Part3

A Simple CUDA Renderer

TEAM MEMBER: Abhishek Vanamala
Jahnavi Valisetty

Instruction to execute:

- `export PATH="/usr/local/cuda/bin:$PATH"`
- `make`
- `make check`

Machine used:

HPCLAB

Image Size 1000

Errors in the given implementation:

1. **Atomicity:** All image update operations must be atomic. The critical region includes reading the four32-bit floating-point values (the pixel's rgba color), blending the contribution of the current circle with the current image value, and then writing the pixel's color back to memory.

2. **Order:** Given renderer will not perform updates to an image pixel in circle input order. That is, if circle 1 and circle 2 both contribute to pixel P, any image updates to P due to circle 1 must be applied to the image before updates to P due to circle 2.

As discussed above, preserving the ordering requirement allows for a correct rendering of transparent circles. A key observation is that the definition of order only specifies the order of updates to an individual pixel.

There is no ordering requirement between circles that do not contribute to the same pixel. These circles can be processed independently.

Problem Decomposition:

The provided CUDA implementation parallelizes computation across all input circles, assigning one circle to each CUDA thread.

Rendering will be serial because that's the only way to guarantee order and compute the color correctly. But determining which circles affect which pixels and coloring different pixels can be done in parallel. That's what we need to fix and improve.

__global__ void kernelRenderRegions():For each region find the circles that will affect it using shared memory.
Advantaged of using Shared Memory:

- We can process the circles in chunks
- For each pixel shade it according to the lists for the regions.

If you don't declare a shared array as `volatile`, then the compiler is free to optimize locations in shared memory by locating them in registers (whose scope is specific to a single thread), for any thread, at it's choosing. This is true whether you access that particular shared element from only one thread or not. Therefore, if you use shared memory as a communication vehicle between threads of a block, it's best to declare it `volatile`.

shadePixel():

Given a pixel and circle , determines the contribution to the pixel from the circle.The image update is done in this method .This method is called by the kernelRenderregions().

__global__ void kernelRenderPixelsSmall() : This function call is used for rendering pixel of the circles which are less that five circles. ShadePixel() method will be called to achieve this.

If you don't declare a shared array as `volatile`, then the compiler is free to optimize locations in shared memory by locating them in registers (whose scope is specific to a single thread), for any thread, at it's choosing. This is true whether you access that particular shared element from only one thread or not. Therefore, if you use shared memory as a communication vehicle between threads of a block, it's best to declare it `volatile`.

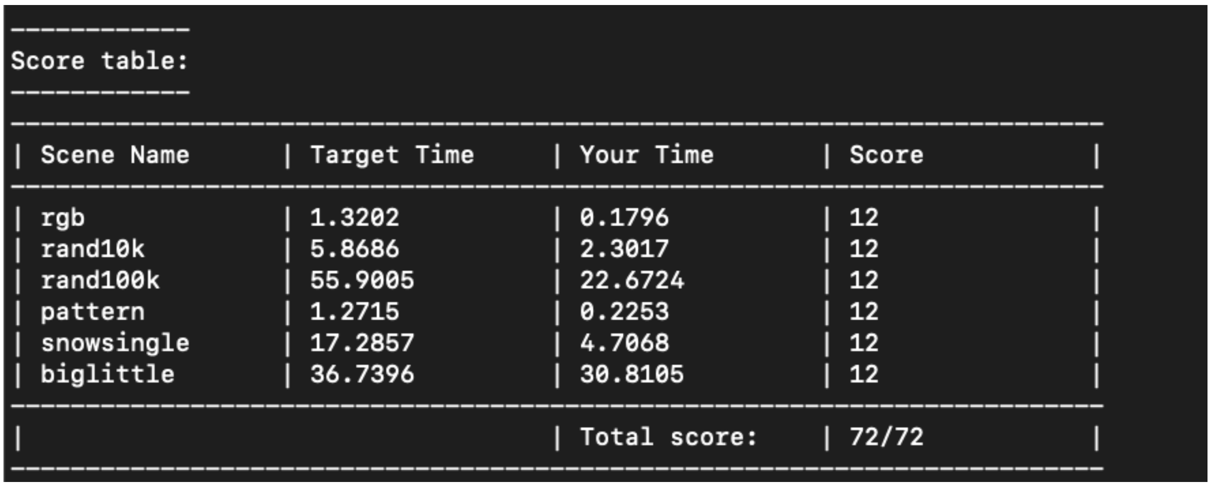
Phase 1:

- Find order of circles for each region , circles that affect the region.
- Allocate local memory based on how many circles assigned to each thread.
- Test how many circles out of circlesperthread affects that region and note down their order.
- Summary each threads private count to array circleIndicator .
- Perform prefix scan on circleIndicator array

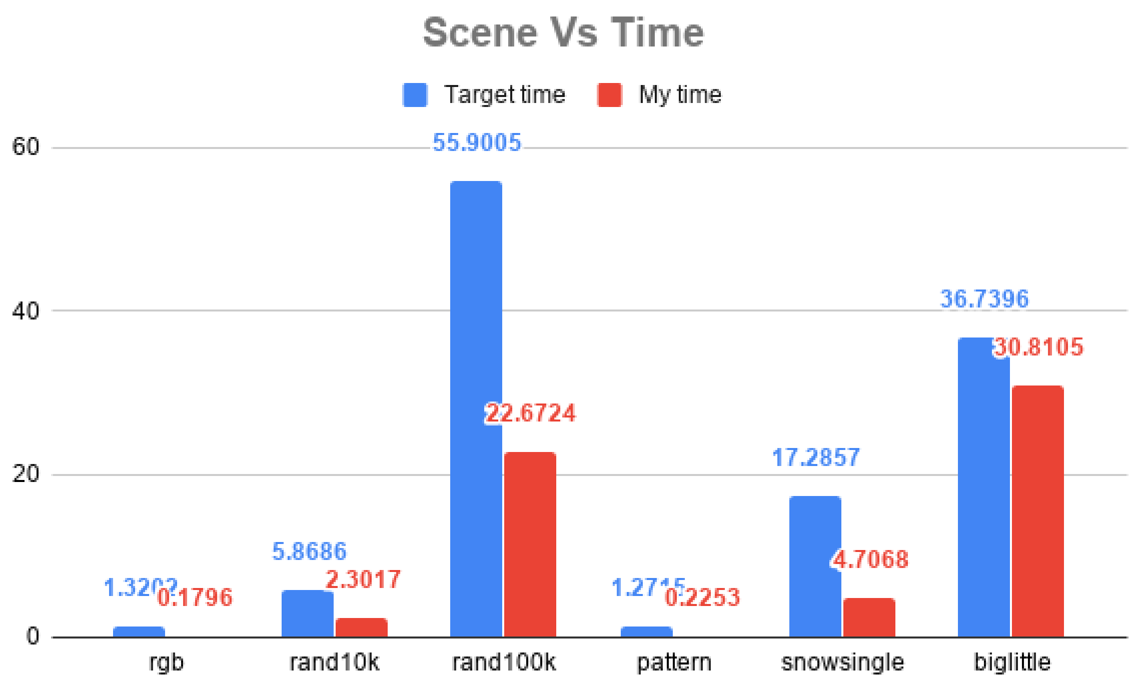
Phase 2:

- Iterate through the necessary circles given by the region corresponding to the pixel and shade the pixel

Score table:



```
------------
Score table:
------------
-------------------------------------------------------------------------
| Scene Name      | Target Time     | Your Time       | Score           |
-------------------------------------------------------------------------
| rgb             | 1.3202          | 0.1796          | 12              |
| rand10k         | 5.8686          | 2.3017          | 12              |
| rand100k        | 55.9005         | 22.6724         | 12              |
| pattern         | 1.2715          | 0.2253          | 12              |
| snowsingle      | 17.2857         | 4.7068          | 12              |
| biglittle       | 36.7396         | 30.8105         | 12              |
-------------------------------------------------------------------------
|                                   | Total score:    | 72/72           |
-------------------------------------------------------------------------
```


Scene Vs Time
Target time
My time
rgb
rand10k
rand100k
pattern
snowsingle
biglittle