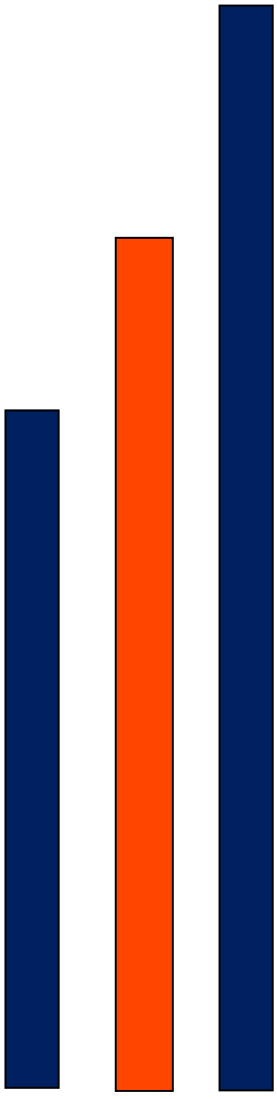
Vulkan Rounded Cell Particle Detection.



User Guide

Jackie Bell

5, 2024

Table of Contents

1 Report of 2024 05 26

I have spent the last couple of weeks solidifying the benchmarking/verification process. Section 2 is a full write up. It is probably important to read this since it shows - 1) The system has been tested in detail for accuracy, 2) the common benchmarking and reporting scheme. This will help to forstall any embarrassment over performance claims.

I have tested the system in three parts. One with no compute pipeline and one with both the compute and vertex pipelines active. The compute pipeline performance is the difference between the total and the graphics. Table 1 is the graphics pipeline performance, table 2 is the compute performance, and table 3 is the combined performance.

This is also reflected in fig. 3 where all three are plotted. It appears that both the compute and graphics pipelines are sub-linear (do you agree?). I investigated the wide variance with NSight Systems which allows me to see the performance of the CPU and GPU together. The red band in fig. 1 is called a *stutter* which occurs allot. NSight reports on the cause illustrated in fig. 2 and it appears if this is fixed the frame rates will have far less variance. The message means the CPU is taking a longer time to build and send a command package to the GPU than the GPU time it takes to process the commands.

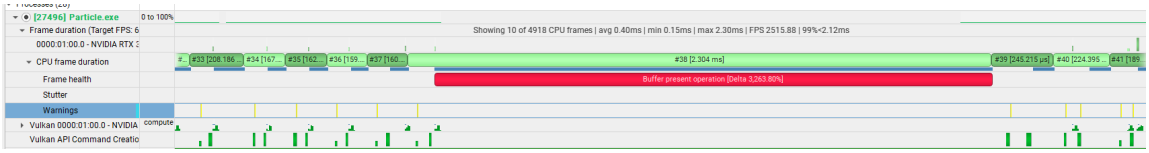


Figure 1: NVidia Insight systems trace of GPU-CPU.

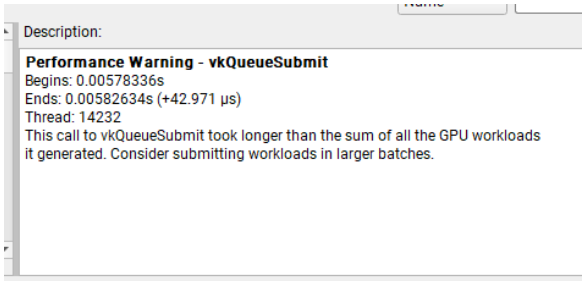


Figure 2: NVidia Insight systems trace message of GPU-CPU.

Table 1: Graphics pipeline performance by number of particles and collisions where max. cell population is 8.

Particles	Collisions	fps	Linearity
1024	0	2815	3.469e-07
17408	0	2309	2.488e-08
33792	0	2045	1.447e-08
50176	0	1445	1.379e-08
82944	0	1009	1.195e-08
115712	0	792	1.091e-08
246784	0	473	8.567e-09
508928	0	308	6.380e-09
754688	0	202	6.560e-09
1000448	0	160	6.247e-09
1246208	0	98	8.188e-09
1508352	0	110	6.027e-09

Table 2: Compute pipeline performance by number of particles and collisions where max. cell population is 8.

Particles	Collisions	fps	Linearity
1024	512	53	0.000e+00
17408	8704	201	3.220e-07
33792	16896	366	1.041e-08
50176	25088	477	7.006e-09
82944	41472	72	2.522e-09
115712	57856	79	1.904e-09
246784	123392	119	5.225e-09
508928	254464	107	3.075e-09
754688	377344	74	5.389e-09
1000448	500224	57	-3.517e-10
1246208	623104	16	2.724e-09
1508352	754176	38	-8.798e-10

Table 3: Graphics and compute pipelines performance by number of particles and collisions where max. cell population is 8.

Particles	Collisions	fps	Linearity
1024	512	2762	3.469e-07
17408	8704	2108	3.469e-07
33792	16896	1679	2.488e-08
50176	25088	968	2.080e-08
82944	41472	937	1.447e-08
115712	57856	713	1.282e-08
246784	123392	354	1.379e-08
508928	254464	201	9.454e-09
754688	377344	128	1.195e-08
1000448	500224	103	5.896e-09
1246208	623104	82	1.091e-08
1508352	754176	72	5.147e-09

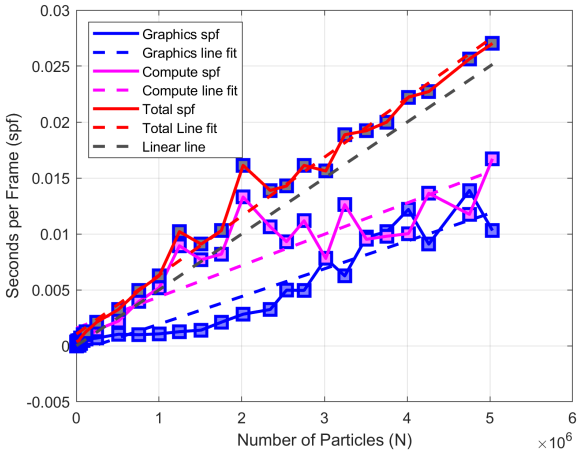


Figure 3: Seconds per frame for 0.5 collision density with 30 particles per cell verses number of particles.

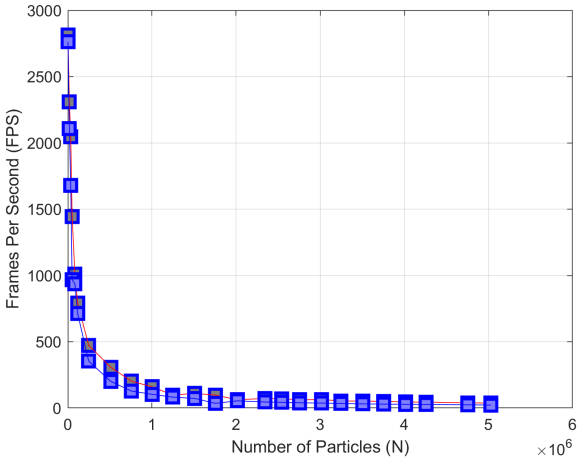


Figure 4: Frames per second for 0.5 collision density with 30 particles per cell verses number of particles.

2 Benchmarking - Verification

Reports of image space methods for particle collision detection do not provide common benchmarking and detailed verification making it difficult or impossible to compare methods. To rectify this issue a common benchmarking methods referred to as the maximum machine render rate which can be applied to any particle collision detection method. An application called ***genApp*** generates benchmarking data and is "light weight" in that it avoids transmission of very large datasets. ***verApp*** is another application provides that serves to verify the number of particles and collisions in datasets. Further, ***verApp*** includes testing algorithms such as location rounding and array indexing. Further a suite of MatLab scripts perform analysis on the collected data and export plots, tables, and images in latex format. The following details the verification process beginning with the maximum machine render rate.

2.1 MMRR

The testing metric will be based on what we refer to as the *maximum machine render rate* (***mmrr***). This rate is based on the time it takes a CPU-GPU combination to render a simple multicolored non-index drawn triangle with vertex information embedded in the vertex stage[]. The test is run for one minute recording the frame-rate every second then taking the mean of the 60 timings. The ***mmrr*** software used here is a simple Vulkan application based on the *Willams triangle*[].

The *maximum application render rate* (***marr***) is the the mean of the one minute timings for the application divided by the ***mmrr***. The ***marr*** is then the fraction of time it takes to render a frame by an application relative to the maximum rendering speeds of the CPU-GPU combination. This application rate ratio (***arr***) is intended to provide a more meaningful metric and to provide portability across various hardware configurations for particle collision only.

A number of studies will be required to fully test the performance and reliability of this method which claims to compute the exact number of collisions. These are configured to perform a number of standard tests. The *particle density* tests the number of particles that occupy a single cell. The *collision density* tests the a varying number of collisions relative to a fixed number of particles. The *particle volume* tests an increwasing number of particles at a constant particle and collision density.

Particle volume

1. Number of particles:variable
2. Particle cell density:fixed

3. Number of collisions:fixed

Particle density

1. Number of particles:variable
2. Particle cell density:variable
3. Number of collisions:fixed

Collision density

1. Number of particles:fixed
2. Particle cell density:fixed
3. Number of collisions:variable

There are situations where particle density varies greatly over the rounded space. For instance a group of cells may be have a high density of particles while other particles are spread in cells far away, and where some cells do not contain any particles. The *spread test* should show high performance since if a cell does not contain a particle the cell is never touched.

The overall process entails generating data, verifying that data, running the application against that data, and finally verifying that the application is actually detecting the parameters dictated by the datasets.

2.2 Generate Benchmark-Verification Data

	1	2	3	4	5	6	7	8	9	10	11	12
1	wx	wy	wz	dx	dy	dz	tot	sel	cols	collision	cdens	radius
2	1	1	1	33	1	1	32	s	10	0	0.5	0.2
3	1	1	1	1025	1	1	1024	s	10	0	0.5	0.2
4	1	1	1	17409	1	1	17408	s	10	0	0.5	0.2
5	1	1	1	33793	1	1	33792	s	10	0	0.5	0.2
6	1	1	1	50177	1	1	50176	s	10	0	0.5	0.2
7	1	1	1	66561	1	1	66560		10	0	0.5	0.2
8	1	1	1	82945	1	1	82944	s	10	0	0.5	0.2
9	1	1	1	99329	1	1	99328		10	0	0.5	0.2
10	1	1	1	115713	1	1	115712	s	10	0	0.5	0.2

Figure 5: Behcmark dataset configuration file.

Fig. 5 shows the format of the *benchmark configuration file* which is read by ***genApp*** to produce datasets. The `wx,wy,wz` columns configure the work-groups in the compute kernel (see fig. **??**). The `dx,dy,dz` columns configure `vkSubmit(..)` dimensions. The `tot` column provides the number of particles to be generated. The `sel` column selects that line for generation. The `cols` column configures the length of the second dimension of the

particle cell hash table which must be equal to the number of particles that can occupy a cell. The `cdens` column configures the percentage of particles that will be colliding. The `radius` column provides the radius of the particles which can be varied to produce different particle cell density.

genAppreads this file line by line, generates the datasets and a *test configuration file*. The test configuration file provides the side length of the cell array which is square int the case of bench marking. The file also provides the particles per cell (cell density), the total number of particles (`pcount`), the number of collisions (`colcount`), and the binary file containing the particle data (`datafile`). For each run the ***rccdApp*** reports the results which are written to the file name in `aprFile`. The `density` field provides collision density as a percentage and `ColArySize` is the particle cell hash table cell length.

```
index = 0;
Sidelen = 3;
PartPerCell = 8;
pcount = 32;
colcount =16;
dataFile = "perfdataT/0000
     CollisionDataSet32X16X3.bin";
aprFile = "perfdataT/0000
     CollisionDataSet32X16X3";
density = 0.50;
ColArySize =10;
dispatchx = 33;
dispatchy = 1;
dispatchz = 1;
workGroupsx =1;
workGroupsy =1;
workGroupsz =1;
```

Figure 6: The benchset test configuration file provides information on each test to be run by ***rccdApp***

2.3 Run Benchmark data

The ***rccdApp*** is run in verification mode where it iterates over each benchmark file in a loop. ***rccdApp*** reads the data from each configuration file, loads the benchmark data, creates a results file, and runs the test. Each test runs for 60 seconds recording the number of particles processed, the number of collisions detected, and the frame rate for each second. Fig. 7 illustrates the format of the report file. The `time`, `fps`, `spf` fields provide the time in second of the frame, the framemrate and time per frame. The `expectedp`, `loadedp`, `shadercomp`, `shadergrp`, report the number of particles generated by ***genApp***, the number of particles loaded by ***rccdApp***, the number of particles processed by the compute and vertex

kernels respectively. Any mismatch of these fields results in the creation of an error file which contains the error data. The `expectedc`, `shaderc`, reports the number of collisions expected from ***genApp*** and the number of collisions actually processed by the compute kernel. If the collsions expected are not equal to the number of collisions processed an error file is written. The `threadcount` reports the number of threads launched by the compute kernal which allows tuning of the workgroup parameters.

Each set of data can be run with only the graphics pipeline or with both the graphics and compute pipeline active in order to determine the performance of these separately (see fig. 3).

time	fps	spf	expectedp	loadedp	shaderp_comp	shaderp_grp	expectedc	shaderc	threadcount	sidelen	density
0	2322	0.430663	32	32	32	32	16	16	33	3	0.5
1	2734	0.365764	32	32	32	32	16	16	33	3	0.5
2	2868	0.348675	32	32	32	32	16	16	33	3	0.5
3	2808	0.356125	32	32	32	32	16	16	33	3	0.5
4	2713	0.368596	32	32	32	32	16	16	33	3	0.5
5	2804	0.356633	32	32	32	32	16	16	33	3	0.5
...	...	...	...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...	...	...	...
59	702	1.4245	32	32	32	32	16	16	33	3	0.5
60	683	1.46413	32	32	32	32	16	16	33	3	0.5
0	0	0	32	32	32	32	16	16	33	3	0.5
Total	910.37	1.42448									

Figure 7: Results reporting file.

2.4 Analyze the data

Matlab scripts (***matApp***) is run over the results data collecting parameters and checking for errors. This data is coalesce into frame rate v number of particles, time per frame versus number of particles at the same collision density and frame rate versus constant number of particles over varying density, Fig. 4, Fig. 3, and result tables 1, 2, and 3.