大作业中期文档

基于GPU的碰撞检测

马喆轩 2021080078 软12

CUDA空间划分的碰撞检测算法

1. 初始化

- 设 r_{max} 为场景中最大的球体的半径
- 设 $d = r_{max} * 1.5$ ，
- 我们可以将场景空间划分为若干个 $d * d * d$ 正方体格子 。因此我们可以保证每个球最多只会 8 个格子里出现。
- **Cell**是什么
  - 一个球的球心所在的格子称为 **Home Cell（简称 H Cell）**。
  - 而由于球的大小不同，其边缘可能会处于其他格子中，这些格子称为 **Phantom Cell （简称 P Cell）**。
  - 而一个小球只有一个 H Cell 和 **最多 7 个 P Cell** (因为前面提到的**"**保证每个球最多只会 8 个格子里出现**"**)
- 每个小球有几个属性：
  - **ID**：用于访问小球的额外属性，如碰撞检测中的包围体（bounding volume）。
  - **Control bits**：控制位用于存储物体的不同状态，例如物体所在的H cell类型等。
  - **Cell IDs**：用于表示物体的包围体所覆盖的单元格（cells）。
- 为了优化内存带宽和排序操作，`cell ID array` 和 `object ID array` 被存储为两个独立的数组：
  - **Cell ID array**：包含每个小球所在的cell ID。
  - **Object ID array**：包含小球的ID及其控制位 （control bits）。

2. 进行空间划分

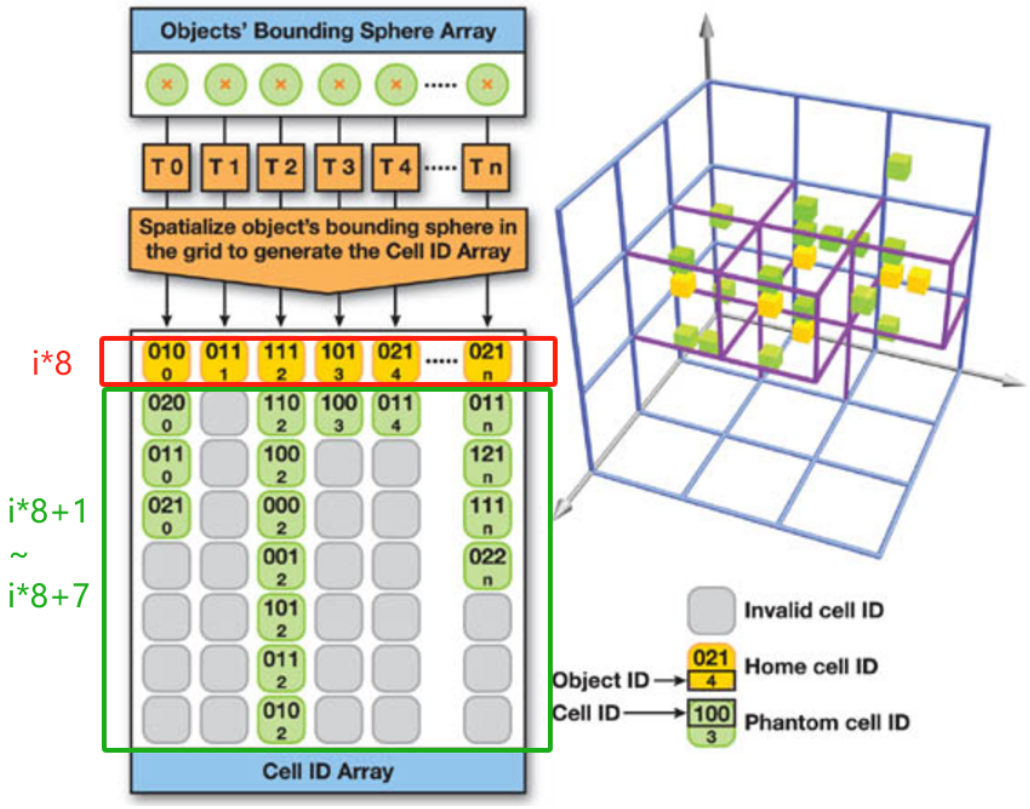
- 如何得到Home Cell Id ？

  每个物体的“home cell”即其包围体的中心所对应的单元格。这个单元格通过哈希化物体的质心坐标来计算。假设质心坐标为 `pos = (pos.x, pos.y, pos.z)`，单元格大小为 `d`，哈希的计算方式如下：

  ```
  GLuint hash = ((GLuint)(pos.x / d) << XSHIFT) |
                ((GLuint)(pos.y / d) << YSHIFT) |
                ((GLuint)(pos.z / d) << ZSHIFT);
  ```

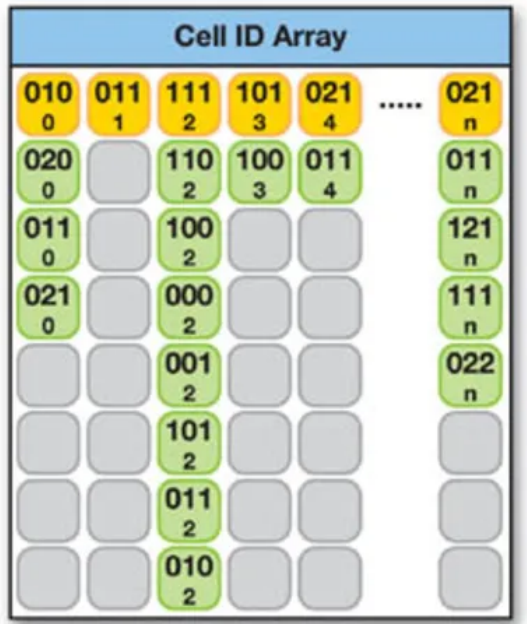
  其中，`XSHIFT`、`YSHIFT` 和 `ZSHIFT` 是预定义常量，控制每个坐标的位移。这个哈希值将成为物体的 H cell ID。

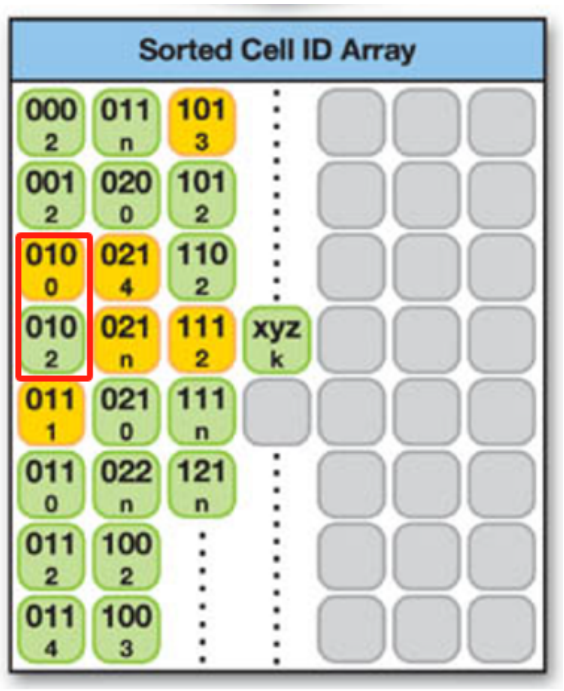
- 如何得到Phantom Cell Id ？
  - 前面提到一个小球可能不止在一个cell里，因此需要记录与其包围体重叠的其他单元格P Cell，都是与H Cell相邻的单元格
  - 如果有重叠的单元格，则将其加入 `Cell ID Array`。
  - 如果物体没有占用所有 $2^d - 1$个 P cells，则这些未使用的 P cell ID 被标记为无效（例如 `0xffffffff`）。
- 如何得到 Cell Id Array？
  - 假设有 i 个小球
  - 下标 $i * 8$ 就是存储 Home Cell ID，而其余位置就是存储 Phantom Cell Id；
  - 下标 $i * 8$ ~ $i * 8 + 7$ 就是第 i 个小球所占的位置
  - 可以通过并行，每个线程负责不同的小球，然后写入 `Cell Id Array` 即可



3. 基数排序 Radix Sort

- 对 `Cell Id Array` 进行基数排序
- 基数排序是一种稳定的排序，使用这个特性按Cell ID大小进行排序，因此若拥有相同的Cell ID，永远可以保证的是Home Cell 一定在 Phantom Cell 前面 （因为构造时就保证了 Home Cell ID必定处于 $i*8$ 的位置）
- 左边是未排序的，右图是排序的结果



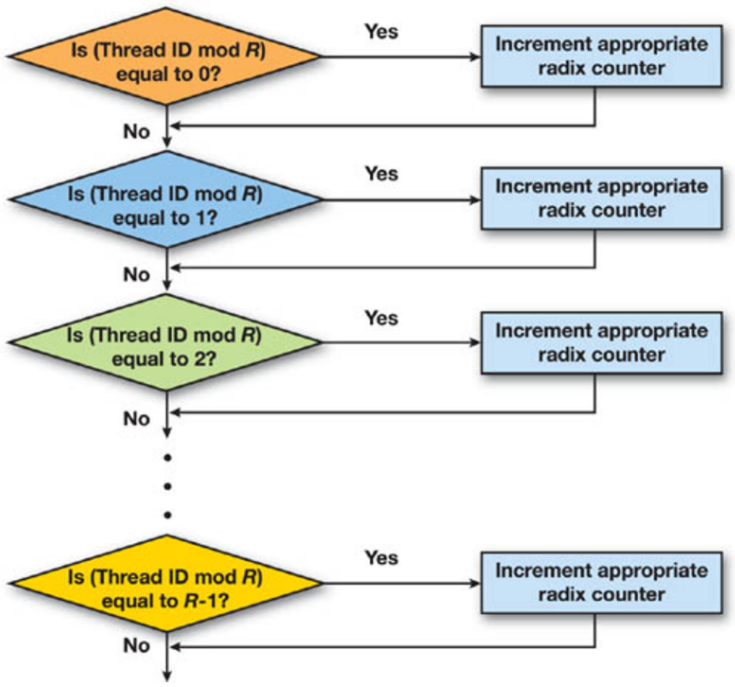


- 同样地，基数排序也是可以使用并行来计算。
  - Cell ID 为32位，我们可以依次取每8位（从最低位开始）作为一个pass来进行排序（这是因为基数排序的特性）
  - **Phase 1: Setup and Tabulation**
    - 引用原文 The resulting array of *radix counters* is then processed to convert each value into offsets from the beginning of the array. In other words, each value of the table is replaced by the sum of all its preceding values; this operation is called a *prefix sum*.
    - 每8位有一组计数器`radix_counter`，大小为256（因为 $2^8 = 256$）。`radix_counter[0] ~ redix_counter[255]` 分别记录了相应的计数，例如`00000010`，则 `radix_counter[2]++`
    - 统计radix_counter完成后，我们需要计算其前缀和 *prefix sum*。

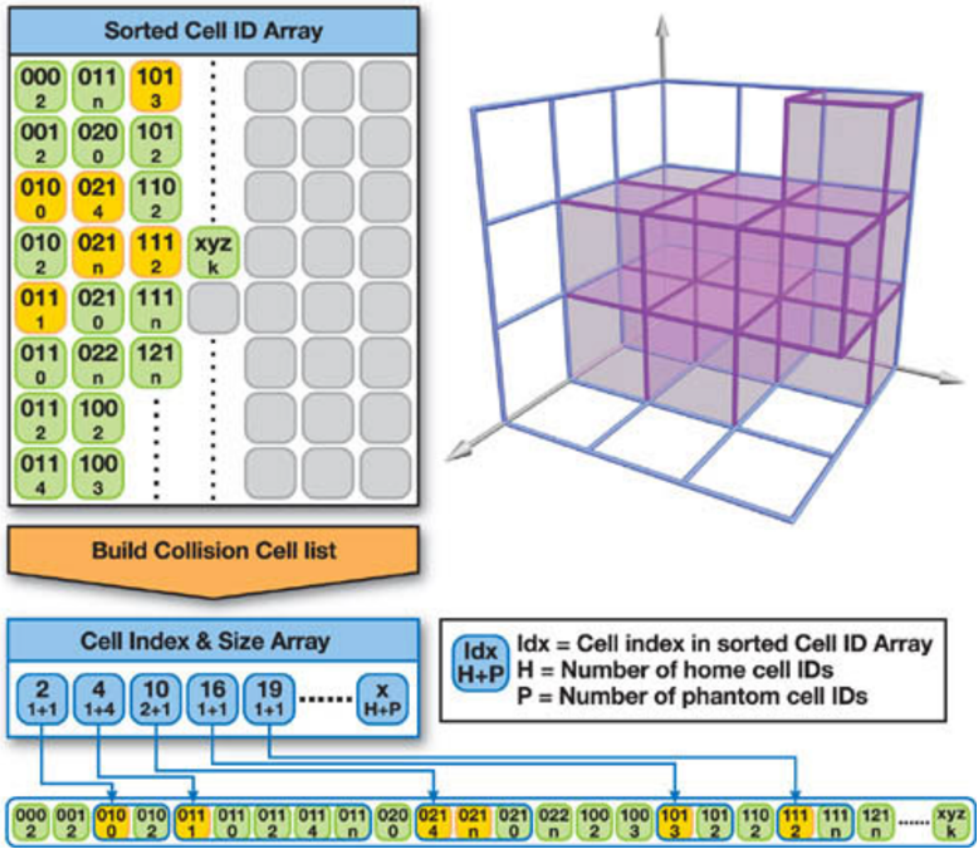
```
# 计算 prefix sum 之前的 radix_counter
index (i):     0  1  2  3  4  5  6  7  ...  255
radix_counter[i]: 0  2  1  4  0  1  0  0  ...  0
```

```
# 计算 prefix sum 完成的 radix_counter
index (i):     0  1  2  3  4  5  6  7  ...  255
radix_counter[i]: 0  0  2  3  7  7  8  8  ...  8
```

- 计算前缀和的意义是什么？
  - **它告诉我们每个元素在最终排序数组中的确切位置。**
- 这个步骤可以使用并行，对应的线程统计自己负责的计数即可



- **Phase 3: Reordering**
  - 将 `Cell Id Array` 按照刚刚所得到的前缀和 `radix_counter`，将 `Cell Id Array` 的元素放至排序后的数组中它应该所在的位置。
  - 获取 `Cell Index & Size Array` 这里简单记为 `Index Array`
    - 如下图，只需遍历一遍基数排序后的 `Cell Id Array` 即可得到
    - 这个 `Index Array` 是记录了 “连续的 Cell Id 开头的子数组” 的索引，其中包括 Cell Id
      - H 即指的是这个Cell 中有多少个 `home cell`
      - P 即指的是这个Cell 中有多少个 `phantom cell`



4. 碰撞检测

我们可以利用 `Index Array` 中的信息，对 `Cell Id Array` 中的 “子数组” 进行碰撞检测。

- 对于子数组中的 cell 进行两两碰撞检测
  - 会优先 `home cell` 与 `home cell` 之间的碰撞检测 （因为排序结果是稳定的，前面定义了home cell 的优先级较高）
  - 再来 `home cell` 与 `phantom cell` 之间的碰撞检测，只有当 phantom cell 的小球的 home id 大于 home cell 的小球的 home id 才会进行碰撞检测。
  - 保证同一对只检测一次，避免重复计算，这里可以使用多线程来处理

难点

1. 环境配置
2. 算法的并行部分

参考资料

官方资料：https://developer.nvidia.com/gpugems/gpugems3/part-v-physics-simulation/chapter-32-broad-phase-collision-detection-cuda