A Refined Path Generation Pipeline for Radio Channel Propagation Modeling

Niklas Vaara*, Pekka Sangi*, Juha Pyhtilä†, Markku Juntti† and Janne Heikkilä*
*Center for Machine Vision and Signal Analysis
†Centre for Wireless Communcations - Radio Technologies
Faculty of Information Technology and Electrical Engineering, University of Oulu
Oulu, Finland
Email: niklas.vaara@oulu.fi

***Abstract*—Ray tracing is a widely used approach for deterministic modelling of radio channels. We present our path generation pipeline, which combines environment discretization-based propagation path search with path refinement, which outputs validated paths fulfilling the Fermat's principle of the least time. We propose a novel gradient descent-based solution for refinement. Whole pipeline is implemented as GPU computations using NVIDIA CUDA and OptiX ray tracing engine, and experimental results show efficacy of the approach.**

***Index Terms*—Ray tracing, ray launching, path optimization, GPU computing**

I. Introduction

Ray tracing (RT) is a reliable and deterministic state-of-the-art method for radio channel propagation modeling. Approximating the features of the radio channel is useful in many contexts, such as helping in the design of beamforming and multiple-input multiple-output systems, where the performance is determined by multidispersive characteristics of the radio channel [1], [2].

Recent interest has been in machine learning (ML) and artificial intelligence solutions for green and efficient communication networks, as the rapidly increasing traffic makes network modeling and monitoring difficult [3]. As ML models require a lot of training data, the results of RT-based radio channel modeling simulations become an attractive alternative to data generated by time consuming measurements.

Although faster than measurements, the time complexity of RT-based simulations is still high, as the number of rays grows exponentially based on the number of interactions. Performing the RT on a graphics processing unit (GPU) has been widely adopted due to the parallel nature of RT [4]. Even with a GPU-based RT solution, the process is still time consuming. Thus, on top of the GPU-side code optimization, other approaches have to be considered to reduce the time complexity, often with the cost of reducing the accuracy of the simulation.

The simulation accuracy is determined by multiple factors such as the precision of the model and its materials, and the chosen path search method. For diffuse interactions, ray launching (RL) implementations such as [5] utilize angular discretization. This results in angular dispersion, which can be reduced by an approach called ray splitting, where the rays are split into multiple new rays when a spatial separation threshold is reached [6], [7]. The drawback is the additional computing required by the new rays. Another path searching approach is tile-based environment discretization [8], [9], where each discretized element is considered when tracing the paths. In this approach the accuracy is purely determined by the discretization resolution, and has the advantage of not missing important paths caused by angular dispersion. The downside is that approximation of the geometry may be required in order to be suitable for the tile-based discretization.

As the paths generated by RL are coarse approximations, an approach known as the image method is often applied to the generated paths to refine the accuracy of the paths [10], [11]. Optimal paths fulfill the Fermat's principle of the least time, from which the Snell's law and the law of diffraction can be derived [12]. Reflections are calculated as shown in [4] and diffractions can be computed by solving systems of nonlinear equations as presented in [13].

In this paper, we present our path generation pipeline where discretization-based path search is combined with a path refinement step. As a basis for path search we use the approach proposed by Lu *et al.* [9], which is implemented using NVIDIA CUDA [14] and OptiX RT engine [15]. Complementary path refinement and validation stages are also implemented as GPU computations. Especially, instead of applying the image method for refinement, we present a straightforward gradient descent-based solution for path optimization. The results suggest that the quality of coarse paths can be improved with this approach.

II. Description of the pipeline

The path generation pipeline consists of input preparation, coarse path search, path refinement, and validation. The coarse path search algorithm is based on DED-RL by Lu *et al.* [9], which is an efficient GPU-based RL algorithm for urban areas. The walls are discretized into tiles and edge segments to speed up the RL process, allowing simple ray collision tests. Another important reason for speedup, especially for diffraction edges and transmitters (TXs), is that the rays are only cast in relevant directions. The whole pipeline is written in C++ and CUDA. For ray tracing we use the OptiX RT engine. The coarse path search, path refinement and path validation stages exploit GPU for fast computation. Further details can be found in [16].

A. Input Preparation

All of the components use an internal format for the geometry representation. The internal format contains information about vertices, indices, materials, walls, and diffraction edges. A tool was developed which analyzes diffraction edges based on a given angle threshold as in [5] and walls from glTF 2.0 [17] models and converts the relevant data to the internal format.

Once the scene is in the internal format, the walls and edges are discretized based on the given tile size and edge segment length similarly to the approach used in [9]. Additionally, TXs and receivers (RXs) are placed into the scene.

B. Coarse Path Search

The coarse path search consists of visibility analysis and propagation stages. These stages are executed for each TX in the input.

1) Visibility Analysis: In the visibility analysis stage the visibilities between tiles are determined and saved into an $N \times N$ symmetric matrix, where N denotes the number of tiles in the geometry [9]. In our implementation, we allocate memory for the elements above the main diagonal of the matrix, where each visibility is stored in a single bit. A ray is cast from each tile to every other tile using OptiX, and the result is written into the matrix based on the respective tile indices. This so-called first-level visibility analysis (FLVA) is also applied to other kinds of point pairs, like tile-edge and edge-edge pairs.

2) Propagation Principle: During the propagation stage the visibility analysis is based on the implementation by Lu *et al.* [9]. The visibility determination consists of two tests, FLVA matrix query and second-level visibility analysis (SLVA). First, the result of FLVA is queried with the respective indices to find out if visibility between the two discretized elements exists. If visibility is found, SLVA is performed. For reflections, SLVA is determined by back-projecting the tile center point to the surface of the spawning tile. If the projected point is inside the spawning tile, the tile is accepted. For diffractions, two Keller's cones [12] are constructed based on the rays cast to both ends of the diffraction edge segment. The volume between these two cones is used to determine whether the discretized element is visible or not. The SLVA for reflection and diffraction is illustrated in Fig. 1(a).

The validity of each interaction is determined by the FLVA result query, SLVA, and a chosen field intensity threshold. Intensity of the complex field vector $\mathbf{E} = [E_\phi, E_\theta]^T$ is calculated with the equation

$$E_m(\mathbf{E}) = 10 \log_{10} \left(|E_\phi|^2 + |E_\theta|^2 \right) \text{ [dB (1 V}^2\text{/m}^2\text{)]}. \quad (1)$$

3) Propagation Implementation: A method called chaining was developed to achieve improved performance. Reflections, diffractions and propagation of rays to RX points are handled in their respective kernels to reduce the divergence. Additionally, each kernel invocation only handles one interaction and each thread handles only one discretized element, resulting in a high GPU occupancy.

The chaining process is divided into transmission and propagation kernels. The propagation starts at depth level 0, where the level denotes the number of interactions. If the depth level is 0, the transmission kernels are used, which determine the initial interactions and line of sight (LOS) connections in the respective kernels. If the interaction is valid, it is added to the parent buffer of the next depth level. Each level will call the RX, reflection, and diffraction kernels in the aforementioned order, which forms the launching logic, as illustrated in Fig. 1(b).

The high level chaining code runs in 32 contexts on the GPU, as it is possible to exploit CUDA streams for asynchronous launching. GPU side launching removes the need for memory transfers and allows direct access to the GPU memory.

Propagation kernels are launched for each interaction, where the total number of relevant discretized elements forms the launch dimensions. Those are resolved dynamically. Each invocation in the launch handles one discretized element, and performs the FLVA result query, SLVA, electromagnetic (EM) calculations, and lastly, checks if the field intensity is above the chosen threshold. If all the conditions are met, the path is added to the parent buffer of the next depth level. The parent buffers should have a decent size. However, the performance is not largely affected by the size as no central processing unit (CPU) synchronizations are needed when they become full.

A buffer of a chosen size is allocated for each RX point, where the valid paths are stored. When the path buffer becomes full, synchronization with the CPU is required. The processing time needed by the synchronization was minimized by implementing a double buffering approach, where the path buffer is divided into two buffers. When CPU synchronization occurs, the active buffer pointer is switched. A thread is then allocated from a thread pool, which will asynchronously download the filled path buffer from the GPU.

The CPU path processing thread calculates a hash for each path to avoid duplicate paths as in [18]. A hash map is used for storing the hashes, which is queried for each hash to check if the path already exists. If an existing path is not found, the hash is inserted into the hash map.

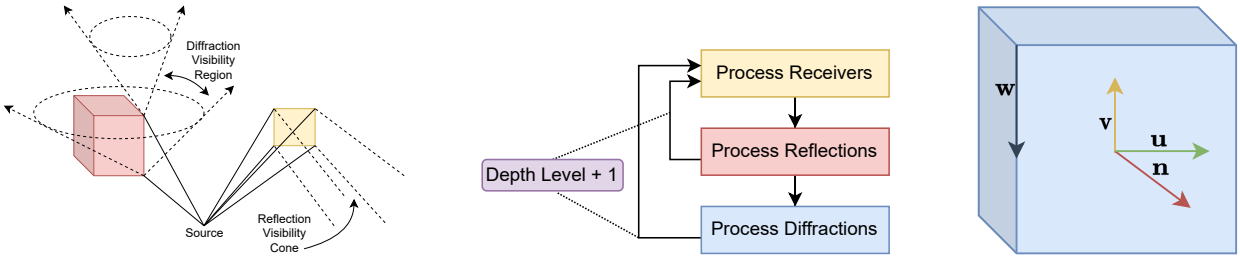
C. Path Refinement and Validation

1) Gradient Descent: Gradient descent algorithm minimizes a function $f(\mathbf{a})$ by updating the parameters in the direction of the function's negative gradient, which is represented by the equation [19]

$$\mathbf{a}_{n+1} = \mathbf{a}_n - \gamma_n \nabla f(\mathbf{a}_n), \; n \geq 0 \quad (2)$$

where n represents the current iteration index, γ is the step size, $\mathbf{a}$ denotes the vector of parameters, and $\nabla f(\mathbf{a})$ represents the gradient of the function that we aim to minimize.

In order to fulfill the Fermat's principle of least time, we are interested in finding the paths that takes the shortest propagation time. As the currently supported interaction types in the coarse path search are reflection or diffraction, we do not consider the index of refraction in the equation due to the



(a) Second-level visibility analysis of reflections and diffractions.

(b) Launching logic of chaining.

(c) Vectors $\mathbf{u}$, $\mathbf{v}$, and $\mathbf{w}$ used in the gradient descent algorithm.

Fig. 1. Illustrations of the pipeline.

homogeneity of the medium. Thus, the length of a path can be represented by the following function

$$f_{path}(\mathbf{x}) = \sum_{k=0}^{N} \|\mathbf{I}_{k+1} - \mathbf{I}_k\|, \tag{3}$$

where N is the number of interactions and $\mathbf{I}_k$ is the notation for the function of the interaction type. The points $\mathbf{I}_0$ and $\mathbf{I}_{N+1}$ correspond to the TX and RX positions respectively. In the case of N_r reflections and N_d diffractions, the vector $\mathbf{x}$ can be represented as

$$\mathbf{x} = [r_1, \ldots, r_{N_r}, s_1, \ldots, s_{N_r}, t_1, \ldots, t_{N_d}]^T, \tag{4}$$

where the elements r_i and s_i represent the variables of reflections, and t_i denotes the variable of diffractions.

In order to minimize the path length at $\mathbf{I}_k$, the interaction point $\mathbf{I}_k$ has to consider the source point $\mathbf{I}_{k-1}$ and the destination point $\mathbf{I}_{k+1}$. Thus, the path length from $\mathbf{I}_{k-1}$ to $\mathbf{I}_{k+1}$ via $\mathbf{I}_k$ is represented by the following function

$$f_k(\mathbf{x}) = \|\mathbf{I}_{k+1} - \mathbf{I}_k\| + \|\mathbf{I}_k - \mathbf{I}_{k-1}\|. \tag{5}$$

Each interaction point is updated with (2) iteratively until the maximum iteration or convergence threshold is reached. The converged path found with the gradient descent represents a unique global minimum, as the minimization for boundless planes and straight edges is convex.

2) Reflections: In the case of reflections, we have the surface point $\mathbf{p}$ and orthonormal vectors $\mathbf{u}$ and $\mathbf{v}$ as in Fig. 1(c). A reflection point $\mathbf{R}_i$ on the plane can be represented as

$$\mathbf{R}_i = \mathbf{p}_i + r_i\mathbf{u}_i + s_i\mathbf{v}_i, \tag{6}$$

where i denotes the ith reflection, and the variables r_i and s_i determine the interaction point position on the plane. $\mathbf{R}_i$ can be interpreted as $\mathbf{I}_k$ if the kth interaction is a reflection. Thus, we need to differentiate the function f_k with respect to the aforementioned variables. We get the following functions

$$f_{r_i} = \left(\frac{(\mathbf{I}_k - \mathbf{I}_{k+1})}{\|\mathbf{I}_k - \mathbf{I}_{k+1}\|} + \frac{(\mathbf{I}_k - \mathbf{I}_{k-1})}{\|\mathbf{I}_k - \mathbf{I}_{k-1}\|} \right) \cdot \mathbf{u}_i \tag{7}$$

and

$$f_{s_i} = \left(\frac{(\mathbf{I}_k - \mathbf{I}_{k+1})}{\|\mathbf{I}_k - \mathbf{I}_{k+1}\|} + \frac{(\mathbf{I}_k - \mathbf{I}_{k-1})}{\|\mathbf{I}_k - \mathbf{I}_{k-1}\|} \right) \cdot \mathbf{v}_i, \tag{8}$$

where $f_{r_i} = \partial f_k / \partial r_i$ and $f_{s_i} = \partial f_k / \partial s_i$. For (2), we can now form the vectors $\mathbf{a} = [r_i, s_i]^T$ and $\nabla f(\mathbf{a}) = [f_{r_i}, f_{s_i}]^T$.

3) Diffractions: For diffractions, we have the surface point $\mathbf{p}$ on the edge and the normalized direction vector $\mathbf{w}$ along the edge, as illustrated in Fig. 1(c). The diffraction points can be represented with the equation

$$\mathbf{D}_i = \mathbf{p}_i + t_i\mathbf{w}_i, \tag{9}$$

where i denotes the ith diffraction and the variable t_i determines the position of the interaction on the edge. $\mathbf{D}_i$ can be interpreted as $\mathbf{I}_k$ if the kth interaction is a diffraction. Thus, the function for updating diffraction points is

$$f_{t_i} = \left(\frac{(\mathbf{I}_k - \mathbf{I}_{k+1})}{\|\mathbf{I}_k - \mathbf{I}_{k+1}\|} + \frac{(\mathbf{I}_k - \mathbf{I}_{k-1})}{\|\mathbf{I}_k - \mathbf{I}_{k-1}\|} \right) \cdot \mathbf{w}_i, \tag{10}$$

where $f_{t_i} = \partial f_k / \partial t_i$. Thus, for (2), we get $\mathbf{a} = t_i$ and $\nabla f(\mathbf{a}) = f_{t_i}$.

4) Step Size Determination: The step size γ_n in (2) is calculated in each iteration for each interaction. We use backtracking line search algorithm to calculate γ_n which is determined once the following condition is satisfied [20]

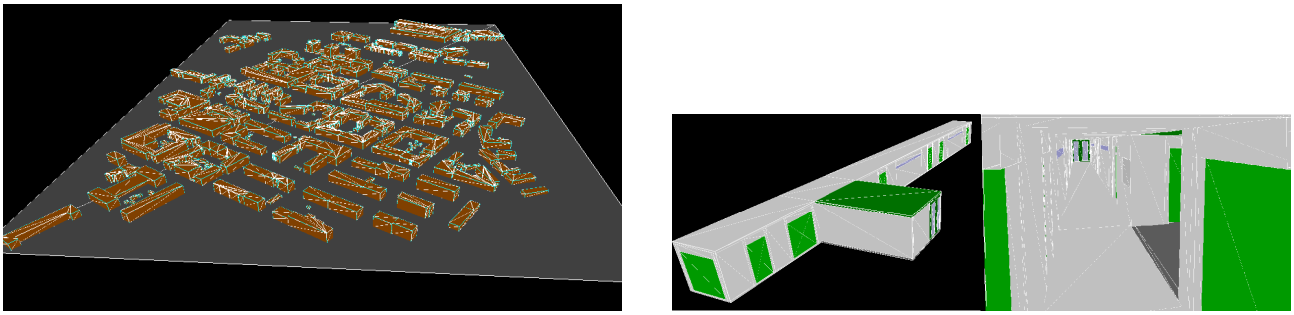
$$f(\mathbf{x}_n) - f(\mathbf{x}_n + \gamma_j(-\nabla f(\mathbf{x}_n))) \geq \gamma_j t, \tag{11}$$

where

$$t = c\|\nabla f(\mathbf{x}_n)\|^2, \ 0 < c < 1. \tag{12}$$

The step size determination is started with an initial γ_j of 1, which is multiplied with τ $(0 < \tau < 1)$ in each iteration until (11) is satisfied. Once the condition is satisfied, the current γ_j is chosen as the step size γ_n.

5) Path Validation: The path refinement algorithm assumes infinite planes and edges, which might result in the interaction points moving outside of the wall and edge bounds. Another problem is that the rays along the path might be occluded after the refinement. Additionally, RL techniques that utilize discretization and cones or similar volumetric ray-based methods for determining valid interactions may suffer from a bouncing problem. In some situations, SLVA may allow a ray to bounce between two discretization points. When such a path does not make up a physically valid route, refinement and validation can prune it.



(a) The outdoor scene.

(b) The indoor scene.

Fig. 2. The scenes used in the experiments.

We use OptiX for the RT. For each interaction point along the path, we shoot a ray to the next point with a small bias added along the direction vector to avoid collision with the surface of the source point. Lastly, for all the valid paths, we perform the EM calculations.

III. Experiments

A. Overview

The experiments were conducted with outdoor and indoor scenes of real locations. The outdoor case shown in Fig. 2(a) is a model of Etu-Lyötty area located near the downtown of Oulu. The indoor scene (Fig. 2(b)) is a model of one corridor at the University of Oulu. The PC platform used in the experiments has an Intel i5-2500k CPU and an NVIDIA GeForce GTX 1070 GPU.

The experiments consist of two separate sections. In the first section, coarse path search experiments are conducted for both the indoor and outdoor model with varying parameters. In the second section, the path refinement experiments are conducted with the resulting paths of the coarse path search experiments.

B. Coarse Path Search

Path buffer size of 5000 was chosen, which is allocated for each RX point. The chosen size is used in both the indoor and outdoor scene experiments. For both scenes, the resulting number of paths can be found in Fig. 3(a) and the execution times in Fig. 3(b).

1) Outdoor Scene: The outdoor scene benchmarks were conducted with $5 \times 5\text{m}^2$ discretization for walls and edges, and $10 \times 10\text{m}^2$ discretization for the ground plane, which resulted in 7389 tiles and 3239 diffraction edge segments. One 1W isotropic TX with 1.4GHz carrier frequency and a regular RX grid of 719 points were inserted into the scene. We used -100dB threshold for the field vector intensity.

2) Indoor Scene: The indoor scene benchmarks were conducted with $1 \times 1\text{m}^2$ discretization for walls which resulted in 611 tiles and 245 diffraction edge segments. One 1W isotropic TX with 110GHz carrier frequency and a regular RX grid of 53 points were inserted into the scene. A -100dB threshold for the field vector intensity was used in the indoor scene as well.

3) Discussion: One interesting scenario seen in Fig. 3(a) and 3(b) is that indoor scene performs worse than the outdoor scene with the limits of 5 interactions and 0 diffractions, even though the number of discretized elements in the indoor scene is evidently smaller. This is caused by the bouncing problem described in Section II-C5 and duplicate paths. Another factor is that in CUDA, each stream has implicit synchronization. This means that all kernels launched within CUDA stream are guaranteed to be executed in the launching order [21], causing additional overhead.

C. Path Refinement

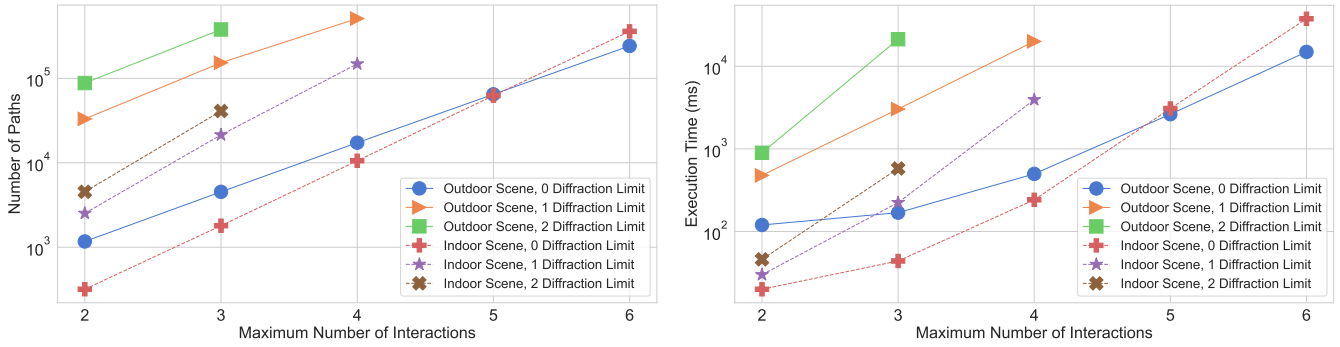
The indoor and outdoor scene path refinement experiments were conducted with c and τ of 0.6, and convergence threshold of 1e-5. Additionally, if both of the sides in (11) are below 1e-4, the current iteration γ_j is chosen as γ_n for (2).

The outdoor scene had 104 LOS paths and the indoor scene had 7. Only non-line of sight (NLOS) paths are considered in Table I. In the aforementioned table, outdoor is abbreviated as OD and indoor as ID, #I stands for the maximum number of interactions, #D denotes the maximum number of diffractions, #P represents the number of NLOS paths before refinement, C stands for NLOS converged paths, and V denotes validated NLOS converged paths.

In the outdoor experiments, 1.7–89.8% (on average 38.7%) of NLOS paths converged. Out of the converged paths 11.7–78.2% (on average 44.6%) remained after validation. In the indoor experiments, 0.8–75.7% (on average 34.5%) of the NLOS paths converged, and 22.0–83.1% (on average 53.8%) of these converged paths passed the validation.

TABLE I
Path refinement statistics in percentages.

#I	#D	OD #P	ID #P	OD C	ID C	OD V	ID V
2	0	1066	311	48.3	63.0	54.0	65.8
3	0	4423	1791	24.4	27.0	36.6	53.0
4	0	17191	10562	11.6	8.9	22.4	41.1
5	0	63579	62479	4.7	2.7	15.4	30.2
6	0	241983	360181	1.7	0.8	11.7	22.0
2	1	32981	2514	80.3	74.1	73.2	74.4
3	1	153297	21403	44.6	39.5	52.2	57.9
4	1	510714	148206	21.7	15.3	40.0	45.4
2	2	87905	4528	89.8	75.7	78.2	83.1
3	2	379077	41033	59.7	37.8	62.3	65.3



(a) Number of paths generated by the experiments.

(b) Execution times of the experiments.

Fig. 3. Coarse path search experiment results.

The significance of path validation can be seen from the results. A large portion of the converged paths become occluded by geometry or move out of the wall and edge bounds. Paths that have diffractions have high convergence and validity percentage compared to paths that only have reflections, as the diffraction can only move along the vector between the edge end points, thus, reducing the possibility of occlusion in the refined path. The paths with multiple reflections may also suffer from the bouncing problem described in II-C5.

IV. Conclusion

We have presented a pipeline for ray tracing-based path computation. Especially, we have presented a method called chaining for coarse path search, which enables high GPU occupancy and low divergence. In addition, we developed a path refiner, which utilizes a gradient descent-based approach for calculating the paths fulfilling the Fermat's principle of the least time. In experiments, the path refiner found suitable paths for about 38% and 34% of coarse paths in the case of outdoor and indoor scene, respectively. Out of these refined paths, approximately 44% remained valid for the outdoor case and 53% for the indoor case, which demonstrates the importance of path refinement and validation.

Acknowledgment

This work was supported by the Ho2020 Project ARIADNE (GA 871464), Celtic Project AI4Green, and Academy of Finland 6G Flagship (Grant 346208).

References

[1] V. Degli-Esposti, "Ray tracing propagation modelling: Future prospects," in *European Conference on Antennas and Propagation (EuCAP)*, 2014, pp. 2232–2232.

[2] F. Fuschini, E. M. Vitucci, M. Barbiroli, G. Falciasecca, and V. Degli-Esposti, "Ray tracing propagation modeling for future small-cell and indoor applications: A review of current techniques," *Radio Science*, vol. 50, no. 6, pp. 469–485, 2015.

[3] "AI4Green Celtic," https://www.celticnext.eu/project-ai4green/, accessed: 26.8.2022.

[4] Z. Yun and M. F. Iskander, "Ray tracing for radio propagation modeling: Principles and applications," *IEEE Access*, vol. 3, pp. 1089–1100, 2015.

[5] M. Schiller, A. Knoll, M. Mocker, and T. Eibert, "GPU accelerated ray launching for high-fidelity virtual test drives of VANET applications," in *Int. Conf. High Performance Comp. & Sim. (HPCS)*, 2015, pp. 262–268.

[6] P. Kreuzgruber, P. Unterberger, and R. Gahleitner, "A ray splitting model for indoor radio propagation associated with complex geometries," in *IEEE Vehicular Technology Conference*, 1993, pp. 227–230.

[7] S. Fortune, "Efficient algorithms for prediction of indoor radio propagation," in *IEEE Vehicular Technology Conference*, vol. 1, 1998, pp. 572–576.

[8] R. Hoppe, G. Wolfle, and F. Landstorfer, "Accelerated ray optical propagation modeling for the planning of wireless communication networks," in *IEEE Radio and Wireless Conference*, 1999, pp. 159–162.

[9] J. S. Lu, E. M. Vitucci, V. Degli-Esposti, F. Fuschini, M. Barbiroli, J. A. Blaha, and H. L. Bertoni, "A discrete environment-driven GPU-based ray launching algorithm," *IEEE Trans. Antennas Propag.*, vol. 67, no. 2, pp. 1180–1192, 2018.

[10] C.-H. Chen, C.-L. Liu, C.-C. Chiu, and T.-M. Hu, "Ultra-wide band channel calculation by SBR/image techniques for indoor communication," *Journal of Electromagnetic Waves and Applications*, vol. 20, no. 1, pp. 41–51, 2006.

[11] Y. Fu-Rong, S. Dan, C. Feng, and L. Yuan-jian, "Study on the propagation characteristics of radio wave in tunnels by the method of SBR/image," in *Proc. Asia-Pacific Conference on Antennas and Propagation*, 2014, pp. 706–709.

[12] J. B. Keller, "Geometrical theory of diffraction," *Josa*, vol. 52, no. 2, pp. 116–130, 1962.

[13] T. Funkhouser, N. Tsingos, I. Carlbom, G. Elko, M. Sondhi, J. E. West, G. Pingali, P. Min, and A. Ngan, "A beam tracing method for interactive architectural acoustics," *The Journal of the Acoustical Society of America*, vol. 115, no. 2, pp. 739–756, 2004.

[14] "NVIDIA Corporation, CUDA Toolkit Documentation," https://docs.nvidia.com/cuda/, accessed: 17.8.2022.

[15] S. G. Parker, J. Bigler, A. Dietrich, H. Friedrich, J. Hoberock, D. Luebke, D. McAllister, M. McGuire, K. Morley, A. Robison *et al.*, "OptiX: A general purpose ray tracing engine," *ACM Transactions on Graphics*, vol. 29, no. 4, pp. 1–13, 2010.

[16] N. Vaara, "Tools for ray tracing based radio channel modeling and simulation," *Master's Thesis*, 2022.

[17] "The Khronos Group, glTF 2.0 Specification," https://www.khronos.org/registry/glTF/specs/2.0/glTF-2.0.html, accessed: 26.8.2022.

[18] J. Tan, Z. Su, and Y. Long, "A full 3-D GPU-based beam-tracing method for complex indoor environments propagation modeling," *IEEE Trans. Antennas Propag.*, vol. 63, no. 6, pp. 2705–2718, 2015.

[19] S. Boyd, S. P. Boyd, and L. Vandenberghe, *Convex optimization*. Cambridge University Press, 2004.

[20] L. Armijo, "Minimization of functions having Lipschitz continuous first partial derivatives," *Pacific Journal of Mathematics*, vol. 16, no. 1, pp. 1–3, 1966.

[21] "Steve Rennich NVIDIA, CUDA C/C++ Streams and Concurrency," https://developer.download.nvidia.com/CUDA/training/StreamsAndConcurrencyWebinar.pdf, accessed: 26.8.2022.