

\mathcal{N} -BVH: Neural ray queries with bounding volume hierarchies

Philippe Weier
weier@cg.uni-saarland.de
Saarland University
Germany

Alexander Rath
rath@cg.uni-saarland.de
Saarland University
Germany

Élie Michel
emichel@adobe.com
Adobe
UK

Iliyan Georgiev
igeorgiev@adobe.com
Adobe
UK

Philipp Slusallek
philipp.slusallek@dfki.de
Saarland University, DFKI
Germany

Tamy Boubekeur
boubek@adobe.com
Adobe
France



Figure 1: \mathcal{N} -BVH provides a compressed representation for ray queries against complex 3D assets, integrating seamlessly into standard ray-tracing pipelines. Here, the rendered image (left) combines responses from our neural model (top right) with classical BVH/triangle ray queries (middle right), providing a faithful approximation to the original scene. Our model achieves a compression rate of 42 \times for the subset of geometry it represents, and 13 \times over the entire scene.

ABSTRACT

Neural representations have shown spectacular ability to compress complex signals in a fraction of the raw data size. In 3D computer graphics, the bulk of a scene’s memory usage is due to polygons and textures, making them ideal candidates for neural compression. Here, the main challenge lies in finding good trade-offs between efficient compression and cheap inference while minimizing training time. In the context of rendering, we adopt a ray-centric approach to this problem and devise \mathcal{N} -BVH, a neural compression architecture designed to answer arbitrary ray queries in 3D. Our compact model

is learned from the input geometry and substituted for it whenever a ray intersection is queried by a path-tracing engine. While prior neural compression methods have focused on point queries, ours proposes neural *ray queries* that integrate seamlessly into standard ray-tracing pipelines. At the core of our method, we employ an adaptive BVH-driven probing scheme to optimize the parameters of a multi-resolution hash grid, focusing its neural capacity on the sparse 3D occupancy swept by the original surfaces. As a result, our \mathcal{N} -BVH can serve accurate ray queries from a representation that is more than an order of magnitude more compact, providing faithful approximations of visibility, depth, and appearance attributes. The flexibility of our method allows us to combine and overlap neural and non-neural entities within the same 3D scene and extends to appearance level of detail.

SIGGRAPH Conference Papers '24, July 27-August 1, 2024, Denver, CO, USA

© 2024 Copyright held by the owner/author(s).

This is the author’s version of the work. It is posted here for your personal use. Not for redistribution. The definitive Version of Record was published in *Special Interest Group on Computer Graphics and Interactive Techniques Conference Conference Papers '24 (SIGGRAPH Conference Papers '24)*, July 27-August 1, 2024, Denver, CO, USA, <https://doi.org/10.1145/3641519.3657464>.

CCS CONCEPTS

• **Computing methodologies** → **Rendering**.

KEYWORDS

ray tracing, neural models, acceleration structures, bounding volume hierarchies, level of detail

ACM Reference Format:

Philippe Weier, Alexander Rath, Élie Michel, Iliyan Georgiev, Philipp Slusallek, and Tamy Boubekeur. 2024. *N*-BVH: Neural ray queries with bounding volume hierarchies. In *Special Interest Group on Computer Graphics and Interactive Techniques Conference Conference Papers '24 (SIGGRAPH Conference Papers '24)*, July 27-August 1, 2024, Denver, CO, USA. ACM, New York, NY, USA, 10 pages. <https://doi.org/10.1145/3641519.3657464>

1 INTRODUCTION

Physically based rendering engines rely on a single fundamental operator to simulate light transport in a 3D scene: ray tracing. For realistic scenes encompassing complex surface meshes (see Fig. 1), ray queries are accelerated using a hierarchical data structure that allows skipping empty space efficiently when tracing the rays in search for an intersection. Nevertheless, the memory footprint of the scene and of this structure is often challenging to manage, given the limited space available on graphics processing units (GPU). Recently, neural methods have demonstrated impressive ability to compress data, in particular spatial samplings, yet have been mostly designed to serve point queries, such as evaluating a shape represented via its *signed distance function*. In this paper, we propose a new neural representation for 3D scene models designed specifically for ray queries that blends naturally into a typical ray tracer. Our key observation is that any neural compression model can be optimized efficiently as long as it is trained on samples that live close to the signal of interest. In our case, 3D surfaces are the signal of interest and are commonly structured in a *bounding volume hierarchy* (BVH) to speed up their intersection test. We take inspiration from this very standard setup and propose to optimize a state-of-the-art neural data structure by embedding it into such a BVH; we call our new structure a *Neural BVH* or *N-BVH*.

At training time, we use the scene’s BVH as a probing machine to generate ray queries/responses training pairs only close to the surfaces for our neural model to learn. At rendering time, our *N-BVH* inherits the natural empty-space skipping behavior of a standard BVH and serves neural ray queries, preventing full, deep BVH traversal and hence storage. Our *N-BVH* can be used concurrently with a standard BVH, to overlap neural and non-neural assets in a single scene. Its training takes only a few minutes even on large scenes and its runtime response includes visibility, depth, and appearance attributes for arbitrary rays. Additionally, *N-BVH* proposes a simple level-of-detail (LoD) scheme, by refining multiple error-driven cuts in its underlying tree structure and optimizing the neural model concurrently at all their nodes.

Contributions. We introduce the following novel elements:

- A new hybrid neural data structure, *N-BVH*, which encodes signals such as depth, normal, or appearance attributes so that they can be efficiently queried by a ray, and focuses its neural capacity on the sparse subset of 3D space spanned by surfaces;
- A neural ray-intersection query mechanism (Section 4) which can serve any path tracer with faithful intersection approximations;
- A fast training scheme driven by a coarse-to-fine tree-cut optimization (Section 5.1), which automatically concentrates the training where the error is highest; this scheme provides a deep (resp. shallow) hierarchy where the geometry is hard (resp. easy) to learn and is oblivious to the actual tessellation of the input meshes, e.g., a densely subdivided plane is perceived as simple;
- The ability to jointly define multiple adaptive neural levels-of-detail (Section 5.3), specifying a target node count for each related tree cut.

Basing *N-BVH* on hash grids copes ideally with BVH empty-space skipping: typically, the vast majority of the backpropagation gradients are non-zero during training even if only a small amount of the scene’s volume is actually sampled. This sparse adaptation translates into encoded points landing close to the geometry, inducing faster and easier learning at a reduced memory cost. This allows for a very lightweight hash grid encoding—in the order of a few megabytes—where we favor BVH depth over neural capacity growth.

Application spectrum. We demonstrate the benefit of our *N-BVH* with two application scenarios: *hybrid path tracing* (Section 7) which combines neural and non-neural assets in a single pipeline; and *neural appearance prefiltering* (Section 8) which alleviates memory consumption and render times while providing equal or lower prefiltering error on large complex models appearance.

The source code of our implementation is publicly available at <https://github.com/WeiPhil/nbvh>.

2 RELATED WORK

Neural implicit representations. Neural radiance fields (NeRF) [Mildenhall et al. 2020] and their recent adaptations to interactive and real-time graphics [Müller et al. 2022] use implicit neural representations forming neural fields [Xie et al. 2022] for novel view synthesis. Neural representations’ adaptability, coupled with sparse [Müller et al. 2022] and compressed [Takikawa et al. 2022] coordinate encodings, efficiently store high-dimensional functions, representing high-frequency content in spatial and angular domains. The necessity for accelerated coordinate-based representation has mainly been explored in NeRF-related techniques where the dense sampling of a volume using ray-marching can lead to a prohibitively high number of network inferences. The (shallow) multi-layer perceptron (MLP) involved in the process is responsible for a large part of the inference cost which Hedman et al. [2021] proposes to avoid by accumulating features along rays to offload the MLP. Yang [2023] also showed that a tree-structured MLP can further improve compression fidelity. Recently, Wang et al. [2023] showed that further speed and accuracy can be gained with an adequate empty-space skipping strategy. Not only does this significantly reduce the number of network queries at runtime, but it can drastically improve the reconstructed signal as sampling is increased near the high-frequency content of the scene. Our approach is similar in spirit and relies on traditional scene acceleration data structures to provide an efficient empty-space skipping strategy.

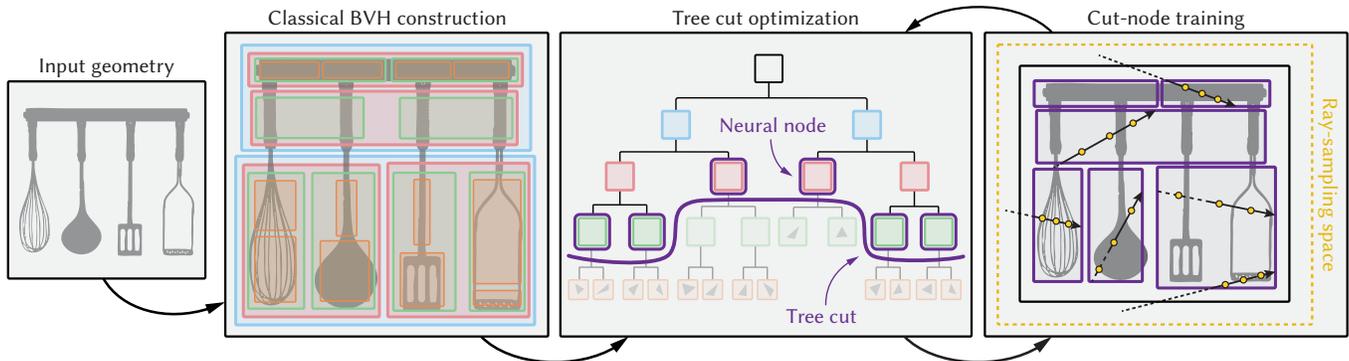


Figure 2: Our lightweight N -BVH is a shallow hierarchy whose leaf nodes simply store bounds within which we search for ray intersections by querying a neural geometry representation. The leaves (in purple outlines) represent a cut in a classical BVH over the input geometry; we optimize the cut by iteratively splitting the leaves with largest inference error. After each splitting step we train our neural model within the cut-node bounds by sampling random rays in the scene.

However, we are not restricted to NeRF applications as we aim at learning general ray-intersection queries which are at the core of many rendering applications.

Geometric simplification. Silhouette and shape-preserving decimation [Garland and Heckbert 1997; Hoppe 1996; Kobbelt et al. 1998] are widely used, often with artist supervision, in production environments. However, they can fail to accurately preserve the appearance of the original asset when high compression rates are demanded or the underlying geometry does not exhibit the geometric characteristic the algorithms are tailored to [Lindstrom 2000; Rossignac and Borrel 1993; Schaefer and Warren 2003]. Point-based [Pauly et al. 2002] or statistical simplification [Cook et al. 2007] supports less structured shapes, and hybrid volume/surface approaches [Gobbetti and Marton 2005; Loubet and Neyret 2017] have been proposed but they usually focus on prefiltering rather than reducing the memory footprint of the scene.

Neural compressed geometric reconstruction. A large body of work has shifted toward representing geometry as a signed distance field (SDF) rather than a more typical triangle representation. Park et al. [2019] were among the first to represent SDFs as deep neural networks, though were able to learn only low-frequency signals and at a high inference cost. Later, Takikawa et al. [2021] demonstrated how coordinate-based networks, implemented as an octree feature volume, allow much higher frequency SDFs to be reconstructed while also providing continuous levels of detail at a low memory footprint. As an alternative geometric representation, ACORN [Martel et al. 2021] proposes another type of coordinate-based network for learning a highly compressed and accurate occupancy field represented by an optimized space partitioning. While providing accurate reconstruction, it is specifically designed to answer point queries, which prevents its application to many rendering domains. The direct reconstruction of compressed triangle-based representation is a challenging problem even for coordinate-based networks as the underlying data is inherently discrete. Weier et al. [2023] propose to compress and represent the prefiltered appearance while taking into account the correlation arising in structured geometry

using a sparse voxel grid paired with a hash grid for effective compression of the learned signal. Feng et al. [2022] tackle the problem of learning visibility and depth for arbitrary ray queries by proposing a ray-foot parameterization that prevents aliasing of different rays sharing the same intersection, thereby improving the learning process. However, their approach does not compress the geometric representation as it requires two large MLPs for inference, making it impractical for real-time use. Furthermore, the ability of their method to reconstruct complex high-frequency geometric signals is limited by the network’s capacity. NeuralVDB [Kim et al. 2022] compresses the volume of a 3D scene with a set of overlapping domains, each equipped with an MLP mapping local voxel coordinates to voxel data. Its hierarchy is shallow and wide, and the method offers a significant compression ratio for point-queried volume data compared to previous iterations of the VDB representation.

Hybrid neural path tracing. The work of [Fujieda et al. 2023] focuses on optimizing ray-tracing performance through the learning of visibility. While their representation achieves real-time inference with a low memory footprint, their model only handles shadow ray queries and overfits to both camera and light positions, preventing any type of relighting or interactivity.

3 METHOD OVERVIEW

We propose a compact learnt representation for 3D surfaces with support for efficient ray-intersection queries (see Fig. 2). This representation replaces the input geometry with a coordinate-based neural model that we sample along rays to infer intersection point, surface normal, and appearance. Our model is based on a multi-resolution hash grid [Müller et al. 2022] which excels at compactly representing sparse but complex 3D signals.

Neural ray inference is subject to inherent reconstruction error. Beyond the capacity of the neural model, the parameterization of its input strongly influences its performance. Consequently, probing the model at sample points close to the geometry helps it to efficiently store intersection information. Dense sampling along the ray achieves high accuracy but is prohibitively costly; the main challenge is thus to sample the model sparsely but close to the

3D surface. To that end, we take inspiration from classical ray-intersection acceleration to cluster geometry in bounding boxes and probe the model only within the ray-box intersection intervals. Organizing these boxes into a bounding volume hierarchy, dubbed \mathcal{N} -BVH, recovers the benefits of traditional acceleration structures, namely (approximate) front-to-back probing for early termination and efficient empty-space skipping.

Note that we use a single global neural model encompassing the entire geometry, with a BVH to provide efficient training and ray inference in the near-field of the model. Since no geometrical primitives, no textures, and only a shallow BVH need to be stored, our method achieves significant compression rates.

We begin by describing our representation and the practical need for its adaptive probing (Section 4). We then present our \mathcal{N} -BVH structure and its error-driven construction, which we extend to support level-of-detail and further accelerate inference (Section 5).

4 NEURAL RAY QUERY

Given a set of geometric elements and their bounding box, we train a neural model to answer intersection queries for rays that intersect the box. Since our queries are low-dimensional, we adopt the current state-of-the-art approach which consists in combining spatial features with a small fully connected decoding module. Below we detail the encoding of our queries, our neural model, as well as its inference and training pipelines.

Motivation. The simplest way to parameterize our query is via the ray-box entry and exit points; that is, to learn features on the bounding box surface. Unfortunately, this encoding exhibits poor correlation with the signal being learnt, i.e., the ray-intersection point. The intuition is that it stores information far away from the signal, which causes blur and loss in accuracy. We illustrate this behavior in the inline figure, where we focus on the yellow ray entry point (the argument for the exit point is analogous). On the left, we see that rays going through this point have vastly different intersections, and the model is forced to aggregate (i.e., average) information across all of them at the (yellow) encoding point. Moving the encoding point inwards (middle subfigure) brings it closer to the surface where the ray-intersection points correlate more with one another. The ideal encoding location is then the intersection itself where we need to learn information only about that one point.

Encoding, inference & training. In practice we obviously do not know the intersection location—it is what we seek to compute; we therefore resort to sampling the ray-box intersection interval at several locations which parameterize our query. We collect a set of features for each and decode the vector of concatenated features via a small multi-layer perceptron (MLP) to obtain the intersection response. The concatenation *order* here is critical as it encodes the ray’s direction. The features are stored in a spatial multi-resolution hash grid [Müller et al. 2022]. We illustrate this inference scheme in Fig. 3. We train the model by sampling the space of potential rays that intersect a node. For each ray, we (i) sample its origin uniformly within the 50%-inflated scene bounding box,

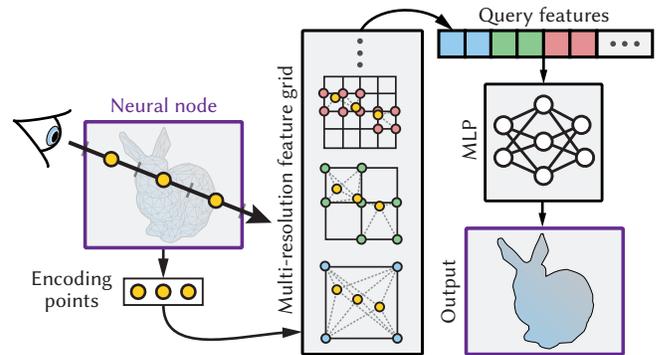


Figure 3: Our neural ray query pipeline. We sample uniformly along a given ray-box intersection interval and at each point collect features from a multi-resolution hash grid. The concatenated features are fed to an MLP to obtain a reconstructed signal (visibility, intersection, normal, appearance).

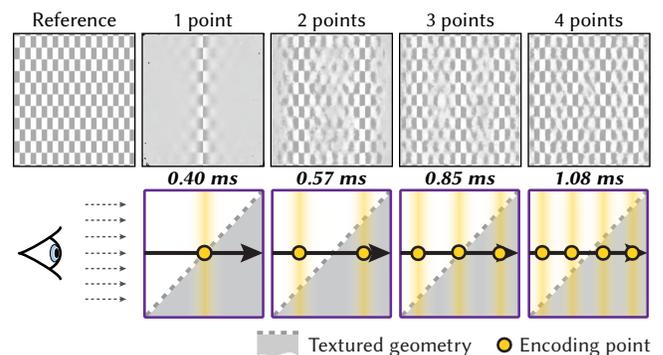


Figure 4: A surface textured with a square checkerboard, observed orthographically at a 45° angle. Reconstruction quality is high when our neural representation is probed close to the surface. Increasing the sampling rate along rays reduces error, but at proportionately higher inference cost.

(ii) sample a uniform direction, and (iii) query our model. The output is compared to the ground truth—obtained by intersecting the actual geometry, and the measured loss is back-propagated to the model’s learnable parameters which are optimized via gradient descent. The loss depends on the specific signal being inferred—visibility, intersection, normal, etc.; we discuss the losses we use in Section 5.2.

Sampling & reconstruction quality. Our ray encoding is obtained via stratified point-sampling along the ray (see Fig. 3, left). We observe that to obtain accurate reconstruction it suffices that one of these points lies close to the surface, as the MLP learns to extract the relevant features from the concatenated vector. To maintain good accuracy we thus need to probe the model close to the surface. A naive way to achieve this is to increase the sampling rate along the ray, as we demonstrate in Fig. 4. Unfortunately, denser sampling also increases inference time, to query the features from the grid and process them. Our \mathcal{N} -BVH structure addresses this challenge.

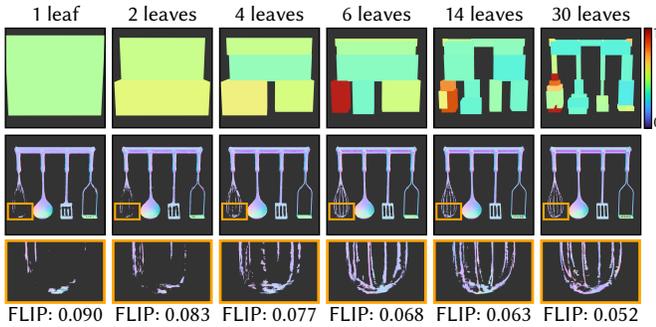


Figure 5: Increasing the number of \mathcal{N} -BVH leaf nodes consistently improves the reconstruction quality. It also improves performance on this scene with low depth complexity where at most one neural inference per ray is needed. The top row visualizes the average training loss per node in false color.

5 NEURAL BOUNDING VOLUME HIERARCHY

Scaling up to large and complex scenes calls for a sampling scheme that has the ability to sample sparsely but close to the geometry. To this end, we draw from decades of ray-tracing acceleration research: We can avoid intersection queries along ray intervals known to be traversing empty space. We split the input geometry into smaller, simpler pieces, enclosing each in a tight bounding box. Probing/training our model then need only consider ray segments inside these smaller boxes. Organizing the boxes into a bounding volume hierarchy (BVH) achieves our goals: it provides front-to-back ray traversal, empty-space skipping, and probing the model closer to the geometry. This lightweight structure is contained within the bounds of our neural model. We call it *neural BVH*, or \mathcal{N} -BVH.

Inference for a given ray proceeds by traversing the \mathcal{N} -BVH and querying the neural representation upon reaching a leaf node as in Fig. 3. The number of neural queries per ray is thus equal to the number of leaf nodes encountered before finding an intersection. In Fig. 5 we demonstrate that a shallow \mathcal{N} -BVH already drastically improves the reconstruction compared to using a single neural node.

We illustrate the structure of our bounding hierarchy in Fig. 2. It resembles that of a classical BVH—the difference is in the depth and the leaf nodes. A classical BVH is usually deep, and its leaves contain geometric primitives. In contrast, our \mathcal{N} -BVH is a very lightweight structure: it is shallow (1–2 orders of magnitude fewer nodes), and its leaves are hollow bounding boxes over larger geometric clusters that are represented implicitly by our neural ray-query model. That model dominates the memory footprint of our combined representation. We next describe how we construct our \mathcal{N} -BVH.

5.1 Error-driven construction

We want our \mathcal{N} -BVH to yield uniform inference error throughout the scene, i.e., its leaves’ error to be roughly equal. We adopt a top-down construction approach which alternates between model training and node splitting.

Base-BVH cut optimization. Instead of building a bounding volume hierarchy from scratch, we leverage the structure of the readily available input-geometry BVH, or *base BVH*. This relieves us from requiring explicit access to the geometry and reduces our task to finding a cut in this BVH. We optimize the cut iteratively starting from the root. In each step, we first train our neural model within the bounds of the nodes along the current cut for a number of iterations (detailed in Section 5.2). We then expand the cut by splitting the nodes with largest error, replacing each with its two children. The number of training iterations between consecutive cut-expansion operations increases progressively by a user-specified factor, and so does the number of node splits (as the tree grows bigger, and the cut longer). We terminate when a target node count is reached. The nodes along the base-BVH cut then become the leaf nodes of our \mathcal{N} -BVH for which we run one final, longer round of training.

Node error. The error introduced by approximating a region in space (i.e., a node’s content) with our representation is the product $q \cdot p$ of the node’s training loss q and the probability p of a random ray hitting it. The loss we discuss in Section 5.2 below; the probability is proportional to the node’s surface area, though we estimate it by the fraction of training rays that hit the node, which gives us the flexibility to adjust the ray distribution (see Section 5.2). In our tests, ranking the nodes by the raw $q \cdot p$ product caused overly aggressive splitting of large and/or high-loss nodes, leading to unbalanced deep trees and low performance. We therefore use a ranking heuristic that dampens the error logarithmically: $r = 2 \log q + \log p$, where the additional factor 2 puts more weight on the loss to discourage splitting large, low-loss nodes too often.

Figure 5 shows a few \mathcal{N} -BVH trees built using the ranking criterion r to select the nodes to be split. In the supplemental document, we confirm on a complex mesh that fixed-depth, uniform node splitting yields worse reconstruction.

5.2 Node training and losses

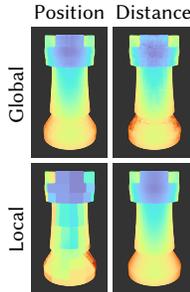
To train our neural model, we sample rays uniformly inside the (inflated) root node, as described in Section 4, and intersect them against the nodes in the current base-BVH cut. We need to train the neural model only within cut-node bounding boxes. Below we describe how we adjust the training to focus on high-error nodes and how we learn the different inference signals.

Ray distribution. The goal of our \mathcal{N} -BVH construction is to achieve spatially uniform inference error. Besides node splitting, we employ two techniques to make better use of our finite training budget. First, for each ray, we train only the first leaf node it intersects, which focuses the effort on more visible nodes. The intersected node is trained only with probability $\max(r/r_{\max}, 0.005)$, where r is that node’s error (Section 5.1) and r_{\max} is the largest error in the cut.

Visibility. The visibility along a ray (segment) can be defined as a binary classification problem. We use a sigmoid activation, thresholding the output against 0.5, and a binary cross-entropy loss which shows better convergence than a typical L_2 loss [Simard et al. 2003]. Visibility (i.e., presence of intersection) is the easiest signal to learn, and we use it during training as an indicator whether intersection information (depth, normal, etc.) should be learnt for each

ray segment. If the ground-truth visibility is 1 (i.e., no intersection), we set the losses for all other data to zero. This ensures that we do not learn unnecessary intersection information.

Intersection point. For the intersection location, we have the choice between learning the 1D distance along the ray or the 3D location directly. Furthermore, each of these can be learned locally, i.e., relative to the node’s extent, or globally. We compare these four options in the inline figure. We find that locally learning the distance, with an L_1 loss, consistently achieves the highest quality. We attribute this behavior to the fact that a 1D signal is easier to learn/represent than a 3D one; constraining the signal to the unit interval also reduces its variation.



Auxiliary intersection data. In our applications, the auxiliary intersection information that we learn are normal and albedo (the other BSDF parameters are fixed by the user). We use a relative L_2 loss for the albedo, which we found to perform better than a regular L_2 loss, even though the albedo signal varies between 0 and 1. For the normal, we use an L_1 loss.

Combined loss. When using our representation in a standard rendering pipeline, e.g., our hybrid path-tracing application, we need to infer all four signals discussed above. The combined training loss we use is $L = 2L_{\text{visibility}} + 2L_{\text{distance}} + L_{\text{normal}} + L_{\text{albedo}}$, with more weight to visibility and distance for better geometric reconstruction.

5.3 Level of detail

Our representation can be trained simultaneously at different scales, or levels of detail (LoD). We apply a simple top-down approach to define *multiple* base-BVH cuts during \mathcal{N} -BVH construction, each defining an LoD. Since our split scheduling increases the number of tree-cut nodes exponentially, we register a new LoD at regular training iteration intervals. This results in a roughly linear increase in \mathcal{N} -BVH depth between LoDs. During training, we randomly select a tree cut and train its nodes.

6 IMPLEMENTATION

We implemented our method in a fully software-based CUDA wave-front path tracer. For training and inference, we use the *tiny-cuda-nn* library [Müller et al. 2022] with half-precision scalars. All our experiments are performed on an NVIDIA RTX 3090 GPU, except for neural prefiltering (Section 8), which is run on an NVIDIA RTX 3080 GPU to match the timings of Weier et al. [2023]. Further implementation details and pseudo-code of our inference pipeline can be found in our supplemental document.

BVH construction. Since our \mathcal{N} -BVH construction requires direct access to the base BVH, we cannot leverage hardware-accelerated construction and traversal. We construct that BVH on the CPU using a sweeping SAH builder [Stich et al. 2009], upload it to the GPU, and traverse it in a dedicated CUDA kernel.

Neural model & training. In all our experiments, we perform gradient descent in batches of 2^{18} rays and employ the Adam optimizer

[Kingma and Ba 2014] with default hyper-parameters and a learning rate of 0.01. The output MLP comprises 4 hidden layers, each containing 64 neurons with ReLU activations. The output layer has sigmoid activation for all ray-query outputs, except for the normal for which a linear activation has shown improved reconstruction quality. The hash grid contains 8 levels, starting from a base resolution of 8^3 to a maximum resolution of 1024^3 , with 4 features per level. To control the network’s memory footprint, we vary only the hash-map size. In scenes where the BVH nodes have extremely thin bounding boxes, training-time node intersection can be subject to floating-point errors. To alleviate this issue, we inflate \mathcal{N} -BVH nodes slightly, following Weier et al. [2023].

7 APPLICATION: HYBRID PATH TRACING

The most direct use of our method is to replace traditional ray-tracing operations for large assets with our \mathcal{N} -BVH, to reduce the overall memory footprint of the scene. We use a two-level hierarchy where the top-level acceleration structure (TLAS) holds several bottom-level structures (BLAS) at its leaves. In this hybrid hierarchy, each BLAS is either a classical BVH or our \mathcal{N} -BVH. Whenever a leaf node in a BLAS is reached, a classical or neural ray intersection query is performed; both query types yield the same type of intersection data. The shading frame and BSDF are instantiated and sampled as usual to determine the next scattering direction for path tracing.

7.1 Results

In addition to the results presented next, our supplemental video demonstrates our real-time \mathcal{N} -BVH construction and inference scheme. The results of our ablation and the renders for all our test scenes can be found in full in the supplemental document and HTML viewer. To compare rendered images to their ground truths, we use the FLIP error metric [Andersson et al. 2020].

Reconstruction quality & performance. In Fig. 6 we plot the rendering error of various \mathcal{N} -BVH configurations on six scenes. For each scene, we show renders and error maps for an \mathcal{N} -BVH configuration that achieves a good trade-off between render time and reconstruction quality. Images for the remaining configurations can be found in the supplemental HTML viewer. These confirm that render times and reconstruction quality correlate most with the \mathcal{N} -BVH node count, less so with the hash-grid size. Note that we deliberately select the most complex assets to be represented neurally; the rest of the scene does not benefit as much from high compression rates. In Table 1 we also compare performance and memory footprint to those of a traditional CUDA path tracer.

Compression rate. The main strength of our approach lies in its ability to compress large and complex scenes into a few tens of megabytes, while maintaining their usability in a traditional rendering pipeline. In Fig. 7 we plot the different compression rates achieved on the scenes from Fig. 6. We distinguish between the rate of the entire scene and the rate of only the portion represented by our neural model. On that portion alone we can achieve over 1000× compression rates over the original geometry footprint. On the entire scene, the compression rates are 5 – 100×. The footprint reduction is mostly due to the geometry representation, though

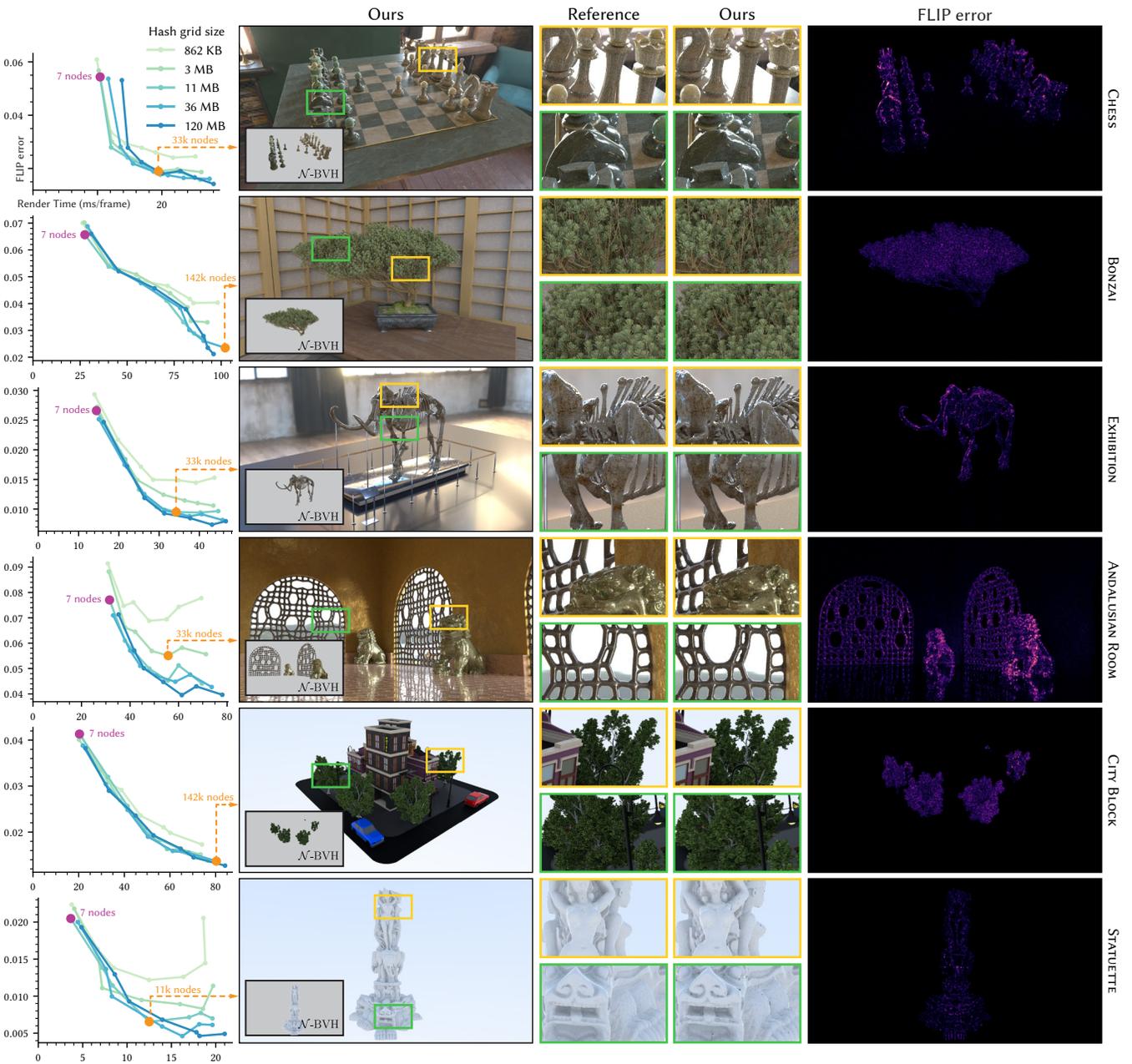


Figure 6: Hybrid path tracing results, rendered at 1920×1080 resolution. On the left, we plot the rendering time of our approach (which is a function of the \mathcal{N} -BVH node count and hash-grid size) with respect to image error. Both the number of nodes in our \mathcal{N} -BVH and the hash-grid size impact performance, the latter however less significantly. On the right, we show the rendered images and their FLIP error for a chosen configuration (●) that achieves a good performance/quality trade-off. In Table 1 we compare the rendering times and memory footprint of two configurations (●, ●) to classical (i.e., non-neural) path tracing.

Table 1: Performance comparison against classical path tracing. Our approach achieves 2–4× higher render times. However, our representation delivers drastic memory compression at low error, allowing the rendering of complex scenes that could not even be uploaded onto low-end GPUs. Reducing the node count of our \mathcal{N} -BVH (●) closes the gap to software path-tracing performance (albeit at a higher error), showing the flexibility of our approach to adapt to strict render-time budgets.

Scene	Path tracing		Our hybrid ●			Our hybrid ●		
	Time	Memory	Time	Memory	FLIP	Time	Memory	FLIP
CHESS	6.6 ms	329 MB	10 ms	36 MB	0.057	19 ms	37 MB	0.019
BONZAI	28 ms	853 MB	29 ms	181 MB	0.069	102 ms	186 MB	0.023
EXHIBITION	13 ms	2.69 GB	14 ms	181 MB	0.027	34 ms	182 MB	0.010
AND. ROOM	28 ms	309 MB	31 ms	108 MB	0.088	55 ms	109 MB	0.055
CITY BLOCK	24 ms	1.55 GB	22 ms	180 MB	0.039	80 ms	185 MB	0.014
STATUETTE	4.9 ms	642 MB	3.7 ms	10.8 MB	0.020	12 ms	11.2 MB	0.007

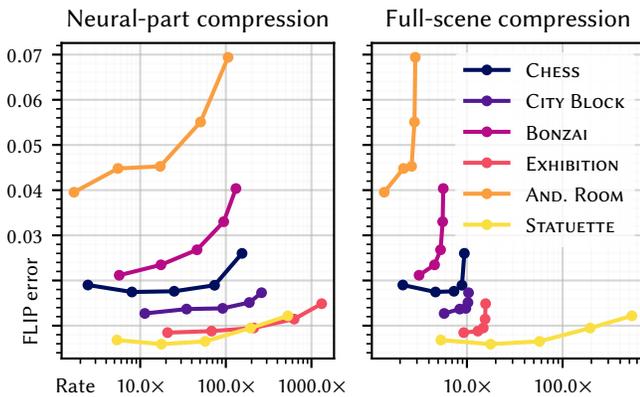


Figure 7: Memory compression rates achieved on our test scenes. We keep the \mathcal{N} -BVH node counts fixed to those chosen in Fig. 6 (●) and vary the hash-grid size along the horizontal axis. We plot the compression rates achieved on the neural part only and on the full scene.

textured albedo and normal data are also learned by our model and contribute to the overall compression. Note that the mammoth in the EXHIBITION scene has extremely high complexity as it is an original consolidated 3D scan from the Smithsonian Institute.

Level of detail. A simple application of LoD already improves the performance of our hybrid path tracing: we switch to a coarser level in our \mathcal{N} -BVH after the primary-ray intersection. Since the number of leaf nodes to query in the \mathcal{N} -BVH can be drastically reduced, we achieve 1.5–2× faster rendering with only slight increase in reconstruction error. In the supplemental document we show additional results on the scene from Fig. 1 to demonstrate the effect on reconstruction quality and render time. Note that no other hybrid path tracing results in the paper use this LoD strategy. We do apply an LoD scheme in our neural prefiltering application (Section 8).

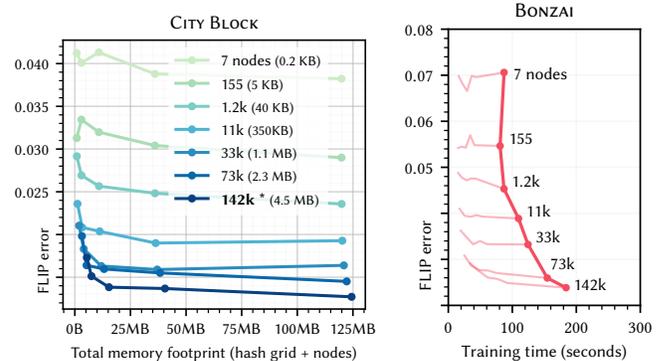


Figure 8: Total memory footprint of our representation vs. rendering error. Along each curve we vary the hash-grid size; that size impacts the error much less than the \mathcal{N} -BVH node count (which is fixed along each curve and its footprint is reported in the legend). The asterisk indicates the node count chosen in Figs. 6 and 7 (●).

Figure 9: Training time vs. error. Hash-grid size is fixed. After an initial 1000-iteration cut optimization to reach a set total node count (indicated on plot), we plot error over the final 5000-iteration training-only stage.

7.2 Ablation study

We next evaluate how the hyperparameters of our model affect its size, training time, inference cost, and reconstruction quality.

Hash-grid size vs. node count. We observe that the biggest improvement in reconstruction quality is achieved by increasing the number of \mathcal{N} -BVH nodes rather than the hash-grid size. Node count impacts memory footprint to a much lesser degree than the number of learnable hash-grid parameters. Figure 8 illustrates how increasing the hash-grid size on the CITY BLOCK scene has diminishing returns on reconstruction quality and rapidly plateaus as the necessary neural capacity is reached. All our test scenes exhibit similar behavior.

Training time. Training our entire pipeline takes only a couple of minutes. We observe that long training times are not necessary to achieve good reconstruction quality. What impacts training time (and error) the most is again the \mathcal{N} -BVH node count, since a larger node count increases the traversal time. In Fig. 9 we fix the hash-grid size and plot the training times achieved for different node counts on the BONZAI scene, our geometrically most complex scene. The same experiment on all our test scenes, with similar results, can be found in the supplemental document.

Hash-grid utilization. While the leaves of our \mathcal{N} -BVH occupy the 3D space covered by the hash-grid model only very sparsely, the model’s *capacity* is still highly utilized thanks to the hash collisions naturally dispersing encoded queries across its learnable features. The supplemental document contains a more detailed table comparing the volume occupied by the \mathcal{N} -BVH leaves on our test scenes and the average number of non-zero gradients during training for different hash-map sizes.

8 APPLICATION: NEURAL PREFILTERING

Weier et al. [2023] showed that neural appearance prefiltering can save large amounts of memory while maintaining low perceptual error. Their method traverses a neural sparse-voxel visibility-only representation to find the first voxel that reports occlusion along the ray. Their appearance sparse-voxel network then predicts the prefiltered appearance (represented as a phase function) for a randomly sampled direction along which the path is continued. We plug into their pipeline, substituting their visibility grid with our *N*-BVH. We keep their appearance network, directly evaluating their publicly available pre-trained weights.

Improved rendering performance. The neural visibility grid of Weier et al. [2023] incurs a major performance overhead as it performs inference in each of potentially many traversed voxels in search for an intersection. Note that they do not require explicit intersection-distance estimation as their appearance network pre-filters the occlusion *inside a voxel* (at a given level of detail).

By replacing their grid with our *N*-BVH, we drastically reduce the number of required visibility inferences. This is due to a single *N*-BVH leaf typically covering much larger space than one of their voxels. Once an intersection is found using our *N*-BVH, we only need to identify in which appearance voxel the intersection took place. We then carefully take into account their voxel-based path-space formulation to trace secondary rays, which requires the ray origin to lie on the previously determined appearance-voxel’s boundary rather than on the intersected surface.

In Fig. 10, we show that our approach achieves similar reconstruction quality on both structured and unstructured geometry; the remaining error is mostly due to the inaccuracies in the appearance network. We matched the seven appearance LoDs with visibility LoDs in our *N*-BVH using as many tree cuts. For the BAY CEDAR scene, our *N*-BVH contains 127 nodes at the coarsest level and 176k nodes at the finest. The ROVER scene requires a much lower node count, 91 and 21k respectively. Hash-map size is 2^{18} for both scenes. On the right of Fig. 11 we report the render times and compare them to the authors’ implementation. Theirs includes a visibility Russian-roulette mechanism to gain a 2–3× speed up over their vanilla implementation. Our approach does not require such a scheme and still easily yields a 2× speed improvement.

Memory footprint & training. In Fig. 11, we compare the memory footprint and training times of the two approaches. Our *N*-BVH exhibits much faster training and lower memory footprint as we can aggressively reduce the hash-grid size without loss of quality.

9 DISCUSSION

Our *N*-BVH delivers intersection distance, normal, and appearance attributes for ray queries. It integrates by design into existing path-tracing pipelines, and offers control over intuitive trade-offs (hierarchy depth, neural capacity) to fit various application scenarios. Its performance is output-sensitive, depending on the amount of allocated storage, the frequency of the scene’s content, and its self-similarity, rather than on the number of input primitives.

Limitations. The main limitation in our neural ray query model lies in the assumption of convexity (or concavity) of the geometry inside a neural node to ensure correct intersection estimation when

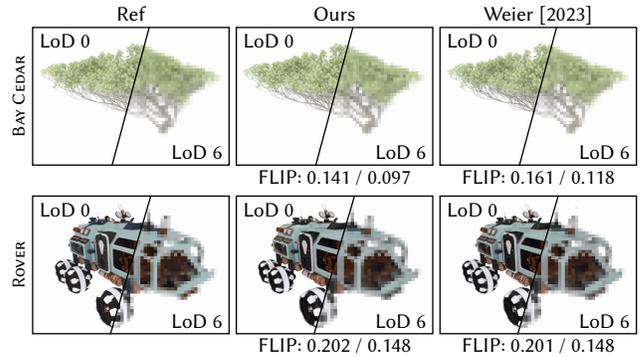


Figure 10: Neural appearance prefiltering comparison. Our *N*-BVH method achieves equal or better reconstruction quality than that of Weier et al. [2023] on both structured and unstructured geometry at all scales, with faster rendering and training times, at lower memory footprint (see Fig. 11).

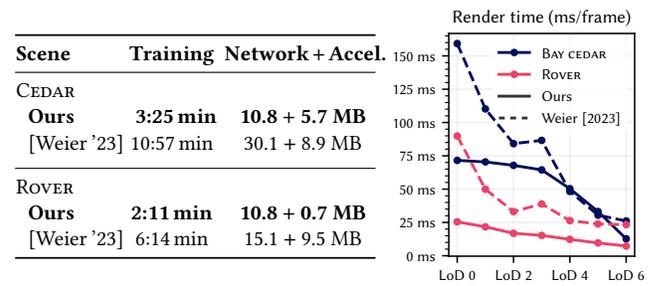


Figure 11: Training-time and memory-footprint comparison of our appearance prefiltering method against that of Weier et al. [2023], at LoD 0 (highest resolution). They use a voxel grid as a ray-acceleration structure, while we use our *N*-BVH. The timing and memory numbers exclude the appearance network. Rendering times are reported for 1024² resolution.

querying the model with a ray originating within the node’s bounds. This can lead to incorrect offsets of secondary ray origins when using a low number of *N*-BVH nodes; as the tree grows deeper, the problem disappears since convexity/concavity is ensured in the limit. In practice, for the scenes we tested, even at a low node count (1.2k), this issue did not manifest. One existing constraint in our encoding scheme is the fixed number of encoded points per node. Although we have identified that three points strike a good balance between inference speed and reconstruction quality, introducing an adaptive sampling approach—increasing points in challenging areas and reducing them in simpler regions—could allow for a more balanced workload between traversal cost and inference time.

Future work. Our approach can be further generalized. Since we fit our representation solely based on the result of ray queries, it is readily applicable to any surface representation that can be intersected. Being based on a standard BVH, it can also inherit future improvements brought to this primitive partitioning structure. Our current method prevents compression of *dynamic* geometry since a naïve solution would require to re-train our representation at every

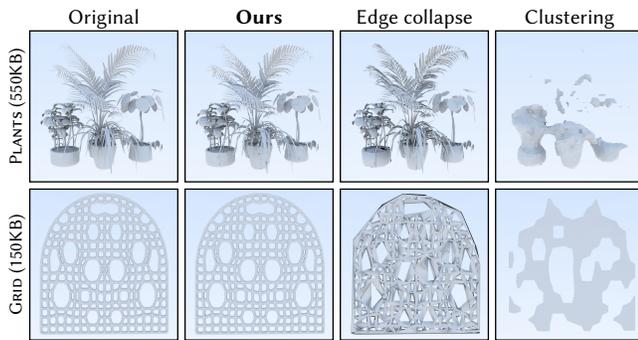


Figure 12: Geometric simplification comparison. We apply classical edge collapse [Garland and Heckbert 1997] and clustering [Schaefer and Warren 2003] to reduce the memory footprint of two scenes from 42 MB to 550 KB and from 9 MB to 150 KB respectively. We compare these to our approach with equal footprint. Our method is more robust to the diversity of signals found in a complex 3D scene than these specialized surface- or spatial-clustering approaches.

frame. Further extending our neural ray-query encoding to handle dynamic content would be an interesting direction for future work.

Conclusion. N -BVH drastically compresses complex scenes while maintaining low rendering error, offering graceful degradation as capacity decreases and blending trivially into classical pipelines. Designed to be efficiently queried by rays, our approach can effectively compress the geometric representation much more accurately than standard geometric simplification techniques such as *edge-collapse* or *spatial clustering* (Fig. 12). Although we cannot yet compete with the speed of raw path-tracing algorithms, we enable the rendering of scenes that would not otherwise fit in memory, and we think that porting our N -BVH on the hardware could benefit from fused neural and hardware ray-tracing operations, i.e., the ability to evaluate our neural model on the fly while traversing the tree, avoiding heavy context switches for each inference.

ACKNOWLEDGMENTS

We thank Jean-Marc Thiery and Loïs Paulin, Pascal Grittmann, Ömercan Yazici, and the Adobe Research team for sharing insights throughout the project. We also thank the anonymous reviewers for their feedback and suggestions. We have used assets from diverse artists: PolyHaven (environment maps), Walt Disney Animation Studio (BAY CEDAR from the Moana Island Scene) and vajrablue (ROVER). This project has received funding from the European Union’s Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 956585.

REFERENCES

Pontus Andersson, Jim Nilsson, Tomas Akenine-Möller, Magnus Oskarsson, Kalle Åström, and Mark D. Fairchild. 2020. FLIP: A Difference Evaluator for Alternating Images. *ACM Comp. Graph. and Interactive Techn.* 3, 2 (2020), 15:1–15:23.

Robert L. Cook, John Halstead, Maxwell Planck, and David Ryu. 2007. Stochastic simplification of aggregate detail. *ACM Trans. Graph.* 26, 3 (jul 2007), 79–es. <https://doi.org/10.1145/1276377.1276476>

Brandon Y. Feng, Yinda Zhang, Danhang Tang, Ruofei Du, and Amitabh Varshney. 2022. PRIF: Primary Ray-based Implicit Function. In *Proceedings of the European*

Conference on Computer Vision (ECCV) (Tel Aviv, Israel). Springer-Verlag, Berlin, Heidelberg, 138–155. https://doi.org/10.1007/978-3-031-20062-5_9

Shin Fujieda, Chih Chen Kao, and Takahiro Harada. 2023. Neural Intersection Function. In *High-Performance Graphics - Symposium Papers*. Jacco Bikker and Christiaan Gribble (Eds.). The Eurographics Association. <https://doi.org/10.2312/hpg.20231135>

Michael Garland and Paul S. Heckbert. 1997. Surface Simplification Using Quadric Error Metrics. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '97)*. ACM Press/Addison-Wesley Publishing Co., USA, 209–216. <https://doi.org/10.1145/258734.258849>

Enrico Gobbetti and Fabio Marton. 2005. Far voxels: a multiresolution framework for interactive rendering of huge complex 3D models on commodity graphics platforms. In *ACM SIGGRAPH 2005 Papers* (Los Angeles, California) (SIGGRAPH '05). Association for Computing Machinery, New York, NY, USA, 878–885. <https://doi.org/10.1145/1186822.1073277>

Peter Hedman, Pratul P. Srinivasan, Ben Mildenhall, Jonathan T. Barron, and Paul Debevec. 2021. Baking Neural Radiance Fields for Real-Time View Synthesis. *ICCV* (2021).

Hugues Hoppe. 1996. Progressive meshes. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '96)*. Association for Computing Machinery, New York, NY, USA, 99–108. <https://doi.org/10.1145/237170.237216>

Doyub Kim, Minjae Lee, and Ken Museth. 2022. NeuralVDB: High-resolution Sparse Volume Representation using Hierarchical Neural Networks. arXiv:2208.04448 [cs.LG]

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

Leif Kobbelt, Swen Campagna, and Hans-Peter Seidel. 1998. A General Framework for Mesh Decimation. In *Proc. Graph. Interface*. 43–50.

Peter Lindstrom. 2000. Out-of-core simplification of large polygonal models. In *ACM SIGGRAPH*. 259–262.

Guillaume Loubet and Fabrice Neyret. 2017. Hybrid mesh-volume LoDs for all-scale pre-filtering of complex 3D assets. *Computer Graphics Forum* 36, 2 (2017), 431–442. <https://doi.org/10.1111/cgf.13138> arXiv:<https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.13138>

Julien N. P. Martel, David B. Lindell, Connor Z. Lin, Eric R. Chan, Marco Monteiro, and Gordon Wetzstein. 2021. Acorn: Adaptive Coordinate Networks for Neural Scene Representation. *ACM Trans. Graph.* 40, 4, Article 58 (jul 2021), 13 pages. <https://doi.org/10.1145/3450626.3459785>

Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. 2020. NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis. In *ECCV*.

Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. 2022. Instant Neural Graphics Primitives with a Multiresolution Hash Encoding. *ACM Trans. Graph.* 41, 4, Article 102 (July 2022). <https://doi.org/10.1145/3528223.3530127>

Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. 2019. DeepSDF: Learning Continuous Signed Distance Functions for Shape Representation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

M. Pauly, M. Gross, and L.P. Kobbelt. 2002. Efficient simplification of point-sampled surfaces. In *IEEE Visualization, 2002. VIS 2002*. 163–170. <https://doi.org/10.1109/VISUAL.2002.1183771>

J. Rossignac and P. Borrel. 1993. Multi-Resolution 3D Approximation for Rendering Complex Scenes. *Modeling in Computer Graphics* (1993), 455–465.

S. Schaefer and J. Warren. 2003. Adaptive Vertex Clustering Using Octrees. In *Proceedings of SLAM Geometric Design and Computing*, 491–500.

P.Y. Simard, D. Steinkraus, and J.C. Platt. 2003. Best practices for convolutional neural networks applied to visual document analysis. In *Doc. Anal. and Recogn.* 958–963.

Martin Stich, Heiko Friedrich, and Andreas Dietrich. 2009. Spatial Splits in Bounding Volume Hierarchies. In *Proceedings of the Conference on High Performance Graphics 2009* (New Orleans, Louisiana) (HPG '09). Association for Computing Machinery, New York, NY, USA, 7–13. <https://doi.org/10.1145/1572769.1572771>

Towaki Takikawa, Alex Evans, Jonathan Tremblay, Thomas Müller, Morgan McGuire, Alec Jacobson, and Sanja Fidler. 2022. Variable Bitrate Neural fields.

Towaki Takikawa, Joey Litalien, Kangxue Yin, Karsten Kreis, Charles Loop, Derek Nowrouzezahrai, Alec Jacobson, Morgan McGuire, and Sanja Fidler. 2021. Neural Geometric Level of Detail: Real-time Rendering with Implicit 3D Shapes. (2021).

Zian Wang, Tianchang Shen, Merlin Nimier-David, Nicholas Sharp, Jun Gao, Alexander Keller, Sanja Fidler, Thomas Müller, and Zan Gojcic. 2023. Adaptive Shells for Efficient Neural Radiance Field Rendering. *ACM Trans. Graph.* 42, 6, Article 259 (2023), 15 pages. <https://doi.org/10.1145/3618390>

Philipp Weier, Tobias Zirr, Anton Kaplanyan, Ling-Qi Yan, and Philipp Slusallek. 2023. Neural Prefiltering for Correlation-Aware Levels of Detail. *ACM Trans. Graph.* 42, 4, Article 78 (jul 2023), 16 pages. <https://doi.org/10.1145/3592443>

Yiheng Xie, Towaki Takikawa, Shunsuke Saito, Or Litany, Shiqin Yan, Numair Khan, Federico Tombari, James Tompkin, Vincent Sitzmann, and Srinath Sridhar. 2022. Neural fields in visual computing and beyond. In *Comp. Graph. Forum*. Vol. 41. 641–676.

Runzhao Yang. 2023. TINC: Tree-structured Implicit Neural Compression. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*.