Microstructure **In**telligent **Des**ign Software

快速指导手册

（材料）微结构智能设计软件

April, 2023

Science center for phase diagram, phase transition, material intelligent design and manufacture, Central South University, China

相图、相变及材料智能设计与制备科学中心，中南大学，中国

MInDes – a program for microstructure intelligent design software

Science center for phase diagram, phase transition, material intelligent design and manufacture, Central South University, China

目录

1. MInDes 介绍

MInDes 是由中南大学相图、相变及材料智能设计与制备实验室主任、教育部长江学者、973 项目首席科学家——杜勇教授提出以耦合实际热力学、扩散、热物性、力学等数据库为目标的，耦合多物理场演化的介观微结构模拟软件。

MInDes 采用 C++高级程序语言搭建了基本程序框架。程序底层设置了必要的数据（函数）类型、运算方法、数据网格结构、程序核心循环、各物理场求解器和并行框架。程序中层是对接各求解器的接口模块，由研究人员进行模型、程序功能模块、数据库的二次开发设计。程序表层将在可视化界面上处理 MInDes 的输入（.mindes）、输出（.log、.vts、.dat 和.txt 等）等。MInDes 可在 windows、linux 双系统上运行，将使用 OpenMP、CUDA 等并行库加速计算，使用差分法、傅里叶谱方法、格子玻尔兹曼法等对各物理场进行求解。

开发者：

MInDes 程序的开发自 2019 年始，是中南大学相图、相变及材料智能设计与制备实验室在读博士黄奇（2018-2023）博士期间的主要成果。多年后续开发期间，许多科研工作者进行了贡献：

黄奇、…

手册更新历史：

2023 年 4 月（第一版）、2023 年 11 月（第二版）、…

合作联系：

杜勇教授: yong-du@csu.edu.cn;

黄奇博士: qihuang0908@163.com;

2. 软件的配置安装、模拟及可视化

2.1. 运行环境

Windows:

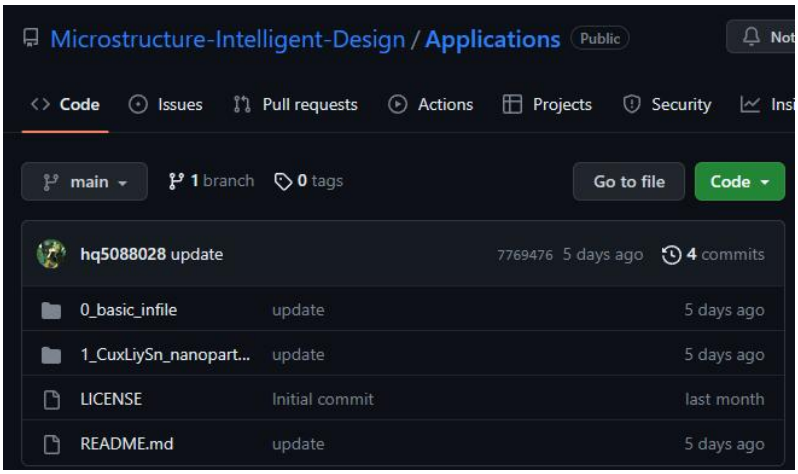
64 位（x64）系统。

Linux:

暂无要求。

2.2. 软件安装

从 Github 下载安装包（Applications-main.zip）：



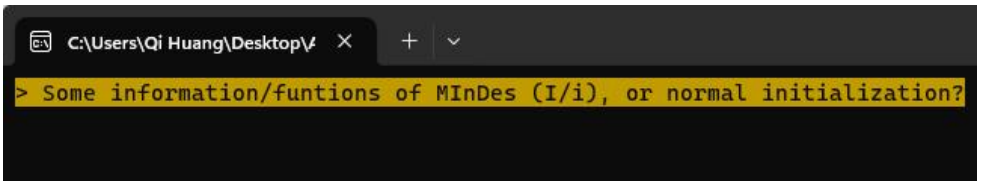
解压安装包（Applications-main.zip）

基本的输入文件都放在文件夹（0_basic_infile）中，其他应用案例放在后续文件夹中，以文件夹（1_CuxLiySn_nanoparticle_lithiation）为例，打开该文件夹

从上至下：该案例的输入文件、动态库（.dll）、MInDes（linux 系统执行文件）、exe 运行文件（x64 windows 系统执行文件）及其他文件。

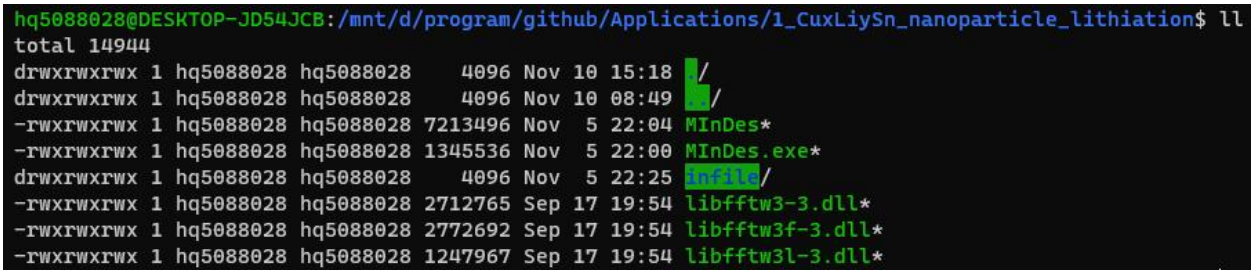
在对应系统中运行执行文件，进入软件配置界面

Windows 系统下，双击打开 MInDes.exe：

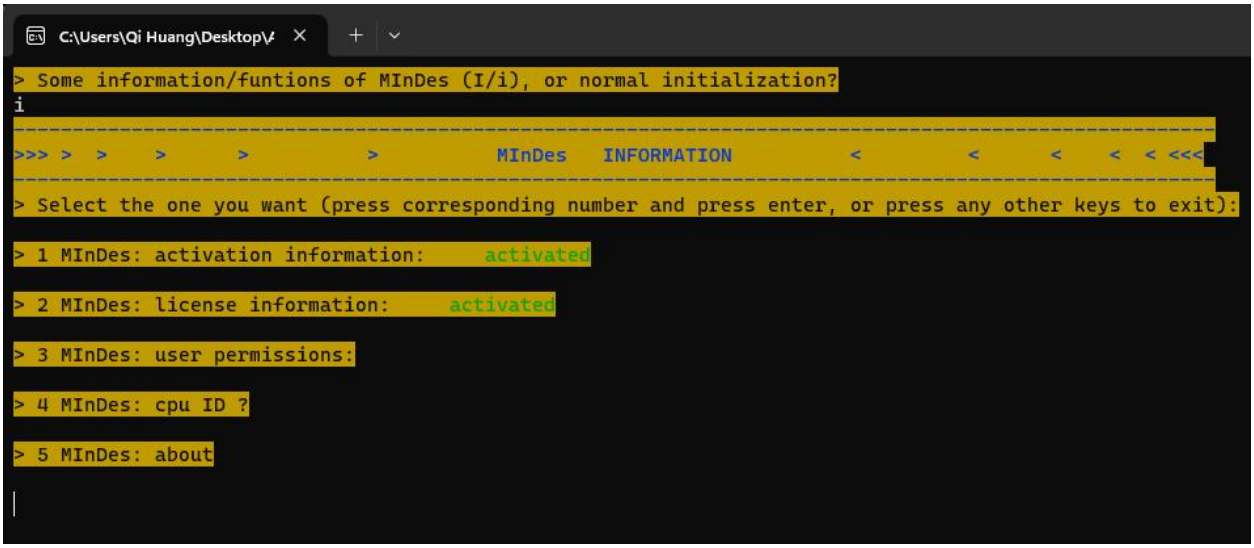


敲击键盘“i”并回车，进入信息界面。

Linux 系统下，在该文件夹内执行 MInDes，

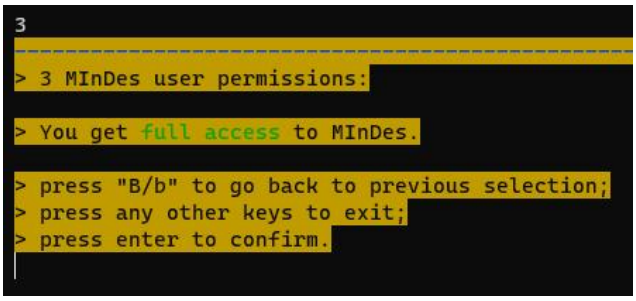


直接进入信息界面，总界面：



可在此界面查看（1）激活信息及（2）license 信息，目前权限全开放。

在此界面可查看当前软件权限（3）：



最后，可查看软件概况：

```
5
------------------------------------------------------------------------------
> 5 MInDes about

> Developer:Qi Huang

> Email:qihuang0908@163.com

> Copyright (c) 2019-2023 Science center for phase diagram, phase transition,
>               material intelligent design and manufacture. Central South University. China

> This program is free software: you can redistribute it and/or modify it under the terms
> of the GNU General Public License as published by the Free Software Foundation, either
> version 3 of the License, or (at your option) any later version.

> This program 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 General Public License for more details.

> You should have received a copy of the GNU General Public License along with this program.
> If not, see <http://www.gnu.org/licenses/>.

                                                  Pikachu says: let's go ! it's time to start a simulation !

> press "B/b" to go back to previous selection;
> press any other keys to exit;
> press enter to confirm.
```

2.3. 模拟流程及可视化

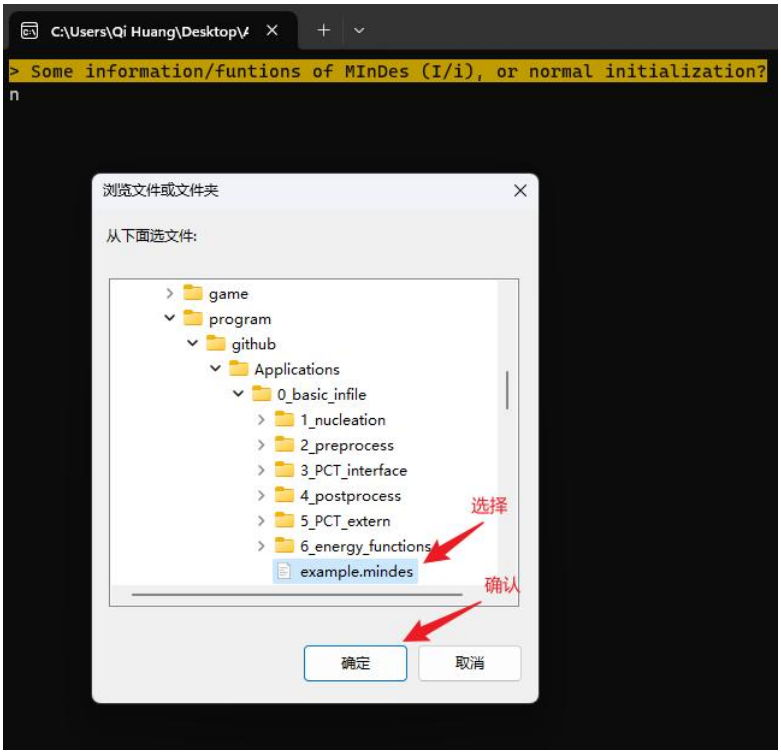
通过 MInDes 执行文件读取.mindes 输入文件

Windows 读取方法：

双击打开 MInDes：

```
C:\Users\Qi Huang\Desktop\M
> Some information/funtions of MInDes (I/i), or normal initialization?
```

输入 N/n 选择 normal initialization，enter 确认，弹出 windows 文件浏览系统，可在该界面找到对应的输入文件的.mindes 文件，比如 0_basic_infile 文件夹中的 example.mindes 文件（其他的案例可以读取对应文件夹中的.mindes 文件）：



软件将自动读取该**输入文件**，并在同目录下生成同名文件夹用于保存**输出文件**：

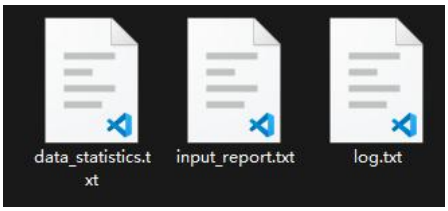
因此也需要注意，同目录下、同名文件夹内的同名输出文件文件会在输出过程中被覆盖，可通过修改该文件夹名称保留输出文件。

Linux 读取方法：

定位到 MInDes 所在文件夹（必须在该文件夹内启动 MInDes 软件），在启动软件时，在后面输入读取.mindes 文件的索引，同样可读取 0_basic_infile 文件夹中的 example.mindes 文件：

```
16:19 ./
15:37 ../
22:31 MInDes*
22:31 MInDes.exe*
15:37 infile/
22:31 libfftw3-3.dll*
22:31 libfftw3f-3.dll*
22:31 libfftw3l-3.dll*
/Desktop/Applications-main/1_CuxLiySn_nanoparticle_lithiation$ ./MInDes ../0_basic_infile/example
```

输出文件与 Windows 系统一致，可以看到 example 输出文件夹中默认存在三个文本文件：



统计模拟过程数据的 data_statistics.txt 文件、对输入文件内所有 keys 的反馈文件 input_report.txt 和日志文件 log.txt 文件。

设置.mindes 文件

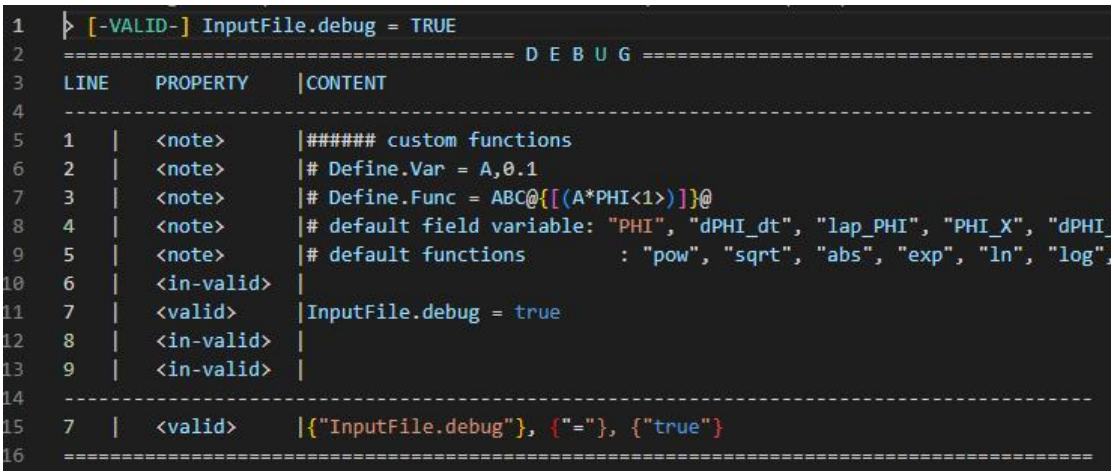
打开 example.mindes 文件，观察到除了上方被注释的 5 行，有一行 key 值：InputFile.debug = true

为与注释行做区分，我们称之为命令行。命令行开头、末尾都不能空格，中间两个空格将命令行分为 3 部分，第一部分为命令行的钥匙（key）：“InputFile.debug”，第二部分统一都为等号：“=”，第三部分为命令行的值

（value）：“true”，为确保命令行不会失效，修改输入文件时不应再引入更多空格。

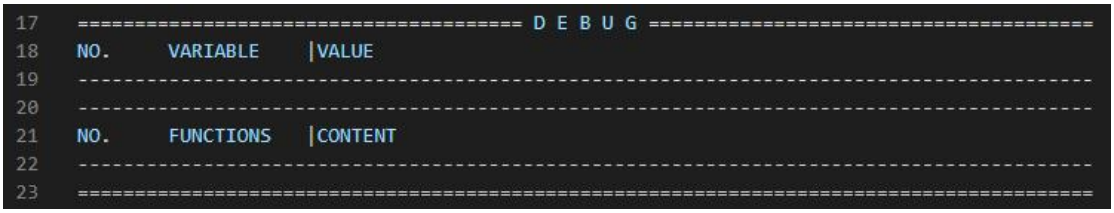
该命令行的功能是对该输入文件进行测试，并在输出在文件 input_report.txt 中，然后使用者就可以通过 input_report.txt 文件来测试、检查所有命令行的的书写情况。

如，输出案例输入文件的 input_report.txt 是一个多行的文本文件，其中第一部分如下图，是对所有输入文本有效性的识别



如图，第 5-13 行是读取自 example.mindes 输入文件中所有文本。这些文本会被分为 3 类：注释行<note>、有效行<valid>、无效行<in-valid>，注释及无效行会在进一步识别过程中被忽略，而有效行会被软件管理（如第 15 行）。

同样我们可以在输入文件中定义变量和函数，以辅助进行一些复杂的输入，这些定义在如下图的容器中被测试（本输入文本未定义变量和函数）：



然后，后面所有的是对输入命令行的识别，我们可以看到有关于命令行的解释，和对应命令的读取情况：

```
######################## define custom variables and functions ########################
# Define.Var = A,0.1
# Define.Func = ABC@{[(A*PHI<1>)]}@
# default functions      : "pow", "sqrt", "abs", "exp", "ln", "log", "sin", "cos", "tan",
#######################################################################################
> [DEFAULT] Solver.Loop.begin_step = 0
> [DEFAULT] Solver.Loop.end_step = 0
> [DEFAULT] Solver.Loop.screen_loop_step = -1
> [DEFAULT] Solver.Loop.screen_output_step = -1
> [DEFAULT] Solver.Loop.vts_output_step = -1
> [DEFAULT] Solver.Loop.data_output_step = -1
# Solver.difference_method : 0 - FIVE_POINT , 1 - NINE_POINT
> [DEFAULT] Solver.difference_method = 0
> [DEFAULT] Solver.Phi.is_normalize = FALSE
> [DEFAULT] Solver.Con.is_normalize = TRUE
> [DEFAULT] Solver.PairWise.accelerate = FALSE
> [DEFAULT] Solver.Parallel.openmp_thread_counts = 1
> [DEFAULT] Solver.Loop.Iterate.Con = 1
> [DEFAULT] Solver.Loop.Iterate.Temp = 1
# Solver.Comps = (c0, c1, c2, ... )
> [DEFAULT] Solver.Comps = ()
# Solver.Phases = {[(phase0),(c0, c1, ... )], [(phase1),(c0, c1, ... )], ... }
> [DEFAULT] Solver.Phases = {}
```

命令前方的[DEFAULT]代表没有读取到这个命令，因此采用默认值。而读取到的命令，前方为[-VALID-]：

```
> [-VALID-] InputFile.debug = TRUE
```

部分命令行的读取意味着开启某个模块，从而会解锁更多的可识别的命令行。因此使用者需要通过 input_report.txt 及已有的案例来反复测试和修改输入文件，来达到想要的模拟效果。

解读日志信息（log.txt）

日志信息会直接被输出在控制台上（不论是 Windows 还是 Linux），同时也会以文件的形式（log.txt）存储在输出文件夹中。例如输出文件夹 example 中的 log.txt：

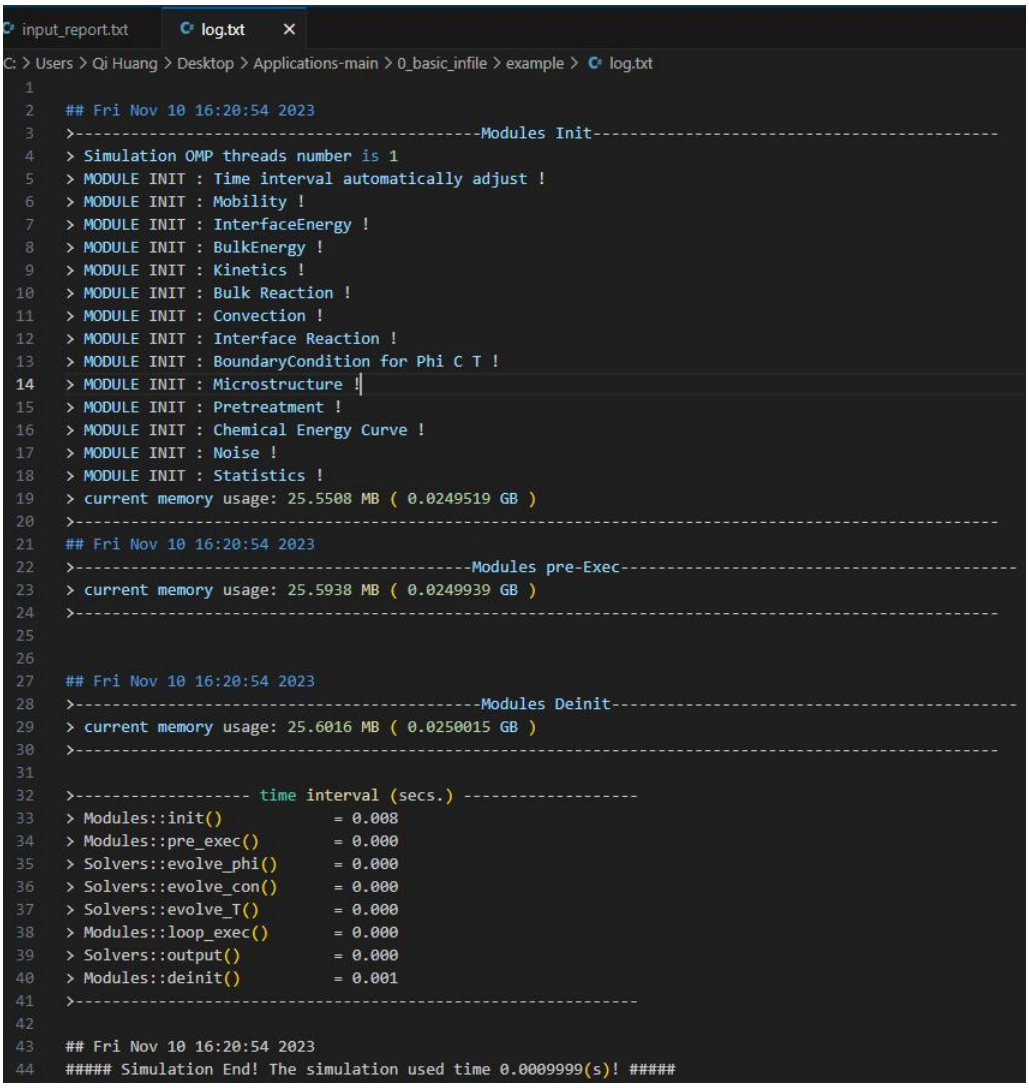
其中，Modules Init 为模块初始化区域，这里将提示 OpenMP 使用的线程数，初始化过程中尝试启动的模块及内存的消耗。

然后是 Modules pre-Exec 区域，这里将执行预处理功能，执行完毕后同样会统计内存的消耗。

接着在 Modules pre-Exec 和 Modules Deinit 之间是程序的循环主体，用于输出模拟过程中的相、组分、温度的演化信息。

最后，执行 Modules Deinit 来释放模拟网格空间。

在程序结束前，会统计模拟各阶段的耗时，以辅助开发者进行程序高效运行设计。



可视化（.vts）

需要安装 paraview 软件辅助 MInDes 进行可视化。

在循环过程中，可以通过一些命令来控制输出：

```
> [DEFAULT] Solver.Loop.screen_loop_step = -1
> [DEFAULT] Solver.Loop.screen_output_step = -1
> [DEFAULT] Solver.Loop.vts_output_step = -1
> [DEFAULT] Solver.Loop.data_output_step = -1
```

其中 screen_loop_step 控制一些无运算消耗的输出，用于提示循环执行到第几步，以及 phi、c、T 的收敛情况；screen_output_step 控制一些相平均分数、

浓度平均分数的统计情况，需要消耗少量运算进行统计；vts_output_step 控制 vts 展示文件的输出；data_output_step 控制网格数据备份输出。在这四个命令默认的情况下，都不会进行输出。

对于展示文件 vts，在主求解器和各个模块中，有相应的命令来控制一些数据的输出：

```
> [DEFAULT] Postprocess.PCT.VTS.phi = FALSE
> [DEFAULT] Postprocess.PCT.VTS.con = FALSE
> [DEFAULT] Postprocess.PCT.VTS.potential = FALSE
> [DEFAULT] Postprocess.PCT.VTS.energy_density = FALSE
> [DEFAULT] Postprocess.PCT.VTS.temperature = FALSE
> [DEFAULT] Postprocess.PCT.VTS.phase_con = FALSE
> [DEFAULT] Postprocess.PCT.VTS.phase_potential = FALSE
> [DEFAULT] Postprocess.PCT.VTS.grains_rev = FALSE
> [DEFAULT] Postprocess.PCT.VTS.phi_index = FALSE
> [DEFAULT] Postprocess.PCT.VTS.phi_gradient = FALSE
> [DEFAULT] Postprocess.PCT.VTS.phi_name = FALSE
> [DEFAULT] Postprocess.PCT.VTS.phi_summary = FALSE
> [DEFAULT] Postprocess.PCT.VTS.interface_flag = FALSE
```

在输出的 vts 中，scalar 代表输出数据为标量，vec3 代表输出数据为向量

打开 vts 文件后，会自动来到 paraview 界面，或者可以在 paraview 软件中批量打开 vts 文件，以形成动图。paraview 的操作需自行学习。

再（套嵌）模拟（.dat）

输出的.dat 文件包含模拟网格中相分数、浓度、温度及自定义变量信息，使用者可以通过命令来控制是否通过.dat 文件来初始化一个模拟网格

```
> [DEFAULT] Preprocess.Microstructure.is_datafile_init = FALSE
```

在初始化过程中，需要注意模拟网格、相和组分等信息与.dat 文件存储的是否匹配，否则将导致难以预料的后果。

在 xxxx\2_preprocess 文件夹中的 3_merge_phis_auto.mindes 案例就用到了该方法，可用作参考。在 Windows 下也可通过文件浏览系统索引需要的.dat 文件。

2.4. 基本输入参数简介

```
3.    > [DEFAULT] Solver.Loop.begin_step = 0
```
模拟开始步

```
4.    > [DEFAULT] Solver.Loop.end_step = 0
```
模拟结束步

```
5.    > [DEFAULT] Solver.Loop.screen_loop_step = -1
```
模拟步 及 phi、c、T 收敛情况提示

```
6.    > [DEFAULT] Solver.Loop.screen_output_step = -1
```
综合屏幕输出步

```
7.    > [DEFAULT] Solver.Loop.vts_output_step = -1
```
vts 文件输出步

```
8.    > [DEFAULT] Solver.Loop.data_output_step = -1
```
网格数据文件输出步

```
9.    # Solver.difference_method : 0 - FIVE_POINT , 1 - NINE_POINT
10.   > [DEFAULT] Solver.difference_method = 0
```
差分方法

```
11.   > [DEFAULT] Solver.Phi.is_normalize = FALSE
12.   > [DEFAULT] Solver.Con.is_normalize = TRUE
```
相及组分是否归一

```
13.   > [DEFAULT] Solver.PairWise.accelerate = FALSE
```
反对称相场模型加速

```
14.   > [DEFAULT] Solver.Parallel.openmp_thread_counts = 1
```
并行线程数

```
15.   > [DEFAULT] Solver.Loop.Iterate.Con = 1
16.   > [DEFAULT] Solver.Loop.Iterate.Temp = 1
```
多步迭代，一个相场步对应的多个浓度场步及温度场步（暂未调试）

```
17.   # Solver.Comps = (c0, c1, c2, ... )
18.   > [DEFAULT] Solver.Comps = ()
```

体系内的组分定义

```
19.    # Solver.Phases = {[(phase0),(c0, c1, ... )], [(phase1),(c0,
       c1, ... )], ... }
20.    > [DEFAULT] Solver.Phases = {}
```

体系内的相定义

```
21.    # Solver.GrainsOrientations = {[(phi_index_0,
       phi_index_2, ... ),(rotation_angle_1, rotation_angle_2,
       rotation_angle_3)],  ... }
22.    > [DEFAULT] Solver.GrainsOrientations = {}
```

定义晶粒的取向（未调试）

```
23.    # Solver.GrainsOrientations.rotation_gauge = 0 - XYX, 1 - XZX, 2 - YXY, 3
       - YZY, 4  - ZXZ, 5  - ZYZ
24.    #                                        6 - XYZ, 7 - XZY, 8 - YXZ, 9
       - YZX, 10 - ZXY, 11 - ZYX
25.    > [DEFAULT] Solver.GrainsOrientations.rotation_gauge = 4
```

晶粒取向的旋转准则

```
26.    > [DEFAULT] Solver.PCT.RealTime.init = 0.000000
```

起始的真实时间

```
27.    > [DEFAULT] Solver.PCT.TimeInterval.dt = 1.000000
```

模拟的时间步长

```
28.    # Solver.PCT.TimeInterval.auto_adjust =
       (delt_step,max_scale,is_reduce_output)
29.    > [DEFAULT] Solver.PCT.TimeInterval.auto_adjust = (100,1e3,true)
```

自动调整时间步长，以使 phi、c、T 演化稳定

```
30.    > [DEFAULT] Solver.PCT.phi_increment_limit = 0.001000
31.    > [DEFAULT] Solver.PCT.con_increment_limit = 0.001000
32.    > [DEFAULT] Solver.PCT.temp_increment_limit = 0.001000
```

自动调整时间步长过程中，期望 phi、c、T 演化的精度

```
33.    > [DEFAULT] Solver.Mesh.Nx = 1
34.    > [DEFAULT] Solver.Mesh.Ny = 1
35.    > [DEFAULT] Solver.Mesh.Nz = 1
```

模拟空间的大小，为使并行有效需以定义 x 方向网格优先

```
36.    > [DEFAULT] Solver.Mesh.dr = 1.000000
```

模拟网格的大小

```
37.    # Solver.Mesh.BoundaryCondition : 0 - FIXED , 1 - PERIODIC , 2 - ADIABATIC
```

```
38.    > [DEFAULT] Solver.Mesh.BoundaryCondition.x_up = 1
39.    > [DEFAULT] Solver.Mesh.BoundaryCondition.x_down = 1
40.    > [DEFAULT] Solver.Mesh.BoundaryCondition.y_up = 1
41.    > [DEFAULT] Solver.Mesh.BoundaryCondition.y_down = 1
42.    > [DEFAULT] Solver.Mesh.BoundaryCondition.z_up = 1
43.    > [DEFAULT] Solver.Mesh.BoundaryCondition.z_dowm = 1
```
边界条件

```
44.    # ModelsManager.Phi.equation : 0 - Const, 1 - AllenCahn Standard, 2 -
       AllenCahn Pairwise, 3 - CahnHilliard Standard
45.    > [DEFAULT] ModelsManager.Phi.equation = 0
```
相场方程

```
46.    # ModelsManager.Con.equation : 0 - Const, 1 - TotalConcentration, 2 -
       PhaseConcentration, 3 - GrandPotential
47.    > [DEFAULT] ModelsManager.Con.equation = 0
```
浓度场方程

```
48.    # ModelsManager.Con.valid_domain : 0 - Standard, 1 - Reverse
49.    > [DEFAULT] ModelsManager.Con.valid_domain = 0
```
浓度场有效区域

```
50.    # ModelsManager.Temp.equation : 0 - Const, 1 - Standard
51.    > [DEFAULT] ModelsManager.Temp.equation = 0
```
温度场方程

```
52.    > [DEFAULT] Preprocess.Microstructure.is_datafile_init = FALSE
```
是否以数据网格文件初始化一个模拟

```
53.    # .matrix = {[(phi_index),(phi_name),(phi_comp_0_value,
       phi_comp_1_value, ... )],[(total_comp_0_value,
       total_comp_1_value, ... )],[(temp_value)]}
54.    > [DEFAULT] Preprocess.Microstructure.matrix = {[()]}
```
模拟基体的定义

```
55.    # .property = [(phi_index_begin, phi_index_end),
       (phi_name, ... ),(phi_weight, ...)]
56.    > [DEFAULT] Preprocess.Microstructure.voronoi.property = [()]
```
生成多晶结构

```
57.    # .property = (bmp_layer, file_name)
58.    > [DEFAULT] Preprocess.Microstructure.bmp24.property = ()
```
从图片读取二维结构

```
59.    > [DEFAULT] Preprocess.Microstructure.geometry_layer_number = 0
```

在几何区域内拓展网格结构和生成网格数据

```
60.    # .property = [(phi_index_begin, phi_index_end),
       (phi_name, ... ),(phi_weight, ...),(in phi_index, ... )]
61.    > [DEFAULT] Preprocess.Microstructure.voronoi_inPhis.property = [()]
```

定义位于某个/些相区内的多晶结构

```
62.    # Preprocess.reconstruct_phis = {[(phi_index_0, phi_index_1, ... ),
       (phi_name)], .... }
63.    > [DEFAULT] Preprocess.reconstruct_phis = {[()]}
```

重构某些晶粒的性质

```
64.    # Preprocess.merge_phis = {[(phi0,phi1,phi2, ... ), is_phi_c_merge], .... }
65.    > [DEFAULT] Preprocess.merge_phis = {[()]}
```

混合某些相形成一个相

```
66.    # Preprocess.auto_merge_phis = (phis_index_0, phis_index_1, ... )
67.    > [DEFAULT] Preprocess.auto_merge_phis = ()
```

自动混合互相不接触的多个相

```
68.    # Preprocess.relax_interface = (relax_steps, output_steps,
       fix_phi_after_relax)
69.    > [DEFAULT] Preprocess.relax_interface = ()
```

弛豫相界面

```
70.    > [DEFAULT] Preprocess.remove_inexistent_phis = FALSE
```

剔除不存在的相，减少内存空间消耗

```
71.    > [DEFAULT] Preprocess.re_ordering_phis_indexs_from = 0
```

对各个相的索引再排序

```
72.    # Preprocess.fill_phis = {[(phi_index_0, phi_index_1, ... ), (phi_con_1,
       phi_con_2, ... ), (total_con_1, total_con_2, ... ), (temperature)], .... }
73.    > [DEFAULT] Preprocess.fill_phis = {[()]}
```

填充相区内的组分、温度数据

```
74.    # Postprocess.physical_fields = (mechanic, fluid dynamic, electric)
75.    > [DEFAULT] Postprocess.physical_fields = (false,false,false)
76.    > [-VALID-] Postprocess.physical_fields(0) = FALSE
77.    > [-VALID-] Postprocess.physical_fields(1) = FALSE
78.    > [-VALID-] Postprocess.physical_fields(2) = FALSE
```

开启外场：机械场、流场、电场

```
79.    > [DEFAULT] Postprocess.Statistics.file_name = data_statistics
80.    > [DEFAULT] Postprocess.Statistics.is_phi_c_t = FALSE
```

```
81.    > [DEFAULT] Postprocess.Statistics.is_electricity = FALSE
82.    # Postprocess.Statistics.datafiles = (datafile1, ... )
83.    > [DEFAULT] Postprocess.Statistics.datafiles = ()
```

统计网格内的数据并进行输出

```
84.    > [DEFAULT] Postprocess.PCT.VTS.phi = FALSE
85.    > [DEFAULT] Postprocess.PCT.VTS.con = FALSE
86.    > [DEFAULT] Postprocess.PCT.VTS.potential = FALSE
87.    > [DEFAULT] Postprocess.PCT.VTS.energy_density = FALSE
88.    > [DEFAULT] Postprocess.PCT.VTS.temperature = FALSE
89.    > [DEFAULT] Postprocess.PCT.VTS.phase_con = FALSE
90.    > [DEFAULT] Postprocess.PCT.VTS.phase_potential = FALSE
91.    > [DEFAULT] Postprocess.PCT.VTS.grains_rev = FALSE
92.    > [DEFAULT] Postprocess.PCT.VTS.phi_index = FALSE
93.    > [DEFAULT] Postprocess.PCT.VTS.phi_gradient = FALSE
94.    > [DEFAULT] Postprocess.PCT.VTS.phi_name = FALSE
95.    > [DEFAULT] Postprocess.PCT.VTS.phi_summary = FALSE
96.    > [DEFAULT] Postprocess.PCT.VTS.interface_flag = FALSE
```

一些可以在 vts 文件中输出的数据，是否输出的开关

```
97.    > [DEFAULT] <AUTO> Solver.Loop.stop = FALSE
```

正常停止一个运行程序的开关，程序会跳出未完成的主循环并进行结束输出和内存释放操作。

3. 模块介绍

3.1. 微结构初始化

在第二节的介绍中，结构初始化由以下命令行控制：

```
# .matrix = {[(phi_index),(phi_name),(phi_comp_0_value,
phi_comp_1_value, ... )],[(total_comp_0_value,
total_comp_1_value, ... )],[(temp_value)]}
> [DEFAULT] Preprocess.Microstructure.matrix = {[()]}
# .property = [(phi_index_begin, phi_index_end),
(phi_name, ... ),(phi_weight, ...)]
> [DEFAULT] Preprocess.Microstructure.voronoi.property = [()]
# .property = (bmp_layer, file_name)
> [DEFAULT] Preprocess.Microstructure.bmp24.property = ()
> [DEFAULT] Preprocess.Microstructure.geometry_layer_number = 0
# .property = [(phi_index_begin, phi_index_end),
(phi_name, ... ),(phi_weight, ...),(in phi_index, ... )]
> [DEFAULT] Preprocess.Microstructure.voronoi_inPhis.property = [()]
```

首先使用者需要声明微结构的基体，所有的结构都会被初始化在这个基体上：

```
Preprocess.Microstructure.matrix
```

接着可以在基体上生成几何体，每个几何体位于一个几何层中，几何层代表了生成几何体的先后顺序，后生成的几何体会覆盖先生成的几何体，因此可以通过设计几何层形成层次感：

```
Preprocess.Microstructure.geometry_layer_number = 1
```

然后，使用者需要声明几何层的一些属性。如仅有一层几何层，只需要声明第一层的属性（按 0 开始排序）

```
# .property = (phi_index, phi_name, geometry_type, rotation_gauge, reverse_region)
#              geometry_type  : 0 - None, 1 - Ellipsoid, 2 - Polyhedron
#              rotation_gauge : 0 - XYX, 1 - XZX, 2 - YXY, 3 - YZY, 4  - ZXZ,
5  - ZYZ
#                               6 - XYZ, 7 - XZY, 8 - YXZ, 9 - YZX, 10 - ZXY, 11
- ZYX
> [-VALID-] Preprocess.Microstructure.geometry_layer_0.property = (0,G0,0,1,false)
```

相的索引（phi_index）用于区分每个晶粒，对应Φ^1、Φ^2、Φ^3等，而相的属性 phi_name 可以认为是具有相同物理性质的晶粒。如具有相同的热力学吉布斯自由能函数，具有相同弹性模量等，用于进一步分类。几何类型（geometry_type）目前支持椭球、多面体的生成。生成步与无量纲模拟步对

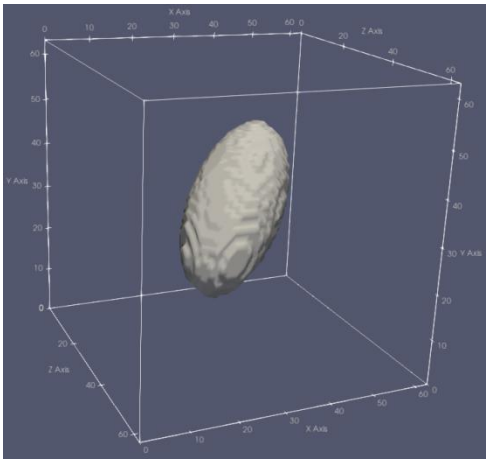
应，一般情况下设置为第 0 步，即在模块预处理过程中就会生成结构。旋转规则（rotation_gauge）可用于二维、三维结构的旋转，一般取几何体的相对坐标轴作为旋转参照物。反向填充（reverse_region）即为填充几何体外的部分。

➢ 初始化一个椭球/椭圆

如果在设置几何层时，定义几何类型为椭球，使用者就需要在该几何层上进一步定义椭球的参数，及几何区域内的数据，如下

```
# .ellipsoid =
[(core_x,core_y,core_z),(radius_x,radius_y,radius_z),(rotation_angle_1,rotation_a
ngle_2,rotation_angle_3)]
> [DEFAULT] Preprocess.Microstructure.geometry_layer_0.ellipsoid =
[(0,0,0),(0,0,0),(0,0,0)]
> [DEFAULT] Preprocess.Microstructure.geometry_layer_0.T = 0.000000
> [DEFAULT] Preprocess.Microstructure.geometry_layer_0.phi = 1.000000
> [DEFAULT] Preprocess.Microstructure.geometry_layer_0.is_normalized = TRUE
# .x = [(comp_0_name,comp_0_value),(comp_1_name,comp_1_value), ...]
> [DEFAULT] Preprocess.Microstructure.geometry_layer_0.x = [()]
# .custom_int = [(custom_0_index, custom_0_value),(custom_1_index,
custom_1_value), ...]
> [DEFAULT] Preprocess.Microstructure.geometry_layer_0.custom_int = [()]
# .custom_double = [(custom_0_index, custom_0_value),(custom_1_index,
custom_1_value), ...]
> [DEFAULT] Preprocess.Microstructure.geometry_layer_0.custom_double = [()]
# .custom_vec3 = [(custom_0_index, custom_0_value_0, custom_0_value_1,
custom_0_value_2), (custom_1_index, custom_1_value_0, custom_1_value_1,
custom_1_value_2), ...]
> [DEFAULT] Preprocess.Microstructure.geometry_layer_0.custom_vec3 = [()]
# .custom_vec6 = [(custom_vec6_index, custom_vec6_value_0, custom_vec6_value_1,
custom_vec6_value_2, custom_vec6_value_3, custom_vec6_value_4,
custom_vec6_value_5), ...]
> [DEFAULT] Preprocess.Microstructure.geometry_layer_0.custom_vec6 = [()]
```

案例详见 1_nucleation/0_ellipse.mindes 及 1_nucleation/1_ellipsoid.mindes

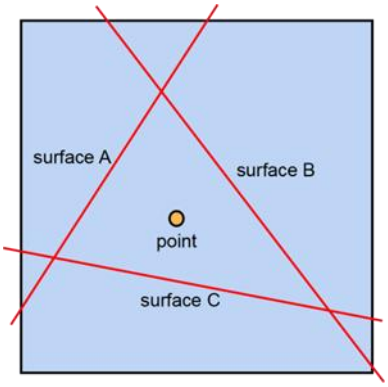


➢ 初始化一个多边形/多面体

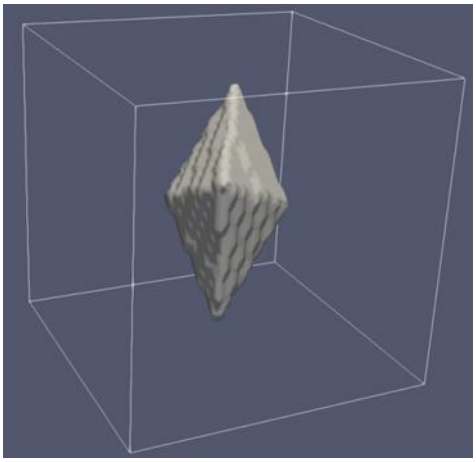
如果在设置几何层时，定义几何类型为多面体，使用者就需要在该几何层上进一步定义多面体的参数

```
# .polyhedron =
{[inside_point],[surf_point,surf_point,surf_point], .... ,[(rotation_angle_1,rota
tion_angle_2,rotation_angle_3)]}
#                point = (position_x,position_y,position_z)
> [DEFAULT] Preprocess.Microstructure.geometry_layer_0.polyhedron = {[(-
1,0,0)],[(0,0,0),(0,1,0),(0,0,1)],[(0,0,0)]}
> [DEFAULT] Preprocess.Microstructure.geometry_layer_0.T = 0.000000
> [DEFAULT] Preprocess.Microstructure.geometry_layer_0.phi = 1.000000
> [DEFAULT] Preprocess.Microstructure.geometry_layer_0.is_normalized = TRUE
# .x = [(comp_0_name,comp_0_value),(comp_1_name,comp_1_value), ...]
> [DEFAULT] Preprocess.Microstructure.geometry_layer_0.x = [()]
# .custom_int = [(custom_0_index, custom_0_value),(custom_1_index,
custom_1_value), ...]
> [DEFAULT] Preprocess.Microstructure.geometry_layer_0.custom_int = [()]
# .custom_double = [(custom_0_index, custom_0_value),(custom_1_index,
custom_1_value), ...]
> [DEFAULT] Preprocess.Microstructure.geometry_layer_0.custom_double = [()]
# .custom_vec3 = [(custom_0_index, custom_0_value_0, custom_0_value_1,
custom_0_value_2), (custom_1_index, custom_1_value_0, custom_1_value_1,
custom_1_value_2), ...]
> [DEFAULT] Preprocess.Microstructure.geometry_layer_0.custom_vec3 = [()]
# .custom_vec6 = [(custom_vec6_index, custom_vec6_value_0, custom_vec6_value_1,
custom_vec6_value_2, custom_vec6_value_3, custom_vec6_value_4,
custom_vec6_value_5), ...]
> [DEFAULT] Preprocess.Microstructure.geometry_layer_0.custom_vec6 = [()]
```

采用面切割+体内点锚定的方法确定一个多面体/多边形，该方法也可轻易拓展到 Voronoi 结构生成逻辑中。

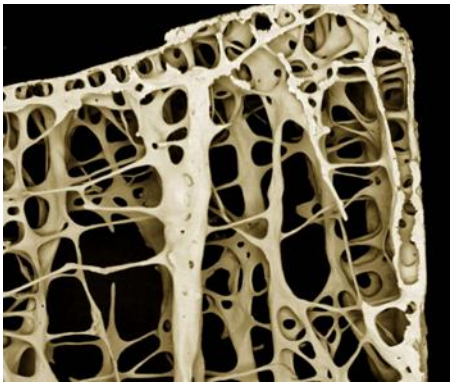


案例见 1_nucleation/2_polygon.mindes 及 1_nucleation/3_polyhedron.mindes



除了几何体形核之外，还可以根据图片（bmp24 格式）来初始化一个更加复杂/自然的结构：

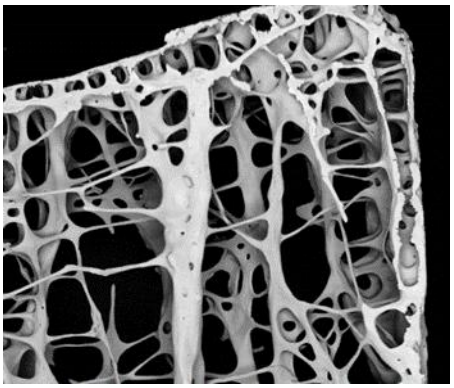
比如面对下图的复杂结构，



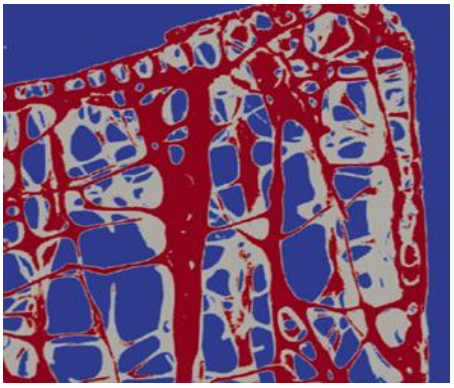
调用 bmp24 方法，

```
# .property = (bmp_layer, file_name)
> [DEFAULT] Preprocess.Microstructure.bmp24.property = ()
# .phi = (phi_index, phi_name, phi_fraction)
> [DEFAULT] Preprocess.Microstructure.bmp24_0.phi = ()
# .x = [(comp_0_name,comp_0_value), ...]
> [DEFAULT] Preprocess.Microstructure.bmp24_0.x = [()]
> [DEFAULT] Preprocess.Microstructure.bmp24_0.gray_threshold = (0.0,1.0)
> [DEFAULT] Preprocess.Microstructure.bmp24_0.temperature = 0.000000
> [DEFAULT] Preprocess.Microstructure.bmp24_0.is_normalized = TRUE
```

该方法会读取 bmp24 格式的图片，并将其转化为灰度图片，如图



bmp24 方法自带图层（bmp_layer），可以分步从图片冲读取不同灰度的区域，然后对于该区域可以设置想生成相的索引（phi_index），相的性质（phi_name），及相的值（phi_fraction）。对于该区域也可以生成对应组分值（x）。可以设置该区域的灰度值范围（gray_threshold，0~1：黑~白）。对于该区域也可以生成对应组分值（temperature）。也可以设置相分数是否有归一的需求。



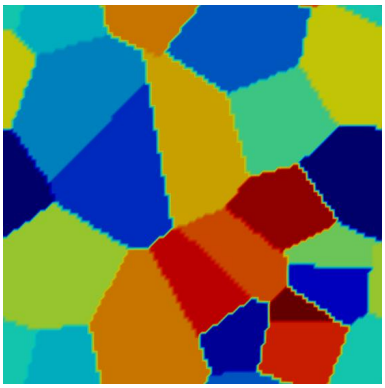
具体案例可见 1_nucleation/ 4_read_bmp24.mindes，通过假设两个相，在背景相（蓝色）中生成第一个相（红色）及第二个相（白色）。

Voronoi 结构是相场模拟中较为常见的结构，可通过如下命令快速生成基本的 voronoi 结构：

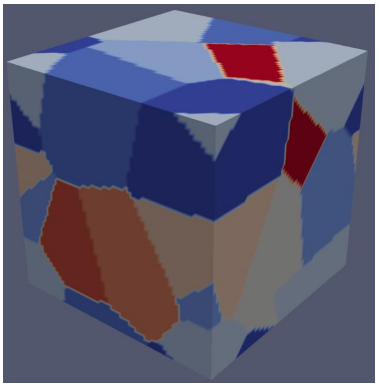
```
# .property = [(phi_index_begin, phi_index_end),
(phi_name, ... ),(phi_weight, ...)]
> [DEFAULT] Preprocess.Microstructure.voronoi.property = [()]
# .box = [(box_origin_point),(box_end_point)]
> [DEFAULT] Preprocess.Microstructure.voronoi.box = [(0,0,0),(0,0,0)]
# .x = [(comp_0_name,comp_0_value), (comp_1_name,comp_1_value), ...]
> [DEFAULT] Preprocess.Microstructure.voronoi.x = [()]
> [DEFAULT] Preprocess.Microstructure.voronoi.temperature = 0.000000
> [DEFAULT] Preprocess.Microstructure.voronoi.rand_seed = 0
> [DEFAULT] Preprocess.Microstructure.voronoi.point_distance = -1.000000
```

首先定义想生成的晶粒索引的范围，随机生成的晶粒的性质和权重。其中盒子为生成 voronoi 结构的区域（box）。在定义的区域内会随机撒点生成晶粒，晶粒数量由相索引起始和终止决定。各个晶粒具体分别属于什么相，由相名称及权重来随机分配。可以设置区域内的组分（x）和温度（temperature）。同时可以设置撒点时点的间距（point_distance）。

具体二维案例见 1_nucleation/ 5_voronoi2d.mindes



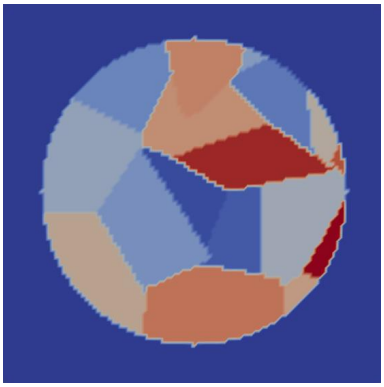
三维案例见 1_nucleation/ 8_voronoi3d.mindes



形核功能的叠加：例生成椭圆中的 Voronoi

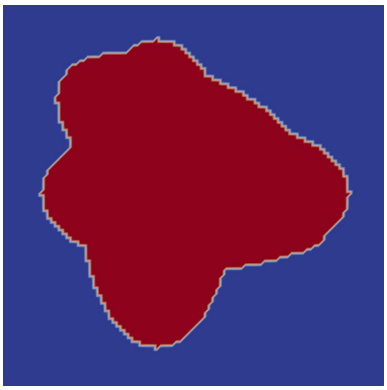
先生成 voronoi 结构，以及使用 ellips 反填充。

见 1_nucleation/ 6_voronoi2d_ellips.mindes



形核功能的叠加 2：例生成已有区域中的 Voronoi

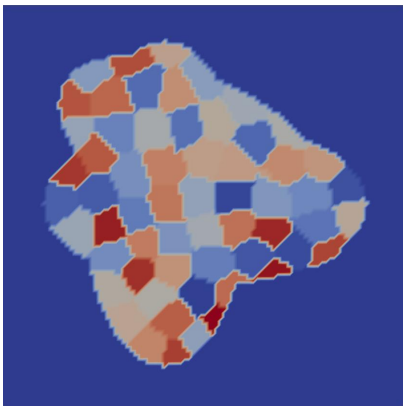
假设的任意几何区域：



在该区域内生成 voronoi 结构：

```
# .property = [(phi_index_begin, phi_index_end),
(phi_name, ... ),(phi_weight, ...),(in phi_index, ... )]
> [-VALID-] Preprocess.Microstructure.voronoi_inPhis.property =
[(1,60),(Grain0),(1.0),(0)]
# .box = [(box_origin_point),(box_end_point)]
> [-VALID-] Preprocess.Microstructure.voronoi_inPhis.box = [(0,0,0),(100,100,0)]
# .x = [(comp_0_name,comp_0_value), (comp_1_name,comp_1_value), ...]
> [DEFAULT] Preprocess.Microstructure.voronoi_inPhis.x = [()]
> [DEFAULT] Preprocess.Microstructure.voronoi_inPhis.temperature = 0.000000
> [-VALID-] Preprocess.Microstructure.voronoi_inPhis.rand_seed = 0
> [-VALID-] Preprocess.Microstructure.voronoi.point_distance = 7
```

与一般情况生成多晶结构类似，不过需要在 property 的设置末尾注明会在哪些相区内生成多晶结构。



3.2. 预处理

在预处理模块中集成了一些辅助处理功能。

相重构

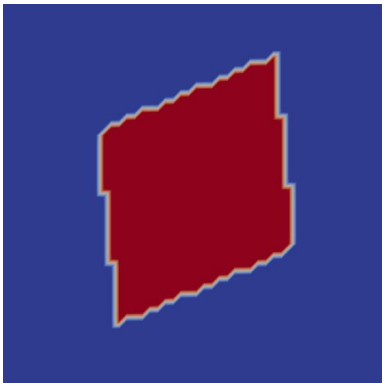
```
# Preprocess.reconstruct_phis = {[(phi_index_0, phi_index_1, ... ),
(phi_name)], .... }
> [DEFAULT] Preprocess.reconstruct_phis = {[()]}
```

重构调整读取已有网格结构（.dat）后，相的性质。

弛豫界面

案例见 2_pretreatment/1_relax_interface.mindes

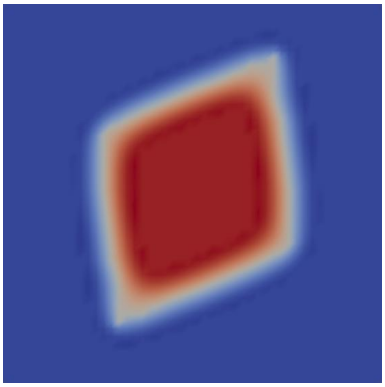
相场模型计算得到的是扩散界面，在相场法中，部分数值问题源自于还没有形成一个扩散的界面，该功能可在模拟前迭代弛豫出一个扩散的界面，以满足部分需求，比如对于一个具有尖锐界面的多边形结构



通过命令：

```
# Preprocess.relax_interface = (relax_steps, output_steps, fix_phi_after_relax)
> [DEFAULT] Preprocess.relax_interface = ()
```

弛豫结果为：

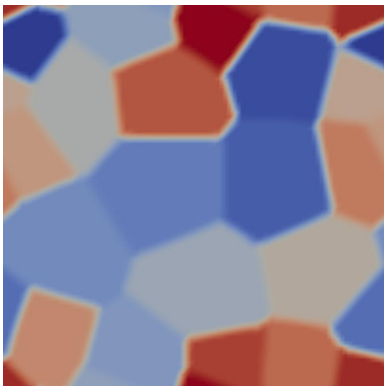


相混合

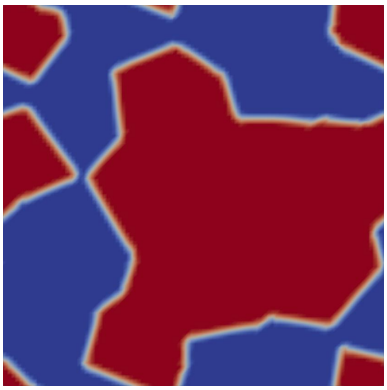
案例见 2_preprocess/2_merge_phis.mindes 和 2_preprocess/3_merge_phis_auto.mindes。

```
# Preprocess.merge_phis = {[(phi0,phi1,phi2, ... ), is_phi_c_merge], .... }
> [DEFAULT] Preprocess.merge_phis = {[()]}
# Preprocess.auto_merge_phis = (phis_index_0, phis_index_1, ... )
> [DEFAULT] Preprocess.auto_merge_phis = ()
```

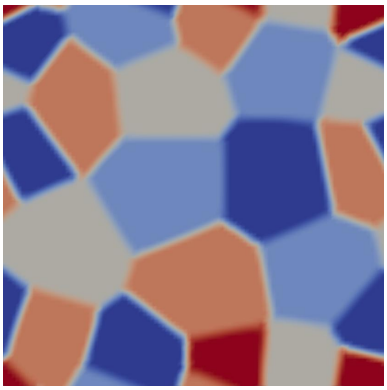
针对如下图所示随机生成的 20 个晶粒，可以将其中部分晶粒合并（在不影响模拟结果的情况下，可以节约计算内存，提高计算效率）



可将随机生成的 20 个晶粒混合为两个晶粒：



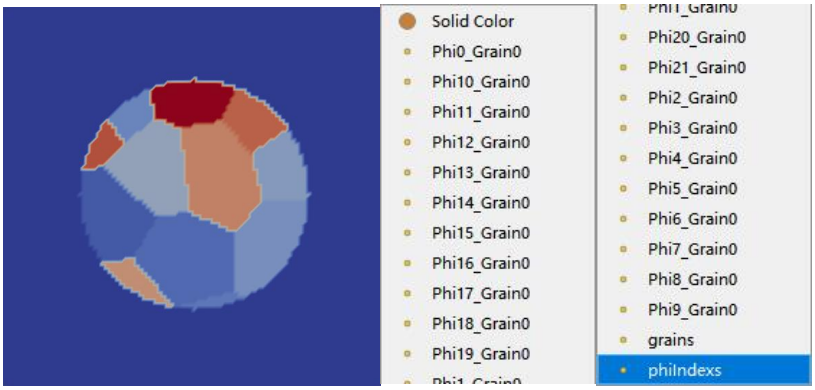
可将随机生成的 20 个晶粒中不接触的晶粒自动混合



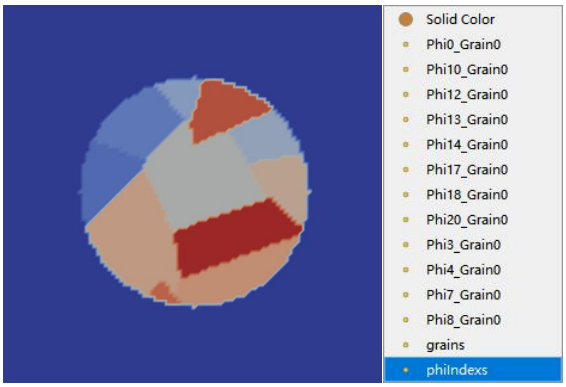
剔除不存在的相

```
> [DEFAULT] Preprocess.remove_inexistent_phis = FALSE
```

在构造复杂结构时难免会有相被覆盖或者消失，为节约内存、提高计算效率，剔除不存在的相是有必要的。以在椭圆中生成 voronoi 结构为例：



可见在该结构中存在 22 个晶粒，然而其中大部分在椭圆形核过程中被覆盖了，是不存在的，因此可使用该功能。相体积分数为 0.0 的晶粒会被自动剔除，结果只剩下 12 个晶粒：



相索引重排序：

```
> [DEFAULT] Preprocess.re_ordering_phis_indexs_from = 0
```

从索引 0 开始重新给晶粒/序参量排序，使其更加有序。

相填充

```
# Preprocess.fill_phis = {[(phi_index_0, phi_index_1, ... ), (phi_con_1,
phi_con_2, ... ), (total_con_1, total_con_2, ... ), (temperature)], .... }
> [DEFAULT] Preprocess.fill_phis = {[()]}
```

该功能可在对应相区中填充组分及温度。

3.3. 相界面

需要模拟相界面，首先需要选择相场动力学方程：

```
# ModelsManager.Phi.equation : 0 - Const, 1 - AllenCahn Standard, 2 - AllenCahn
Pairwise, 3 - CahnHilliard Standard
> [-VALID-] ModelsManager.Phi.equation = 2
```

一维、二维相界面

首先，可以选择界面能的模型：

```
# ModelsManager.Phi.InterfaceEnergy.int_gradient : 0 - Steinbach_1996 , 1 -
Steinbach_1999 , 2 - Steinbach_G2009
> [-VALID-] ModelsManager.Phi.InterfaceEnergy.int_gradient = 0
# ModelsManager.Phi.InterfaceEnergy.int_potential : 0 - Nestler_Well , 1 -
Nestler_Obstacle , 2 - Steinbach_P2009
> [-VALID-] ModelsManager.Phi.InterfaceEnergy.int_potential = 1
```

梯度能：0 - Steinbach_1996, 1 - Steinbach_1999, 2 - Steinbach_G2009

势能：0 - Nestler_Well, 1 - Nestler_Obstacle, 2 - Steinbach_P2009

```
# ModelsManager.Phi.Lij.const  = Lij_value
#                     .matrix = [(phi_i, phi_j, Lij_value), ... ]
#                     .block = [(phi_begin, phi_end, Lij_value), ... ]
> [-VALID-] ModelsManager.Phi.Lij.const = 1
# ModelsManager.Phi.Lij.Const.block = [(phi_index1, phi_index2, ... ), ... ]
> [DEFAULT] ModelsManager.Phi.Lij.Const.block = [()]
```

设置界面迁移率

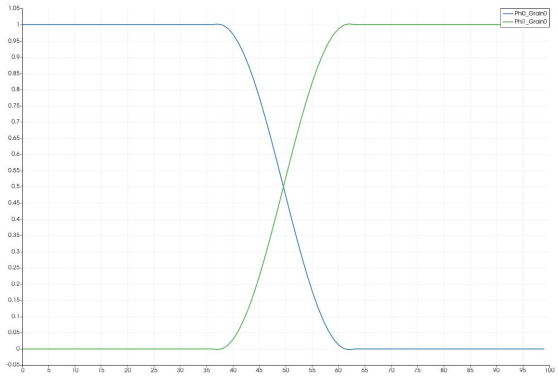
```
# ModelsManager.Phi.xi_ab.const  = xi_ab
#                       .matrix = [(phi_a, phi_b, xi_ab_value), ...]
> [-VALID-] ModelsManager.Phi.xi_ab.const = 1
# ModelsManager.Phi.xi_abc.const  = xi_ab
#                        .matrix = [(phi_a, phi_b, phi_c, xi_abc_value), ...]
> [-VALID-] ModelsManager.Phi.xi_abc.const = 0
```

设置界面能

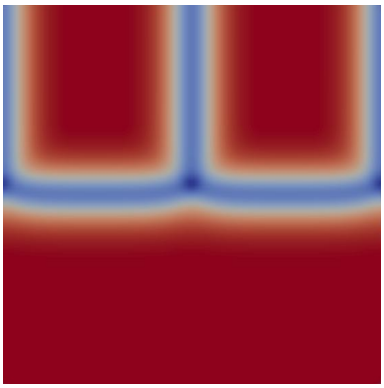
```
> [-VALID-] ModelsManager.Phi.InterfaceEnergy.int_width = 10
```

设置界面宽度

案例见 3_PCT_interface \1_two_phis_diffuse_interface\
allen_cahn_pairwise_1d.mindes

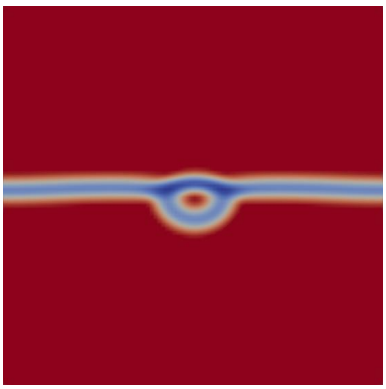


案例见 3_PCT_interface\2_three_phis_junction\allen_cahn_pairwise_2d.mindes



多相结

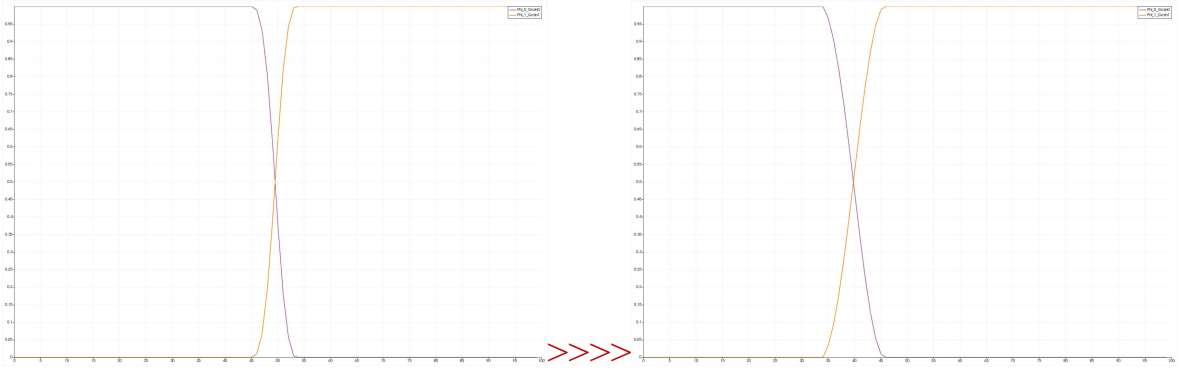
案例见
3_PCT_interface\2_three_phis_junction\allen_cahn_pairwise_curvature.mindes



给一个界面恒定驱动

相场框架下，界面的驱动力由体相能及组分势来驱动，假设相具有一个恒定的体能

```
# ModelsManager.Phi.BulkEnergy.const = [(phi_name, bulk_energy), ... ]
> [-VALID-] ModelsManager.Phi.BulkEnergy.const = [(Grain0,0.0),(Grain1,-0.01)]
```



其中紫色（Grain0）、黄色（Grain1）实线为两相体积分数分布，设定中 Grain1 体能量密度更低，所以黄色相长大。

界面上的相浓度

浓度场的演化，需先选择浓度场动力学方程：

```
# ModelsManager.Con.equation : 0 - Const, 1 - TotalConcentration, 2 -
PhaseConcentration, 3 - GrandPotential
> [DEFAULT] ModelsManager.Con.equation = 0
```

然后需要定义参数

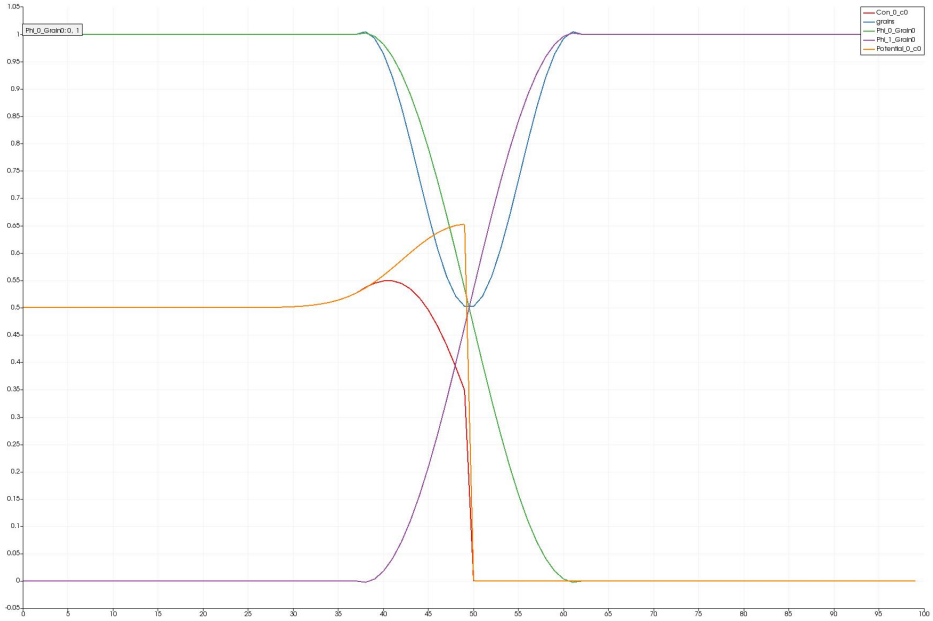
```
# ModelsManager.Con.valid_domain : 0 - Standard, 1 - Reverse
> [DEFAULT] ModelsManager.Con.valid_domain = 0
> [-VALID-] ModelsManager.Con.ValidDomain.phase_indexes = (0)
> [-VALID-] ModelsManager.Con.ValidDomain.threshold = 0.5
> [DEFAULT] ModelsManager.Con.GrandPotential.range = (-100000.0,100000.0)
```

其中，valid_domain 是设置标准有效区域还是反转的有效区域。
phase_indexes 对应的是有效的相，反转则是排除这个相是有效区域。
Threshold 是判断有效相相分数的判据，range 是对大势方程值的截断。

```
# ModelsManager.Con.Mii = [(phase_0_M_00 , phase_0_M_11, ...) , ... ]
> [DEFAULT] ModelsManager.Con.Mii = [()]
```

为每个相每个组分定义化学扩散速率。

案例见
3_PCT_interface\5_concentration_on_interface\3_smooth_grand_potential\allen_cahn_pair_wise_1d.mindes 大势方程的演化：



其中绿色、紫色实线为两相体积分数分布，红色线为总组分的分布，黄色线为绿色相区的扩散势分布。其中绿色相区被设定为有效区域。

3.4. 固体力学

打开物理场中的机械场开关

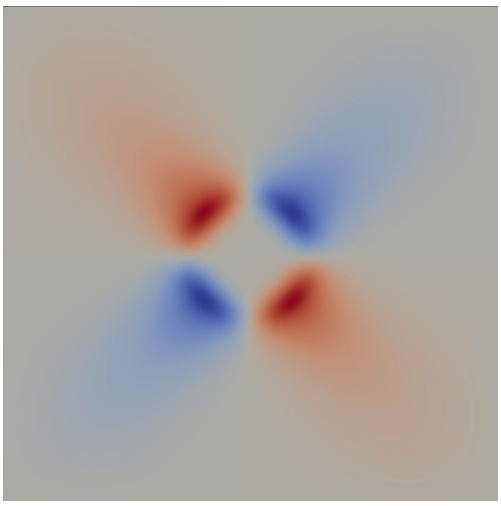
```
# Postprocess.physical_fields = (mechanic, fluid dynamic, electric)
> [-VALID-] Postprocess.physical_fields = (true,false,false)
```

然后设置机械场-弹性求解参数

```
# Postprocess.SolidMechanics.momentum_balance = 0 - None , 1 - Explicit , 2 -
Implicit (Ingo Steinbach) , 3 - Implicit (Armen G. Khachaturyan)
> [-VALID-] Postprocess.SolidMechanics.momentum_balance = 2
> [DEFAULT] Postprocess.SolidMechanics.restart_iterator_in_loop = FALSE
> [DEFAULT] Postprocess.SolidMechanics.Implicit.bc_ite_rate = 1.000000
# Postprocess.SolidMechanics.fix_boundary.type = (BC_X, BC_Y, BC_Z) , 0 -
Average , 1 - Strain , 2 - Stress
> [-VALID-] Postprocess.SolidMechanics.fix_boundary.type = (0,0,0)
> [-VALID-] Postprocess.SolidMechanics.write_displacement_field = TRUE
> [-VALID-] Postprocess.SolidMechanics.max_iteration_steps = 100
> [-VALID-] Postprocess.SolidMechanics.debug = TRUE
> [-VALID-] Postprocess.SolidMechanics.strain_accuracy = 1e-07
> [-VALID-] Postprocess.SolidMechanics.solid_phases = (Grain0,Grain1)
> [-VALID-] Postprocess.SolidMechanics.plasticity = FALSE
```

```
> [-VALID-] Postprocess.SolidMechanics.Stiffness.Grain0 =
[(C11,C12,C12,0,0,0),(C12,C11,C12,0,0,0),(C12,C12,C11,0,0,0),(0,0,0,Gm,0,0),(0,0,
0,0,Gm,0),(0,0,0,0,0,Gm)]
> [-VALID-] Postprocess.SolidMechanics.EigenStrain.Grain0 = (0,0,0,0,0,0)
> [-VALID-] Postprocess.SolidMechanics.Stiffness.Grain1 =
[(C11,C12,C12,0,0,0),(C12,C11,C12,0,0,0),(C12,C12,C11,0,0,0),(0,0,0,Gm,0,0),(0,0,
0,0,Gm,0),(0,0,0,0,0,Gm)]
> [-VALID-] Postprocess.SolidMechanics.EigenStrain.Grain1 = (0.01,0.01,0,0,0,0)
# Postprocess.SolidMechanics.StiffnessEigenStrain.model = (model_1,
model_2, ...)   0 - Normal, 1 - PhaseDependent_MolarVolume , 2 - RegionDependent
> [DEFAULT] Postprocess.SolidMechanics.StiffnessEigenStrain.model = (0)
```

得如下结果



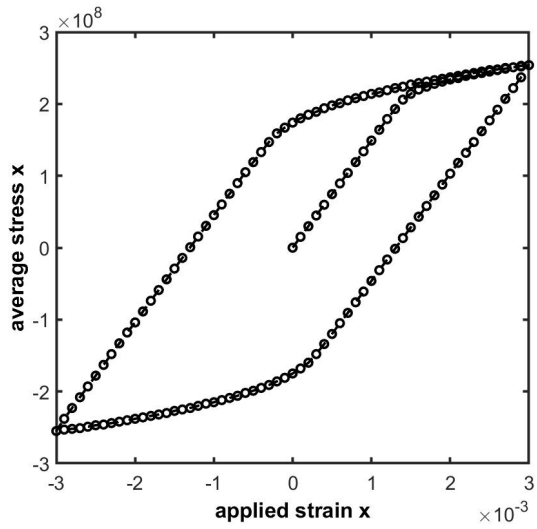
打开塑性求解开关

```
> [-VALID-] Postprocess.SolidMechanics.plasticity = TRUE
```

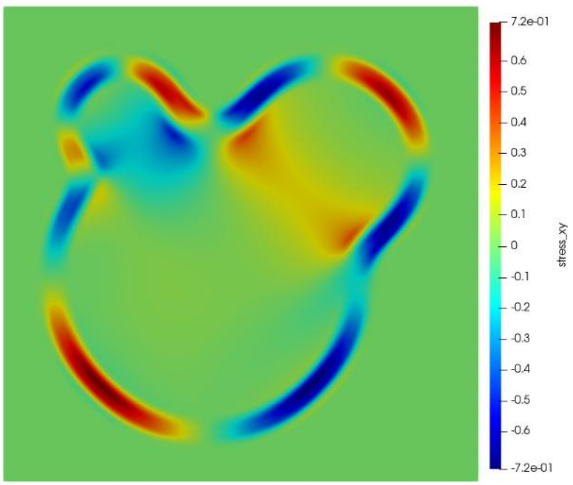
然后设置机械场-塑性求解参数

```
> [DEFAULT] Postprocess.SolidMechanics.Elastoplasticity.mapping_steps = 1
> [-VALID-] Postprocess.SolidMechanics.Plasticity.max_iteration_steps = 1
# Postprocess.SolidMechanics.Plasticity.yield_stress = ( phase_0_stress, ... )
#                                     .hardening_modulus = (phase_0_modulus, ...)
#                                     .shear_modulus = (phase_0_modulus, ...)
> [-VALID-] Postprocess.SolidMechanics.Plasticity.yield_stress =
(m_yeild_stress,i_yeild_stress)
> [-VALID-] Postprocess.SolidMechanics.Plasticity.yield_stress(0) = 2.75e+08
> [-VALID-] Postprocess.SolidMechanics.Plasticity.yield_stress(1) = 2.75e+11
> [-VALID-] Postprocess.SolidMechanics.Plasticity.hardening_modulus =
(m_hardening,i_hardening)
> [-VALID-] Postprocess.SolidMechanics.Plasticity.shear_modulus = (mG,iG)
```

在对材料体拉伸挤压时得到塑性求解得平均应力结果



也有对复杂颗粒得应力求解结果



案例详见：

0_basic_infile\4_postprocess\1_solid_mechanics_solvers\1_elastic_field_implicit.mindes

0_basic_infile\4_postprocess\1_solid_mechanics_solvers\2_elastic_complex_structure_implicit.mindes

0_basic_infile\4_postprocess\1_solid_mechanics_solvers\3_elastic_plastic_flow_implicit.mindes

等

3.5. 流体力学

打开流体力学开关：

```
# Postprocess.physical_fields = (mechanic, fluid dynamic, electric)
> [-VALID-] Postprocess.physical_fields = (false,true,false)
```

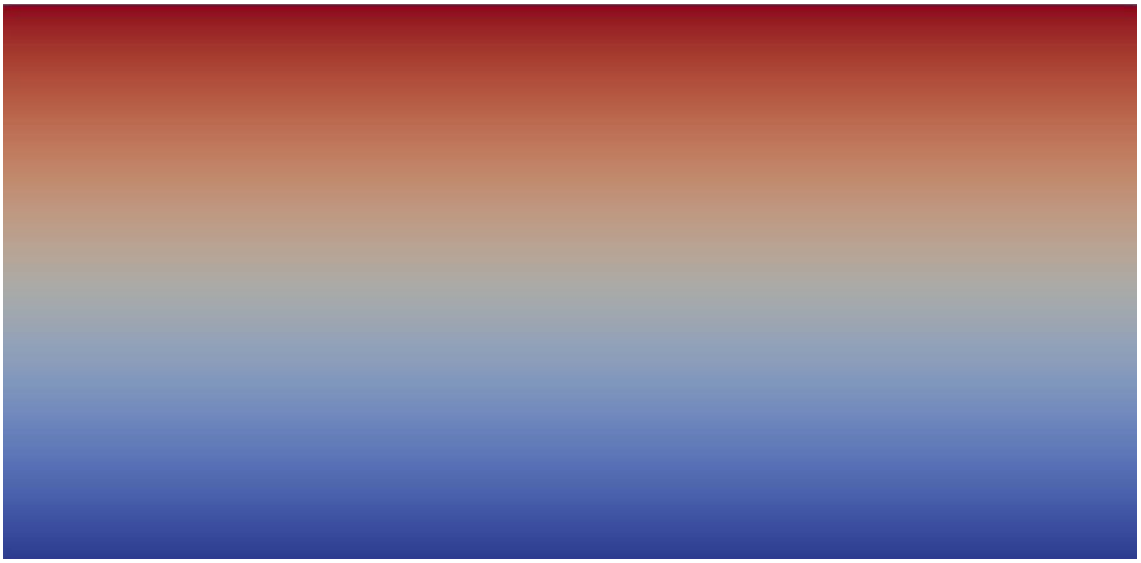
设置流体力学求解参数：

```
# Postprocess.FluidDynamics.solver = 0 - None , 1 - Pressure_Correction , 2 -
Lattice_Boltzmann
> [-VALID-] Postprocess.FluidDynamics.solver = 2
> [-VALID-] Postprocess.FluidDynamics.LatticeBoltzmann.max_iterate_steps = 10000
> [-VALID-] Postprocess.FluidDynamics.LatticeBoltzmann.debug_solver = TRUE
> [-VALID-] Postprocess.FluidDynamics.LatticeBoltzmann.debug_output_step = 100
> [-VALID-] Postprocess.FluidDynamics.LatticeBoltzmann.momentum_accuracy = 1e-06
# Postprocess.FluidDynamics.LatticeBoltzmann.solid_phases = (phase_name, ... )
> [-VALID-] Postprocess.FluidDynamics.LatticeBoltzmann.solid_phases = ()
# tau = viscosity / fluid_dt / Cs2 + 0.5
> [-VALID-] Postprocess.FluidDynamics.LatticeBoltzmann.liquid_viscosity = 0.1
> [-VALID-] Postprocess.FluidDynamics.LatticeBoltzmann.liquid_density = 1
# .LatticeBoltzmann.boundary_condition = (down_x,up_x,down_y,up_y)
#                                       0 - Wall, 1 - Period, 2 - Free, 3 -
Pressure, 4 - Normal_Flow
#                           .pressure = p0 , density0 = p0 / Cs^2 , Cs = 1 /
sqrt(3)
> [-VALID-] Postprocess.FluidDynamics.LatticeBoltzmann.boundary_condition =
(1,1,0,0)
> [-VALID-] Postprocess.FluidDynamics.LatticeBoltzmann.BC_Down_Y.wall_roughness =
1
> [-VALID-] Postprocess.FluidDynamics.LatticeBoltzmann.BC_Down_Y.wall_speed = 0
> [-VALID-] Postprocess.FluidDynamics.LatticeBoltzmann.BC_Up_Y.wall_roughness = 1
> [-VALID-] Postprocess.FluidDynamics.LatticeBoltzmann.BC_Up_Y.wall_speed = 0.1
# Postprocess.FluidDynamics.LatticeBoltzmann.source = ()
#             0 - Forces
> [DEFAULT] Postprocess.FluidDynamics.LatticeBoltzmann.source = ()
> [DEFAULT] Postprocess.FluidDynamics.LatticeBoltzmann.two_phase_flow = FALSE
```

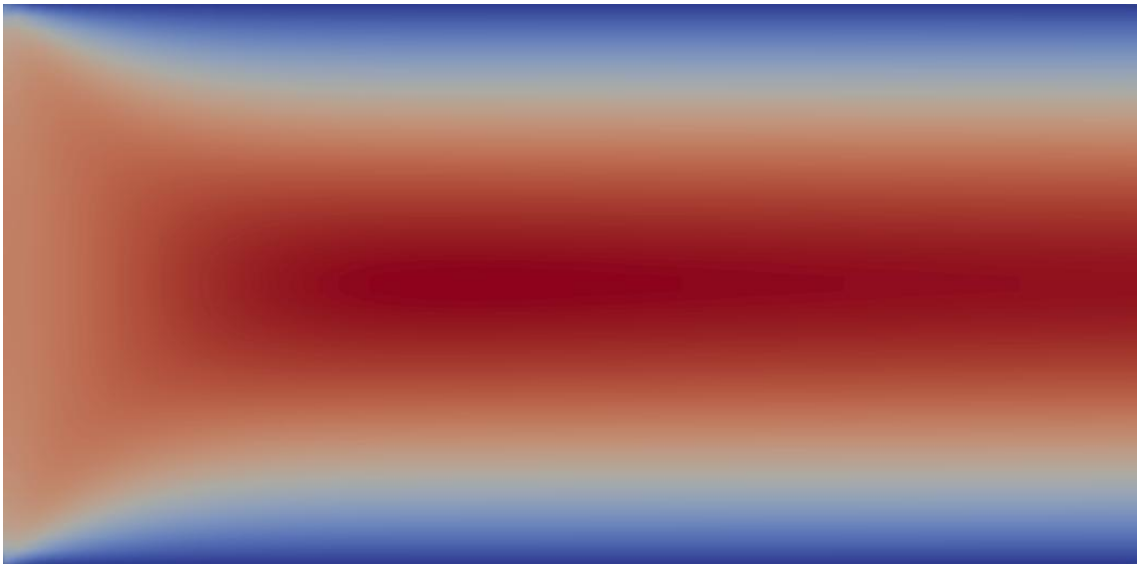
案例详见：

0_basic_infile\4_postprocess\2_fluid_dynamics_solvers

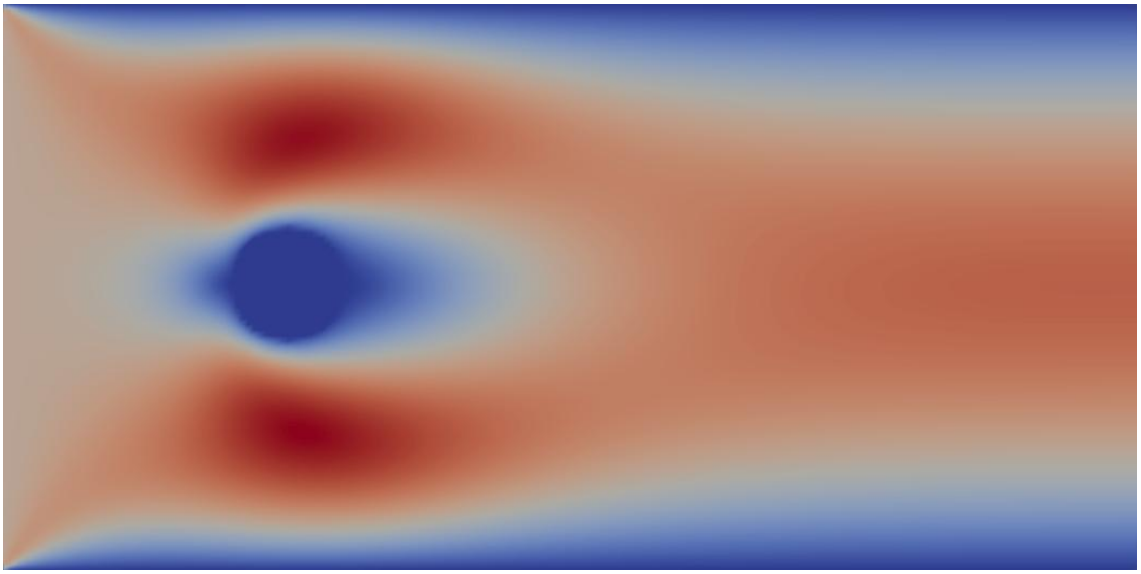
求解 Couette 流：



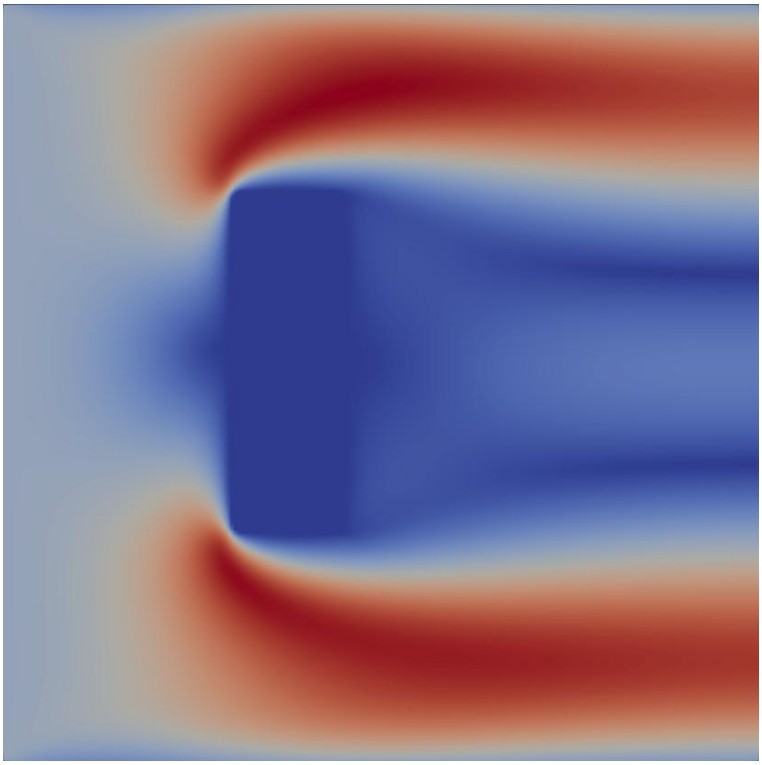
求解二维管道流：



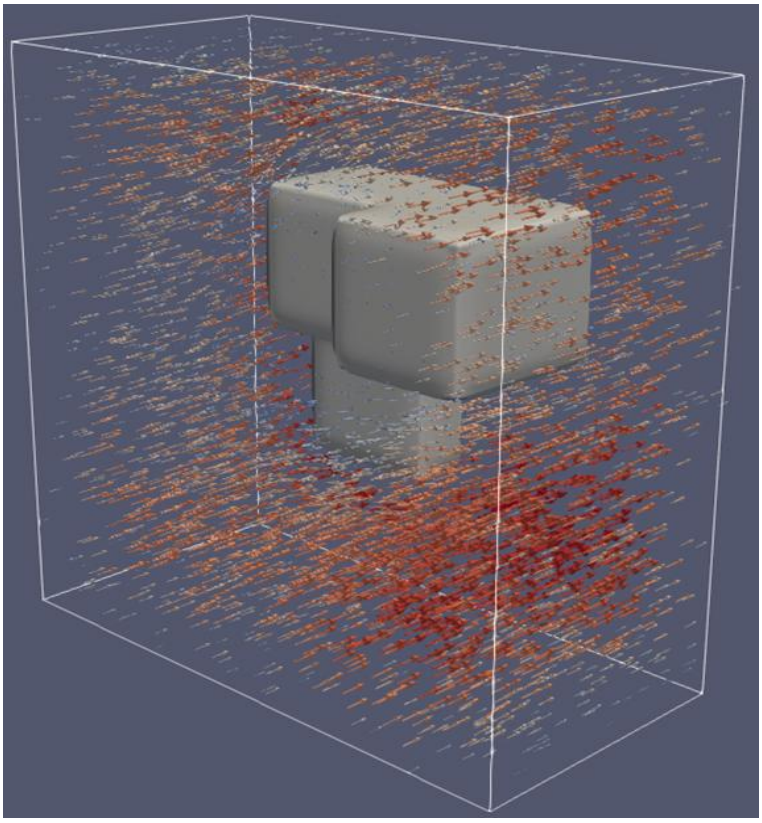
求解圆固体绕流（二维）：



求解方固体绕流（二维）：



求解方固体绕流（三维）：



叠加温度场，自然对流：