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:

- Inject
- Advect
- Pressure
- Vorticity
- Evolve

2x Velocity

2x Pressure

1x Vorticity
Eulerian Simulation #2

- Add fluid to simulation
- Move data at, \( XYZ \rightarrow (XYZ + \text{Velocity}) \)
- 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
  - $64^3 = \sim 0.25m$ cells
  - $128^3 = \sim 2m$ cells
  - $256^3 = \sim 16m$ 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<\text{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* $\rightarrow$ *occupied*)
  - if \( \{ |V_{x,y,z}| > |D_{\text{brick}}| \} \)
  - expand in that axis

- Reduction (*occupied* $\rightarrow$ *unoccupied*)
  - inverse of Expansion
  - handled automatically
Sparse Simulation

- Clear Tiles
  - Reset all tiles to 0 (unoccupied) in brick map.
- Inject
- Advect
- Pressure
- Vorticity
- Evolve*
  - Read value from brick map.
  - Append brick coordinate to list if occupied.
- Fill List

*Includes expansion

```c
Texture3D<uint> g_BrickMapRO;
AppendStructuredBuffer<uint3> g_ListRW;
if (g_BrickMapRO[idx] != 0)
{
    g_ListRW.Append(idx);
}
```
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 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**)
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
Gotcha: Tile mappings must be updated from CPU
Latency Resistant Simulation #1

Naïve Approach:

- **clamp velocity to** $V_{\text{max}}$
- **CPU Read-back:**
  - occupied bricks.
  - 2 frames of latency!
- 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

Diagram:
- CPU Readback Ready?
  - Yes:
    - Readback Brick List
    - Emitter Bricks
    - Prediction Engine
    - UpdateTile Mappings
  - No:
    - Sparse Eulerian Simulation

CPU and GPU interactions:
- CPU
- GPU
- Feedback loop
Demo
Performance #1

NOTE: Numbers captured on a GeForce GTX980
Performance #2

NOTE: Numbers captured on a GeForce GTX980
Scaling

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

<table>
<thead>
<tr>
<th>Grid Resolution</th>
<th>128^3</th>
<th>256^3</th>
<th>384^3</th>
<th>512^3</th>
<th>1024^3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scaling Sim.</td>
<td>78.14%</td>
<td>76.46%</td>
<td>75.01%</td>
<td>NA</td>
<td>NA</td>
</tr>
</tbody>
</table>
Summary

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

- Other latency resistant algorithms with tiled resources?
Questions?

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

Thanks for attending.