Numerical Simulation of Velocity-stress Equation in 3D Based on Staggered-Grid Finite Difference

学生姓名 王文强 学号 11849528

指导教师 张振国

院　　系 地球与空间科学系

年　　级 2018 级博士研究生

Abstract：

Staggered-grid finite difference has double precision while keeping computational and memory constant. In this paper, the staggered-grid finite difference is used to discretize the elastic wave equation. Then CPU and GPU are used to realize the numerical simulation of the elastic wave equation, and we compare the accuracy of CPU and GPU. Due to the large number of variables of speed and stress, the 3D implementation will consume a lot of memory. Therefore, it is necessary to implement the PML absorption boundary by optimizing the memory. By establishing a random medium model, we apply this velocity model to the elastic wave forward modeling.

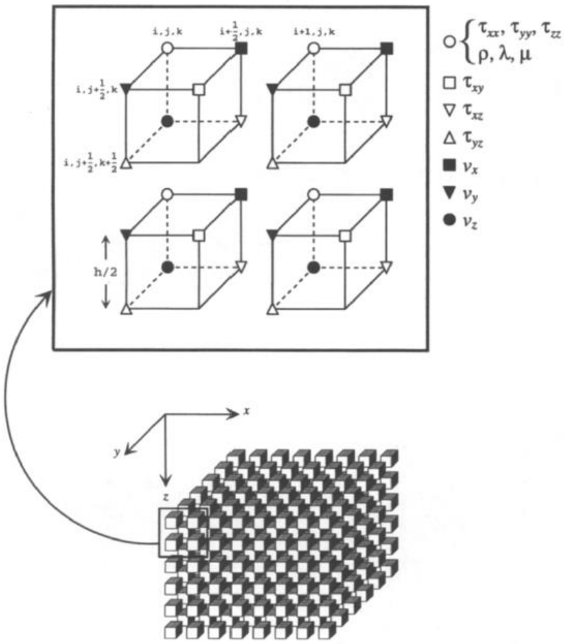
Keywords：Velocity-stress Equation, 3D, Staggered-Grid, Finite Difference, GPU

Algorithm implementation：

1. Velocity-stress Equation:

$$\begin{cases} \rho \dfrac{\partial V_x}{\partial t} = \dfrac{\partial \sigma_{xx}}{\partial x} + \dfrac{\partial \tau_{xy}}{\partial y} + \dfrac{\partial \tau_{xz}}{\partial z} + f_x \\ \rho \dfrac{\partial V_y}{\partial t} = \dfrac{\partial \tau_{xy}}{\partial x} + \dfrac{\partial \sigma_{yy}}{\partial y} + \dfrac{\partial \tau_{yz}}{\partial z} + f_y \\ \rho \dfrac{\partial V_z}{\partial t} = \dfrac{\partial \tau_{xz}}{\partial x} + \dfrac{\partial \tau_{yz}}{\partial y} + \dfrac{\partial \sigma_{zz}}{\partial z} + f_z \end{cases} \tag{1}$$

$$\begin{cases} \dfrac{\partial \sigma_{xx}}{\partial t} = (\lambda + 2\mu)\dfrac{\partial V_x}{\partial x} + \lambda(\dfrac{\partial V_y}{\partial y} + \dfrac{\partial V_z}{\partial z}) \\ \dfrac{\partial \sigma_{yy}}{\partial t} = (\lambda + 2\mu)\dfrac{\partial V_y}{\partial y} + \lambda(\dfrac{\partial V_x}{\partial x} + \dfrac{\partial V_z}{\partial z}) \\ \dfrac{\partial \sigma_{zz}}{\partial t} = (\lambda + 2\mu)\dfrac{\partial V_z}{\partial z} + \lambda(\dfrac{\partial V_x}{\partial x} + \dfrac{\partial V_y}{\partial y}) \\ \dfrac{\partial \tau_{xy}}{\partial t} = \mu(\dfrac{\partial V_x}{\partial y} + \dfrac{\partial V_y}{\partial x}) \\ \dfrac{\partial \tau_{xz}}{\partial t} = \mu(\dfrac{\partial V_x}{\partial z} + \dfrac{\partial V_z}{\partial x}) \\ \dfrac{\partial \tau_{yz}}{\partial t} = \mu(\dfrac{\partial V_y}{\partial z} + \dfrac{\partial V_z}{\partial y}) \end{cases} \tag{2}$$



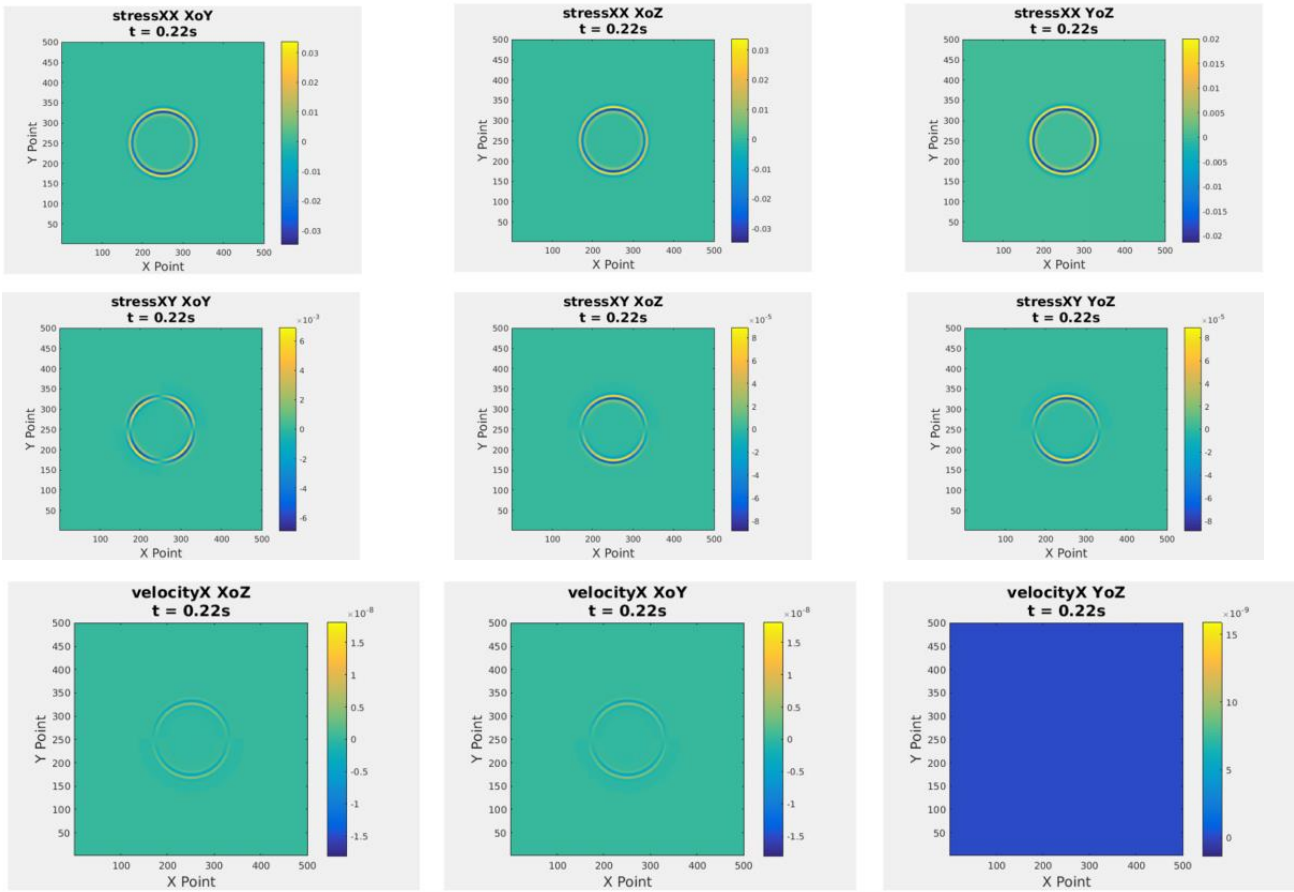
2. Staggered-Grid Finite Difference:

As is shown in the picture, all variables are distributed on the grid points. Then we can obtain the following finite difference equation from the above equation.

$$\begin{cases} \rho V_{x}{}_{i+\frac{1}{2},j,k}^{n+1/2} = \rho V_{x}{}_{i+\frac{1}{2},j,k}^{n-1/2} + \Delta t[D_x\tau_{xx} + D_y\tau_{xy} + D_z\tau_{xz} + f_x]_{i+\frac{1}{2},j,k}^{n} \\ \rho V_{y}{}_{i,j+\frac{1}{2},k}^{n+1/2} = \rho V_{y}{}_{i,j+\frac{1}{2},k}^{n-1/2} + \Delta t[D_x\tau_{xy} + D_y\tau_{yy} + D_z\tau_{yz} + f_y]_{i,j+\frac{1}{2},k}^{n} \\ \rho V_{z}{}_{i,j,k+\frac{1}{2}}^{n+1/2} = \rho V_{z}{}_{i,j+\frac{1}{2},k}^{n-1/2} + \Delta t[D_x\tau_{xz} + D_y\tau_{yz} + D_z\tau_{zz} + f_z]_{i,j,k+\frac{1}{2}}^{n} \end{cases}$$

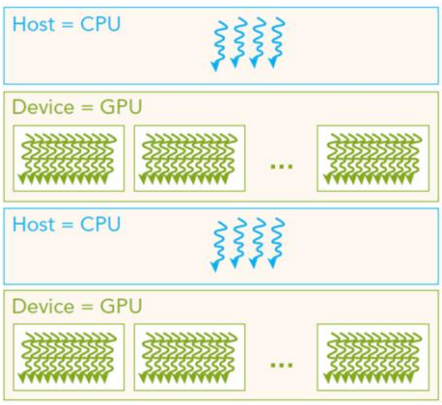
$$\begin{cases} \sigma_{xx}{}_{i,j,k}^{n+1} = \sigma_{xx}{}_{i,j,k}^{n-1} + \Delta t[(\lambda + 2\mu)D_xV_x + \lambda(D_yV_y + D_zV_z)]_{i,j,k}^{n+1/2} \\ \sigma_{yy}{}_{i,j,k}^{n+1} = \sigma_{yy}{}_{i,j,k}^{n-1} + \Delta t[(\lambda + 2\mu)D_yV_y + \lambda(D_xV_x + D_zV_z)]_{i,j,k}^{n+1/2} \\ \sigma_{zz}{}_{i,j,k}^{n+1} = \sigma_{zz}{}_{i,j,k}^{n-1} + \Delta t[(\lambda + 2\mu)D_zV_z + \lambda(D_xV_x + D_yV_y)]_{i,j,k}^{n+1/2} \\ \tau_{xy}{}_{i+\frac{1}{2},j+1/2,k}^{n+1} = \tau_{xy}{}_{i+\frac{1}{2},j+1/2,k}^{n+1} + \Delta t\mu(D_yV_x + D_yV_x)_{i+\frac{1}{2},j+1/2,k}^{n+1} \\ \tau_{xz}{}_{i+\frac{1}{2},j,k+1/2}^{n+1} = \tau_{xy}{}_{i+\frac{1}{2},j,k+1/2}^{n+1} + \Delta t\mu(D_zV_x + D_xV_z)_{i+\frac{1}{2},j,k+1/2}^{n+1} \\ \tau_{yz}{}_{i,j+1/2,k+\frac{1}{2}}^{n+1} = \tau_{yz}{}_{i,j+1/2,k+\frac{1}{2}}^{n+1} + \Delta t\mu(D_zV_y + D_yV_z)_{i,j+1/2,k+\frac{1}{2}}^{n+1} \end{cases}$$

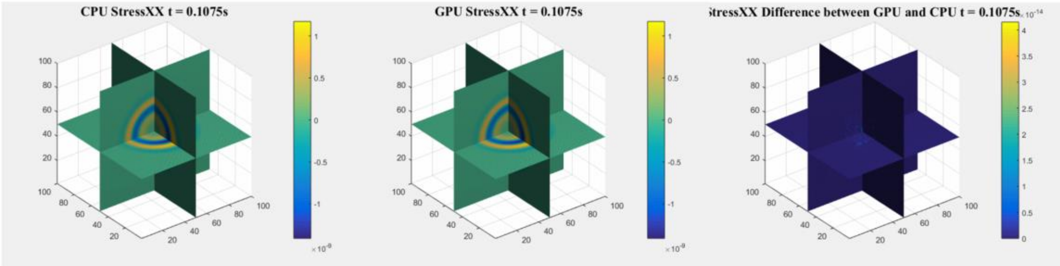
Firstly, we use C++ codes to implement the staggered-grid finite difference equation, and we get the snapshot in the following pictures.



3. Implement of CPU and GPU

CUDA is a set of parallel programming framework provided by NVIDIA. We can transfer the CPU data to the GPU, accelerate it in parallel in the GPU, and then retransmit the data back to CPU. CUDA C is a C-like language. NVIDIA has already mapped it to hardware. We don't need to think too much about hardware details when programming. More attention is paid to the algorithm logic of parallel programs.





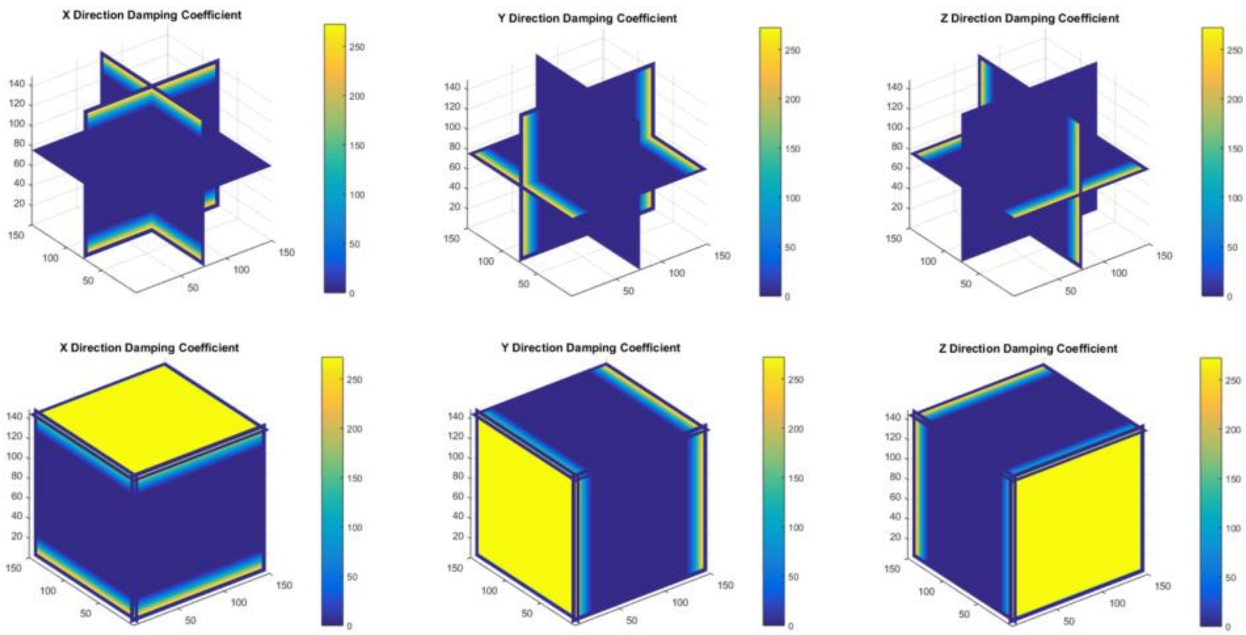
As is shown in the above picture, the difference between GPU and CPU is small enough to omit.

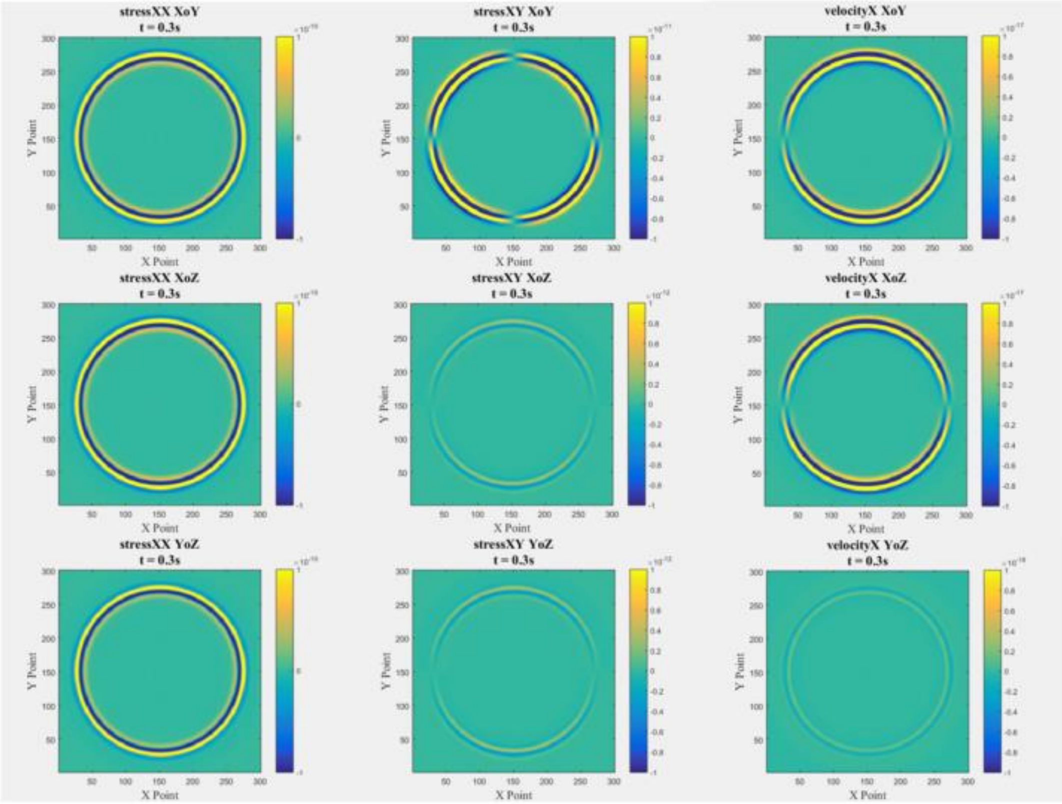
When we calculate a grid model 500*500*500, the time loss is about 7 hours on the CPU and about 100 seconds on the GPU. As the time loss on CPU is very big, we didn't calculate so many models. We calculate several modes on GPU, and the time loss is in the following table.

Grid	500 * 500 * 500	500 * 500 * 400	500 * 400 * 400	400 * 400 * 400	400 * 400 * 300	400 * 300 * 300	300 * 300 * 300	300 * 300 * 200	300 * 200 * 200	200 * 200 * 200	100 * 100 * 100
Time Loss (s)	109	89	73	61	47	37	31	23	18	14	8

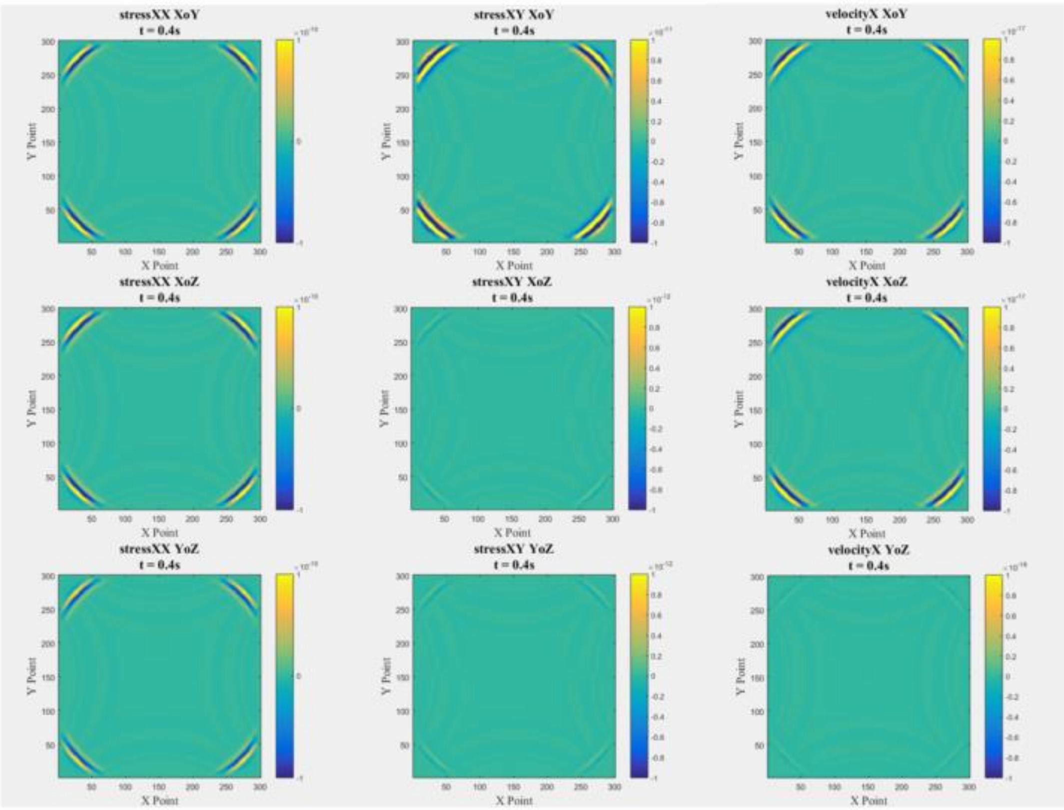
It's obvious that the GPU performs much well than CPU.

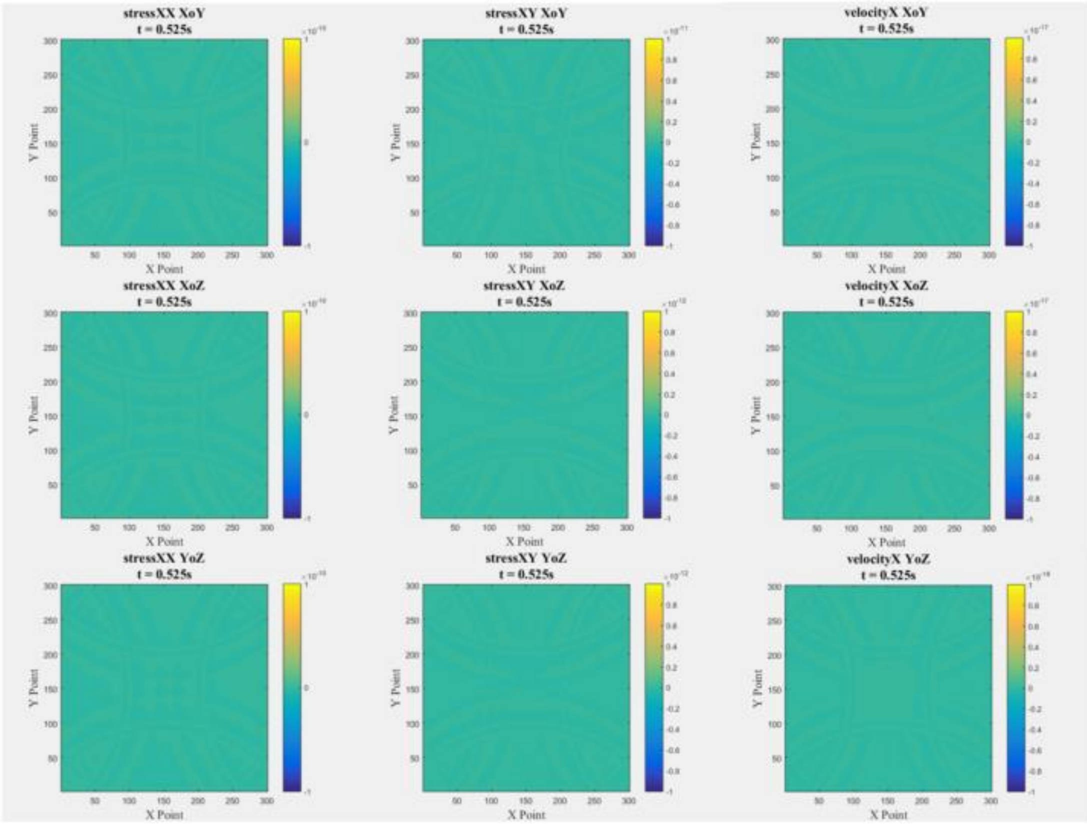
4. PML

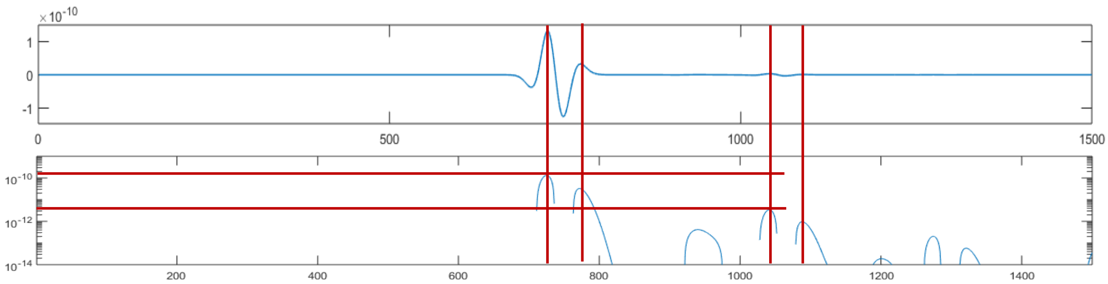




If we do not add artificial absorption boundaries, we will generate pseudo-reflected waves.

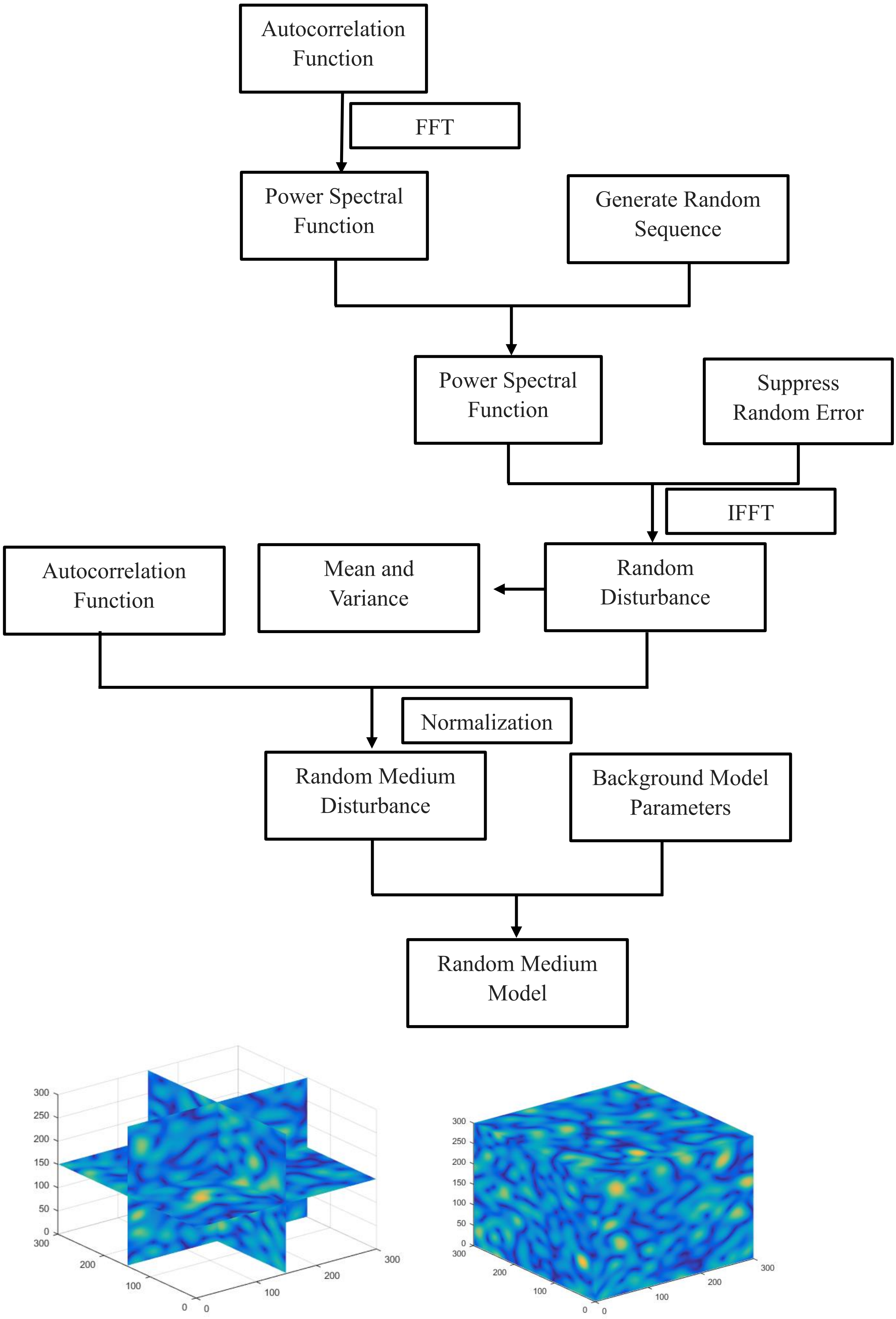


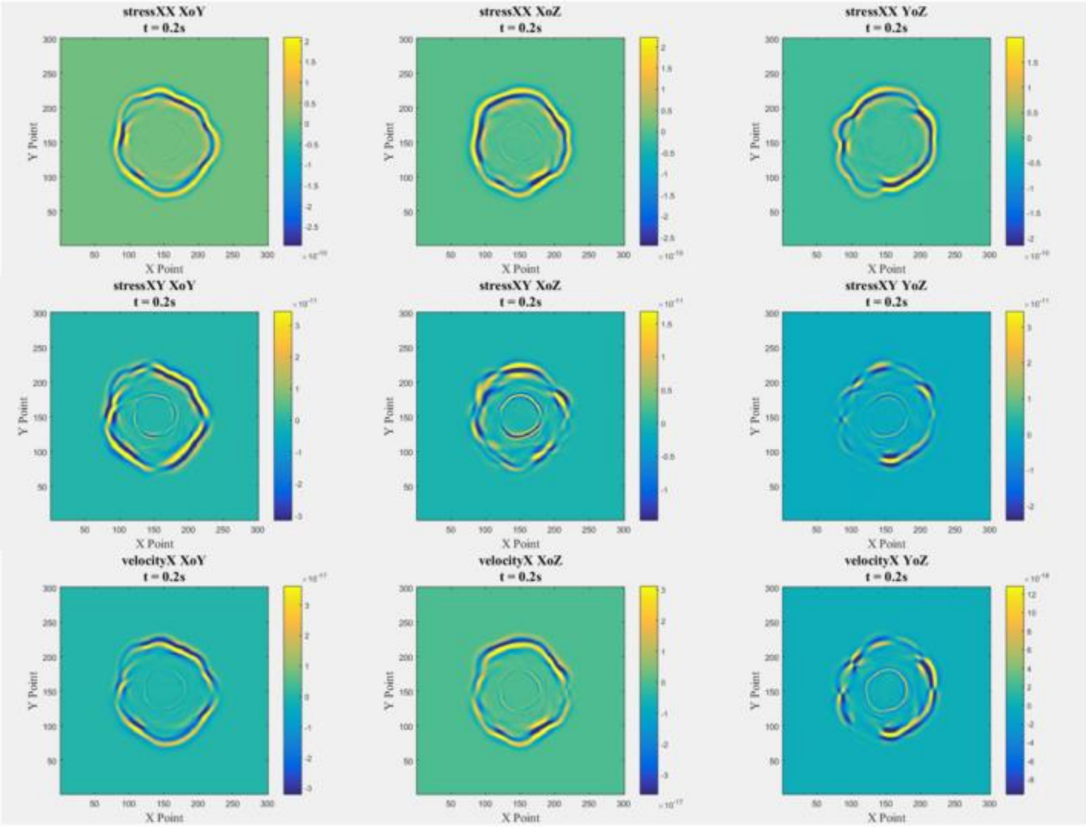


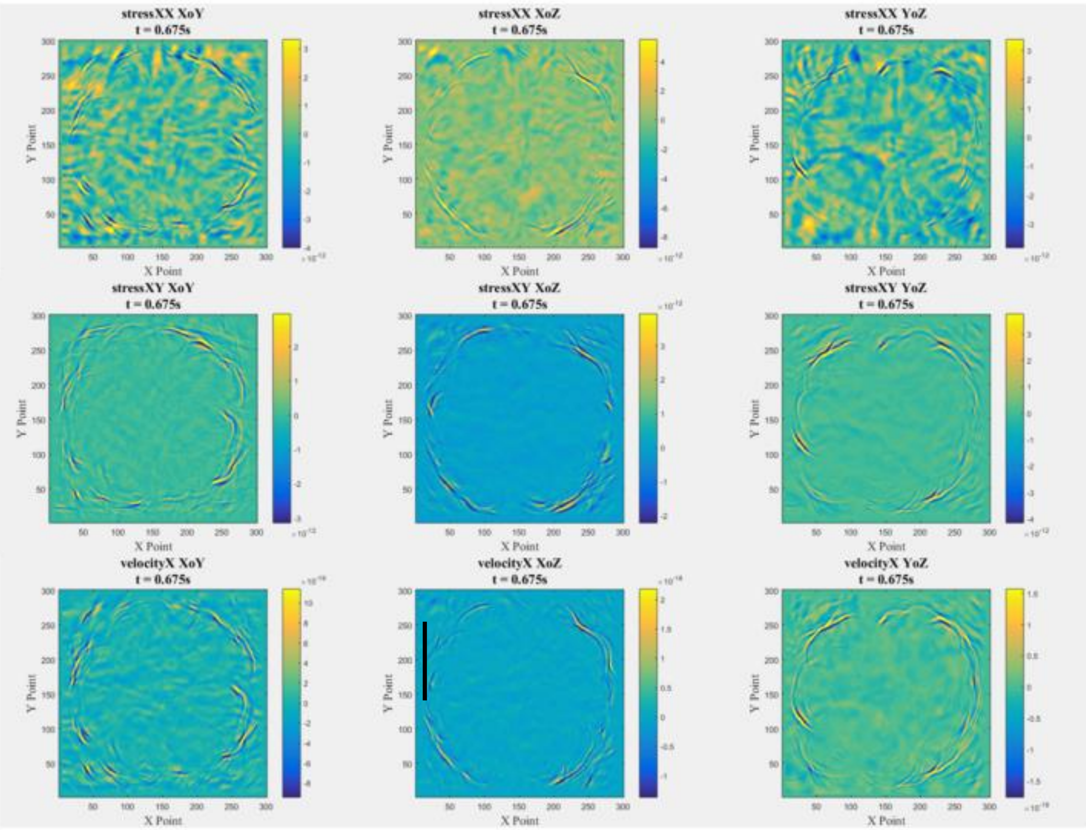


5 Random Medium

We establish a velocity model based on random medium theory. Then, we use the velocity model for numerical simulation of elastic equations. The algorithm flow chart is show below:


Autocorrelation Function
FFT
Power Spectral Function
Generate Random Sequence
Power Spectral Function
Suppress Random Error
IFFT
Autocorrelation Function
Mean and Variance
Random Disturbance
Normalization
Random Medium Disturbance
Background Model Parameters
Random Medium Model


stressXX XoY
t = 0.2s
stressXX XoZ
t = 0.2s
stressXX YoZ
t = 0.2s
stressXY XoY
t = 0.2s
stressXY XoZ
t = 0.2s
stressXY YoZ
t = 0.2s
velocityX XoY
t = 0.2s
velocityX XoZ
t = 0.2s
velocityX YoZ
t = 0.2s
X Point
Y Point


stressXX XoY
t = 0.675s
stressXX XoZ
t = 0.675s
stressXX YoZ
t = 0.675s
stressXY XoY
t = 0.675s
stressXY XoZ
t = 0.675s
stressXY YoZ
t = 0.675s
velocityX XoY
t = 0.675s
velocityX XoZ
t = 0.675s
velocityX YoZ
t = 0.675s
X Point
Y Point