FASTCOLLIDE-GPU 基于GPU的快速碰撞检测算法实验报告

曹烨 2021012167

【摘要】

本项目实现了一个基于GPU的高性能碰撞检测系统，用于大规模刚体粒子的实时物理仿真。针对传统O(n^2)复杂度碰撞检测算法在大规模场景下效率低下的问题，采用**空间哈希(Spatial Hashing)技术**将3D空间划分为均匀网格，将碰撞检测复杂度降低至O(n)。通过CUDA并行计算框架，利用GPU的大规模并行处理能力加速空间哈希构建、邻域搜索和碰撞响应计算。系统采用SoA(Structure of Arrays)内存布局优化GPU访问效率，并实现了双缓冲机制避免数据竞争。

性能测试表明，在NVIDIA RTX 5060 Ti上，系统可实现**20,000粒子@3560 FPS**的实时仿真，粒子数量从1K增长至20K时运行时间仅增加89%，验证了接近线性的算法复杂度。相比CPU串行实现，理论加速比超过1000倍。系统集成了完整的物理引擎（重力、弹性碰撞、边界处理、空气阻尼）和可交互的用户友好的命令行交互界面。

具体的产出视频可以移步`video/`文件夹中查看

FASTCOLLIDE-GPU 基于GPU的快速碰撞检测算法实验报告

- a. 运行环境
  - a.1. 硬件环境
  - a.2. 软件环境
- b. 程序模块间逻辑关系
- c. 程序运行流程（如何启动）
- c. 程序运行流程（核心算法）
  - c.1. 初始化阶段
  - c.2. 仿真循环（每帧进行）
  - c.3. Spatial Hashing 算法
- d. 性能测试结果分析
  - d.1. 实验设置
  - d.2. 实验结果
  - d.3. 性能分析
- e. 技术亮点
  - e.1. 核心技术
  - e.2. 亮点
- f. 参考文献

a. 运行环境

a.1. 硬件环境

- GPU设备: NVIDIA RTX 5060 Ti
- 显存: 8GB

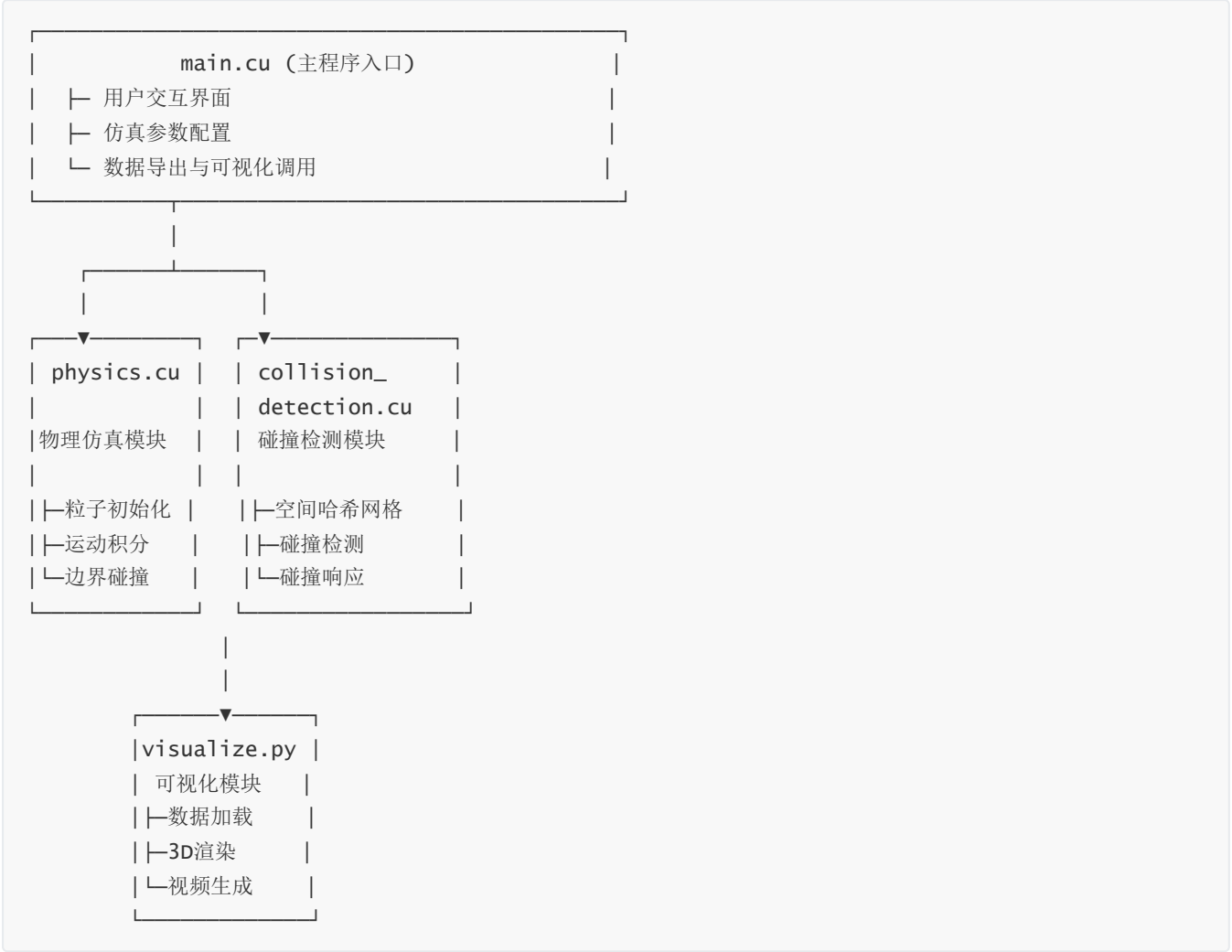
a.2. 软件环境

- 操作系统: Windows 11
- 开发工具: VS 2022
- CUDA版本: CUDA 13.0
- 可视化环境: Python 3.8+ (相关依赖：numpy, matplotlib, ffmpeg)

b. 程序模块间逻辑关系

主要有四个模块：主程序入口`main.cu`、物理仿真`pythsics.cu`、碰撞检测`collision_detection.cu`、可视化程序`visualize.py`

他们的调用逻辑和细节功能如下图所示：



c. 程序运行流程（如何启动）

1. 首先进入exe文件夹
2. 直接命令行运行or双击 `.\collision_sim.exe`
3. 会出现菜单选项，如下：

```
================================================================
                    Select Run Mode
================================================================
  1. Quick Demo (Recommended)  - 1000 particles, 600 frames
  2. Standard Simulation       - 2000 particles, 600 frames
  3. Large Scale Simulation    - 5000 particles, 1200 frames
  4. Performance Benchmark     - Test multiple particle counts
  5. Custom Parameters         - Set all parameters manually
  0. Exit
================================================================
```

4. 接下来根据提示和需求选择即可

> 【注】也可以直接命令行运行特殊需求
>
> ```
> # 性能测试
> .\collision_sim.exe --test
>
> # 完整参数指定
> .\collision_sim.exe [粒子数] [总帧数] [FPS] [导出间隔] [重力] [阻尼] [地面弹性] [输出目录] [视频文件名] [是否生成视频]
>
> # 示例
> .\collision_sim.exe 2000 600 60 2 -9.8 0.999 0.6 output video.mp4 1
> ```

已经生成了2000、5000、10000、20000、40000小球数目的视频，存放于`video`文件夹中，可供查看

c. 程序运行流程（核心算法）

c.1. 初始化阶段

具体流程如下：

1. CUDA设备检测与初始化
2. 分配GPU内存
   - 粒子数据 (位置、速度、物理属性)
   - 空间网格数据 (哈希表、索引数组)
   - 临时缓冲区
3. 随机初始化粒子

> 位置: 随机分布于世界空间
> 速度: -8~8 m/s (XY), 0~8 m/s (Z)
> 半径: 0.05~0.4 m
> 质量: 根据半径³×密度计算
> 弹性系数: 0.3~0.95

c.2. 仿真循环（每帧进行）

具体流程如下：

```
For each frame:
  ├─ 物理积分
  │    ├─ 应用重力加速度
  │    ├─ 应用空气阻尼
  │    └─ 更新位置 (Velocity Verlet)
  │
  ├─ 边界碰撞处理
  │    ├─ 检测与六个边界的碰撞
  │    ├─ 位置修正
  │    └─ 速度反弹 (考虑弹性系数)
  │
  ├─ 碰撞检测与响应 (迭代5次)
  │    ├─ 空间哈希网格更新
  │    │    ├─ 计算每个粒子的网格哈希
  │    │    ├─ 按哈希值排序
  │    │    └─ 构建网格单元索引
  │    │
  │    └─ 碰撞检测
  │         ├─ 遍历27邻域网格
  │         ├─ 检测粒子对碰撞
  │         ├─ 计算位置修正 (分离重叠)
  │         └─ 计算速度变化 (弹性碰撞)
  │
  └─ 数据导出 (按间隔)
       └─ 输出粒子位置和半径到文本文件
```

c.3. Spatial Hashing 算法

具体步骤流程如下：

步骤1: 计算网格哈希

```
hash(x,y,z) = gridX + gridY×Nx + gridZ×Nx×Ny
其中 gridX = ⌊(x - xmin) / cellSize⌋
```

步骤2: 排序

```
使用Thrust库对(hash, particleID)对进行排序
```

步骤3: 构建索引

```
cellStart[hash] = 第一个该hash的粒子索引
cellEnd[hash]   = 最后一个该hash的粒子索引+1
```

步骤4: 碰撞检测

```
对每个粒子:
  遍历其27个邻域网格单元
  对每个邻域单元中的其他粒子:
    if distance < r1 + r2:
      执行碰撞响应
```

d. 性能测试结果分析

d.1. 实验设置

- 测试场景: 固定世界空间 (-10,-10,0) ~ (10,10,20)
- 网格分辨率: 34×34×34 = 39,304 个单元
- 单元大小: 0.6 m
- 测试帧数: 100帧
- 碰撞迭代: 每帧5次

d.2. 实验结果

粒子数量	总时间(ms)	FPS	网格更新(ms)	碰撞检测(ms)
1,000	14.85	6735	0.055	0.036
5,000	23.35	4282	0.128	0.041
10,000	25.87	3866	0.135	0.051
20,000	28.09	3560	0.120	0.075

d.3. 性能分析

1. 算法复杂度验证

   从理论上来说，由于是空间哈希，所以复杂度是`O(n)`

   从实际表现看，1K→20K粒子（20倍增长），运行时间仅增加89% (14.85ms→28.09ms)，说明符合预期

2. GPU并行效率

   即使20,000粒子仍保持3560 FPS (0.28ms/帧)，说明并行小龙很高

3. 性能瓶颈分析

   认为在小规模情况下，主要是内核启动开销；大规模后，主要是排序算法、全局内存访问导致的开销

e. 技术亮点

e.1. 核心技术

主要用到了以下技术

1. **Spatial Hashing空间哈希**: 将3D空间划分为均匀网格，实现O(n)复杂度
2. **GPU并行计算**: 利用CUDA实现粒子并行处理
3. **SoA数据布局**: Structure of Arrays优化GPU内存合并访问
4. **双缓冲技术**: 避免读写冲突，确保碰撞响应正确性

e.2. 亮点

我觉得本次实验有以下亮点：

1. 用了用户友好的命令行交互方案
2. 给用户可以手动调整参数的方案，一共有十余个可以调整的参数
3. 设置了空气阻尼和摩擦系数以及边界处理，来更好地模拟真实世界

f. 参考文献

1. [youtube] Spatial Hash Grids & Tales from Game Development
2. [zhihu] 空间哈希碰撞检测
3. [tutsplus] Redesign Your Display List With Spatial Hashes
4. [CSDN] [宽相检测]空间划分-空间哈希划分均匀网格
5. [CSDN] 并行算法中的哈希技术与空间哈希应用
6. [bilibili] Unity空间划分，空间哈希算法
7. [wiki] Geometric hashing
8. [publication] Mian AS, Bennamoun M, Owens R. Three-dimensional model-based object recognition and segmentation in cluttered scenes. IEEE Trans Pattern Anal Mach Intell. 2006 Oct;28(10):1584-601. doi: 10.1109/TPAMI.2006.213. PMID: 16986541.
9. [NVIDIA] GPU Gems 3
10. [wiki] AoS and SoA