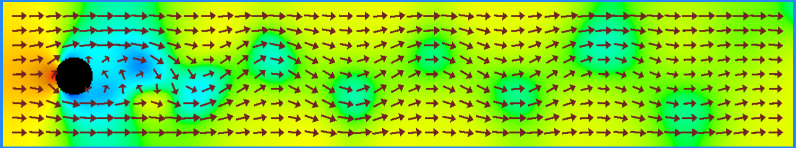


Optimized Visualization of Fluid Simulations

Samuel Stark - u1800081 - 10th March 2020

Disclaimer

- The scope of this project is *huge!*
- 8,500 lines of code over 146 files (not including comments, blank space, libraries)
- I can't talk about everything interesting in 15 minutes.
- This is going to be a whistle-stop tour of the best bits.
- Ask me anything after the presentation and I can talk your ear off.

Timestep calculations	Agnostic sim runners
CUDA Unified Memory	Origin-aware pointers
Parallel Reductions	CUDA Graphs
`const __restrict__`	Frame allocation
Image Layout Transfers	Vulkan Memory Model
CUDA Warps	Push Constants
Specialization Constants	Indirect Dispatch/Draw
Indexed Rendering	Semaphores
Fences	Vulkan Memory Allocation
Memory Alignment	Atomic Variables
and more!	

Table 1: Interesting things I could talk about

CFD, Simulations, and High-Speeds

- Equations modelling real-world phenomena have been around for centuries.
- Computational Fluid Dynamics programs (CFD) solve the Navier-Stokes equations to simulate fluid flow.
- Used in many fields:
  - Aerodynamics [Jameson et al. 2002]
  - Fire Spread Modelling [Sullivan 2009]
  - Entertainment Industry ['Fluid Dynamics on the Big Screen' 2008; Medvecký-Heretik Jakub 2018]
- Generally **interactive speeds** and **precise simulation** not pursued together.

Project Motivation

- CS257 coursework presented a fluid simulation from [Griebel et al. 1998], tasked students with optimizing it for a 6-core CPU.
- My solution [Stark 2020] ran 64x faster than the original, and 7.9x faster than real-time, on the given input data.
- But the simulation was still limited:
  - We were prevented from running it on a GPU for greater speedups.
  - Results could only be visualized after the fact, even though it was fast enough to render in real time.

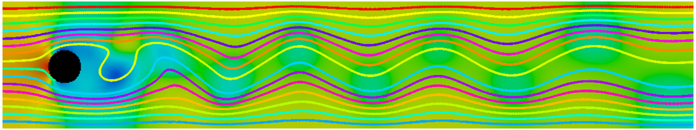
Project Goals/Achievements

Port the simulation to the GPU.

Exploit the speedup to improve accuracy and increase sim resolution.

Intuitively visualize the simulation in real time.[1]

All goals were achieved!



[1] Use games industry techniques for efficient rendering.

Table of Contents

Simulation Overview

- Simulation code preserved from CS257 submission.
- Simulates "laminar flows of viscous, incompressible fluids".
- Fluid is represented by a 2D array of cells.
- Fluid flows around static 'obstacle' cells.
- Generates values for velocity (u, v) and relative pressure p.

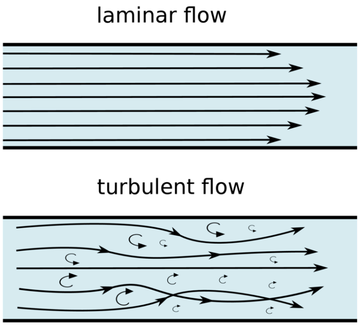


Figure 1: Laminar vs. turbulent fluid flow. Reproduced from cfdsupport.com

Simulation Structure

- Simulation runs in ‘ticks’, each representing a discrete timestep δt.
- Each ‘tick’ has multiple sequential execution stages.
- Each stage has been optimized to be embarassingly parallel.
- Poisson Solver runs for a constant amount of iterations each tick.

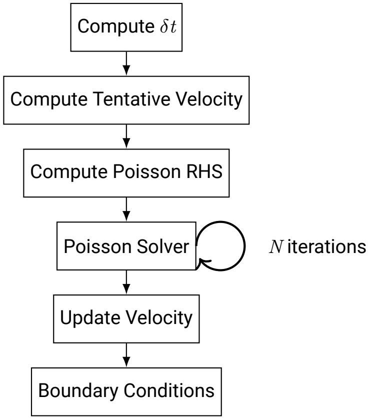


Figure 2: An example simulation tick

Simulation Kernels

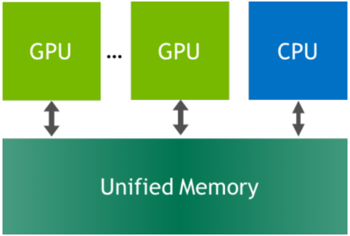
- This maps incredibly well to CUDA 'kernels'[2].
- Each stage is implemented as one or more kernels, run over every element in parallel.

```
// Computing delta-t is done slightly differently (ask me about it at the end!)

__global__ void computeTentativeVelocity_apply(...);
__global__ void computeTentativeVelocity_postproc_vertical(...);
__global__ void computeTentativeVelocity_postproc_horizontal(...);

__global__ void computeRHS_1per(...);

__global__ void poisson_single_tick(...);

__global__ void updateVelocity_1per(...);

__global__ void boundaryConditions_preproc_vertical(...);
__global__ void boundaryConditions_preproc_horizontal(...);
__global__ void boundaryConditions_apply(...);
__global__ void boundaryConditions_inputflow_west_vertical(...);
```

[2] https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#kernels

CUDA Unified Memory

- CUDA provides Unified Memory allocations[3]
- Paged between the Host and Device on-demand.
- Same performance as normal GPU memory when present on the device.
- Used to mix CPU and GPU implementations while testing and debugging.



[3] https://developer.nvidia.com/blog/unified-memory-cuda-beginners/

`const __restrict__` pointers

- CUDA exposes a fast "read-only data cache"[4].
- To ensure the compiler knows memory is read only, use the `const` and `__restrict__` qualifiers on all pointers.
- Shown to speed up execution times in [Diarra 2018].

```
template<typename T>
using in_matrix =
    const T* const __restrict__;

template<typename T>
using out_matrix =
    T* const __restrict__;
```

Figure 3: Helper templates used in kernel definitions

Ask me about `const __restrict__` pointers at the end!

[4] https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#global-memory-3-0

Parallel Reductions

- Computing δt requires the maximum values of u, v.
- We can do this in parallel on the GPU!
- Find the values on the GPU, then copy them to the CPU to calculate δt.
- Implementation taken from [Harris n.d.].

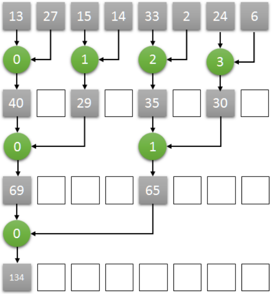
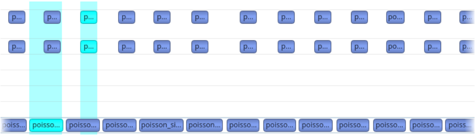


Figure 4: Example of parallel reduction for sum. Reproduced from eximiaco.tech

CUDA Graphs

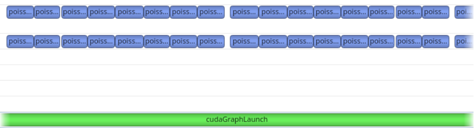
- CPU overhead when launching Poisson kernels caused large GPU bubbles.
- Instead of launching N times, record a CUDA Graph[5] that runs N iterations, and launch it once.
- Theoretical 2x speedup.

```
for (int i = 0; i < 100; i++) {
    launch poisson on stream;
}
```



Individual Launches

```
(record poisson100Iters if not present)

cudaGraphLaunch(poisson100Iters, stream);
```



With CUDA Graphs

[5] https://developer.nvidia.com/blog/cuda-graphs/

Table of Contents

Visualization Research I

- This program is an example of ‘tightly-coupled in-situ visualization’ [Kress 2017].
- Academia hasn’t recently innovated in fluid visualization, only in methods for running faster such as [Shyh-Kuang Ueng et al. 1996].
- This was noted in [Gaither 2004], which states ‘feature detection’ would be a key element going forward rather than new visualization methods.

Visualization Research II

- Industry seems to match this assessment.
- Tools such as Autodesk CFD, Tecplot, ParaView all visualize data with the same general methods...
- but they allow the data to be *filtered* to extract relevant values.
- Methods can be combined to show a range of information.

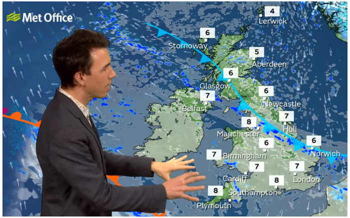


Figure 5: Weather Forecast showing wind speed, weather fronts, and cloud cover.[6]

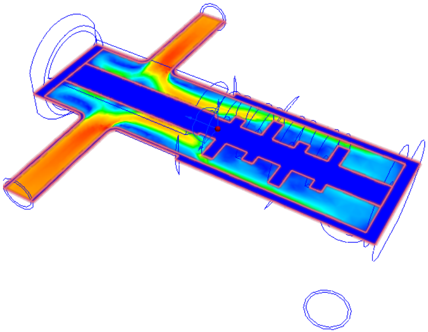
[6] https://youtu.be/y_1--MkiNjQ, Met Office 10 Day Trend for March 3rd.

Visualization Research

What can Autodesk CFD do?

Result Planes - Scalar

- Place a plane in 3D space
- Select a scalar quantity (pressure, temperature etc.)
- The cross-section of the model shows the selected quantity, with a color scale

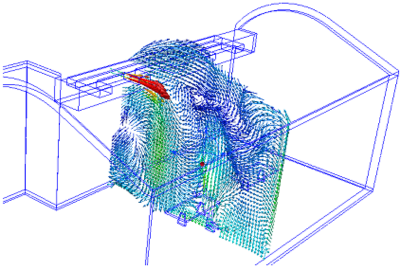


Visualization Research

What can Autodesk CFD do?

Result Planes - Vector

- Place a plane in 3D space
- Select a *vector* quantity (velocity etc.)
- The cross-section of the model shows a vector field of the selected velocity.

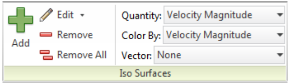


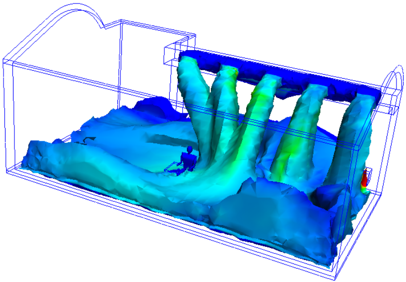
Visualization Research

What can Autodesk CFD do?

Isosurfaces

- Select a scalar quantity X.
- Select a value $X = x$.
- This surface is displayed with a color based on another quantity Y.
- A vector quantity can also be added to the surface.



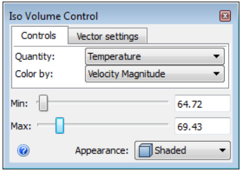


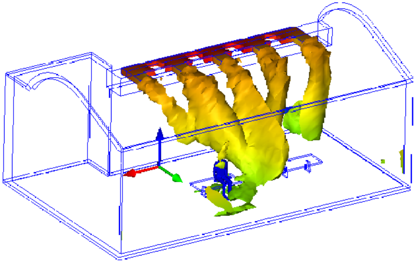
Visualization Research

What can Autodesk CFD do?

Isovolumes

- Select a scalar quantity X.
- Select a *range* $x_{min} \leq X \leq x_{max}$.
- This *volume* is displayed with a color based on another quantity Y.
- A vector quantity can also be added to the *volume*.



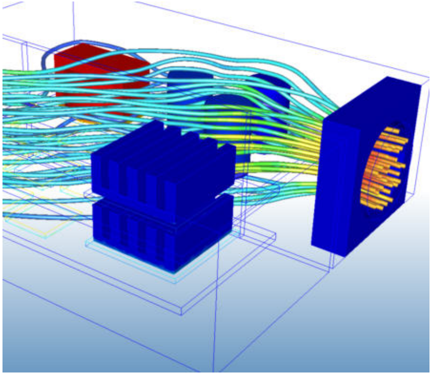


Visualization Research

What can Autodesk CFD do?

Particles

- Place particle spawn points (‘seeds’).
- Select a scalar quantity to display, or a solid color.
- Points along the particle paths show the specified quantity.
- Can choose many kinds of path:
  - Cylinders
  - Ribbons
  - Comets
  - etc.

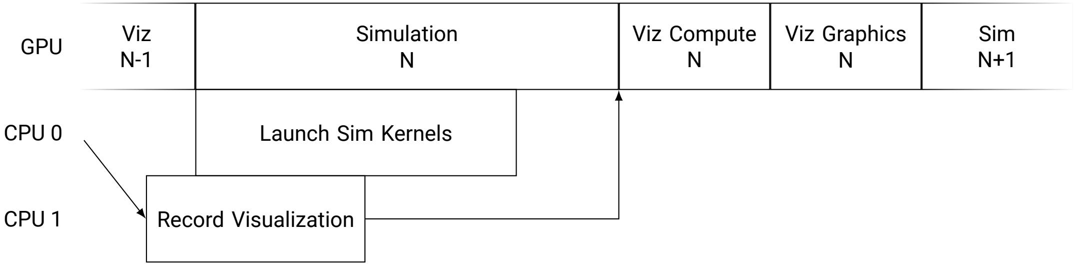


Selected Features

Separate the visualization into layers:

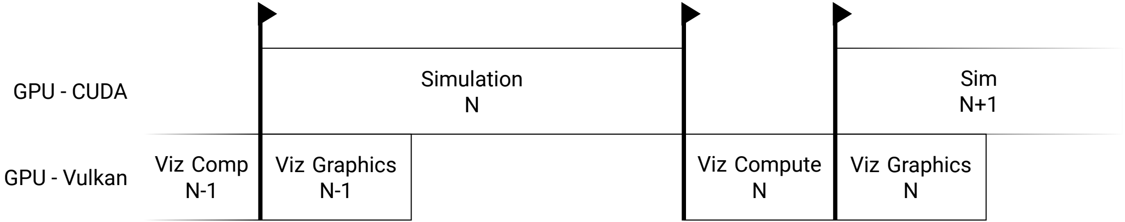
- Background
- Scalar Quantity
  - Display a quantity X using a colormap when $x_{min} \leq X \leq x_{max}$
  - Allow the user to select a range, or calculate a range containing all values
  - Equivalent to Results Plane (Scalar) + 2D Isovolume
- Vector Quantity
  - Display a vector field of X when $x_{min} \leq X \leq x_{max}$
  - Allow the user to select a range, or calculate a range containing all values
  - Equivalent to Results Plane (Vector) + 2D Isovolume
- Particles
  - Editable ‘seeds’
  - Planned for particle trace options, didn’t have time.

Anatomy of a Frame



- CPU 0 launches the simulation, which requires some CPU/GPU sync at the start.
- CPU 1 enqueues the visualization work to start right after the simulation.
- Sim and Visualization share memory, architecture is zero-copy.
- Maintains near-100% GPU Utilization.

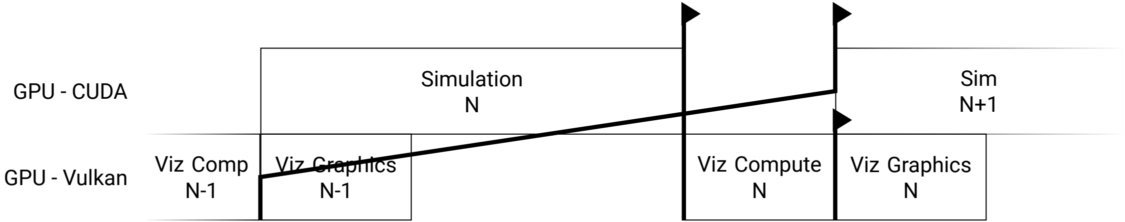
GPU Synchronization



- Synchronization between overall workloads is performed via *semaphores*[7].
- One workload waits on a semaphore until another workload signals it.
- Compute workloads cannot overlap on my graphics card[8]
- Simulation and Viz Graphics *could* overlap, but don't in practice.

[7] `https://www.khronos.org/registry/vulkan/specs/1.2-extensions/man/html/VkSemaphore.html`
[8] Running parallel compute workloads was introduced in [*NVIDIA AMPERE GA102 GPU ARCHITECTURE* 2020]

GPU Synchronization - Less Misleading



- Synchronization between overall workloads is performed via *semaphores*[9].
- One workload waits on a semaphore until another workload signals it.
- Compute workloads cannot overlap on my graphics card[10]
- Simulation and Viz Graphics *could* overlap, but don't in practice.

[9] `https://www.khronos.org/registry/vulkan/specs/1.2-extensions/man/html/VkSemaphore.html`
[10] Running parallel compute workloads was introduced in [*NVIDIA AMPERE GA102 GPU ARCHITECTURE* 2020]

Extracting Simulation Data

- First part of Viz Compute.
- Transfer + interpolate data from 1D arrays to a 2D texture.
- More complex than a simple copy.
- Allows arbitrary sampling, using built-in texture filtering for free interpolation.

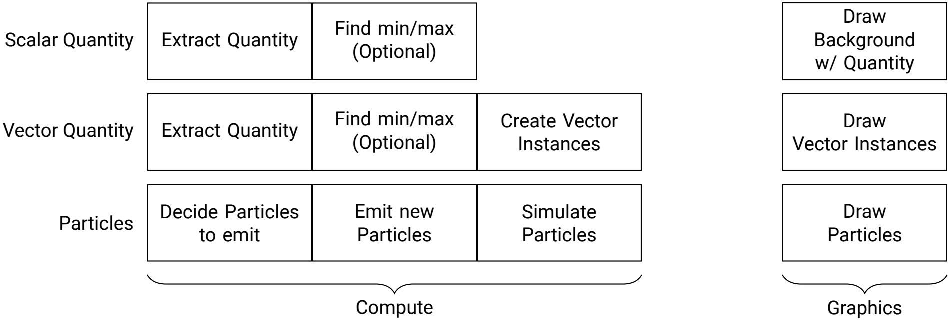
```
float u[], v[], p[], isfluid[];

int idx = i * pConsts.height + j;
vec2 velocity = vec2(u[idx], v[idx]);
```

↓

```
uniform sampler2D simDataSampler;
 // = (u, v, p, isfluid);

 // 50% across, 20% up the image
vec2 sampleAt = (0.5, 0.2);
vec2 velocity =
    texture(simDataSampler, sampleAt).xy;
```

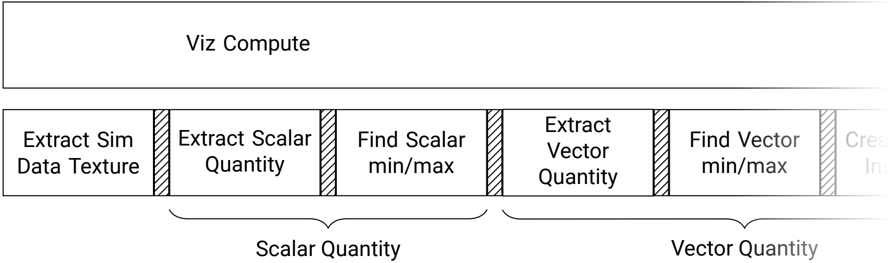
Ask me about Simulation Data Textures at the end!

Per-Layer Viz Work



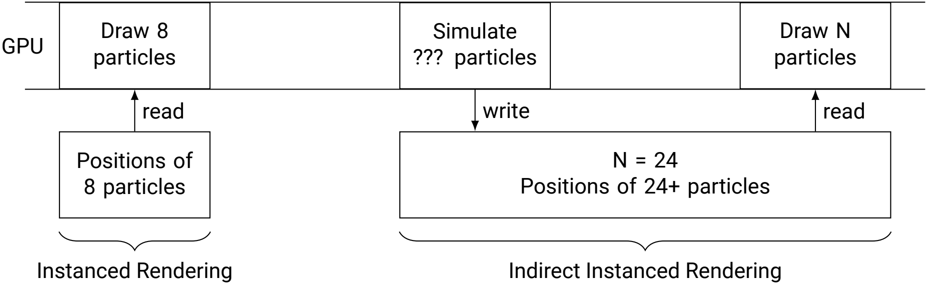
- Compute Pipelines use one Compute Shader, roughly equivalent to CUDA Kernels.
- Graphics Pipelines use a Vertex Shader and a Fragment Shader to draw to a render target.
- There is also a ‘final composite’ stage which renders the GUI with the viz output.

Viz Compute Order



- Computer work for layers is done serially, not in parallel (which could be improved in the future).
- Vulkan uses Execution and Memory Barriers to ensure ordering. (Ask me about this at the end!)
- Vectors and Particles are drawn with Indirect Instanced rendering.

Indirect Instanced Rendering



- We don’t know how many Vectors/Particles exist at record time.
- Tell the GPU to look somewhere in memory to find how many copies to render.

Ask me about indirect/instanced/indexed rendering at the end!

Result!

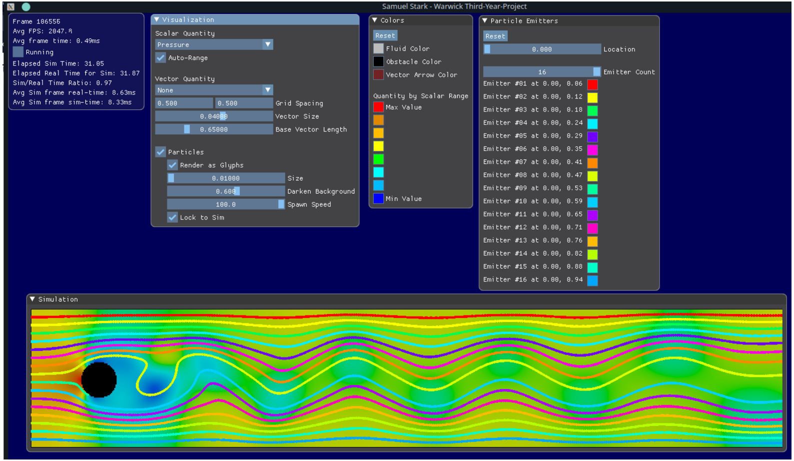
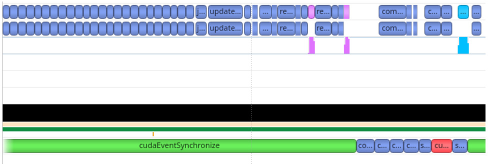

Samuel Stark - Warwick Third-Year-Project
Frame 106556
Avg FPS: 2047.8
Avg frame time: 0.49ms
Running
Elapsed Sim Time: 31.05
Elapsed Real Time for Sim: 31.87
Sim/Real Time Ratio: 0.97
Avg Sim frame real-time: 8.63ms
Avg Sim frame sim-time: 8.33ms
Visualization
Scalar Quantity
Pressure
Auto-Range
Vector Quantity
None
0.500
0.500
Grid Spacing
0.04000
Vector Size
0.65000
Base Vector Length
Particles
Render as Glyphs
0.01000
Size
0.600
Darken Background
100.0
Spawn Speed
Lock to Sim
Colors
Reset
Fluid Color
Obstacle Color
Vector Arrow Color
Quantity by Scalar Range
Max Value
Min Value
Particle Emitters
Reset
0.000
Location
16
Emitter Count
Emitter #01 at 0.00, 0.06
Emitter #02 at 0.00, 0.12
Emitter #03 at 0.00, 0.18
Emitter #04 at 0.00, 0.24
Emitter #05 at 0.00, 0.29
Emitter #06 at 0.00, 0.35
Emitter #07 at 0.00, 0.41
Emitter #08 at 0.00, 0.47
Emitter #09 at 0.00, 0.53
Emitter #10 at 0.00, 0.59
Emitter #11 at 0.00, 0.65
Emitter #12 at 0.00, 0.71
Emitter #13 at 0.00, 0.76
Emitter #14 at 0.00, 0.82
Emitter #15 at 0.00, 0.88
Emitter #16 at 0.00, 0.94
Simulation

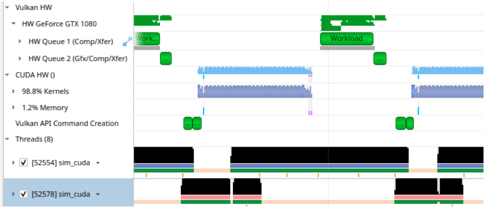
Table of Contents

GPU Utilization

- GPU Utilization is close to 100% where possible.
- At tick boundaries some bubbles appear as the CPU calculates the next δt.
- When visualizing, the Vulkan work hides this.



Tick Boundary



Overall Visualization Pipeline

Speed

- Simulates the original CS257 input 2.47-2.86x faster than the original code.
- Visualization takes 1.35ms per frame (740 FPS) at highest iteration count $N = 1000$
- Individual visualization features are quick, and combined take less time than the simulation.[11]

	Base Frame	with Sim	Scalar Quantity	Vector Field	Particles
Mean Time (ms)	0.30	1.18	0.39	0.46	0.42
$\triangle$ from base (ms)	-	$+$0.88	$+$0.09	$+$0.16	$+$0.12

[11] All points measured here in worst-case: with auto-range on where possible, and with maximum particles onscreen.

Difference vs. Original

- The program contains a comparison tool for checking similarity.
- Simulating the original CS257 test has a mean square error of 10^{-14} for velocities, and 10^{-9} for pressure.
- As iteration count and simulation time increases, the error becomes larger.
- Multiple potential causes in algorithm and implementation, but haven't researched further.

N	100	200	300	1000
Velocity MSE (u,v)	10^{-14}	10^{-14}	10^{-14}	10^{-14}
Pressure MSE (p)	10^{-9}	10^{-8}	10^{-7}	10^{-6}

Mean Square Error for original CS257 input data, simulated for 10 s

Table of Contents

Project Management

- Schedule defined as part of the Specification, planned for coding and writing reports.
- Code Freeze on Week 22 was very helpful
- Gave me enough time to finish the presentation!

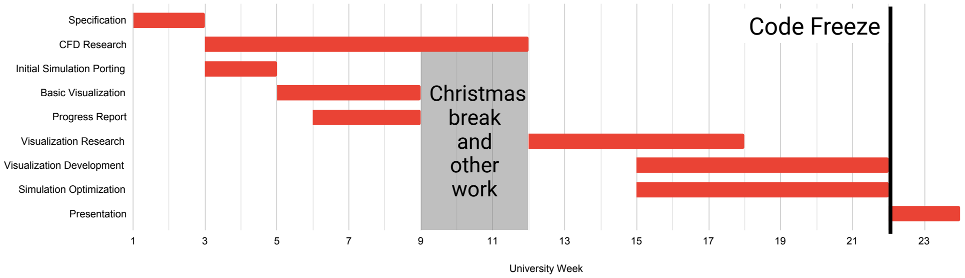


Table of Contents

Conclusion

- Overall, the project was a success.
- CUDA is a very intuitive API, especially for those without prior compute experience.
- Vulkan requires more heavy lifting, but it seems to have been worth it.
- Looking to the games industry for advice in i.e. particle rendering is helpful.
- For the scientific community to start using Vulkan, simple abstraction layers will be needed.
  - VTK, a popular visualization library, has a Vulkan branch that seems to be dead.
  - Datoviz is a new library with Python bindings that renders with Vulkan.
- CUDA-Vulkan interoperability is nice! Resources should be allocated from Vulkan to maintain full control.

Future Work

Simulation

- Investigate simulation accuracy and algorithm.
- Re-introduce the Poisson accuracy check.
- Optimize parallel reductions.

Visualization

- Investigate colorblindness options.
- Better memory allocation, potentially using a helper library.
- Run different layer computations in parallel with separate command buffers?

Demo + Questions

References I

‘Fluid Dynamics on the Big Screen’. In: *ANSYS Advantage* II.2 (2008), pp. 52–53. URL: https://www.ansys.com/-/media/ansys/corporate/resourcelibrary/article/aa-v2-i2-fluid-dynamics-on-big-screen.pdf.

NVIDIA AMPERE GA102 GPU ARCHITECTURE. Tech. rep. 2020. URL: https://www.nvidia.com/content/dam/en-zz/Solutions/geforce/ampere/pdf/NVIDIA-ampere-GA102-GPU-Architecture-Whitepaper-V1.pdf (visited on 22/02/2021).

Rokiatou Diarra. ‘Towards Automatic Restrictification of CUDA Kernel Arguments’. In: *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*. ASE 2018. New York, NY, USA: Association for Computing Machinery, 2018, pp. 928–931. ISBN: 9781450359375. DOI: 10.1145/3238147.3241533. URL: https://doi.org/10.1145/3238147.3241533.

References II

K. Gaither. 'Visualization's role in analyzing computational fluid dynamics data'. In: *IEEE Computer Graphics and Applications* 24.3 (2004), pp. 13–15. DOI: 10.1109/MCG.2004.1297005.

Michael Griebel, Thomas Dornseifer and Tilman Neunhoeffer. *Numerical simulation in fluid dynamics: a practical introduction*. SIAM, 1998.

Mark Harris. *Optimizing Parallel Reduction in CUDA*. Tech. rep. URL: https://developer.download.nvidia.com/assets/cuda/files/reduction.pdf.

Antony Jameson, Luigi Martinelli and J Vassberg. 'Using computational fluid dynamics for aerodynamics–a critical assessment'. In: *Proceedings of ICAS*. 2002, pp. 2002–1.

James Kress. *In Situ Visualization Techniques for High Performance Computing*. 2017. URL: www.cs.uoregon.edu/Reports/AREA-201703-Kress.pdf (visited on 06/03/2021).

References III

Medvecký-Heretik Jakub. ‘Real-time Water Simulation in Game Environment’. PhD thesis. Masaryk University, Faculty of Informatics, 2018.

Shyh-Kuang Ueng, C. Sikorski and Kwan-Liu Ma. ‘Efficient streamline, streamribbon, and streamtube constructions on unstructured grids’. In: *IEEE Transactions on Visualization and Computer Graphics* 2.2 (1996), pp. 100–110. DOI: 10.1109/2945.506222.

S. Stark. ‘CS257 Report - Reducing the Execution Time of a Fluid Simulation Program’. In: (2020).

Andrew L. Sullivan. ‘Wildland surface fire spread modelling, 1990 - 2007. 1: Physical and quasi-physical models’. In: *International Journal of Wildland Fire* 18.4 (2009), p. 349. ISSN: 1049-8001. DOI: 10.1071/wf06143. URL: http://dx.doi.org/10.1071/WF06143.

Simulation Data Texture

- Simulation stores data points from a staggered grid.
- Visualization wants to get data at arbitrary locations, which texture hardware is really good at.
- Convert the original data to a texture 2x the resolution, and interpolate when values aren't present.