# 基于CUDA实现N-Body并行计算

## 1. 实验内容

1）了解N-Body问题，设计算法进行模拟计算；

2）使用CUDA开发并行计算程序；

3）使用OpenGL编写图形化程序，呈现动画效果。

## 2. 实验目的

1）深入了解并行编程及CUDA。

2）在项目的实践过程中，掌握更多开发技能。

## 3. 实验环境

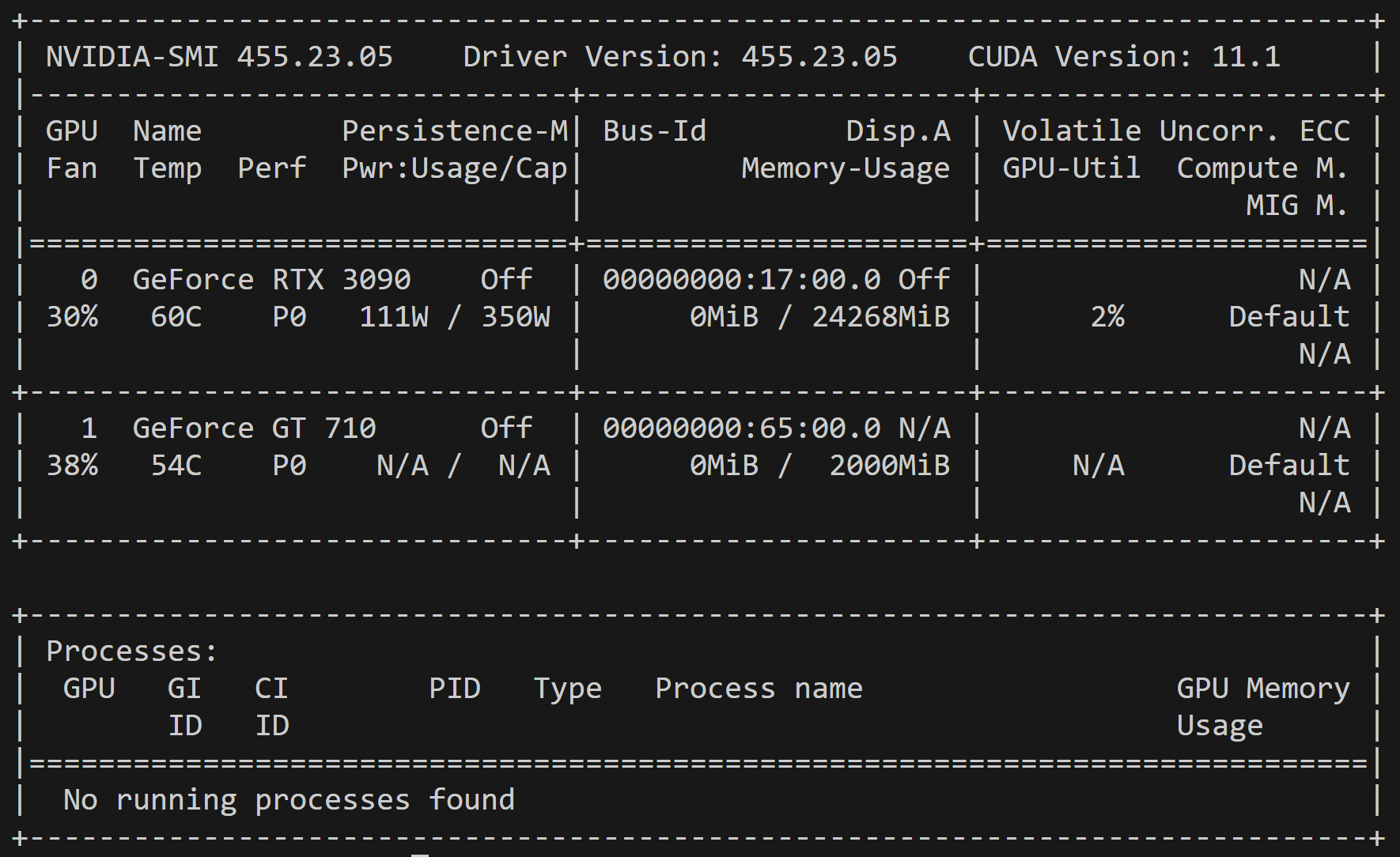
我们在远程主机上运行CUDA程序，获得模拟N-Body过程的数据，再使用本机的OpenGL程序解析数据并绘图。

### 3.1 CUDA及GPU环境

远程主机为Linux系统，Ubuntu版本为 20.04.5 LTS。

GCC版本为7.5.0。

CUDA版本为11.1，GPU型号为GeForce RTX 3090。详细信息见下图。



### 3.2 OpenGL环境

在64位Windows11系统上运行，仅使用自带的OpenGL，没有使用额外的库。

GCC版本为9.2.0。

## 4. 实验流程

### 4.1 分析N-Body问题

N-Body Simulation（多体运动模拟）需要在计算多个粒子运动参数的同时处理粒子之间的相互作用，可用于模拟宇宙中星体的运动。N-Body问题没有解析解，又对初值敏感，一般使用模拟算法解决。

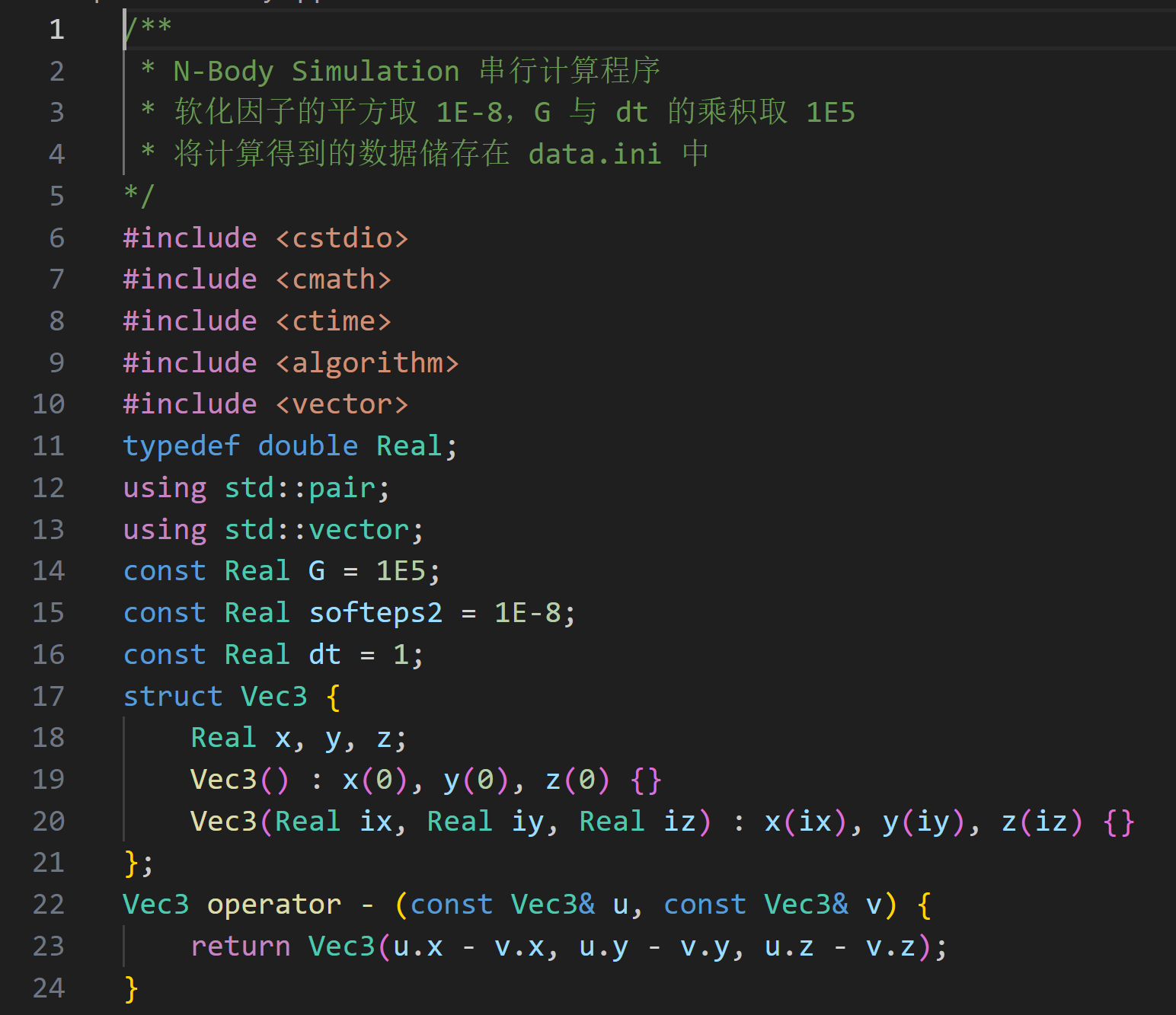
具体来说，本实验需要模拟三维空间中多个质点的运动，质点具有质量、位置矢量和速度矢量三个属性，质点之间的相互作用仅有经典力学下的万有引力。万有引力的大小与物体质量成正比，与距离成反比。每一时刻，物体会根据速度改变位置，同时根据本身与其他物体之间的万有引力产生的合加速度改变速度。在模拟过程中，可以将时间划分为多个小段，每个迭代周期中，基于一个时间小段计算位置与速度的改变量，并更新运动参数。

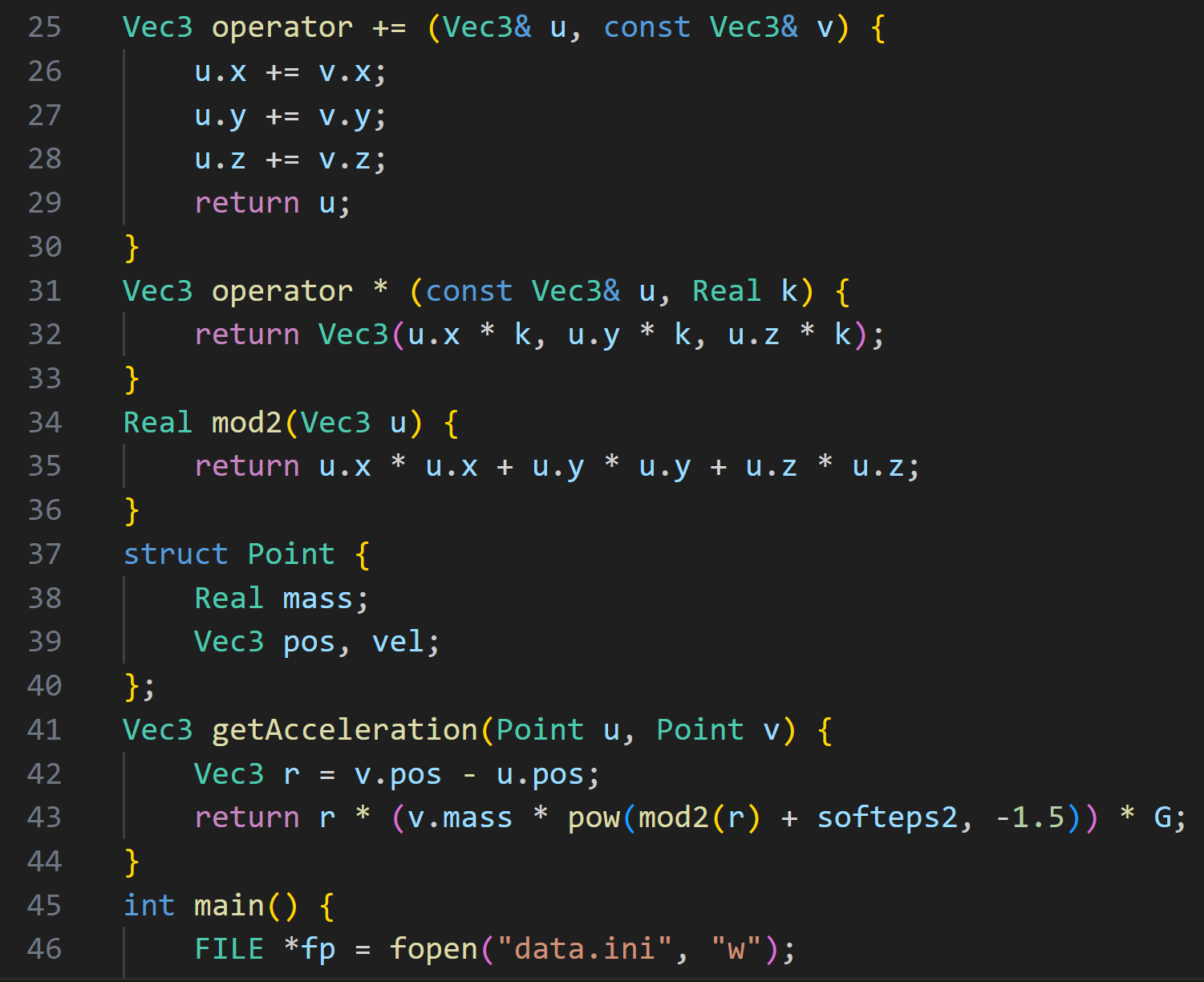
值得注意的是，朴素的平方反比公式不方便处理距离趋近于零的情况。在精度有限的模拟中，这很可能产生问题。同时，这意味着物体对自身会产生一个未定义的力，在运算时必须用逻辑运算排除自身，降低运算效率。因此，我们可以引入软化因子。使用如下的公式计算万有引力产生的加速度时，在距离大的情况几乎没有影响，同时可以较好地处理距离极小的情况。

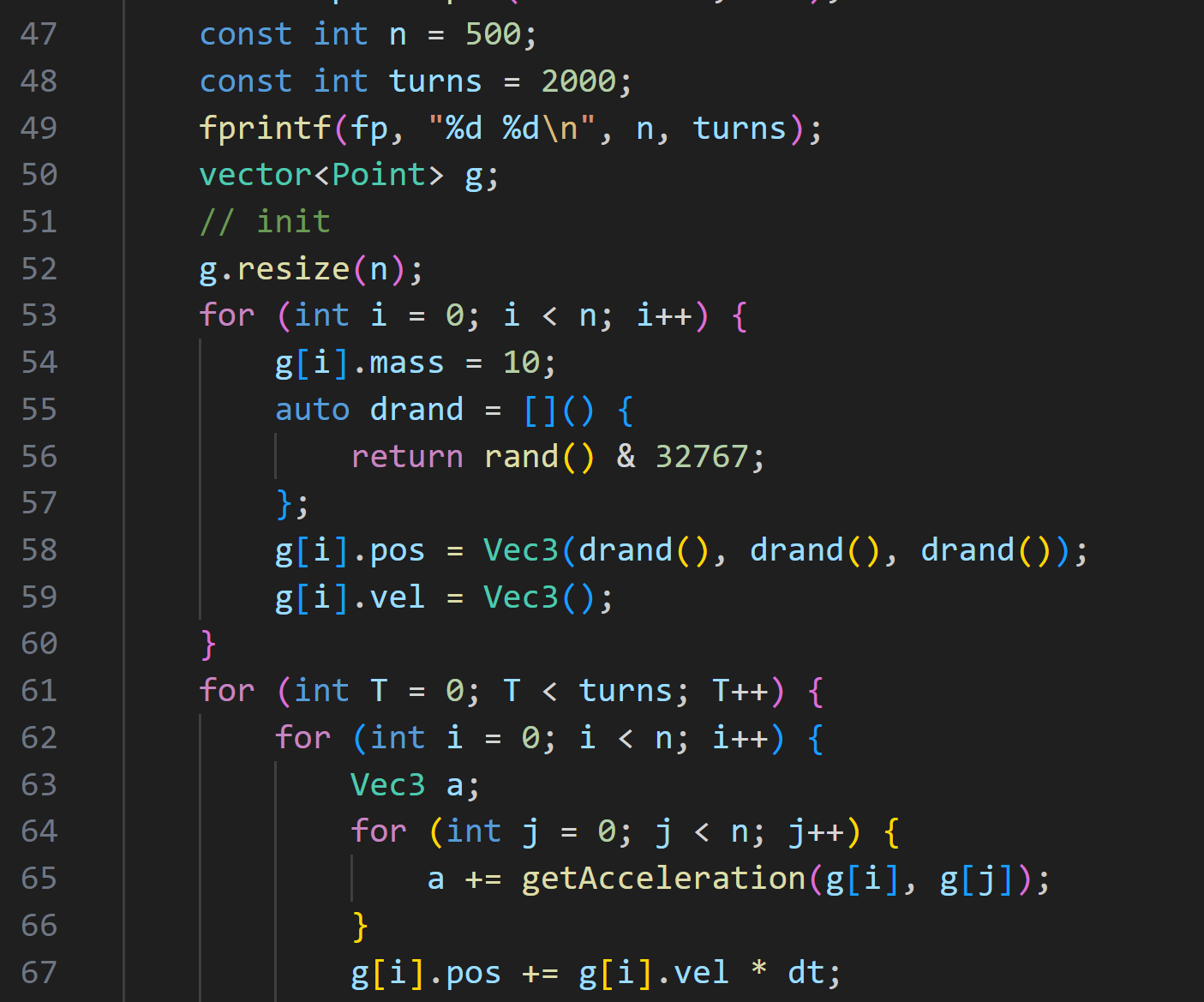
在此公式中，代表由质点指向质点的向量，是一个接近于的软化因子。

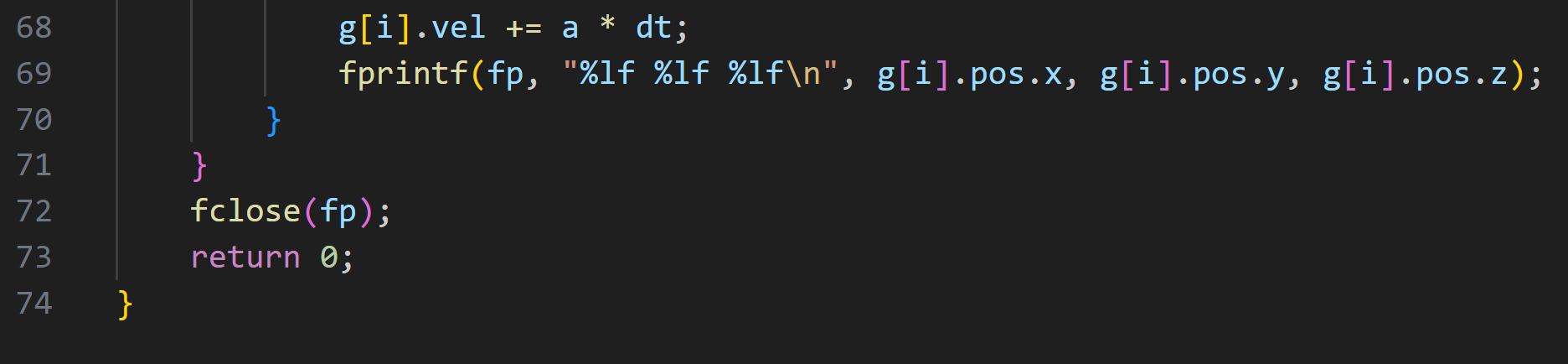
由于实验的主要目的是使用CUDA，因此我们用最朴素的All-Pairs模拟法，在每一时刻计算每对质点之间的相互作用，复杂度为 。

### 4.2 编写串行程序









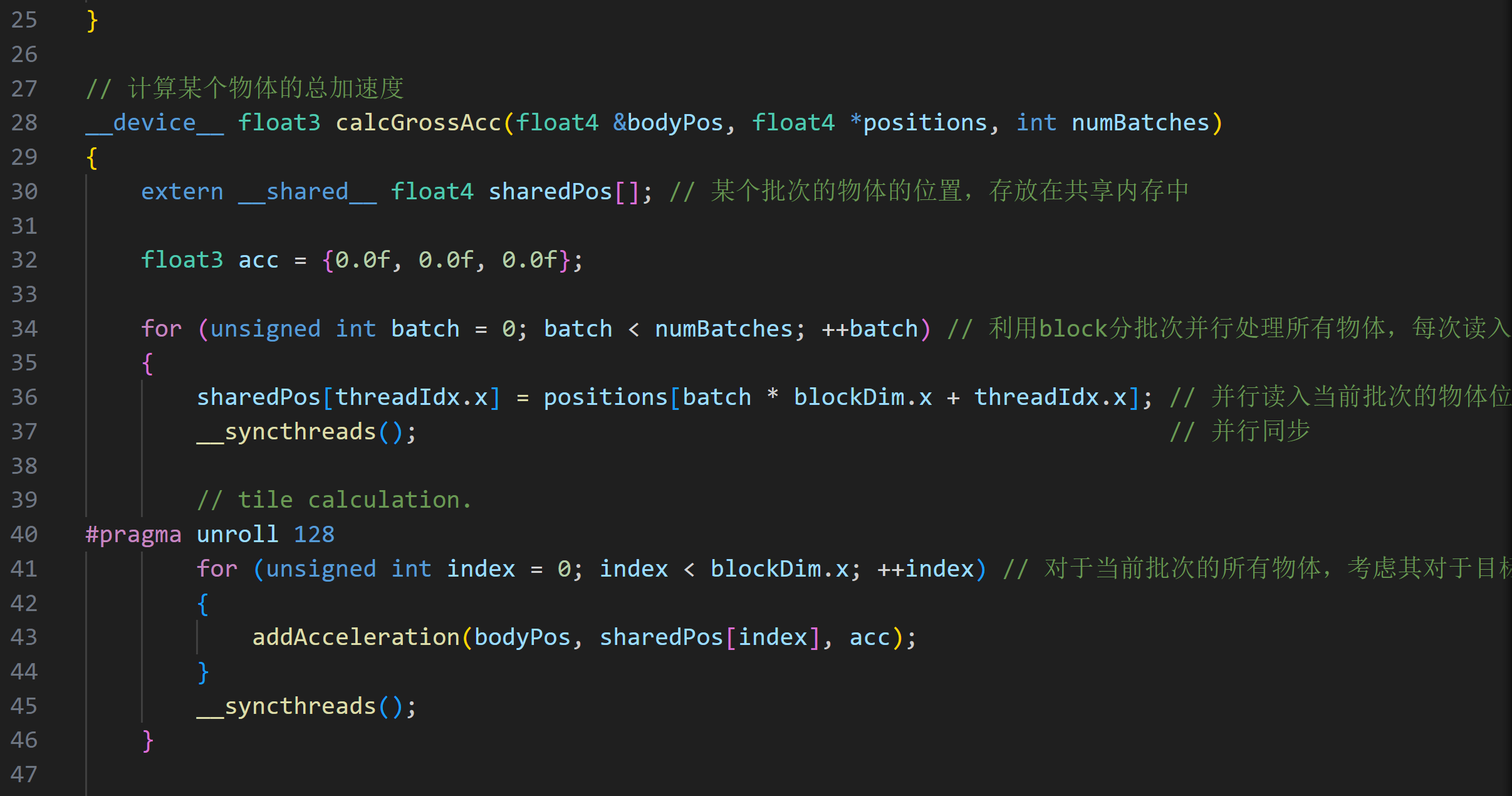
这是普通的C++程序，按照常规方式运行即可。

运行结果：按照预定格式输出数据到data.ini。

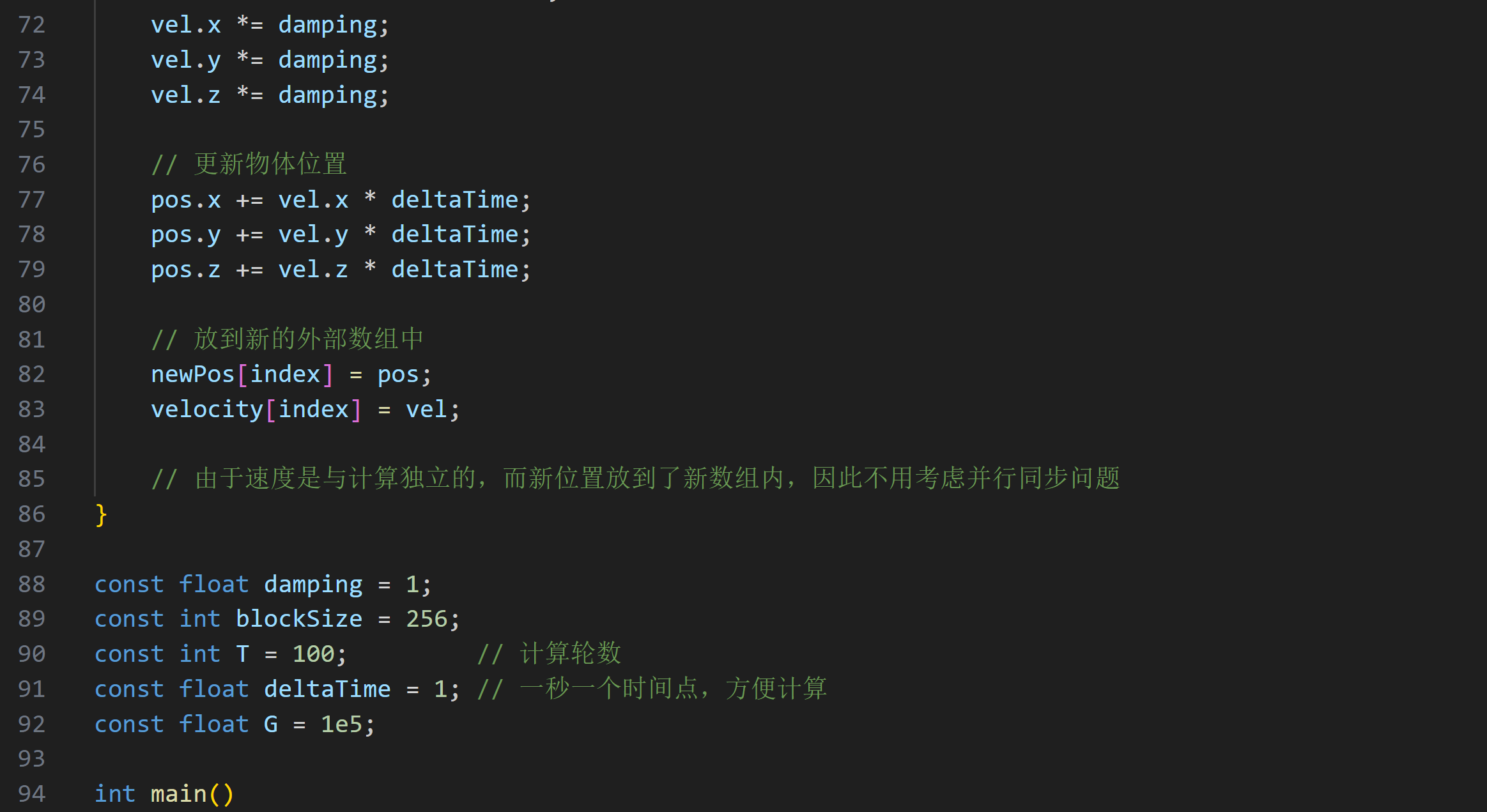
在截图后又更新了程序，最终版本的代码略有差别，请以附件提供的源代码为准。最终版程序加入了Linux下的计时函数，因此可能无法在Windows下编译运行。

### 4.3 编写并行程序



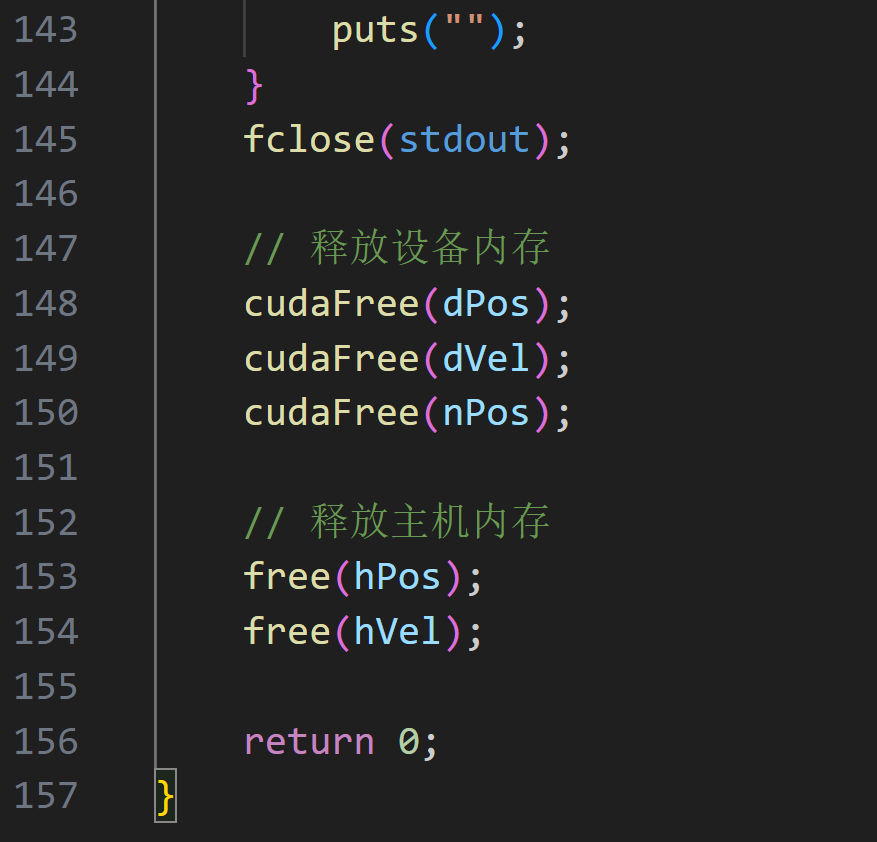






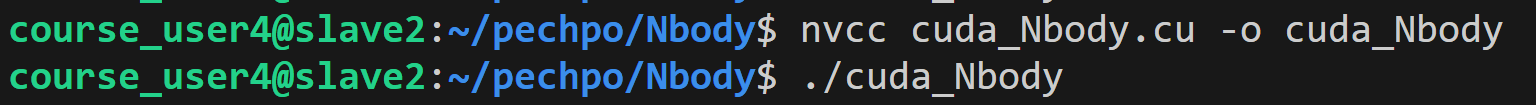




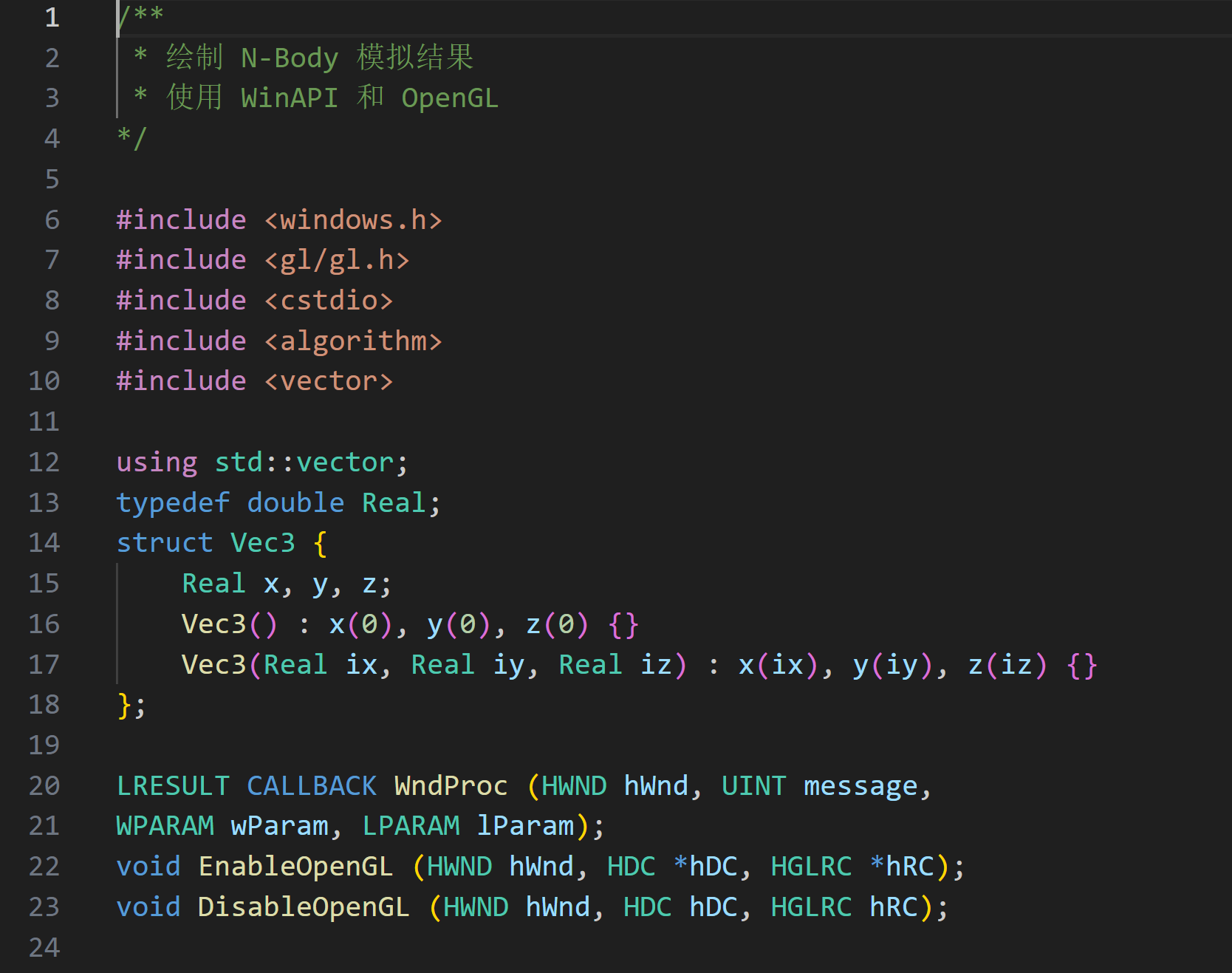


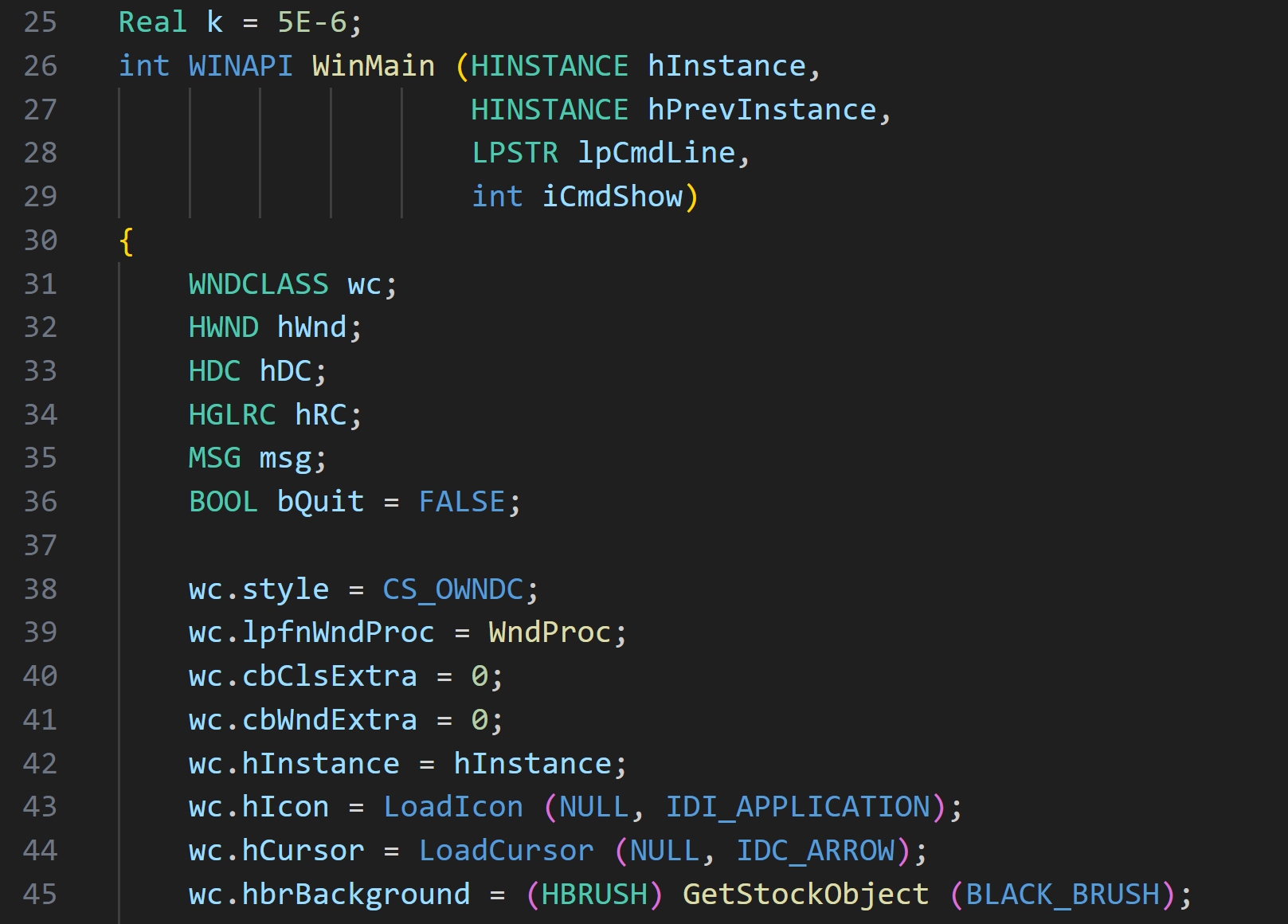
在截图后又更新了程序，最终版本的代码略有差别，请以附件提供的源代码为准。

编译指令为“nvcc \*.cu -o \*”。

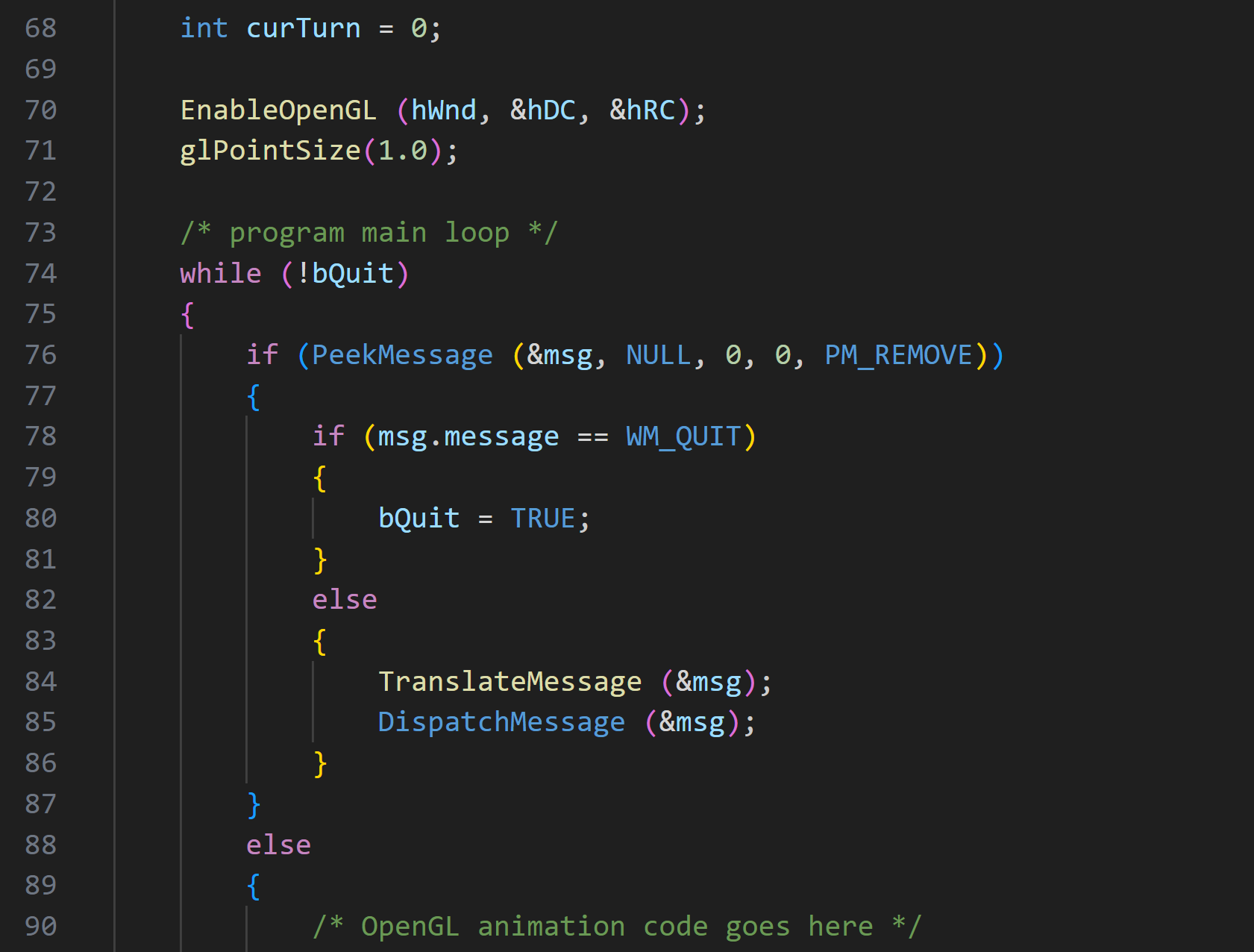


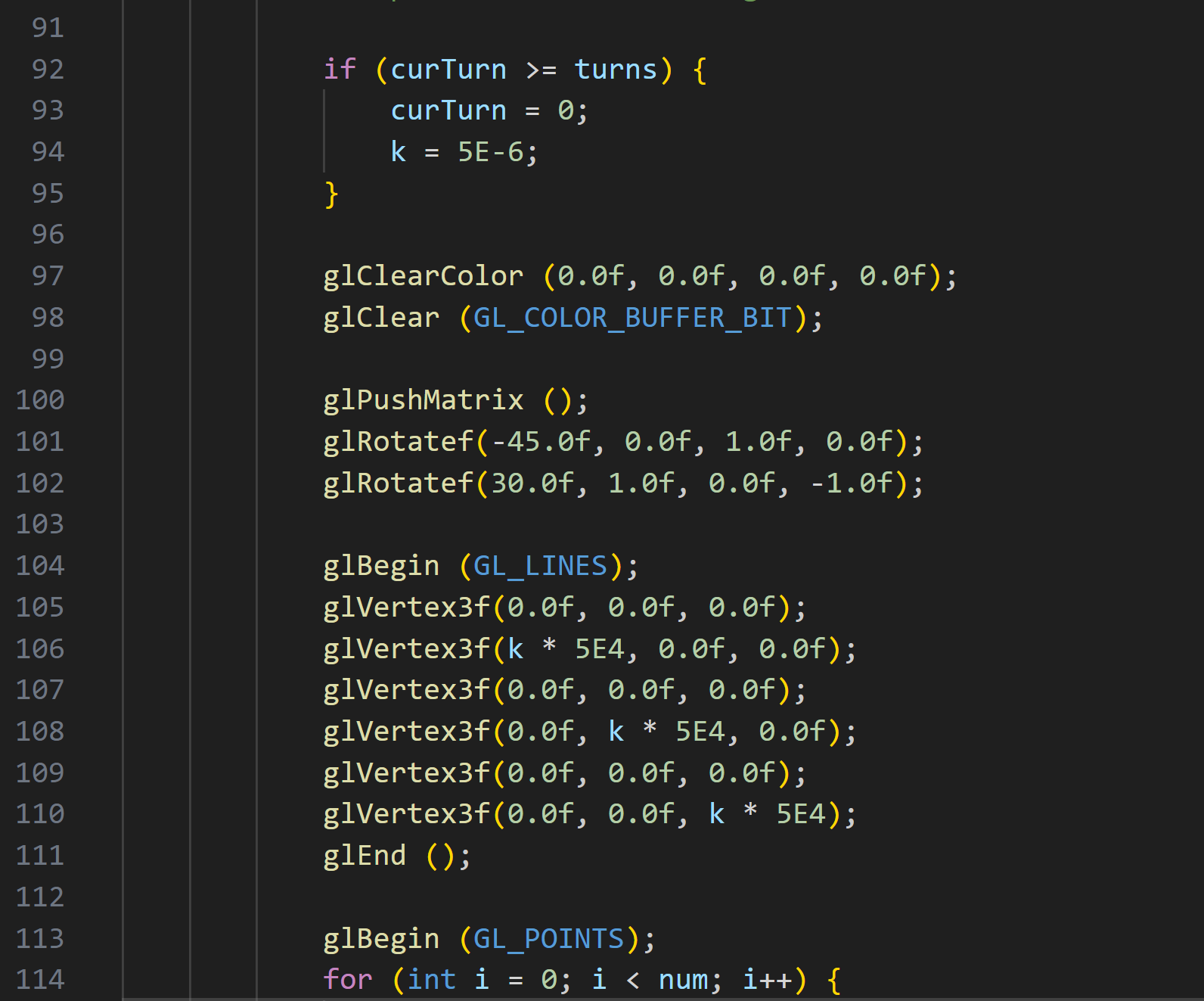
### 4.4 编写图形化程序

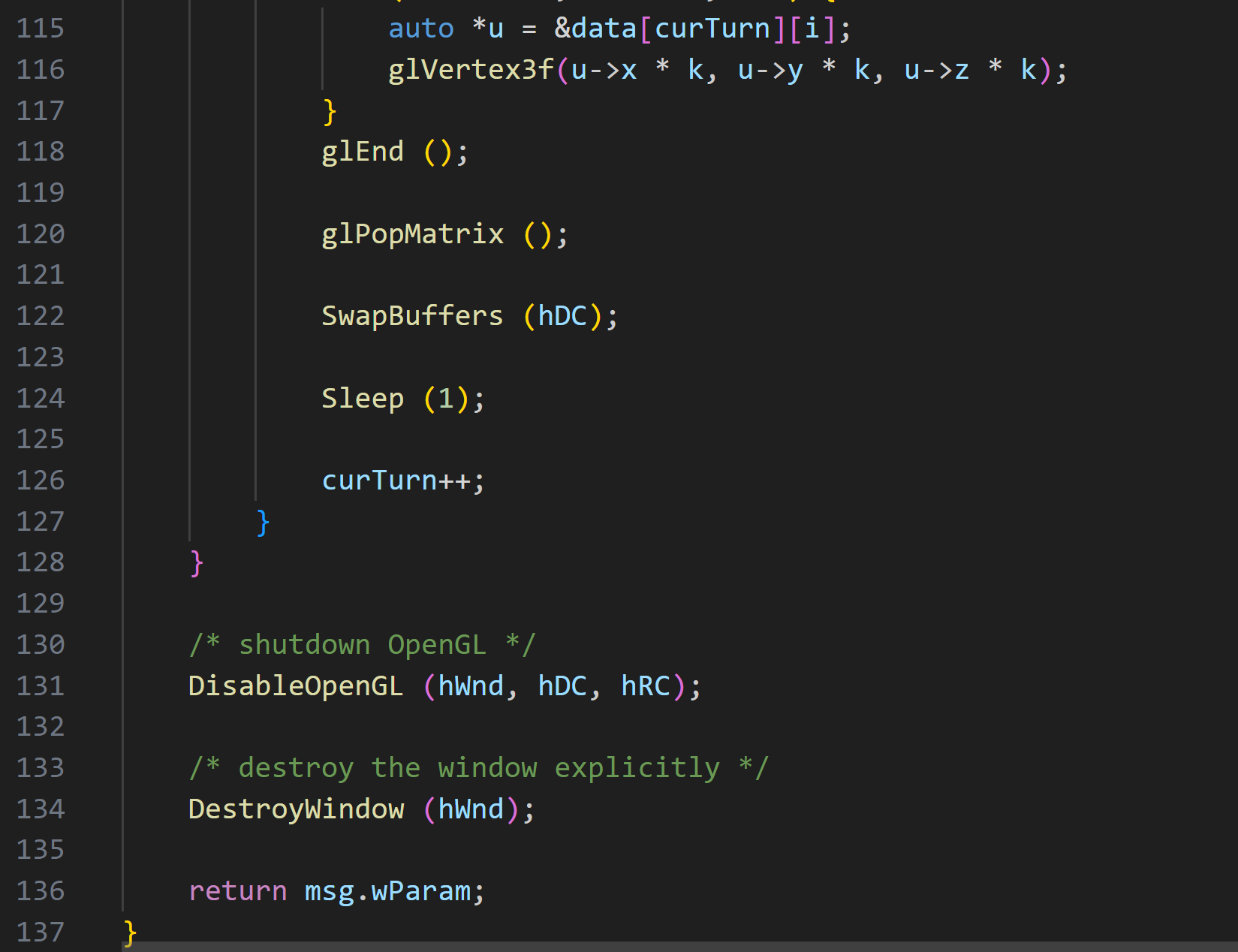


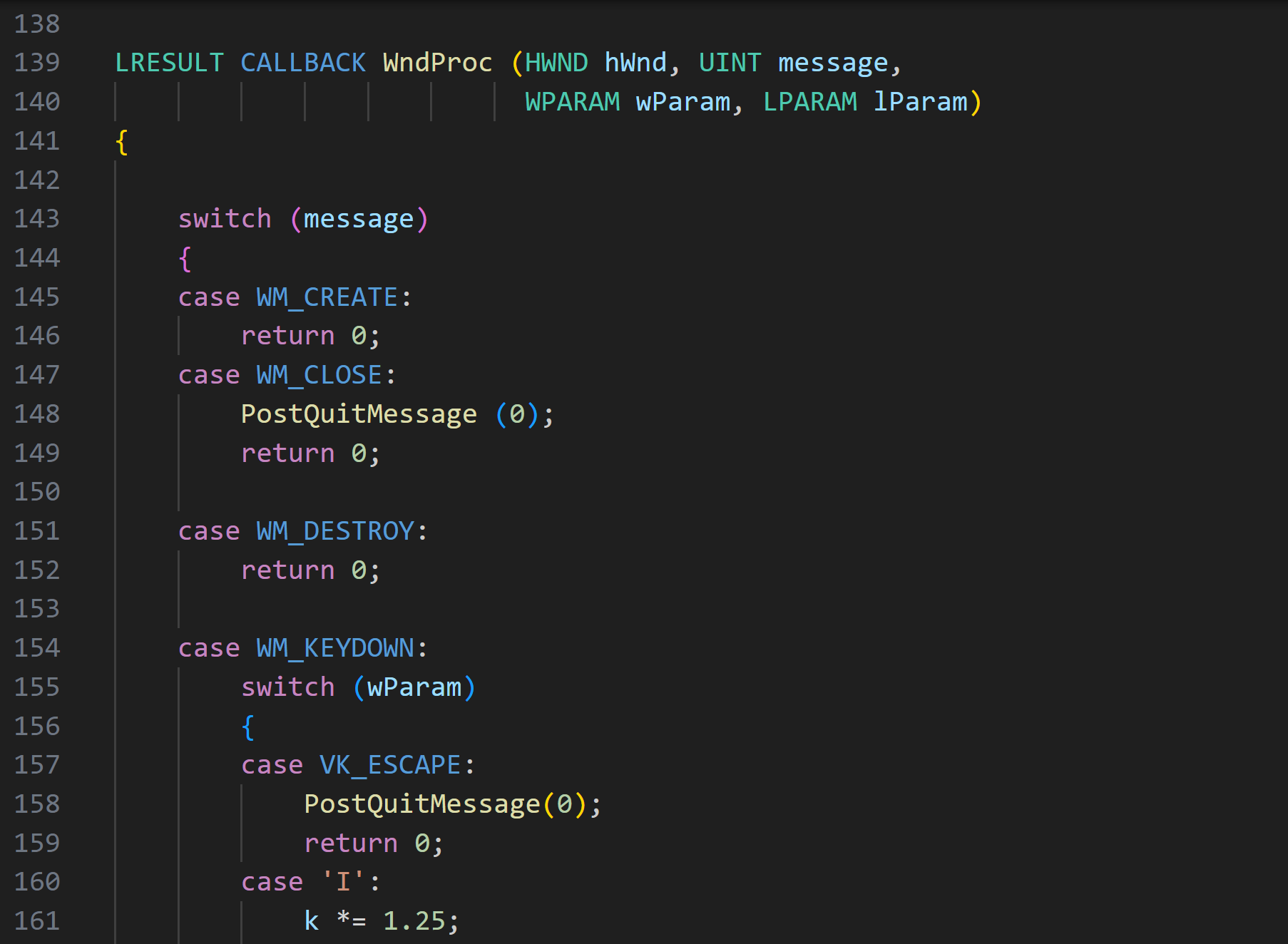




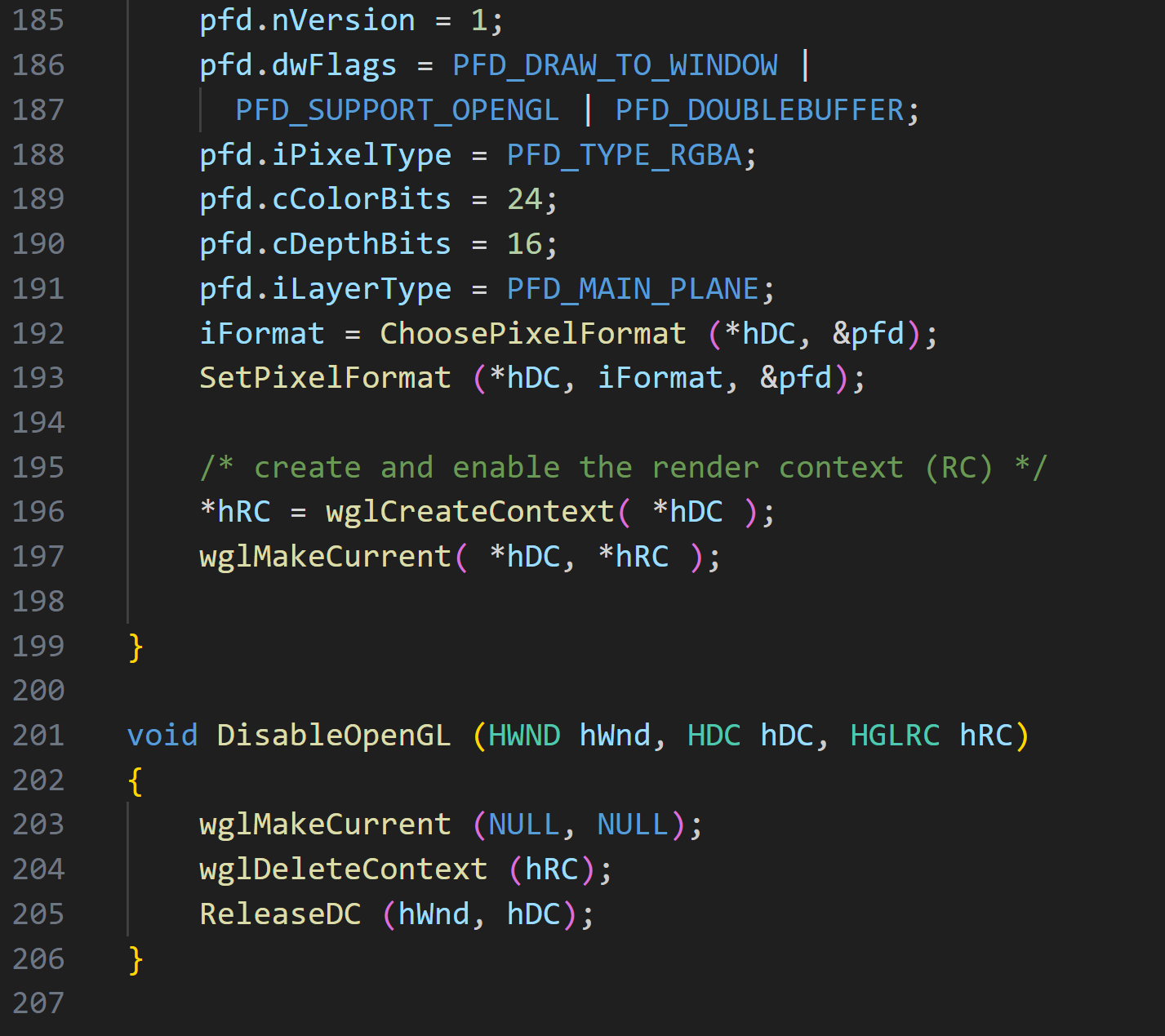












在截图后又更新了程序，最终版本的代码略有差别，请以附件提供的源代码为准。

编译时需要指令“-mwindows -lopengl32”。在Windows系统中应该无需安装额外的库即可运行。



运行后弹出窗口并播放粒子运动过程。

## 5. 实验结果展示

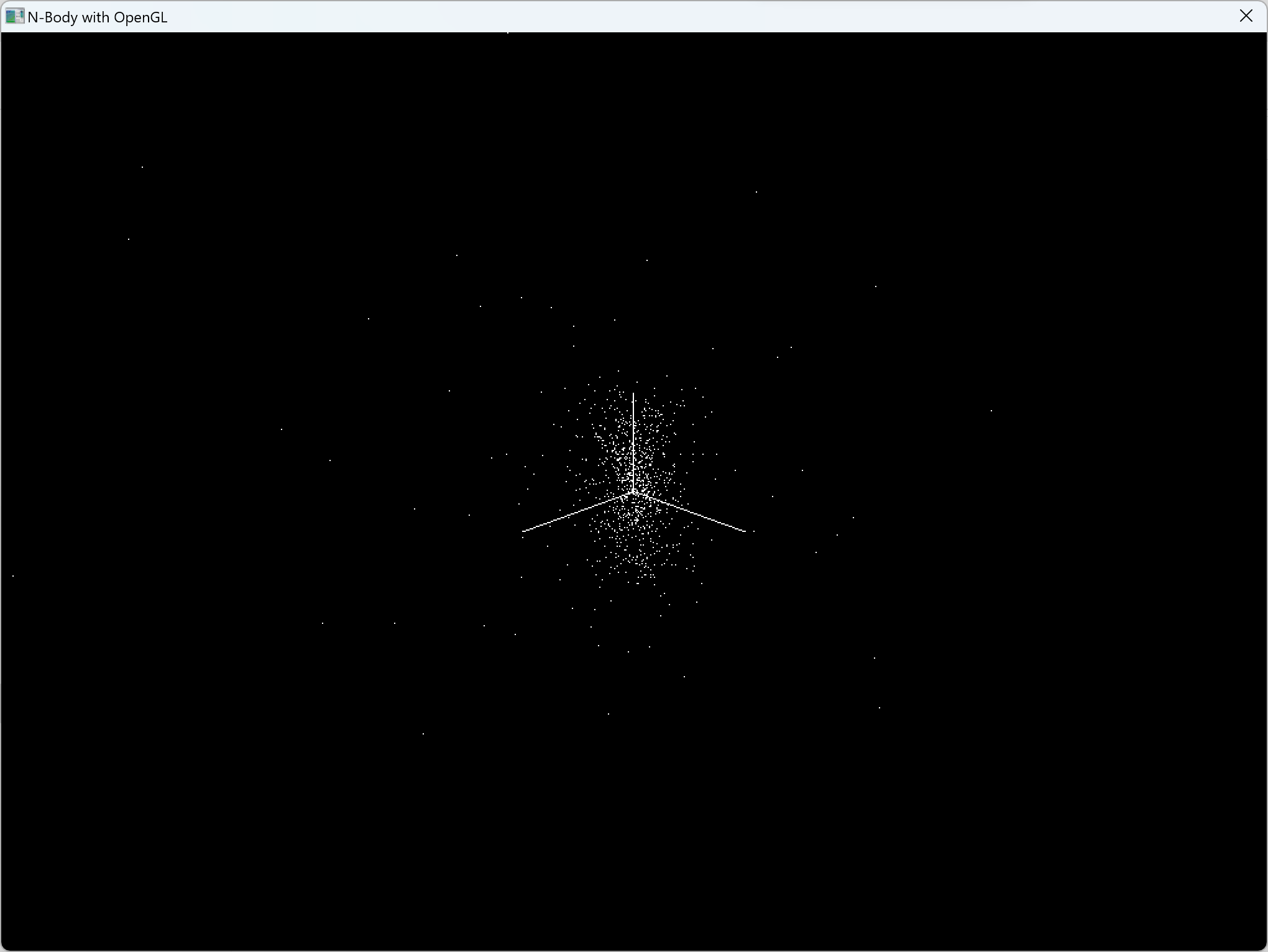
首先编译运行gen.cpp，生成测试数据data.in。

接下来可以运行串行程序seq.cpp，读入data.in，输出data.out。

同样的，可以运行并行程序cuda\_Nbody.cu来读入data.in，输出data.out。

若要观看动画效果，需要将data.out传输至Windows系统，和animation.exe放在同一目录下，运行animation.exe即出现动画。程序刚开始运行时，需要载入数据，会有一段时间的卡顿。

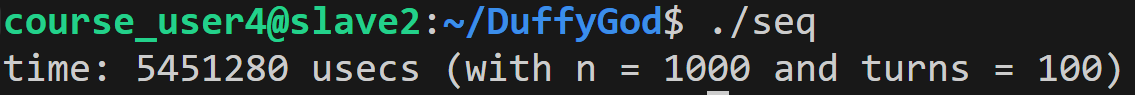
播放期间，按下键盘上的I键或O键可以放大或缩小图形。



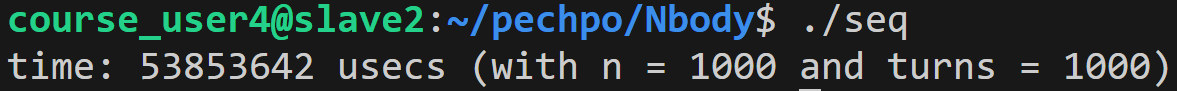
在压缩包中，为减少提交文件中冗余信息，我们附带了一份数据量较小的data.out。如果需要测试大量粒子的运动过程，可以编译运行并行计算程序来生成一份更大的数据。

## 6. 实验分析

串行程序在点数取1000且轮次数取100时，用时为 5.45秒。

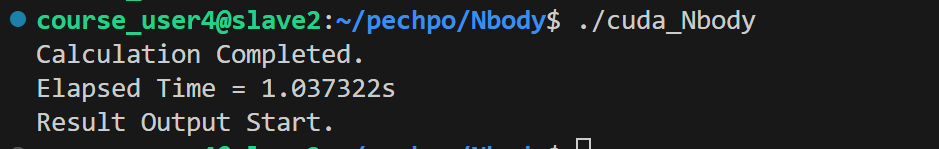


在点数取1000且轮次数取1000时，用时为 53.85秒。



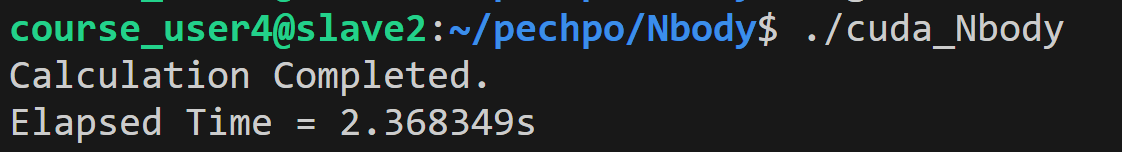
同样是点数取1000且轮次数取1000时，并行程序的用时为1.04秒。

**可见，在点数1000时，加速比已经达到约50倍。**

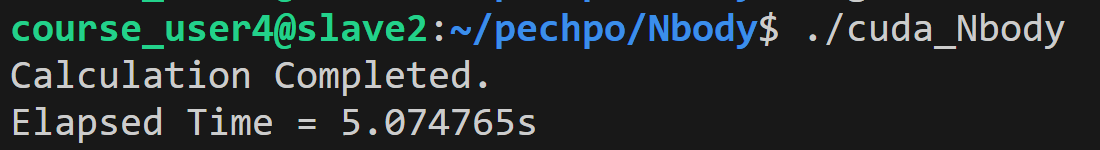


不过，即使大幅增加数据范围，并行程序的用时依然很短。以下轮次数始终为1000，点数不断上升。

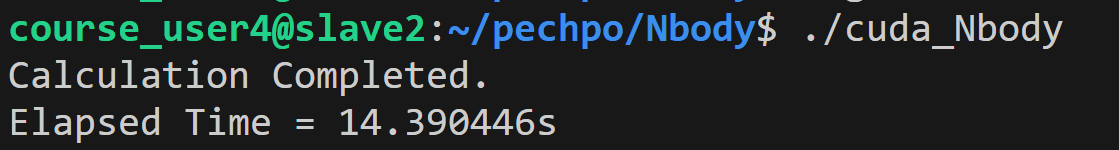
当点数取25000时，用时为2.37秒。



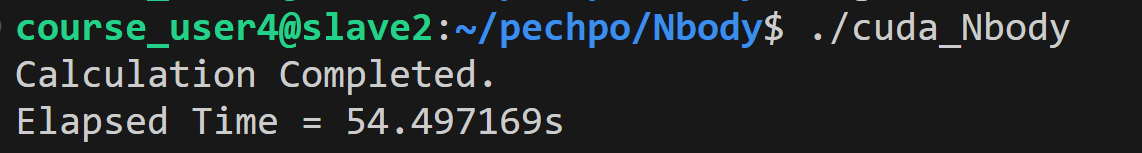
当点数取50000时，用时为5.07秒。



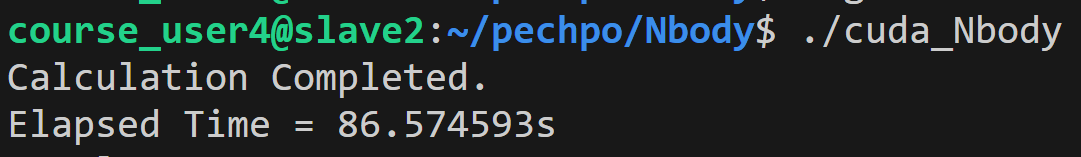
当点数取100000时，用时为14.39秒。



当点数取200000时，用时为54.50秒。



当点数取250000时，用时为86.57秒。



点数达到200000后，用时随点数的增长关系近似于平方，与算法的时间复杂度一致，可以认为此时CUDA的性能几乎得到充分发挥。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 点数 | 25000 | 50000 | 100000 | 200000 | 250000 |
| 用时/秒 | 2.37 | 5.07 | 14.39 | 54.50 | 86.57 |

程序中定义每个block的线程数为256。进行计算可知，在n=10000时，block数量为10000/256≈40，而RTX3090的流式处理器（SM，Stream Multiprocessor）数量为82，因此此时点数过少，没有让GPU跑满，所以这个数据段的计算时间是大约线性增长的。在n更小时，甚至不增长（维持在1s左右）。

在n=200000时，block数量为20000/256≈782，此时平均每个SM运行9.5个block，逼近性能极限。因此这个数据段的计算时间是大约平方增长的，符合算法的理论复杂度O(n2)。

## 7. 问题与讨论

1）将数据存储在文本文件中的读取效率太低，导致animation.exe开始运行时卡顿一段时间。可以尝试使用二进制文件或数据库存取进行优化。

2）为了追求最好的呈现效果，需要让物理参数和初始粒子的质量、速度、位置的量级相匹配，否则形成的系统可能非常不稳定。需要不断调整来完善。

3）要在OpenGL绘图程序中实现更好的呈现效果（如辉光特效），会涉及到更深入的计算机图形学知识。

4）如果增加实验细节，还可以优化一半的运行时间（力是对称的），并且还可以考虑调整block的线程数等参数，尝试找到更佳的实现模式。

## 参考资料：

[1] CUDA C++ Programming Guide https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html

[2] https://github.com/ZeusYang/NBodySimulation/tree/master