ON THE USE OF RAY TRACING PROGRAMMABLE FRAMEWORKS FOR RADIO WAVE PROPAGATION PREDICTION

[1]MARCOS BARRANQUERO, [2]JUAN CASADO, [3]JOSEFA GÓMEZ, [4]ABDELHAMID TAYEBI

[1,3,4]Computer Science Department, University of Alcalá, Alcalá de Henares, Spain
[2]Starleaf, Building 7, Hatters Lane Watford WD18 8YN, United Kingdom
E-mail: [1]marcos.barranquero@edu.uah.es, [1]josefa.gomezp@uah.es, [1]hamid.tayebi@uah.es, [2]juan.ballesteros@starleaf.com

Abstract - This work focuses on the use of ray tracing programmable frameworks, like Nvidia Optix, for radio wave propagation prediction. Although these frameworks are oriented to graphics visualization, they can be tailored to calculate specular reflections on walls and diffractions in edges of buildings. Once all the paths between the transmitter and the receiver are obtained, Maxwell's equations can be applied to compute the path loss or the received power.

Keywords - Propagation, Ray Launching, Ray Tracing.

I. INTRODUCTION

A huge increase in the development of ray tracing simulation tools has been seen in the last decades [1-5]. Since the main drawback of these tools is the high computational requirements when analyzing complex scenarios, lots of efforts have been done to accelerate and optimize the algorithms.

This work focuses on deterministic models to calculate the radio wave propagation because they provide more accurate results. On the other hand, empirical models are much more faster but their accuracy is not so high.

Deterministic models can be classified in two categories: ray launching based on the shooting and bouncing rays technique and ray tracing based on the image theory concept. Ray launching is also known as brute force ray tracing method. Ray launching is easier to implement using ray tracing programmable frameworks since it considers a bundle of transmitted rays that may or may not reach the receiver. The receiver is modeled as a sphere with an appropriate radius. If the radius is too small, it is possible that none of the rays will reach the sphere. However, if the radius is too big, two or more similar rays could reach the sphere. Fig. 1 shows the partial results of a ray launching simulation between a transmitter (red sphere) and a receiver (grey sphere). It can be seen that four direct rays reach the receiver. This is not correct since there must only be one direct ray between them. Finally, all the rays that intersect the reception sphere contribute to the total electrical field received at that concrete point. It is important to remark that all duplicated rays must be removed before applying the electromagnetic equations.

There are many ray tracing frameworks available such as Optix [6], POVRay [7] or OSPRay [8]. The most popular is Optix because it takes advantage of the GPUs (Graphics Processing Unit). Therefore, the pipeline of the ray tracing algorithms that use Optix can be accelerated with an optimal performance. OptiX is a programmable framework that allows developers to create ray launching applications that run on NVIDIA GPUs at very high speeds, thus drastically reducing simulation time [9]. OptiX was initially created to improve graphics rendering, but now can be extended by allowing to collect and send custom data related to the rays between the transmitter and the receiver. This flexibility allows radio propagation prediction algorithms taking advantage of the parallelization capabilities of the CUDA (Compute Unified Device Architecture) technology. In this way, it will be possible to carry out precise electromagnetic simulations of large urban areas in much shorter time than those currently obtained using similar tools.

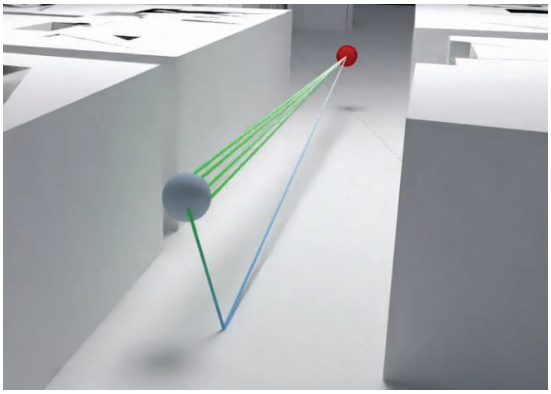

Fig.1. Example of a wrong simulation because the radius of the reception sphere is too big.

Some related works can be found in the literature based on Optix [6, 10-15]. However, none of them uses the Python PlotOptix interface [16], which facilitates the implementation of the software by allowing a very high number of rays to be handled with a relatively low computational cost in each simulation. PlotOptix is a Python package that allows combining components prototyped in the high level language with the optimized workflow of the underlying OptiX ray tracing engine. The ray

launching code proposed in this work is based on the use of PlotOptix.

II. PROPOSED ALGORITHM

The algorithm can be implemented in Python language. Monte-Carlo approach, typical for the ray tracing in graphics, can been used. Specular reflection from flat faces are implemented using the face normal direction. A wireframe of relatively thin, linear segments must be added to the scene in order to implement diffraction. The unidirectional pipeline is described as follows:

a) Start with a relatively dense distribution of rays starting from a measurement point. Save in the output the hit position, object id, and normal information. Split reflection hits on faces and diffraction hits on edges.
b) Continue with shooting reflected rays, each time saving edge hits for later. Repeat b) until some max reflections.
c) Process the buffer of edge hits in batches:
1. Calculate directions of a number or outgoing segments for each edge hit; follow these rays like in b), with a lower number of segments.
2. Continue until the buffer of edge hits is empty.

It is important to remark that all faces are triangles, if something is a polygon, it is divided into a number of triangles. Also, ray-triangle intersection operation is shifted to hardware because in the RTX generation chips the operation costs almost nothing. Still, in practice there are too many intersections for a brute force method. The common approach, equivalent to the Space Volumetric Partitioning or Z-buffer [2], is a hierarchical structure of bounding volumes (BVH), where all the triangles from the scene mesh are sorted. There are a few methods to calculate it. Once the BVH is ready, the ray-BVH intersection is again hardware supported. There are dedicated cores (called RT) in RTX GPUs that do this intersection very fast. So only very few triangles need to be checked for the intersection.

Algorithms in the hit program are unique to the application and are written in a normal way, meaning a small piece of code that will calculate the reflection (or diffraction) and prepare the ray to shoot the next segment.

Three main contributions are required to be calculated: direct ray, reflected rays and diffracted rays. Of course, there can be combinations between reflections and diffractions. Typically, in situations of non-line of sight, a ray which is launched from the transmitter to the receiver is composed of several segments because the ray can be reflected and diffracted several times. Specular reflection from flat faces is trivial, because the normal direction of the facet is the only data required to obtain the reflected ray. Diffraction is much more complex since we need to calculate outgoing direction of the ray taking into account the Keller's cone. Instead of sampling visibility of each edge in the scene, a wireframe of thin segments along the edges is added to the scene. The unidirectional pipeline to compute the diffracted rays is the following:

a) Start with a relatively dense distribution of rays starting from a measurement point. In the output there is the hit position, object id, and normal information. It is easy to know if a face or a wireframe segment (an edge) have been hit, and at which angle.
b) Continue with shooting reflected rays, each time saving edge hits for later. Repeat b) until some max reflections.
c) At this point multi-reflection cases are completed, hopefully some ended at the transmitter. There must be a buffer of primary, secondary, and so on edge hits, hopefully not enormous. Now edge hits from the buffer are processed in batches:
1. Calculate directions of a number of outgoing segments for each edge hit. They will cover a cone with a constant beta angle and a range of phi (Keller's cone). Then, follow these rays like in b), maybe with a lower number of segments.
2. Continue until the buffer of edge hits is empty.

This pipeline should allow for multiple diffraction, reflection, and mixed cases. At each step there is possibility of checking also the direct visibility of the. It is also possible to implement the pipeline avoiding memory re-allocations (e.g. circular buffer for edge hits), and keep high efficiency.

One can do multiple passes of such a pipeline, with a jitter in the initial segment directions to improve coverage of the scene with the secondary segments.

III. CONCLUSIONS

An accelerated ray launching approach is described in this paper. It is expected that the combination of ray-tracing algorithms and parallelization techniques on GPUs would permit real-time simulations for future wireless systems, as well as multidimensional coverage maps in 3D for both indoor and outdoor environments.

ACKNOWLEDGMENTS

This work is supported by the program "Programa de Estímulo a la Investigación de Jóvenes Investigadores" of Vice rectorate for Research and Knowledge Transfer of the University of Alcala and by the Comunidad de Madrid (Spain) through project CM/JIN/2019-028.

REFERENCES

[1] F. Saez de Adana, O. Gutierrez Blanco, I. Gonzalez Diego, J. Perez Arriaga and M. F. Catedra, "Propagation model based on ray tracing for the design of personal communication systems in indoor environments," IEEE Transactions on Vehicular Technology, vol. 49, no. 6, pp. 2105-2112, Nov. 2000, doi: 10.1109/25.901882.

[2] M. F. Catedra, J. Perez, F. Saez de Adana and O. Gutierrez, "Efficient ray-tracing techniques for three-dimensional analyses of propagation in mobile communications: application to picocell and microcell scenarios," IEEE Antennas and Propagation Magazine, vol. 40, no. 2, pp. 15-28, April 1998, doi: 10.1109/74.683539.

[3] C. Dong, L. Guo, X. Meng and Y. Wang, "An Accelerated SBR for EM Scattering From the Electrically Large Complex Objects," IEEE Antennas and Wireless Propagation Letters, vol. 17, no. 12, pp. 2294-2298, Dec. 2018. doi: 10.1109/LAWP.2018.2873119

[4] A. Navarro, D. Guevara and J. Gómez, "A Proposal to Improve Ray Launching Techniques," IEEE Antennas and Wireless Propagation Letters, vol. 18, no. 1, pp. 143-146, Jan. 2019. doi: 10.1109/LAWP.2018.2883235.

[5] CHANG, K.R.; KIM, H.T. 'Improvement of the computation efficiency for a ray-launching model'. IEE Proceedings on Microwaves and Antennas Propagation, Vol. 145, nº 4, Agosto 1998, pp. 303-308. N. Cadavid, D. G. Ibarra and S. L. Salcedo, "Using 3-D Video Game Technology in Channel Modeling," in IEEE Access, vol. 2, pp. 1652-1659, 2014. doi: 10.1109/ACCESS.2014.2370758

[6] J. Opiła, "Prototyping of visualization designs of 3D vector fields using POVRay rendering engine," 2016 39th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), Opatija, Croatia, 2016, pp. 343-348. doi: 10.1109/MIPRO.2016.7522164

[7] I. Wald et al., "OSPRay - A CPU Ray Tracing Framework for Scientific Visualization," in IEEE Transactions on Visualization and Computer Graphics, vol. 23, no. 1, pp. 931-940, Jan. 2017. doi: 10.1109/TVCG.2016.2599041

[8] NVIDIA driver. [Online]. Available: https://www.nvidia.com/Download/index.aspx

[9] J. Tan, Z. Su and Y. Long, "A Full 3-D GPU-based Beam-Tracing Method for Complex Indoor Environments Propagation Modeling," in IEEE Transactions on Antennas and Propagation, vol. 63, no. 6, pp. 2705-2718, June 2015. doi: 10.1109/TAP.2015.2415036

[10] Y. Liu, D. Shi, A. Li and P. Zeng, "Radio Wave Propagation Prediction Based on the NVIDIA OptiX GPU Ray Tracing Engine," 2019 IEEE 6th International Symposium on Electromagnetic Compatibility (ISEMC), Nanjing, China, 2019, pp. 1-3. doi: 10.1109/ISEMC48616.2019.8986068

[11] C. Y. Kee and C. Wang, "Efficient GPU Implementation of the High-Frequency SBR-PO Method," in IEEE Antennas and Wireless Propagation Letters, vol. 12, pp. 941-944, 2013. doi: 10.1109/LAWP.2013.2274802

[12] J. S. Lu et al., "A Discrete Environment-Driven GPU-Based Ray Launching Algorithm," in IEEE Transactions on Antennas and Propagation, vol. 67, no. 2, pp. 1180-1192, Feb. 2019. doi: 10.1109/TAP.2018.2880036

[13] V. Degli-Esposti et al., "Efficient RF coverage prediction through a fully discrete, GPU-parallelized ray-launching model," 12th European Conference on Antennas and Propagation (EuCAP 2018), London, UK, 2018, pp. 1-5. doi: 10.1049/cp.2018.0579

[14] M. Schiller, A. Knoll, M. Mocker and T. Eibert, "GPU accelerated ray launching for high-fidelity virtual test drives of VANET applications," 2015 International Conference on High Performance Computing & Simulation (HPCS), Amsterdam, Netherlands, 2015, pp. 262-268. doi: 10.1109/HPCSim.2015.7237048

[15] R. Sulej, PlotOptiX: ray tracing and data visualization package for Python. [Online]. Available: https://plotoptix.rnd.team.

★★★