MOL^T Based Fast High-order Three Dimensional A-Stable Scheme for Wave Propagation

M. Thavappiragasam[1,2], A. Viswanathan[4], and A. Christlieb[1,3]

[1]*Department of Computational Mathematics Science and Engineering, Michigan State University, MI, United States*
[2]*Department of Electrical and Computer Engineering, Michigan State University, MI, United States*
[3]*Department of Mathematics, Michigan State University, MI, United States*
[4]*Department of Mathematics and Statistics, University of Michigan, MI, United States and*

Invited Paper 22/11/17, Accepted 8/12/17

We present a fast (linear-time), high-order, multi-dimensional, A-stable implicit wave solver applicable for electromagnetic (EM) problems, specifically targeting plasma science. This approach uses a Method Of Lines Transpose (MOL^T) formulation combined with an Alternating Direction Implicit (ADI) scheme. In this scheme, a PDE is first discretized in time, and then the resulting boundary-value problems are solved using a Green's function method. In particular, inverse of the resulting modified Helmholtz operator is analytically constructed and evaluated efficiently using an $O(N)$ recursive fast convolution algorithm. Extension to multi-dimensions is formed using an ADI scheme, and each line is solved independently. In this work, we propose a higher order 3D scheme which is able to deal with complicated geometries. The scheme is successfully evaluated using several 3D test problems.

Keywords: Method of Lines Transpose, Transverse Method of Lines, Implicit Methods, Boundary Integral Methods, Alternating Direction Implicit Methods, ADI Schemes, Higher Order Schemes, Multi-Dimensional Schemes, Complex Geometry.

PACS: 65M20, 78A40, 35G15

MSC: 41.20.Jb, 52.35.Hr, 96.60.Tf

1. INTRODUCTION

The method of lines transpose (MOL^T) methodology (also known as the transverse method of lines or Rothe's method)[1–3] can be used in designing scalable direct implicit methods that are as efficient as explicit solvers, ideal for multicore computing. These implicit methods are unconditionally stable to arbitrary order accuracy and offer a unique approach to bridging scales in applications such as plasma science. By making use of clever factorizations of operators and efficient convolution methods, MOL^T avoids the use of matrices that typically result from the discretization of the spatial parts of a problem and thus eliminates the main bottleneck in scaling implicit methods. In this approach, a PDE is first discretized in time, and then the coupled set of resulting boundary-value problems are solved using a Green's function method[4]. In particular, we use an analytic inversion of the associated differential operator implemented by utilizing an $\mathcal{O}(N)$, recursive fast convolution algorithm. Extension to multi dimension is formed using an alternating direction implicit (ADI) scheme, and each line is solved independently. In order to take large time steps in problems we need higher order accuracy in time. The high order scheme is achieved by utilizing a Lax-Wendroff approach to exchange time derivatives with spatial derivatives using even powers of the Laplacian [5]. The inversions of higher derivatives can be performed analytically, so that the resulting scheme is made explicit, even at the semi-discrete level. In constructing the analytical convolution operators, we incorporate the boundary conditions directly. In this way, without additional complexity Dirichlet and periodic boundary conditions can be implemented to higher order. Since dealing with outflow boundary conditions depend on previous time step values at the boundaries, we need to do little more work to implement it in higher order. In this paper, we give a brief introduction for the higher order outflow boundary conditions for 1D schemes. A detailed explanation for multidimensional scheme with outflow boundary condition will be talked in upcoming communications. Complex geometries are handled via an embedded boundary approach (see [6] for second order accurate 2D scheme).

In Section 2, we give the implicit semi-discrete solution for 3D wave equation, in Section 3 we derive the high-order scheme and deal with outflow boundary condition, in Section 4 we explain how we can handle multidimensional problems with complex geometries, and finally we report a set of test cases in Section 5.

2. THREE-DIMENSIONAL IMPLICIT WAVE EQUATION SOLVER USING ADI SCHEME

The three dimensional implicit scheme is formed using an alternating direction implicit (ADI) scheme (see [4] for one and two dimensional schemes), and each line is solved independently. The multi-dimensional wave equation can be represented using the initial boundary value problem

$$\frac{1}{c^2}\frac{\partial^2 u}{\partial t^2} - \nabla^2 u = S(\mathbf{k}, t), \quad \mathbf{k} \in \Omega, \quad t > 0 \qquad (1)$$

$$u(\mathbf{k},0) = f(\mathbf{k}), \quad \mathbf{k} \in \Omega$$
$$u_t(\mathbf{k},0) = g(\mathbf{k}), \quad \mathbf{k} \in \Omega$$

with consistent boundary conditions. Suppose the domain $\Omega = [x_a, x_b] \times [y_a, y_b] \times [z_a, z_b]$ for three dimensional wave equation, Dirichlet and outflow boundary conditions can be defined by,

1. Dirichlet boundary condition:
   $u(x_a,t) = U_{Xa}(t), \quad u(x_b,t) = U_{Xb}(t),$
   $u(y_a,t) = U_{Ya}(t), \quad u(y_b,t) = U_{Yb}(t),$
   $u(z_a,t) = U_{Za}(t), \quad u(z_b,t) = U_{Zb}(t).$

2. Outflow boundary condition:
   $u_t(x_a,t) = cu_x(x_a,t), \quad u_t(x_b,t) = -cu_x(x_b,t),$
   $u_t(y_a,t) = cu_y(y_a,t), \quad u_t(y_b,t) = -cu_y(y_b,t),$
   $u_t(z_a,t) = cu_z(z_a,t), \quad u_t(z_b,t) = -cu_z(z_b,t).$

Using MOL^T, we first perform a temporal discretization and then approximate modified Helmotz operator. Let us consider a scheme for spatial dimension three. The discretization begins with a time-centered finite different approximation,

$$\frac{u^{n+1} - 2u^n + u^{n-1}}{(c\Delta t)^2} - \nabla^2\Big(u^n + \frac{u^{n+1} - 2u^n + u^{n-1}}{\beta^2}\Big) = S^n(\mathbf{k},t) \tag{2}$$

where the averaging parameter $\beta > 0$ and bounded with a specific upper limit based on temporal order of accuracy [5]. It gives us a semi-discrete equation,

$$\Big(1 - \frac{1}{\alpha^2}\nabla^2\Big)\Big(\beta^2 u^n + u^{n+1} - 2u^n + u^{n-1}\Big) = \beta^2 u^n + \frac{\beta^2}{\alpha^2} S^n(\mathbf{k},t) \tag{3}$$

where $\alpha = \frac{\beta}{c\Delta t}$. Next, we approximate the modified Helmotz operator using ADI splitting,

$$\begin{aligned} 1 - \frac{1}{\alpha^2}\nabla^2 &= 1 - \frac{1}{\alpha^2}\Big(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2}\Big) \\ &= \Big(1 - \frac{1}{\alpha^2}\frac{\partial^2}{\partial x^2}\Big)\Big(1 - \frac{1}{\alpha^2}\frac{\partial^2}{\partial y^2}\Big)\Big(1 - \frac{1}{\alpha^2}\frac{\partial^2}{\partial z^2}\Big) \\ &\quad + \frac{1}{\alpha^4}\Big(\frac{\partial^4}{\partial x^2 \partial y^2} + \frac{\partial^4}{\partial x^2 \partial z^2} + \frac{\partial^4}{\partial y^2 \partial z^2}\Big) \\ &\quad - \frac{1}{\alpha^6}\frac{\partial^6}{\partial x^2 \partial y^2 \partial z^2} \end{aligned}$$

Hence the approximation gives,

$$1 - \frac{1}{\alpha^2}\nabla^2 = \mathcal{L}_x \mathcal{L}_y \mathcal{L}_z + \mathcal{O}\big((c\Delta t)^4\big) \tag{4}$$

where $\mathcal{L}_x$, $\mathcal{L}_y$, and $\mathcal{L}_z$ are one dimensional univariate modified Helmholtz operators ([4]) applied in the indicated spatial variable,

$$\mathcal{L}_x = 1 - \frac{\partial_{xx}}{\alpha^2}, \quad \mathcal{L}_y = 1 - \frac{\partial_{yy}}{\alpha^2}, \quad and \quad \mathcal{L}_z = 1 - \frac{\partial_{zz}}{\alpha^2}$$

Now, equation (3) can be expressed by,

$$\mathcal{L}_x \mathcal{L}_y \mathcal{L}_z \Big[\beta^2 u^n + u^{n+1} - 2u^n + u^{n-1}\Big] = \beta^2 u^n + \frac{\beta^2}{\alpha^2} S^n(\mathbf{k},t) \tag{5}$$

Upon taking the inverse of the modified Helmholtz operators $\mathcal{L}_x$, $\mathcal{L}_y$, and $\mathcal{L}_z$, we obtain the semi-discrete solution for three dimensions with second-order temporal accuracy,

$$u^{n+1} - 2u^n + u^{n-1} = -\beta^2 \mathcal{D}^{(1)}_{xyz}[u^n] + \beta^2 \mathcal{L}_z^{-1}\mathcal{L}_y^{-1}\mathcal{L}_x^{-1}\Big[\frac{1}{\alpha^2} S^n\Big](\mathbf{k},t) \tag{6}$$

where the three dimensional operator is,

$$\mathcal{D}^{(1)}_{xyz}[u] := u - \mathcal{L}_z^{-1}\mathcal{L}_y^{-1}\mathcal{L}_x^{-1}[u], \tag{7}$$

with superscript 1 which denotes first level of computing or just the second order scheme, while in two dimensions the scheme is

$$u^{n+1} - 2u^n + u^{n-1} = -\beta^2 \mathcal{D}^{(1)}_{xy}[u^n] + \beta^2 \mathcal{L}_y^{-1}\mathcal{L}_x^{-1}\Big[\frac{1}{\alpha^2} S^n\Big](\mathbf{k},t), \tag{8}$$

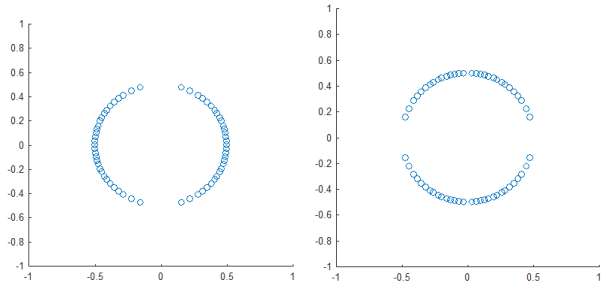
$$\mathcal{D}^{(1)}_{xy}[u] := u - \mathcal{L}_y^{-1}\mathcal{L}_x^{-1}[u].$$

Using the free space Green function, the inverse modified Helmholtz operator $\mathcal{L}_x^{-1}$ in x direction can be defined by,

$$\mathcal{L}_x^{-1}[u] := \underbrace{\frac{\alpha}{2}\int_a^b e^{-\alpha|x-x'|}u(x')dx'}_{\text{Particular solution}} + \underbrace{Ax_1 e^{-\alpha(x-a)} + Bx_1 e^{-\alpha(b-x)}}_{\text{Homogeneous solution}} \tag{9}$$

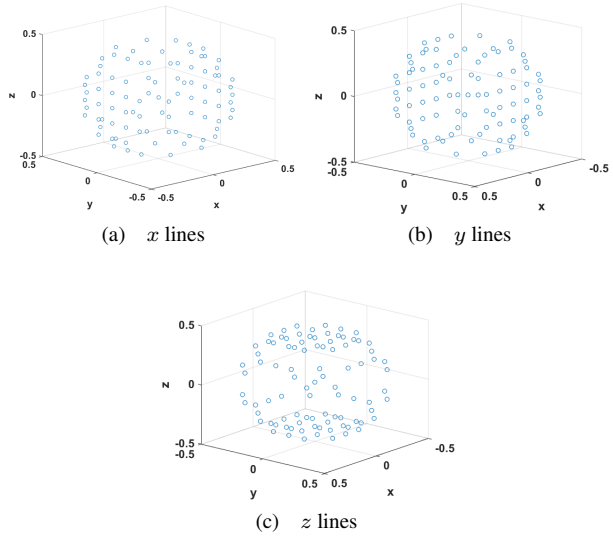
where the Ax_1 and Bx_1 are homogeneous boundary coefficients along the x direction. Since subscripts of the coefficients denote the proper level of computing, Ax_1 and Bx_1 are dealt in the first level of computing. We can define $\mathcal{L}_y^{-1}$ and $\mathcal{L}_z^{-1}$ in the similar way using homogeneous boundary coefficients (Ay_1, By_1) and (Az_1, Bz_1) respectively.
Inversion of the operator $\mathcal{L}_{\mathbf{k}}^{-1}$ is performed sequentially leading to the usual x, y and z "sweeps" of the ADI algorithm that is represented by a combination of the operator $\mathcal{L}_{\mathbf{k}}^{-1}$. Because the inverse operators are defined along lines, it is quite natural to discretize 2D and 3D regions along Cartesian lines. However, the end points of these lines

are not restricted to residing at mesh points and so can always be chosen to lie on the boundary $\partial\Omega$. For example, the boundary points for a circle are shown in Figure 1 and for a sphere in Figure 2. Because the 1D convolution algorithm can incorporate unstructured meshes locally without incurring time-step restrictions, it is of no concern to have the boundary lie arbitrarily close to a mesh point. Because inversion along each line only requires the solution of a two-point boundary-value problem, the usual boundary integral is never actually constructed. Instead, the effects of the boundaries are passed throughout the domain via the ADI sweeps. The multi-D domain decomposition simply requires that each direction be complete before starting the next.



(a) x lines (b) y lines

FIG. 1: Boundary points, intersections of the mesh lines with the boundary of a circle used for the ADI x (a) and y (b) sweeps.



(a) x lines (b) y lines

(c) z lines

FIG. 2: Boundary points, intersections of the mesh lines with the boundary of a sphere used for the ADI x (a), y (b) and z (b) sweeps.

Let us see this scheme in a little more detail in two dimensions, The two dimensional rectangular domain with corner nodes (x_a, y_a), (x_a, y_b), (x_b, y_b), (x_a, y_b) is split as one dimensional lines in x and y directions as shown in the Figure 3. Grid points fall in the Cartesian domain $(x_a, x_b) \times (y_a, y_b)$ with spatial distance Δx and Δy along x and y lines respectively. The horizontal lines and vertical lines are solved using the fast convolution line integration during x sweeps and y sweeps respectively. Those x, y sweeps should be performed one by one in any order, after completion of a sweep the result will be saved in an intermediate variable which will then be used to do the other sweep. In order to improve the accuracy of the ADI solve, the inversion of the x and y Helmholtz operators is symmetrized, by averaging the results of x—y and y —x solves. That means we need to apply the operators D_{xy} and D_{yx} and then take the average.

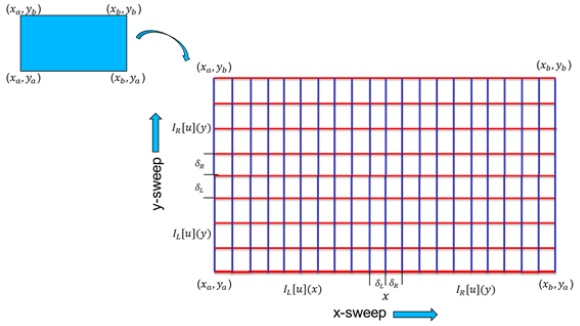


FIG. 3: ADI splitting and performs x - y sweeps

The general approach for implementation of the 2D ADI scheme follows the steps shown below: Assume the number of x and y lines are n_x and n_y respectively.

1. Perform the x-sweep
   At each time step, solve the result in a temporary variable $Linv_x(x, y_i)$ for $1 \le i \le n_y$. The boundary conditions are imposed at $x = a_i$ and b_i.

2. Perform the y-sweep
   For $1 \le j \le n_x$ using $Linv_y(x_j, y)$, solve for the equation $u_{n+1} = 2u_n + u_{n-1} - \beta^2 D_{xy}$, where $D_{xy} = u_n - Linv_y Linv_x$. The boundary conditions are now applied at $y = c_j$ and d_j.

For the case of three dimensions, The rectangular cuboid domain with corner nodes (x_a, y_a, z_a), (x_a, y_b, z_a), (x_b, y_b, z_a), (x_a, y_b, z_a), (x_a, y_a, z_b), (x_a, y_b, z_b), (x_b, y_b, z_b), and (x_a, y_b, z_b) is split as one dimensional lines in x, y, and z directions as shown in the Figure 4, and perform z-sweeps in addition to x and y-sweeps.

We give the detailed algorithm for the two dimensional implicit wave equation solver in Algorithm 1 and relevant supporting functions in Algorithms 2 and 3. Algorithm 2 computes solution u at time t_{n+1} using our two dimensional implicit solver that calls the function *Linv* given in Algorithm 3 to apply the operator $\mathcal{L}^{-1}$ to perform x and

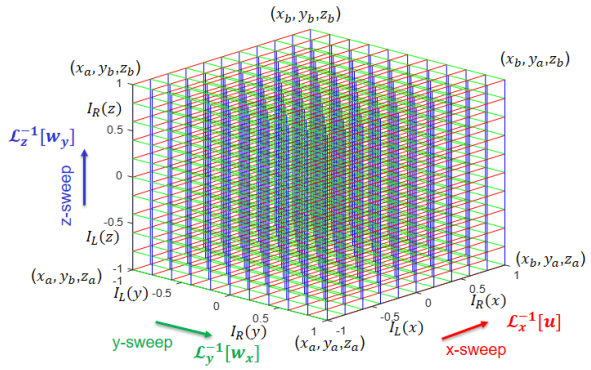


FIG. 4: ADI splitting and performs x, y, and z sweeps

y sweeps. The functions *fastconvol* and *applyBC* called in Algorithm 3 are used to find the particular solution and the homogeneous coefficients A^n and B^n for appropriate boundary conditions respectively. The function *fastconvol* is an implementation of the $O(N)$ recursive compact Simpson's quadrature based approach defined in Algorithm 7. The boundary coefficients A^n and B^n are retrieved as the first and second element of the vector H respectively.

Algorithm 1 Two Dimensional Solver

function *TwoDiWES*$(x_A, x_B, y_A, y_B, n_x, n_y, \Delta t, T, c, \beta, p_s)$
x_A, x_B and y_A, y_B - boundary points along x and y respectively,
n_x, n_y - number of grid points along x and y respectively,
Δt - time step size,
T - total time,
c - wave speed,
β - averaging parameter,
p_s - order of accuracy in space,
  $x = FormGridVector(x_A, x_B, n_x)$
  $y = FormGridVector(y_A, y_B, n_y)$
  $setInitialvalue(u_{n-1}, t_0)$
  $setInitialvalue(u_n, t_1)$
  $\alpha = \frac{\beta}{c\Delta t}$
  $n_t = \frac{T}{\Delta t}$
  $\mu_x = e^{-\alpha(x(n)-x(0))}$
  $\mu_y = e^{-\alpha(y(n)-y(0))}$
  $expA_x = e^{-\alpha(x-x(0))}$
  $expB_x = e^{-\alpha(x(n)-x)}$
  $expA_y = e^{-\alpha(y-y(0))}$
  $expB_y = e^{-\alpha(y(n)-y)}$
  $\nu_x = \alpha(x(2:n_x) - x(1:n_x-1))$
  $\nu_y = \alpha(y(2:n_y) - y(1:n_y-1))$
  $expweight_x = expWeights(\nu_x, p_s)$
  $expweight_y = expWeights(\nu_y, p_s)$
  for $t_n = 1$ to n_t **do**
    $u_{n+1} = computeU(u_{n-1}, u_n, n_x, n_y, \beta, \mu_x, \mu_y, x_A, x_B, y_A, y_B, expA_x, expB_x, expA_y, expB_y, \nu_x, \nu_y, expweight_x, expweight_y, p_s)$
    $u_{n-1} = u_n$
    $u_n = u_{n+1}$

Algorithm 2 Compute u at time t_{n+1}

function *computeU*$(u_{n-1}, u_n, n_x, n_y, \beta, \mu_x, \mu_y, x_A, x_B, y_A, y_B, expA_x, expB_x, expA_y, expB_y, \nu_x, \nu_y, expweight_x, expweight_y, p_s)$
u_{n-1}, u_n - solution at time t_{n-1} and t_n respectively,
n_x, n_y - number of grid points along x and y respectively,
β - averaging parameter,
$\mu_x = e^{-\alpha(x(n)-x(0))}$, $\mu_y = e^{-\alpha(y(n)-y(0))}$,
x_A, x_B and y_A , y_B - boundary points along x and y respectively,
$expA_x$, $expB_x$ and $expA_y$, $expB_y$ - exponential vector of grid distance from left and right boundaries along x and y respectively,
ν_x, ν_y - array of weighted nodes along x and y respectively,
$expweight_x$, $expweight_y$ - array of exponentially weighted nodes along x and y respectively,
p_s -order of accuracy in space
  $Linv_x = Linv(u_n, n_y, \beta, \mu_x, x_A, x_B, expA_x, expB_x, \nu_x, expweight_x, p_s, 0)$
  $Linv_y Linv_x = Linv(Linv_x, n_x, \beta, \mu_y, y_A, y_B, expA_y, expB_y, \nu_y, expweight_y, p_s, 1)$
  $D_{xy} = u_n - Linv_y Linv_x$
  $u_{n+1} = 2u_n + u_{n-1} - \beta^2 D_{xy}$

Algorithm 3 $\mathcal{L}$ inverse; $\mathcal{L}^{-1}$

function *Linv*$(u, n, \beta, \mu, bdry_A, bdry_B, expA, expB, \nu, expweight, p_s, dir)$
u_n - solution at time t_n,
n - number of lines that need to be sweep,
$bdry_A$, $bdry_B$ - boundary points along the specific line,
$expA$ $expB$- exponential vector of grid distance from left and right boundaries respectively along the line,
ν - array of weighted nodes along the line,
$expweight$ - array of exponentially weighted nodes along the line,
p_s - order of accuracy in space,
dir - determine which sweep is going to do, 0 for along x and 1 for y
  if dir == 0 **then** ▷ Do x sweep
    for $i = 1$ to n **do**
      $I = fastconvol(u(i,:), \nu, expweight, p_s)$
      $H = applyBC(u(bdry_A), u(bdry_B), I(bdry_A), I(bdry_B), \beta, bdry_A, bdry_B, \mu)$
      $Linv(i,:) = I + H(1)expA + H(2)expB$
  else ▷ Do y sweep
    for $j = 1$ to n **do**
      $I = fastconvol(u(:,j), \nu, expweight, p_s)$
      $H = applyBC(u(bdry_A), u(bdry_B), I(bdry_A), I(bdry_B), \beta, bdry_A, bdry_B, \mu)$
      $Linv(:,j) = I + H(1)expA + H(2)expB$

3. HIGHER ORDER SCHEME

The higher order accurate solutions was achieved by exchanging time derivation into spatial derivation. As shown in [5], we can introduce a new operator $\mathcal{C}_{xyz}$,

$$\mathcal{C}_{xyz} = \mathcal{L}_y^{-1}\mathcal{L}_z^{-1}\mathcal{D}_x + \mathcal{L}_z^{-1}\mathcal{L}_x^{-1}\mathcal{D}_y + \mathcal{L}_x^{-1}\mathcal{L}_y^{-1}\mathcal{D}_z$$

in combined with $\mathcal{D}_{xyz}$ (equation 7) to get high-order scheme. The $2p$th order scheme as presented in [5] is,

$$u^{n+1} - 2u^n + u^{n-1} = \sum_{p=1}^{\infty}\sum_{m=1}^{p}(-1)^m \frac{2\beta^{2m}}{(2m)!}\binom{p-1}{m-1}\mathcal{C}^m\mathcal{D}^{p-m}[u^n]. \quad (10)$$

This scheme unconditionally stables for all Δt (the 2D scheme was proved in [5]), and the same range for β as given in [5].
The fourth order version in three dimensions cab be expressed by,

$$u^{n+1} - 2u^n + u^{n-1} = -\beta^2\mathcal{C}^{(1)}_{xyz}[u^n] - \left(\beta^2\mathcal{D}^{(2)}_{xyz} - \frac{\beta^4}{12}\mathcal{C}^{(2)}_{xyz}\right)\mathcal{C}^{(1)}_{xyz}[u^n] \quad (11)$$

where superscripts of the operators C and D denote level numbers over the computation. The fourth order scheme has to be implemented in two levels ($p = 2$). At the first level, we compute $\mathcal{C}^{(1)}_{xyz}[u^n]$ and apply the operators $\mathcal{C}^{(2)}_{xyz}$ and $\mathcal{D}^{(2)}_{xyz}$ on the computed $\mathcal{C}^{(1)}_{xyz}[u^n]$ at the second level. Using these level numbers, we can define boundary coefficients related to level 2 computing, (Ax_2, Bx_2), (Ay_2, By_2), and (Az_2, Bz_2) along x, y, and z directions respectively.

Now we consider the variable speed scheme, suppose the wave speed c_{xyz} is bounded with $c_1 \le c_{xyz} \le c_2$, we first normalize it by c_{xyz}/c_2 and use it during the higher order treatment with the term of spatial derivatives for the derivative exchange from time to spatial. This approach gives,

$$u^{n+1} - 2u^n + u^{n-1} = -\beta^2 c^2_{xyz}\mathcal{C}^{(1)}_{xyz}[u^n] - \left(\beta^2 c^2_{xyz}\mathcal{D}^{(2)}_{xyz} - \frac{\beta^4 c^4_{xyz}}{12}\mathcal{C}^{(2)}_{xyz}\right)\mathcal{C}^{(1)}_{xyz}[u^n] \quad (12)$$

3.1. Higher Order Outflow Boundary Condition

In order to obtain higher order outflow boundary conditions, we consider more terms in the Taylor series than second order equation in [4]. For this procedure, we need to know the solution at time steps, t_{n+1}, t_{n+2}, and so on, if we choose the centred finite difference stencil to derive equations with higher order accuracy. Instead we prefer to use one-sided backward finite difference stencils to obtain higher order accuracy and obtain boundary coefficients explicitly. In this paper, we give a higher order scheme for outflow boundary conditions in one dimension only. Because of the extension to multi-dimensions should require additional works, we will see a scheme for higher order outflow in multi-dimensions in our following communications.
Let us derive fourth order outflow boundary conditions in one dimension. First, we construct a time interpolant at the right boundary ($x > b$) using a Taylor series expression of the form

$$u(b, t_n - z\Delta t) \approx u(b, t_n) - z\Delta t u_t(b, t_n) + \frac{z^2\Delta t^2}{2}u_{tt}(b, t_n) - \frac{z^3\Delta t^3}{6}u_{ttt}(b, t_n) + \frac{z^4\Delta t^4}{24}u_{tttt}(b, t_n)$$

By truncating higher order error terms, we only need to approximate the first time derivative to fourth order accuracy, second time derivative to third order accuracy, and so on. To perform this approximation, we use a five point backward finite difference stencil to approximate u_t, u_{tt}, u_{ttt}, and u_{tttt} to the desired order of accuracy. We obtain,

$$\begin{aligned} u(b, t_n - z\Delta t) \approx{}& u^n(b) - z\Big(\frac{25}{12}u^n(b) - 4u^{n-1}(b) + 3u^{n-2}(b) - \frac{4}{3}u^{n-3}(b) + \frac{1}{4}u^{n-4}(b)\Big) \\ &+ \frac{z^2}{2}\Big(\frac{35}{12}u^n(b) - \frac{26}{3}u^{n-1}(b) + \frac{19}{2}u^{n-2}(b) - \frac{14}{3}u^{n-3}(b) + \frac{11}{12}u^{n-4}(b)\Big) - \frac{z^3}{6}\Big(\frac{5}{2}u^n(b) - 9u^{n-1}(b) \\ &+ 12u^{n-2}(b) - 7u^{n-3}(b) + \frac{3}{2}u^{n-4}(b)\Big) + \frac{z^4}{24}\Big(u^n(b) - 4u^{n-1}(b) + 6u^{n-2}(b) - 4u^{n-3}(b) + u^{n-4}(b)\Big) \quad (13) \end{aligned}$$

We need to integrate this expression analytically using lemma 3.0.1 (see [4] for a proof) to compute the boundary coefficient B^n at the right boundary from the equation given in [4],

$$B^n = \frac{\beta}{2}\int_0^1 e^{-\beta z}u(b, t_n - z\Delta t)dz + e^{-\beta}B^{n-1},$$

The equation was derived form the outflow boundary condition at the right boundary $u_t(b, t) = -cu_x(b, t)$ by extending it to $b + ct_n$ after time $t = t_n$, and switch the spatial integrals into time integrals at b.

Lemma 3.0.1 *For integers $m \ge 0$ and real $v > 0$,*

$$E_m := v\int_0^1 \frac{z^m}{m!}e^{-vz}dz = \frac{1}{v^m}\Big(1 - e^{-v}P_m(v)\Big)$$

where $P_m(v) = \sum_{l=0}^{m}\frac{v^l}{l!}$ is the Taylor series expansion of order m of e^v.

Hence we arrive at

$$B^n = e^{-\beta}B^{n-1} + \gamma_0 u^n(b) + \gamma_1 u^{n-1}(b) + \gamma_2 u^{n-2}(b) + \gamma_3 u^{n-3}(b) + \gamma_4 u^{n-4}(b) \quad (14)$$

where,

$$\gamma_0 = E_0(\beta) - \frac{25}{12}E_1(\beta) + \frac{35}{12}E_2(\beta) - \frac{5}{2}E_3(\beta) + E_4(\beta),$$
$$\gamma_1 = 4E_1(\beta) - \frac{26}{3}E_2(\beta) + 9E_3(\beta) - 4E_4(\beta),$$
$$\gamma_2 = -3E_1(\beta) + \frac{19}{2}E_2(\beta) - 12E_3(\beta) + 6E_4(\beta),$$
$$\gamma_3 = \frac{4}{3}E_1(\beta) + \frac{14}{3}E_2(\beta) + 7E_3(\beta) - 4E_4(\beta),$$
$$\gamma_4 = -\frac{1}{4}E_1(\beta) + \frac{11}{12}E_2(\beta) - \frac{3}{2}E_3(\beta) + E_4(\beta). \quad (15)$$

Likewise, by considering the left boundary $x < a$, we get

$$A^n = e^{-\beta}A^{n-1} + \gamma_0 u^n(a) + \gamma_1 u^{n-1}(a) + \gamma_2 u^{n-2}(a) + \gamma_3 u^{n-3}(a) + \gamma_4 u^{n-4}(a). \quad (16)$$

Now we have equations to compute homogeneous boundary coefficients A^n and B^n to fourth order accuracy. Note that the boundary constants corresponding to operator $\mathcal{D}^{(1)}[u]$ (denoted by A_1^n, B_1^n) are independent of the boundary constants corresponding to operator $\mathcal{D}^{(2)}[u]$ (denoted by A_2^n, B_2^n). Therefore, our fourth order wave solution can be constructed in two levels.

- Level 1
  Compute $\mathcal{D}^{(1)}[u]$ using u; A_1^n and B_1^n are obtained by second order solution implicitly [4] or explicitly,

$$A_1^n = e^{-\beta}A_1^{n-1} + \gamma_0 u^n(a) + \gamma_1 u^{n-1}(a) + \gamma_2 u^{n-2}(a)$$
$$B_1^n = e^{-\beta}B_1^{n-1} + \gamma_0 u^n(b) + \gamma_1 u^{n-1}(b) + \gamma_2 u^{n-2}(b)$$

  where $\gamma_0 = E_0(\beta) - \frac{3}{2}E_1(\beta) + E_2(\beta)$, $\gamma_1 = 2E_1(\beta) - 2E_2(\beta)$, $\gamma_2 = -\frac{1}{2}E_1(\beta) + E_2(\beta)$.

- Level 2
  Compute $\mathcal{D}^{(2)}[u]$ using $D^{(1)}[u]$; A_2^n and B_2^n are obtained by fourth order solution using (14) and (16) explicitly,

$$A_2^n = e^{-\beta}A_2^{n-1} + \gamma_0 u^n(a) + \gamma_1 u^{n-1}(a) + \gamma_2 u^{n-2}(a) + \gamma_3 u^{n-3}(a) + \gamma_4 u^{n-4}(a)$$
$$B_2^n = e^{-\beta}B_2^{n-1} + \gamma_0 u^n(b) + \gamma_1 u^{n-1}(b) + \gamma_2 u^{n-2}(b) + \gamma_3 u^{n-3}(b) + \gamma_4 u^{n-4}(b)$$

Now, we provide an algorithm for fourth order outflow boundary conditions in one dimension (Algorithm 4). This algorithm computes u at time t_{n+1} which works for outflow boundary conditions with fourth order accuracy in time. The algorithm computes γ coefficients (see 15) using second order centered, and fourth order backward finite different stencils in Level 1 and Level 2 computations respectively. For this purpose we use the function *gamma* that computes the γ coefficients for any order based on a given finite difference stencil. The function *LinvOut* defined in Algorithm 5 is used to apply $\mathcal{L}^{-1}$ operator on u^n at a specific computing level k. This algorithm utilizes the function *fastconvol* to compute particular solution, and another function *applyBCoutflow* defined in Algorithm 6 is used to obtain homogeneous boundary coefficients. Initially outflow boundary coefficients (vector H) are set to zero, meaning boundary coefficients at time step t_0 is zero, and previous solutions at boundaries are maintained (ubp_A and ubp_B) for the computation.The function *polyHO* computes the polynomial of coefficients, $A_P(\beta)$ defined in [5].

$$A_p(\beta) = 2\sum_{m=1}^{p}(-1)^m \frac{\beta^{2m}}{(2m)!}\binom{p-1}{m-1}.$$

Algorithm 4 Compute u at time t_{n+1} with fourth order accuracy in time using outflow boundary conditions

function *computeUHOO*($u_n, u_{n-1}, \beta, x_A, x_B, expA, expB, \nu, expweight, H, ubp_A, ubp_B, p_s, p_t$)
u_n, u_{n-1} - solution at time t_n and t_{n-1} respectively,
β - averaging parameter,
x_A, x_B - left and right boundary points respectively,
$expA$ $expB$- exponential vector of grid distance from left and right boundaries respectively,
ν - array of weighted nodes along the line,
$expweight$ - array of exponentially weighted nodes,
H - value of boundary coefficients for each level at time t_n,
ubp_A, ubp_B - solution at the boundary points at previous time steps,
p_s - order of accuracy in space,
p_t - order of accuracy in time.
  $simp_2 = 1 : -1 : 1$ ▷ 2nd order centered FD stencil
  $sexp_4 = 0 : -1 : -4$ ▷ 4th order backward FD stencil
  $g(1,:) = gamma(\beta, 2, simp_2)$
  $g(2,:) = gamma(\beta, 4, sexp_4)$
  $P = \frac{p_t}{2}$ ▷ Number of levels
  $Dterms = 0$
  for $k = 1$ to P **do**
    $D = u - LinvOut(u_n, \beta, x_A, x_B, expA, expB, \nu, expweight, p_s, k, H, ubp_A, ubp_B, g, maxit, tol)$
    $Dterms = Dterms + polyHO(\beta, P)D$
    $u = D$
  Update previous solutions at the boundary points
    for $j = 0$ to p_t-1 **do**
      $ubp_A(j,k) = ubp_A(j+1,k)$ ▷ Left shift
      $ubp_B(j,k) = ubp_B(j+1,k)$
    $ubp_A(p_t,k) = u(x_A)$
    $ubp_B(p_t,k) = u(x_{B)}$
  $u_{n+1} = 2u_n - u_{n-1} + Dterms$

4. DOMAINS WITH COMPLEX GEOMETRIES

We now consider domains with complex geometries such as a model for A6M (we hope to show a simulation of this model in our next paper) that has a set of arch areas join together in 2D. We chose embedded boundary method to

Algorithm 5 Compute $\mathcal{L}^{-1}$ for outflow boundary condition

function *LinvOut*$(u, \beta, x_A, x_B, expA, expB, \nu, expweight, p_s, k, H, ubp_A, ubp_B, g)$
u - solution u at time t_n,
β - averaging parameter,
x_A, x_B - boundary points,
$expA$, $expB$- exponential vector of grid distance from left and right boundaries respectively along the line,
ν - array of weighted nodes along the line,
$expweight$ - array of exponentially weighted nodes along the line,
p_s - order of accuracy in space,
k - current level of computation.
H - boundary coefficients at time step t_n,
ubp_A, ubp_B - solution at the boundary points at previous time steps,
g - a vector of γ coefficients for a given order of accuracy.
Compute particular solution
    $I = fastconvol(u, \nu, expweight, p_s)$
Compute boundary coefficients
    $H = applyBCOutflow(u(x_A), u(x_B), \beta, H, ubp_A, ubp_B, g, k)$
Operate $\mathcal{L}^{-1}$
    $Linv = I + H(k,0)expA + H(k,1)expB$

Algorithm 6 Compute boundary coefficients by applying outflow boundary conditions

function *applyBCOutflow*$(u_A, u_B, \beta, H, ubp_A, ubp_B, g, k)$
u_A, u_B - solutions at boundary points,
H - boundary coefficients at time step t_n,
ubp_A, ubp_B - solution at the boundary points at previous time steps,
g - a vector of γ coefficients for a given order of accuracy,
β - averaging parameter,
k - current level of computation.
    $p = 2 * k$
    $H(k,0) = H(k,0)e^{-\beta} + g(k,0)u_A$
    $H(k,1) = H(k,1)e^{-\beta} + g(k,0)u_B$
    for j = 1 to p **do**
        $H(k,0) = H(k,0) + g(k,j)ubp_A(p-j,k)$
        $H(k,1) = H(k,1) + g(k,j)ubp_B(p-j,k)$

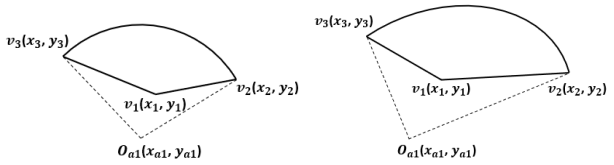
Algorithm 7 Fast Convolution

function *fastconvol*$(u_n, \nu, expweight, p_s)$
u_n - solution at time t_n,
ν - array of weighted nodes along the specific direction,
$expweight$ - array of exponentially weighted nodes along the specific direction,
p_s - order of accuracy in space
    for $j = 1$ to n **do**
        Compute $J_L(j+1)$ and $J_R(j)$ using quadrature defined in [4] and $expweight$ based on p_s.
    $d = e^{-\nu}$
    for $j = 1$ to n **do**
        $I_L(j+1) = d(j)I_L(j) + J_L(j+1)$
        $I_R(n-j+1) = d(n-j+2)I_R(n-j+2) + J_L(n-j+1)$
    for $j = 1$ to n **do**
        $I(j) = \frac{1}{2}\big(I_L(j) + I_R(j)\big)$

solve such complex problems. This paper mainly focuses on varying wave speed over the domains instead of embedding solutions of the domains along their interfaces. The complex geometries in higher dimensional problems may be even harder. In our approach, however, since the higher dimensional problems are solved with ADI schemes, we need to worry about one dimensional lines. Thus, we need to know all relevant properties of each line in order to solve it. However, these lines are going to be broken into several segments due to boundary lines of complex objects. First of all, intersection points (can be felt in off-grid), where the grid line intersects with presenting object boundaries, need to be computed using proper geometrical functions.
We give a detail explanation for 3D complex geometries in this section. In our approach, first we decompose higher dimensional geometry to 2D slices and then represent it using 2D graphs with a set of vertices and edges. The edges can be classified as straight line edges and curves or arches. The graph G can be given as,

$$G(V, E(E_S, E_A)), \tag{17}$$

with edges $E_S \equiv (v_{si}, v_{sj}), \quad and \quad E_A \equiv (O_{ij}, v_{ai}, v_{aj})$
where v_{si}, v_{sj}, v_{ai}, and v_{aj} are the vertices and O_{ij} is center of the arch (v_{ai}, v_{aj}) which may be a circular arch (Figure 5 (a)) or an elliptical arch (Figure 5 (b)). In Figure 5, each object has three vertices (v_1, v_2, v_3), two straight edges $e_{s1}(\equiv (v_1, v_2)), e_{s2}(\equiv (v_3, v_1))$, and one arch edge $e_{a1}(\equiv (v_2, v_3, O_{a1}))$ that can be represented as a graph $G([v_1, v_2, v_3], [[e_{s1}, e_{s2}], [e_{a1}])])$



(a) with circular arch (b) with elliptical arc

FIG. 5: 2D graph with 2 straight edges and 1 circular arch (a) or elliptical arch (b) edge

4.1. Pre-computing

Before begining the PDE evolution, we need to perform a pre-computation to identify key characteristics of the geometry such as boundary and relevant parameter values along the grid lines in each direction. The flow diagram in Figure 6 shows major tasks performed in pre-computation and results obtained at the end of each task.

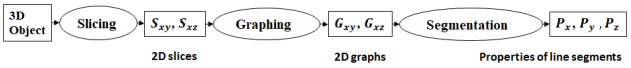


FIG. 6: Flow diagram of pre-computation for 3D problems with complex geometries

1. **Slicing**
   This task decomposes a given 3D object into a set of 2D objects by slicing along the grid lines. We can form three set of 2D slices S_{yz}, S_{xz}, and S_{xy} in each direction x, y, and z respectively, but two sets are sufficient for our computation (if the object is uniform along any direction, we need to process slicing in that direction only - yielding a set of slices). Suppose we apply slicing along z and y directions, we will have a set of xy slices, S_{xy} placed along z with distance Δz apart and a set of xz slices, S_{xz} placed along y with distance Δy

2. **Graphing**
   This task defines the geometry of each 2D slice using 2D graphs as expressed in equation 17. We have two sets of 2D slices, S_{xy} of size n_z and S_{xz} of size n_y where n_z and n_y are number of spatial grid points along the z and y directions respectively. Each slice should have at least one 2D object. Suppose we assume a 3D object has only convex surfaces along primary directions x, y, and z, then 2D objects in the slices will be defined by

$$\begin{aligned} G_{xy}(k), &\quad k = 1, 2, ...n_z \\ G_{xz}(j), &\quad j = 1, 2, ...n_y \end{aligned}$$

   Therefore, we will obtain, $n_z + n_y$ graphs.

3. **Segmentation**
   This task generates line segments along each grid line in each direction x, y, and z using intersections between the grid lines and surface of the object. The segments along x and y can be computed using the graph G_{xy} and along x and z can be computed using the graph G_{xz}, but we should avoid unneeded duplicate computations for x in order to reduce computing cost. In the final stage of the pre-computation all relevant parameters/properties P_x of size $n_x \times n_{sx}$, P_y of size $n_y \times n_{sy}$ and P_z of size $n_z \times n_{sz}$ for each line segment will be computed. Here n_{sx}, n_{sy} and n_{sz} are number of line segments along x, y, and z respectively. Since, however, each line in the same direction does not need to have the same amount of segments, n_{sx}, n_{sy} and n_{sz} are not single variables, they are vectors/arrays to keep track of the number of segments in each line. Now, the sizes of P_x, P_y, and P_z are $\sum_{i=1}^{n_x} n_{sx}(i)$, $\sum_{j=1}^{n_y} n_{sy}(j)$, and $\sum_{k=1}^{n_z} n_{sz}(k)$ respectively.

After we are done with the pre-compution, we will have three set of line segments with their properties (P_x, P_y, and P_z) for each direction. These segments will be utilized by our higher order solver to obtain a solution with higher order accuracy. Actually, during each sweep, each line segment is treated individually with their own piecewise constant wave speed, and then the 3D higher order equation 11 is applied to compute the solution u^{n+1} at next time step t_{n+1}.

5. NUMERICAL RESULTS

We present a set of test cases that evaluates our framework; specifically we consider problems with variable/piecewise constant wave speed in one, two, and three dimensions, using a Gaussian pulse as an initial source or a point source. For 3D, we obtain fourth order convergent graphs for Dirichlet boundary conditions.

5.1. Waves with variable speeds

5.1.1. One Dimensional Case

A one dimensional wave travels through two different consecutive domains, $\Omega_1 \in [-1, 0]$ and $\Omega_2 \in [0, 1]$ with piecewise constant speed $c_1 = 1.0$ and $c_2 = 2.0$ respectively. The solution until the wave reaches the location of discontinuity, $x = 0$ is,

$$u(x, t) = a_i e^{-\eta(x - c_1(t - t_0))^2} \tag{18}$$

where a_i is the initial amplitude, η is the Gaussian shape parameter, and t_0 is a time offset which delays the Gaussian pulse.
A portion of the travelling wave in domain one, Ω_1 will be reflected at the location of interface, $x = 0$, while the remaining waves are transmitted to domain two, Ω_2. Thus, we need to consider the combination of transmitted, reflected, and incident waves from the location of interface, $x = 0$ and onward. The left domain (Ω_1) will have components of incident and reflected waves, and the right domain (Ω_2) will have the transmitted component of the original wave.
In this way, the solution obeys the following equations,

$$\begin{aligned} u_1(t) &= a_i e^{\eta(x - c_1(t - t_0))^2} + a_r e^{\eta(x + c_1(t - t_0))^2} \quad x < 0 \\ u_2(t) &= a_t e^{\bar{\eta}(x - c_2(t - t_0))^2} \quad x \geq 0, \end{aligned}$$

where, a_r and a_t are amplitudes of the reflected and transmitted wave respectively and $\bar{\eta}$ is the shape parameter of the transmitted Gaussian pulse. Because of the zero transverse

displacement at the interface, $u_1(0,t) = u_2(0,t)$, that gives us,

$$(a_i + a_r)e^{\eta c_1^2 (t-t_0)^2} = a_t e^{\bar{\eta} c_2^2 (t-t_0)^2}.$$

By equating the coefficients we get,

$$a_i + a_r = a_t \quad and \quad \eta c_1^2 = \bar{\eta} c_2^2. \tag{19}$$

From energy conservation (energy of the incident wave should be equal to sum of the energy of reflected and transmitted waves), we have

$$\sqrt{\eta} a_i^2 = \sqrt{\eta} a_r^2 + \sqrt{\bar{\eta}} a_t^2. \tag{20}$$

Upon solving equations (19), and (20), we obtain

$$a_t = \frac{2a_i c_2}{(c_1 + c_2)}, \quad a_r = a_i - a_t.$$

For our first numerical example, we choose $a_i = 1$, so $a_t = \frac{2c_2}{c_1+c_2}$, and $a_r = 1 - a_t$. Further, we chose $c_1 = 1.0$ and $c_2 = 2.0$, and apply our solver along the domain $\Omega_1 \cup \Omega_2$ with 1024 uniform grid points of size $\Delta x = 0.002$, while maintaining a CFL of 2.5 by choosing time step $\Delta t = 0.005$. Figure 7 shows time snapshots of the moving wave using outflow boundary conditions at the left and right boundaries. The numerical results agree closely with the theoretical solutions as can be seen in the figure.

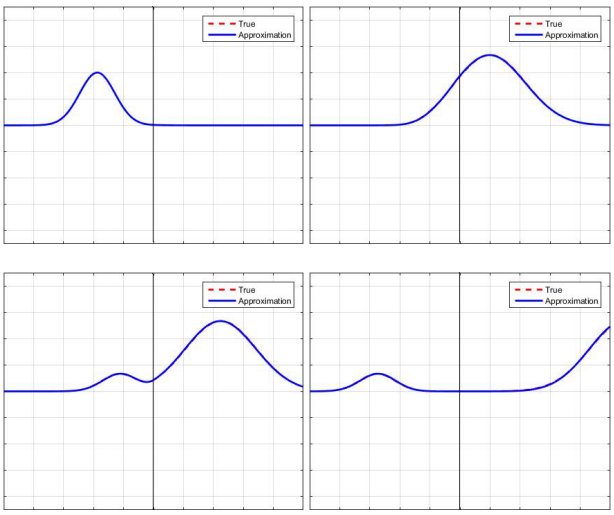


FIG. 7: Piecewise constant speed ($c_1 = 1.0$ and $c_2 = 2.0$) wave in 1D

5.1.2. *Two Dimensional Case*

In this section, we consider the two dimensional variable speed solver. A square domain with a square patch is chosen to assess the two dimensional variable speed solver. Due to the material properties of the patch, waves travel with different speeds through the patch than in the remaining area. We choose a Gaussian pulse $e^{-25(x^2+y^2)}$ as an initial solution and apply outflow boundary condition along the border of the square domain ($\Omega = [-1,1] \times [-1,1]$). Since we use spatial grid size $\Delta x = \Delta y = 0.02$ and time step size $\Delta t = 0.005$, applicable CFL is 0.25. We demonstrate a test case with one patch placed as shown in the Figure 8

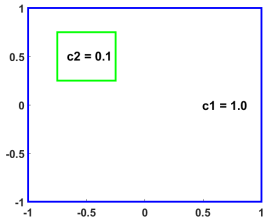


FIG. 8: Geometrical view of one square patch in the square domain

A 0.5 x 0.5 square patch is placed at the left top corner centered at, (-0.5, 0.5). The wave speed is set to be $c_2(= 0.1)$ on the patch, and $c_1(= 1.0)$ in the remaining area. Figure 9 shows time snapshots of the solution

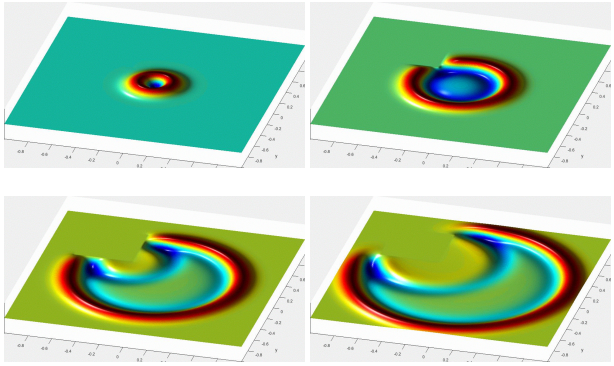
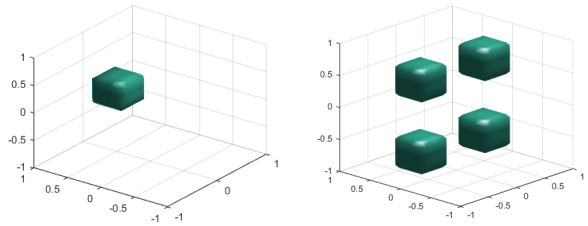


FIG. 9: Time evolution of a Gaussian field with a square patch.

5.1.3. *Three Dimensional Case*

In this section, we discuss the three dimensional variable speed wave solver. A cubic domain with cubic patches is chosen to assess the three dimensional variable speed solver. Due to the material property of patches, waves travel with different speed through these patches than the remaining area. We place a Gaussian pulse $e^{-36(x^2+y^2+z^2)}$ as an initial solution and apply outflow

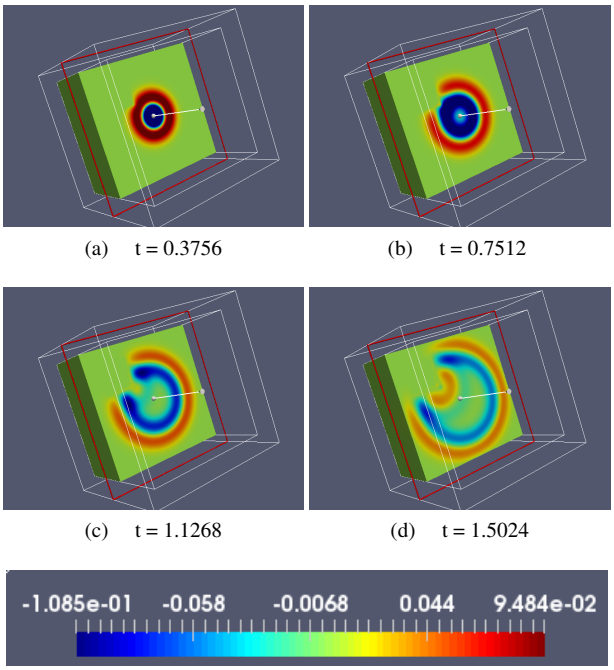
boundary conditions along the border of the cubic domain ($\Omega = [-1, 1] \times [-1, 1] \times [-1, 1]$). Since we use spatial grid size $\Delta x = \Delta y = \Delta z = 0.0625$ ($32 \times 32 \times 32$ grid points) and time step size $\Delta t = 0.0313$, applicable CFL is 0.5. We demonstrate two test cases: one, and four patches for case-i and case-ii respectively as shown in the Figure 10



(a) Single patch (b) 4 patches

FIG. 10: Geometrical view of (a) one and (b) four square patches in the square domain

Case-i - Single patch
A $0.5 \times 0.5 \times 0.5$ cubic patch is placed at the left top corner centered at, (-0.5, 0, 0.5). The wave speed is set to be $c_2(= 0.1)$ on the patch, and $c_1(= 1.0)$ in the remaining area. Figure 11 shows snapshots of the wave at different time instants

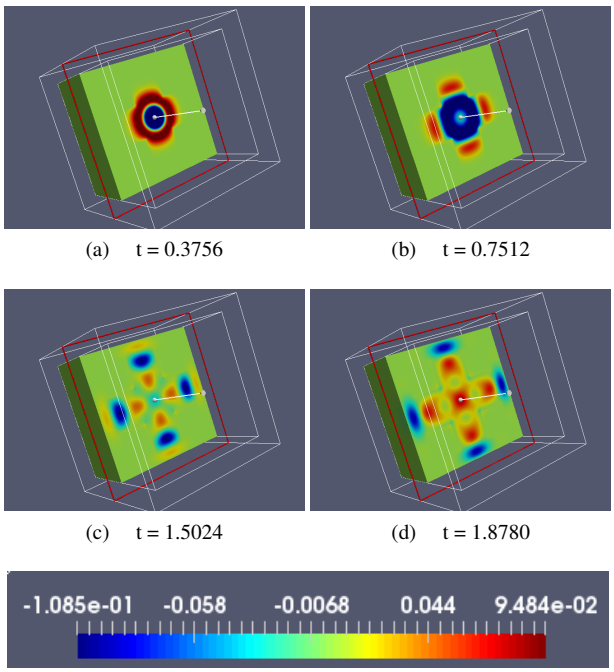


(a) t = 0.3756 (b) t = 0.7512

(c) t = 1.1268 (d) t = 1.5024

FIG. 11: Time evolution of a Gaussian field with a cubic patch.

Case-ii - Four patches
There are four $0.5 \times 0.5 \times 0.5$ cubic patches are placed at the four corners; i.e., centered at (-0.5, 0, 0.5), (-0.5, 0, -0.5), (0.5, 0, 0.5), and (0.5, 0, -0.5). The wave speed is set to be $c_2(= 0.1)$ on every patch, and $c_1(= 1.0)$ in the remaining area. Figure 12 shows snapshots of the wave at different time instants.



(a) t = 0.3756 (b) t = 0.7512

(c) t = 1.5024 (d) t = 1.8780

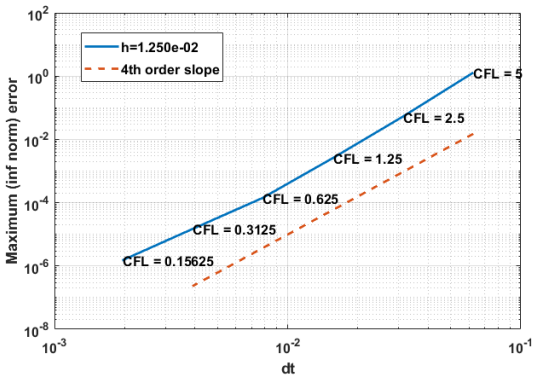
FIG. 12: Time evolution of a Gaussian field with four cubic patches.

5.2. Refinement Studies

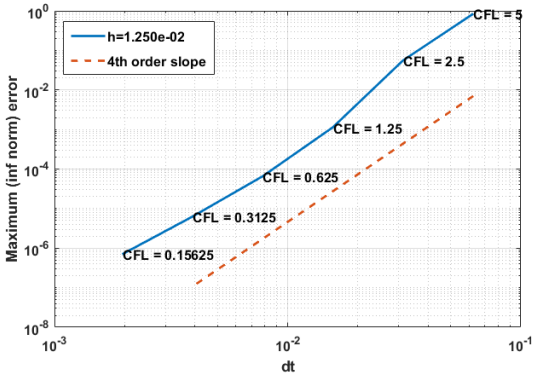
We show fourth order convergence by performing a refinement study on a cubic domain $\Omega = [-1, 1] \times [-1, 1] \times [-1, 1]$ with a point source $cos(\omega t)$ at center of the domain, $(0, 0, 0)$. This runs up to time $T = 2.0$, with a fixed spatial resolution of $160 \times 160 \times 160$ spatial points. The discrete L^2 norm of the error is constructed at each time step and maximum error over all time step is used to graph Figure 13 for $\Delta t = \frac{6.25}{2^k}$, $k = 1$ to 5, with Dirichlet boundary conditions.

6. CONCLUSIONS

In this paper, we have proposed a Higher order 3D scheme for wave propagation which deals with complex geometries. This A-stable implicit scheme is developed based on MOL^T approach and it utilizes a fast convolution recursive algorithm [4] which consumes $O(N)$ operations for a line segment with N grid points. We evaluated the performance of the fourth order 3D scheme using wave



(a) $\omega = 0.1$



(b) $\omega = 1$

FIG. 13: Fourth order convergence of 3D wave solver using Dirichlet boundary conditions with $\Delta x = \Delta y = \Delta z = 1.25 \times 10^{-2}$ for (a) $\omega = 0.1$ and (b) $\omega = 1$

guides on cubic patches and refinement studies for convergent tests. We are focusing with higher order embedded Neumann boundary scheme for complicated geometries as our immediate following work, and multi-scale adeptness of this scheme should support us to bring it on multi-core architectures such as GP^2Us which is our next target.

References

[1] Ascher, U. and Mattheij, R. and Russell, R., *Numerical Solution of Boundary Value Problems for Ordinary Differential Equations*, Society for Industrial and Applied Mathematics, (1995).

[2] R. Coifman and V. Rokhlin and S. Wandzura, *The fast multipole method for the wave equation: a pedestrian prescription*, IEEE Antennas and Propagation Magazine **35**, 3, pp. 7-12 (1993).

[3] Annamaria Mazzia and Francesca Mazzia, *High-order transverse schemes for the numerical solution of PDEs*, Journal of Computational and Applied Mathematics, **82**, 1 , pp. 299 - 311, (1997).

[4] Matthew F. Causley and Andrew J. Christlieb, and Ong, B. and VanGroningen, L., *Method of Lines Transpose: An Implicit Solution to the Wave Equation*, Mathematics of Computation. **83**, 290, pp. 2763-2786 (2014).

[5] Matthew F. Causley and Andrew J. Christlieb, *Higher Order A-Stable Schemes for the Wave Equation Using a Successive Convolution Approach*, SIAM Journal on Numerical Analysis. **52**, 1, pp. 220-235 (2014).

[6] Matthew F. Causley and Andrew J. Christlieb, and Wolf, Eric *Method of Lines Transpose: An Efficient Unconditionally Stable Solver for Wave Propagation*, Journal of Scientific Computing. **70**, 2, pp. 896-921 (2017).