Alma Mater Studiorum Università di Bologna
Archivio istituzionale della ricerca

A Discrete Environment-Driven GPU-Based Ray Launching Algorithm

(Article begins on next page)

31 January 2025

A Discrete Environment-Driven GPU-Based Ray Launching Algorithm

J. S. Lu, *Member, IEEE*, E. M. Vitucci, Senior *Member, IEEE*, V. Degli-Esposti, *Senior Member, IEEE*, F. Fuschini, M. Barbiroli, J. Blaha, H. L. Bertoni, *Life Fellow, IEEE*

Abstract— **We present here a novel, fully discrete Ray Launching field prediction algorithm that takes advantage of environment preprocessing to efficiently trace rays undergoing both specular and diffuse interactions. The algorithm is "environment-driven" because rays are traced from the ray source according to the presence and distribution of obstacles in the surrounding space, therefore adapting ray density to the environment's characteristics. The environment is discretized into simple regular shapes to facilitate faster geometric computations, to allow for visibility preprocessing and for the algorithm to be parallelized in a straightforward way. These innovative features combined together and implemented on a NVIDIA Graphical Processing Unit (GPU) are shown to speed-up computation by several orders of magnitude compared to more conventional algorithms, while retaining a similar accuracy level. The speed-up and prediction accuracy achieved in reference cases is presented in comparison to a pre-existing ray-based model and RF-coverage measurements.**

Key words — Radio Propagation, Ray Tracing, Ray Launching, Cellular Radio, GPU

I. INTRODUCTION

Radio propagation models for path-loss prediction, such as [1], where developed since the eighties and nineties of the past century to assist the deployment of public mobile radio systems. After that, there have been a long line of measurement campaigns that reduce the small area averaged received signal strength to a simple power law range dependence together with lognormal shadow fading, see for example [2]. However, when the base station antenna is at or below the surrounding buildings, shadowing becomes even more important in determining signal strength. Since the 1990's site-specific deterministic RF propagation prediction models using ray-optical approximations [3] where introduced and tested with success [4]-[7]. With respect to simpler path-loss models, they also had the potential to simulate multipath propagation, and therefore the dispersion characteristics of the radio channel in both time and space at the same time.

More recently, due to the advent of Multiple-Input Multiple-Output (MIMO) transmission schemes and to the use of higher frequency bands, ray-based models have been proposed for a variety of uses, including the design and planning of new-generation wireless systems, fingerprinting and multipath-based localization methods [8],[9], and real-time use for the optimization of wireless systems performance [10]. Nevertheless, they still have not achieved widespread application mainly because of their high computation time.

Although classification criteria and terminology are not unequivocally defined, ray-based methods can be divided in two main groups [4],[11],[12]:

(i) *Ray Tracing (RT)*: Algorithms where the exact position(s) of both the transmitting (Tx) and the receiving (Rx) point(s) are taken into account from the beginning, and thus rays satisfying Geometrical Optics (GO) rules [13],[14] for those specific Rx/Tx locations are searched using various techniques, including the image method and the so called *visibility tree* technique [9]. Sometimes RT is also referred to as "image-Ray Tracing".

(ii) *Ray Launching (RL)*: Algorithms where rays start from the Tx with a pre-determined angular separation and are traced regardless of the position of the Rx(s). These rays are ideally propagated by the algorithm along their trajectory until they encounter an obstacle, where they are reflected, diffracted, transmitted or scattered. The subsequent rays are propagated following GO rules. RL is also referred to as brute-force ray tracing, shooting and bouncing rays, pincushion or beam-launching method [4],[11],[15]. Since rays are not traced specifically to an exact Rx position(s), a space discretization is assumed that in principle limits field prediction accuracy. When a ray is traced and its field is computed, it represents the field over an entire ray tube volume. Therefore RL is more efficient – although theoretically less accurate – than RT to perform RF coverage prediction over vast areas.

Some RL implementations, at times referred to as *ray-tube tracing, beam tracing* or *frustum ray tracing,* make use of ray-tubes of properly arranged triangular or polygonal cross sections so as to realize tessellation of the whole space [16]–[18]. In these methods ray-tube cross sections are properly cut using polygon clipping techniques when a ray tube propagates across the edge of an obstacle in order to exactly determine which portion of the tube hits the obstacle and which should continue undisturbed forward-propagation. These methods can achieve a high accuracy level, but the discretization error is always present and can only be reduced by launching a very high number of beams with small cross section. Moreover, polygon clipping or similar algorithms to clip ray tubes are computationally expensive.

Both RT and RL suffer from long running times because of the many rays that must be tested in order to find the few that contribute. In the case of RT each wall in the entire building database must be examined to find those intersected by relevant rays. On the other hand, RL

E. M. Vitucci, V. Degli Esposti, F. Fuschini, M. Barbiroli are with the Alma Mater Studiorum - Università di Bologna, Dipartimento dell'Ingegneria Elettrica e dell'Informazione (DEI), IT-40136 Bologna, Italy (e-mail: enricomaria.vitucci, v.degliesposti, franco.fuschini, marina.barbiroli @unibo.it).

J. S. Lu and J. A. Blaha are with Polaris Wireless, Inc., Mountain View, CA, USA (email: jlu, jblaha @polariswireless.com).

H.L. Bertoni is with the NYU Wireless Center at Tandon School of Engineering, New York University, Brooklyn NY, 11215 USA (hb752@nyu.edu)

may spend time to track a lot of rays providing no contribution at the Rx(s) in practice. Trying to include diffuse scattering from building surfaces in either RT or RL raises computation time even further.

In order to improve running time, and include diffuse scattering in simple way, the present Discrete Environment-Driven Ray Launching model (DED-RL) has been developed during a collaboration between the University of Bologna and Polaris Wireless Inc. In addition to seamless space tessellation, other advanced features have been implemented in order to achieve a very high accuracy while drastically reducing computation time, as listed below.

1) *Environment discretization*. Environment surfaces (walls streets, etc.) are discretized into "tiles" to simplify the ray launching algorithm and perform basic operations such as beam cross-section cutting across obstacles' edges without complex polygon clipping algorithms (see section II). A similar solution is proposed in [19], but there the scenario is rasterized into cubes instead of surface tiles, as proposed here. Discretization also opens the way to the following features 2) and 3).

2) *Environment-driven ray launching*. Efficiency has been improved by launching rays on the base of the geometrical distribution of the obstacles (tiles) present in the environment, instead of using a constant angular discretization: this feature might be called "environment driven RL" and is particularly useful for outdoor application where obstacles are sparse or there are large open sky sectors where launching many rays would be useless.

3) *Visibility preprocessing*. Since the environment is simplified into a set of tiles, the potential existence of a propagation path between a generic pair of tiles – i.e. a "*visibility relation*" – can be pre-computed for a given environment and saved in a file (the "*visibility matrix*"), thus greatly simplifying and speeding-up the computation of multiple-bounce rays at run-time.

4) *Parallelization on a Graphical Processing Unit (GPU)*. Since RL approach is inherently fit to parallel computing [18]–[25] the whole algorithm – including visibility preprocessing – has been parallelized on NVIDIA GPUs using the CUDA language extension, thus further reducing computation time with respect to traditional implementations.

Although almost all the features listed above have been to some extent already studied and implemented in the past, to the best knowledge of the Authors they have never been combined into a single model where synergies amplify the advantages of individual solutions in order to reach an unprecedented level of computational efficiency for a 3D ray-based approach.

The new model has also inherited some advanced features from a pre-existing RT model developed at the University of Bologna [10], such as the Effective Roughness diffuse-scattering model [26].

Using a NVIDIA K80 GPU platform and environment discretization with approximately 10x10m tiles, DED-RL can handle accurate prediction over all building surfaces of a 20 km^2 dense-urban area with a computation time lower than one hour virtually without memory swapping.

The main purpose of this paper is to present the new algorithm with its innovative features and to explain why such features work in synergy to achieve high computational efficiency. Therefore most of the results in Sections III and IV show the computation time speed up achievable using both individual features and all of them combined in the complete algorithm. As benchmarks to validate the performance of the new RL model we use the cited pre-existing RT model and RF coverage measurements performed by Polaris Wireless.

The main features of the DED-RL algorithm are presented in section II and described in more detail together with GPU parallelization and speed-up characteristics in section III. Some efficiency and accuracy validation results are then shown in section IV. Conclusions are finally drawn in section V.

II. Algorithm Principles

As with all RL algorithms, DED-RL is suitable for prediction over large areas or volumes. More specifically, it has been designed to perform fast deterministic propagation prediction on 3D outdoor surfaces of all buildings and streets in a given prediction area (or "target area"), to enable multi-frequency RF coverage design and optimization, or the application of RF-based localization methods in urban environments such as fingerprinting techniques [7]. Prediction on outdoor surfaces can be extended indoors using empirical or deterministic models [27]. DED-RL's input requires location and antenna details of all transmitters, a 3D building vector database and a raster terrain-database.

A. Environment-driven vs. traditional ray launching

In conventional RL algorithms, rays are launched from the Tx – or from diffraction and scattering Virtual sources (VTx) [10] – into the surrounding space according to a pre-set angular discretization with a uniform ray density regardless of the obstacles' positions (see Fig. 1 where the discretization grid is triangular). As the rays propagate in space, intersections with surrounding objects (walls/edges) are searched in order to determine obstruction and bouncing.
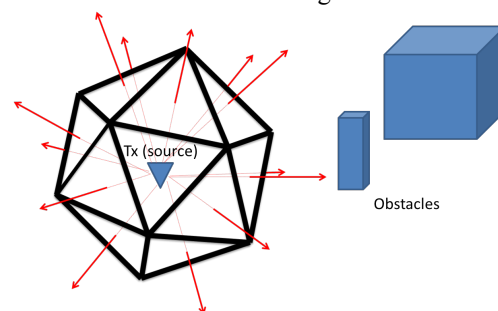
Fig. 1. Traditional Ray Launching: ray tubes and center rays launched from the Tx according to a pre-set angular discretization, independently of the obstacles in the environment. In this case: triangular cross-section ray tubes are used. Rays can also be launched through triangle vertexes instead of centroids.

As a result there can be oversampling with multiple rays in regions where obstacles are not present, e.g. rays at elevated angles in outdoor applications were large open-sky sectors are present, and under-sampling where obstacles are smaller than the distance between rays (ray

spacing). Under-sampling especially occurs far away from the Tx as ray spacing increases with distance. In conventional RL algorithms, "beam splitting" techniques must be enforced after a given propagation distance or according to a given criterion to artificially increase the density of rays and reduce the discretization error.

In environment-driven RL algorithms rays are launched only toward obstacles (Fig. 2), i.e. where they are actually needed to properly describe the field, and with a density that is proportional to the spatial density of obstacles in each direction. Each ray-tube cross section is inherited from the shape of the reflecting obstacle (rectangular in Fig. 2). This solution does not require beam splitting because spatial resolution is automatically matched to environment complexity at each new bounce and does not degrade with distance. With this solution, the total number of rays to be launched is minimized. Moreover the algorithm can keep track of the exact cross-section geometry of each ray tube bounced off, or partially obstructed by, a wall, by clipping it according to the wall's shape. Therefore field representation accuracy can be kept to the maximum level.

This kind of environment-driven algorithm however has the following drawbacks:

a) polygon clipping algorithms, necessary to determine the polygonal cross section of ray tubes, are computationally intensive

b) accuracy degradation might occur when obstacles are very large or very close to the Tx, where the incidence angles and therefore the reflection/diffraction coefficients might appreciably vary over different points of the same obstacle.

c) no ray (or ray section) can be pre-computed for a given environment independently of the position of the Tx because the cross-section shape of each ray tube after one or more bounces (and clipping operations) depends on the position of the Tx.
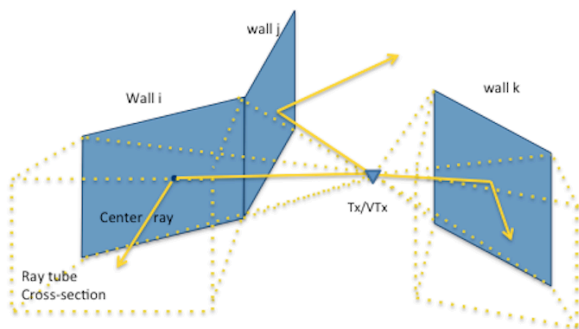
Fig.2. Environment-driven Ray Launching: rays (thick dashed lines) or corresponding ray tubes (thinner dashed lines) are launched only toward obstacles, here generating reflected rays

B. The advantages of discretization

All the above-mentioned drawbacks are solved or at least eased using environment discretization and a discrete RL algorithm: this is another distinctive feature of the DED-RL algorithm.

Each wall is subdivided into rectangular tiles (or pixels) with a typical size of some square meters using the algorithm described in section III.A (see Fig. 3). Since tiles are relatively small, their center-points (stars in Fig. 3, also called centroids) can be considered representative of the whole tile and all operations can be performed in a simplified pixel-based fashion using the centroid instead of the tile's surface. As walls are discretized, ray tubes bouncing off-walls also become discretized and therefore the use of polygon clipping algorithms can be avoided. Of course there is a discretization error with respect to a fully analogue, environment-driven RL, but it can be kept low enough using small-enough tiles.

One drawback is a higher number of surfaces to be considered, and therefore rays to be traced. This problem is mitigated by the use of parallel computation. In fact, discrete ray launching is very suitable to parallelization as a large number of similar operations must be performed at the same time and can be parallelized on multi-core computing platforms such as GPUs. This solution has been chosen for DED-RL, as explained in detail in section III.
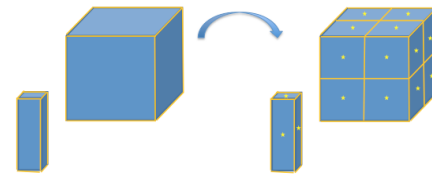
Fig. 3. Environment discretization example

Last but not least, discretization opens the way to another very important feature that has been implemented in DED-RL: visibility pre-processing.

A great deal of computation time in RL algorithms is spent in determining the visibility of objects, i.e. the existence of a Line-of-Sight (LoS) propagation path between a Tx/Vtx and a given object, or portion of it, as rays are bounced only at visible objects. For example, one wall might be only partly visible from the Tx as an interposed object might obstruct. Since the starting view-point for visibility is the Tx, in conventional RL algorithms visibility and therefore rays can only be determined when the Tx position is known. Not so in discrete RL. Since objects are tiles, the mutual visibility between a couple of tiles – and therefore the potential existence of a ray – can be determined a priori based on the environment topology and regardless of the position of the Tx. Therefore a given discretized environment database can be pre-processed and visibility relations between each pair of tiles can be saved in a matrix, the *visibility matrix*, in order to speed-up the tracing of rays at a later time for one or more Tx positions. Of course direct visibility from the Tx, also called *first-level visibility*, must be re-determined for each Tx position, while subsequent visibility levels corresponding to one or more bounces, can make use of information stored in the visibility matrix.

Consider for example the case shown in Fig. 4 where visibility determination and tracing of rays up to the second level (after 1 bounce) is shown for a simple case. In Fig. 4(A) first-level visibility from Tx and corresponding rays are shown for wall w1 and one

obstructing object only (other objects are neglected here for simplicity). In Fig. 4(B) tile t1 of wall w1 (spawning tile) is considered and second level visibility is determined using visible tiles from t1 stored in the visibility matrix and the reflection cone that impinges on surrounding obstacles. Only a few objects are shown for simplicity. We can focus on target wall w4 to determine the visible portion of it and ultimately which rays need to be traced after one bounce off tile t1.

(A)

w1

Non visible tiles

Tx (source)

(B)

VTx

b1

spawning tile (t1)

w1

non visible tiles

b3

w3

reflection visibility region (cone)

shadow boundary

b2

Tx (source)

non visible tiles

visible tiles

w4 (target wall, back view)

(C)

VTx

spawning tile

w1

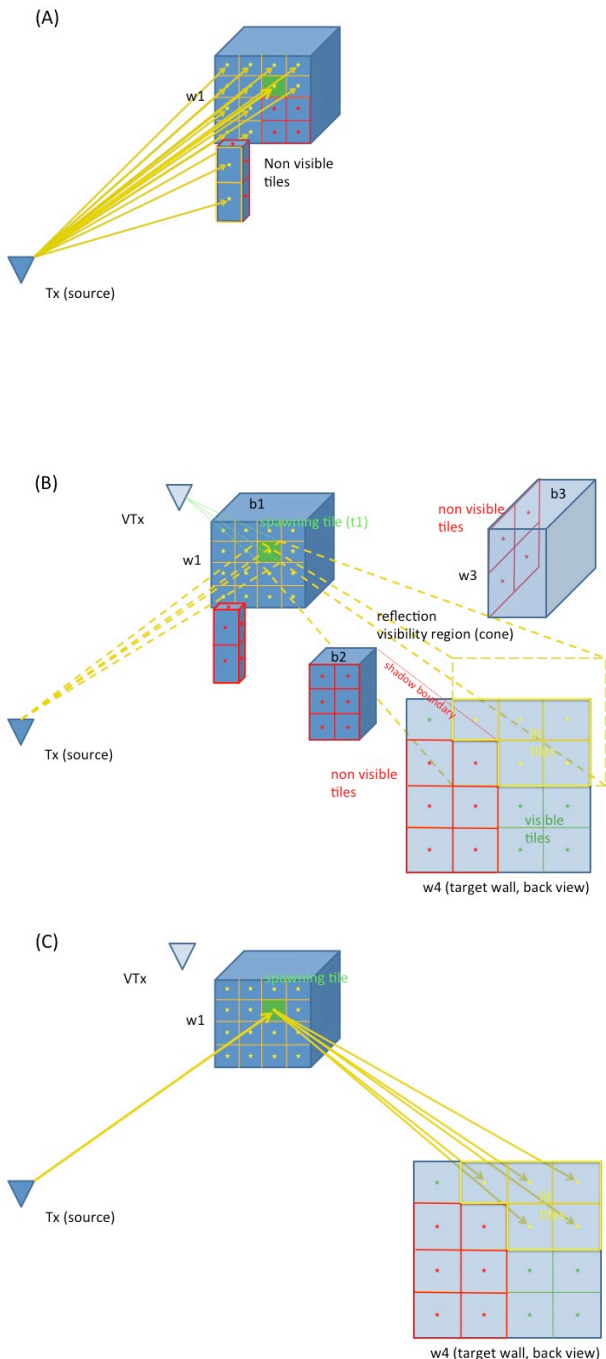Tx (source)

w4 (target wall, back view)

Fig. 4. DED-RL functioning example: (A) first-level visibility from Tx and corresponding rays; (B) spawning tile, related visible and non-visible tiles, reflection visibility cone and lit tiles; (C) traced rays corresponding to (B) for wall w4 only.

In traditional RL algorithms the whole environment (in practice thousands of walls!) would have to be checked for visibility from the spawning tile. Here,

since the visibility matrix has been pre-computed, tiles (centroids) visible from the spawning tile are known and highlighted in green: tiles of wall w3 are not visible because building b1 itself shadows them, while some of the tiles in wall w4 are shadowed by building b2. Not all the remaining, 10 visible tiles must be considered for launching rays however, because only some of them fall within the reflection visibility region (reflection cone), i.e. are visible through reflection from tile t1.

The reflection cone shape is easily determined with a projection using the VTx position and the tile's shape and all tile centroids within the cone can be easily determined by checking only the 10 visible tiles' centroids. Therefore the visible portion of wall w4 through reflection from the spawning tile is determined by the 5 lit tiles highlighted in yellow. As in digital computer graphics, such a wall portion is determined without any polygon clipping, simply as a composition of pixels.

Rays to be traced for this computation step are shown in Fig. 4(C): the 5 bounced rays from tile t1 addressing the lit tiles of wall w3. Note that both ray splitting (to keep ray-spacing within a limited range) and polygon clipping (to determine the lit portion of walls at the generic computation step) are avoided and replaced by this discrete, environment-driven RL procedure.

If diffuse scattering from tile t1 instead of reflection is considered the pre-processing speed-up advantage is even greater because visible tiles through diffuse scattering exactly correspond to the visible tiles stored in the visibility matrix since diffuse scattering's visibility region corresponds to the whole 2π external hemisphere [10]. Similar considerations would hold for diffraction off a tile's edge, where the visibility region is a large region comprised between the two Keller's cones [14] corresponding to the edge's end points (see Fig. 5). Note that edges, like surfaces, are also discretized into segments. Edge segments correspond to the common edge side of a pair of border tiles located on two walls sharing a common edge. Border tiles are properly flagged when discretizing the environment, and the IDs of the 2 tiles generating the edge segment (*parent tiles*) are stored. With this approach, no operation is needed to determine the visibility of edges: edge visibility is the OR of the parent tiles' visibility.

upper Keller's cone

diffraction visibility region
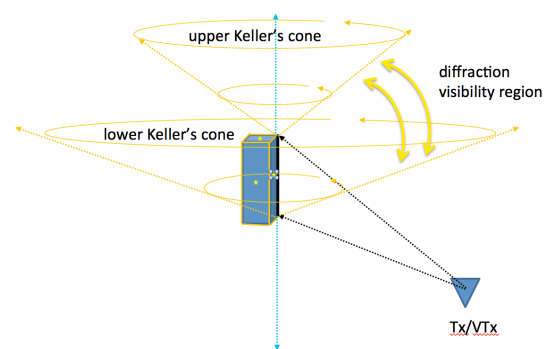
lower Keller's cone

Tx/VTx

Fig. 5. Visibility region for diffraction off a tile's edge. The region is comprised between two Keller's cones.

In order to determine tiles visible *from* an edge, the visibility matrix is queried for both of the parent tiles and the union of all tiles visible from parent tiles is made. Then, only those visible tiles are selected that fall inside the diffraction visibility region, (Fig. 5).

At each iteration step each ray's field and other information such as the propagation delay, the ray geometry etc. are saved in an output file. Note that, as long as the field is only needed on environment surfaces (radio coverage prediction), no operation to determine which ray tubes hit a given Rx point must be performed here: tiles represent both bouncing and target elements (Rx points) at the same time and therefore the field contribution can be automatically saved at each computation step whenever a ray addresses a tile.

More details on the different features of the DED-RL algorithm are given in section III. An overall scheme of the DED-RL algorithm with the different input, output and computational blocks is shown in Fig. 6. The RL engine uses the following inputs: tile geometry information from the discretization algorithm, tile visibility information from the visibility matrix and the Tx/Rx data (position, antenna diagram and polarization, radiated power etc.). The outputs, which can include total field strength and received power as well as each ray's field, delay and geometry for each Rx point, are saved in the output matrix
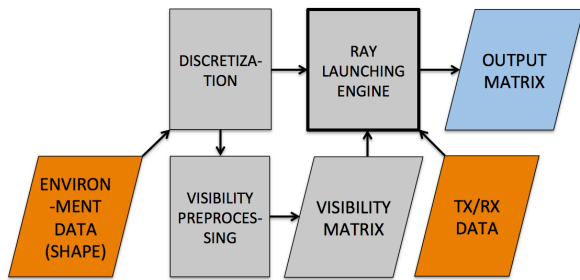

Fig. 6. Simplified scheme of the DED-RL Algorithm. Input files in orange, output file in blue.

III. ALGORITHM DETAILS AND IMPLEMENTATION

In this section, select key features of the DED-RL algorithm are described in some detail and their contribution to the algorithm's efficiency is discussed.

A. Environment Discretization

Propagation prediction requires terrain, clutter, vegetation and building GIS databases in order to accurately account for significant propagation effects. In this work only terrain and building databases are considered, but clutter and vegetation can be considered as well.

Terrain databases are commonly available in the form of raster databases created from satellite imagery. These raster databases are essentially terrain heights uniformly sampled in an area. The horizontal resolution of the sampling depends on the data source. For example, Shuttle Radar Tomography Mission (SRTM) data is available in 30 m resolution [28].

Building databases on the other hand are usually given as a set of right prisms with polygonal base and vertical sides, so that a single building is represented by one or more of them stacked on top of each other. These databases are also known as building vector databases that usually come in the ESRI SHAPE format [29] with varying accuracy. As an example, the Google street view of a building is shown in Fig. 7(a) and the corresponding prisms outlines are drawn in red in Fig. 7(b).
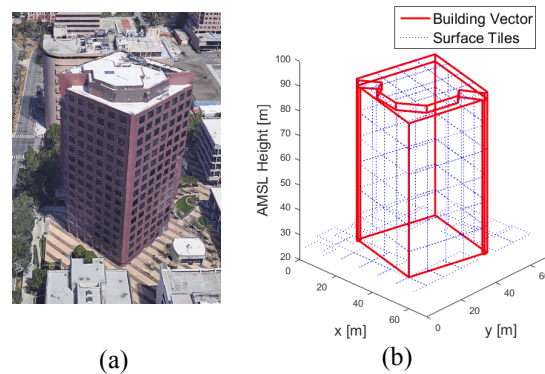

(a)   (b)

Fig. 7. (a) Example building and the surrounding terrain; (b) Tiling the surfaces of the example building and the surrounding terrain.

To tile the terrain and building surfaces a tile shape (e.g., triangle, rectangle) and desired tile area ΔA should first be chosen. Every surface is then overlaid with a uniform grid of non-overlapping tiles. As far as buildings are concerned, in the present work we tile SHAPE-format building walls with as-square-as-possible rectangular tiles with the following procedure.

For vertical building walls, which are rectangles of sides u,v, we first define the desired tile-side length Δl as follows

$$\Delta l = \sqrt{\Delta A} \qquad (1)$$

i.e. the side length of a square tile of area ΔA. Then we divide each of the wall sides by Δl and save both the integer quotient q and the reminder length l_R, so that for example:

$$u = q \cdot \Delta l + l_R \qquad (2)$$

The side "u" is then subdivided into segments of length $\Delta \ell'_u$ as follows:

$$\begin{cases} \Delta l'_u = u/q & \text{if} \quad l_R \le \dfrac{\Delta l}{2} \\[2mm] \Delta l'_u = u/(q+1) & \text{if} \quad l_R > \dfrac{\Delta l}{2} \end{cases} \qquad (3)$$

In this way we ensure that the subdivision length $\Delta l'_u$ of wall length u is close enough to desired length Δl. This discretization procedure also generates the discretization of the wall edge already presented in section II.

Once we have discretized both dimensions u, v of the wall into segments of length $\Delta l'_u, \Delta l'_v$, respectively, using equation (3), we have obtained a discretization of

the wall into $\Delta l'_u \times \Delta l'_v$ tiles. Such tiles are as similar as possible to squares of side Δl. If a given wall is too narrow, or its area is too small according to given minimum parameters, then the wall is dropped from the tiling procedure and is therefore disregarded in the RL algorithm.

For polygonal roofs, we draw the minimum circumscribed rectangle, and proceed with the tiling procedure as described above. At the end of the process all tiles whose centroids falls outside the roof's polygon are discarded. Conversely, for terrain tiles, all tiles whose centroids falls inside the building's footprint polygon are discarded.

For the building shown in Fig. 7, the tiles are depicted in Fig. 7(b), with a desired tile area $\Delta A=100$ m^2. Note that $\Delta A=100$m^2 is used for most RL simulations in the present work, except where specified otherwise. For the simulations discussed in Section IV, 100 m^2 rectangular tiles were used to tile the 20 km^2 area surrounding the high-rise core of San Francisco. This resulted in 466,529 tiles, which corresponds to ~23,000 tiles/km^2. Overall computation time for tiling was ~5 s on a standard PC using a 3.2 GHz, INTEL Xeon CPU.

Of course, approximation of the surface shape using tiles is a source of error, especially for roofs. To mitigate this error, the tiling algorithm can orient tiles and use a variable tile area.

The choice of the proper ΔA is obviously a trade-off between computation time and accuracy. According to our investigations $\Delta A=100$ m^2 can be considered a good compromise. In fact, considering a medium-sized urban database with 30 different cellular sites – not shown here for brevity – using $\Delta A= 25$ m^2 instead of $\Delta A=100$ m^2 can yield a small decrease of about 1 dB in prediction error's standard deviation with respect to measurements, but at the cost of a nearly quadruple computation time.

B. Visibility Pre-processing

A ray incident on a tile (spawning tile) can only bounce to tiles that belong to a subset of those visible to that spawning tile. Visibility is predetermined on the base of the visibility of the centroids of the considered tiles and saved in the visibility matrix as presented in section II.

If there are N tiles, the pre-processed visibilities for all tiles can be saved in a NxN visibility matrix of ones and zeros where the ith row or column corresponds to the ith tile. For example, if the ij element in the matrix is 1 it means that the ith tile is visible to the jth tile and vice versa.

As an example, for a 20 km^2 dense urban environment (central S. Francisco), using GPU parallelization it took an average of 69 ms to compute visibility from a tile for a total of approximately 9 hours. The resulting average number of visible tiles per spawning tile is 952. The amount of memory needed to store the matrix depends on the variable type (e.g., Boolean, character). If variable type is Boolean (1 byte each), the amount of memory occupation for the matrix is N^2 bytes. For the example, (466529)^2 bytes = 202.7

GB are needed. Thus, depending on the size of the environment, the size of the matrix can be prohibitively large to access or keep in RAM.

Because the matrix is generally sparse (~0.2% non-zero for the example), a sparse storage method such as *compressed sparse row* (CSR) [29] is used in the present work. This reduces the memory size dramatically (by ~99.2% to 1.65 GB for the example discussed here).

The benefit of using the pre-processed visibility matrix is significant with respect to the case when the visibility check is done "online" while tracing the rays. In fact, according to our calculations it would takes ~10ms to check the obstructions for each ray segment, and this will need to be repeated for every single ray. Thus, the computation time saved scales directly with the number of rays considered in a simulation. If a single TX with a maximum of 3 bounces per ray is considered for the 20 km^2 area in San Francisco, ~10^8 rays need to be computed, and ~80 days would be needed only for visibility calculations!

The drawbacks to using a pre-processed visibility matrix are the initial pre-processing time and memory needed to store it, but these figures can be kept reasonably low using the methods described above. Moreover visibility matrix computation must be done only once for a given environment database.

As an example, in Fig. 8 the computation times and the memory occupation for different map sizes are shown, with reference to the San Francisco scenario discretized into relatively small tiles of $\Delta A=25$ m^2. It is evident from the figure that even in this challenging case pre-processing is feasible for an urban area of 10 km^2, with a limited effort in term of computation time (less than one day), and memory occupation (about 10 GB, which can be handled by modern PCs).
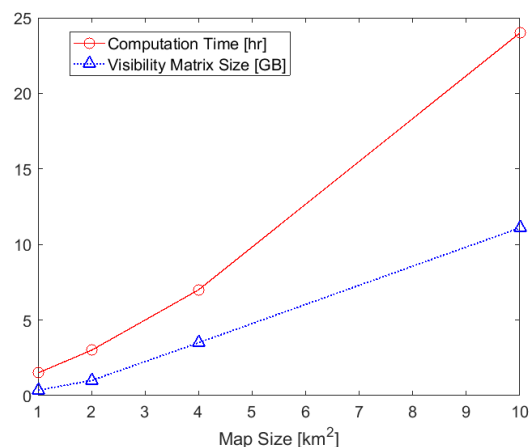
Fig. 8 – Preprocessing Computation times and memory occupation for different sizes of the map, assuming an average tile area $\Delta A=25$ m^2

C. Bouncing and saving the field

As mentioned in section II, one peculiar feature of the DED-RL is that bouncing and saving the field are performed simultaneously at each computation step.

First of all, at each iteration the centroid of the current visible tile is set as a Receiver, then the incident field from each ray is computed, and the total incident ray field, including time delay and angle of arrival on

the tile is stored. When computing the total field strength a simple incremental vector addition can be performed without any further memory occupation in the output matrix.

The field contribution for each receiving tile is computed taking into account the current ray trajectory as well as the number and the kind of bounces (or interactions) it has undergone on the whole. The impinging field for a single ray is therefore computed as [31]:

$$\vec{E}(RX) = SF \cdot \left(\prod_{i=1}^{N_b} \underline{\underline{C_i}} \right) \cdot \vec{E}_0 \cdot e^{-j\beta r_{tot}} \tag{4}$$

where $\vec{E}_0$ is the field emitted by the Tx antenna in the direction of departure of the ray, and $\underline{\underline{C_i}}$ is a dyadic coefficient (Fresnel reflection coefficients, UTD diffraction coefficients, or scattering coefficients) accounting for the amplitude reduction and phase shift experienced by the propagating field at the i-th interaction. In (4) SF is the spreading (or divergence) factor representing the overall propagation loss along the ray due to the spatial broadening of the wavefront [11]. Also, N_b is the number of bounces (or interactions) undergone by the ray, β is the wave number and r_{tot} is the total length of the ray. The Tx antenna radiation pattern (directional gain) is accounted for in eq. (4) through the field amplitude $\vec{E}_0$, i.e. the field emitted by the Tx antenna assuming far field conditions ($r > 2A/\lambda$) at the incident tile, as usual for ray tracing models. In case far-field conditions are not satisfied, the corresponding tiles can be excluded from field computation, or the Tx antenna could be decomposed into smaller radiating elements.

After incident field computation, bouncing is applied, i.e. each visible tiles becomes a "spawning" tile: the visibility matrix is queried to find the potentially visible tiles from the spawning tile, and then the actually visible tiles are selected among them with simple processing operations, according to the different interaction types.

Reflection: In order to determine if a visible tile belongs to the reflection cone, the center of this tile is back-projected to the wall plane of the spawning tile following the line that connects it with the VTx, and we then check whether the projected point falls inside the spawning tile or not.

Diffraction: Using a similar processing we check whether the tile's centroid falls inside the visibility region shown in Fig. 5 for the current VTx using a ray-unfolding technique [10].

Diffuse Scattering: No further visibility processing is needed for diffuse scattering, since all the visible tiles stored in the visibility matrix can be reached through scattering.

Finally, the visible edges are automatically set by identifying the "border" tiles among all the visible tiles, as mentioned in Section II.

Once the visible tiles and edges are found according to the different interaction mechanisms, the incident field for all of them is computed, and the procedure described above is iterated.

The tracing of a multiple-bounce ray continues until the incident power falls below a minimum power threshold, or when the maximum number of interactions N_{bMAX} is reached. Both the minimum power threshold and the maximum number of interactions are input parameters of the DED-RL algorithm.

D. GPU parallelization

All the main features of DED-RL algorithm described in the previous sections – visibility preprocessing, first level visibility, ray bouncing and field computation – are suitable for code parallelization.

Traditional parallelization techniques are based on multi-threading and multi-core computing, exploiting the characteristics of modern processors. More sophisticated techniques rely on the use of multi-processor architectures, computer clusters, distributed and grid computing.

In recent years, an alternative approach has become widespread, which is based on the use of the Graphics Processing Unit (GPU) - in addition to the CPU - for general purpose computing. GPU computing is often used in complex mathematical and geometric calculations, due to its ability to process vectors or matrices with extreme efficiency. This great speed and power for mathematical calculations comes from the fact that modern GPUs have more processing circuits with data caching and basic flow control than the CPU. GPU computing is particularly suitable for problems that may be expressed in terms of intensive computations on multiple parallel data, i.e. the same operations are executed for each data element in parallel. Such approach is often called GP-GPU (General-Purpose computing on Graphics Processing Units), and it is the one adopted for the present work: all the features of the DED-RL have been implemented for parallelization on NVIDIA GPUs using the CUDA C++ language [32].

CUDA (acronym for Computer Unified Device Architecture) is a computing platform and a model of parallel programming that allows a dramatic increase in performance by exploiting the computing power of the CUDA-enabled GPUs. The CUDA platform is a software layer that gives direct access to the GPU's virtual instruction set and parallel computational elements for the execution of compute kernels. It is specifically conceived for NVIDIA GPU architectures. CUDA processors are usually programmed in CUDA C/C++, which is basically the standard C/C++ programming language, with some CUDA extensions [32].

In CUDA C/C++, the code is executed on the GPU using special functions called "kernels", which are synchronous computation "batches" containing the parallel code. Each instance of the parallel code is executed by a "thread", and several threads are grouped into "blocks". Different thread blocks constitute a kernel "grid". This is a "logical" subdivision, of course. From the physical point of view, each thread is mapped onto a core of the GPU, and each thread block is assigned to a different "Streaming Multiprocessor" (SM). When a kernel is launched on the GPU, the blocks are distributed to all the available SMs. The threads of a block execute in

groups of 32 threads, known as warps, and multiple warps can execute concurrently on a SM, as long as resources (registers, shared memory, etc.) are available. As soon as a thread block terminates its execution, it is immediately replaced by a new block. It is responsibility of the programmer to create enough blocks to keep all of the SMs occupied. Of course, the maximum number of blocks and threads that can run in parallel, depends on the number of SMs and CUDA cores available on the GPU.

The Tesla K80 GPU Accelerator used in the present work has 2 GPU units, each one equipped with 12 GB of internal RAM memory, 13 SMs and 2496 CUDA cores. Among all the operations done by the DED-RL algorithm, the geometrical calculation of visibility between tiles is one of the most computationally expensive. In order to do this operation in an efficient way, well known acceleration methods such as kd-trees or Bounding Volume Hierarchies (BVH) can be used, and several implementations of these methods on the GPU have been proposed in the literature. In our case, we implemented on the GPU the kd-tree method with a short stack in order to handle the GPU resources in an efficient way, similarly to what done in [18].

The parallelization potential of the visibility computation is shown in Fig. 9 for a simple task: computation of the visible tiles from a Tx (first level visibility). The full 20 km^2 map of San Francisco is used, and 6 different Tx sites with different characteristics (micro- and macro-cellular) and located in different zones of the city have been considered (Tx A, B, C, D, E, F). For each site, the computation time with parallel computation (red bars in the histogram) is compared to the reference case of computation using a single block - single thread kernel, which in means in practice that no parallelization is applied (blue bars in the histogram). The computation time for first level visibility ranges from 30ms to 180ms depending on site and environment, and the achievable parallelization speed-up ranges from ~500x to ~1600x.
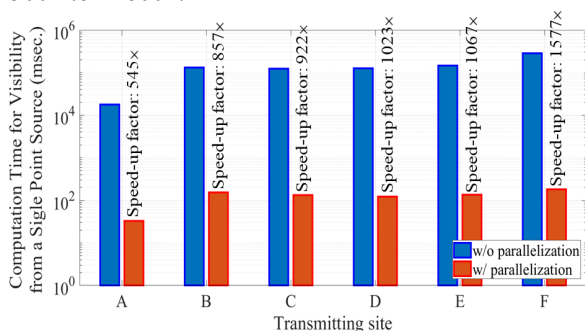

Fig. 9. Parallelization speed-up factors of first level visibility for different sites in San Francisco. Blue bars refer to computation time using a single block-single thread kernel.

It is worth noting that the highest speed-up gains are achieved for macrocellular sites (Tx D,E,F), which are able to "see" a higher number of tiles, since they are located above the average height of the surrounding buildings. In such cases the parallelization potential of the GPU can be fully exploited, with a better occupation of the available resources and a good balance in the execution times of the different threads. For example, Tx F is a macrocellular base station located in a very dense

urban area close to Union Square and surrounded by very tall buildings, and in fact we get the highest parallelization gain in this case.

E. Flow-Chart of the DED-RL engine

The flow-chart of the DED-RL core, corresponding to the "Ray Launching Engine" block of Fig.6, is shown in Fig. 10. The algorithm consists in the following steps:

a. The tiles visible from the Tx antenna are found
b. Initial ray tubes are launched toward the visible tiles
c. The power and delay of the center rays to the visible tiles' centroids are computed and saved.
d. For each visible tile, if the incident power at the centroid is greater than a user-specified threshold and the maximum number of allowed bounces is not exceeded, the corresponding ray tube is allowed to be bounced.
e. If there are no ray tubes to be bounced, the program ends
f. For ray tubes that are allowed to bounce, the geometry of the bounced ray tubes are found.
g. For each bounced ray tube, using the preprocessed tile visibility matrix those tiles that are visible to the spawning tile and also lie inside the bounced ray tube are found.
h. The original bounced ray tubes are split into multiple bounced ray tubes that are incident on the newly found tiles.
i. Restart from c), where the initial ray tubes are now the bounced ray tubes found in h), and the visible tiles are the tiles that the bounced ray tubes are incident on.

The computation of visibility (a), propagation characteristics computation (c) and ray tube bouncing (f & g) are the most computationally intensive parts of the algorithm, and therefore they are run in parallel into CUDA kernels (highlighted in red in Fig. 10).

IV. PERFORMANCE FOR WIDE-AREA PREDICTION

Some tests were initially carried out to ensure DED-RL results were in agreement with theory in some simple, basic environments, such as free-space with flat terrain reflection [2]. After that, to evaluate the computation speed and accuracy of the DED-RL model in realistic cases, we compared DED-RL predictions to measurements and RT predictions, using the model described in [10]. This RT model is used as a reference because it represents a classical, non parallelized image-RT algorithm developed in our research group, which can take into account the same kind of interactions as the DED-RL model, including diffuse scattering.

A. Evaluation of DED-RL Speed-up Gain

In order to evaluate the relative computation efficiency of the DED-RL algorithm with respect to RT in different parameter configurations, a 0.5 km^2 area in central San Francisco has been selected as a reference environment. The Tx site was located on the top of a building facing Union Square, and the environment was discretized using 100m^2 tiles, for a total of 5100 tiles. DED-RL computation time was 2s, 6s and 11s for 1, 2 and 3 bounces, respectively.
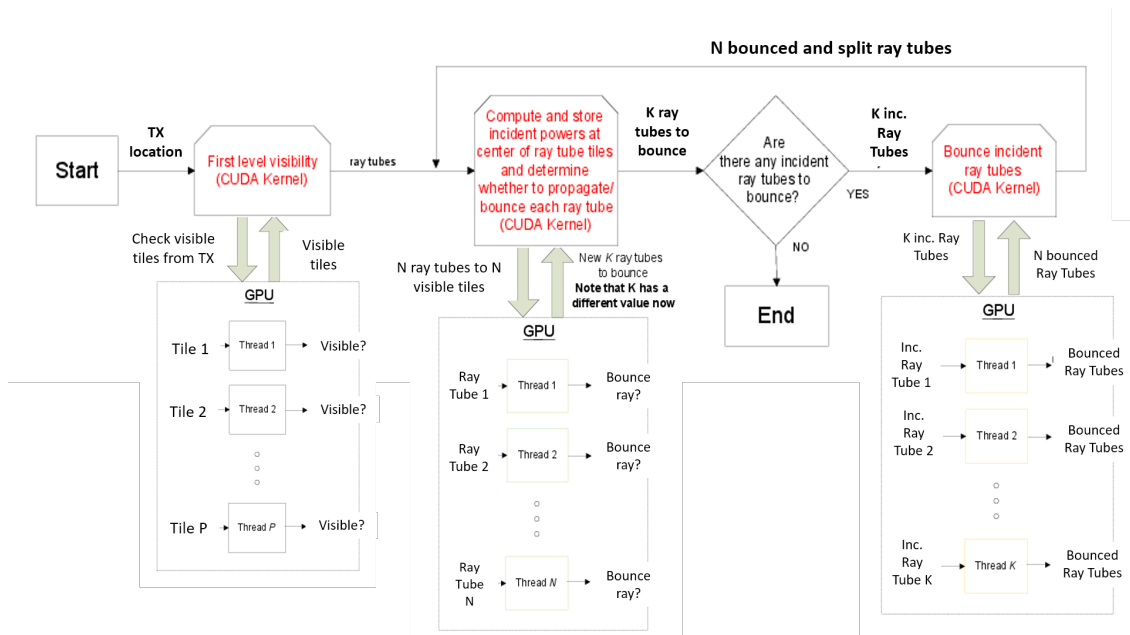
Fig. 10. Detailed flow-chart of the DED-RL engine

Since RL intrinsically performs prediction over the whole area while RT can perform prediction for a specific Rx point, in order to provide a fair and complete comparison we decided to run RT several times for different number of receivers starting from 1 up to 5100 Rx points (all the tiles in the area). DED-RL is of course more efficient for area prediction, but what if prediction on only 1 or a few points (e.g. a Rx route) is needed? Is RT faster for such a point-specific prediction?
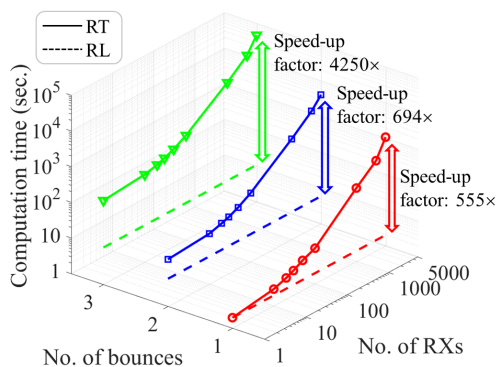

Fig. 11. Computation times and speed up factors of DED-RL with respect to non-parallelized Image-RT run for different number of bounces (Union Square Scenario). DED-RL computation times are represented with dashed lines.

In Fig. 11 RT computation time vs. number of receivers is shown using a log-log scale, while DED-RL computation time is represented as a horizontal dashed line. It is evident that RT computation time is similar to DED-RL computation time only for a single receiver and for a single-bounce prediction, while the DED-RL outperforms RT when thousands of receivers and multiple bounces are considered, with a speed-up gain of up to several thousands. This gain could be even greater – about 4 orders of magnitude – for simulations on larger areas and more bounces. This confirms that 3D

predictions on very large urban areas would be unfeasible using a non-parallelized image-ray tracing, while DED-RL is able to handle them in a few minutes.

B. San Francisco Cellular Measurements

The measurements used in the DED-RL performance evaluations are from two cell sites we will refer to as TX1 and TX2 with the characteristics reported in Table I. The cell sites are located just west of the San Francisco financial district. In the vicinity of the sites, there is large building height variation with buildings as short as 15m and buildings taller than 150m. The terrain also has large variations from 11m above mean sea level to 105m.

Table I – transmitters and receiver characteristics.

TRANSMITTERS					
	Lat. [deg]	Lon. [deg]	Band [MHz]	Height [m]	ERP [dBm]
TX1	37.7904	-122.4053	850	41	40
TX2	37.7853	-122.4080	850	34	40
RECEIVER					
PCTEL Seegull LX GSM 850 MHz scanner with a PCTEL OP178H					
Antenna type		Ominidirectional			

Two types of measurements were recorded: 1) ground level (GL) and 2) above ground level (AGL). The GL measurements were recorded as a vehicle with the scanner traveled around the streets in the vicinity of the cell sites. 2063 and 1143 GL measurements were recorded from each cell site, respectively. The AGL measurements were recorded as the scanner was carried up and down an open staircase on the side of a 120 m building that was serviced by both cell sites. A total of 13 AGL measurements locations were considered.

C. Wide-Area Prediction Accuracy

To evaluate the absolute and relative accuracy of DED-RL predictions were performed for the two cell sites over the whole 20 km^2 area which encompassed the measurement locations for both cell sites. In the simulations, the tile size was $\Delta A=100m^2$ and two different maximum bounce criteria were used: A) "3 bounces" where $N_{bMAX}=3$ with a maximum of 3 reflections, 1 diffraction and 1 diffuse scattering and B) "5 bounces" where $N_{bMAX}=5$ with maximums of 5 reflections, 2 diffractions and 1 diffuse scattering. Such criteria are considered typical for ray-based prediction in dense urban environment. It is worth noting that, due to building obstructions and to the relatively short distances, terrain diffraction is not as important in urban environment as it is in rural-environment propagation, where it often represents a dominant propagation process. In urban street canyons on hilly areas however, where the LoS path is obstructed by terrain, the diffracted path can be dominant and multiple diffractions on terrain tiles' edges must be considered in the DED-RL algorithm to achieve good results.
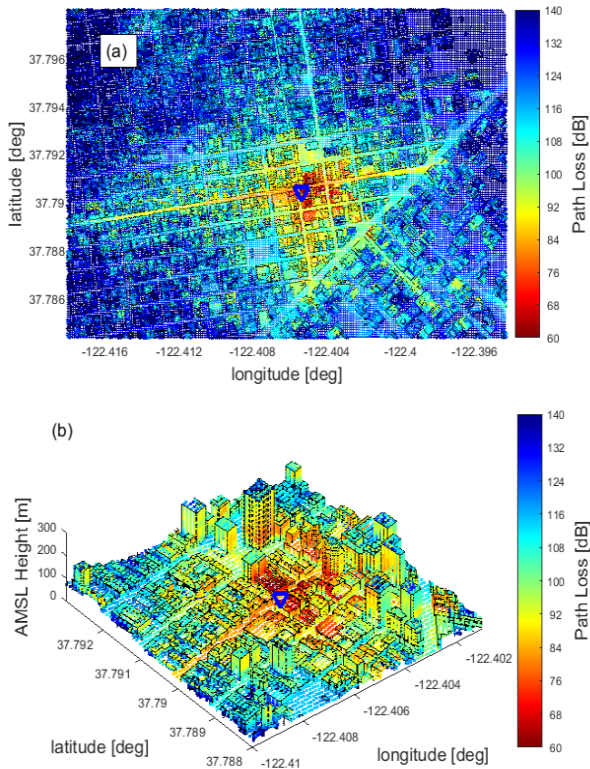

Fig. 12 (a) 2D view of DED-RL Path Loss predictions from TX1 (blue triangle) located in central San Francisco; (b) Zoomed-in 3D view of predictions in the vicinity of TX1.

Table II. RL Computation Time

Tx	DED-RL 3 Bounces	DED-RL 5 Bounces
	Time [s]	Time [s]
1	1584.0	3044.0
2	1281.0	2345.0

The DED-RL 5-bounce Path Loss (PL) predictions for TX1 are shown in Figs. 12(a), (b). Fig. 12(a) depicts a

2D view of the predictions, while Fig. 12(b) shows a zoomed in 3D view of the area around TX1. PL (dB) is defined here using the following equation: PL(dB)=ERP-P$_R$, where ERP is the Effective Radiated Power of the Base Station antenna (see Table I), and P$_R$ is received power. Note that for directive antennas the original rays from the transmitter and the arriving rays to the receiver are weighted according to the antenna gains when computing the received power. The computation times for each Tx are given in Table II. The corresponding prediction errors computed versus measurements for TX1 and TX2 are shown in Figs. 13 and 14, and tabulated in Tables III and IV. The measurements path loss ranged from 74.8 to 132.0 dB with a median of 108.1 dB.

The simulated power values corresponding to the measurement locations shown in Figs. 13 and 14 have been extracted from the DED-RL predictions by snapping the power values from the nearest tiles to each of the measurement points.

Overall, for both TX1 and TX2, the DED-RL with 5 bounces has good prediction accuracy. For TX1, mean prediction error is -3.3 dB and standard deviation of error is 8.6 dB for GL measurements, and mean error is 0.9 dB and standard deviation is 7.2 dB for AGL measurements. There are several areas in which DED-RL predictions can improve such as vegetation and street clutter attenuation and over roof top propagation. From Fig. 13, at locations on streets adjacent to TX1, signal strength predictions are generally too high. These locations were generally line-of-sight or close to line-of-sight. We expect the over-estimation to be caused by vegetation and street clutter that were not accounted for in the DED-RL predictions. Another area of improvement is over roof top propagation. An example of this is the underestimation of the received power for locations to the south-east of TX1. We think that the block grid configuration change from north/south to north-west/south-east causes rays travelling in the street canyon to have large losses, so that the dominant rays travelled over the roof tops. Though, to predict these rays, more diffractions than the maximum limit of 2 in our DED-RL predictions are needed.

The results for DED-RL with 3 bounces are also given in Tables III and IV. Comparing results for 5 bounces and 3 bounces, the DED-RL computation time is approximately half but the error standard deviations degrade a little from 8.6 to 8.8 dB. There is also more underestimation which is evident from the mean error going from -3.3 to -5 dB. This is expected due to the lower number of rays that are considered. Overall, the biggest impact on performance was not from changing the maximum number of reflections from 5 to 3, but from decreasing the maximum number of diffractions from 2 to 1. If the maximum limit of 2 diffractions was instead increased, more over-roof-top rays could exist and performance is expected to improve over the 5 bounce predictions. Unfortunately, diffraction is computationally intensive, so computation time will exponentially increase. Further work will have to address this issue with computationally efficient over-roof top models.

RT was also run for the measurement locations. DED-RL generally had slightly better performance than RT for the same number of bounces. As opposed to the DED-RL simulations, RT simulations have been done only for specific measurement locations instead of the whole 20 km^2 area, and the maximum number of bounces has been limited to 3: in fact, such predictions on very large areas and with many bounces are not feasible with RT. For comparison, RT computation time in the TX 2 case was already almost 1 day for the 1143 GL locations using the 3 bounces setting.
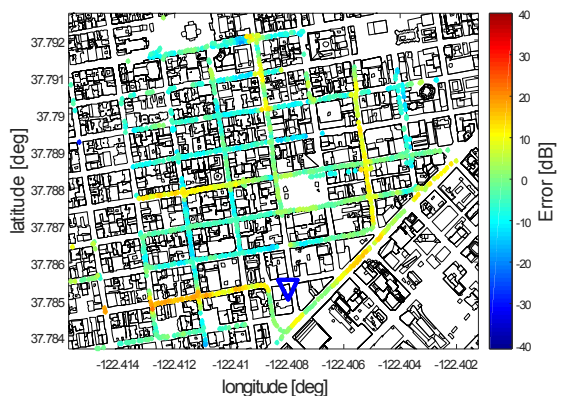

Fig. 13 DED-RL prediction error for TX1

Fig. 14 DED-RL prediction error for TX2

Table III. Ground Level Prediction Error Statistics

Tx	DED-RL 3 Bounces		DED-RL 5 Bounces		Ray-Tracing 3 Bounces	
	Mean [dB]	STD [dB]	Mean [dB]	STD [dB]	Mean [dB]	STD [dB]
1	-5	8.8	-3.3	8.6	-0.02	11.7
2	-1.6	8	-0.7	7.8	1.0	8.9

Table IV. Above Ground Level Prediction Error Statistics

TX	DED-RL 3 Bounces		DED-RL 5 Bounces		Ray-Tracing 3 Bounces	
	Mean [dB]	STD [dB]	Mean [dB]	STD [dB]	Mean [dB]	STD [dB]
1	0.5	7.3	0.9	7.2	3.0	10.4
2	-4.7	3.4	-4.5	3.5	4.9	6.0

V. CONCLUSIONS

A novel Ray Launching algorithm for fast 3D prediction in wide urban areas, called Discrete Environment-Driven Ray Launching (DED-RL), is proposed.

Using the combination of different techniques, i.e. environment discretization, visibility pre-processing and GPU parallelization, DED-RL is able to achieve very high levels of computational efficiency, up to four orders of magnitude compared to a reference ray tracing algorithm. Typical computation times for complete predictions over all building surfaces in a urban cell site range from a few seconds to tens of minutes, depending on the size of the urban scenario and the characteristics of the Tx site.

The accuracy level is found to be similar to ray tracing, despite some approximations that are intrinsic to the discrete approach as compared to the more rigorous image-based approach. These approximations can be over-compensated by the higher number of bounces and combinations of different mechanisms, which could not be considered with ray tracing due to the high computation times.

Further studies will be carried out in the future to further assess the potential and the prediction accuracy of DED-RL.

REFERENCES

[1] M. Hata, "Empirical formula for propagation loss in land mobile radio services," in IEEE Transactions on Vehicular Technology, vol. 29, no. 3, pp. 317-325, Aug. 1980.
[2] M. N. Abdallah et al., "Further Validation of an Electromagnetic Macro Model for Analysis of Propagation Path Loss in Cellular Networks Using Measured Driving-Test Data," in IEEE Antennas and Propagation Magazine, vol. 56, no. 4, pp. 108-129, Aug. 2014.
[3] Felsen, L.B., and N. Marcuvitz, Radiation and Scattering of Waves, Prentice Hall; IEEE Press, 1973.
[4] T. K. Sarkar, Zhong Ji, Kyungjung Kim, A. Medouri and M. Salazar-Palma, "A survey of various propagation models for mobile communication," in IEEE Antennas and Propagation Magazine, vol. 45, no. 3, pp. 51-82, June 2003.
[5] J. P. Rossi, J. C. Bic, A. J. Levy, Y. Gahillet, and M. Rosen , "A Ray Launching Method for Radio-mobile Propagation in Urban Area," Antennas and Propagation Society Symposium 1991 Digest, London, Ontario, Canada, pp. 1540-1543 vol.3, 1991.
[6] M. F. Catedra, J. Perez, F. Saez de Adana, O.Gutierrez, "Efficient Ray-Tracing techniques for three-dimensional analyses of propagation in mobile communication: application to picocell and microcell scenarios," IEEE Antennas and Propagation Magazine, Vol. 40, No. 2, pp. 15-28, April 1998.
[7] G. Liang and H. L. Bertoni, "A new approach to 3-D ray tracing for propagation prediction in cities," IEEE Trans. Antennas Propagat., vol. 46, no. 6, pp. 853-863, June 1998.
[8] M. Raspopoulos, "Multidevice Map-Constrained Fingerprint-Based Indoor Positioning Using 3-D Ray Tracing," in IEEE Transactions on Instrumentation and Measurement, Vol. 67, No. 2, pp. 466-476, Feb.2018 doi: 10.1109/TIM.2017.2774181
[9] P. Meissner et al., "On the use of ray tracing for performance prediction of UWB indoor localization systems," 2013 IEEE International Conference on Communications Workshops (ICC), pp. 68-73, Budapest, June 9-13, 2013
[10] 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, Jun. 2015.
[11] H. L. Bertoni, Radio Propagation for Modern Wireless Systems, Prentice Hall, Upper Saddle River, NJ, USA, 2000.
[12] Z. Yun and M. F. Iskander, "Ray Tracing for Radio Propagation Modeling: Principles and Applications," in IEEE Access, vol. 3, pp. 1089-1100, 2015.

[13] C. A.Balanis, *Avanced Engineering Electromagnetics*, Wiley & Sons, 1989.

[14] Derek A. McNamara, Carl W. I. Pistorius, J. A. G. Malherbe, *Introduction to the Uniform Geometrical Theory of Diffraction*, Artech House, 1990.

[15] S. Y. Seidel and T. S. Rappaport, "Site-specific propagation prediction for wireless in-building personal communication system design," in IEEE Transactions on Vehicular Technology, Vol. 43, No. 4, pp. 879-891, Nov 1994.

[16] H. Suzuki, A. Mohan, "Ray tube tracing method for predicting indoor channel characteristics map", Electron. Lett. Bol. 33, pp. 1495-1496, August 1997.

[17] H. Suzuki, A. S. Mohan, "Measurement and prediction of high spatial resolution indoor radio channel characteristic map", IEEE Transactions on Vehicular Technology, vol. 49, no. 4, pp. 1322-1333, July 2000.

[18] J. Tan, Z. Su, Y. Long, "A Full 3-D GPU-based Beam-Tracing Method for Complex Indoor Environments Propagation Modeling", IEEE Trans. on Ant. and Propagat., vol. 63, No. 6, pp. 2705-2718, June 2015.

[19] Z. Lai, N. Bessis, G. de la Roche, H. Song, J. Zhang, G. Clapworthy, "An intelligent Ray Launching for urban prediction", 3rd European Conf. on Ant. and Propagat., 23-27 March 2009, Berlin (GE)-

[20] Z. Lai, N. Bessis, P. Kuonen, G. de la Roche, J. Zhang, G. Clapworthy, "A Performance Evaluation of a Grid-enabled Object-Oriented Parallel Outdoor Ray Launching for Wireless Network Coverage Prediction", 5th Int. Conf. on Wireless and Mobile Communications, 2009.

[21] C. Y. Kee, C. Wang, "Efficient GPU Implementation of the High-Frequency SBR-PO Method", IEEE Wireless Propagat. Letters, vol. 12, pp. 941-944, 2013.

[22] Y Tao, H. Lin, H. Bao, "GPU-Based Shooting and Bouncing Ray Method for Fast RCS Prediction", IEEE Trans. on Ant. and Propagat., vol. 58, No. 2, pp. 494-502, February 2010.

[23] M. Schiller, A. Knoll, M. Mocker, T. Eibert, "GPU Accelerated Ray Launching for High-Fidelity Virtual Test Drives of VANET Applications", Int. Conf. on High Performance Computing and Simulations, 20-24 July 2015, Amsterdam (NL).

[24] A. Schmitz, L. Kobbelt, "Efficient and accurate urban outdoor radio wave propagation", Int. Conf. on Electromagnetics in Advanced Applications (ICEAA), 12-16 September 2011, Torino (IT).

[25] R. Felbecker, L. Raschkowski. W. Keusgen, M. Peter, "Electromagnetic Wave Propagation in the Millimeter Wave Band Using the NVIDIA OptiX GPU Ray Tracing Engine", 6th European Conf. on Ant. and Propagat., 26-30 March 2012, Prague (CZ).

[26] V. Degli-Esposti, F. Fuschini, E. M. Vitucci, and G. Falciasecca, "Measurement and Modelling of Scattering From Buildings," IEEE Trans. Antennas Propagat., Vol. 55 No 1, pp. 143-153, Jan. 2007.

[27] V. Degli-Esposti, J. S. Lu, J. N. Wu, J. J. Zhu, J. A. Blaha, E. M. Vitucci, F. Fuschini, M. Barbiroli, "A Semi-deterministic Model for Outdoor-to-Indoor Prediction in Urban Areas", IEEE Antennas and Wireless Propagation Letters, vol. 16, 2017, pp 2412-2415.

[28] Shuttle Radar Topography Mission, U.S. Geological Survey, accessed on January 24th, 2018. Available: https://lta.cr.usgs.gov/SRTM .

[29] ESRI Shapefile Technical Description—An ESRI White Paper—July 1998, accessed on Apr. 15, 2017. Available: https://www.esri.com/library/whitepapers/pdfs/shapefile.pdf.

[30] D. Rose (Ed.), *Sparse Matrices and their Applications*, Springer-Verlag New York Inc, 1972.

[31] V. Degli-Esposti, F. Fuschini, E. M. Vitucci, M. Barbiroli, M. Zoli, L. Tian, X. Yin, D. Dupleich, R. Müller, C. Schneider, R.S. Thomä, "Ray-tracing based mm-wave beamforming assessment," IEEE Access, Vol. 2, Nov. 2014, pp. 1314 – 1325.

[32] J. Cheng, M. Grossmann, T. McKercher, *Professional CUDA C Programming*, Wiley & Sons, 2014.