OpenVDB

Ken Museth

DreamWorks Animation SKG

SIGGRAPH2013

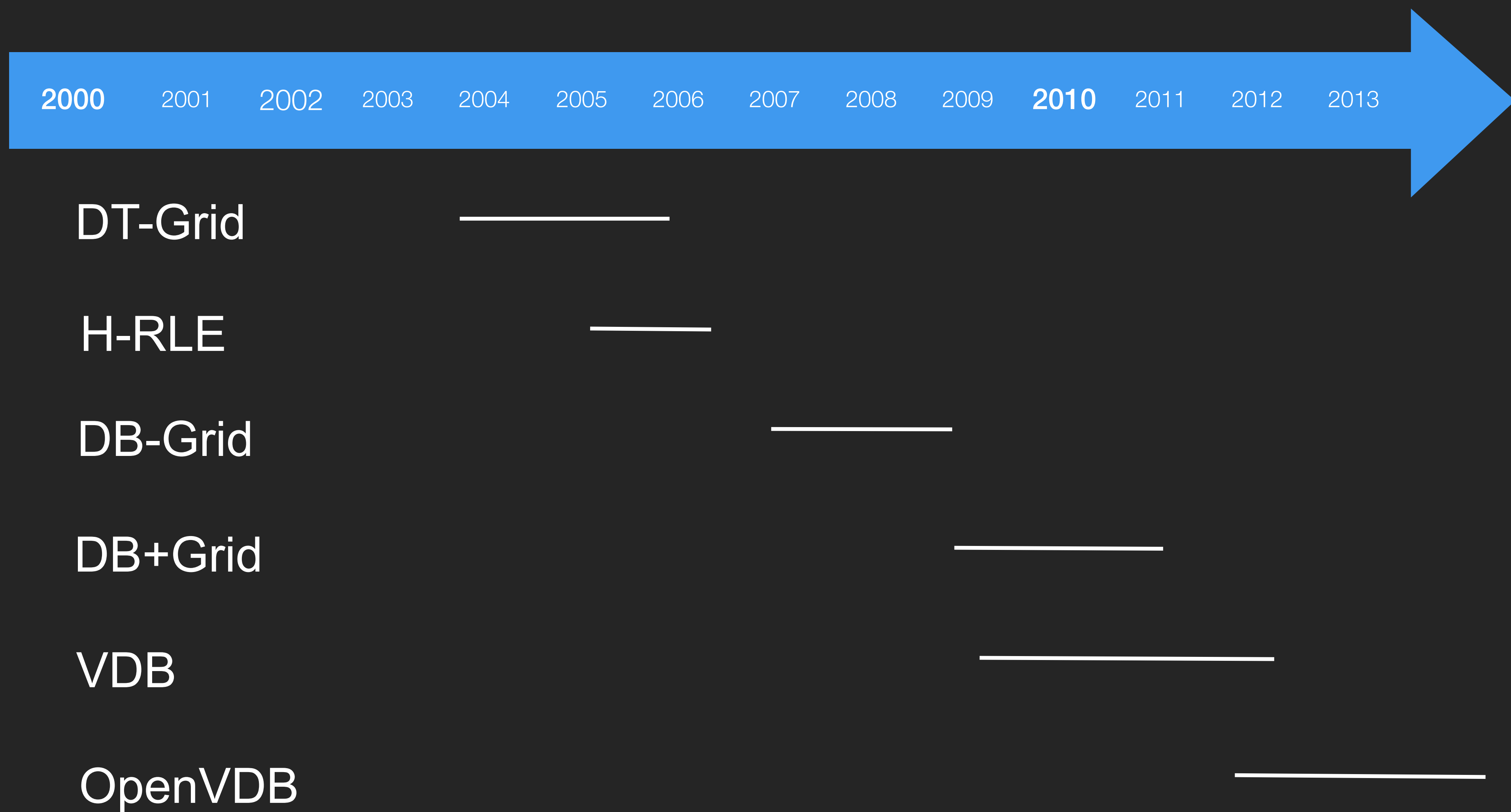
Schedule

- Part 1
  - **2:00pm** Introduction to OpenVDB, Ken Museth (DWA)
  - **2:45pm** Overview of toolset, Mihai Alden (DWA)
  - **3:15pm** Break
- Part 2
  - **3:30pm** Adoption at DreamWorks Animation, Jeff Budsberg (DWA)
  - **4:00pm** Adoption at Digital Domain, John Johansson (DD)
  - **4:30pm** Adoption in Houdini, Edward Lam (SideFX Software)
  - **5:00pm** Concluding remarks and questions, Ken Museth (DWA)

Course Material

- Main site: http://www.openvdb.org
  - Course slides and hip files
  - Coding cookbook
  - FAQ
  - Google group: “OpenVDB Forum”
- Technical paper on VDB (http://ken.museth.org)

History of VDB



Motivating VDB

- Consider extremes
  - DT-Grid / H-RLE (level sets only)
  - DB-Grid (doesn't scale and non-hierarchical)
  - Octrees (slow tree traversal)
- Dichotomy
  - Static versus dynamic data structure

Facts

- DWA feature films using VDB
  - *Puss in Boots*, *Rise of the Guardians*, *The Croods*, *Turbo*
- OpenVDB developers
  - K. Museth, P. Cucka, M. Aldén and D. Hill
- Availability
  - http://www.openvdb.org
  - https://github.com/dreamworksanimation/openvdb_dev
- License
  - Mozilla Public License version 2 and CLA

Library Versioning

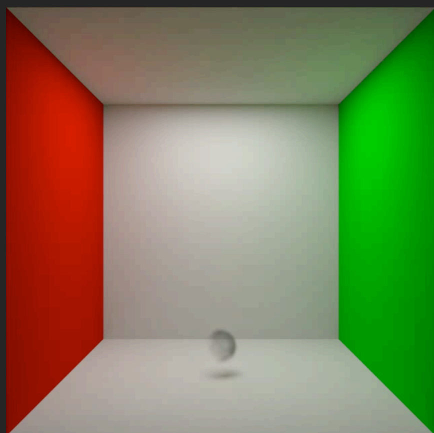
major.minor.patch

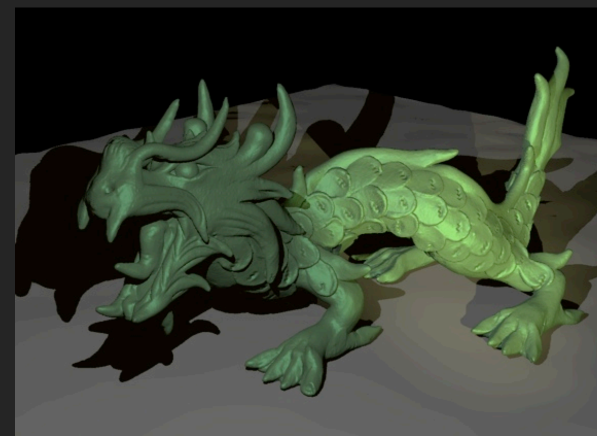
- Patch:
  - No change to API, file format or ABI of Grid or its member classes
- Minor:
  - Change to API but not Grid ABI; backward-compatible file format
- Major:
  - Change to ABI of Grid or non-backward-compatible file format
- No release guarantees complete ABI compatibility!
  - Library is namespaced on the complete version number

SideEffects Houdini

Scott Keating & Jeff Wagner, SideFX

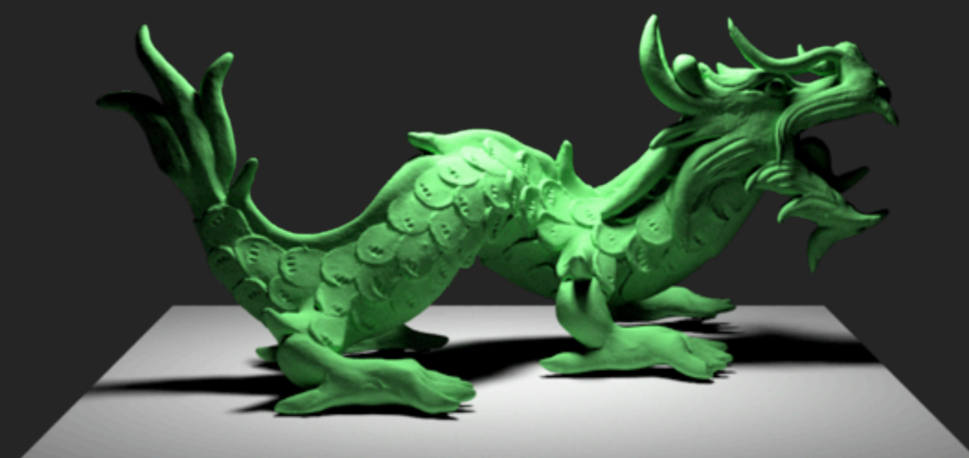
Commercial Renderers





SideFX Mantra

arnold
SOLIDANGLE

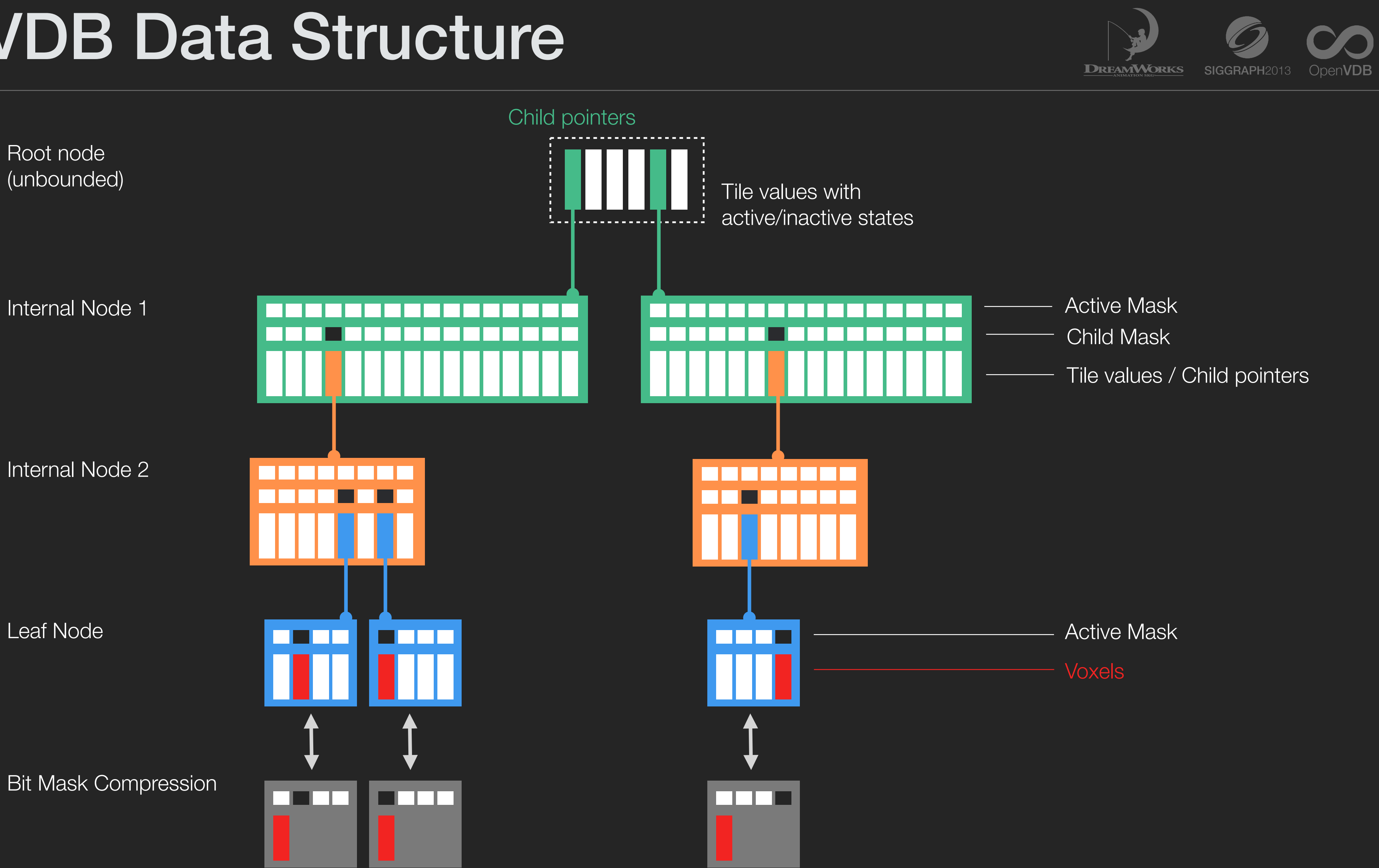


PIXAR's RenderMan

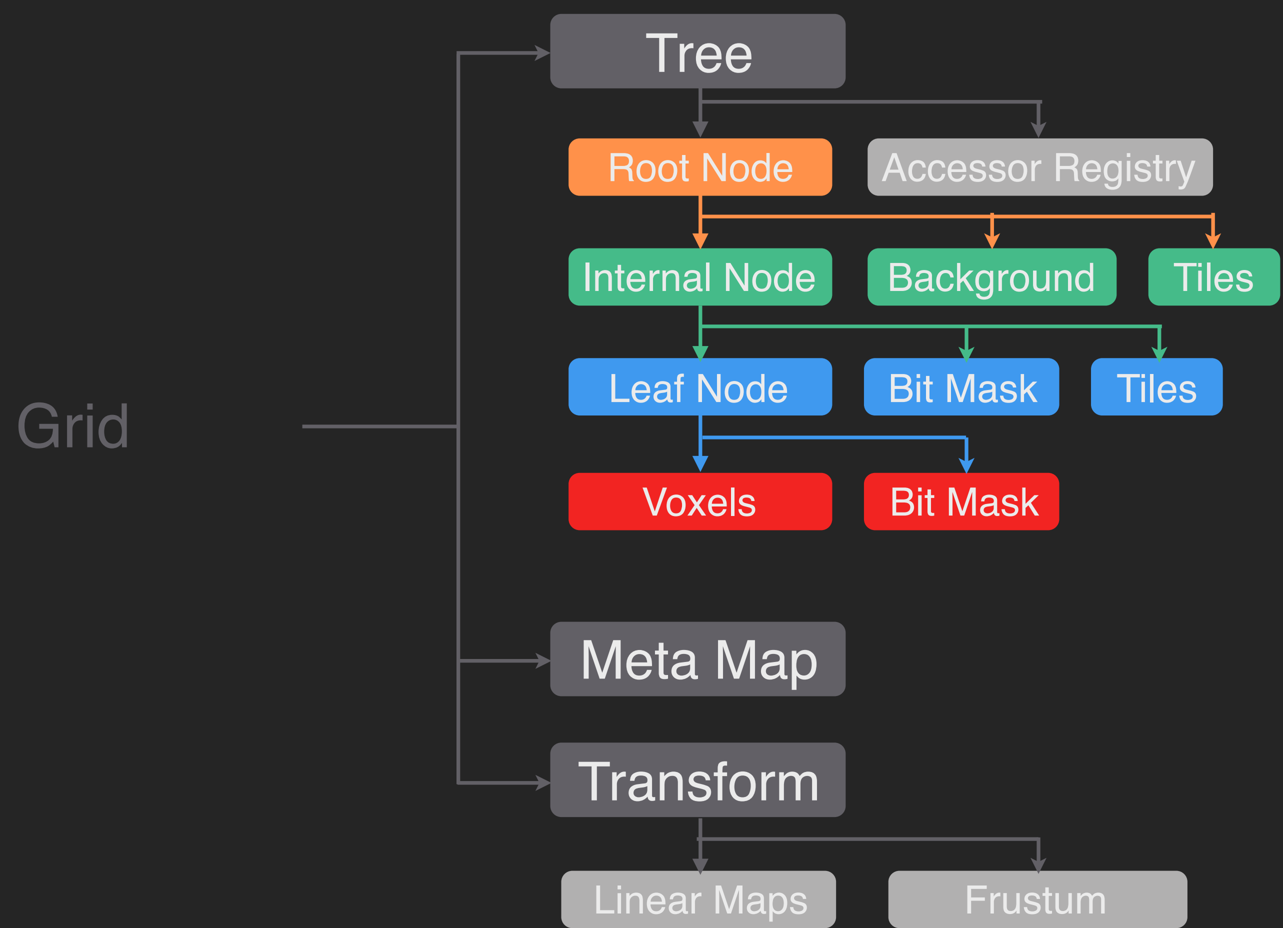
Terminology

- Voxel
  - Smallest addressable unit of index space
  - Resides at the leaf node level
- Tile
  - Larger constant domain of index space
  - Resides at the upper (non-leaf) tree levels
- Active state
  - All values (voxels and tiles) have a binary state
  - Interpretation of state is application-defined

VDB Data Structure



Grid/Tree Class Structure



Sparsity and Efficiency

- Memory efficiency
  - Sparse, hierarchical tree structure
  - Allocation on insert
  - Custom compression schemes on I/O
- Computational efficiency
  - Hierarchical algorithms
  - Sparse computations
  - C++ template metaprogramming
  - Bit tricks, multithreading and SIMD

Grid Types

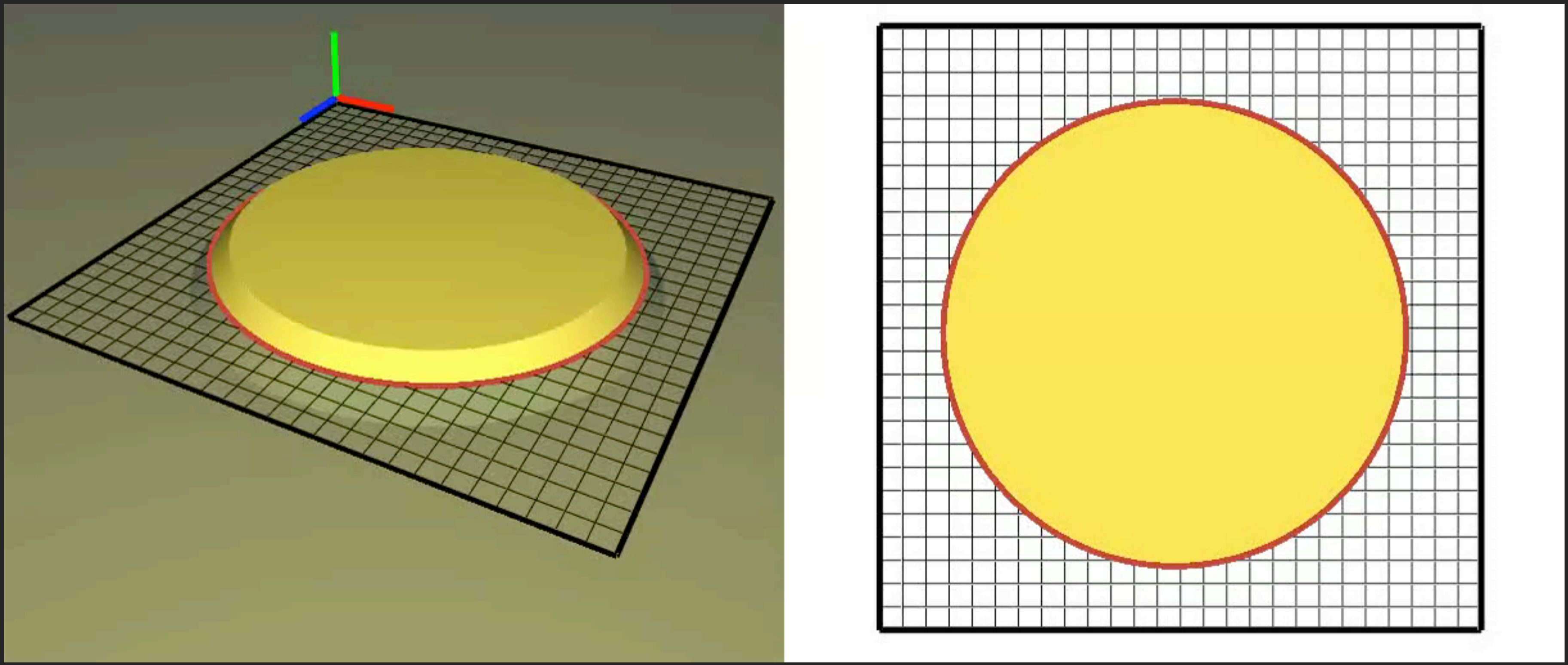
- Fully compile-time configurable
  - Value type
  - Node size, Tree depth, e.g. tile-grid, Octree, N-tree etc.

```
typedef LeafNode<float,3>  N0;
typedef InternalNode<N0,4> N1;
typedef InternalNode<N1,5> N2;
typedef RootNode<N2> RootType;
typedef Tree<RootType> TreeType; // FloatTree
typedef Grid<TreeType> GridType; // FloatGrid
```

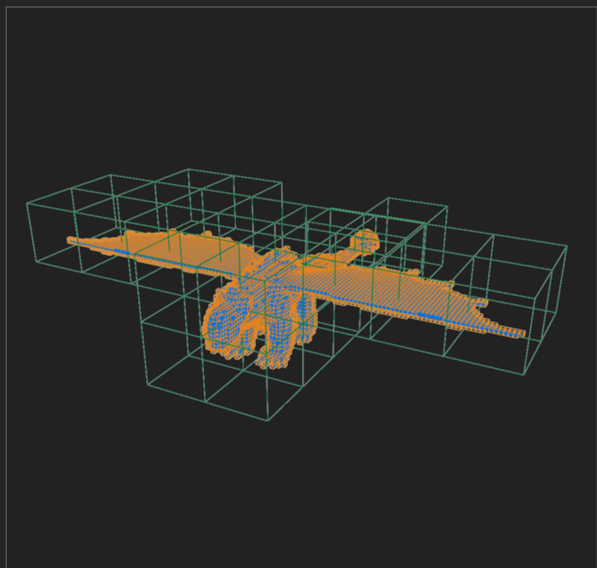
Common Scalar Grid Categories

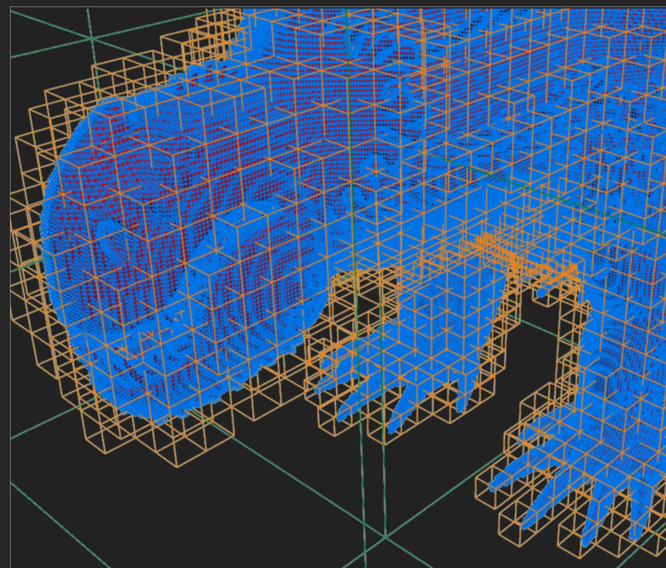
- Fog or density volumes
  - Scalar normalized density [0–1]
  - Background zero
  - Interior voxels have value 1
  - Active tiles
- SDFs or narrow band level sets
  - Scalar truncated signed distance $]-\beta/2, \beta/2[$
  - Background is the positive half-width of the narrow band $(\beta/2)$
  - Interior is the negative half-width of the narrow band $(-\beta/2)$
  - No active tiles

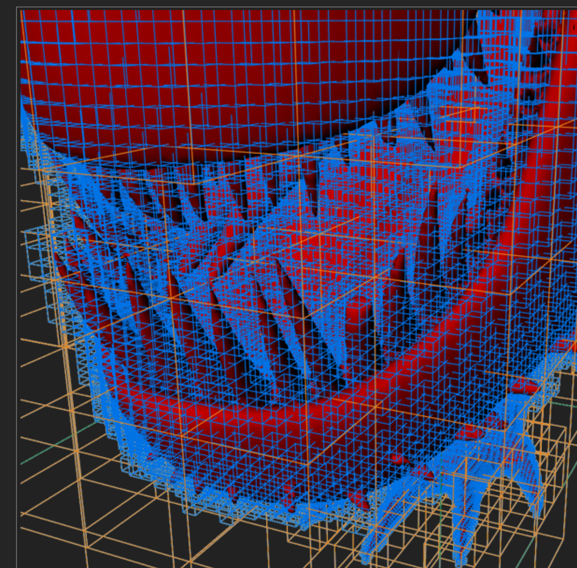
Fundamentals of Level Sets



Narrow Band Level Sets







InternalNodes: 1024^3

InternalNodes: 64^3

LeafNodes: 4^3

Voxels: 1^3

Active voxel span: 7897 x 1504 x 5774

228 million sparse voxel
vs
69 billion dense voxels

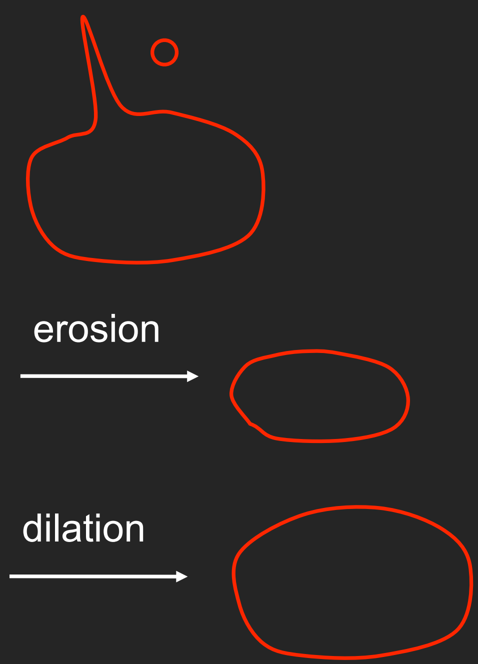
>1GB memory footprint
vs
1/4 TB dense grid

Properties of Level Sets

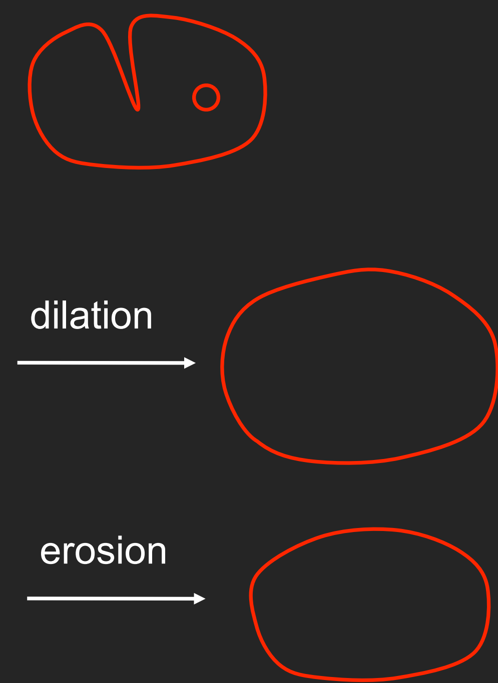
- Closest signed distance $\phi(\vec{x}) \qquad |\nabla\phi(\vec{x})| = 1$
- Interface normals $$\vec{n} = \frac{\nabla\phi}{|\nabla\phi|} = \nabla\phi$$
- Closest Distance Transform $$\mathrm{CPT}(\vec{x}) = \vec{x} - \phi(\vec{x})\nabla\phi(\vec{x})$$
- Mean curvature $$K = \tfrac{1}{2}\nabla \bullet \frac{\nabla\phi}{|\nabla\phi|}$$
- Surface deformations $$\frac{\partial\phi}{\partial t} = \begin{cases} \vec{V} \bullet \vec{\nabla}\phi \\ \Gamma|\vec{\nabla}\phi| \end{cases}$$

Morphological Operations

Opening



Closing



math::Transform

- Transforms Index Space (coordinates) ⟷ World Space (locations)
  - `Vec3d Transform::indexToWorld(const Vec3d& xyz) const;`
  - `Vec3d Transform::indexToWorld(const Coord& ijk) const;`
  - `Vec3d Transform::worldToIndex(const Vec3d& xyz) const;`
- Supports
  - Affine Transforms (mix of scaling, rotation, shearing, translation)
  - Frustum Transforms
  - Updates with *pre* and *post* semantics
- Wraps a minimal representation “`BaseMap`”
  - Virtual hit in calling `BaseMap`

Maps: Reduced Representation

- Distinct implementations for common transforms
  - MapType: Scale, Translation, Frustum, etc.
  - Minimal number of floating point operations
- For higher performance, template on MapType
  - No virtual hits
  - `math::processTypedMap`
- Used in vector calculus operators (Gradient, Curl…) in world space
  - Computes first and second derivatives of the transform
  - Chain Rule-based logic implemented for various operators

Data Access

- Sequential access
  - Fundamental to simulations
  - Access elements according to memory layout
- Random access
  - Coordinate-based access: `getValue(x,y,z)`
  - Spatial coherence
- Stencil access
  - Fundamental to finite differencing, filtering and interpolation
  - Often combined with sequential or random access

Iterators

- Bit mask
  - Located in all nodes
  - Facilitates efficient iterators

```
Grid/Tree/NodeType::ValueOnIter
Grid/Tree/NodeType::ValueOffIter
Grid/Tree/NodeType::ValueAllIter
NodeType = {RootNode,InternalNode,LeafNode}

Tree::LeafIter     Tree::NodeIter
```

`tree::LeafManager`

- Fundamental assumptions
  - Voxel processing only, so safe to ignore tiles
  - Static topology
- Linearize tree as an array of leaf nodes
  - Lightweight
  - Very fast iteration and trivial split operator for multithreading
- Temporal value buffers for each leaf node
  - Solve time-dependent PDEs, filtering and convolution
  - Multithreaded swapping of buffers

tree::ValueAccessor

- Random or coordinate-based access
  - Should never be truly random!
  - Virtually always spatially coherent
- Improved random access
  - Cache nodes
  - Inverted tree traversal
  - On average shorter path
  - Amortize overhead of slow dynamic RootNode

Stencil Operations

- Finite differencing
  - Gradient, curvature, Laplacian, curl, etc.
  - Central-difference schemes (2nd–6th order)
  - Upwind schemes (1st–5th order)
- Interpolation
  - Nearest neighbor: `tools::PointSampler`
  - Trilinear: `tools::BoxSampler`
  - Triquadratic: `tools::QuadraticSampler`
  - Staggered variants for MAC grids

Optimization

- Use `ValueAccessor` for random access
- Use sparse iterators for sequential access
- Use `LeafManager` when applicable (i.e., no tiles and static topology)
- Disable asserts (`-DNDEBUG`)
- Use library tools and routines (don't reinvent the wheel!)
- If possible, partition algorithm into topology and value updates
  - Use topology copy constructors
  - Use `setValueOnly`
- Use multithreading

Thread Safety

- Non-thread-safe operations
  - Insert new values (due to allocate-on-insert)
  - Change state of a value (due to bit masks)
  - Read or write via `ValueAccessor` (due to internal node caching)
- Thread-safe operations
  - Random or sequential read of constant values and states
  - Modify values in *existing* nodes

Threading Strategies

- Thread over values or nodes using sparse iterators
- Consider using `tree::LeafManager` if applicable
- Assign one `tree::ValueAccessor` to each thread
- Reuse `ValueAccessor`s as much as possible
  - Never allocate a new `ValueAccessor` per access operation
  - In extreme cases use thread-safe access methods on `Tree`

Threading Strategies

- If the topology of the output grid is known
  - Pre-allocate output grid with correct topology (use `topologyCopy`)
  - Use `tools::foreach` or `tbb::parallel_for` over output grid
  - Use `setValueOnly`
- If the topology of the output grid is unknown
  - Assign an empty grid to each computational thread
  - Use `tools::transformValues` or `tbb::parallel_reduce`
  - Use hierarchical merge or compositing in thread join
  - Use concurrent malloc, e.g., `tbb_malloc` or `jemalloc`

End

Questions?