Code_TYMPAN™ open source software dedicated to the calculation of industrial noise in the environment

Solène Le Bourdiec[1] and Denis Thomasson[1]

[1] EDF R&D

Clamart, France

ABSTRACT

Code_TYMPAN™ is an open source software for industrial noise calculation in the environment. It allows to deal with 3D realistic geometries and has a convenient Human Machine Interface to help engineers to build 3D models to achieve analysis needed in environmental noise studies. Code_TYMPAN™ allows to define the user's simulation from basic components and geometrical solvers. It includes different solvers: a classical one based on ISO 9613 extended to industrial applications and a more sophisticated one called ANIME3DSolver. The latter one uses an efficient 3D ray tracer taking into account multiple diffraction and meteorological effects. This ray tracer is based on acceleration structures.

The aim of this paper is to present the possibilities offered by Code_TYMPAN™. Some simulation results on more or less complex scenes will be shown together with the 3D ray tracer computing time. Outlooks for further developments will also be discussed.

1. INTRODUCTION

Code_TYMPAN™ is an open source software for industrial noise calculation in the environment developed the R&D division of the French electric utility EDF [1]. It allows to deal with 3D realistic geometries and has a convenient Human Machine Interface to help engineers to build 3D models and to achieve analysis needed in environmental noise studies. Code_TYMPAN™ includes different acoustical solvers based on geometrical solvers to find acoustic paths. Finding acoustic paths in 3D is expensive. Code_TYMPAN™ offers an efficient method: a ray tracer based on acceleration structures to reduce the computing time.

In this paper, first the abilities offered by Code_TYMPAN™ are presented. Then the optimized 3D ray tracer implemented in Code_TYMPAN™ is described. In the last part, simulation results obtained with the ray tracer on more or less complex scenes are shown and the influence of the complexity of the scene on the computing time is discussed.

2. Code_TYMPAN™

2.1 General description

Code_TYMPAN™ is used to calculate the noise impact of industrial site on the environment. It

[1] solene.le-bourdiec@edf.fr and denis.thomasson@edf.fr

generates noise contributions for each source on discrete receiver points and noise maps with isophones (see figure 1).

Code_TYMPAN's data structure ensures the needed durability of the models along the long time scale industrial processes. The ability of reusing and extending olden projects is considered as essential. It allows especially to integrate evolutions of industrial sites in the models.

Code_TYMPAN recently turned into an 'open' software: significant changes have been achieved in order to integrate or implement one's own numerical solver: geometrical or acoustical propagation solver.

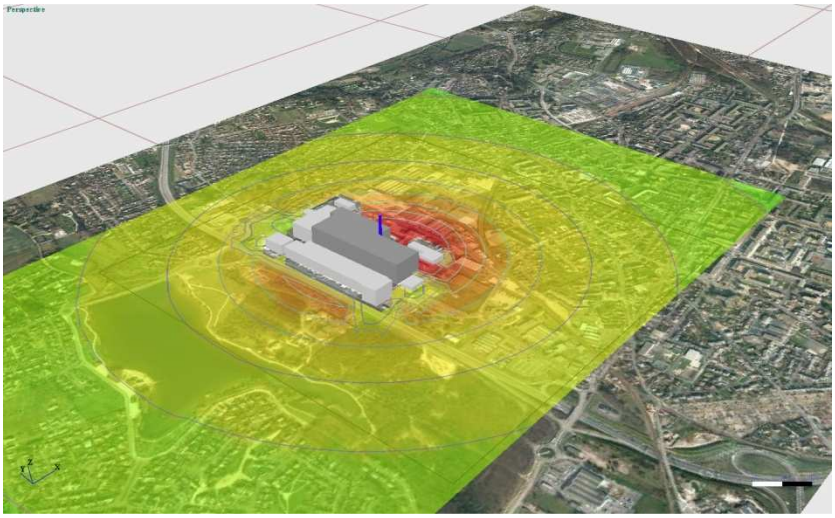


Figure 1 – Noise map result in Code_TYMPAN

2.2 Computation methods

Code_TYMPAN gives access to different simulation methods through solvers. These solvers are based on an object-oriented library of geometrical and acoustical objects more or less complex (lines, faces, buildings, machines, roads, acoustic boxes, lakes, sources, receivers, chimney, ground…).

Simulations are divided in two steps: the geometrical step to find acoustic paths from sources to receivers and the acoustic step to compute acoustic attenuation on each obtained path.

Code_TYMPAN includes two geometrical methods that can be used with different acoustic calculation methods: a convex hull finding and an optimized 3D ray tracing which will be described in section 3.

Code_TYMPAN has two acoustical solvers. The first one is based on an improved version of the ISO9613 method which includes 3D diffraction and improved ground effects. This software is generally used with the convex hull finding method. The second solver is a new one and is called *ANIME3DSolver.* It is based on the Assessing Noise Impact on the Environment in 3D method (ANIME3D). This method takes into account meteorological effects, 3D diffraction and influence of inhomogeneous ground using Fresnel area [2]. It uses a 3D optimized ray tracer to find acoustic ways.

3. OPTIMIZED ACOUSTIC RAY TRACER

3.1 Description

In the acoustic ray tracing method, all 3D acoustical paths propagating from sources to receivers have to be found. These paths are called rays and are straight if we assume that the environment is homogeneous (no meteorological effects).

The ray tracer principle is to propagate many rays through a 3D scene and to manage the interactions between rays and scene elements.

Rays loose energy and undergo changes of phases along their ways because of wavefront expansion, atmospheric absorption, diffraction and reflection.

Reflection is assumed specular. Diffraction satisfies the Fermat principle. In case of diffraction, secondary rays are launched on the surface of the Keller cone whose axis is the diffraction edge. The aperture angle of this cone is equal to the angle of arrival of the incident ray. Diffraction increases the number of rays and is also expensive. It introduces exponential complexity.

Rays are stopped according to the following criteria:

- Number of diffractions
- Number of reflections
- Total number of events
- Maximum ray length

The cost of the ray tracing algorithm is linked to the intersections between rays and surfaces searches. Basically, the solution is to test the intersection between a ray and each primitive (We called primitives the basic surface elements of the scene: building faces and topography triangles). This solution is very expensive. The complexity is in *O(n)* where n is the number of the primitives of the scene.

This ray tracing basic algorithm, named brute-force algorithm, is given in figure 4.

```
for all rays do
    Find the first intersection with the scene
        if intersection found then
            Scene interaction
            Secondary rays generation
        end if
and for
```

Figure 4 - Ray Tracing basic algorithm

3.2 Acceleration methods

Accelerating structures are used to reduce the cost of the brute-force algorithm. They are methods of primitives' classification. Thanks to these structures, intersection tests are less expensive because only a small relevant subset of the primitives is tested. These methods also provide a significant reduction of the computation time.

The classification is done by building a tree of bounding boxes which contain subsets of primitives. The root of this tree is the bounding box containing the full scene, leaves are primitives and middle boxes contain less and less primitives.

One of the methods used to build the tree is called Bounding Volume Hierarchy method (BVH). It constructs a tree taking into account the volumes of the bounding boxes [3]. Another available method, called KdTree (K dimensional Tree) is based on space splitting [4]. Examples of both BVH and KdTree tree building methods are given in figures 5 and 6.

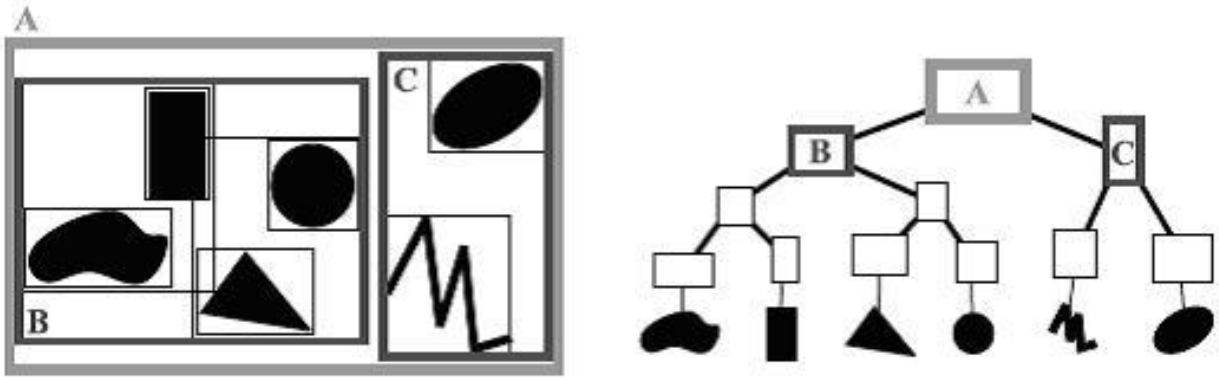


Figure 5 – Tree building in the Bounding Volume Hierarchy method (BVH)

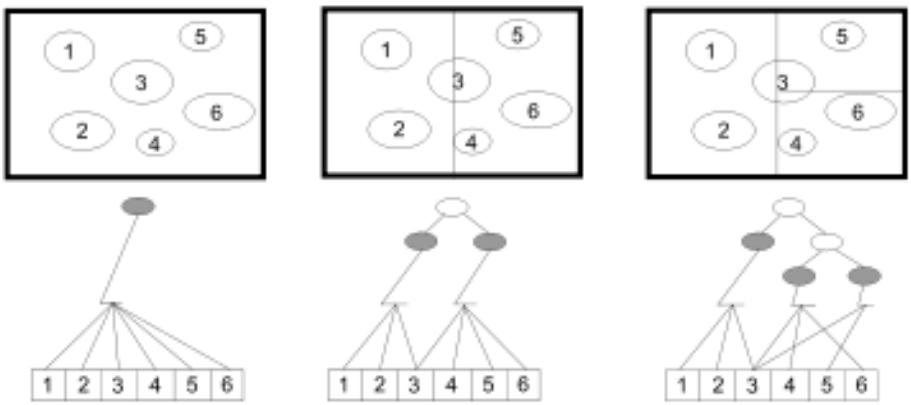


Figure 6 – Tree building in the KdTree method

Once the tree is built, it is read through testing the intersections with the faces of the bounding boxes. If the intersection is found, subsets of boxes are successively tested. This process saves intersection tests on all primitives inside the unintersected boxes.

Complexity of the search algorithm is now in *O(log(n))* and the complexity of the construction of the tree is in *O(n.log(n))*.

These acceleration methods can be optimized by using a cost function such as the Surface Area Heuristic method (SAH) [5] which assumes that the probability to intersect a volume is based on its area. To optimize the way to subdivide space, the cost function estimates the cost of different possibilities and keeps the best one. SAH method assumes that the probability to intersect a volume is based on its area.

If the number of receivers is smaller than the number of sources launching direction is reversed, reducing time calculation.

4. RESULTS

4.1 Optimized Ray Tracer in Code_TYMPAN

The optimized Ray Tracer in Code_TYMPAN is called *AcousticRayTracer*. It includes three acceleration methods: Regular grid, KdTree and BVH. Both are optimized with the SAH method.

Both reflection and diffraction behavior have been tested and validated as we can see in figures 7 and 8.

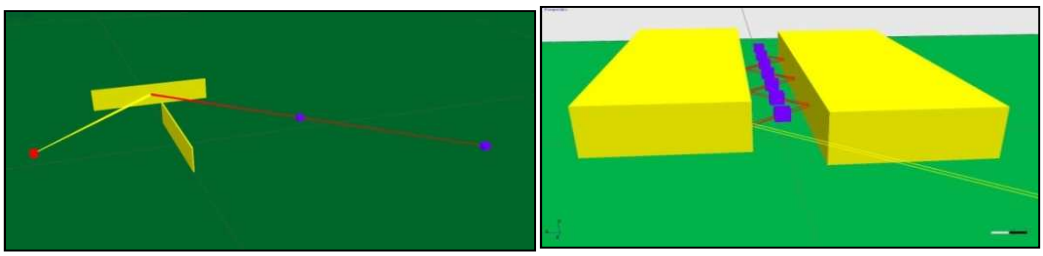


Figure 7 – Reflection validation

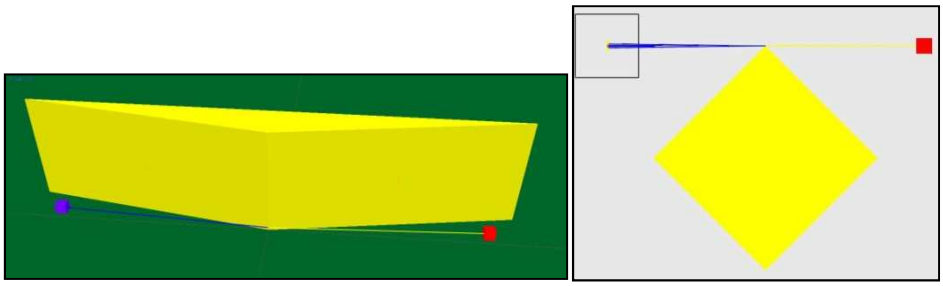


Figure 8 – Diffraction validation

Parameters of *AcousticRayTracer* are stop criteria mentioned in part 3 and also:

- Number of rays
- Size of receivers (radius of the sphere)
- Thickness of diffraction edges (cylinder are designed around edges)
- Number of secondary rays launching on the Keller cone (in case of diffraction)
- Choice of accelerating structure (no structure, regular grid, BVH, KdTree)
- Use of floor for reflections (some acoustic methods like NF S 31.133 integrate ground effect along ray course and do not need ground reflected rays)

These criteria influence the computation time, particularly the number of rays. It is also important to find the best agreement between precision and computation time.

According to Fermat's principle, the valid ray is the shorter path between the source, a set of elements and the receiver. Invalid rays have to be filtered also.

The first filter implemented suppresses duplicated rays. For instance when several rays having the same history (same events and close primitives intersected), we keep the shortest path.

The second filter suppresses rays which have two events on the same face (for example a reflection close to a diffraction on the same face).

Code_TYMPAN GUI displays paths from sources to receivers. The path color changes according to the event type.

In figure 9, we can see acoustic paths obtained with *AcousticRayTracer* applied to a quite complex scene containing 969 primitives and named *Scene1*.

In the first simulation, there are one source (in red) and one receiver (in pink). 20000 rays are launched. Rays are initially yellow. They turn red after a reflection and white after a diffraction. The computation time of this simulation is 60s (single thread on INTEL core i5 processor at 2.3 GHz).

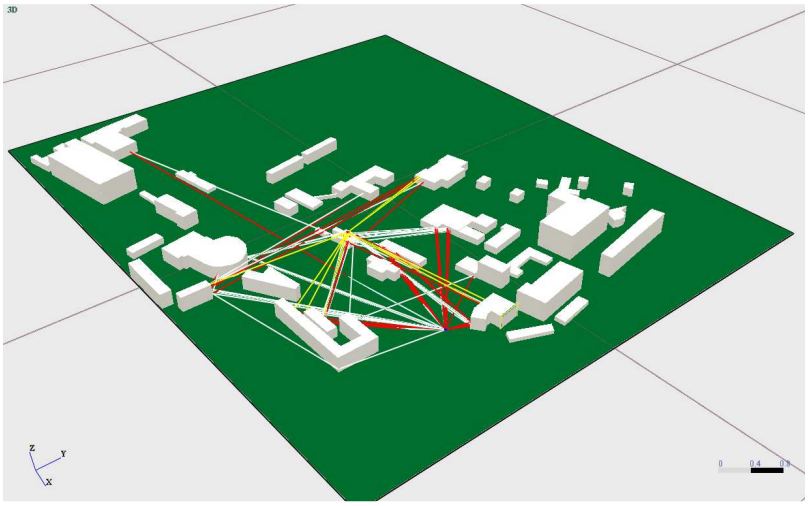


Figure 9 – Acoustic paths obtained with Code_TYMPAN's ray tracer for the *Scene1*.

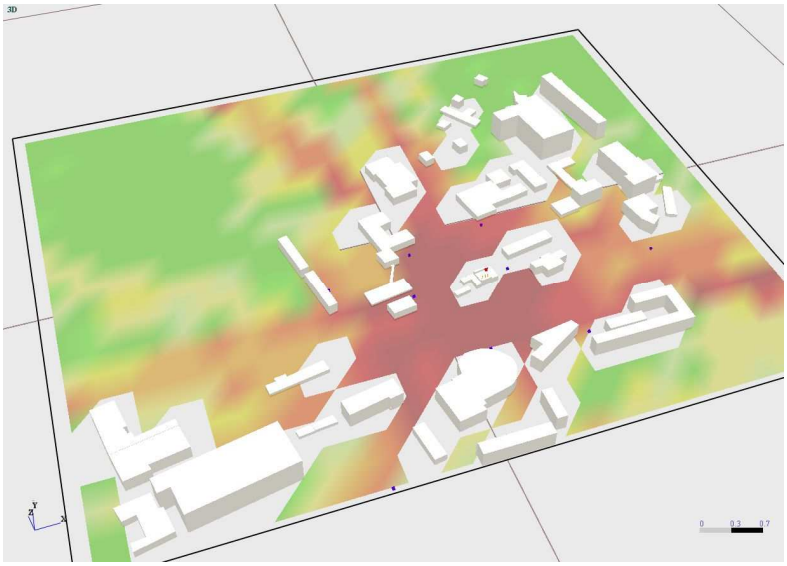


Figure 10 – Acoustic map obtained with Code_TYMPAN's ray tracer for the *Scene1*.

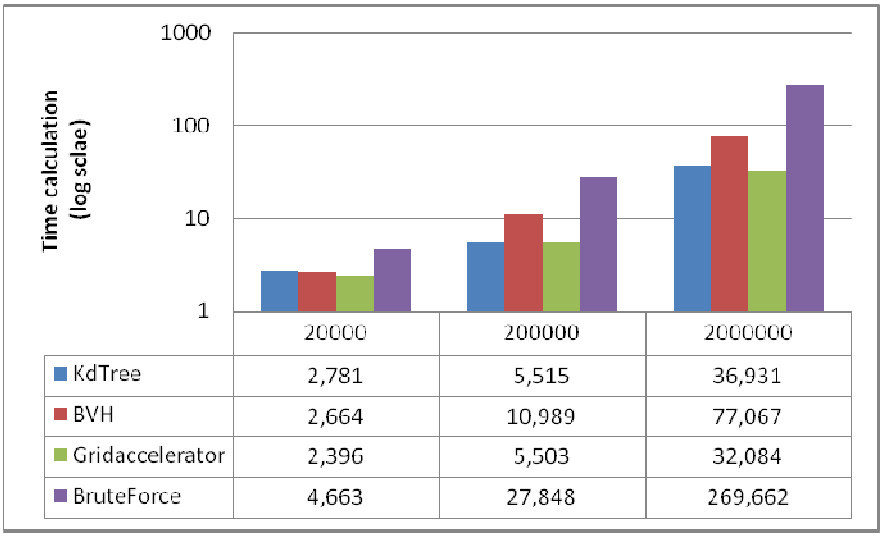
4.2 Code_TYMPAN's optimized ray tracer efficiency

In this part the performance of the optimized 3D ray tracer is presented. Firstly, brute-force and accelerating methods time calculation are compared. Secondly, the scene complexity influence is studied: different number of primitives and receivers. Then the calculation complexity is considered through the number of rays.

4.2.1 Comparison Brut force / accelerating methods

First the computing time of the ray tracer without optimization (Brute-force) is compared to the ray tracer with accelerating methods: BVH, KdTree or Grid accelerator in the *Scene 1* with 9 receivers and 3450 faces for different rays' number (2.10^4, 2.10^5 and 2.10^6).

Results are given in figure 11.



	20000	200000	2000000
KdTree	2,781	5,515	36,931
BVH	2,664	10,989	77,067
Gridaccelerator	2,396	5,503	32,084
BruteForce	4,663	27,848	269,662

Figure 11: Time calculation comparison between Brute Force method and accelerating methods (KdTee, BVH, Grid Accelerator) for 2.10^4, 2.10^5 and 2.10^6 rays.

It can be seen in figure 11 that accelerating methods time calculation grow less quick than the brute-force time calculation.

4.2.2 Influence of rays' number

Then the influence of the rays' number on computation time on the *Scene 1* for the four methods is evaluated: Brute-force, BVH, KdTree and Grid Accelerator.

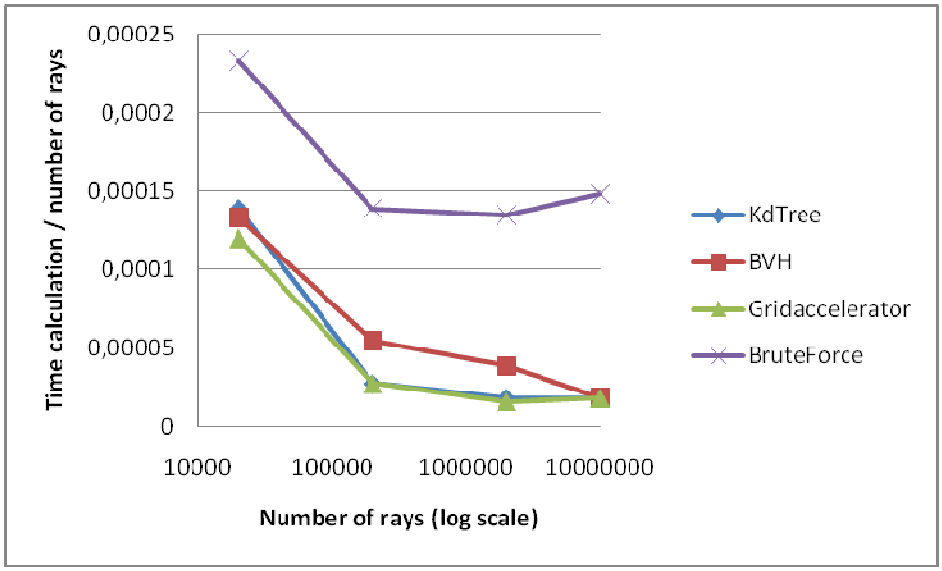


Figure 12: Evolution of the time calculation/ rays' number ratio for KdTee, BVH, Grid Accelerator and Brute-force methods

Figure 12 shows that the larger the number of rays is, the more efficient the accelerating methods are.

Computation time of accelerating methods is composed of the time to construct the tree and the identification of all paths. Tree construction time is costly and does not depend on the number of rays. So the more rays are launched, the more efficient accelerating methods are.

4.2.3 Influence of receivers' and primitives' number

In figure 13, the time calculation of the optimized ray tracer versus the number of receivers is presented. Figure 14 gives the time calculation of the optimized ray tracer related to primitives' number.

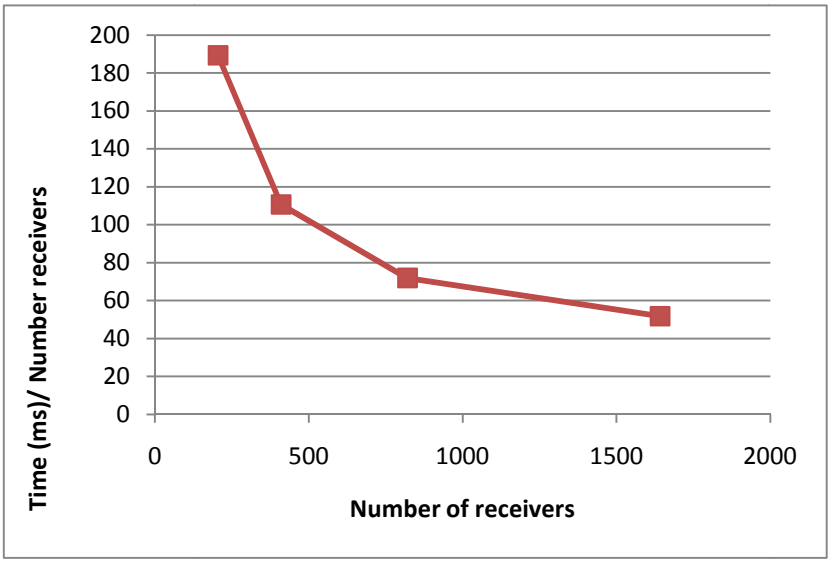


Figure 13 - Computation time of the Code_TYMPAN's optimized ray tracer related to the receivers' number

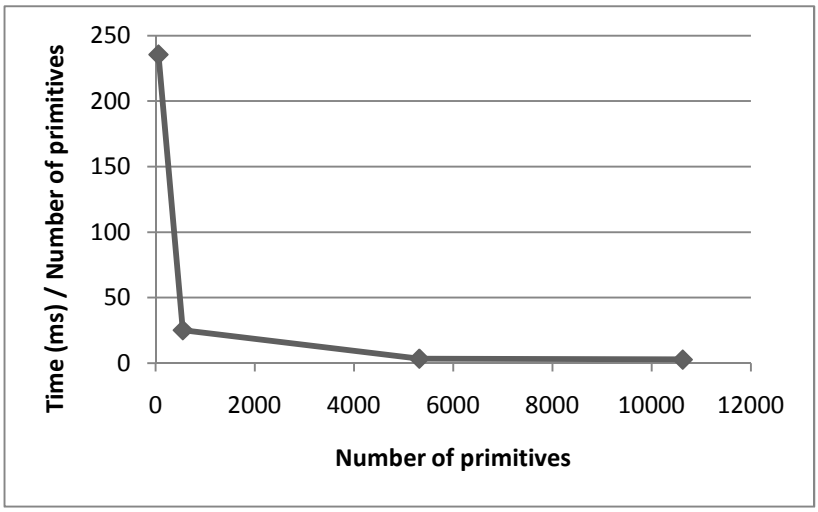


Figure 14 – Computation time of the Code_TYMPAN's optimized ray tracer related to primitives' number

Both figures here above show that the more complex the scene is (number of triangles or number of receivers) the more efficient the ray tracer is.

5. CONCLUSION AND FUTURE WORKS

Code_TYMPAN^{TM} is an open-source software which offers a promising development plateform to the acoustic community. Its 3D optimized ray tracer allows a substantial reduction of the computation time compared to a brute-force ray tracer. Moreover, the efficiency increases with the complexity of the scene.

Several new developments are in progress to further open Code_TYMPAN's architecture to split between modeling (CAO, acoustical parameters…) and solvers (paths identification and acoustic attenuation calculation). This work will allow importing CAO from others CAO softwares, parametric calculations, and easy connection of other solvers.

Further works will enlarge Code_TYMPAN applications fields including noise perception, uncertainties, parametric calculations and indoor acoustics. NMPBSolver based on NMPB2008 [6], will also be included together with road and railway dedicated interface.

Code_TYMPAN is open source. So you can extend the code to add your own numerical solver or by writing new functionalities within the interface. You can also experience Code_TYMPAN to make your own impact study.

REFERENCES

[1] « Code_TYMPAN - Logiciel - EDF R&D ». [Online].

http://innovation.edf.com/recherche-et-communaute-scientifique/logiciels/code-tympan-94425.html.

[2] F. Junker, « ANIME3D: A full 3D method for calculating the impact of industrial noise on the environment», in *INTERNOISE, 2013*, Innsbruck, Austria, 2013.

[3] Herman Johannes Haverkort, "Results on geometric networks and data structures", 2004. Chapter 1: Introduction, page 9-10 + 16.

[4] I. Wald and V. Havran. "On building fast kd-trees for ray tracing, and on doing that in O(NlogN) On building fast kd-trees for ray tracing, and on doing that in O(NlogN)." IEEE Symposium on Interactive Ray

Tracing, pp. 61-69, 2006.

[5] M. Dreher, G. Dutilleux, and F. Junker, « Optimized 3D ray tracing algorithm for environmental acoustic studies », in *Acoustics 2012*, 2012.

[6] G. Dutilleux, J. Defrance, D. Ecotière, B. Gauvreau, M. Bérengier, F. Besnard, et E. L. Duc, « NMPB-Routes-2008: The Revision of the French Method for Road Traffic Noise Prediction », Acta Acustica united with Acustica, vol. 96, n° 3, p. 452‑462, 2010.