Efficient Computation of Radio Coverage Zone Using a Spatial Partitionment Approach

L. Aveneau, P. Combeau, R. Vauzelle, M. Mériaux
IRCOM-SIC, CNRS UMR 6615, SP2MI
Bd Marie et Pierre Curie, BP 30179, 80962 Futuroscope Chasseneuil Cedex – France
Email : aveneau@sic.univ-poitiers.fr

***Abstract*— In previous contributions, we have proposed a geometrical approach to the problem of coverage zone computation. It is a three step solution : in the first one, the computation of the spatial propagation of a wave is done, using only a geometrical approach. Then, a partitionment is realised, by use of an image segmentation technique. Finally, the evaluation of the coverage zone is computed using external computations for a reduced number of receiver locations.**

Since the first step of spatial propagation is very time consuming, we develop an efficient optimization, based on discrete geometric results. Some results are presented showing a very high time reduction.

I. Introduction

In previous works [1], Combeau *et al.* investigate a spatial partitionment tool which allows to compute a coverage zone for any transmitter location. Their main hypothesis is that the environment can be partitioned into some geometrical zones of quasi-constant reception level, hypothesis which is verified in all the situations tested.

The first step of this tool is a purely geometrical wave propagation, *i.e.* a recursive computation of the different "zones" where some propagation phenomenon is perceptible. The phenomena allowed here are the line of sight propagation, the reflection and the diffraction on boundaries of any scene object (see Fig. 1). This tool allows to compute and to store each geometrical zone into which the received signals have the same history (*i.e.* the same interaction with the environment, with the same obstacles in the same order). Moreover, it allows to obtain a valid radio coverage zone in an acceptable time, *e.g.* 10 minutes for Paris downtown. It can be understood as a time optimisation for a classical ray-tracing approach.

In this paper we present the optimisation of the computation of this first step in the following way: first, we recall the spatial partitionment tool; then, we present the core of our optimisation contribution and the main discrete mathematical results we used; in the third section we discuss some results, and notice the strong improvement of the computing time.

II. Spatial Partitionment

In this section we recall the spatial partitionment tool, and mainly detail the core of the geometrical propagation step.

A. The three step process

The tool is based on three successive steps. Firstly, the geometrical propagation is done, as described in §II-B. It leads to a list of geometrical zones.

Next, the resulting zones are *embedded* in a regular 2D grid, which is a representation of the final coverage zone; at each corresponding pixel, we use a statistical attenuation considering the history of the zone. The resulting "image" is then segmented in order to obtain a spatial partitionment, where each element represents a portion of space with roughly a constant received relative attenuation.

The third and last step consists in the computation of the reception levels for all the different partition elements, using an external computation tool (*e.g. a ray-tracing software*).

In this paper we focus only on the first step, which is described in the next subsection.

B. The geometrical propagation step

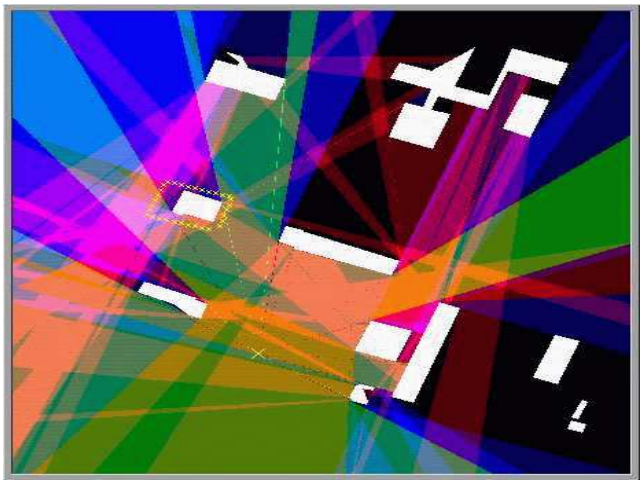

Fig. 1. An example of geometrical propagation obtained with our spatial partitionment tool. Notice that the white blocks are buildings

A *geometrical zone* allows the representation of all the propagation phenomena. They are stored in a tree, where each node can leads to any number of children. Thus the tree root is the transmitter location. The first level nodes contain only the line of sight zones: they represent all the geometrical points directly visible from the transmitter location. The next levels contain all the reflected and diffracted zones.

The construction of such a tree is done in a simple way, using a main function allowing to reduce a "*hypothetical*" zone in a good decomposition of true zones, by using blockers for cutting it. These blockers are provided by the building

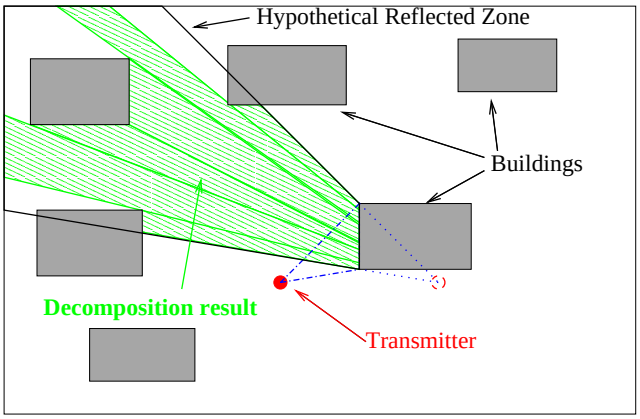

Fig. 2. Reflected zones deduced from an initial hypothetical zone

boundaries in outdoor, or walls and objects in indoor. For instance, in FIG. 2 it can be observed that the hypothetical reflected zone (with contours in dark) is decomposed into five true reflected zones (green and hatched).

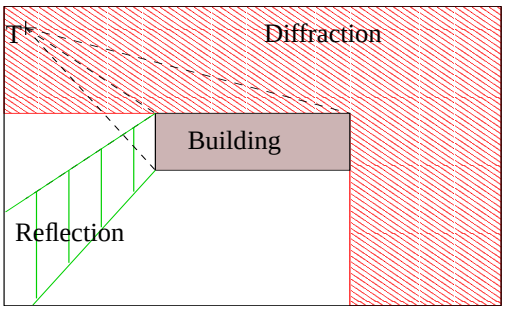

Fig. 3. Example for reflection (in green) and diffraction (red) zones

The different propagation phenomena are also simulated in a simple way; for the reflection case, we use the source-image method to obtain the reflected zone on the blocker; in the same way, the diffracted zones are obtained by attempting to diffract at the blocker corners. Two such results can be seen in FIG. 3.

III. DISCRETE GEOMETRICAL OPTIMISATION

The decomposition of a hypothetical zone into the correct zones is a difficult process. In a naive way, it is done by scanning all the building edges. So its complexity is $O(n)$, where n is the number of such edges.

For a large number of hypothetical zones (and it is rapidly the case even for simple scenes by using many reflections and diffractions), this complexity leads to an important computation time. Then, our purpose is to optimize this step of zone decomposition.

Our solution is based on a 2D discrete mathematical approach [2] [3]. It consists in embedding of all edges into a uniform grid, using an algorithm for correctly discretizing a line (see §III-A).

Then, in order to find all the edges that can cut the zone, we superpose the zone on the regular grid (see §III-B).

In the following subsections, we give an overview of the implementation principles.

A. Line discretization : the supercover model

In a 2D discrete mathematical approach, all the geometrical elements are described using pixels. A pixel is obviously defined as a couple of integer coordinates. The conversion of a 2D point of $\mathbb{R}^2$ to such a pixel may be done in many ways. Here, for a reel point (P_x, P_y) of $\mathbb{R}^2$, a pixel (x, y) of $\mathbb{N}^2$ is defined as the integer parts $(P_x = \lfloor x \rfloor \ , \ P_y = \lfloor y \rfloor)$.

A more powerful notion is the *supercover* of a pixel, given by the two following inequations:

$$\begin{array}{rcccl} \lfloor x - \frac{1}{2} \rfloor & \leq & P_x & \leq & \lceil x + \frac{1}{2} \rceil \\ \lfloor y - \frac{1}{2} \rfloor & \leq & P_y & \leq & \lceil y + \frac{1}{2} \rceil \end{array}$$

These inequations give all the pixels that can be considered for a 2D real point, including the ambiguous cases occuring when a real coordinate is a natural number (we obtain 2 pixels), or when the two coordinates are naturals (we obtain 4 pixels). For our purpose, the pixel supercover ensures that we never miss an interesting edge.

In a same way, it is possible to define the supercover of a 2D line Δ of equation $ax+by+c=0$, where $a, b, c \in \mathbb{R}^3$ are the line coefficients. It is composed by all the pixels between two 2D lines D' and D", defined by:

$$\begin{array}{rcccl} D' & : & a(x+\frac{1}{2}) + b(y+\frac{1}{2}) + c & = & 0 \\ D'' & : & a(x-\frac{1}{2}) + b(y-\frac{1}{2}) + c & = & 0 \end{array}$$

In other words, the supercover of Δ is composed by all the pixels (x, y) given by the following double inequation:

$$-\frac{(|a| \ + \ |b|)}{2} \quad \leq \quad ax \ + \ by \ + \ c \quad \leq \quad \frac{(|a| \ + \ |b|)}{2}$$

This double inequation (which is usually called a *diophantic inequation*) is used for the first step of our optimisation, *i.e.* for embedding the building egdes into a 2D regular grid. Notice that using of a line supercover is necessary in order to ensure that we do not miss any edge during the zone decomposition step.

Here, there is no problem with computation time, since this is done only one time for each edge. Nevetheless, this computation will be reused in the next step of our optimisation method. So we must use an efficient implementation, based on the well-known Bresenham's one [4], but with the supercover account.

B. Zone discretization

Our optimisation is based on the ability to find all the edges which can be inside a given zone. As in the previous subsection, it is done using a discrete geometrical approach. Here, the problem is double:

- To be able to produce an efficient discretization of a zone, *i.e.* without missing any pixel for the precision, and with a computation as fast as possible.
- To scan the discretization in the best way.

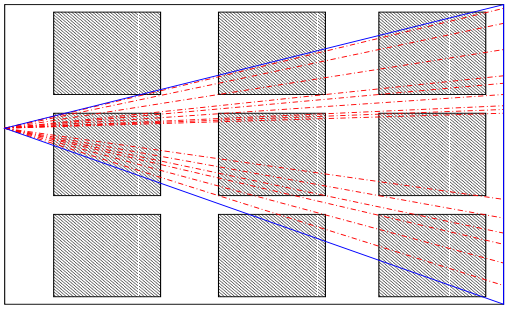

Fig. 4. The use of far pixels (and thus far edges) for cutting the zone (contour in blue) leads to too many subdivisions (dashed contours in red)

This last problem is based on an obvious fact: assume we scan a zone discretization from the farthest pixel to the nearest one, then we probably first find an edge which is, in fact, shadowed by many other ones. On the other side, scanning from the nearest to the farthest pixel ensures that we probably find first a really visible edge. In this last case, the advantage is that we do not cut too many times a zone with shadowed edges. So the gain is obviously in computation time, which is reduced, but also in the reduction of memory usage. Indeed, cuting a zone by an edge generally leads to two new zones on the blocker border, plus the original one which is modified.

For instance, using the farthest pixels lead to the complex decomposition in FIG. 4, whereas using the nearest edge may leads only to the modification of the zone. It is noticeable that the largest number of false subdivisions is equal to the number of building corners falling inside the zone. In the figure FIG. 4, there are 19 vertices, and so 19 subdivisions in the worst case.

In our implementation, we solve these problems with two tips. The first one consists in scanning the discretization in the best way. In order to do this, we define 4 privilegied directions:

- from minimum absciss to maximum absciss,
- from maximum absciss to minimum absciss,
- from minimum ordinate to maximum ordinate,
- from maximum ordinate to minimum ordinate.

The second tip consists in using a special data structure for storing the border of the discretization zone. Indeed, since the zones are allways triangles or quadrangles, then they are always convex. So the knowledge of a zone discretization can be easily obtained with the knowledge of the discretization border. This data structure is an array, where each cell is a couple of integers. The semantic of these couples depends on the best scanning direction, as defined in the first tip.

Using these two tips leads to a very simple optimization process:

- Firstly, we compute the best direction for scanning the zone discretization.
- Secondly, we compute the size of the data structure border.
- Thirdly, we compute the supercover of the zone border. The computation of these segments of line gives extremum values that are stored into our data structure.
- Finally, we can scan the zone discretization, by scanning the array with respect to the best direction obtained in step 1.

In the last step we must evaluate, for each building edge stored in the corresponding pixel, if it is a valid blocker. If yes, then we attempt to decompose the zone. If the decomposition is valid, then we stop to evaluate the discretization since the zone is modified, and the modified zone is re-discretized later.

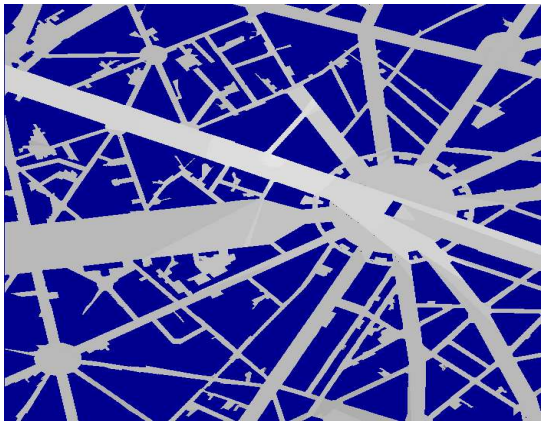

Fig. 5. Scene C: the paris downtown

IV. DISCUSSION

A. Computation time results

We make some comparisons between the old naive solution and our proposed optimized method. We use 3 test scenes:

- A is a zoom of the campus of the University of Poitiers, France, as depicted in FIG. 1. It consists in 111 horizontal building edges.
- B is the scientific part of the same campus. It consists in 451 horizontal building edges.
- C represents Paris downtown, and contains 9515 building edges (see FIG. 5).

All the computation time are obtained on a Xeon at 2 Ghz with 1Go of RAM.

As shown in TABLES I, the new solution leads to a very important acceleration factor in terms of computation. We show both the different computation times and the number of zones produced. This last information is very relevant, since a large number of zones implies both time and memory overconsumption. The left column denotes the interactions processed during the computation, where R is a reflection, and D a diffraction. For instance, the first line for scene A denotes computation for neither reflection nor diffraction account, while the next one shows results for 1 reflection and 1 diffraction.

For the first scene A, it is noticeable that even with a very small number of building edges, our optimization reduces the computation time. This result is, in our opinion, directly correlated to the reduction of the zone number. Another fact of interest is the importance of the diffraction phenomenon. Each diffraction account implies that the computation time and the number of zones grow highly. This observation remains true

TABLE I

COMPARISON BETWEEN THE NAIVE AND THE NEW OPTIMIZED ZONE DECOMPOSITION

		Naive algorithm		Optimised solution	
Scene	Param	Time	# Zones	Time	# Zones
A	0R 0D	0"04	42	0"02	28
	1R 1D	0"14	3.371	0"07	1.942
	2R 1D	0"32	8.024	0"13	4.529
	3R 1D	0"52	14.331	0"22	7.924
	4R 1D	0"81	21.585	0"33	11.760
	1R 1D	2"87	85.577	1"20	46.184
	2R 2D	8"72	249.386	3"33	132.455
	3R 2D	19"84	536.636	7"24	280.416
	4R 2D	37"55	954.536	13"23	491.728
B	0R 0D	0"05	234	0"05	80
	1R 1D	5"89	51.247	0"70	18.335
	2R 1D	18"83	160.674	2"10	58.660
	3R 1D	44"79	375.543	4"77	140.151
	4R 1D	89"90	736.023	9"66	279.855
	1R 2D	308"16	2.711.251	34.69	959.438
	2R 2D	1236"01	10.844.342	136"71	3.887.361
	3R 2D	NA	NA	406"99	7.515.342
C	0R 0D	4"93	1.879	0"50	71
	1R 1D	631"67	281.564	1"61	19.566
	2R 1D	1933"49	841.105	3"21	58.821
	3R 1D	4567"35	1.964.082	6"42	136.352
	4R 1D	9162"92	3.893.005	11"87	270.168
	2R 2D	NA	NA	209"27	3.167.696

both for the naive and the optimized algorithm. This crucial fact is due to the important opening angle of the diffraction zones with respect to the reflection one. Thus, there are much more decompositions in the first case, and therefore more computation time and more resulting zones.

With the scenes B and C, all the previous observations remain valid. The computation time is drastically reduced when the edge number grows up, as with the scene C (notice that NA means Not Available, *i.e.* computation time too large, or not enough memory). The most important result is that, for a scene with many edges, the computation can be done for any number of interactions. This was not the case without our optimization, since then the computation time is more than a week or a month. Notice that the memory limitation can be broken out by using the zone without storage, for instance for a ray-tracing prediction tool as the Ray-Tube method [5].

B. Influence of the pixel width

Another fact of importance is the pixel width: for small scenes, as scene A, the acceleration is not as important as for big ones, since the grid scanning is, then, time consuming. So we have to reduce the grid size – which is directly correlated to the pixel width –. In any case the optimal size is then obtained with respect to the number of building edges.

For instance we show the importance of the pixel width for the scene A and B, for 1 reflection and 1 diffraction, in FIG. 6.

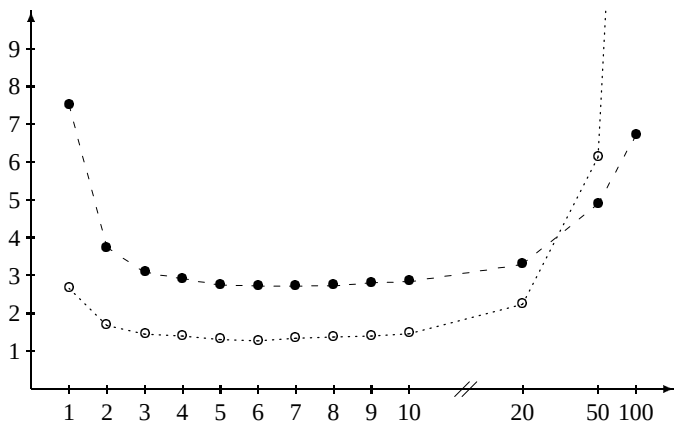

Fig. 6. Importance of the pixel width (in abscis) on the computation time (in ordinate) for the scene A (dark bullets) for 2R and 2D, and the scene C (white bullets) for only 1R and 1D.

Clearly there is an optimal value for each scene. Finding it is a very hard problem, as it is very hard to find the best size of a regular grid in a ray tracing implementation [4]. For a given scene, only some experiences can produce it (with a small number of interactions). Nevertheless, the best value we currently use is a pixel width of 5 meters: with big scenes, the computation time seems almost optimal; with small ones the computation time can be reduced, but not really in an important way.

V. CONCLUSION

As a main conclusion, it must be noticed that this efficient time optimization leads to an efficient tool, allowing a radio coverage zone computation in an interesting time; moreover it may be extended in order to optimize a classical ray-tracing software, by diminishing the number of the model application points [1].

Another use may be to obtain the interesting combinations of edges and faces, which is the main problem for a ray-tracing software. Then the memory limitation of this method can be broken out, by dealing with all the coverage receivers for each computed zone. This may be a very powerful solution to the efficient coverage prediction problem, and can be, for instance, a good solution to the limitation problems of the Ray-Tube method [5].

REFERENCES

[1] P. Combeau, R. Vauzelle, L. Aveneau and Y. Pousset, *An acceleration technique for the radioelectric coverage prediction in small and microcell configurations*, IEEE European Workshop on Integrated Radio Communication Systems, Angers, France, 6-7 May 2002.

[2] J.-P. Reveillès, *Combinatorial pieces in digital lines and planes*, SPIE Vision Geometry IV, vol. 2573, 1995.

[3] Eric Andres, Raj Acharya, and Claudio Sibata. Discrete analytical hyperplanes. *Graphical Models And Image Processing*, 59(5):302–309, September 1997.

[4] J.D. Foley and A. van Dam and S.K. Feiner and J.F. Hughes, *Computer Graphics: principles and Practice*, Addison-Wesley Publishing Company, 1995.

[5] Hae-Won Son and Noh-Hoon Myung A Deterministic Ray Tube Method for Microcellular Wave Propagation Prediction Model *IEEE Transactions on Antennas and Propagation*, 47(8):1344–1350.