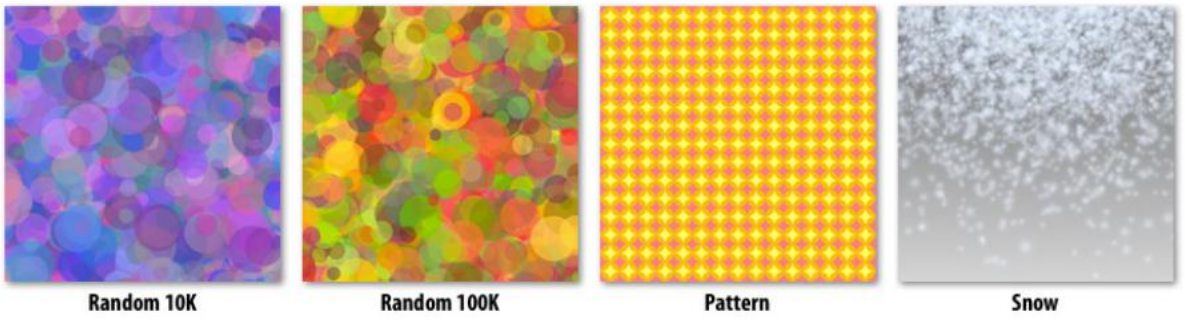
任务3：一个简单的CUDA渲染器

截止日期:11月8日星期三太平洋标准时间晚上11:59

总分100分



概述

在本作业中，您将在CUDA中编写一个绘制彩色圆圈的并行渲染器。虽然这个呈现器非常简单，但是并行化呈现器需要您设计和实现数据结构
可以有效地并行构造和操作。这是一个具有挑战性的任务，所以建议你早点开始。**说真的，建议您尽早开始。**好运！

环境 设置

1. 您将收集结果(i;e。在Amazon Web Services (AWS)上启用gpu的虚拟机上运行性能测试)。请按照cloud_readme中的说明操作。用于设置一台机器以运行分配。
2. 从课程Github下载作业启动代码，使用:

```
Git克隆https://github。com/stanford-cs149/asst3
```

CUDA C程序员指南PDF 版本或网页 version是学习如何在CUDA中编程的优秀参考。在网络上(只有Google!)和NVIDIA上有大量的CUDA教程和SDK示例 开发人员 网站。特别是，你可能会喜欢免费的Udacity课程Introduction 来 平行 编程 在 CUDA。

CUDA中的表G.1 C 编程 指南是一个方便的参考，每个线程块的最大CUDA线程数，线程块的大小，共享内存等NVIDIA T4 gpu，你将在这个任务中使用。NVIDIA T4 gpu支持CUDA 7.5计算能力。

对于c++问题(比如virtual关键字是什么意思)，c++ 超级 FAQ是一个很好的资源，它以一种详细而又容易理解的方式解释了事情(不像很多c++资源)，它是由c++的创造者Bjarne Stroustrup共同编写的!

警告

为节省资源，虚拟机在CPU活动时间< 2%后15分钟会自动关闭。

这意味着如果您不执行CPU密集型工作(如写入)，则VM将关闭
代码。

因此，我们建议您在本地开发代码，或者手动将代码复制到机器中，或者使用git连接将提交拉入虚拟机。使用git很好，因为您可以返回到以前版本的代码。

如果你还没有建立一个私有的git仓库，这里有一些资源可以帮助你开始。确保github的repo是私有的，以确保你没有破坏荣誉代码。

设置git的有用链接:

- 添加 一个 远程 存储库连接到您的私有仓库。
- 添加 ssh 设置SSH密钥。 我们建议这样做不需要密码和
  使用默认名称id_rsa。

拥有ssh密钥并知道如何连接到远程存储库之后，需要执行以下两项操作来设置环境。

1. 将您的私钥复制到服务器上。SSH文件夹(id_rsa)。ssh)文件
2. 在服务器和本地用以下代码创建一个名为config的文件。

```
主机github。com
    主机名github。com
    git用户
    IdentityFile ~ /。ssh / id_rsa
```

现在您应该能够从服务器和本地拉入和推送提交了!

部分 1： CUDA 热身 1： SAXPY （5 分）

为了获得编写CUDA程序的一些练习，您的热身任务是重新实现SAXPY函数
从作业1在CUDA。这部分作业的起始代码位于/saxpy目录中
分配存储库的。您可以通过调用make和来构建和运行这个时髦的CUDA程序 /saxpy目录下的。
/cudaSaxpy

请在SAXPY中的saxpyCuda函数中完成SAXPY的实现。铜。您将需要分配设备全局内存数组并复制主机输入数组的内容 在执行计算之前，将X、Y和结果存入CUDA设备内存。CUDA计算完成后，结果必须复制回主机内存。请参阅《程序员指南》(web版)3.2.2节中cudaMemcpy函数的定义，或者看看赋值启动器代码中指向的有用教程。

作为实现的一部分，在saxpyCuda中为CUDA内核调用添加计时器。添加完成后，程序应该为两次执行计时:

> 所提供的启动器代码包含计时器，用于测量将数据复制到GPU、运行内核和将数据复制回CPU的整个过程。
>
> 您还应该插入计时器，它只测量运行内核所花费的时间。(它们不应该包括cpu到gpu数据传输的时间，也不应该包括从
>
> GPU回到CPU。)

-
-

*当在后一种情况下添加计时代码时，您需要小心:默认情况下，CUDA内核在GPU上的执行与主应用程序线程在CPU上运行是异步的。*例如，如果你写这样的代码:

```
double startTime = CycleTimer::currentSeconds();saxpy_kernel<< blocks,
threadsPerBlock>>>(N, alpha, device_x, device_y, device_result);
double endTime = CycleTimer::currentSeconds();
```

您将测量一个似乎非常快的内核执行时间！(因为您只计时API调用本身的成本，而不是在GPU上实际执行结果计算的成本。

因此，您将希望调用来同步（）
内核调用等待GPU上所有CUDA工作的完成。这是对　　当GPU上所有先前的CUDA工作完成时，cudadevicesynize()返回。请注意，cudadevicesynsynchronize()在cudaMemcpy()之后没有必要确保内存传输到GPU完成，因为cudaMempy()在我们使用它的条件下是同步的。(对于那些希望了解更多的人，请参阅此 证明某事属实的证据

```
双启动时间=自行车计时器：：当前秒（）；saxpy_kernel<<<块，线程p块>>>（N，alpha，device_x
，device_y，设备_结果）；
cudaDeviceSynchronize();
双端时间=自行车计时器：：当前秒（）；
```

请注意，在您的测量中，包括到CPU的传输时间，在最终计时器之前不需要调用（）(在调用之后　　因为cudaMemcpy（）在复制完成后不会返回到调用线程。

问题1。与基于cpu的顺序实现相比，您观察到的性能如何
SAXPY(还记得作业1中程序5的SAXPY结果吗?)

问题2。比较并解释结果之间的差异
由两组计时器提供(只对内核执行进行计时vs。除了内核执行之外，还要对将数据移动到GPU和返回的整个过程进行计时)。观察到的带宽值是否与机器的不同部件可用的报告带宽大致一致？(你应该使用web来追踪NVIDIA T4 GPU的内存带宽。提示：　https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/tesla-t4/t4-tensor-co重新数据表-951643。pdf。AWS对内存总线的期望带宽为4gb /s，与16通道PCIe不匹配 3.0.有几个因素会阻止峰值带宽，包括CPU主板芯片组性能，以及用作传输源的主机CPU内存是否“固定”——后者允许GPU直接访问内存，而无需经过虚拟内存地址转换。如果你感兴趣，你可以在这里找到更多的信息:http://kth . infrastructure。com/courses/12406/pages/optimizing-host-device-data-communication-i-pinn ed-host-memory)

部分 2：梭达 热身运动 2：平行 前缀总和（10 分）

现在你熟悉CUDA程序的基本结构和布局，作为第二个练习你被要求想出并行实现函数find_重复，给定整数列表，返回所有指数的列表我[我]==[我+1]。

例如，给定数组{1,2,2,1,1,1,3,5,3,3}，程序应该输出该数组{1、3、4、8}。

排他前缀和

我们希望您通过首先实现并行独占前缀和操作来实现find_repast。

复杂的前缀和取一个数组A并产生一个新的数组输出，该输出在每个索引i上具有到但不包括A[i]的所有元素的和。例如，给定的数组{1、4、6、8、2}，排他前缀和输出输出={0、1、5、11、19}。

A=

下面的“类似c”代码是扫描的迭代版本。在之前的伪代码中，我们使用palan_for来表示潜在的并行循环。这是我们在类中讨论的相同的算法： http://cs149。斯坦福大学。edu/fall23/lecture/dataparallel/slide_17

```
Void exclusive_scan_iterative(int* start, int* end, int* output) {

     int N =端启动；
     memmove（输出、开始、N号（int））；

     //上扫描阶段
     For (int two_d = 1;two_d <= N/2;two_d * = 2) {
          因特two_dplus1 = 2*two_d；
          pacany_for（inti=0；i < N；i += two_dplus1）{
               output[i+two_dplus1-1] += output[i+two_d-1];
          }
     }

     output[N-1] = 0;

     //下行扫描阶段
     为（int two_d = N/2；two_d >= 1；two_d /= 2）{
          因特two_dplus1 = 2*two_d；
          pacany_for（inti=0；i < N；i += two_dplus1）{
               int t = output[i+two_d-1];
               输出[i+two_d-1] =输出[i+two_dplus1-1];
               output[i+two_dplus1-1] += t;
          }
     }
}
```

我们希望您使用这个算法来实现CUDA中并行前缀和的一个版本。您必须在扫描/扫描中实现排他性的扫描功能。cu .您的实现将包括主机代码和设备代码。该实现将需要多个CUDA内核启动（在上面的伪代码中，每个并行_for循环对应一个）。

注意:在起始代码中，上面的参考解扫描实现假设输入数组的长度(N)是2的幂。在cudaScan函数中，当在GPU上分配相应的缓冲区时，我们通过将输入数组长度四舍五入到下一个2的幂来解决这个问题。但是，代码只将N个元素从GPU缓冲区复制回CPU缓冲区。这个事实应该简化你的CUDA实现。

编译生成二进制cudaScan。命令行的用法如下：

```
用法：./cudaScan

方案选择:
  -m——测试<TYPE>在输入上运行指定的函数。有效的测试是：扫描，
查找重复序列（默认值：扫描）
  -i——input <NAME>对给定的输入类型运行测试。有效输入为：1个，
随机（默认值：随机）
  -n----------排列大小<INT>数组中的元素数
  -t-推力的使用推力库的实现
  -？ ——帮助这个消息
```

使用前缀和实现“查找重复”

编写完exclusive_scan之后，在中实现函数find_repeats scan/scan.cu .这除了需要一个或多个调用外，还需要编写更多的设备代码。您的代码应该将重复元素的列表写入提供的输出指针(在设备内存中)，然后返回输出列表的大小。

在调用独占_scan实现时，请记住的内容启动数组将被复制到输出数组中。此外，传递给排他性_scan的数组被假定在设备内存中。

评分:我们将测试你的代码在随机输入数组上的正确性和性能。

作为参考，下面提供了一个扫描分数表，显示了在K80 GPU上的简单CUDA实现的性能。来检查您的正确性和性能分数扫描和查找_重复操作的实现，运行扫描和 ./checker.pl ./checker.pl Find_分别重复。这样做将产生如下所示的参考表；你的分数是仅仅基于代码的性能。为了获得完整的信用，您的代码必须在所提供的参考解决方案的20%范围内执行。

```
--------------------------
扫描评分表：
--------------------------
------------------------------------------------------------------------
|元素计数|参考时间|学生时间|分数|
------------------------------------------------------------------------
| 1000000          | 0.766          | 0.143 (f)          | 0          |
| 10000000         | 8.876          | 0.165 (F)          | 0          |
| 20000000         | 17.537         | 0.157 (F)          | 0          |
| 40000000         | 34.754         | 0.139 (F)          | 0          |
------------------------------------------------------------------------
|                                   |总分：  | 0/5                    |
------------------------------------------------------------------------
```

这部分的作业主要是关于以数据并行的方式编写CUDA和思考问题，而不是关于性能调优代码。在这部分分配中获得完整的性能点不需要太多（或任何）性能调优，只需要算法伪代码到CUDA的直接端口。然而，有一个技巧:简单的scan实现可能会为伪代码中的并行循环的每次迭代启动N个CUDA线程，并在内核中使用条件执行来确定哪些线程真正需要工作。这样的解决方案将无法实现！（考虑向上扫描阶段的最后一个最外循环迭代，其中只有两个线程可以工作！）。一个完整的信用解决方案将只为最内部的并行循环的每次迭代启动一个CUDA线程。

测试线束：默认情况下，测试线束在一个每次都相同的伪随机生成的阵列上运行该程序正在运行，以帮助调试。您可以随机传递该参数以进行运行 `-i` 一个随机数组——我们在评分时也会这样做。我们鼓励您为您的项目提出替代输入来帮助您评估它。您还可以使用-n <size>选项来更改输入数组的长度。

这个论点，推力将使用推力 图书馆的独家实现 粗略地看**最多任何能够创造实现并与Thrust竞争的人都可以获得2分的额外学分。**

部分 3: A 简单 圆圈 渲染器 (85 pts)

现在开始真正的表演!

赋值启动器代码的目录/渲染包含绘制彩色渲染器的实现圆构建代码，并使用以下命令行运行渲染过程：`rgb`。该程序将输出一个图像output_0000。PPM包含三个圆。现在运行使用命令行为雪的渲染器。现在，输出的图像将会下降 `./render -r cpuref` 雪PPM图像可以通过预览的方式直接在OSX上查看。对于windows，您可能需要下载一个查看器。

注意：您还可以使用-i选项将渲染器输出发送到显示器，而不是一个文件。（在雪的情况下，你会看到一个下雪的动画。）但是，要使用交互模式，您需要能够设置X-windows转发到您的本地机器。这个 参考资料 参考可能会有所帮助。）

赋值启动器代码包含两个版本的呈现器:一个顺序的、单线程的c++参考实现，在重新渲染器中实现。和一个不正确的并行CUDA实现 `cudaRenderer。cu .`

渲染器概述

`./render -r cpuref`

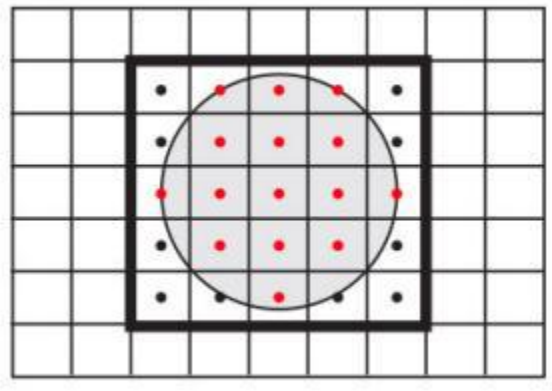
我们鼓励您通过检查参考文献来熟悉渲染器代码库的结构在refRenderer中实现。`cpp .`在渲染第一帧之前调用该方法设置。在你的CUDA加速渲染器，这个方法很可能包含你所有的渲染器初始化代码（分配缓冲区等）。致使

被称为每一帧，并负责将所有的圆圈绘制到输出图像中。的另一个主要功能渲染器，高级动画，也会在每一帧中被调用一次。它会更新圆圆的位置和速度。在这个作业中，你不需要修改advanceAnimation。

渲染器接受圆数组（三维位置、速度、半径、颜色）作为输入。基本顺序各帧的渲染算法为：

```
清晰的图像
每个圆圈
    更新位置和速度
每个圆圈
    计算屏幕边界框
    对于边界框中的所有像素
        计算像素中心点
        如果中心点在圆圈内
            计算点处圆的颜色
            为这个像素混合圆对图像的贡献
```

下图说明了使用圆中点测试计算圆像素覆盖的基本算法。请注意，只有当像素的中心位于圆内时，圆才会为输出像素提供颜色。

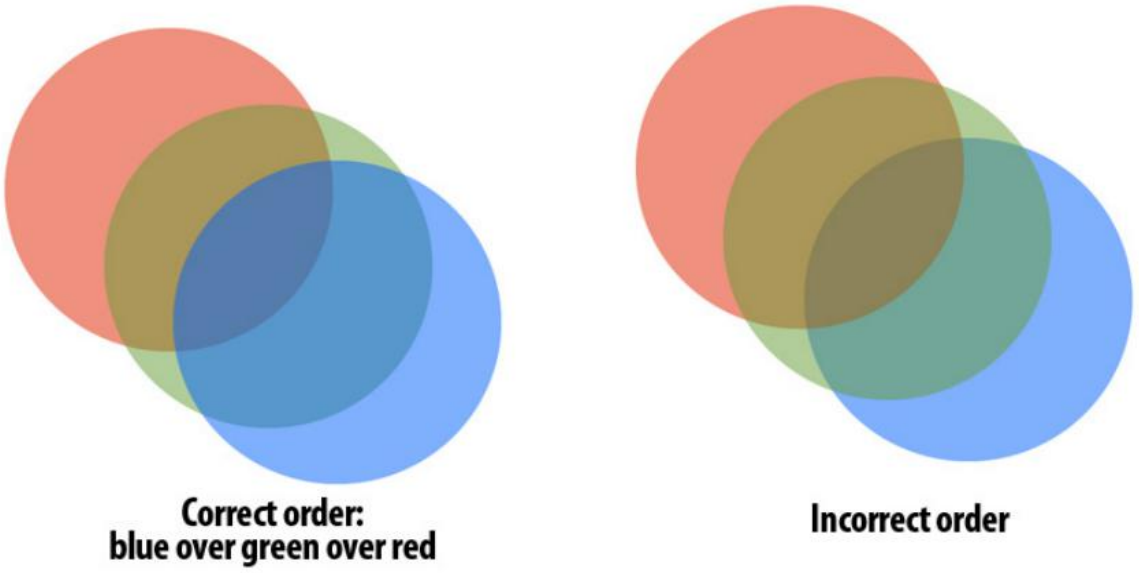


渲染器的一个重要细节是它渲染半透明的圆圈。因此，任何一个像素的颜色都不是单个圆的颜色，而是将所有与像素重叠的半透明圆的贡献混合的结果(注意上面伪代码的“混合贡献”部分)。渲染器通过一个包含红色(R)、绿色(G)、蓝色(B)和不透明度（alpha）值（RGBA）的4个元组来表示圆的颜色。Alpha = 1对应于一个完全不透明的圆圈。 α = 0对应于一个完全透明的圆圈。要在颜色为P_b的像素上绘制带有颜色为C_alpha的半透明圆，渲染器使用以下数学：

```
result_r = C_alpha * C_r + (1.0 - C_alpha) * P_r
result_g = C_alpha * C_g + (1.0 - C_alpha ) * P_g
result_b = C_alpha * C_b + (1.0 - C_alpha ) * P_b
```

请注意，组合不是可交换的（对象X到对象Y看起来与对象Y不一样），所以渲染绘制圆按照应用程序提供的顺序是很重要的。(您可以假设应用程序按深度顺序提供圆圈。)例如，考虑下面的两幅图像，一个蓝色圆圈画在一个绿色圆圈上 (C_r, C_g, C_b, (P_r, P_g,

它被画在一个红色的圆圈上。在左边的图像中，圆圈以正确的顺序被绘制到输出图像中。在右边的图像中，圆圈以不同的顺序绘制，输出的图像看起来不正确。



CUDA渲染器

在熟悉了在参考代码中实现的圆圈渲染算法之后，现在研究在可爱的渲染器中提供的渲染器的CUDA实现。铜。您可以运行梭达使用-渲染器cuda（或-r cuda）cuda程序选项实现渲染器。

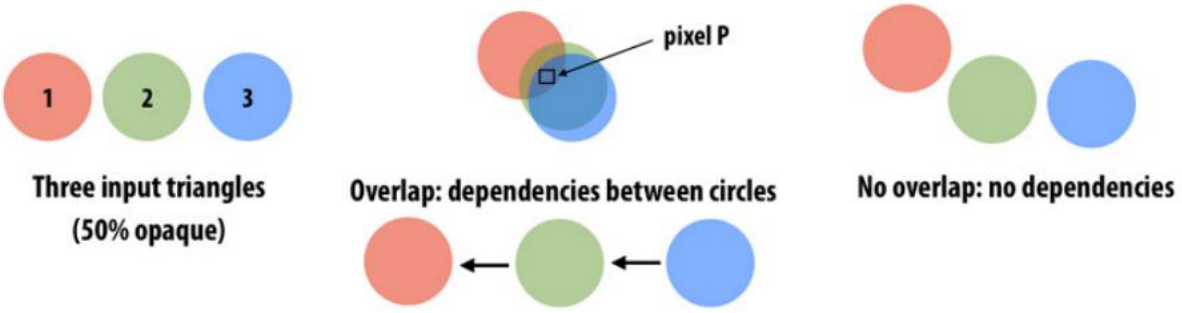
提供的CUDA实现在所有输入圆上并行计算，将一个圆分配给每个CUDA线程。虽然这个CUDA实现是一个数学的完整实现一个圆的渲染器，它包含几个主要的错误，您将修复在这个分配。具体来说：当前实现不确保映像更新是一个原子操作，并且它不保留所需的操作图像更新的顺序(排序要求将在下面描述)。

渲染器要求

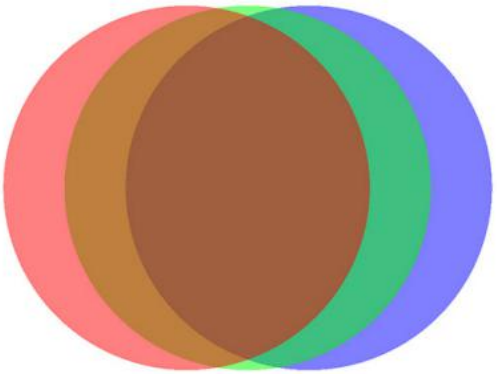
并行CUDA渲染器实现必须维护两个不变量，这两个不变量在顺序实现。

1. 原子性：所有图像更新操作都必须是原子性操作。关键区域包括阅读四个32位浮点值（像素的rgba颜色），混合了当前圆的贡献与将当前的图像值写入图像，然后将像素的颜色写回内存中。
2. 顺序：渲染器必须按圆圈输入顺序对图像像素执行更新。也就是说，如果圆1和圆2都对像素P有贡献，任何由于圆1而对P进行的图像更新都必须应用于

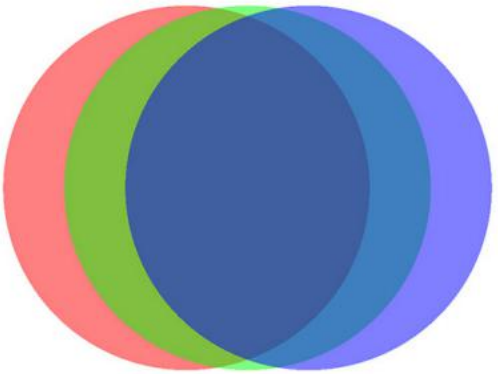
图像之前更新到P由于圆圈2。如上所述，保留订购要求
允许正确渲染透明圆。(它对图形还有许多其他的好处
系统**如果好奇，可以和Kayvon谈谈。)一个关键的观察是，order的定义只指定对相同像素的更新顺序。**因此，如下图所示，不构成相同像素的圆之间没有排序要求。这些圆可以独立处理。



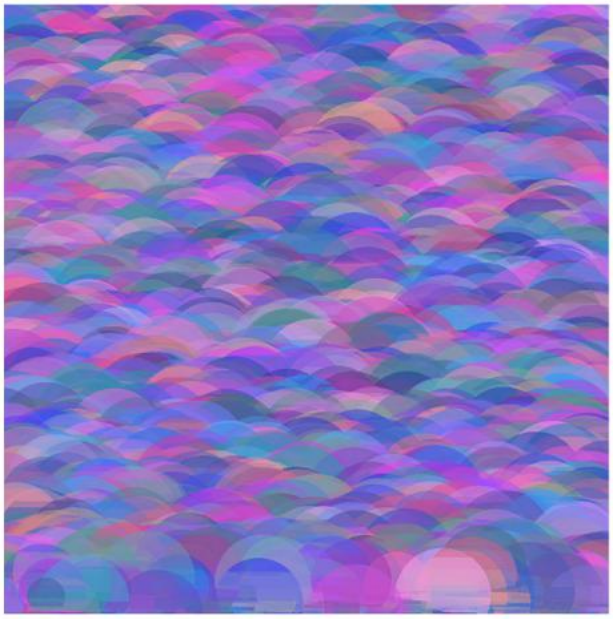
由于所提供的CUDA实现不满足这两种要求，因此结果不正确
通过在rgb和圆形场景上运行CUDA渲染器实现，可以看到尊重顺序或原子性。
您将在生成的图像中看到水平条纹，如下图所示。这些条纹会随着每一帧的变化而改变。



Incorrect (RGB)



Correct (RGB)



Incorrect (rand10K)

Correct (rand10K)

你需要做些什么

你的工作是编写最快的，正确的CUDA渲染器实现。你可以采取任何方法参见适合的，但您的渲染器必须遵守上面指定的原子性和顺序要求。不满足这两个要求的解决方案将在分配的第3部分中被给予不超过12分。我们已经给了你这样的解决方案！

一个很好的开始就是通过阅读指南。并说服自己它不符合正确性要求。特别是，看看CudaRenderer:render如何启动CUDA内核kernelRenderCircles。（角膜渲染圈子是所有工作发生的地方。）要直观地看到违反上述两个要求的效果，请使用make编译程序。然后运行。/渲染-r cuda rand10k，它应该显示带有10K圆圈的图像，显示在上面的图像的底部一行。将此（不正确的）图像与通过运行的顺序代码生成的图像进行比较。

我们建议您：

1. 首先重写CUDA启动程序代码实现，以便它在并行运行时在逻辑上是正确的（我们推荐一种不需要锁或同步的方法）
2. 然后确定您的解决方案有什么性能问题。

3....在这一点上，对作业的真正思考开始了(提示：在圆盒测试中提供给您的圆交叉盒测试。cu_inl是你的朋友。我们鼓励您使用这些子程序。)

以下是执行的命令行选项：

```
使用:./render [options] scenename
有效的场景名称是： rgb，橄榄球，兰德10k，兰德100k，大小，小大，图案，
                        弹球，烟花，催眠，雪，雪单身
程序选项：
  -r-渲染器<cpuref/cuda>选择渲染器：参考或cuda（默认的=cuda）

  -s-大小<INT>                          制作渲染图像<INT>x<INT>像素
(默认= 1024)
  -b——bench <START:END> -f—             运行帧[开始，END）（默认值[0、1））输出文件名
  —file <FILENAME> -c———                (FILENAME_xxxx。ppm)
  check                                 根据CPU检查CUDA输出的正确性
参考
  我——互动                             将输出呈现为交互式显示
  -?——帮助                             这个消息
```

检查代码：为了检测程序的正确性，渲染有一个方便的——检查选项。此选项运行参考CPU渲染器的顺序版本和CUDA渲染器，然后比较结果的图像，以确保正确性。您的CUDA渲染器实现所花费的时间也会被打印出来。

我们总共提供了五个圆形数据集，你将根据它们进行评分。然而，为了获得完整的信用，您的代码必须通过我们所有的正确性测试。要检查代码的正确性和性能分数，请运行。/检查器。Py(注意。在渲染目录中。如果您在启动代码上运行它，程序将打印如下类似的表，以及我们整个测试集的结果：

```
------------
评分表：
------------
-------------------------------------------------------------------------
|场景名称|裁判时间(T_ref) |你的时间(T) |得分|
-------------------------------------------------------------------------
| rgb                | 0.2321          | (F)              | 0              |
| rand10k            | 5.7317          | (F)              | 0              |
| rand100k           | 25.8878         | (F)              | 0              |
|模式                | 0.7165          | (F)              | 0              |
| snowsingle         | 38.5302         | (F)              | 0              |
| biglittle          | 14.9562         | (F)              | 0              |
-------------------------------------------------------------------------
|                                      |总分：  | 0/72                    |
-------------------------------------------------------------------------
```

注意：在某些运行中，您可能会得到这些场景的奖励，因为所提供的渲染器的运行时是不确定性的，有时它可能是正确的。这并不会改变当前的CUDA渲染器通常是不正确的事实。

“Ref time”是我们的参考解决方案在当前机器上的性能(在提供的render_ref可执行文件中)。“您的时间”是您当前的CUDA渲染器解决方案的性能，其中一个(F)表示一个不正确的解决方案。您的成绩将取决于您的实现与这些参考实现相比的性能（请参见分级指南）。

除了您的代码，我们希望您提供一个关于您的实现如何工作的清晰、高级的描述，以及您是如何实现这个解决方案的简要描述。具体解决您一路上尝试的方法，以及如何确定如何优化代码（例如，您执行了什么度量来指导优化工作？）。

你应该在文章中提到的工作方面包括：

1. 在你的文章的顶部包括合作伙伴的名字和SUNetid。
2. 复制为您的解决方案生成的计分表，并指定您在哪台机器上运行代码。
3. 描述您如何分解问题，以及如何将工作分配给CUDA线程块和线程（甚至可能是扭曲）。
4. 描述在您的解决方案中同步发生的位置。
5. 如果有的话，你采取了什么措施来减少通信需求(e。g., 同步或主内存带宽要求）？
6. 简要描述一下您是如何得出最终的解决方案的。一路上你还尝试了什么其他的方法。他们怎么了？

分级指南

- 这个作业的评分值7分。
- 你的实现值72分。这些分数被平均分成12分

  场景如下：
  - 每个场景2个正确性点。
  - 每个场景有10个性能点（只有在解决方案正确时才能获得）。

    您的性能将根据所提供的基准参考渲染器的性能进行评分。不可食用的:

    对于时间(T)为10倍的解决方案，不会给出性能点特雷夫.

    对于优化解决方案20%以内的解决方案，将得到完整的性能点(T <1.20 * Tref)

    对于T的其他值(对于1.20 Tref<=T <10 * Tref),你的表现1到10分的分数将计算为： 10 * T_ref / T。
- 你的实现在课堂排行榜上的表现值得得到最后的6分。

  排行榜的提交和分级细节将在随后的一篇文章中详细说明。
- 高达5分的额外学分（教师的自由裁量权）的解决方案，实现显著

  性能高于要求。你的写作必须清楚地彻底地解释你的方法。

-
-
-

- 高达5分的额外学分（教练的自由裁量权）为一个高质量的并行cpu只
  渲染器实现，实现了良好的利用所有核和核的SIMD向量单位。请随意使用任何工具。例如，SIMD intrinsic, ISPC, pthreads)。为了获得好评，您应该分析基于GPU和基于cpu的解决方案的性能，并讨论在实现选择中存在差异的原因。

因此，本项目的总得分如下：

- 第一部分(5分)
- 第2部分(10分）
- 第3部分的记录（7分）
- 第3部分的实施情况(72分）
- 第三部分排行榜(6分)
- 潜在额外信贷（最高达10分）

作业提示和提示

以下是前几年整理的一组技巧和提示。请注意，有各种方法可以实现渲染器，但并不是所有的提示都适用于您的方法。

- 在这个赋值中有两个潜在的并行性轴。*一个轴是平行的*
  跨像素的另一个问题是跨圆之间的并行性（如果重叠的圆符合排序要求）。解决方案将需要
  利用这两种类型的并行性，可能是在计算的不同部分。
  在圆盒测试中提供给您的圆交叉盒测试。我是你的朋友。我们鼓励您使用这些子例程。
  exclusiveScan提供的共享内存前缀和操作。Cu_inl可能是
  对您的这个任务很有价值（并不是所有的解决方案都可以选择使用它）。请参见这里对前缀和的简单描述。我们
  在共享内存中为两倍大小的数组提供独占前缀和的实现。**所提供的代码不能在非双功率输入上工作，而且它还要求线程中的线程数**
  块是数组的大小。请阅读代码中的注释。
- 您可以使用推力装置_如果你选择的话，库。
  无需使用推力来实现优化后的CUDA参考实现的性能。解决这个问题的一种流行方法是使用我们提供的共享内存前缀和实现。还有另一种流行的方法是使用Thrust库中的前缀和例程。这两者都是有效的解决方案策略。
  渲染器中是否有数据重用?我们可以做些什么来利用这种重用呢？
  由于没有CUDA语言原语原子执行图像更新操作的逻辑，您如何确保图像更新的原子性？构建锁定全局内存的原子操作是一种解决方案，但请记住，即使映像更新是原子的，也必须按照所需的顺序执行更新。**我们建议您首先考虑确保并行解决方案中的顺序，然后只考虑解决方案中的原子性问题（如果它仍然存在的话）。**
- 如果你发现自己有空闲时间，享受制作自己的场景吧！

捕获CUDA错误

默认情况下，如果您访问一个数组，分配过多的内存，或导致错误，CUDA通常不会通知您；相反，它只会无声地失败并返回一个错误代码。您可以使用以下宏（请随意修改它）来包装CUDA调用：

```
#定义调试

#ifdef调试
#define cudaCheckError(ans) {cudaAssert((ans), __FILE__, __LINE__);} inline void
cudaAssert(cudaError_t代码，const char *文件，int行，bool abort=true)
{
    如果（代码！=cuda成功）
    {
        “CUDA错误： %s在%s：%d\n”，
          cudaGetErrorString(代码)，文件，行);
       If (abort) exit(code);
    }
}
其他#
定义cuda检查错误（ans）ans
#endif
```

请注意，一旦代码正确可以提高性能，您可以取消定义调试以禁用错误检查。

然后，您可以包装CUDA API调用来处理它们返回的错误：

```
cudaCheckError ( cudaMalloc(&a, size*sizeof (int)) );
```

注意，不能直接包装内核启动。相反，他们的错误将在你所包装的下一个CUDA调用中被捕获：

```
内核<<<1,1>>>(a)；//假设内核导致了一个错误！
错误（同步（））；//错误打印在这一行
```

所有CUDA API功能、同步、视频、拥抱等都可以包装。

重要的是：如果一个CUDA函数错误之前，但没有被捕获，该错误将显示在下一个错误检查中，即使它包装了一个不同的函数。例如：

```
...
第742行：cudaMalloc（&a，-1）；//执行，然后继续
第743行：CUDA检查错误（a，b））；//打印“CUDA错误：记忆缺失”。铜:743”
...
```

因此，在调试时，建议您包装所有CUDA API调用(至少在
你写的代码)。

(来源:改编自此 堆栈 上溢 邮政

3.4传入说明

请使用梯度望远镜提交您的工作。如果你正在和伙伴一起工作，请记住在分级镜上标记你的伴侣。

1. **请将您的文件作为文件提交。**`pdf。`
2. `请执行sh create_submission。`**生成一个要提交到分级处理的zip。**注意，这将在您的代码目录中运行，使清理，所以您将不得不再次运行make来运行您的代码。如果脚本错误说“权限被拒绝”，您应该运行chmod +x创建_提交。sh，然后尝试重新运行该脚本。

我们的评分脚本将重新运行检查代码，允许我们验证您的分数匹配您提交的写体。`pdf .`我们也可能会尝试在其他数据集上运行您的代码，以进一步检查其正确性。