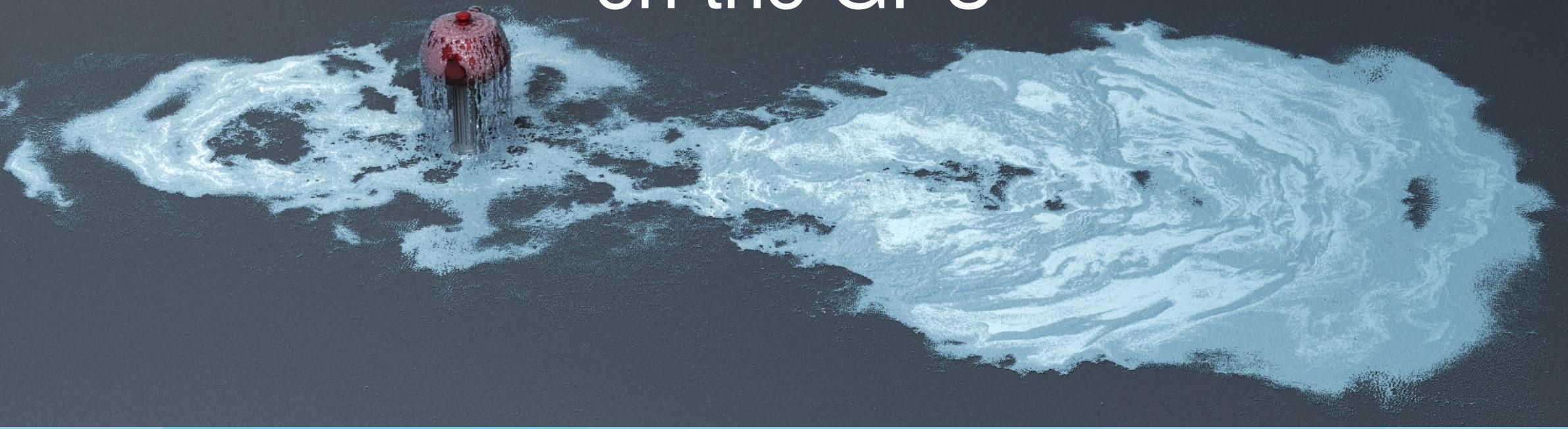


# Fast Fluid Simulation with Sparse Volumes on the GPU

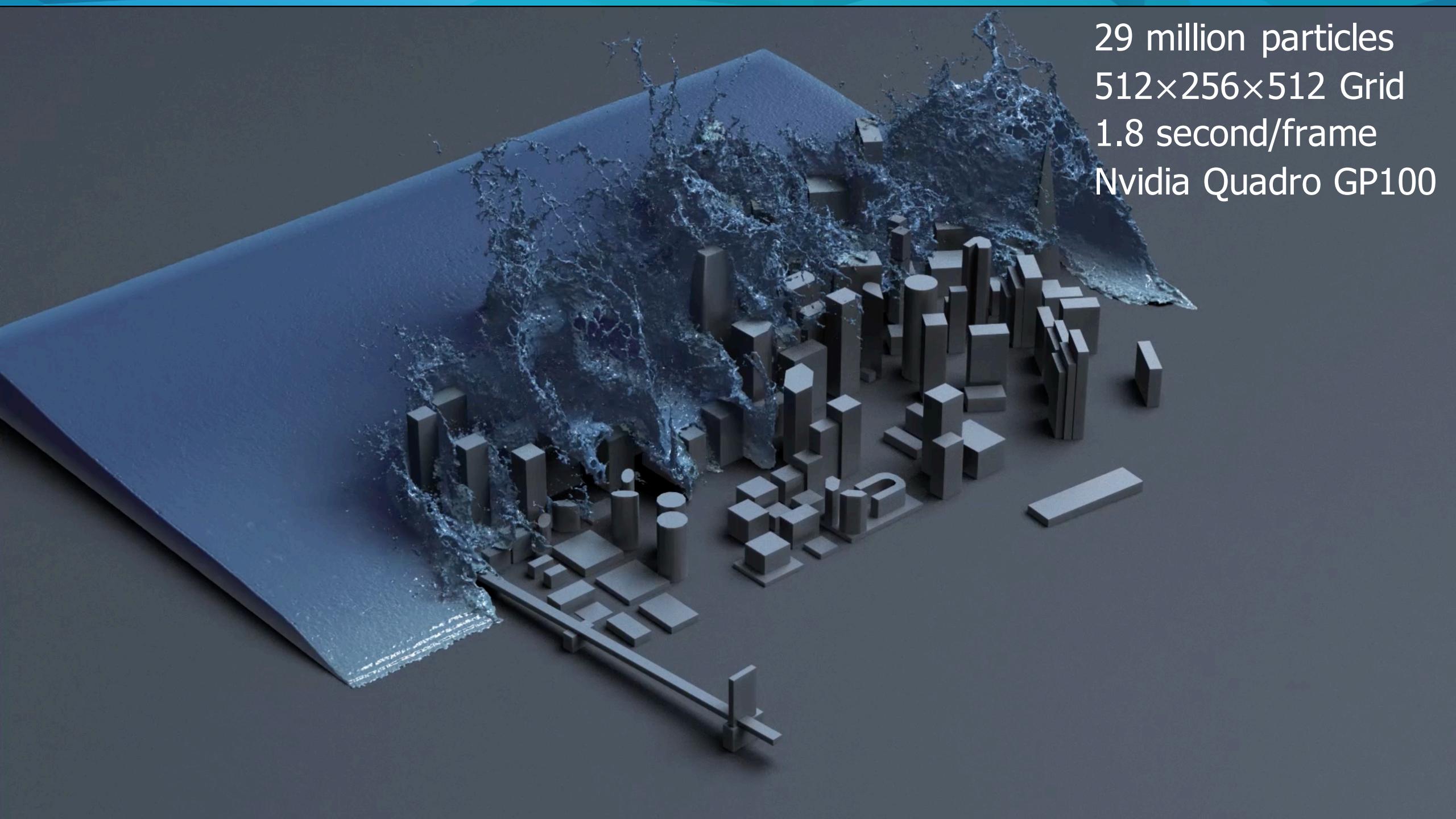


Kui Wu<sup>1</sup>, Nghia Truong<sup>1</sup>, Cem Yuksel<sup>1</sup>, Rama Hoetzlein<sup>2</sup>

<sup>1</sup> University of Utah, USA

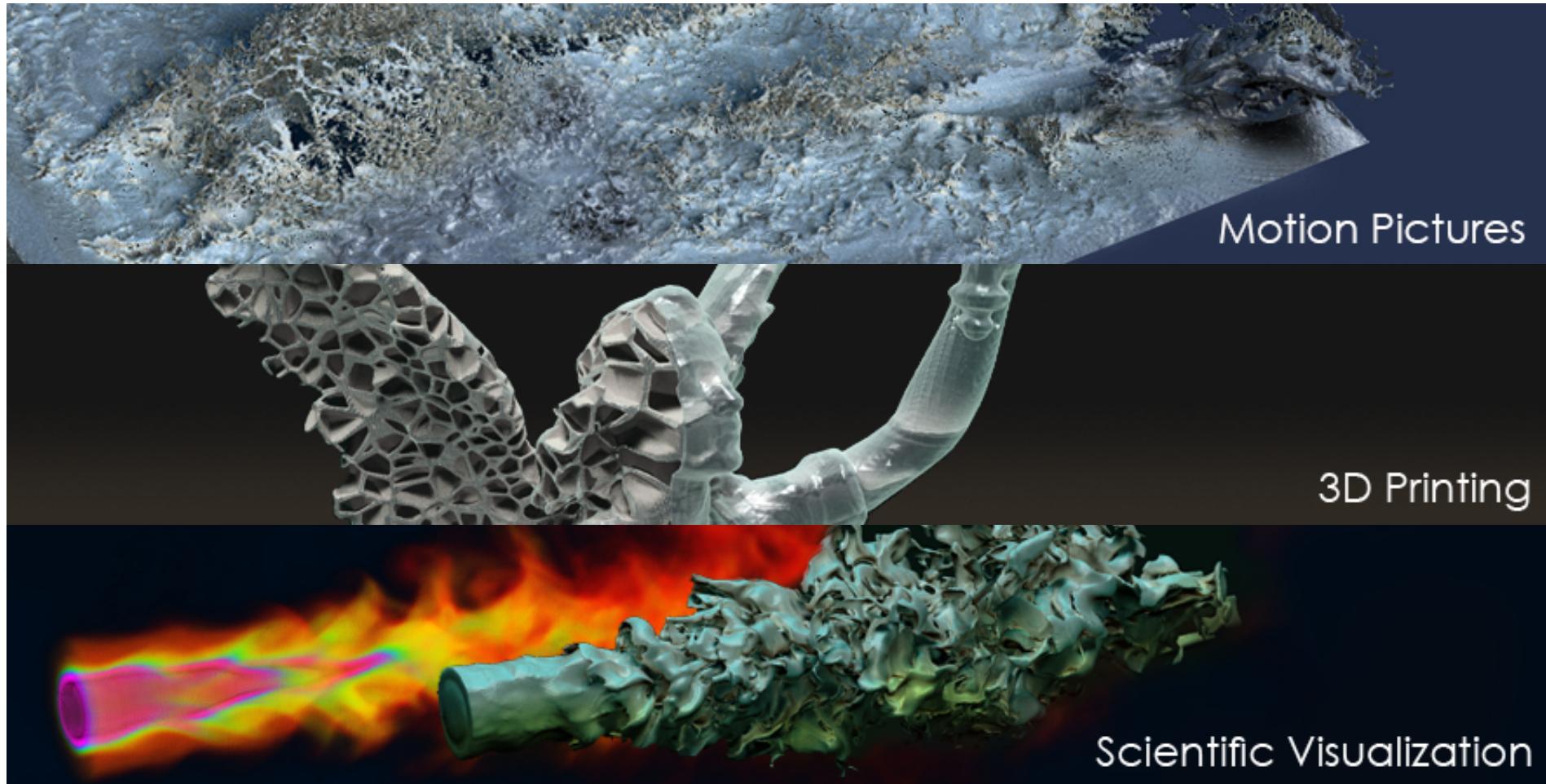
<sup>2</sup> NVIDIA Corporation





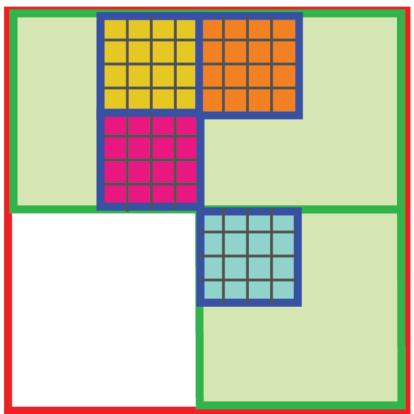
29 million particles  
512×256×512 Grid  
1.8 second/frame  
Nvidia Quadro GP100

# NVIDIA® GVDB Voxels

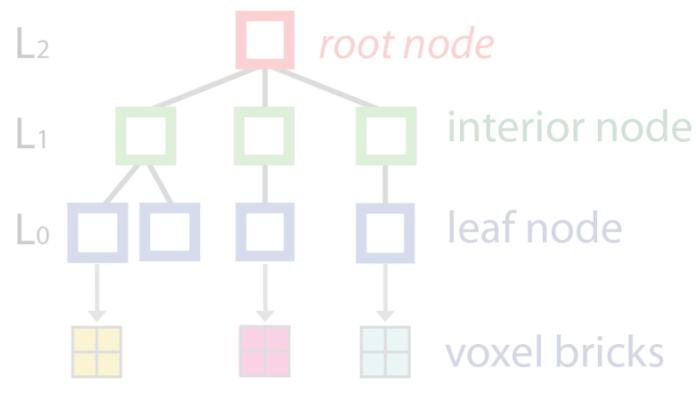


A new, open source NVIDIA SDK for compute, simulation and rendering of sparse volumes based on CUDA

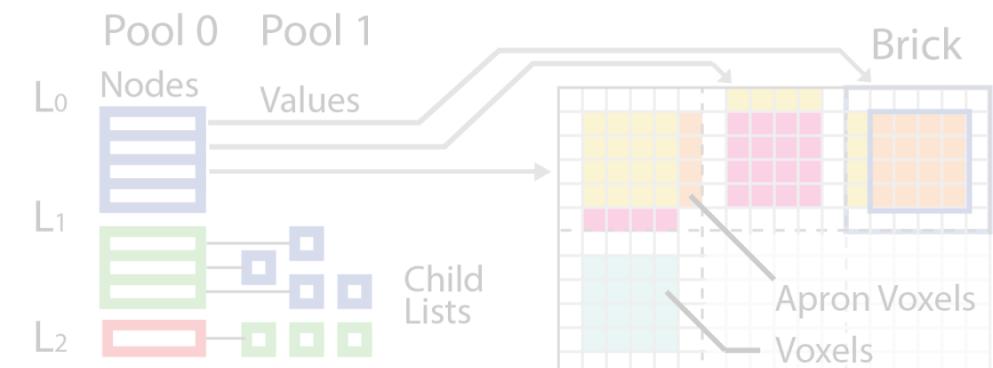
# NVIDIA® GVDB Voxels



a) Spatial domain



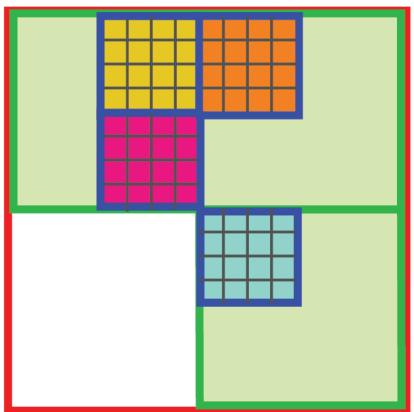
b) Tree schematic



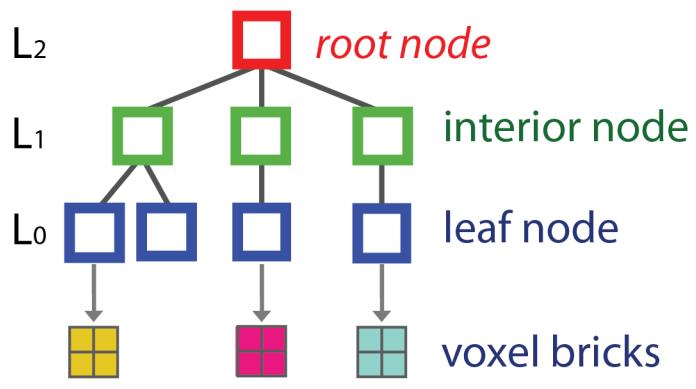
c) GVDB Data Structure

d) GVDB Voxel Atlas

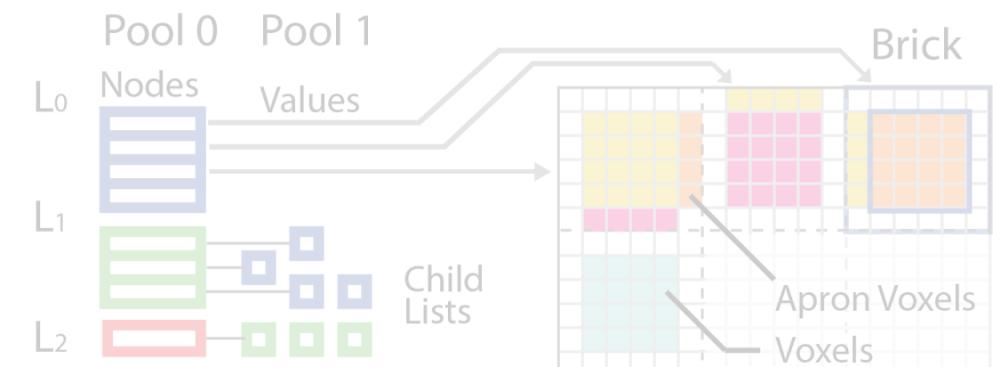
# NVIDIA® GVDB Voxels



a) Spatial domain



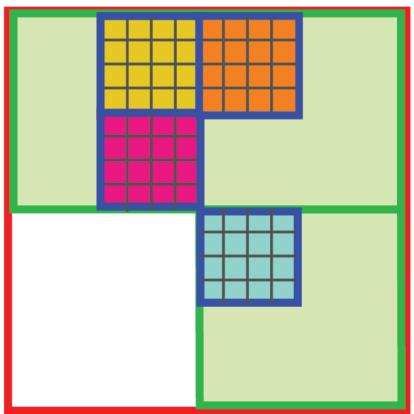
b) Tree schematic



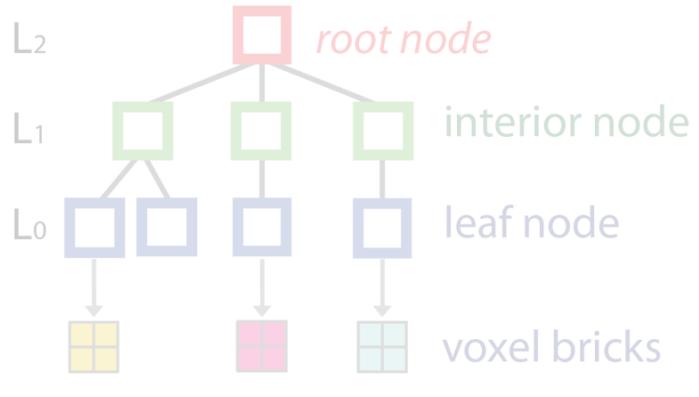
c) GVDB Data Structure

d) GVDB Voxel Atlas

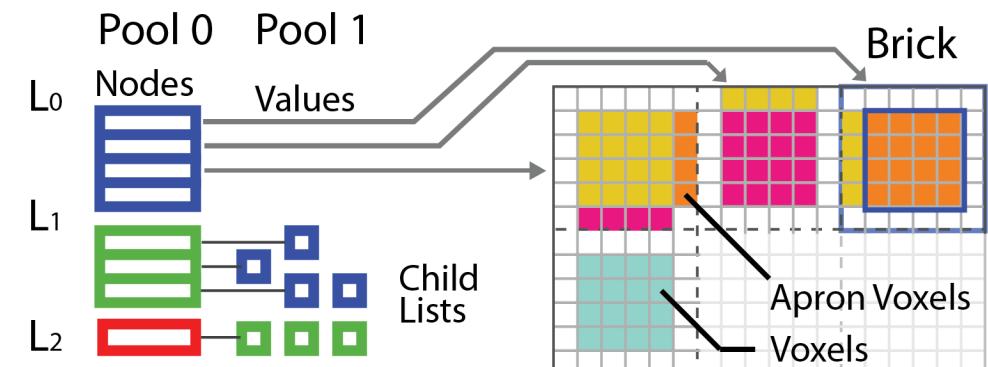
# NVIDIA® GVDB Voxels



a) Spatial domain



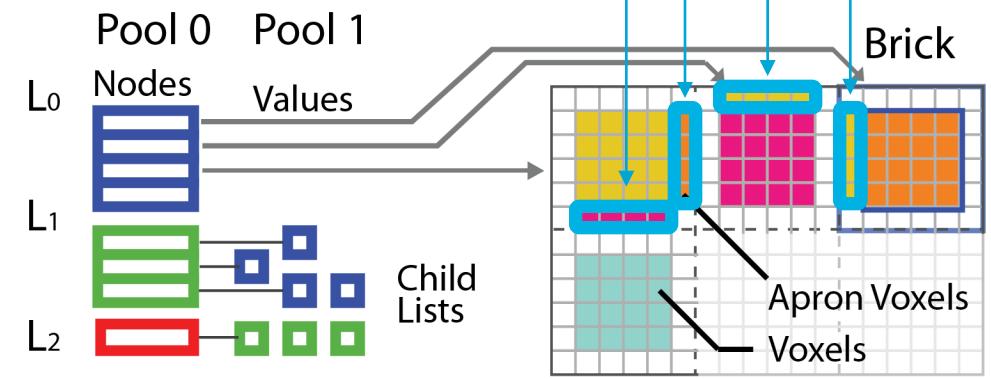
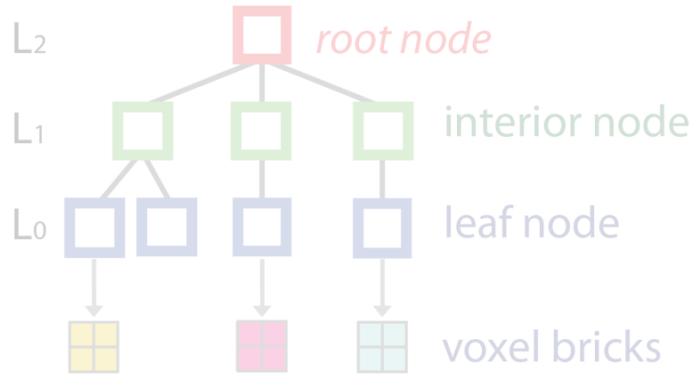
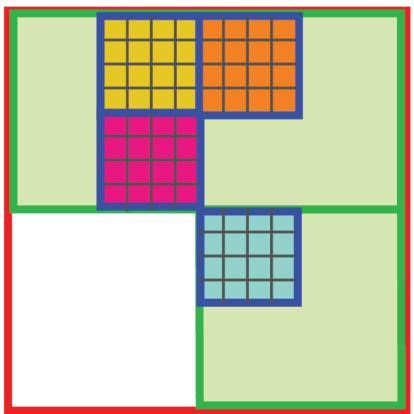
b) Tree schematic



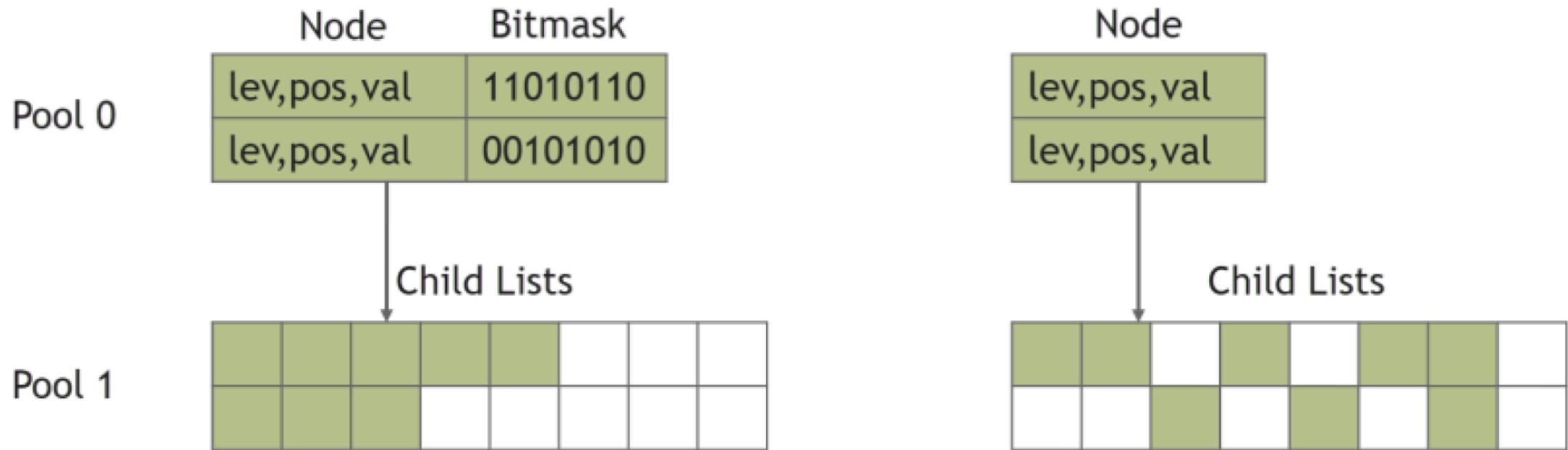
c) GVDB Data Structure

d) GVDB Voxel Atlas

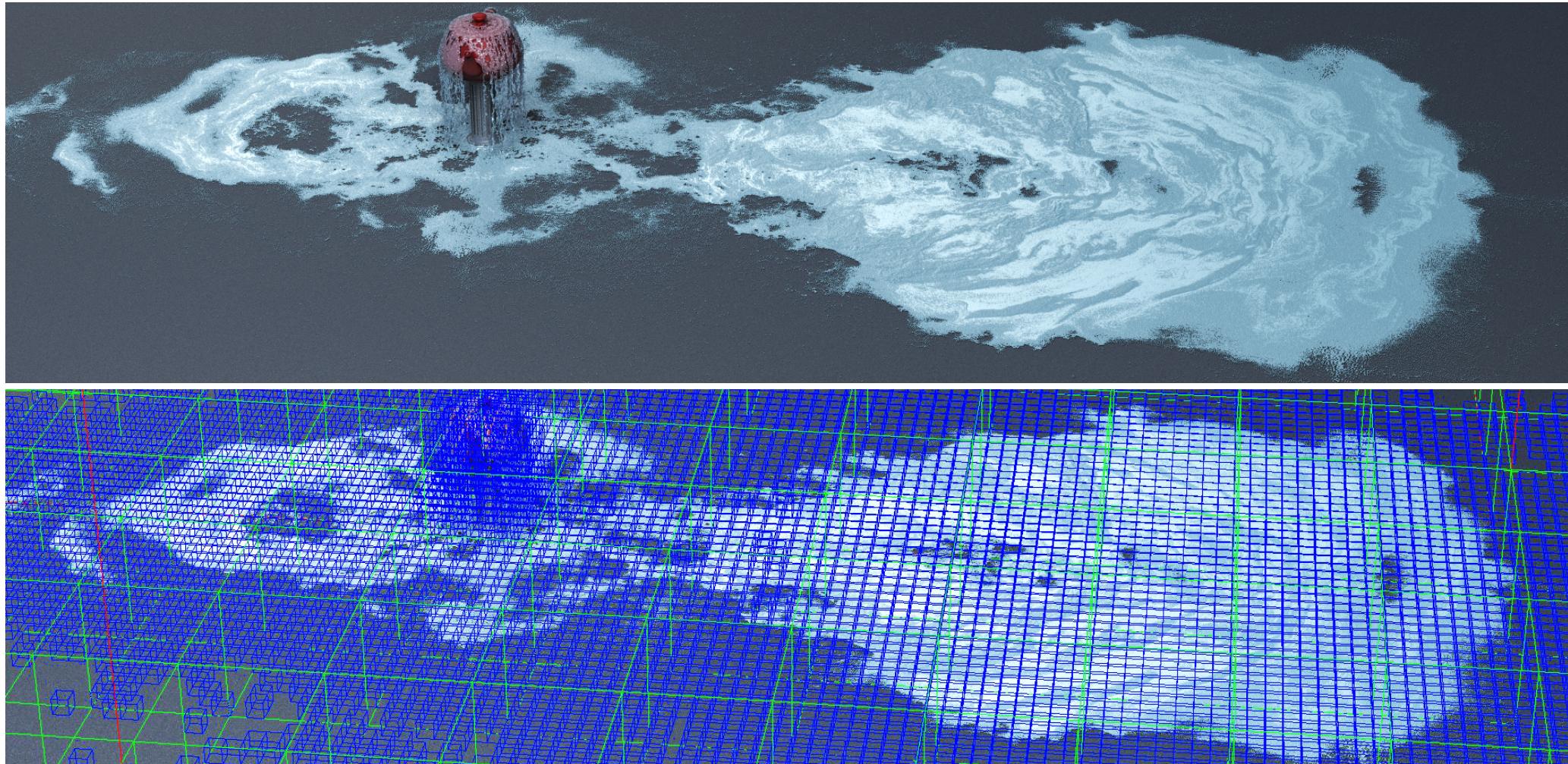
# NVIDIA® GVDB Voxels



# NVIDIA® GVDB Voxels

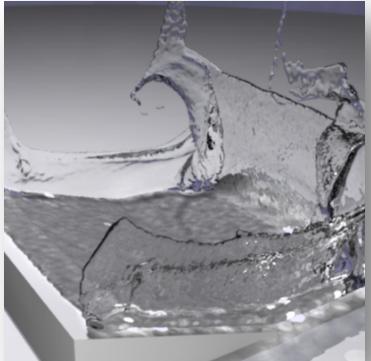


# FLIP Simulation with NVIDIA® GVDB Sparse Voxels

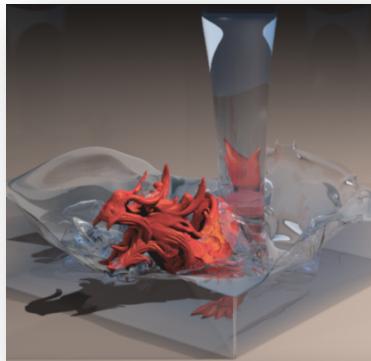


# Prior Work

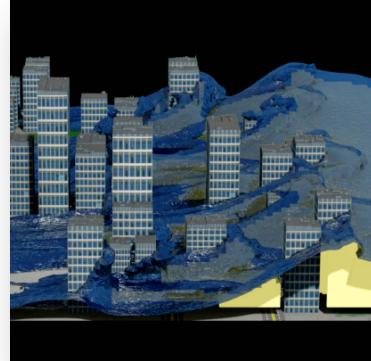
## CPU-based methods for large scale simulation



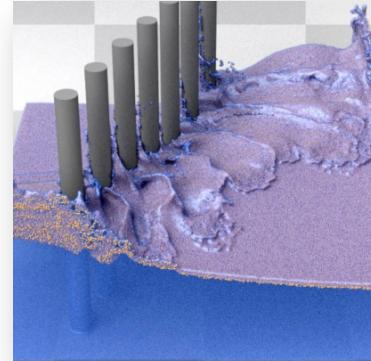
[Zhu and Bridson 2005]  
Fluid-implicit particle method (FLIP)



[McAdams et al. 2010]  
Multigrid PCG



[Ferstl et al. 2014]  
Domain Decomposition



[Ferstl et al. 2016]  
Narrow Band FLIP



Schur Complement Solver



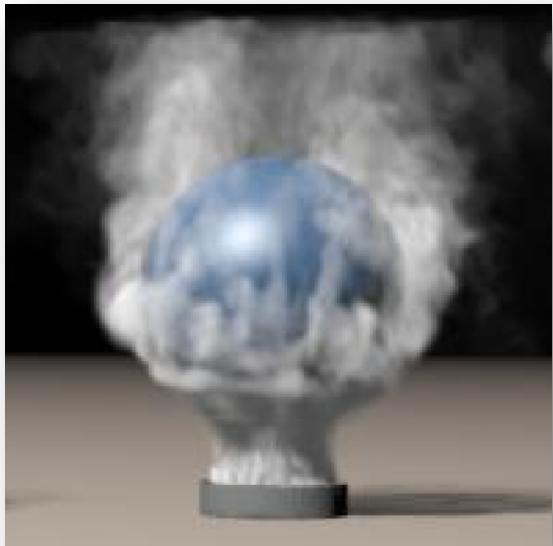
Liu et al. 2016



Aanjaneya and Gao et al. 2017  
SPGrid

# Prior Work

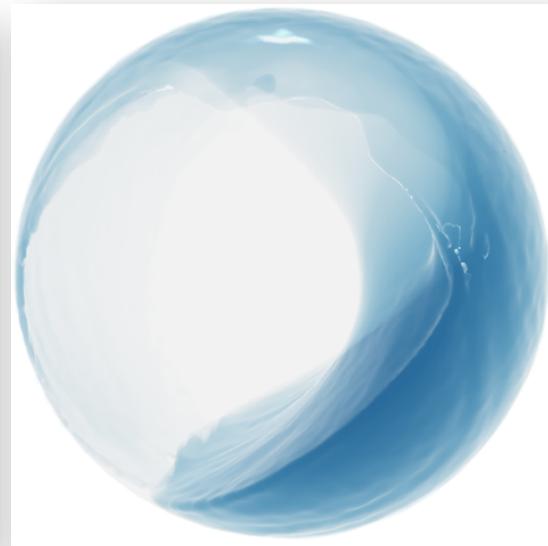
## GPU-based methods



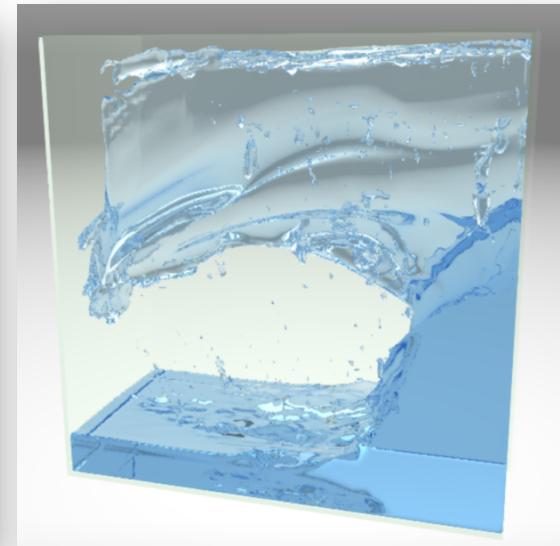
[Molemaker et al. 2008]



[Horvath and Geiger 2009]



[Chentanez and Müller 2011]



[Chentanez and Müller 2012]

# FLIP with Sparse Volumes on the GPU

```
1: procedure SparseFLIP( )
2:    $P \leftarrow$  initial points
3:    $V \leftarrow$  GVDB structure
4:   for each frame do
5:     if first frame then
6:        $V_{topo} \leftarrow$  full rebuild ( $P$ )
7:     else
8:        $V_{topo} \leftarrow$  incremental build ( $P$ )
9:     end if
10:     $V \leftarrow$  resize and clear ( $V_{topo}$ )
11:     $S \leftarrow$  insert points in subcells ( $V, P$ )
12:     $V(vel) \leftarrow$  particles-to-voxels ( $S, P$ )
13:     $V \leftarrow$  update apron ( $\rho, vel, marker$ )
14:     $V(vel_{old}) \leftarrow V(vel)$ 
15:     $V(div) \leftarrow$  divergence ( $V(vel)$ )
16:     $V(\rho) \leftarrow$  CG pressure solve ( $V, div$ )
17:     $V(vel) \leftarrow$  pressure-to-velocity ( $V(\rho)$ )
18:     $V \leftarrow$  update apron ( $V(vel)$ )
19:     $P \leftarrow$  advance ( $V(vel), V(vel_{old})$ )
20:   end for
21: end procedure
```

Dynamic Topology

Particles-to-Voxels

Pressure Solver (CG)

# FLIP with Sparse Volumes on the GPU

```
1: procedure SparseFLIP()
2:    $P \leftarrow$  initial points
3:    $V \leftarrow$  GVDB structure
4:   for each frame do
5:     if first frame then
6:        $V_{topo} \leftarrow$  full rebuild ( $P$ )
7:     else
8:        $V_{topo} \leftarrow$  incremental build ( $P$ )
9:     end if
10:     $V \leftarrow$  resize and clear ( $V_{topo}$ )
11:     $S \leftarrow$  insert points in subcells ( $V, P$ )
12:     $V(vel) \leftarrow$  particles-to-voxels ( $S, P$ )
13:     $V \leftarrow$  update apron ( $\rho, vel, marker$ )
14:     $V(vel_{old}) \leftarrow V(vel)$ 
15:     $V(div) \leftarrow$  divergence ( $V(vel)$ )
16:     $V(\rho) \leftarrow$  CG pressure solve ( $V, div$ )
17:     $V(vel) \leftarrow$  pressure-to-velocity ( $V(\rho)$ )
18:     $V \leftarrow$  update apron ( $V(vel)$ )
19:     $P \leftarrow$  advance ( $V(vel), V(vel_{old})$ )
20:   end for
21: end procedure
```

## Dynamic Topology

```
if first frame then
   $V_{topo} \leftarrow$  full rebuild ( $P$ )
else
   $V_{topo} \leftarrow$  incremental build ( $P$ )
end if
```

Particles-to Voxels

Pressure Solver (CG)

# FLIP with Sparse Volumes on the GPU

```
1: procedure SparseFLIP( )
2:    $P \leftarrow$  initial points
3:    $V \leftarrow$  GVDB structure
4:   for each frame do
5:     if first frame then
6:        $V_{topo} \leftarrow$  full rebuild ( $P$ )
7:     else
8:        $V_{topo} \leftarrow$  incremental build ( $P$ )
9:     end if
10:     $V \leftarrow$  resize and clear ( $V_{topo}$ )
11:     $S \leftarrow$  insert points in subcells ( $V, P$ )
12:     $V(vel) \leftarrow$  particles-to-voxels ( $S, P$ )
13:     $V \leftarrow$  update apron ( $\rho, vel, marker$ )
14:     $V(vel_{old}) \leftarrow V(vel)$ 
15:     $V(div) \leftarrow$  divergence ( $V(vel)$ )
16:     $V(\rho) \leftarrow$  CG pressure solve ( $V, div$ )
17:     $V(vel) \leftarrow$  pressure-to-velocity ( $V(\rho)$ )
18:     $V \leftarrow$  update apron ( $V(vel)$ )
19:     $P \leftarrow$  advance ( $V(vel), V(vel_{old})$ )
20:   end for
21: end procedure
```

Dynamic Topology

Particles-to-Voxels

$S \leftarrow$  insert points in subcells ( $V, P$ )  
 $V(vel) \leftarrow$  particles-to-voxels ( $S, P$ )

Pressure Solver (CG)

# FLIP with Sparse Volumes on the GPU

```
1: procedure SparseFLIP( )
2:    $P \leftarrow$  initial points
3:    $V \leftarrow$  GVDB structure
4:   for each frame do
5:     if first frame then
6:        $V_{topo} \leftarrow$  full rebuild ( $P$ )
7:     else
8:        $V_{topo} \leftarrow$  incremental build ( $P$ )
9:     end if
10:     $V \leftarrow$  resize and clear ( $V_{topo}$ )
11:     $S \leftarrow$  insert points in subcells ( $V, P$ )
12:     $V(vel) \leftarrow$  particles-to-voxels ( $S, P$ )
13:     $V \leftarrow$  update apron ( $\rho, vel, marker$ )
14:     $V(vel_{old}) \leftarrow V(vel)$ 
15:     $V(div) \leftarrow$  divergence ( $V(vel)$ )
16:     $V(\rho) \leftarrow$  CG pressure solve ( $V, div$ )
17:     $V(vel) \leftarrow$  pressure-to-velocity ( $V(\rho)$ )
18:     $V \leftarrow$  update apron ( $V(vel)$ )
19:     $P \leftarrow$  advance ( $V(vel), V(vel_{old})$ )
20:   end for
21: end procedure
```

Dynamic Topology

Particles-to-Voxels

Pressure Solver (CG)

$V(\rho) \leftarrow$  CG pressure solve ( $V, div$ )

# FLIP with Sparse Volumes on the GPU

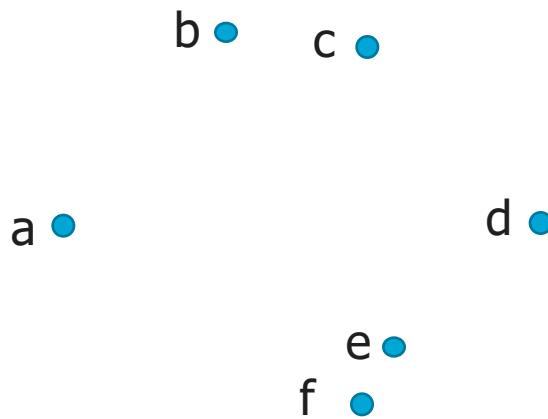
```
1: procedure SparseFLIP( )
2:    $P \leftarrow$  initial points
3:    $V \leftarrow$  GVDB structure
4:   for each frame do
5:     if first frame then
6:        $V_{topo} \leftarrow$  full rebuild ( $P$ )
7:     else
8:        $V_{topo} \leftarrow$  incremental build ( $P$ )
9:     end if
10:     $V \leftarrow$  resize and clear ( $V_{topo}$ )
11:     $S \leftarrow$  insert points in subcells ( $V, P$ )
12:     $V(vel) \leftarrow$  particles-to-voxels ( $S, P$ )
13:     $V \leftarrow$  update apron ( $\rho, vel, marker$ )
14:     $V(vel_{old}) \leftarrow V(vel)$ 
15:     $V(div) \leftarrow$  divergence ( $V(vel)$ )
16:     $V(\rho) \leftarrow$  CG pressure solve ( $V, div$ )
17:     $V(vel) \leftarrow$  pressure-to-velocity ( $V(\rho)$ )
18:     $V \leftarrow$  update apron ( $V(vel)$ )
19:     $P \leftarrow$  advance ( $V(vel), V(vel_{old})$ )
20:   end for
21: end procedure
```

## Dynamic Topology

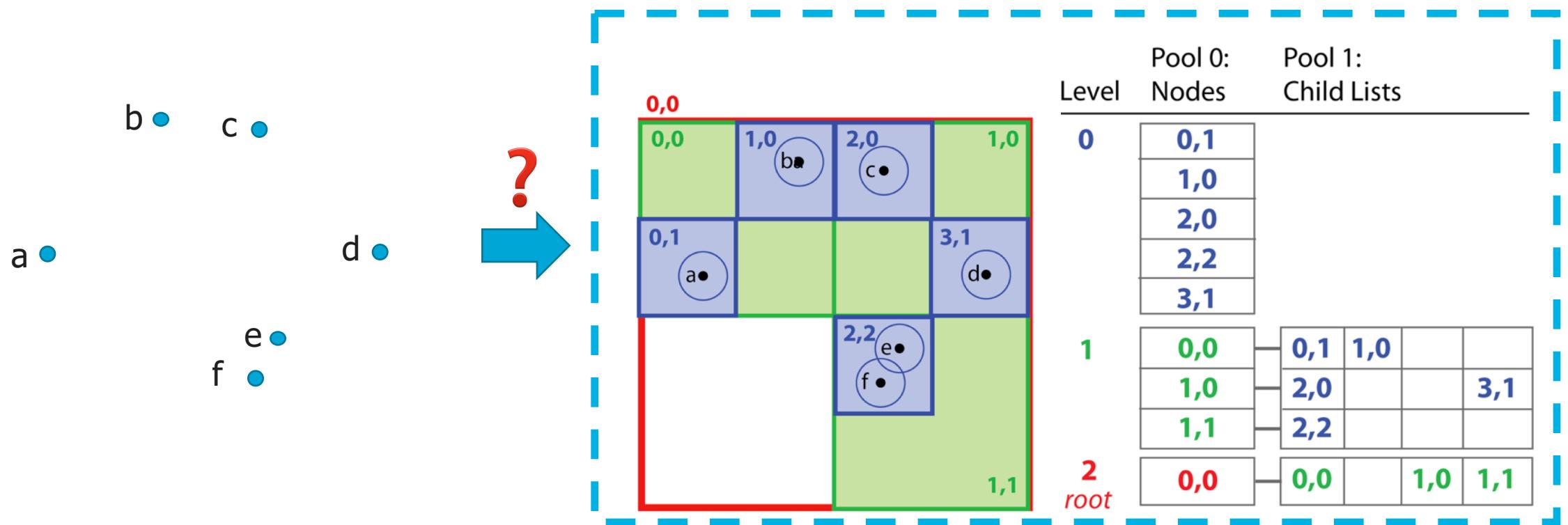
Particles-to-Voxels

## Pressure Solver (CG)

# Full Topology Rebuild

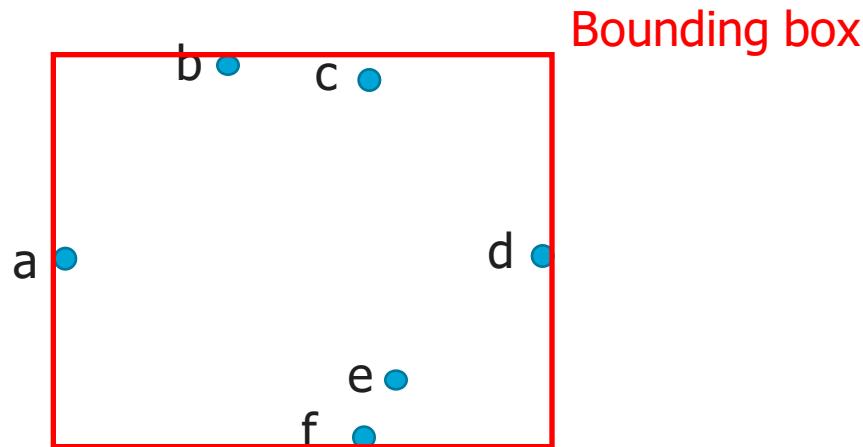


# Full Topology Rebuild



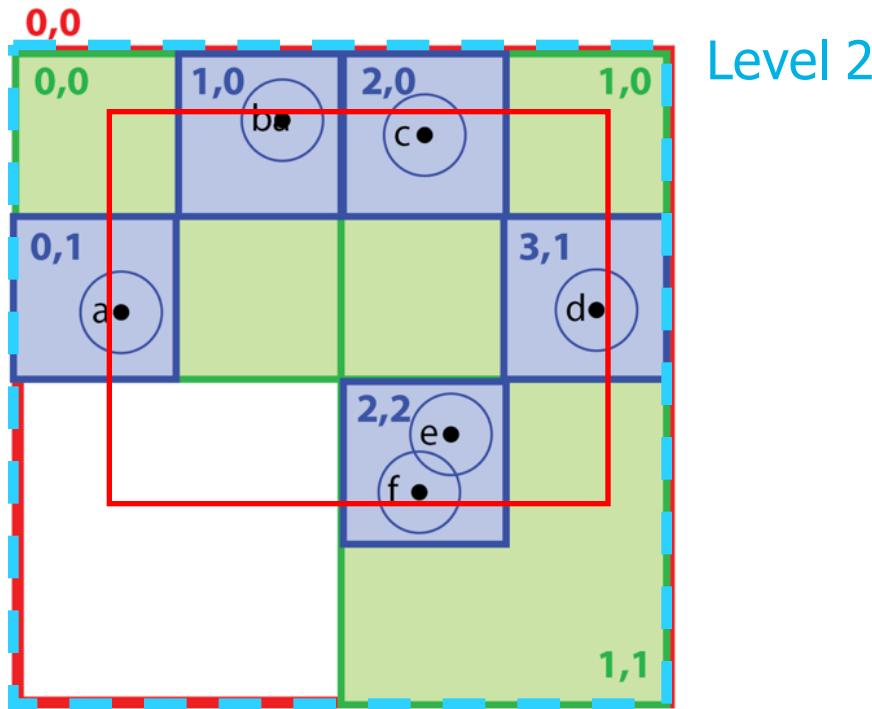
# Full Topology Rebuild

- Determine the number of tree levels L
- Generate the level-index list
- Compact the level-index list
- Allocate and initialize nodes
- Set child node lists



# Full Topology Rebuild

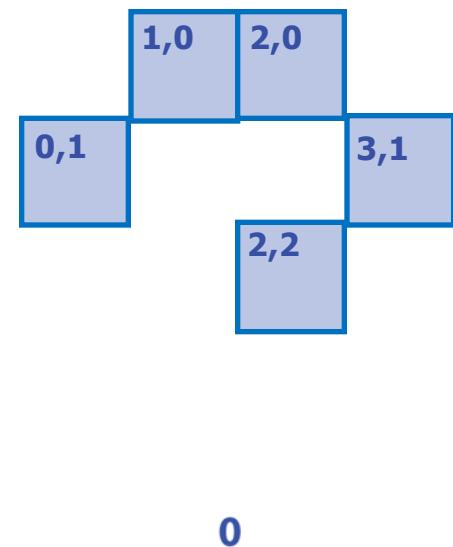
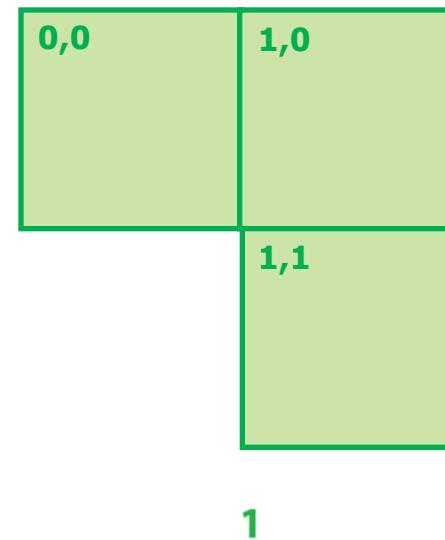
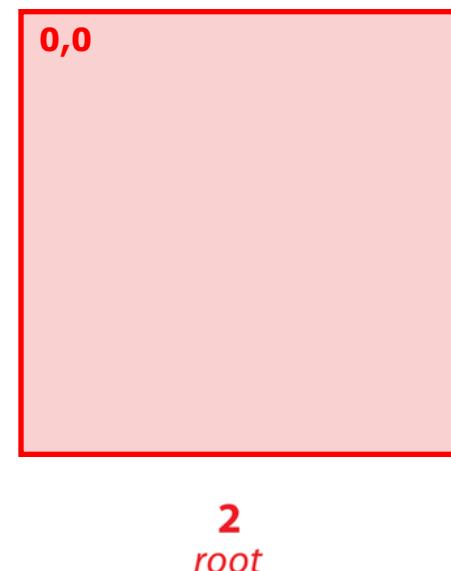
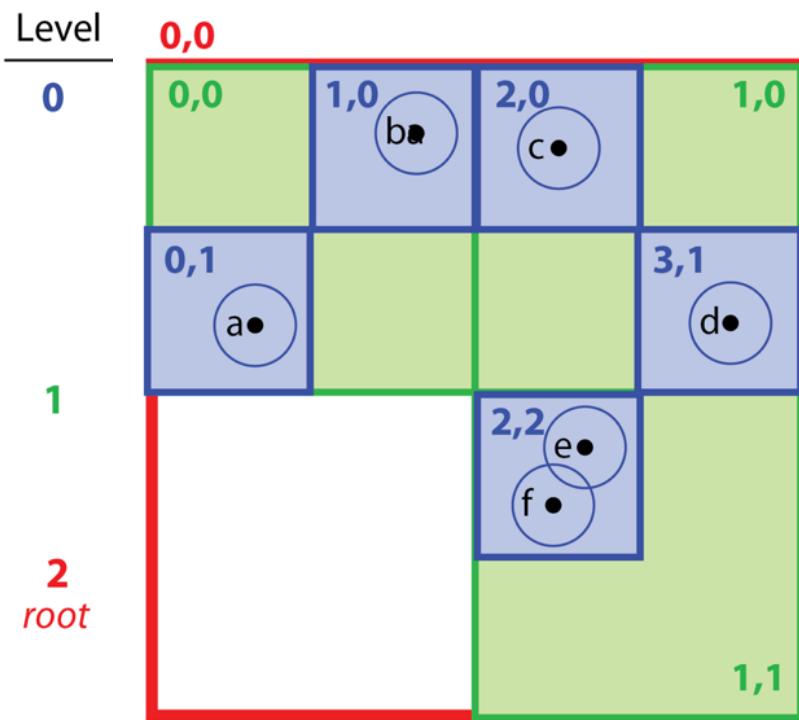
- Determine the number of tree levels L
- Generate the level-index list
- Compact the level-index list
- Allocate and initialize nodes
- Set child node lists



# Full Topology Rebuild

- Determine the number of tree levels  $L$
- Generate the level-index list
- Compact the level-index list

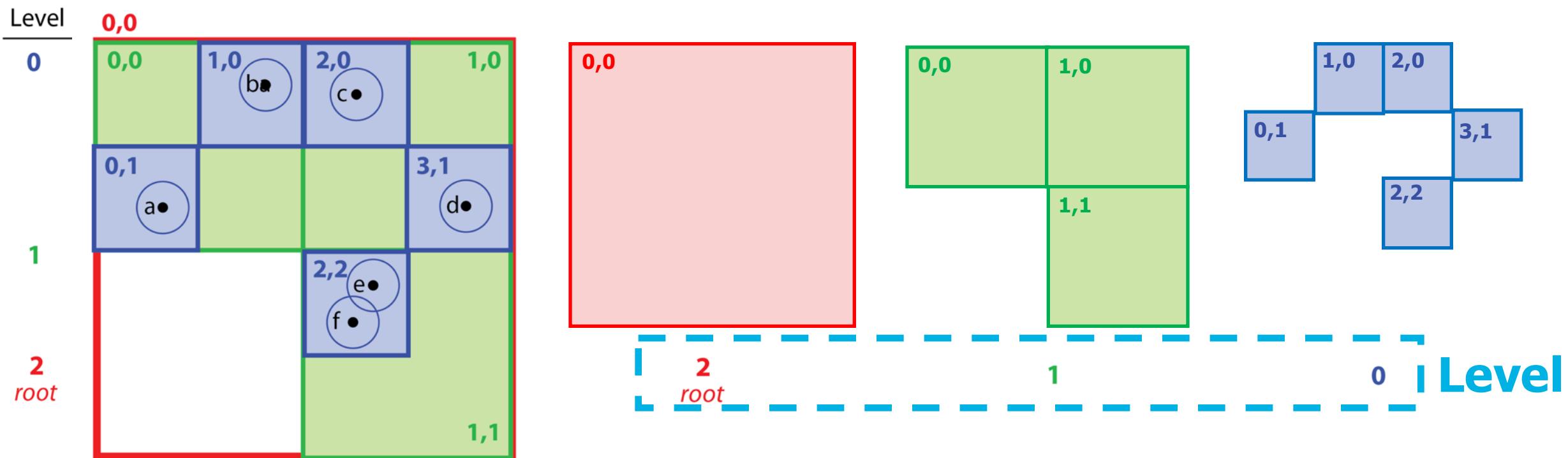
- Allocate and initialize nodes
- Set child node lists



# Full Topology Rebuild

- Determine the number of tree levels  $L$
- Generate the level-index list
- Compact the level-index list

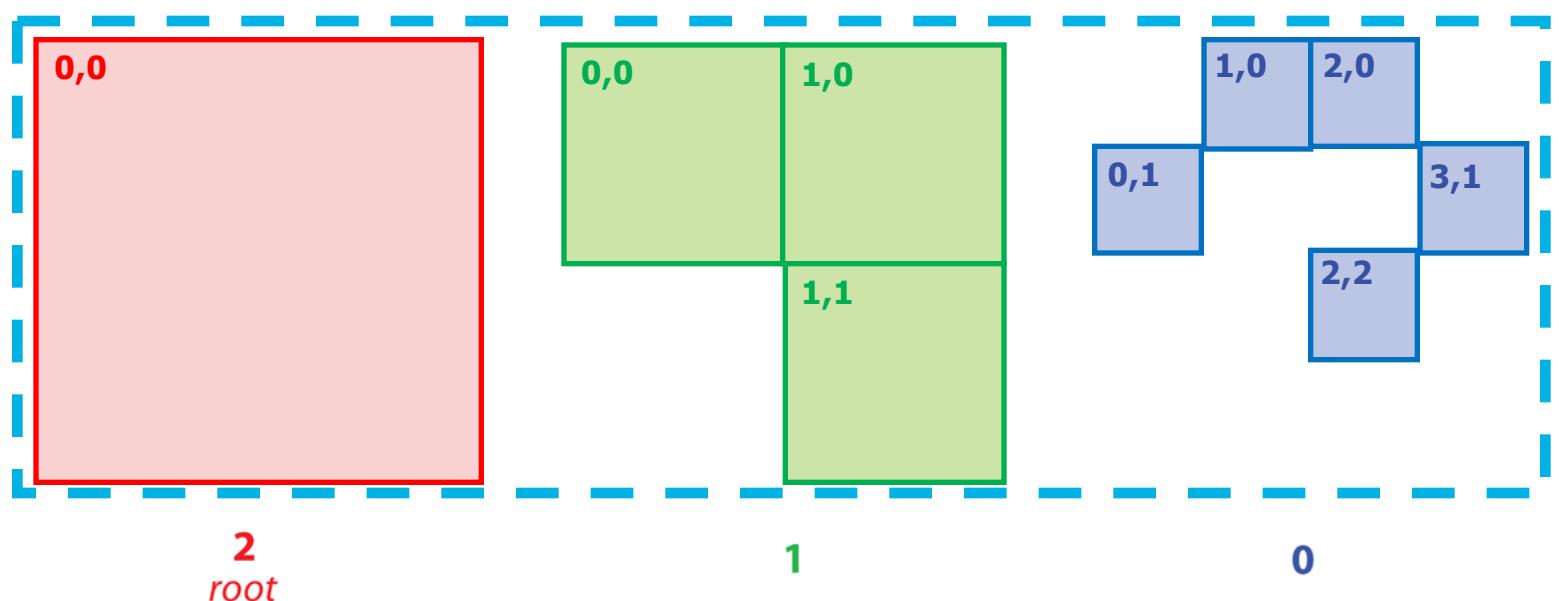
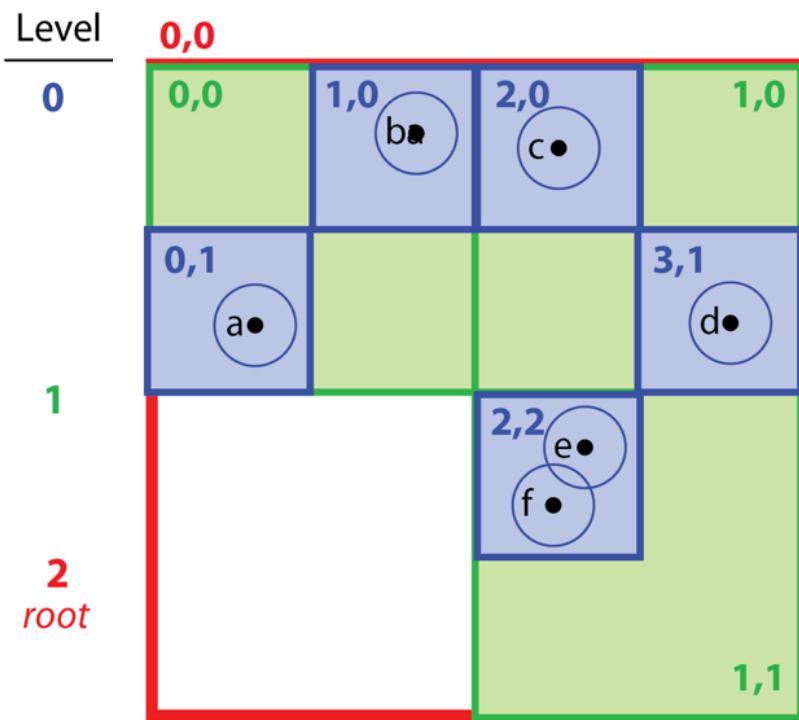
- Allocate and initialize nodes
- Set child node lists



# Full Topology Rebuild

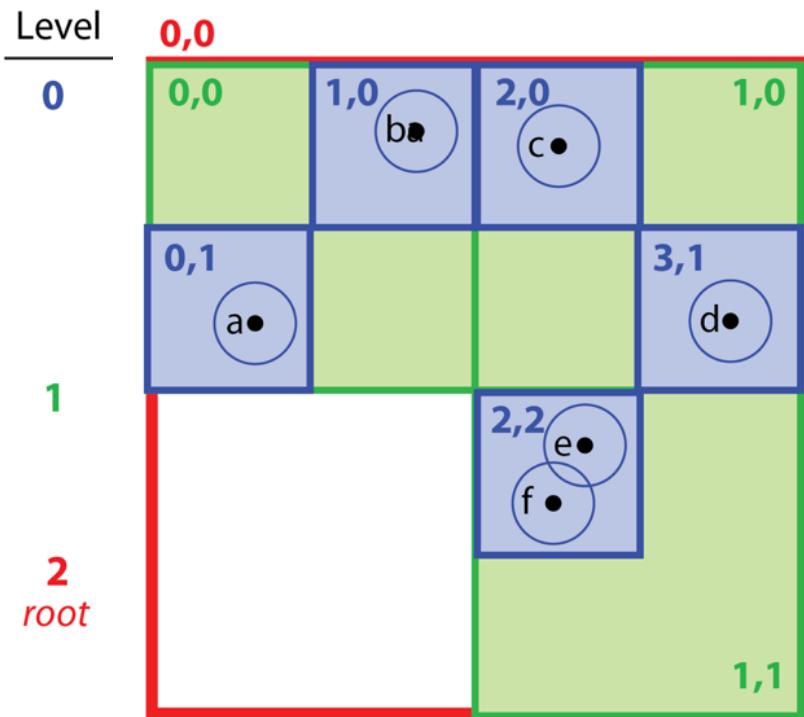
- Determine the number of tree levels  $L$
- Generate the level-index list
- Compact the level-index list

- Allocate and initialize nodes
- Set child node lists



# Full Topology Rebuild

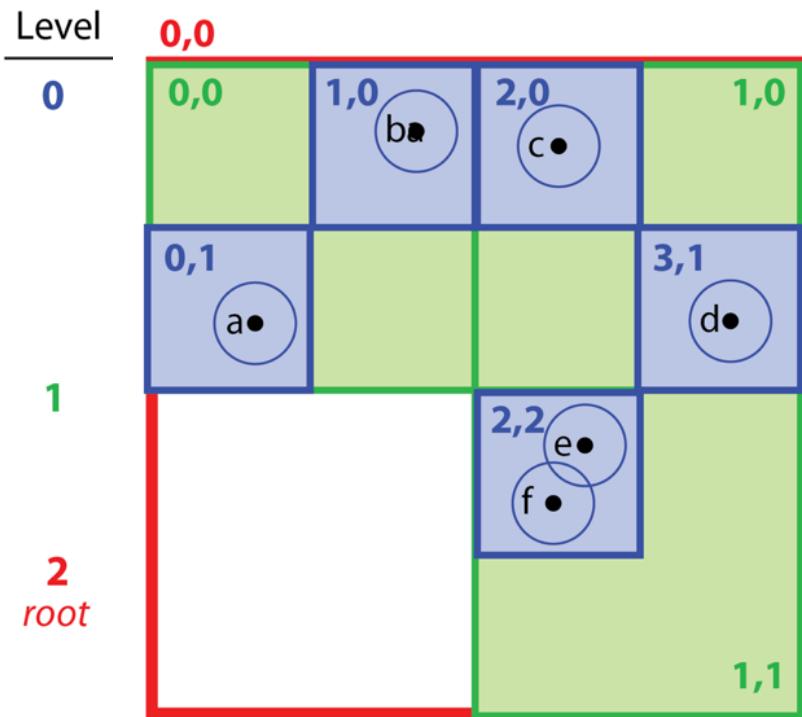
- Determine the number of tree levels L
- Generate the level-index list
- Compact the level-index list



# Full Topology Rebuild

- Determine the number of tree levels L
- Generate the level-index list
- Compact the level-index list

- Allocate and initialize nodes
- Set child node lists



Level-Index  
 $\langle l, x, y \rangle$

a    001

100

**Level + Indexing coordinates**

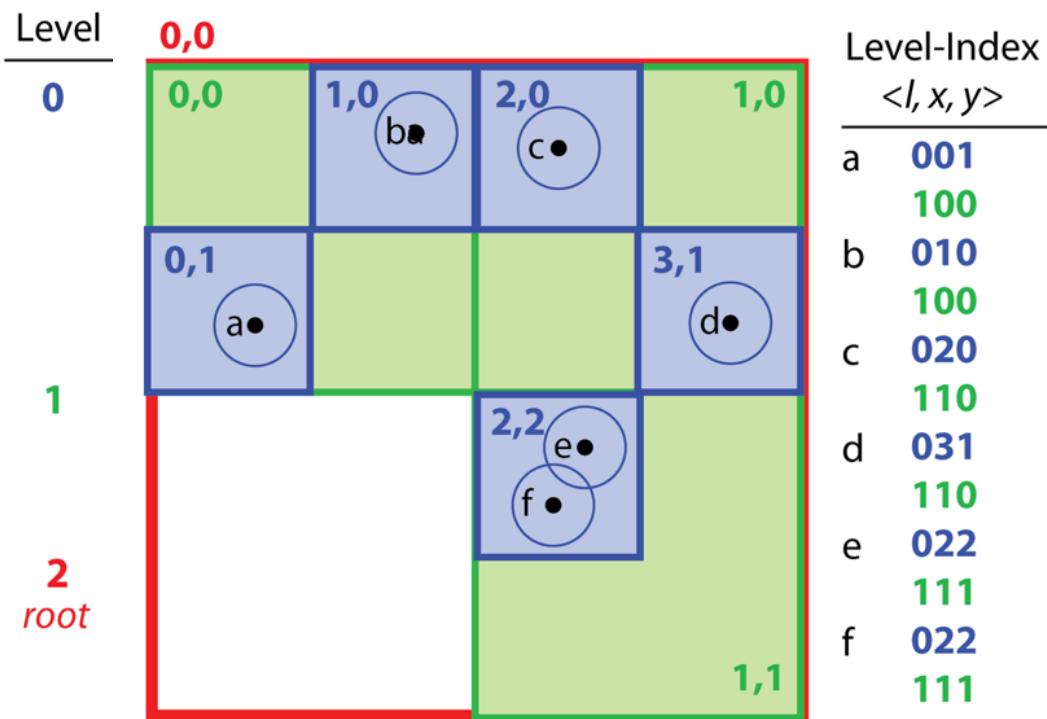
**0 + (0,1)**

**1 + (0,0)**

# Full Topology Rebuild

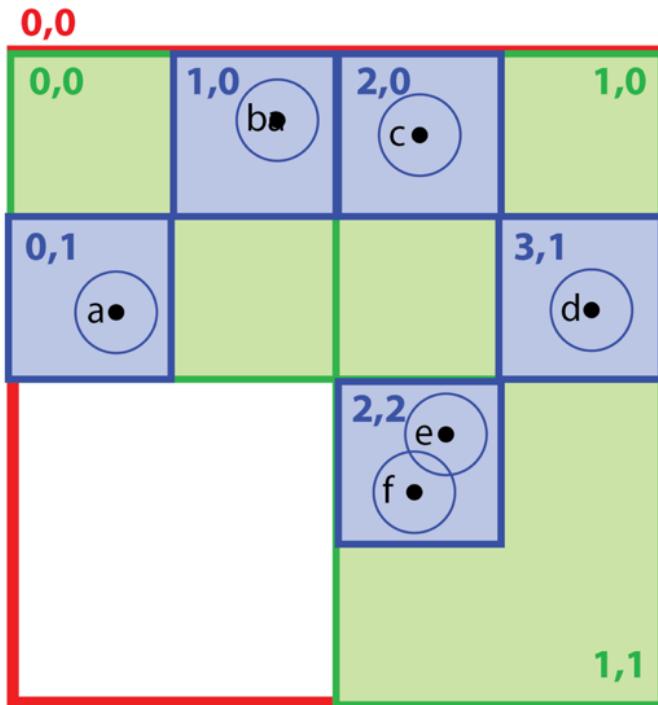
- Determine the number of tree levels  $L$
  - Generate the level-index list
  - Compact the level-index list

- Allocate and initialize nodes
  - Set child node lists



# Full Topology Rebuild

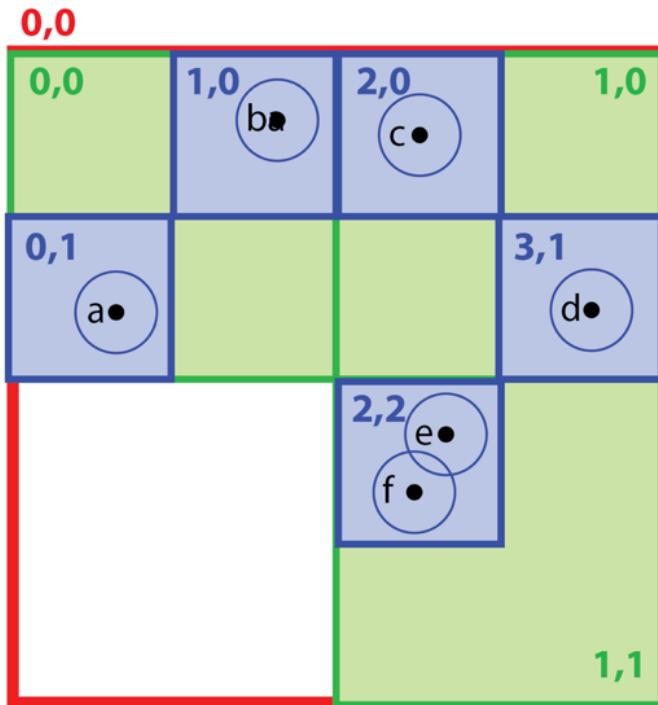
- Determine the number of tree levels L
- Generate the level-index list
- Compact the level-index list
- Allocate and initialize nodes
- Set child node lists



Level-Index $\langle l, x, y \rangle$	Sorted List	Marker List	Prefix Sum	Compact List
a 001	001	1	0 — 0	001
	100	1	1 — 1	010
b 010	020	1	2 — 2	020
	100	1	3 — 3	022
c 020	022	0	3 — 4	031
	110	1	4 — 5	100
d 031	100	1	5 — 6	110
	110	0	5 — 7	111
e 022	110	1	6 —	
	111	0	6 —	
f 022	111	1	7 —	
	111	0	7 —	

# Full Topology Rebuild

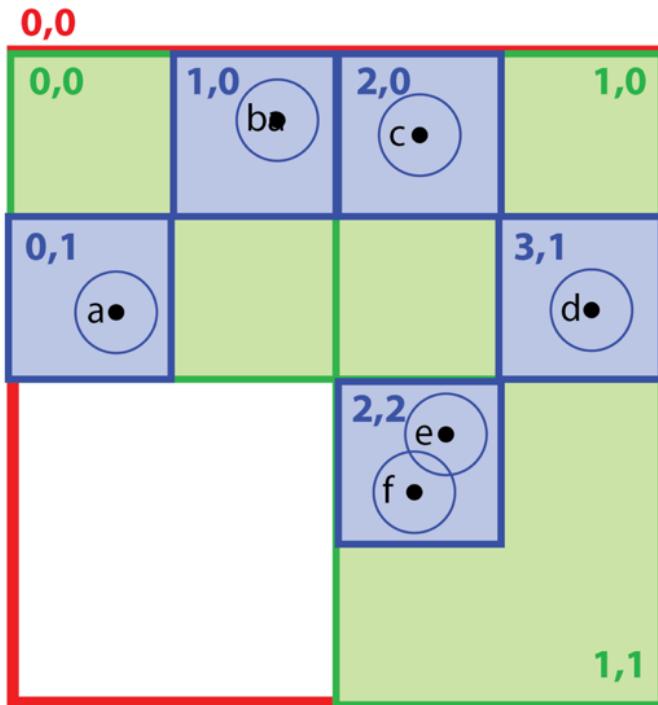
- Determine the number of tree levels L
- Generate the level-index list
- Compact the level-index list
- Allocate and initialize nodes
- Set child node lists



	Level-Index $\langle l, x, y \rangle$	Sorted List	Marker List	Prefix Sum	Compact List
a	001	001	1	0 — 0	001
	100	010	1	1 — 1	010
b	010	020	1	2 — 2	020
	100	022	1	3 — 3	022
c	020	022	0	3 — 4	031
	110	031	1	4 — 5	100
d	031	100	1	5 — 6	110
	110	100	0	5 — 7	111
e	022	110	1	6 —	
	111	110	0	6 —	
f	022	111	1	7 —	
	111	111	0	7 —	

# Full Topology Rebuild

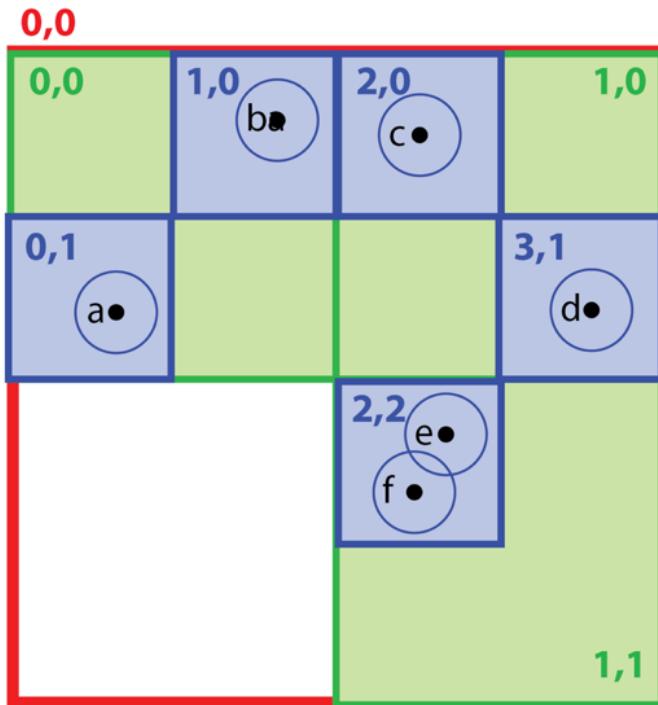
- Determine the number of tree levels L
- Generate the level-index list
- Compact the level-index list
- Allocate and initialize nodes
- Set child node lists



	Level-Index $\langle l, x, y \rangle$	Sorted List	Marker List	Prefix Sum	Compact List
a	001	001	1	0	0 001
	100	010	1	1	1 010
b	010	020	1	2	2 020
	100	022	1	3	3 022
c	020	022	0	3	4 031
	110	031	1	4	5 100
d	031	100	1	5	6 110
	110	100	0	5	7 111
e	022	110	1	6	
	111	110	0	6	
f	022	111	1	7	
	111	111	0	7	

# Full Topology Rebuild

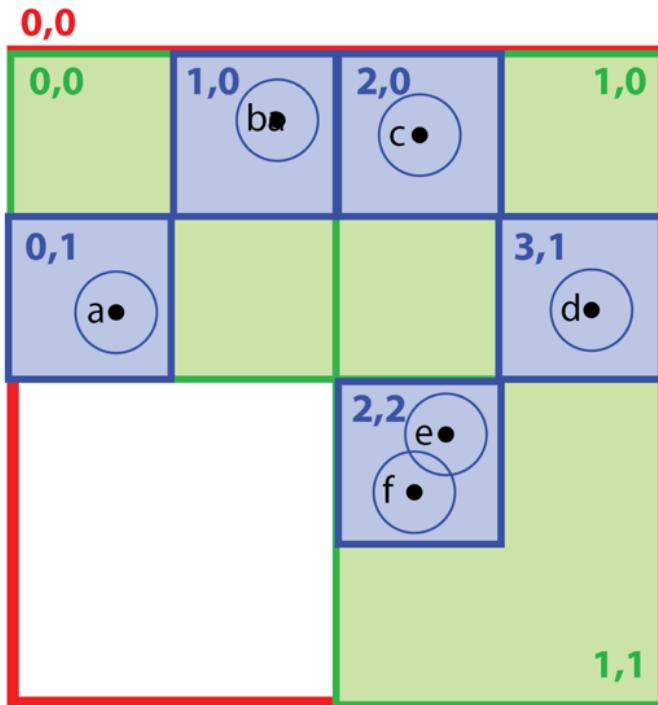
- Determine the number of tree levels L
- Generate the level-index list
- Compact the level-index list
- Allocate and initialize nodes
- Set child node lists



	Level-Index $\langle l, x, y \rangle$	Sorted List	Marker List	Prefix Sum	Compact List
a	001 100	001 010	1 1	0 1	0 001 1 010
b	010 100	020 022	1 1	2 3	2 020 3 022
c	020 110	022 031	0 1	3 4	4 031 5 100
d	031 110	100 100	1 0	5 5	6 110 7 111
e	022 111	110 110	1 0	6 6	
f	022 111	111 111	1 0	7 7	

# Full Topology Rebuild

- Determine the number of tree levels L
- Generate the level-index list
- Compact the level-index list
- Allocate and initialize nodes
- Set child node lists

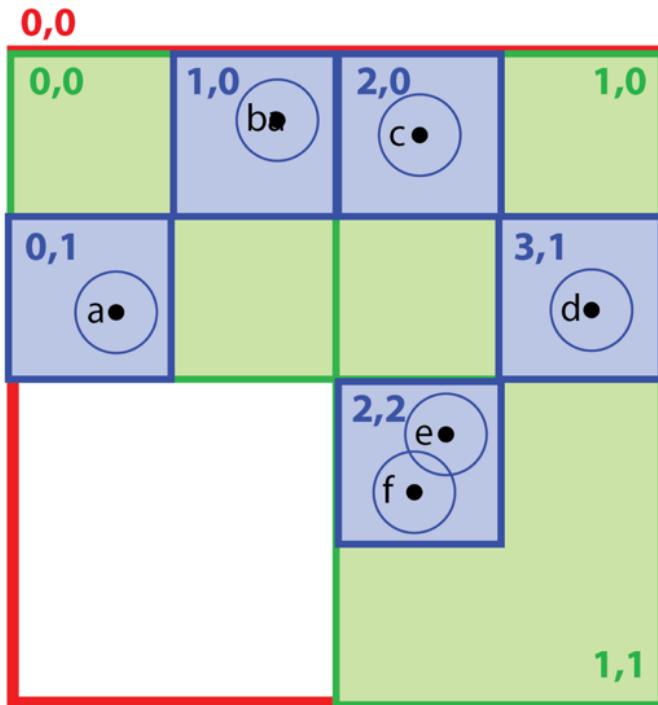


	Level-Index $\langle l, x, y \rangle$	Sorted List	Marker List	Prefix Sum	Compact List	Level	Pool 0: Nodes	Pool 1: Child Lists
a	001	001	1	0	0 001	0	0,1	
	100	010	1	1	1 010		1,0	
	010	020	1	2	2 020		2,0	
		100	1	3	3 022		2,2	
	020	022	0	3	3 031		3,1	
		110	1	4	4 100			
	031	100	1	5	5 110			
		110	0	5	6 111			
	022	110	1	6		1	0,0	
		111	0	6			1,0	
e	022	111	1	7		2	1,1	
	111	111	0	7			0,0	

root

# Full Topology Rebuild

- Determine the number of tree levels L
- Generate the level-index list
- Compact the level-index list
- Allocate and initialize nodes
- Set child node lists

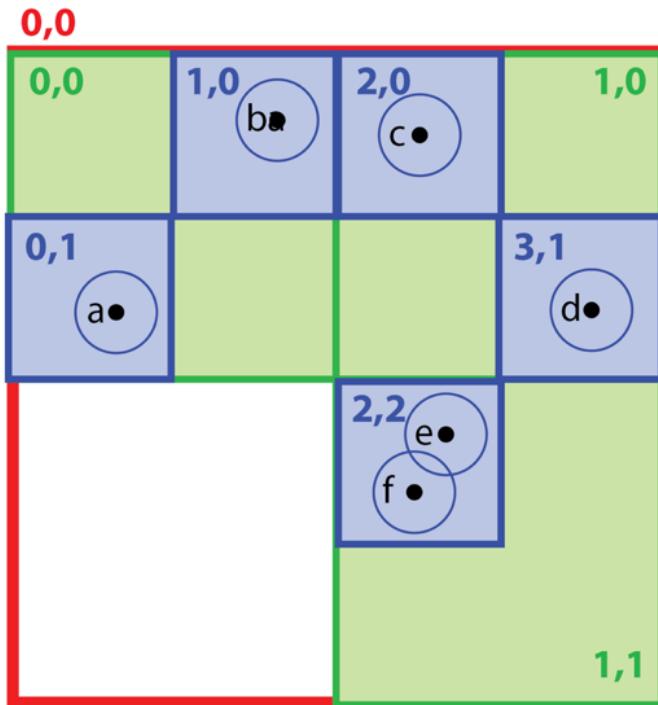


	Level-Index $\langle l, x, y \rangle$	Sorted List	Marker List	Prefix Sum	Compact List	Level	Pool 0: Nodes	Pool 1: Child Lists
a	001	001	1	0	0 001	0	0,1	
	100	010	1	1	1 010		1,0	
	010	020	1	2	2 020		2,0	
		100	1	3	3 022		2,2	
	020	022	0	3	3 031		3,1	
		110	1	4	4 100			
	031	100	1	5	5 110			
		110	0	5	6 111			
	022	110	1	6		1	0,0	
		111	0	6			1,0	
e	022	111	1	7			1,1	
	111	111	0	7			0,0	

The table shows the compacted level-index list. The 'Compact List' column contains the resulting binary strings. The 'Level' column indicates the level of each node. The 'root' node is highlighted in red.

# Full Topology Rebuild

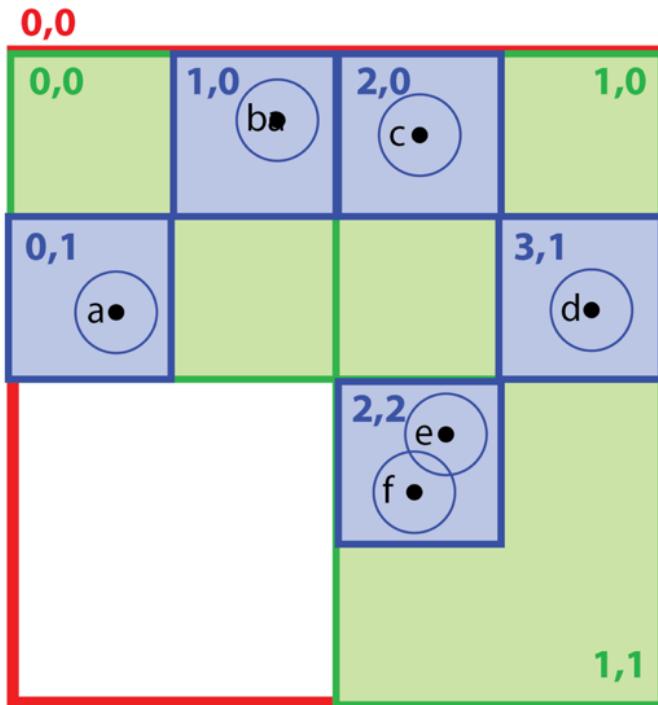
- Determine the number of tree levels L
- Generate the level-index list
- Compact the level-index list
- Allocate and initialize nodes
- Set child node lists



	Level-Index $\langle l, x, y \rangle$	Sorted List	Marker List	Prefix Sum	Compact List	Level	Pool 0: Nodes	Pool 1: Child Lists
a	001	001	1	0	0 001	0	0,1	
	100	010	1	1	1 010		1,0	
	010	020	1	2	2 020		2,0	
		100	1	3	3 022		2,2	
	020	022	0	3	3 031		3,1	
		110	1	4	4 100			
	031	031	1	5	5 110			
		100	1	5	5 111			
	022	100	0	6	6	1	0,0	
		110	1	6	6		1,0	
	022	110	0	6	6		1,1	
		111	1	7	7		0,0	0,0 1,0 1,1
<i>root</i>								

# Full Topology Rebuild

- Determine the number of tree levels L
- Generate the level-index list
- Compact the level-index list
- Allocate and initialize nodes
- Set child node lists

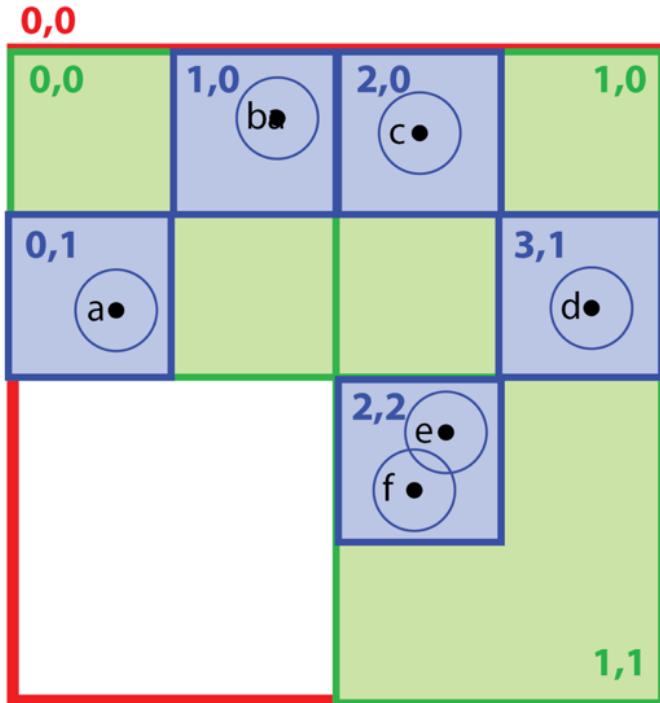


	Level-Index $\langle l, x, y \rangle$	Sorted List	Marker List	Prefix Sum	Compact List	Level	Pool 0: Nodes	Pool 1: Child Lists
a	001	001	1	0	0 001	0	0,1	
	100	010	1	1	1 010		1,0	
	010	020	1	2	2 020		2,0	
		100	1	3	3 022		2,2	
	020	022	0	3	3 031		3,1	
		110	1	4	4 100			
	031	100	1	5	5 110		0,0	0,1 1,0
		110	0	5	6 111		1,0	2,0
	022	110	1	6	6		1,1	3,1
		111	0	6			2,2	
	022	111	1	7	7		0,0	0,0 1,0 1,1
		111	0	7				

root

# Full Topology Rebuild

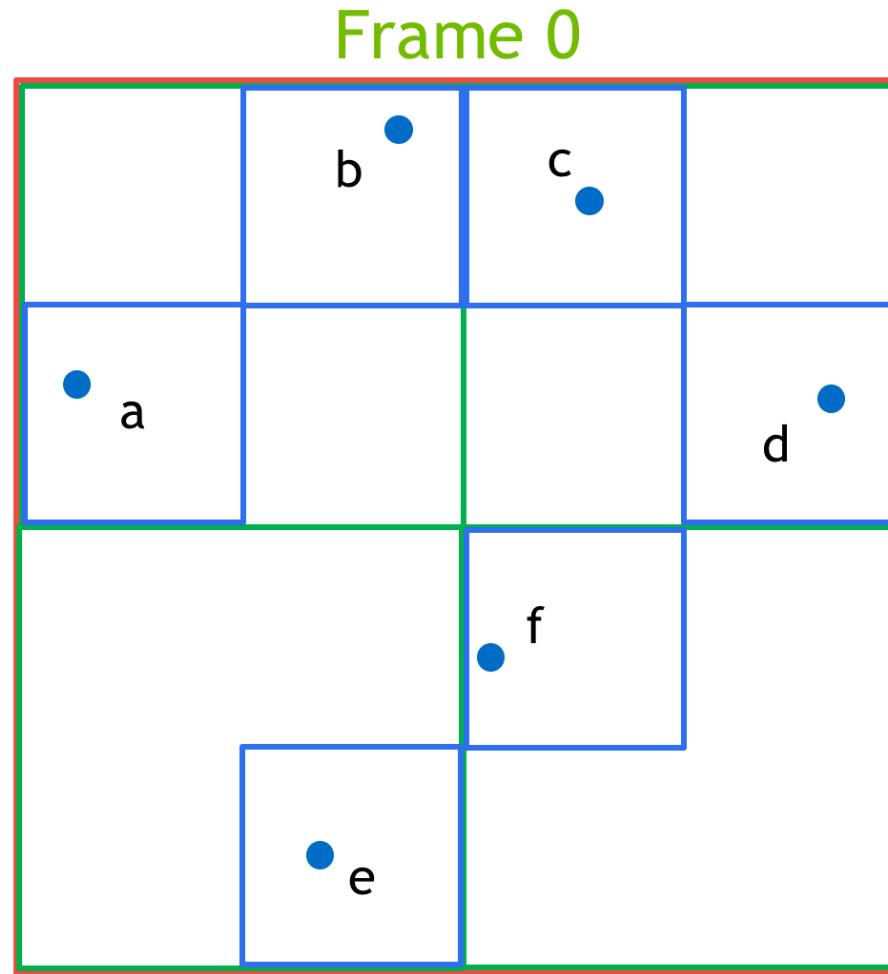
- Determine the number of tree levels L
- Generate the level-index list
- Compact the level-index list
- Allocate and initialize nodes
- Set child node lists



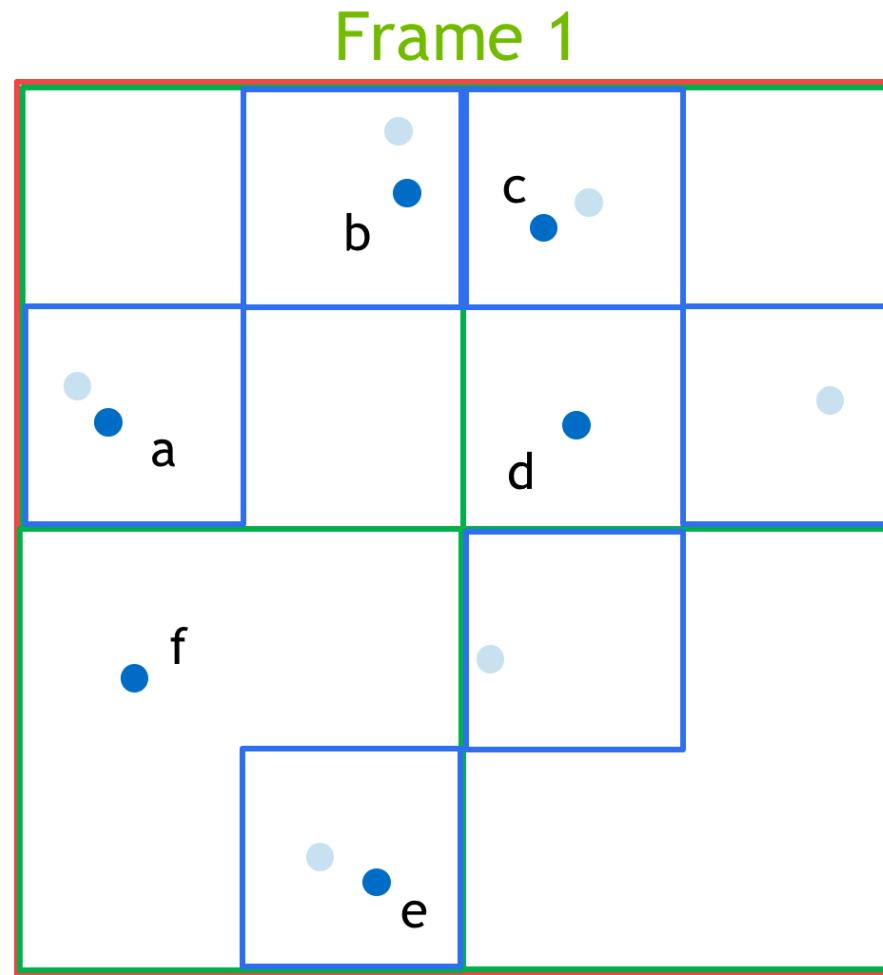
Very long list       $n=P*L$

Level-Index	Sorted List	Marker List	Prefix Sum	Compact List	Pool 0: Nodes	Pool 1: Child Lists
$\langle l, x, y \rangle$	List	List	Sum	List	Level	
a 001	001	1	0	0 001	0	0,1
	010	1	1	1 010		1,0
b 010	020	1	2	2 020	1	2,0
	022	1	3	3 022		2,2
c 020	022	0	3	3 031	2	3,1
	031	1	4	4 100		
d 031	100	1	5	5 110	1	0,0 0,1 1,0
	100	0	5	5 111		2,0 3,1
e 022	110	1	6	6	1	1,0 2,0 2,2
	110	0	6			1,1
f 022	111	1	7		2	0,0 0,0 1,0 1,1
	111	0	7		root	

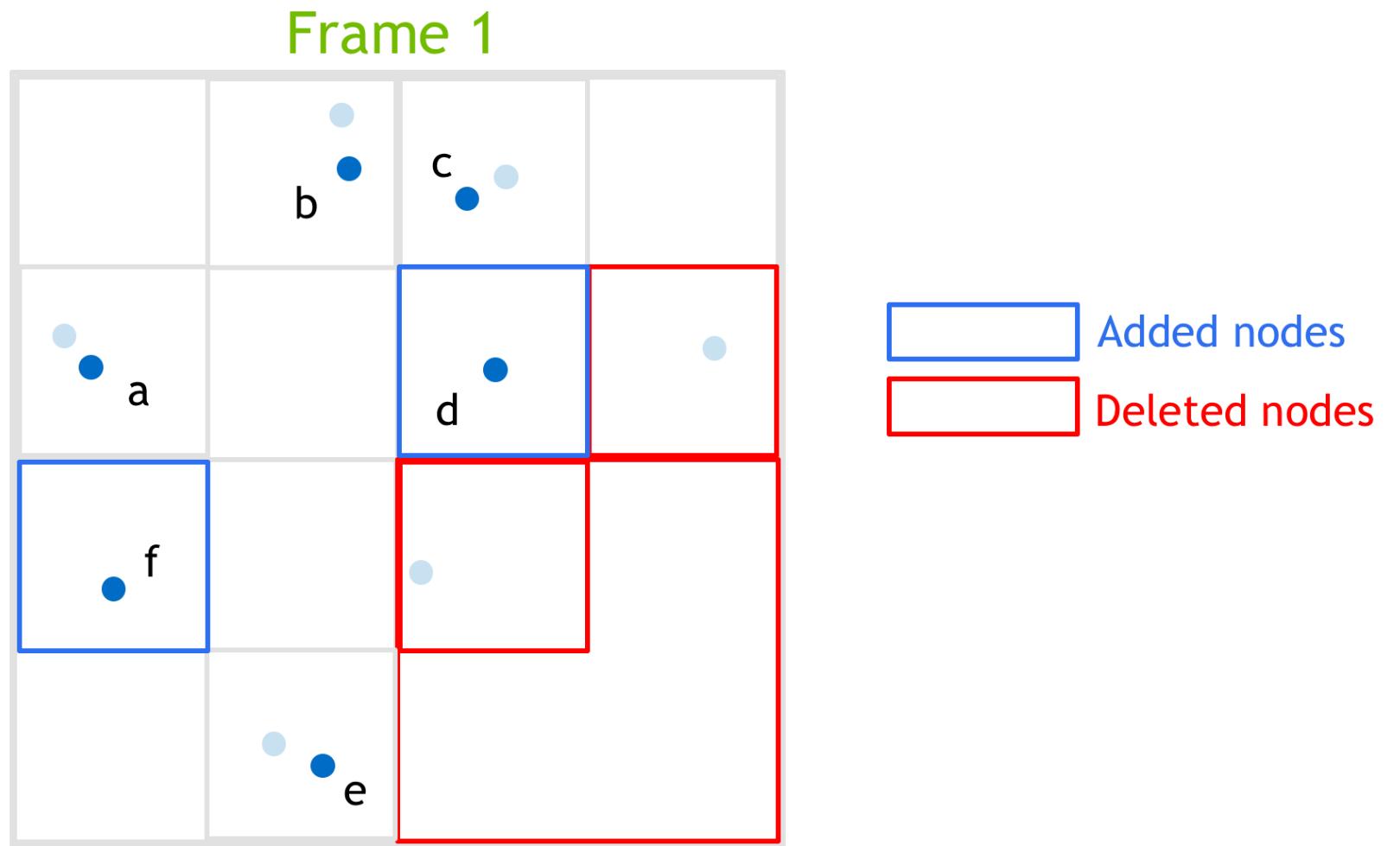
# Incremental Rebuild



# Incremental Rebuild

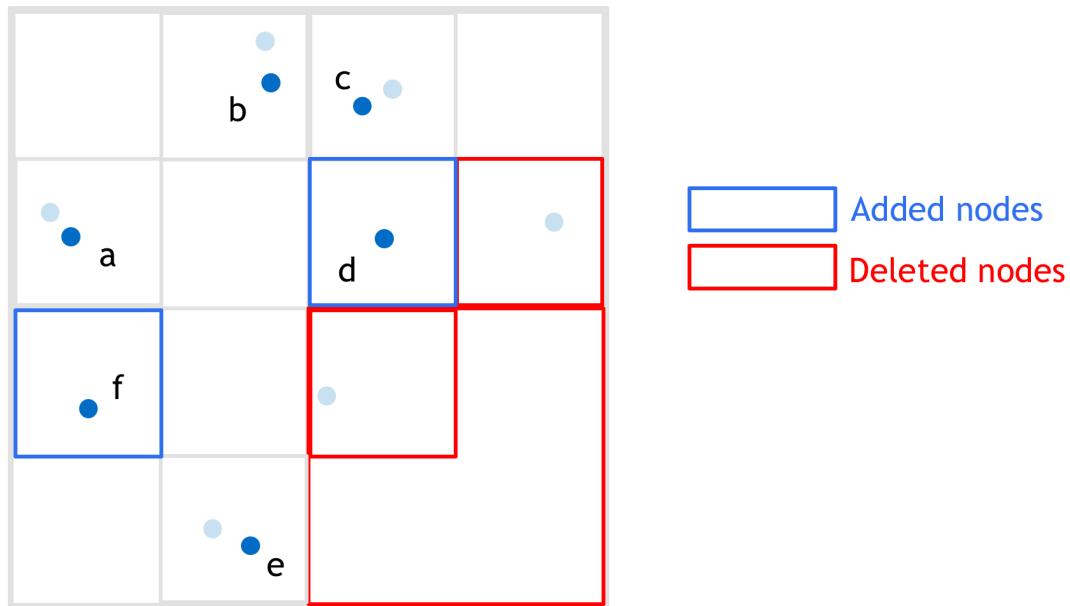


# Incremental Rebuild



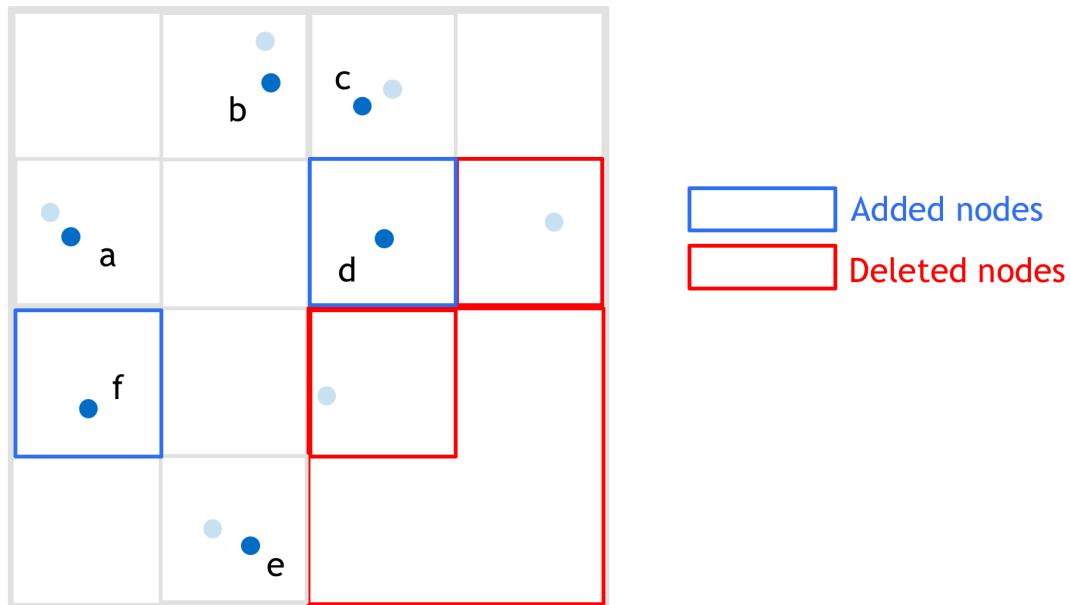
# Incremental Rebuild

- Determine the number of tree levels L
- Generate the level-index list
- Compact the level-index list
- Allocate and initialize nodes
- Set child node lists



# Incremental Rebuild

- Determine the number of tree levels L
- Generate the level-index list\*
- Compact the level-index list
- Allocate and initialize nodes
- Set child node lists\*

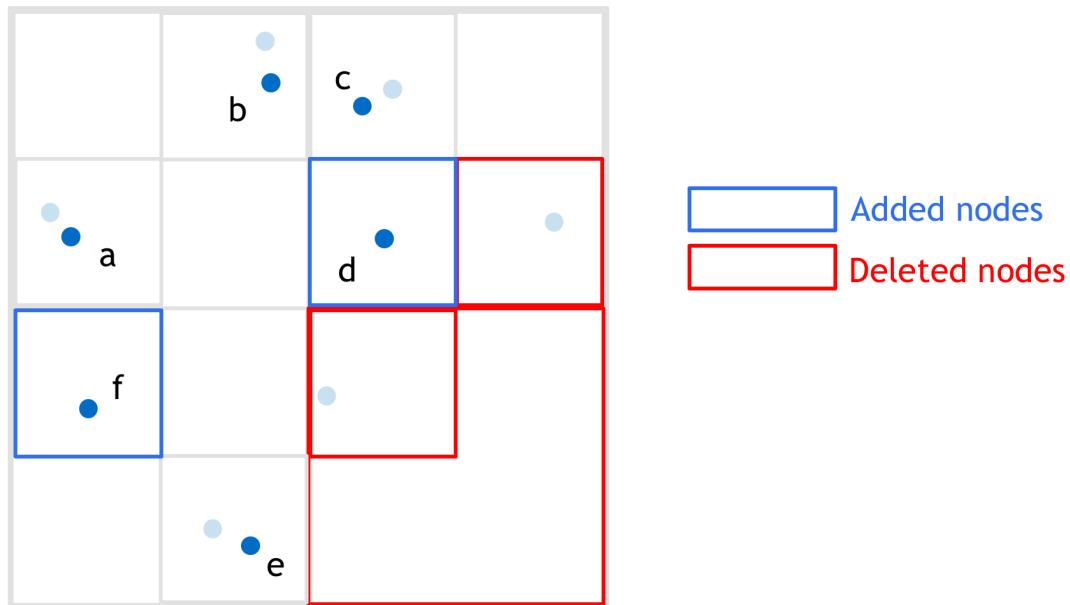


For each particle, check whether already exists a node at that level

- if so, mark the node
- otherwise, add to level-index list

# Incremental Rebuild

- Determine the number of tree levels L
- Generate the level-index list\*
- Compact the level-index list
- Allocate and initialize nodes
- Set child node lists\*



- For each particle, check whether already exists a node at that level
- if so, mark the node
  - otherwise, add to level-index list

# Topology Construction

Particles #	CPU Full	GPU Full	GPU Incremental
4M	384	56 (7×)	<b>4.8</b> (80×)
8M	807	96 (8×)	<b>5.6</b> (144×)
16M	1598	204 (8×)	<b>9.6</b> (167×)
32M	3249	313 (10×)	<b>18.7</b> (174×)
65M	6368	366 (17×)	<b>35.5</b> (179×)

in millisecond

# FLIP with Sparse Volumes on the GPU

```
1: procedure SparseFLIP( )
2:    $P \leftarrow$  initial points
3:    $V \leftarrow$  GVDB structure
4:   for each frame do
5:     if first frame then
6:        $V_{topo} \leftarrow$  full rebuild ( $P$ )
7:     else
8:        $V_{topo} \leftarrow$  incremental build ( $P$ )
9:     end if
10:     $V \leftarrow$  resize and clear ( $V_{topo}$ )
11:     $S \leftarrow$  insert points in subcells ( $V, P$ )
12:     $V(vel) \leftarrow$  particles-to-voxels ( $S, P$ )
13:     $V \leftarrow$  update apron ( $\rho, vel, marker$ )
14:     $V(vel_{old}) \leftarrow V(vel)$ 
15:     $V(div) \leftarrow$  divergence ( $V(vel)$ )
16:     $V(\rho) \leftarrow$  CG pressure solve ( $V, div$ )
17:     $V(vel) \leftarrow$  pressure-to-velocity ( $V(\rho)$ )
18:     $V \leftarrow$  update apron ( $V(vel)$ )
19:     $P \leftarrow$  advance ( $V(vel), V(vel_{old})$ )
20:   end for
21: end procedure
```

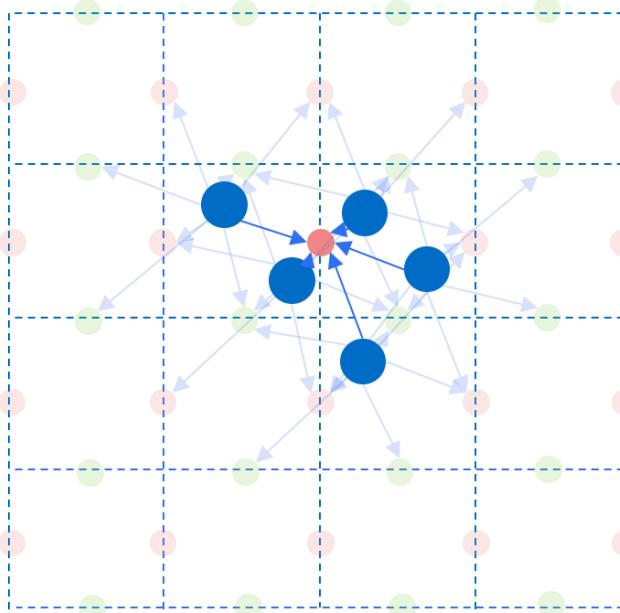
Dynamic Topology

Particles-to-Voxels

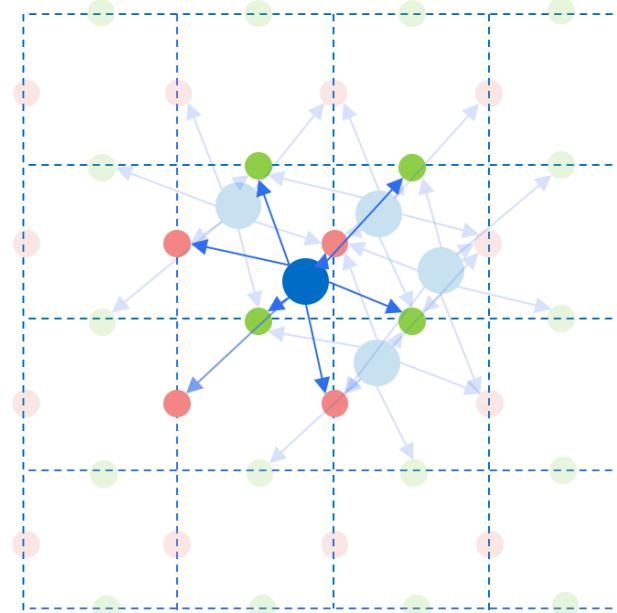
Pressure Solver (CG)

# Particles-to-Voxels

Gathering

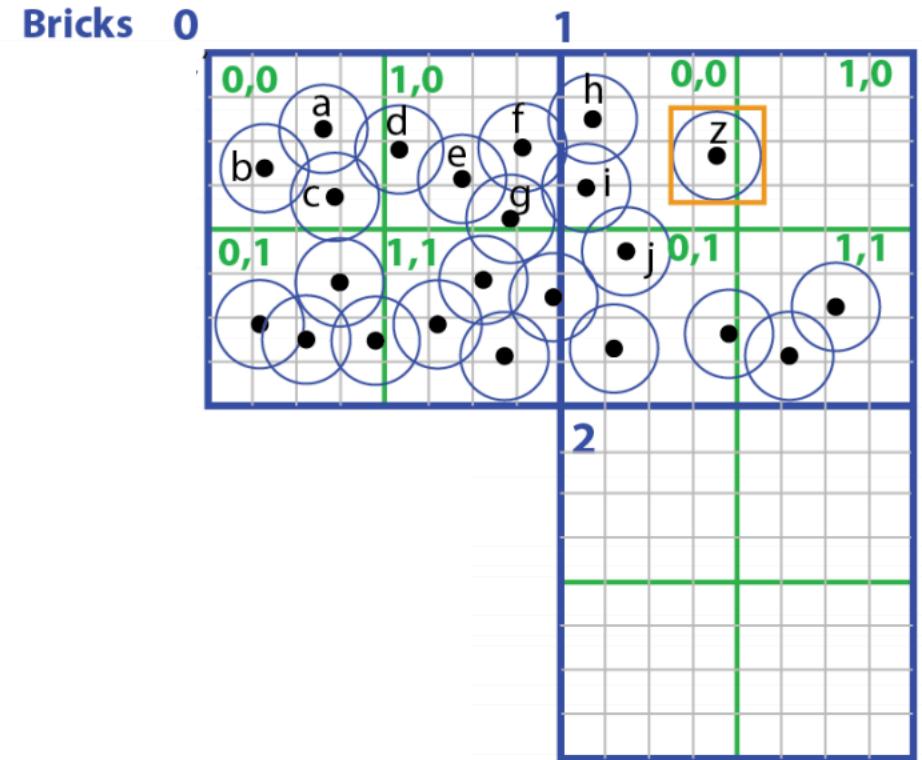


Scattering

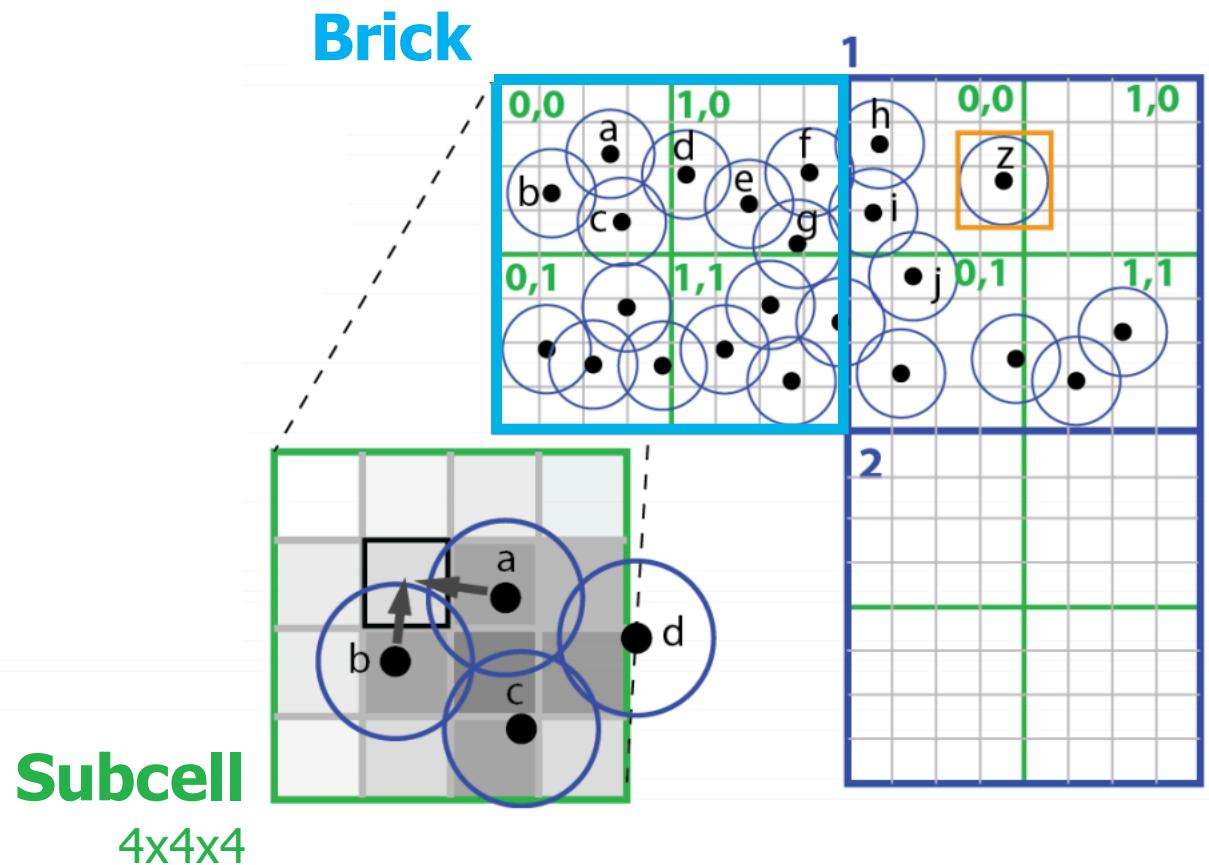


Better for texture writing in parallel

# Subcell



# Subcell



# Subcell

Subcell Count & Scan

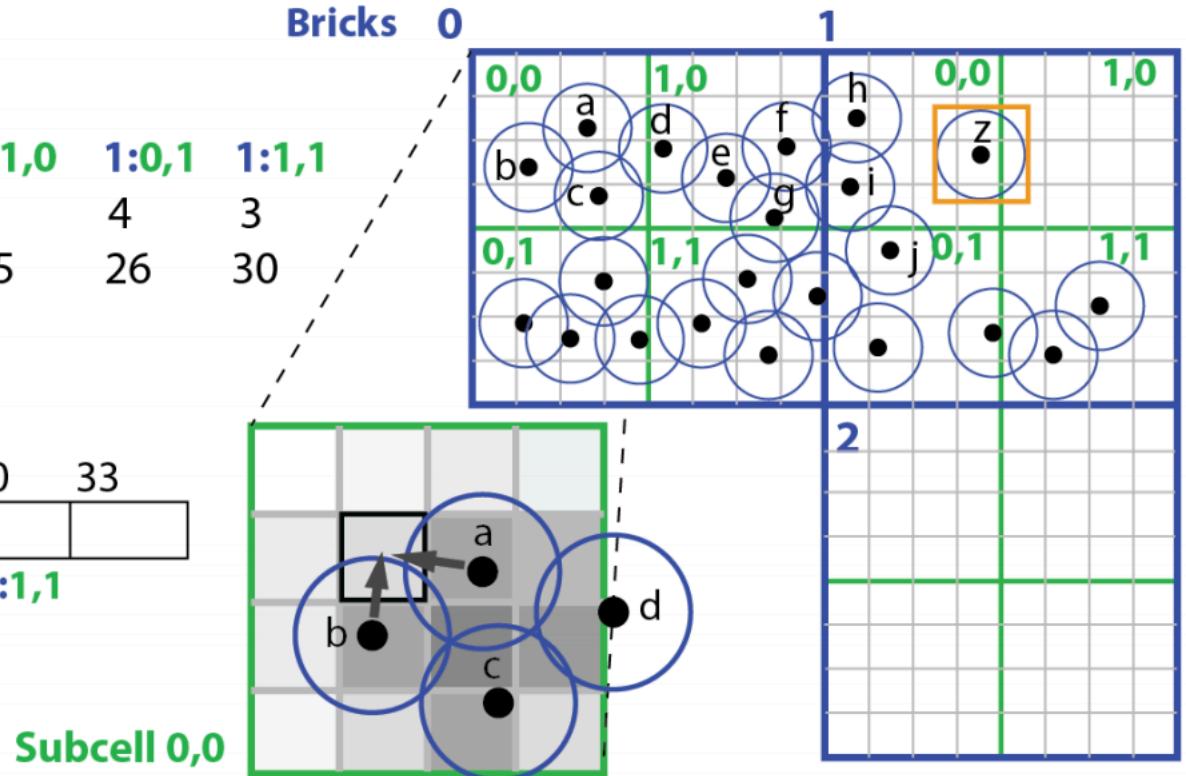
Brick	0	1						
Subcell	0:0,0	0:1,0	0:0,1	0:1,1	1:0,0	1:1,0	1:0,1	1:1,1
Count	4	6	5	6	4	1	4	3
Scan	0	4	10	15	21	25	26	30

Subcell Particle List

0	4	10	15	21	25	30	33
a b c d	d e f g h i	c.....	...g..	h i j z	z	j...	

0:0,0	0:1,0	0:0,1	0:1,1	1:0,0	1:0,1	1:1,1
-------	-------	-------	-------	-------	-------	-------



# FLIP with Sparse Volumes on the GPU

```
1: procedure SparseFLIP( )
2:    $P \leftarrow$  initial points
3:    $V \leftarrow$  GVDB structure
4:   for each frame do
5:     if first frame then
6:        $V_{topo} \leftarrow$  full rebuild ( $P$ )
7:     else
8:        $V_{topo} \leftarrow$  incremental build ( $P$ )
9:     end if
10:     $V \leftarrow$  resize and clear ( $V_{topo}$ )
11:     $S \leftarrow$  insert points in subcells ( $V, P$ )
12:     $V(vel) \leftarrow$  particles-to-voxels ( $S, P$ )
13:     $V \leftarrow$  update apron ( $\rho, vel, marker$ )
14:     $V(vel_{old}) \leftarrow V(vel)$ 
15:     $V(div) \leftarrow$  divergence ( $V(vel)$ )
16:     $V(\rho) \leftarrow$  CG pressure solve ( $V, div$ )
17:     $V(vel) \leftarrow$  pressure-to-velocity ( $V(\rho)$ )
18:     $V \leftarrow$  update apron ( $V(vel)$ )
19:     $P \leftarrow$  advance ( $V(vel), V(vel_{old})$ )
20:   end for
21: end procedure
```

Dynamic Topology

Particles-to-Voxels

Pressure Solver (CG)

# GPU Matrix-Free Conjugate Gradient Solver

---

**Algorithm 2** Matrix-free Conjugate Gradient Solver

---

```
1: > Given the inputs divergence b, starting value x, maximum of iterations
   imax, and error tolerance ε
2: procedure MatrixFreeConjugateGradientSolver( $b, x, i_{\max}, \epsilon$ )
3:    $i = 0$ 
4:    $r = b - \text{SpMV}(x)$ 
5:    $d = r$ 
6:    $\delta_{\text{new}} = \text{InnerProduct}(r, r)$ 
7:    $\delta_0 = \delta_{\text{new}}$ 
8:   while  $i < i_{\max}$  do
9:     UpdateApron( $d$ )
10:     $q = \text{SpMV}(d)$ 
11:     $\alpha = \delta_{\text{new}} / \text{InnerProduct}(d, q)$ 
12:     $x = x + \alpha d$ 
13:     $r = b - \alpha q$ 
14:     $\delta_{\text{old}} = \delta_{\text{new}}$ 
15:     $\delta_{\text{new}} = \text{InnerProduct}(r, r)$ 
16:     $\beta = \delta_{\text{new}} / \delta_{\text{old}}$ 
17:     $d = r + \beta d$ 
18:     $i = i + 1$ 
19:    if  $i \bmod 10 == 0$  then
20:      if  $\delta_{\text{new}} > \epsilon^2 \delta_0$  then > Read back to CPU for check
21:        Stop solver and return
22:      end if
23:    end if
24:  end while
25: end procedure
```

---

# GPU Matrix-Free Conjugate Gradient Solver

---

**Algorithm 2** Matrix-free Conjugate Gradient Solver

---

```
1: Given the inputs divergence b, starting value x, maximum of iterations
   imax, and error tolerance ε
2: procedure MatrixFreeConjugateGradientSolver(b,x,imax,ε)
3:   i = 0
4:   r = b - SpMV(x)
5:   d = r
6:   δnew = InnerProduct(r,r)
7:   δ0 = δnew
8:   while i < imax do
9:     UpdateApron(d)
10:    q = SpMV(d)
11:    α = δnew/InnerProduct(d,q)
12:    x = x + αd
13:    r = b - αq
14:    δold = δnew
15:    δnew = InnerProduct(r,r)
16:    β = δnew/δold
17:    d = r + βd
18:    i = i + 1
19:    if i mod 10 == 0 then
20:      if δnew > ε2δ0 then      ▷ Read back to CPU for check
21:        Stop solver and return
22:      end if
23:    end if
24:  end while
25: end procedure
```

---

# Sparse Matrix Vector Multiplication (SpMV)

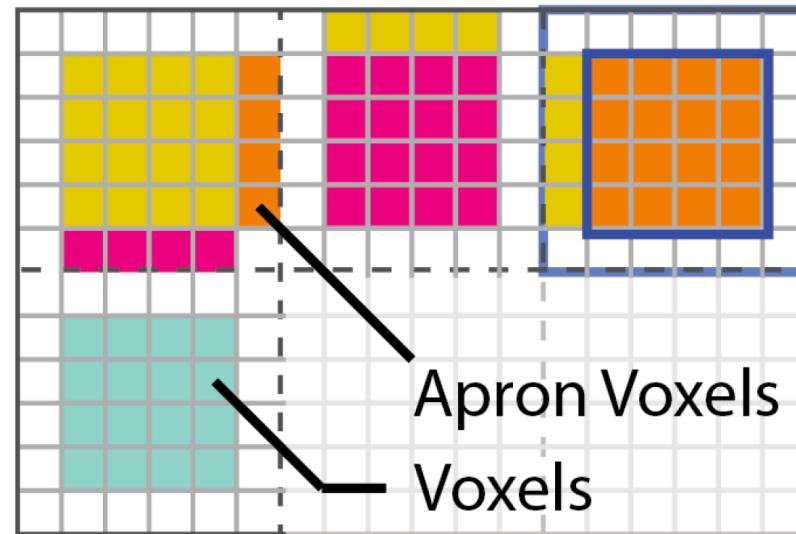
---

**Algorithm 2** Matrix-free Conjugate Gradient Solver

```
1: ▷ Given the inputs divergence  $b$ , starting value  $x$ , maximum of iterations  
    $i_{\max}$ , and error tolerance  $\epsilon$   
2: procedure MatrixFreeConjugateGradientSolver( $b, x, i_{\max}, \epsilon$ )  
3:    $i = 0$   
4:    $r = b - \text{SpMV}(x)$  ← SpMV  
5:    $d = r$   
6:    $\delta_{\text{new}} = \text{InnerProduct}(r, r)$   
7:    $\delta_0 = \delta_{\text{new}}$   
8:   while  $i < i_{\max}$  do  
9:     UpdateApron( $d$ )  
10:     $q = \text{SpMV}(d)$  ← SpMV  
11:     $\alpha = \delta_{\text{new}} / \text{InnerProduct}(d, q)$   
12:     $x = x + \alpha d$   
13:     $r = b - \alpha q$   
14:     $\delta_{\text{old}} = \delta_{\text{new}}$   
15:     $\delta_{\text{new}} = \text{InnerProduct}(r, r)$   
16:     $\beta = \delta_{\text{new}} / \delta_{\text{old}}$   
17:     $d = r + \beta d$   
18:     $i = i + 1$   
19:    if  $i \bmod 10 == 0$  then  
20:      if  $\delta_{\text{new}} > \epsilon^2 \delta_0$  then      ▷ Read back to CPU for check  
21:        Stop solver and return  
22:      end if  
23:    end if  
24:  end while  
25: end procedure
```

---

- Instead of storing of the matrix, we do this by directly examining voxel values as needed on-the-fly
- Vector is stored in GVDB texture atlas



# Sparse Matrix Vector Multiplication (SpMV)

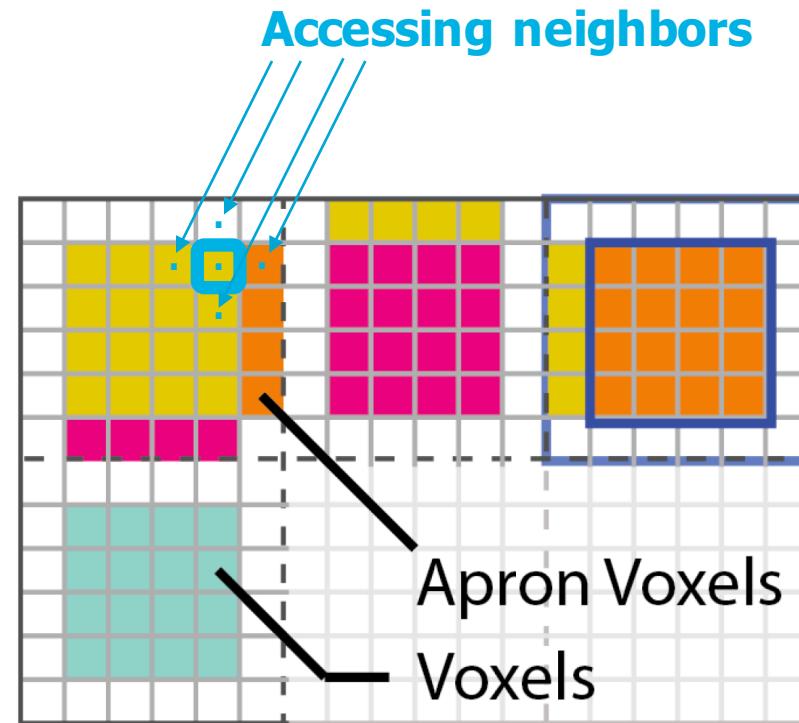
---

**Algorithm 2** Matrix-free Conjugate Gradient Solver

```
1: ▷ Given the inputs divergence  $b$ , starting value  $x$ , maximum of iterations  
    $i_{\max}$ , and error tolerance  $\epsilon$   
2: procedure MatrixFreeConjugateGradientSolver( $b, x, i_{\max}, \epsilon$ )  
3:    $i = 0$   
4:    $r = b - \text{SpMV}(x)$  ← SpMV  
5:    $d = r$   
6:    $\delta_{\text{new}} = \text{InnerProduct}(r, r)$   
7:    $\delta_0 = \delta_{\text{new}}$   
8:   while  $i < i_{\max}$  do  
9:     UpdateApron( $d$ )  
10:     $q = \text{SpMV}(d)$  ← SpMV  
11:     $\alpha = \delta_{\text{new}} / \text{InnerProduct}(d, q)$   
12:     $x = x + \alpha d$   
13:     $r = b - \alpha q$   
14:     $\delta_{\text{old}} = \delta_{\text{new}}$   
15:     $\delta_{\text{new}} = \text{InnerProduct}(r, r)$   
16:     $\beta = \delta_{\text{new}} / \delta_{\text{old}}$   
17:     $d = r + \beta d$   
18:     $i = i + 1$   
19:    if  $i \bmod 10 == 0$  then  
20:      if  $\delta_{\text{new}} > \epsilon^2 \delta_0$  then      ▷ Read back to CPU for check  
21:        Stop solver and return  
22:      end if  
23:    end if  
24:  end while  
25: end procedure
```

---

- Instead of storing of the matrix, we do this by directly examining voxel values as needed on-the-fly
- Vector is stored in GVDB texture atlas



# Apron Update

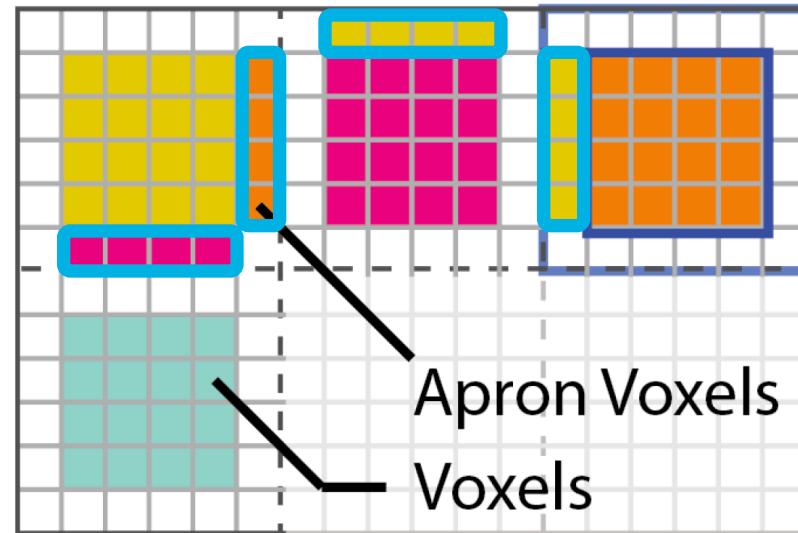
---

**Algorithm 2** Matrix-free Conjugate Gradient Solver

```
1: Given the inputs divergence b, starting value x, maximum of iterations imax, and error tolerance ε
2: procedure MatrixFreeConjugateGradientSolver(b,x,imax,ε)
3:   i = 0
4:   r = b - SpMV(x)
5:   d = r
6:   δnew = InnerProduct(r,r)
7:   δ0 = δnew
8:   while i < imax do
9:     UpdateApron(d) ←———— UpdateApron
10:    q = SpMV(d)
11:    α = δnew/InnerProduct(d,q)
12:    x = x + αd
13:    r = b - αq
14:    δold = δnew
15:    δnew = InnerProduct(r,r)
16:    β = δnew/δold
17:    d = r + βd
18:    i = i + 1
19:    if i mod 10 == 0 then
20:      if δnew > ε2δ0 then      Read back to CPU for check
21:        Stop solver and return
22:      end if
23:    end if
24:  end while
25: end procedure
```

---

- Update apron voxels before using it



# Inner Product

---

**Algorithm 2** Matrix-free Conjugate Gradient Solver

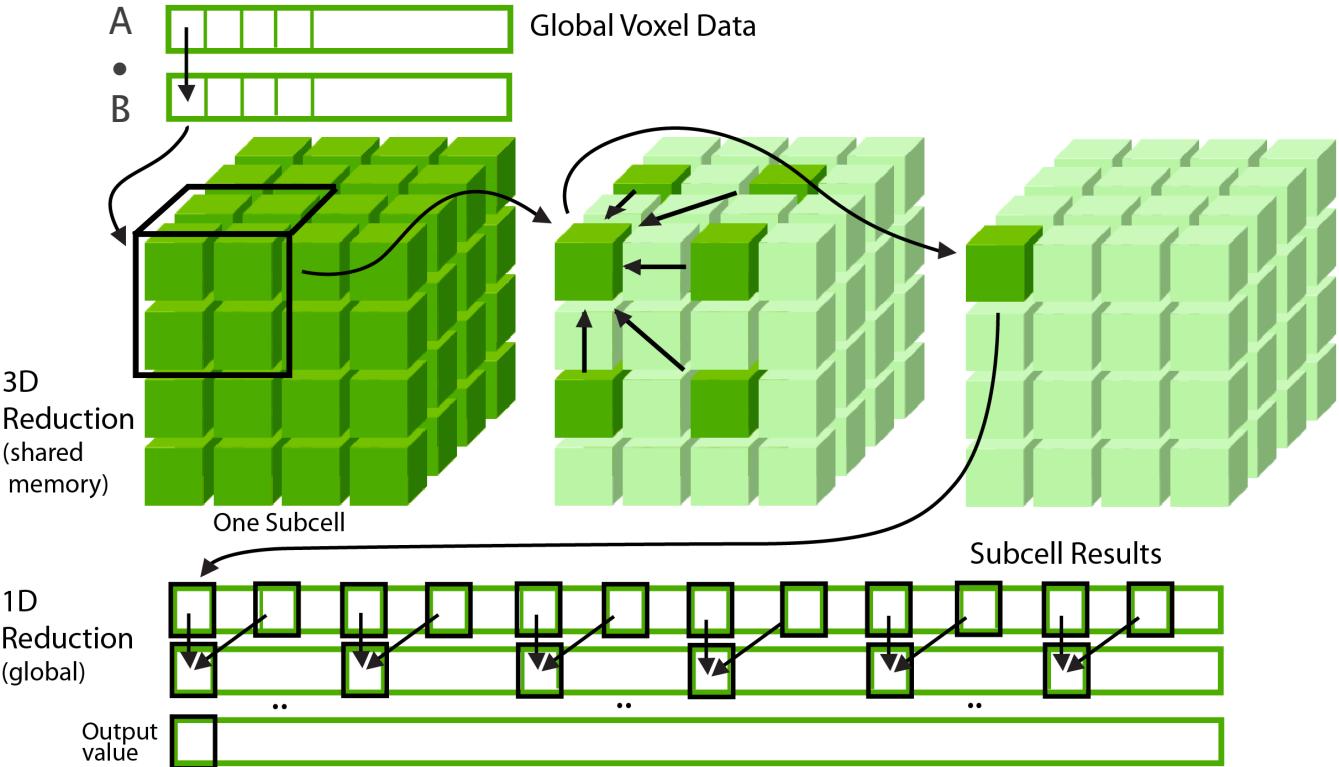
---

```

1: ▷ Given the inputs divergence b, starting value x, maximum of iterations
imax, and error tolerance ε
2: procedure MatrixFreeConjugateGradientSolver(b,x,imax,ε)
3:   i = 0
4:   r = b - SpMV(x)
5:   d = r
6:   δnew = InnerProduct(r,r) ▼
7:   δ0 = δnew
8:   while i < imax do
9:     UpdateApron(d)
10:    q = SpMV(d)
11:    α = δnew/InnerProduct(d,q) ▼
12:    x = x + αd
13:    r = b - αq
14:    δold = δnew
15:    δnew = InnerProduct(r,r) ▼
16:    β = δnew/δold
17:    d = r + βd
18:    i = i + 1
19:    if i mod 10 == 0 then
20:      if δnew > ε2δ0 then ▷ Read back to CPU for check
21:        Stop solver and return
22:      end if
23:    end if
24:  end while
25: end procedure

```

**InnerProduct**



# GPU Matrix-Free Conjugate Gradient Solver

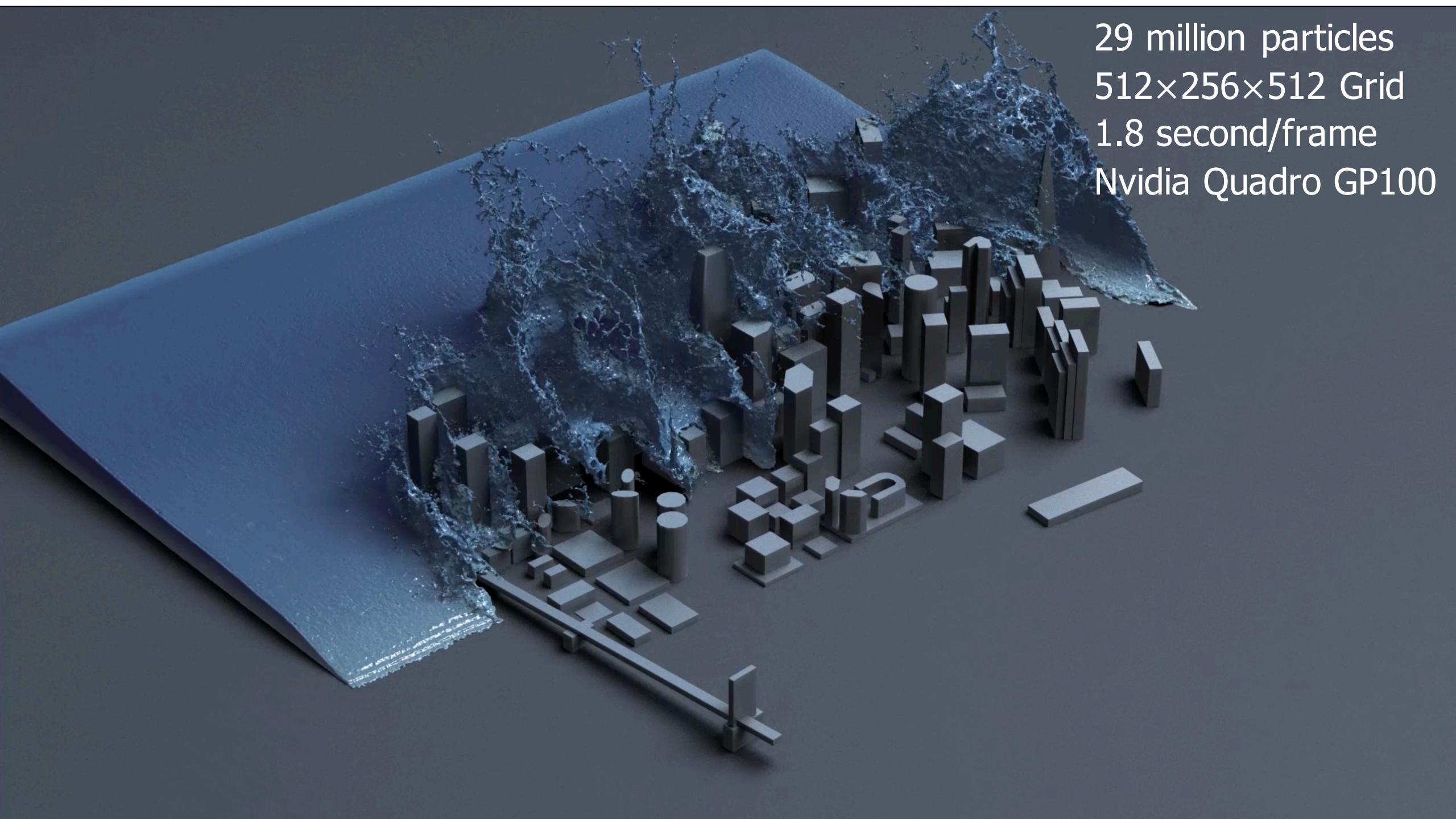
---

**Algorithm 2** Matrix-free Conjugate Gradient Solver

---

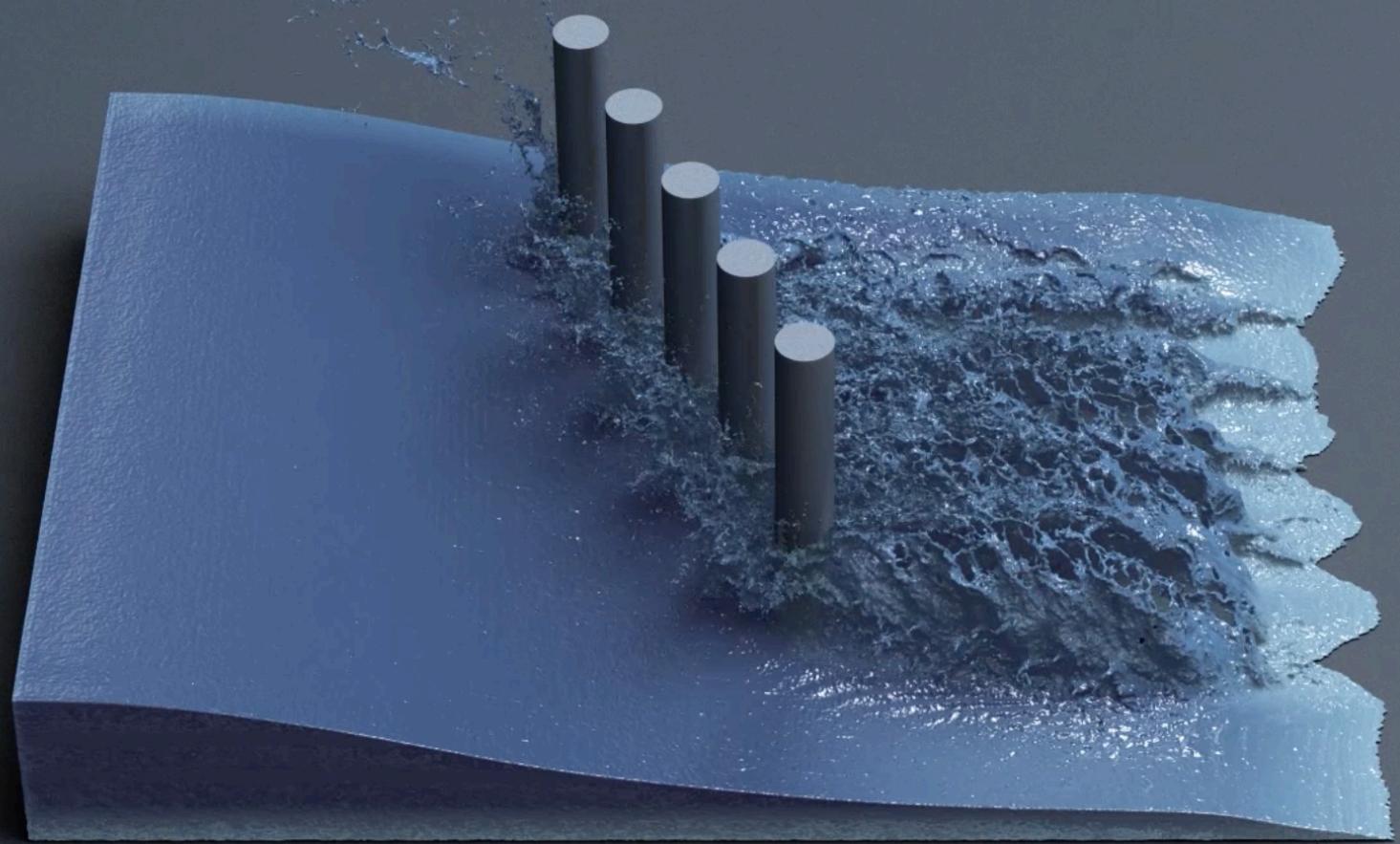
```
1: Given the inputs divergence b, starting value x, maximum of iterations
   imax, and error tolerance ε
2: procedure MatrixFreeConjugateGradientSolver(b,x,imax,ε)
3:   i = 0
4:   r = b - SpMV(x)
5:   d = r
6:   δnew = InnerProduct(r,r)
7:   δ0 = δnew
8:   while i < imax do
9:     UpdateApron(d)
10:    q = SpMV(d)
11:    α = δnew/InnerProduct(d,q)
12:    x = x + αd
13:    r = b - αq
14:    δold = δnew
15:    δnew = InnerProduct(r,r)
16:    β = δnew/δold
17:    d = r + βd
18:    i = i + 1
19:    if i mod 10 == 0 then
20:      if δnew > ε2δ0 then      ▷ Read back to CPU for check
21:        Stop solver and return
22:      end if
23:    end if
24:  end while
25: end procedure
```

---

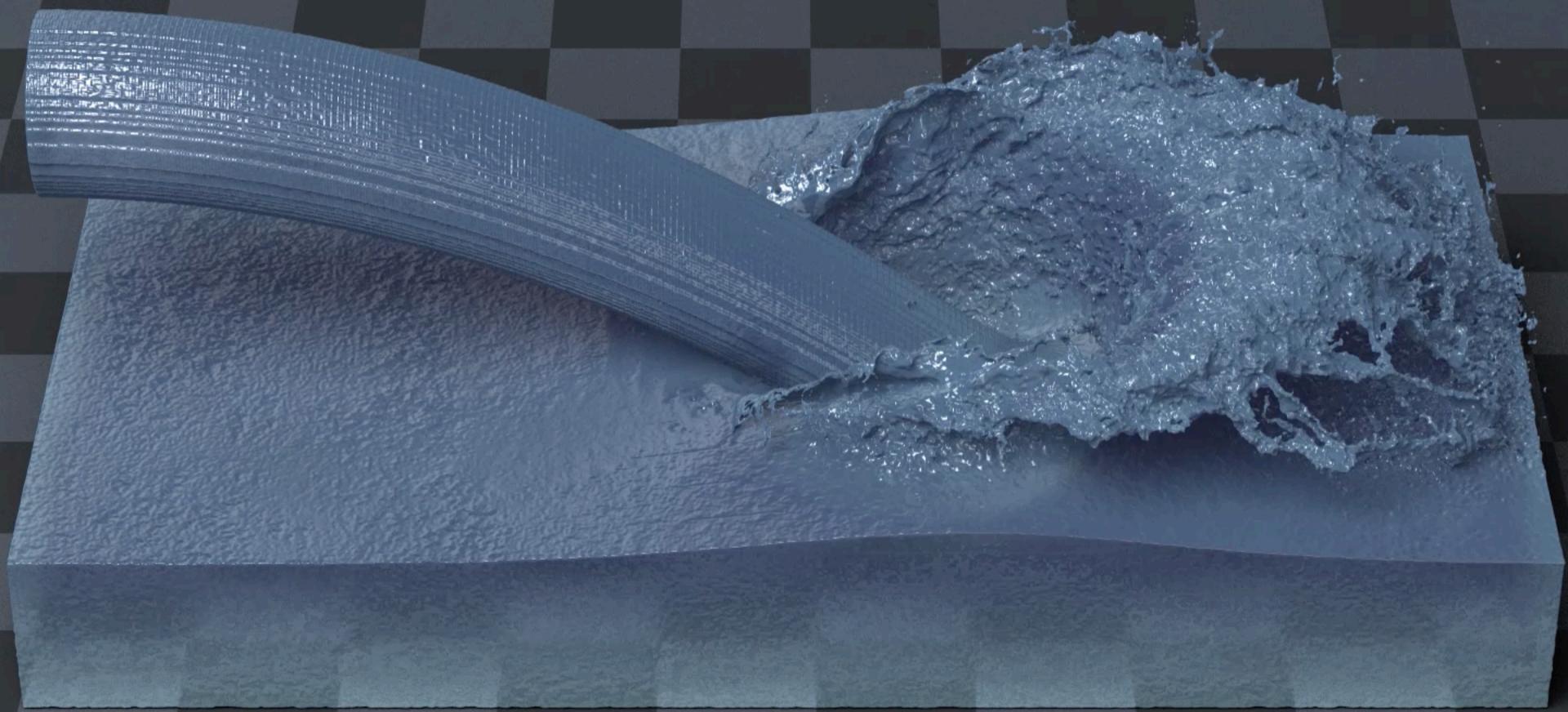


29 million particles  
512×256×512 Grid  
1.8 second/frame  
Nvidia Quadro GP100

29 million particles  
450×300×300 Grid  
2.4 second/frame  
Nvidia Quadro GP100



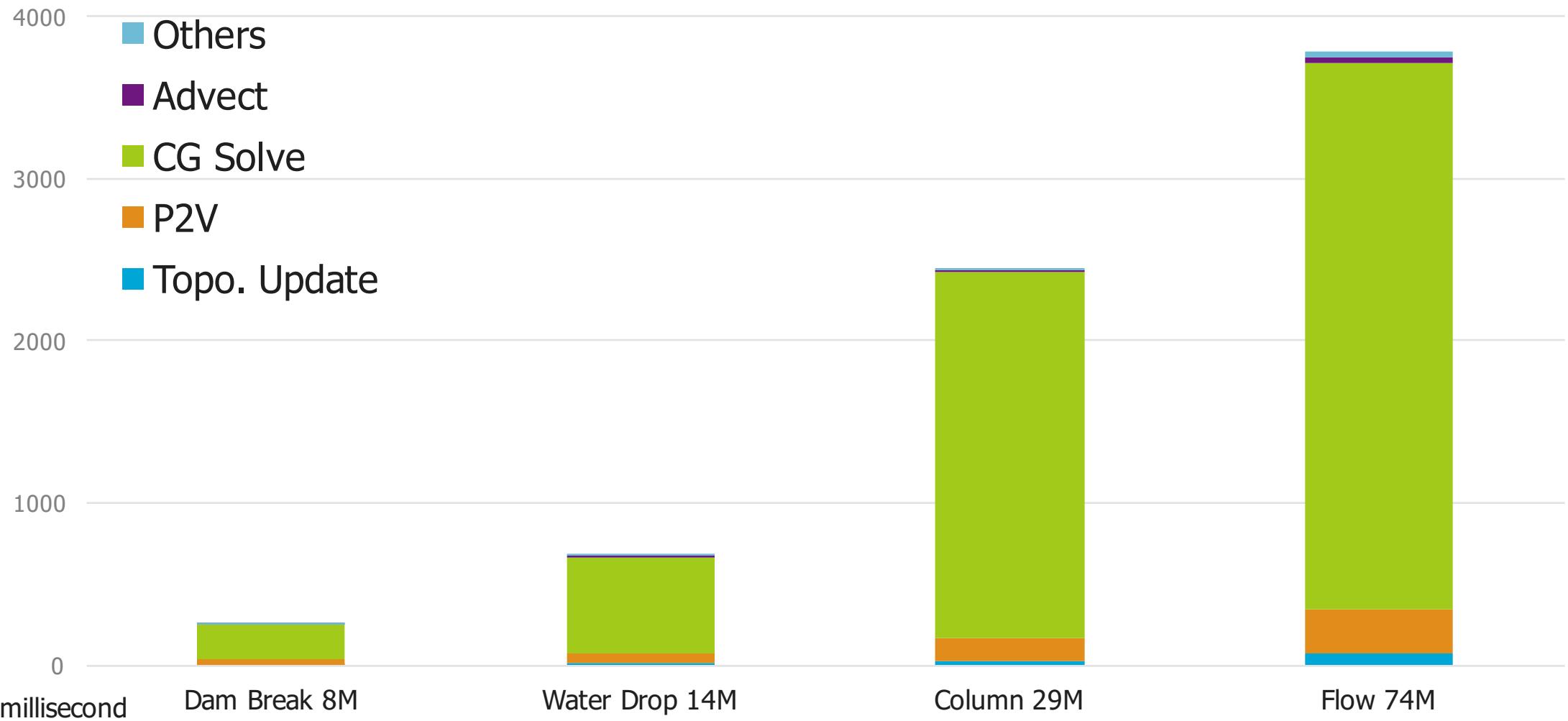
74 million particles  
1056×288×768 Grid  
3.8 second/frame  
Nvidia Quadro GP100



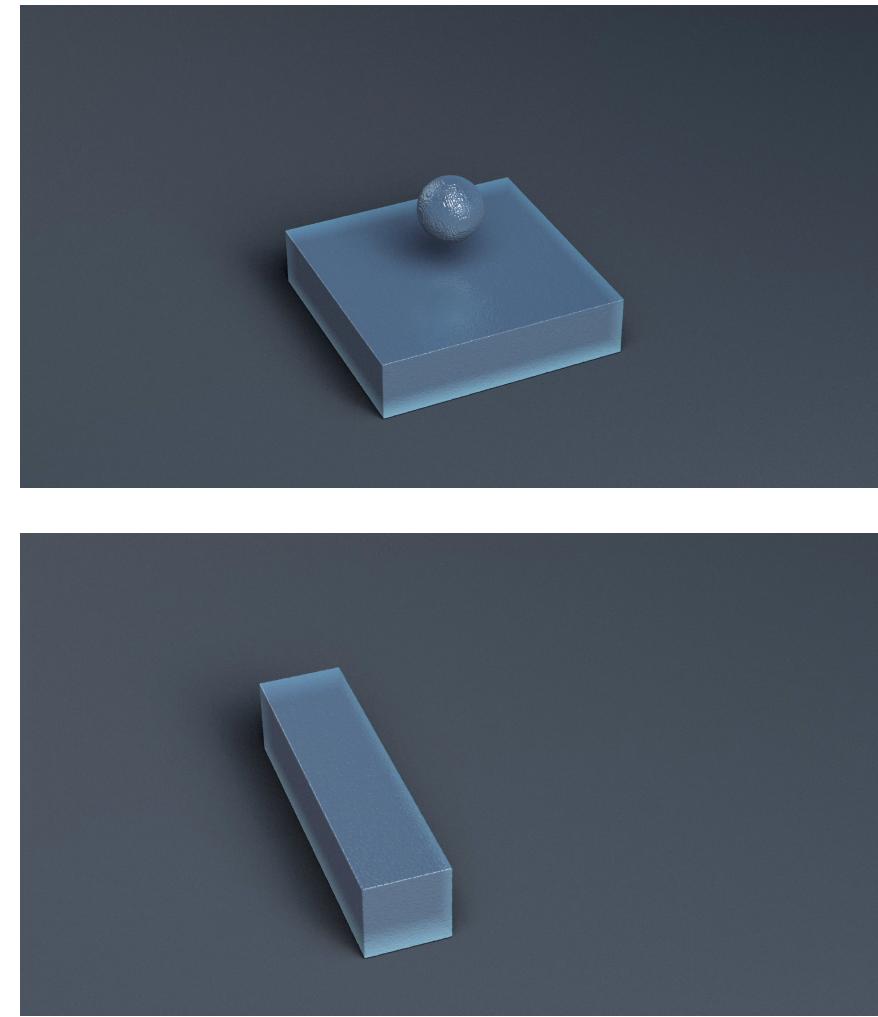
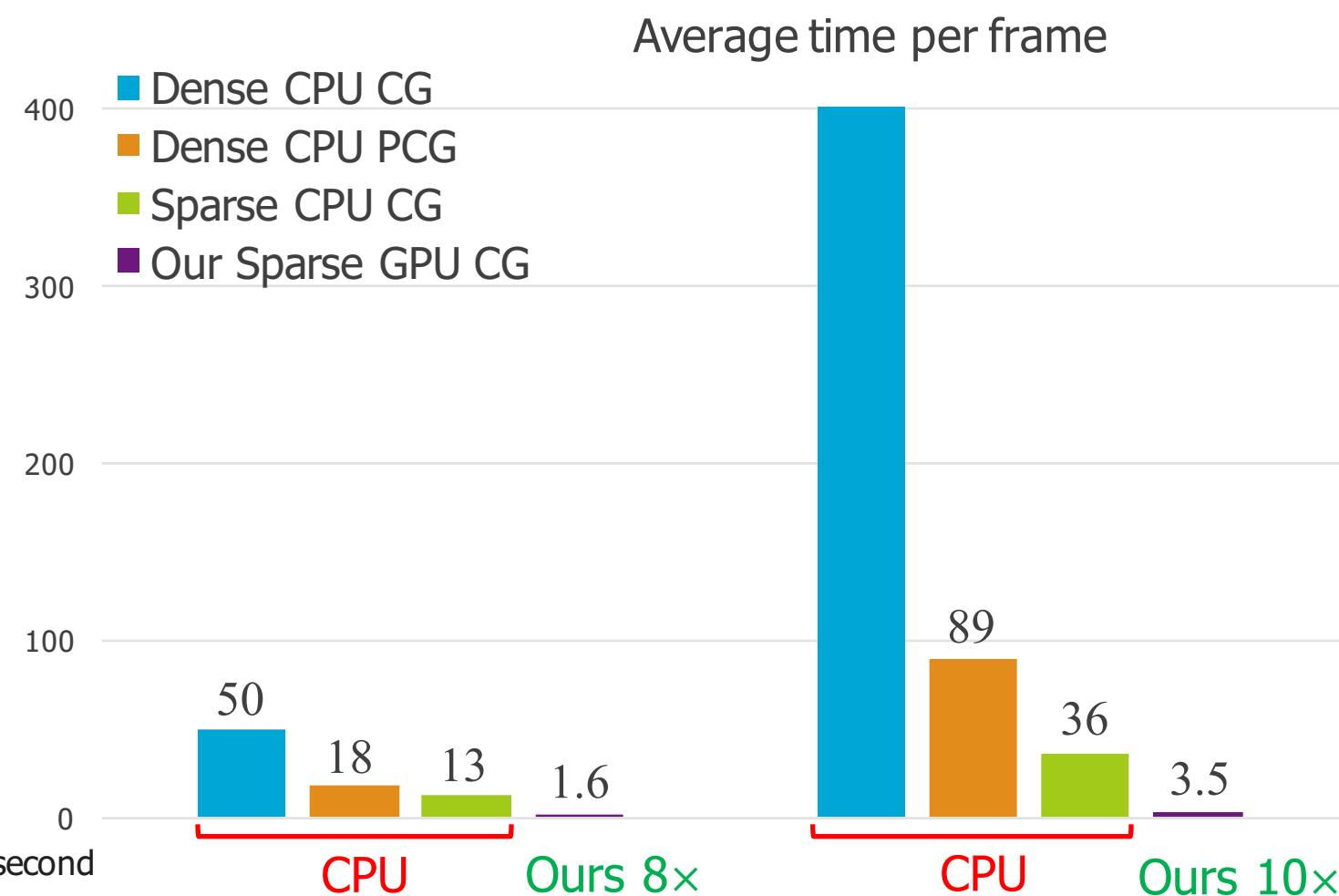
2 million particles  
3360×160×2272 Grid  
1 second/frame  
Nvidia Quadro GP100



# Average Time per Frame



# GPU Matrix-free CG Solver



# Conclusions

We have presented a sparse, efficient, GPU-based FLIP simulation for fluids on virtually unbounded domains

- Dynamic Topology
- Particle-to-Grid Rasterization on Subcell
- GPU Matrix-free CG solver
- An order of magnitude faster than on the CPU

# Future work

- GPU-friendly Preconditioner
- Narrow band FLIP on the GPU

# Thank you!



<https://developer.nvidia.com/gvdb>



NVIDIA® GVDB Voxels 1.1

- Dynamic Topology
- Points-to-Voxels



nVIDIA®

