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:

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

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Texture3D<

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.

Simulate? = |Velocity| > 0
TRACKING BRICKS #3

Expansion (*ignore* → *simulate*)

\[
\text{if } \{ V_{|x|y|z|} > |D_{\text{brick}}| \},
\]

expand simulation in that axis

Reduction (*simulate* → *ignore*)

inverse of Expansion

handled automatically by clear
SPARSE SIMULATION

- Clear BrickMap: Reset all to 0 (ignore) in brick map.
- Inject
- Advect
- Pressure
- Vorticity
- Evolve*
- Fill List

*Includes expansion

```cpp
Texture3D<uint> g_BrickMapRO;
AppendStructuredBuffer<uint3> g_ListRW;
if(g_BrickMapRO[idx] != 0)
{
    g_ListRW.Append(idx);
}
```
PROBLEM!

N-order problem

- $64^3 = \sim 0.25 \text{m cells}$
- $128^3 = \sim 2 \text{m cells}$
- $256^3 = \sim 16 \text{m cells}$

... 

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

```c++
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)

<table>
<thead>
<tr>
<th>BPP</th>
<th>Tile Dimensions</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>64x32x32</td>
</tr>
<tr>
<td>16</td>
<td>32x32x32</td>
</tr>
<tr>
<td>32</td>
<td>32x32x16</td>
</tr>
<tr>
<td>64</td>
<td>32x16x16</td>
</tr>
<tr>
<td>128</td>
<td>16x16x16</td>
</tr>
</tbody>
</table>
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 all tiles every frame.

Track tile deltas and use the range flags;

- Ignore (unmapped)  → D3D11_TILE_RANGE_NULL
- Simulate (mapped)  → D3D11_TILE_RANGE_REUSE_SINGLE_TILE
- Unchanged         → D3D11_TILE_RANGE_SKIP
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
LATTENCY 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

Better Approach:

CPU Read-back:

- occupied bricks.
- \( \max\{|V|\} \) within brick.
- 2 frames of latency!

extrapolate “probable” tiles.
LATENCY RESISTANT SIMULATION #3
DEMO
PERFORMANCE #1

**Sim. Time (ms)**

<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><strong>Full Grid</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Num. Bricks</td>
<td>256</td>
<td>2048</td>
<td>6,912</td>
<td>16,384</td>
<td>131,072</td>
</tr>
<tr>
<td>Memory (MB)</td>
<td>80</td>
<td>640</td>
<td>2,160</td>
<td>5,120</td>
<td>40,960</td>
</tr>
<tr>
<td>Simulation</td>
<td>2.29ms</td>
<td>19.04ms</td>
<td>64.71ms</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td><strong>Sparse Grid</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Num. Bricks</td>
<td>36</td>
<td>146</td>
<td>183</td>
<td>266</td>
<td>443</td>
</tr>
<tr>
<td>Memory (MB)</td>
<td>11.25</td>
<td>45.63</td>
<td>57.19</td>
<td>83.13</td>
<td>138.44</td>
</tr>
<tr>
<td>Simulation</td>
<td>0.41ms</td>
<td>1.78ms</td>
<td>2.67ms</td>
<td>2.94ms</td>
<td>5.99ms</td>
</tr>
</tbody>
</table>

**Scaling Sim.**

|               | 78.14% | 76.46% | 75.01% | NA    | NA     |

**NOTE:** Numbers captured on a GeForce GTX980
PERFORMANCE #2

NOTE: Numbers captured on a GeForce GTX980
Thank you!

Alex Dunn - adunn@nvidia.com

Twitter: @AlexWDunn

Other “latency resistant” techniques using tiled resources??