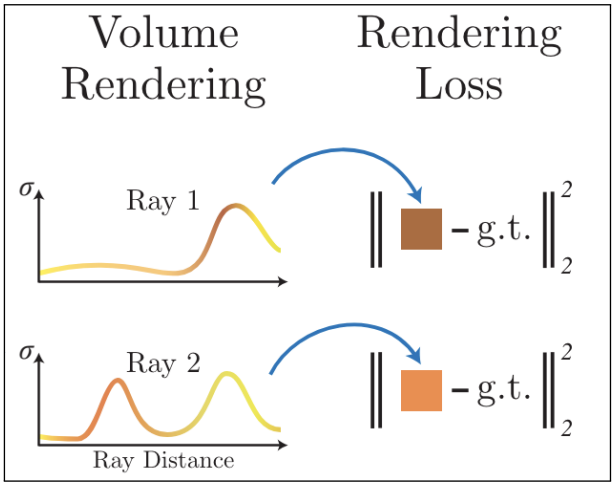

Figure 4: Updating the Neural Network [1]

The following Figure 5 gives a more theoretical visualization of the key step involving looking at a 3D space through 2D lines that describe volume density.

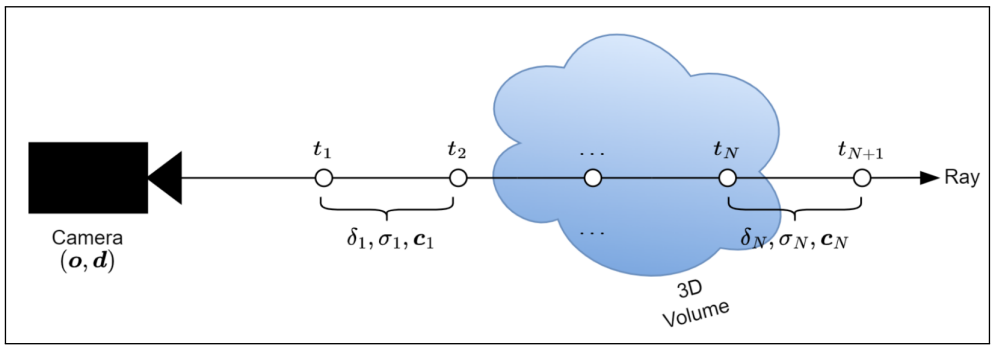

Figure 5: Volume Density Computation [2]

This step is called volume rendering.

A ray starts at the camera position and moves through the scene along a specified direction. The ray is divided into several evenly spaced intervals or points, which are sampled along its path (labeled $t_1, t_2, \ldots, t_N$ in the figure).

Each point along the ray represents a location in 3D space and is associated with three values:

1. Density (σ): This indicates how much light is absorbed or scattered at that point. Higher density typically means the ray has encountered an object or some material.

2. Color (c): The color emitted or reflected by the material at that point, which may depend on the viewing direction.

3. Distance (δ): The small interval between consecutive sampled points along the ray.

For each point, the accumulated light or radiance (the 'r' in NeRF) is calculated by combining these values. The idea is to blend the contributions of all sampled points along the ray to determine the final color of the pixel. The blending takes into account, how much light passes through or is blocked by earlier points, called transmittance and the emitted light and color from each sampled point, which contribute to the pixel's final color.

The process essentially integrates the densities and colors along the ray to simulate how light would realistically interact with the scene. The final pixel color is a combination of all these contributions, weighted by their densities and transmittance, so brighter and denser regions dominate the color while transparent or empty spaces contribute less. (like t_1, t_2 if the object was the blue cloud) [2]

Most importantly for the MLP, this process is differentiable, which means NeRF can optimize the parameters of its neural network by comparing the predicted pixel color to the actual observed color in the input images and adjusting its internal MLP weights to minimize the error. This is what allows NeRF to learn a highly accurate representation of the scene from multiple viewpoints.

The Multilayer Perceptron

The following Figure 6 shall help as a short refresher on neural networks and their ability to encode data and update their weights with a loss function and optimizing with gradient descent:

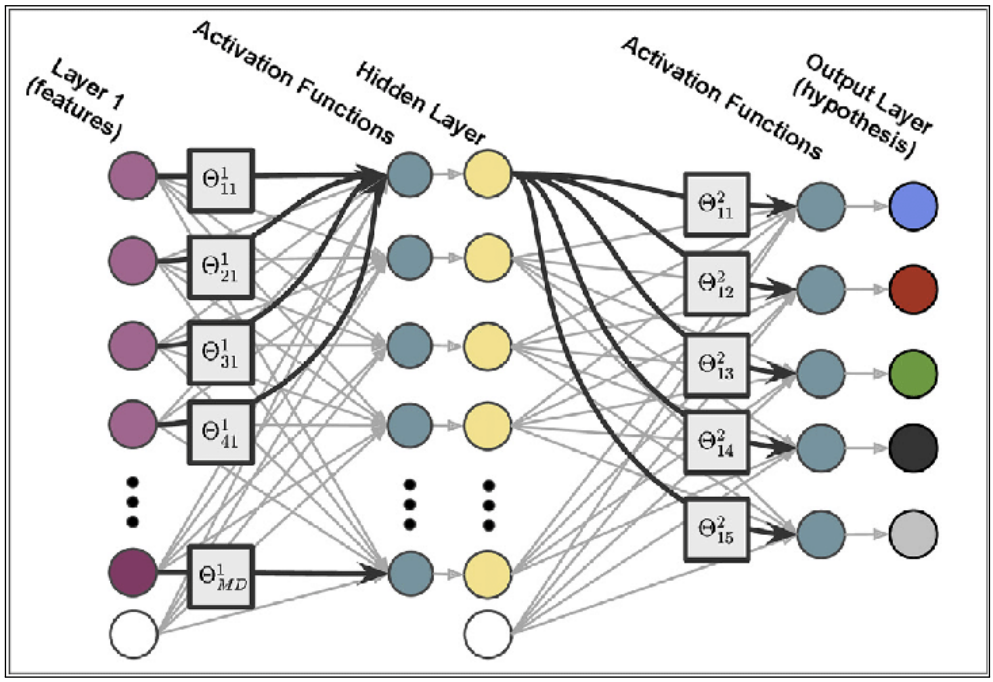

Figure 6: Multilayer Perceptron [10]

In Terms of vocabulary, neural networks that are feedforward, meaning unidirectional have a defined set of input neurons, five in our case and a defined set of output neurons. They are fully connected and depending on their amount of hidden layers (deepness) they are either called multi-level perceptrons (MLP) with at least three hidden layers or deep neural networks for higher amounts of hidden layers. The lines in those definitions are blurry, it is mostly because of convention that neural networks of NeRFs are called MLP. [11]

To wrap up this subchapter it is interesting to have a look at the general use-cases of NeRF before focusing on consumer-level applications, like in the introduction and the possibility for one single developer to eventually be able to efficiently sample 3D objects

to import into computer games.

Applications of NeRF

The following applications came up the most often, after a thorough internet search:

- Filmmaking: Creating realistic 3D environments for virtual production.
- Virtual Reality: Transforming real-world scenes into virtual experiences. [6]
- Heritage Preservation: Digitizing cultural artifacts and historical sites.
- Medical Imaging: 3D reconstructions of anatomical structures from 2D scans. [7]
- Augmented Reality: Integrating 3D models into real-world contexts.
- Architecture: Interactive 3D walkthroughs based on blueprints. [8]
- Gaming: Generating immersive 3D worlds from real-world data.
- Autonomous Driving: Generating detailed 3D maps for navigation.
- E-commerce: Providing 360-degree product visualizations.
- Entertainment: Enabling users to generate custom 3D scenes from photographs. [9]

NeRF's technological and application landscape demonstrates its wide-ranging impact across industries, but its resource-intensive nature highlights the importance of ongoing research into optimization for broader adoption.

The techniques described until now (called Vanilla NeRF) are mostly shared for every other variants. Other models can improve performance or other metrics, but to stay aligned with the focus of this paper we will only look at variants and advances that primarily improve efficiency. The most obvious efficiency gain can be obtained through hierarchical sampling.

2.1.1 Hierarchical Sampling for Efficiency

Hierarchical Volume Sampling is a method used in NeRF to make the rendering process more efficient by focusing computational resources on the most important parts of a scene. Instead of treating all points along a ray equally, this method uses a two-step

process to first identify regions likely to contain relevant information and then refine the details in those areas. The Figure 7 illustrates this concept through three stages:

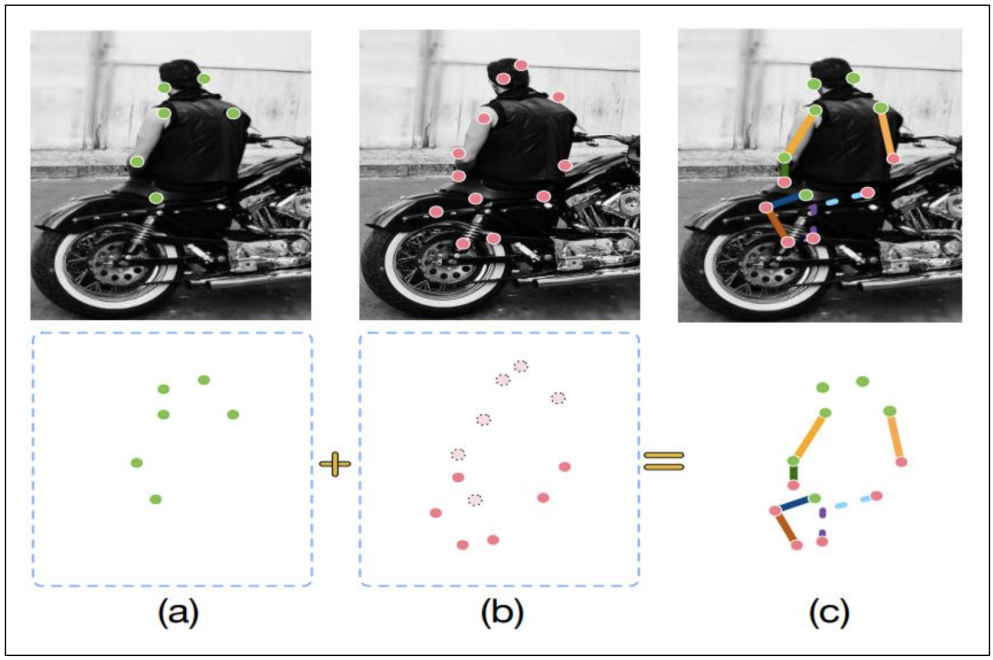

Figure 7: Hierarchical Sampling Visualized [12]

This image followed by the explanations are based on the work of Huang et al 2017. [12]

1. Coarse Sampling (Step A in Figure 7): Initially, a small number of points are sampled along the ray (shown in green). These points provide a rough estimate of where significant features, such as object surfaces or high-density areas, are located in the scene. This stage quickly identifies regions that are likely to contribute to the final image.

2. Fine Sampling (Step B in Figure 7): Using the information from the coarse sampling step, a denser set of points (shown in pink) is selected, focusing only on the regions identified as important in the coarse stage. This avoids sampling in empty or unimportant areas, which saves computational resources. The fine sampling allows the method to capture subtle details, such as edges or textures.

3. Combining Results (Step C in Figure 7): The outputs from both the coarse and fine samplings are combined to create a detailed and efficient rendering of the scene.

By concentrating effort on key areas while maintaining coverage across the ray, this hierarchical process ensures high image quality without excessive computation.

On a technical side, those features like density areas don't match well with the straightforward MLP architecture. This is why coarse sampling usually augments the neural network to a so called convolutional neural network. While the model's complexity rises it can now extract hierarchical feature in the lower layers (basic features like, edges), while deeper layers combine these into higher-level features (like, faces or objects). More importantly it can do so by utilizing less parameters, due to weight sharing.

Although this results in a more optimized NeRF that is more accurate; while utilizing less disk space, the decoding of the model remains very compute heavy, even more resource intensive as the simpler MLP model.

2.1.2 Neural Light Field

The following insights on Neural Light Fields are based on the work of Cao et al. 2023. [13]

Neural Light Fields (NLFs) represent an evolution in neural rendering techniques, offering significant efficiency advantages over NeRFs, even NeRFs that undergo hierarchical sampling. NeRFs have demonstrated impressive results in synthesizing novel views of a scene by modeling the 3D environment through a volumetric rendering process. However, the method's reliance on sampling a large number of data points along each ray and performing computationally intensive integration operations results in slow inference speeds. This inefficiency becomes particularly problematic in real-time applications or when deployed on resource-constrained devices such as mobile phones for real-time inference.

Slow NeRF Inference

NeRF is slow at inference because of the way it processes 3D scenes to create an image. For every pixel in the image there is one ray with multiple data points along the ray. At

each of these points, the model predicts two things: how dense the point is (which tells if it's part of an object or empty space) and the color of light coming from that point in the direction of the camera. These predictions are made by a neural network, which must be run separately for every sampled point. Since each ray needs many points to be sampled, and there are thousands or even millions of rays in an image, this results in a huge number of calculations. This shows how reducing the number of datapoints per ray can have big performance gains. With hierarchical sampling the removed data-points are computed to be insignificant, thus gaining in performance with minimal visual impact.

After predicting the density and color for all the points along a ray, NeRF combines these results to figure out the final color of the pixel. This step involves a lot of math, as the model blends the contributions from each point based on how much light they let through. To make things even more complex, NeRF also accounts for the direction of the camera when predicting colors, so the calculations change depending on the viewing angle. This adds more steps to the process for each point.

Because NeRF relies on a neural network to calculate these properties for every point in the scene, and because it needs to sample so many points to create one image, the process takes a long time. The neural network has to run millions of times for a single image, and each of these runs involves detailed computations. This approach also uses a lot of memory because it has to keep track of all the sampled points and their properties.

Fast NeLF Inference

In simpler models like NeLF, the process is much faster because the model skips these detailed steps. Instead of sampling lots of points along each ray, NeLF predicts the color of the entire ray in one single go. This avoids the need for all the extra calculations and makes it much better suited for real-time rendering. NeRF's detailed method allows for very high-quality results, but it's this complexity that makes it so slow during inference.

NLFs address the performance issues by shifting the focus from modeling volumetric density to directly mapping a ray to its corresponding pixel color in the image. Instead

of sampling multiple points along a ray (like in Figure 2 and accumulating their contributions, NLF processes each ray with a single forward pass through the neural network. This simplification substantially speeds up the rendering process and at the same time admittedly the image quality regresses notably. Luckily there is a remedy.

Overcoming the missing density information

To overcome these training difficulties, methods like Radiance-to-Light Field R2L employ a distillation process where a pre-trained NeRF acts as a "teacher" model. The NeRF generates pseudo-labels for a large dataset of rays, including their origins, directions, and corresponding colors. These labels are then used to train the NeLF, effectively transferring the knowledge from the volumetric model to the light field model. This process is visualized in the Figure 8 down below.

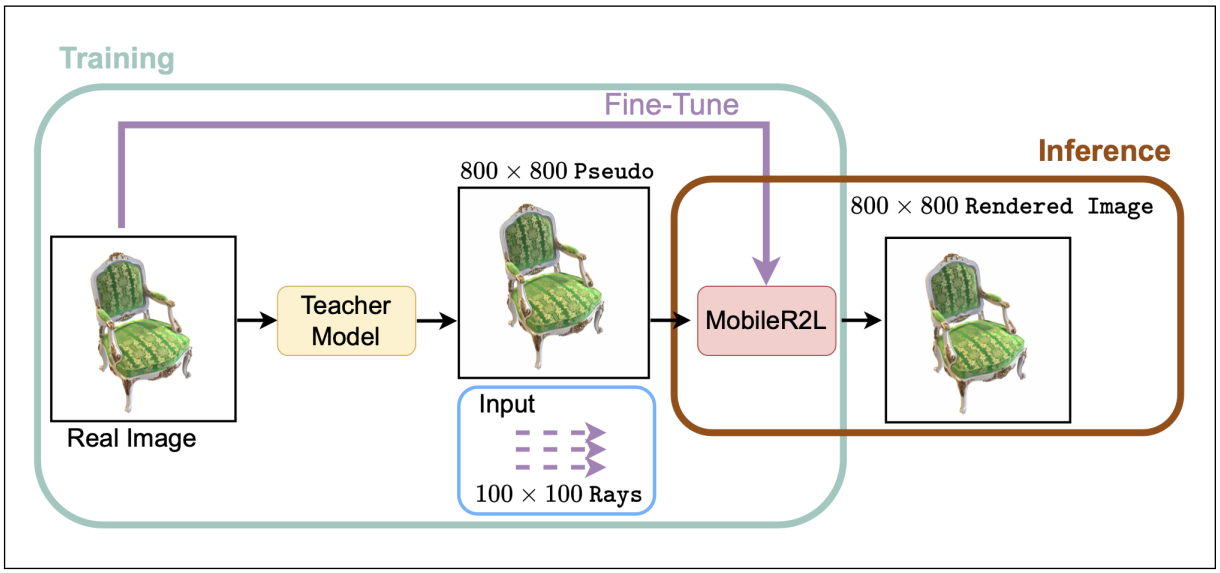

Figure 8: Inference with NLF [12]

The following end goal in simple terms: A NeRF trained on a dataset with only 100 images will model the 3D structure with the best possible visual fidelity. A NeLF based on the same 100 images will have far superior inference performance but also lack in visuals. This is why the NeLF won't be trained on the same dataset. A good balance for

a 100 image NeRF is to let it generate 5000 images. Those are the one's then used for the NeLF. Interestingly the NeLF keeps its large performance advantage while matching in visual fidelity. Despite these advances, early implementations of NeLF, such as R2L, still suffered from high computational costs due to large network sizes, making them unsuitable for mobile deployment.

Modern adaptations, as highlighted in the referenced paper, further refine the concept by using a lightweight network architecture better suited for mobile hardware. For example, the paper introduces MobileR2L, which combines convolutional layers and super-resolution modules to reduce memory and computational demands (with similar concepts to the hierarchical sampling for efficiency on NeRFs). These innovations allow NLF-based systems to achieve real-time rendering speeds on mobile devices without sacrificing image quality. By up sampling low-resolution light field volumes to high-resolution images, MobileR2L efficiently balances speed, memory usage, and visual fidelity.

The key take-away is that NLF's direct ray-to-color mapping fundamentally improves rendering efficiency over NeRF by avoiding computationally expensive volumetric operations. While it introduces challenges in training and model design, innovations like distillation-based learning and hardware-friendly architectures make NLF an attractive choice for scenarios requiring fast and efficient neural rendering, especially on constrained platforms such as mobile devices.

2.2 Hardware Requirements for Inference

There is a general understanding in the field of computer science that machine learning (ML) models based on artificial neural networks (not unlike the NeRF architecture) necessitate, so called "specialized Hardware" instead of a traditional CPU. While it is absolutely possible to run the inference of a NeRF or NLF on a general-purpose CPU, the speed and efficiency, on the other hand are lackluster. This chapter dives into the topic of what calculations neural networks depend on and how the semiconductor industry came

up with new innovating architectures that tackle those computations more efficiently. To provide a complete understanding on generational improvements that are at times due silicon improvements, the first subchapter starts the introduction to ML Hardware with Moore's Law.

2.2.1 Types of Silicon and Process Technologies

Before starting to look at the different architectures, there is a baseline in the chip industry responsible for efficiency improvements in computing called Moore's Law. As depicted in the Figure 9 below, it is an observation by Gordon Moore that the number of transistors on computer chips doubles approximately every two years.

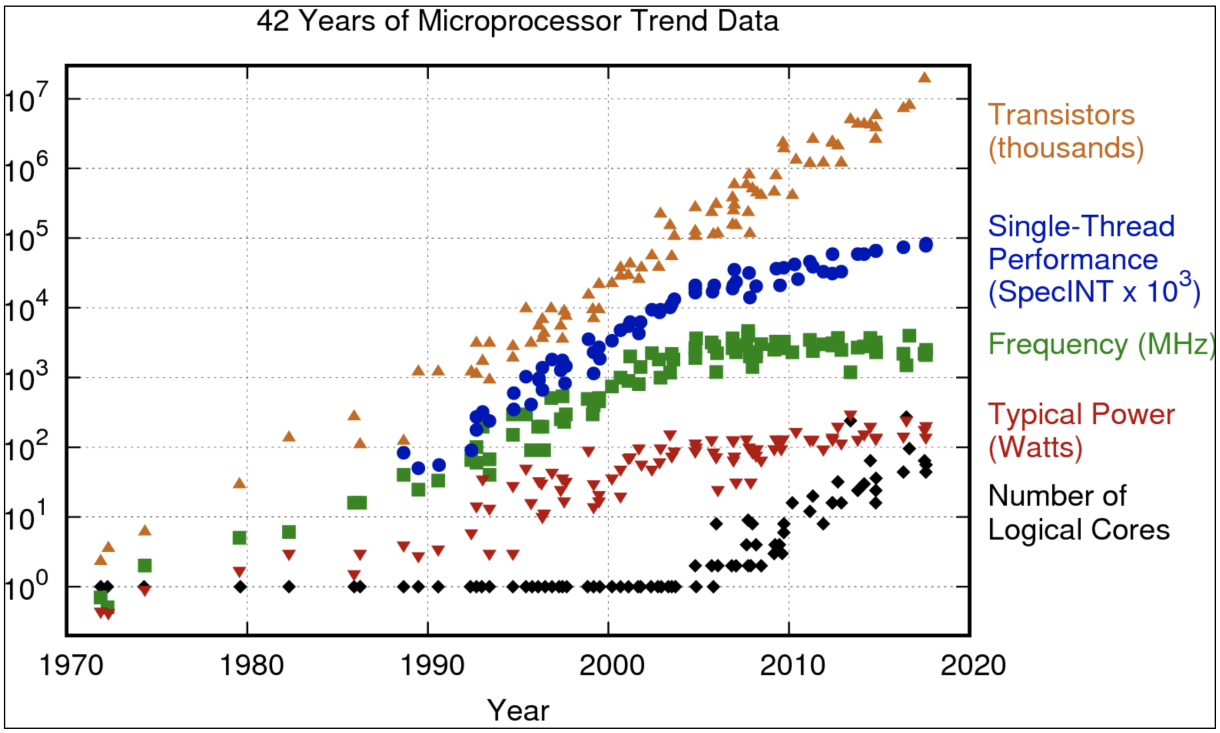

Figure 9: Moore's Law Visualized with Empirical Data [14]

Moore's Law isn't a performance indicator. It is the number of transistors (orange line) doubling every two years that is the key takeaway from this observation. This in turn is due to the technical advancements at making the building blocks of the chips smaller and thus more efficient, allowing to increase the amount of transistors per chip.

Performance metrics on the other hand, can only be evaluated when looking at actual chip implementations of those transistors. Here architectural advances have a far larger impact on performance gains.

The work by Sevilla et al. (2022) gives a brilliant overview of performance improvements through time and differentiates between three eras. [15] Before 2010 compute grew in line with Moore's law, doubling roughly every 20 months. Since the advent of deep neural networks in the early 2010s, the scaling of training compute has accelerated, doubling approximately every six months. In late 2015, a new trend emerged as firms developed large-scale ML models with 10 to 100-fold larger requirements in training compute. Here performance improvements in Hardware are estimated at a doubling every 3.4 month. [16] Based on these observations Sevilla splits the history of compute in ML into three eras: the Pre Deep Learning Era, the Deep Learning Era and the Large-Scale Era.

The pre deep learning Era evolved primarily around CPU architecture, defined by the architectures inability to outpace Moore's Law.

Limitations of the CPU

When analyzing a model like a feedforward artificial neural network (the one used for NeRFs), it becomes clear which steps demand significant computational resources. First, the data for every neuron and every weight must be loaded into random-access memory (RAM) for efficient processing. With modern models often containing billions of parameters, this scalability poses challenges. Frequent data transfers between RAM and disk lead to severe performance bottlenecks, which must be avoided to maintain efficiency.

At a higher level of the memory hierarchy, the computation of neuron values and weights requires additional resources. This process involves loading data into the chip's cache system, ensuring it is readily available for the algorithmic logic unit (ALU) to perform the required computations. The fundamental operation here is called

"multiply-accumulate" (MAC), which involves multiplying two numbers and adding the result to a running total, as depicted in Figure 10.

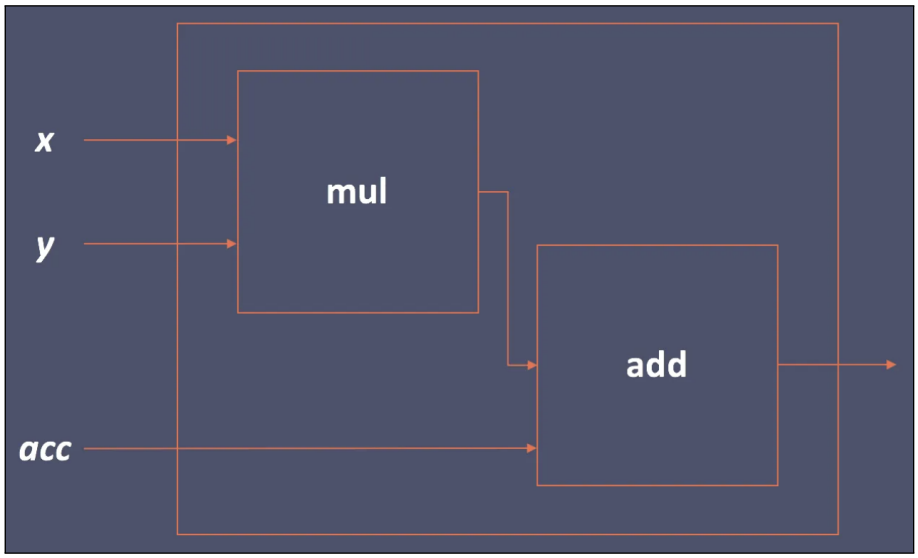

Figure 10: Multiply Accumulate [17]

MAC is central to ML workloads, underpinning critical processes such as matrix-vector multiplication, and is used extensively in both the training and inference stages of neural networks. For perspective, even a small model with 4.2 million parameters requires approximately 569 million MAC operations. [18]

However, CPUs are not optimized for the parallelism required in ML tasks. With limited cores and constrained memory bandwidth, CPUs struggle to handle workloads involving repetitive, high-volume tasks like MAC operations. This challenge is exacerbated by the von Neumann architecture, where the separation of memory and logic circuits creates a bottleneck. Each result from the ALU must be written back to system memory, adding further delays. While cache hierarchies are designed to keep frequently accessed data close to the processor and preemptively load needed information, they are inefficient for ML computations where MAC variables are typically used only once. This results in low arithmetic intensity, where memory access times, rather than computation speeds, become the primary limiting factor.

Moreover, this inefficiency is reflected in power consumption. Data transfers for a

single MAC operation can consume up to 400 times more energy than the computation itself. This demonstrates how the mismatch between conventional CPU architectures and ML workloads leads to inefficiencies in both speed and energy usage, highlighting the need for hardware specifically designed to handle such operations. [18]

The following subchapter focusses on architectural changes during the deep learning and large-scale Era that made for the biggest performance improvements. Machine Learning Accelerators are in essence just GPU's (graphics processing unit) that have come a long way.

The following chapter delves into its inner working by leaning on the work of Matthews, Newhall, and Webb [19] in the chapter "Hardware Accelerators". [20]

2.2.2 Overview on Machine Learning Accelerators

GPUs, originally designed for processing 3D graphics, possess a far greater number of arithmetic logic units (ALUs) than CPUs and are equipped with high-speed memory interfaces. These characteristics make GPUs inherently more efficient and faster for machine learning tasks, as they can process significantly more multiply-accumulate (MAC) operations simultaneously.

A GPU is composed of multiple Streaming Multiprocessors (SMs), which are further subdivided into numerous Streaming Processors (SPs). The SM acts as a group of processing units capable of executing parallel computations, while each SP within an SM is responsible for specific tasks, such as performing arithmetic operations or accessing memory. The structure is designed to maximize parallelism by dividing workloads across its many cores.

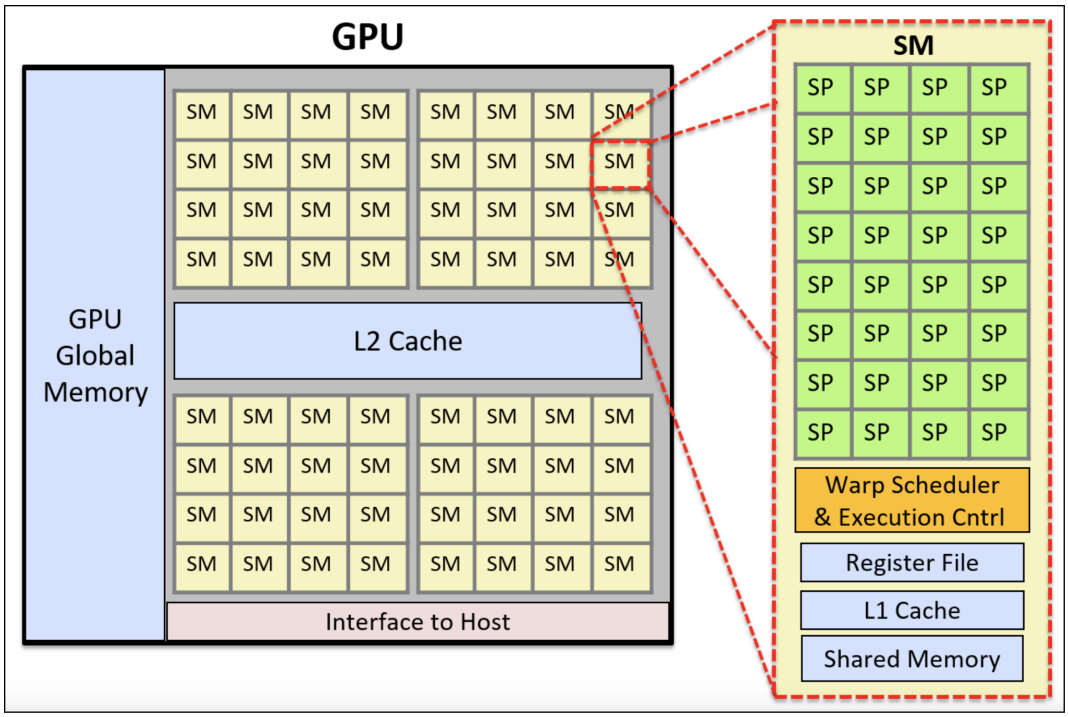

Figure 11: Inside the GPU Cores [20]

The diagram (referenced in Figure 11 above) illustrates the architecture of a typical GPU. This midrange GPU, with 2048 cores, far exceeds the core count of even the most advanced CPUs. The architecture ensures that these cores remain continuously active, optimizing their utilization. Instructions from machine learning workloads are divided into parallel threads, which are then executed by individual SPs. These threads are grouped into blocks, and each block is assigned to an SM for execution. Programmers can specify the number of threads per block and the number of blocks per grid, ideally matching the number of SPs available in the GPU's architecture.

When an SM executes a block, the threads within the block are further divided into warps, which are groups of 32 threads. Each warp is processed by an SP using "Single Instruction, Multiple Data" (SIMD) execution, where the same instruction is applied to different data across the threads. This approach avoids duplicating instruction code for every core, reducing the bandwidth load and improving efficiency.

As SPs execute these threads, they access data from the GPU's global memory,

which serves as the primary memory. However, global memory access is relatively slow compared to registers or shared memory. To mitigate this bottleneck, GPUs employ optimization techniques, such as caching frequently accessed data and utilizing shared memory to reduce the frequency of global memory access. The performance of the GPU's SPs depends on the arithmetic intensity of the workload, which measures the number of operations per data access. In an optimized system, the workload is large enough to fully utilize the SPs' computational and memory pipelines.

Optimizations are also critical. Without sufficient workload saturation, the GPU's cores risk being underutilized. In such cases, performance becomes constrained by memory bandwidth or latency rather than computational throughput. Properly structured workloads ensure that GPUs achieve their full potential, delivering the high levels of parallelism and efficiency they are designed for.

2.2.3 Types of Edge-Device Hardware for Local Inference

Edge-Device are usually battery powered and therefore constraint in their consumption. Besides GPUs there are two notable architectures that aim at repeating the improvements GPUs have brought over CPUs. Those chips are called NPUs and TPUs. [21]

Both NPUs and TPUs improve throughput significantly by tailoring their architectures to operate on 8-bit integer (INT8) precision, which is sufficient for most inference tasks. Reducing precision from traditional 32-bit or 16-bit floating-point to 8-bit integers result in minimal accuracy degradation during inference, while drastically increasing computational efficiency and reducing power consumption. This design choice makes both types of processors ideal for high-throughput applications.

NPUs

NPUs are specialized processors designed to optimize the computation of neural network operations, particularly for inference in mobile and embedded systems. A key advantage of NPUs lies in their efficient handling of synaptic weights, which represent the strength

of connections between neurons in a neural network. Unlike GPUs, NPUs are specifically designed to minimize the inefficiencies associated with weight storage and computation.

They integrate dedicated weight storage circuits and use optimized memory hierarchies, such as on-chip static RAM (SRAM) or tightly coupled weight buffers, to store synaptic weights close to the processing cores. This reduces the need for frequent data transfers between system memory (dynamic RAM or DRAM) and the compute units, a common bottleneck in GPUs. Additionally, NPUs employ weight quantization, reducing the precision of weights from floating-point formats to 8-bit integers, which reduces memory usage and accelerates computation without significantly affecting accuracy.

NPUs also excel at handling sparse neural networks by skipping computations involving zero or near-zero weights. This sparsity-aware design avoids wasting resources on irrelevant operations, something that is difficult to achieve with GPUs, which process data in fixed blocks. Furthermore, NPUs parallelize weight operations at the hardware level, enabling highly efficient matrix-vector multiplications and other neural network computations.

TPUs

TPUs, Google's proprietary accelerator, share the efficiency improvements of INT8 precision while taking a different architectural approach. They are designed around systolic arrays, which consist of a grid of processing nodes that perform partial computations and pass the intermediate results to the next nodes. This architecture optimizes tensor operations, such as the afore mentioned matrix multiplications.

The systolic array design allows for continuous data flow between processing nodes, reducing memory access and maximizing utilization, a common trend by now.

It is worth noting that by now modern GPUs, like the H100 from Nvidia have integrated hardware accelerated matrix multiplications in their SP-cores (Figure 11). Those GPUs that are now also specialized in tensor operations can be regarded

as Tensor-GPUs, blurring the line between GPU and TPU. [22] Though, modern GPUs haven't adopted the processing node grid design that make the TPU an application-specific integrated circuit (ASIC), and they have therefore kept their general purpose nature. Their broader applicability, commercial success and non-proprietary status have made them the default hardware accelerator. Solely the newer NPU accelerators are competing in the inference market, due to their superior power efficiency.

The following chapter takes a more pragmatic approach and shows the current state on using NeRFs from a consumer standpoint, utilizing the most prevalent consumer hardware on the market.

This experience is necessary to quantize the pace of improvement over the last years in this case comparing to a time before the NeRF model from 2020. This in turn will be used to give the audience an idea on the possible evolutionary path this technology is on.

3 Methodology

The following chapter is about achieving a real-time neural light field (NeLF) rendering model for mobile devices. With the guide provided by the Github page [23] based on the work [13] of Cao et al. 2023 the goal of this chapter is to recreate the pipeline progresses through several well-defined stages.

Starting with a collection of input images and culminating in an efficient, trained model capable of high-speed inference.

Below is a technical breakdown of the process as described in the provided paper:

3.1 Training of a Real-Time NeLF Rendering Model

The process begins with setting up the system and preparing the necessary tools. The project leverages a teacher-student model architecture, where the teacher model (a more computationally intensive neural rendering framework) generates pseudo data to train the lightweight student model, MobileR2L. The pipeline comprises multiple key stages, from dataset preparation to real-time inference on mobile devices.

Pretraining

The pretraining phase consists of obtaining a set of preprocessed images. First, a set of multi-view images is captured using a standard camera setup. These images serve as input for learning the scene's geometry and appearance. For instance, in a practical application like shoe rendering for virtual try-on, around 100 images are captured with a mobile device. These images are preprocessed by segmenting the foreground (like the shoe) from the background to ensure clean input data for training.

Since it is beyond the scope of this paper to creating a new set of 100 pictures and preprocess them, the next steps will be using the test data set provided by the repository.

Teacher Model Training

A Neural Radiance Field (NeRF) model is trained on the collected images. The method used here is precisely the one explained in the theoretical background's chapter.

The NeRF represents the scene implicitly using a multi-layer perceptron network, which maps the 5D inputs, the 3D spatial coordinates (x, y, z) and 2D view direction to 1D density and 3D RGB radiance. The model is then trained to predict pixel colors by sampling and compositing multiple points along camera rays.

As this is the model that will be used to generate the synthetic images needed to create the NeLF it is called a teacher model. In this specific project the NeRF is actually replaced with a more efficient version called NGP (Neural Graphics Primitives). It replaces NeRF's dense volumetric sampling with hash encoding for spatial data. This allows it to represent the scene efficiently with fewer parameters and faster convergence.

This step takes one hour on a Nvidia V100 GPU that computes at a speed of 130 TFLOPS (130*10 $\times$ 10^{12} FP16 operations per seconds) [24] On my local machine the hardware accelerator is a NVIDIA GTX1050ti with a compute capacity of 2.1 TFLOPS (almost 64x slower) [25], resulting in compute duration of almost three days.

Generating Pseudo Data with the Teacher Model

Once the teacher model is trained, it is used to generate pseudo data. The teacher model synthesizes high-resolution outputs by rendering scenes using camera poses defined in the dataset. These outputs serve as a pseudo dataset for training the student model. Depending on disk storage, between 2,000 to 10,000 pseudo images can be generated. For MobileR2L, this number is typically set to 5,000 to balance quality and efficiency.

This step takes roughly an additional hour on the V100 and once again close to 3 days on the GTX1050ti.

Training the Student Model (MobileR2L)

The pseudo dataset is used to finally train MobileR2L. This is the lightweight model that replaces the computationally intensive volumetric rendering of the NeRF with a direct ray-to-color mapping. The model employs a convolutional backbone, optimized for mobile hardware, which eliminates the inefficiencies of fully connected layers. Training the student model requires GPUs and may take one to two days, depending on available resources. The model is fine-tuned to produce high-resolution images while maintaining low latency. On the GTX1050ti this process would take up to 100 days.

3.2 NeLF Inference on Local Hardware

Once training is complete, the model is exported in the ONNX format for compatibility purposes. The exported ONNX model can be deployed to mobile devices for real-time rendering. During inference, the MobileR2L model efficiently computes the pixel colors of a 3D scene from input camera positions using that single forward pass. Low-resolution outputs are generated first and upscaled to high resolution using super-resolution modules integrated into the framework. This design achieves real-time rendering speeds, such as 18 milliseconds per frame on an iPhone 13 for a 1008×756 resolution image.

By leveraging a distillation process, optimized convolutional architecture, and super-resolution techniques, MobileR2L balances high image quality with computational efficiency, making it suitable for real-time applications on mobile devices. This pipeline represents the complete approach to deploying neural rendering in constrained environments while maintaining high visual fidelity.

4 Results

This chapter focusses on the results of the NeLF model, where the time component is predominantly used as a benchmark for positive or negative outcome. The pipeline for training a NeLF model, ready to be used for real-time inferencing consists of five different steps that have the following lead times.

1. Pretraining consists of gathering the initial set of images and segmenting the foreground from the background. This step requires some quality control but is a relatively simple operation that runs in close to real time on current hardware

2. Teacher model training builds the NeRF or in this case the better suited NGP and takes up from one to 64 hours of time to complete

3. Pseudo data generation runs inferences on the NGP to generate 5000 images covering every possible blind spot of the image and also takes up from one to 64 hours

4. Student model training utilizes the new data set of 5000 images and builds the NeLF while taking from one to two days all the way to 100 days.

5. NeLF inference then again provides the best possible results and runs the model with a refresh rate of almost 60FPS (18.3ms).

From these results it is clear that NeLFs have good enough performance for everyday uses. The test shows functioning real-time performance with an edge device delivering 15TOPS at 5 watts. [26]

On the other hand, the points 2-5 make it clear that consumer adoption to create the NeLF models locally is something only for the future A system running the V100 will take almost two days while a more modest one running the popular GTX1050ti will take approximately 105 days.

The next subchapter takes those lead times to provide figures on power consumptions.

4.1 Performance and Power Consumptions

The V100, according to the datasheet consumes 400 watts at full load, whereas the GTX1050ti runs at 60 Watt. Their power consumptions can be evaluated with the simple formula of Energy = power × time.

The V100 therefore consumes 19.2kW (400w x 48h) and the GTX1050ti 151kW (60W x 2520h). This amount of power is not negligible in fact at a price of 115€/MWh [27] the electricity bill would be at 2,2€ (19.2 / 1000 x 115) or 17,3€, just for a single NeLF model.

This is why the previously mentioned computing gains of the Deep Learning and Large-Scale Era are important. Although those values look ridiculously expensive now, they will become irrelevant very quickly.

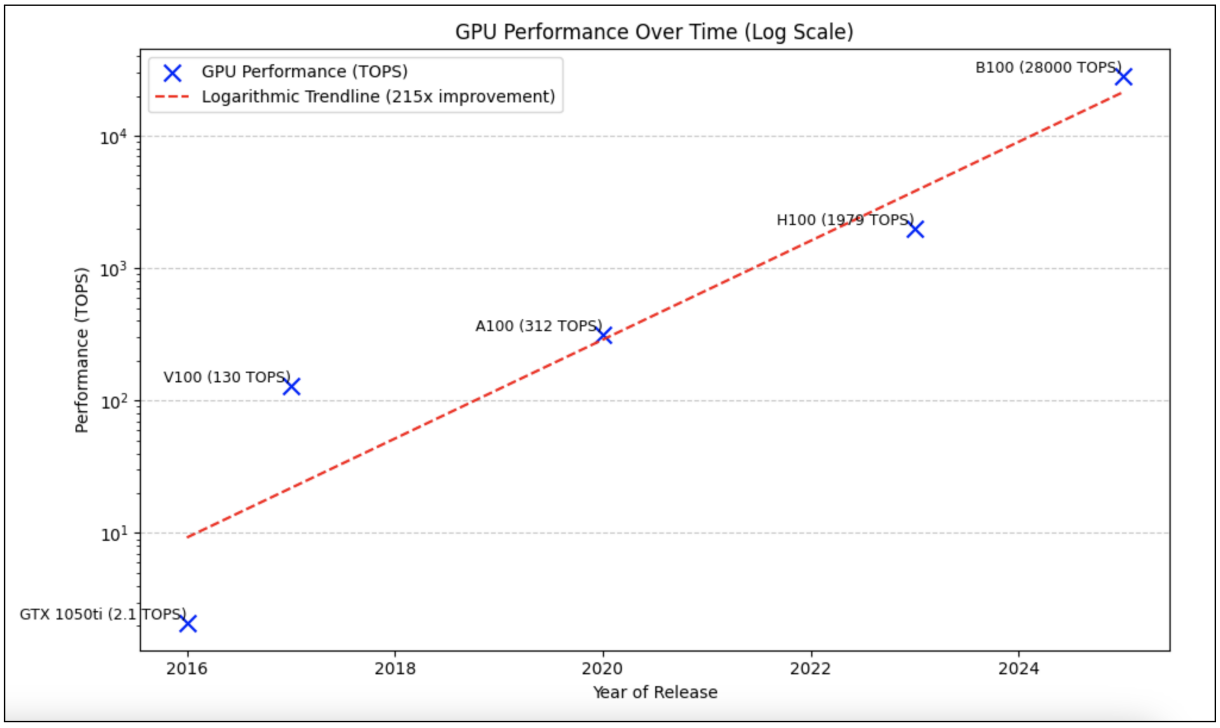

Figure 12: Trendline of NVIDIA Compute Improvements

According to the trendline in Figure 12 generated by accessing publicly available data on generational NVIDIA GPU improvements, there is a computing increase from

the V100 to the upcoming B100 of the factor 215x. At the same power consumption this would bring down the cumulative training time from 48h down to 13 minutes and the previous energy costs down from 2,2€ to 1ct!

When looking again at the pipeline for training a NeLF model we can make the following conclusion. Point 5 (local inference) is already possible and will only get faster and allow for higher resolutions in time Point 2-4 (training) are barely possible right now but will most definitely be accessible to anyone in the coming years. This leaves point one, the gathering of the initial image dataset as the real limiting factor.

4.2 Limitations and Outlook

Despite the significant advancements in Neural Radiance Field (NeRF) technology and future hardware, a significant limitation remains. That is the dependence on extensive and precisely annotated datasets. The requirement for at least 100 high-resolution images with accurate positional data presents a barrier for users lacking specialized equipment or expertise. This issue is worsened when attempting to capture dynamic scenes or environments with variable lighting and occlusion. Developing methods to reduce the data volume requirement or utilizing synthetic augmentation could alleviate these constraints.

Looking forward, possible research could be on a hybrid approach combining the models with physics-based rendering techniques. For instance, leveraging partial scene priors or domain-specific knowledge could enable NeRFs to add missing details and enhance realism, even in under-sampled regions.

Additionally, there could be advancements using ML models that could facilitate automatic generation of positional data, further democratizing the technology.

In summary, while NeRF technology is on a clear upward trajectory, realizing its full potential will primarily depend on overcoming barriers in the current software stack.

5 Conclusion

This paper has explored the capabilities and challenges of applying Neural Radiance Fields (NeRF) for 3D scene reconstruction, with a focus on making this technology accessible for consumer-level applications. It began by discussing the theoretical background of NeRFs and their potential to generate high-quality 3D models from 2D images, addressing both the advantages and limitations of this approach. While NeRFs provide a significant improvement over traditional 3D reconstruction techniques like photogrammetry, especially when accurate textures are needed, they are still limited by high computational requirements and long training times, especially on consumer hardware.

The methodology described in this paper illustrates the process of implementing a real-time Neural Light Field (NeLF) model using a teacher-student architecture to optimize inference speed. By leveraging models such as NGP (Neural Graphics Primitives) and MobileR2L, it demonstrated that it is possible to achieve real-time performance on mobile devices, with inference speeds suitable for consumer use. However, as the results showed, the training process is still a major bottleneck, requiring significant computational resources that are not yet feasible for most consumer users.

With the V100 GPU, training times are reduced, but the GTX1050ti still results in training durations that can stretch into weeks. However, these results are put in the context of generational advancements which show, that the long lead times will soon be shortened to a few minutes and thus make this problem a thing of the past.

In conclusion, while NeRF and NeLF technologies show great promise for 3D scene reconstruction, there are still significant software hurdles to overcome before they can be fully adopted at the consumer level. Solving these obstacles will be essential in bringing these advanced techniques to a wider audience and opening new possibilities for fields like gaming digital content creation.

References

[1] Ben Mildenhall et al. "NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis". In: *CoRR* abs/2003.08934 (2020). arXiv: 2003.08934. URL: https://arxiv.org/abs/2003.08934.

[2] *Hugging Face - Neural Radiance Fields (NeRFs)*. Accessed: October 29, 2024. 2024. URL: https://huggingface.co/learn/computer-vision-course/unit8/nerf.

[3] [Online; accessed November 8, 2024]. URL: https://cdn.artec3d.com/styles/1468_webp/s3/content-hub-images/articles/content/photogrammetry_2.png.webp?VersionId=dMTSjlwsuo_S3VlSfUrfVyaRkmeg0rc8&itok=qbax9_-p.

[4] [Online; accessed November 8, 2024]. URL: https://cdn.artec3d.com/styles/1468_webp/s3/content-hub-images/articles/content/photogrammetry_4_new.png.webp?VersionId=6KDU0Eui0X6q.SPV8lI8UazqOCGQxk7k&itok=7Xww-EU7.

[5] *What is Photogrammitry*. Accessed: November 8, 2024. 2024. URL: https://www.artec3d.com/learning-center/what-is-photogrammetry.

[6] *What is a Neural Radiance Field (NeRF)*. Accessed: November 10, 2024. 2023. URL: https://www.techtarget.com/searchenterpriseai/definition/neural-radiance-fields-NeRF.

[7] *Neural Radiance Fields (NeRF) A Breakthrough in 3D Reconstruction*. Accessed: November 10, 2024. 2024. URL: https://www.tooli.qa/insights/neural-radiance-fields-nerf-a-breakthrough-in-3d-reconstruction.

[8] *What is NeRF - Neural Radiance Fields Explained*. Accessed: November 10, 2024. 2023. URL: https://aws.amazon.com/what-is/neural-radiance-fields/#:~:text=A%20neural%20radiance%20field%20(NeRF,set%20of%20two%2Ddimensional%20images..

[9] *Neural Radiance Fields (NeRFs) - Hugging Face Community Computer Vision Course*. Accessed: November 10, 2024. 2024. URL: https://huggingface.co/learn/computer-vision-course/unit8/nerf.

[10] Alex Witsil and Jeffrey Johnson. "Volcano video data characterized and classified using computer vision and machine learning algorithms". In: *Geoscience Frontiers* 11 (Feb. 2020). DOI: 10.1016/j.gsf.2020.01.016.

[11] *MLP-vs-DNN*. Accessed: November 8, 2024. 2024. URL: https://www.baeldung.com/cs/mlp-vs-dnn.

[12] Shaoli Huang, Mingming Gong, and Dacheng Tao. "A Coarse-Fine Network for Keypoint Localization". In: *2017 IEEE International Conference on Computer Vision (ICCV)*. 2017, pp. 3047–3056. DOI: 10.1109/ICCV.2017.329.

[13] Junli Cao et al. *Real-Time Neural Light Field on Mobile Devices*. 2023. arXiv: 2212.08057 [cs.CV]. URL: https://arxiv.org/abs/2212.08057.

[14] [Online; accessed November 16, 2024]. URL: https://www.karlrupp.net/wp-content/uploads/2018/02/42-years-processor-trend.png.

[15] Jaime Sevilla et al. "Compute Trends Across Three Eras of Machine Learning". In: *2022 International Joint Conference on Neural Networks (IJCNN)*. IEEE, July 2022, pp. 1–8. DOI: 10.1109/ijcnn55064.2022.9891914. URL: http://dx.doi.org/10.1109/IJCNN55064.2022.9891914.

[16] *ai-and-scaling-the-compute-for-the-new-moores-law/*. Accessed: November 21, 2024. 2023. URL: https://blog.mi.hdm-stuttgart.de/index.php/2023/03/03/ai-and-scaling-the-compute-for-the-new-moores-law/.

[17] [Online; accessed November 17, 2024]. URL: https://medium.com/analytics-vidhya/not-all-tops-are-created-equal-e1911ffb4a82.

[18] *multiply-accumulate*. Accessed: November 16, 2024. 2020. URL: https://medium.com/analytics-vidhya/not-all-tops-are-created-equal-e1911ffb4a82.

[19] S.J. Matthews, T. Newhall, and K.C. Webb. *Dive Into Systems: A Gentle Introduction to Computer Systems*. No Starch Press, 2022. ISBN: 9781718501362. URL: https://books.google.de/books?id=7RVOEAAAQBAJ.

[20] *Features of the GPU*. Accessed: November 16, 2024. 2020. URL: https://diveintosystems.org/book/C15-Parallel/gpu.html.

[21] *Gpu-vs-Tpu-vs-Npu*. Accessed: November 16, 2024. 2023. URL: https://www.backblaze.com/blog/ai-101-gpu-vs-tpu-vs-npu/.

[22] *Whitepaper NVIDIA H100 Tensor Core GPU Architecture*. Accessed: November 18, 2024. 2022. URL: https://resources.nvidia.com/en-us-tensor-core.

[23] *Github of MobileR2L*. Accessed: November 20, 2024. 2023. URL: https://github.com/snap-research/MobileR2L.

[24] *NVIDIA V100 - Datasheet*. Accessed: November 15, 2024. 2020. URL: https://images.nvidia.com/content/technologies/volta/pdf/volta-v100-datasheet-update-us-1165301-r5.pdf.

[25] *NVIDIA GTX1050ti - Datasheet*. Accessed: November 15, 2024. 2019. URL: https://www.techpowerup.com/gpu-specs/geforce-gtx-1050-ti.c2885.

[26] *Apple A15Bionic - Specifications*. Accessed: November 17, 2024. 2021. URL: https://en.wikipedia.org/wiki/Apple_A15.

[27] *Electricity Prices in Germany*. Accessed: November 17, 2024. 2024. URL: https://www.statista.com/statistics/1267541/germany-monthly-wholesale-electricity-price/.