

NeuralPVS: Learned Estimation of Potentially Visible Sets

XIANGYU WANG, University of Stuttgart, Germany

THOMAS KÖHLER and JUN LIN QIU, Graz University of Technology, Austria

SHOHEI MORI, University of Stuttgart, Germany

MARKUS STEINBERGER, Graz University of Technology, Austria

DIETER SCHMALSTIEG, University of Stuttgart, Germany

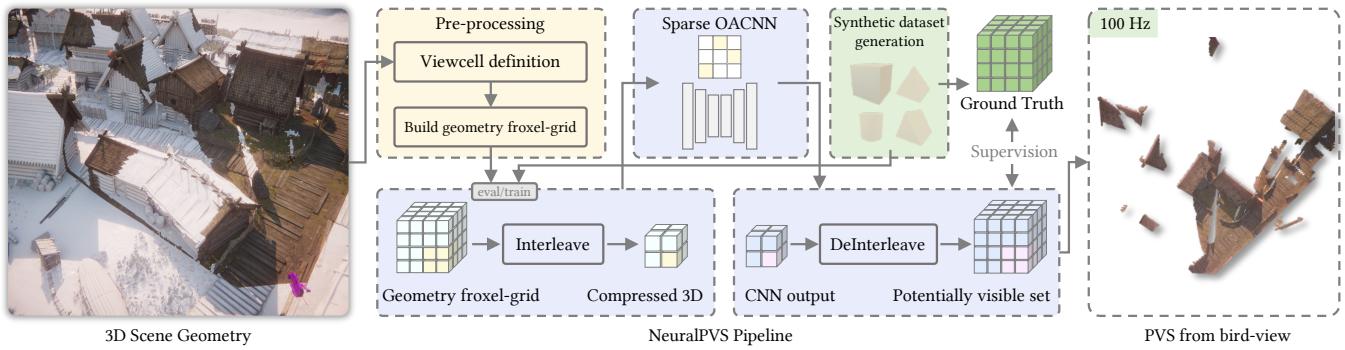


Fig. 1. Overview of the NeuralPVS pipeline. The left side illustrates the overall system and task, where the camera is colored purple, and the white rendering indicates geometry invisible to the camera. A froxelized representation of the input scene is fed into the neural network with interleaving layers and outputs the potentially visible set (PVS) in froxelized form, as displayed in the middle. The network is trained with pairs consisting of a froxelized scene and the corresponding ground-truth PVS in froxelized form. The network runs at 100 Hz (10 ms per frame) on the GPU and generates less than 1% error rate, without introducing noticeable artifacts in the rendered images. The right side shows the rendered PVS of the frame from a bird's-eye view.

Real-time visibility determination in expansive or dynamically changing environments has long posed a significant challenge in computer graphics. Existing techniques are computationally expensive and often applied as a precomputation step on a static scene. We present NeuralPVS, the first deep-learning approach for visibility computation that efficiently determines from-region visibility in a large scene, running at approximately 100 Hz processing with less than 1% missing geometry. This approach is possible by using a neural network operating on a froxelized representation of the scene. The network's performance is achieved by combining sparse convolution with a 3D volume-preserving interleaving for data compression. Moreover, we introduce a novel repulsive visibility loss that can effectively guide the network to converge to the correct data distribution. This loss provides enhanced robustness and generalization to unseen scenes. Our results demonstrate that NeuralPVS outperforms existing visibility methods in terms of both accuracy and efficiency.

Authors' Contact Information: Xiangyu Wang, xiangyu.wang@visus.uni-stuttgart.de, University of Stuttgart, Stuttgart, Germany; Thomas Köhler, t.koehler@tugraz.at; Jun Lin Qiu, qiu@student.tugraz.at, Graz University of Technology, Graz, Austria; Shohei Mori, s.mori.jp@ieee.org, University of Stuttgart, Stuttgart, Germany; Markus Steinberger, markus.steinberger@icg.tugraz.at, Graz University of Technology, Graz, Austria; Dieter Schmalstieg, dieter.schmalstieg@visus.uni-stuttgart.de, University of Stuttgart, Stuttgart, Germany.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

© 2025 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM XXXX-XXXX/2025/9-ART
<https://doi.org/10.1145/nnnnnnn.nnnnnnn>

CCS Concepts: • Computing methodologies → Visibility.

Additional Key Words and Phrases: Convolutional neural network

1 Introduction

Visibility computation is one of the fundamental problems in computer graphics, as it enables a broad spectrum of applications such as shadow mapping, light-field rendering, global illumination, or collision detection. A common approach is to determine a potentially visible set (PVS), which contains an approximation of the visible portion of a scene [Teller and Séquin 1991]. Many methods have been proposed for computing a PVS [Cohen-Or et al. 2003], either from a single viewpoint or from a region (a so-called *viewcell*). Commercial game engines frequently determine a from-point PVS before shading to reduce shading load. Computing a from-region PVS has much broader use for applications such as frame extrapolation, prefetching or streaming rendering. Unfortunately, algorithms for from-region PVS have an inherently high computational complexity [Durand et al. 2002], and most algorithms rely on precomputation [Bittner et al. 2009; Mattausch et al. 2006], including solutions used in commercial game engines such as Unreal [Epic Games 2022]. Some recent work [Hladky et al. 2019; Kim and Lee 2023; Koch and Wimmer 2021; Voglreiter et al. 2023] achieves promising online visibility computation, but the limitations of the underlying algorithms still restrict their applicability, especially for high resolutions and scenes with high geometric complexity.

In recent years, neural networks have revolutionized many areas of computer graphics, such as material representation [Dou et al.

2024; Sztrajman et al. 2021], denoising [Bako et al. 2017; Chaitanya et al. 2017], and global illumination [Diolatzis et al. 2022; Ren et al. 2024; Zheng et al. 2024]. Even complex and ill-posed problems can be successfully handled by neural networks, if enough training data is available for supervised learning. In computer graphics, the problem of procuring enough training data can often be solved by generating synthetic images or datasets. With sufficient optimization, the resulting trained network may be able to run on a GPU at rates that are competitive to conventional algorithms. For example, NVIDIA DLSS [Liu 2020] uses a neural network for frame upsampling.

In this paper, we explore the use of a convolutional neural network (CNN) to compute a from-region PVS. We rasterize the given scene into a froxel grid, i.e., a frustum-aligned grid expressed in normalized device coordinates [Evans 2015]. Then, we compute a ground-truth PVS as another grid in the same space, by sampling visibility from many viewpoints in the viewcell. Pairs of geometry grid and ground-truth PVS are derived from synthetic scenes and used for training. At inference time, the network predicts the PVS for the scene as seen from the current viewpoint.

Our approach, called *NeuralPVS*, makes use of the most recent refinements to CNN architectures, which rely on sparse and adaptive convolutions [Peng et al. 2024]. To boost the speed of our network to real-time performance, we extend the network with 3D interleaving of the froxelized input data. Our end-to-end training pipeline allows the model to learn the visibility patterns directly from synthetic scenes, which consists of basic geometric primitives with varying complexity and occlusion patterns. The trained network can be applied to arbitrary scenes without any need for fine-tuning. This design makes our network a drop-in replacement for a conventional PVS generator. We also introduce a novel repulsive visibility loss function that encourages the model to focus on the most relevant regions and stay away from invisible regions. Our main technical contributions can be summarized as follows:

- We propose a novel neural network for from-region PVS computation, which has significantly improved accuracy and speed compared to existing methods.
- We explore 3D volumetric interleaving and repulsive visibility loss to enhance the efficiency of the neural network, while preserving the accuracy of visibility estimation.
- We develop a rendering pipeline and application with our proposed method, which demonstrates the effectiveness of our approach through extensive experiments on large scenes, showing that our method outperforms existing state-of-the-art methods in terms of both accuracy and efficiency.

2 Background

We begin by surveying prior work on PVS computation, distinguishing between offline methods and those designed for real-time execution. We then review recent advances in integrating deep learning into the traditional graphics pipeline, highlighting how neural architectures have been employed to solve graphics tasks. Finally, we discuss contemporary techniques in 3D geometric learning, particularly those that inform the backbone network architecture adopted in our work.

2.1 PVS computation

For performance reasons, computing PVS offline for the region is a common practice to speed up rendering. The space of possible viewpoints can be subdivided into static viewcells, and the PVS per viewcell can be computed and stored [Teller and Séquin 1991]. Early offline PVS computation methods are restricted to handling 2.5D geometry [Wonka et al. 2000] and often imposed additional constraints such as watertight geometry [Schaufler et al. 2000] or particular data distribution schemes [Cohen-Or et al. 1998]. Subsequent research aimed to remove these restrictions and address more general geometric scenarios using various techniques based on rasterization [Bittner et al. 2005; Nirenstein and Blake 2004] or raytracing [Bittner et al. 2009]. The main problem of offline methods remains that they can only work with static scenes.

Modern graphics applications, e.g., 3D streaming and dynamic 3D scenes, call for PVS computation methods that can operate in real time. From-point methods often rely on depth buffering to test and store occlusions. For example, rendering engines can use a geometry pre-pass to establish visible fragments [Burns and Hunt 2013]. If a full-resolution geometry pre-pass is considered too expensive, an efficient software rasterizer working at lower resolution [Hasselgren et al. 2016] or a hierarchical depth buffer [Greene et al. 1993] can be used. The latter can be accelerated with hardware occlusion queries, although the need for GPU synchronization limits scalability [Mattausch et al. 2006]. Another approach reduces the required geometry processing by replacing complex scene geometry with simpler occlusion proxies [Koltun et al. 2000]. Finding good proxies is still an active research topic [Tan et al. 2025].

Only a few methods address online computation of a from-region PVS. Simple solutions can try to heuristically sample multiple predicted camera positions [Hladky et al. 2022; Mueller et al. 2018; Reinert et al. 2016], but this approach is not very scalable.

The camera offset space of Hladky et al. [2019] uses per-pixel linked lists as a scene representation to determine the PVS. Maintaining sorted lists is expensive and difficult to scale to high resolutions. Koch et al. [2021] use hardware ray-tracing to find visible surfaces. Their method is stochastic, and its convergence depends on the speed of the raytracing hardware. Kim and Lee [2023] propose track disocclusions through depth peeling. Scalability of depth peeling is limited by the need to repeatedly rasterize the scene. Voglreiter et al. [2023] combine traversal of a coarse octree in world space with k-buffering in image space. Peeling octree layers instead of traditional depth peeling allows them to build the PVS with a single geometry pass. However, the scalability of their method is still affected by the need for GPU synchronization after each layer.

Recently, a disocclusion-based approach was proposed [Anonymous 2025], introducing the disocclusion buffer as a sparse, layered representation that allows order-independent and fully parallel computation of PVS, achieving speed-ups with comparable accuracy. Although both this concurrent work and ours advance PVS generation, our neural network-based approach will benefit even more from future GPU advancements, as improved hardware directly accelerates inference performance.

Our method builds on the observation that visibility can be efficiently expressed in a froxelized view frustum. A froxel grid requires

only one bit per froxel to indicate a disoccluded region. Such a froxel grid can be built much more efficiently than linked lists, k-buffers or depth peeling layers. We use the froxel grid as input to a CNN, which has a run-time cost that is only dependent on the froxel grid resolution. This resolution is independent of both the target image resolution and the complexity of the scene.

2.2 Deep learning in graphics

With the ability to learn and model complex mappings, deep learning is increasingly gaining recognition as a new approach to overcome bottlenecks in computer graphics because of its predictable computational cost and speed.

Deep Shading [Nalbach et al. 2017] is one of the earliest works to introduce a CNN into graphics tasks to compute the shading effect on pixels from shading buffers. A CNN has also been used to efficiently denoise Monte Carlo rendering [Bako et al. 2017; Chaitanya et al. 2017]. Similar neural network-based techniques are widely used in post-processing, such as by DLSS [Liu 2020]. BRDF methods based on deep learning [Dou et al. 2024; Hu et al. 2020; Sztrajman et al. 2021] use neural layers to represent BRDF to replace the large tabulated dataset and can achieve a better trade-off between memory and quality. Neural global illumination [Diolatzis et al. 2022; Ren et al. 2024; Zheng et al. 2024] can now compete in quality for dynamic lighting simulation [Ren et al. 2024] and deliver convincing results on dynamic geometry [Zheng et al. 2024].

To the best of our knowledge, no existing work addressed visibility computing with a neural network. In this paper, we will focus on applying deep learning methods to the field of PVS computation.

2.3 3D geometric learning

Our PVS estimation network combines several key techniques from 3D geometric learning, including sparse 3D convolution and the latest improvements to the convolutional network design.

Classical 3D convolutional networks [Dai et al. 2017; Tchapmi et al. 2017] use dense representation to learn 3D geometry patterns. Dense representations have extremely high memory consumption and lead to slow inference speed. Most geometric models have ample empty space and do not require convolutions to be applied in the empty areas. Therefore, a sparse CNN architecture is usually preferred for this kind of problem.

Instead of storing the entire space as one tensor, a sparse 3D convolution network stores its data, the so-called sparse tensor, either in an octree [Riegler et al. 2017] or in a hash table [Chen et al. 2022; Choy et al. 2019; Graham and Maaten 2017; Spconv Contributors 2022]. Sparse 3D convolutions on hash tables are optimized by placing the kernel center at the activated positions [Graham and Maaten 2017], dilating to the activated position’s neighbors [Choy et al. 2019], dynamically dilating the reception field [Chen et al. 2022], or combinations of these factors [Choy et al. 2019; Spconv Contributors 2022]. Recent work [Peng et al. 2024] on an omni-adaptive convolutional neural network (OA-CNN) demonstrates that advances in transformer networks can be retrofitted to CNN architectures by adding adaptive receptor fields and a form of self-awareness. Our method benefits from the speed and efficiency

afforded by these architectures, since a volumetric scene typically has an occupancy of less than 5% of the froixels.

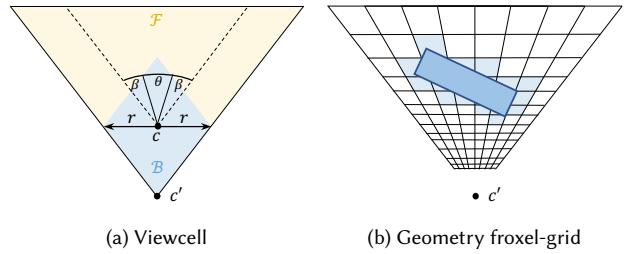


Fig. 2. (a) A frustum enclosing all primitives that are potentially visible from a viewcell (blue area) with radius r around the current viewpoint c is created by displacing the viewpoint backwards to c' . (b) The scene primitives are conservatively rasterized into a regular grid.

3 Method

Our goal is to significantly improve the performance of PVS computation by replacing previous algorithms operating on analytic or sampled geometry with a robust neural network operating on a froxelized scene representation. The network is pre-trained with synthetic geometry grids corresponding to random scenes that loosely resemble the structure of the target scenes. A key advantage of using a CNN comes from the fact that converting the polygonal scene into a froxel grid of fixed resolution makes the time needed to compute a PVS largely independent of geometric scene complexity. Overall, our pipeline proceeds as follows (Figure 3):

- (1) **Viewcell definition:** build a geometry grid G by rasterizing the scene into G
- (2) **PVS generation:** a single forward pass through the neural network infers the PVS
- (3) **Novel view synthesis:** rasterize only primitives contained in the PVS

Step 3 is repeated until the camera leaves the current viewcell, then the process is restarted with step 1.

3.1 Preliminaries

We represent the scene geometry by a grid $G(\mathbf{x})$, $\mathbf{x} \in \mathbb{X}$, in normalized device coordinates, which is 1 if the froxel at \mathbf{x} is occupied, and 0, otherwise. G is defined over a discrete domain $\mathbb{X} \equiv [1..N_x] \times [1..N_y] \times [1..N_z]$, which splits the view frustum into a grid of discrete froixels.

The potentially visible set can be defined as the union of visible polygons for all the viewpoints in a cell [Airey et al. 1990]. Based on the definition, we represent the PVS in volumetric form as a grid $V(\mathbf{x})$, which is 1 for visible froixels, and 0 otherwise. Let $N = N_x \cdot N_y \cdot N_z$. Our goal is to learn a mapping from G to the ground-truth per-froxel visibilities $\hat{V} : \{0, 1\}^N \rightarrow \{0, 1\}^N$. We train a 3D convolutional network f_θ that produces per-froxel probabilities and obtain $V = \mathbf{1}(f_\theta(G))$, where $\mathbf{1}$ is an indicator function $[\hat{V}(\mathbf{x}) \geq \tau]$ with a threshold τ .

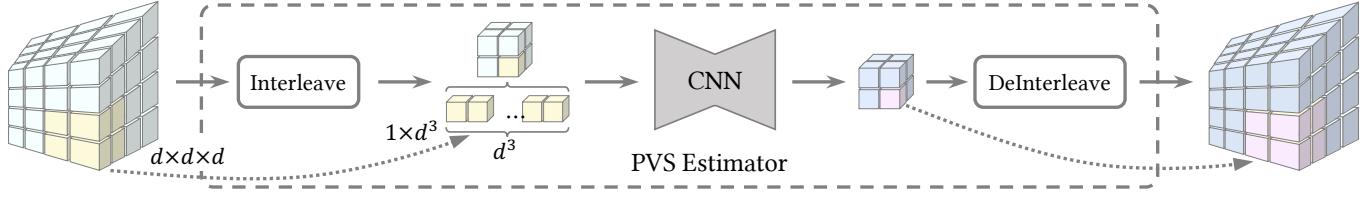


Fig. 3. NeuralPVS pipeline. For each viewcell, the scene’s geometry is voxelized into a GV, which is input to the PVS estimator network. A 3D interleaving function first compresses the GV channels; a CNN then predicts the visible part of the geometry grid; afterwards, a 3D deinterleaving function reconstructs the full PVS. Geometric primitives in voxels marked invisible in the PVS are culled from all further rendering computations.

3.2 Pre-processing

Our definition of the viewcell \mathcal{B} (Figure 2a) considers a lateral motion of the camera from an original viewpoint c up to radius r [Wonka et al. 2001]. If we assume that the camera has a field of view of θ , the view frustum \mathcal{F} associated with a viewpoint c' displaced backward by $r/\tan(\theta/2)$ encloses the PVS associated with \mathcal{B} . To accommodate camera rotations up to a maximum angle of β on either side, we enlarge the field of view to $\theta + 2\beta$.

We rasterize the primitives contained in \mathcal{F} into a geometry grid G stored as a 3D texture, according to the concept of voxelization [Evans 2015]. Each fragment $(x, y, z)^\top$ in normalized device coordinates is quantized to the voxel coordinates at $(\lfloor uN_x \rfloor, \lfloor vN_y \rfloor, \lfloor wN_z \rfloor)^\top$, and the texture at the corresponding location is set to 1 to indicate an occupied voxel. To ensure a gap-free rasterization, we supersample the scene at a resolution of (sN_x, sN_y, sN_z) . For storage efficiency, we pack eight consecutive voxels along the x -axis into an 8-bit integer using atomic bitwise-OR for concurrent texture writes. For the ground-truth generation during training, the fragments are generated using orthographic projections and then reprojected on the fly into \mathcal{F} to ensure a uniform sample distribution across all distances from the camera. At test time, we prioritize speed and use a conventional perspective projection to generate the samples for the geometry grid.

By construction, $\text{supp}(V(x)) \subseteq \text{supp}(V)$, which allows immediate occlusion culling: Any fragment mapping to a voxel where $\hat{V} = 0$ can be discarded. As proposed by Hladky et al. [2019], the ground truth PVS is computed by dense sampling of the viewcell. A large number ($M = 1000$) of viewpoints $c_m \in \mathcal{B}$ is selected. For each viewpoint, the primitives in \mathcal{F} are rendered and a depth buffer is produced. The fragments indicated in the depth buffer are re-projected to the coordinate system of c , and the corresponding voxels in G are marked as occupied.

3.3 Neural PVS estimation

Dense volumetric CNN architectures, such as VNet [Milletari et al. 2016], are designed for offline operation, such as segmentation of medical scans. Even their sparse variants are too slow for applications in real-time graphics. Therefore, we adopted OA-CNN [Peng et al. 2024] as a backbone. OA-CNN is built on a highly optimized kernel for sparse convolution. Furthermore, it introduces adaptive receptive fields and dynamically adjusts convolutional kernel weights to deliver performance that reflects modern transformer networks.

The network accepts a geometry grid G as input and provides volumetric probabilities V as output. The complete pipeline is described in Figure 3.

3D volume-preserving interleaving. The CNN inference time complexity is linearly dependent on the resolution of the input features, while the number of features has less impact on the speed. To further improve the inference speed, we adopt a 3D volume-preserving interleaving, generalizing the mechanism proposed by Xiao et al. [2018]. As shown in Figure 3, we place an interleaving function g_d before the convolutional layers and a corresponding de-interleaving function after the convolutional layers:

$$V = g_d^{-1}(f_\theta(g_d(G))).$$

The interleaving function

$$g_d : \mathbb{R}^{N_x \times N_y \times N_z} \rightarrow \mathbb{R}^{\frac{N_x}{d} \times \frac{N_y}{d} \times \frac{N_z}{d}}$$

takes as input a grid of $N_x \times N_y \times N_z$ voxels. It divides the grid into blocks of dimension $d \times d \times d$ and stacks the voxels in a block into a one-dimensional feature vector. The de-interleaving function

$$g_d^{-1} : \mathbb{R}^{\frac{N_x}{d} \times \frac{N_y}{d} \times \frac{N_z}{d}} \rightarrow \mathbb{R}^{N_x \times N_y \times N_z}$$

inverts this process. We choose $d \in \{8, 16, 32\}$ for optimal memory alignment. A value $d < 8$ is not practical, as the setup overhead becomes too high [Xiao et al. 2018]. The interleaving preserves the relative positional information of the geometry, while shrinking the dimension of the input by a factor of d^3 . For the typical setup of $d = 16$, the first convolution step after interleaving further shrinks the size of each input vector from d^3 to around 200 elements, leading to a significantly reduced processing time.

Weighted Dice loss. We adopt a weighted Dice loss, which weighs false negatives (FN) much more strongly than false positives (FP) by a factor α . In our context, false negatives are voxels included in the ground truth PVS, but missing from the predicted PVS. False positives are voxels not included in the ground truth, but included in the predicted PVS. True positives (TP) are voxels which are included both in the ground truth and the predicted PVS, and ground truth positives (GTP) are all voxels marked in the ground truth \hat{V} :

$$\text{TP}(V, \hat{V}) = \sum_{x \in \mathbb{X}} \hat{V}(x) \cdot V(x), \quad \text{FP}(V, \hat{V}) = \sum_{x \in \mathbb{X}} \hat{V}(x) \cdot (1 - V(x)),$$

$$\text{FN}(V, \hat{V}) = \sum_{x \in \mathbb{X}} (1 - \hat{V}(x)) \cdot V(x), \quad \text{GTP}(\hat{V}) = \sum_{x \in \mathbb{X}} \hat{V}(x).$$

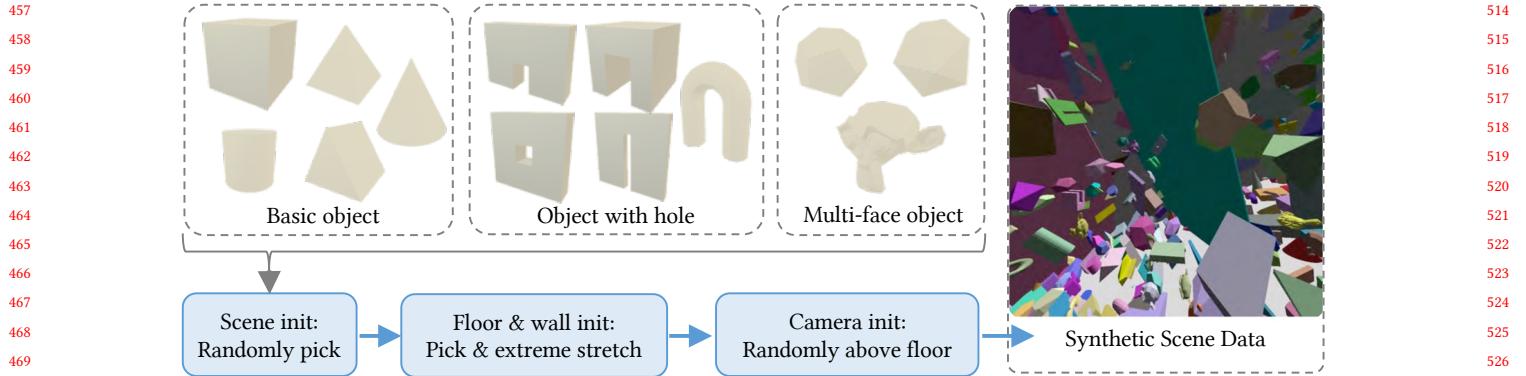


Fig. 4. The pipeline of our proposed synthetic 3D data generation. To ensure the data distribution diversity, we applied a multi-step scene generation strategy in order to cover as many occlusion patterns as possible.

Following the definition of Taha and Hanbury [2015], our modified Dice loss is given as

$$\mathcal{L}_{dice}(V, \hat{V}) = 1 - \frac{2 \operatorname{TP}(V, \hat{V})}{2 \operatorname{TP}(V, \hat{V}) + \alpha \operatorname{FP}(V, \hat{V}) + (1 - \alpha) \operatorname{FN}(V, \hat{V})}.$$

Repulsive visibility loss. Our training data is necessarily very imbalanced, because the number of visible froxels in real scenes is usually between 0.5% and 10% and can vary significantly between scenes and viewpoints. This imbalance makes it difficult to achieve convergence, as the network can easily get stuck in a local minimum during training. The conditioning to avoid high false negative counts can lead to overprediction of visibility, or the network never leaves the initial state.

To overcome these imbalance issues, we propose a repulsive visibility loss (RVL) inspired by the work of Wang et al. [2018]. Unlike Dice loss, which globally supervises all froxels, RVL aggressively encourages only prediction in local grids where $V = 1$ to match the overall distribution of FN/FP. This loss is given as

$$\mathcal{L}_{rv} = \mathcal{L}_{attr} + \mathcal{L}_{rep}, \quad \mathcal{L}_{attr} = 1 - \frac{\operatorname{FN}}{\operatorname{GTP}}, \quad \mathcal{L}_{rep} = \frac{\operatorname{FP}}{\operatorname{GTP}},$$

where \mathcal{L}_{attr} attracts the prediction to match the ground truth, and \mathcal{L}_{rep} pushes the prediction away from non-visible areas. We train the model by minimizing a combination of \mathcal{L}_{dice} and \mathcal{L}_{rv} :

$$\mathcal{L} = \lambda \cdot \mathcal{L}_{dice}(V, \hat{V}) + (1 - \lambda) \cdot \mathcal{L}_{rv}(V, \hat{V}).$$

4 Implementation

4.1 Datasets

The training dataset must include a wide range of geometric variations to ensure generalization. To achieve this, we use synthetic data. We randomly place selected objects at varying frequencies and apply random scales and rotations with a consistent distribution across the local object dimensions (Figure 4). Each training set is composed of 1,000 frames.

Our objects consist of primitive and complex shapes, including cubes, cones, pyramids, cylinders, dodecahedrons, icosahedrons, and round arches, as well as the monkey head model from Blender, a wall with a door-shaped cutout, and a cube with a square hole

resembling a window. To simulate architectural structures such as walls, floors, and ceilings, we scale certain objects significantly along two of their three dimensions. Finally, we introduce planes that serve as a global base for the floor, wall, and ceiling, extending across the scene. These planes are initialized to ensure that they remain visible within the viewcell. The viewcell is initialized at the center of the scene, positioned above the floor at a random height, and randomly rotated for variations.

We generate evaluation datasets using widely used scenes, such as Viking Village and Robot Lab, for comparison with previous work. For both the training and evaluation sets, we create pairs of a geometry grid and ground-truth PVS (Section 3.2).

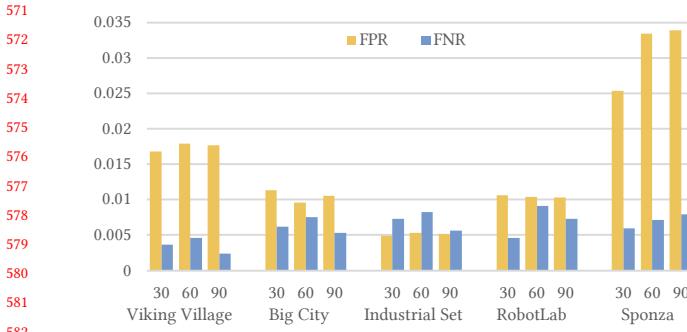
4.2 Training and evaluation

Our neural network implementation is based on OA-CNN [Pointcept Contributors 2023] (see supplementary material for details). For performance reasons, we use an aspect ratio of 1:1 for our viewport. Specifically, we test grid resolutions of $A \times A \times A$ with $A = 256$ and a viewcell size $r \in \{30 \text{ cm}, 60 \text{ cm}, 90 \text{ cm}\}$.

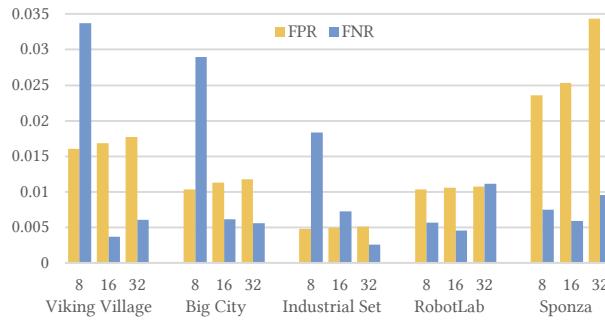
All training was performed on an Oracle Linux 7.9 server with 2 AMD EPYC 7662, 1 TB RAM, and 4 NVIDIA A100 SXM4 GPU; only one GPU was used for training. All evaluations were run on a desktop computer running Oracle Linux 9.5, equipped with AMD Ryzen 9 7900X, 64 GB RAM, and an NVIDIA RTX 5090 GPU. We optimized the model using $\lambda = 0.99$, with an initial learning rate 0.001, learning rate decay rate 10^{-10} , and batch size 3. We performed 200 epochs of training, which took approximately 15 hours for our setup. We used $\tau = 0.5$ throughout all experiments.

4.3 Rendering

We use Unity's Universal Rendering Pipeline (URP) version 17.0 for all rendering tasks. The main camera is configured with a resolution of 1024×1024 pixels, a field of view (FOV) of 60°, a near plane of 0.3 m, and a far plane of 1000 m. For the rendering of GV and PVS, we employ a wider FOV of 90° to account for up to 15° of camera rotation within a viewcell. To avoid visibility gaps, we render these buffers at a higher resolution of 2048×2048.



(a) Performance metrics for various viewcell size on different scenes. The horizontal axis is grouped by viewcell size $r \in 30\text{ cm}, 60\text{ cm}, 90\text{ cm}$ for each scene. Given that our network uses a fixed geometry grid size, performance remains relatively insensitive to changes in r .



(b) Froxel space performance metrics for various interleaving grid size d on different scenes. The horizontal axis is grouped by viewcell size $d \in 8, 16, 32$ for each scene. The performance is affected by changes in d ; setting $d = 16$ gives the overall best performance.

Fig. 5. Performance for different viewcell sizes and interleaving (Section 3.3) grid sizes. All metrics were averaged over the entire frame sequence.

The predicted view frustum spans up to a threshold (empirically chosen at 30 m). For distances larger than the threshold, we render the far-field scene geometry into a visibility buffer once when the PVS is created, and add the geometry indicated in the ID attachment of the visibility buffer to the PVS directly. This optimization allows the CNN to focus its predictive power in the near- and mid-ranges, where disocclusions are expected. The ground truth PVS is sampled using 1000 evenly distributed camera positions per viewcell.

5 Results

We evaluate our method on two indoor scenes (Sponza, Robot Lab) and three outdoor scenes (Viking Village, Big City, Industrial Set v3.0), as illustrated in Figure 8. For each scene, we render a 60-second animation at 60 Hz along a pre-recorded camera path, resulting in 3600 frames per scene.

5.1 Performance metrics

To evaluate the performance of the PVS estimation, we define the false negative rate $\text{FNR} = \text{FN}/\text{GTP}$ (\downarrow), the false positive rate



Fig. 6. (a) Average inference time (ms) and (b) average peak allocated memory (MB) for different interleaving factors $d = \{8, 16, 32\}$.

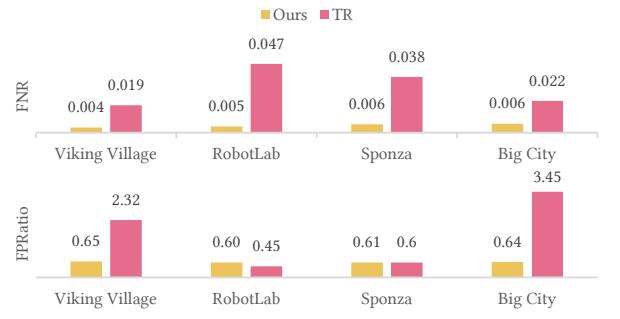


Fig. 7. Comparison of FNR and FPR (false positive as a multiple of the ground truth) between our results and baseline TR [Voglreiter et al. 2023]. In general, our method produces better FNR, similar or better FPR.

Table 1. Image space performance metrics SSIM for various interleaving grid size d on different scenes.

Scenes	SSIM \uparrow		
	d=8	d=16	d=32
Viking Village	0.9990	0.9996	0.9985
City	0.9991	0.9988	0.9972
Industrial	0.9963	0.9963	0.9996
Robot Lab	0.9985	0.9984	0.9995
Sponza	0.9998	0.9981	0.9913

FPR=FP/GTP (\downarrow) and the pixel error rate PER (\downarrow) for comparison with previous work. The pixel error rate reports the pixels showing incorrect primitives because of primitives missed in the PVS. It is computed as a fraction of the screen resolution and averaged over a sequence of frames through the scene. Moreover, we render shaded scenes from the PVS and the original geometry and compare the results using SSIM (\uparrow) [Wang et al. 2004], which considers local luminance, contrast, and structure over a sliding window.

Table 2. Comparison of time (ms), memory (MB) and pixel error rate (PER) between our method and Trim Regions [Voglerreiter et al. 2023]. "Ours" uses $r = 30$, $d = 16$ and "Ours*" uses $r = 30$, $d = 8$ to reduced memory. Pixel error comparison between our method and baseline TR [Voglerreiter et al. 2023]. Although the baseline has better PER, SSIM comparisons on the image confirm that the errors are minor and almost unnoticeable by humans.

Scene	ms↓		MB↓			PER/%↓		SSIM↑
	Ours	TR	Ours	Ours*	TR	TR	Ours	Ours
Viking	9.9	19.3	328.8	293.7	359.2	0.006	0.032	0.9996
Robot Lab	9.9	17.2	318.0	285.1	483.8	0.030	0.255	0.9984
Sponza	10.1	17.8	323.4	292.0	563.9	0.021	0.151	0.9981
City	10.3	16.4	333.5	295.3	330.5	0.006	0.142	0.9988



Fig. 8. The five test scenes used in our evaluation, shown with their respective primitive counts.

Figure 9 shows the metrics for an animated camera path of 3,600 frames in the Viking village scene. For keyframes such as the best and the worst case of FNR, the rendering results of the computed PVS are shown to better illustrate the performance of the method. As indicated by the metrics and the rendered images, our method is robust in handling different geometry distributions and occlusion relations in the complex scene, with even the worst case in this challenging scene exhibiting only a small amount of pixel errors. The corresponding SSIM value confirm our observation that the error is usually unnoticeable to humans.

Figure 5a shows the evaluation results on all scenes with different viewcell sizes r . We find that $r = 30$ cm gives the best performance in all scenes, while $r = 60$ cm has a slightly higher false negative rate. A potential explanation is that the size of the viewcell affects the subdivided grid shape in the geometry grid, where a too small grid size likely breaks the global geometry structure and causes the loss of global information, while a too large grid size makes it harder to learn the local structures within the grid.

Figure 5b shows the evaluation results on all scenes with different interleaving grid size d . With different d on the same scene, the performance can vary significantly. Choosing $d = 16$ gives the best performance in most scenes. With a too large interleaving factor d , too few elements remain after downsampling in the CNN, which may cause the network to overfit and has a negative effect on performance. For smaller d , more elements remain after interleaving; assuming the same training period, the false negative will be higher due to greater difficulties of converging.

Table 3. Ablation study on Viking Village. We use $r = 30$, $d = 16$. For "No OA-CNN" and "VNet", VNet [Milletari et al. 2016] is used instead of OA-CNN. Time and memory for "No RVLoss" is omitted because the loss does not affect the performance of inference.

	FNR ↓	FPR ↓	ms ↓	MB ↓
Ours	0.00368	0.01683	10.1	328.8
No Interleave	0.00227	0.01940	25.4	1076.2
No OA-CNN	0.01870	0.01677	19.0	509.8
No RVLoss	0.00000	0.61785	-	-
VNet	0.03235	0.01771	326.4	2135.9



Table 1 shows the SSIM of all scenes with different d . SSIM is always very close to 1, which indicates that the rendering quality is very good. Additional perceptual (FILP [Andersson et al. 2020]) and temporal metrics (VMAF [Li et al. 2016], CGVQM [Jindal et al. 2025]) are provided in the supplementary material, all confirming a similarly strong performance. Note that all our test scenes are completely unknown to the CNN, which is exclusively trained on purely synthetic scenes with simple, randomly generated geometries. Results show that our method is highly robust and generalizable to different and unknown data patterns while providing reliable estimation quality.

5.2 Speed and memory

We measure the runtime speed and memory in various scenes with different viewcell sizes. Section 4.3 presents the inference time per frame with the overall statistics and breakdown time per stage, including interleaving, CNN inference, and deinterleaving. The overall time per frame remains at approximately 10 ms (100 Hz), despite the scene scale and geometry variations. This makes our method more efficient than a naive depth pre-pass, which is 1.5 ms slower in our experiments after both methods are averaged between frames.

Following the argument of Voglerreiter et al. [2023], a running speed of 3 m/s translates into 5 cm/s camera movement at 60 Hz. At this speed of camera motion, the PVS is valid for 6 frames (100 ms), when $r=30$ cm, and, for 18 frames (300 ms) when $r=90$ cm. Amortized over these valid periods, the PVS inference only takes approximately 3.3% of the computation time for an application that generates new frames at a rate of 60 Hz. Therefore, we can assume that our method

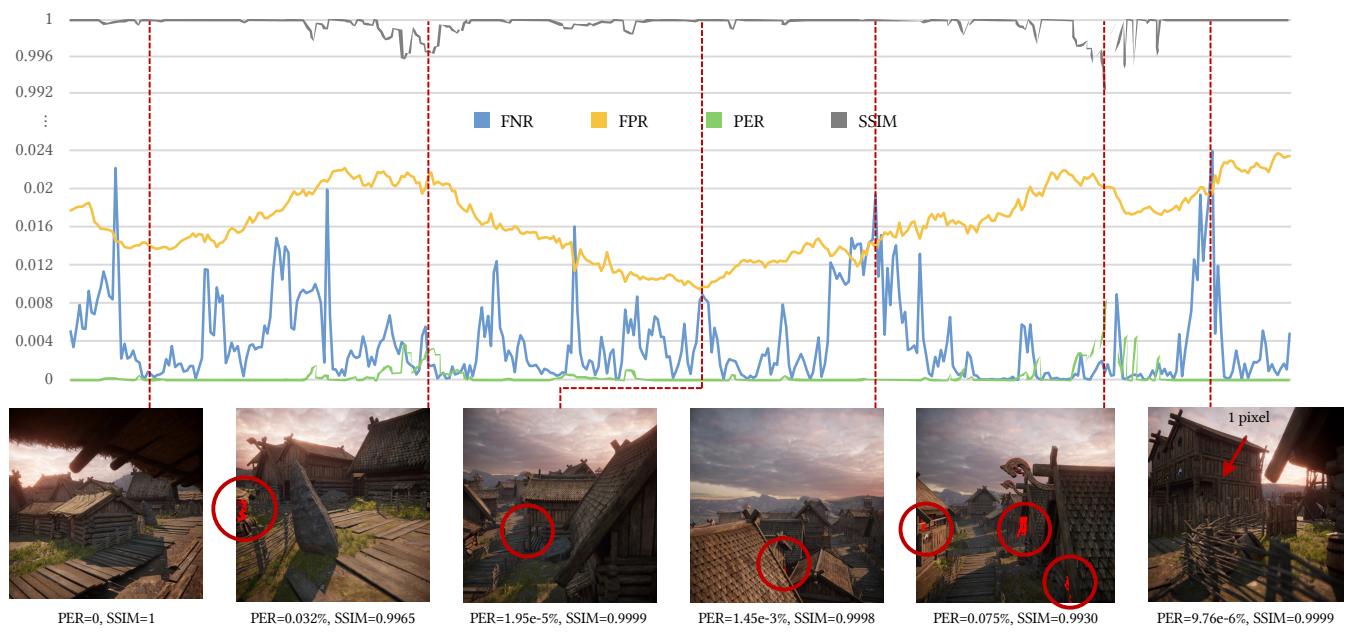


Fig. 9. Per-frame PVS estimation performance with key frames images of the Viking Village scene. The sequence has 1800 frames in total, with 410 frames of PVS computed shown in the figure. The error pixels of the key frames are marked in red on the rendered image.

is very well suited for frame extrapolation or streaming applications in terms of speed.

Figure 5 shows the time and peak memory allocated during inference. The adaptive relation convolution of OA-CNN will use more parameters if we add more feature channels, leading to increased inference time and memory usage when we use an interleaving factor of $d = 32$. Although memory requirements can vary depending on the occupancy rate of the input $G(x)$, the memory consumption is largely insensitive to scene geometry, as long as the viewcell configuration remains unchanged. The sparse tensor ensures that only the positions where geometry exists will be recorded in memory. Consequently, there is no memory overhead for empty grids.

5.3 Comparisons to the state of the art

To the best of our knowledge, the trim region method of Voglreiter et al. [2023] is the fastest to date for from-region PVS computation. We compare our results with those reported in the original paper. For a fair comparison, we run an experiment using the same GPU (NVIDIA RTX 4090) and the same scenes as ours, kindly provided by the authors. Figure 7 shows the comparisons of metrics in these scenes, namely, Viking Village, Robot Lab, Sponza, and Big City. Table 2 shows the speed and memory comparisons on these scenes.

Our method outperforms trim regions with respect to per-frame processing speed. The trim region method delivers 57 Hz (18 ms) per frame on average, while we achieve over 100 Hz (10 ms) on average (an improvement of 76%). We also obtain better quantitative performance, with significantly smaller false negatives and similar or better false positive rates. Compared to trim regions, our method has 83.8% less FNR and 63.4% less FPR on average. Although our

PER is slightly higher, the overall image quality is not noticeably affected, as shown by the SSIM column in Table 2.

5.4 Ablation studies

To evaluate the effectiveness of the OA-CNN network and the interleaving function, we carried out experiments on the classical CNN backbone VNet [Milletari et al. 2016] baseline and the original OA-CNN model with and without the interleaving function, as well as with and without RVL. Results are shown in Table 3.

The model trained without RVL has a significantly higher false positive rate, which means that it simply predicts (almost) the whole scene to be visible. The result shows that RVL is crucial in preventing the network from overfitting to the local minimum, and thus maintains a better balance between minimal FNR and reasonable FPR. Our results indicate that the interleaving function is the key to efficient inference. Its use increases the inference rate from 39 Hz to 100 Hz (2.5× speed-up) at 70% reduced memory usage.

The OA-CNN backbone is crucial for the performance of the network. Replacing OA-CNN with VNet leads to with 15% increased FNR, while the FPR is almost unaffected. OA-CNN also contributes to decreasing inference time and memory usage. Without the interleaving function and OA-CNN backbone, a completely naïve VNet network is too slow for real-time applications.

5.5 Challenging cases

To better understand the limitations of our method, we list some typical challenging cases in Figure 10. Like classical heuristic methods, the neural network also finds it harder to precisely estimate PVS in edge cases involving complex occlusion relationships (for example,

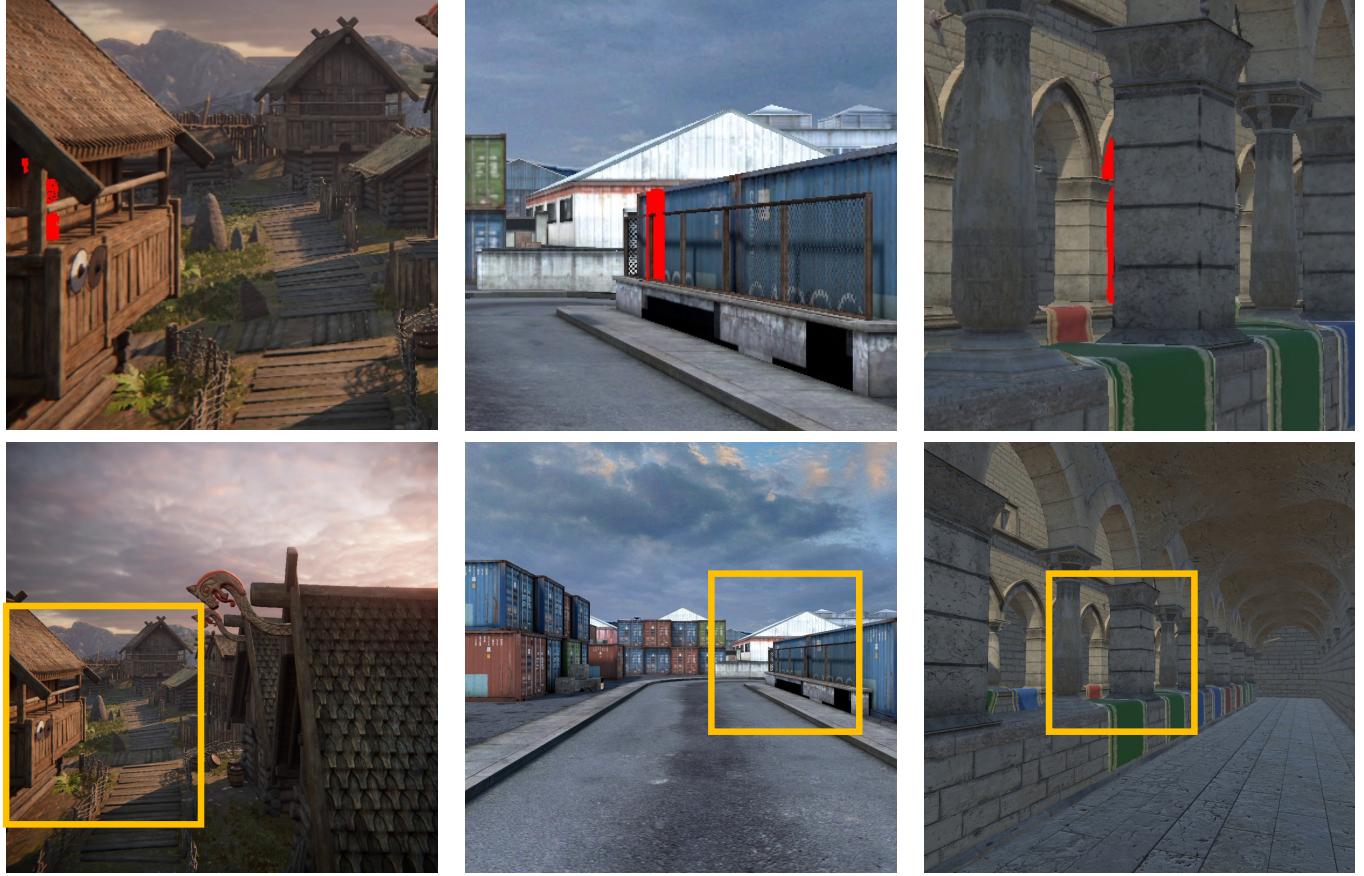


Fig. 10. Challenging situations for NeuralPVS model. left column: Multi-layer occlusion from Viking Village scene, 1104 false negative pixels. center column: complex occlusion structure from Industrial scene, 1152 false negative pixels. right column: almost occluded background from Sponza scene, 1102 false negative pixels.

multiple thin occluders in the left column). In the center column, the structure of the metal fence is not covered by our synthetic dataset and thus causes pixel errors. Further geometry has a lower resolution due to the geometry grid generation process, far-away geometry has a larger chance of getting misestimated, as shown in the right column. Moreover, machine learning methods are always dependent to some degree on the training data. While we found our randomized synthetic training set robust for a wide variety of scenes, fine-tuning our method for specific scenes can potentially help minimize errors.

5.6 Dynamic scenes

Since our view cell is defined purely in space, visibility changes caused by a dynamic object require special treatment. A simple approach unconditionally adds all dynamic objects to the PVS after visibility computation. However, this approach does not consider dynamic objects as occluders or occludees. While the first case – dynamic occluders – is likely too complicated to yield a speed-up, the second case – dynamic occludees – is rather simple to exploit. We construct a temporal bounding volume (TBV) for the moving object,

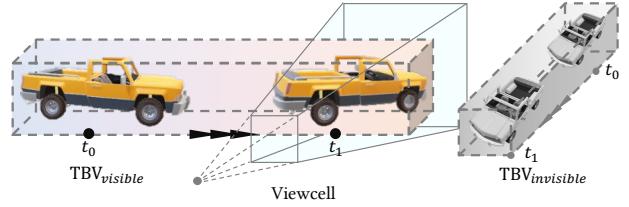


Fig. 11. Dynamics scenes visibility computation with temporal bounding volumes. The yellow vehicle is visible during the period of time t_0 to t_1 , while the grey vehicle is invisible.

which conservatively encloses the object during the predicted period (Figure 11). At runtime, the TBV is froxelized and tested against the froxelized PVS. If no froxel occupied by the TVB is deemed visible, the dynamic object can be pruned.

6 Conclusion and future work

We have presented a novel approach to compute from-regions potentially visible sets using a convolutional neural network. Our proposed method applies the OA-CNN network design with an

additional interleaving function and a repulsive visibility loss to perform end-to-end estimation of the PVS. Our method achieves high accuracy and fast inference.

We see several directions for future work. With the development of deep learning, adopting a more featured network design would potentially bring better accuracy, while balancing the computational cost could be challenging. Moreover, additional spatial data structures, such as octrees or hash grids, might bring about further improvements in both performance and speed. We also consider replacing the simple froxelization with a more advanced different feature representation of the local geometry. Finally, it is natural to expect some performance gain by introducing neural methods to other rendering tasks related to visibility, e.g. radiance transfer, shadow rendering, and global illumination.

Acknowledgements. This work was supported by the Alexander von Humboldt Foundation funded by the German Federal Ministry of Research, Technology and Space, the German Research Foundation DFG (grant 528364066, SFB TRR161), and the Austrian Science Fund FWF (grant I6663). The authors thank the International Max Planck Research School for Intelligent Systems (IMPRS-IS) for supporting Xiangyu Wang.

References

- John M. Airey, John H. Rohlff, and Frederick P. Brooks. 1990. Towards image realism with interactive update rates in complex virtual building environments. In *Proceedings of the 1990 Symposium on Interactive 3D Graphics - SI3D '90*. ACM Press, Snowbird, Utah, United States, 41–50. doi:10.1145/91385.91416
- Pontus Andersson, Jim Nilsson, Tomas Akenine-Möller, Magnus Oskarsson, Kalle Åström, and Mark D. Fairchild. 2020. FLIP: A Difference Evaluator for Alternating Images. *Proc. ACM Comput. Graph. Interact. Tech.* 3, 2 (2020), 15:1–15:23. doi:10.1145/3406183
- Anonymous. 2025. Title Withheld for Anonymous Review. Anonymous submission.
- Steve Bakó, Thijs Vogels, Brian McWilliams, Mark Meyer, Jan Novák, Alex Harvill, Pradeep Sen, Tony Deroose, and Fabrice Rousselle. 2017. Kernel-predicting convolutional networks for denoising Monte Carlo renderings. *ACM Transactions on Graphics* 36, 4 (Aug. 2017), 1–14. doi:10.1145/3072959.3073708
- Jiří Bittner, Oliver Mattausch, Peter Wonka, Vlastimil Havran, and Michael Wimmer. 2009. Adaptive global visibility sampling. *ACM Trans. Graph.* 28, 3 (2009), 94:1–94:10. doi:10.1145/1531326.1531400
- Jiří Bittner, Peter Wonka, and Michael Wimmer. 2005. Fast exact from-region visibility in urban scenes. In *Proceedings of the Sixteenth Eurographics Conference on Rendering Techniques (EGSR '05)*. Eurographics Association, Goslar, DEU, 223–230.
- Christopher A Burns and Warren A Hunt. 2013. The visibility buffer: a cache-friendly approach to deferred shading. *Journal of Computer Graphics Techniques (JCGT)* 2, 2 (2013), 55–69.
- Chakravarty R, Alla Chaitanya, Anton S. Kaplanyan, Christoph Schied, Marco Salvi, Aaron Lefohn, Derek Nowrouzezahrai, and Timo Aila. 2017. Interactive reconstruction of Monte Carlo image sequences using a recurrent denoising autoencoder. *ACM Trans. Graph.* 36, 4 (2017), 98:1–98:12. doi:10.1145/3072959.3073601
- Yukang Chen, Yanwei Li, Xiangyu Zhang, Jian Sun, and Jiaya Jia. 2022. Focal sparse convolutional networks for 3d object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 5428–5437. http://openaccess.thecvf.com/content_CVPR2022/html/Chen_Focal_Sparse_Convolutional_Networks_for_3D_Object_Detection_CVPR_2022_paper.html
- Christopher Choy, JunYoung Gwak, and Silvio Savarese. 2019. 4d spatio-temporal convnets: Minkowski convolutional neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 3075–3084. http://openaccess.thecvf.com/content_CVPR_2019/html/Choy_4D_Spatio-Temporal_ConvNets_Minkowski_Convolutional_Neural_Networks_CVPR_2019_paper.html
- D. Cohen-Or, Y.L. Chrysanthou, C.T. Silva, and F. Durand. 2003. A survey of visibility for walkthrough applications. *IEEE Transactions on Visualization and Computer Graphics* 9, 3 (July 2003), 412–431. doi:10.1109/TVCG.2003.1207447
- Daniel Cohen-Or, Gadi Fibich, Dan Halperin, and Eyal Zadicario. 1998. Conservative Visibility and Strong Occlusion for Viewspace Partitioning of Densely Occluded Scenes. *Computer Graphics Forum* 17, 3 (1998), 243–253. doi:10.1111/1467-8659.00271
- Angela Dai, Angel X. Chang, Manolis Savva, Maciej Halber, Thomas Funkhouser, and Matthias Nießner. 2017. Scannet: Richly-annotated 3d reconstructions of indoor scenes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 5828–5839. http://openaccess.thecvf.com/content_cvpr_2017/html/Dai_ScanNet_Richly-Annotated_3D_CVPR_2017_paper.html
- Stavros Diolatzis, Julien Philip, and George Drettakis. 2022. Active Exploration for Neural Global Illumination of Variable Scenes. *ACM Trans. Graph.* 41, 5 (2022), 171:1–171:18. doi:10.1145/3522735
- Yishun Dou, Zhong Zheng, Qiaoqiao Jin, Bingbing Ni, Yugang Chen, and Junxiang Ke. 2024. Real-Time Neural BRDF with Spherically Distributed Primitives. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 4337–4346. http://openaccess.thecvf.com/content_CVPR2024/html/Dou_Real-Time_Neural_BRDF_with_Spherically_Distributed_Primitives_CVPR_2024_paper.html
- Frédéric Durand, George Drettakis, and Claude Puech. 2002. The 3D visibility complex. *ACM Transactions on Graphics* 21, 2 (April 2002), 176–206. doi:10.1145/508357.508362
- Epic Games. 2022. Precomputed Visibility Volumes. <https://docs.unrealengine.com/5.1/en-US/precomputed-visibility-volumes-in-unreal-engine/>
- Alex Evans. 2015. Learning from Failure: a Survey of Promising, Unconventional and Mostly Abandoned Renderers for ‘Dreams PS4’, a Geometrically Dense, Painterly UGC Game’. <https://advances.realtimerendering.com/s2015/>
- Benjamin Graham and Laurens van der Maaten. 2017. Submanifold Sparse Convolutional Networks. doi:10.48550/arXiv.1706.01307 arXiv:1706.01307 [cs].
- Ned Greene, Michael Kass, and Gavin Miller. 1993. Hierarchical Z-buffer visibility. In *Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '93)*. Association for Computing Machinery, New York, NY, USA, 231–238. doi:10.1145/166117.166147
- Jon Hasselgren, Magnus Andersson, and Tomas Akenine-Möller. 2016. Masked Software Occlusion Culling. In *Eurographics/ ACM SIGGRAPH Symposium on High Performance Graphics*. Ulf Assarsson and Warren Hunt (Eds.). The Eurographics Association. doi:10.2312/hpg.20161189 ISSN: 2079-8679.
- Jozef Hladky, Hans-Peter Seidel, and Markus Steinberger. 2019. The camera offset space: real-time potentially visible set computations for streaming rendering. *ACM Trans. Graph.* 38, 6 (2019), 231:1–231:14. doi:10.1145/3355089.3355530
- Jozef Hladky, Michael Stengel, Nicholas Vining, Bernhard Kerbl, Hans-Peter Seidel, and Markus Steinberger. 2022. QuadStream: A Quad-Based Scene Streaming Architecture for Novel Viewpoint Reconstruction. *ACM Transactions on Graphics* 41, 6 (Dec. 2022), 1–13. doi:10.1145/3550454.3555524
- Bingyang Hu, Jie Guo, Yanjun Chen, Mengtian Li, and Yanwen Guo. 2020. DeepBRDF: A Deep Representation for Manipulating Measured BRDF. *Computer Graphics Forum* 39, 2 (2020), 157–166. doi:10.1111/cgf.13920
- Akshay Jindal, Nabil Sadaka, Manu Mathew Thomas, Anton Sochenov, and Anton Kaplanyan. 2025. CGVQM-D: Computer Graphics Video Quality Metric and Dataset. doi:10.48550/arXiv.2506.11546 arXiv:2506.11546 [cs].
- Janghun Kim and Sungkil Lee. 2023. Potentially Visible Hidden-Volume Rendering for Multi-View Warping. *ACM Transactions on Graphics* 42, 4 (2023). doi:10.1145/3592108
- Thomas Koch and Michael Wimmer. 2021. Guided Visibility Sampling++. *Proceedings of Computer Graphics and Interactive Techniques* 4, 1 (2021). doi:10.1145/3451266 Place: New York.
- Vladlen Koltun, Yiorgos Chrysanthou, and Daniel Cohen-Or. 2000. Virtual Occluders: An Efficient Intermediate PVS representation. In *Rendering Techniques 2000*, Bernard Péroche and Holly Rushmeier (Eds.). Springer, Vienna, 59–70. doi:10.1007/978-3-7091-6303-0_6
- Zhi Li, Anne Aaron, Ioannis Katsavounidis, Anush Moorthy, and Megha Manohara. 2016. Toward a practical perceptual video quality metric, 2016. *Netflix Techblog* (2016). <https://netflixtechblog.com/toward-a-practical-perceptual-video-quality-metric-653f208b9652>
- Edward Liu. 2020. DLSS 2.0 – Image Reconstruction for Real-Time Rendering with Deep Learning. <https://developer.download.nvidia.com/video/gputechconf/gtc/2020/presentations/s22698-dlss-image-reconstruction-for-real-time-rendering-with-deep-learning.pdf> Published: GPU Technology Conference (Session S22698), NVIDIA.
- Oliver Mattausch, Jiří Bittner, and Michael Wimmer. 2006. *Adaptive Visibility-Driven View Cell Construction*. The Eurographics Association. <https://doi.org/10.2312/EGWR/EGSR06/195-205> ISSN: 1727-3463.
- Fausto Milletari, Nassir Navab, and Seyed-Ahmad Ahmadi. 2016. V-Net: Fully Convolutional Neural Networks for Volumetric Medical Image Segmentation. In *International Conference on 3D Vision*. arXiv, 565–571. doi:10.1109/3DV.2016.79 shortConferenceName: 3DV.
- Joerg H. Mueller, Philip Vogelreiter, Mark Dokter, Thomas Neff, Mina Makar, Markus Steinberger, and Dieter Schmalstieg. 2018. Shading Atlas Streaming. *ACM Transactions on Graphics* 37, 6 (Dec. 2018). doi:10.1145/3272127.3275087 Place: New York Publisher: Association for Computing Machinery.
- O. Nalbach, E. Arabadzhyska, D. Mehta, H.-P. Seidel, and T. Ritschel. 2017. Deep Shading: Convolutional Neural Networks for Screen Space Shading. *Computer Graphics Forum* 36, 4 (2017), 65–78. doi:10.1111/cgf.13225
- S. Nirenstein and E. Blake. 2004. *Hardware Accelerated Visibility Preprocessing using Adaptive Sampling*. The Eurographics Association. <https://doi.org/10.2312/EGWR/EGSR04/207-216> ISSN: 1727-3463.

- 1141 Bohao Peng, Xiaoyang Wu, Li Jiang, Yukang Chen, Hengshuang Zhao, Zhuotao Tian,
 1142 and Jiaya Jia. 2024. OA-CNNs: Omni-Adaptive Sparse CNNs for 3D Semantic Segmentation. In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, Seattle, WA, USA, 21305–21315. doi:10.1109/CVPR52733.2024.02013
- 1144 Pointcept Contributors. 2023. Pointcept: a codebase for point cloud perception research.
 1145 <https://github.com/Pointcept/Pointcept>
- 1146 Bernhard Reinert, Johannes Kopf, Tobias Ritschel, Eduardo Cuervo, David Chu, and
 1147 Hans-Peter Seidel. 2016. Proxy-guided Image-based Rendering for Mobile Devices.
Computer Graphics Forum 35, 7 (Oct. 2016), 353–362. doi:10.1111/cgf.13032
- 1148 Haocheng Ren, Yuchi Huo, Yifan Peng, Hongtao Sheng, Weidong Xue, Hongxiang
 1149 Huang, Jingzhen Lan, Rui Wang, and Hujun Bao. 2024. LightFormer: Light-Oriented
 1150 Global Neural Rendering in Dynamic Scene. *ACM Trans. Graph.* 43, 4 (2024), 75:1–
 1151 75:14. doi:10.1145/3658229
- 1152 Gernot Riegler, Ali Osman Ulusoy, and Andreas Geiger. 2017. Octnet: Learning deep 3d
 1153 representations at high resolutions. In *Proceedings of the IEEE Conference on Computer
 Vision and Pattern Recognition*. 3577–3586. http://openaccess.thecvf.com/content_cvpr_2017/html/Riegler_OctNet_Learning_Deep_CVPR_2017_paper.html
- 1154 Gernot Schaufler, Julie Dorsey, Xavier Decoret, and Francois X. Sillion. 2000. Conservative
 1155 volumetric visibility with occluder fusion. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '00)*. ACM Press/Addison-Wesley Publishing Co., USA, 229–238. doi:10.1145/344779.344886
- 1156 Spconv Contributors. 2022. Spconv: Spatially Sparse Convolution Library. <https://github.com/traveller59/spconv>
- 1157 Alejandro Sztrajman, Gilles Rainer, Tobias Ritschel, and Tim Weyrich. 2021. Neural
 1158 BRDF Representation and Importance Sampling. *Computer Graphics Forum* 40, 6
 1159 (2021), 332–346. doi:10.1111/cgf.14335
- 1160 Abdel Aziz Taha and Allan Hanbury. 2015. Metrics for evaluating 3D medical image
 1161 segmentation: analysis, selection, and tool. *BMC Medical Imaging* 15, 1 (Aug. 2015),
 1162 29. doi:10.1186/s12880-015-0068-x
- 1163 Zhipeng Tan, Yongxiang Zhang, Fei Xia, and Fei Ling. 2025. Differentiable Rendering
 1164 based Part-Aware Occlusion Proxy Generation. *Computer Graphics Forum* 44, 2
 (2025), e70077. doi:10.1111/cgf.70077
- 1165
- 1166
- 1167
- 1168
- 1169
- 1170
- 1171
- 1172
- 1173
- 1174
- 1175
- 1176
- 1177
- 1178
- 1179
- 1180
- 1181
- 1182
- 1183
- 1184
- 1185
- 1186
- 1187
- 1188
- 1189
- 1190
- 1191
- 1192
- 1193
- 1194
- 1195
- 1196
- 1197
- 1198 Lyne Tchapmi, Christopher Choy, Iro Armeni, JunYoung Gwak, and Silvio Savarese.
 1199 2017. Segcloud: Semantic segmentation of 3d point clouds. In *2017 International Conference on 3D Vision (3DV)*. IEEE, 537–547. <https://ieeexplore.ieee.org/abstract/document/8374608/>
- 1200 Seth J. Teller and Carlo H. Séquin. 1991. Visibility preprocessing for interactive walk-
 1201 throughs. In *Proceedings of the 18th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '91)*. Association for Computing Machinery, New York, NY, USA, 61–70. doi:10.1145/122718.122725
- 1202 Philip Voglreiter, Bernhard Kerbl, Alexander Weinrauch, Joerg Hermann Mueller,
 1203 Thomas Neff, Markus Steinberger, and Dieter Schmalstieg. 2023. Trim Regions for Online Computation of From-Region Potentially Visible Sets. *ACM Transactions on Graphics* 42, 4 (Aug. 2023), 1–15. doi:10.1145/3592434
- 1204 Xinlong Wang, Tete Xiao, Yuning Jiang, Shuai Shao, Jian Sun, and Chunhua Shen. 2018.
 1205 Repulsion Loss: Detecting Pedestrians in a Crowd. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. IEEE, Salt Lake City, UT, 7774–7783. doi:10.1109/CVPR.2018.00811
- 1206 Zhou Wang, A.C. Bovik, H.R. Sheikh, and E.P. Simoncelli. 2004. Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing* 13, 4 (April 2004), 600–612. doi:10.1109/TIP.2003.819861
- 1207 Peter Wonka, Michael Wimmer, and Dieter Schmalstieg. 2000. Visibility Preprocessing with Occluder Fusion for Urban Walkthroughs. In *Rendering Techniques 2000*, Bernard Péroche and Holly Rushmeier (Eds.). Springer, Vienna, 71–82. doi:10.1007/978-3-7091-6303-0_7
- 1208 Peter Wonka, Michael Wimmer, and Francois X. Sillion. 2001. Instant Visibility. *Computer Graphics Forum* 20, 3 (2001), 411–421. doi:10.1111/1467-8659.00534
- 1209 Lei Xiao, Anton Kaplanyan, Alexander Fix, Matthew Chapman, and Douglas Lanman. 2018. DeepFocus: learned image synthesis for computational displays. *ACM Transactions on Graphics* 37, 6 (Dec. 2018), 1–13. doi:10.1145/3272127.3275032
- 1210 Chuankun Zheng, Yuchi Huo, Hongxiang Huang, Hongtao Sheng, Junrong Huang,
 1211 Rui Tang, Hao Zhu, Rui Wang, and Hujun Bao. 2024. Neural Global Illumination via Superposed Deformable Feature Fields. In *SIGGRAPH Asia 2024 Conference Papers (SA '24)*. Association for Computing Machinery, New York, NY, USA, 1–11. doi:10.1145/3680528.3687680
- 1212
- 1213
- 1214
- 1215
- 1216
- 1217
- 1218
- 1219
- 1220
- 1221
- 1222
- 1223
- 1224
- 1225
- 1226
- 1227
- 1228
- 1229
- 1230
- 1231
- 1232
- 1233
- 1234
- 1235
- 1236
- 1237
- 1238
- 1239
- 1240
- 1241
- 1242
- 1243
- 1244
- 1245
- 1246
- 1247
- 1248
- 1249
- 1250
- 1251
- 1252
- 1253
- 1254