Fast Parallel Index Based GPU Particle Collision Detection

CEN5035

Project Proposal

FPIBGUtility

Project Team

Back End

```
Jack Bell - jb2ndr@ufl.edu
TBD
```

Front end

```
Efren Lopez - efren.lopez@ufl.edu
Kristen Moy - kristenmoy@ufl.edu
```

Project Proposal

Overview

I have a software application (called FPIBG) that uses the Graphics Processor to perform particle flow simulations. It is not a Web application so the back and front ends are defined differently. This application is written in C++ using the Vulkan API but this will not be part of the project. Instead we are writing code to control it in Python.

It is a great project for people wanting to go into reaserch becasue there is allot of analysis of algoritm performance (time complexity) and formating analysis data. Nothing is complicated and all of the code is prototyped in Matlab.

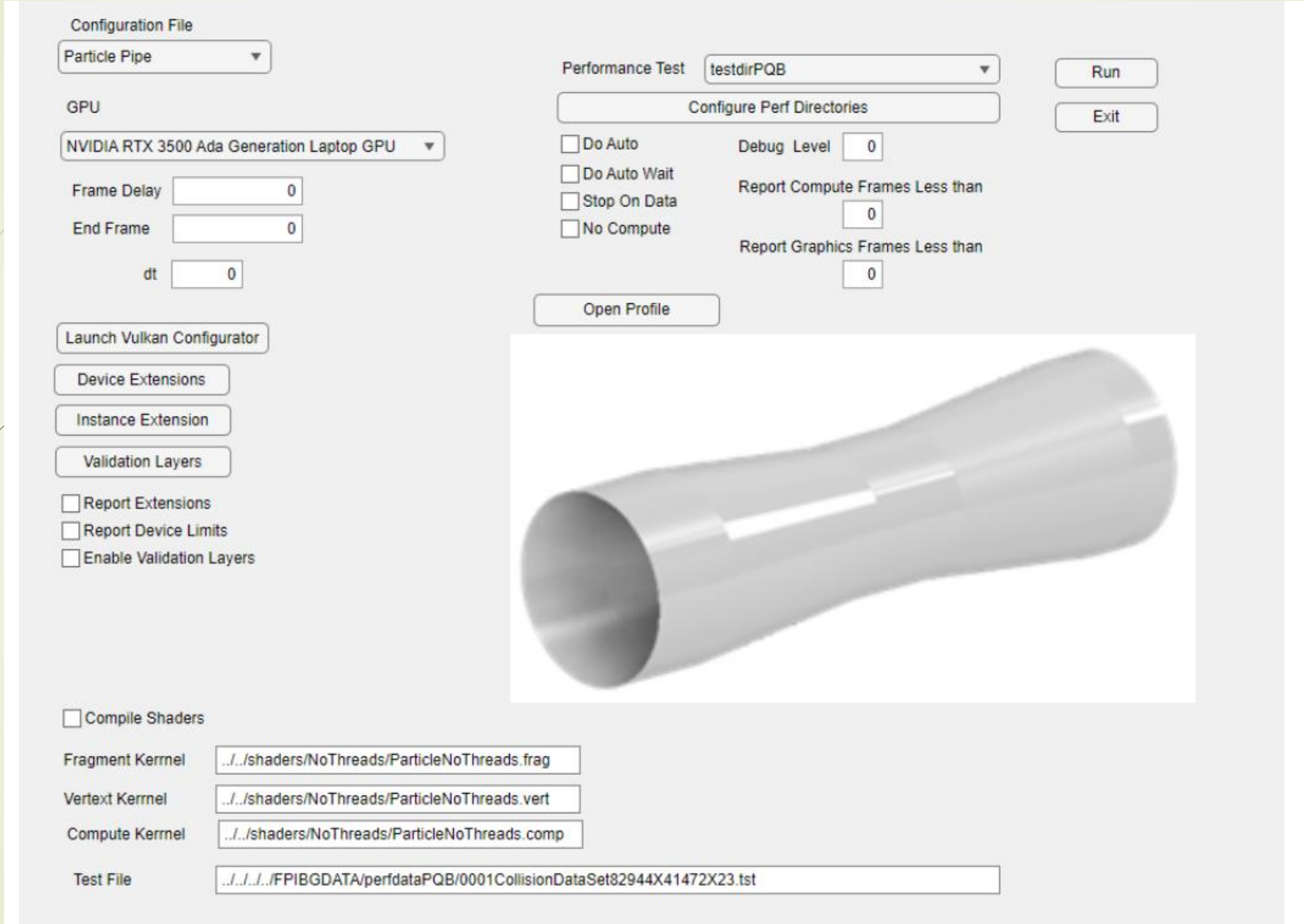
The paper on the application can be found here: https://github.com/fieldparticle/FPIBG/blob/main/2024_10_22_FPIBG_For%20_Publish.pdf

The application needs a front-end to configure and launch the FPIBG simulation run and a back-end to produce reports from the generated data. I need to use Pyton because the research comunity in the particle simulation field uses it for these kinds of utilities. It also needs to be portatble across MSWIN, LIMUX, and MACOS.

Front-end

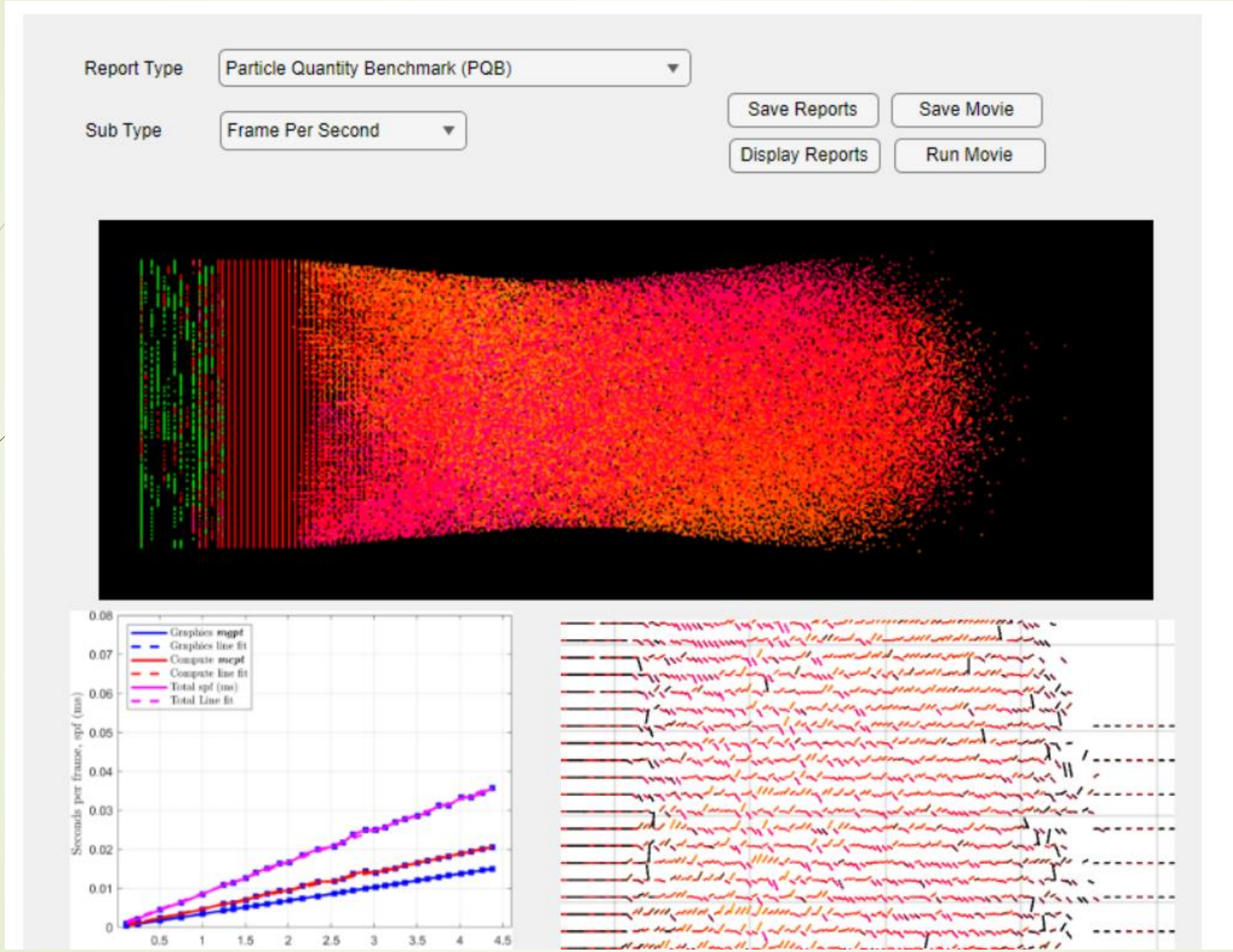
```
The front provides the following facilities:
1. Reads and writes to a configuration file via the libcofig API.
2. Reads an image of the flow device in a 3D file format called Wavefront (*.obg) being studied
       https://en.wikipedia.org/wiki/Wavefront_.obj_file
3. Launch and control the simulation via tcp/ip.

The configuration file provides the data for the FPIBG run. One of those configuration items is the pro
device through which the particles will flow. In the example GUI below shows a Converging-Diverging Noz
anything like a section of round pipe. The front-end must be a separate utility that
communicates with the application via TCP/IP becasue it can run the FPIBG application
locally or it can launch multiple tasks of the application in a High Performance Computing
environment (something we won't do here).
```


Configuration File
Particle Pipe
GPU
NVIDIA RTX 3500 Ada Generation Laptop GPU
Frame Delay 0
End Frame 0
dt 0
Launch Vulkan Configurator
Device Extensions
Instance Extension
Validation Layers
Report Extensions
Report Device Limits
Enable Validation Layers
Performance Test
testdirPQB
Configure Perf Directories
Do Auto
Do Auto Wait
Stop On Data
No Compute
Debug Level 0
Report Compute Frames Less than 0
Report Graphics Frames Less than 0
Open Profile
Run
Exit
Compile Shaders
Fragment Kerrnel ../../shaders/NoThreads/ParticleNoThreads.frag
Vertext Kerrnel ../../shaders/NoThreads/ParticleNoThreads.vert
Compute Kerrnel ../../shaders/NoThreads/ParticleNoThreads.comp
Test File ../../../../FPIBGDATA/perfdataPQB/0001CollisionDataSet82944X41472X23.tst

Back-end

```
The back-end provides the following features.
1. Reads data generated by the simulation run.
2. Produces and displays plots of the data.
3. Writes those plots in Latex format so that they can be ncluded in report documents.
4. Reads captured image files of each frame and converts the color values
at each pixel into scalar or vector fields.
5. Plots the scalar or vector plots to Latex format for inclusion in reports.
6. Assembles image files into movies.
```



Report Type
Particle Quantity Benchmark (PQB)
Sub Type
Frame Per Second
Save Reports
Save Movie
Display Reports
Run Movie
Graphics mgpt
Graphics line fit
Compute mcpt
Compute line fit
Total spf (ms)
Total Line fit
Seconds per frame, spf (ms)
0.08
0.07
0.06
0.05
0.04
0.03
0.02
0.01
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5

Project Process Overview

I. Each sprint (probably a week and called a module) there are two assignments.
   A. Perform the coding for the sub module.
   B. Create a sub module test bed to test and exercise.
II. Each module will have a branch named MODNNN where NNN is the module number.
III. Each module will have a test procedure (maybe google test?)
IV. At the end of the module the everyone's work is merged and tested.
V. Upon successful testing start the new module.

Baby Steps!

Module 001 Deliverables

Module 001 Task Description

M001Task 01 – Create base data object.

M001Task02 – Create base tcpip object.

M001Task03 – Incorporate libconfig.

M001Task04 – Create logfile class

Base Data Class

MODULE 001 BaseClass

CLASS
FPIBGBase __init__(workingDirectory, dataDirectory, logFileHandle, configFileHandle)

MEMBER VARIABLES
m_logFileHandle
m_configFileHandle
m_workingDirectory
m_dataDirectory

MEMBER FUNCTIONS

*Items in red are new elements this module.

Log File Class

MODULE 001 Logfile class

CLASS
logData

MEMBER VARIABLES
m_baseClass

MEMBER FUNCTIONS
openLogfile
closeLogFile
setAppName
setFunctionName
setErrorCode
writeEntery

*Items in red are new elements this module.

MODULE 001 Logfile class

```
Line Number:Date:24hrtime:Application:Object:Function:ErrCode:Error String
```

Example:

```
001:010825:1400:FPIBGAPPFRONT:DATAObject:getData:0005:Get Data Fields – File Not Open
```

Configuration File Class

MODULE 001 Configuration file class

CLASS
configData
MEMBER VARIABLES
m_baseClass
m_getFieldName
m_getFieldData
m_setFieldName
m_setFieldData
MEMBER FUNCTIONS
openDatafile
closeDataFile
readDataFile

*Items in red are new elements this module.

MODULE 001 Configuration file class – Configuration File

```
name = "particleOnly";
version = "11";
application =
{
      window =
      {
            title       = "MPS";
            // Size of the window.
            size  = { w = 1000; h = 1000; };
      };
      phys_device             = "NVIDIA RTX 3500 Ada Generation Laptop GPU";
      //phys_device           = "Intel(R) HD Graphics P530"
      frame_delay             = 0;
      end_frame               = 0;
      dt                = 0.1;
      cap_name                = "cube";
      cap_num                 = 0;
      cap_frames              = 237;
      framesBuffered          = 1;
      frag_kernParticle             = "../../shaders/NoThreads/ParticleNoThreads.frag";
      frag_kernParticlespv          = "frag2.spv";
      vert_kernParticle             = "../../shaders/NoThreads/ParticleNoThreads.vert";
      vert_kernParticlespv          = "vert2.spv";
      comp_kernParticle             = "../../shaders/NoThreads/ParticleNoThreads.comp";
      comp_kernParticlespv          = "comp.spv";
      doAuto                  = false;
      doAutoWait              = 0;
      testfile                = "../../../../FPIBGDATA/perfdataPQB/0001CollisionDataSet82944X41472X23.tst";
```

CONTINUED
NEXT PAGE

MODULE 001 Configuration file class – Configuration File

```
	//perfTest			= "testdirPQB";
	//perfTest			= "testdirPCD";
	//perfTest			= "testdirCFB";
	perfTest		= "testdirDUP";
	testdirPQB			= "../../../../FPIBGDATA/perfdataPQB"
	testdirCFB			= "../../../../FPIBGDATA/perfdataCFB"
	testdirPCD			= "../../../../FPIBGDATA/perfdataPCD"
	testdirDUP			= "../../../../FPIBGDATA/perfdataDUP"
	compileShaders			= true;
	enableValidationLayers	= true;
	stopondata		= true;
	noCompute		= false;
	debugLevel		= 0;
	reportCompFramesLessThan	= 3;
	reportGraphFramesLessThan	= 0;
	framesInFlight		= 1;
	device_extensions		= (  "VK_NV_shader_sm_builtins",
				"VK_NV_shading_rate_image",
				"VK_EXT_shader_subgroup_ballot",
				"VK_KHR_swapchain",
	"VK_KHR_shader_non_semantic_info",
				"VK_EXT_fragment_shader_interlock",
	"VK_EXT_shader_image_atomic_int64",
				"VK_EXT_shader_atomic_float"  );
	instance_extensions = ( "VK_EXT_debug_utils","VK_EXT_debug_report");
	validation_layers	= ( "VK_LAYER_KHRONOS_validation" );
	printExtension		= false;
	printDevLimtits			= true;
	verbose_rpt		= false;
};
```

Performance Data Class

MODULE 001 Performance data class

CLASS
DataClass
MEMBER VARIABLES
m_baseClass
MEMBER METHODS
openDatafile(self)
closeDataFile(self)
readDataFile(self)
getDataColumn(self,ColumnName)
printDataColumn(self,ColumnName)

*Items in red are new elements this module.

Collision Fraction Benchmark Data Format, Plots, Tables

MODULE 001 Performance data class

CFB – Collision Fraction Benchmark

time	fps	cpums	cms	gms	expectedp	loadedp	shaderp_comp	shaderp_grph	expectedc	shaderc	threadcount	sidelen	density	PERR	CERR
0	763	0.001311	0.000568	0.00044	117648	117648	0	0	58824	0	0	26	0.5	0	0
1	847	0.001181	0.000524	0.000431	117648	117648	0	0	58824	0	0	26	0.5	0	0
2	870	0.001149	0.000523	0.00043	117648	117648	0	0	58824	0	0	26	0.5	0	0

MODULE 001 Performance data class

CFB – Collision Fraction Benchmark

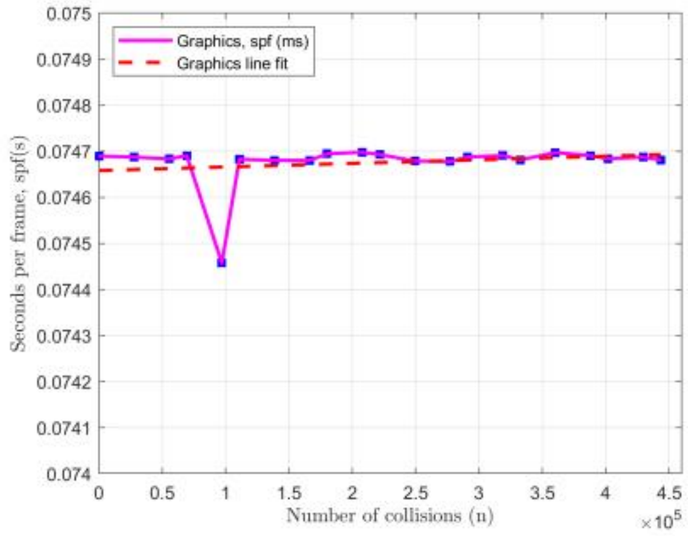


Figure 13. *Collision Fraction Benchmark (CFB), graphics execution time versus number of collisions. For 443392 particles. The collision fraction,* $\boldsymbol{F}$ *varies from 0.0 to 1.0 in 0.05 increments and where particle-cell density,* $\boldsymbol{max_p}$ *is 8.*

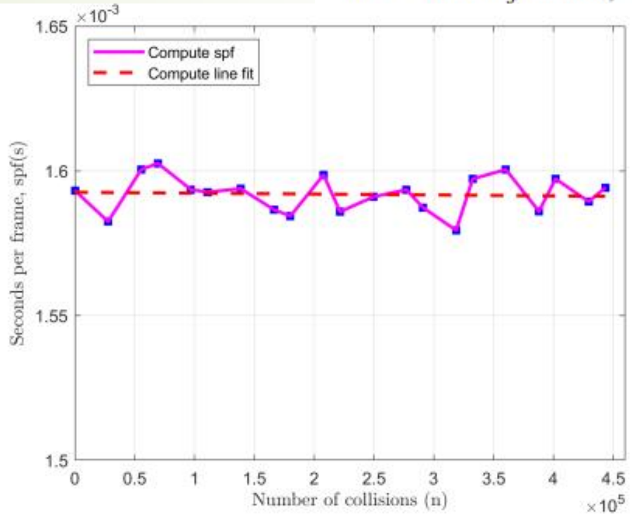


Figure 14. *Collision Fraction Benchmark (CFB), compute execution time versus number of collisions. For 443392 particles. The collision fraction,* $\boldsymbol{F}$ *varies from 0.0 to 1.0 in 0.05 increments and where particle-cell density,* $\boldsymbol{max_p}$ *is 8.*

Particle Cell Density Benchmark Format, Plots, Tables

MODULE 001 Performance data class

PCD – Particle Cell Density Benchmark

time	fps	cpums	cms	gms	expectedp	loadedp	shaderp_comp	shaderp_grph	expectedc	shaderc	threadcount	sidelen	density	PERR	CERR
0	14	0.071429	0.075328	0.001619	443392	443392	0	0	0	0	0	21	0	0	0
1	13	0.076923	0.075264	0.001645	443392	443392	0	0	0	0	0	21	0	0	0
2	13	0.076923	0.074712	0.001642	443392	443392	0	0	0	0	0	21	0	0	0

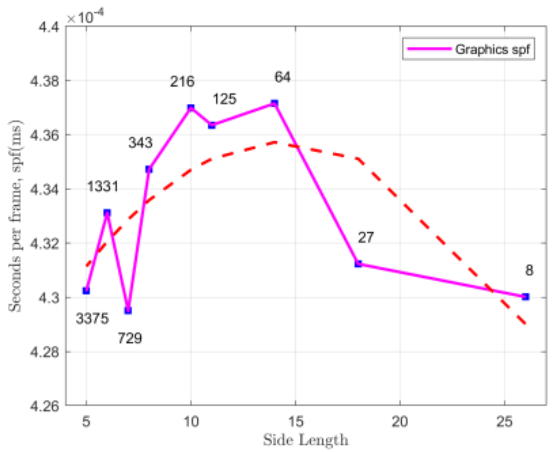


Figure 15. Particle Cell Density Benchmark graphics(PCB). Measures the execution time as the number of particles per cell increases. The numbered points are the particles per cell, $\boldsymbol{max_p}$. Execution time in seconds per frame versus the side length, $\boldsymbol{L_s}$. The side length squared equals the number of cells, $\boldsymbol{l_c}$. Table 3 lists the data points showing the number of particles per cell, $\boldsymbol{max_p}$, as the side length decreases. Graphics time, ***mgpt***, to process 117648 paricles, as particle per cell, $\boldsymbol{max_p}$, increase from 8 to 3375.

Table 3. *Particle radius versus particles per cell max_p.*

L_s	Radius R Range	max_p	$mcpt$(s)	$mgpt$(s)
91	0.16-0.2	8	0.03	0.02
61	0.11-0.15	27	0.07	0.02
46	0.91-0.11	64	0.15	0.02
37	0.071-0.09	125	0.27	0.02
31	0.061-0.07	216	0.47	0.02
27	0.051-0.06	343	0.71	0.02
21	0.041-0.05	729	1.44	0.02
18	0.031-0.04	1331	2.56	0.02
13	0.021-0.03	3375	6.26	0.02
9	0.011-0.02	12167	22.57	0.02

MODULE 001 Performance data class

PCD – Particle Cell Density Benchmark

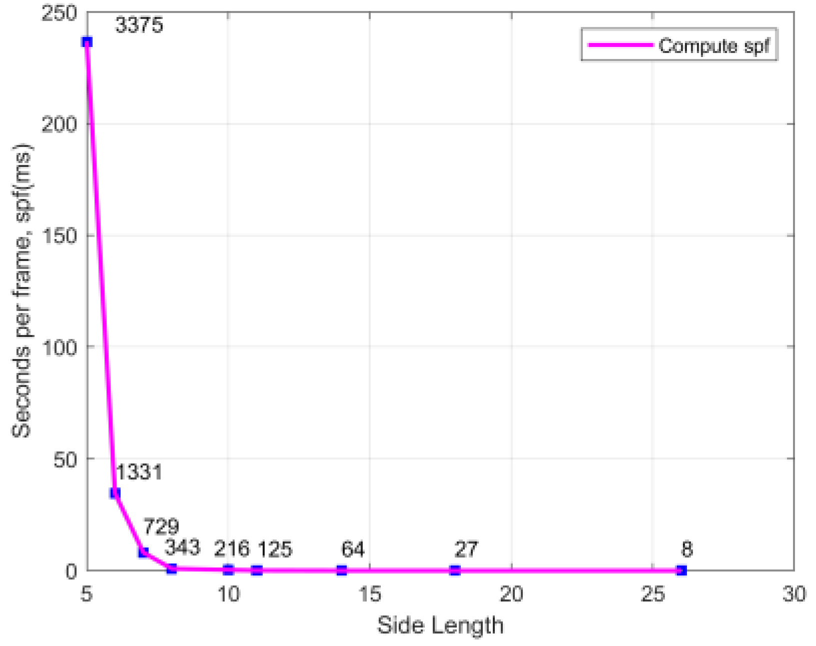


Figure 16. Particle Cell Density Benchmark compute (PCB). Measures the execution time as the number of particles per cell increases. The numbered points are the particles per cell, max_p Execution time in seconds per frame versus the side length, L_s. The side length squared equals the number of cells, l_c. Table 3 lists the data points showing the number of particles per cell, max_p, as the side length decreases. Compute time, $mgpt$, to process 117648 paricles, as particle per cell, max_p, increase from 8 to 3375.

Particle Quantity Benchmark Format, Plots, Tables

MODULE 001 Performance data class

PQB - Particle Quantity Benchmark

time	fps	cpums	cms	gms	expectedp	loadedp	shaderp_comp	shaderp_graph	expectedc	shaderc	threadcount	sidelen	density	PERR	CERR
0	5123	0.000195	7.04E-06	1.12E-05	32	32	0	0	16	0	0	3	0.5	0	0
1	6291	0.000159	6.34E-06	1.08E-05	32	32	0	0	16	0	0	3	0.5	0	0
2	6299	0.000159	7.23E-06	1.09E-05	32	32	0	0	16	0	0	3	0.5	0	0
3	6315	0.000158	7.07E-06	1.03E-05	32	32	0	0	16	0	0	3	0.5	0	0

Table 2. *Compute (**mcpt**) and graphics(**mgpt**), processing times. With maximum frames per second (**maxfps**), mimimum seconds per frame (**minspf**), application render rate. performance by number of particles where particle-cell density,* max_p *is 8, collision fraction,* F*, is 0.5, and* **mmrr** *is* 7.18×10^{-2} *ms.*

Particles in Dataset	Compute (Narrow) *mcpt* (*s*)	Graphics (Broad) *mgpt* (*s*)	*maxfps* (s^{-1})	*minspf* (*s*)	*arr*
32	6.14e-06	1.02e-05	6512	1.54e-04	46.7816
82944	3.70e-04	3.02e-04	1179	8.48e-04	8.4698
115712	5.15e-04	4.15e-04	891	1.12e-03	6.4009
246784	1.26e-03	8.81e-04	420	2.38e-03	3.0172
508928	2.71e-03	1.80e-03	206	4.85e-03	1.4799
754688	4.06e-03	2.68e-03	140	7.14e-03	1.0057
1000448	5.41e-03	3.54e-03	106	9.43e-03	0.7615
1246208	7.06e-03	4.44e-03	83	1.20e-02	0.5963
1360896	7.71e-03	4.85e-03	77	1.30e-02	0.5532
1508352	8.56e-03	5.36e-03	69	1.45e-02	0.4957
1623040	8.81e-03	5.72e-03	66	1.52e-02	0.4741
1754112	9.98e-03	6.25e-03	60	1.67e-02	0.4310
1901568	1.03e-02	6.70e-03	57	1.75e-02	0.4095
2016256	1.10e-02	7.10e-03	54	1.85e-02	0.3879
2163712	1.23e-02	7.65e-03	49	2.04e-02	0.3520
2343936	1.30e-02	8.26e-03	46	2.17e-02	0.3305
2540544	1.40e-02	8.91e-03	43	2.33e-02	0.3089
2638848	1.51e-02	9.29e-03	40	2.50e-02	0.2874
2753536	1.57e-02	9.74e-03	39	2.56e-02	0.2802
2900992	1.58e-02	1.02e-02	38	2.63e-02	0.2730

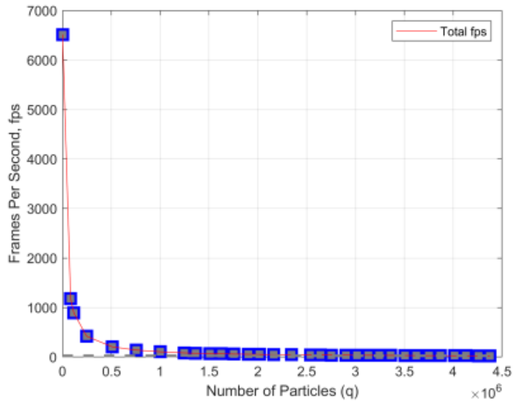


Figure 8. Frame per (*spf*) for 0.5 collision density with 8 particles per cell verses number of particles where particle-cell density, max_p is 8, collision fraction, F, is 0.5, and **mmrr** is4.35 x 10^{-3} ms

PQB - Particle Quantity Benchmark

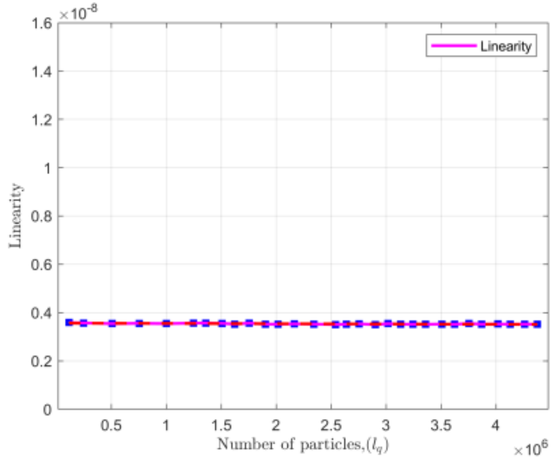


Figure 11. *Linearity of particle quantity benchmark graphics (PQB) where linearity is* $\frac{\mathbf{mgpt}}{q}$. *Linearity versus number of particles for graphics pipline,* ***mgpt****. The orange dashed line is the linear fit line. The fit equation is, grapics:* $-1.1 \times 10^{-17} q + 3.56 \times 10^{-9}$ *Conditions are particle-cell density,* $\boldsymbol{max_p}$ *is 8, collision fraction,* ***F****, is 0.5, and* ***mmrr*** *is* 7.18×10^{-5} *ms*

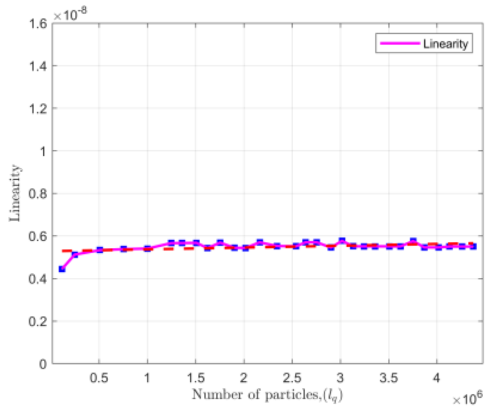


Figure 10. *Linearity of particle quantity benchmark (PQB) compute pipeline where linearity is* $\frac{\mathbf{mcpt}}{q}$. *Linearity versus number of particles for compute pipline,* ***mcpt****. The orange dashed line is the linear fit line. The fit equation is, compute: Linear fit: 8.50e-17q+5.28e-09 Conditions are particle-cell density,* $\boldsymbol{max_p}$ *is 8, collision fraction,* ***F****, is 0.5, and* ***mmrr*** *is* 7.18×10^{-5} *ms*