MPM-Geomechanics Manual (v0.1)

MPM-Geomechanics: An open-source Material Point Method code for geomechanics.

Dr. Fabricio Fernández

November 3, 2025

1 Introduction to the Material Point Method (MPM)

The Material Point Method is an hybrid Lagrangian-Eulerian numerical method, that allows for simulating continuum mechanics processes involving large deformations and displacements without issues related to computational mesh distortion. In MPM, the material domain to be simulated is discretized into a set of material points that can move freely within a computational mesh, where the equations of motion are solved. The material points store all variables of interest during the simulation, such as stress, pore pressure, temperature, etc., giving the method its Lagrangian characteristic. In an MPM computational cycle, all variables stored in the material points are computed at the computational mesh nodes using interpolation functions, and then the equation of motion is solved at the nodes. The nodal solution obtained is interpolated back to the particles, whose positions are updated, and all nodal variables are discarded. This method, enables the numerical solution of the motion equation in continuum mechanics by using the nodes of an Eulerian mesh for integration and Lagrangian material points to transfer and store the properties of the medium.

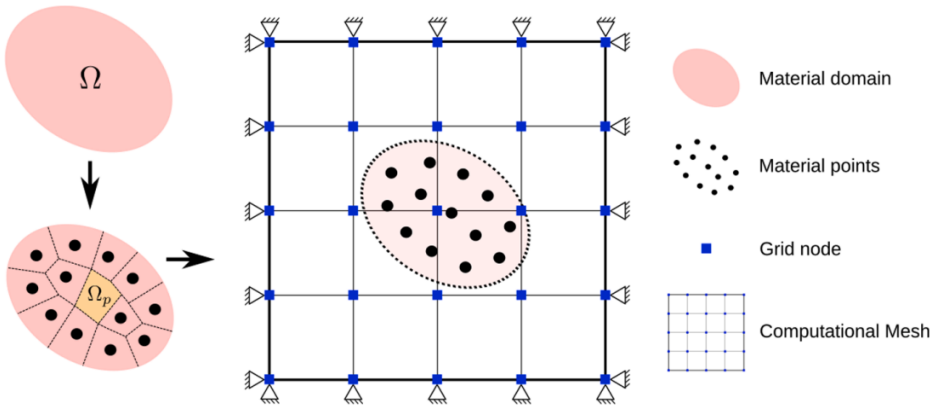

Figure 1: Eulerian and Lagrangian discretization

2 MPM Formulation

In continuum mechanics, the 3D motion equation of movement is

$$\frac{\partial \sigma_{ij}}{\partial x_j} + \rho b_i = \rho a_i$$

or

$$\frac{\partial \sigma_{ij}}{\rho \partial x_j} + b_i = a_i$$

The internal forces are related with the σ_{ij}, the Cauchy stress tensor, and ρ is the density of the material, b_i is a mass field force and a_i is the acceleration.

This equations are presented in tensor notation, but, vector and matrix can be used as notation in the formulation.

The discrete form of the motion equation, can be obtained using finite difference methods or by using a weak form, such as used by the finite element method, or the material point method.

The weak form is obtained multiplyng the motion equation by arbitrary weighting functions and integrating this product over the domain. In this procedure the integration by parts reduces the order the stress tensor and introduces the natural boundary conditions.

$$-\int_\Omega \sigma_{ij}\delta u_{i,j}dV + \int_\Gamma t_i \delta u_i dA + \int_\Omega \rho b_i \delta u_i dV = \int_\Omega \rho a_i \delta u_i dV$$

Here δu_i are arbitrary displacements functions, whose value in the boundary are $\delta u_i|_\Gamma = 0$ and t_i is an external traction acting on the boundary Γ.

In the MPM context any field or space property $f(x_i)$ can approximated using the value stored in the particle f_p:

$$f(x_i) = \sum f_p \chi_p(x_i)$$

where χ_p is the particle characteristic function that defines the volume occupied by the material point:

$$V_p = \int_{\Omega_p \cap \Omega} \chi_p(x_i) dV$$

In consequent, density, acceleration and stress fields can be approximated by the values stored in particles:

$$\rho(x_i) = \sum_p \frac{m_p}{V_{ip}} \chi_p(x_i)\rho(x_i)a_i(x_i) = \sum_p \frac{\dot{p_{ip}}}{V_p}\chi_p(x_i)\sigma_{ij}(x_i) = \sum_p \sigma_{ijp}\chi_p(x_i)$$

where $\dot{p_{ip}} = m_p \dot{v}_{ip} = m_p a_{ip} = f_{ip}$ is the momentum variation in time that is equal to the total force, regarding the second Newton's law.

Replacing these fields in the weak form of the motion equation we have:

$$-\sum_p \int_{\Omega_p \cap \Omega} \sigma_{ijp}\chi_p \delta u_{i,j} dV + \int_\Gamma t_i \delta u_i dA + \sum_p \int_{\Omega_p \cap \Omega} \frac{m_p}{V_p} b_{ip}\chi_p \delta u_i dV = \sum_p \int_{\Omega_p \cap \Omega} \frac{\dot{p}_p}{V_p}\chi_p a_i dV$$

In the generalized interpolation material point method (GIMP), the resolution of this equation is carried out using a Petrov–Galerkin scheme where the characteristic functions $\chi_p(x_i)$ are the trial functions and the nodal interpolation functions $N_I(x_i)$ are the test functions.To arrive at this scheme, the virtual displacements are expressed using nodal interpolation functions:

$$\delta u_i = \sum_I N_{Ip}\delta u_{iI}$$

The trial and test functions are such that:

$$\sum_I N_I(x_i) = 1 \sum_p \chi_p(x_i) = 1$$

The resulting discrete form of the motion equation then is:

$$f_{iI}^{int} + f_{iI}^{ext} = \dot{p}_{iI}$$

where

$$p_{iI} = \sum_p S_{Ip} p_{Ip}$$

is the nodal momentum,

$$f_{iI}^{int} = -\sum_p \sigma_{ijp} S_{Ip,j} V_p$$

is the nodal internal force, and

$$f_{iI}^{ext} = \sum_p m_p S_{Ip} b_{ip} + \int_\Gamma t_i N_I(x_i) dA$$

is the external force at node I.

The function S_{Ip} and its gradients $S_{Ip,j}$ are the weighting functions of node I evaluated at the position of particle p.

The GIMP shape functions are defined by

$$S_{Ip} = \frac{1}{V_p} \int_{\Omega_p \cap \Omega} \chi_p(x_i) N_I(x_i) dV$$

and

$$S_{Ip,j} = \frac{1}{V_p} \int_{\Omega_p \cap \Omega} \chi_p(x_i) N_{I,j}(x_i) dV$$

These two functions are also a partition of the unity $\sum_I S_{Ip} = 1$.

The the weighting function need to be integrated over the particle domain by choosing different characteristic functions and interpolation functions in a Petrov–Galerkin scheme.

In the contiguous particle GIMP (cpGIMP) the characteristic function in defined as step function and the interpolation function is defined as linear function:

$$\chi_p(x) = \begin{cases} 1, & x \in \Omega_p, \\ 0, & x \notin \Omega_p. \end{cases}$$

$$N_I(x) = \begin{cases} 0, & |x - x_I| \geq L, \\ 1 + \dfrac{x - x_I}{L}, & -L < x - x_I \leq 0, \\ 1 - \dfrac{x - x_I}{L}, & 0 < x - x_I < L. \end{cases}$$

Where the integration is performed analytically within the particle domain.

$$S_{Ip} = \begin{cases} 0 & |\xi| \geq L + l_p \\ \left(L + l_p + \xi\right)^2 / 4 L l_p & -L - l_p < \xi \leq -L + l_p \\ 1 + \xi / L & -L + l_p < \xi \leq -l_p \\ 1 - \left(\xi^2 + l_p^2\right) / 2 L l_p & -l_p < \xi \leq l_p \\ 1 - \xi / L & l_p < \xi \leq L - l_p \\ \left(L + l_p - \xi\right)^2 / 4 L l_p & L - l_p < \xi \leq L + l_p \end{cases}$$

and

$$\nabla S_{Ip} = \begin{cases} 0 & |x_p - x_I| \geqslant L + l_p, \\ \frac{L+l_p+(x_p-x_I)}{2Ll_p} & -L - l_p < x_p - x_I \leqslant -L + l_p, \\ \frac{1}{L} & -L + l_p < x_p - x_I \leqslant -l_p, \\ -\frac{x_p-x_I}{Ll_p} & -l_p < x_p - x_I \leqslant l_p, \\ -\frac{1}{L} & l_p < x_p - x_I \leqslant L - l_p, \\ -\frac{L+l_p-(x_p-x_I)}{2Ll_p} & L - l_p < x_p - x_I \leqslant L + l_p. \end{cases}$$

In which $2lp$ is the particle domain, L is the mesh size in 1D, and is the relative particle position to node.

Weighting functions in 3D are obtained by the product of three one-dimensional weighting functions:

$$S_{Ip}(x_{ip}) = S_{Ip}(\xi)S_{Ip}(\eta)S_{Ip}(\zeta)$$

where $\xi = x_p - x_I, \eta = y_p - y_I$ and $\zeta = z_p - z$.

3 Explicit MPM integration

The discrete form of the motion equation

3.1 Central difference Method

The displacement, the velocity and the acceleration at time $t = 0, t^1, t^2, ..., t^n$ are knows, and the values at time t^{n+1} are required, namely the solution of the problem.

In central difference method, the velocity at $t^{n+1/2}$ can be approximated as:

$$\dot{u}^{n+1/2} = (u^{n+1} - u^n)\Delta t$$

and, the acceleration in t^n can be approximated as:

$$\ddot{u}^n = (\dot{u}^{n+1/2} - \dot{u}^{n-1/2})\Delta t$$

and therefore, the required displacement at t^{n+1} can be calculated as:

$$u^{n+1} = u^n + \dot{u}^{n+1/2}\Delta t$$

, where

$$\dot{u}^{n+1/2} = \dot{u}^{n-1/2} + \ddot{u}^n\Delta t$$

.

The motion equation in time t^n is:

$$m\,\ddot{u}^n = f^n$$

So,

$$\ddot{u}^n = f^n/m$$

and

$$\dot{u}^{n+1/2} = \dot{u}^{n-1/2} + f^n/m\,\Delta t$$

4 Numerical implementation

For one Δt, the updated position can be obtained as:

- Calculate force f_n
- Calculate acceleration $\ddot{u}^n = f^n/m$
- Impose essential boundary conditions in terms of $\ddot{u}^n$
- Update velocity $\dot{u}^{n+1/2} = \dot{u}^{n-1/2} + \ddot{u}^n \, \Delta t$
- Update position $u^{n+1} = u^n + \dot{u}^{n+1/2} \, \Delta t$
- Let $n = n + 1$
- Update position $u^{n+1} = u^n + \dot{u}^{n+1/2} \, \Delta t$

5 Stability Requirement

The central difference method is explicit here and conditionally stable, so the time step must be less that a certain value for avoiding error amplification.

For linear systems this critical time step value depends on the natural period of the system, in particular for undamped linear systems the critical time step is:

$$\Delta t_{cr} = T_n$$

T_n is the smallest natural period of the system. For finite element method the critical time step of the central difference method can be expressed as:

$$\Delta t_{cr} = \min_e(l^e/c)$$

Where l^e is the characteristic length of the element and c is the sound speed. This time step restriction implies that time step has to be limited such that a disturbance can travel across the smallest characteristic element length withing a single time step.

This condition is known as CFL condition, or Courant-Friedrichs-Lewy condition. For linear elastic material the sound speed is:

$$c = \sqrt{racE(1-\nu)(1+\nu)(1-2\nu)\rho}$$

In the MPM, the particles can has velocities in any time step, so the critical time speed can be written with this velocity plus:

$$\Delta t_{cr} = l^e/\max_p(c_p + |v_p|)$$

In a structured regular mesh, l^e is the grid cell dimension.

6 Numerical damping

Real materials dissipates energy during movement. Temperature is another source of dissipation. In numerical analysis numerical damping can be used.

7 Local damping

The numerical damping is a technique for getting a stationary solution of the dynamic system. In this simulator we have two type of numerical damping: the local (viscous) and the kinetic (dynamic relaxation) damping.

7.1 Quasi-static solution with local damping

The local damping is used to get a quasi-static solution of the dynamic system using a viscous nodal force. In each time step, a viscous force is applied in each node, whose magnitude is proportional and opposite to the nodal velocity.

$$f_{iI}^{dnplocal} = -\alpha |f_{iI}^{unb}| \hat{v}_{iI}$$

, where the unbalanced nodal force is:

$$f_{iI}^{unb} = f_{iI}^{int} + f_{iI}^{ext}$$

Therefore, the resulting discrete form of the motion equation with viscous nodal damping is:

$$\dot{p}_{iI} = f_{iI}$$

in which

$$\dot{p}_{iI} = f_{iI}^{int} + f_{iI}^{ext} + f_{iI}^{dnplocal}$$

7.2 Using local damping

For activate local damping force with $\alpha = 0.145$:

Listing 1: Defining local damping in json file.

```
"damping":{"type"=":"local", "value":0.145}
```

The stress field obtained from a local damping forced system, cam be used as initial stress field value.

In geostatic analysis, the quasi static analysis can be store in to a simulation state:

Listing 2: input JSON

```
"save_state":true,
```

And for loading the salved state

Listing 3: input JSON

```
  "load_state":true,
```

8 Explicit MPM Scheme

1. Calculate nodal mass and momentum

$$m_I^k = \sum_{p=1}^{n_p} m_p N_{Ip}^k$$

$$p_{iI}^{k-1/2} = \sum_{p=1}^{n_p} m_p v_{ip}^{k-1/2} N_{Ip}^k$$

2. Impose boundary conditions in terms of $p_{iI}^{k-1/2}$.

3. The stress can be updated using the vorticity tensor at beginning or at the end of the time step. For USL scheme calculate the particle strain increment and the vorticity increment using the current nodal velocity:

$$\Delta\varepsilon_{ijp}^{k-1/2} = \frac{1}{2}\left(N_{Ip,j}^{k} v_{iI}^{k-1/2} + N_{Ip,i}^{k} v_{jI}^{k-1/2}\right)\Delta t$$
$$\Delta\Omega_{ijp}^{k-1/2} = \frac{1}{2}\left(N_{Ip,j}^{k} v_{iI}^{k-1/2} - N_{Ip,i}^{k} v_{jI}^{k-1/2}\right)\Delta t$$

Where the nodal velocity is calculated with the nodal momentum:

8.1 Update Stress First - USF - Scheme

In the USF scheme the velocities in $n - 1/2$ are used to update the stress state:

8.2 Update Stress Last - USL - Scheme

In the USL scheme the updated velocities in nodes $n + 1/2$ are used to update the stress state:

8.3 Modified Update Stress Last - MUSL - Scheme

In the Modified USL scheme, the updated particle velocities are used to update the stress state:

9 Project structure

The project is organized such as:

- `src/ and inc/` sources and headers C++ files
- `manual/` program manual (e.g., STL), DEMs, etc.
- `tests/` simulation results of analytical and numerical tests
- `examples/` implementation verification and application examples

10 Build commands

Instruction commands using Linux/WSL:

```
cmake -S . -B build -DCMAKE_BUILD_TYPE=Release
cmake --build build -j
./build/MPM-Geomechanics --input inputs/base_case.json
```

11 Program features

The main features of the program are:

- Three-dimensional 3D formulation
- Dynamic formulation
- Shared memory parallelization using OpenMP
- Many ways to crate bodies (polygons, particle list, pre-defined bodies)
- Several constitutive models for soil and rock materials
- Softening/hardening models to represent weakness during large deformations
- Coupled fluid-mechanical formulation (*under development*)

12 Compiled binaries

For downloading the compiled binaries

- Go to the [Actions page](https://github.com/fabricix/MPM-Geomechanics/actions).
- Select the latest run of the **MSBuild** workflow for Window, or **CI** for Linux.
- At the bottom, you will find the available artifacts under the **Artifacts** section.
- Download the 'compiled-binaries' artifact to get the compiled code.

13 Required Programs

The following instructions outline the steps required to install all necessary programs to build the MPM-Geomechanics project and execute its test suite.

13.1 Required Programs - Windows

Program	Installation	Command
Winget	Microsoft's official website	Native of Windows
Git	via Winget	`winget install -e -id Git.Git -e -source winget`
MSYS2	via Winget	`winget install -e -id MSYS2.MSYS2 -source winget`
GitHub CLI	via MSYS2 MINGW64	`pacman -S mingw-w64-x86_64-github-cli`
Python	via MSYS2 MINGW64	`pacman -S mingw-w64-x86_64-python`
CMake	via MSYS2 MINGW64	`pacman -S mingw-w64-x86_64-cmake`
GCC	via MSYS2 MINGW64	`pacman -S mingw-w64-x86_64-gcc`
G++	via MSYS2 MINGW64	`pacman -S mingw-w64-x86_64-gcc`
Make	via MSYS2 MINGW64	`pacman -S make`

Table 1: Programs required for Windows installation.

Make sure you have Winget installed. You can verify this by running `winget -version`. If you don't have Winget installed, you can get it from Microsoft's official website.

Step 1: Install Git, GitHub, and MSYS2

Run the following commands:

```
winget install -e --id Git.Git -e --source winget
winget install -e --id MSYS2.MSYS2 --source winget
```

Step 2: Install required packages in MSYS2 MINGW64

```
pacman -S mingw-w64-x86_64-github-cli
pacman -S mingw-w64-x86_64-python
pacman -S mingw-w64-x86_64-cmake
pacman -S mingw-w64-x86_64-gcc
pacman -S make
```

Step 3: Verify the installations

```
git --version
gh --version
python --version
cmake --version
make --version
gcc --version
g++ --version
```

13.2 Required Programs - Linux

Program	Installation	Command
Git	via apt	`sudo apt install git`
GitHub CLI	via apt	`sudo apt install gh`
Python	via apt	`sudo apt install python3 python3-pip`
CMake	via apt	`sudo apt install cmake`
GCC	via apt	`sudo apt install build-essential`
G++	via apt	`sudo apt install build-essential`
Make	via apt	`sudo apt install build-essential`

Table 2: Programs required for Linux installation.

Step 1: Install all packages via apt

```
sudo apt install git
sudo apt install gh
sudo apt install python3 python3-pip
sudo apt install cmake
sudo apt install build-essential
```

Step 2: Verify the installations

```
git --version
gh --version
python3 --version
cmake --version
make --version
gcc --version
g++ --version
```

13.3 Compilation in Windows

Prior to proceeding with these instructions, please consult the windows required programs section in this manual.

The simplest way to compile on windows is by using the '.bash' file at '/build/CMake' with 'MSYS2 MINGW64' console line, just execute:

```
  cd project-directory/build/CMake
./cmake-build.bash
```

Alternatively, you can use the following commands:

```
  cd project-directory/build/CMake
  cmake -G "Unix Makefiles" -B build
  cmake --build build
```

13.4 Compiling on Linux

Prior to proceeding please consult the Linux Required Programs section in this manual.

The simplest way to compile on Linux is by using the ‘.bash‘ file at ‘/build/CMake‘, just execute:

```
cd build/CMake
./cmake-build.bash
```

Alternatively, you can use the following commands:

```
cd build/CMake
cmake -G "Unix Makefiles" -B build
cmake --build build
```

14 Visual Studio Solution

For compiling the code in windows you can use the Visual Studio solution file ‘/build/MPM-Geomechanics.sln‘, and build it by pressing ‘Ctr+B‘. Alternatively you can compile it by using command in a *Developer Command Prompt*:

```
msbuild MPM-Geomechanics.sln -p:Configuration=Release
```

15 Make Compilation

For compile the code in a linux environment, execute the makefile inside the make folder: `MPM-Geomechanics\build\make\makefile`.

16 Documentation

The program documentation is generated using Doxygen documentation generator.

```
doxygen Doxyfile
```

The HTML generated documentation is located in ‘/docs/index.html‘.

17 Execution

In order to run simulations in several terminal, you can add the compiled code in the system ‘PATH‘.

18 Testing Compilation and Benchmarking

19 How to Compile

Prior to proceeding with these instructions, please consult required programs in manual.

The tests use **GoogleTest**. It is necessary to import this library by cloning the official repository into the folder ‘/external‘. Each developer must clone this repository independently.

“‘ cd external git clone https://github.com/google/googletest.git “‘

The simplest way to compile on Windows and Linux is by using the ‘.bash‘ file at ‘/build/qa‘ with ‘MSYS2 MINGW64‘ console line, simply execute the following command in the directory ‘MPM-Geomechanics/build/qa‘: “‘ ./cmake-build.bash “‘

Alternatively, you can use the following commands: “‘ cmake -G "Unix Makefiles" -B build “‘ “‘ cmake –build build “‘

These commands will generate two executables: ‘MPM-Geomechanics-tests‘ and ‘MPM-Geomechanics-benchmark‘.

- ‘MPM-Geomechanics-tests‘: Run testing using GoogleTest. All files ending with ‘.test.cpp‘ are testing files, you can find them in the directory ‘qa/tests‘.

- ‘MPM-Geomechanics-benchmark‘: Run benchmark using GoogleTest. All files ending with ‘.benchmark.cpp‘ are performance files. You can find them in the directory ‘qa/benchmark‘.

20 How does benchmarking work?

If you are using Windows OS, make sure to use the MINGW64 command-line console.

20.1 Executable MPM-Geomechanics-benchmark

To run the benchmark correctly, a JSON file named `benchmark-configuration.json` is required. This file allows the user to specify values for each test. If the file does not exist or a value is missing, a default value will be used.

The executable `MPM-Geomechanics-benchmark` allows the following command-line arguments:

`<directory>`: Specifies which file should be used to run the benchmark. If no file is specified, the program will use `benchmark-configuration.json` located in the same directory as the executable. Example: `MPM-Geomechanics-benchmark configuration-file.json`

The executable `MPM-Geomechanics-benchmark` allows the following command-line flags:

`-log`: Shows more information about missing keys in the `benchmark-configuration.json` file

20.2 Script (executable): start-multi-benchmark.py

The performance test can also be executed using the ‘start-multi-benchmark.py‘ script, which allows running benchmarks with one or more executables downloaded as artifacts from GitHub and stores the log results in separate folders. Each executable is automatically downloaded from GitHub as an artifact, using an ID specified in `start-multi-benchmark-configuration.json`. Additionally, the benchmark configuration (number of martial points and number of threads) can be defined in the same file.

Example of a `start-multi-benchmark-configuration.json` file:

```
{
  "executables": {
    "win_benchmark_executable": "MPM-Geomechanics-benchmark.exe",  -> An executable located at build/qa/MPM-Geomechanics-benchmark.exe
      "win_github_id": "17847226555"  -> An artifact ID that represent it. can check out if it exists via gh run view <ID>
  },
  "parameters": {
    "particles": [ 15000, 25000, 50000],  -> material point to test
    "threads": [1, 5, 10]  -> number of threads to test
  }
}
```

This setup will run 3x3 (particles x threads) totaling 18 tests (3x3x2)

The executable `start-multi-benchmark.py` allows the following command-line flags:

`-clear`: Removes the **benchmark folder** (if it exists) before executing the performance tests. `-cache`: Uses the previously created cache instead of the `start-multi-benchmark-configuration.json` file.

21 Verification Problems

21.1 Boussinesq's Problem

Introduction

In Geomechanics, the Boussinesq's problem refers to the point load acting on a surface of an elastic half-space. The boundary conditions for this problem are:

- The load P is applied only at one point, at the origin.
- The load is zero at any other point.
- For any point infinitely distant from the origin, the displacements must all vanish.

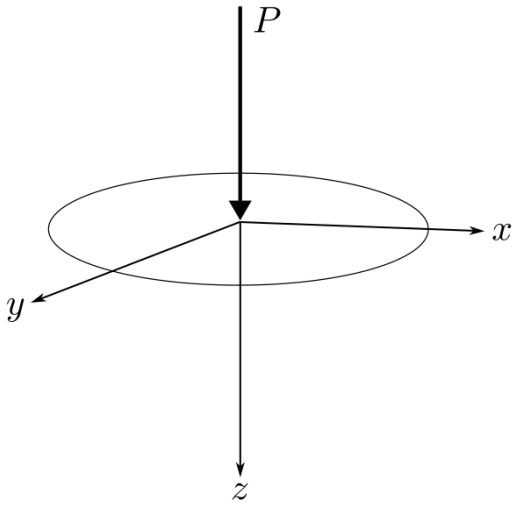

Figure 2: Boussinesq's problem.

Analytical Solution

The analytical solution of the vertical displacement field is:

$$u_z(x, y, z) = \frac{P}{4\pi G d}\left(2(1-\nu) + \frac{z^2}{d^2}\right)$$

where $G = \frac{E}{2(1+\nu)}$ is the shear modulus of the elastic material, ν is the Poisson ratio, and $d = \sqrt{x^2 + y^2 + z^2}$ is the total distance from the load to the point.

MPM Model and Result Comparison

To model the displacement field generated by the point load, we create an elastic body with dimensions $l_x = l_y = l_z = 1\,\mathrm{m}$, using the keyword `"cuboid"` with Point 1 at (0,0,0) and Point 2 at (1,1,1).

For the elastic parameters:

$$E = 200 \times 10^6\,\mathrm{Pa}, \quad \rho = 1500\,\mathrm{kg/m^3}, \quad \nu = 0.25$$

The computational mesh has cell dimensions $\Delta x = \Delta y = \Delta z = 0.1\,\mathrm{m}$. A nodal load of magnitude 1 is applied in the vertical direction at the midpoint of the upper surface. The plane Z_n is free, while all other planes are sliding (only tangential displacements allowed). Dynamic relaxation is used to reach a static solution, through the keyword `"damping"` with type `"kinetic"`.

Input File

```
{
    "body": {
        "elastic-cuboid-body": {
            "type":"cuboid",
            "id":1,
            "point_p1":[0.0,0.0,0],
            "point_p2":[1,1,1],
            "material_id":1
        }
    },
    "materials": {
        "material-1": {
            "type":"elastic",
            "id":1,
            "young":200e6,
            "density":1500,
            "poisson":0.25
        }
    },
    "mesh": {
        "cells_dimension":[0.1,0.1,0.1],
        "cells_number":[10,10,10],
        "origin":[0.0,0.0,0.0],
        "boundary_conditions": {
            "plane_X0":"sliding",
            "plane_Y0":"sliding",
            "plane_Z0":"sliding",
            "plane_Xn":"sliding",
            "plane_Yn":"sliding",
            "plane_Zn":"free"
        }
    },
    "time":0.025,
    "time_step_multiplier":0.3,
    "results": {
        "print":50,
        "fields":["all"]
    },
    "nodal_point_load": [[[0.5, 0.5, 1.0], [0.0, 0.0, -1.0]]],
    "damping": {
        "type":"kinetic"
    }
}
```

The MPM numerical results show good agreement with the analytical solution, with small deviations due to discretization and the representation of the domain by particles with finite volume. Errors decrease with mesh refinement.

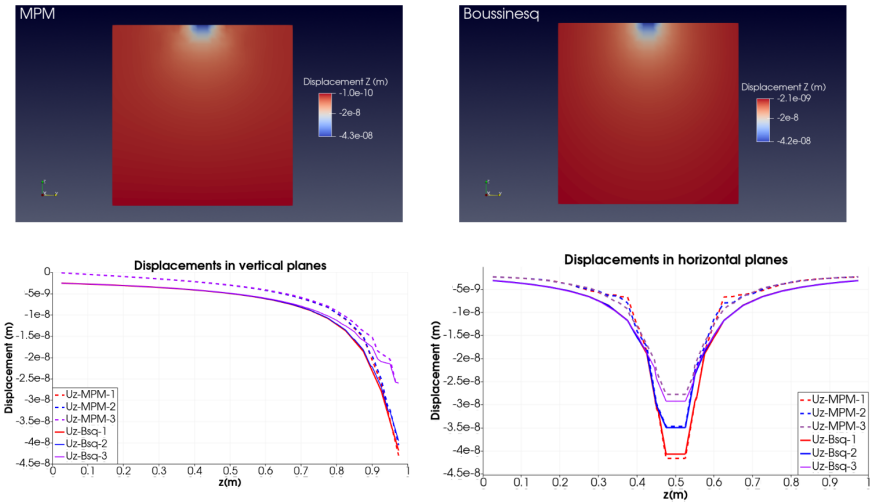

Figure 3: Comparison between analytical and MPM results.

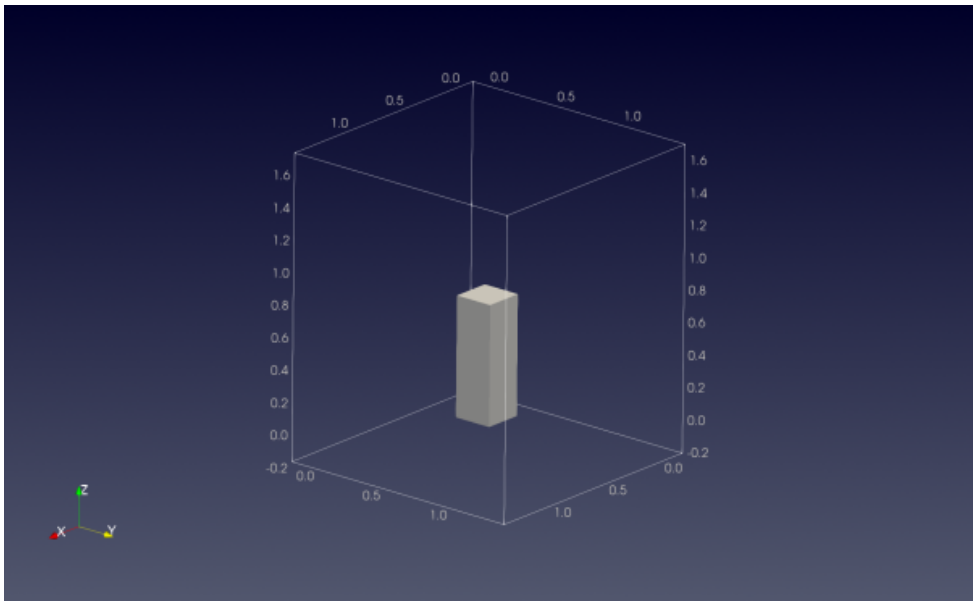

Figure 4: Geometry of the elastic body.

21.2 Base Acceleration Example

Introduction

In this example we model the motion of an elastic body subjected to a dynamic boundary condition. The body is a cuboid with dimensions $l_x = l_y = 0.3\,\text{m}$ and $l_z = 0.8\,\text{m}$, with lower coordinate point $p_{min} = (0.4, 0.4, 0.0)\,\text{m}$.

The base acceleration is defined as:

$$\ddot{u} = A\,2\pi f \cos(2\pi f t + \alpha)$$

The total simulation time is $T = 2\,\text{s}$, with a time step $\Delta t = 10^{-4}\,\text{s}$. Material properties:

$$\rho = 2500\,\text{kg/m}^3, \quad E = 100 \times 10^6\,\text{Pa}, \quad \nu = 0.25$$

MPM Model

The MPM model consists of uniformly distributed particles inside the body, created with the keywords `"body"` and `"cuboid"`. The mesh dimensions are $\Delta x = \Delta y = \Delta z = 0.1\,\text{m}$, and it covers the full expected displacement range of the body. The mesh uses $n_x = 12, n_y = 12, n_z = 15$.

Input File

```
{
  "stress_scheme_update":"USL",
  "shape_function":"GIMP",
  "time":10,
  "time_step":0.0005,
  "gravity":[0.0,0.0,0.0],
  "n_threads":1,
  "damping": {
    "type":"local",
    "value":0.0
  },
```

```
  "results": {
    "print":100,
    "fields":["id","displacement","velocity","material","active","body"]
  },
  "n_phases":1,
  "mesh": {
    "cells_dimension":[0.1,0.1,0.1],
    "cells_number":[10,10,15],
    "origin":[0,0,0]
  },
  "earthquake": {
    "active": true,
    "file": "base_acceleration.csv",
    "header": true
  },
  "material": {
    "elastic_1": {
      "type":"elastic",
      "id":1,
      "young":10e6,
      "density":2500,
      "poisson":0.25
    }
  },
  "body": {
    "columns_1": {
      "type":"cuboid",
      "id":1,
      "point_p1":[0.2,0.2,0],
      "point_p2":[0.5,0.5,1.0],
      "material_id":1
    }
  }
}
```

Earthquake Block Parameters

```
"earthquake": {
  "active": true,
  "file": "base_acceleration.csv",
  "header": true
}
```

Where:

- **active**: Enables or disables seismic loading.
- **file**: Path to the CSV file containing time, acceleration_x, acceleration_y, and acceleration_z.
- **header**: True if the CSV has a header row.

Example of the first lines of the record:

```
t,ax,ay,az
0.0,-1.8849555921538759,-0.9424777960769379,-0.0
5e-05,-1.8849554991350466,-0.9424777844495842,-0.0
0.0001,-1.884955220078568,-0.9424777495675233,-0.0
0.00015000000000000001,-1.8849547549844674,-0.9424776914307561,-0.0
...
```

Results and Visualization

After simulation, particle results are found in the `/particle` directory, and grid results in the `/grid` directory. Particle results (`particle_1.vtu`, `particle_2.vtu`, ..., `particle_41.vtu`) are referenced in `particleTimeSerie.pvd`, which can be opened in ParaView:

`File → Open → particleTimeSerie.pvd`

The mesh can be loaded with:

`File → Open → eulerianGrid.vtu`

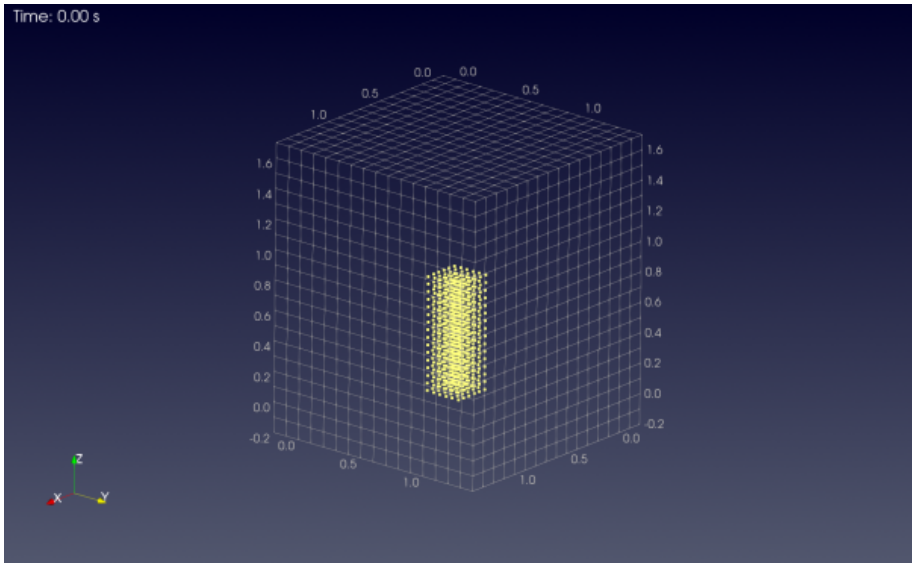

Figure 5: Particles and mesh of the analyzed case.

Verification of Dynamic Boundary Condition

The velocity from the analytical input function

$$\dot{u} = A \sin(2\pi f t + \alpha)$$

was compared with the particle velocity at the base of the model. The results show excellent agreement between analytical and MPM-calculated velocities.

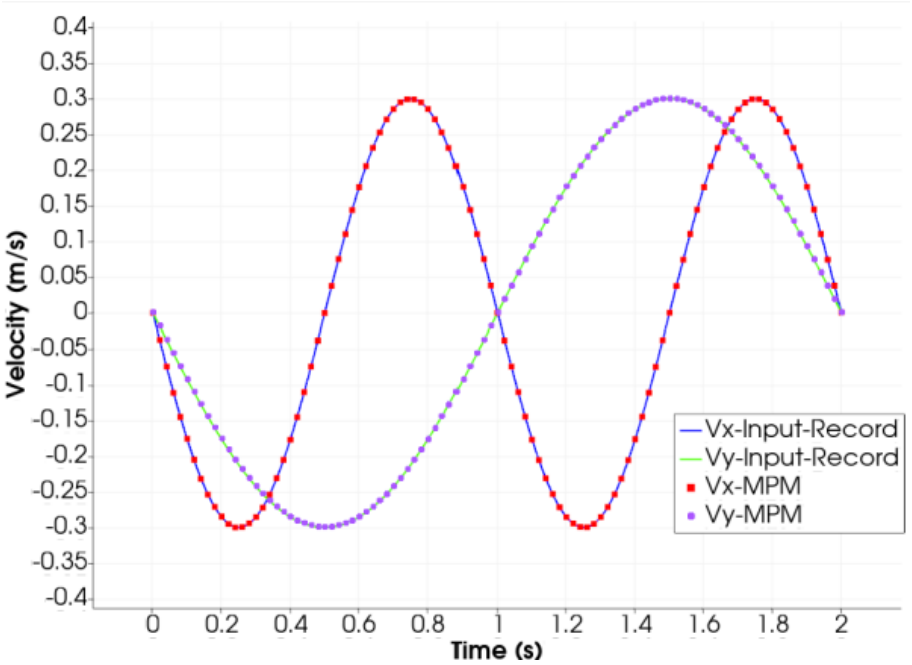

Figure 6: Verification of velocities obtained with MPM simulation.