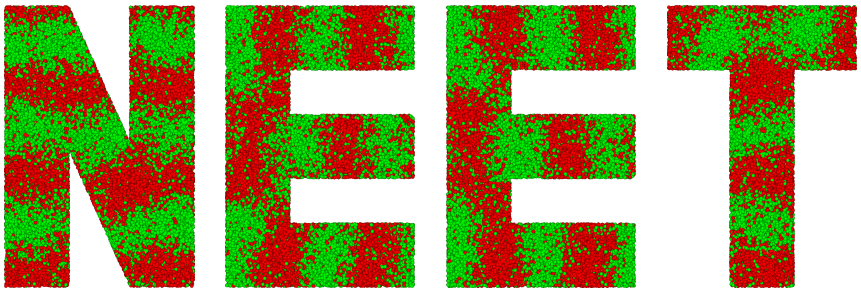
SpaghettiCord

— User Manual —



Yuki Norizoe

AIST Tsukuba, 2014 - 2015

Cover illustration:
Examples of 3-dimensional simulation results.
Details are given in samples 6, 7, and 8.

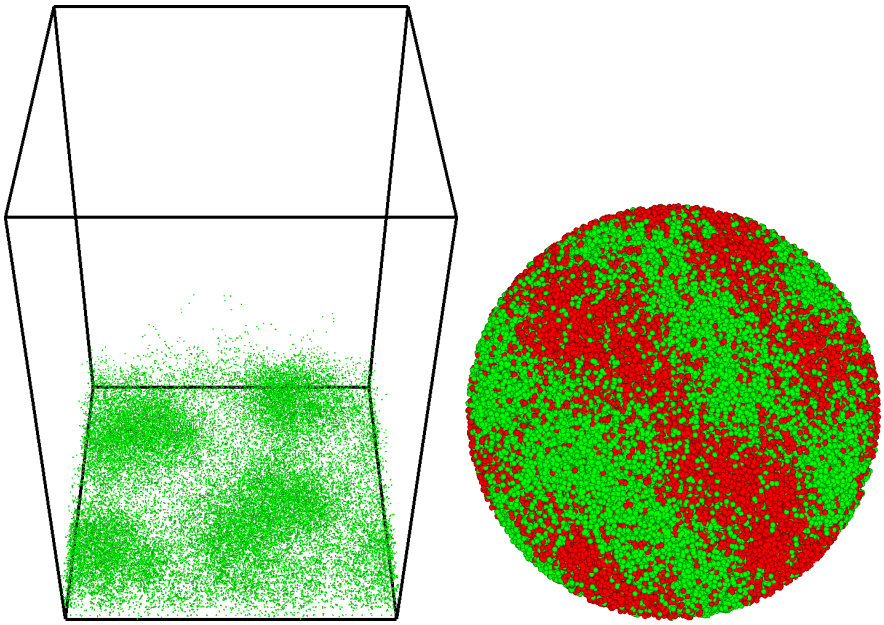


Fig. 1: (left panel) 2-dimensional microphase separation of single-component homopolymers grafted onto a planar substrate. Taken from sample 1. The same, but smaller, system as the polymer brush shown in Fig. 4(a) of a reference [2]. (right panel) Microphase separation of diblock copolymers confined in a spherical system box. Taken from sample 4.

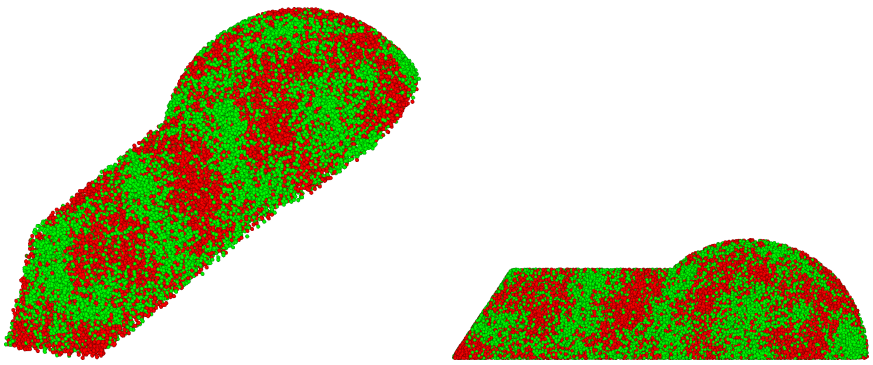


Fig. 2: Microphase separation of diblock copolymers confined in a Japanese ancient tomb, *zen-pou-kou-en-fun*. Both the panels are taken from sample 5.

Contents

Chapter 1

Introduction

This document is a user manual of "SpaghettiCord" – a molecular Monte Carlo simulation program to simulate collective phenomena of polymers in a coarse-grained scale.

Executable binary files of SpaghettiCord for Linux (x64) and Windows (x64) are provided. This program runs in collaboration with OCTA udf files. Simulation parameters are read from, and results written to the udf files. The definition of the udf parameters is also included in the present program package. Features of the program as well as these udf parameters are illustrated in chapter 2.

Source codes of SpaghettiCord are closed.

1.1 Basic requirements for utilizing SpaghettiCord

A basic understanding of both the standard NVT molecular Monte Carlo simulation method [1] and solvent-free model discussed in a reference [2] is required.

1.2 Usage

The simulation is performed through the following 5 steps:

1. A user sets simulation parameters to the udf file.

2. The executable file is run. The udf file name is given through the standard input after automatic prompting. The program automatically finishes the following steps 3 and 4, after this step 2 ends.

3. The simulation parameters are read from the udf file. The total number of polymers in the system and some other parameters are automatically determined and written to the udf file.

4. The simulation runs and finishes. Particle configurations are successively written to the udf file during the simulation run. Time evolutions of the

acceptance ratio, square end-to-end distance of polymers, and other physical quantities are also output to other text files during the run.

5. The simulation results are analysed.

Chapter 2

Features and parameters

Molecular Monte Carlo simulations of linear polymers are performed with the canonical ensemble in 3 dimensions *via* the standard Metropolis algorithm [1]. The thermal energy, k_BT, is chosen as the unit energy. (L_x, L_y, L_z) denotes the size of the rectangular parallelepiped system box. A periodic boundary condition is applied to the system. The system box is placed in spatial regions of $0 \leq \lambda < L_\lambda$, where λ denotes the Cartesian coordinates x, y and z. ΔL is the grid spacing to calculate the non-bonded interaction among the particles (segments). The Mersenne Twister algorithm is selected as a random number generator for the simulations [3–5]. In one simulation step, a particle is picked at random and given a uniform random trial displacement within a cube of edge length $2\Delta l$. A Monte Carlo step (MCS) is defined as N_{total} trial moves, during which each particle is selected for the trial displacement once on average, where N_{total} denotes the total number of particles of both the grafted and non-grafted ones in the system.

P0, P1, . . . , P9 denote the molecular (polymer) species, which illustrates that 10 molecular species can be simultaneously designed and utilized in the simulation. $N^{(\mathrm{P}j)}$ is the number of segments per polymer of Pj. Different molecular architecture can be set to each polymer species. For example, when P0 and P1 are a monomer and homopolymer, homopolymer solution can be simulated by mixing molecules of P0 and P1 in the system box. Polydisperse systems, mixture of diblock and triblock copolymers, and other complicated systems can also be simulated.

Each segment in the molecules can be spatially fixed, *i.e.* grafted, at any position. When one end of each polymer is grafted onto a hard substrate, polymer brushes are simulated. When monomers are fixed in the free space, this results in soft walls.

2.1 Molecular architecture

The molecular architecture of each polymer (molecular) species is described by three elements: a bead-spring potential, bending rigidity (chain stiffness) along the linear polymer chain, and configurations of the segment species along the chain.

Homogeneous springs and homogeneous chain stiffness are given to each polymer.

2.1.1 Linear springs

Linear springs linearly connect the adjacent segments in each polymer. The potential of each spring of Pj is denoted by,

$$\frac{H_{\rm spr}^{({\rm P}j)}}{k_BT} = \frac{1}{2}k_{\rm spr}^{({\rm P}j)}\left(l_{\rm adj} - l_{\rm natural}^{({\rm P}j)}\right)^2, \tag{2.1}$$

where $k_{\rm spr}^{({\rm P}j)}$ denotes the spring constant, $l_{\rm natural}^{({\rm P}j)}$ is the natural length of the springs, and $l_{\rm adj}$ is the distance between the centres of the pair of the connected segments. A value set of $k_{\rm spr}^{({\rm P}j)} = 3(N^{({\rm P}j)} - 1)/R_e^2$ and $l_{\rm natural}^{({\rm P}j)} = 0.0$, *e.g.* $k_{\rm spr}^{({\rm P}j)} = 3 \times 31/1 = 93.0$ with $N^{({\rm P}j)} = 32$, is typically chosen for monodisperse systems, where R_e denotes the root mean square of the end to end distance of an ideal chain with the same molecular architecture. For the monodisperse systems, this value set results in R_e = the unit length of the simulation system. This potential energy vanishes at $N^{({\rm P}j)} = 1$, i.e. monomers.

2.1.2 Chain stiffness

The chain stiffness of each polymer is provided as three-body potential among adjacent segment triples in a chain [6],

$$\frac{H_{\rm bending}^{({\rm P}j)}}{k_BT} = k_{\rm stiff}^{({\rm P}j)}\left\{1 - \cos\left(\phi - \phi_0^{({\rm P}j)}\right)\right\}, \tag{2.2}$$

where the angle ϕ is defined by the scalar product between the two bonds from an end to the centre of the triples and from the centre to the other end. $k_{\rm stiff}$ denotes the bending constant and $\phi_0^{({\rm P}j)}$ is the preferred angle. The chain stiffness vanishes when the polymerization degree ≤ 2.

2.1.3 Configuration of the segment species along the chain

The following configurations of the segment species along each polymer chain can be constructed:

- homopolymer (or monomer) of SA,
- homopolymer (or monomer) of SB,
- SA-SB (or SB-SA) diblock copolymer,
- SA-SB-SA triblock copolymer,

where SA and SB are segment species. Arbitrary value sets of the number of segments of each block and the total number of segments along the chain can be chosen.

2.2 Non-bonded interaction

Non-bonded interaction (or external interaction), denoted by $H_{\text{non-bonded}}$, is calculated utilizing the grid-based density. This calculation technique is discussed in a reference [2].

Three types of the non-bonded interactions are provided:

1. single-component solvent-free model,
2. binary solvent-free model,
3. interaction potential using χ-parameter.

Users of SpaghettiCord select one of the three for each simulation run. Simultaneous or combination use of two or more types is disallowed.

2.2.1 Single-component solvent-free model

The non-bonded interaction of the single-component solvent-free model is given as a functional of the local segment density. A third-order expansion of the non-bonded interaction free energy in a form of powers of the local segment density is employed:

$$\frac{H_{\text{non-bonded}}}{k_B T} := \int_V dV \left(-\frac{1}{2} v \left(\rho_{\text{SA}}(\boldsymbol{r})\right)^2 + \frac{1}{3} w \left(\rho_{\text{SA}}(\boldsymbol{r})\right)^3 \right), \tag{2.3}$$

where $\rho_\alpha(\boldsymbol{r})$ is the local volumetric number density of the segments of α segment species at the spatial position $\boldsymbol{r}$ and α denotes the segment species SA, SB. The positive constants, v and w, correspond to the attractive and repulsive interaction strengths among the segments, respectively. In the following calculation, dimensionless physical quantities are utilized: the local segment density $\rho'_\alpha(\boldsymbol{r}) = \rho_\alpha(\boldsymbol{r}) R_e^3$, parameter $w'' = w/{R_e}^6$, and parameter $v'' = v/R_e^3$. These dimensionless quantities reduce eq. (2.3) to:

$$\frac{H_{\text{non-bonded}}}{k_B T} = \int_V \frac{dV}{{R_e}^3} \left(-\frac{1}{2} v'' \left(\rho'_{\text{SA}}(\boldsymbol{r})\right)^2 + \frac{1}{3} w'' \left(\rho'_{\text{SA}}(\boldsymbol{r})\right)^3 \right). \tag{2.4}$$

Note that values of these dimensionless interaction parameters, v'' and w'', are directly set to the corresponding parameters in the udf file, although double prime symbols of these parameters are eliminated from the names of the udf parameters.

2.2.2 Binary solvent-free model

The non-bonded interaction of the binary solvent-free model [7, 8] is defined as a natural extension of the single-component solvent-free model. Dimensionless interaction parameters, $v''_{\alpha\beta} = v_{\alpha\beta}/R_e^3$ and $w''_{\alpha\beta\gamma} = w_{\alpha\beta\gamma}/R_e^6$ (where $\alpha, \beta, \gamma = \text{SA}, \text{SB}$

denote the segment species), reduce this non-bonded interaction of the binary model to:

$$\begin{aligned}\frac{H_{\text{non-bonded}}}{k_BT} &= \int_V \frac{dV}{{R_e}^3}\left[-\frac{1}{2}v''_{\text{AA}}\left(\rho'_{\text{SA}}(\boldsymbol{r})\right)^2 - \frac{1}{2}v''_{\text{BB}}\left(\rho'_{\text{SB}}(\boldsymbol{r})\right)^2 + v''_{\text{AB}}\rho'_{\text{SA}}(\boldsymbol{r})\rho'_{\text{SB}}(\boldsymbol{r})\right. \\ &\left. +\frac{1}{3}\left\{w''_{\text{AAA}}\left(\rho'_{\text{SA}}(\boldsymbol{r})\right)^3 + 3w''_{\text{AAB}}\left(\rho'_{\text{SA}}(\boldsymbol{r})\right)^2\rho'_{\text{SB}}(\boldsymbol{r}) + 3w''_{\text{ABB}}\rho'_{\text{SA}}(\boldsymbol{r})\left(\rho'_{\text{SB}}(\boldsymbol{r})\right)^2 + w''_{\text{BBB}}\left(\rho'_{\text{SB}}(\boldsymbol{r})\right)^3\right\}\right].\end{aligned} \tag{2.5}$$

Note that values of these dimensionless interaction parameters, $v''_{\alpha\beta}$ and $w''_{\alpha\beta\gamma}$, are directly set to the corresponding parameters in the udf file, although double prime symbols of these parameters are eliminated from the names of the udf parameters.

2.2.3 Interaction potential using χ-parameter

The non-bonded interaction potential using χ-parameter is defined as [9]:

$$\frac{H_{\text{non-bonded}}}{k_BT} = \sum_{\boldsymbol{a}}\left[\rho\Delta V\left(\chi\phi_A\phi_B + \frac{1}{2}\kappa\left(\phi_A + \phi_B - \phi_0\right)^2\right)\right] \tag{2.6}$$

where $\boldsymbol{a}$ denotes the collocation grid, ρ is the average number density of the total segments in the system, and $\Delta V = (\Delta L)^3$ is the volume of a cubic cell of the collocation grid. $\phi_\alpha = U_\alpha/(\rho\Delta V)$ where U_α denotes the number of the α-segments in the cubic cell. The reference density, ϕ_0, is fixed at unity, *i.e.* $\phi_0 = 1$.

Value sets of χ, κ, and $\rho\Delta V$ are set to the udf file, where $\rho\Delta V$ is the average number of the total segments per cubic cell.

2.3 System size and grid spacing

The system size L_x is directly set to the udf file. Approximate values of $L_y/L_x, L_z/L_x$, and the grid spacing for the non-bonded interaction ΔL are also set to the udf file. Values of L_y, L_z, and ΔL actually utilized in the simulation are automatically determined through the following code using these input parameters. This results in the homogeneous and isotropic grids in the system.

```
// ( Lx, Ly, Lz ) denotes the system size, and "GridSpacing"
// is the grid spacing.
// "LyOverLx" and "LzOverLx" are the given approximate values
// of Ly/Lx and Lz/Lx, respectively.
// "ApproxGridSpacing" is the given approximate value
// of the grid spacing.

// The number of grids along x-axis
```

```
mx = (int)( Lx / ApproxGridSpacing ) ;

// Actual value of the grid spacing is determined.
GridSpacing = Lx / mx ;

// The number of grids along y and z-axis respectively
my = (int)( LyOverLx * mx ) ;
mz = (int)( LzOverLx * mx ) ;

// Values of Ly and Lz are determined.
Ly = my * GridSpacing ;
Lz = mz * GridSpacing ;
```

2.4 Data structure of particles

Here `class CParticle`, which is the data structure of the particles written to the udf file as records, is illustrated.

`ParticleNumber` denotes the serial number of the particles for each molecular species Pj. This ranges in $[0, n_p^{(\mathrm{P}j)} N^{(\mathrm{P}j)})$, where $n_p^{(\mathrm{P}j)}$ is the number of polymers of Pj in the system. The value of this parameter divided by $N^{(\mathrm{P}j)}$ equals the index of the polymer (polymer number) to which the particle belongs. This index ranges in $[0, n_p^{(\mathrm{P}j)})$.

`SegmentNumber` is the segment number of the particle in each polymer. This parameter ranges in $[0, N^{(\mathrm{P}j)})$. For example, this parameter of monomers is always fixed at 0.

The rectangular parallelepiped system box occupies the spatial region of $0 \leq \lambda < L_\lambda$. This box is often referred to as the basic cell. On the other hand, an infinite number of the identical boxes also lie in and fill the whole space because the periodic boundary condition is applied to the system. These boxes outside the region of the basic cell are referred to as image cells. Each image cell is represented by indices (i_x, i_y, i_z) and occupies the spatial region of $i_\lambda L_\lambda \leq \lambda < (i_\lambda + 1)L_\lambda$. The image cell at $i_x = i_y = i_z = 0$ corresponds to the basic cell. Udf parameters `x, y, z` of `class CParticle` denote the coordinates of the particle in the basic cell. Udf parameters `ImageCellX, ImageCellY, ImageCellZ` are the indices of the image cell, (i_x, i_y, i_z), where the particle is located.

2.5 Time evolution of some physical quantities

In addition to the particle configurations written to the udf file, time evolution of some physical quantities are also written to other plain text files. Each text file begins with one line of the header information of the file. This header shows what is recorded in each column of the text file. The 1st column is always “MCS”.

Udf parameter `Logging.SamplingRateEnergyETC` determines the interval of sampling for these physical quantities in MCS. Arbitrary values can be set to this parameter independently of the sampling rate for the particle configuration, `Logging.SamplingRate`.

Udf parameter `Logging.FileNamePrefix` determines the prefix of the names of these text files. If empty, the present udf file name is chosen. Here we assume that "sample" is given to this parameter.

`sample-AcceptanceRatio.txt` shows the time evolution of the acceptance ratio of the simulation. Note that the grafted particles always reject the trial. This decreases the acceptance ratio recorded in this file.

The computational time is recorded in `sample-ComputationalTime.txt`. This is measured using the wall-clock time.

The time evolution of the average square end to end distance of a polymer for each molecular species is recorded in `sample-SquareEndToEndDistance.txt`.

The time evolution of the average square gyration radius of a polymer for each molecular species is recorded in `sample-SquareGyrationRadius.txt`.

`sample-PotentialEnergy.txt` shows the time evolution of the total potential energy in the system. Each column, excluding the 1st one, of this file is an element of the total energy. For example, when the non-bonded interaction potential using χ-parameter is chosen for the simulation, the 2nd column of this file shows the total energy in the system resulting from the first term of the right-hand side of eq. (2.6). The other cases are illustrated in tables 2.1 and 2.2.

`sample-InitialEnsembleAndSystem.log` is a log file for debugging purposes.

Table 2.1: Meanings of columns when the single-component or binary solvent-free model is chosen.

column	meanings
1	MCS
2	all the terms including v'' or $v''_{\alpha\beta}$ in the right-hand sides of equations (2.4) and (2.5)
3	all the terms including w'' or $w''_{\alpha\beta\gamma}$ in the right-hand sides of equations (2.4) and (2.5)
4	always zero (reserved)
5	the total non-bonded interaction
6	linear springs, eq. (2.1)
7	chain stiffness, eq. (2.2)
8	always zero (reserved)
9	the total interaction potential energy in the system

Table 2.2: Meanings of columns when the non-bonded interaction potential using χ-parameter is chosen.

column	meanings
1	MCS
2	the first term of the right-hand side of eq. (2.6)
3	the second term of the right-hand side of eq. (2.6)
4	always zero (reserved)
5	the total non-bonded interaction
6	linear springs, eq. (2.1)
7	chain stiffness, eq. (2.2)
8	always zero (reserved)
9	the total interaction potential energy in the system

2.6 Global udf parameters

Here the global udf parameters are illustrated. udf classes utilized in these udf parameters are shown in sec. 2.8.

2.6.1 `SystemSize`

System size. The rectangular system box is placed in spatial regions of $0 \leq \alpha < L_\alpha$, where α denotes x, y, and z. See also sec. 2.3.

- `Lx: double`
  System size, L_x.

- `LyOverLx: double`
  Approximate value of L_y/L_x.

- `LzOverLx: double`
  Approximate value of L_z/L_x.

- `ApproxGridSpacing: double`
  Approximate value of the grid spacing for the external (or non-bonded) interaction.

2.6.2 `Resume`

Parameters to resume the simulation.

- `ResumeSimulation: select { "true", "false" }`
  If true, the simulation is resumed from the last particle configuration. If false, the initial particle configuration is automatically arranged according to given parameters for initialization.

- `LastParticleConfigFileName: string`
  The udf file name from which the last particle configuration is loaded if `"ResumeSimulation"` is true. When this parameter is left empty, the present udf file is automatically chosen. If `"ResumeSimulation"` is false, this parameter is ignored.

2.6.3 ExternalInteraction

Parameters for external or non-bonded interaction.

- `HamiltonianType: select { "OFF", "Chi", "SolventFree1Component", "SolventFreeBinary" }`
  Type of Hamiltonian, i.e. external or non-bonded interaction. When `"OFF"` is chosen, the interaction is off, which results in ideal polymers.

- `Chi:`
  - `ParameterChi: double`
    Interaction parameter χ.
  - `ParameterKappa: double`
    Interaction parameter κ.
  - `AverageNumberOfParticlesPerFieldCell: double`
    The average number of particles per cell.

- `SolventFree1Component:`
  - `ParameterW: double`
    Interaction parameter w''.
  - `ParameterV: double`
    Interaction parameter v''.

- `SolventFreeBinary:`
  - `ParameterVAA: double`
    Interaction parameter v''_{AA}.
  - `ParameterVBB: double`
    Interaction parameter v''_{BB}.
  - `ParameterVAB: double`
    Interaction parameter v''_{AB}.
  - `ParameterWAAA: double`
    Interaction parameter w''_{AAA}.
  - `ParameterWAAB: double`
    Interaction parameter w''_{AAB}.
  - `ParameterWABB: double`
    Interaction parameter w''_{ABB}.

  - `ParameterWBBB: double`
    Interaction parameter w''_{BBB}.

2.6.4 Logging

Parameters to control logging.

- `SamplingRate: int`
  Interval of sampling in MCS.
- `NumberOfSamples: int`
  The number of samples, i.e. records, excluding the initial configuration.
- `SamplingRateEnergyETC: int`
  Interval of sampling for energy etc. in MCS.
- `FileNamePrefix: string`
  Prefix of the log file names. If empty, the present udf file name is chosen.

2.6.5 Miscellaneous

Miscellaneous.

- `TrialDisplacementMax: double`
  Maximum trial displacement. In one simulation step, a particle is chosen at random and given a uniform random trial displacement within a cube of edge length = 2 × (this parameter).

2.6.6 MolecularArch[]: CPolymerStructure

Molecular architecture. 1 to 10 polymer species can be simultaneously set and used: P0, P1, P2, ... , P9. The polymer species which are not set here are ignored. For example, when P0, P1, and P2 species are set here, only these species can be utilized in the simulation, and the other species cannot. When more than 10 polymer species are set, only the first 10 species are utilized, and the others are ignored.

2.6.7 HardWalls

Hard walls, i.e. hard planes and hard blocks. The hard planes are placed parallel to *xy*-plane.

- `MasterSwitch: select { "ON", "OFF" }`
  Master switch, i.e. switch for all the hard walls.
- `SwitchHardPlanes: select { "ON", "OFF" }`
  Switch for planar hard walls, i.e. hard planes.

- `Switch3DHardBlocks: select { "ON", "OFF" }`
  Switch for 3D hard walls, i.e. hard blocks.

- `HardPlanesZ[]: double`
  z-coordinates of hard planes parallel to xy-plane. Hard planes outside the system box, i.e. in regions of $z < 0$ or $L_z \leq z$, are ignored.

- `HardBlocks[]: CGeometricShape3D`
  Hard blocks. Set 3D geometric shapes as the hard blocks. Non-grafted particles are disallowed to be positioned on the surface of or inside the hard blocks. Regions of these 3D shapes outside the system box are ignored. Note that the non-grafted particles can pass through or leap the hard blocks. This means that the trial displacement of the non-grafted particles is allowed when both the starting and ending points of the displacement are located outside the hard blocks, even though the hard blocks are found between these points.

- `CellSizeForHardBlock: double`
  Cell size for the cell lists for the hard blocks. An arbitrary positive value smaller than L_x, L_y, and L_z can be set. Typically, the value of the grid spacing for the interaction potential can be an acceptable value.

2.6.8 IniParticleConf[]: CInitialNumberAndArrangementParticles

Initial particle configuration.

2.6.9 MersenneTwister

Parameters for Mersenne Twister, a pseudorandom number generator.

- `AutoSeed: select { "true", "false" }`
  Option mostly for debugging purposes. This should be fixed at `"true"` for usual production runs. If true, the random seed is automatically determined utilizing the system clock. If false, the seed is manually given through the following variable.

- `RandomSeed: long`
  Random seed of Mersenne Twister, zero or a positive value. This parameter is ignored when `"AutoSeed"` is true.

2.6.10 Output

Output data. These parameters are automatically determined based on the input data.

- `NumberOfParticlesAndPolymers:` The numbers of particles and polymers.
  - `n_P0: int`
    The number of polymers, P0.
  - `n_P1: int`
    The number of polymers, P1.
  - `n_P2: int`
    The number of polymers, P2.
  - `n_P3: int`
    The number of polymers, P3.
  - `n_P4: int`
    The number of polymers, P4.
  - `n_P5: int`
    The number of polymers, P5.
  - `n_P6: int`
    The number of polymers, P6.
  - `n_P7: int`
    The number of polymers, P7.
  - `n_P8: int`
    The number of polymers, P8.
  - `n_P9: int`
    The number of polymers, P9.
  - `N_SA: int`
    The number of segments, SA.
  - `N_SB: int`
    The number of segments, SB.
  - `N_SC: int`
    The number of segments, SC.
  - `N_total: arraysize`
    The total number of particles.
- `SystemSize:` System size. See also sections 2.3 and 2.6.1.
  - `Ly: double`
    System size, L_y.
  - `Lz: double`
    System size, L_z.
  - `V: double`
    System volume, V.

  - `GridSpacing: double`
    Grid spacing.

- `RandomSeed: long`
  Random seed of Mersenne Twister, zero or a positive value.

2.7 udf parameters for records

Here udf parameters for records are illustrated. These are all output data, i.e. simulation results.

- `MCS: int`
  MCS at the present record. 1 MCS (Monte Carlo step) is defined as `N_total` trial moves, during which each particle is selected for the trial displacement once on average, where `N_total` denotes the total number of particles (segments) in the system. This `N_total` includes both the grafted and non-grafted particles.

- `Particles[ N_total ]: CParticle`
  Array for particles. See sections 2.4 and 2.8.4.

2.8 udf classes

2.8.1 udf classes for geometric shapes

2.8.1-a `class CPoint_2D`

Point in 2-D.

- `x: double`
  x-coordinate.

- `y: double`
  y-coordinate.

2.8.1-b `class CPoint_3D`

Point in 3-D.

- `x: double`
  x-coordinate.

- `y: double`
  y-coordinate.

- `z: double`
  z-coordinate.

2.8.1-c class CCircle

Circle.

- `Centre: CPoint_2D`
  Centre coordinates.
- `r: double`
  Radius.

2.8.1-d class CRectangle

Rectangle. Each side is placed parallel to x or y-axis.

- `x_min: double`
  min. x-coordinate.
- `x_max: double`
  max. x-coordinate.
- `y_min: double`
  min. y-coordinate.
- `y_max: double`
  max. y-coordinate.

2.8.1-e class CTriangle

Triangle.

- `Vertex1st: CPoint_2D`
  1st vertex.
- `Vertex2nd: CPoint_2D`
  2nd vertex.
- `Vertex3rd: CPoint_2D`
  3rd vertex.

2.8.1-f class CCircularSector

Circular sector. This ranges in [`phi` − `theta`/2, `phi` + `theta`/2], where `theta` denotes the central angle and `phi` does the direction. `theta` should be set in $[0, 2\pi]$.

- `Centre: CPoint_2D`
  Centre coordinates.
- `r: double`
  Radius.

- `theta: double [rad]`
  Central angle. This should be set in $[0, 2\pi]$.
- `phi: double [rad]`
  Direction.

2.8.1-g class CQuadrilateral

Quadrilateral. This is composed of a pair of triangles: triangles with the 1st - 3rd vertices and with 2nd - 4th vertices. The line segment between the 2nd and 3rd vertices is shared between these two triangles.

- `Vertex1st: CPoint_2D`
  1st vertex.
- `Vertex2nd: CPoint_2D`
  2nd vertex.
- `Vertex3rd: CPoint_2D`
  3rd vertex.
- `Vertex4th: CPoint_2D`
  4th vertex.

2.8.1-h class CBasicGeometricShape2D

2-D basic geometric shape.

- `BasicShape2D: select { "circle", "rectangle", "triangle", "CircularSector", "quadrilateral" }`
  Basic 2-D shape.
- `circle: CCircle`
- `rectangle: CRectangle`
- `triangle: CTriangle`
- `CircularSector: CCircularSector`
- `quadrilateral: CQuadrilateral`
- `Inverse: select { "ON", "OFF" }`
  If ON, the region of the geometric shape is inverted.

2.8.1-i class CMultipleShapeIntersection2D

The intersection of 2-D geometric shapes, i.e. a 2-D multiple geometric shape.

- `ComponentShapes2D[]: CBasicGeometricShape2D`
  Array of component 2-D basic shapes. Regions of the component shapes with "`Inverse == ON`" are removed from the resulting multiple shape.
- `Inverse: select { "ON", "OFF" }`
  If ON, the region of the geometric shape is inverted.

2.8.1-j class CMultipleShapeUnion2D

The union of 2-D geometric shapes, i.e. a 2-D multiple geometric shape.

- `ComponentShapes2D[]: CBasicGeometricShape2D`
  Array of component 2-D basic shapes. Regions of the component shapes with "`Inverse == ON`" are removed from the resulting multiple shape.
- `Inverse: select { "ON", "OFF" }`
  If ON, the region of the geometric shape is inverted.

2.8.1-k class CGeometricShape2D

2-D geometric shape.

- `Shape2D: select { "basic", "MultipleIntersection", "MultipleUnion" }`
  2-D shape.
- `basic: CBasicGeometricShape2D`
- `MultipleIntersection: CMultipleShapeIntersection2D`
- `MultipleUnion: CMultipleShapeUnion2D`

2.8.1-l class CSphere

Sphere.

- `Centre: CPoint_3D`
  Centre coordinates.
- `r: double`
  Radius.

2.8.1-m `class CCuboid`

Cuboid. Each edge is placed parallel to x, y, or z-axis.

- `x_min: double`
  min. x-coordinate.
- `x_max: double`
  max. x-coordinate.
- `y_min: double`
  min. y-coordinate.
- `y_max: double`
  max. y-coordinate.
- `z_min: double`
  min. z-coordinate.
- `z_max: double`
  max. z-coordinate.

2.8.1-n `class CCylinder`

Cylinder. The axis is placed parallel to z-axis.

- `Base: CCircle`
  Base.
- `z_min: double`
  min. z-coordinate.
- `z_max: double`
  max. z-coordinate.

2.8.1-o `class CRectangularFrustum`

3-D geometric shape including, similar to, and more general than rectangular frustums, oblique rectangular prisms, etc. The lower and upper rectangular bases of this shape are placed parallel to xy-plane. Each edge of the bases is placed parallel to x or y-axis. These two rectangular bases can be independently set. This means that these bases can be independent in the size as well as the position. Four side surfaces of this shape are all trapezoids. This 3-D shape is equivalent to a rectangular frustum when the bases are similar, and an oblique rectangular prism when congruent.

- `LowerBase: CRectangle`
  Lower rectangular base.

- `z_min: double`
  min. z-coordinate.

- `UpperBase: CRectangle`
  Upper rectangular base.

- `z_max: double`
  max. z-coordinate.

2.8.1-p class CBasicGeometricShape3D

3-D basic geometric shape.

- `BasicShape3D: select { "sphere", "cuboid", "cylinder", "RectangularFrustum" }`
  Basic 3-D shape.

- `sphere: CSphere`

- `cuboid: CCuboid`

- `cylinder: CCylinder`

- `RectangularFrustum: CRectangularFrustum`

- `Inverse: select { "ON", "OFF" }`
  If ON, the region of the geometric shape is inverted.

2.8.1-q class CMultipleShapeIntersection3D

The intersection of 3-D geometric shapes, i.e. a 3-D multiple geometric shape.

- `ComponentShapes3D[]: CBasicGeometricShape3D`
  Array of component 3-D basic shapes. Regions of the component shapes with "`Inverse == ON`" are removed from the resulting multiple shape.

- `Inverse: select { "ON", "OFF" }`
  If ON, the region of the geometric shape is inverted.

2.8.1-r class CMultipleShapeUnion3D

The union of 3-D geometric shapes, i.e. a 3-D multiple geometric shape.

- `ComponentShapes3D[]: CBasicGeometricShape3D`
  Array of component 3-D basic shapes. Regions of the component shapes with `"Inverse == ON"` are removed from the resulting multiple shape.

- `Inverse: select { "ON", "OFF" }`
  If ON, the region of the geometric shape is inverted.

2.8.1-s `class CGeometricShape3D`

3-D geometric shape.

- `Shape3D: select { "basic", "MultipleIntersection", "MultipleUnion" }`
  3-D shape.

- `basic: CBasicGeometricShape3D`

- `MultipleIntersection: CMultipleShapeIntersection3D`

- `MultipleUnion: CMultipleShapeUnion3D`

2.8.2 udf classes for molecular architecture

2.8.2-a `class CPolymerStructure`

Polymer structure. Each linear polymer has the homogeneous bonds and homogeneous chain stiffness over the polymer chain. When the polymerization degree equals one, the polymer is equivalent to a monomer.

- `BlockLengthA1: int`
  The number of segments in the first A-block, >= 0. The linear molecular architecture is: (1st A-block)-(B-block)-(2nd A-block). The length of each block can be zero. For example, the molecule is equivalent to a homopolymer of A-species at B-block = zero, and equivalent to an A-B diblock copolymer at either (1st A-block) = zero or (2nd A-block) = zero. When the total number of the segments, i.e. the polymerization degree, equals one, the molecule is equivalent to a monomer. When zero, the polymer species cannot be utilized.

- `BlockLengthB: int`
  The number of segments in the B-block, >= 0.

- `BlockLengthA2: int`
  The number of segments in the second A-block, >= 0.

- `SpringConstant: double`
  Spring constant of linear springs between the segments. The same spring const. and natural length are given to all the springs in each polymer. These parameters of the springs are ignored when the polymerization degree equals one, i.e. for a monomer.

- `NaturalLengthOfSpring: double`
  Natural length of the linear springs.

- `BendingConstant: double`
  Bending constant of the chain stiffness potential. The chain stiffness of the polymer is provided as three-body potential among three neighboring segments. Therefore, the parameters for the chain stiffness are ignored when the polymerization degree <= 2.

- `PreferredBondAngle: double`
  Preferred bond angle of the chain stiffness potential. When this parameter equals zero, the minimum of the chain stiffness potential occurs for parallel bonds in a chain.

2.8.3 udf classes for initial particle configuration

2.8.3-a class CInitialPolymerConformationGaussianRandom

Gaussian random polymer conformation.

- `SignOf0thTo1stSegmentZ: select { "POSITIVE", "NEUTRAL", "NEGATIVE" }`
  Sign of the relative z-coordinate from the head segment (0th particle) to the next segment (1st particle) of each polymer. If `"POSITIVE"` (or `"NEGATIVE"`), always positive (or negative). This means that, when the gaussian random number generator provides a negative (or positive) value, this is automatically turned positive (or negative) keeping the same absolute value. For example, this is chosen when a single-sided polymer brush on a planar substrate is simulated. If `"NEUTRAL"`, the above automatic adjustment of the sign is switched off.

2.8.3-b class CInitialPolymerConformationHomogeneousLinear

Segments of each polymer are placed at regular intervals on a straight line from the head segment.

- `IntervalBetweenSegments: double`
  Intervals between the adjacent segments.

- `Longitude: double`
  Direction of the straight line, longitude (ϕ in typical geometric notation).

- `Latitude: double`
  Direction of the straight line, latitude (θ in typical geometric notation).

2.8.3-c `class CPolymerConformation`

Polymer conformation.

- `ConformationType: select { "gaussian", "straight" }`
  Type of polymer conformation.
- `gaussian: CInitialPolymerConformationGaussianRandom`
- `straight: CInitialPolymerConformationHomogeneousLinear`

2.8.3-d `class CInitialHeadOn3DPoint`

The head segments are placed on a given 3-D point.

- `PolymerSpecies: select { "P0", "P1", "P2", "P3", "P4", "P5", "P6", "P7", "P8", "P9" }`
  Polymer species.
- `NumberOfPolymers: int`
  The number of placed polymers.
- `Graft: select { "OFF", "FIRST_END", "LAST_END", "BOTH_ENDS", "MIDDLE_POINT", "ALL" }`
  Grafting type.

  `"OFF"`: No segments are grafted. In other words, no segments are spatially fixed.

  `"FIRST_END"`: The first end, i.e. the 0th (head) segment, of each polymer (or each segment as a polymer species) is grafted.

  `"LAST_END"`: The last end, i.e. the N-th ($(N-1)$-th) segment, of each polymer (or each segment as a polymer species) is grafted.

  `"BOTH_ENDS"`: Both the ends, i.e. both the 0th (head) and N-th ($(N-1)$-th) segments, of each polymer (or each segment as a polymer species) are grafted.

  `"MIDDLE_POINT"`: The middle point between both the ends, i.e. $(N/2)$-th segment, of each polymer (or each segment as a polymer species) is grafted.

  `"ALL"`: All the segments of each polymer (or each segment as a polymer species) are grafted.

- `Point3DForHead: CPoint_3D`
  The 3-D point where the head segments are placed.
- `PolymerConformation: CPolymerConformation`
  Polymer conformation.
- `IgnoredHardBlockIndex: int`
  The index of the hard block to ignore for the arrangement of the head (0th) segments. When a negative value or one larger than the maximum is given to this parameter, no hard blocks are ignored, i.e. all the hard blocks are checked.

2.8.3-e class CInitialHeadOnSquareLattice2D

Arranging the head (0th) segments of each polymer on 2-D geometric shape lying parallel with *xy*-plane. The head segments are placed on 3-D points at square lattice points on or inside the 2-D shape, i.e. including the perimeter of the shape.

- `PolymerSpecies: select { "P0", "P1", "P2", "P3", "P4", "P5", "P6", "P7", "P8", "P9" }`
  Polymer species.
- `Graft: select { "OFF", "FIRST_END", "LAST_END", "BOTH_ENDS", "MIDDLE_POINT", "ALL" }`
  Grafting type.

  `"OFF"`: No segments are grafted. In other words, no segments are spatially fixed.

  `"FIRST_END"`: The first end, i.e. the 0th (head) segment, of each polymer (or each segment as a polymer species) is grafted.

  `"LAST_END"`: The last end, i.e. the N-th ($(N-1)$-th) segment, of each polymer (or each segment as a polymer species) is grafted.

  `"BOTH_ENDS"`: Both the ends, i.e. both the 0th (head) and N-th ($(N-1)$-th) segments, of each polymer (or each segment as a polymer species) are grafted.

  `"MIDDLE_POINT"`: The middle point between both the ends, i.e. $(N/2)$-th segment, of each polymer (or each segment as a polymer species) is grafted.

  `"ALL"`: All the segments of each polymer (or each segment as a polymer species) are grafted.
- `ShapeForHeadSegment: CGeometricShape2D`
  2-D geometric shape, on which the head segments are placed.
- `z: double`
  z-coordinate, at which the 2-D geometric shape is placed. Actually, z-coordinate of the head segments.

- `PolymerConformation: CPolymerConformation`
  Polymer conformation.

- `LatticeConst: double`
  Lattice constant.

- `NumberOfHeadsPerLattice: int`
  The number of the head segments distributed to each lattice point. The number of heads per lattice point. This equals the number of polymers per lattice point.

- `IgnoredHardBlockIndex: int`
  The index of the hard block to ignore for the arrangement of the head (0th) segments. When a negative value or one larger than the maximum is given to this parameter, no hard blocks are ignored, i.e. all the hard blocks are checked.

2.8.3-f `class CInitialHeadOnCubicLattice3D`

Arranging the head (0th) segments of each polymer on 3-D geometric shape. The head segments are placed on 3-D points at cubic lattice points inside the 3-D shape, including the surface.

- `PolymerSpecies: select { "P0", "P1", "P2", "P3", "P4", "P5", "P6", "P7", "P8", "P9" }`
  Polymer species.

- `Graft: select { "OFF", "FIRST_END", "LAST_END", "BOTH_ENDS", "MIDDLE_POINT", "ALL" }`
  Grafting type.

  `"OFF"`: No segments are grafted. In other words, no segments are spatially fixed.

  `"FIRST_END"`: The first end, i.e. the 0th (head) segment, of each polymer (or each segment as a polymer species) is grafted.

  `"LAST_END"`: The last end, i.e. the N-th ($(N-1)$-th) segment, of each polymer (or each segment as a polymer species) is grafted.

  `"BOTH_ENDS"`: Both the ends, i.e. both the 0th (head) and N-th ($(N-1)$-th) segments, of each polymer (or each segment as a polymer species) are grafted.

  `"MIDDLE_POINT"`: The middle point between both the ends, i.e. $(N/2)$-th segment, of each polymer (or each segment as a polymer species) is grafted.

  `"ALL"`: All the segments of each polymer (or each segment as a polymer species) are grafted.

- `ShapeForHeadSegment: CGeometricShape3D`
  3-D geometric shape, on which the head segments are placed.
- `PolymerConformation: CPolymerConformation`
  Polymer conformation.
- `LatticeConst: double`
  Lattice constant.
- `NumberOfHeadsPerLattice: int`
  The number of the head segments distributed to each lattice point. The number of heads per lattice point. This equals the number of polymers per lattice point.
- `IgnoredHardBlockIndex: int`
  The index of the hard block to ignore for the arrangement of the head (0th) segments. When a negative value or one larger than the maximum is given to this parameter, no hard blocks are ignored, i.e. all the hard blocks are checked.

2.8.3-g class CInitialHeadRandomSurface2D

Arranging the head (0th) segments of each polymer on 2-D geometric shape lying parallel with *xy*-plane. Random distribution using rejection sampling. Note that the acceptance ratio of this rejection sampling depends on the geometric shapes. This could result in the extremely low acceptance ratio. The simulation program automatically aborts when the random numbers cannot be accepted in acceptable computational time.

- `PolymerSpecies: select { "P0", "P1", "P2", "P3", "P4", "P5", "P6", "P7", "P8", "P9" }`
  Polymer species.
- `NumberOfPolymers: int`
  The number of placed polymers.
- `Graft: select { "OFF", "FIRST_END", "LAST_END", "BOTH_ENDS", "MIDDLE_POINT", "ALL" }`
  Grafting type.

  `"OFF"`: No segments are grafted. In other words, no segments are spatially fixed.

  `"FIRST_END"`: The first end, i.e. the 0th (head) segment, of each polymer (or each segment as a polymer species) is grafted.

  `"LAST_END"`: The last end, i.e. the N-th ($(N-1)$-th) segment, of each polymer (or each segment as a polymer species) is grafted.

"BOTH_ENDS": Both the ends, i.e. both the 0th (head) and N-th ($(N-1)$-th) segments, of each polymer (or each segment as a polymer species) are grafted.

"MIDDLE_POINT": The middle point between both the ends, i.e. $(N/2)$-th segment, of each polymer (or each segment as a polymer species) is grafted.

"ALL": All the segments of each polymer (or each segment as a polymer species) are grafted.

- ShapeForHeadSegment: CGeometricShape2D
  2-D geometric shape, on which the head segments are placed.

- z: double
  z-coordinate, at which the 2-D geometric shape is placed. Actually, z-coordinate of the head segments.

- PolymerConformation: CPolymerConformation
  Polymer conformation.

- IgnoredHardBlockIndex: int
  The index of the hard block to ignore for the arrangement of the head (0th) segments. When a negative value or one larger than the maximum is given to this parameter, no hard blocks are ignored, i.e. all the hard blocks are checked.

2.8.3-h class CInitialHeadRandomSurface3D

Arranging the head (0th) segments of each polymer on 3-D geometric shape. Random distribution on surface of the 3-D shape using rejection sampling. Note that the acceptance ratio of this rejection sampling depends on the geometric shapes. This could result in the extremely low acceptance ratio. The simulation program automatically aborts when the random numbers cannot be accepted in acceptable computational time.

- PolymerSpecies: select { "P0", "P1", "P2", "P3", "P4", "P5", "P6", "P7", "P8", "P9" }
  Polymer species.

- NumberOfPolymers: int
  The number of placed polymers.

- Graft: select { "OFF", "FIRST_END", "LAST_END", "BOTH_ENDS", "MIDDLE_POINT", "ALL" }
  Grafting type.

  "OFF": No segments are grafted. In other words, no segments are spatially fixed.

"FIRST_END": The first end, i.e. the 0th (head) segment, of each polymer (or each segment as a polymer species) is grafted.

"LAST_END": The last end, i.e. the N-th ($(N-1)$-th) segment, of each polymer (or each segment as a polymer species) is grafted.

"BOTH_ENDS": Both the ends, i.e. both the 0th (head) and N-th ($(N-1)$-th) segments, of each polymer (or each segment as a polymer species) are grafted.

"MIDDLE_POINT": The middle point between both the ends, i.e. $(N/2)$-th segment, of each polymer (or each segment as a polymer species) is grafted.

"ALL": All the segments of each polymer (or each segment as a polymer species) are grafted.

- ShapeForHeadSegment: CGeometricShape3D
  3-D geometric shape, on which the head segments are placed.
- PolymerConformation: CPolymerConformation
  Polymer conformation.
- IgnoredHardBlockIndex: int
  The index of the hard block to ignore for the arrangement of the head (0th) segments. When a negative value or one larger than the maximum is given to this parameter, no hard blocks are ignored, i.e. all the hard blocks are checked.

2.8.3-i class CInitialHeadRandomVolumetric3D

Arranging the head (0th) segments of each polymer on 3-D geometric shape. Random distribution inside the 3-D shape, including the surface, using rejection sampling. Note that the acceptance ratio of this rejection sampling depends on the geometric shapes. This could result in the extremely low acceptance ratio. The simulation program automatically aborts when the random numbers cannot be accepted in acceptable computational time.

- PolymerSpecies: select { "P0", "P1", "P2", "P3", "P4", "P5", "P6", "P7", "P8", "P9" }
  Polymer species.
- NumberOfPolymers: int
  The number of placed polymers.
- Graft: select { "OFF", "FIRST_END", "LAST_END", "BOTH_ENDS", "MIDDLE_POINT", "ALL" }
  Grafting type.

  "OFF": No segments are grafted. In other words, no segments are spatially fixed.

"FIRST_END": The first end, i.e. the 0th (head) segment, of each polymer (or each segment as a polymer species) is grafted.

"LAST_END": The last end, i.e. the N-th ($(N-1)$-th) segment, of each polymer (or each segment as a polymer species) is grafted.

"BOTH_ENDS": Both the ends, i.e. both the 0th (head) and N-th ($(N-1)$-th) segments, of each polymer (or each segment as a polymer species) are grafted.

"MIDDLE_POINT": The middle point between both the ends, i.e. $(N/2)$-th segment, of each polymer (or each segment as a polymer species) is grafted.

"ALL": All the segments of each polymer (or each segment as a polymer species) are grafted.

- ShapeForHeadSegment: CGeometricShape3D
  3-D geometric shape, on which the head segments are placed.

- PolymerConformation: CPolymerConformation
  Polymer conformation.

- IgnoredHardBlockIndex: int
  The index of the hard block to ignore for the arrangement of the head (0th) segments. When a negative value or one larger than the maximum is given to this parameter, no hard blocks are ignored, i.e. all the hard blocks are checked.

2.8.3-j class CInitialNumberAndArrangementParticles

Initial particle configuration.

- ConfigurationType: select { "point3D", "SquareLattice2D", "CubicLattice3D", "RandomSurface2D", "RandomSurface3D", "RandomVolumetric3D" }
  Type of the initial particle configuration.

- point3D: CInitialHeadOn3DPoint

- SquareLattice2D: CInitialHeadOnSquareLattice2D

- CubicLattice3D: CInitialHeadOnCubicLattice3D

- RandomSurface2D: CInitialHeadRandomSurface2D

- `RandomSurface3D: CInitialHeadRandomSurface3D`

- `RandomVolumetric3D: CInitialHeadRandomVolumetric3D`

2.8.4 udf classes for particles

See also sec. 2.4.

2.8.4-a `class CParticle`

- `ParticleNumber: int`
  Particle number.
- `SegmentNumber: int`
  Segment number.
- `SegmentSpecies: select { "SA", "SB" }`
  Segment species.
- `PolymerSpecies: select { "P0", "P1", "P2", "P3", "P4", "P5", "P6", "P7", "P8", "P9" }`
  Polymer species.
- `x: double`
  Particle coordinate, *x*.
- `y: double`
  Particle coordinate, *y*.
- `z: double`
  Particle coordinate, *z*.
- `ImageCellX: int`
  Index of the image cell, *x*.
- `ImageCellY: int`
  Index of the image cell, *y*.
- `ImageCellZ: int`
  Index of the image cell, *z*.
- `GraftSwitch: select { "ON", "OFF" }`
  Switch of the grafting.

Bibliography

[1] Daan Frenkel and Berend Smit. *Understanding molecular simulation: from algorithms to applications*. Academic Press, London, 2002.

[2] Y. Norizoe, H. Jinnai, and A. Takahara. Molecular simulation of 2-dimensional microphase separation of single-component homopolymers grafted onto a planar substrate. *EPL*, 101:16006, 2013.

[3] Makoto Matsumoto and Takuji Nishimura. Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Trans. Model. Comput. Simul.*, 8(1):3–30, 1998.

[4] Makoto Matsumoto and Yoshiharu Kurita. Twisted gfsr generators. *ACM Trans. Model. Comput. Simul.*, 2(3):179–194, 1992.

[5] Makoto Matsumoto and Yoshiharu Kurita. Twisted gfsr generators ii. *ACM Trans. Model. Comput. Simul.*, 4(3):254–266, 1994.

[6] J. C. Shillcock and R. Lipowsky. Equilibrium structure and lateral stress distribution of amphiphilic bilayers from dissipative particle dynamics simulations. *J. Chem. Phys.*, 117(10):5048–5061, 2002.

[7] Kostas Ch. Daoulas and Marcus Müller. Comparison of simulations of lipid membranes with membranes of block copolymers. In Wolfgang Peter Meier and Wolfgang Knoll, editors, *Polymer Membranes/Biomembranes*, volume 224 of *Advances in Polymer Science*, pages 43–85. Springer Berlin Heidelberg, 2010.

[8] Kostas Ch. Daoulas and Marcus Müller. Exploring thermodynamic stability of the stalk fusion-intermediate with three-dimensional self-consistent field theory calculations. *Soft Matter*, 9:4097–4102, 2013.

[9] Marcus Müller, Kostas Ch. Daoulas, and Yuki Norizoe. Computing free energies of interfaces in self-assembling systems. *Phys. Chem. Chem. Phys.*, 11(12):2087–2097, 2009.

Acknowledgments

This work is supported by New Energy and Industrial Technology Development Organization (NEDO), Japan.