

LEMPS – User Guide (1.0.170221)

Balázs Tóth (BME)  
e-mail: [Toth.Balazs@epito.bme.hu](mailto:Toth.Balazs@epito.bme.hu)

Budapest, 02.25.2017

Table of Contents

Introduction 2

License 2

Installation 2

Mathematical model 3

Simulation workflow 4

Structure of the XML-document 5

List of functions 7

Running the solver 8

Example 9

Appendix 11

# Introduction

LEMPS (Lagrangian Engine for Methods of ParticleS) is a general-purpose open-source C++ library that facilitates the application and implementation of meshless numerical methods. The goal of the LEMPS-project is to establish a meshless solver that can solve user-defined system of PDE-s over a set of spatially distributed nodes (particles) in one, two, or three dimensions. Solutions can be governed by priori implemented schemes like SPH, DEM, etc. By emerging the system of equations to the user’s level LEMPS becomes a flexible meshless simulation tool for a wide range of applications.

To read a short introduction of the project with simple examples please visit the project page: <https://bitbucket.org/lempsproject/>

# License

LEMPS is free software: you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version LEMPS is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details You should have received a copy of the GNU Lesser General Public License along with LEMPS. If not, see <<http://www.gnu.org/licenses/>> For more information please visit: <<https://bitbucket.org/lempsproject/>>

# Installation

Please note that currently, LEMPS cannot be installed on Windows, it supports only MAC OS and Ubuntu systems.

The installation requires a very few dependencies to be installed. These dependencies are:

* Visualization Toolkit 7.0.0 ([link](http://www.vtk.org/files/release/7.0/VTK-7.0.0.zip)),
* Common utilities ([link](https://bitbucket.org/lempsproject/commonutils/downloads/commonutils-1.0.170221.zip)),
* ProLog ([link](https://bitbucket.org/lempsproject/prolog/downloads/prolog-1.0.170221.zip)),
* HandyXML ([link](https://bitbucket.org/lempsproject/handyxml/downloads/handyxml-1.0.170221.zip)).

You can perform the installation process including the dependencies using the Installation tool for LEMPS ([link](https://bitbucket.org/lempsproject/installation/downloads/install-1.0.170221.sh)). By typing the command

$ sh intall-1.0.170221.sh

in terminal in the folder, you would like to install LEMPS, the installer downloads and installs the necessary files in your system. After installation, the executable (**pmsimple)** will be placed in the bin directory and a **start.sh** file will be generated in the installation directory. The installer also adds the bin directory to your environmental variables so you can run it from different directories. In certain cases, when environmental variables are not available (through ssh-connection) you can perform computations using the **start.sh** file in the desired folder:

$ sh start.sh

The post-processing of the results can be carried out with Paraview, which is strongly recommended to be installed with LEMPS ([link](http://www.paraview.org/)).

# Mathematical model

This section explains the fundamental structure of the solver with the most important notions from the user point of view.

## Definitions

### Tensor

In LEMPS tensor is considered as an at most second order matrix in one, two, or three dimensions. Consequently, it can have any shape within a 3 by 3 matrix, forming an atomic and fundamental quantity in LEMPS. Scalars, vectors and tensors can be treated in arbitrary dimensions (one, two or three) using this quantity and the operations (like summation, inner or outer product, etc.) between tensors of same or different sizes can be governed by mathematical rules. Note: LEMPS currently does not support higher order or complex valued tensors.

### Constant

Global value which preserves its initial value during the simulation. Warning on changing attempt is muted. Constants can be named or unnamed (e.g. , is named, is unnamed).

## Variable

Global value which can be changed during the simulation.

### Particle

Particle is a point-like object with position in space.

### Field

Values interpreted over a set of spatially distributed particles. Field values are assigned to particles. Can be initialized using constant and variable values as a value assignment.

### Domain

Domain is an axis-aligned rectangular box in which the particles can occupy their locations. No particles can exist out of this box. The domain can be one, two, or three dimensional and bounds the included space with periodic or symmetric boundaries. The domain must be divided into cells whom edge length must be equal to the particle interaction radius.

### Particle system

A special field that stores particle positions in space. It performs the neighbour search and all the necessary sorting procedures. A particle system always requires a domain assigned to it. Only one particle system can be defined in a simulation.

### Workspace

This is a database which stores constants, variables, fields, and the particle system. Other objects are present in the workspace for convenience. Only one workspace can be defined in a simulation.

### Expression

Expression is an object built up using mathematical operators acting on constants, variables and fields.

### Function

Function is an assignment of an expression to a variable or field.

### Function space

Function space wraps the workspace and the list of functions. It also performs the evaluation of the functions.

### Parameter space

It stores the parameters related to the simulation flow, like the physical time or the result saving methods.

## Solution

Using tensors all necessary quantities (global of field) can be constructed forming an interpretation basis of the desired physical model and the governing equations are formulated as functions.

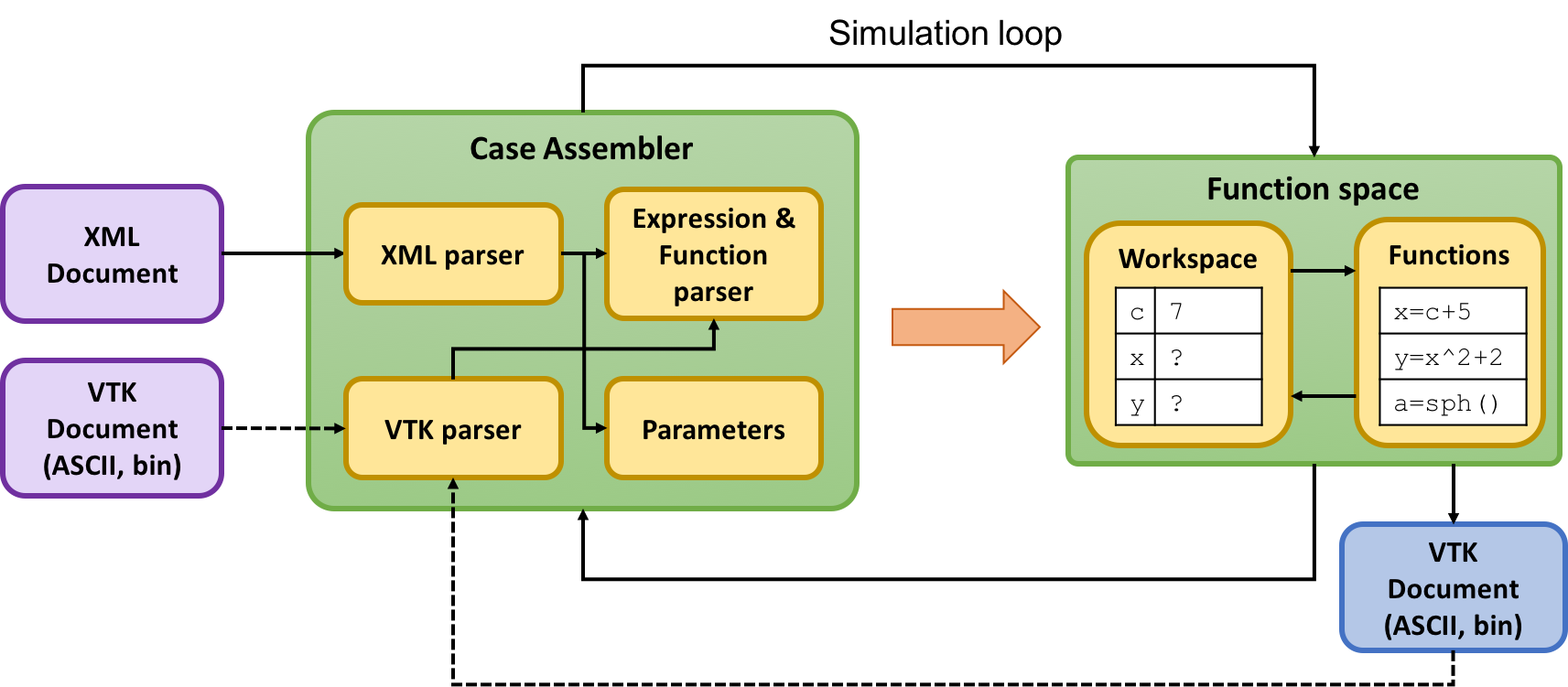


Figure 1: Simulation loop, structure of the solver

The solution of any function in the function space appears in the workspace immediately without any further update request. During the calculation, the function space is periodically triggered to solve the functions in the defined order and prints the function space itself to VTK-files with a user-defined sampling interval (print\_interval). This procedure is shown in Figure 1.

# Simulation workflow

It is strongly recommended to read the Mathematical model section before this section.

LEMPS aims the goal of generality at two individual levels: the user, and the intermediate developer levels. Throughout this document only the user-level will be investigated and explained. During the application procedure of LEMPS the user is not required to code or re-compile any parts of the solver before performing calculations. The geometrical layout, equations and simulation parameters are provided to the solver through XML and ASCII or binary VTK documents.

When the user defines a problem to solve, the following steps should be always done as a workflow (more or less in a strict order) before the calculation could be started:

1. Define **constant** (preserves its initialization value during the simulation) and **variable** quantities that are global values during the calculation.
2. Create a calculation **domain**.
3. Generate **particle system** inside the domain over a uniform grid. The particle system handles neighbour search, sorting, etc. Note that only one particle system can exist in a simulation.
4. Define **field**s over the particle system. The definition can depend on constants and variables.
5. Interpret your equations considering the discrete particle system and define them as **functions** using the defined constants, variables and fields.
6. Finally, define the **simulation parameters** to govern the calculation flow and the registration of the results.

Simulations can be initialized in two ways:

* through XML-document consisting of the problem definition,
* XML-document pointing to a result VTK-file of former simulation.

Note: This is a short summary of these fundamental steps, more information about the modelling process will be placed in the following sections.

# Structure of the XML-document

In this section the elements of the input XML-document is discussed. The structure and the blocks of the XML-document is shown in Figure 3 and Figure 2. Since the XML-parser reads the blocks in a certain order, they can be placed inside the XML document in an arbitrary order. However, the structure must be properly defined.

The sequence of the block reading:

1. Function space
   1. Workspace
      1. Constants
      2. Variables
      3. Particle system
         1. Domain
         2. Particle grid
      4. Fields
   2. Functions
2. Parameter space.

The values defined in the constants’ and variables’ blocks are considered as known values in all subsequent blocks including the parameter space. The particle system defines three fields in the workspace:

* : particle positions,
* : particle identifier,
* : grid identifier,

which are not allowed to be overridden by the user.

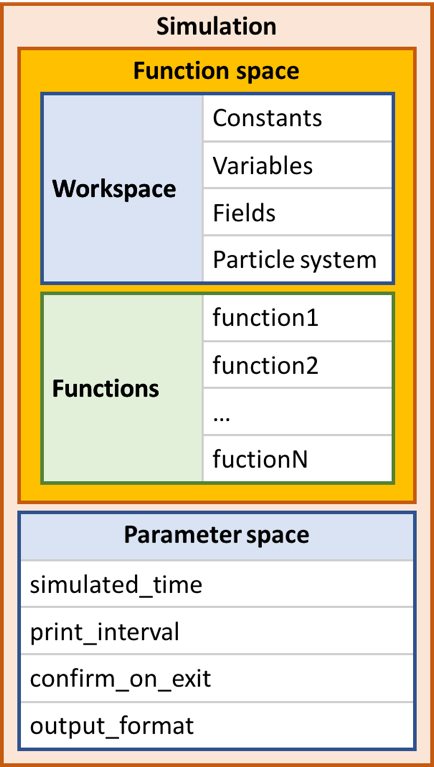


Figure 2: Structure of the XML-document

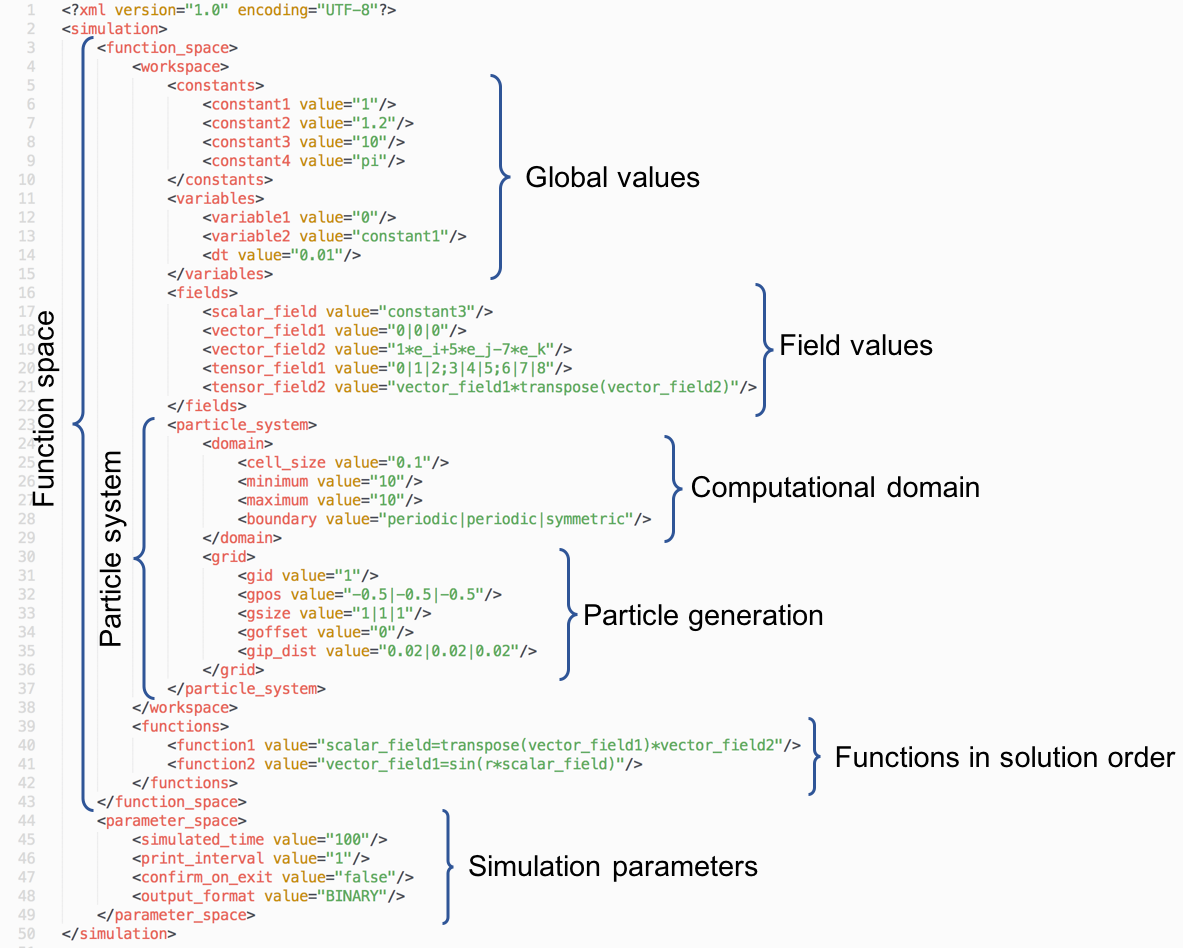


Figure 3: Blocks of the XML-document

# List of functions

The most important functions implemented in LEMPS are shown in Table 1.

Table 1: List of functions in LEMPS

|  |  |  |
| --- | --- | --- |
|  | name | Desription |
| 1 | abs | one operand function |
| 2 | acos |
| 3 | acot |
| 4 | and |
| 5 | asin |
| 6 | atan |
| 7 | cos |
| 8 | cosh |
| 9 | cot |
| 10 | coth |
| 11 | exp |
| 12 | floor |
| 13 | gt | two operands: 1: a, 2: b (a>b) |
| 14 | if | three operands: 1: logical test, 2: if true, 3: if false |
| 15 | log | one operand function |
| 16 | lt | two operands: 1: a, 2: b (a<b) |
| 17 | magnitude | vector length |
| 18 | max | two operands |
| 19 | min | two operands |
| 20 | mod | two operands |
| 21 | not | logical negation |
| 22 | or | logical or |
| 23 | sgn | sign function |
| 24 | sin | one operand function |
| 25 | sinh |
| 26 | sqrt |
| 27 | tan |
| 28 | tanh |
| 29 | trace | trace of matrix |
| 30 | transpose | transpose of vector of tensor |
| 31 | trunc | truncation to integer |
| 32 | xor | logical xor |
| 33 | cross | cross product |
| 34 | inverse | inverse of matrix |
| 35 | determinant | determinant of nxn matrix |
| 36 | eq | two operands: 1: a, 2: b (a==b) |

The complete list with interaction operators and pre-defined values can be found in the Appendix.

# Running the solver

Once LEMPS is properly installed the solver is ready to launch from terminal. Considering that the installed bin directory is defined in the list of environment variable paths, the user can launch the solver by typing the executable name in terminal in the desired directory:

$ pmsimple

This command runs the executable that looks for a “config.xml” file in the same folder. During the simulation, the solver writes the results in the same directory (by default).

The user can pass optional commands to the solver right after the name of the executable:

$ pmsimple –xmlname document.xml

This command overrides the default “config.xml” name of the XML document with “document.xml” as the simulation input file.

The list of commands is summarized in Table 2.

Table 2: Runtime commands

|  |  |  |
| --- | --- | --- |
|  | Command | Description |
| 1 | -help | Display LEMPS information. |
| 2 | -wsres | Lists all reserved names in workspace. |
| 3 | -numthreads <number> | Defines the number of threads to use. Default is detected. |
| 4 | -xmlname <filename> | Defines the name of the XML input file. |
| 5 | -logfile <filename> | Defines the name of the output log file. |
| 6 | -wdir <directory> | Defines the working directory. FULL path of an EXISTING directory is required. |
| 7 | -version | Prints the version number. |

Note 1: former simulation results in the working directory are overwritten without prompt.

Note 2: number of threads must be less than the number of particles.

# Examples

In this section the application of the solver is introduced through two simple test cases.

## Three-body problem

This test case simulates three point-like bodies forming a self-gravitating system in two dimensions. The governing equations are

where is the gravitational constant, kg is the particle mass and , where is the position of the -th particle. The initial conditions of the particles are presented in Table 3.

Table 3: Initial conditions

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Position | | Velocity | |
| p1 | 0.9700436 | -0.24308753 | 0.466203685 | 0.43236573 |
| p2 | -0.9700436 | 0.24308753 | 0.466203685 | 0.43236573 |
| p3 | 0.9700436 | 0.24308753 | -0.932407370 | -0.86473146 |

These initial conditions are related to the non-trivial eight-shaped choreography of three nodes. The initial conditions are read from the “eightshaped.vtk” file by the extremely simple “eightshaped.xml” document:

<?xml version="1.0" encoding="UTF-8"?>

**<simulation>**

**<initial\_condition>**

**<file** value="eightshaped.vtk"**/>**

**</initial\_condition>**

**<parameter\_space>**

**<simulated\_time** value="10"**/>**

**<print\_interval** value="0.1"**/>**

**<confirm\_on\_exit** value="0"**/>**

**<output\_format** value="ASCII"**/>**

**</parameter\_space>**

**</simulation>**

The simulation can be launched by typing the following in terminal:

$ pmsimple –xmlname eightshaped.xml –numthreads 3

After the calculation is finished the results can be visualized in Paraview ([link](http://www.paraview.org/)). The first 20 results are plotted in Figure 4.

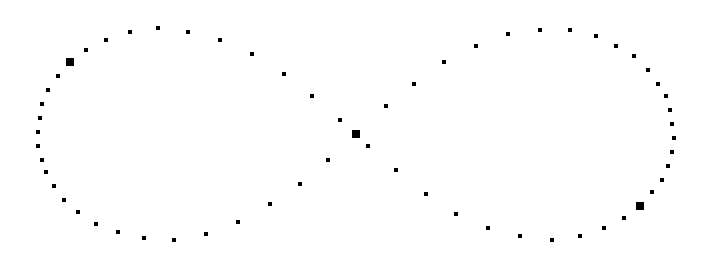


Figure 4: First 20 result steps of three body problem solution (thick dots show the initial positions).

## SPH – Linear wave equation

In this test case the linear wave equation is solved in two dimensions. The governing equation is a second order hyperbolic PDE:

where is a scalar field and is the wave propagation constant.

The following initial condition is applied:

over the uniformly distributed particle set of the domain . The boundaries are considered as periodic and symmetric conditions in and directions respectively:

The solver can be launched with the “linwave.xml” document (see Appendix) to build the whole simulation problem:

$ pmsimple –xmlname linwave.xml

Finally, the results in different time instants with a triangulated surface over is shown in Figure 5.

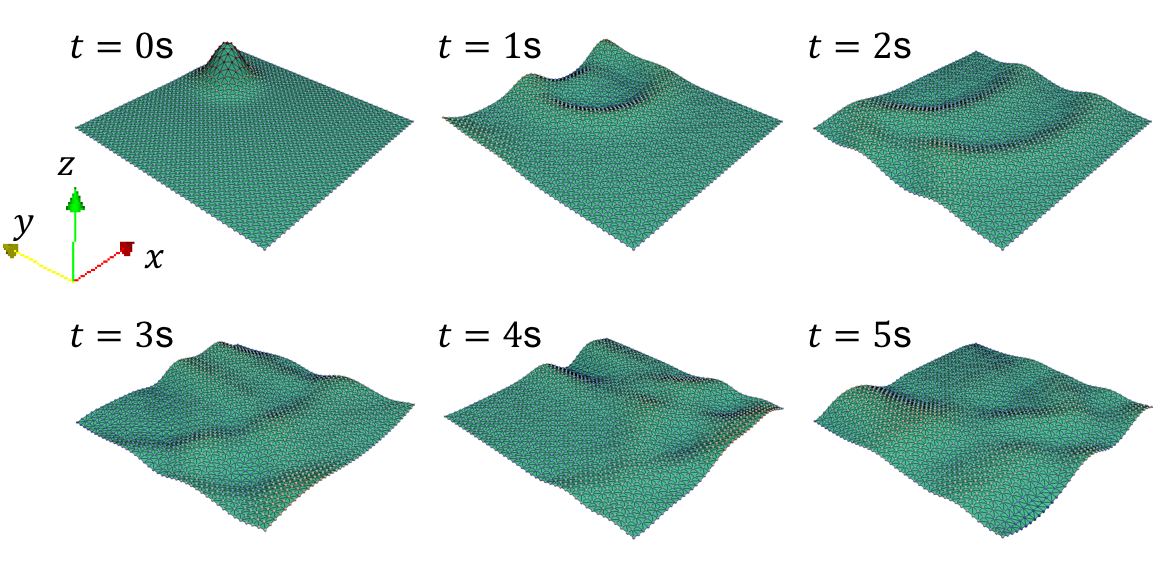


Figure 5: Solution of the linear wave equation at different instants.

# Appendix

1. The complete list of reserved names and pre-defined values in LEMPS:

|  |  |  |
| --- | --- | --- |
|  | Name | Description |
| 1 | id | particle identifier |
| 2 | TRUE | logical true |
| 3 | FALSE | logical false |
| 4 | pi | 3.14159265359 |
| 5 | Wp22210 | SPH kernel functions |
| 6 | Wp22220 |
| 7 | Wp22230 |
| 8 | Wp32210 |
| 9 | Wp32220 |
| 10 | Wp32230 |
| 11 | Wp52210 |
| 12 | Wp52220 |
| 13 | Wp52230 |
| 14 | domain\_min | minimum corner of the domain |
| 15 | domain\_max | maximum corner of the domain |
| 16 | cell\_size | size of the cells in the domain |
| 17 | ASCII |  |
| 18 | BINARY |  |
| 19 | periodic |  |
| 20 | symmetric |  |
| 21 | e\_i | Basis vectors. Dimensions are based on the domain dimensions. |
| 22 | e\_j |
| 23 | e\_k |
| 24 | abs | one operand function |
| 25 | acos |
| 26 | acot |
| 27 | and |
| 28 | asin |
| 29 | atan |
| 30 | cos |
| 31 | cosh |
| 32 | cot |
| 33 | coth |
| 34 | dem | Discrete element operator |
| 35 | div |  |
| 36 | elem | three operands: 1: name, 2: column, 3: row |
| 37 | exp | one operand function |
| 38 | floor |
| 39 | fmax | Field maximum |
| 40 | fmean | Field mean |
| 41 | fmin | Field minimum |
| 42 | grad |  |
| 43 | gt | two operands: 1: a, 2: b (a>b) |
| 44 | if | three operands: 1: logical test, 2: if true, 3: if false |
| 45 | log | one operand function |
| 46 | lt | two operands: 1: a, 2: b (a<b) |
| 47 | magnitude | vector length |
| 48 | max | two operands |
| 49 | min | two operands |
| 50 | mod | two operands |
| 51 | not | logical negation |
| 52 | or | logical or |
| 53 | rand | two operands: 1: a, 2: b. Random values between a and b |
| 54 | sgn | sign function |
| 55 | sin | one operand function |
| 56 | sinh |
| 57 | sph\_D00 | SPH operators |
| 58 | sph\_D01 |
| 59 | sph\_D10 |
| 60 | sph\_D11 |
| 61 | sph\_G00 |
| 62 | sph\_G01 |
| 63 | sph\_G10 |
| 64 | sph\_G11 |
| 65 | sph\_L00 |
| 66 | sph\_S00 |
| 67 | sph\_X00 |
| 68 | sqrt | one operand function |
| 69 | tan |
| 70 | tanh |
| 71 | trace | trace of matrix |
| 72 | transpose | transpose of vector of tensor |
| 73 | trunc | truncation to integer |
| 74 | xor | logical xor |
| 75 | identity | one operand: 1: a. Identity tensor of size a |
| 76 | neighbours | Gives the number of neighbors |
| 77 | simulation | Reserved for blocks |
| 78 | workspace |
| 79 | function\_space |
| 80 | variables |
| 81 | constants |
| 82 | fields |
| 83 | particle\_system |
| 84 | parameter\_space |
| 85 | domain |
| 86 | cell\_size |
| 87 | grid |
| 88 | functions |
| 89 | nbody | n-body interaction operator |
| 90 | cross | cross product |
| 91 | inverse | inverse of matrix |
| 92 | determinant | determinant of nxn matrix |
| 93 | eq | two operands: 1: a, 2: b (a==b) |
| 94 | euler | numerical integration operators. |
| 95 | predictor |
| 96 | corrector |

1. The “eightshaped.vtk” file:

# vtk DataFile Version 4.0

vtk output

ASCII

DATASET POLYDATA

FIELD FieldData 4

domain 1 4 string

domain\_min:-10|-10

domain\_max:10|10

cell\_size:1

boundary:0|0

variables 1 1 string

T:0

constants 1 3 string

coef:1

mass:1

dt:0.0001

functions 1 3 string

function1:a=(nbody(mass,coef)/mass)

function2:v=euler(v,a,dt)

function3:r=euler(r,v,dt)

POINTS 3 float

0.9700436 -0.24308753 0 -0.9700436 0.24308753 0 0 0 0

VERTICES 6 6

1 0

1 1

1 2

POINT\_DATA 3

SCALARS id float

LOOKUP\_TABLE default

0 1 2

FIELD FieldData 4

r 3 3 float

0.9700436 -0.24308753 0 -0.9700436 0.24308753 0 0 0 0

gid 1 3 float

0 0 0

a 3 3 float

0 0 0 0 0 0 0 0 0

v 3 3 float

0.466203685 0.43236573 0 0.466203685 0.43236573 0 -0.932407370 -0.86473146 0

1. Liner wave equation; “linwave.xml”:

<?xml version="1.0" encoding="UTF-8"?>

**<simulation>**

**<function\_space>**

**<workspace>**

**<constants>**

**<L** value="1"**/>**

**<csize** value="L/20"**/>**

**<dx** value="csize/2.1"**/>**

**<rho0** value="1000"**/>**

**<mass** value="dx^2\*rho0"**/>**

**<dt** value="0.002"**/>**

**<a** value="0.1"**/>**

**</constants>**

**<variables>**

**<T** value="0"**/>**

**</variables>**

**<fields>**

**<phi** value="10\*exp(-((elem(r,0,0)-0.2)^2+(elem(r,1,0)-0.3)^2)/0.01)"**/>**

**<phi\_dot** value="0"**/>**

**<phi\_dot\_dot** value="0"**/>**

**</fields>**

**<particle\_system>**

**<domain>**

**<cell\_size** value="csize"**/>**

**<minimum** value="-L/2/csize|-L/2/csize"**/>**

**<maximum** value="L/2/csize|L/2/csize"**/>**

**<boundary** value="periodic|symmetric"**/>**

**</domain>**

**<grid>**

**<gid** value="0"**/>**

**<gpos** value="-L/2|-L/2"**/>**

**<gsize** value="L|L"**/>**

**<goffset** value="0|0"**/>**

**<gip\_dist** value="dx|dx"**/>**

**</grid>**

**</particle\_system>**

**</workspace>**

**<functions>**

**<eq0** value="T=T+dt"**/>**

**<eq1** value="phi\_dot\_dot=if(gid,0,a\*sph\_L00(phi,mass,rho0,Wp22220))"**/>**

**<p2** value="phi\_dot=euler(phi\_dot,phi\_dot\_dot,dt)"**/>**

**<p3** value="phi=euler(phi,phi\_dot,dt)"**/>**

**</functions>**

**</function\_space>**

**<parameter\_space>**

**<simulated\_time** value="5"**/>**

**<print\_interval** value="0.02"**/>**

**<confirm\_on\_exit** value="0"**/>**

**<output\_format** value="ASCII"**/>**

**</parameter\_space>**

**</simulation>**