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

PHILIPPE WEIER, ALEXANDER RATH, ÉLIE MICHEL, ILIYAN GEORGIEV, PHILIPP SLUSALLEK, and TAMY BOUBEKEUR

In this supplemental document we provide further evaluation of the different scenes presented in our paper. In particular, we compare a naive fixed-depth tree cut against our hierarchical optimization and provide pseudo-code and the parameters used for our tree-cut scheduling. We also provide further insights into the different terms of the optimization loss used in our hybrid path-tracing pipeline. Finally, we show the results of our full ablation study for the scenes presented in the paper.

1 INTRODUCTION

Besides the additional evaluations of our method presented next, we encourage the reader to check out our supplemental video showcasing a real-time demo of our approach and our tree-cut optimisation training scheme. The supplemental HTML viewer contains the results of all our tested configurations with memory footprints, FLIP error maps and a side-by-side comparison with reference images.

2 TREE-CUT SCHEDULING

In our paper, we show the ability of our error-driven tree-cut optimisation to automatically adapt to the underlying complexity of the signal (Section 5.1 in the paper). Here in Fig. 1 we demonstrate the benefit of such an approach when compared to a fixed tree depth at an equal node count.

Our error-driven construction starts from a tree cut with a single node (the root), and splits nodes until a target training iteration t is reached. Three user parameters drive the number of nodes that get split at a subsequent training iteration, namely **(i)** the frequency f at which a batch of splits occurs, **(ii)** the scaling factor $s_n > 1$ applied to the current number of splits to perform in a batch and **(iii)** a scaling factor $s_t > 1$ applied to f every time a batch of splits is performed. As a result, our scheduler allocates a growing number of splits every f iterations while simultaneously decreasing the frequency at which splits occur. Beyond the stopping iteration t, the model keeps on learning with no further split of the BVH. For our hybrid path tracing results, we use $t = 3000$ and $s_t = 2$ as well as the following varying parameters depending on our target node count:

Target node count	f	s_n
7	3000	2.0
155	300	2.0
1.2k	100	2.0
11k	50	2.3
33k	40	2.5
73k	50	3.2
142k	90	5.0

In Algorithm 1 we provide a pseudo-code to generate the number of splits at a given iteration using the previously defined parameters.

Algorithm 1 Pseudo code for number of splits performed at training iteration *iter*.

```
function getNumberOfSplits(iter)
    if iter == 0 then
        data ← InitSplitSchedulerData(splitNum = 1 ,
                                       splitInterval = f )
    if iter > t then
        return 0
    numSplits ← 0
    if iter % f == 0 then
        numSplits ← data.splitNum
        data.splitNum ← data.splitNum × s_n
        data.splitInterval ← data.splitInterval × s_t
    return numSplits
```

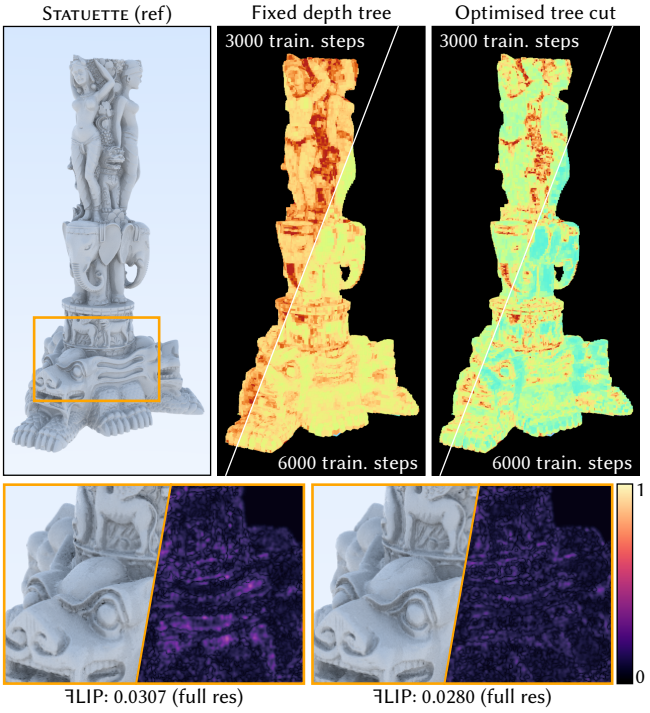

Fig. 1. Our error-driven tree-cut optimization vs a fixed-depth tree cut at equal node count. Our tree-cut scheduling favors splitting high-error nodes instead of low-error nodes. The effect of this can be seen in the loss error maps shown after 3000 and 6000 iterations. While a naive fixed depth tree cut leads to roughly uniform error reduction during training, our tree-cut scheduling ensures that difficult-to-learn regions are split more often to reduce the overall error in the $\mathcal{N}$-BVH.

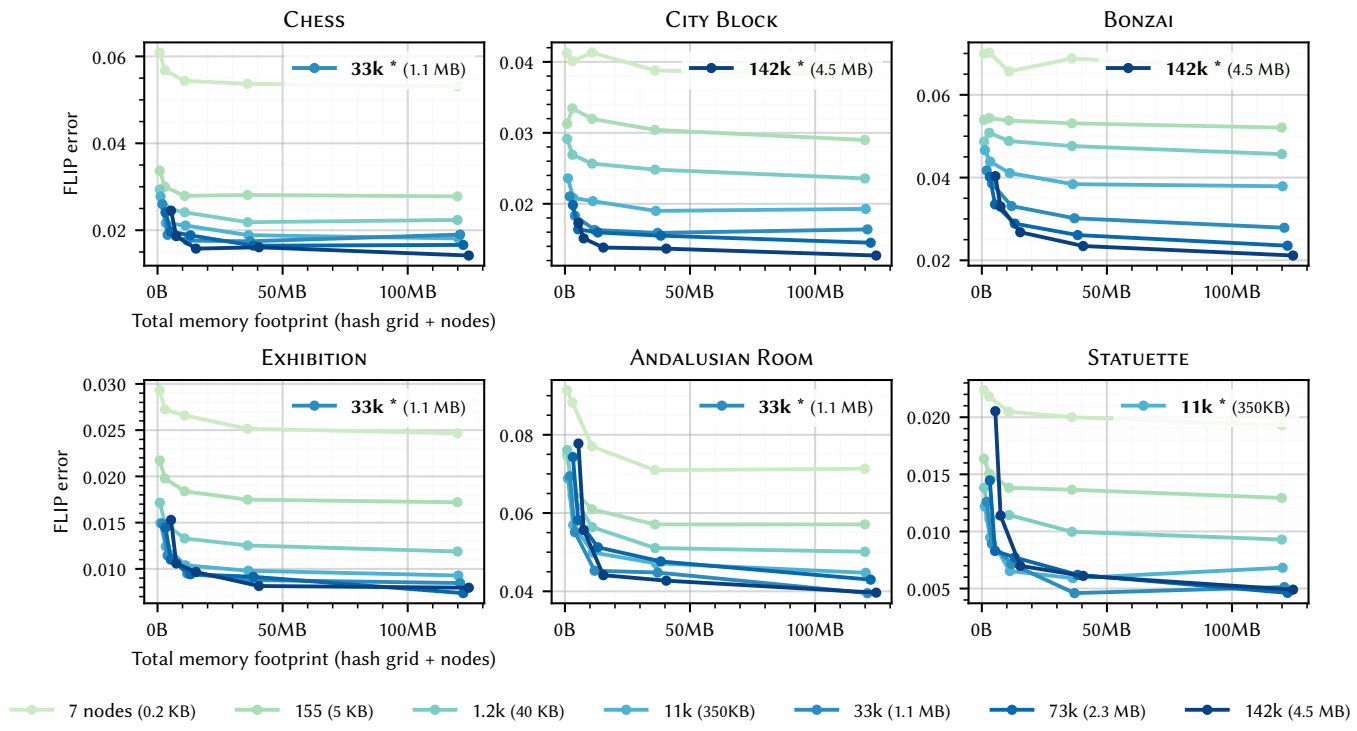

Fig. 2. Total memory footprint of our representation vs. reconstruction error. Along each curve we vary the hash-grid size; that size impacts the error much less than the N-BVH node count. The asterisk indicates the node count chosen in the main results of the paper.

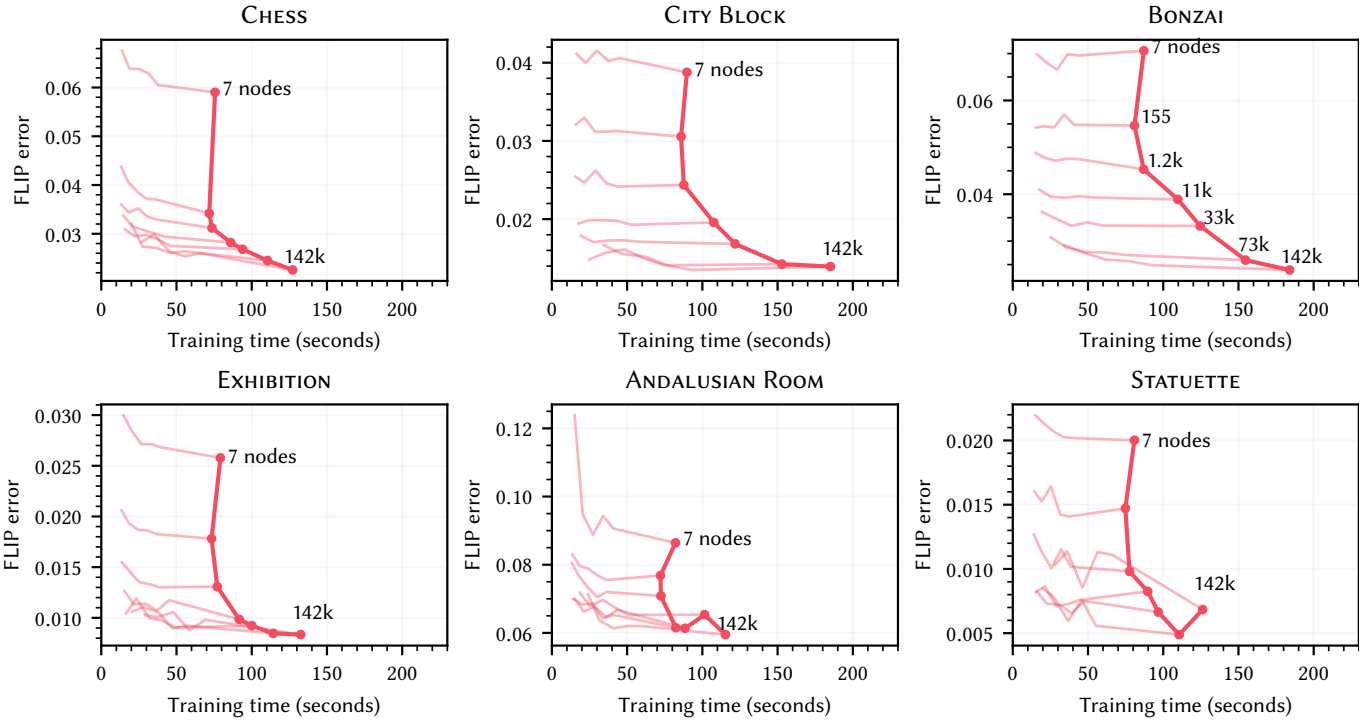

Fig. 3. Training time vs. error. After an initial 1000-iteration cut optimization to reach a set total node count (indicated on plot), we plot FLIP error over an additional 5000 training-only iterations.

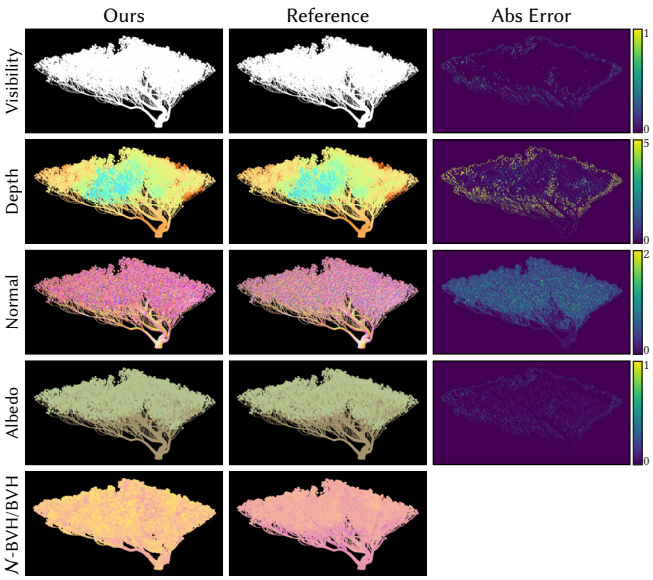

Fig. 4. Hybrid path tracing loss decomposition. We optimise the Bonzai scene for a target node count of 40k nodes and show the absolute error achieved for the visibility, depth, normal and albedo loss separately.

Table 1. We report the hash grid utilization by comparing the mean percentage of non-zero gradients over 100 batches of 2^{18} rays sampled for different hashmap sizes T, and the percentage of space occupied by our $\mathcal{N}$-BVH (140k nodes) relative to the root node.

Scene	Occupied Volume	Non-zero gradients $T = 2^{14}$	$T = 2^{16}$	$T = 2^{18}$
Chess	3.2%	87%	82%	71%
City Block	4.5%	84%	79%	70%
Bonzai	8.1%	87%	85%	78%
Exhibition	2.1%	82%	76%	65%
Andalusian Room	1.7%	80%	74%	62%
Statuette	2.9%	84%	79%	69%

3 HASH-GRID SIZE VS. NODE COUNT

In Fig. 2 we present the additional results of our network memory footprint ablation. As mentioned in the paper, the error decreases most significantly by increasing the number of $\mathcal{N}$-BVH nodes rather than increasing the hash-grid size. This behaviour is consistent over the scenes we tested and demonstrates the ability of our $\mathcal{N}$-BVH to focus the underlying hash grid's capacity on the sparse 3D occupancy swept by the original surfaces.

4 TRAINING TIME VS ERROR

For the scenes presented in our paper, we plot the training times for different node counts in Fig. 3. Training our entire pipeline ranges from a few tens of seconds to 2-3 minutes for all the tested scenes. 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 sample-traversal time.

5 LOSS ABLATION

In Fig. 4 we analyse the different components of our hybrid path tracing loss and compute the absolute error with respect to their corresponding reference. We observe that visibility and albedo are generally simpler to learn than depth and normal. The latter often presents high-frequency details that only a larger number of nodes in the $\mathcal{N}$-BVH allows to learn accurately.

6 HASH GRID UTILIZATION

In Table 1 we report the hash grid utilization for the different scenes we tested. We note that even if the $\mathcal{N}$-BVH only occupies a small portion of the 3D space, we consistently query a much larger volume of the hash grid's occupancy thanks to the hash grid collisions.

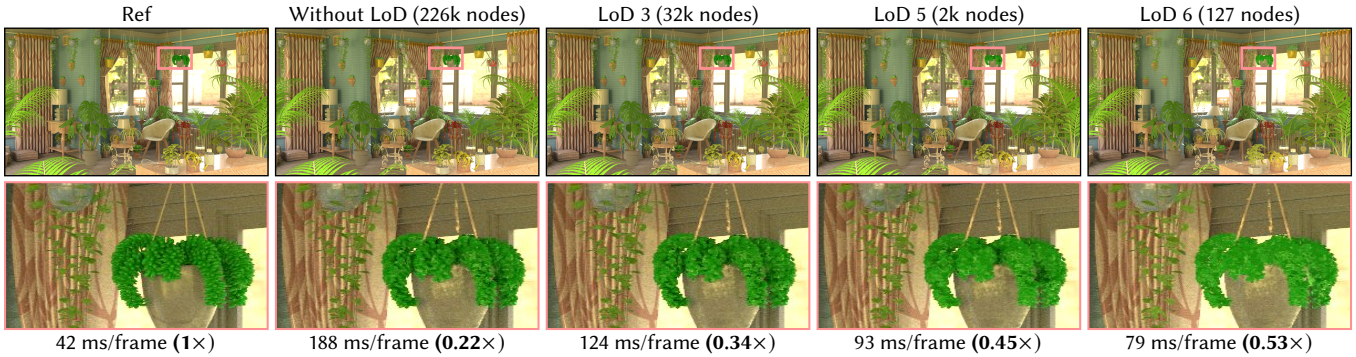

Fig. 5. Primary ray intersection LoD switch. We train our $\mathcal{N}$-BVH for a total of 7 levels of detail, each corresponding to a different number of nodes in the learned tree cut. We then render using the most detailed LoD for primary rays, an one of the coarser LoD for secondary rays. For finer LoDs the perceived error only slightly decreases while already providing a drastic speed-up. For coarse LoDs, close-up renders reveal missing shadows on high-frequency geometry but still provide a solid approximation in the distance.

Algorithm 2 Pseudo code of our Neural Path Tracing pipeline.

```
function NEURALPATHTRACING
    for all pixel ∈ image do
        ray ← cameraRaygen()
        while ray != terminated do
            BLAS ← intersectTLAS(ray)
            if BLAS != neural then
                its ← intersectNonNeuralBLAS(ray)
            else
                its ← intersectNBVH(ray)
            ray ← scatterRay(ray, its)
        pixel ← splatPathContribution(ray)

function INTERSECTNBVH(ray)
    traversalStack ← NBVHRoot
    closestIts ← Its(dist = ∞)
    while traversalStack != ∅ do
        node ← getNextNBVHNode(ray)
        rayQuery ← intersectNode(ray)
        its ← NBVHNetworkInference(rayQuery)
        if its.dist < closestIts.dist then
            closestIts ← its
    return closestIts
```

Table 2. We separately report the average percentage of time spent on the inference alone and the entire $\mathcal{N}$-BVH traversal (including inference time). The configurations used are the same as in our paper's main comparison (Fig. 6), corresponding to a low number of nodes (●) and a high number of nodes (●).

	Low node count ●		High node count ●	
Scene	**Traversal**	**Inference**	**Traversal**	**Inference**
CHESS	56.3%	32.6%	72.9%	33.1%
CITY BLOCK	39.5%	19.3%	75.1%	30.6%
BONZAI	46.3%	23.3%	73.6%	29.5%
EXHIBITION	54.8%	26.9%	76.3%	30.7%
AND. ROOM	35.8%	21.1%	59.8 %	26.7%
STATUETTE	66.7%	36.5%	87.7%	37.9%

7 PER-RAY LEVEL OF DETAIL

Finally, we describe a simple application of our $\mathcal{N}$-BVH LoD scheme to accelerate our hybrid path tracing pipeline. While primary rays have a large impact on the perceived error in the final rendered image, most secondary rays can be less accurate without introducing high error. Therefore, we propose to switch the $\mathcal{N}$-BVH to a coarser level of detail (corresponding to a tree-cut with fewer nodes) during secondary ray intersection. In Fig. 5 we show on our teaser that this simple heuristic can drastically reduce the number of inference queries and easily achieve 1.5–2× faster rendering times for only a slight decrease in reconstruction quality. Transitioning to a tree-cut with a low node count is predominantly noticeable in the shadows, as they still demand a precise visibility estimate to accurately replicate their high-frequency characteristics.

8 $\mathcal{N}$-BVH TRAVERSAL PROFILING

In Table 2 we provide further insights into the performance of our $\mathcal{N}$-BVH by profiling the time spent on traversal. We report the timings for traversing the $\mathcal{N}$-BVH (including inference) and the time spent on inference solely. The remaining time is spent on ray generation and scattering, as well as TLAS and non-neural BLAS traversal.

9 FURTHER IMPLEMENTATION DETAILS

Depth estimation. The *order* of the ray-query's concatenated features allows us to estimate depth from within nodes. The default order follows the ray direction; when a ray query is issued from within a node, we reverse that order. Then, an intersection occurs inside the node only if the inferred distance is smaller than the distance from the ray origin to the node's exit point.

Neural path tracing inference pipeline. A pseudo code of our neural path tracing inference pipeline is described in Algorithm 2. Note that when an $\mathcal{N}$-BVH is intersected with a ray (function INTERSECTNBVH), we first get the next intersected node, compute the ray-query encoding, pause the traversal to perform inference (in a separate kernel), update the closest intersection and finally resume the traversal until the traversal stack is empty.