基于CUDA的碰撞检测报告

软73 沈冠霖 2017013569

1.运行环境与方法

1.1 运行环境

- 系统：Win10
- 开发环境：VS2017
- OpenGL版本：4.6.0 NVIDIA 441.66
- OGLU工具库版本：1.2.2.0 Microsoft Corporation
- CUDA版本：10.2
- 使用GPU device 0: GeForce GTX 950M
- SM的数量：5
- 每个线程块的共享内存大小：48 KB
- 每个线程块的最大线程数：1024
- 每个EM的最大线程数：2048
- 每个EM的最大线程束数：64

1.2 运行方法

打开bin中的可执行文件，选择4种模式，能看到动画渲染效果，并且运行几分钟后能看到一万次碰撞检测平均时间。在动画渲染的时候可以用键盘WASD/上下左右还有滑动鼠标来更新视角。

2.实现原理

2.1 碰撞处理

我们需要做两方面的碰撞处理，一是球和边界的碰撞处理，二是球和球的碰撞处理。

前者较为简单，因为边界都是横平竖直的，我们直接把对应的速度取反，其他不变就行。

对于后者，我们假设球是均匀的，其质量为$\frac{4}{3}\pi\rho r^3$。

我们还假设所有的碰撞都是对心完全弹性碰撞。我们先把两个球的速度划分为切向速度（沿半径连线的）和法向速度。根据动量守恒定律，碰撞后法向速度不变，切向速度满足下列公式：

$$v_1' = \frac{v_1(m_1 - m_2) + 2m_2 v_2}{m_1 + m_2}$$

$$v_2' = \frac{v_2(m_2 - m_1) + 2m_1 v_1}{m_1 + m_2}$$

即可完成碰撞处理。

2.2 基于空间划分的CUDA碰撞检测

我们的碰撞检测方法和cuda官网教程https://developer.nvidia.com/gpugems/gpugems3/part-v-physics-simulation/chapter-32-broad-phase-collision-detection-cuda一致。

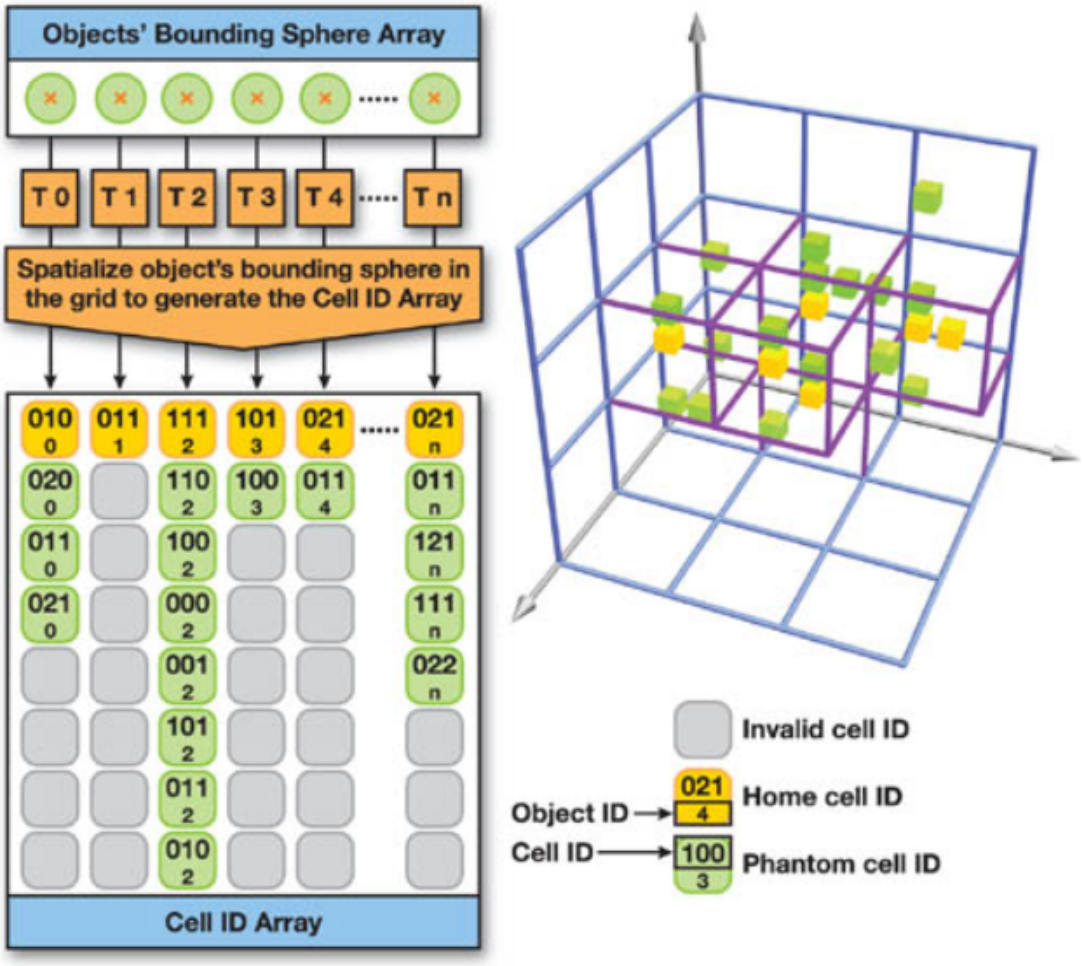
对于球的运动和球与边界的碰撞，我们直接让每个线程处理不同的球的运动和边界碰撞即可实现。

对于球之间的碰撞检测，我们方法如下：

2.2.1 空间划分

我们将空间划分为若干正方体格子，为了保证后续检测的效率，我们定义每个格子的边长为球最大半径的1.5倍。这样每个球就只可能出现在2 * 2 * 2 = 8个格子里。碰撞检测算法的第一步就是求出每个球所在的格子，格子被分为两类，一类是主格子（home cell），也就是球心所在的格子，另一类是从格子（phantom cell），也就是球所在的其他格子。因为格子边长足够大，所以每个球最多在8个格子里。

Figure 32-9 illustrates the result of cell ID array construction.



这一步进行并行也不难，只需要让不同的线程计算不同的球所在的格子，然后存到一个数组里就行。第i个球存到数组第8i到第8i+7个位置。最终得到8N个格子。

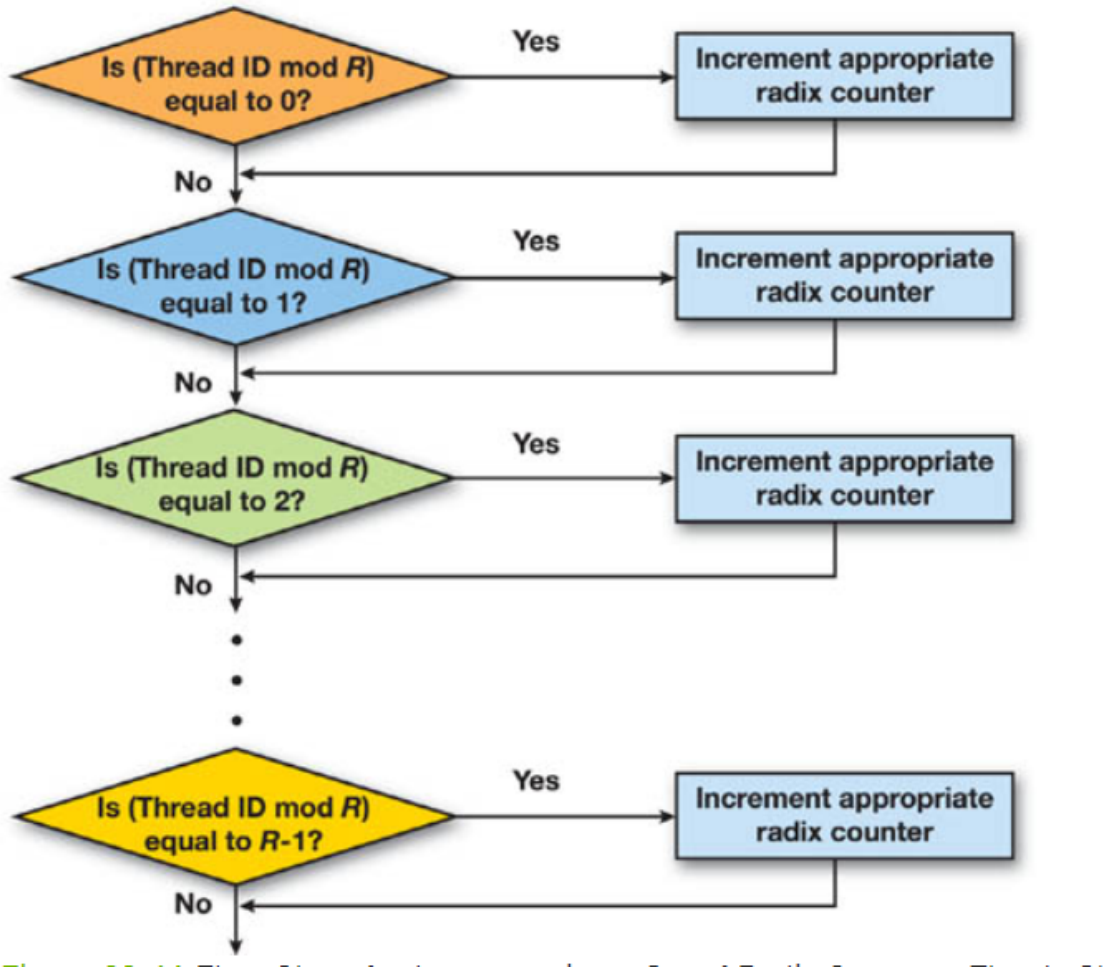
2.2.2 基数排序

之后我们需要对格子数组进行排序，保证有如下优先级：格子编号小的在前面，如果格子编号一样，home的格子在phantom的格子前面。

因为我们的格子数目有限（一般是每一维就几十个，不超过256），因此可以把格子编号+home/phantom信息用一个32位整数表示出来。这时候使用时间复杂度为O（N）的基数排序就很好。我们使用L=8的基数排序，排4趟。

首先，我们需要计算对应位置结果为0-255的元素个数，使得radix_sums[i]变为对应位置结果为i的个数。使用cuda计算需要使用如下的代码来避免读-写冲突和写-读冲突。

```
for (int j = 0; j < blockDim.x; j++)
{
    if (threadIdx.x % blockDim.x == j)
    {
        int current_radix_num = (cells[i] >> shift) & (num_indices - 1);
        radix_sums[current_radix_num] ++;
    }
}
```



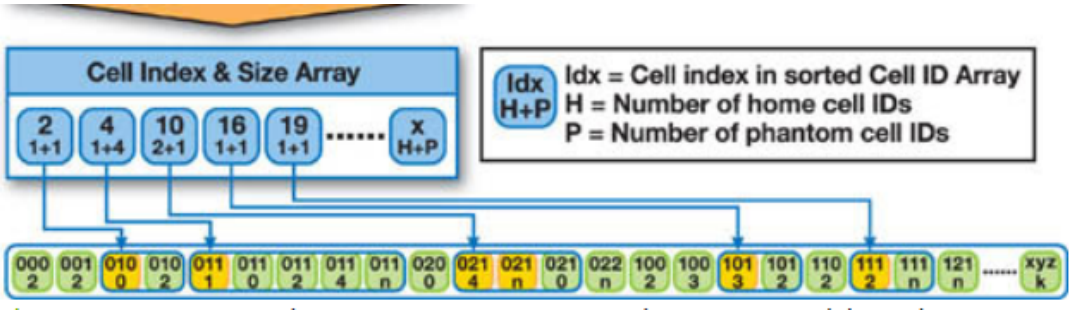
其次，我们需要求前缀和，让radix_sums[i]为结果为0到i-1的元素个数之和。我参考https://www.cnblogs.com/biglucky/p/4283473.html的介绍实现了一个reduction-reverse两阶段的前缀和求和方法。

最后，我们需要给每个元素分配到对应的位置。因为要保证稳定排序，我们这里使用串行方法执行。

排序之后，我们还串行遍历了整个cell数组，得到了数组索引，也就是数组从哪里到哪里属于一个格子，哪些是home，哪些是phantom。

2.2.3 碰撞检测

经过基数排序，我们就得到了如下的一个有序数组：



这个数组按照先是格子编号，后是home优先phantom的顺序排序，而且我们知道每个格子对应的元素从哪开始到哪结束，几个home几个phantom，这样就可以方便的进行并行碰撞检测了。我们给每个线程分配几个格子，进行碰撞检测和处理，这样不会产生冲突，只要我们保证每个碰撞只被处理一次。

为了保证碰撞只被处理一次，我们要进行冲突检测。首先，我们只处理home和home，还有home和phantom的碰撞。其次，对于home和phantom的碰撞，假设home的物体为A，phantom的物体为B，A的home编号为hA，B的home编号为hB，当仅当hA<hB我们才进行碰撞检测。使用这种方法可以有效避免冲突。

3.实验结果

3.1 正确性

我们使用OpenGL渲染得到动画，分析结果，我们的碰撞检测结果基本正确，碰撞都能被处理到。但是碰撞处理结果有一些瑕疵，就是因为我们只更新速度没有更新位置，这样导致如果有的球速度过大，嵌入另一个球太多了，碰撞处理后他们没法分开，会有一些小问题。

//渲染的时候有时候物体会陷进去，小问题

3.2 效率比较

为了比较，我们还实现了遍历碰撞检测的串行，并行形式，在不同的设置下进行实验。我们的结果都是碰撞检测10000次取平均值的，结果如下：

	O(n^2)的遍历算法（串行）	**O(n^2)的遍历算法（并行）**	**空间划分算法（串行）**	**空间划分算法（并行）**
12 * 12 * 12个格子，8个球	0.0013ms	2.1655ms	0.3705ms	2.5561ms
24 * 24 * 24个格子，64个球	0.0281ms	2.2212ms	2.2734ms	4.2536ms
48 * 48 * 48个格子，256个球	0.9157ms	2.5086ms	22.9631ms	18.4446ms

分析结果，首先，因为我们的球比较稀疏，因此格子数量带来的影响比较大（空间划分算法时间效率为O(num_cell)，遍历算法为O(num_ball^2)），因此空间划分算法效率不如遍历。其次，因为我们数据规模没有那么大，cuda优化不够明显，再加上需要cpu-gpu的内存交换占用了大量时间，因此cuda并没有实际加速。但是随着格子和球数量的增加，使用cuda加速的时间增加并没有串行那么多，可以预见，当数据规模足够大，cuda效率会超越串行。

4.总结

4.1 总结

这次作业，让我基本了解了空间划分碰撞检测的流程，cuda的基本使用方法，进一步熟悉了OpenGL渲染方法。

但是还是有很多不足，因为我对cuda不了解，对于基数排序等的一些优化做的不是特别充分，空间划分方法也比较简单，碰撞处理也有可以改进的空间。希望之后能有机会进一步改进。

4.2 参考文献

[1]CUDA的配置按照这个教程 https://blog.csdn.net/u013165921/article/details/77891913

[2]CUDA学习按照这个教程 https://zhuanlan.zhihu.com/p/34587739

[3]算法按照这个流程 https://developer.nvidia.com/gpugems/gpugems3/part-v-physics-simulation/chapter-32-broad-phase-collision-detection-cuda

[4]prefix sum算法参考了这里 https://www.cnblogs.com/biglucky/p/4283473.html

[5]代码实现参考了这个repo https://github.com/deeptoaster/cuda-collisions