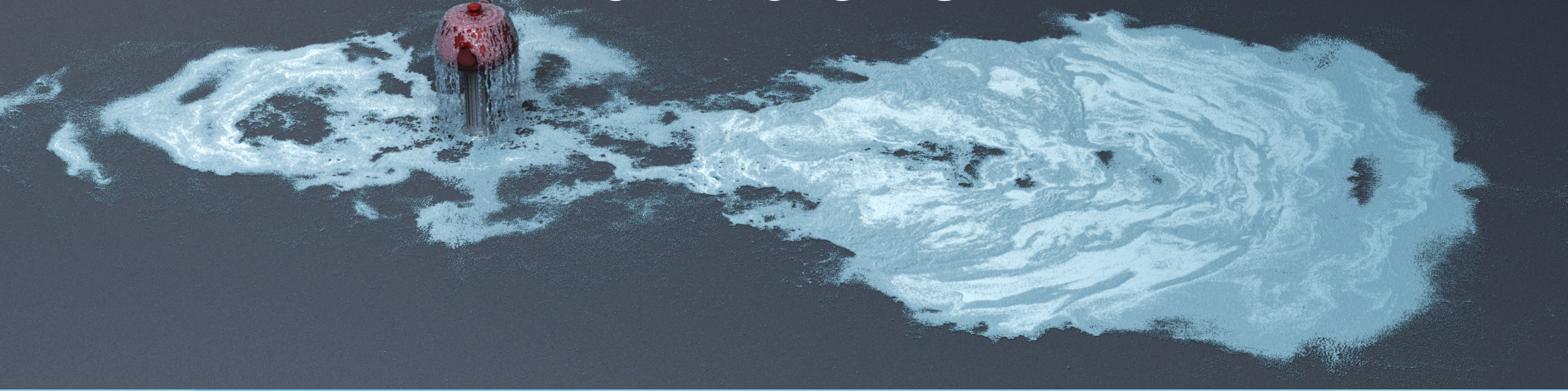
# Fast Fluid Simulation with Sparse Volumes on the GPU



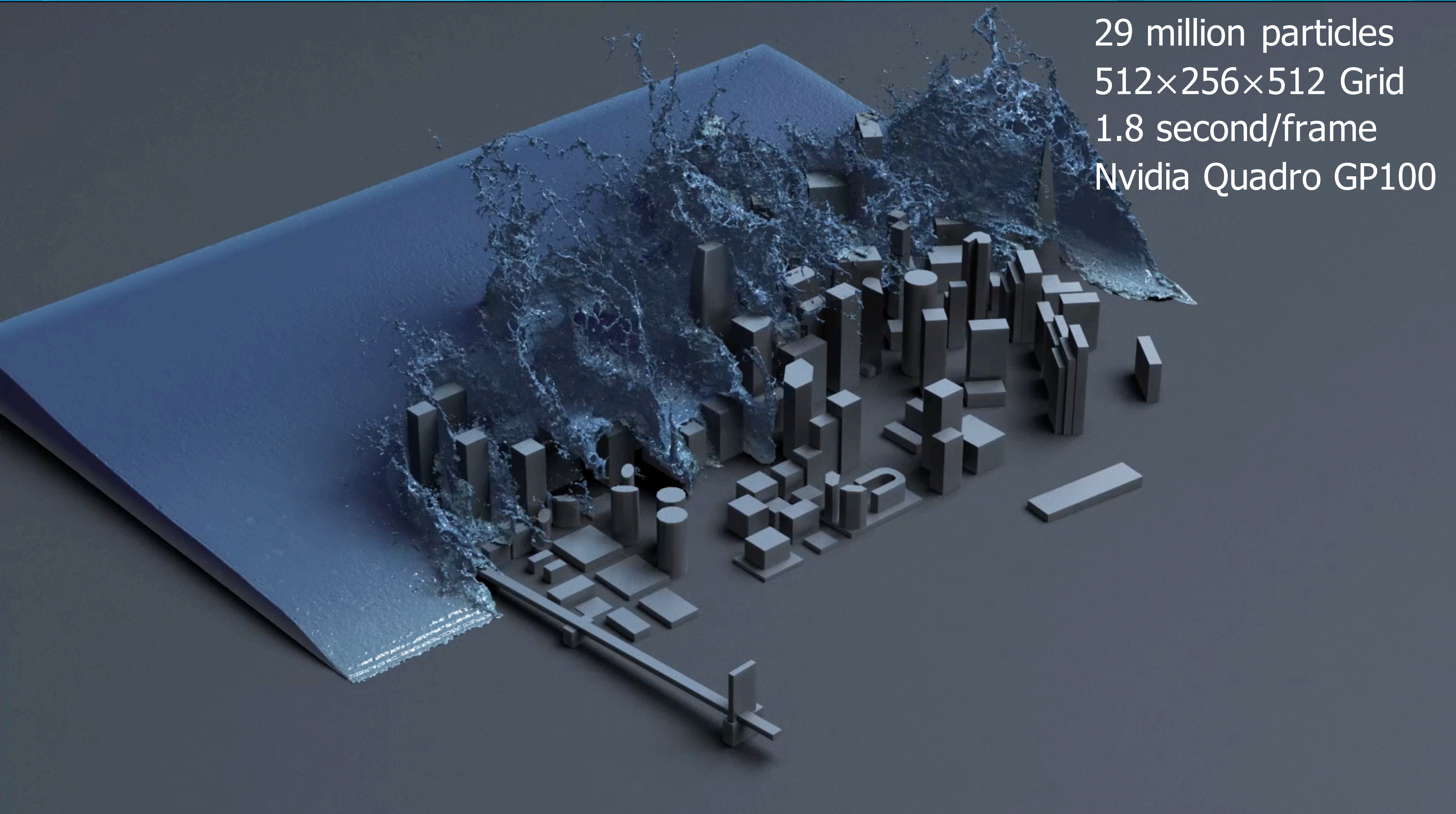
Kui Wu[1], Nghia Truong[1], Cem Yuksel[1], Rama Hoetzlein[2]

[1] University of Utah, USA

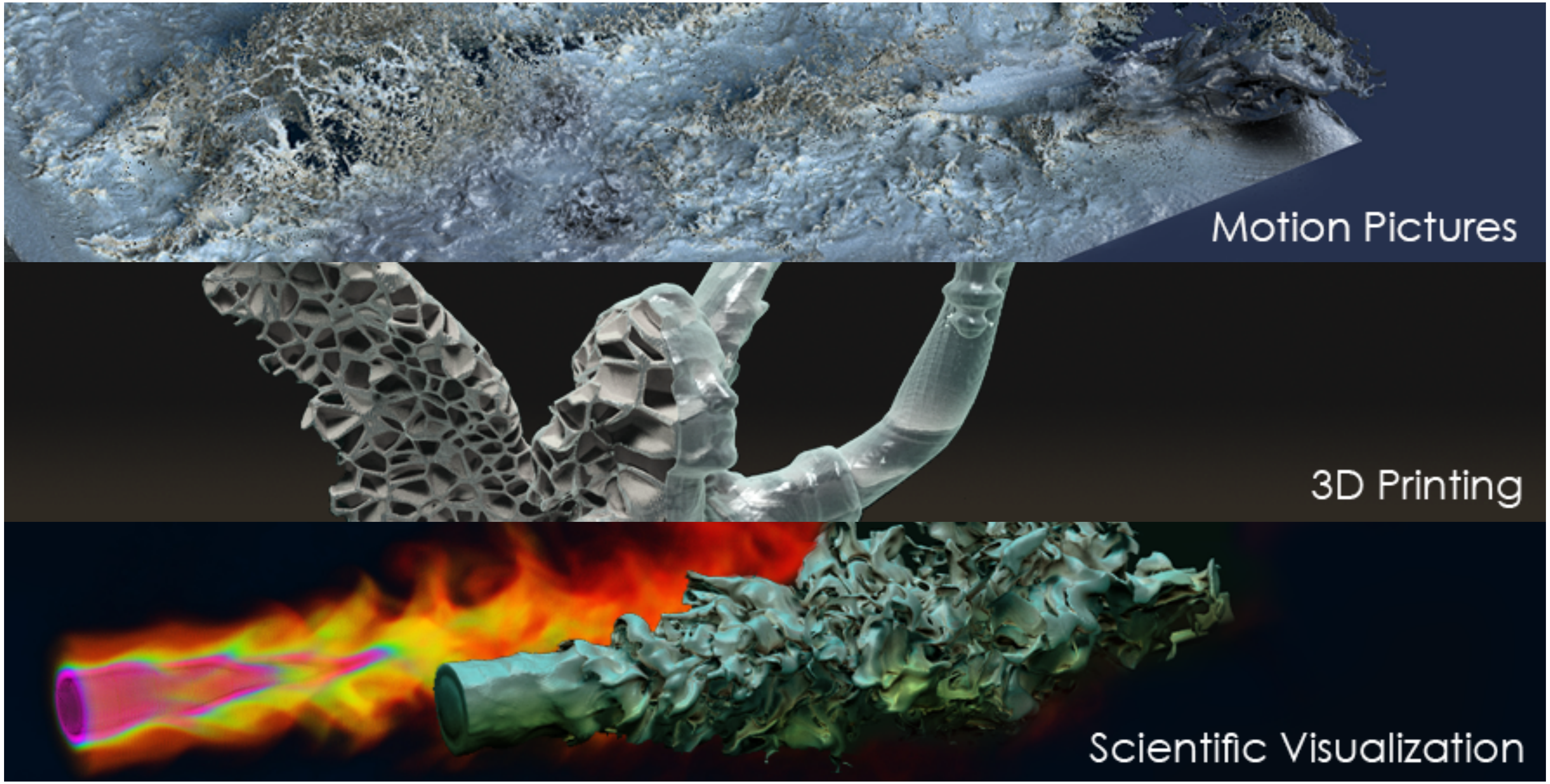
[2] NVIDIA Corporation

EUROGRAPHICS 2018

NVIDIA

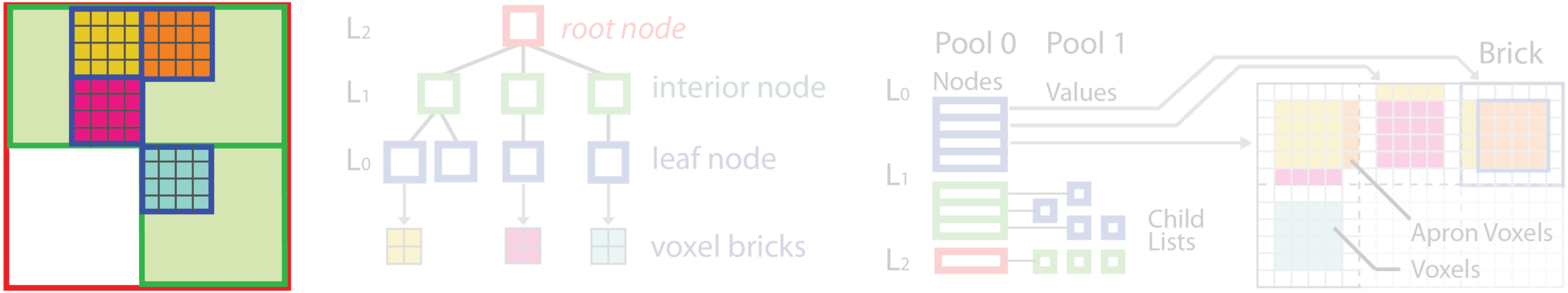

29 million particles
512×256×512 Grid
1.8 second/frame
Nvidia Quadro GP100

# NVIDIA® GVDB Voxels



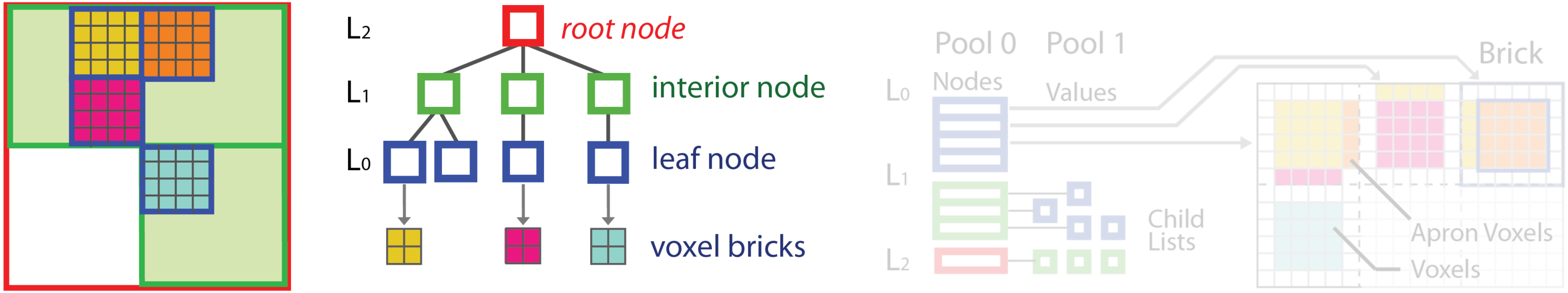

A new, open source NVIDIA SDK for compute, simulation and rendering of sparse volumes based on CUDA

# NVIDIA® GVDB Voxels




a) Spatial domain b) Tree schematic c) GVDB Data Structure d) GVDB Voxel Atlas

# NVIDIA® GVDB Voxels

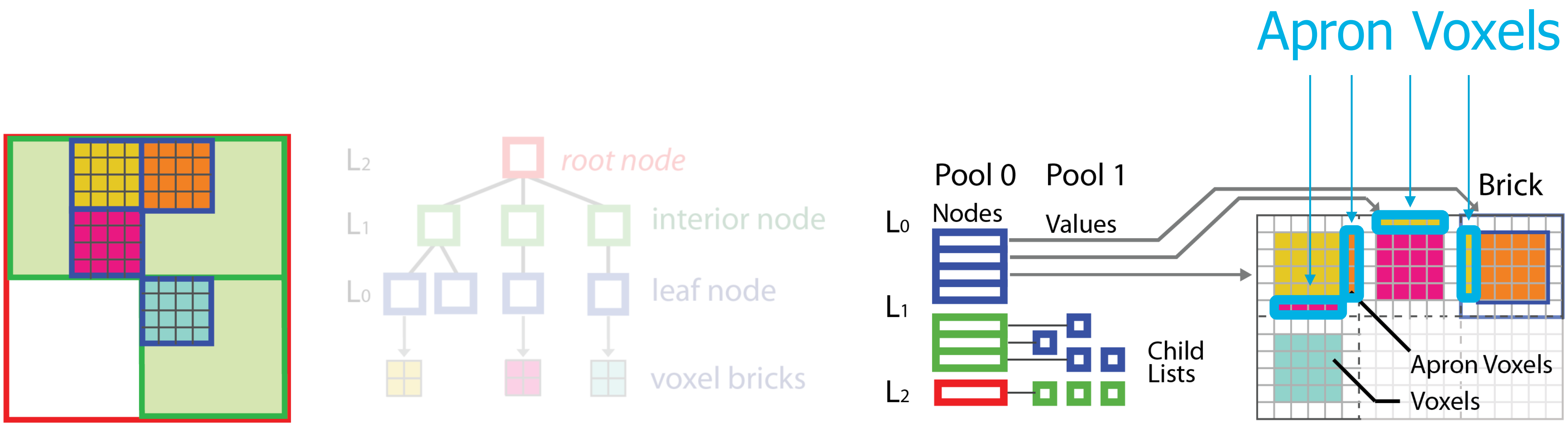



*a) Spatial domain* *b) Tree schematic* *c) GVDB Data Structure* *d) GVDB Voxel Atlas*

# NVIDIA® GVDB Voxels

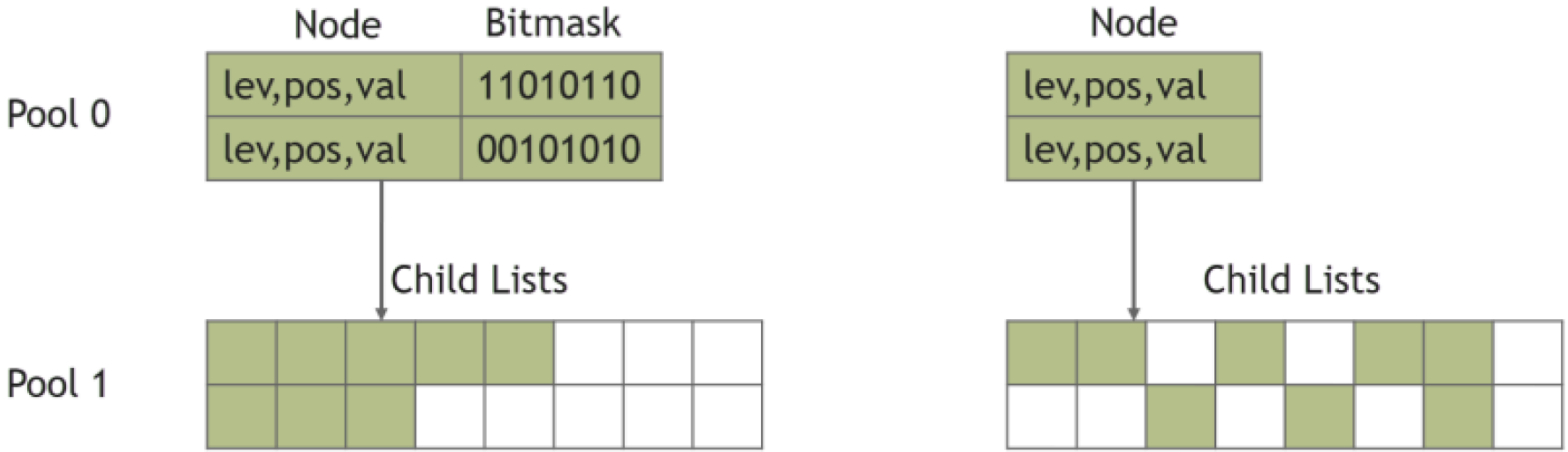

a) Spatial domain b) Tree schematic c) GVDB Data Structure d) GVDB Voxel Atlas

# NVIDIA® GVDB Voxels

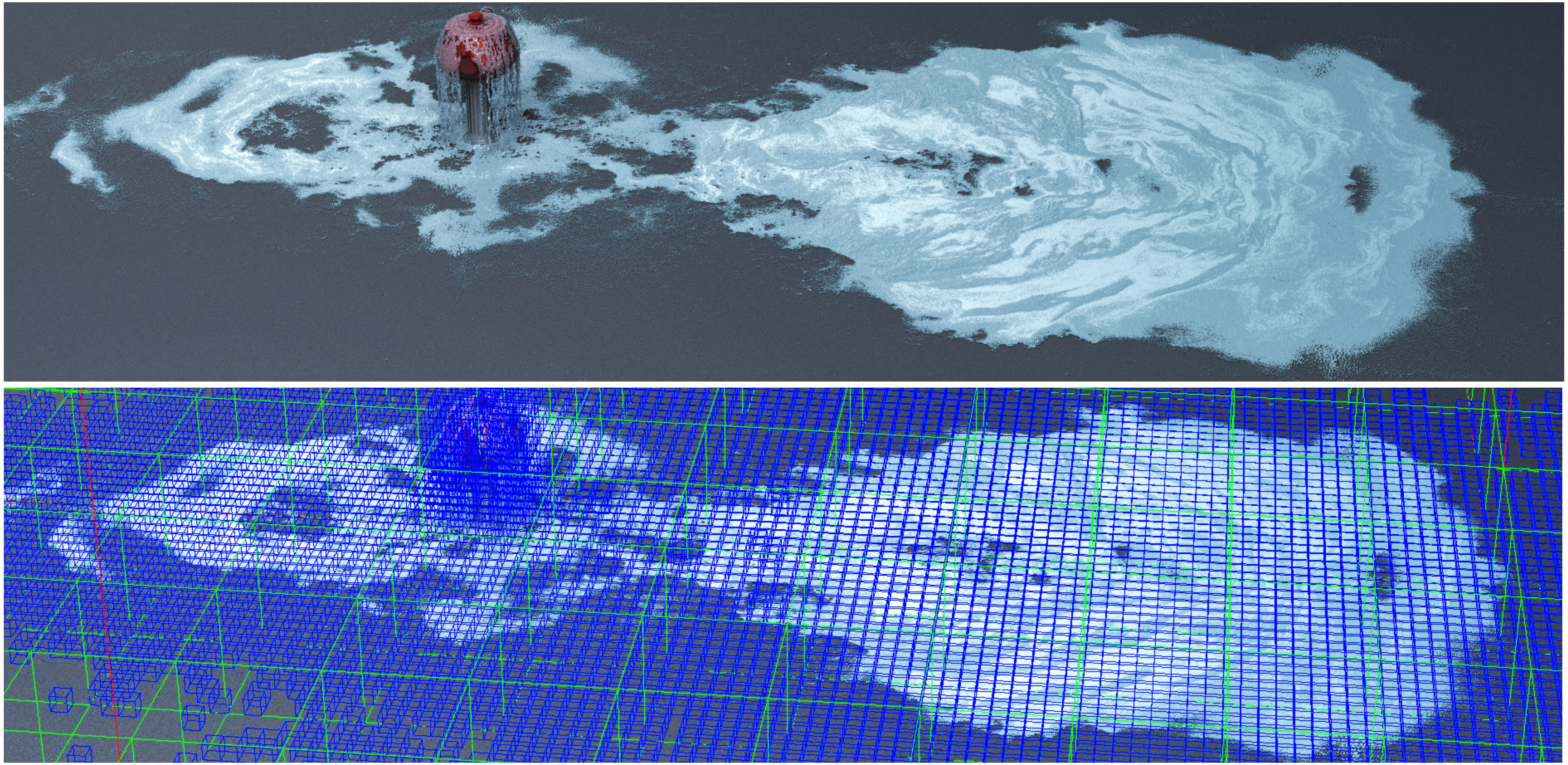



a) Spatial domain b) Tree schematic c) GVDB Data Structure d) GVDB Voxel Atlas

# NVIDIA® GVDB Voxels

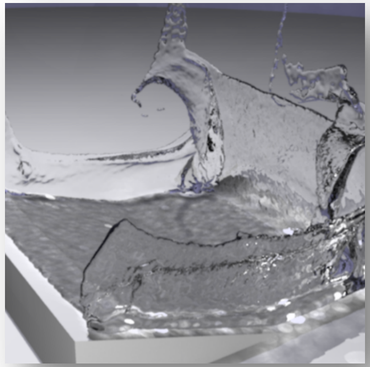


# FLIP Simulation with NVIDIA® GVDB Sparse Voxels

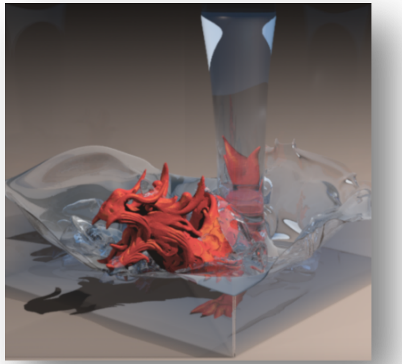


# Prior Work

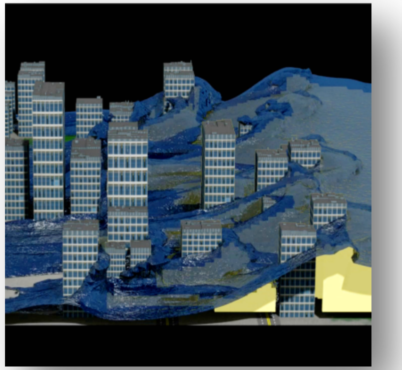
## CPU-based methods for large scale simulation



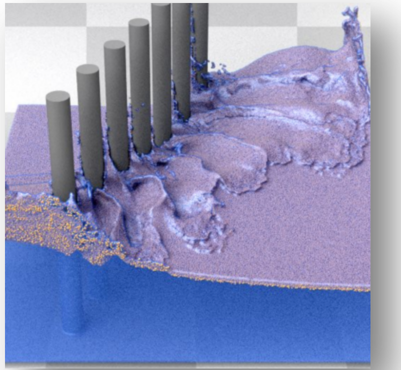
[Zhu and Bridson 2005]
Fluid-implicit particle method (FLIP)



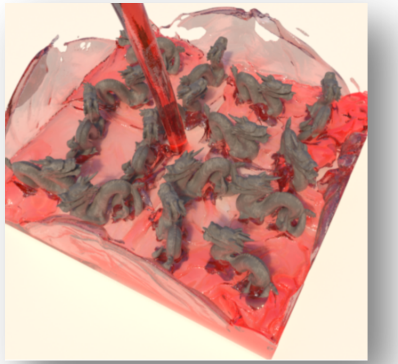
[McAdams et al. 2010]
Multigrid PCG



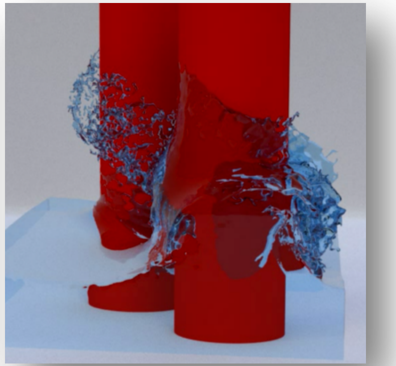
[Ferstl et al. 2014]
Domain Decomposition



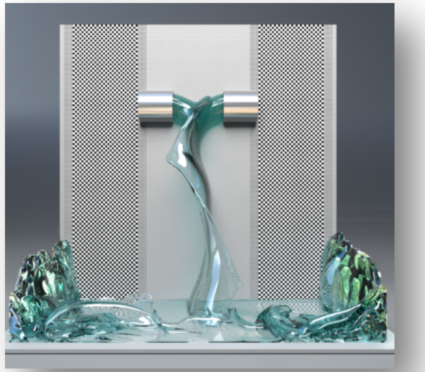
[Ferstl et al. 2016]
Narrow Band FLIP



Schur Complement Solver [Liu et al. 2016]



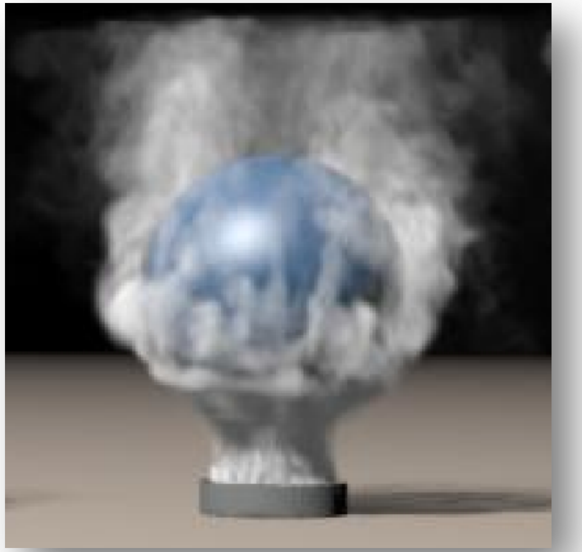
[Chu et al. 2017]



[Aanjaneya and Gao et al. 2017]
SPGrid

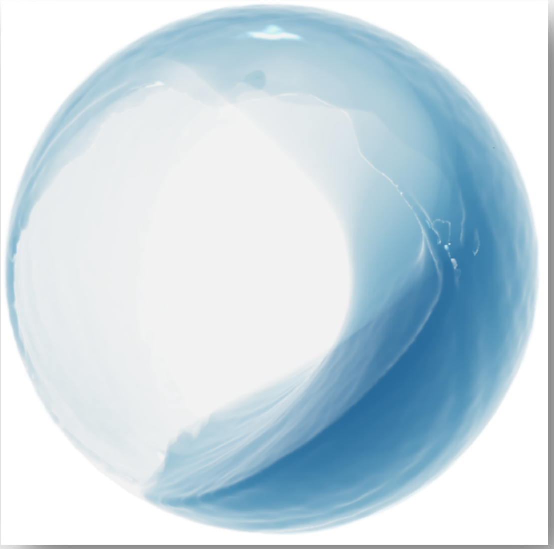
# Prior Work

## GPU-based methods

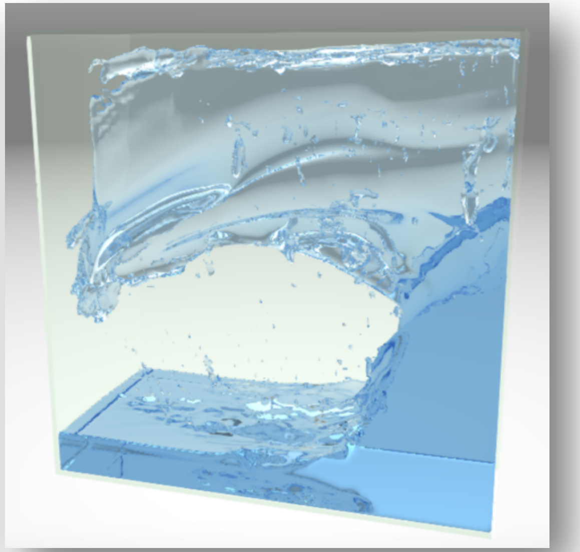


[Molemaker et al. 2008]

[Horvath and Geiger 2009]



[Chentanez and Müller 2011]



[Chentanez and Müller 2012]

# FLIP with Sparse Volumes on the GPU

**procedure** SparseFLIP( )
  $P \leftarrow$ initial points
  $V \leftarrow$ GVDB structure
  **for** *each frame* **do**
    **if** first frame **then**
      $V_{topo} \leftarrow$ full rebuild $(P)$
    **else**
      $V_{topo} \leftarrow$ incremental build $(P)$
    **end if**
    $V \leftarrow$ resize and clear $(V_{topo})$
    $S \leftarrow$ insert points in subcells $(V, P)$
    $V(vel) \leftarrow$ particles-to-voxels $(S, P)$
    $V \leftarrow$ update apron $(\rho, vel, marker)$
    $V(vel_{old}) \leftarrow$ V(vel)
    $V(div) \leftarrow$ divergence $(V(vel))$
    $V(\rho) \leftarrow$ CG pressure solve $(V, div)$
    $V(vel) \leftarrow$ pressure-to-velocity $(V(\rho))$
    $V \leftarrow$ update apron $(V(vel))$
    $P \leftarrow$ advance $(V(vel), V(vel_{old}))$
  **end for**
**end procedure**

Dynamic Topology

Particles-to-Voxels

Pressure Solver (CG)

# FLIP with Sparse Volumes on the GPU

```
procedure SparseFLIP( )
    P ← initial points
    V ← GVDB structure
    for eachframe do
        if first frame then
            V_topo ← full rebuild (P)
        else
            V_topo ← incremental build (P)
        end if
        V ← resize and clear (V_topo)
        S ← insert points in subcells (V, P)
        V(vel) ← particles-to-voxels (S, P)
        V ← update apron (ρ, vel, marker)
        V(vel_old) ← V(vel)
        V(div) ← divergence (V(vel))
        V(ρ) ← CG pressure solve (V, div)
        V(vel) ← pressure-to-velocity (V(ρ))
        V ← update apron (V(vel))
        P ← advance (V(vel), V(vel_old))
    end for
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

**procedure** **SparseFLIP**( )
  $P \leftarrow$ initial points
  $V \leftarrow$ GVDB structure
  **for** $each frame$ **do**
    **if** first frame **then**
      $V_{topo} \leftarrow$ full rebuild $(P)$
    **else**
      $V_{topo} \leftarrow$ incremental build $(P)$
    **end if**
    $V \leftarrow$ resize and clear $(V_{topo})$
    $S \leftarrow$ insert points in subcells $(V, P)$
    $V(vel) \leftarrow$ particles-to-voxels $(S, P)$
    $V \leftarrow$ update apron $(\rho, vel, marker)$
    $V(vel_{old}) \leftarrow$ V(vel)
    $V(div) \leftarrow$ divergence $(V(vel))$
    $V(\rho) \leftarrow$ CG pressure solve $(V, div)$
    $V(vel) \leftarrow$ pressure-to-velocity $(V(\rho))$
    $V \leftarrow$ update apron $(V(vel))$
    $P \leftarrow$ advance $(V(vel), V(vel_{old}))$
  **end for**
**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

**procedure** **SparseFLIP**( )
  $P \leftarrow$ initial points
  $V \leftarrow$ GVDB structure
  **for** *eachframe* **do**
    **if** first frame **then**
      $V_{topo} \leftarrow$ full rebuild $(P)$
    **else**
      $V_{topo} \leftarrow$ incremental build $(P)$
    **end if**
    $V \leftarrow$ resize and clear $(V_{topo})$
    $S \leftarrow$ insert points in subcells $(V, P)$
    $V(vel) \leftarrow$ particles-to-voxels $(S, P)$
    $V \leftarrow$ update apron $(\rho, vel, marker)$
    $V(vel_{old}) \leftarrow$ V(vel)
    $V(div) \leftarrow$ divergence $(V(vel))$
    $V(\rho) \leftarrow$ CG pressure solve $(V, div)$
    $V(vel) \leftarrow$ pressure-to-velocity $(V(\rho))$
    $V \leftarrow$ update apron $(V(vel))$
    $P \leftarrow$ advance $(V(vel), V(vel_{old}))$
  **end for**
**end procedure**

Dynamic Topology

Particles-to-Voxels

**Pressure Solver (CG)**

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

# FLIP with Sparse Volumes on the GPU

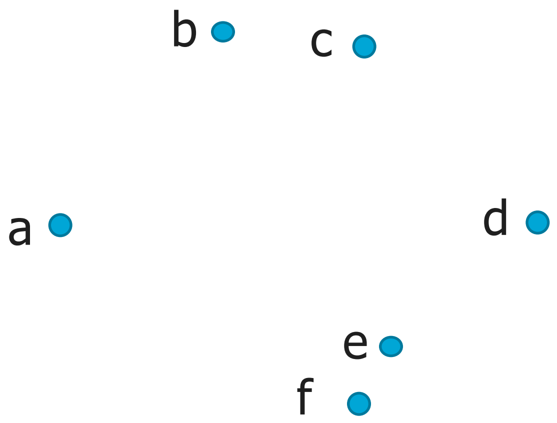
**procedure** SparseFLIP( )
    $P \leftarrow$ initial points
    $V \leftarrow$ GVDB structure
    **for** *each frame* **do**
        **if** first frame **then**
            $V_{topo} \leftarrow$ full rebuild $(P)$
        **else**
            $V_{topo} \leftarrow$ incremental build $(P)$
        **end if**
        $V \leftarrow$ resize and clear $(V_{topo})$
        $S \leftarrow$ insert points in subcells $(V, P)$
        $V(vel) \leftarrow$ particles-to-voxels $(S, P)$
        $V \leftarrow$ update apron $(\rho, vel, marker)$
        $V(vel_{old}) \leftarrow$ V(vel)
        $V(div) \leftarrow$ divergence $(V(vel))$
        $V(\rho) \leftarrow$ CG pressure solve $(V, div)$
        $V(vel) \leftarrow$ pressure-to-velocity $(V(\rho))$
        $V \leftarrow$ update apron $(V(vel))$
        $P \leftarrow$ advance $(V(vel), V(vel_{old}))$
    **end for**
**end procedure**

Dynamic Topology

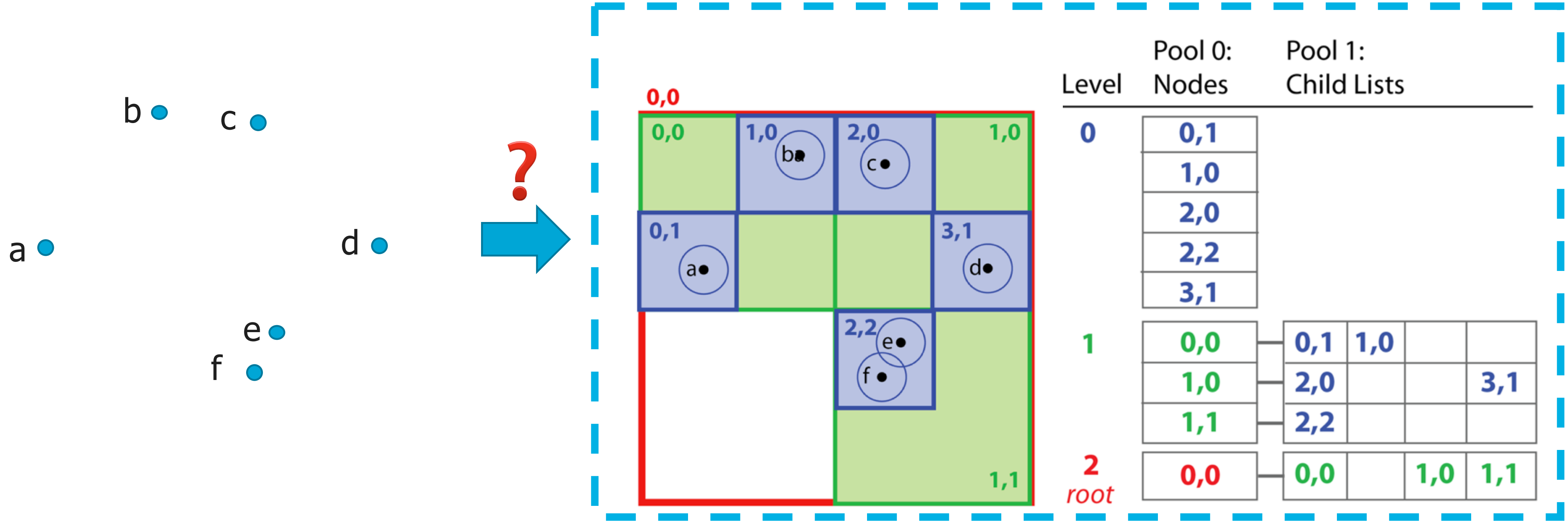
Particles-to-Voxels

Pressure Solver (CG)

# Full Topology Rebuild

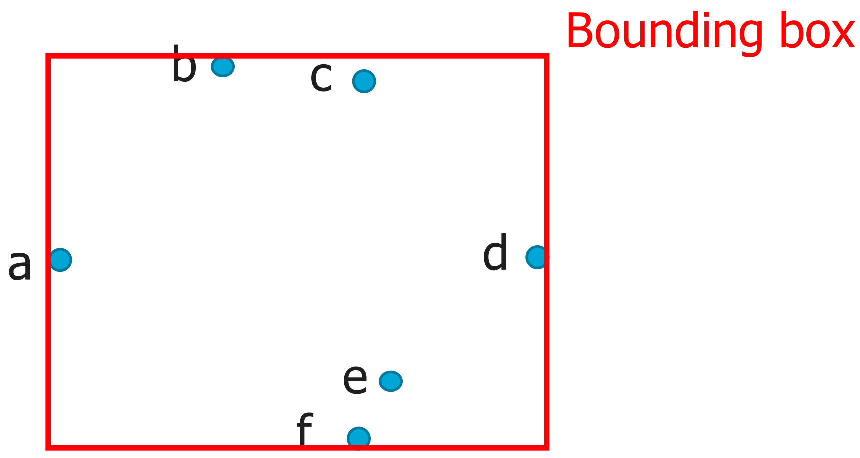


# Full Topology Rebuild



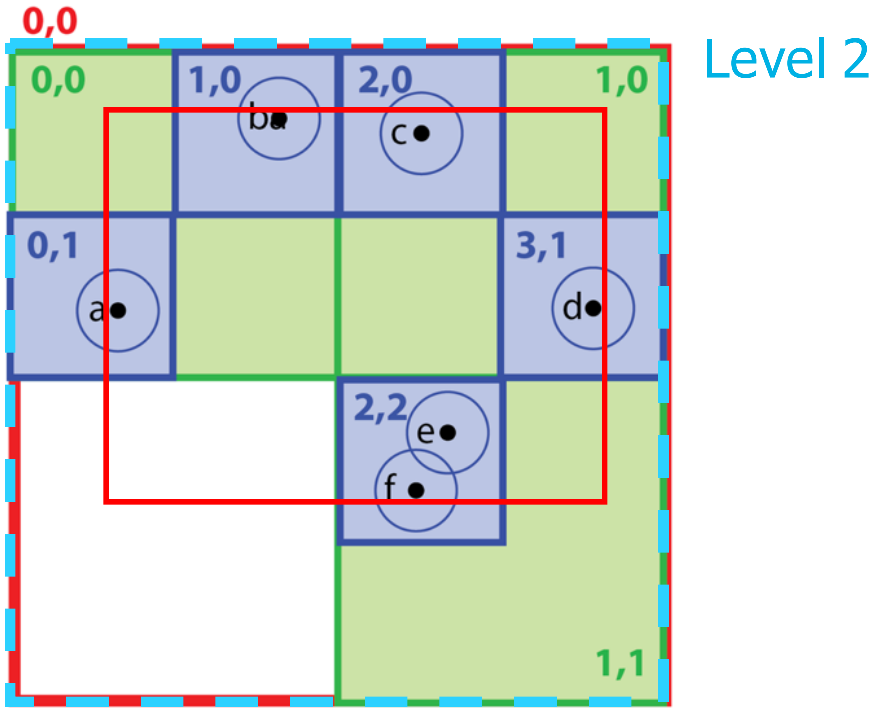
# Full Topology Rebuild

- 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



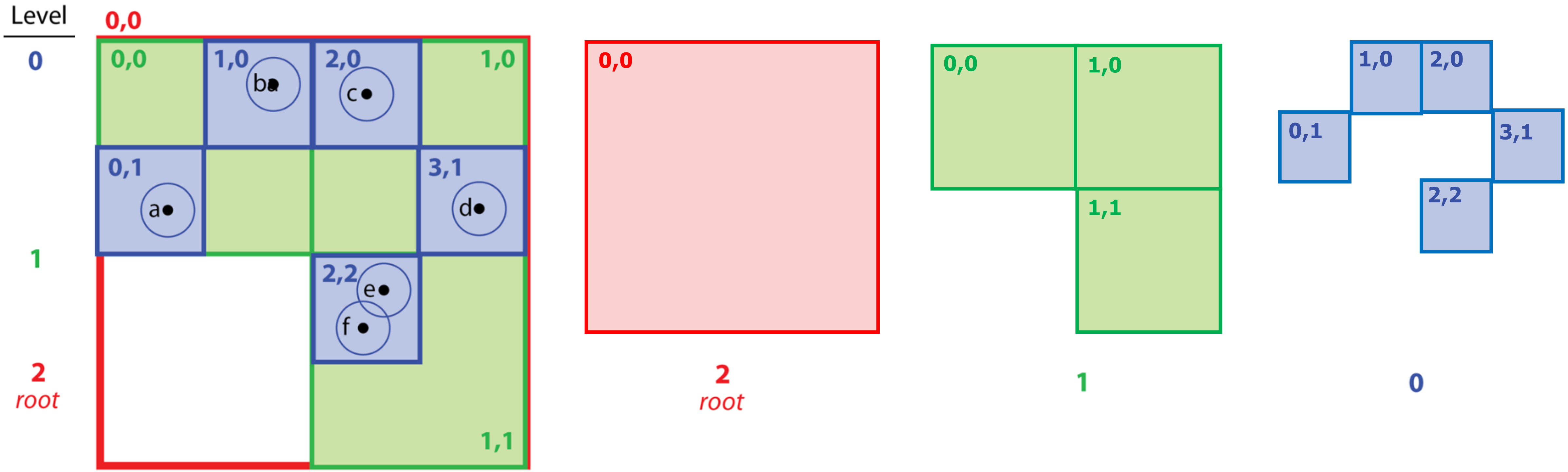
# Full Topology Rebuild

- 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



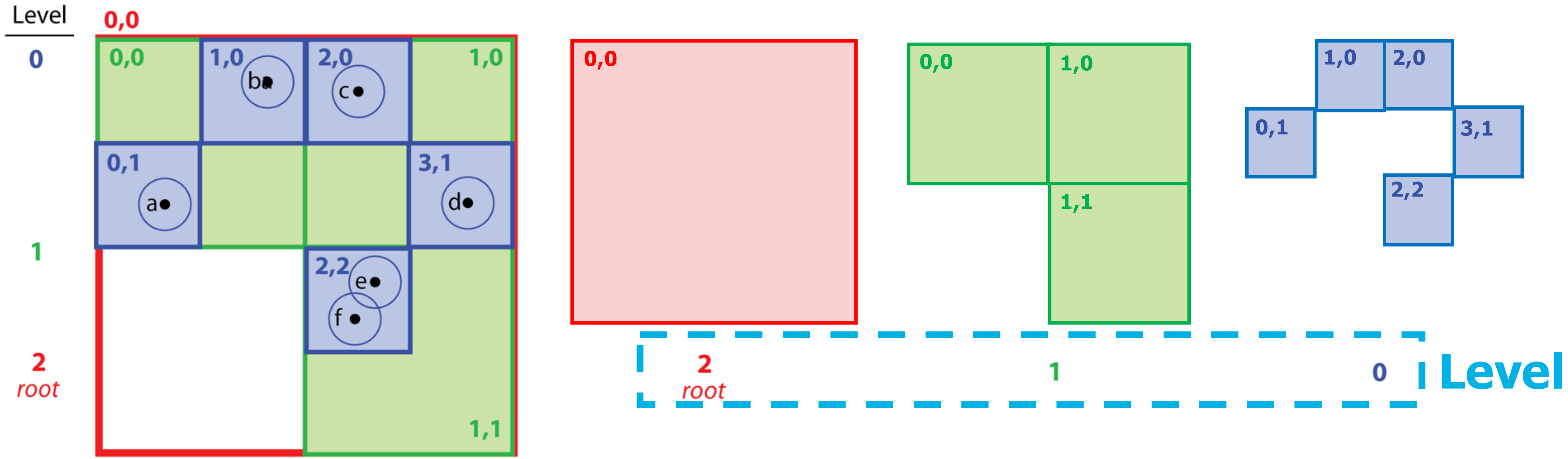
# Full Topology Rebuild

- 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



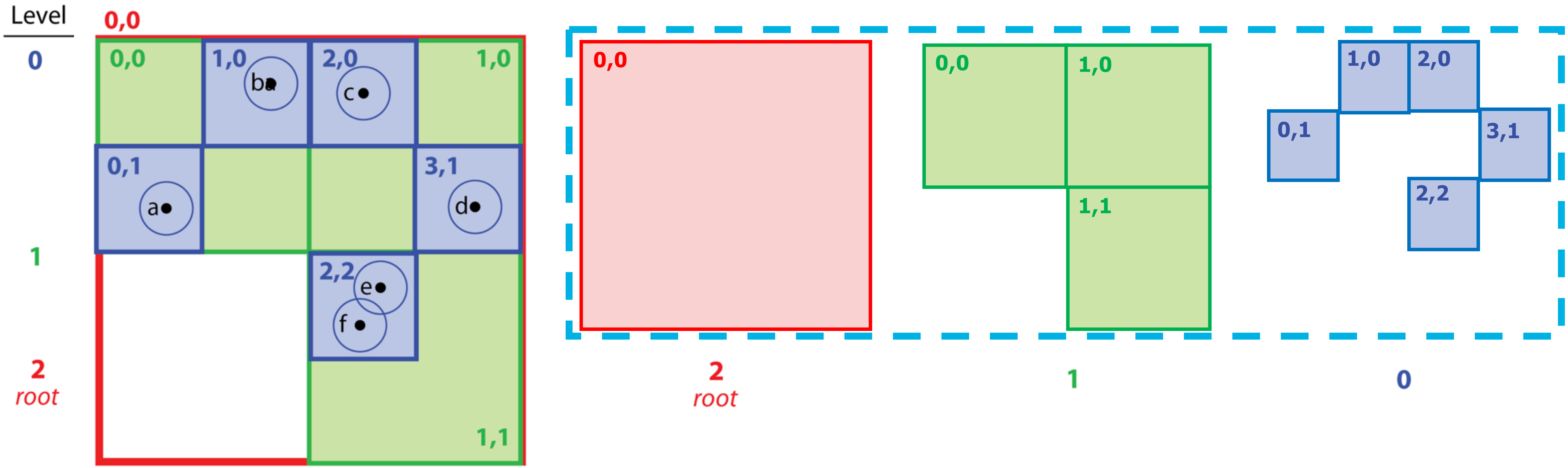
# Full Topology Rebuild

- 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



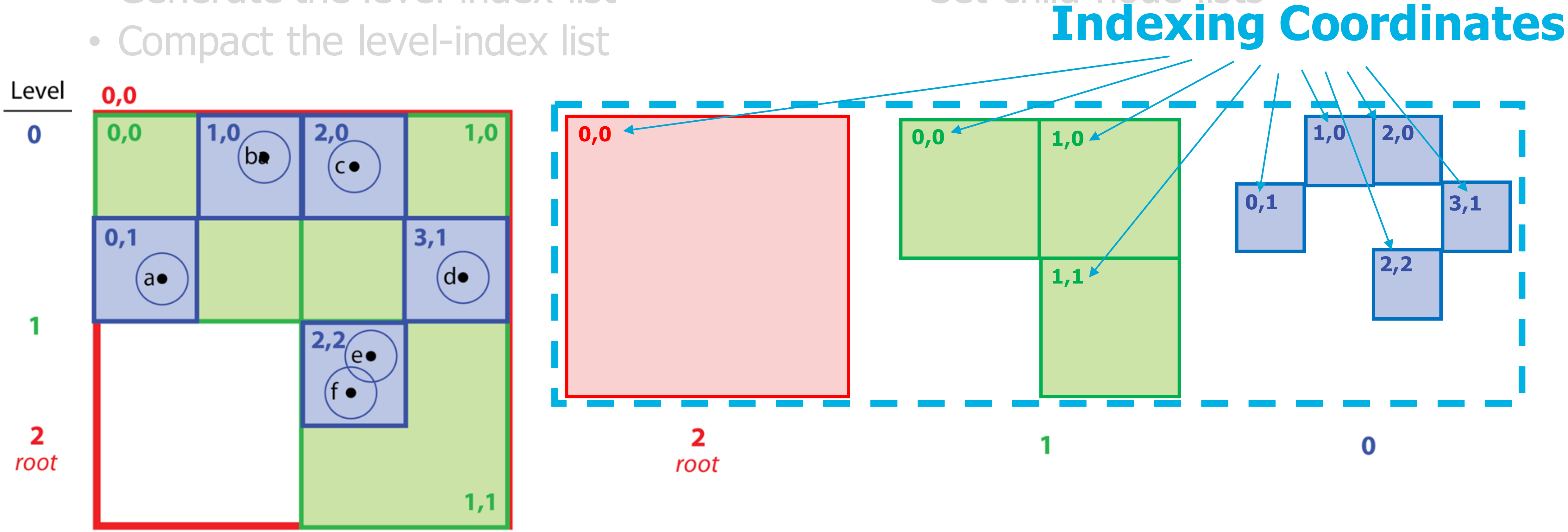
# Full Topology Rebuild

- 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



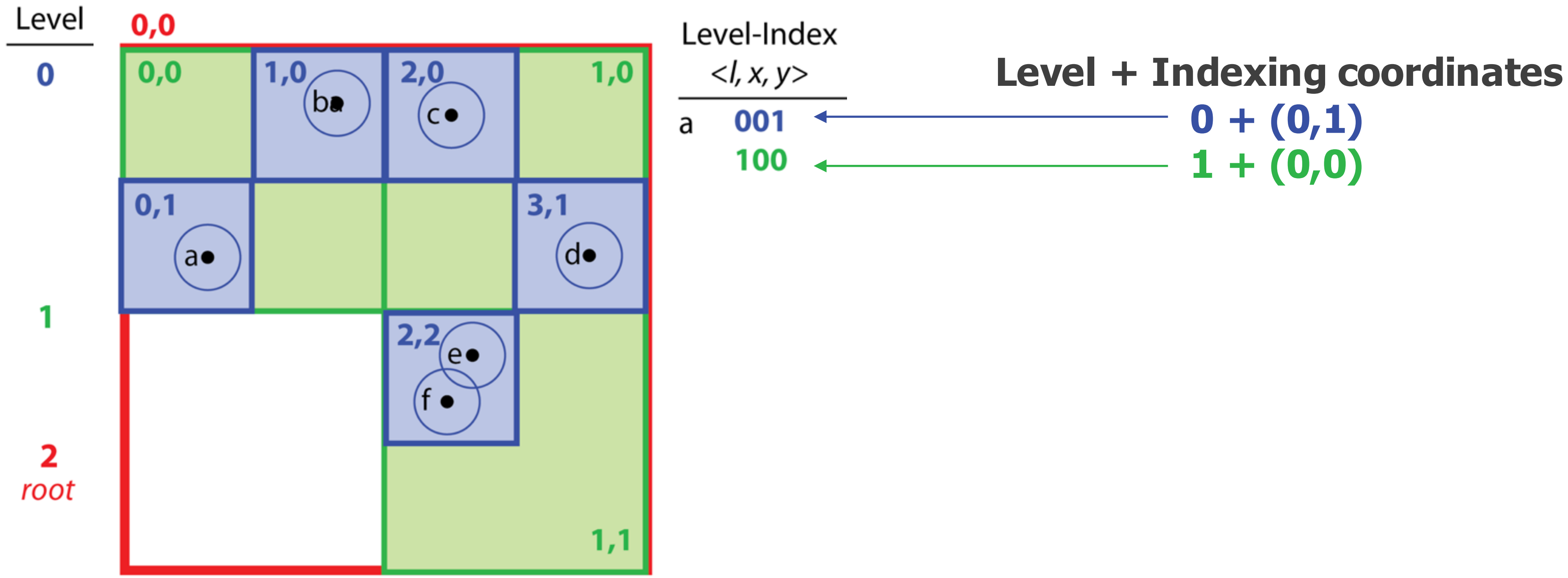
# Full Topology Rebuild

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



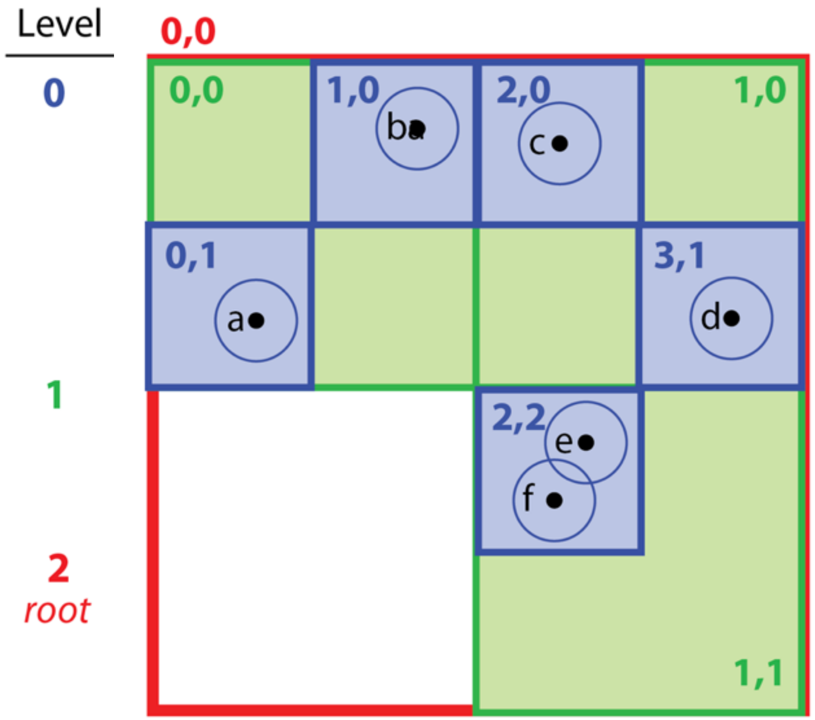
# Full Topology Rebuild

- 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

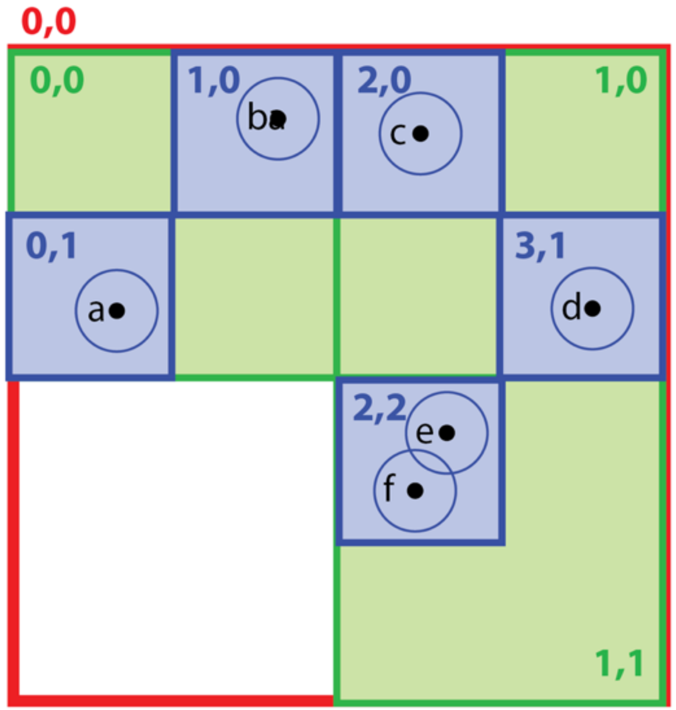
- Determine the number of tree levels L
- Generate the level-index list
- Compact the level-index list
- Allocate and initialize nodes
- Set child node lists




| | Level-Index <l, x, y> |
|---|---|
| a | 001 |
| | 100 |
| b | 010 |
| | 100 |
| c | 020 |
| | 110 |
| d | 031 |
| | 110 |
| e | 022 |
| | 111 |
| f | 022 |
| | 111 |

# 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

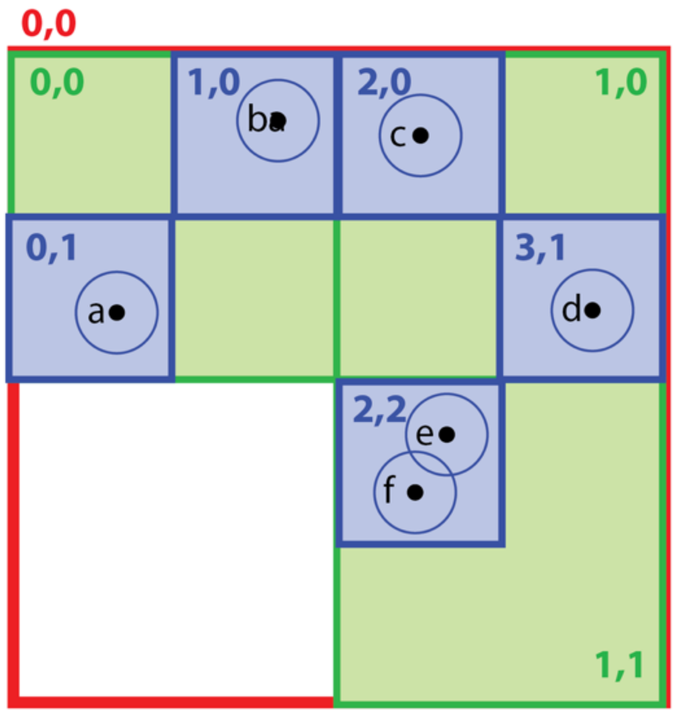



| | Level-Index <*l*, *x*, *y*> | Sorted List | Marker List | Prefix Sum | Compact List | |
|---|---|---|---|---|---|---|
| a | 001 | 001 | 1 | 0 | 0 | 001 |
| | 100 | 010 | 1 | 1 | 1 | 010 |
| b | 010 | 020 | 1 | 2 | 2 | 020 |
| | 100 | 022 | 1 | 3 | 3 | 022 |
| c | 020 | 022 | 0 | 3 | 4 | 031 |
| | 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

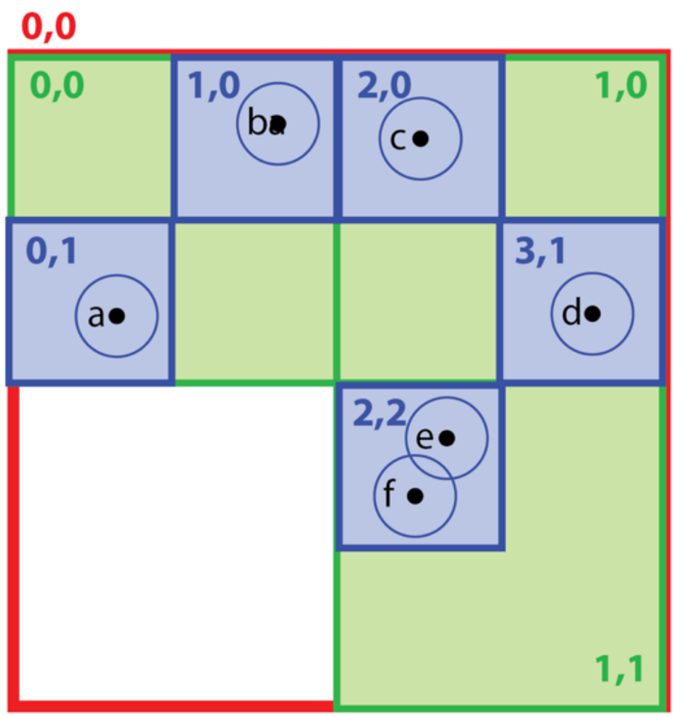
- Allocate and initialize nodes
- Set child node lists




| | Level-Index <*l*, *x*, *y*> | 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

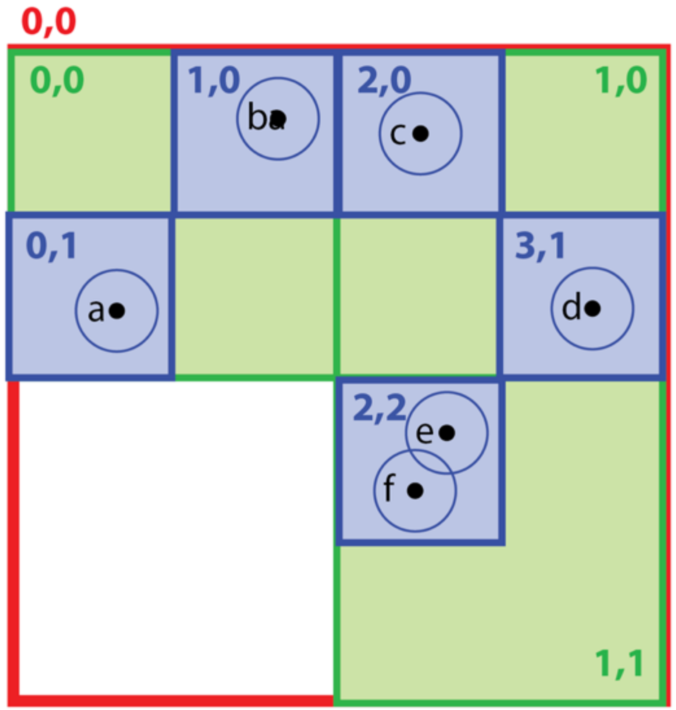
- Determine the number of tree levels L
- Generate the level-index list
- Compact the level-index list
- Allocate and initialize nodes
- Set child node lists




| | Level-Index <l, x, y> | 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

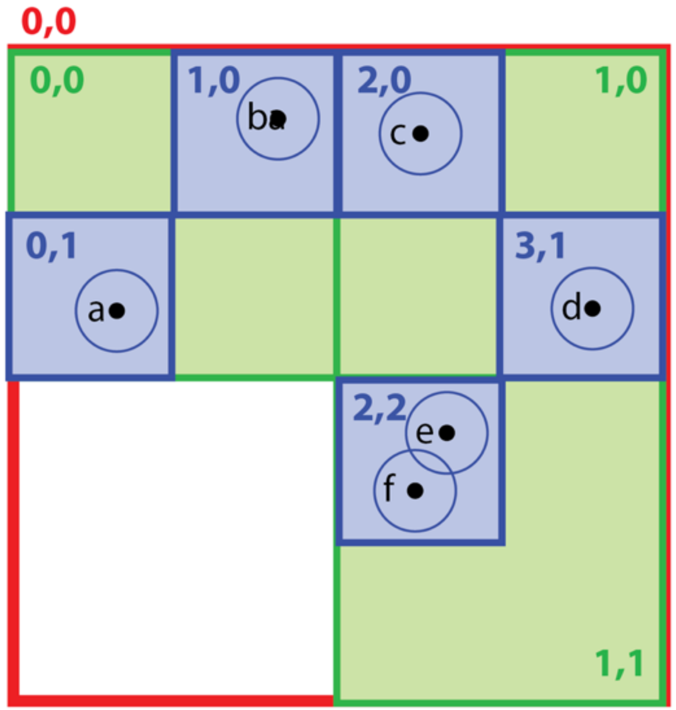
- Determine the number of tree levels L
- Generate the level-index list
- Compact the level-index list
- Allocate and initialize nodes
- Set child node lists



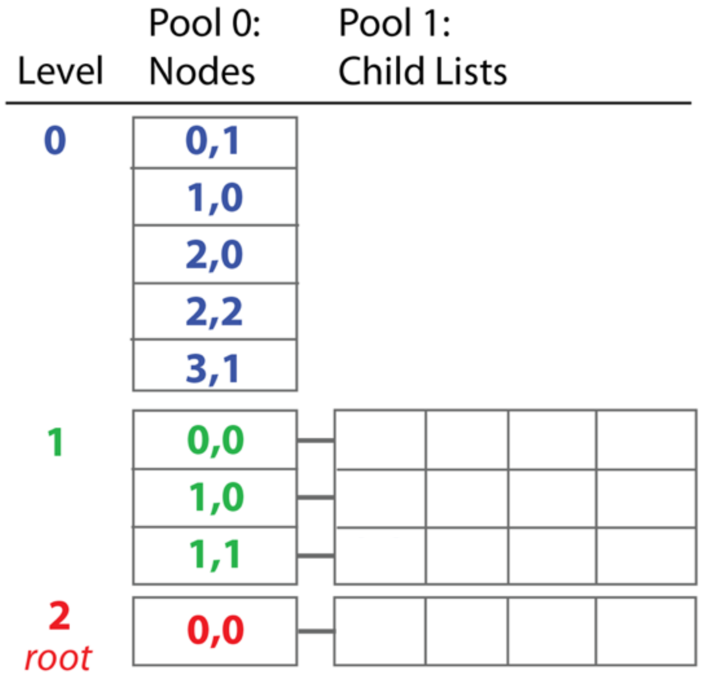

| | Level-Index <l, x, y> | Sorted List | Marker List | Prefix Sum | Compact List |
|---|---|---|---|---|---|
| a | 001 | 001 | 1 | 0 | 0 001 |
| | 100 | 010 | 1 | 1 | 1 010 |
| b | 010 | 020 | 1 | 2 | 2 020 |
| | 100 | 022 | 1 | 3 | 3 022 |
| c | 020 | 022 | 0 | 3 | 4 031 |
| | 110 | 031 | 1 | 4 | 5 100 |
| 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

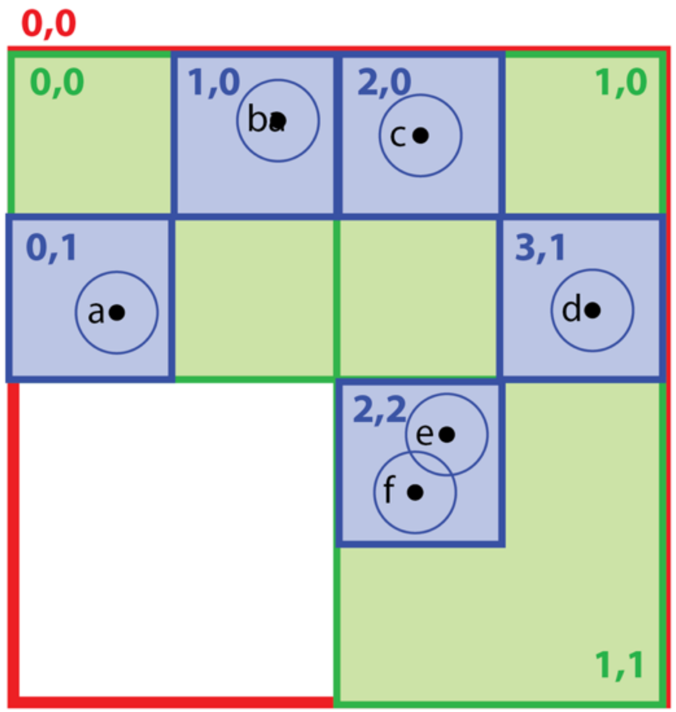



| | Level-Index <l, x, y> | Sorted List | Marker List | Prefix Sum | Compact List | |
|---|---|---|---|---|---|---|
| a | 001 | 001 | 1 | 0 | 0 | 001 |
| | 100 | 010 | 1 | 1 | 1 | 010 |
| b | 010 | 020 | 1 | 2 | 2 | 020 |
| | 100 | 022 | 1 | 3 | 3 | 022 |
| c | 020 | 022 | 0 | 3 | 4 | 031 |
| | 110 | 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

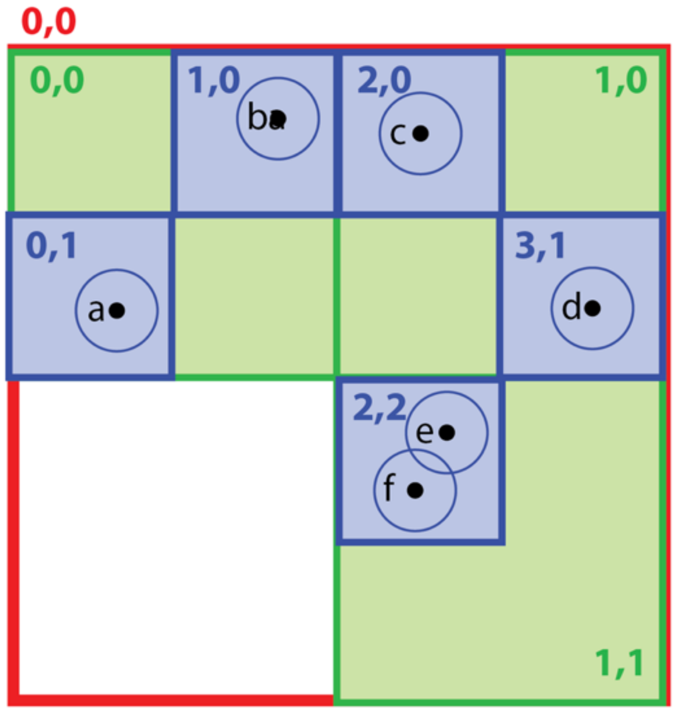



| | Level-Index <*l, x, y*> | Sorted List | Marker List | Prefix Sum | Compact List |
|---|---|---|---|---|---|
| a | 001 | 001 | 1 | 0 | 0 001 |
| | 100 | 010 | 1 | 1 | 1 010 |
| b | 010 | 020 | 1 | 2 | 2 020 |
| | 100 | 022 | 1 | 3 | 3 022 |
| c | 020 | 022 | 0 | 3 | 4 031 |
| | 110 | 031 | 1 | 4 | 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 | |

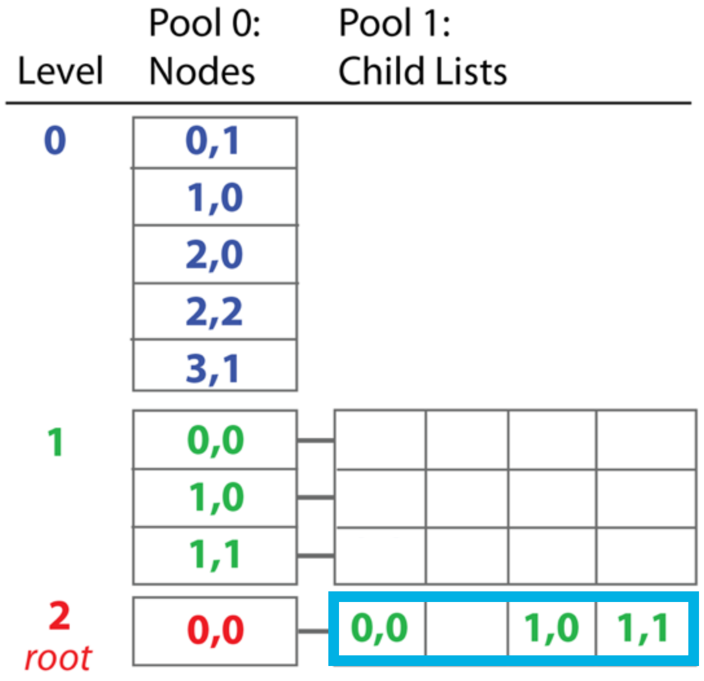
| Level | Pool 0: Nodes | Pool 1: Child Lists |
|---|---|---|
| 0 | 0,1 | |
| | 1,0 | |
| | 2,0 | |
| | 2,2 | |
| | 3,1 | |
| 1 | 0,0 | |
| | 1,0 | |
| | 1,1 | |
| 2 *root* | 0,0 | |

# 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

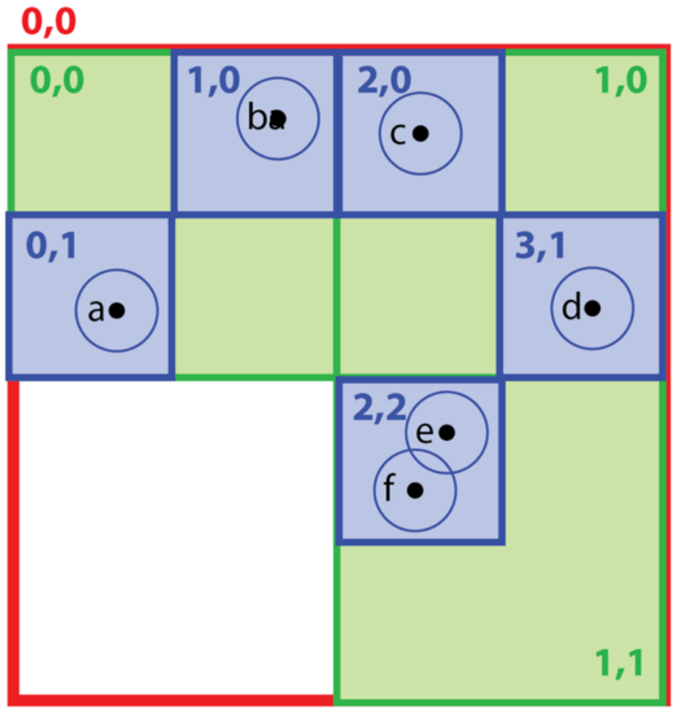



| | Level-Index <*l*, *x*, *y*> | Sorted List | Marker List | Prefix Sum | Compact List |
|---|---|---|---|---|---|
| a | 001 | 001 | 1 | 0 | 0 001 |
| | 100 | 010 | 1 | 1 | 1 010 |
| b | 010 | 020 | 1 | 2 | 2 020 |
| | 100 | 022 | 1 | 3 | 3 022 |
| c | 020 | 022 | 0 | 3 | 4 031 |
| | 110 | 031 | 1 | 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 | |

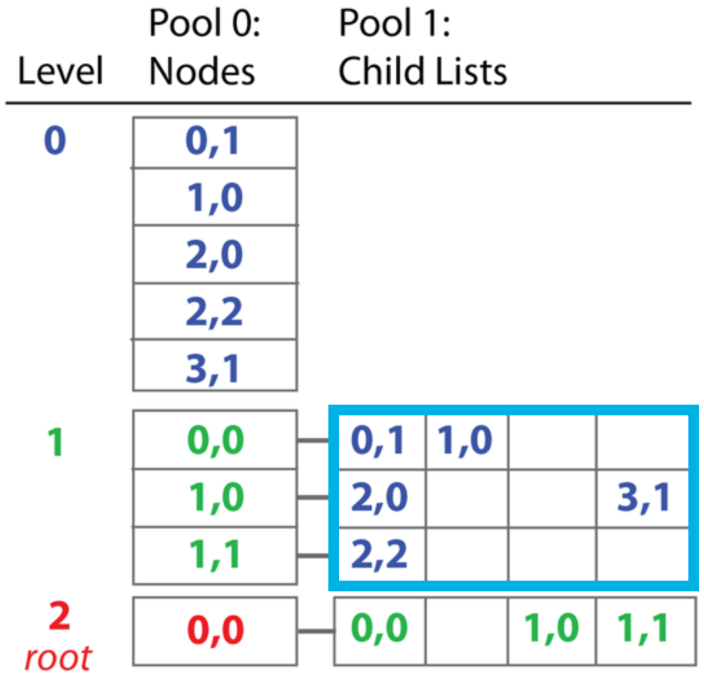


# Full Topology Rebuild

- 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



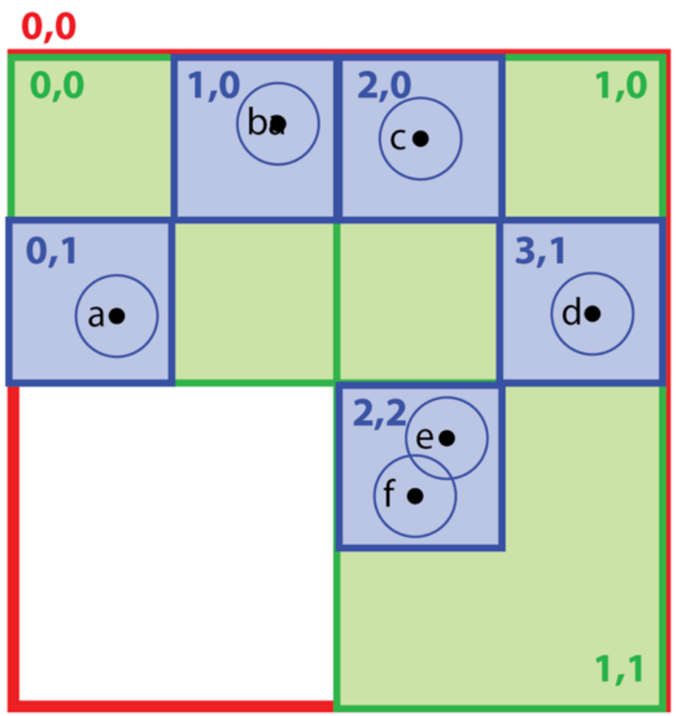

| | Level-Index <*l, x, y*> | Sorted List | Marker List | Prefix Sum | Compact List | |
|---|---|---|---|---|---|---|
| a | 001 | 001 | 1 | 0 | 0 | 001 |
| | 100 | 010 | 1 | 1 | 1 | 010 |
| b | 010 | 020 | 1 | 2 | 2 | 020 |
| | 100 | 022 | 1 | 3 | 3 | 022 |
| c | 020 | 022 | 0 | 3 | 4 | 031 |
| | 110 | 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 | | |



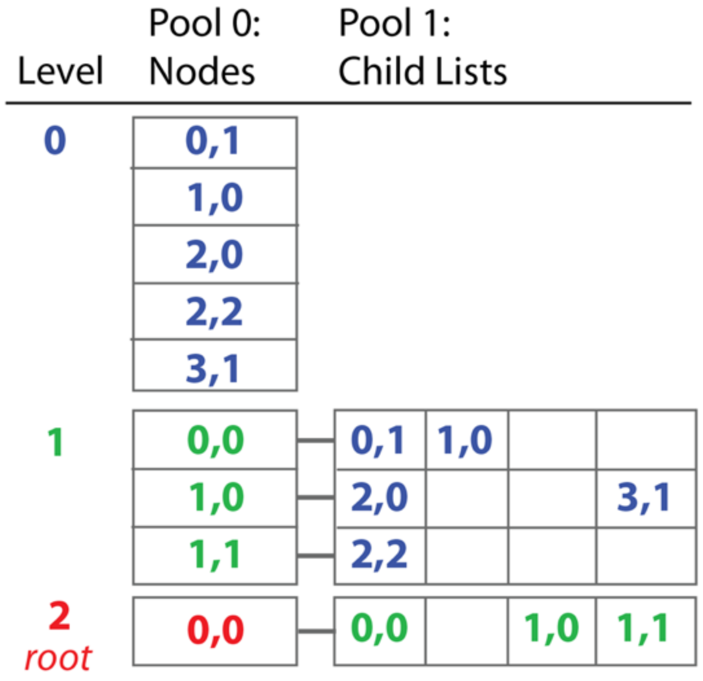
# Full Topology Rebuild

- 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

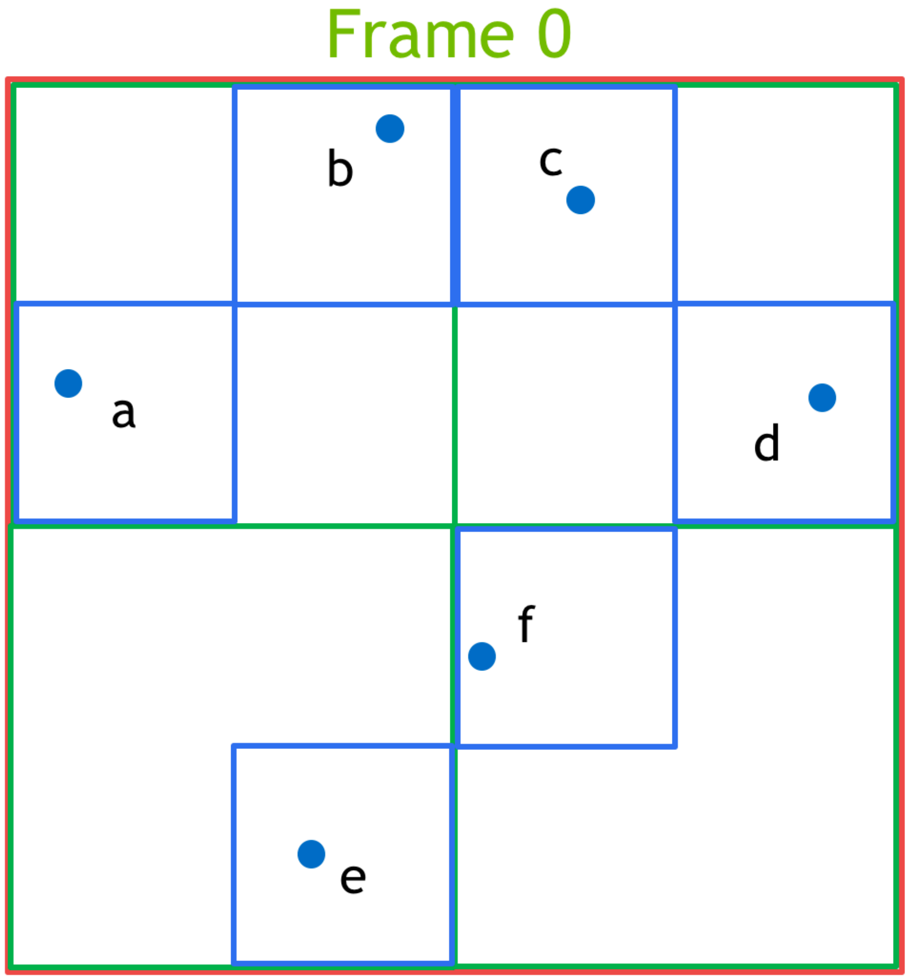
**Very long list** **n=P*L**



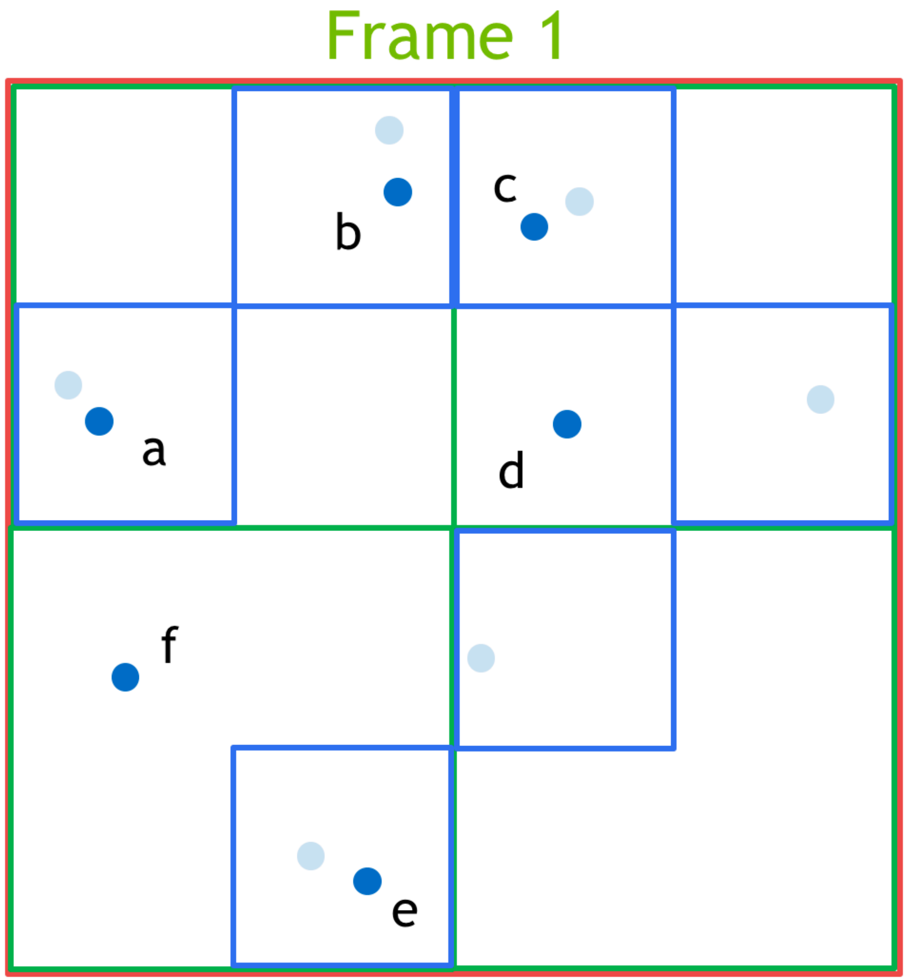

| | Level-Index <l, x, y> | Sorted List | Marker List | Prefix Sum | Compact List |
|---|---|---|---|---|---|
| a | 001 | 001 | 1 | 0 | 0 001 |
| | 100 | 010 | 1 | 1 | 1 010 |
| b | 010 | 020 | 1 | 2 | 2 020 |
| | 100 | 022 | 1 | 3 | 3 022 |
| c | 020 | 022 | 0 | 3 | 4 031 |
| | 110 | 031 | 1 | 4 | 5 100 |
| 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 | |



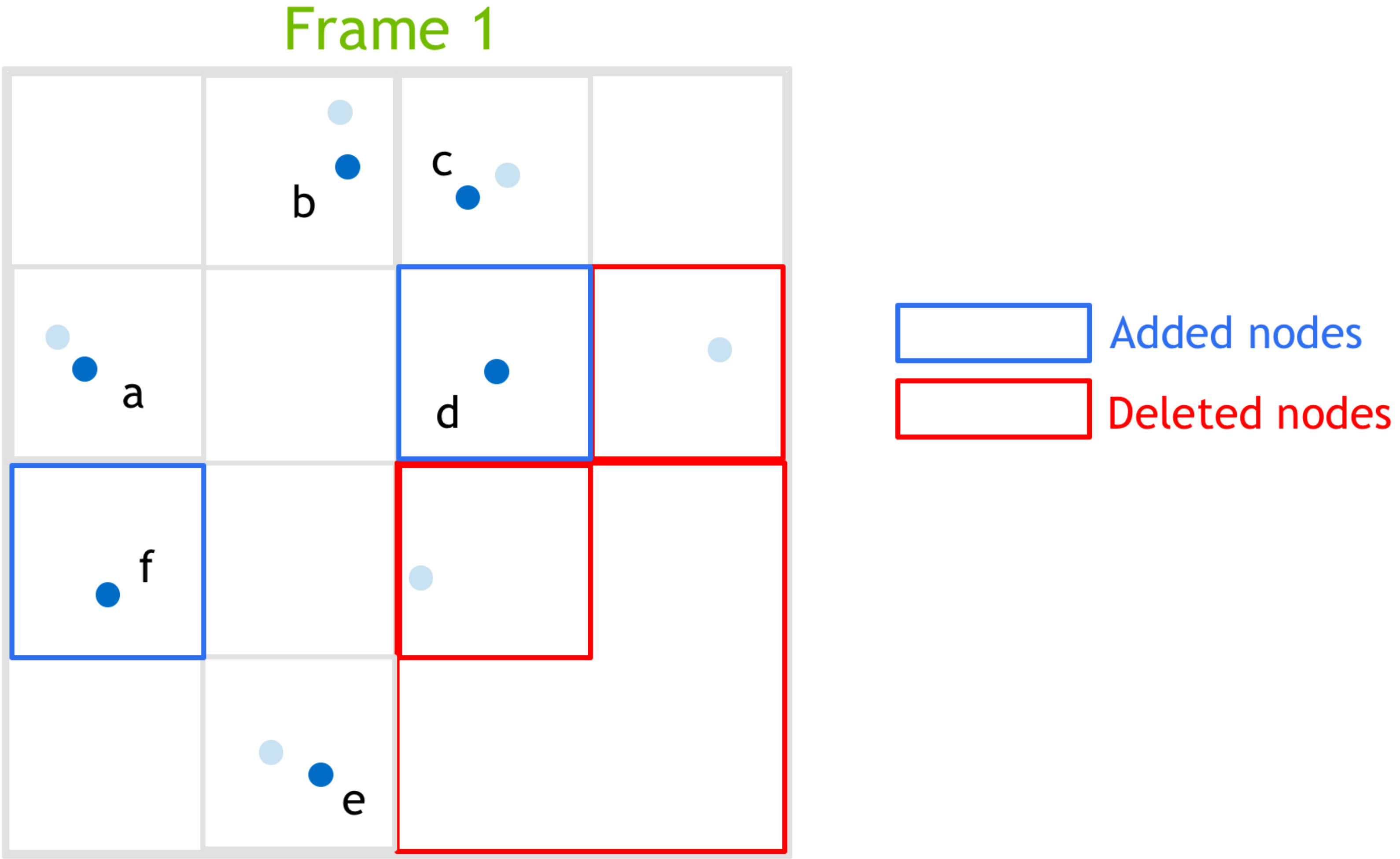
# Incremental Rebuild



# Incremental Rebuild

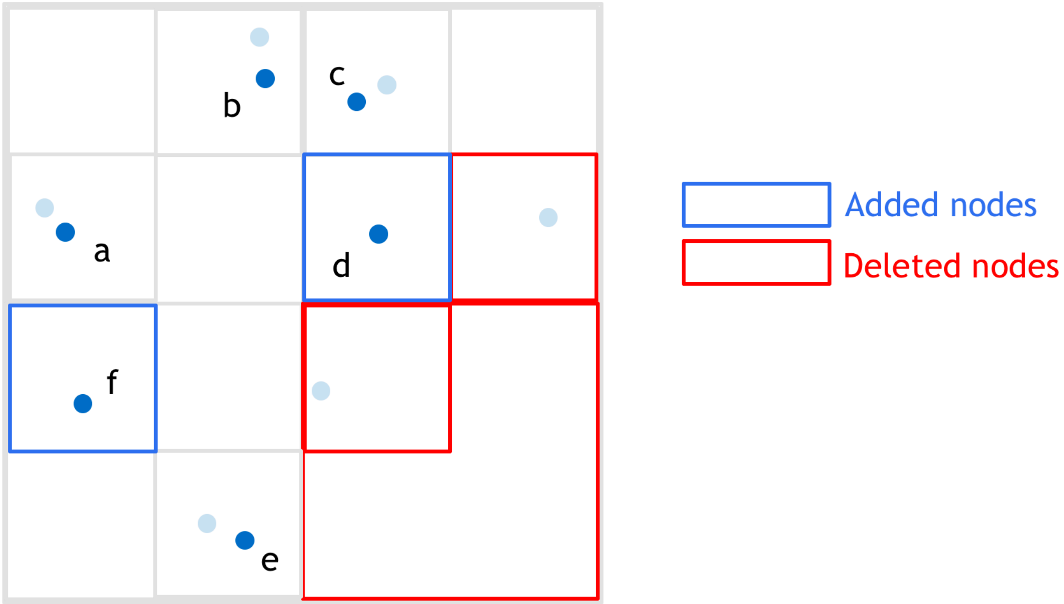


# Incremental Rebuild



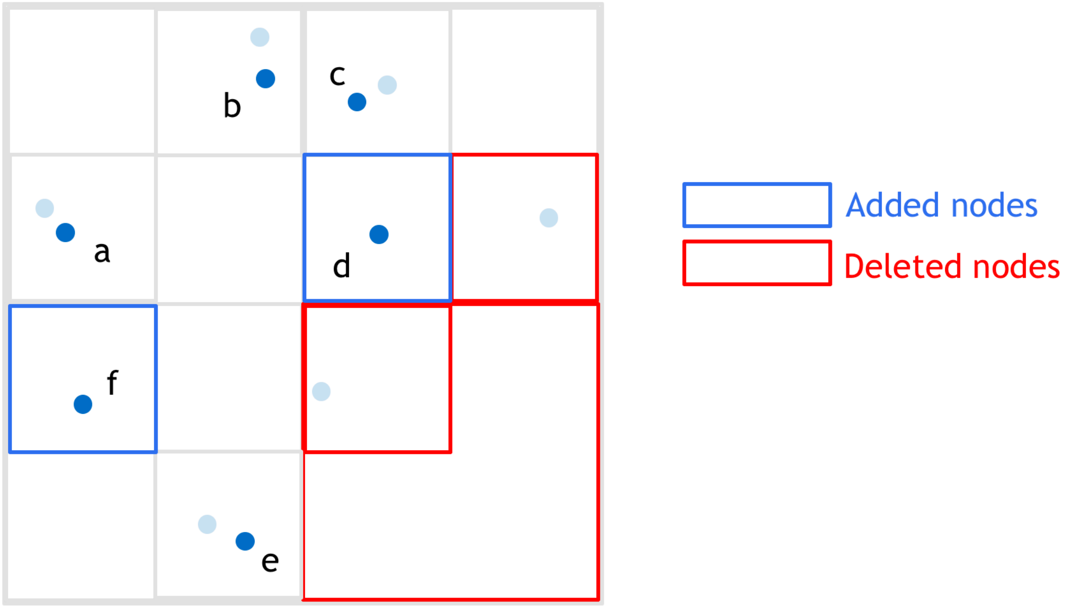
# Incremental Rebuild

- 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*

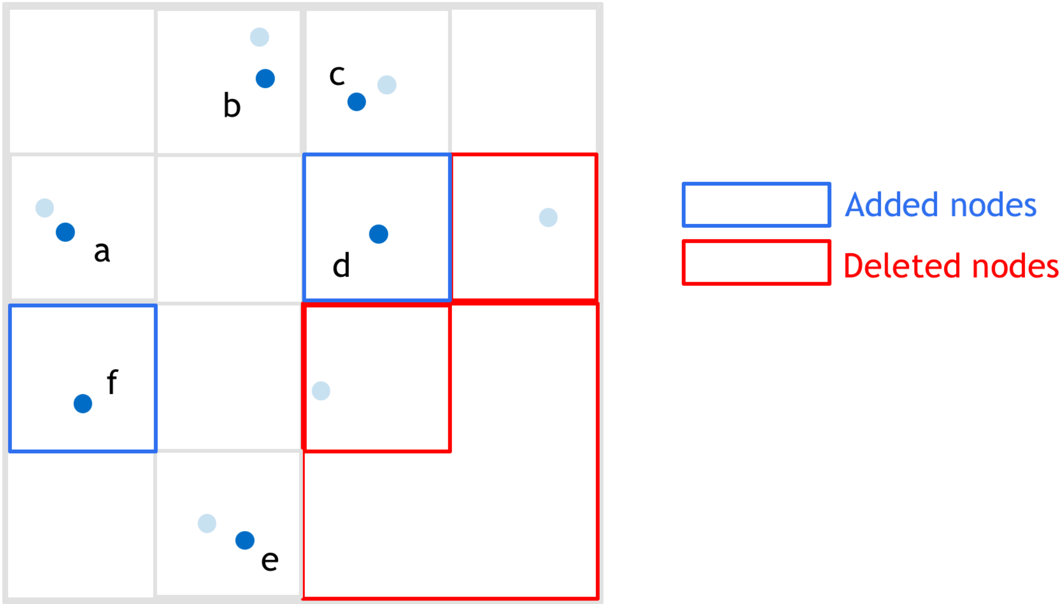



For each particle, check whether already exists a node at that level

- 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×) | **4.8** | (80×) |
| 8M | 807 | 96 | (8×) | **5.6** | (144×) |
| 16M | 1598 | 204 | (8×) | **9.6** | (167×) |
| 32M | 3249 | 313 | (10×) | **18.7** | (174×) |
| 65M | 6368 | 366 | (17×) | **35.5** | (179×) |

in millisecond

# FLIP with Sparse Volumes on the GPU

```
procedure SparseFLIP( )
    P ← initial points
    V ← GVDB structure
    for eachframe do
        if first frame then
            V_topo ← full rebuild (P)
        else
            V_topo ← incremental build (P)
        end if
        V ← resize and clear (V_topo)
        S ← insert points in subcells (V, P)
        V(vel) ← particles-to-voxels (S, P)
        V ← update apron (ρ, vel, marker)
        V(vel_old) ← V(vel)
        V(div) ← divergence (V(vel))
        V(ρ) ← CG pressure solve (V, div)
        V(vel) ← pressure-to-velocity (V(ρ))
        V ← update apron (V(vel))
        P ← advance (V(vel), V(vel_old))
    end for
end procedure
```

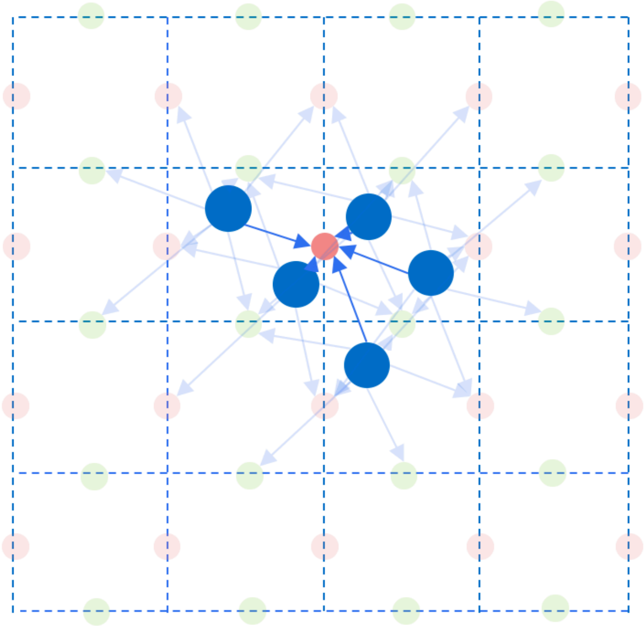
Dynamic Topology

**Particles-to-Voxels**

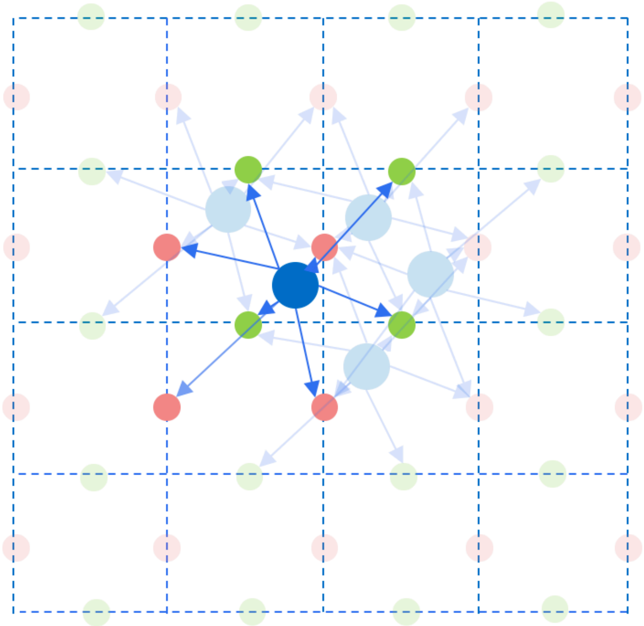
Pressure Solver (CG)

# Particles-to-Voxels

Gathering ✓

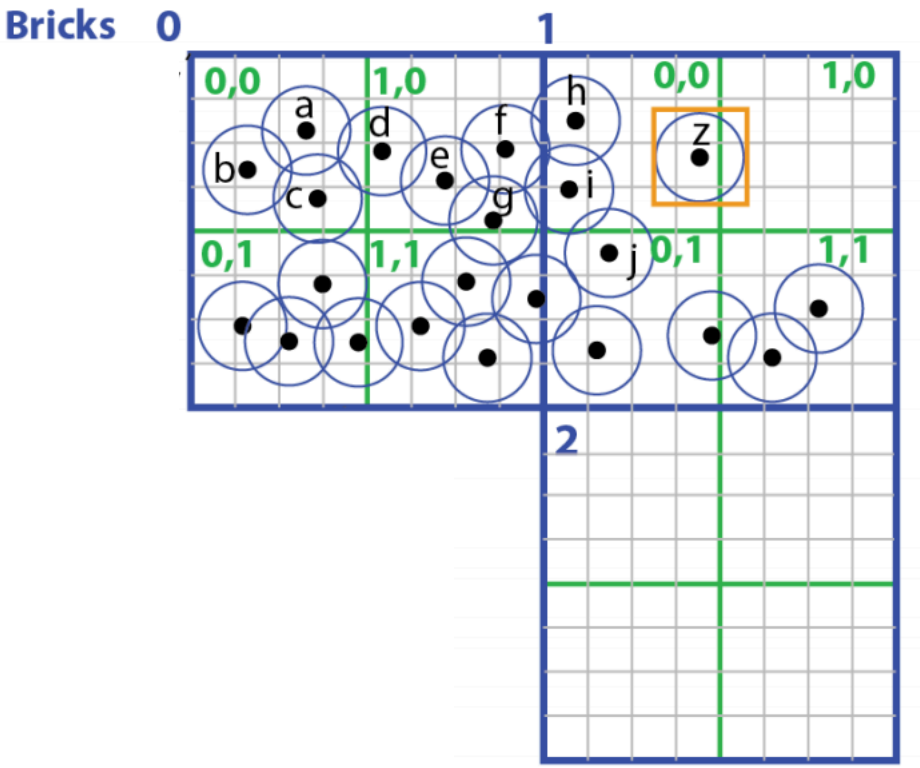


Scattering

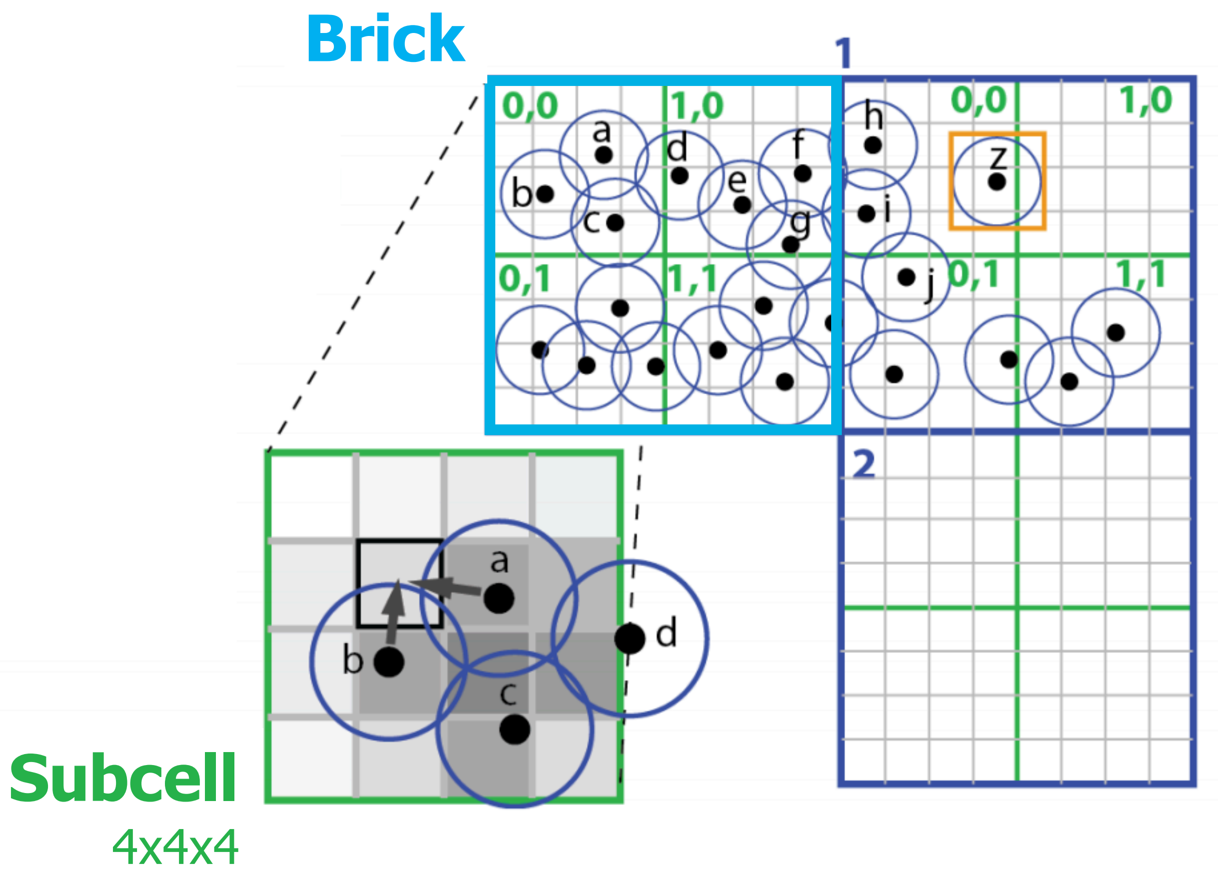


Better for texture writing in parallel

# Subcell



# Subcell



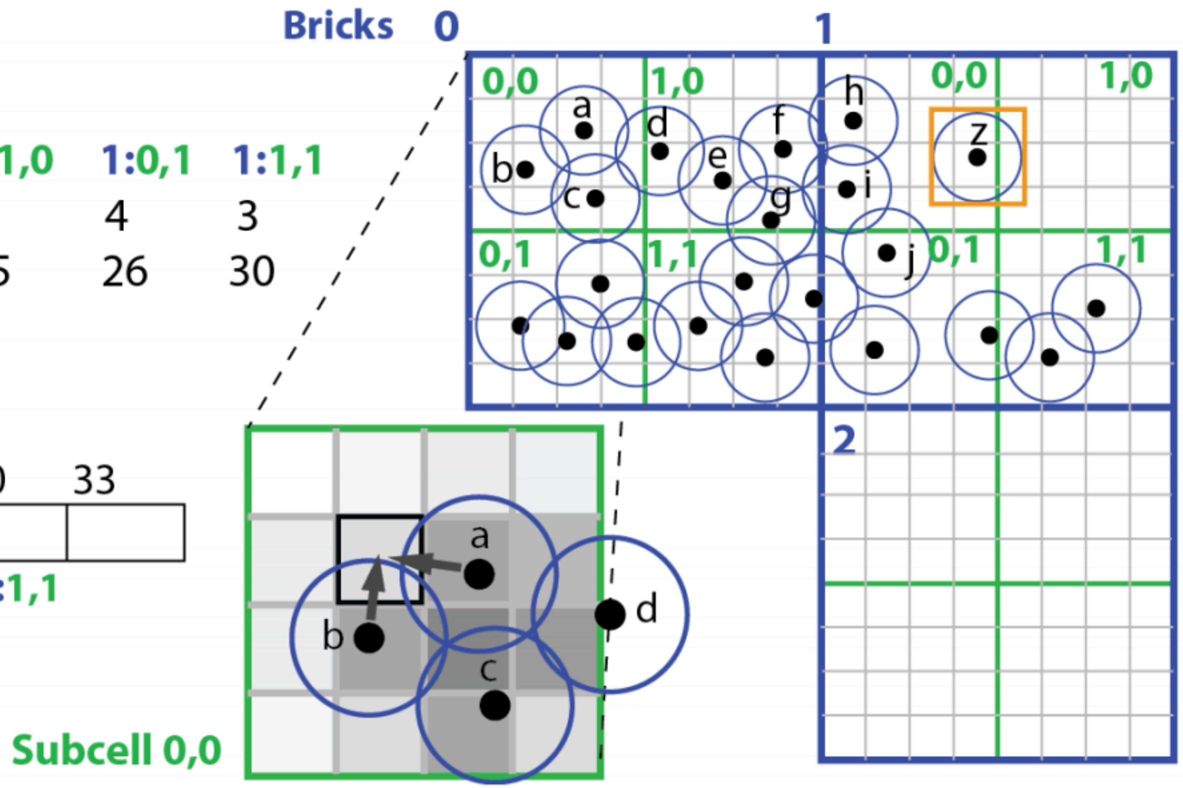
# Subcell

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 | 26 | 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

```
procedure SparseFLIP( )
    P ← initial points
    V ← GVDB structure
    for eachframe do
        if first frame then
            V_topo ← full rebuild (P)
        else
            V_topo ← incremental build (P)
        end if
        V ← resize and clear (V_topo)
        S ← insert points in subcells (V, P)
        V(vel) ← particles-to-voxels (S, P)
        V ← update apron (ρ, vel, marker)
        V(vel_old) ← V(vel)
        V(div) ← divergence (V(vel))
        V(ρ) ← CG pressure solve (V, div)
        V(vel) ← pressure-to-velocity (V(ρ))
        V ← update apron (V(vel))
        P ← advance (V(vel), V(vel_old))
    end for
end procedure
```

Dynamic Topology

Particles-to-Voxels

**Pressure Solver (CG)**

# GPU Matrix-Free Conjugate Gradient Solver

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

```
▷ Given the inputs divergence b, starting value x, maximum of iterations
  i_max, and error tolerance ε
procedure MatrixFreeConjugateGradientSolver(b, x, i_max, ε)
    i = 0
    r = b − SpMV(x)
    d = r
    δ_new = InnerProduct(r, r)
    δ_0 = δ_new
    while i < i_max do
        UpdateApron(d)
        q = SpMV(d)
        α = δ_new / InnerProduct(d, q)
        x = x + αd
        r = b − αq
        δ_old = δ_new
        δ_new = InnerProduct(r, r)
        β = δ_new / δ_old
        d = r + βd
        i = i + 1
        if i mod 10 == 0 then
            if δ_new > ε^2 δ_0 then          ▷ Read back to CPU for check
                Stop solver and return
            end if
        end if
    end while
end procedure
```

# GPU Matrix-Free Conjugate Gradient Solver

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

```
▷ Given the inputs divergence b, starting value x, maximum of iterations
  i_max, and error tolerance ε
procedure MatrixFreeConjugateGradientSolver(b, x, i_max, ε)
    i = 0
    r = b − SpMV(x)
    d = r
    δ_new = InnerProduct(r, r)
    δ_0 = δ_new
    while i < i_max do
        UpdateApron(d)
        q = SpMV(d)
        α = δ_new / InnerProduct(d, q)
        x = x + αd
        r = b − αq
        δ_old = δ_new
        δ_new = InnerProduct(r, r)
        β = δ_new / δ_old
        d = r + βd
        i = i + 1
        if i mod 10 == 0 then
            if δ_new > ε^2 δ_0 then          ▷ Read back to CPU for check
                Stop solver and return
            end if
        end if
    end while
end procedure
```

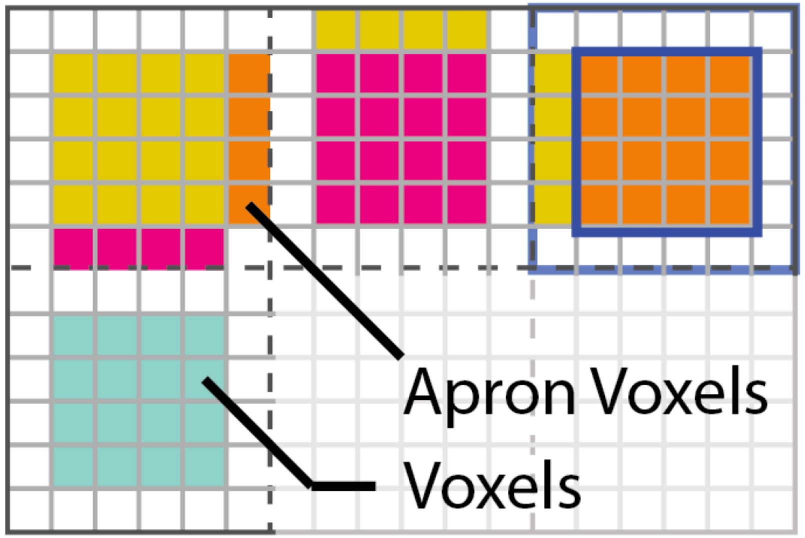
# Sparse Matrix Vector Multiplication (SpMV)

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

▷ *Given the inputs divergence b, starting value x, maximum of iterations* $i_{\max}$*, and error tolerance* $\varepsilon$
**procedure** **MatrixFreeConjugateGradientSolver**$(b, x, i_{\max}, \varepsilon)$
    $i = 0$
    $r = b -$ **SpMV**$(x)$
    $d = r$
    $\delta_{new} =$ **InnerProduct**$(r, r)$
    $\delta_0 = \delta_{new}$
    **while** $i < i_{\max}$ **do**
        **UpdateApron**$(d)$
        $q =$ **SpMV**$(d)$
        $\alpha = \delta_{new} /$ **InnerProduct**$(d, q)$
        $x = x + \alpha d$
        $r = b - \alpha q$
        $\delta_{old} = \delta_{new}$
        $\delta_{new} =$ **InnerProduct**$(r, r)$
        $\beta = \delta_{new} / \delta_{old}$
        $d = r + \beta d$
        $i = i + 1$
        **if** $i$ mod $10 == 0$ **then**
            **if** $\delta_{new} > \varepsilon^2 \delta_0$ **then** ▷ *Read back to CPU for check*
                Stop solver and return
            **end if**
        **end if**
    **end while**
**end procedure**

**SpMV**

- 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



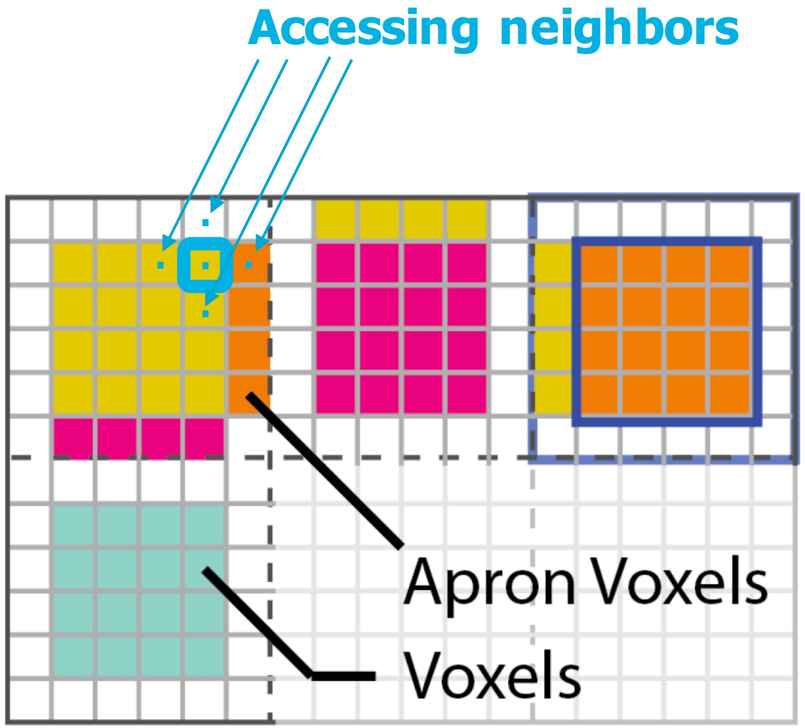
# Sparse Matrix Vector Multiplication (SpMV)

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

```
▷ Given the inputs divergence b, starting value x, maximum of iterations
  i_max, and error tolerance ε
procedure MatrixFreeConjugateGradientSolver(b, x, i_max, ε)
    i = 0
    r = b − SpMV(x)
    d = r
    δ_new = InnerProduct(r, r)
    δ_0 = δ_new
    while i < i_max do
        UpdateApron(d)
        q = SpMV(d)
        α = δ_new / InnerProduct(d, q)
        x = x + αd
        r = b − αq
        δ_old = δ_new
        δ_new = InnerProduct(r, r)
        β = δ_new / δ_old
        d = r + βd
        i = i + 1
        if i mod 10 == 0 then
            if δ_new > ε²δ_0 then            ▷ Read back to CPU for check
                Stop solver and return
            end if
        end if
    end while
end procedure
```

SpMV

- 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

▷ *Given the inputs divergence b, starting value x, maximum of iterations* $i_{\max}$, *and error tolerance* $\varepsilon$

**procedure** **MatrixFreeConjugateGradientSolver**$(b, x, i_{\max}, \varepsilon)$

  $i = 0$

  $r = b - \textbf{SpMV}(x)$

  $d = r$

  $\delta_{new} = \textbf{InnerProduct}(r, r)$

  $\delta_0 = \delta_{new}$

  **while** $i < i_{\max}$ **do**

    **UpdateApron**$(d)$ ← **UpdateArpon**

    $q = \textbf{SpMV}(d)$

    $\alpha = \delta_{new} / \textbf{InnerProduct}(d, q)$

    $x = x + \alpha d$

    $r = b - \alpha q$

    $\delta_{old} = \delta_{new}$

    $\delta_{new} = \textbf{InnerProduct}(r, r)$

    $\beta = \delta_{new} / \delta_{old}$

    $d = r + \beta d$

    $i = i + 1$

    **if** $i \bmod 10 == 0$ **then**

      **if** $\delta_{new} > \varepsilon^2 \delta_0$ **then** ▷ *Read back to CPU for check*

        Stop solver and return

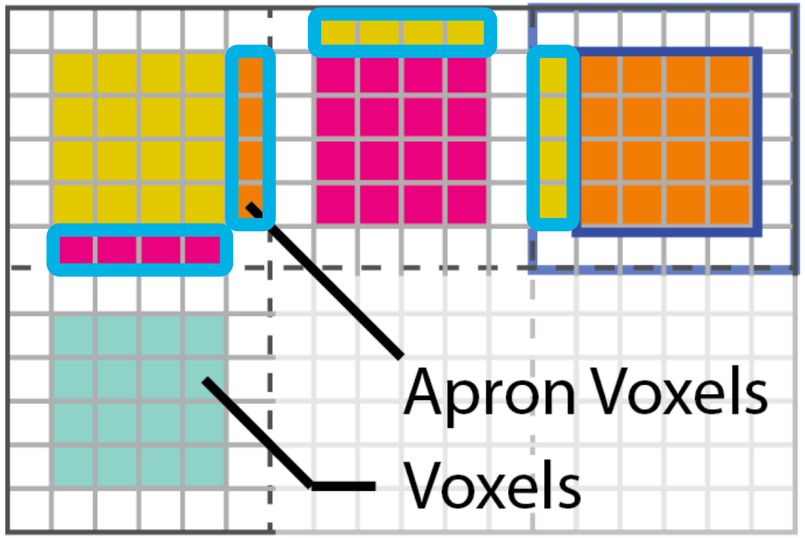
      **end if**

    **end if**

  **end while**

**end procedure**

- Update apron voxels before using it

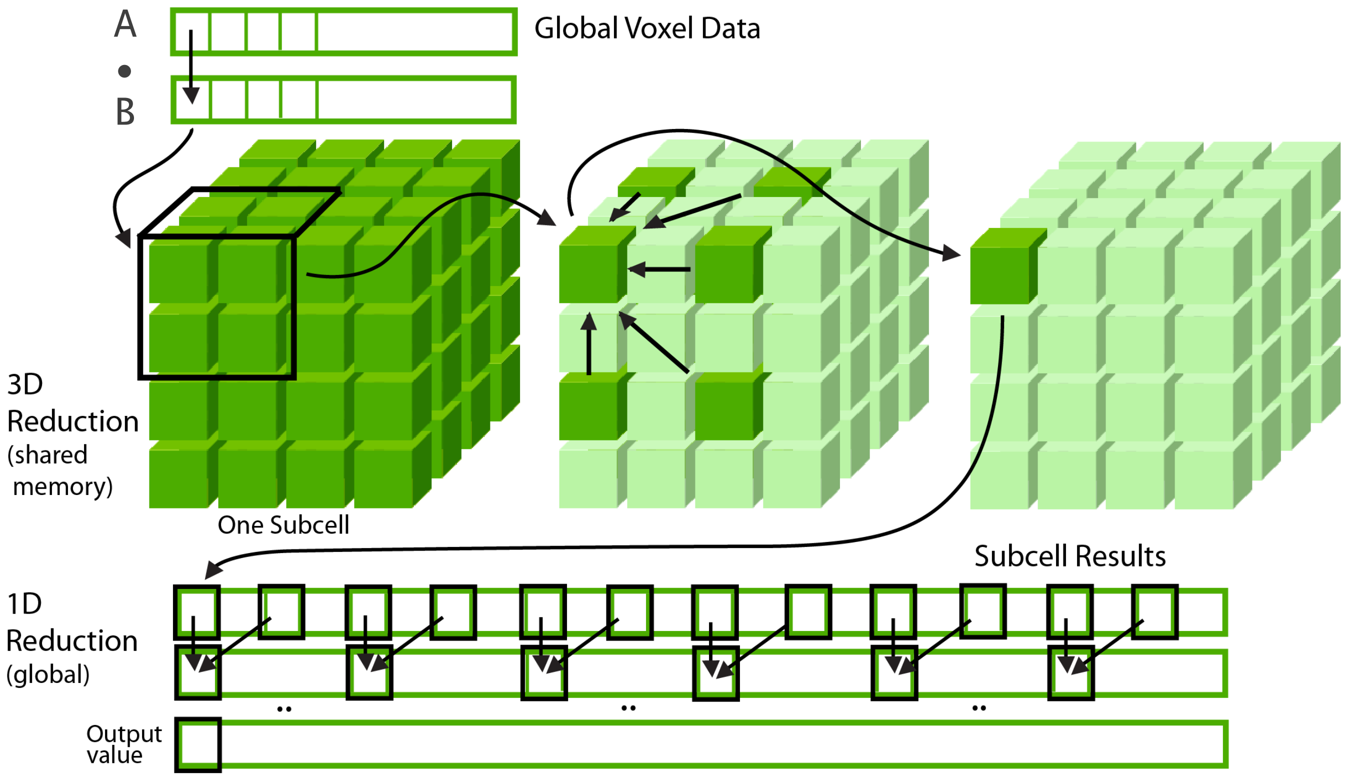


# Inner Product

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

```
▷ Given the inputs divergence b, starting value x, maximum of iterations
  i_max, and error tolerance ε
procedure MatrixFreeConjugateGradientSolver(b, x, i_max, ε)
    i = 0
    r = b − SpMV(x)
    d = r
    δ_new = InnerProduct(r, r)
    δ_0 = δ_new
    while i < i_max do
        UpdateApron(d)
        q = SpMV(d)
        α = δ_new / InnerProduct(d, q)
        x = x + αd
        r = b − αq
        δ_old = δ_new
        δ_new = InnerProduct(r, r)
        β = δ_new / δ_old
        d = r + βd
        i = i + 1
        if i mod 10 == 0 then
            if δ_new > ε²δ_0 then                ▷ Read back to CPU for check
                Stop solver and return
            end if
        end if
    end while
end procedure
```

**InnerProduct**

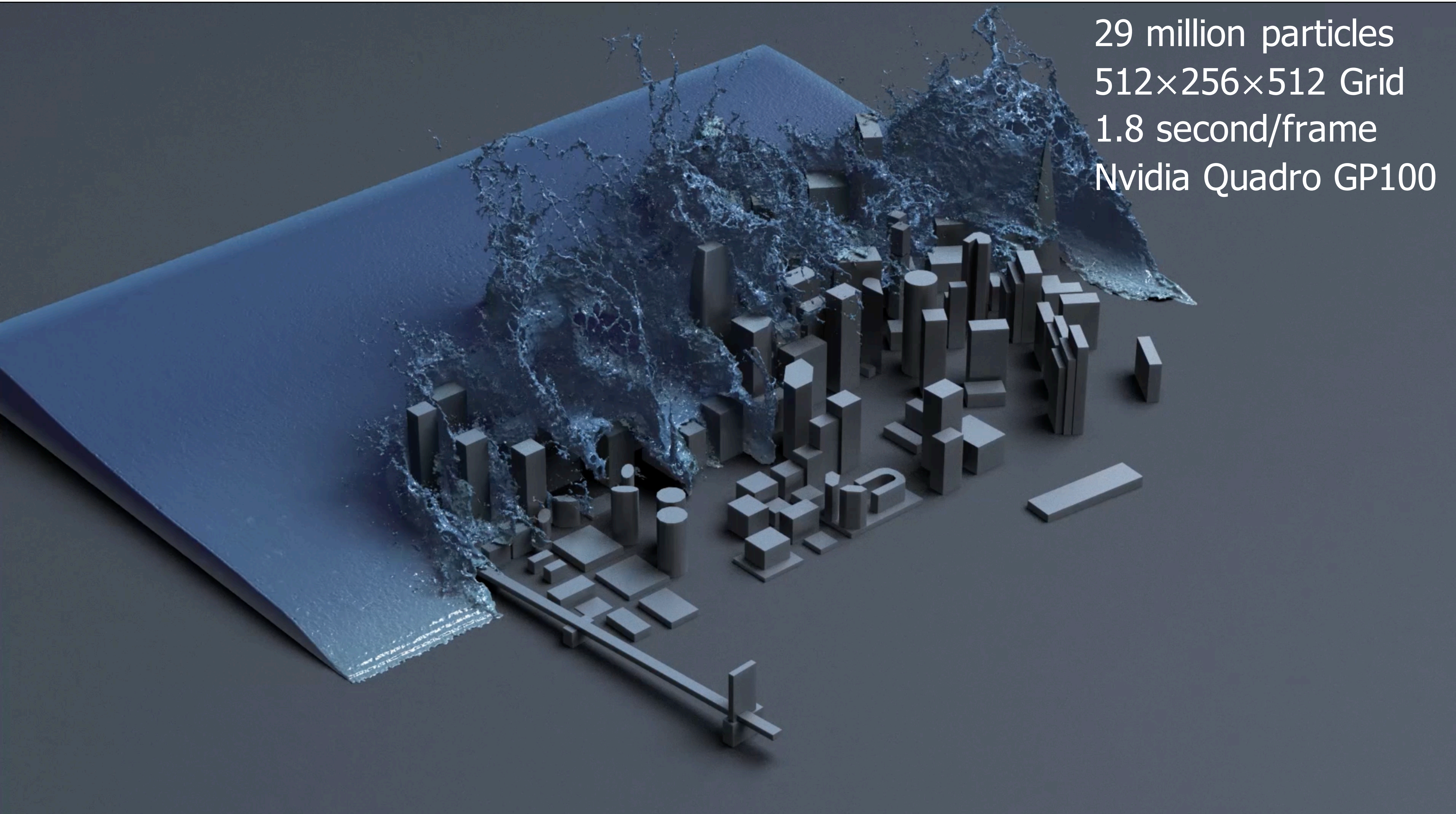


# GPU Matrix-Free Conjugate Gradient Solver

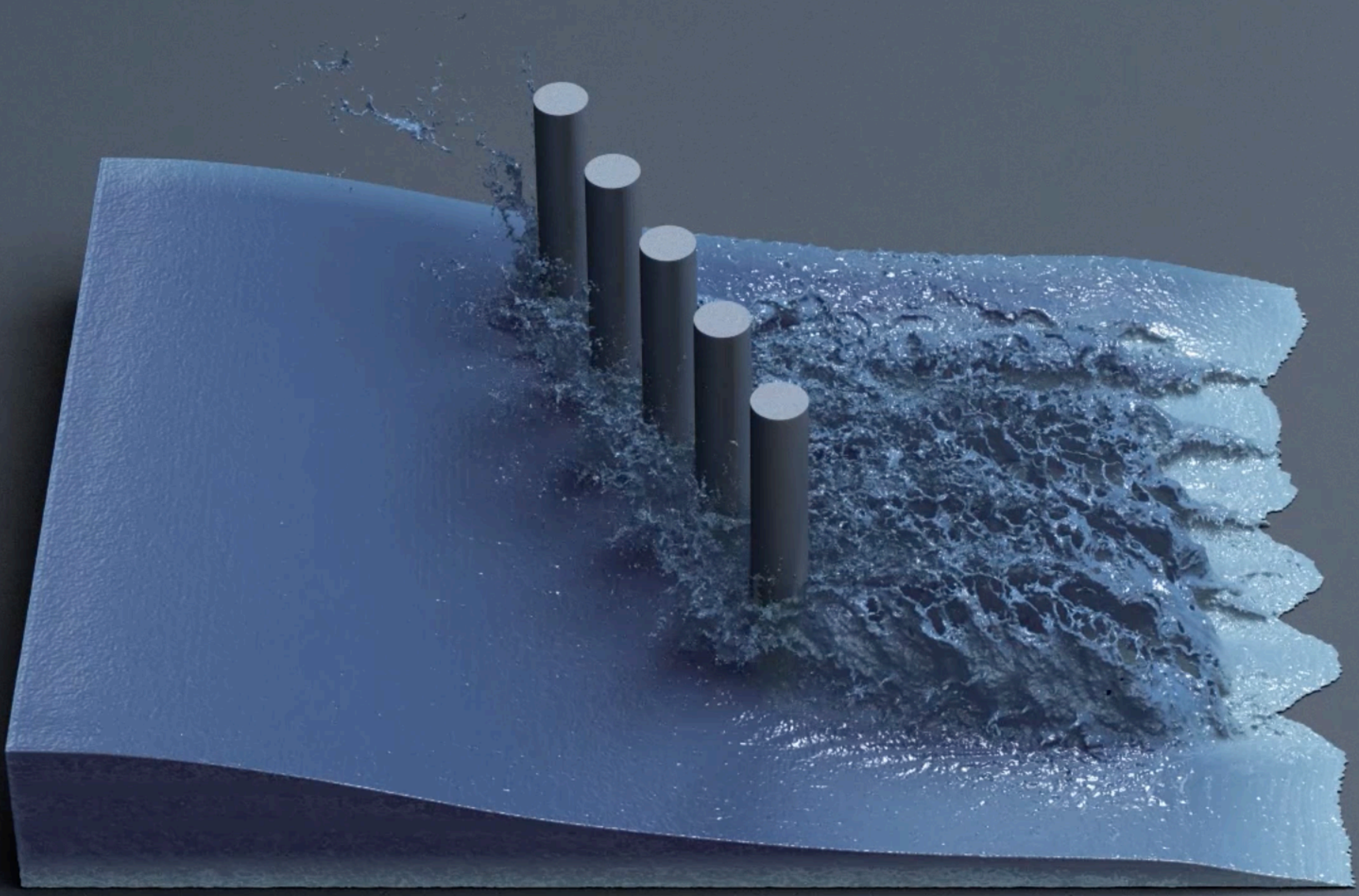
**Algorithm 2** Matrix-free Conjugate Gradient Solver

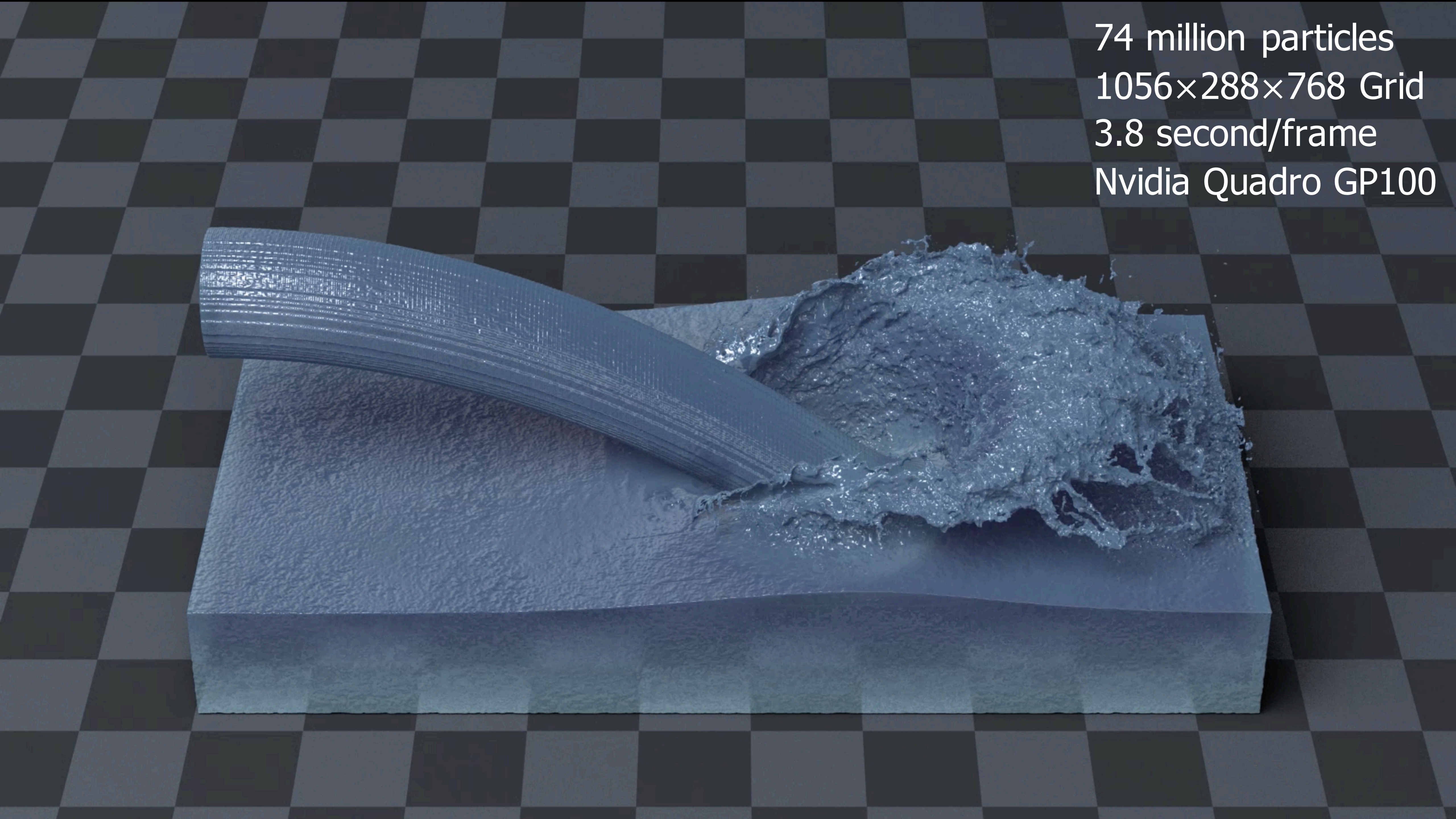
▷ *Given the inputs divergence b, starting value x, maximum of iterations* $i_{\max}$, *and error tolerance* $\varepsilon$

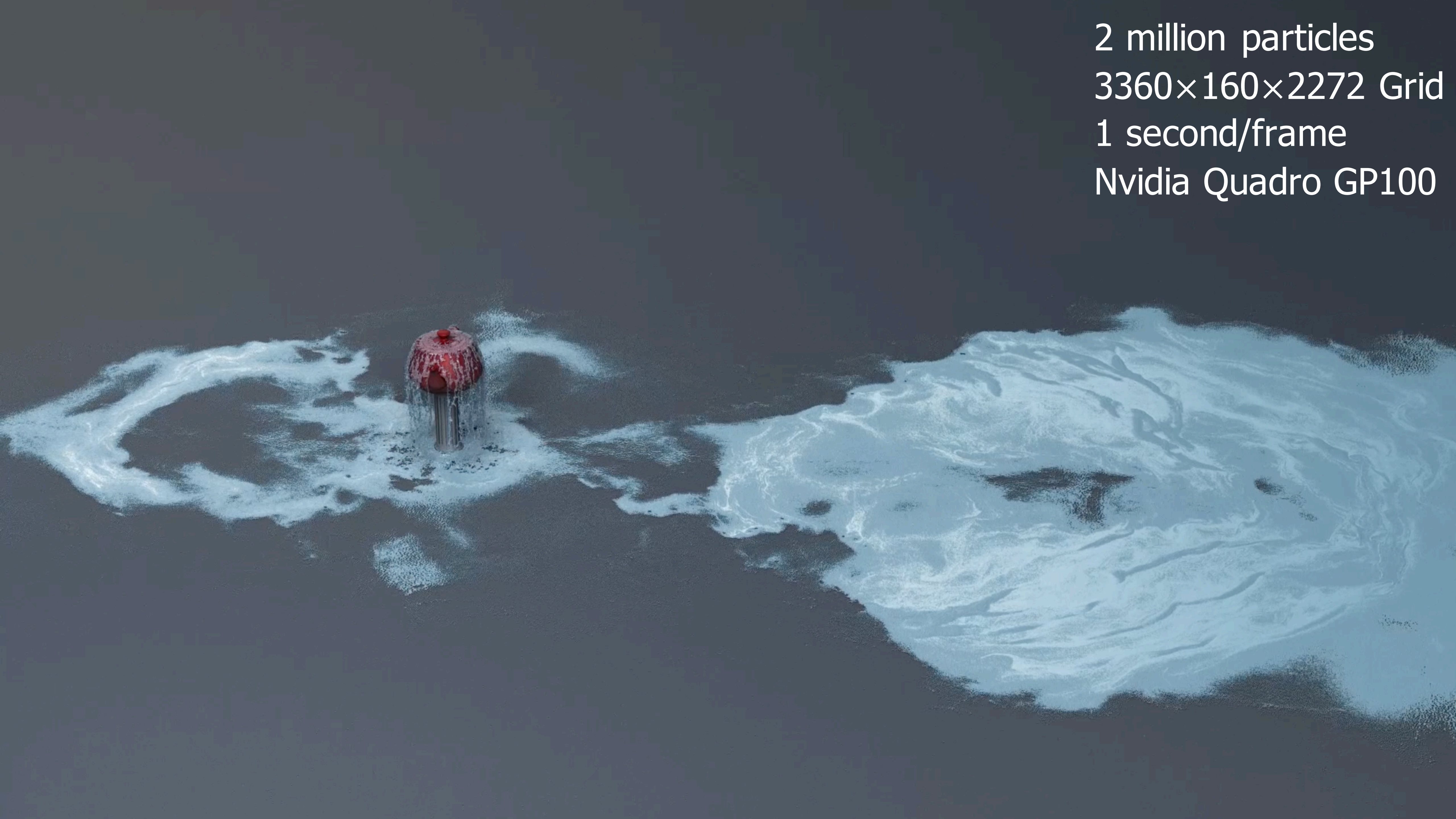
**procedure** **MatrixFreeConjugateGradientSolver**$(b, x, i_{\max}, \varepsilon)$
    $i = 0$
    $r = b -$ **SpMV**$(x)$
    $d = r$
    $\delta_{new} =$ **InnerProduct**$(r, r)$
    $\delta_0 = \delta_{new}$
    **while** $i < i_{\max}$ **do**
        **UpdateApron**$(d)$
        $q =$ **SpMV**$(d)$
        $\alpha = \delta_{new}/$**InnerProduct**$(d, q)$
        $x = x + \alpha d$
        $r = b - \alpha q$
        $\delta_{old} = \delta_{new}$
        $\delta_{new} =$ **InnerProduct**$(r, r)$
        $\beta = \delta_{new}/\delta_{old}$
        $d = r + \beta d$
        $i = i + 1$
        **if** $i \bmod 10 == 0$ **then**
            **if** $\delta_{new} > \varepsilon^2 \delta_0$ **then** ▷ *Read back to CPU for check*
                Stop solver and return
            **end if**
        **end if**
    **end while**
**end procedure**


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

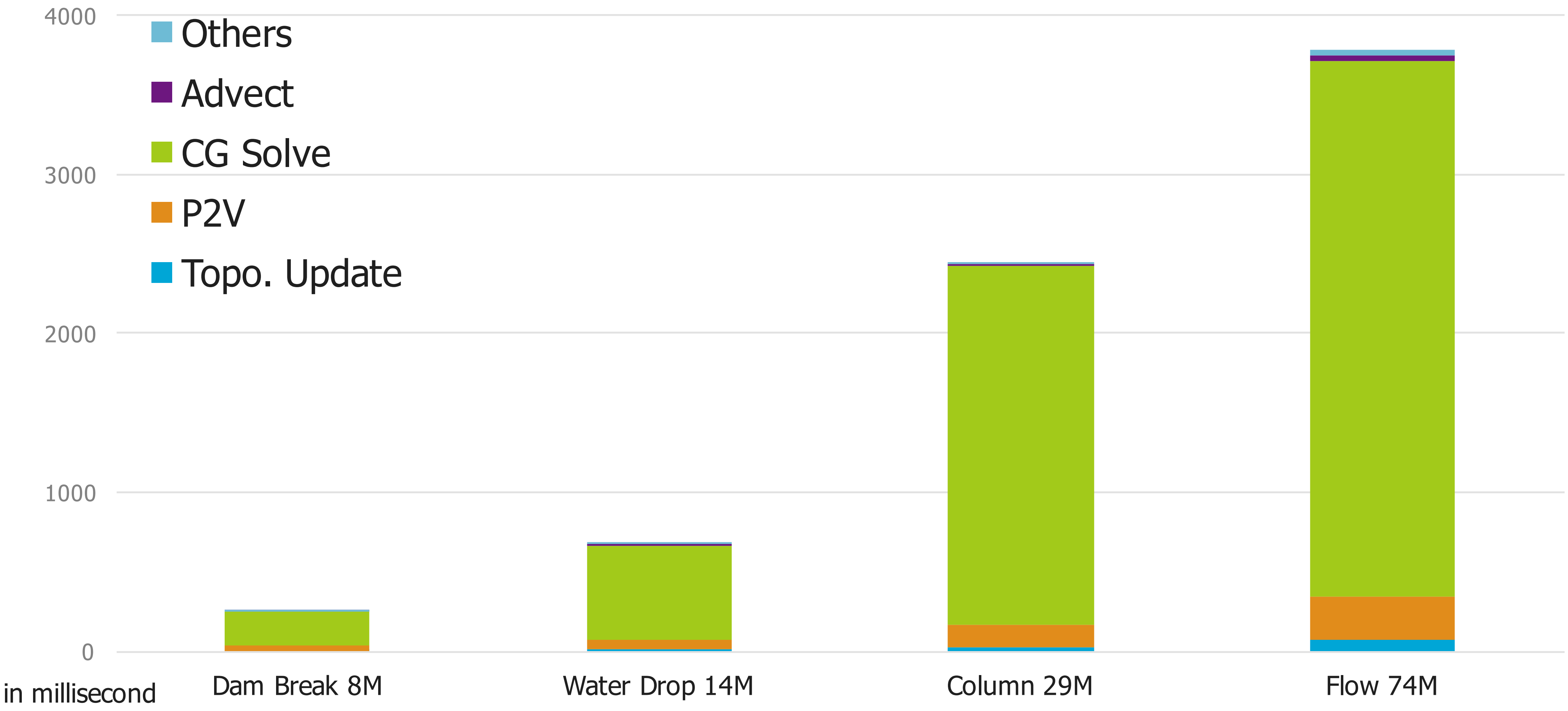
29 million particles
450×300×300 Grid
2.4 second/frame
Nvidia Quadro GP100



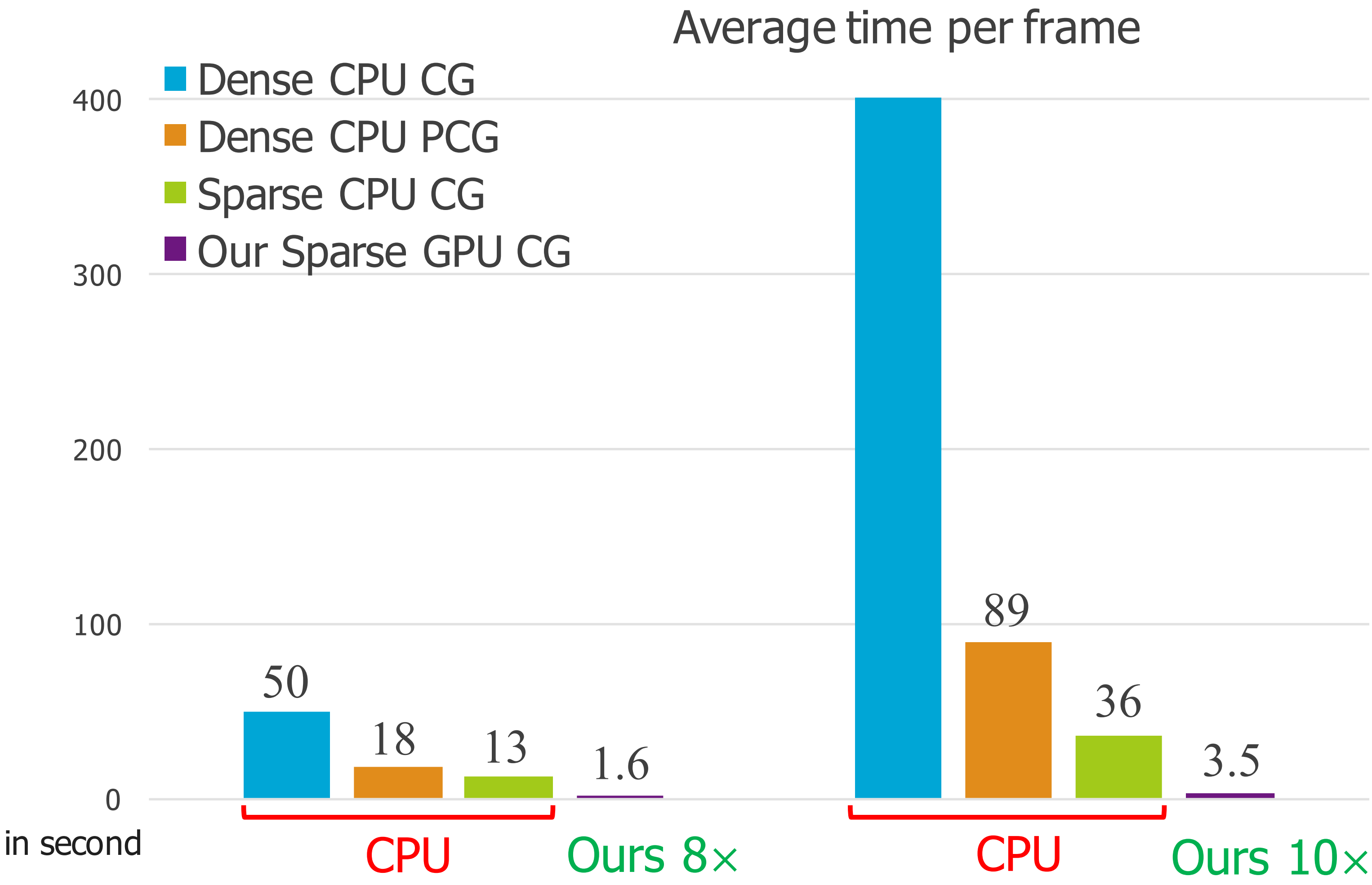

74 million particles
1056×288×768 Grid
3.8 second/frame
Nvidia Quadro GP100

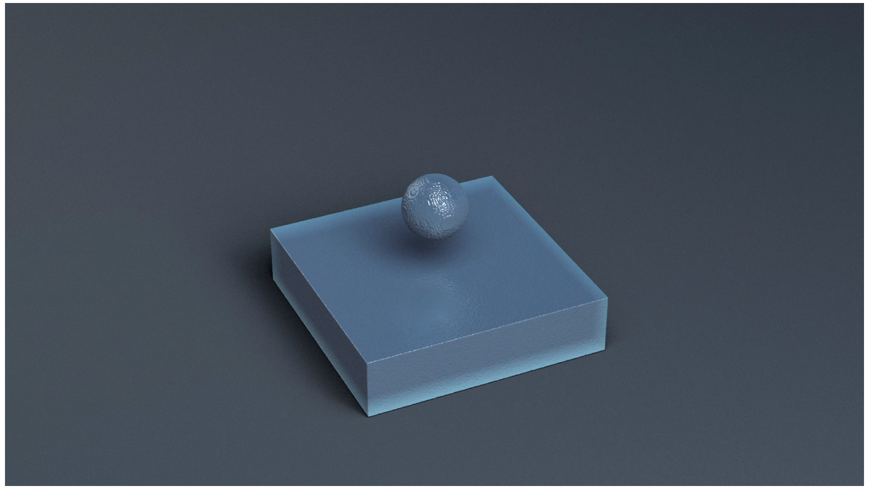

2 million particles
3360×160×2272 Grid
1 second/frame
Nvidia Quadro GP100

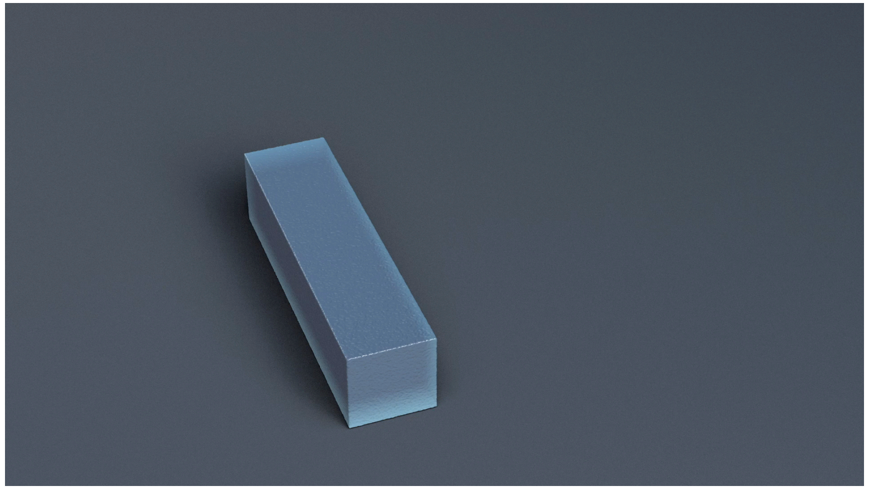
# Average Time per Frame



# GPU Matrix-free CG Solver





# Conclusions

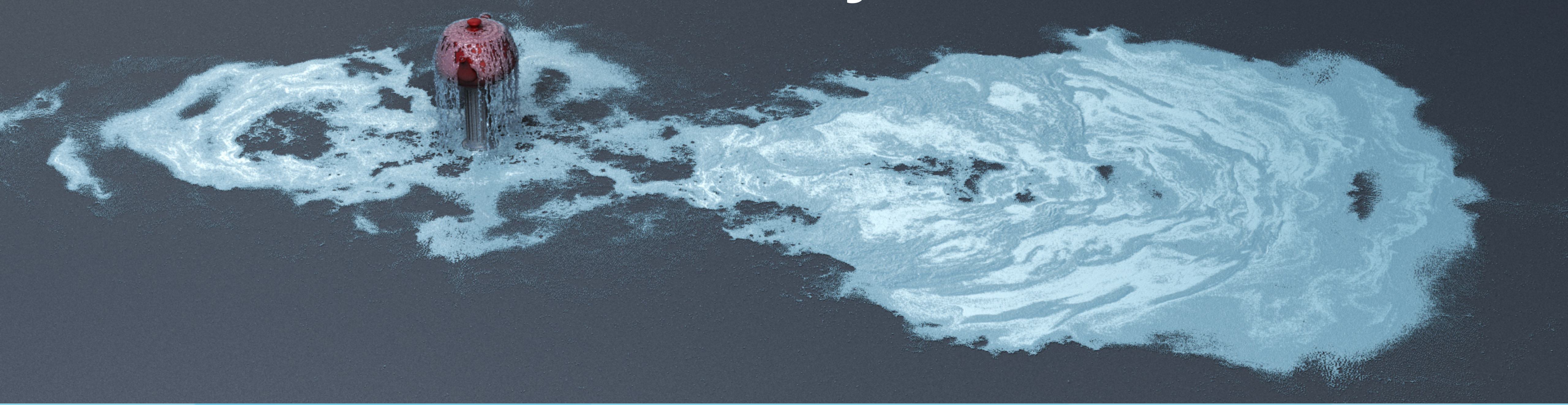
We have presented a sparse, efficient, GPU-based FLIP simulation for fluids on virtually unbounded domains

- 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

EUROGRAPHICS 2018

NVIDIA®