

ME543: Sound and Vibration Project Report: Computational Modelling of Acoustic Logic Gates

Ahmet Burak Yildirim¹

¹*Department of Mechanical Engineering, Bilkent University, Ankara 06800, Turkey*
(Dated: March 23, 2023)

This report aims to demonstrate a computational model for the passive logic gates, in particular, OR and XOR gates, which were described in [1]. As the main difference, this project aims to model the logic gates in MATLAB, whereas [1] realizes the passive logic gates both in experiments and COMSOL simulations. The computational model relies on discretizing the wave equation in 2D and embeds the associated rigid boundary conditions throughout the gate geometry. The simulations show that the computational model for the OR gate shows the expected behavior; however model for the XOR gate is incapable of demonstrating the XOR gate characteristics.

INTRODUCTION TO ACOUSTIC LOGIC GATES

Development of logic gates may be considered as one of the most fundamental advancements in modern electronics. Due to the importance of logic gates, novel ways of designing and controlling logic gates have been a hot topic for research for a long time. Recent developments in optics showed that optical logic gates, which are to be used in photonic circuits, can be achieved both theoretically and experimentally, simply relying on the constructive and destructive interference of light [1]. These developments in optics then drew some attention in the field of acoustics and yielded research in acoustic logic gates, which are now being realized thanks to the self-collimation effect [2], curved metamaterial surfaces [3], or passive structures consisting of Helmholtz resonator arrays that were proposed to be modeled in this project [1]. The acoustic logic gates are furthermore mentioned to have potential applications in computing, information processing, and integrated acoustics [1, 3]. With this, one can see the reasoning behind the efforts in the computational modeling of logic gates and its further implications.

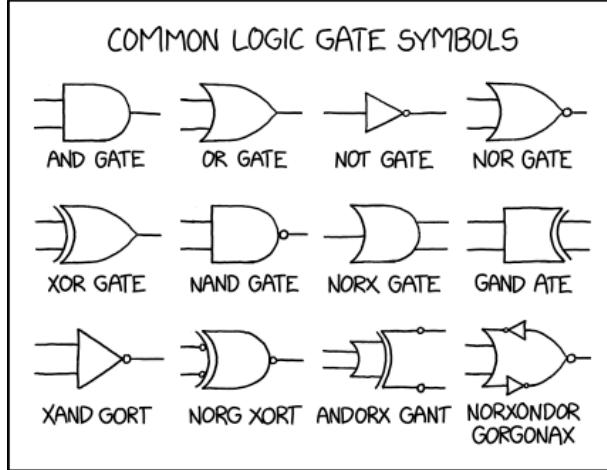


FIG. 1. Comic on logic gates, from xkcd [4]

MODELLING

The modeling procedure consists of the choice of the logic gates of interest, the reasoning behind the choices, the discretization scheme for the wave equation and simulation details, and lastly, the design of the gate geometries. In summary, the choice of the logic gates was determined to **keep the computational demands within the affordable limit**; the discretization is done using **finite difference method** for the spatial solution and **explicit Euler method for temporal solution** of the wave equation; and the geometries were **extracted from the main article** by assigning rigid boundary conditions.

Logic gate selection

The researchers claimed that the passive structures, including several Helmholtz resonators with different width ratios, may be turned into OR, XOR, AND, and NOT logic gates [1].

The parameters which are used to define the geometry are the width and length of a single channel (w, l), the width, gap length, and total length of a single Helmholtz resonator (w_2, l_1, l_2), the inner-channel width (w_1), the wall thickness (w_1), and the width of output channels and length of intersection region (d, d_1). Despite the high number of parameters in the geometric description, these parameters are necessary but insufficient to model the structure. The regions for the incident wave at the inlet and the transmitted wave at the outlet are ill-defined, and their lengths are not prescribed. For simplicity, these regions are assumed to be squares by using the width information. Moreover, the prescribed wall thickness is observed not to satisfy the total length (l). This means that the wall thickness in the lateral direction (w_3) may differ from the wall thickness in the longitudinal direction.

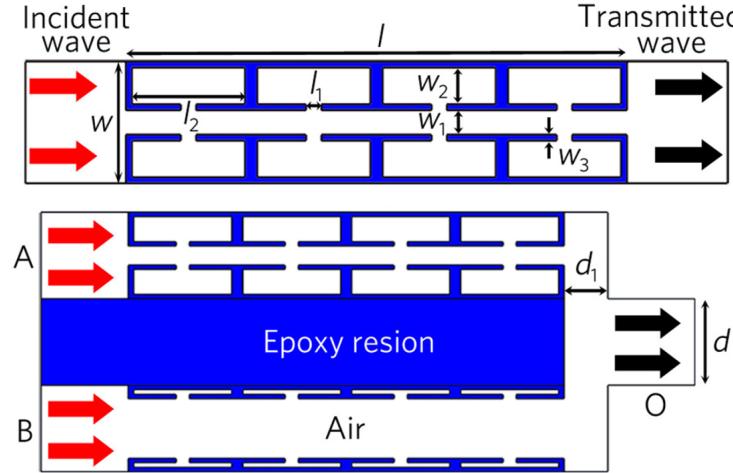


FIG. 2. Schematic of the logic gate geometries. Upper and lower channels may have different width ratios (w_1/w) according to the logic gate of interest. Taken from [1].

In this project, due to the complexity of AND gate geometry, which would have required finer meshing and more computational power, and the triviality of NOT gate geometry which has the identical geometrical structure as XOR gate, modeling of the logic gates is limited to OR and XOR gates [1]. The geometries for OR and XOR logic gates which are included in this project can be found in Figure 3, whereas AND and NOT logic gates which are excluded from this project, can be found in Figures 4 and 5.

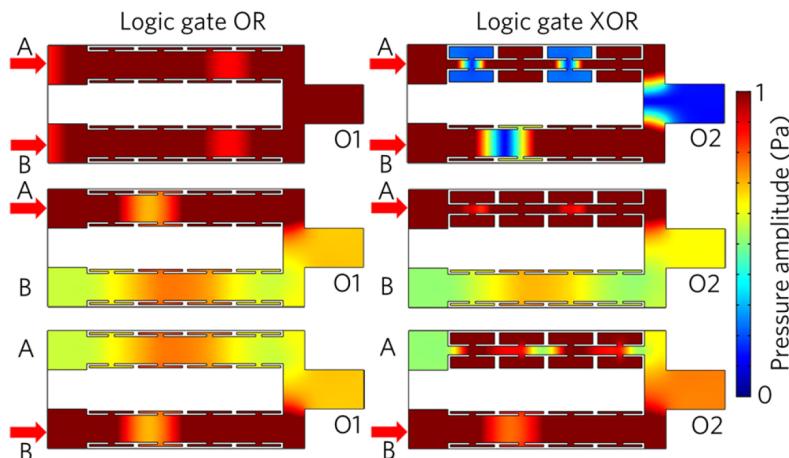


FIG. 3. Exemplary OR and XOR logic gates including the spatial pressure amplitude distributions for different input signals. Taken from [1].

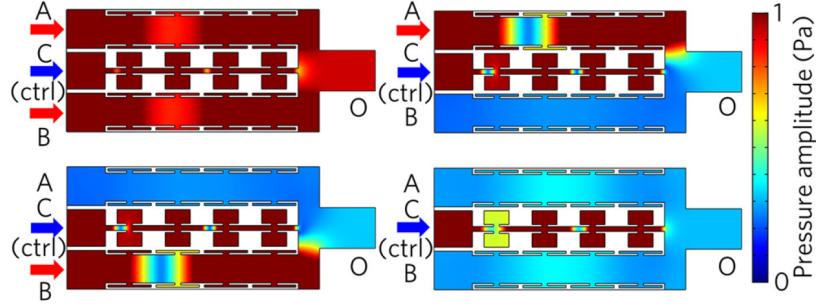


FIG. 4. Exemplary AND logic gate including the spatial pressure amplitude distribution for different input signals. The structure includes a third channel (C) as the control signal. Taken from [1].

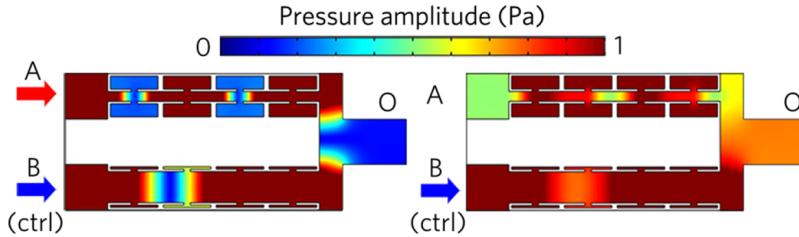


FIG. 5. Exemplary NOT logic gate including the spatial pressure amplitude distribution for different input signals. As will be seen by the following figures, the NOT gate structure is identical to the XOR gate, and the researchers have considered two cases already present in their XOR gate study. Taken from [1].

The proposed logic gate geometries for XOR and OR gates can be found in Figure 5. One can easily model the logic gate for the NOT gate by introducing the width ratios associated with the model into the script. For the AND gate, on the other hand, since the structure consists of not 2 but 3 channels, the current script does not support its modeling of it.

Wave equation

To model the propagation of waves within the logic gate geometry in time, one needs to numerically solve the wave equation in 2D (x, y):

$$\frac{1}{c^2} \frac{\partial^2 p}{\partial t^2} = \frac{\partial^2 p}{\partial x^2} + \frac{\partial^2 p}{\partial y^2} \quad (1)$$

where p is the pressure and c is the speed of sound.

For the numerical solution, there are multiple schemes present: finite difference method, finite element method, and finite volume method. The finite difference method is selected and used in this project to ease the implementation. Time advancement is obtained using the implicit method (backward Euler method) that is known to be unconditionally stable. With these methods, the error propagation within the solution is 1st order in time and 2nd order in space.

The implicit method has the following equation for time advancement:

$$\frac{p_{i,j}^{t+1} - p_{i,j}^t}{\Delta t} + \frac{c}{2\Delta x} (p_{i+1,j+1}^{t+1} - p_{i-1,j-1}^{t+1}) = 0 \quad (2)$$

where Δt is the timestep and Δx is the grid size.

This equation can be used to determine the stability condition, often called CFL (Courant–Friedrichs–Lowy) condition, that is $CFL = c \frac{\Delta t}{\Delta x} < 1$. For higher stability, the CFL value is often selected as ≤ 0.7 , which will be set to 0.7 in this

project and used to determine the timestep Δt for the used grid size $\Delta x = 1$.

The finite difference discretization results in the following set of equations:

$$p_{i,j,t+\Delta t} = 2p_{i,j,t} - p_{i,j,t-\Delta t} + CFL^2 (p_{i+1,j,t} + p_{i,j+1,t} + p_{i-1,j,t} + p_{i,j-1,t} - 4p_{i,j,t}) \quad (3)$$

$$p_{0,j,t+\Delta t} = p_{1,j,t} - \frac{CFL - 1}{CFL + 1} (p_{1,j,t+\Delta t} - p_{0,j,t}) \quad (4)$$

$$p_{L,j,t+\Delta t} = p_{L-1,j,t} - \frac{CFL - 1}{CFL + 1} (p_{L-1,j,t+\Delta t} - p_{L,j,t}) \quad (5)$$

$$p_{i,0,t+\Delta t} = p_{i,1,t} - \frac{CFL - 1}{CFL + 1} (p_{i,1,t+\Delta t} - p_{i,0,t}) \quad (6)$$

$$p_{i,L,t+\Delta t} = p_{i,L-1,t} - \frac{CFL - 1}{CFL + 1} (p_{i,L-1,t+\Delta t} - p_{i,L,t}) \quad (7)$$

where eqn. (3) is to be used for the grids not near the boundaries, eqn. (4) and eqn. (5) are to be used for the grids neighboring the boundaries for $x = 0$ and $x = L$, and eqn. (6) and eqn. (7) to be used for the grids neighboring the boundaries for $y = 0$ and $y = L$ given that 0 and L are some coordinates corresponding to the nearby of boundaries. One can note that with eqn.s (4-7), the boundaries are assumed to satisfy **rigid boundary conditions**, as prescribed by the researchers [1].

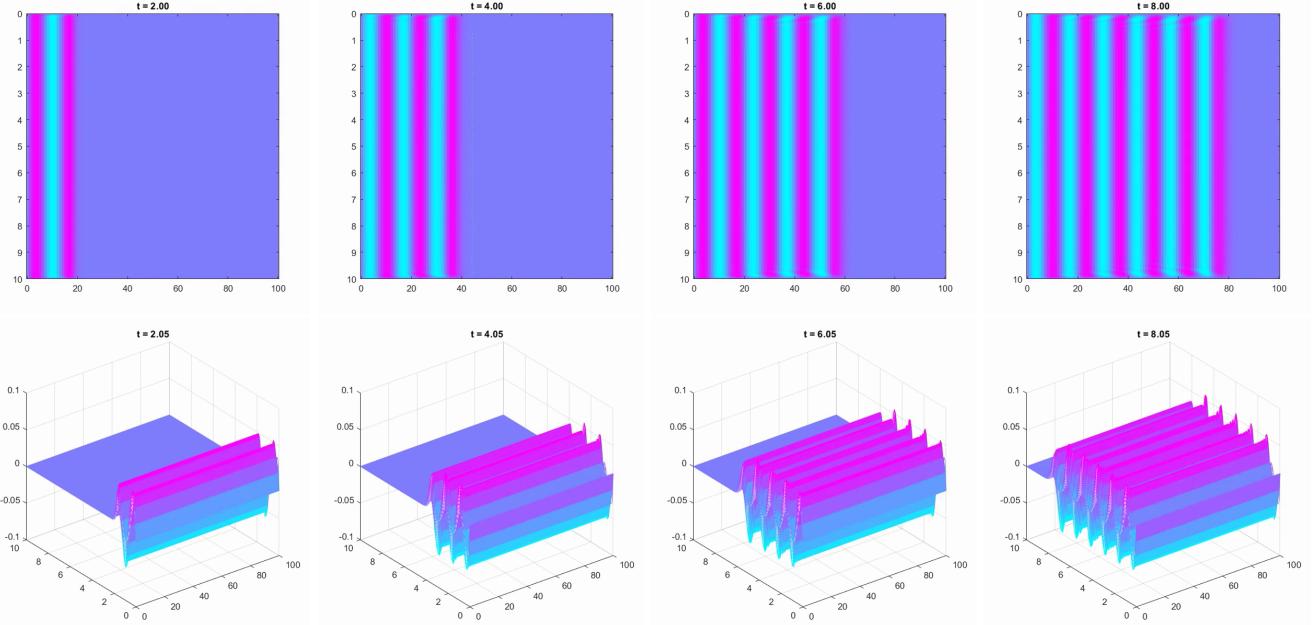


FIG. 6. Propagation of a sinusoidal wave in an environment without boundaries using implicit and finite difference methodology for the numerical solution. Upper images show the propagation in 2D with pressure magnitude, and lower images show the corresponding sine waves in 3D.

Boundaries

To construct the logic gate geometries shown for OR and XOR gates in Figure 3, one needs to define the geometric parameters to be used as an input and will determine the coordinates of the rigid boundaries.

As mentioned before, the geometric information published by the researchers is insufficient to generate the geometries [1]. Thus, while filling the missing parts with the most reasonable values by visually investigating the proposed

geometries, see Figure 3, one must note that this discrepancy may lead to symmetry and wave cancellation errors. The geometric parameters which are used to construct the simulations are tabulated as follows. One can note that all the parameters are nondimensional with respect to the input signal wavelength, which requires determining the input sine wave frequencies so that frequency in air satisfies the dimensional relation, $f_{air} = 3430$ Hz.

TABLE I. Geometric Inputs Used for Simulation Construction

Geometric Property	Value	Geometric Property	Value
Input signal wavelength (λ)	<i>to be set</i>	Vertical wall thickness (w_3)	$0.05 w$
Length of upper/lower channel (l)	0.5λ	Horizontal wall thickness (w_4)	$\frac{1}{5} (l - 4 l_2)$
Width of upper/lower channel (w)	0.5λ	Width of upper channel Helmholtz resonator ($w_{2,upper}$)	$\frac{1}{2} (w - w_{1,upper} - 4 w_3)$
Width of output channel (d)	0.1λ	Width of lower channel Helmholtz resonator ($w_{2,lower}$)	$\frac{1}{2} (w - w_{1,lower} - 4 w_3)$
Width of intersection region (d_1)	0.055λ	Total length of logic gate (L_x)	$w + l + d_1 + d$
Gap length of Helmholtz resonator (l_1)	$0.03 l$	Total width of logic gate (L_y)	$2 w + d$
Total length of Helmholtz resonator (l_2)	$0.23 l$	Grid size in x-direction (ΔX)	1
Width of upper channel Helmholtz resonator ($w_{1,upper}$)	OR Gate: 0.69λ XOR Gate: 0.69λ	Grid size in y-direction (ΔY)	1
Width of lower channel Helmholtz resonator ($w_{1,lower}$)	OR Gate: 0.69λ XOR Gate: 0.22λ	Speed of sound ($c = \lambda f$)	1

With the parameters given in Table 1, the following OR and XOR geometries were constructed in a MATLAB environment, which then used rigid boundary coordinates for wave propagation simulations.

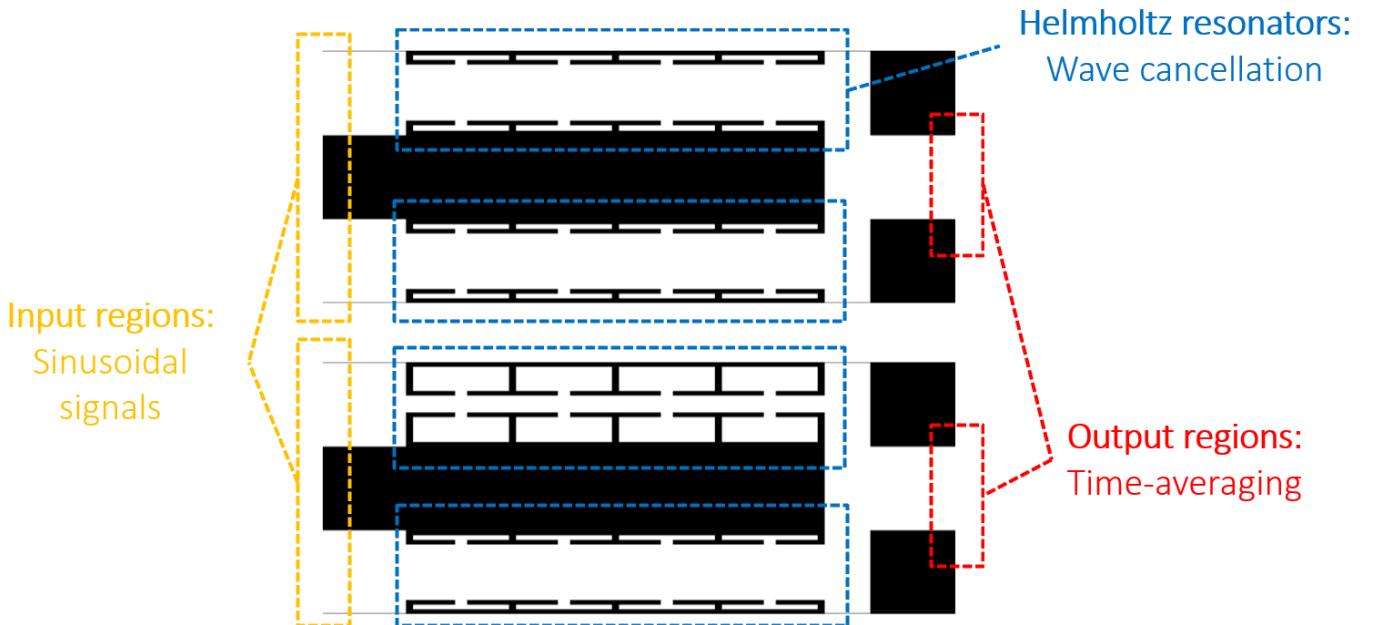


FIG. 7. OR and XOR logic gates geometries constructed in MATLAB. The upper geometry belongs to the OR gate, and the lower gate belongs to the XOR gate. Helmholtz resonators and input and output regions are depicted in the figure.

RESULTS AND COMPARISON

Recalling that the geometric parameters were nondimensional with respect to the input wavelength λ , one must determine the input sine wave frequency in terms of dimensional numbers to replicate the input conditions provided by the researchers [1]. The sinusoidal input wave has the following form:

$$p(x = 0, \text{inlet}_{y=0} \text{ to } y=\text{end}, t) = A \sin(2\pi t f); \quad (8)$$

The amplitude term, A , in eqn. (8) may be replaced with either 1 or 0 according to the input signal conditions, which may have the following combinations: {Upper channel, Lower channel} = {1, 1}, {1, 0}, {0, 1}. Note that the {0, 0} condition is disregarded due to its triviality. During the simulations, the pressure amplitudes have been considered for visualization and averaging due to the importance of the amplitude but not the magnitude in the acoustical sense. At this point, one can further define a nondimensional frequency by using the frequency of air (f_{air}), as researchers used air to fill the channels, the speed of sound c and the wavelength of the input signal, which determines the geometric parameters: $f' = f \times \lambda/c$. This approach becomes convenient when dealing with the working band, where the researchers showed that OR and XOR gates may be realized for the range $f \approx 3000 - 3800$ Hz. This range approximately corresponds to $f' = 0.9 - 1.1$ regarding the nondimensional frequency. After the definition of nondimensional frequency, the wave propagation simulations were conducted by sweeping over 10 frequency values ($f' = 0.1, 0.3, \dots, 1.7, 1.9$). The average pressure values at the outlet region were measured at the end of each simulation for a particular gate (OR or XOR) and a particular input state ({1, 1}, {1, 0} or {0, 1}). Each simulation lasted for 20000 timesteps, and a total of 60 simulations were conducted with this runtime for each (10 frequencies, 2 gates, 3 input states). Due to the visualization efforts and better readability of the script, the script was not optimized, and each of the 60 simulations required 20+ minutes of runtime with the available computational resources. Consequently, it became inconvenient to create a graphical user interface and control every geometric parameter (Table 1) and input parameter (eqn. 8), although there were efforts.

By sweeping over the frequencies, one determines whether the logic gates can be realized with the used numerical scheme without worrying about the dimensional numbers or possible problems due to geometric and meshing. Simulation snapshots from 5 different frequency values ($f' = 0.1, 0.5, 0.9, 1.3, 1.7$) are presented in the following figures, Figures 8-13.

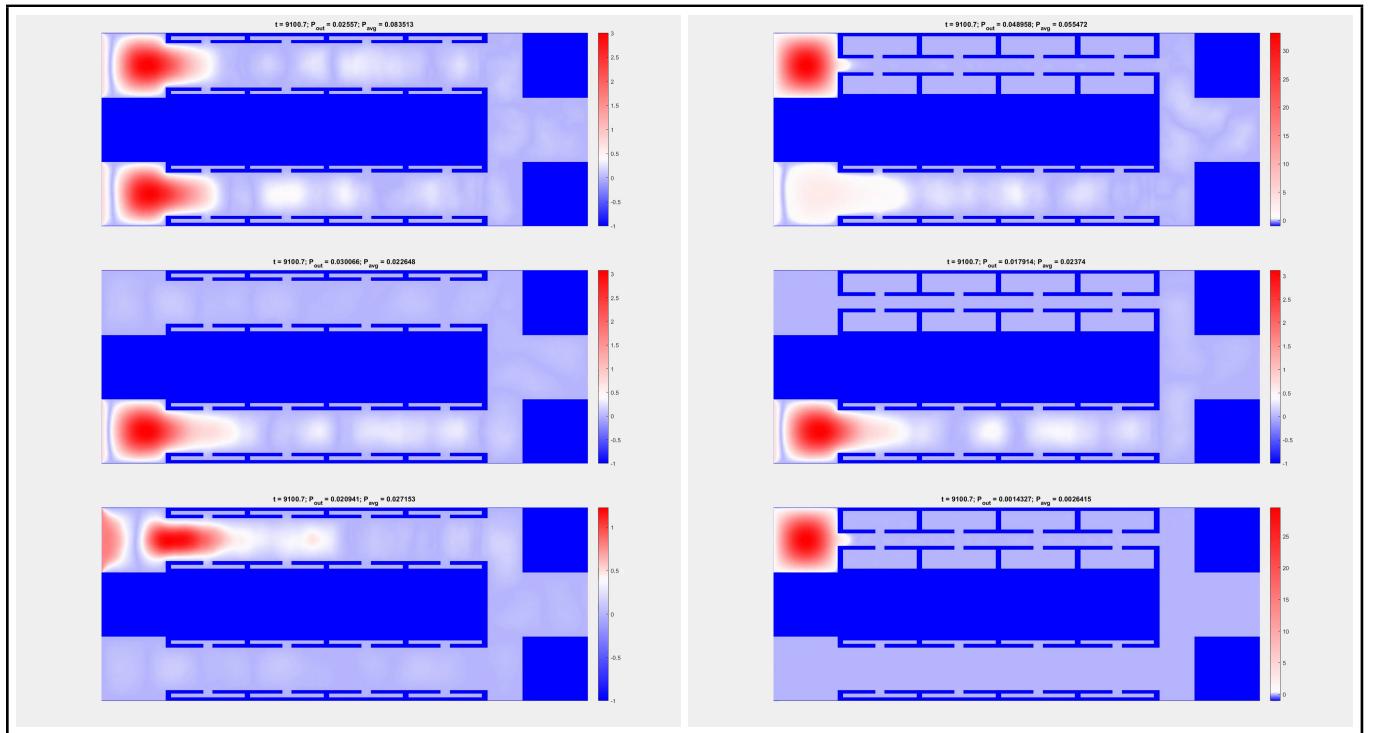


FIG. 8. Snapshots from the simulations with $f' = 0.1$ at a timestep approximately 60% of the total runtime. Left: OR gate, Right: OR gate. Top: Input signal of {1, 1}, Middle: Input signal of {1, 0}, Bottom: Input signal of {0, 1}.

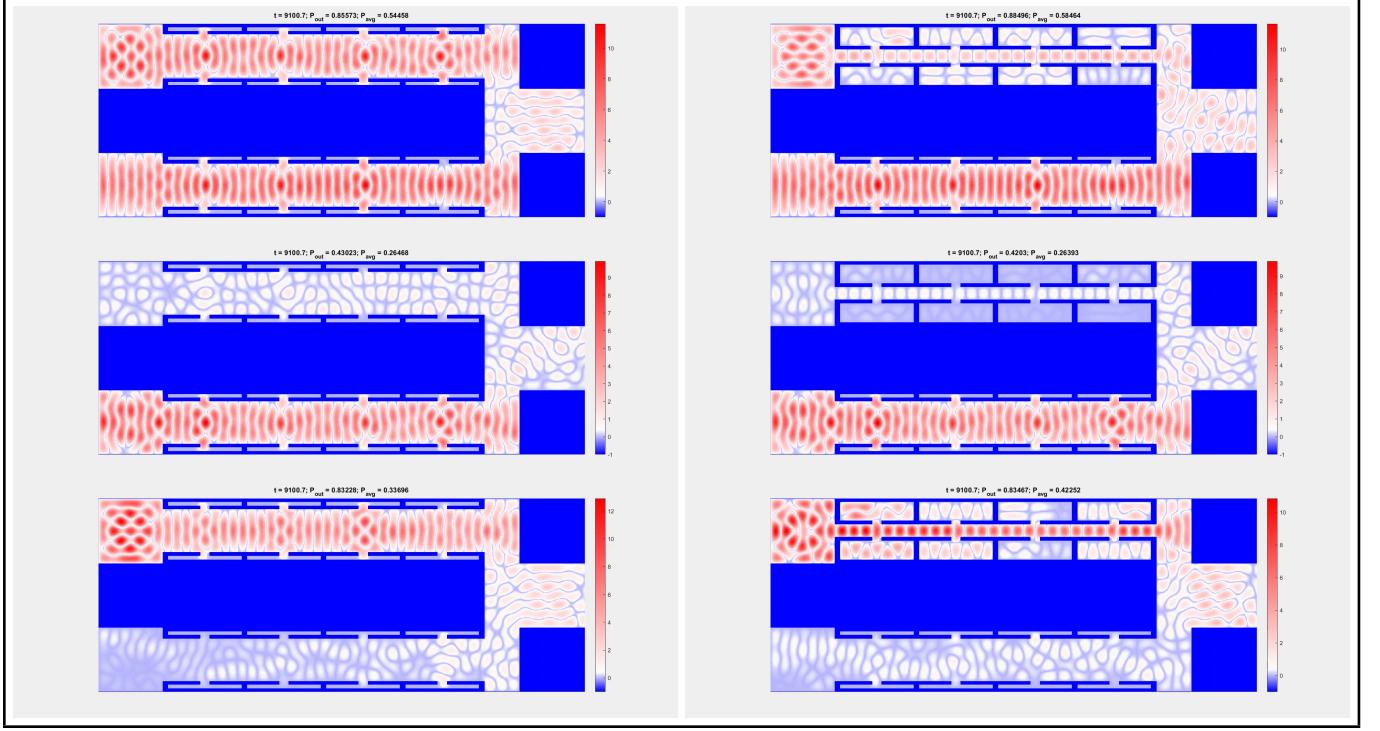


FIG. 9. Snapshots from the simulations with $f' = 0.5$ at a timestep approximately 60% of the total runtime. Left: OR gate, Right: OR gate. Top: Input signal of $\{1, 1\}$, Middle: Input signal of $\{1, 0\}$, Bottom: Input signal of $\{0, 1\}$.

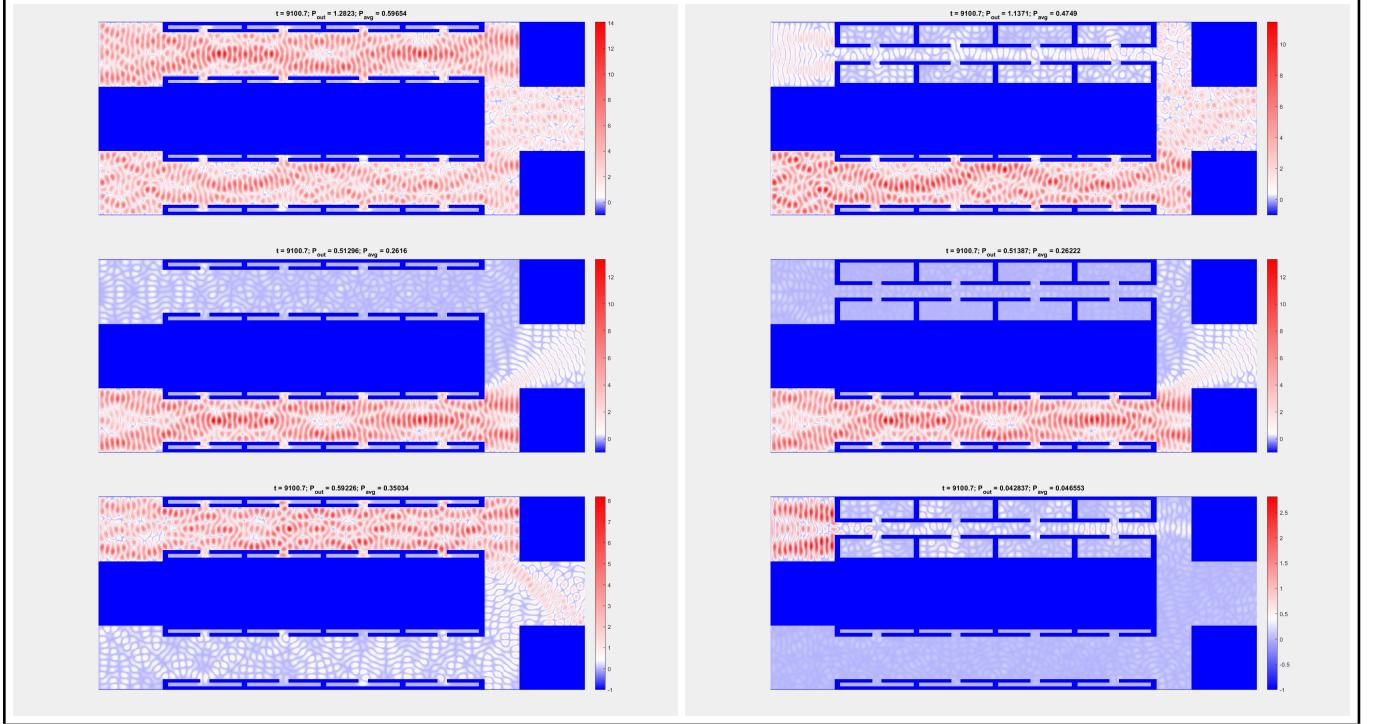


FIG. 10. Snapshots from the simulations with $f' = 0.9$ at a timestep approximately 60% of the total runtime. Left: OR gate, Right: OR gate. Top: Input signal of $\{1, 1\}$, Middle: Input signal of $\{1, 0\}$, Bottom: Input signal of $\{0, 1\}$. Note that this frequency value is within the limits of the working band, and the output pressure time averages should satisfy the truth table accordingly unless there is no restraint due to the numerical scheme or the geometry.

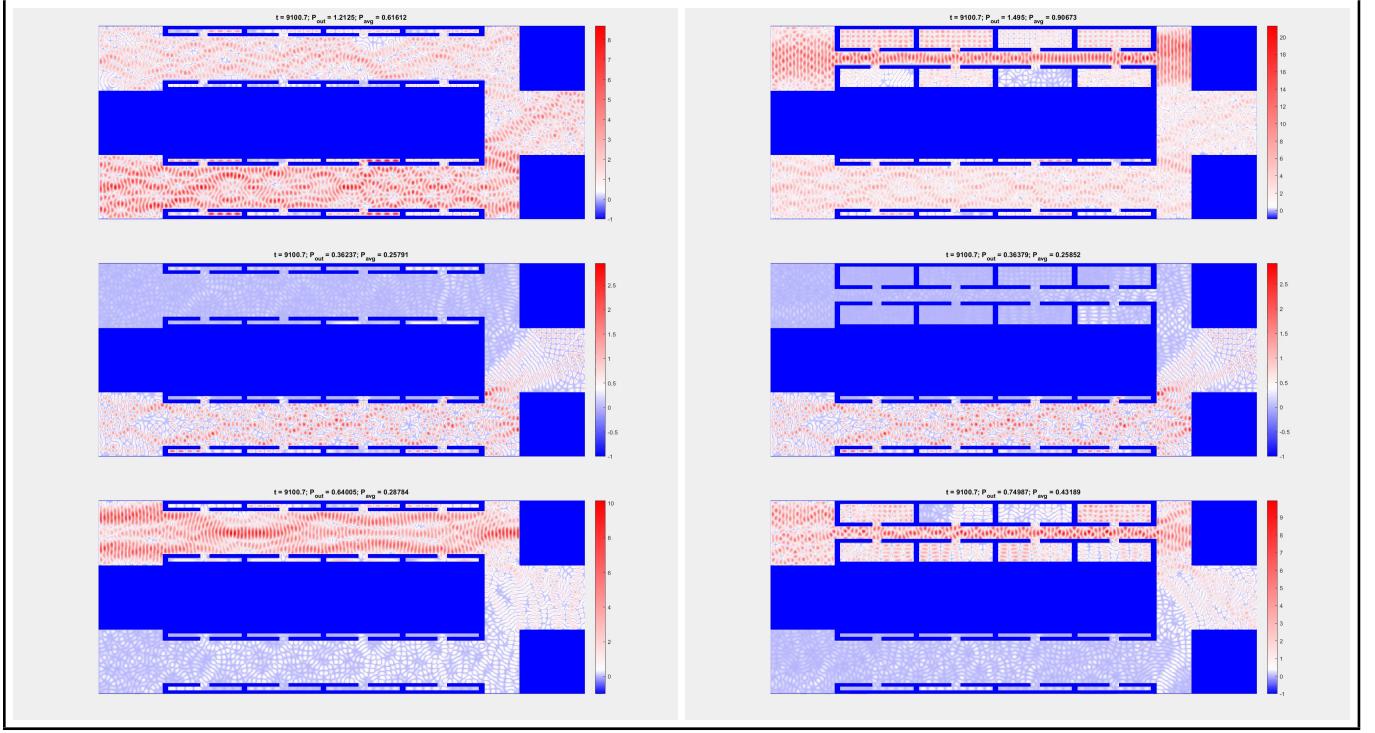


FIG. 11. Snapshots from the simulations with $f' = 1.3$ at a timestep approximately 60% of the total runtime. Left: OR gate, Right: OR gate. Top: Input signal of $\{1, 1\}$, Middle: Input signal of $\{1, 0\}$, Bottom: Input signal of $\{0, 1\}$.

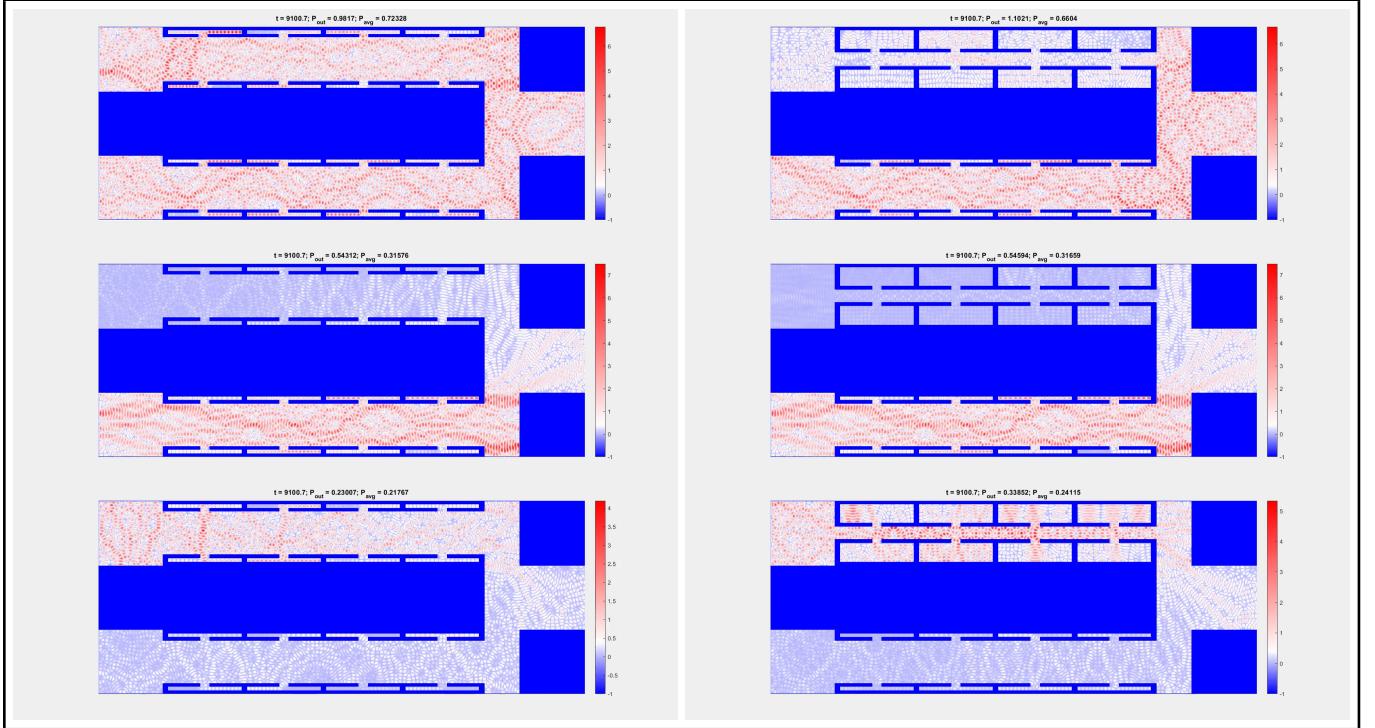


FIG. 12. Snapshots from the simulations with $f' = 1.7$ at a timestep approximately 60% of the total runtime. Left: OR gate, Right: OR gate. Top: Input signal of $\{1, 1\}$, Middle: Input signal of $\{1, 0\}$, Bottom: Input signal of $\{0, 1\}$. Note that as the frequency increases, the Helmholtz resonators become useless to create a phase difference and cancel waves, but rather become regions that waves can travel without significantly interrupting the other waves.

The time-averaged pressure amplitudes at the outlet region, where neither the averaging region nor the averaging methodology was defined by the researchers, may be obtained for the simulations done over a range of frequencies.

The results presented in the following figure show that computational modeling of the OR gate was **successful** as the OR gate results in the following truth table:

Input signal: {1, 1} → Output signal: {1}
 Input signal: {1, 0} → Output signal: {1}
 Input signal: {0, 1} → Output signal: {1}

where the threshold pressure amplitude value of 0.4 Pa was used in accordance with the researchers [1].

On the other hand, the following figure shows that computational modeling of the XOR gate was **unsuccessful** as the XOR gate did not result in the ideal truth table:

Input signal: {1, 1} → Output signal: {0}
 Input signal: {1, 0} → Output signal: {1}
 Input signal: {0, 1} → Output signal: {1}

It rather resulted in the following truth table:

Input signal: {1, 1} → Output signal: {1}
 Input signal: {1, 0} → Output signal: {0}
 Input signal: {0, 1} → Output signal: {1}

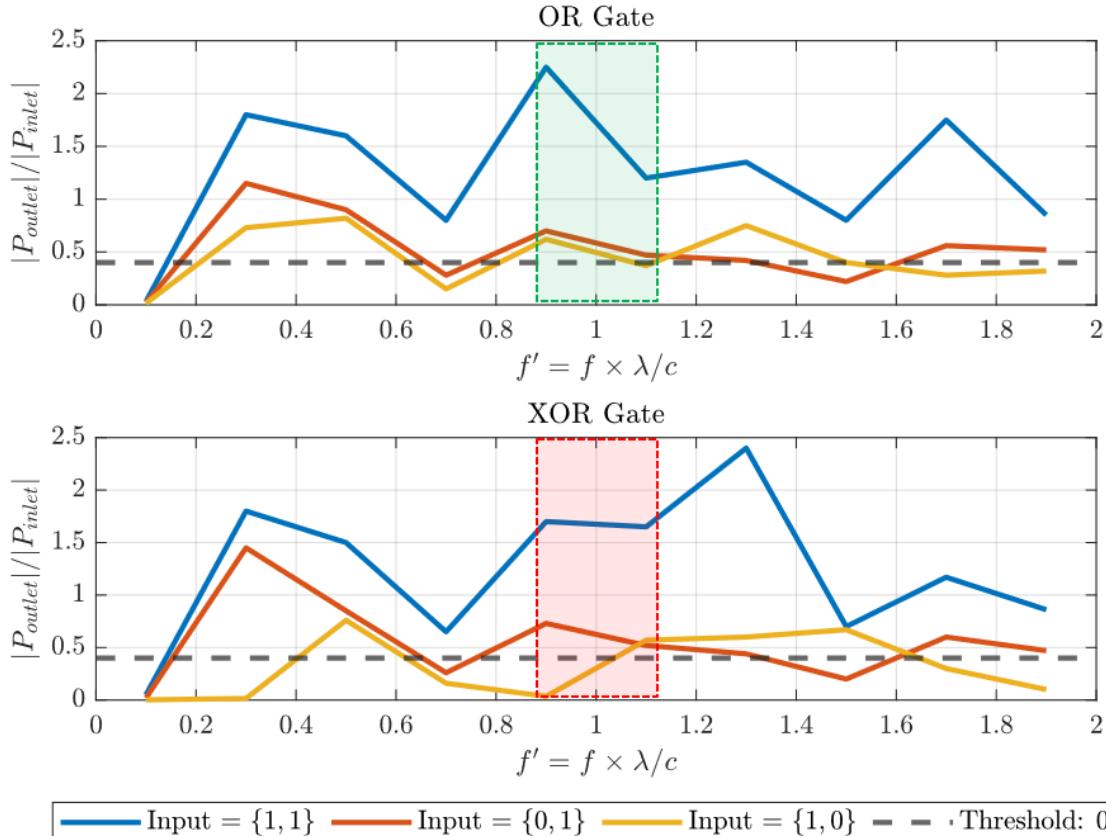


FIG. 13. Time-averaged output pressures over the range of nondimensional frequencies. The working ranges that are proposed by the researchers are depicted on the plots, where the computational model of OR gate was able to realize the expected results, whereas XOR gate was not able to realize not only in the working band but over non of the studied frequency values.

CONCLUSION

To conclude this study, we may consider some observations done throughout the project, starting from the construction of the simulations to the obtained results:

As can be seen in Figure 14, the wave canceling and waves which are emerging in the same space periodically in time are observed. This may be argued due to the Helmholtz resonators placed in the channels. However, as the time-averaged pressure amplitudes within the whole logic gate geometry are not studied, but the focus was only on the outlet region due to memory limitations, there is no quantitative analysis of the pressure distribution over the logic gates. One may note from Figures 3-5 that the COMSOL analysis done by the researchers showed that the pressure distribution was not uniform but had low amplitude regions, such as some parts of the regions or inside of the Helmholtz resonators.

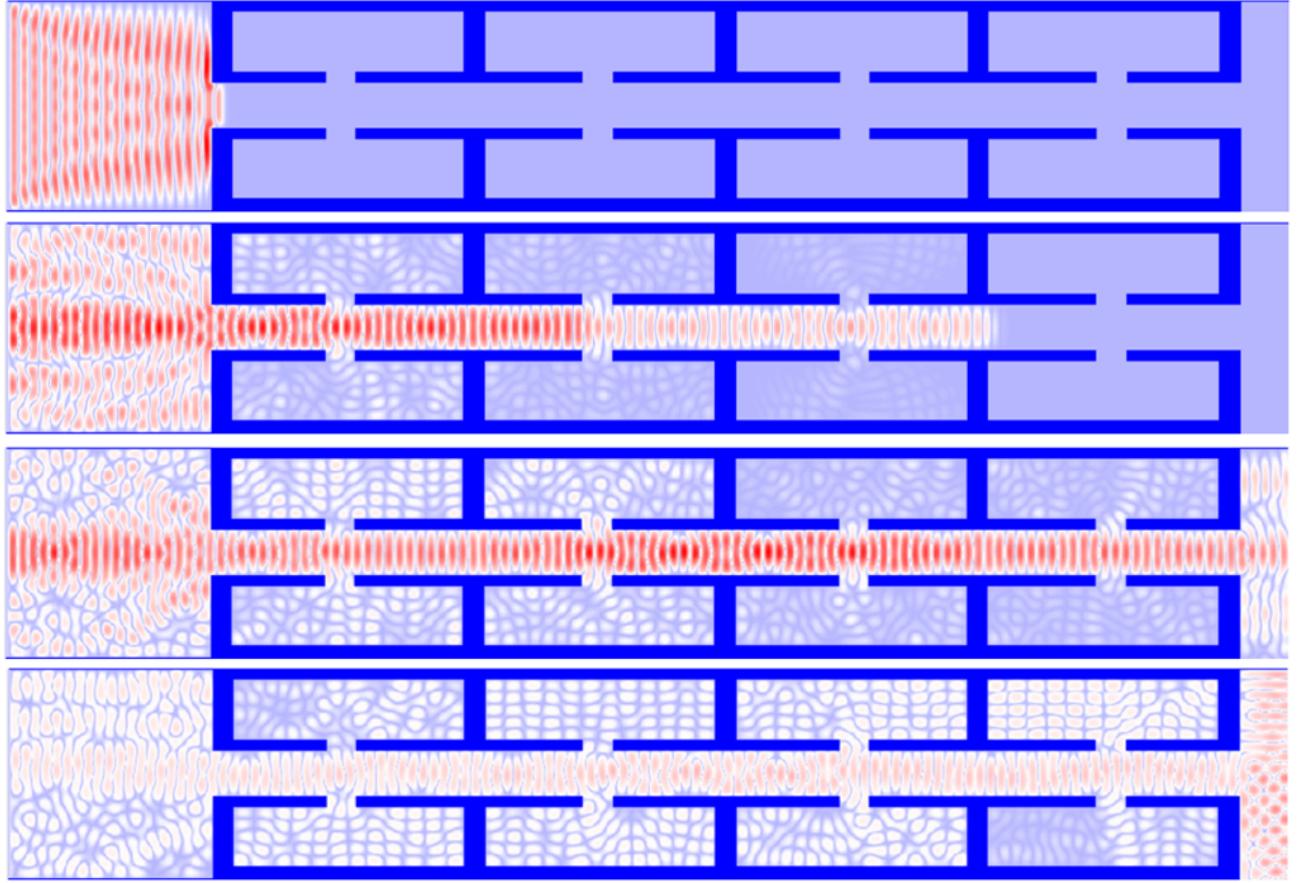


FIG. 14. $f=1.9$.

As the main goal of this project, one of the logic gates was successfully simulated in MATLAB, which was achieved regardless of the uncertainties in geometry, working band, and averaging methodology. On the other hand, despite the OR gate model showing the expected behavior, the XOR gate model was not able to reproduce the results for none of the studied frequency values. This observation can be interpreted as there was no shift in the working band for any reason, so the expected behavior might have been observed for a different working band. The insufficiency is due to the numerical method and logic gate geometry, such as meshing.

The last observation may be further supported by the fact that the gates did not preserve the symmetry of pressure amplitudes, either due to geometric asymmetries that are present but neglected in the script or due to the accumulation of errors in the numerical solution of the wave equation. A naive approach to eliminate the current inefficiencies may be using the finite element method for the solution of the wave equation and decreasing the grid sizes, which should result in fewer errors within the numeric and better meshing.

REFERENCES

- [1] Wang, Y., Xia, Jp., Sun, Hx. et al., "Binary-phase acoustic passive logic gates," *Nature Sci Rep*, 2019. [Online]. Available: <https://doi.org/10.1038/s41598-019-44769-0>. [Accessed: 19-May-2022].
- [2] Ting Zhang, Ying Cheng, Jian-zhong Guo, et al., "Acoustic logic gates and Boolean operation based on self-collimating acoustic beams," *Appl. Phys. Lett.*, 2015. [Online]. Available: <https://doi.org/10.1063/1.4915338>. [Accessed: 19-May-2022].
- [3] Yu-Jing Lu et al., "Acoustic logic gates by a curved waveguide with ultrathin metasurfaces," *J. Phys. D: Appl. Phys.*, 2020. [Online]. Available: <https://doi.org/10.1088/1361-6463/ab483e>. [Accessed: 19-May-2022].
- [4] "Logic Gates," *xkcd*. [Online]. Available: <https://xkcd.com/2497/>. [Accessed: 16-May-2022].
- [5] Komatsuzaki T., Iwata Y., Morishita S., "Modelling of Incident Sound Wave Propagation around Sound Barriers Using Cellular Automata," *Springer*, 2012. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-642-33350-7_40. [Accessed: 19-May-2022].
- [6] Chapra, S. C., & Canale, R. P., "Numerical Methods for Engineers," *McGraw-Hill Higher Education*, 2015.

APPENDIX: MATLAB SCRIPT

```

1 clc; clear
2
3 %% Wave equation: (d^2/dx^2 +cd d^2/dy^2) p = 1/c^2 (d^2/dt^2) p
4
5 %% OR Gate + XOR Gate, Total runtime: 20000
6 % Domain definition
7 c = 1;
8 lambda = 2000; % 10 cm ... 10 cm to 2000 units lambda
9 prefactors = (0.1:0.2:1.9); % as a prefactor for frequency sweep
10 OUTPUT.VALUES = []; % save output pressure values
11
12 for i = 1:size(prefactors,2)
13
14 prefactor = prefactors(i); disp("prefactor is " + num2str(prefactor) + ".");
15 freq = prefactor*c/lambda;
16
17 tic
18 input = 11;
19 filename = "OR.input" + num2str(input) + "prefactor." + num2str(prefactor) + ".gif";
20 filename2 = "OR.input" + num2str(input) + "prefactor." + num2str(prefactor) + ".png";
21 out = solver11.OR(lambda, filename, freq, filename2); disp("solver11.OR is completed.");
22 OUTPUT.VALUES = [OUTPUT.VALUES; out];
23
24 filename = "XOR.input" + num2str(input) + "prefactor." + num2str(prefactor) + ".gif";
25 filename2 = "XOR.input" + num2str(input) + "prefactor." + num2str(prefactor) + ".png";
26 out = solver11.XOR(lambda, filename, freq, filename2); disp("solver11.XOR is completed.");
27 OUTPUT.VALUES = [OUTPUT.VALUES; out];
28
29 input = 10;
30 filename = "OR.input" + num2str(input) + "prefactor." + num2str(prefactor) + ".gif";
31 filename2 = "OR.input" + num2str(input) + "prefactor." + num2str(prefactor) + ".png";
32 out = solver10.OR(lambda, filename, freq, filename2); disp("solver10.OR is completed.");
33 OUTPUT.VALUES = [OUTPUT.VALUES; out];
34
35 filename = "XOR.input" + num2str(input) + "prefactor." + num2str(prefactor) + ".gif";
36 filename2 = "XOR.input" + num2str(input) + "prefactor." + num2str(prefactor) + ".png";
37 out = solver10.XOR(lambda, filename, freq, filename2); disp("solver10.XOR is completed.");
38 OUTPUT.VALUES = [OUTPUT.VALUES; out];
39
40 input = 01;
41 filename = "OR.input01" + "prefactor." + num2str(prefactor) + ".gif";
42 filename2 = "OR.input01" + "prefactor." + num2str(prefactor) + ".png";
43 out = solver01.OR(lambda, filename, freq, filename2); disp("solver01.OR is completed.");
44 OUTPUT.VALUES = [OUTPUT.VALUES; out];
45
46 filename = "XOR.input01" + "prefactor." + num2str(prefactor) + ".gif";
47 filename2 = "XOR.input01" + "prefactor." + num2str(prefactor) + ".png";
48 out = solver01.XOR(lambda, filename, freq, filename2); disp("solver01.XOR is completed.");
49 OUTPUT.VALUES = [OUTPUT.VALUES; out];
50 toc
51
52 end
53
54 filename = 'savefile.mat';
55 save(filename)
56
57 function out = solver11.OR(lambda, filename, freq, filename2)
58
59 % Geometric constraints
60 l = lambda / 2;
61 w = lambda / 10;
62 d = 0.1 * lambda;
63 d1 = 0.055 * lambda;
64
65 l1 = (0.03 * 1);
66 l2 = (0.23 * 1);
67 w1.upper = (0.69 * w); % changing
68 w1.lower = (0.69 * w); % changing
69 w3 = (0.05 * w);
70 w2.upper = ((w - w1.upper - 4 * w3) / 2);
71 w2.lower = ((w - w1.lower - 4 * w3) / 2);
72 w4 = ((l-4*l2)/5);

```

```

73 id2 = ((1-4*11)/8);
74 id3 = ((12-11)/2);
75
76 Lx = w + l + d1 + d;
77 Ly = 2*w + d;
78 dx = 1; dy = 1;
79 nx = fix(Lx/dx) + 2; ny = fix(Ly/dy) + 2;
80
81 x = linspace(0, Lx, nx); y = linspace(0, Ly, ny);
82
83 % Field definition
84
85 p = 0*ones(nx,ny);
86 p.prev = p; p.next = p;
87
88 CFL = 0.7; % c * dt/dx
89 c = 1; %c/lambda; freq1 = c/lambda; freq2 = c/lambda;
90 dt = CFL*dx/c;
91 output.pres = {};
92 time = {};
93
94 % Initial conditions
95 T = 20000*dt; % Runtime
96
97 t = 0;
98 f = figure(1);
99 set(gcf,'Position',[100 100 1400 500]);
100
101 while t<T
102     % Absorbing Boundaries
103     p.next(1,:) = p(2,:)+((CFL-1)/(CFL+1))*(p.next(2,:)-p(1,:));
104     p.next(end,:) = p(end-1,:)+((CFL-1)/(CFL+1))*(p.next(end-1,:)-p(end,:));
105
106     % Rigid Boundaries
107     p(1:end, [1 end]) = 0; % Upper and lower boundaries
108     %p([1 end], 1:end) = 0; % Left and right boundaries
109     p(l:w:l, w:dx:end-w-2*dx) = 0; % Middle region
110     p(w:l:d1:end, [l:w w+d:dx:end]) = 0; % Right bottom and top regions
111     p(w:w+l, [l+dx:w3+dx w-w3-dx:w]) = 0; p(w:w+l, [end-w3:end-dx end-w-dx:end-w+w3-dx]) = 0; % 1/3
112
113     p([w:w+w4 w+w4+12:w2+w4+12 w+2*w4+2*12:w+3*w4+2*12 w+3*w4+3*12:w+4*w4+3*12 w+4*w4+4*12:w+5*w4+4*12], [w3+2*dx:w3+dx+w2.upper ...
114         -w-3-1*dx-w2.upper:w-w3-2*dx]) = 0; % 2/3
115     p([w:w+w4 w+w4+12:w2+w4+12 w+2*w4+2*12:w+3*w4+3*12:w+4*w4+3*12 w+4*w4+4*12:w+5*w4+4*12], [end-w3-w2.lower:end-w3-dx ...
116         -end-w+w3+0*dx:end-w+w3+w2.lower+1*dx]) = 0; % 2/3
117     p([w:w+w4+1:d1 w+w4+12-1d3:w+2*w4+12+1d3 w+2*w4+2*12-1d3:w+3*w4+2*12+1d3 w+3*w4+3*12-1d3:w+4*w4+3*12+1d3 w+4*w4+4*12-1d3:w+5*w4+4*12], ...
118         [2*dx+w2.upper:w3:w3+2*dx+w2.upper+w-3-w2.upper-w3-2*dx:w-w2.upper-w3-2*dx]) = 0; % 3/3
119     p([w:w+w4+1:d3 w+w4+12-1d3:w+2*w4+12+1d3 w+2*w4+2*12-1d3:w+3*w4+2*12+1d3 w+3*w4+3*12-1d3:w+4*w4+3*12+1d3 w+4*w4+4*12-1d3:w+5*w4+4*12], ...
120         [end-w3-w2.lower-w3-1*dx:end-w2.lower-w3-1*dx end-w+2*dx+w2.lower+w3:end-w+w3+2*dx+w2.lower+w3]) = 0; % 3/3
121
122 t = t+dt;
123 p.prev = p; p = p.next;
124
125 % Source 1
126 p(2, end-lambda/10:end-1) = 1*sin(2*pi*t*freq);
127
128 % Source 2
129 p(2, 2:lambda/10-1) = 1*sin(2*pi*t*freq);
130
131 % Rigid Boundaries
132 p(1:end, [1 end]) = 0; % Upper and lower boundaries
133 %p([1 end], 1:end) = 0; % Left and right boundaries
134 p(l:w:l, w:dx:end-w-2*dx) = 0; % Middle region
135 p(w:l:d1:end, [l:w w+d:dx:end]) = 0; % Right bottom and top regions
136 p(w:w+l, [l+dx:w3+dx w-w3-dx:w]) = 0; p(w:w+l, [end-w3:end-dx end-w-dx:end-w+w3-dx]) = 0; % 1/3
137
138     p([w:w+w4 w+w4+12:w2+w4+12 w+2*w4+2*12:w+3*w4+3*12:w+4*w4+3*12 w+4*w4+4*12:w+5*w4+4*12], [w3+2*dx:w3+dx+w2.upper ...
139         -w-3-1*dx-w2.upper:w-w3-2*dx]) = 0; % 2/3
140     p([w:w+w4 w+w4+12:w2+w4+12 w+2*w4+2*12:w+3*w4+2*12 w+3*w4+3*12:w+4*w4+3*12 w+4*w4+4*12:w+5*w4+4*12], [end-w3-w2.lower:end-w3-dx ...
141         -end-w+w3+0*dx:end-w+w3+w2.lower+1*dx]) = 0; % 2/3
142     p([w:w+w4+1:d3 w+w4+12-1d3:w+2*w4+12+1d3 w+2*w4+2*12-1d3:w+3*w4+2*12+1d3 w+3*w4+3*12-1d3:w+4*w4+3*12+1d3 w+4*w4+4*12-1d3:w+5*w4+4*12], ...
143         [2*dx+w2.upper:w3:w3+2*dx+w2.upper+w-3-w2.upper-w3-2*dx:w-w2.upper-w3-2*dx]) = 0; % 3/3
144     p([w:w+w4+1:d3 w+w4+12-1d3:w+2*w4+12+1d3 w+2*w4+2*12-1d3:w+3*w4+2*12+1d3 w+3*w4+3*12-1d3:w+4*w4+3*12+1d3 w+4*w4+4*12-1d3:w+5*w4+4*12], ...
145         [end-w3-w2.lower-w3-1*dx:end-w2.lower-w3-1*dx end-w+2*dx+w2.lower+w3:end-w+w3+2*dx+w2.lower+w3]) = 0; % 3/3
146
147 % Finite difference solution
148 for i = 2:nx-1
149     for j = 2:ny-1
150         p.next(i,j) = 2*p(i,j) - p.prev(i,j) + CFL^2 * ( p(i+1,j) + p(i,j+1) - 4*p(i,j) + p(i-1,j) + p(i,j-1));
151     end
152 end
153
154 k = abs(p);
155 % Rigid Boundaries
156 k(1:end, [1 end]) = -1; % Upper and lower boundaries
157 %p([1 end], 1:end) = 0; % Left and right boundaries
158 k(l:w:l, w:dx:end-w-2*dx) = -1; % Middle region
159 k(w:l:d1:end, [l:w w+d:dx:end]) = -1; % Right bottom and top regions
160 k(w:w+l, [l+dx:w3+dx w-w3-dx:w]) = -1; K(w:w+l, [end-w3:end-dx end-w-dx:end-w+w3-dx]) = -1; % 1/3
161
162 k([w:w+w4 w+w4+12:w2+w4+12 w+2*w4+2*12:w+3*w4+3*12:w+4*w4+4*12:w+5*w4+4*12], [w3+2*dx:w3+dx+w2.upper ...
163         -w-3-1*dx-w2.upper:w-w3-2*dx]) = -1; % 2/3
164 k([w:w+w4 w+w4+12:w2+w4+12 w+2*w4+2*12:w+3*w4+2*12 w+3*w4+3*12:w+4*w4+4*12:w+5*w4+4*12], [end-w3-w2.lower:end-w3-dx ...
165         -end-w+w3+0*dx:end-w+w3+w2.lower+1*dx]) = -1; % 2/3
166 k([w:w+w4+1:d3 w+w4+12-1d3:w+2*w4+12+1d3 w+2*w4+2*12-1d3:w+3*w4+2*12+1d3 w+3*w4+3*12-1d3:w+4*w4+3*12+1d3 w+4*w4+4*12-1d3:w+5*w4+4*12], ...
167         [2*dx+w2.upper:w3:w3+2*dx+w2.upper+w-3-w2.upper-w3-2*dx:w-w2.upper-w3-2*dx]) = -1; % 3/3
168 k([w:w+w4+1:d3 w+w4+12-1d3:w+2*w4+12+1d3 w+2*w4+2*12-1d3:w+3*w4+2*12+1d3 w+3*w4+3*12-1d3:w+4*w4+3*12+1d3 w+4*w4+4*12-1d3:w+5*w4+4*12], ...
169         [end-w3-w2.lower-w3-1*dx:end-w2.lower-w3-1*dx end-w+2*dx+w2.lower+w3:end-w+w3+2*dx+w2.lower+w3]) = -1; % 3/3
170
171 if round(mod(t/dt,25)) == 1
172     output = mean(mean(k(end-d:2:end,w:dx:w+d))); %max(max(k(w+l:d1:end,w:dx:w+d)));
173     output.pres = [output.pres output];
174     time = [time t];
175 end
176 if round(mod(t/dt,25)) == 1
177     % Visualize
178     % COLORBAR
179     indexValue = 0.4;      % value for which to set a particular color
180     topColor = [1 0 0];    % color for maximum data value (red = [1 0 0])

```

```

172      indexColor = [1 1 1];          % color for indexed data value (white = [1 1 1])
173      bottomColor = [0 0 1];        % color for minimum data value (blue = [0 0 1])
174      largest = max(max(k));
175      smallest = min(min(k));
176      L = size(p,1);
177      index = L*abs(indexValue-smallest)/(largest-smallest);
178      customCMap1 = [linspace(bottomColor(1),indexColor(1),100*index)',...
179                      linspace(bottomColor(2),indexColor(2),100*index)',...
180                      linspace(bottomColor(3),indexColor(3),100*index)'];
181      customCMap2 = [linspace(indexColor(1),topColor(1),100*(L-index))',...
182                      linspace(indexColor(2),topColor(2),100*(L-index))',...
183                      linspace(indexColor(3),topColor(3),100*(L-index))'];
184      customCMap = [customCMap1;customCMap2]; % Combine colormaps
185      colormap(customCMap)
186
187      imagesc(x,y,k'); colorbar; caxis([-1 max(max(abs(k)))]); % caxis([-1 1]);
188      colorbar; axis equal;
189      grid off; box off; axis off
190      ax = gca;
191      ax.YDir = 'normal';
192      tstring = num2str(t); pstring = num2str(output); astring = num2str(mean(nonzeros(output_pres)));
193      txt = " " + tstring + "; P.out" = " " + pstring + "; P.avg" = " " + astring;
194      title(txt); %title(sprintf('t = %.6f',t));
195      drawnow limitrate
196      frame = getframe(f);
197      im = frame2im(frame);
198      [imind,cm] = rgb2ind(im,256);
199      if t == dt
200          imwrite(imind,cm,filename,'gif','DelayTime',0.01, 'Loopcount',inf);
201      else
202          imwrite(imind,cm,filename,'gif','DelayTime',0.01, 'WriteMode','append');
203      end
204  end
205 end
206 figure(2); plot(time,output.pres); xlabel("Time"); ylabel("Output Pressure"); saveas(gcf,filename2)
207 out = output.pres;
208 close all
209 end
210
211 function out = solver10_OR(lambda, filename, freq, filename2)
212
213     % Geometric constraints
214     l = lambda / 2;
215     w = lambda / 10;
216     d = 0.1 * lambda;
217     d1 = 0.055 * lambda;
218
219     l1 = (0.03 * 1);
220     l2 = (0.23 * 1);
221     w1_upper = (0.69 * w); % changing
222     w1_lower = (0.69 * w); % changing
223     w3 = (0.05 * w);
224     w2_upper = ((w - w1_upper - 4 * w3) / 2);
225     w2_lower = ((w - w1_lower - 4 * w3) / 2);
226     w4 = ((l-4*l2)/5);
227     ld2 = ((l-4*l1)/8);
228     ld3 = ((l2-l1)/2);
229
230     Lx = w + l + d1 + d;
231     Ly = 2*w + d;
232     dx = 1; dy = 1;
233     nx = fix(Lx/dx) + 2; ny = fix(Ly/dy) + 2;
234
235     x = linspace(0, Lx, nx); y = linspace(0, Ly, ny);
236
237     % Field definition
238
239     p = 0*ones(nx,ny);
240     p.prev = p; p.next = p;
241
242     CFL = 0.7; % c * dt/dx
243     c = 1; %c/lambda; freq1 = c/lambda; freq2 = c/lambda;
244     dt = CFL*dx/c;
245     output.pres = {};
246     time = {};
247
248     % Initial conditions
249     T = 20000*dt; % Runtime
250
251     t = 0;
252     f = figure(1);
253     set(gcf, 'Position', [100 100 1400 500]);
254
255     while t < T
256         % Absorbing Boundaries
257         p.next(1,:) = p(2,:); + ((CFL-1)/(CFL+1))*(p.next(2,:)-p(1,:));
258         p.next(end,:) = p(end-1,:); + ((CFL-1)/(CFL+1))*(p.next(end-1,:)-p(end,:));
259
260         % Rigid Boundaries
261         p(1:end, [1 end]) = 0; % Upper and lower boundaries
262         %p([1 end], 1:end) = 0; % Left and right boundaries
263         p(l:w1, wdx:end-w-2*dx) = 0; % Middle region
264         p(w1:ld1:end, [1:w w+d1:d1:end]) = 0; % Right bottom and top regions
265         p(w:w+l, [1:dx:w3+dx:w-w3-dx:w]) = 0; p(w:w+l, [end-w3:end-dx end-w-dx:end-w+w3-dx]) = 0; % 1/3
266
267         p([w:w+w4 w+w4+12:w+2*w4+12 w+2*w4+2*w4+12:w+3*w4+2*w4+12 w+3*w4+3*w4+3*w4+12 w+4*w4+4*w4+12:w+5*w4+4*w4+12], [w3+2*dx:w3+dx+w2.upper ...
268             -w3-1*dx-w2.upper:w-w3-2*dx]) = 0; % 2/3
269         p([w:w+w4 w+w4+12:w+2*w4+12 w+2*w4+2*w4+12:w+3*w4+3*w4+12 w+4*w4+3*w4+12 w+4*w4+4*w4+12:w+5*w4+4*w4+12], [end-w3-w2.lower:end-w3-dx ...
270             -end-w3+3*0*dx:end-w3+w2.lower+1*dx]) = 0; % 2/3
271         p([w:w+w4+1:ld1 w+w4+12-ld3:w+2*w4+12+ld3 w+2*w4+2*w4+12-ld3:w+3*w4+2*w4+12+ld3 w+3*w4+3*w4+12-ld3:w+4*w4+3*w4+12+ld3 w+4*w4+4*w4+12], ...
272             [2*dx+w2.upper:w3:w3+2*dx+w2.upper+w3 w-w3-w2.upper-w3-2*dx:w-w2.upper-w3-2*dx]) = 0; % 3/3
273         p([w:w+w4+1:ld3 w+w4+12-ld3:w+2*w4+12+ld3 w+2*w4+2*w4+12-ld3:w+3*w4+3*w4+12-ld3:w+4*w4+3*w4+12+ld3 w+4*w4+4*w4+12], ...
274             [end-w3-w2.lower-w3-1:dx:end-w2.lower-w3-1:dx end-w+2*dx+w2.lower+w3:end-w+w3+2*dx+w2.lower+w3]) = 0; % 3/3
275
276         %p.next(1:end,1) = p(1:end,2) + ((CFL-1)/(CFL+1))*(p.next(1:end,2)-p(1:end,1));
277         p.next(1:end, end) = p(1:end, end-1) + ((CFL-1)/(CFL+1))*(p.next(1:end, end-1)-p(1:end, end));
278
279     % Source 1

```

```

279 p(2, end-lambda/10:end-1) = 1*sin(2*pi*t*freq);
280
281 % Source 2
282 %p(2, 2:lambda/10-1) = 1*sin(2*pi*t*freq);
283
284 % Rigid Boundaries
285 p(l:end, [l end]) = 0; % Upper and lower boundaries
286 %p([l end], l:end) = 0; % Left and right boundaries
287 p(l:w+l, wdx:end-w-2*dx) = 0; % Middle region
288 p(w+l:d:end, [l:w w+d:dx:end]) = 0; % Right bottom and top regions
289 p(w:w+l, [l+dx:w+3*dx w-w3-dx:w]) = 0; p(w:w+l, [end-w3:end-dx end-w-dx:end-w+w3-dx]) = 0; % 1/3
290
291 p([w:w+w4 w+w4+12:w+2*w4+2*w12:w+3*w4+2*w12 w+3*w4+3*w12 w+4*w4+4*w12:w+5*w4+4*w12], [w3+2*dx:w3+dx+w2_upper ...
292     -w-3-1*dx-w2_upper:w-w3-2*dx]) = 0; % 2/3
293 p([w:w+w4 w+w4+12:w+2*w4+2*w12:w+3*w4+2*w12 w+3*w4+3*w12 w+4*w4+4*w12:w+5*w4+4*w12], [end-w3-w2_lower:end-w3-dx ...
294     -end-w+w3+0*dx:end-w+w3+w2_lower+1*dx]) = 0; % 2/3
295 p([w:w+w4+1+dx:w+w4+12+1d3:w+2*w4+2*w12+1d3 w+2*w4+2*w12+1d3:w+3*w4+2*w12+1d3 w+3*w4+3*w12-1d3:w+4*w4+3*w12+1d3:w+5*w4+4*w12], ...
296     [2*dx+w2_upper:w3:w+2*dx+w2_upper+w3 w-w3-w2_upper:w-3-2*dx:w-w2_upper:w-3-2*dx]) = 0; % 3/3
297 p([w:w+w4+1d3 w+w4+12-1d3:w+2*w4+2*w12+1d3 w+2*w4+2*w12+1d3:w+3*w4+3*w12-1d3:w+4*w4+3*w12+1d3:w+5*w4+4*w12], ...
298     [end-w3-w2_lower:w3-1*dx:end-w2_lower-w3-1*dx end-w+2*dx+w2_lower+w3:end-w+w3+2*dx+w2_lower+w3]) = 0; % 3/3
299
300 % Finite difference solution
301 for i = 2:nx-1
302     for j = 2:ny-1
303         p.next(i,j) = 2*p(i,j) - p.prev(i,j) + CFL^2 * ( p(i+1,j) + p(i,j+1) - 4*p(i,j) + p(i-1,j) + p(i,j-1));
304     end
305
306 k = abs(p);
307 % Rigid Boundaries
308 k(l:end, [l end]) = -1; % Upper and lower boundaries
309 %p([l end], l:end) = 0; % Left and right boundaries
310 k(l:w+l, wdx:end-w-2*dx) = -1; % Middle region
311 k(w+l:d:end, [l:w w+d:dx:end]) = -1; % Right bottom and top regions
312 k(w:w+l, [l+dx:w+3*dx w-w3-dx:w]) = -1; % 1/3
313 k([w:w+w4+12:w+2*w4+2*w12:w+3*w4+2*w12 w+4*w4+4*w12:w+5*w4+4*w12], [w3+2*dx:w3+dx+w2_upper ...
314     -w-3-1*dx-w2_upper:w-w3-2*dx]) = -1; % 2/3
315 k([w:w+w4 w+w4+12:w+2*w4+2*w12:w+3*w4+2*w12 w+3*w4+3*w12 w+4*w4+4*w12:w+5*w4+4*w12], [end-w3-w2_lower:end-w3-dx ...
316     -end-w+w3+0*dx:end-w+w3+w2_lower+1*dx]) = -1; % 2/3
317 k([w:w+w4+1d3 w+w4+12-1d3:w+2*w4+2*w12+1d3 w+2*w4+2*w12+1d3:w+3*w4+2*w12+1d3 w+3*w4+3*w12-1d3:w+4*w4+3*w12+1d3:w+5*w4+4*w12], ...
318     [2*dx+w2_upper:w3:w+2*dx+w2_upper+w3 w-w3-w2_upper:w-3-2*dx:w-w2_upper:w-3-2*dx]) = -1; % 3/3
319 k([w:w+w4+1d3 w+w4+12-1d3:w+2*w4+2*w12+1d3 w+2*w4+2*w12+1d3:w+3*w4+3*w12-1d3:w+4*w4+3*w12+1d3:w+5*w4+4*w12], ...
320     [end-w3-w2_lower:w3-1*dx:end-w2_lower-w3-1*dx end-w+2*dx+w2_lower+w3:end-w+w3+2*dx+w2_lower+w3]) = -1; % 3/3
321
322 if round(mod(t/dt,25)) == 1
323     output = mean(mean(k(end-d/2:end,w:dx:w+d))); %max(max(k(w+l:d:end,w:dx:w+d)));
324     output.pres = [output.pres output];
325     time = [time t];
326 end
327 if round(mod(t/dt,25)) == 1
328     % Visualize
329     % COLORBAR
330     indexValue = 0.4; % value for which to set a particular color
331     topColor = [1 0 0]; % color for maximum data value (red = [1 0 0])
332     indexColor = [1 1 1]; % color for indexed data value (white = [1 1 1])
333     bottomColor = [0 0 1]; % color for minimum data value (blue = [0 0 1])
334     largest = max(max(k));
335     smallest = min(min(k));
336     L = size(p,1);
337     index = L*abs(indexValue-smallest)/(largest-smallest);
338     customCMap1 = [linspace(bottomColor(1),indexColor(1),100*index)',...
339                     linspace(bottomColor(2),indexColor(2),100*index)',...
340                     linspace(bottomColor(3),indexColor(3),100*index)';
341     customCMap2 = [linspace(indexColor(1),topColor(1),100*(L-index)',...
342                     linspace(indexColor(2),topColor(2),100*(L-index)'),...
343                     linspace(indexColor(3),topColor(3),100*(L-index))'];
344     customCMap = [customCMap1;customCMap2]; % Combine colormap
345     colormap(customCMap)
346
347     imagesc(x,y,k'); colorbar; caxis([-1 max(max(abs(k)))]); % caxis([-1 1]);
348     colorbar; axis equal;
349     grid off; box off; axis off
350     ax = gca;
351     ax.Ydir = 'normal';
352     tstring = num2str(t); pstring = num2str(output); astring = num2str(mean(nonzeros(output.pres)));
353     txt = "t = " + tstring + "; P.out = " + pstring + "; P.avg = " + astring;
354     title(txt); %title(sprintf('t = %.6f',t));
355     drawnow limitrate
356     frame = getframe(f);
357     im = frame2im(frame);
358     [imind,cm] = rgb2ind(im,256);
359     if t == dt
360         imwrite(imind,cm,filename,'gif','DelayTime',0.01, 'Loopcount',inf);
361     else
362         imwrite(imind,cm,filename,'gif','DelayTime',0.01, 'WriteMode','append');
363     end
364 end
365 figure(2); plot(time,output_pres); xlabel("Time"); ylabel("Output Pressure"); saveas(gcf,filename2)
366 out = output_pres;
367 close all
368
369 function out = solver01_OR(lambda, filename, freq, filename2)
370
371 % Geometric constraints
372 l = lambda / 2;
373 w = lambda / 10;
374 d = 0.1 * lambda;
375 d1 = 0.055 * lambda;
376
377 l1 = (0.03 * l);
378 l2 = (0.23 * l);
379 w1_upper = (0.69 * w); % changing
380 w1_lower = (0.69 * w); % changing
381 w3 = (0.05 * w);
382 w2_upper = ((w - wl_upper - 4 * w3) / 2);
383 w2_lower = ((w - wl_lower - 4 * w3) / 2);
384 w4 = ((l-4*l)/5);
385 ld2 = ((l-4*l1)/8);

```

```

382 ld3 = ((12-11)/2);
383
384 Lx = w + 1 + dl + d;
385 Ly = 2*w + d;
386 dx = 1; dy = 1;
387 nx = fix(Lx/dx) + 2; ny = fix(Ly/dy) + 2;
388
389 x = linspace(0, Lx, nx); y = linspace(0, Ly, ny);
390
391 % Field definition
392
393 p = 0*ones(nx,ny);
394 p.prev = p; p.next = p;
395
396 CFL = 0.7; % c * dt/dx
397 c = 1; %c/lambda; freq1 = c/lambda; freq2 = c/lambda;
398 dt = CFL*dx/ct;
399 output.pres = [];
400 time = [];
401
402 % Initial conditions
403 T = 20000*dt; % Runtime
404
405 t = 0;
406 f = figure(1);
407 set(gcf,'Position',[100 100 1400 500]);
408
409 while t<T
410     % Absorbing Boundaries
411     p.next(:,1) = p(2,:); + ((CFL-1)/(CFL+1))*(p.next(2,:)-p(1,:));
412     p.next(:,end) = p(end-1,:); + ((CFL-1)/(CFL+1))*(p.next(end-1,:)-p(end,:));
413
414     % Rigid Boundaries
415     p(:,end, [1 end]) = 0; % Upper and lower boundaries
416     %p([1 end], 1:end) = 0; % Left and right boundaries
417     p(:,w1, wdx:end-w-2*dx) = 0; % Middle region
418     p(w1:d1:end, [1:w w+d+dx:end]) = 0; % Right bottom and top regions
419     p(w1:w1, [1+dx+w3+dx w-w3-dx:w1]) = 0; p(w1:w1, [end-w3:end-dx end-w-dx:end-w+w3-dx]) = 0; % 1/3
420
421 p([w:w+w4 w+w4+12:w+2*w4+12 w+2*w4+2*12 w+3*w4+3*12:w+4*w4+4*12:w+5*w4+4*12], [w3+2*dx:w3+dx+w2.upper ...
    w-w3-1*dx-w2.upper:w-w3-2*dx]) = 0; % 2/3
422 p([w:w+w4 w+w4+12:w+2*w4+12 w+2*w4+2*12 w+3*w4+2*12 w+3*w4+3*12:w+4*w4+3*12 w+4*w4+4*12:w+5*w4+4*12], [end-w3-w2.lower:end-w3-dx ...
    end-w+w3+0*dx:end-w+w3+w2.lower+1*dx]) = 0; % 2/3
423 p([w:w+w4+1d3 w+w4+12-1d3:w+2*w4+12+1d3 w+2*w4+2*12-1d3:w+3*w4+2*12+1d3 w+3*w4+3*12-1d3:w+4*w4+3*12+1d3 w+4*w4+4*12-1d3:w+5*w4+4*12], ...
    [2*dx+w2.upper+w3:w3+2*dx+w2.upper+w3 w-w3-w2.upper-w3-2*dx:w-w2.upper-w3-2*dx]) = 0; % 3/3
424 p([w:w+w4+1d3 w+w4+12-1d3:w+2*w4+12+1d3 w+2*w4+2*12-1d3:w+3*w4+2*12+1d3 w+3*w4+3*12-1d3:w+4*w4+3*12+1d3 w+4*w4+4*12-1d3:w+5*w4+4*12], ...
    [end-w3-w2.lower-w3-1*dx:end-w2.lower-w3-1*dx end-w+2*dx+w2.lower+w3:end-w+w3+2*dx+w2.lower+w3]) = 0; % 3/3
425
426 t = t+dt;
427 p.prev = p; p = p.next;
428
429 %p.next(1:end,1) = p(1:end,2) + ((CFL-1)/(CFL+1))*(p.next(1:end,2)-p(1:end,1));
430 p.next(1:end,end) = p(1:end,end-1) + ((CFL-1)/(CFL+1))*(p.next(1:end,end-1)-p(1:end,end));
431
432 % Source 1
433 %p(2, end-lambda/10:end-1) = 1*sin(2*pi*t*freq);
434
435 % Source 2
436 p(2, 2:lambda/10-1) = 1*sin(2*pi*t*freq);
437
438 % Rigid Boundaries
439 p(:,end, [1 end]) = 0; % Upper and lower boundaries
440 %p([1 end], 1:end) = 0; % Left and right boundaries
441 p(:,w1, wdx:end-w-2*dx) = 0; % Middle region
442 p(w1:d1:end, [1:w w+d+dx:end]) = 0; % Right bottom and top regions
443 p(w1:w1, [1+dx+w3+dx w-w3-dx:w1]) = 0; p(w1:w1, [end-w3:end-dx end-w-dx:end-w+w3-dx]) = 0; % 1/3
444
445 p([w:w+w4 w+w4+12:w+2*w4+12 w+2*w4+2*12 w+3*w4+3*12:w+4*w4+4*12:w+5*w4+4*12], [w3+2*dx:w3+dx+w2.upper ...
    w-w3-1*dx-w2.upper:w-w3-2*dx]) = 0; % 2/3
446 p([w:w+w4 w+w4+12:w+2*w4+12 w+2*w4+2*12 w+3*w4+2*12 w+3*w4+3*12:w+4*w4+3*12 w+4*w4+4*12:w+5*w4+4*12], [end-w3-w2.lower:end-w3-dx ...
    end-w+w3+0*dx:end-w+w3+w2.lower+1*dx]) = 0; % 2/3
447 p([w:w+w4+1d3 w+w4+12-1d3:w+2*w4+12+1d3 w+2*w4+2*12-1d3:w+3*w4+2*12+1d3 w+3*w4+3*12-1d3:w+4*w4+3*12+1d3 w+4*w4+4*12-1d3:w+5*w4+4*12], ...
    [2*dx+w2.upper+w3:w3+2*dx+w2.upper+w3 w-w3-w2.upper-w3-2*dx:w-w2.upper-w3-2*dx]) = 0; % 3/3
448 p([w:w+w4+1d3 w+w4+12-1d3:w+2*w4+12+1d3 w+2*w4+2*12-1d3:w+3*w4+2*12+1d3 w+3*w4+3*12-1d3:w+4*w4+3*12+1d3 w+4*w4+4*12-1d3:w+5*w4+4*12], ...
    [end-w3-w2.lower-w3-1*dx:end-w2.lower-w3-1*dx end-w+2*dx+w2.lower+w3:end-w+w3+2*dx+w2.lower+w3]) = 0; % 3/3
449
450 % Finite difference solution
451 for i = 2:nx-1
452     for j = 2:ny-1
453         p.next(i,j) = 2*p(i,j) - p.prev(i,j) + CFL^2 * ( p(i+1,j) + p(i,j+1) - 4*p(i,j) + p(i-1,j) + p(i,j-1));
454     end
455 end
456
457 k = abs(p);
458
459 % Rigid Boundaries
460 k(:,end, [1 end]) = -1; % Upper and lower boundaries
461 %p([1 end], 1:end) = 0; % Left and right boundaries
462 k(:,w1, wdx:end-w-2*dx) = -1; % Middle region
463 k(w1:d1:end, [1:w w+d+dx:end]) = -1; % Right bottom and top regions
464 k(w1:w1, [1+dx+w3+dx w-w3-dx:w1]) = -1; k(w1:w1, [end-w3:end-dx end-w-dx:end-w+w3-dx]) = -1; % 1/3
465
466 k([w:w+w4 w+w4+12:w+2*w4+12 w+2*w4+2*12 w+3*w4+3*12:w+4*w4+4*12:w+5*w4+4*12], [w3+2*dx:w3+dx+w2.upper ...
    w-w3-1*dx-w2.upper:w-w3-2*dx]) = -1; % 2/3
467 k([w:w+w4 w+w4+12:w+2*w4+12 w+2*w4+2*12 w+3*w4+2*12 w+3*w4+3*12:w+4*w4+3*12 w+4*w4+4*12:w+5*w4+4*12], [end-w3-w2.lower:end-w3-dx ...
    end-w+w3+0*dx:end-w+w3+w2.lower+1*dx]) = -1; % 2/3
468 k([w:w+w4+1d3 w+w4+12-1d3:w+2*w4+12+1d3 w+2*w4+2*12-1d3:w+3*w4+2*12+1d3 w+3*w4+3*12-1d3:w+4*w4+3*12+1d3 w+4*w4+4*12-1d3:w+5*w4+4*12], ...
    [2*dx+w2.upper+w3:w3+2*dx+w2.upper+w3 w-w3-w2.upper-w3-2*dx:w-w2.upper-w3-2*dx]) = -1; % 3/3
469 k([w:w+w4+1d3 w+w4+12-1d3:w+2*w4+12+1d3 w+2*w4+2*12-1d3:w+3*w4+2*12+1d3 w+3*w4+3*12-1d3:w+4*w4+3*12+1d3 w+4*w4+4*12-1d3:w+5*w4+4*12], ...
    [end-w3-w2.lower-w3-1*dx:end-w2.lower-w3-1*dx end-w+2*dx+w2.lower+w3:end-w+w3+2*dx+w2.lower+w3]) = -1; % 3/3
470
471 if round(mod(t/dt,25)) == 1
472     output = mean(mean(k(end-d/2:end,w+dx:w+d))); %max(max(k(w1:d1:end,w+dx:w+d)));
473     output.pres = [output.pres output];
474     time = [time t];
475 end
476 if round(mod(t/dt,25)) == 1
477     % Visualize
478     % COLORBAR
479     indexValue = 0.4; % value for which to set a particular color
480     topColor = [1 0 0]; % color for maximum data value (red = [1 0 0])
481     indexColor = [1 1 1]; % color for indexed data value (white = [1 1 1])

```

```

481      bottomcolor = [0 0 1];      % color for minimum data value (blue = [0 0 1])
482      largest = max(max(k));
483      smallest = min(min(k));
484      L = size(p,1);
485      index = L*abs(indexValue-smallest)/(largest-smallest);
486      customCMap1 = [linspace(bottomcolor(1),indexColor(1),100*index)',...
487                      linspace(bottomcolor(2),indexColor(2),100*index)',...
488                      linspace(bottomcolor(3),indexColor(3),100*index)'];
489      customCMap2 = [linspace(indexColor(1),topColor(1),100*(L-index))',...
490                      linspace(indexColor(2),topColor(2),100*(L-index))',...
491                      linspace(indexColor(3),topColor(3),100*(L-index))'];
492      customCMap = [customCMap1;customCMap2]; % Combine colormaps
493      colormap(customCMap)
494
495      imagesc(x,y,k'); colorbar; caxis([-1 max(max(abs(k)))]); % caxis([-1 1]);
496      colorbar; axis equal;
497      grid off; box off; axis off
498      ax = gca;
499      ax.VDir = 'normal';
500      tstring = num2str(t); pstring = num2str(output); astring = num2str(mean(nonzeros(output_pres)));
501      txt = "t = " + tstring + "; P.out = " + pstring + "; P.avg = " + astring;
502      title(txt); %title(sprintf('t = %.6f',t));
503      drawnow limitrate
504      frame = getframe(f);
505      im = frame2im(frame);
506      [imind,cm] = rgb2ind(im,256);
507      if t == dt
508          imwrite(imind,cm,filename,'gif','DelayTime',0.01, 'Loopcount',inf);
509      else
510          imwrite(imind,cm,filename,'gif','DelayTime',0.01, 'WriteMode','append');
511      end
512  end
513
514  figure(2); plot(time,output_pres); xlabel("Time"); ylabel("Output Pressure"); saveas(gcf,filename2)
515  out = output_pres;
516  close all
517 end
518
519 function out = solver11_XOR(lambda, filename, freq, filename2)
520
521     % Geometric constraints
522     l = lambda / 2;
523     w = lambda / 10;
524     d = 0.1 * lambda;
525     d1 = 0.055 * lambda;
526
527     l1 = (0.03 * 1);
528     l2 = (0.23 * 1);
529     w1_upper = (0.69 * w); % changing
530     w1.lower = (0.22 * w); % changing
531     w3 = (0.05 * w);
532     w2_upper = ((w - w1.upper - 4 * w3) / 2);
533     w2.lower = ((w - w1.lower - 4 * w3) / 2);
534     w4 = ((l-4*l2)/5);
535     l1d2 = ((l-4*l1)/8);
536     l1d3 = ((l12-l1)/2);
537
538     Lx = w + l + d1 + d;
539     Ly = 2*w + d;
540     dx = 1; dy = 1;
541     nx = fix(Lx/dx) + 2; ny = fix(Ly/dy) + 2;
542
543     x = linspace(0, Lx, nx); y = linspace(0, Ly, ny);
544
545     % Field definition
546
547     p = 0*ones(nx,ny);
548     p.prev = p; p.next = p;
549
550     CFL = 0.7; % c * dt/dx
551     c = 1; %c/lambda; freq1 = c/lambda; freq2 = c/lambda;
552     dt = CFL*dx/c;
553     output_pres = [];
554     time = [];
555
556     % Initial conditions
557     T = 20000*dt; % Runtime
558
559     t = 0;
560     f = figure(1);
561     set(gcf,'Position',[100 100 1400 500]);
562
563     while t<T
564
565         % Absorbing Boundaries
566         p.next(1,:) = p(2,:); + ((CFL-1)/(CFL+1))*(p.next(2,:)-p(1,:));
567         p.next(end,:) = p(end-1,:); + ((CFL-1)/(CFL+1))*(p.next(end-1,:)-p(end,:));
568
569         % Rigid Boundaries
570         p(:,end, [1 end]) = 0; % Upper and lower boundaries
571         %p([1 end, 1:end]) = 0; % Left and right boundaries
572         p(1:w1, wdx:end-w-2*dx) = 0; % Middle region
573         p(w1:d1:end, [1:w w+dx:w+end]) = 0; % Right bottom and top regions
574         p(w1:w+1, [1:dx:w3:dx w-w3-dx:w]) = 0; % 1/3
575         p([w:w+4 w+w4+1:2:w+2*w4+12 w+2*w4+2*12:w+3*w4+3*12:w+4*w4+3*12 w+4*w4+4*12], [w3+2*dx:w3+dx+w2_upper ...
576             w-w3-1*dx-w2_upper:w-w3-2*dx]) = 0; % 2/3
577         p([w:w+4 w+w4+1:2:w+2*w4+12 w+2*w4+2*12:w+3*w4+2*12:w+3*w4+3*12:w+4*w4+3*12 w+4*w4+4*12], [end-w3-w2_lower:end-w3-dx ...
578             end-w+w3+3+dx:end-w+w3+w2_lower+1*dx]) = 0; % 2/3
579         p([w:w+4+l3 w+w4+12-l3:w+2*w4+12+l3 w+2*w4+2*12+l3:w+3*w4+3*12+l3:w+4*w4+3*12+l3:w+5*w4+4*12], ...
580             [2*dx+w2_upper:w3:w3+2*dx+w2_upper:w3 w-w3-w2_upper-w3-2*dx:w-w2_upper-w3-2*dx]) = 0; % 3/3
581         p([w:w+4+l3 w+w4+12-l3:w+2*w4+12+l3 w+2*w4+2*12+l3:w+3*w4+3*12+l3:w+4*w4+3*12+l3:w+5*w4+4*12], ...
582             [end-w3-w2_lower-w3-1*dx:end-w2_lower-w3-1*dx end-w+2*dx+w2_lower:w3:end-w+w3+2*dx+w2_lower+w3]) = 0; % 3/3
583
584         t = t+dt;
585         p.prev = p; p = p.next;
586
587         % Source 1
588         p(2, end-lambda/10:end-1) = 1*sin(2*pi*t*freq);

```

```

588 % Source 2
589 p(2, 2:lambda/10-1) = 1*sin(2*pi*t*freq);
590
591 % Rigid Boundaries
592 p(1:end, [1 end]) = 0; % Upper and lower boundaries
593 %p([1 end], 1:end) = 0; % Left and right boundaries
594 p(1:w1, wdx:end-w-2*dx) = 0; % Middle region
595 p(w1:d1:end, [1:w w+d+dx:end]) = 0; % Right bottom and top regions
596 p(w1:w1, [1+dx:w3+dx w-w3-dx:w]) = 0; p(w1:w1, [end-w3:end-dx end-w-dx:end-w+w3-dx]) = 0; % 1/3
597
598 p([w:w+w4 w+w4+12:w+2*w4+12 w+2*w4+2*12 w+3*w4+3*12:w+4*w4+3*12 w+4*w4+4*12:w+5*w4+4*12], [w3+2*dx:w3+dx+w2.upper ...
599 w-w3-1*dx-w2.upper:w-w3-2*dx]) = 0; % 2/3
600 p([w:w+w4 w+w4+12:w+2*w4+12 w+2*w4+2*12:w+3*w4+3*12:w+4*w4+3*12 w+4*w4+4*12:w+5*w4+4*12], [end-w3-w2.lower:end-w3-dx ...
601 end-w3+w3+0*dx:end-w3+w2.lower+1*dx]) = 0; % 2/3
602 p([w:w+w4+1:d1 w+w4+12:w+2*w4+12+w1:d3 w+2*w4+2*12-1:d3:w+3*w4+2*12+1:d3 w+3*w4+3*12-1:d3:w+4*w4+3*12+1:d3 w+4*w4+4*12-1:d3:w+5*w4+4*12], ...
603 [2*dx+w2.upper:w3:w3+2*dx+w2.upper+w3 w-w3-w2.upper-w3-2*dx:w-w2.upper-w3-2*dx]) = 0; % 3/3
604 p([w:w+w4+d3 w+w4+12-1:d3:w+2*w4+12+w1:d3 w+2*w4+2*12-1:d3:w+3*w4+2*12+1:d3 w+3*w4+3*12-1:d3:w+4*w4+3*12+1:d3 w+4*w4+4*12-1:d3:w+5*w4+4*12], ...
605 [end-w3-w2.lower-w3-1:dx:end-w2.lower-w3-1:dx end-w+2*dx+w2.lower+w3:end-w+w3+2*dx+w2.lower+w3]) = 0; % 3/3
606
607 % Finite difference solution
608 for i = 2:nx-1
609     for j = 2:ny-1
610         p.next(i,j) = 2*p(i,j) - p.prev(i,j) + CFL^2 * ( p(i+1,j) + p(i,j+1) - 4*p(i,j) + p(i-1,j) + p(i,j-1));
611     end
612 end
613 k = abs(p);
614 % Rigid Boundaries
615 k(1:end, [1 end]) = -1; % Upper and lower boundaries
616 %p([1 end], 1:end) = 0; % Left and right boundaries
617 k(1:w1, wdx:end-w-2*dx) = -1; % Middle region
618 k(w1:d1:end, [1:w w+d+dx:end]) = -1; % Right bottom and top regions
619 k(w1:w1, [1+dx:w3+dx w-w3-dx:w]) = -1; k(w1:w1, [end-w3:end-dx end-w-dx:end-w+w3-dx]) = -1; % 1/3
620
621 k([w:w+w4 w+w4+12:w+2*w4+12 w+2*w4+2*12:w+3*w4+3*12:w+4*w4+3*12 w+4*w4+4*12:w+5*w4+4*12], [w3+2*dx:w3+dx+w2.upper ...
622 w-w3-1*dx-w2.upper:w-w3-2*dx]) = -1; % 2/3
623 k([w:w+w4 w+w4+12:w+2*w4+12 w+2*w4+2*12:w+3*w4+3*12:w+4*w4+3*12 w+4*w4+4*12:w+5*w4+4*12], [end-w3-w2.lower:end-w3-dx ...
624 end-w3+w3+0*dx:end-w3+w2.lower+1*dx]) = -1; % 2/3
625 k([w:w+w4+1:d3 w+w4+12-1:d3:w+2*w4+2*12+1:d3 w+2*w4+2*12-1:d3:w+3*w4+2*12+1:d3 w+3*w4+3*12-1:d3:w+4*w4+3*12+1:d3 w+4*w4+4*12-1:d3:w+5*w4+4*12], ...
626 [2*dx+w2.upper:w3:w3+2*dx+w2.upper+w3 w-w3-w2.upper-w3-2*dx:w-w2.upper-w3-2*dx]) = -1; % 3/3
627 k([w:w+w4+d3 w+w4+12-1:d3:w+2*w4+12+w1:d3 w+2*w4+2*12-1:d3:w+3*w4+2*12+1:d3 w+3*w4+3*12-1:d3:w+4*w4+3*12+1:d3 w+4*w4+4*12-1:d3:w+5*w4+4*12], ...
628 [end-w3-w2.lower-w3-1:dx:end-w2.lower-w3-1:dx end-w+2*dx+w2.lower+w3:end-w+w3+2*dx+w2.lower+w3]) = -1; % 3/3
629
630 if round(mod(t/dt,25)) == 1
631     output = mean(mean(k(end-d/2:end,w+dx:w+d))); %max(max(k(w1+d1:end,w+dx:w+d)));
632     output.pres = [output.pres output];
633     time = [time t];
634 end
635 if round(mod(t/dt,25)) == 1
636     % Visualize
637     % COLORBAR
638     indexValue = 0.4; % value for which to set a particular color
639     topColor = [1 0 0]; % color for maximum data value (red = [1 0 0])
640     indexColor = [1 1 1]; % color for indexed data value (white = [1 1 1])
641     bottomColor = [0 0 1]; % color for minimum data value (blue = [0 0 1])
642     largest = max(max(k));
643     smallest = min(min(k));
644     L = size(p,1);
645     index = L*abs(indexValue-smallest)/(largest-smallest);
646     customCMap1 = [linspace(bottomColor(1),indexColor(1),100*index)',...
647                     linspace(bottomColor(2),indexColor(2),100*index)',...
648                     linspace(bottomColor(3),indexColor(3),100*index)'];
649     customCMap2 = [linspace(indexColor(1),topColor(1),100*(L-index))',...
650                     linspace(indexColor(2),topColor(2),100*(L-index))',...
651                     linspace(indexColor(3),topColor(3),100*(L-index))'];
652     customCMap = [customCMap1;customCMap2]; % Combine colormaps
653     colormap(customCMap)
654
655     imagesc(x,y,k'); colorbar; caxis([-1 max(max(abs(k)))]); % caxis([-1 1]);
656     colorbar; axis equal;
657     grid off; box off; axis off
658     ax = gca;
659     ax.YDir = 'normal';
660     tstring = num2str(t); pstring = num2str(output); astring = num2str(mean(nonzeros(output.pres)));
661     txt = "t = " + tstring + "; P.out" + pstring + "; P.avg" + astring;
662     title(txt); %title(sprintf('t = %.6f',t));
663     drawnow limitrate
664     frame = getframe(f);
665     im = frame2im(frame);
666     [imind,cm] = rgb2ind(im,256);
667     if t == dt
668         imwrite(imind,cm,filename,'gif','DelayTime',0.01, 'Loopcount',inf);
669     else
670         imwrite(imind,cm,filename,'gif','DelayTime',0.01, 'WriteMode','append');
671     end
672 end
673 figure(2); plot(time,output.pres); xlabel("Time"); ylabel("Output Pressure"); saveas(gcf,filename2)
674 out = output.pres;
675 close all
676
677 function out = solver10_XOR(lambda, filename, freq, filename2)
678
679 % Geometric constraints
680 l = lambda / 2;
681 w = lambda / 10;
682 d = 0.1 * lambda;
683 d1 = 0.055 * lambda;
684
685 l1 = (0.03 * 1);
686 l2 = (0.23 * 1);
687 w1.upper = (0.69 * w); % changing
688 w1.lower = (0.22 * w); % changing
689 w3 = (0.05 * w);
690 w2.upper = ((w - w1.upper - 4 * w3) / 2);
691 w2.lower = ((w - w1.lower - 4 * w3) / 2);
692 w4 = ((l-4*l2)/5);
693 l1d2 = ((l-4*l1)/8);
694 l1d3 = ((l1-l1)/2);

```

```

691
692 Lx = w + l + dl + d;
693 Ly = 2*w + d;
694 dx = 1; dy = 1;
695 nx = fix(Lx/dx) + 2; ny = fix(Ly/dy) + 2;
696
697 x = linspace(0, Lx, nx); y = linspace(0, Ly, ny);
698
699 # Field definition
700
701 p = 0*ones(nx,ny);
702 p.prev = p; p.next = p;
703
704 CFL = 0.7; % c * dt/dx
705 c = 1; %c/lambda; freq1 = c/lambda; freq2 = c/lambda;
706 dt = CFL*dx/c;
707 output.pres = [];
708 time = [];
709
710 # Initial conditions
711 T = 20000*dt; % Runtime
712
713 t = 0;
714 f = figure(1);
715 set(gcf,'Position',[100 100 1400 500]);
716
717 while t < T
718     % Absorbing Boundaries
719     p.next(1,:) = p(2,:); + ((CFL-1)/(CFL+1))*(p.next(2,:)-p(1,:));
720     p.next(end,:) = p(end-1,:); + ((CFL-1)/(CFL+1))*(p.next(end-1,:)-p(end,:));
721
722     % Rigid Boundaries
723     p(1:end, [1 end]) = 0; % Upper and lower boundaries
724     %p([1 end], 1:end) = 0; % Left and right boundaries
725     p(1:w+1, w:dx:end-w-2*dx) = 0; % Middle region
726     p(w+1:d:end, [1:w w+d:dx:end]) = 0; % Right bottom and top regions
727     p(w:w+1, [1:dx:w3+dx w-w3-dx:w]) = 0; [end-w3:end-dx end-w-dx:end-w+w3-dx] = 0; % 1/3
728
729 p([w:w+4 w+w4+12:w+2*w4+12:w+2*w4+2*12:w+3*w4+3*12:w+4*w4+3*12 w+4*w4+4*12:w+5*w4+4*12], [w3+2*dx:w3+dx+w2_upper ...
    w-w3-1*dx-w2_upper:w-w3-2*dx]) = 0; % 2/3
730 p([w:w+4 w+w4+12:w+2*w4+2*12:w+3*w4+2*12 w+3*w4+3*12:w+4*w4+3*12 w+4*w4+4*12:w+5*w4+4*12], [end-w3-w2_lower:end-w3-dx ...
    end-w+w3+0*dx:end-w+w+w2_lower+1*dx]) = 0; % 2/3
731 p([w:w+w4+1d3 w+w4+12-1d3:w+2*w4+12+1d3 w+2*w4+2*12-1d3:w+3*w4+2*12+1d3 w+3*w4+3*12-1d3:w+4*w4+3*12+1d3 w+5*w4+4*12], ...
    [2*dx+w2_upper:w3:w3+2*dx+w2_upper+w3 w-w3-w2_upper-w3-2*dx:w-w2_upper-w3-2*dx]) = 0; % 3/3
732 p([w:w+w4+1d3 w+w4+12-1d3:w+2*w4+12+1d3 w+2*w4+2*12-1d3:w+3*w4+2*12+1d3 w+3*w4+3*12-1d3:w+4*w4+3*12+1d3 w+5*w4+4*12], ...
    [end-w3-w2_lower-w3-1*dx:end-w2_lower-w3-1*dx end-w+2*dx+w2_lower+w3:end-w+w3+2*dx+w2_lower+w3]) = 0; % 3/3
733
734 t = t+dt;
735 p.prev = p; p = p.next;
736
737 %p.next(1:end,1) = p(1:end,2); + ((CFL-1)/(CFL+1))*(p.next(1:end,2)-p(1:end,1));
738 p.next(1:end,end) = p(1:end,end-1); + ((CFL-1)/(CFL+1))*(p.next(1:end,end-1)-p(1:end,end));
739
740 % Source 1
741 p(2, end-lambda/10:end-1) = 1*sin(2*pi*t*freq);
742
743 % Source 2
744 %p(2, 2:lambda/10-1) = 1*sin(2*pi*t*freq);
745
746 % Rigid Boundaries
747 p(1:end, [1 end]) = 0; % Upper and lower boundaries
748 %p([1 end], 1:end) = 0; % Left and right boundaries
749 p(1:w+1, w:dx:end-w-2*dx) = 0; % Middle region
750 p(w+1:d:end, [1:w w+d:dx:end]) = 0; % Right bottom and top regions
751 p(w:w+1, [1:dx:w3+dx w-w3-dx:w]) = 0; [end-w3:end-dx end-w-dx:end-w+w3-dx] = 0; % 1/3
752
753 p([w:w+4 w+w4+12:w+2*w4+2*12:w+2*w4+2*12:w+3*w4+3*12:w+4*w4+3*12 w+4*w4+4*12:w+5*w4+4*12], [w3+2*dx:w3+dx+w2_upper ...
    w-w3-1*dx-w2_upper:w-w3-2*dx]) = 0; % 2/3
754 p([w:w+4 w+w4+12:w+2*w4+2*12:w+3*w4+2*12 w+3*w4+3*12:w+4*w4+3*12 w+4*w4+4*12:w+5*w4+4*12], [end-w3-w2_lower:end-w3-dx ...
    end-w+w3+0*dx:end-w+w+w2_lower+1*dx]) = 0; % 2/3
755 p([w:w+w4+1d3 w+w4+12-1d3:w+2*w4+12+1d3 w+2*w4+2*12-1d3:w+3*w4+2*12+1d3 w+3*w4+3*12-1d3:w+4*w4+3*12+1d3 w+5*w4+4*12], ...
    [2*dx+w2_upper:w3:w3+2*dx+w2_upper+w3 w-w3-w2_upper-w3-2*dx:w-w2_upper-w3-2*dx]) = 0; % 3/3
756 p([w:w+w4+1d3 w+w4+12-1d3:w+2*w4+12+1d3 w+2*w4+2*12-1d3:w+3*w4+2*12+1d3 w+3*w4+3*12-1d3:w+4*w4+4*12-1d3:w+5*w4+4*12], ...
    [end-w3-w2_lower-w3-1*dx:end-w2_lower-w3-1*dx end-w+2*dx+w2_lower+w3:end-w+w3+2*dx+w2_lower+w3]) = 0; % 3/3
757
758 % Finite difference solution
759 for i = 2:nx-1
760     for j = 2:ny-1
761         p.next(i,j) = 2*p(i,j) - p.prev(i,j) + CFL^2 * (p(i+1,j) + p(i,j+1) - 4*p(i,j) + p(i-1,j) + p(i,j-1));
762     end
763 end
764
765 k = abs(p);
766 % Rigid Boundaries
767 k(1:end, [1 end]) = -1; % Upper and lower boundaries
768 %p([1 end], 1:end) = 0; % Left and right boundaries
769 k(1:w+1, w:dx:end-w-2*dx) = -1; % Middle region
770 k(w+1:d:end, [1:w w+d:dx:end]) = -1; % Right bottom and top regions
771 k(w:w+1, [1:dx:w3+dx w-w3-dx:w]) = -1; [end-w3:end-dx end-w-dx:end-w+w3-dx] = -1; % 1/3
772
773 k([w:w+w4 w+w4+12:w+2*w4+2*12:w+2*w4+2*12:w+3*w4+3*12:w+4*w4+4*12:w+5*w4+4*12], [w3+2*dx:w3+dx+w2_upper ...
    w-w3-1*dx-w2_upper:w-w3-2*dx]) = -1; % 2/3
774 k([w:w+4 w+w4+12:w+2*w4+12:w+2*w4+2*12:w+3*w4+2*12 w+3*w4+3*12:w+4*w4+3*12 w+4*w4+4*12:w+5*w4+4*12], [end-w3-w2_lower:end-w3-dx ...
    end-w+w3+0*dx:end-w+w+w2_lower+1*dx]) = -1; % 2/3
775 k([w:w+w4+1d3 w+w4+12-1d3:w+2*w4+12+1d3 w+2*w4+2*12-1d3:w+3*w4+2*12+1d3 w+3*w4+3*12-1d3:w+4*w4+3*12+1d3 w+5*w4+4*12], ...
    [2*dx+w2_upper:w3:w3+2*dx+w2_upper+w3 w-w3-w2_upper-w3-2*dx:w-w2_upper-w3-2*dx]) = -1; % 3/3
776 k([w:w+w4+1d3 w+w4+12-1d3:w+2*w4+12+1d3 w+2*w4+2*12-1d3:w+3*w4+2*12+1d3 w+3*w4+3*12-1d3:w+4*w4+4*12-1d3:w+5*w4+4*12], ...
    [end-w3-w2_lower-w3-1*dx:end-w2_lower-w3-1*dx end-w+2*dx+w2_lower+w3:end-w+w3+2*dx+w2_lower+w3]) = -1; % 3/3
777
778 if round(mod(t/dt,25)) == 1
779     output = mean(mean(k(end-d/2:end,w:dx:w+d))); %max(max(k(w+l+d1:end,w:dx:w+d));
780     output.pres = [output.pres output];
781     time = [time t];
782 end
783 if round(mod(t/dt,25)) == 1
784     % Visualize
785     % COLORBAR
786     indexValue = 0.4; % value for which to set a particular color
787     topColor = [1 0 0]; % color for maximum data value (red = [1 0 0])
788     indexColor = [1 1 1]; % color for indexed data value (white = [1 1 1])
789     bottomColor = [0 0 1]; % color for minimum data value (blue = [0 0 1])

```

```

790     largest = max(max(k));
791     smallest = min(min(k));
792     L = size(p,1);
793     index = L*abs(indexValue-smallest)/(largest-smallest);
794     customCMap1 = [linspace(bottomcolor(1),indexColor(1),100*index)',...
795         linspace(bottomcolor(2),indexColor(2),100*index)',...
796         linspace(bottomcolor(3),indexColor(3),100*index)'];
797     customCMap2 = [linspace(indexColor(1),topColor(1),100*(L-index))',...
798         linspace(indexColor(2),topColor(2),100*(L-index))',...
799         linspace(indexColor(3),topColor(3),100*(L-index))'];
800     customCMap = [customCMap1;customCMap2]; % Combine colormaps
801     colormap(customCMap)
802
803     imagesc(x,y,k'); colorbar; caxis([-1 max(max(abs(k)))]); % caxis([-1 1]);
804     colorbar; axis equal;
805     grid off; box off; axis off
806     ax = gca;
807     ax.Ydir = 'normal';
808     tstring = num2str(t); pstring = num2str(output); astring = num2str(mean(nonzeros(output_pres)));
809     txt = "t = " + tstring + "; P.out" = + pstring + "; P.avg" = + astring;
810     title(txt); %title(sprintf('t = %.6f',t));
811     drawnow limitrate
812     frame = getframe(f);
813     im = frame2im(frame);
814     [imind,cm] = rgb2ind(im,256);
815     if t == dt
816         imwrite(imind,cm,filename,'gif','DelayTime',0.01, 'Loopcount',inf);
817     else
818         imwrite(imind,cm,filename,'gif','DelayTime',0.01, 'WriteMode','append');
819     end
820 end
821 end
822 figure(2); plot(time,output_pres); xlabel("Time"); ylabel("Output Pressure"); saveas(gcf,filename2)
823 out = output_pres;
824 close all
825 end
826
827 function out = solver01XOR(lambda, filename, freq, filename2)
828
829     % Geometric constraints
830     l = lambda / 2;
831     w = lambda / 10;
832     d = 0.1 * lambda;
833     d1 = 0.055 * lambda;
834
835     l1 = (0.03 * 1);
836     l2 = (0.23 * 1);
837     w1_upper = (0.69 * w); % changing
838     w1_lower = (0.22 * w); % changing
839     w3 = (0.05 * w);
840     w2_upper = ((w - w1_upper - 4 * w3) / 2);
841     w2_lower = ((w - w1_lower - 4 * w3) / 2);
842     w4 = ((l-d+12)/5);
843     ld1 = ((l-4*d+11)/8);
844     ld3 = ((l2-l1)/2);
845
846     Lx = w + l + d1 + d;
847     Ly = 2*w + d;
848     dx = 1; dy = 1;
849     nx = fix(Lx/dx) + 2; ny = fix(Ly/dy) + 2;
850
851     x = linspace(0, Lx, nx); y = linspace(0, Ly, ny);
852
853     % Field definition
854
855     p = 0*ones(nx,ny);
856     p.prev = p; p.next = p;
857
858     CFL = 0.7; % c * dt/dx
859     c = 1; %c/lambda; freq1 = c/lambda; freq2 = c/lambda;
860     dt = CFL*dx/c;
861     output_pres = [];
862     time = {};
863
864     % Initial conditions
865     T = 20000*dt; % Runtime
866
867     t = 0;
868     f = figure(1);
869     set(gcf, 'Position', [100 100 1400 500])
870
871     while t < T
872         % Absorbing Boundaries
873         p.next(1,:) = p(2,:); + ((CFL-1)/(CFL+1))*(p.next(2,:)-p(1,:));
874         p.next(end,:) = p(end-1,:); + ((CFL-1)/(CFL+1))*(p.next(end-1,:)-p(end,:));
875
876         % Rigid Boundaries
877         p(1:end, [1 end]) = 0; % Upper and lower boundaries
878         %p([1 end], 1:end) = 0; % Left and right boundaries
879         p(l:w:l, w:dx:end-w-2*dx) = 0; % Middle region
880         p(w1:d1:end, [1:w w+d:dx:end]) = 0; % Right bottom and top regions
881         p(w:w:l, [1:dx:w3:dx - w3-dx:w]) = 0; p(w:w+l, [end-w3:end-dx end-w-dx:end-w+w3-dx]) = 0; % 1/3
882
883         p([w:w+l2:w+w4+l2:w+2*w4+2*l2:w+3*w4+3*l2:w+4*w4+3*l2 w+4*w4+4*l2:w+5*w4+4*l2], [w3+2*dx:w3+dx+w2.upper ...
884             w-w3-1*dx-w2.upper:w-w3-2*dx]) = 0; % 2/3
885         p([w:w+w4 w+w4+l2:w+2*w4+l2 w+2*w4+2*l2:w+3*w4+2*l2 w+3*w4+3*l2:w+4*w4+3*l2 w+4*w4+4*l2:w+5*w4+4*l2], [end-w3-w2.lower:end-w3-dx ...
886             end-w+w3+0*dx:end-w+w3+w2.lower+1*dx]) = 0; % 2/3
887         p([w:w+w4+l3 w+w4+l2-ld3:w+2*w4+l2+l3 w+2*w4+2*l2-ld3:w+3*w4+2*l2+l3 w+3*w4+3*l2-1d3:w+4*w4+3*l2+1d3 w+4*w4+4*l2:w+5*w4+4*l2], ...
888             [2*dx+w2.upper:w3:w3+2*dx+w2.upper+w3 w-w3-w2.upper-w3-2*dx:w-w2.upper-w3-2*dx]) = 0; % 3/3
889         p([w:w+w4+l3 w+w4+l2-ld3:w+2*w4+l2+l3 w+2*w4+2*l2-ld3:w+3*w4+2*l2+l3 w+3*w4+3*l2-1d3:w+4*w4+3*l2+1d3 w+4*w4+4*l2:w+5*w4+4*l2], ...
890             [end-w3-w2.lower-w3-1:dx end-w+2*dx+w2.lower+w3:end-w+w3+2*dx+w2.lower+w3]) = 0; % 3/3
891
892         t = t+dt;
893         p.prev = p; p = p.next;
894
895         %p.next(1:end,1) = p(1:end,2) + ((CFL-1)/(CFL+1))*(p.next(1:end,2)-p(1:end,1));
896         p.next(1:end,end) = p(1:end,end-1) + ((CFL-1)/(CFL+1))*(p.next(1:end,end-1)-p(1:end,end));
897
898         % Source 1
899         %p(2, end-lambda/10:end-1) = 1*sin(2*pi*t*freq);
900
901

```

```

897 % Source 2
898 p(2, 2:lambd/10-1) = 1*sin(2*pi*t*freq);
899
900 % Rigid Boundaries
901 p(l:end, [1 end]) = 0; % Upper and lower boundaries
902 %p([l end], 1:end) = 0; % Left and right boundaries
903 p(l:w+1, wdx:end-w-2*dx) = 0; % Middle region
904 p(w+1:d1:end, [1:w w+d1:dx:end]) = 0; % Right bottom and top regions
905 p(w:w+1, [1+dx:w3:dx w-w3-dx:w]) = 0; p(w:+1, [end-w3:end-dx end-w-dx:end-w+w3-dx]) = 0; % 1/3
906
907 p([w:w+w4+12:w+w2+2*w+12 w+2*w4+2*12:w+w3*w4+3*12:w+4*w4+2*12 w+3*w4+3*12:w+4*w4+4*12], [w3+2*dx:w3+dx+w2.upper ...
908 w-w3-1*dx-w2.upper:w-w3-2*dx]) = 0; % 2/3
909 p([w:w+w4 w+w4+12:w+2*w4+12 w+2*w4+2*12:w+3*w4+2*12 w+3*w4+3*12:w+4*w4+3*12 w+4*w4+4*12], [end-w3-w2.lower:end-w3-dx ...
910 end-w+w3+0*dx:end-w+w3+w2.lower+1*dx]) = 0; % 2/3
911 p([w:w+w4+12:w+w4+12-1d3:w+2*w4+2*12+1d3 w+2*w4+2*12-1d3:w+3*w4+2*12+1d3 w+3*w4+3*12-1d3:w+4*w4+3*12+1d3 w+4*w4+4*12-1d3:w+5*w4+4*12], ...
912 [2+dx+w2.upper:w3:w3+2*dx+w2.upper+w3 w-w3-w2.upper:w-w3-2*dx]) = 0; % 3/3
913 p([w:w+w4+12:w+w4+12-1d3:w+2*w4+2*12+1d3 w+2*w4+2*12-1d3:w+3*w4+2*12+1d3 w+3*w4+3*12-1d3:w+4*w4+3*12+1d3 w+4*w4+4*12-1d3:w+5*w4+4*12], ...
914 [end-w3-w2.lower:w3-1*dx:end-w2.lower:w3-1*dx end-w+2*dx+w2.lower+w3:end-w+w3+2*dx+w2.lower+w3]) = 0; % 3/3
915
916 % Finite difference solution
917 for i = 2:nx-1
918     for j = 2:ny-1
919         p.next(i,j) = 2*p(i,j) - p.prev(i,j) + CFL^2 * ( p(i+1,j) + p(i,j+1) - 4*p(i,j) + p(i-1,j) + p(i,j-1));
920     end
921 end
922
923 k = abs(p);
924 % Rigid Boundaries
925 k(l:end, [1 end]) = -1; % Upper and lower boundaries
926 %p([l end], 1:end) = 0; % Left and right boundaries
927 k(l:w+1, wdx:end-w-2*dx) = -1; % Middle region
928 k(w+1:d1:end, [1:w w+d1:dx:end]) = -1; % Right bottom and top regions
929 k(w:w+1, [1+dx:w3:dx w-w3-dx:w]) = -1; k(w:w+1, [end-w3:end-dx end-w-dx:end-w+w3-dx]) = -1; % 1/3
930
931 k([w:w+w4 w+w4+12:w+w2+2*w+12 w+2*w4+2*12:w+w3*w4+3*12:w+4*w4+2*12 w+3*w4+3*12:w+4*w4+4*12], [w3+2*dx:w3+dx+w2.upper ...
932 w-w3-1*dx-w2.upper:w-w3-2*dx]) = -1; % 2/3
933 k([w:w+w4 w+w4+12:w+2*w4+12 w+2*w4+2*12:w+3*w4+2*12 w+3*w4+3*12:w+4*w4+3*12 w+4*w4+4*12], [end-w3-w2.lower:end-w3-dx ...
934 end-w+w3+0*dx:end-w+w3+w2.lower+1*dx]) = -1; % 2/3
935 k([w:w+w4+12-1d3:w+2*w4+2*12+1d3 w+2*w4+2*12-1d3:w+3*w4+2*12+1d3 w+3*w4+3*12-1d3:w+4*w4+3*12+1d3 w+4*w4+4*12-1d3:w+5*w4+4*12], ...
936 [2+dx+w2.upper:w3:w3+2*dx+w2.upper+w3 w-w3-w2.upper:w-w3-2*dx]) = -1; % 3/3
937 k([w:w+w4+12-1d3:w+2*w4+2*12+1d3 w+2*w4+2*12-1d3:w+3*w4+2*12+1d3 w+3*w4+3*12-1d3:w+4*w4+3*12+1d3 w+4*w4+4*12-1d3:w+5*w4+4*12], ...
938 [end-w3-w2.lower:w3-1*dx:end-w2.lower:w3-1*dx end-w+2*dx+w2.lower+w3:end-w+w3+2*dx+w2.lower+w3]) = -1; % 3/3
939
940 if round(mod(t/dt,25)) == 1
941     output = mean(mean(k(end-d/2:end,w+dx:w+d))); %max(max(k(w+1:d1:end,w+dx:w+d)));
942     output.pres = [output.pres output];
943     time = [time t];
944 end
945 if round(mod(t/dt,25)) == 1
946     % Visualize
947     % COLORBAR
948     indexValue = 0.4; % value for which to set a particular color
949     topColor = [1 0 0]; % color for maximum data value (red = [1 0 0])
950     indexColor = [1 1 1]; % color for indexed data value (white = [1 1 1])
951     bottomColor = [0 0 1]; % color for minimum data value (blue = [0 0 1])
952     largest = max(max(k));
953     smallest = min(min(k));
954     L = size(p,1);
955     index = L*abs(indexValue-smallest)/(largest-smallest);
956     customCMap1 = [linspace(bottomcolor(1),indexColor(1),100*index)',...
957                     linspace(bottomcolor(2),indexColor(2),100*index)',...
958                     linspace(bottomcolor(3),indexColor(3),100*index)'];
959     customCMap2 = [linspace(indexColor(1),topColor(1),100*(L-index))',...
960                     linspace(indexColor(2),topColor(2),100*(L-index))',...
961                     linspace(indexColor(3),topColor(3),100*(L-index))'];
962     customCMap = [customCMap1;customCMap2]; % Combine colormaps
963     colormap(customCMap)
964
965 imagesc(x,y,k'); colorbar; caxis([-1 max(max(abs(k)))]); % caxis([-1 1]);
966 colorbar; axis equal;
967 grid off; box off; axis off
968 ax = gca;
969 ax.YDir = 'normal';
970 tstring = num2str(t); pstring = num2str(output); astring = num2str(mean(nonzeros(output.pres)));
971 txt = "t = " + tstring + "; P.out" = + pstring + "; P.avg" = + astring;
972 title(txt); %title(sprintf('t = %.6f',t));
973 drawnow limitrate
974 frame = getframe(f);
975 im = frame2im(frame);
976 [imind,cm] = rgb2ind(im,256);
977 if t == dt
978     imwrite(imind,cm,filename,'gif','DelayTime',0.01, 'Loopcount',inf);
979 else
980     imwrite(imind,cm,filename,'gif','DelayTime',0.01, 'WriteMode','append');
981 end
982 end
983 end
984
985 figure(2); plot(time,output.pres); xlabel("Time"); ylabel("Output Pressure"); saveas(gcf,filename2)
986 out = output.pres;
987 close all
988 end

```