PicComb GPU加速实验报告

刘柯 12021232

**实验环境：**

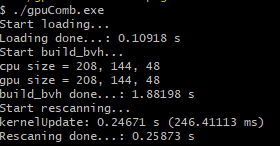
CPU:Intel(R) Core(TM) i7-8700 CPU @ 3.20GHz, 6核心12线程

GPU：NVIDIA GeForce RTX 2070 SUPER

**Baseline：**

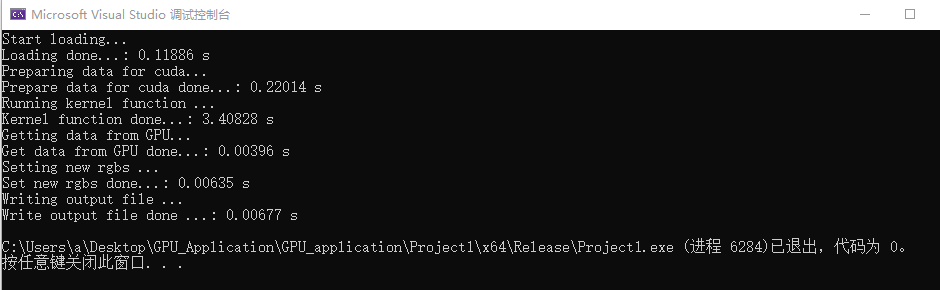
老师提供的CPU版本运行时间：626.56s

老师提供的GPU版本运行时间：约2.5s，各步骤耗时如下图：



**我的GPU版本程序：**

整体思路：从整体上来看，整个程序的作用就是对于1024\*1024个目标点，在1024\*1024个数据点中寻找到一个最近的数据点，如果该数据点和我们的目标点之间的距离少于预先设置好的阙值，则认为找到，并把数据点的rgb值复制到目标点。于是，我就写了一个非常naive的GPU算法：GPU的每一个线程计算一个目标点的最近数据点，在kernel函数中直接遍历1024\*1024个数据点，记录下最近那个数据点的index，最后在CPU中进行是否找到的判断和RGB的复制。在使用512线程的参数设置下，这个方法的效果如下：



总体时间大概为3.7s，其中主要的时间消耗在Kernel function的计算上。

**优化方法：**

为了提升速度，这里主要用了两个优化方法，shared memory和对目标点进行去重处理。

* Shared memory:

前面最naive的方法中，每一个线程在遍历数据点的时候都是回到global memory中去寻找，于是这里试图使用shared memory来加速。在每一个block中申请了大小为4\*512\*3的shared memory，因此512个线程每一个线程首先load 4个数据点坐标到shared memory中（这里的4是在shared memory最大容量允许下的最好值），经过线程\_\_syncthreads()后，每一个线程在shared memory中找到距自己的目标点最近的数据点，并和当前最近的数据点对比决定是否更新，所有线程都完成shared memory的扫描后，会循环这个更新shared memory，扫描shared memory的过程，直到扫描完全部1024\*1024个数据点。

* 对原数据进行去重处理：

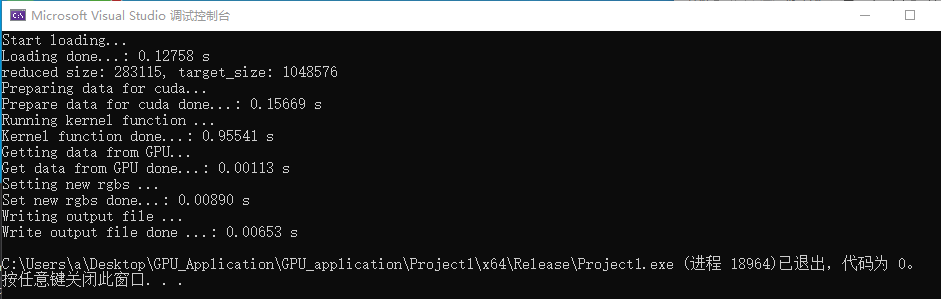
在对目标点进行观察后，发现目标点中存在大量的重复坐标，于是试图提取出所有唯一的xyz坐标，组成新的需要查询的xyz array，并记录下1024\*1024的目标点所对应的唯一的xyz坐标。仅将去重的后的xyz坐标array传入kernel，然后再通过之前计算好的对应关系可以重新计算出每一个目标点的最近点。

**实验结果：**

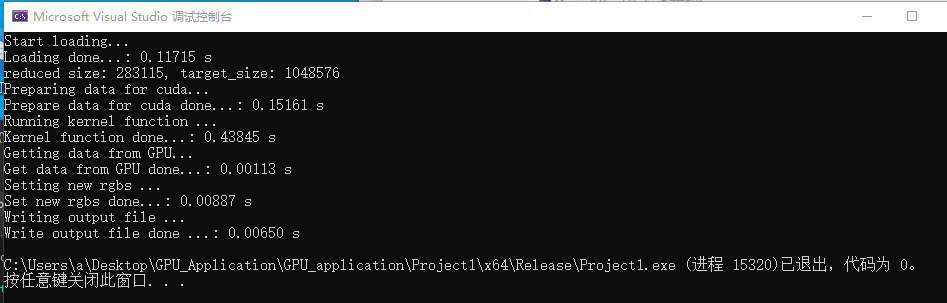
单单使用shared memory，结果如下图，总时间约为1.8s，主要加速的部分是kernel部分，计算效率是原来的2.2倍，



单单使用去重处理加速，结果如下图，总时间约为1.23s，可以看见，去重后的目标点只有283115个，是原来的27%，kernel函数运行时间变为原来的28%，基本上是按比例减小，同时cuda的数据准备阶段占用内存也变少，运行时间也有所减少，整体的运行时间变为原来的33%。



同时使用shared memory和去重加速，结果如下,总时间约为0.69s，可以看见这两个加速手段互不影响，最终的运行时间变为原来的18.6%，kernel运行的查找函数基本上和load数据，cuda准备数据两个部分的运行时间是一个量级。



四个算法跑出来的图如下：跟老师给的图片是基本一致的，在一些细节上会有一点点差别。

（1）老师的算法 （2）Naive GPU算法 （3）Shared memory优化



（4）去重操作处理 （5）Shared memory+去重

以下列出6种算法的时间：

|  |  |  |
| --- | --- | --- |
| 算法 | 运行时间（s） | 加速比 |
| CPU BVH(baseline, provided by teacher) | 626.56 | x1 |
| GPU BVH(provided by teacher) | 2.5 | x250.6 |
| Naive GPU | 3.7 | x169.3 |
| Shared Memory GPU | 1.8 | x348.1 |
| 去重GPU | 1.23 | x509.4 |
| Shared Memory+去重GPU | 0.69 | X908 |

**总结与反思：**

我并没有在GPU上建BVH树，但仍然把程序加速到了CPU算法的900倍，也是老师提供的GPU算法的4倍左右的速度。最后输出的结果基本上和老师给定的是一致的。稍微有一些区别的地方我认为是因为我的限制条件不够以及一些限制参数设置和老师的不同，比如老师的BVH CPU算法中有判断包围盒是否overlap，但这个限制条件我并没有用到，这一点应该会对我的结果造成影响。