粒子系统（CUDA + SFML 3）

一个使用 **CUDA 进行粒子更新/邻域构建**、使用 **SFML 3** 做可视化的示例项目。项目目标是：在 GPU 上实时更新粒子位置，构建最近邻列表并做简单的弹性碰撞响应。

目录结构

```
├── include/                # 头文件（.cuh/.hpp/.h）
│   ├── particle_gpu.cuh     # 粒子 GPU 侧接口 & 结构体声明
│   ├── particle_GPU.cuh     # （若存在）与 GPU 计算相关的内核声明
│   ├── ParticleSystem.h     # 粒子系统封装（Host 侧）
│   └── ...
├── src/                    # 源码（.cpp/.cu）
│   ├── main.cpp             # 程序入口（SFML 渲染循环）
│   ├── particle_gpu.cu      # updateParticles / build_neighbors_for_idx /
resolveCollisions 实现
│   └── ...
├── CMakeLists.txt          # CMake 构建脚本
└── build/                  # 构建输出目录（由 CMake 生成）
```

> **命名建议**：避免既有 `particle_GPU.cuh` 又有 `particle_gpu.cuh` 的大小写混用，容易引起重复包含或歧义。建议统一命名。

依赖项

- **CUDA 13.0**（nvcc 13.x；示例以 `/usr/local/cuda-13.0` 为例）
- **NVIDIA 驱动**（支持 Ada 架构，RTX 50 系列）
- **SFML 3.x**（已安装到 `/usr/local`，例如：`/usr/local/lib/cmake/SFML/SFMLConfig.cmake`）
- CMake ≥ 3.21（建议 ≥ 3.24）

快速开始

1) 配置 & 生成

```
cd ~/VScode/liziyundong
rm -rf build && mkdir build && cd build

cmake ..
 -DCMAKE_BUILD_TYPE=Release
 -DCMAKE_CUDA_COMPILER=/usr/local/cuda-13.0/bin/nvcc
 -DCUDAToolkit_ROOT=/usr/local/cuda-13.0
 -DCMAKE_CUDA_ARCHITECTURES=120
```

```
-DSFML_DIR=/usr/local/lib/cmake/SFML
-DSFML_STATIC_LIBRARIES=TRUE
```

2) 编译

```
cmake --build . -j$(nproc)
```

3) 运行

```
./MyApp
```

关键实现思路

时间步管线（单帧）

1. **更新动力学**（`updateParticles<<<>>>`）：根据速度积分位置 + 边界反弹；只做本粒子的状态更新。
2. **构建邻居**（Host 循环调用 `build_neighbors_for_idx`）：
3. 启动 `compute_distance<<<>>>`，为目标粒子 idx 计算与所有粒子 j 的距离平方 `d2`，写入：
   - `partticles_squre_distance[j]`（double）
   - `partticles_distance_idx[j]`（`Particle*`，指向 j 所在的设备地址）
4. **注意**：`j == idx` 的槽位写为 `INF / nullptr`，避免把自己作为邻居。
5. Host 端用 **Thrust**：`thrust::sort_by_key(double*, ..., Particle***)` 按 `d2` 升序排序。
6. **碰撞响应**（`resolveCollisions<<<>>>`）：每个粒子只遍历前 **K** 个最近邻，计算法向相对速度并做对称弹性冲量，使用 `atomicAdd` 原子更新双方速度，附带少量位置纠正以减少穿透。

邻居存储策略

- 本项目按你的要求：**邻居存储为设备指针 `Particle*`**（不是索引）。
- 这些指针**只能在 GPU 侧使用**；不要 `cudaMemcpy` 回主机后在 CPU 解引用。

可配置参数

- **`NN_CAP`**：邻居数组容量（必须 ≥ 最大粒子数 `count`），建议在头文件里：

  ```
  #ifndef NN_CAP
  #define NN_CAP 2048
  #endif
  ```

- **`K`**（参与碰撞的最近邻个数）：调用 `resolveCollisions<<<>>>` 时传入，例如 `K=16`。
- **`restitution`**（恢复系数）：例如 `0.2f`，越大越弹。

  若 `count > NN_CAP`：- 在 `compute_distance` 中跳过 `j >= NN_CAP`；- 在排序处使用 `n = min(count, NN_CAP)`；- 在 `resolveCollisions` 处限制 `K <= n-1`。

常见问题（FAQ）

1) 运行时报错：`thrust::system_error: D->D: cudaErrorInvalidValue`

原因：排序区间越界。`partticles_squre_distance[]` 和 `partticles_distance_idx[]` 的容量小于 `count`。 **修复**：- 提高 `NN_CAP`，保证 `NN_CAP >= count`； - 或在排序处改为 `n = min(count, NN_CAP)` 并在 kernel 中跳过 `j >= NN_CAP`。

2) 编译时报：`too few arguments in function call` / `identifier ... undefined`

原因：`compute_distance` 与 `compute_distance_for_idx` 版本混用。 **修复**：只保留一种签名，并统一声明/定义/调用。

3) `SFML_FOUND to FALSE` / `Requested SFML configuration (Shared) was not found`

原因：找到了 `SFMLConfig.cmake`，但组件或构建类型（动态/静态）不匹配。 **修复**：- 确保 `find_package(SFML 3 REQUIRED COMPONENTS Graphics Window System)`； - 配置时传 `-DSFML_DIR=/usr/local/lib/cmake/SFML`； - 静态/动态按安装方式一致：`-DSFML_STATIC_LIBRARIES=TRUE` 或去掉该项尝试动态链接。

4) `CUDA_ARCHITECTURES is set to "native", but no NVIDIA GPU was detected`

原因：机器/容器中未检测到 GPU，或 `native` 在你的 CMake 版本不可用。 **修复**：显式指定：`-DCMAKE_CUDA_ARCHITECTURES=120`（RTX 50 系列）。

5) `ptxas fatal : Value 'sm_52' is not defined for option 'gpu-name'`

原因：CUDA 工具链/目标架构不匹配。 **修复**：确保使用 CUDA 13.x，并传 `-DCMAKE_CUDA_ARCHITECTURES=120`。

性能与调优

- **K 值**：一般 `K=8~32` 足够；K 越大越耗时。
- **分批构建邻居**：大规模粒子时每帧只更新一部分粒子邻居（滚动覆盖），降低波动。
- **SoA（结构分离）**：后续可把 `x[], y[], vx[], vy[]` 拆分成独立数组，提升访存效率。
- **空间栅格/哈希**：若 `count` 很大，改用均匀网格做邻域裁剪，把复杂度从近似 `O(N^2)` 降到近似 `O(N)`。

GPU 使用率监控

- 快速查看：`nvidia-smi` 或实时 `watch -n 1 nvidia-smi`
- 交互界面：`sudo apt install nvtop -y && nvtop`
- 记录日志：

```
nvidia-smi --query-
gpu=timestamp,utilization.gpu,utilization.memory,temperature.gpu,memory.used,memory.total
     --format=csv -l 1 > gpu_log.csv
```

CMake 要点（摘录）

- 需要：`project(... LANGUAGES CXX CUDA)`
- C++/CUDA 版本：`set(CMAKE_CXX_STANDARD 17)`、`set(CMAKE_CUDA_STANDARD 17)`
- CUDA：`find_package(CUDAToolkit REQUIRED)`
- SFML 3：`find_package(SFML 3 REQUIRED COMPONENTS Graphics Window System)`
- 目标：

```
add_executable(MyApp ${CPP_SOURCES} ${CU_SOURCES})
target_include_directories(MyApp PRIVATE ${CMAKE_SOURCE_DIR}/include)
target_link_libraries(MyApp PRIVATE SFML::Graphics SFML::Window SFML::System
CUDA::cudart)
```

许可证

本项目仅用于学习与研究，许可证按需自定（MIT/Apache-2.0/GPL-3.0 等）。

致谢

感谢社区与相关开源项目。若你在使用中遇到问题，欢迎提交 issue 或直接联系维护者。