Parallel implementation of simplified 3D digital waveguide mesh room acoustic models

Digital waveguide mesh (DWM) modelling is based on discretisation (of both space and time): the room to be modelled is represented by a contiguous set of particles (***nodes***, also called *junctions* – see Figure 1) on a regular 3D grid.

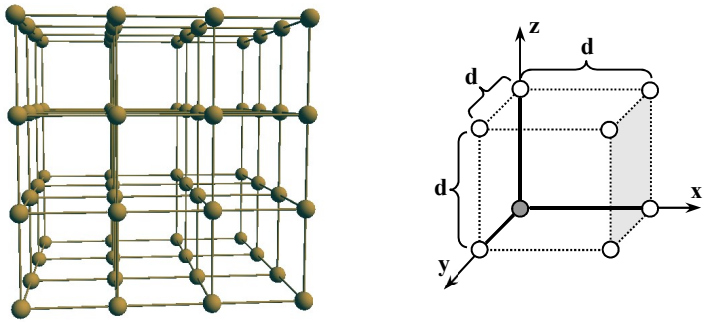

Figure 1 - 3D rectilinear mesh. Each node corresponds to a cubic volume of edge *d* (node spacing).

Node spacing, *d*, depends on the speed of sound (*c*) in the given propagation medium and the chosen audio sampling rate (which determines the mesh update frequency f_u):

$$d = \frac{c.\sqrt{3}}{f_u} \qquad \text{Equation 1}$$

[Considering the standard audio sampling rate (f_u = 44100Hz) and propagation in air (c≈344 m/s), mesh nodes will be spaced by approximately 1,35cm]

Consider a 3D node array with dimensions *X, Y* and *Z*. Node spacing being *d*, it corresponds to a cuboid with edges *W'*, *D'* and *H'* (*m*) along the coordinate axes, as shown in Figure 2:

$$W' = d.X = \frac{c.\sqrt{3}.X}{f_u} \qquad D' = d.Y = \frac{c.\sqrt{3}.Y}{f_u} \qquad H' = d.Z = \frac{c.\sqrt{3}.Z}{f_u} \qquad \text{Equations 2}$$

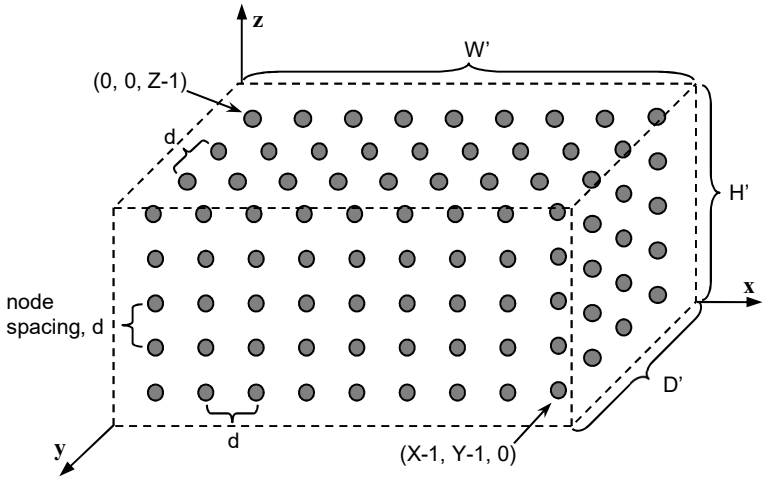

Figure 2 – Cuboid of volume *W'.D'.H'* discretised as a 3D rectilinear grid of *X.Y.Z* nodes.

The node array must cover the whole volume of the room to be modelled. Its relevant acoustic properties (namely shape and nature of its boundaries and position of sound sources and receivers) are then defined through appropriate node configuration. Nodes fall into two basic categories:

- *air nodes* – representing the propagation medium;
- *boundary nodes* – corresponding to the surfaces delimiting the room.

Air nodes can be assigned special functions, namely *mesh excitation* (modelling sound *sources*) and *sound pickup* (modelling *receivers*).

Boundary nodes are assigned an acoustic reflection coefficient ($0 \leq \rho \leq 1$) according to the physical properties of the corresponding material.

Task 1 – Develop a C program to create **room configuration data files**.

The user should specify the dimensions – width (W), depth (D) and height (H) in *m* – of a cuboid covering the room to be modelled and choose the mesh update frequency (f_u) [preferably from a menu of standard options: 48000Hz, 44100Hz, 22050Hz, 16000Hz, 11025Hz, 8000Hz…].

These data define the dimensions of the array to be allocated (in number of nodes): X (along the width axis), Y (along the depth axis) and Z (along the height axis). These are necessarily integer, and so *W*, *D* and *H* will normally have to be adjusted (the user can be informed of these adjustments) to the closest values *W'*, *D'* and *H'* satisfying Equations 2:

$$W \cong W' = \frac{c.\sqrt{3}.X}{f_u} \qquad D \cong D' = \frac{c.\sqrt{3}.Y}{f_u} \qquad H \cong H' = \frac{c.\sqrt{3}.Z}{f_u}$$

The module must then allow node configuration. Each node can be represented by an 8-bit *char* variable, as this allows 256 types of node – more than enough at this stage (see Table 1).

Table 1 – Suggested node configuration codes

Node type		Code	Node type		Code
Air nodes	standard	' ' [space]	Boundary nodes	ρ=0.8	'I'
	source	'S'		ρ=0.9	'J'
	receiver	'R'		ρ=0.91	'1'
Boundary nodes	Fully absorptive (ρ=0)	'A'		ρ=0.92	'2'
	ρ=0.1	'B'		ρ=0.93	'3'
	ρ=0.2	'C'		ρ=0.94	'4'
	ρ=0.3	'D'		ρ=0.95	'5'
	ρ=0.4	'E'		ρ=0.96	'6'
	ρ=0.5	'F'		ρ=0.97	'7'
	ρ=0.6	'G'		ρ=0.98	'8'
	ρ=0.7	'H'		ρ=0.99	'9'
				Fully reflective (ρ=1)	'Z'

Configuration can be based on initialising all nodes as regular air nodes and then allowing the user to specify successive features (boundary regions and source/receiver positions). Table 2 lists the bare minimum set of options that must be supported.

Table 2 – Minimal feature specification menu options

Feature	Parameters			
Sphere	centre (x, y, z)	radius (r)		Absorption coefficient (ρ)
Cuboid (faces parallel to the coordinate axes)	x range (x_{min}, x_{max})	y range (y_{min}, y_{max})	z range (z_{min}, z_{max})	Absorption coefficient (ρ)
Source	x position	y position	z position	N/A
Receiver	x position	y position	z position	N/A

At least one source and one receiver must be specified. Similarly to *W*, *D* and *H*, source/receiver positions specified by the user will normally have to be adjusted to the position of the nearest node (the user can be informed of this adjustment).

To work out whether a node (*i*, *j*, *k*) belongs to a specified boundary region, its Cartesian coordinates (*x*, *y*, *z*) can be calculated as follows:

$$\begin{cases} x = i.d + \frac{d}{2} = \frac{c.\sqrt{3}}{f_u}\left(i + \frac{1}{2}\right) & (0 \le i \le X-1) \\ y = j.d + \frac{d}{2} = \frac{c.\sqrt{3}}{f_u}\left(j + \frac{1}{2}\right) & (0 \le j \le Y-1) \\ z = k.d + \frac{d}{2} = \frac{c.\sqrt{3}}{f_u}\left(k + \frac{1}{2}\right) & (0 \le k \le Z-1) \end{cases}$$

Equations 3

The configured array should be stored in a data file in the format (*.dwm*) described in Table 3.

Table 3 - Room configuration data file format

	N. of elements	Type	Symbol	Description
Header	1	int	*X*	array dimension along *xx* axis
	1	int	*Y*	array dimension along *yy* axis
	1	int	*Z*	array dimension along *zz* axis
	1	long	f_u	mesh update frequency
Data	*X.Y.Z*	char		Node configuration codes: stored sequentially from node (0,0,0) to node (*X*-1, *Y*-1, *Z*-1)

The room visualised in Figure 3 was configured using a module operating as described (notice its boundary regions formed only by spheres and cuboids as per Table 2). The figure itself is a snapshot of a simple *OpenGL* visualisation module for room files under this format.

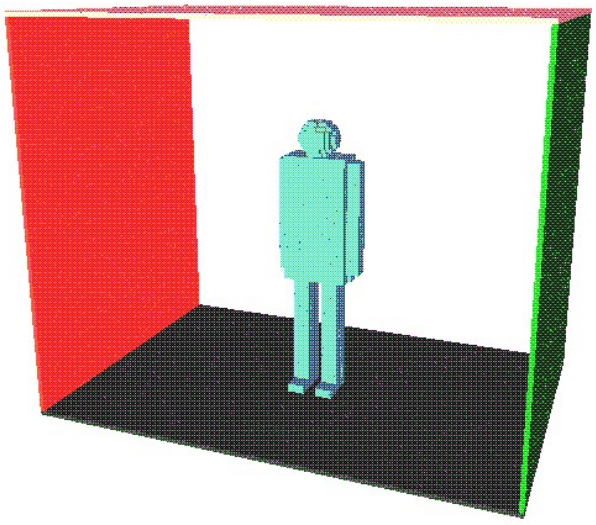

Figure 3 - Room with *W*=3m, *D*=2m and *H*=2.5m, including simplified representation of a listener, modelled at f_u=22050Hz (node spacing *d*≈2.7cm).

Task 2 – Develop a C program to perform **room partitioning**

The program should allow the user to choose a previously created room (by specifying its configuration data filename, e.g. `My_Room.dwm`) and define how it should be partitioned by specifying the dimensions – x_g , y_g and z_g – of a 3D array of blocks.

The total number of blocks, *N*, will be:

$$N = x_g . y_g . z_g \qquad \text{Equation 4}$$

The position (*x*, *y*, *z*) of block *i* ($0 \leq i \leq N - 1$) will be given by Equations 5 (where '/' represents integer division and 'mod' represents remainder of integer division)

$$\begin{cases} x = i \;/\; (y_g . z_g) \\ y = \left[i \; mod \; (y_g . z_g) \right] / \; z_g \\ z = \left[i \; mod \; (y_g . z_g) \right] \; mod \; z_g \end{cases} \qquad \text{Equations 5}$$

Conversely, the index of the block occupying position (*x*, *y*, *z*) will be

$$i = x . y_g . z_g + y . z_g + z \qquad \text{Equation 6}$$

The program should then divide the room, as evenly as possible, into *N* blocks and create *N* files (one per block) named `My_Room_i.dwm`, *i* being the block index.

Challenge (think about it later): can you propose a more sophisticated program which allows the user to specify the total number of blocks N and then works out the **optimal x_g , y_g and z_g values** depending on room size (X, Y, Z)?

Recall the DWM room spatial discretisation illustrated in Figure 1. Sound wave flow can be simulated in the *x*, *y* and *z* directions because, as illustrated in Figure 4, each node communicates with its immediate neighbours via bidirectional *delay units*.

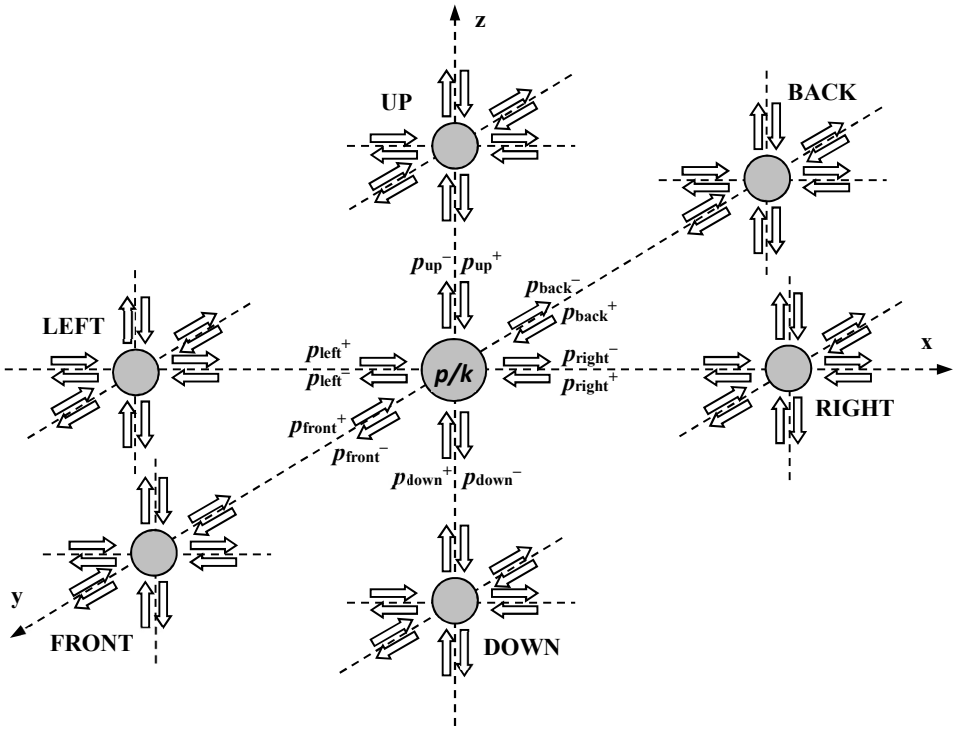

Figure 4 – Structure of a 3D rectilinear mesh scattering junction (centre of the image) seen in articulation with its 6 neighbours. Superscripts '+' and '–' denote flow to and from the junction, respectively. Depending on whether the junction is configured as an air node or a boundary node, the field denoted p/*k* is used to store node pressure *p* at the corresponding point in space or the acoustic reflection factor *k* characterising the respective boundary material.

Time is discretised by those delay units; after each time step in the simulation, the value of the sound wave variable (say *pressure*, *p*) must be computed for each and every node in the mesh. Operation is governed by a very simple iterative algorithm based on a two-pass computation:

Delay pass: the outgoing wave components placed at the output ports are transported to the opposite input ports of their respective neighbouring nodes (the opposite of *up*, *front* and *right* being respectively *down*, *back* and *left* and vice-versa).

$$p^{+}_{neigh.opp} = p^{-}_{node,i} \qquad (i\text{=}up, down, front, back, right, left) \qquad \text{Equation 7}$$

Scattering pass: each node takes its incoming wave components, received in the previous delay pass, to compute outgoing wave components to be passed back to its neighbours.

In **air nodes**, this involves calculating its own new value of the wave variable (Equation 8):

$$p = \frac{1}{3}(p^{+}_{up} + p^{+}_{down} + p^{+}_{front} + p^{+}_{back} + p^{+}_{right} + p^{+}_{left}) \qquad \text{Equation 8}$$

$$p^{-}_{i} = p - p^{+}_{i} \qquad (i\text{=}up, down, front, back, right, left) \qquad \text{Equation 9}$$

In **boundary nodes**, input/output port pairs are governed by the following transfer function, where *k* is the reflection coefficient of the boundary.

$$p^{-}_{i} = k.p^{+}_{i} \qquad (i\text{=}up, down, front, back, right, left) \qquad \text{Equation 10}$$

Modelling sound sources and receivers amounts to injecting and recording audio signal at the desired locations in the room, as illustrated in Figure 5.

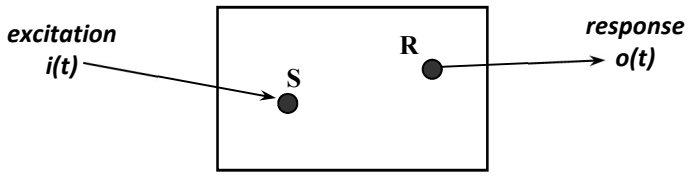

Figure 5 – Model input-output. An excitation signal *i(t)* is injected at point *S* and the resulting response *o(t)* is recorded at receiver location *R*.

Equation 8 shows a Δp pressure change at a given node can be forced by adding $\Delta p/2$ to the content of all its input ports prior to the scattering pass, as shown in Figure 6. Consider a source fixed at node *J* and characterised by the stream of sound samples *excit[n]* (read from the corresponding audio file).

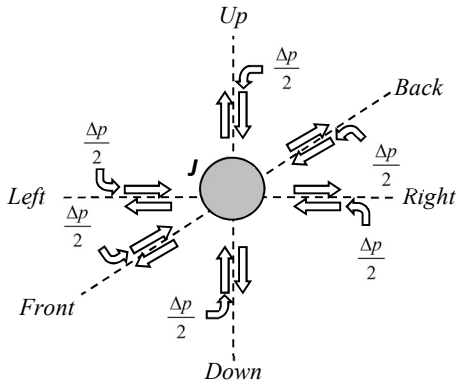

Figure 6 – Injection of a mesh excitation sample Δp at source node *J*.

Mesh excitation is achieved by successively adding the samples of *excit* at that node, using the procedure just described.

The response signal can be obtained from any node in the mesh by simply recording its pressure value *p* in successive iterations. To obtain a *binaural* response, one needs to pickup sound at a pair of receiver nodes appropriately positioned at opposite sides of the virtual listener's head; the corresponding signals can be recorded as a *stereo* soundfile.

Figure 7 outlines the basic structure of the DWM algorithm.

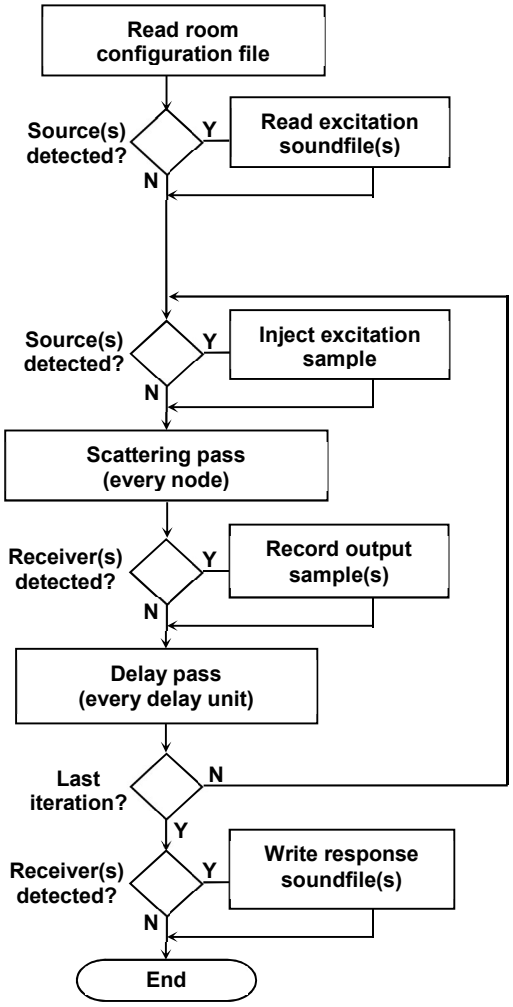

Figure 7 – DWM modelling flowchart.

Task 3 – Develop a 3D DWM room acoustic modelling program in C

The user should choose a previously created room (by specifying its configuration data filename, e.g. `My_Room.dwm`).

Establish a convenient way of specifying the excitation soundfile(s). At least one source must be present. The program may be designed to handle multiple sources.

Establish a convenient way of specifying the response soundfile(s). At least one receiver must be present; it should be possible to specify binaural (i.e. stereo) responses. The program may be designed to handle multiple receivers.

The number of iterations should be greater than the length of the excitation file by an amount corresponding to the room reverberation time (estimated by the user, for example).