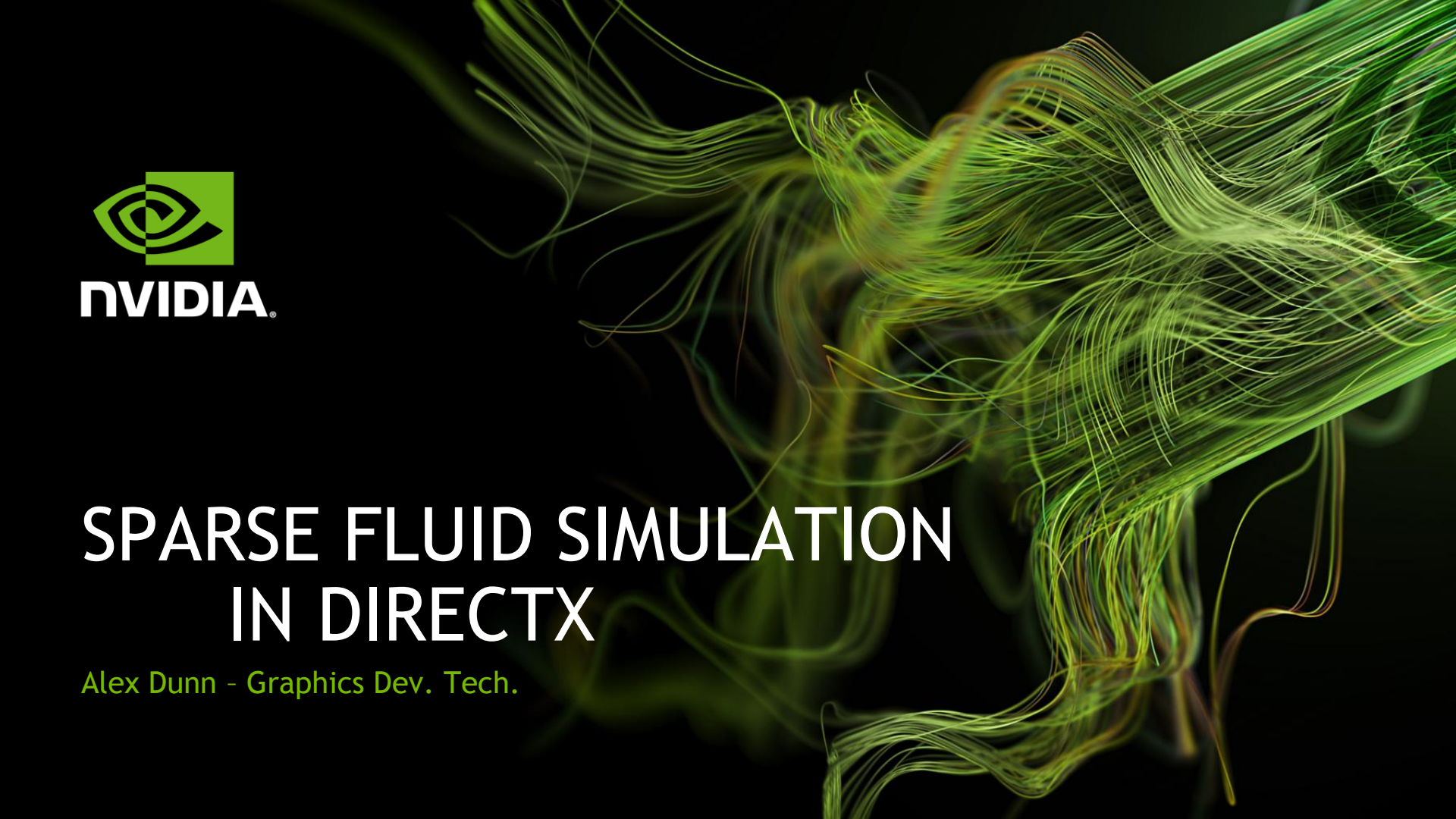


SPARSE FLUID SIMULATION IN DIRECTX

Alex Dunn – Graphics Dev. Tech.

AGENDA

We want more fluid in games! ☺

Eulerian Fluid Simulation.

Sparse Eulerian Fluid.

Feature Level 11.3 Enhancements.

WHY DO WE NEED FLUID IN GAMES?

Replace particle kinematics!

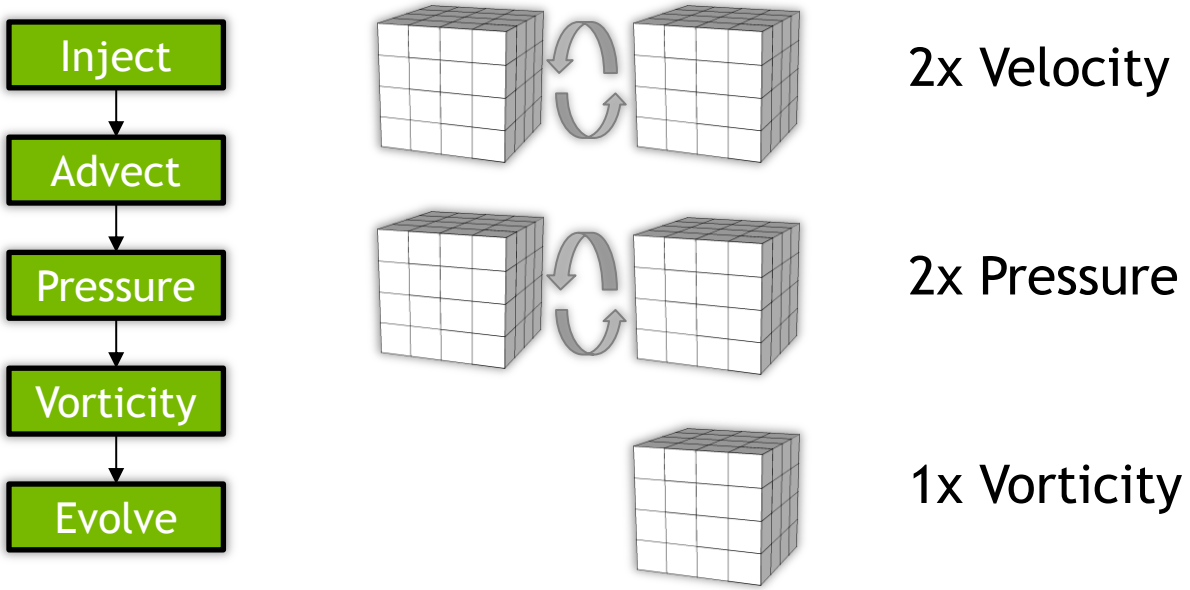
more realistic == better user immersion

More than just eye candy?

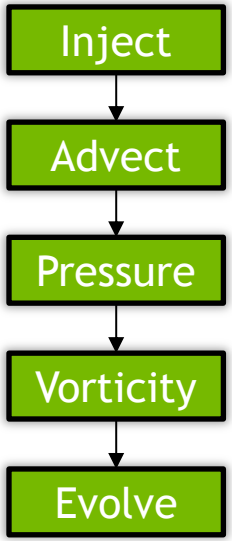
game mechanics?

EULERIAN SIMULATION #1

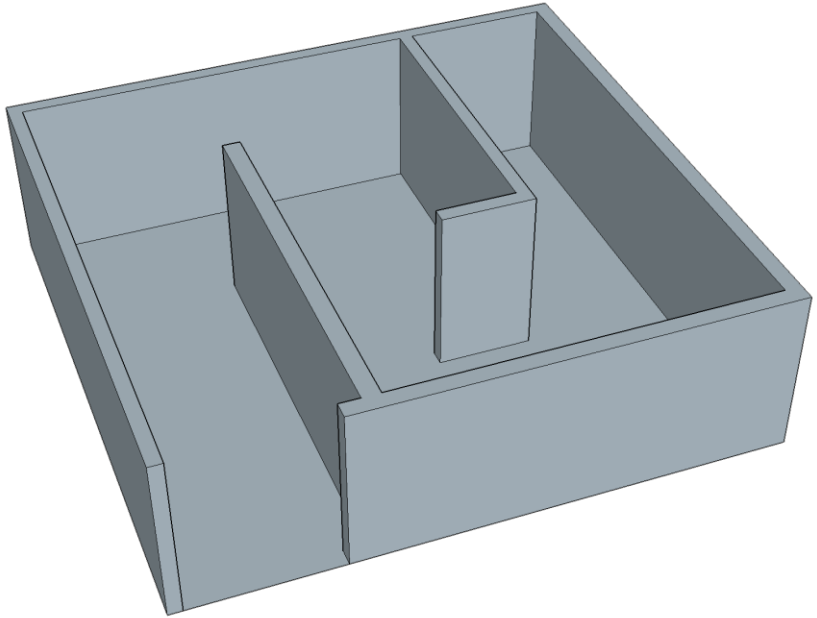
My (simple) DX11.0 eulerian fluid simulation:



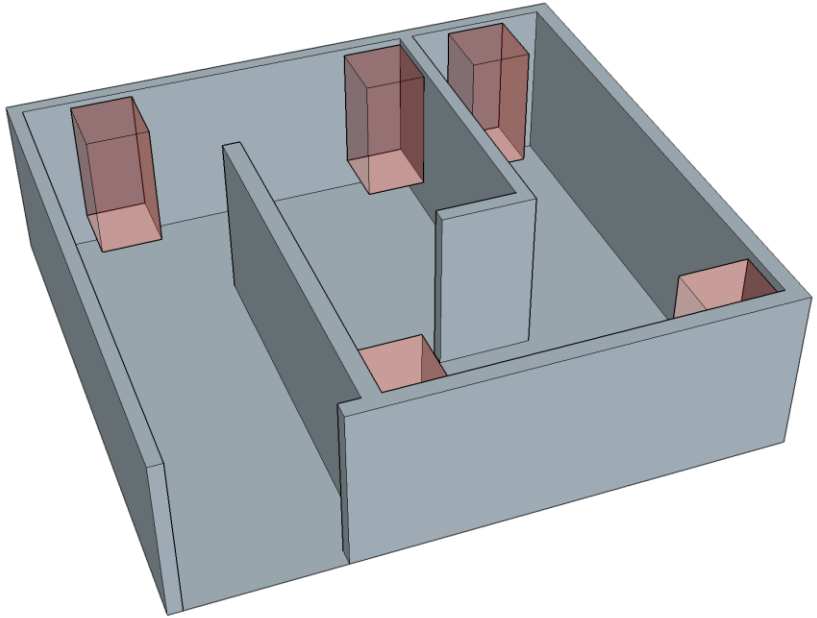
EULERIAN SIMULATION #2



- 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…

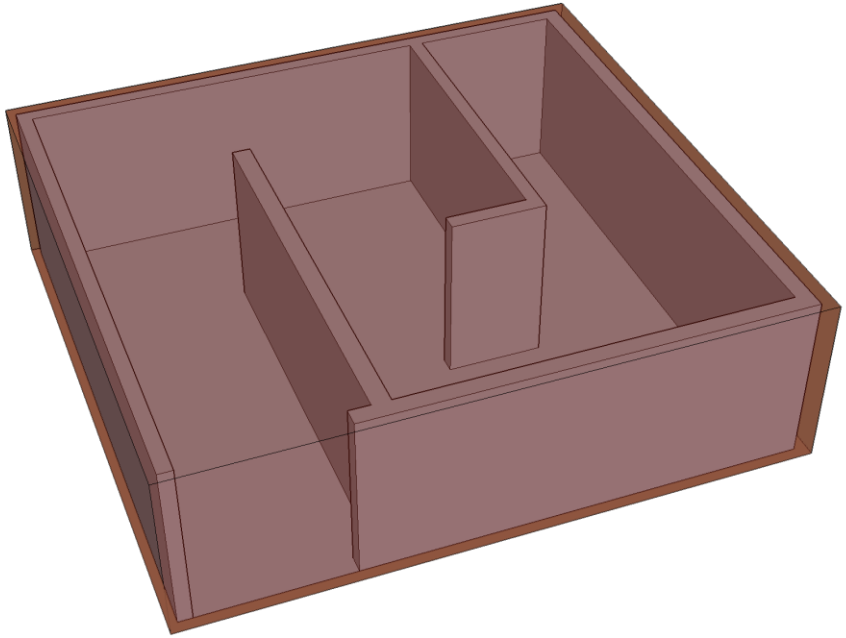
Fluid isn't box shaped.

- 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

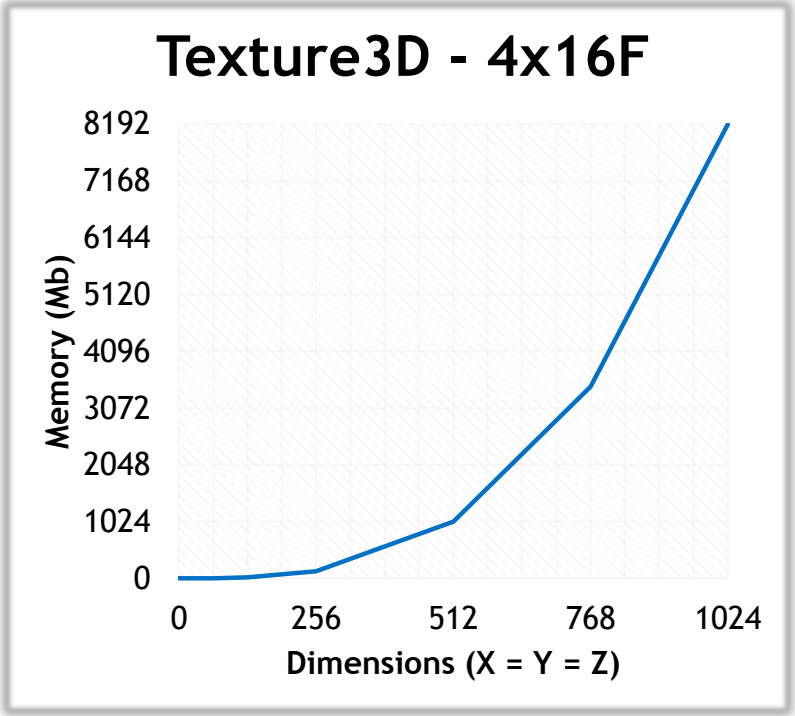
256^3 = ~16m cells

…

Applies to:

computational complexity

memory requirements

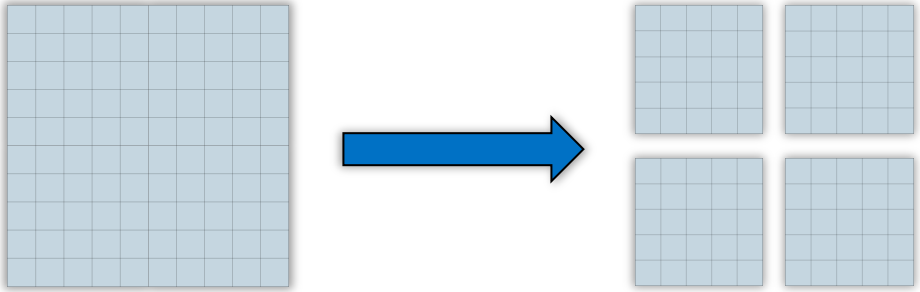


And that's just 1 texture…

BRICKS

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

Simulate each brick independently.



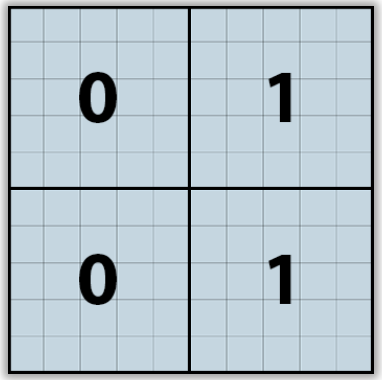
BRICK MAP

Need to track which bricks contain fluid

Texture3D<uint>

1 voxel per brick

- 0 → Ignore
- 1 → Simulate

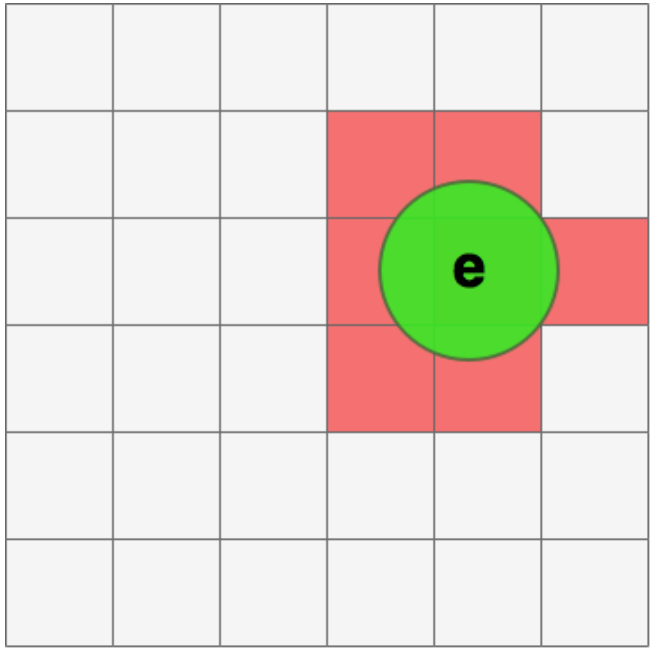


Could also use packed binary grids [Gruen15], but this requires atomics

TRACKING BRICKS #1

Initialise using fluid emitters.

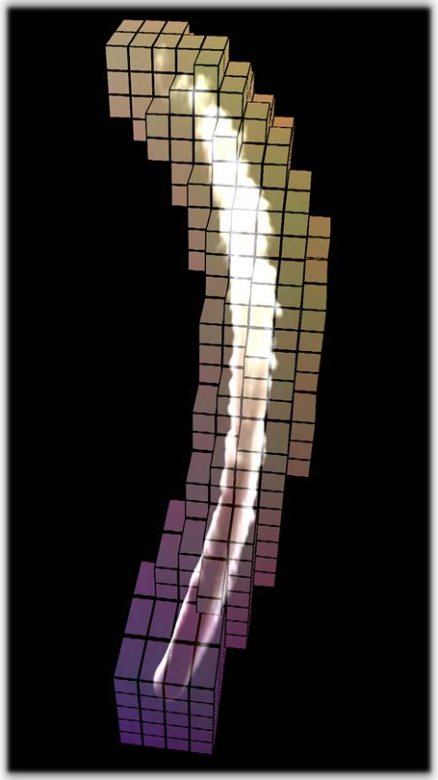
(easy with primitives)



TRACKING BRICKS #2

Simulating air is important for accuracy.

$$\text{Simulate?} = |\text{Velocity}| > 0$$

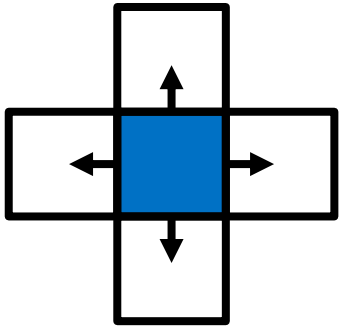


TRACKING BRICKS #3

Expansion (*ignore* → *simulate*)

if { $V_{|x|y|z|} > |D_{brick}|$ }

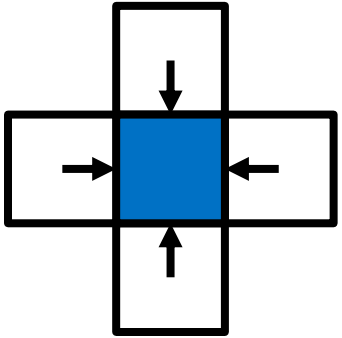
expand simulation in that axis



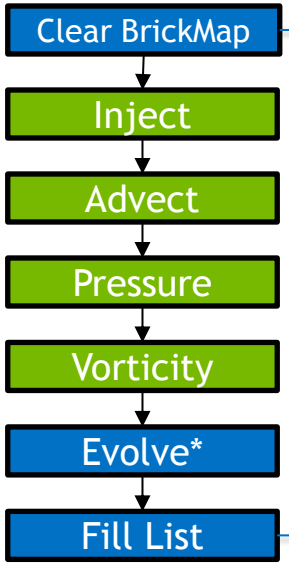
Reduction (*simulate* → *ignore*)

inverse of *Expansion*

handled automatically by clear



SPARSE SIMULATION



Reset all to 0 (ignore) 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*

PROBLEM!

N-order problem

64^3 = ~0.25m cells

128^3 = ~2m cells

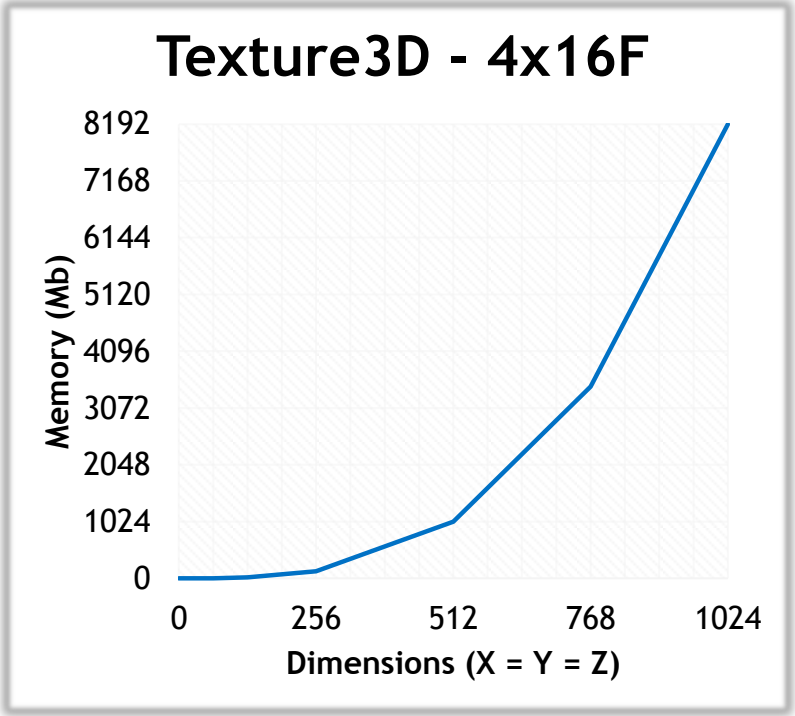
256^3 = ~16m cells

…

Applies to:

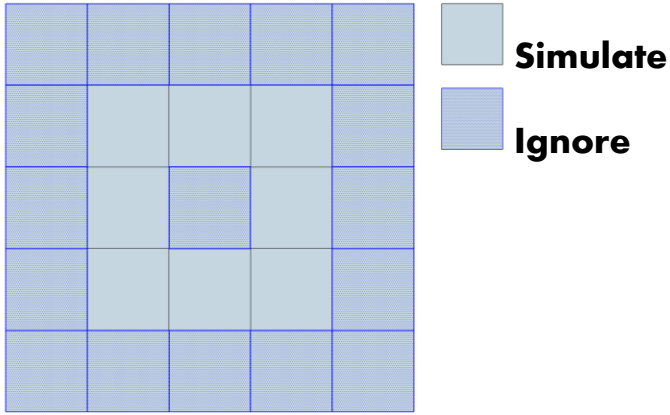
computational complexity

memory requirements



And that's just 1 texture…

UNCOMPRESSED STORAGE



Allocate everything; forget about unoccupied cells ☹

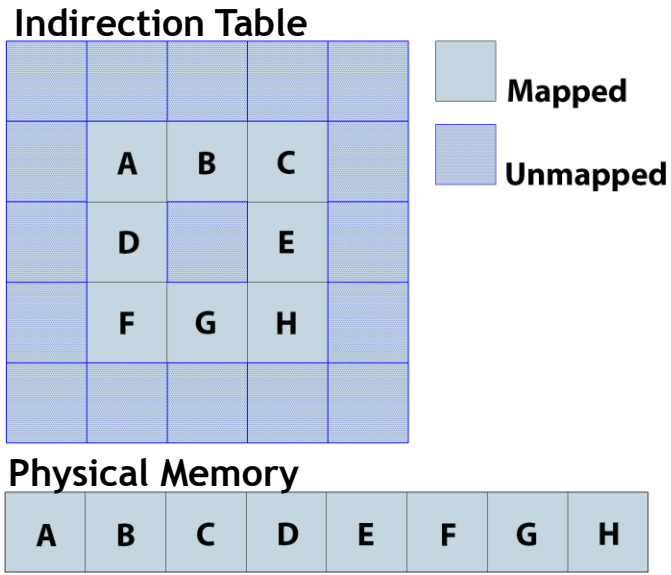
Pros:

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

Cons:

- no reduction in memory usage.

COMPRESSED STORAGE



Similar to, List<Brick>

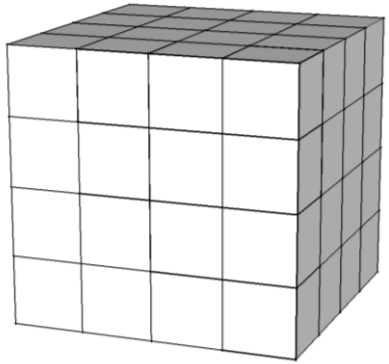
Pros:

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

Cons:

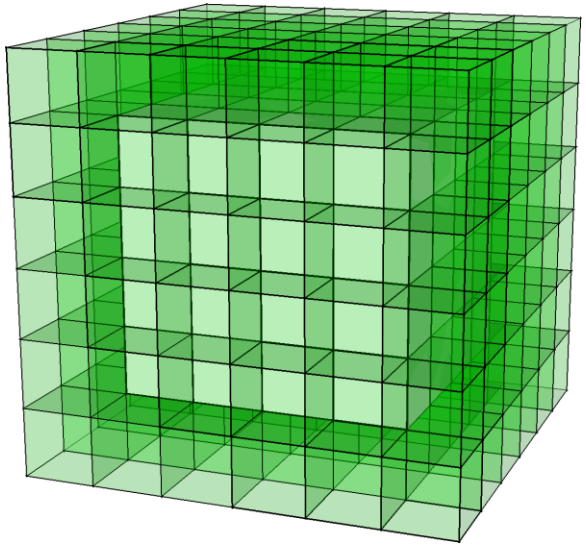
- allocation strategies.
- indirect lookup.
  - “software translation”
  - filtering particularly costly

PADDING TO REDUCE EDGE CASES



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

PADDING TO REDUCE EDGE CASES



1 Brick = $(1+4+1)^3 = 216$

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

<u>Can we do better?</u>

ENTER; FEATURE LEVEL 11.3

Volume Tiled Resources (VTR)! ☺

Extends 2D functionality in FL11.2

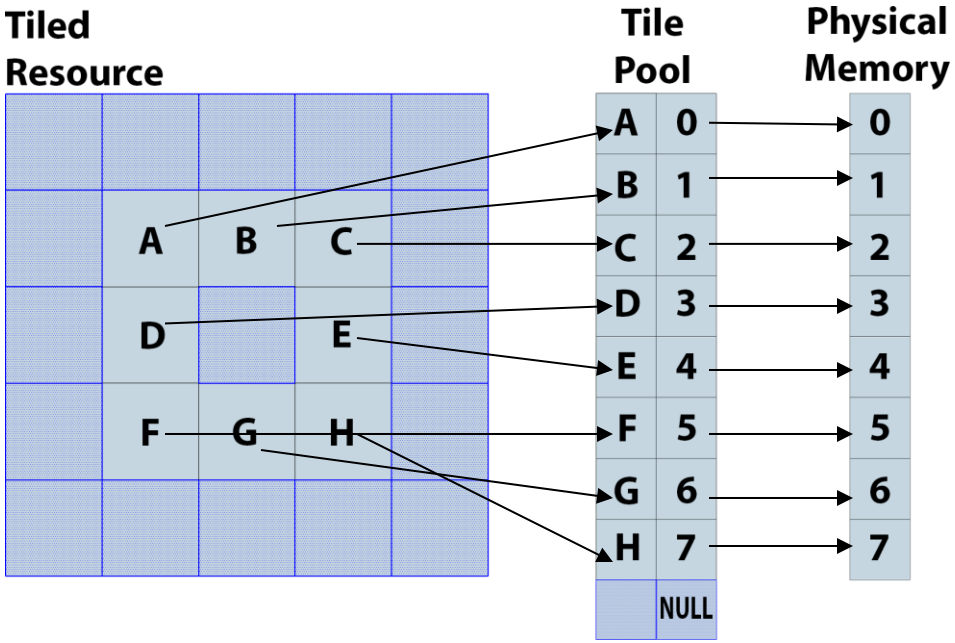
Must query HW support: **(DX11.3 != FL11.3):**

```
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;
```

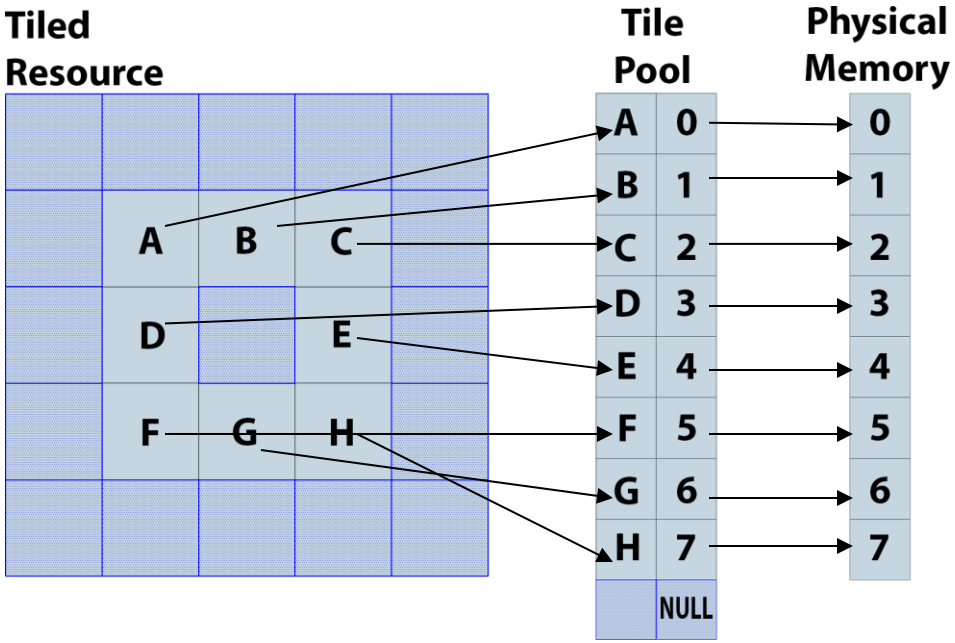
TILED RESOURCES #1



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

TILED RESOURCES #3

Letting the driver know which bricks/tiles should be resident:

```
HRESULT ID3D11DeviceContext2::UpdateTileMappings(
           ID3D11Resource                          *pTiledResource,
           UINT                                    NumTiledResourceRegions,
           const D3D11_TILED_RESOURCE_COORDINATE *pTiledResourceRegionStartCoordinates,
           const D3D11_TILE_REGION_SIZE            *pTiledResourceRegionSizes,
           ID3D11Buffer                            *pTilePool,
           UINT                                    NumRanges,
           const UINT                              *pRangeFlags,
           const UINT                              *pTilePoolStartOffsets,
           const UINT                              *pRangeTileCounts,
           UINT                                    Flags );
```

UPDATE TILE MAPPINGS – TIP

Don't update <u>all</u> tiles every frame.

```
const UINT                                  *pRangeFlags
```

Track tile deltas and use the range flags;

Ignore (unmapped) → D3D11_TILE_RANGE_NULL

Simulate (mapped) → D3D11_TILE_RANGE_REUSE_SINGLE_TILE

<u>Unchanged</u> → **<u>D3D11_TILE_RANGE_SKIP</u>**

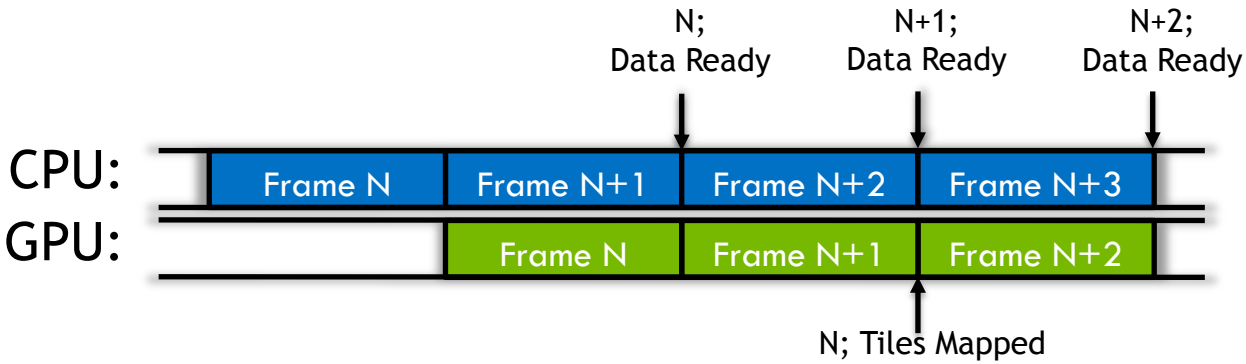
CPU READ BACKS

Taboo in real time graphics

CPU read backs are fine, if done correctly! (and bad if not)

2 frame latency (more for SLI)

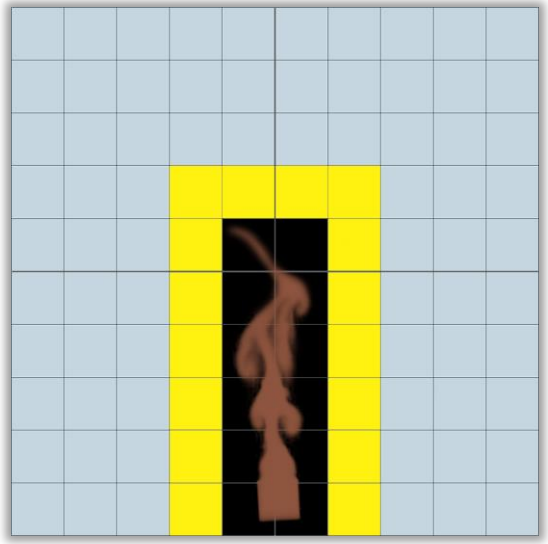
Profile map/unmap calls if unsure



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

Better Approach:

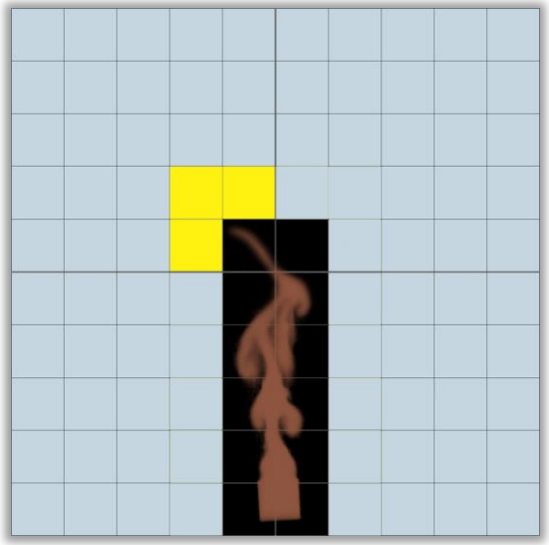
CPU Read-back:

occupied bricks.

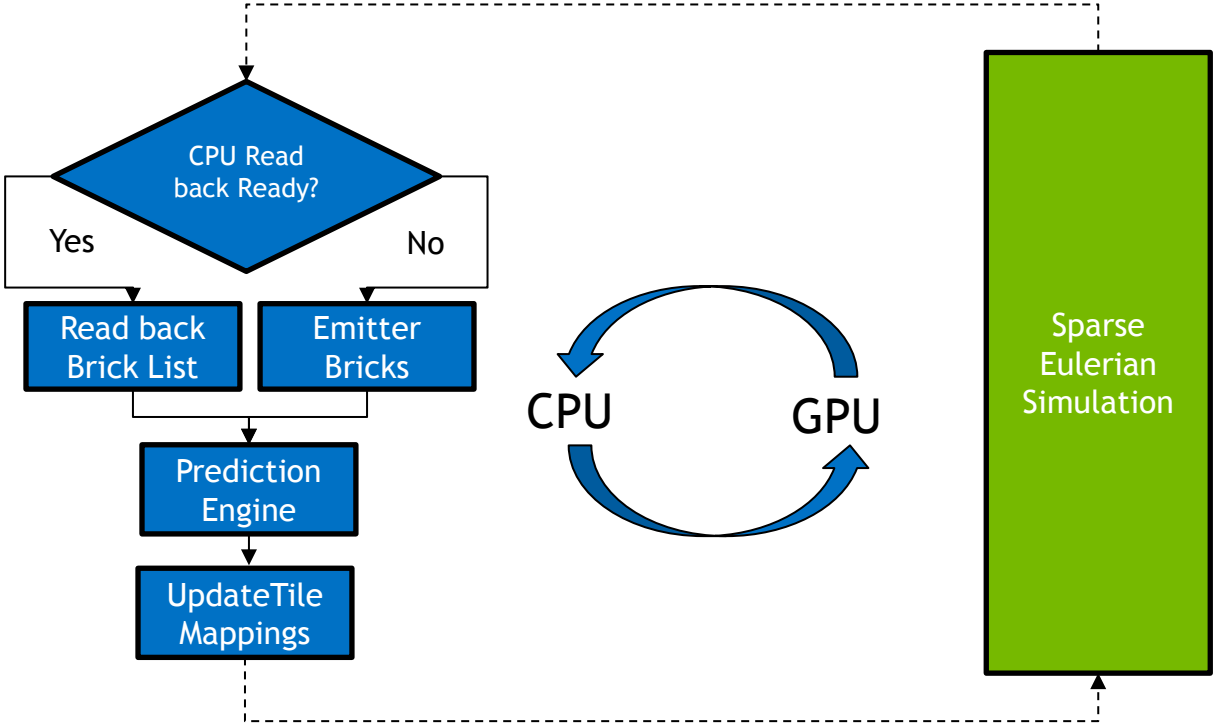
max{|V|} within brick.

<u>2 frames of latency!</u>

extrapolate “probable” tiles.

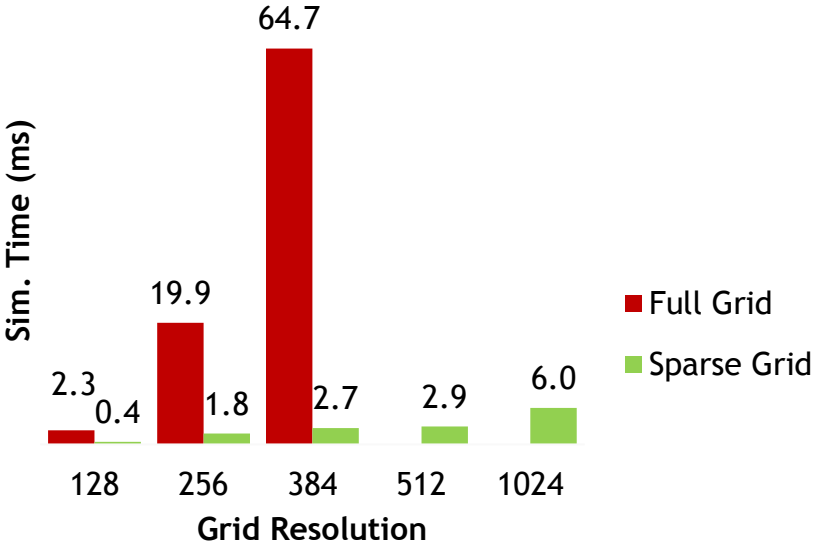


LATENCY RESISTANT SIMULATION #3



DEMO

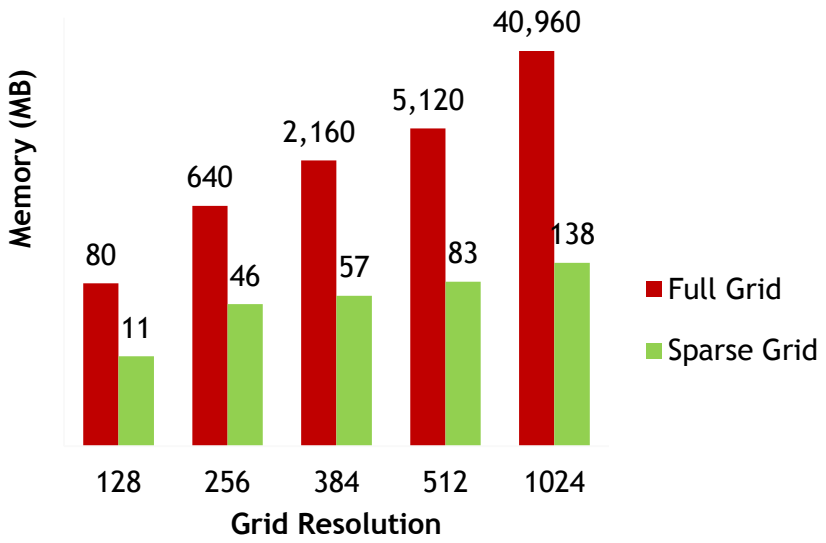
PERFORMANCE #1



	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

Other “latency resistant” techniques using tiled resources??

Thank you!

Alex Dunn - adunn@nvidia.com

Twitter: @AlexWDunn