**Naming:**

Prefix k: constant value (const, constexpr, enum)

Prefix k0: value will be constant after initialization

Output variables through vector pointer arguments for dimensional related functions, i.e. 2D or 3D must give the size of vectors, generally 2 for 2D and 3 for 3D, before calling the function;

Struct: contain variables only (google style)

**Modules:**

1. grid:

grid\_generation\_manege\_serial: Generate

虚函数其实最主要的性能开销在于它阻碍了编译器内联函数和各种函数级别的优化，导致性能开销较大，在普通函数中log(10)会被优化掉，它就只会被计算一次，而如果使用虚函数，log(10)不会被编译器优化，它就会被计算多次。如果代码中使用了更多的虚函数，编译器能优化的代码就越少，性能就越低。

虚函数通常通过虚函数表来实现，在虚表中存储函数指针，实际调用时需要间接访问，这需要多一点时间。然而这并不是虚函数速度慢的主要原因，真正原因是编译器在编译时通常并不知道它将要调用哪个函数，所以它不能被内联优化和其它很多优化，因此就会增加很多无意义的指令（准备寄存器、调用函数、保存状态等），而且如果虚函数有很多实现方法，那分支预测的成功率也会降低很多，分支预测错误也会导致程序性能下降。

如果你想要写出高性能代码并频繁的调用虚函数，注意如果用其它的方式（例如if-else、switch、函数指针等）来替换虚函数调用并不能根本解决问题，它还有可能会更慢，真正的问题不是虚函数，而是那些不必要的间接调用。

正常的函数调用：

复制栈上的一些寄存器，以允许被调用的函数使用这些寄存器；

将参数复制到预定义的位置，这样被调用的函数可以找到对应参数；

入栈返回地址；

跳转到函数的代码，这是一个编译时地址，因为编译器/链接器硬编码为二进制；

从预定义的位置获取返回值，并恢复想要使用的寄存器。

而虚函数调用与此完全相同，唯一的区别就是编译时不知道函数的地址，而是：

从对象中获取虚表指针，该指针指向一个函数指针数组，每个指针对应一个虚函数；

从虚表中获取正确的函数地址，放到寄存器中；

跳转到该寄存器中的地址，而不是跳转到一个硬编码的地址。

通常，使用虚函数没问题，它的性能开销也不大，而且虚函数在面向对象代码中有强大的作用。

但是不能无脑使用虚函数，特别是在性能至关重要的或者底层代码中，而且大项目中使用多态也会导致继承层次很混乱。

那么有什么好方法替代虚函数呢？这里提供几个思路，读者请持续关注，后续会具体讲解：

使用访问者模式来使类层次结构可扩展；

使用普通模板替代继承和虚函数；

C++20中的concepts用来替代面向对象代码；

使用variants替代虚函数或模板方法。

My: 在大量循环体内使用虚函数，编译器无法优化