Fast, Differentiable, GPU-Accelerated Ray Tracing for Multiple Diffraction and Reflection Paths

Jérome Eertmans*, Sophie Lequeu*, Benoît Legat*, Laurent Jacques*, Claude Oestges*
*ICTEAM, Université catholique de Louvain, Louvain-la-Neuve, Belgium, jerome.eertmans@uclouvain.be

arXiv:2510.16172v2 [eess.SP] 31 Dec 2025

***Abstract*—We present a fast, differentiable, GPU-accelerated optimization method for ray path tracing in environments containing planar reflectors and straight diffraction edges. Based on Fermat's principle, our approach reformulates the path-finding problem as the minimization of total path length, enabling efficient parallel execution on modern GPU architectures. Unlike existing methods that require separate algorithms for reflections and diffractions, our unified formulation maintains consistent problem dimensions across all interaction sequences, making it particularly suitable for vectorized computation. Through implicit differentiation, we achieve efficient gradient computation without differentiating through solver iterations, significantly outperforming traditional automatic differentiation approaches. Numerical simulations demonstrate convergence rates comparable to specialized Newton methods while providing superior scalability for large-scale applications. The method integrates seamlessly with differentiable programming libraries such as JAX and DrJIT, enabling new possibilities in inverse design and optimization for wireless propagation modeling. The source code is openly available at https://github.com/jeertmans/fpt-jax.**

***Index Terms*—Parallel programming, Electromagnetic propagation, Radio propagation, Ray tracing.**

I. Introduction

When determining possible ray paths connecting two communicating nodes, such as a transmitter (TX) and a receiver (RX), exhaustive Ray Tracing (RT) algorithms typically explore all combinations of object interactions in the scene. For each candidate path, the corresponding ray is traced while ignoring non-interacting objects, and subsequently discarded if it is obstructed or if the computed interaction points fall outside the boundaries of the interacting objects.

According to Fermat's principle, the path taken by a ray between two points makes the traversal time stationary with respect to small variations of the path [1]. In homogeneous media, where wave velocity is constant, this reduces to selecting the path whose length is a local extremum. Although this suggests that ray paths can be computed via optimization, the procedure must be computationally efficient, as the number of candidate rays grows exponentially with the number of objects and interactions. Hence, path tracing methods must prioritize speed.

For specular reflection on planar objects, the image method provides exact solutions with computational complexity linear in the number of reflections [2]. However, for diffraction—an essential interaction in wireless communications [3]—the image method does not apply. Instead, minimization-based approaches have been proposed and have demonstrated promising performance [4], [5], [6].

Most existing approaches, however, are not designed to scale efficiently on GPUs due to extensive branching, nor are they well-suited for integration with differentiable frameworks. To the best of our knowledge, most existing ray tracers, such as Sionna RT [7], separate the solution into two distinct processes: one for reflection-only paths, and another for reflection-and-diffraction paths. Furthermore, diffraction is usually supported only in a restricted form, often limited to a single diffraction occurring at a fixed position in the interaction chain (typically the last).

In this work, we introduce a generic optimization method capable of handling an arbitrary number of reflections and diffractions in any order, without compromising performance on diffraction-only cases. Moreover, the structure of our problem allows us to compute derivatives efficiently through implicit differentiation, resulting in faster performance than relying on automatic differentiation (AD).

Our main contributions are:

- A generic differentiable solver for ray paths with arbitrary reflections and diffractions;
- A performance comparison with state-of-the-art methods;
- A clean open-source implementation in JAX [8], also available through our DiffeRT RT tool [9].

II. Related Work

The fundamental principle underlying ray path determination in free-space is Fermat's principle. For scenarios involving multiple interactions with planar scattering objects, such as planar reflectors and straight edges, the stationary path strictly corresponds to the global minimum of the total length. This naturally leads to a minimization problem whose objective is to find the shortest path between the source and observation points.

Carluccio and Albani [4] presented important work on efficient ray tracing for multiple straight wedge diffraction. Their approach formulates the ray tracing problem as the minimization of the total path length, leveraging the strict convexity of the cost function to guarantee a unique global minimum. The authors used a modified Newton search algorithm with near-quadratic convergence, achieving remarkable computational efficiency for scenarios involving only diffraction interactions. However, their method is specifically

designed for edge diffraction and does not address mixed reflection-diffraction scenarios.

Building upon this foundation, Puggelli, Carluccio, and Albani [5] extended their approach to handle scenarios comprising both planar reflectors and straight wedges. Their generalized algorithm applies the image method to eliminate reflective surfaces, effectively reducing the problem to a pure diffraction scenario that can be solved using the original Carluccio-Albani method. While this approach maintains computational efficiency, it introduces several limitations for modern parallel computing architectures. Specifically, the treatment of reflections through image theory requires different computational paths depending on the sequence of interaction types, leading to extensive branching operations that are inherently inefficient on GPU architectures, where uniform execution across parallel threads is essential for optimal performance.

Furthermore, the Newton-based approaches in [4], [5] exhibit problem sizes that depend on the number and order of diffractions within the interaction sequence. This variability in problem dimensionality creates additional challenges for GPU implementation, as efficient parallel execution typically requires uniform problem shapes across all threads. The dependency on interaction order also complicates the implementation of vectorized operations, which are crucial for achieving high throughput on modern hardware accelerators.

Despite the computational elegance of Fermat-based path tracing—where paths are determined by minimizing path length according to physical principles—relatively little research has been conducted to improve algorithmic performance or adapt these methods for emerging computational paradigms. Most existing ray tracing frameworks continue to rely on separate algorithms for pure reflection (using the image method) and mixed reflection-diffraction scenarios, leading to code that cannot be efficiently parallelized on GPUs.

Another line of recent research is the advent of AD frameworks, which have revolutionized scientific computing by enabling efficient computation of derivatives for complex algorithms [10], [11]. In the context of RT, AD capabilities are increasingly important for applications such as inverse design, optimization of wireless networks, and machine learning-based propagation modeling [7], [12], [13]. However, directly applying AD to iterative solvers—such as the Newton methods used in existing Fermat-path tracing approaches—often yields suboptimal performance due to the need to differentiate through the entire iteration sequence.

In this work, we propose to bridge these existing contributions by (1) developing a unified framework that handles arbitrary sequences of reflections and diffractions within a single algorithmic procedure that can be accelerated on GPUs, while (2) enabling fast derivative computation through implicit differentiation rather than AD through solver iterations. Our approach builds upon the convex optimization foundations established by Carluccio and Albani while addressing the computational architecture requirements of modern ray tracing applications.

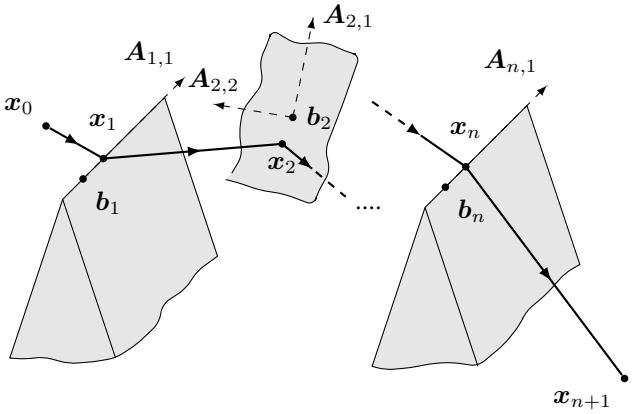

Fig. 1. Illustration of a ray path with n interactions, including reflections and diffractions, inspired from [4, Fig. 1]. For conciseness, $\boldsymbol{A}_{i,j}$ is shorthand notation for $\boldsymbol{A}_{i,:,j}$, i.e., the j-th base vector of the i-th object.

III. Methodology

The path tracing task consists of finding all feasible paths between two fixed points, namely the TX and RX antennas, subject to at most n_{MAX} specular reflection or diffraction interactions with surrounding objects. In practice, we first list all ordered sequences of n objects, corresponding to all candidate paths subject to exactly n interactions, for $n = 1, \ldots, n_{\text{MAX}}$. For each of these ordered sequences, we find the coordinates of the interaction points that lead to the shortest path (Fermat's principle). The search for the shortest path is formulated as a minimization problem, which we solve using an iterative optimization method. This optimization procedure is detailed hereafter, and is applied in parallel to many different ordered sequences of n objects. As is common in minimization-based approaches [4], [5], [14], as well as with the image method [2], objects are first assumed to be infinitely large when solving for the paths, with possible occlusions by other objects being ignored. The ray paths are then post-processed using standard ray-object intersection tests. This work does not study this post-processing step, as it is common to all methods. Finally, gradients of the solution can be obtained through implicit differentiation.

A. Notation

We use bold uppercase letters (e.g., $\boldsymbol{A}$) to denote matrices and tensors, bold lowercase letters (e.g., $\boldsymbol{a}$) for vectors, and non-bold lowercase letters (e.g., a) for scalars. The i-th element (resp. row) of a vector $\boldsymbol{a}$ (resp. matrix $\boldsymbol{A}$) is denoted by a_i (resp. $\boldsymbol{A}_i$), while the (i,j)-th element of a matrix $\boldsymbol{A}$ is denoted by $a_{i,j}$. The transpose of a vector or matrix is indicated by the superscript $^\top$. The Euclidean norm of a vector $\boldsymbol{a}$ is denoted by $\|\boldsymbol{a}\|$.

B. Problem Formulation

Formally, a ray path $\boldsymbol{X}^*$ is obtained as the solution of the minimization problem

$$\boldsymbol{X}^* = \underset{\boldsymbol{X} \in \mathbb{R}^{(n+2)\times 3}}{\arg\min} \; L(\boldsymbol{X}), \tag{1}$$

where $L(\boldsymbol{X})$ denotes the total path length

$$L(\boldsymbol{X}) = \sum_{i=0}^{n} \|\boldsymbol{x}_{i+1} - \boldsymbol{x}_i\|, \tag{2}$$

with $\boldsymbol{x}_i$, $1 \leq i \leq n$, representing the intermediate interaction points, and $\boldsymbol{x}_0$ and $\boldsymbol{x}_{n+1}$ corresponding to the start and end points (typically the TX and RX coordinates in wireless propagation scenarios), see Fig. 1.

In the common case of planar interactions[1], such as specular reflection on planes or diffraction on straight edges, the minimization problem (1) is strictly convex when expressed in a parametric space [4].

Indeed, each interaction point $\boldsymbol{x}_i$ can be expressed as an affine transformation of parametric variables $\boldsymbol{t}_i \in \mathbb{R}^d$:

$$\boldsymbol{x}_i = \boldsymbol{A}_i \boldsymbol{t}_i + \boldsymbol{b}_i, \tag{3}$$

where $\boldsymbol{b}_i$ is a reference point on the i-th planar object, $\boldsymbol{A}_i \in \mathbb{R}^{3\times d}$ is a matrix of basis vectors spanning the object, and d is the intrinsic dimension of the object ($d = 2$ for planar reflections, $d = 1$ for edge diffractions), see Fig. 1.

A key advantage of this formulation is that all matrices $\boldsymbol{A}_i$ (resp. vectors $\boldsymbol{t}_i$ and $\boldsymbol{b}_i$) can be organized into a single tensor $\boldsymbol{A} = \{\boldsymbol{A}_i\}_{i=1}^n$ (resp. matrices $\boldsymbol{T}$ and $\boldsymbol{B}$). When $d = 2$, interactions corresponding to diffractions are easily handled by using $\mathbf{0}$ as the second base vector. While this may increase the apparent number of unknowns for some interaction types, it makes the problem particularly well-suited for massively parallel execution on GPUs, where many ray paths are solved simultaneously. We discuss the effect of this parametrization on convergence speed in later sections.

The minimization problem thus becomes

$$\boldsymbol{T}^* = \arg\min_{\boldsymbol{T}} L(\boldsymbol{T}; \boldsymbol{A}, \boldsymbol{B}), \tag{4}$$

where the objective function is given by:

$$L(\boldsymbol{T}; \boldsymbol{A}, \boldsymbol{B}) = \sum_{i=0}^{n} \|\boldsymbol{A}_{i+1}\boldsymbol{t}_{i+1} + \boldsymbol{b}_{i+1} - \boldsymbol{A}_i\boldsymbol{t}_i - \boldsymbol{b}_i\|, \tag{5}$$

where $\boldsymbol{A}_0\boldsymbol{t}_0$ and $\boldsymbol{A}_{n+1}\boldsymbol{t}_{n+1}$ are set to zero, and $\boldsymbol{b}_0$ and $\boldsymbol{b}_{n+1}$ are defined as the start and end points, respectively.

C. Iterative Solver

We utilize the Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm [15, Chapter 3], a quasi-Newton method, to solve the path length minimization problem (4). Unlike the custom Newton method used in [4], [5], BFGS does not require computing the Hessian matrix, which can be ill-conditioned in certain scenarios, particularly when interaction points are close together, when the path segments are nearly collinear, or more importantly when one of the base vectors is zero, which occurs for diffraction interactions. Instead, BFGS approximates the inverse Hessian $\boldsymbol{H}$ using only gradient evaluations, making it more robust when the Hessian is ill-conditioned.

Unlike traditional BFGS implementations, which perform an inexact line search along the descent direction $\boldsymbol{P}$, we use a fixed-point iteration scheme

$$\alpha^{k+1} = -\frac{\sum_{i=0}^{n} (\Delta\boldsymbol{A}_i\boldsymbol{p}_i)^\top (\Delta\boldsymbol{x}_i) \big/ \left\|\Delta\boldsymbol{x}_i + \alpha^k \Delta\boldsymbol{A}_i\boldsymbol{p}_i\right\|}{\sum_{i=0}^{n} (\Delta\boldsymbol{A}_i\boldsymbol{p}_i)^\top (\Delta\boldsymbol{A}_i\boldsymbol{p}_i) \big/ \left\|\Delta\boldsymbol{x}_i + \alpha^k \Delta\boldsymbol{A}_i\boldsymbol{p}_i\right\|}, \tag{6}$$

where $\Delta\boldsymbol{A}_i\boldsymbol{p}_i = \boldsymbol{A}_{i+1}\boldsymbol{p}_{i+1} - \boldsymbol{A}_i\boldsymbol{p}_i$, and $\Delta\boldsymbol{x}_i = \boldsymbol{x}_{i+1} - \boldsymbol{x}_i$, to find the optimal step size α^* that minimizes the objective function along the direction $\boldsymbol{P}$

$$\alpha^* = \arg\min_{\alpha} L(\boldsymbol{T} + \alpha\boldsymbol{P}; \boldsymbol{A}, \boldsymbol{B}). \tag{7}$$

In practice, we find that the fixed point iterations converge (in most cases, the right-hand side is locally continuously differentiable and its derivative is smaller than 1 in absolute value). We observe that one iteration is often sufficient to reach the optimal step size with a tolerance of 1%.

Finally, and unlike traditional iterative solvers on CPUs, we run the algorithm on a fixed number of iterations, to ensure uniform execution time across all paths in a batch. Indeed, on GPUs, relying on convergence criteria can lead to significant performance degradation as all threads must wait for the slowest one to finish.

D. Implicit Differentiation for Gradient Computation

In many applications, such as inverse problems or parameter optimization, it is necessary to compute the gradient of one scalar quantity (e.g., the received power) with respect to many problem parameters (e.g., object positions or orientations). A straightforward approach is to use reverse-mode AD to differentiate through all the iterations of the solver. However, this can be computationally expensive and memory-intensive, especially when the direct method (here, radio propagation via RT) involves many iterations, as AD must record and backpropagate through every operation in the iterative process.

To address this, we leverage the fact that, at convergence, our solver produces a solution $\boldsymbol{T}^*(\boldsymbol{\theta})$ (with $\boldsymbol{\theta} = (\boldsymbol{A}, \boldsymbol{B})$) that satisfies the first-order optimality condition:

$$\nabla_{\boldsymbol{T}} L(\boldsymbol{T}^*(\boldsymbol{\theta}); \boldsymbol{\theta}) = \mathbf{0}, \tag{8}$$

where L is the objective function.

Assuming the solver has converged to such a stationary point, we can use the implicit function theorem [16, Theorem 7-6, p. 146] to compute the total derivative of $\boldsymbol{T}^*$ with respect to the parameters $\boldsymbol{\theta}$, without differentiating through the entire sequence of solver steps. However, we still rely on AD to compute the Jacobian vectors of $\nabla_{\boldsymbol{T}} L$.

Applying the implicit function theorem to (8) yields:

$$\frac{\partial}{\partial\boldsymbol{\theta}} \nabla_{\boldsymbol{T}} L(\boldsymbol{T}^*(\boldsymbol{\theta}); \boldsymbol{\theta}) + \frac{\partial \nabla_{\boldsymbol{T}} L}{\partial \boldsymbol{T}} \frac{\partial \boldsymbol{T}^*}{\partial\boldsymbol{\theta}} = \mathbf{0}. \tag{9}$$

Rearranging, we obtain:

$$\frac{\partial \boldsymbol{T}^*}{\partial\boldsymbol{\theta}} = -\left[\frac{\partial \nabla_{\boldsymbol{T}} L}{\partial \boldsymbol{T}}\right]^{-1} \frac{\partial}{\partial\boldsymbol{\theta}} \nabla_{\boldsymbol{T}} L. \tag{10}$$

[1] Although this may appear restrictive, most 3D models—particularly outdoor urban environments—are described by polygonal objects, which are inherently planar. Detailed non-planar geometries are relatively rare. In [14], we proposed a minimization-based approach that extends to more complex objects, such as spheres, though at the cost of losing convexity.

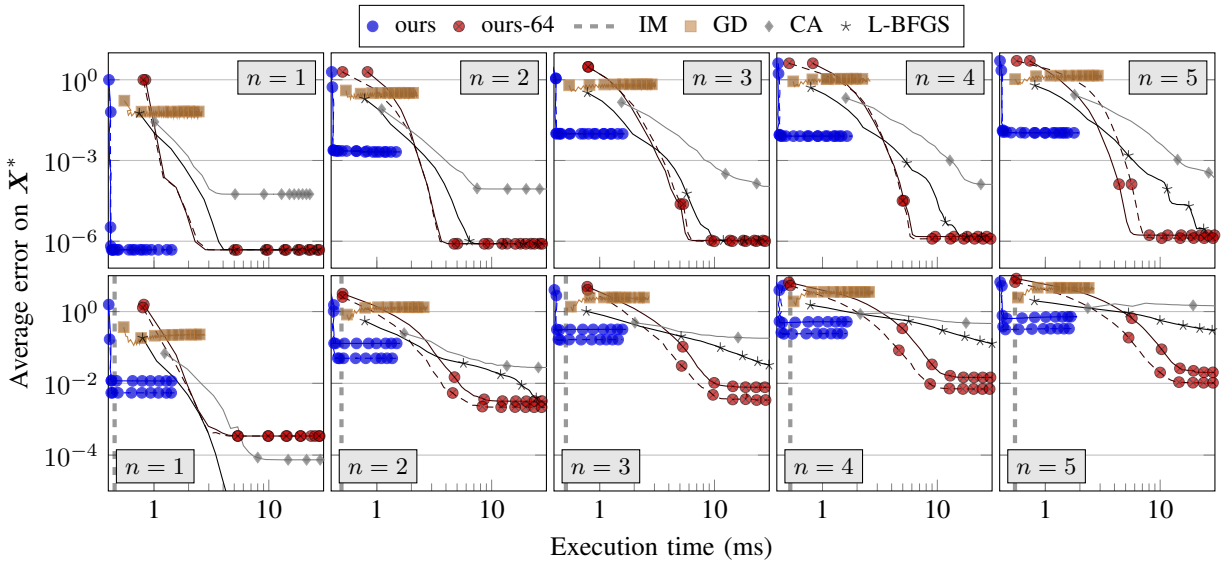

Fig. 2. Average error on 1000 paths against time taken by different solvers, for n increasing from 1 to 5. Top row corresponds to 1D problems (diffractions), bottom row to 2D problems (reflections). Diffractions solved with $d = 2$ are shown with dashed lines in the top row, only with our solver as other methods fail. Similarly, mixed reflections and diffractions cases are show in dashed in the bottom row, for our solver only. Our solver is run in two configurations: standard (ours) and with 64 fixed-point iterations per step (ours-64). Because the image method is exact, a vertical line is shown to indicate its execution time.

In practice, we are often interested in computing vector-Jacobian products of the form $\boldsymbol{v}^\top \frac{\partial \boldsymbol{T}^*}{\partial \boldsymbol{\theta}}$, as required by reverse-mode AD. This can be done efficiently by solving the following linear system:

$$\boldsymbol{u}^\top = -\boldsymbol{v}^\top \left[\frac{\partial \nabla_{\boldsymbol{T}} L}{\partial \boldsymbol{T}}\right]^{-1}, \tag{11}$$

or equivalently

$$\left[\frac{\partial \nabla_{\boldsymbol{T}} L}{\partial \boldsymbol{T}}\right]^\top \boldsymbol{u} = -\boldsymbol{v}, \tag{12}$$

where $\frac{\partial \nabla_{\boldsymbol{T}} L}{\partial \boldsymbol{T}}$ is a symmetric positive semi-definite matrix, being the Hessian of the convex function L, and then computing

$$\boldsymbol{v}^\top \frac{\partial \boldsymbol{T}^*}{\partial \boldsymbol{\theta}} = \boldsymbol{u}^\top \frac{\partial}{\partial \boldsymbol{\theta}} \nabla_{\boldsymbol{T}} L. \tag{13}$$

This approach avoids differentiating through the entire solver, significantly reducing computational cost and memory usage, while providing exact gradients under the assumption of solver convergence.

IV. Simulation Results

We evaluate our method on a variety of random scenarios to demonstrate its effectiveness and computational efficiency. All simulations are conducted on an NVIDIA GeForce RTX 3070 GPU with 8 GB memory. We implemented the algorithm with JAX [8], leveraging its just-in-time compilation and convenient support for generating GPU-compatible code. While JAX provides built-in AD capabilities, it also allows defining custom derivative rules, so we can implement implicit differentiation manually to avoid differentiating through solver iterations in backward-mode AD.

A. Performance Comparison

We benchmark our approach against the following solvers:

- the image method (IM), in reflection-only scenarios;
- a standard gradient descent (GD);
- the Carluccio and Albani (CA) method [4], extended to $d = 2$ when relevant;
- and a limited-memory BFGS (L-BFGS) solver implemented in the Optax library [17].

Each solver processes 1000 ray paths in parallel, performing up to 100 iterations. We record the average error and execution time using single-precision floating-point (32-bit) arithmetic. The average error is defined as the mean Euclidean distance between the estimated interaction points and the ground-truth points obtained from a high-precision solver on a CPU.

Fig. 2 summarizes the results for increasing path complexity, with the number of interactions n ranging from 1 to 5. The top row corresponds to diffraction-only problems ($d = 1$), and the bottom row to reflection-only problems ($d = 2$). Dashed lines indicate cases involving diffractions represented using $d = 2$ instead of $d = 1$, or mixed reflections and diffractions ($d = 2$), which are only supported by our method.

As shown in Fig. 2, the standard gradient descent (GD) consistently fails to converge. In diffraction-only configurations ($d = 1$), our solver achieves both the lowest error and the fastest convergence across all cases. For $n > 1$, increasing the number of fixed-point iterations enables convergence down to machine precision. Notably, extending the optimization to $d = 2$ has negligible impact on accuracy or speed for our method, as the $d = 1$ and $d = 2$ curves almost coincide, while other solvers fail to handle the additional dimension.

In reflection-only scenarios, a similar trend is observed. Except for the simplest case ($n = 1$), where L-BFGS reaches

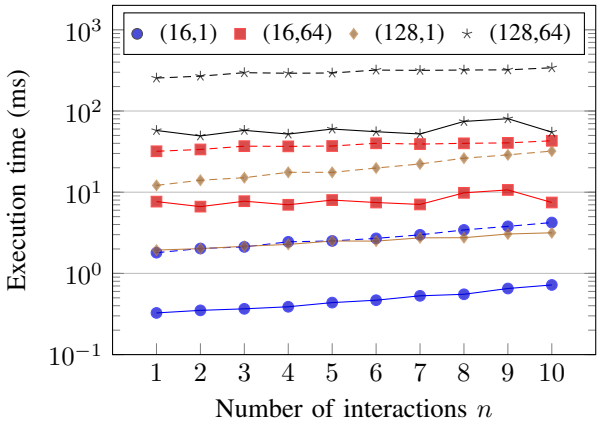

Fig. 3. Execution time for computing the gradient of 1000 path lengths at $\boldsymbol{T}^*(\boldsymbol{\theta})$ with respect to all object parameters, i.e., $\nabla_{\boldsymbol{\theta}} \boldsymbol{L}(\boldsymbol{T}^*(\boldsymbol{\theta}); \boldsymbol{\theta})$, using implicit differentiation (solid lines) and automatic differentiation (dashed lines), as a function of the number of interactions n, for $d = 2$. The solver is run for 16 or 128 iterations, with 1 or 64 fixed-point iterations per step.

machine precision, our solver consistently delivers higher accuracy in less time. Increasing the number of fixed-point iterations again improves precision for larger n. The image method remains the most efficient, achieving comparable or better accuracy at an order-of-magnitude lower runtime.

Finally, in mixed reflection-diffraction configurations (dashed curves in the bottom row of Fig. 2), the average convergence is even faster than in reflection-only cases, likely because diffractions introduce only one unknown per interaction, whereas reflections introduce two.

B. Implicit vs Automatic Differentiation

When comparing the execution time of gradient computation using implicit differentiation versus AD, we observe significant performance improvements with our approach (nearly 10 times faster) that remain constant regardless of the number of solver iterations, the number of fixed-point iterations per step, or the number of considered interactions; see Fig. 3. We also observed very similar results for $d = 1$.

V. Conclusion and Future Work

We have presented a unified, differentiable ray tracing method that reformulates path finding as a convex optimization problem and leverages implicit differentiation for efficient gradient computation. The proposed approach handles arbitrary sequences of reflections and diffractions within a single optimization framework, eliminating branching complexity and enabling efficient parallel execution on GPUs.

Simulation results demonstrate that the solver achieves fast and accurate convergence across a wide range of interaction scenarios, outperforming existing minimization-based approaches in both speed and robustness. Nevertheless, the results also reveal certain limitations, including slower convergence than the image method in reflection-only cases.

Future work will focus on reformulating the current nonsmooth optimization problem into a second order cone program, to be solved with dedicated solvers [18], [19] to improve convergence properties while preserving the advantages of implicit differentiation [20]. Additional efforts will aim at refining the selection of initial candidates, improving line-search strategies, and analyzing the convergence behavior of the proposed fixed-point scheme.

Acknowledgment

Part of this research was funded by the FRS-FNRS (PDR, QuadSense, T.0160.24).

References

[1] D. McNamara, C. Pistorius, and J. Malherbe, *Introduction to the Uniform Geometrical Theory of Diffraction*. Artech House, 1990.

[2] J. Borish, "Extension of the image model to arbitrary polyhedra," *The Journal of the Acoustical Society of America*, vol. 75, no. 6, pp. 1827–1836, jun 1984.

[3] J.-P. Rossi, J. Wiart, and F. Eynard, "In situ measurement of reflection and diffraction coefficients of UHF radio waves on buildings using a ring array," *Radio Science*, vol. 35, no. 2, pp. 361–369, mar 2000.

[4] G. Carluccio and M. Albani, "An Efficient Ray Tracing Algorithm for Multiple Straight Wedge Diffraction," *IEEE Transactions on Antennas and Propagation*, vol. 56, no. 11, pp. 3534–3542, nov 2008.

[5] F. Puggelli, G. Carluccio, and M. Albani, "A Novel Ray Tracing Algorithm for Scenarios Comprising Pre-Ordered Multiple Planar Reflectors, Straight Wedges, and Vertexes," *IEEE Transactions on Antennas and Propagation*, vol. 62, no. 8, pp. 4336–4341, aug 2014.

[6] N. Vaara, P. Sangi, J. Pyhtilä, M. Juntti, and J. Heikkilä, "A refined path generation pipeline for radio channel propagation modeling," in *2023 17th European Conference on Antennas and Propagation (EuCAP)*, 2023, pp. 1–5.

[7] J. Hoydis, F. A. Aoudia, S. Cammerer, M. Nimier-David, N. Binder, G. Marcus, and A. Keller, "Sionna RT: Differentiable ray tracing for radio propagation modeling," 2023.

[8] J. Bradbury *et al.*, "JAX: composable transformations of Python+NumPy programs," http://github.com/jax-ml/jax, 2018.

[9] J. Eertmans, "DiffeRT: A Differentiable Ray Tracing Toolbox for Radio Propagation Simulations," https://github.com/jeertmans/DiffeRT, 2025.

[10] R. Frostig, M. Johnson, and C. Leary, "Compiling machine learning programs via high-level tracing," in *2018 Conference on Systems and Machine Learning (SysML)*, 2018.

[11] W. Jakob, S. Speierer, N. Roussel, and D. Vicini, "Dr.Jit: A Just-In-Time Compiler for Differentiable Rendering," feb 2022.

[12] S. Lequeu, J. Eertmans, C. Oestges, and L. Jacques, "Wireless indoor source localization as an inverse problem: An optimization approach via ray tracing simulations," Master's thesis, Ecole polytechnique de Louvain, 2025.

[13] X. Han, T. Zheng, T. X. Han, and J. Luo, "RayLoc: Wireless indoor localization via fully differentiable ray-tracing," 2025.

[14] J. Eertmans, C. Oestges, and L. Jacques, "Min-Path-Tracing: A Diffraction Aware Alternative to Image Method in Ray Tracing," in *2023 17th European Conference on Antennas and Propagation (EuCAP)*, 2023, pp. 1–5.

[15] R. Fletcher, *Practical Methods of Optimization*. John Wiley & Sons, Ltd, 2000.

[16] T. M. Apostol, *Mathematical analysis : a modern approach to advanced calculus*. Reading, Massachusetts: Addison Wesley, 1957.

[17] DeepMind *et al.*, "The DeepMind JAX Ecosystem," http://github.com/google-deepmind, 2020.

[18] A. Domahidi, E. Chu, and S. Boyd, "ECOS: An SOCP solver for embedded systems," in *2013 European Control Conference (ECC)*. Zurich: IEEE, jul 2013, pp. 3071–3076.

[19] B. O'Donoghue, E. Chu, N. Parikh, and S. Boyd, "Conic Optimization via Operator Splitting and Homogeneous Self-Dual Embedding," *Journal of Optimization Theory and Applications*, vol. 169, no. 3, pp. 1042–1068, jun 2016.

[20] A. Agrawal, S. Barratt, S. Boyd, E. Busseti, and W. M. Moursi, "Differentiating through a cone program," *Journal of Applied and Numerical Optimization*, vol. 1, no. 2, pp. 107–115, 2019.