

# The TLM method for acoustics : local and distributed implementations in Scilab

G. Dutilleux<sup>a</sup> and J. Waechter<sup>b</sup>

<sup>a</sup>Laboratoire Régional des Ponts et Chaussées – 11, Rue Jean Mentelin  
BP 9 – F 67035 STRASBOURG cedex 2 – France  
[guillaume.dutilleux@equipement.gouv.fr](mailto:guillaume.dutilleux@equipement.gouv.fr)

<sup>b</sup>1, rue des jardins – 67350 La Walck - France  
[waechtej@free.fr](mailto:waechtej@free.fr)

**Abstract :** In the fields of outdoor sound propagation prediction and noise abatement the simulation of real-world problems usually requires an amount of memory that is larger than what is possible on a high-end personal computer. If no supercomputer is available, distributed computing is the only way to perform simulations on the whole frequency range of interest. Due to its formulation at the fluid element level, the Transmission Line Matrix (TLM) method is easy to implement on a network of computers. The present paper deals with the implementation of TLM models in Scilab for sound propagation. A short introduction to the principles of TLM is provided in the 2D and 3D cartesian homogeneous cases. Some details of the local (i.e on a single computer) implementation are discussed. It is shown that Scilab is suitable for TLM. A comparison against Sabine's reverberation theory in a shoebox-shaped 3D cavity validates the implementation. The last part deals with a platform-independent distributed implementation of the TLM method in Scilab which is based on a client/server architecture, socket inter-process communication and the Scilab/Java interface. The operation of the whole system is illustrated on the academic example of the radiation of a point source in a 2D homogenous medium.

## INTRODUCTION

In the fields of outdoor sound propagation prediction and noise abatement there is a growing need for large scale computation tools that can deal with complex geometries, inhomogeneous propagation media and work at high frequency (usually up to 5000 Hz), occasionally in the time domain. The wish to simulate inhomogeneous media and arbitrary geometries makes it difficult to avoid a volumic discretization of the fluid domain. Therefore, real-world problems usually require an amount of memory that is larger than what is possible on a high-end personal computer. If no supercomputer is available, distributed computing is the only way to perform simulations on the whole frequency range of interest. Due to its local formulation the Transmission Line Matrix method (TLM) is easy to implement on a network of computers.

The aim of this contribution is to show first that the matrix orientation of Scilab make it suitable for simulating acoustic problems by TLM ; second that, with moderate additional effort, large scale TLM problems can be addressed in a distributed way over a network of computers running Scilab.

In the following, the basics of TLM theory adapted to acoustics are first outlined. The second section presents a computationally effective implementation of a 3D homogeneous TLM in Scilab which is validated with respect to the reverberation theory in rooms. The last section is dedicated to the presentation of a distributed implementation of the TLM method in Scilab with a Java interface for execution over a computer network. The operation of the whole system is demonstrated on a academic situation.

## BASICS OF TLM

### Huygens' principle

The TLM method is based on the Huygens' principle [Kag-1998]. Given a point source that radiates spherically in a propagation medium, the wave front consists of a set of secondary point sources which in turn emit spherical wavelets whose envelopes form a new spherical wavefront which again gives rise to a new generation of spherical wavelets (cf Figure 1).

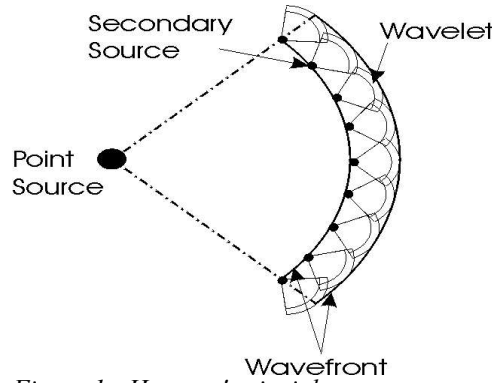


Figure 1 : Huygens' principle

### TLM model

Let us first assume a 2D propagation with no loss of generality. For acoustics, the natural translation of the principle above on a digital computer is to discretize the propagation medium as a cartesian grid made of channels filled with air that connect at nodes or junctions, as shown on Figure 2. All channels have the same length  $\Delta l$  and acoustic impedance  $Z$ . A wave pulse arriving at a junction will be reflected and transmitted according to a local rule which is defined by computing the plane wave reflection coefficient at an impedance discontinuity (cf. Figure 3) and the energy conservation principle.

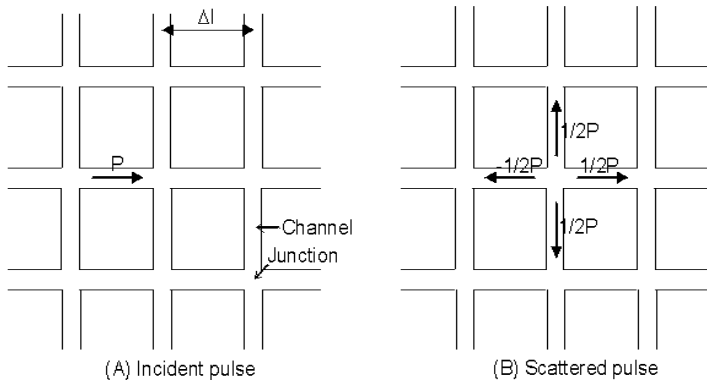


Figure 2 : 2D TLM model for acoustics

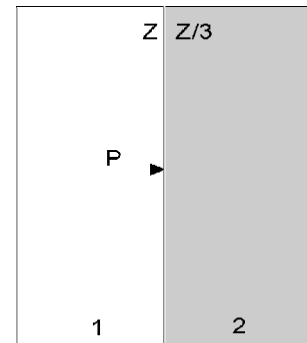


Figure 3 : impedance discontinuity at a node

Considering the different possible incidences at a given junction reference as in a matrix by a row and a column index  $(i,j)$ , one can establish a scattering matrix, which gives for incident waves at instant  $t$  the resulting waves in the 4 channels connected to this point at instant  $t + \Delta t$  :

$${}_{t+\Delta t} \begin{bmatrix} R^w \\ R^n \\ R^e \\ R^s \end{bmatrix}_{i,j} = \frac{1}{2} \begin{bmatrix} -1 & 1 & 1 & 1 \\ 1 & -1 & 1 & 1 \\ 1 & 1 & -1 & 1 \\ 1 & 1 & 1 & -1 \end{bmatrix} {}_t \begin{bmatrix} I^w \\ I^n \\ I^e \\ I^s \end{bmatrix}_{i,j} \quad (1)$$

Where  $R$  stands for reflected pulse and  $I$  for incident,  $w,n,e,s$  for the cardinal directions West, North, East and South.

The pressure at node  $(i,j)$  is evaluated as :

$$P_{i,j} = \frac{1}{2}(I^w + I^n + I^e + I^s) \quad (2)$$

For the pulse to move from one node to its neighbours, one must add a series of propagation rules. For instance, in the case of two connected east and west channels of adjacent nodes the propagation rules are expressed as follows :

$$\begin{aligned} {}_{t+\Delta t}I_{i,j+1}^w &= {}_tR_{i,j}^e \\ {}_{t+\Delta t}I_{i,j}^e &= {}_tR_{i,j+1}^w \end{aligned} \quad (3)$$

With this combination of a scattering matrix and propagation rules at the microscopic level, the normal wave motion in free-field can appear at the macroscopic level. It is of course possible to introduce boundaries in the propagation medium by defining particular propagation rules for the nodes in contact with boundaries. For instance, if one assumes that node  $(i,j)$  is at the vicinity of a vertical border with reflection coefficient  $r$ , the propagation rule above rewrites :

$${}_{t+\Delta t}I_{i,j}^e = r {}_tR_{i,j}^e \quad (4)$$

From a numerical point of view, the accuracy of the TLM scheme is close to the one of a second order finite difference scheme, but with a much better stability. Propagation at the macroscopic level is dispersive, i.e. speed of sound is frequency dependent, but when properly meshed the propagation speed can be kept constant over frequency.

For more details on TLM the reader shall refer to [Chri-1995] and [Kag-1998].

### 3D homogeneous case

It is straightforward to generalize to the 3D case which leads to the following scattering matrix with the same naming conventions as for the 2D case plus  $h$  and  $l$  which stand for high and low :

$${}_{t+\Delta t} \begin{bmatrix} R^w \\ R^n \\ R^e \\ R^s \\ R^h \\ R^l \end{bmatrix}_{i,j,k} = \frac{1}{3} \begin{bmatrix} -2 & 1 & 1 & 1 & 1 & 1 \\ 1 & -2 & 1 & 1 & 1 & 1 \\ 1 & 1 & -2 & 1 & 1 & 1 \\ 1 & 1 & 1 & -2 & 1 & 1 \\ 1 & 1 & 1 & 1 & -2 & 1 \\ 1 & 1 & 1 & 1 & 1 & -2 \end{bmatrix} {}_t \begin{bmatrix} I^w \\ I^n \\ I^e \\ I^s \\ I^h \\ I^l \end{bmatrix}_{i,j,k} \quad (5)$$

The pressure at node  $(i,j,k)$  is evaluated as :

$$P_{i,j,k} = \frac{1}{3}(I^w + I^n + I^e + I^s + I^h + I^l) \quad (6)$$

This case is presented in more details in [Kag-1999]. It is the one chosen for the local implementation outlined in the next section.

### Refinements and extensions

TLM is in no way limited to propagation in a homogeneous time-invariant atmosphere. Due to its local formulation it is simple to implement isotropic sound speed gradients. Some authors have also proposed TLM schemes that include anisotropic gradient [Kag-2001]. In the field of outdoor sound propagation the latter gradient correspond to a wind effect, the former to a temperature effect.

Introducing atmospheric turbulence, i.e stochastic fluctuations of the sound speed, should be easy too. Modelling dissipative media is possible [Kag-1998]. Regarding boundaries, some authors have defined frequency dependent ones. Others have introduced walls with diffuse reflection [Dut-2002], [Dut-2003].

The major hindrance to the application of TLM for sound propagation outdoors is the volumic mesh it requires. For this issue, an axi-symmetric variant of TLM has been developed [Kag-1998]. With this scheme a 3D propagation can be described by a 2D mesh with distant dependent channel impedances.

By the way, TLM has also a wide spectrum of applications in other areas of physics [Chri-1995] [DeC-1998].

## LOCAL IMPLEMENTATION IN SCILAB

### Code for 3D TLM

This section presents the implementation in Scilab of a 3D version of a TLM scheme. We describe the data structure first and then the most important points of the algorithm.

#### Data structure

The data structure is based on a set of 3D matrices. The main variables are listed below :

- $geo$  : stores the type of node with respect to the propagation rules. In 3D there are 27 possible kinds of rules. They are defined by the number of fluid neighbours for a given node and the orientation of the fluid neighbours. To be more specific, the rules can be divided into 4 categories :
  - 1 « fluid » (all neighbours are fluid nodes),
  - 6 « side » rules (all but one neighbours are fluid nodes),
  - 12 « edge » rules (all but 2...),
  - 8 « corner » rules (all but 3...).

The type of node is identified by a key.

- 12 « channel » matrices :
  - XI incident pulses : 1 matrix par channel orientation, i.e. 6 matrices,
  - XR reflected pulses : 1 matrix by channel orientation, i.e. 6 matrices,
  - where  $X \in \{W, N, E, S, H, L\}$  .
- P : matrix for the global pressure at each node. This matrix is not mandatory if one is only interested in values of the acoustic field at isolated locations.

#### Algorithm

The code uses the global index notation for referencing groups of elements in a matrix. We remind that, assuming a 3D matrix with  $L$  rows,  $M$  columns and  $N$  layers, the global index for element  $(i,j,k)$  is given by  $I = i + (j-1)L + (k-1)LM$ . With this notation the neighbours can be easily referenced (cf. Figure 4).

The code requires a version of Scilab with fully a functional hypermatrix data type i.e. a release that does not make any difference between a classical 2D matrix and a higher dimension object, in particular for the operation of `find`, and the referencing

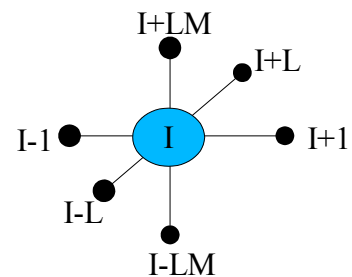


Figure 4 : global indexes of neighbours for element I

of matrix subsets. This is the case since version 3.0.

As shown below, with the chosen data structure it is possible to express the main iteration loop with only matrix operations on matrix subsets. No other `for` loop is necessary. Although not necessary, `find` provides a very convenient way to define these subsets at the initialisation step.

## 1. Initialization

1. Global initialization
2. Defining `geo` with respect to the geometry of the problem
3. Group the elements of `geo` by key value

```
//extracting the nodes with the same propagation
case0=find(geo==0);
...
//caseX=find(geo==X) for all possible X
...
```

## 2. For each iteration do

1. Input energy in the grid if necessary
2. Compute the overall pressure (cf. Eq. 6)

```
P=1/3*(WI+NI+EI+SI+LI+HI);
```

## 3. Compute the scattered components (cf. Eq. 5)

```
//scattering matrix
WR=1/3*(-2*WI+NI+EI+SI+LI+HI);
NR=1/3*(WI-2*NI+EI+SI+LI+HI);
ER=1/3*(WI+NI-2*EI+SI+LI+HI);
SR=1/3*(WI+NI+EI-2*SI+LI+HI);
LR=1/3*(WI+NI+EI+SI-2*LI+HI);
HR=1/3*(WI+NI+EI+SI+LI-2*HI);
```

## 4. Apply the propagation formulae for each key-grouped node

<pre>//fluid volume WI(case0)=ER(case0-1); NI(case0)=SR(case0+L); EI(case0)=WR(case0+1); SI(case0)=NR(case0-L); LI(case0)=HR(case0-LM); HI(case0)=LR(case0+LM);  //////////////////// //side conditions //i=1; WI(case1)=R1*WR(case1); // reflection NI(case1)=SR(case1+L); EI(case1)=WR(case1+1); SI(case1)=NR(case1-L); LI(case1)=HR(case1-LM); NI(case1)=SR(case1+LM); .... //5 other side conditions ....</pre>	<pre>//edge conditions //i=1,j=1 WI(case13)=R1*WR(case13); //reflection NI(case13)=SR(case13+L); EI(case13)=WR(case13+1); SI(case13)=R3*SR(case13); //reflection LI(case13)=HR(case13-LM); HI(case13)=LR(case13+LM); .... //11 other edge conditions //////////////////// //corner conditions WI(case135)=R1*WR(case135); //reflection NI(case135)=SR(case135+L); EI(case135)=WR(case135+1); SI(case135)=R3*SR(case135); //reflection LI(case135)=R5*LR(case135); //reflection HI(case135)=LR(case135+LM); .... //7 other corner conditions ....</pre>
---	--

## 5. End of the loop

### Validation : simulating reverberation in a room

In order to check the validity of the implementation described above, one has performed a simulation with the following parameters :

- shoe-box shaped cavity with dimensions 4 m x 3 m x 2.5 m
- reflection coefficient  $r$  constant over all walls (case  $r=0.5$  and  $r=0.8$ )
- frequency 100 Hz, mesh size  $\Delta l = \lambda/10$
- computation of the mean quadratic pressure on every point of the mesh as a function of time.

$$p_{rms} = \sqrt{\frac{1}{LMN} \sum_I p_I^2} \quad (7)$$

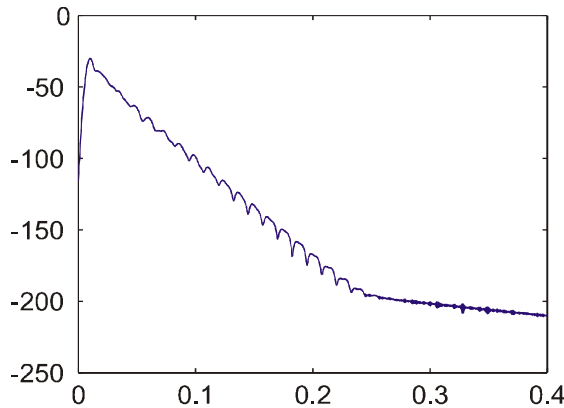


Figure 5 : mean quadratic pressure (dB re 1) as a function of time for  $r=0.5$

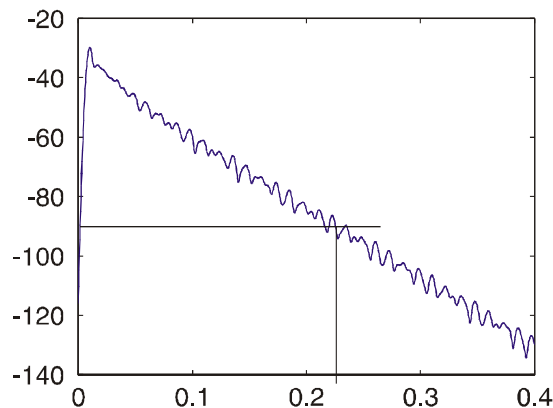


Figure 6 : mean quadratic pressure (dB re 1) as a function of time for  $r=0.8$

Assume that in a room a sound source has been turned on long enough to establish a steady-state level. The reverberation time is the time required for the level of the sound to drop by 60 dB. Sabine's reverberation theory allows for the computation of the reverberation time in the cavity by the following formula, under the assumption of diffuse field :

$$D_R = \frac{0.16 V}{S(1 - r^2)} \quad (8)$$

Where  $V$  (resp.  $S$ ) is the volume (resp. the surface of the walls) of the cavity. More details on reverberation theory can be found in any introductory book on acoustics, for example in [Kin-2000].

With the considered reflection coefficients, one obtains respectively  $D_R = 0.1$  s and 0.23 s, which corresponds to the results given by TLM simulation (Cf Figure 5 and Figure 6). For  $r=0.8$ , the decrease slows down after about 0.25 s. The following corresponds to numerical noise which is normal due to the very low level reached. This first validation is satisfying.

## DISTRIBUTION OVER A COMPUTER NETWORK

This section presents an inexpensive solution when the data structure implied by a particular TLM simulation is too large to hold in the RAM of the most powerful computer at hand : distribution over a computer network. Due to its local formulation, the distribution of a TLM scheme is quite straightforward. The principle is to cut each matrix in blocks of suitable size and to have them processed by different computers. In the following, the infrastructure for communication between the computers involved is first presented. The next subsection is dedicated to some aspects of the implementation. Finally, a concrete example demonstrates the operation of the whole distributed system.

## A client/server infrastructure in Java

A communication protocol between the computers in charge of the different submatrices is necessary both for synchronicity and for the propagation of the pressure field from one system to another. Concerning this software layer, the Java programming language was chosen for two reasons. The first one is that this is a multi-platform language, so it facilitates interoperability over an *a priori* heterogeneous computer network. The second one is that it is well adapted to network programming. The price of platform independence is that Java is not fast, so it would not be reasonable to attempt to make TLM calculations in Java. Therefore, using Scilab for this is fully justified.

The framework used to connect several machines together is of the « TCP socket » kind. A socket represents the network address of an application. Sockets are convenient for the creation of connection points between computers that are members of a local area network or even connected through the internet. Here this implies for each computer taking part to the computations to have Scilab and a Java run-time environment installed.

In the client/server architecture, the client uses a socket for the connection to the server. The creation of a socket by the client requires the address of the server and a port number. Once the

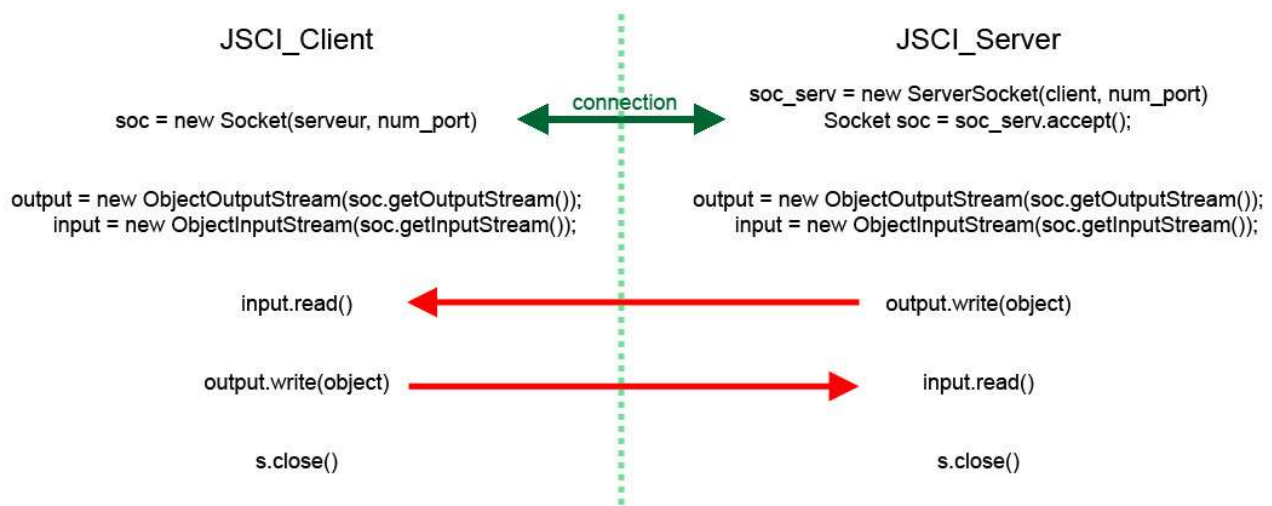


Figure 7 : Java communication interface between clients and server

connection is established, it is possible to retrieve the input and output streams. Therefore, the client can exchange data with the server using the reading and writing primitives of the package `java.io` classes.

On the server side the « `ServerSocket` » class is used for creating a communication point on a particular port. This communication point waits for connections from clients. Contrarily to the `Socket` class, the `ServerSocket` class does not open any connection.

The server is launched as a thread. The run method of the server is an endless loop waiting for clients. For each client, the server opens a specific socket for the new client and creates an instance of the `Connection` class whose responsibility is to manage the input and output streams on the socket (cf. Figure 7). In order to manage several clients at a time, the `Service` class implements the `Runnable` class. This allows for the task of reception to take place in another process.

The data exchanged between clients and server are objects. These are Java classes which encapsulate all necessary variables. This corresponds to the method of serialization. For instance, in order to send a Scilab command to the client, the server creates a new instance of the `ScilabCmd` class that implements `Serializable`. Once the `ScilabCmd` object is properly initialized, the server sends a `writeObject()` to the output stream of the target client. The client receives this object and identifies its type, with the help of the reflection method, before carrying out the requested

operation.

More information on sockets can be found in [Ma-2003] for example.

### Interface between Scilab and Java

Now that the communication layer is available, it is necessary to link the Java client to Scilab. For this, the Java Native Interface (JNI) package is used. The interface requires to write some C code.

To be consistent with the data structure in Scilab, the objects manipulated in Java belong to the Matrix class defined as an unidimensionnal array of double. The basis functions are :

- `scilabSend()` for sending a matrix to the client,
- `scilabGet()` for retrieving a matrix from Scilab,
- `scilabExec(String job)` for having Scilab execute a command.

To couple C code with Java code, the Java Development Kit provides `javah`. This utility generates a header file from Java code (cf. Figure 8). It remains to write the functions in C that correspond to

```
public class Matrix
{
    static { System.loadLibrary("javasci"); }
    [...]
    // Send to Scilab the matrix referenced by object Matrix.
    public native void scilabSend();
    // copy in a Matrix object the value of the Scilab object.
    public native void scilabGet();
    // Execution by Scilab.
    public static native void scilabExec(String job);
}
```

Figure 8 : Java header file given to `javah` for generating the C code

the prototypes appearing in the header file generated by `javah` and to link this program with Scilab. `Javah` generates most of the necessary code, but thing get tricky as soon as there is a data exchange between the different environments. This issue is addressed by using the functions for data manipulation and conversion brought by the JNI package. They cover the following aspects : adaptation between ASCII (C) and UNICODE (Java), or Java strings that do not exist in C.

### Principle of distribution

We consider here only one of the matrices necessary for a TLM computation. In order to limit the data transfer between computers, each block of the main matrix is only sent once to the clients. Then, for each iteration, each client stores the state of the submatrix it is in charge of and sends to the server the part of the submatrix which is common to another client. When all the « computer borders » have been sent to the server, the server dispatches them to the relevant clients and requests the computation of the next TLM iteration to the clients. For the sake of simplicity, we shall limit ourselves to the case of a matrix distributed over 2 clients called PC1 and PC2.



On Figure 9 the matrix XI is sent to PC1 and PC2.

Each PC receives half XI plus a column called the « cut ». The points located in the cut zone are never computed by the client that owns it.

For each iteration, the clients send their own « pre-cut » zone. When the server has received both, it dispatches the one from PC1 to PC2 and vice-versa. This way, the pre-cut zone of PC1 becomes the cut zone of PC2. So the continuity of the simulated fluid medium is reconstructed.

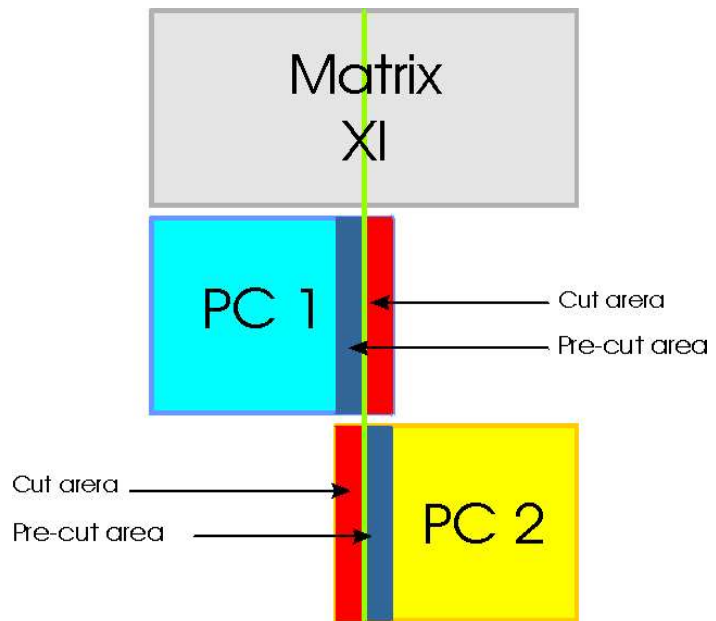


Figure 9 : illustration of the operation with 2 clients

### Scilab implementation

The Java client/server code provides the communication layer. Scilab is in charge of the core of TLM computations at the level of each client. The process of TLM computations is associated to Scilab scripts managed by the server :

- scripts for the initialization of clients : each script correspond to the initial definition of a part of the global simulation. Each script defines the relevant boundary conditions and initial state on a given part of the simulated fluid medium. As mentioned, it is sent to a client only once.
- scripts for the source signals : they give the position of the sources and the signal they emit as a function of time. The server sends this information to the target client at each iteration.

In the current implementation, each client uses the following scripts :

- topgm.sci : stores the current pressure field in a pgm ASCII file. This is a convenient way to follow the evolution of the pressure field over iterations,
- tlm.sci : computes a single TLM iteration. This is the main script.

After receiving the initial matrix, each client starts a connection with Scilab through a the C link program. Once the connexion is started, all communications are carried out through this link.

### Demonstration

To conclude this section on distributed TLM, an example of simulation is provided. The simulation has been performed with two computers running Windows 2000 (PC1) and Windows NT4 (PC2). PC1 hosted the server and one client, PC2 the remaining client. The simulation concerns the academic situation of the radiation of a point source in a 2D homogeneous domain, without any obstacle. The sound source is located on PC1. Table 1 gives a view of the pressure field for an early and a late iteration.

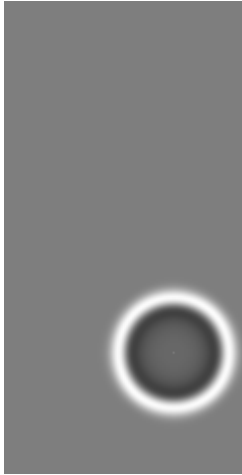

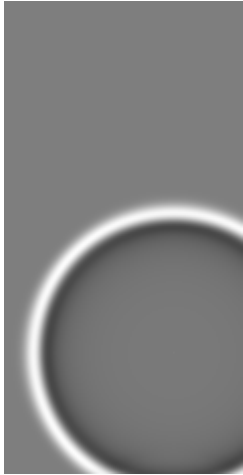
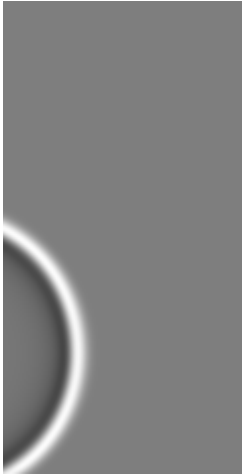
<i>Iteration 60</i>		<i>Iteration 120</i>	
			
PC1	PC2	PC1	PC2

Table 1 : radiation of point source in free field. Computation distributed over two computers.

## CONCLUSIONS AND PERSPECTIVES

The implementation of TLM in Scilab has been described. The matrix orientation of Scilab leads to a quite concise and efficient code. For large scale simulations that would not fit in the memory of a single computer, a client/server infrastructure has been developed in Java with the Scilab/Java interface. Given this additional piece of software, it is not difficult to distribute a simulation over a network of computers. The operation of the whole system has been demonstrated on an academic example. Although this feature has not been tested, it must be noticed that the distributed TLM proposed here is designed to be platform independent.

Ongoing work at LRPC Strasbourg is focussing on the introduction of inhomogeneities in the fluid medium both isotropic (temperature) and anisotropic (wind). Regarding the boundary conditions, it is necessary to define an absorbing layer that simulates a Sommerfeld conditions. Another important point is to formulate frequency dependent boundaries.

It must be emphasized that, due to the elementary mathematical background required and the definition in the time domain, TLM can be a tool of choice for education. It provides indeed a convenient way to illustrate phenomena like interferences, diffraction, refraction in sound speed gradients.

## REFERENCES

- [Chri-1995] Christopoulos C., *The Transmission-Line Modeling Method : TLM*. IEEE Computer Society Pr, 1995. 232 p.
- [DeC-1998] De Cogan D., *Transmission Line Matrix (TLM) : Techniques for Diffusion Applications*. CRC Press, 1998. 224 p.
- [Dut-2002] Dutilleux G. Simulation of sound propagation in urban areas by TLM method – Contribution to the modelling of a diffusing boundary”, G. Dutilleux, 2002, Internal Report NTNU, Trondheim, Norway.
- [Dut-2003] Dutilleux G., Kristiansen U.R., Implementation of a boundary with diffuse reflection in TLM. *In Proc. 10<sup>th</sup> Int. Congress on Sound and Vibration*, Stockholm 2003, 3655-3662.
- [Kag-1998] Kagawa Y., Tsuchiya T., Fuji B., Takeuchi M., Discrete Huygens' Model Approach to Sound Wave Propagation. *Journal of Sound and Vibration*, 1998, 218(3)419-444.

[Kag-1999] et al. 1999 : Kagawa Y., Tsuchiya T., Fujioka K., Takeuchi M., Discrete Huygens' Model Approach to Sound Wave Propagation – Reverberation in a room, sound source identification and tomography in time reversal. Journal of Sound and Vibration, 1999, 225(1)61-78.

[Kag-2001] Kagawa Y. *et al.*, Discrete Huygens' modelling simulation of sound wave propagation in velocity varying environments, Journal of Sound and Vibration 2001, 246(3)419-439.

[Kin-2000] Kinsler L., Frey, A., Coppens A.B., Sanders J.V., *Fundamentals of Acoustics* 4<sup>th</sup> edition. Wiley:New York, 2000. 548p.

[Ma-2003] Matthew N., Stones R., *Beginning Linux Programming*, Wiley:New York, 2003. 848 p.