Computationally Cheap Hair Simulation On GPU

Ahmet Dara VEFA

***Abstract*—Hair on computer games are often made without physics. This generally makes the hair look bad and makes the game character unrealistic. On the other hand there are physics based hair on some games like Witcher 3, however in my experience it was quite costly even though it was only enabled on the main character. This project aims to show, not create, that a GPU based hair simulation using constant point method can be used for fast and realistic hair/grass/fur simulation in video games.**

***Index Terms*—CUDA, Simulation, journal, paper.**

I. Introduction

THIS paper is intended to show you that a GPU based hair simulation using constant point method can be used for fast and realistic hair/grass/fur simulation in video games. I will go through motivation & significance, problem statement, prior works & limitations, theory/algorithm(s) and implementation, common mistakes, experiments or other evidence of success, discussion and future work, conclusion, references in that order.

mds
January 13, 2019

A. Motivation & Significance

Hairs in games are more often than not made without physics. Even when they are made with physics, it doesn't look that good and it decreases performance.

If this project is successful it will still look not that good, however it won't cost performance and will be a step towards "better" games/graphics.

II. Problem Statement

A. Prior Works & Limitations

Prior works on hair simulation exist, yet they are not that great for games. These works include NVidia HairWorks, which is the most known implementation of hair simulation in games. Like I said in introduction, HairWorks implementation is not good and causes performance decrease.

III. Theory/Algorithm(s)

The main algorithm is given in Algorithm 1.

Wind algorithm is given in Algorithm 2.

//TODO EXPLAIN STEPS TO IMPROVE GPU PERFORMANCE

Algorithm 1 Main Algorithm

```
procedure MAIN
    Set starting points of hairs.
    Set interpolated points and end points of hairs.
    Apply Wind() method.
    Check if all all the points are where they need to be.
    ▷ Since we aren't rendering we should check this within the code
end procedure
```

Algorithm 2 Wind

```
procedure WIND
    for each smoothing step do
        for each hair do
            calculate interpolated points and end points' new position.
        end for
        for each hair do        ▷ this is impelented as Collision Detection method
            push the hair outside of the head if any of the points is within boundary values.
        end for
    end for
end procedure
```

IV. Implementation

You will have to read the comments on the source code to get a really good understanding of what I did but here are some extra information:

A. Things To Note

In my implementation I used made many aspects expandable such as each hair can have a unique interpolated point size. These features forced me to not make CUDA(GPU) implementation as fast as it could be.

In my implementation head is set as a sphere for simplicity.

My implementation sets the hair as a curve around the head from the front to the back. If the hair is long enough it goes down the head when it can't touch the head.

In my implementation end points and interpolated points are elevated slightly above the head(if they are touching with the head) to give it "volume". Then again we can't see this unless we render.

In the source code you will only see methods to apply wind in +Z and -Z directions, however I wrote the way you can implement +-X and +-Y direction winds as well.

Right now you can't stop the winds, however all you have to do is stop all methods/kernels applying winds then apply

a strong wind from +Y axis. This would hopefully make the hair like it should be. Another way to stop the winds would be stopping all methods/kernels then either re-setting all hair points or returning the hair to its original position(we must save these positions before hand).

B. Common Mistakes

You will make mistakes when calculating the angles of rotation between two points of the hair. You must take into account the radius of the circle your starting point is touching; also be careful about distances between points on the hair and center of the circle your starting points is touching.
Also you might get confused and not set hair points on GPU correctly(since hair points are not a fixed size). Making hair points fixed size would also possibly give us a performance increase.

V. Experiments Or Other Evidence Of Success

Experiment results are given as an XML file, link in the appendix. Please take a look at the file while reading this section. These results are calculated on GTX 1060 3GB with 96.000 hairs that has 16 points in total with 30 step smoothing. I will be talking about Wind method in the subsections. Before we get to GPU report, I would like to mention CPU vs GPU performance. Note that CPU is Intel Core i3 7100 @3.9 GHZ and the tests are done with multithreaded CPU implementation.
In the tests I ran, I found out that when in release mode of visual studio, CPU would be about 2 times faster than GPU. However in debug mode GPU would be 2 times faster than CPU. So release mode is about 8 times faster for CPU while only being 1.25 times faster for GPU. So overall CPU is actually faster for this operation. However please don't forget that we are making smoothing(in this test case 30) amount of kernel calls for each wind method and collision detection method. In the following subsections you will understand other reasons why GPU is slow and ways to speed up the process.

A. Occupancy

When we examine these results we see theoretical occupancy of 75% and achieved occupancy of 63.11% on Wind method. The main reason for not being able to increase occupancy is max register count per SM(streaming multiprocessor) and max warps per block.

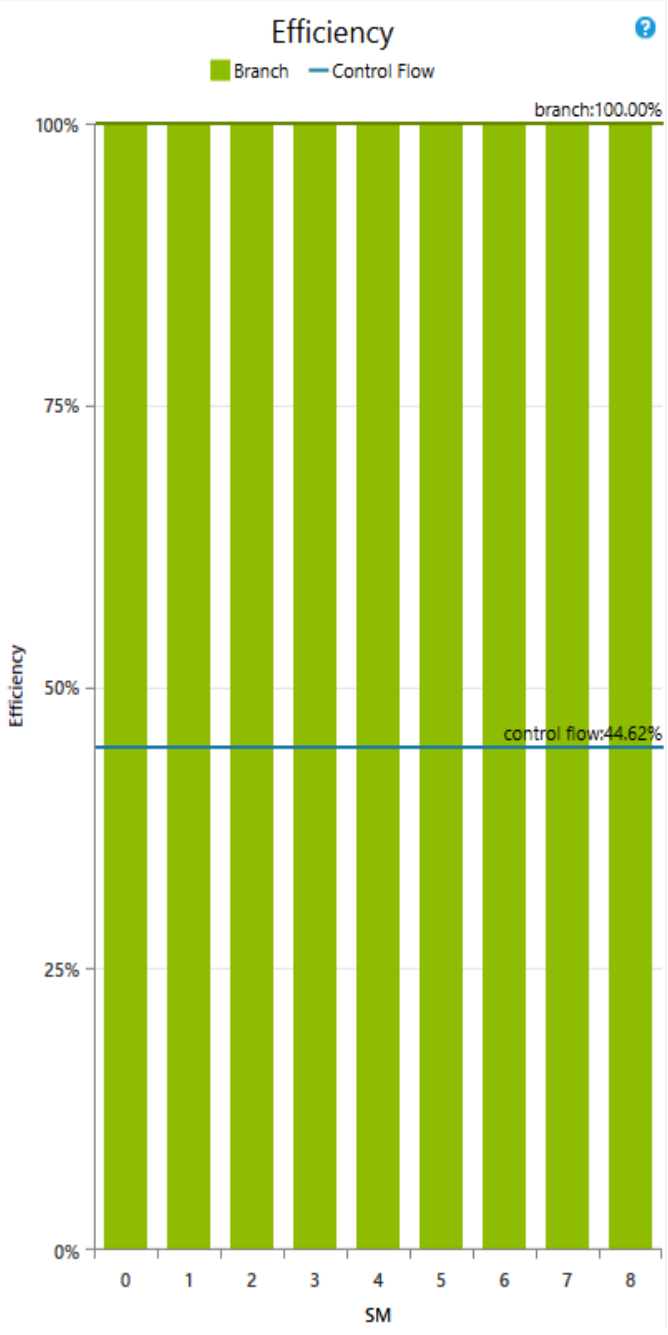
Variable	Achieved	Theoretical	Device Limit
^ Occupancy Per SM			
Active Blocks		2	32
Active Warps	40.39	48	64
Active Threads		1536	2048
Occupancy	63.11%	75.00%	100.00%
^ Warps			
Threads/Block		768	1024
Warps/Block		24	32
Block Limit		2	32
^ Registers			
Registers/Thread		33	255
Registers/Block		30720	65536
Registers/SM		61440	65536
Block Limit		2	32
^ Shared Memory			
Shared Memory/Block		0	49152
Shared Memory/SM		0	98304
Block Limit		∞	32

B. Instructions Per Clock

When we look at IPC(instructions per clock) graph we see issued instruction equal to executed instruction which is the most optimal outcome. However we see that the executed instruction is low at 1.01 meaning that we haven't efficiently used all of the available resources. Also when we look at SM activity graph(not included in this journal, you can find it in source-github), we see that not all SM's are equally active, to mitigate this we could fuse collision detection method with wind method to give all SM's time to catch up with others.

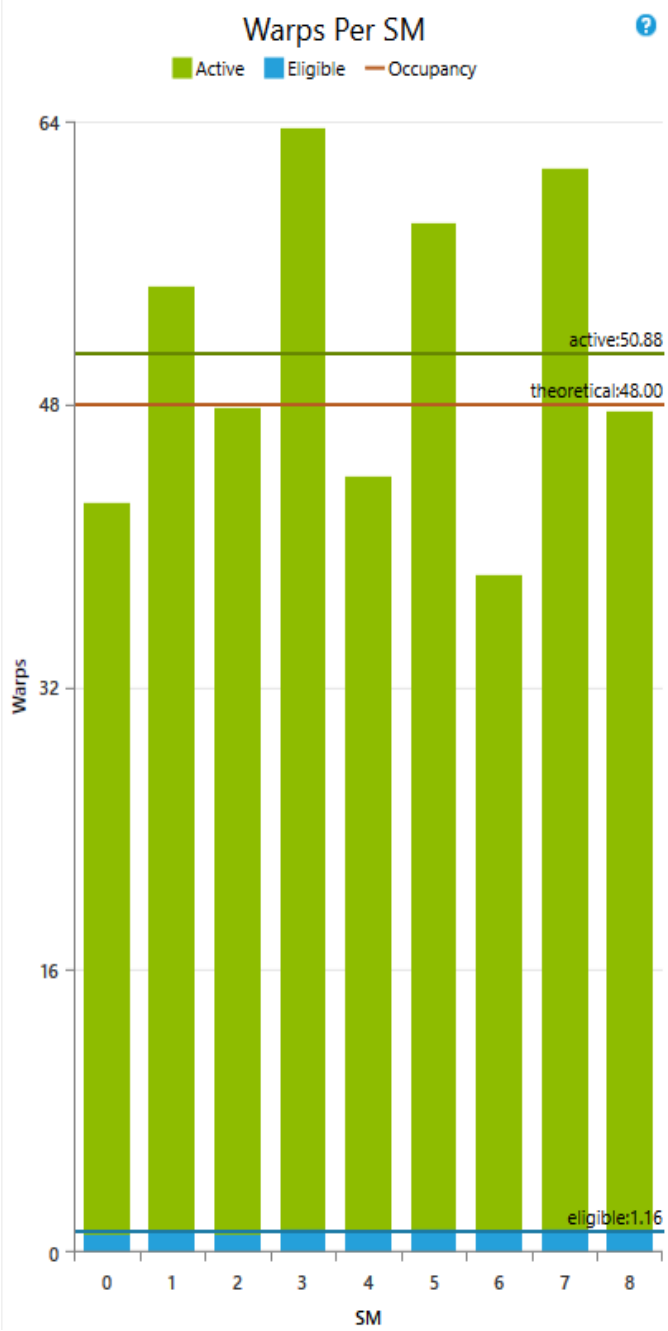
C. Branch Statistics

We see 100% branch efficiency meaning all the threads within the same warp followed the same path. However we see control flow efficiency as 44.62% if we examine this together with branch conditions(taken/not taken– not included in this journal, you can find it in source-github) we understand that the compiler mostly can't predict the path of our program correctly, which causes a huge decrease in performance(if you take pipelines in the account).



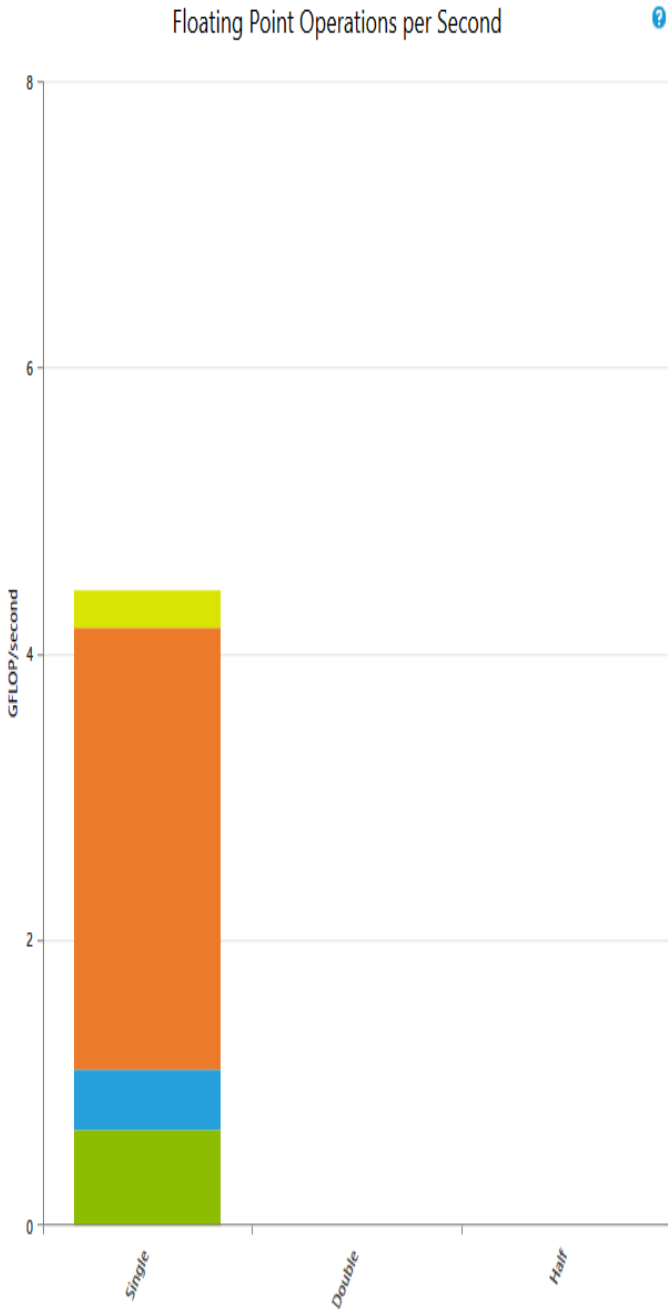
D. Issue Efficiency

When we take a look at warps per SM we see theoretical limit as 48, yet some of the SM's were able to surpass this limit(probably because they had faster CUDA cores), with only 1 of them getting close to maximum limit. We also see the memory dependency(not included in this journal, you can find it in source-github) is 7.52% of our issue stall reason. We can potentially reduce this by optimizing memory alignment and access patterns(like bank conflicts), but I don't see a way we can do that since all the hairs will change their positions when wind is applied causing all banks to be full.



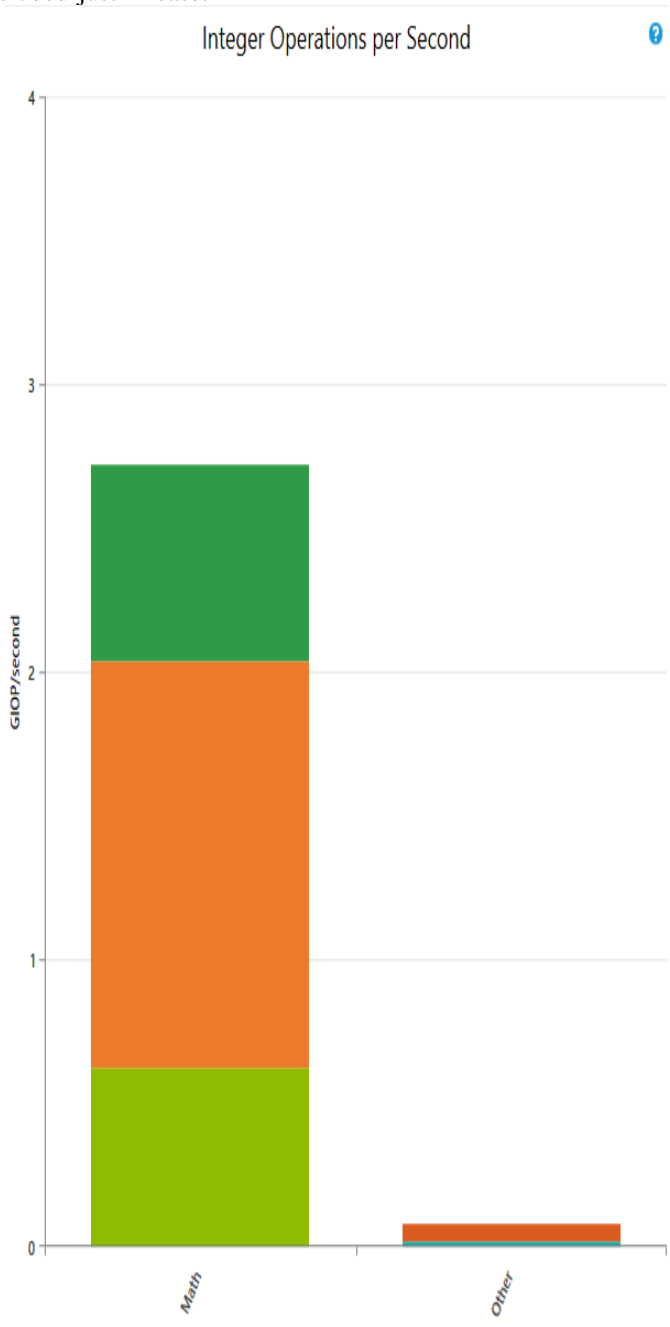
E. Achieved FLOPS(floating point operations per second)

We see that our FLOPS is about 4.1 GFLOPS/second. Which is about the same as max theoretical GFLOPS(for single precision only).



F. Achieved IOPS(integer operations per second)

I can't talk much about IOPS since I couldn't find any document indicating max IOPS count, however the graph is included just in case.



G. Memory Statistics

Even though I didn't explicitly specify to use texture cache compiler decided to use it. As we can see in the graph we have only 20% cache hits which is really really low, and 5.5 billion bank conflicts which is really high. As I mentioned before these bank conflicts probably can't be optimized but perhaps we could increase cache hits to get a better performance.

Name	Total	Per Warp	Per Second
Texture			
Fetches	0.00	0.00	0.00
L2 Transactions	701,612,200.0(	233,870.70	8,081,481,000.
Size	0.00 B	0.00 B	0.00 B/s
Cache Bank Conflicts	5,543,949.00	1,847.98	63,857,670.00
Cache Hit Rate	20.06%		

VI. Discussion And Future Work

It is clear from CPU and GPU implementation speeds that this project is not suitable for GPU at first. Given the time and effort I am pretty sure GPU implementation will be faster. Here are some things that would possibly make GPU implementation faster(note that these could be implementation specific):

- Fixed interpolated point size.
- not recalculating things like length between 2 points etc.
- sin/cos/arc tan lookup tables.
- unifying kernel calls(of wind and collision detection).
- implementing smoothing steps that will work with 1 kernel call.

There are also some more performance and memory improvement methods marked with TODOs in source code, however these are implementation specific.

Here are some things I made to make GPU version faster:

- creating hairs by groups of 32(note that I am still calculating wind effects for each hair).
- using #pragma unroll for loops– didn't seem to have an effect on performance.

Here are some additional features needed to make this project a viable option for hair/fur/grass simulation:

- rendering.
- creating more hair types(like wavy, curvy hair).

There are also some additional features marked with TODOs in source code.

VII. Conclusion

In conclusion this project needs time and effort to make it a viable option for hair simulation. In my opinion this could be successful.

Appendix A
Source Code & Performance Results & Poster & Everything Related With Project

These are located at github or go to the link https://github.com/venediklee/CUDA---Hair-Simulation

Acknowledgment

I would like to thank Ali Bahadr OZDOL for his help in solving some of the 3D coordinate related problems.

References

NVidia HairWorks https://developer.nvidia.com/hairworks