

# 使用 CUDA 的 SPH 流体模拟

生命科学学院 计算机科学与技术双学位 1500012204 李博

## 一、简介

在本项目中，实现了一个简单的约束在立方体中的流体模拟程序。可以用 OpenGL 实时观看渲染的粒子，或者把某一帧的流体粒子重建表面，导出为.obj 模型。在 results 文件夹中有几张用 Maya 渲染过的流体图片，实时粒子渲染截图，以及导出的.obj 模型。

## 二、环境搭建

本项目在 Windows 10 上编写，使用 Visual Studio 2015 Community，尚未做跨平台考虑。请在该环境下测试使用。VS 2017 应该也是可以的。

本项目使用了 CUDA，所以要求有 Nvidia 显卡才能运行（我的显卡是 GTX 960M）。另外还必须安装 CUDA Toolkit，请到参考 2 给出的地址下载安装，我使用的版本是 8.0.61。

另外还使用了 freeglut 和 glew，我在 VS 项目链接器的设置中加了 freeglut.lib, glew32.lib, 可执行的 sph.exe 所在目录中添加了 freeglut.dll 和 glew32.dll，应该无需多操心了。

## 三、一些实现细节

SPH (Smoothed Particle Hydrodynamic, 光滑粒子流体动力学)是一种用粒子模拟流体的方法。我们不会在这里详细介绍 SPH，在参考 1 的博客中有一个简明而周到的介绍，想要详细了解的话可以参看。实现了博客中描述的重力、压力、黏滞力，没有考虑表面张力（真实情况下表面张力与前面的力相比是很小的）。

SPH 方法得名于它使用“光滑核函数”进行模拟计算。一个粒子对空间某点的某种“属性”（密度、压强等）的贡献可以用光滑核函数来决定。这种函数的值随粒子距离而下降，到某一个半径  $h$  之后变成 0。SPH 考虑的是 Navier-Stokes 方程的一种简单形式，经过推导后最终得出的粒子  $i$  的加速度为：

$$\vec{a}(r_i) = \vec{g} + m \frac{45}{\pi h^6} \sum_j \left( \frac{p_i + p_j}{2\rho_i \rho_j} (h - r)^2 \frac{\vec{r}_i - \vec{r}_j}{r} \right) + m \mu \frac{45}{\pi h^6} \sum_j \frac{\vec{u}_j - \vec{u}_i}{\rho_i \rho_j} (h - r)$$

with  $r = |\vec{r}_i - \vec{r}_j|$

相关符号的含义请到 blog 中查看。

首先，我们计算出每个粒子的加速度，然后在时间  $dt$  上步进数值积分得到速度和新的位置。每一个粒子的加速度和积分都可以独立完成，而且粒子数量很大，所以非常适合 GPU 并行计算。每一个粒子的更新在一个线程上完成。积分格式是 LeapFrog 蛙跳格式，比单步 Euler 格式有更好的精度：

$$\mathbf{v}^{i+1/2} = \mathbf{v}^{i-1/2} + \mathbf{a}^i \Delta t$$

$$\mathbf{r}^{i+1} = \mathbf{r}^i + \mathbf{v}^{i+1/2} \Delta t,$$

由于一个粒子只能影响半径  $h$  以内的其他粒子，所以计算一个粒子和其余的其他所有粒子是效率极其低的。因此，我们会把空间划分成边长为  $h$  的小格子。模拟时，先会确定粒子在哪个格子(x,y,z)里，以  $z+y*\text{dimZ}+x*\text{dimY}*\text{dimZ}$  作为该格子的 hash，根据 hash 对粒子排序，然后在  $dStart[hash]$ ,  $dEnd[hash]$ 中存放起止的粒子索引。这样，我们就知道每个格子中有哪些粒子了。计算加速度时，只需要对该粒子所在的格子周围的 27 个格子算一遍就可以了，大大提高了效率。

模拟步骤完成后，如果要重建表面，可以用 marching cubes 算法。它的大概思想是在一个标量场中划分格子，每个格子的八个顶点上采样，根据该处的场值和设定的临界值比较，确定这顶点在等值面内还是等值面外。之后的处理就比较繁琐了，我没怎么仔细看，直接用了个开源实现，本身代码量不多（参考 3）。在代码中可以设定划分的分辨率，但由于这是个 CPU 实现，稍稍提高分辨率会让速度慢很多。

那么这个标量场怎么建立呢？本来我是用密度场作这个标量场的，但实验结果是重建出来的表面非常不平整，如果提高分辨率，表面能看到一个一个的球，和直接用球渲染粒子的效果差不多。后来，我查了些资料，换用 metaball 的方法建立标量场。这里面每个粒子能影响的半径  $r$  要比光滑核半径  $h$  大一些，要不然表面会坑坑洼洼的。Metaball 的介绍可见参考 4。

另外，我以前没接触过 GPU 编程，参考 5、6 中的代码以及 7 的 Nvidia CUDA Toolkit 官方文档对我有不小的帮助。

## 四、项目结构

param.h, param.cpp

在粒子系统中会用到的各种参数。

particlesystem.h, particlesystem.cpp

粒子系统类，管理相关的存储空间，提供各种接口。

particlesystem.cuh, particlesystem.cu

模拟用到的 GPU 函数与算法。

shader.h, shader.cpp

GLSL 的 vertex shader 和 fragment shader，在 OpenGL 中使用。

rendergl.h, rendergl.cpp

用 OpenGL 把粒子渲染为小球 (Point Sprite)。

timer.h, timer.cpp

简单的计时程序, 使用了 windows 的 `getTickCount()` 计时, 计算 FPS。

ofxMarchingCubes.h, ofxMarchingCubes.cpp, mcTables.h

引用的 marching cubes 算法。

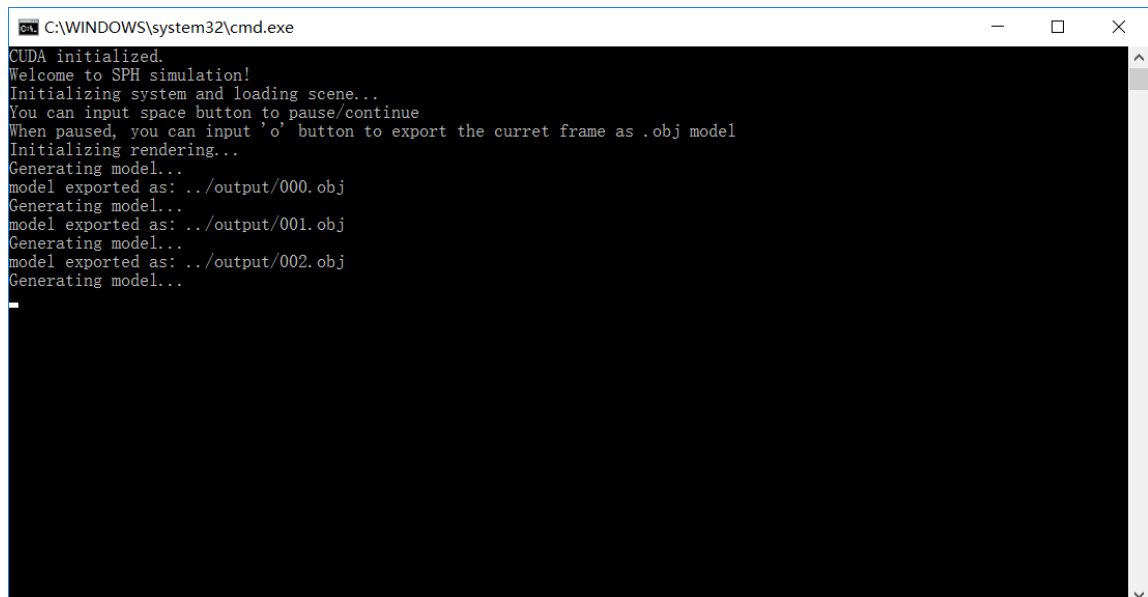
main.cpp

入口。

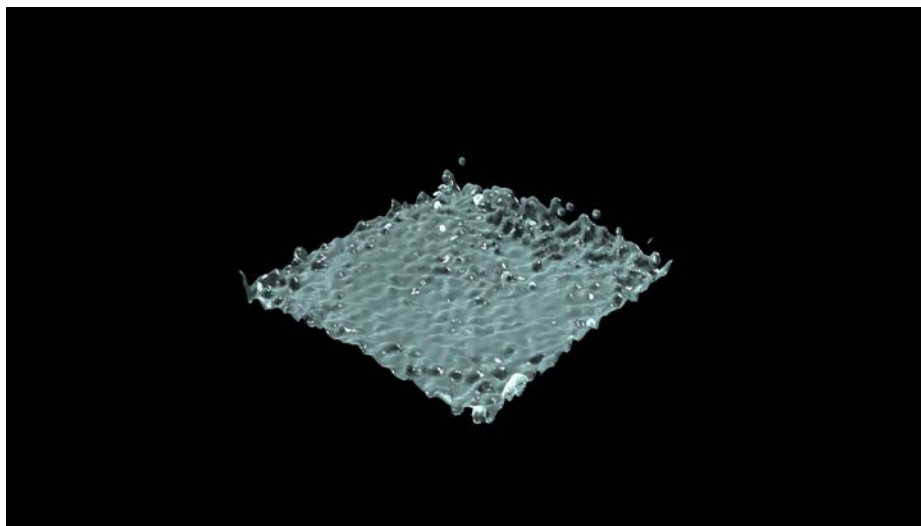
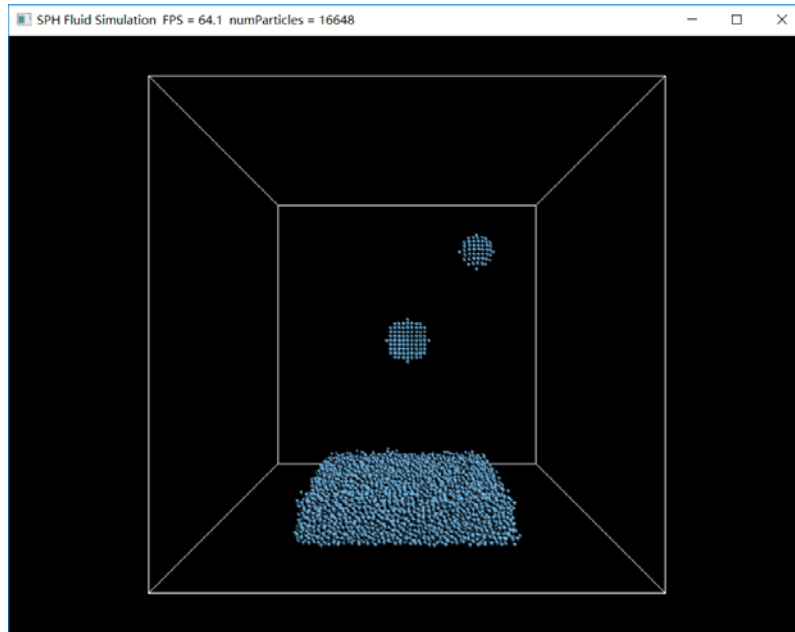
实现上尽量遵从 OOP 原则, 各部分还是有一定的独立复用性的。代码可能稍稍有些乱, 没太顾得上整理。

## 五、运行结果展示

在 results 文件夹中有实时渲染截图、导出的 obj、用 Maya 渲染的结果等。



```
C:\WINDOWS\system32\cmd.exe
CUDA initialized.
Welcome to SPH simulation!
Initializing system and loading scene...
You can input space button to pause/continue
When paused, you can input 'o' button to export the current frame as .obj model
Initializing rendering...
Generating model...
model exported as: ../output/000.obj
Generating model...
model exported as: ../output/001.obj
Generating model...
model exported as: ../output/002.obj
Generating model...
```



## 参考

1. <https://thecodeway.com/blog/?p=83>
2. <https://developer.nvidia.com/cuda-downloads>
3. <https://github.com/larsberg/ofxMarchingCubes>
4. <http://www.geisswerks.com/ryan/BLOBS/blobs.html#2>
5. <https://github.com/finallyjustice/sphfluid>
6. 官方的 CUDA samples 中的 particles 项目
7. <http://docs.nvidia.com/cuda/index.html#axzz4m2sxbXj>