Sparse Fluid Simulation in DirectX

Alex Dunn
Dev. Tech. – NVIDIA
adunn@nvidia.com

Agenda

- We want more fluid in games ☺
- Eulerian (grid based) fluid.
- Sparse Eulerian Fluid.
- Feature Level 11.3 Enhancements!

- (Not a talk on fluid dynamics)

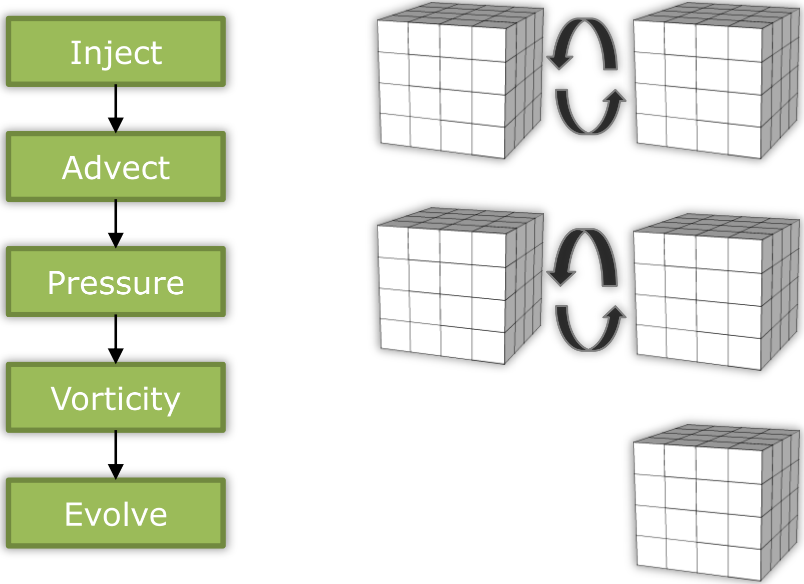
Why Do We Need Fluid in Games?

- Replace particle kinematics!
  - more realistic == better immersion
- Game mechanics?
  - occlusion
    - smoke grenades
    - interaction
  - Dispersion
    - air ventilation systems
    - poison, smoke

- Endless opportunities!

Eulerian Simulation #1

My (simple) DX11.0 eulerian fluid simulation:

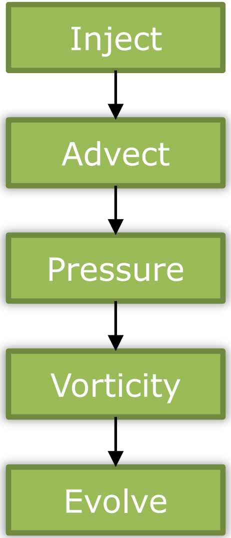


2x Velocity

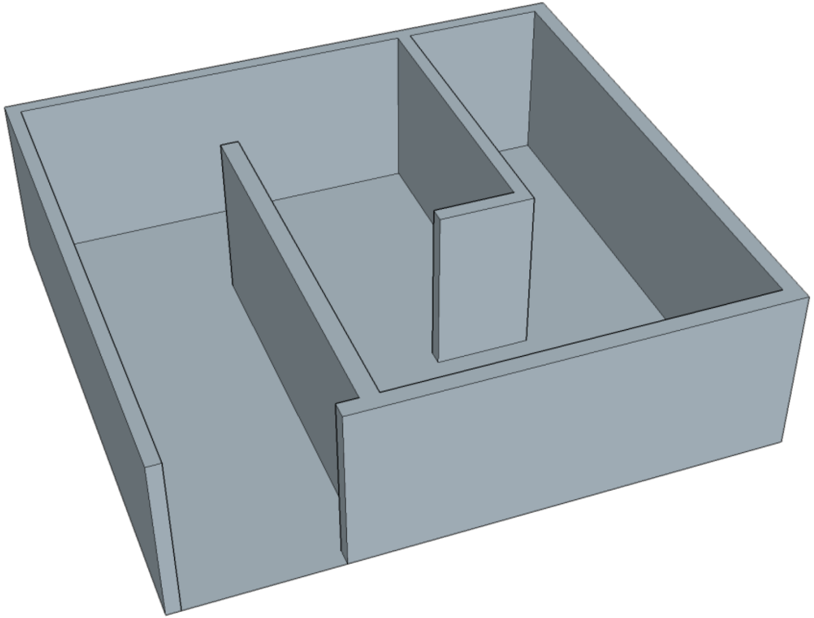
2x Pressure

1x Vorticity

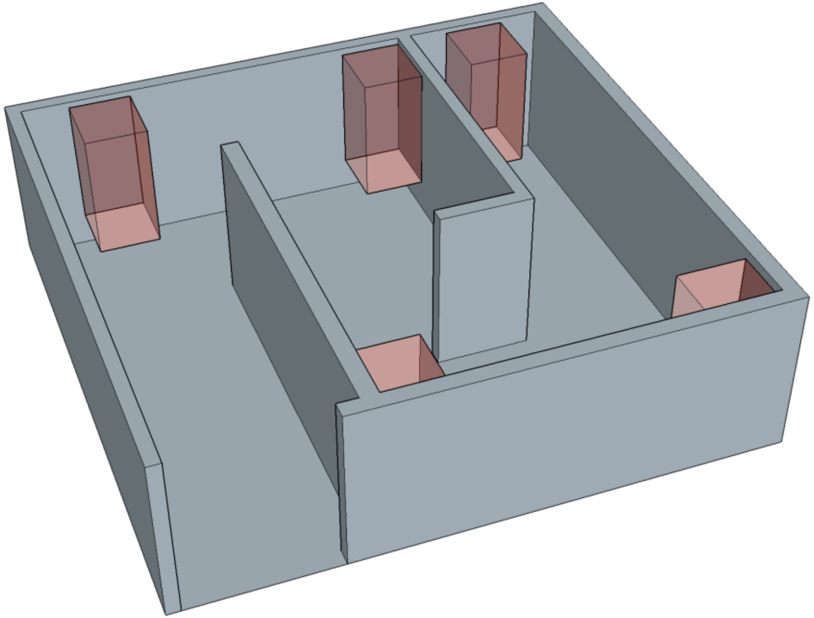
Eulerian Simulation #2



- Add fluid to simulation
- Move data at, XYZ → (XYZ+Velocity)
- Calculate localized pressure
- Calculates localized rotational flow
- Tick Simulation

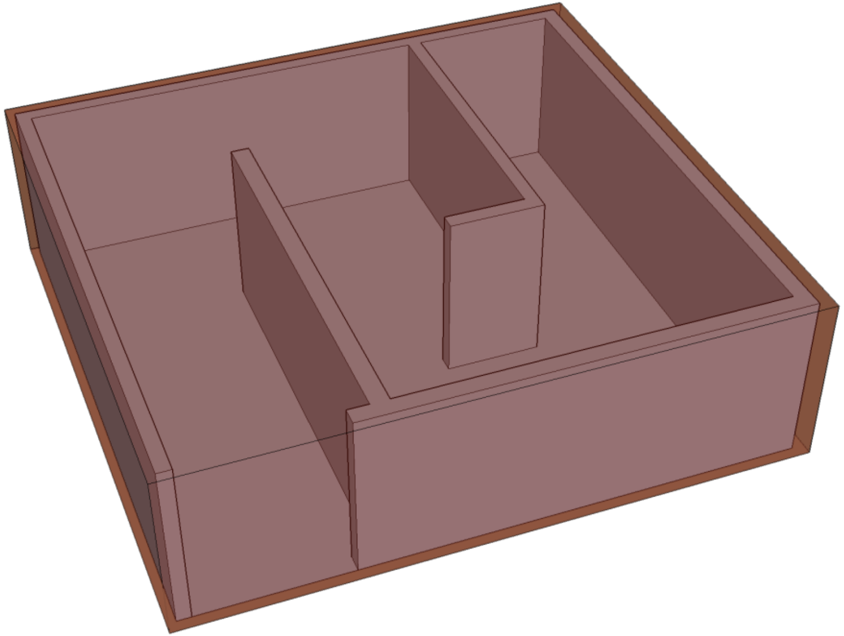


(some imagination required)



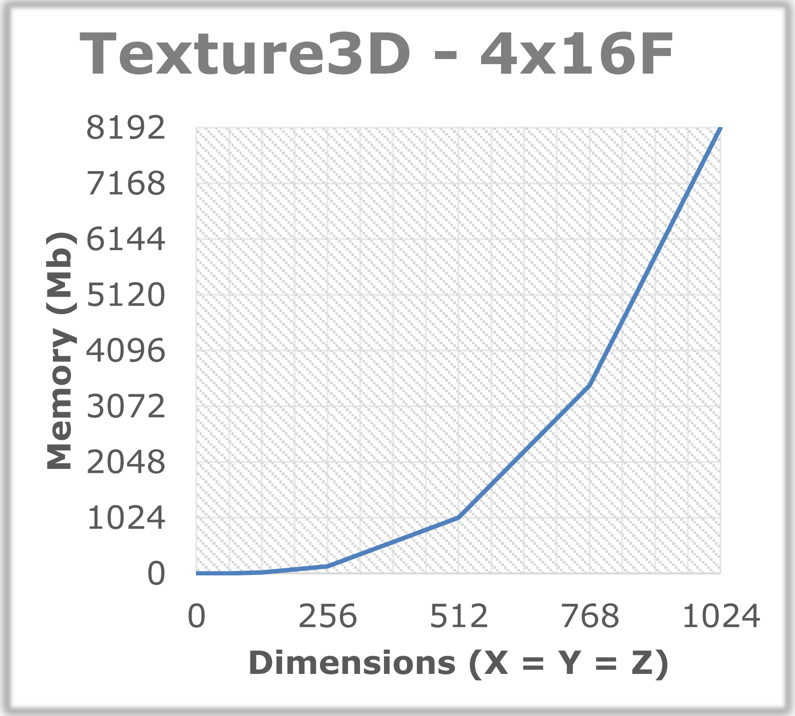
Too Many Volumes Spoil the…

- Fluid isn't box shaped.
  - clipping
  - wastage
- Simulated separately.
  - authoring
  - GPU state
  - ~~volume-to-volume interaction~~
- Tricky to render.



Problem!

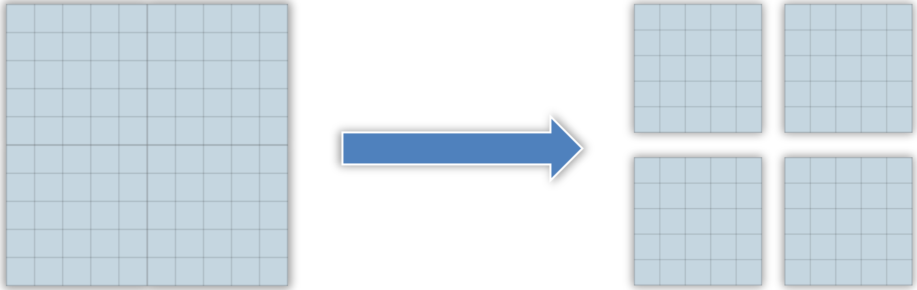
- N-order problem
  - 64^3 = ~0.25m cells
  - 128^3 = ~2m cells
  - 256^3 = ~16m cells
  - …
- Applies to:
  - computational complexity
  - memory requirements



And that's just 1 texture…

Bricks

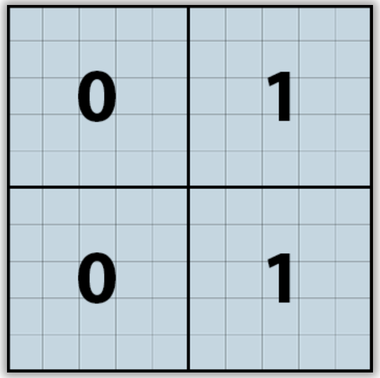
- Split simulation space into groups of cells (each known as a brick).
- Simulate each brick independently.



Brick Map

- Need to track which bricks contain fluid

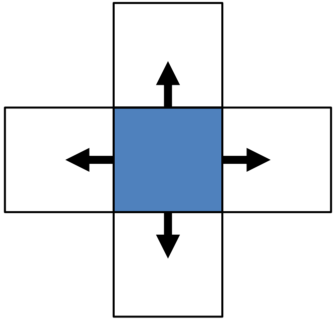
- Texture3D<uint>
- 1 voxel per brick
  - 0 → Unoccupied
  - 1 → Occupied



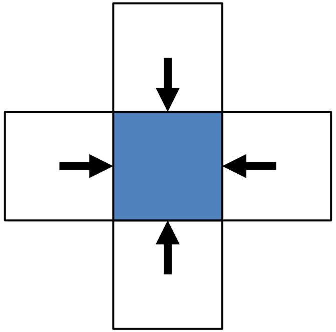
- *Could also use packed binary grids [Gruen15], but this requires atomics ☹*

Tracking Bricks

- Initialise with emitter



- Expansion (*unoccupied* → *occupied*)
  - if { $V_{|x|y|z|} > |D_{brick}|$ }
  - expand in that axis



- Reduction (*occupied* → *unoccupied*)
  - inverse of Expansion
  - handled automatically

Sparse Simulation

- Clear Tiles
- Inject
- Advect
- Pressure
- Vorticity
- Evolve*
- Fill List

Reset all tiles to 0 (unoccupied) in brick map.

Read value from brick map.

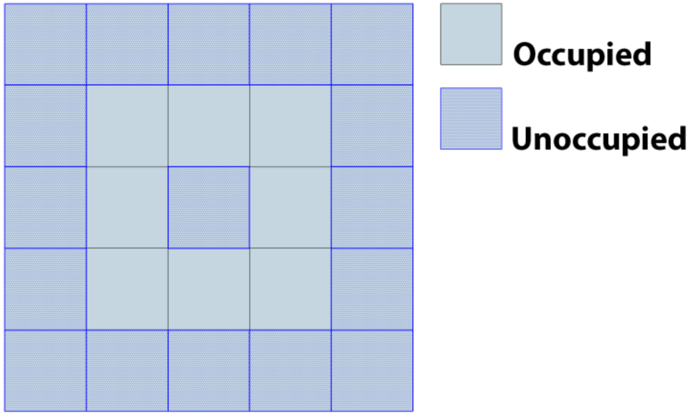
Append brick coordinate to list if occupied.

```
Texture3D<uint> g_BrickMapRO;
AppendStructredBuffer<uint3> g_ListRW;

if(g_BrickMapRO[idx] != 0)
{
      g_ListRW.Append(idx);
}
```

Includes expansion

Uncompressed Storage



Allocate everything; forget about unoccupied cells ☹

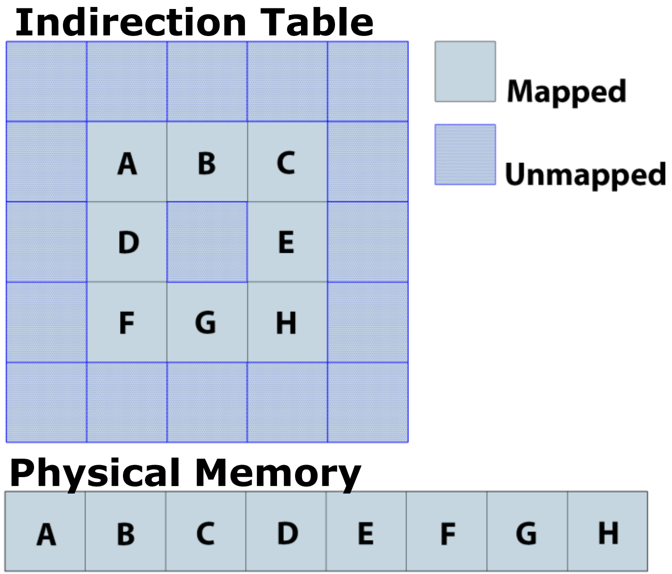
Pros:

- simulation is coherent in memory.
- works in DX11.0.

Cons:

- no reduction in memory usage.

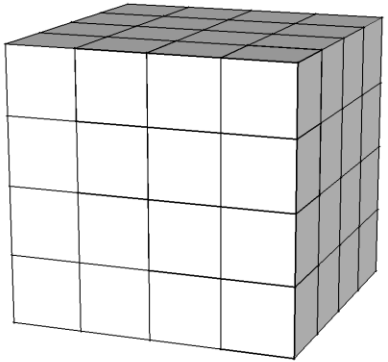
Compressed Storage



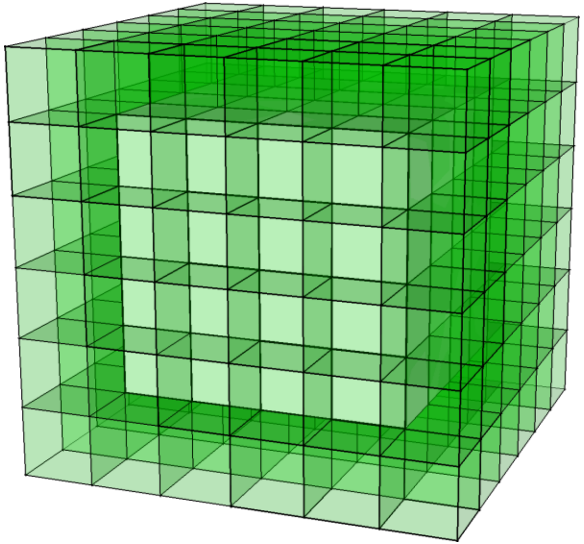
Similar to, List<Brick>

Pros:
- good memory consumption.
- works in DX11.0.

Cons:
- allocation strategies.
- indirect lookup.
  - "software translation"
  - filtering particularly costly



1 Brick = $(4)^3$ = 64



$1 \text{ Brick} = (1+4+1)^3 = 216$

- New problem;
- "$6n^2 + 12n + 8$" problem.

Can we do better?

Enter; Feature Level 11.3

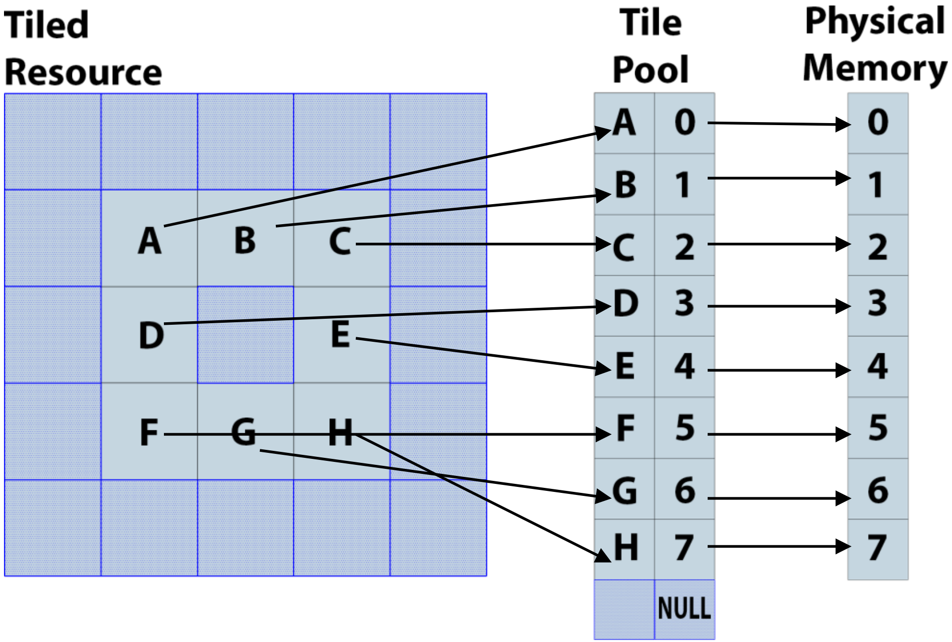
- Volume Tiled Resources (VTR)! ☺
- Extends 2D functionality in FL11.2
- Must check HW support: **(DX11.3 != FL11.3)**

```
ID3D11Device3* pDevice3 = nullptr;
pDevice->QueryInterface(&pDevice3);

D3D11_FEATURE_DATA_D3D11_OPTIONS2 support;
pDevice3->CheckFeatureSupport(D3D11_FEATURE_D3D11_OPTIONS2,
                              &support,
                              sizeof(support));

m_UseTiledResources = support.TiledResourcesTier ==
                              D3D11_TILED_RESOURCES_TIER_3;
```

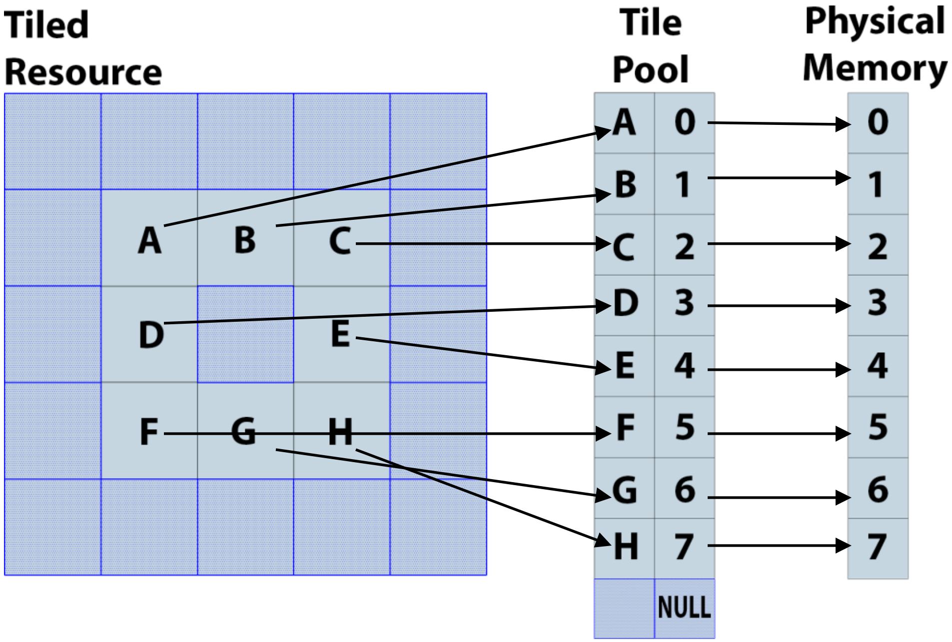
Tiled Resources #1



Pros:

- only mapped memory is allocated in VRAM
- “hardware translation”
- logically a volume texture
- all samplers supported
- 1 Tile = 64KB (= 1 Brick)
- fast loads

Tiled Resources #2



1 Tile = 64KB (= 1 Brick)

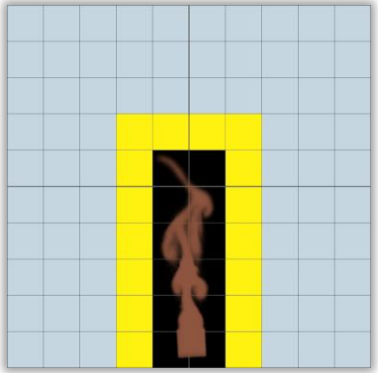
BPP	Tile Dimensions
8	64x32x32
16	32x32x32
32	32x32x16
64	**32x16x16**
128	16x16x16

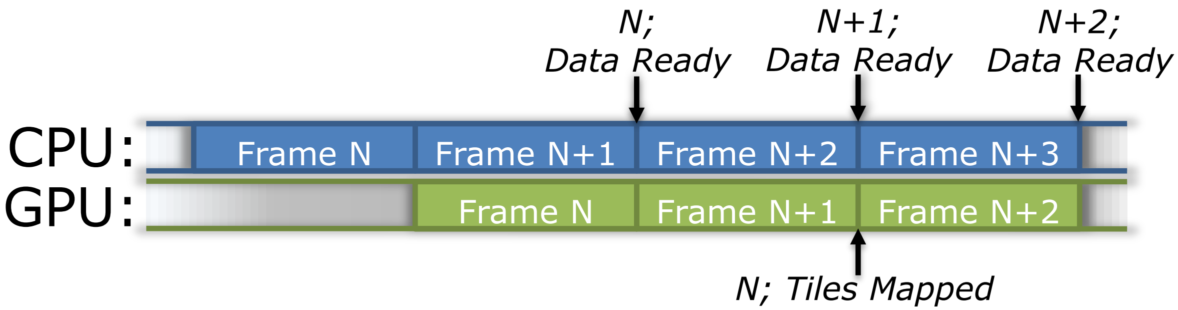
Gotcha: *Tile mappings must be updated from CPU*

Latency Resistant Simulation #1

Naïve Approach:

- **clamp velocity to V_{max}**
- CPU Read-back:
  - occupied bricks.
  - <u>2 frames of latency!</u>
- extrapolate "probable" tiles.

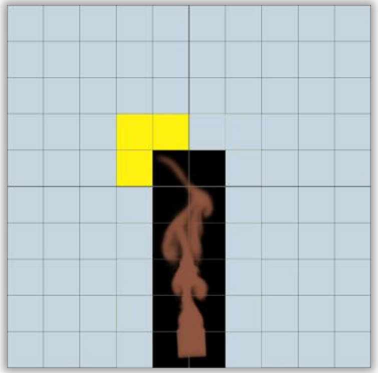


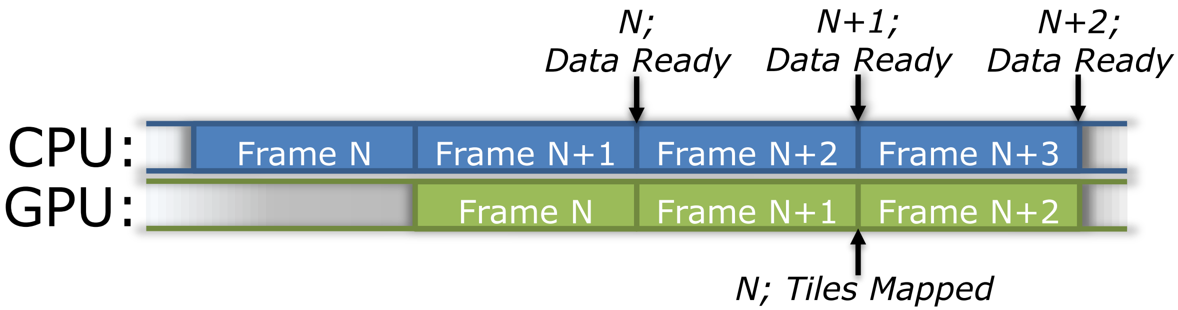


Latency Resistant Simulation #2

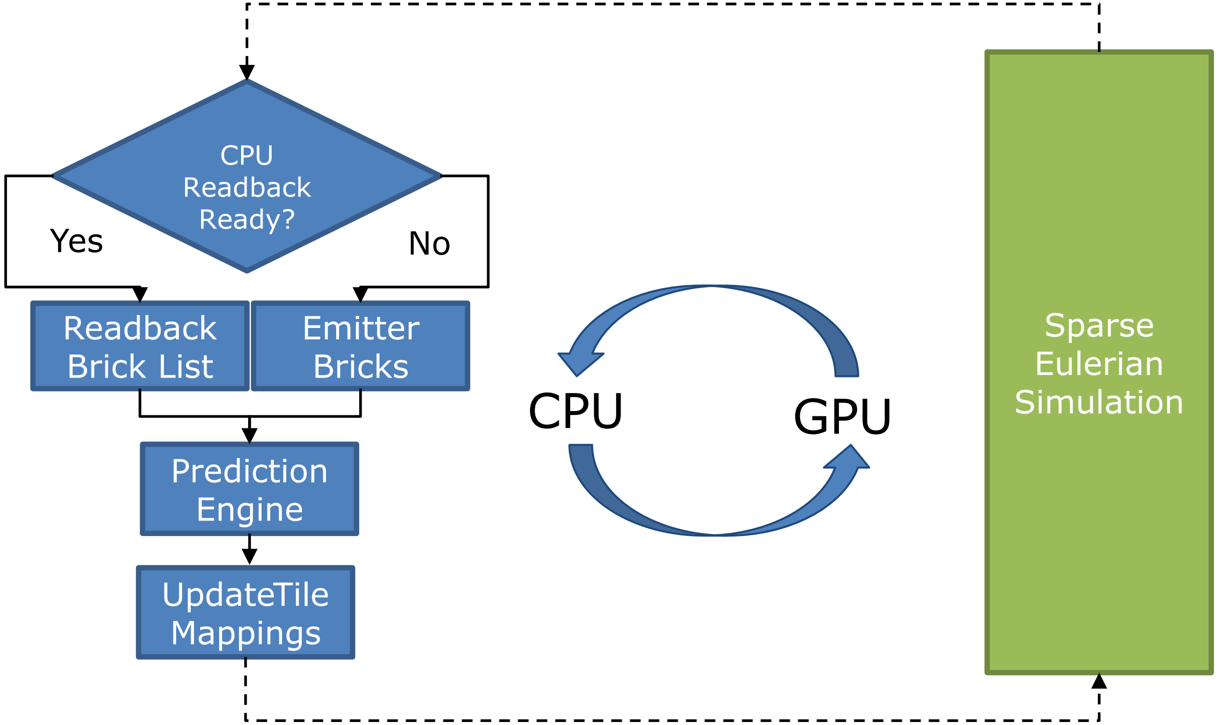
Tight Approach:

- CPU Read-back:
  - occupied bricks.
  - **max{|V|} within brick.**
  - 2 frames of latency!
- extrapolate "probable" tiles.



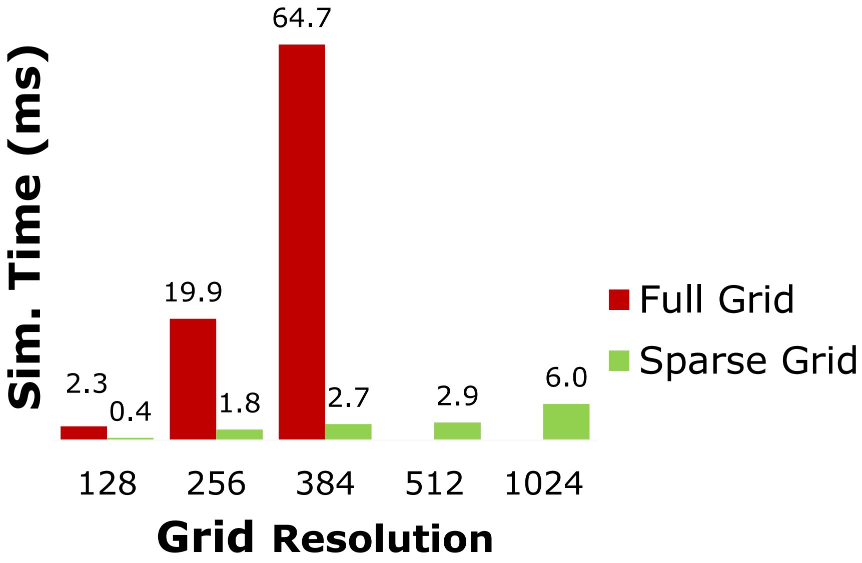


Latency Resistant Simulation #3



Demo

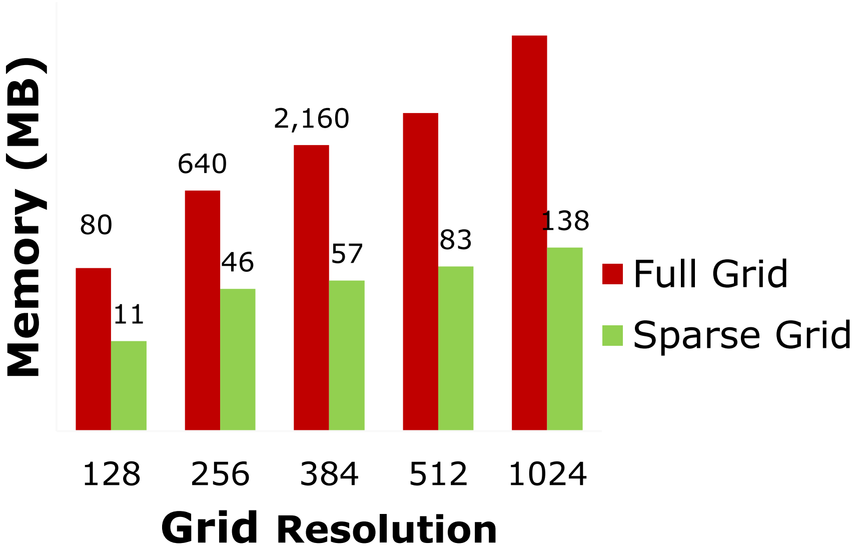
Performance #1



	Grid Resolution				
	128^3	256^3	384^3	512^3	$1,024^3$
	Full Grid				
Num. Bricks	256	2048	6,912	16,384	131,072
Memory (MB)	80	640	2,160	5,120	40,960
Simulation	2.29ms	19.04ms	64.71ms	NA	NA
	Sparse Grid				
Num. Bricks	36	146	183	266	443
Memory (MB)	11.25	45.63	57.19	83.13	138.44
Simulation	0.41ms	1.78ms	2.67ms	2.94ms	5.99ms
Scaling Sim.	78.14%	76.46%	75.01%	NA	NA

NOTE: Numbers captured on a GeForce GTX980

Performance #2



	Grid Resolution				
	128^3	256^3	384^3	512^3	$1{,}024^3$
	Full Grid				
Num. Bricks	256	2048	6,912	16,384	131,072
Memory (MB)	80	640	2,160	5,120	40,960
Simulation	2.29ms	19.04ms	64.71ms	NA	NA
	Sparse Grid				
Num. Bricks	36	146	183	266	443
Memory (MB)	11.25	45.63	57.19	83.13	138.44
Simulation	0.41ms	1.78ms	2.67ms	2.94ms	5.99ms
Scaling Sim.	78.14%	76.46%	75.01%	NA	NA

NOTE: Numbers captured on a GeForce GTX980

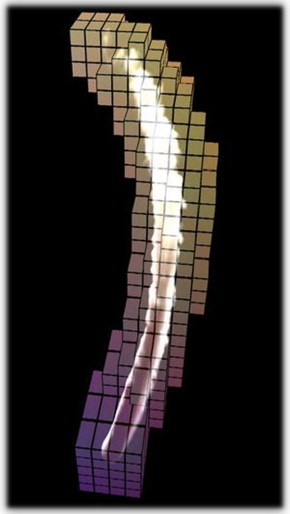
Scaling

- Speed ratio (1 Brick) $= \dfrac{\text{Time}\{\text{Sparse}\}}{\text{Time}\{\text{Full}\}}$
- ~75% across grid resolutions.

	Grid Resolution				
	128^3	256^3	384^3	512^3	$1{,}024^3$
Scaling Sim.	78.14%	76.46%	75.01%	NA	NA

Summary

- Fluid simulation in games is justified.
- Fluid is <u>not</u> box shaped!
- One volume is better than many small.
- Un/Compressed storage a viable fallback.
- VTRs great for fluid simulation.



- <u>Other latency resistant algorithms with tiled resouces?</u>

Questions?

Alex Dunn - adunn@nvidia.com
Twitter: @AlexWDunn

Thanks for attending.