

Data Structures for Sparse Volumetric Data

J.Andreas Bærentzen, DTU Informatics

Random Facts about Volumetric Data Structures

J.Andreas Bærentzen, DTU Informatics

Selected Tools for Representing Sparse Volume Data in Given Scenarios

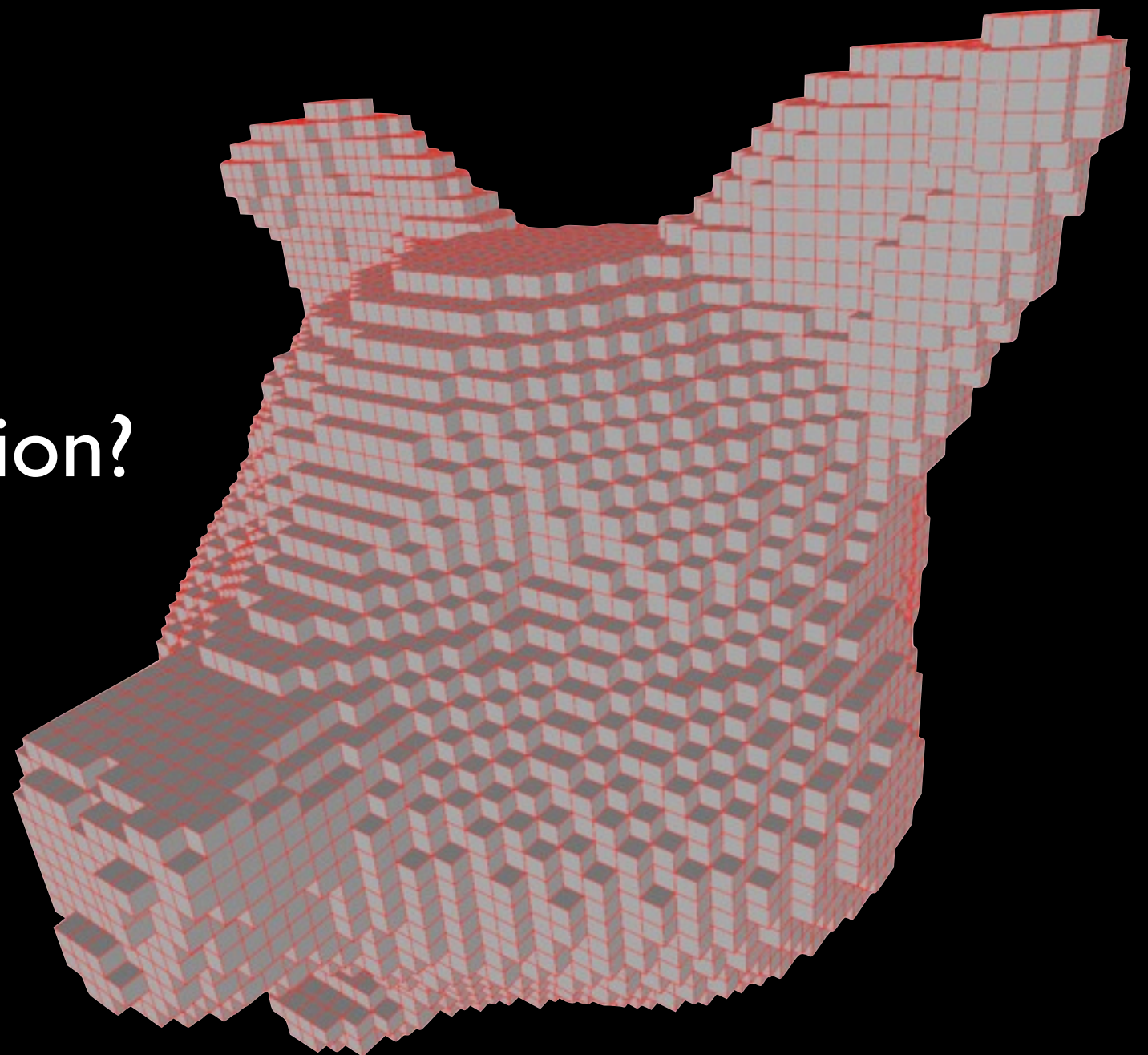
J.Andreas Bærentzen, DTU Informatics

Overview

- Visualization: Real-time or interactive rendering of (medical) volume data
- Distance fields in conjunction with the level set method
- Very high resolution and adaptive distance fields
- Beyond distance fields: Deformable Simplicial Complexes

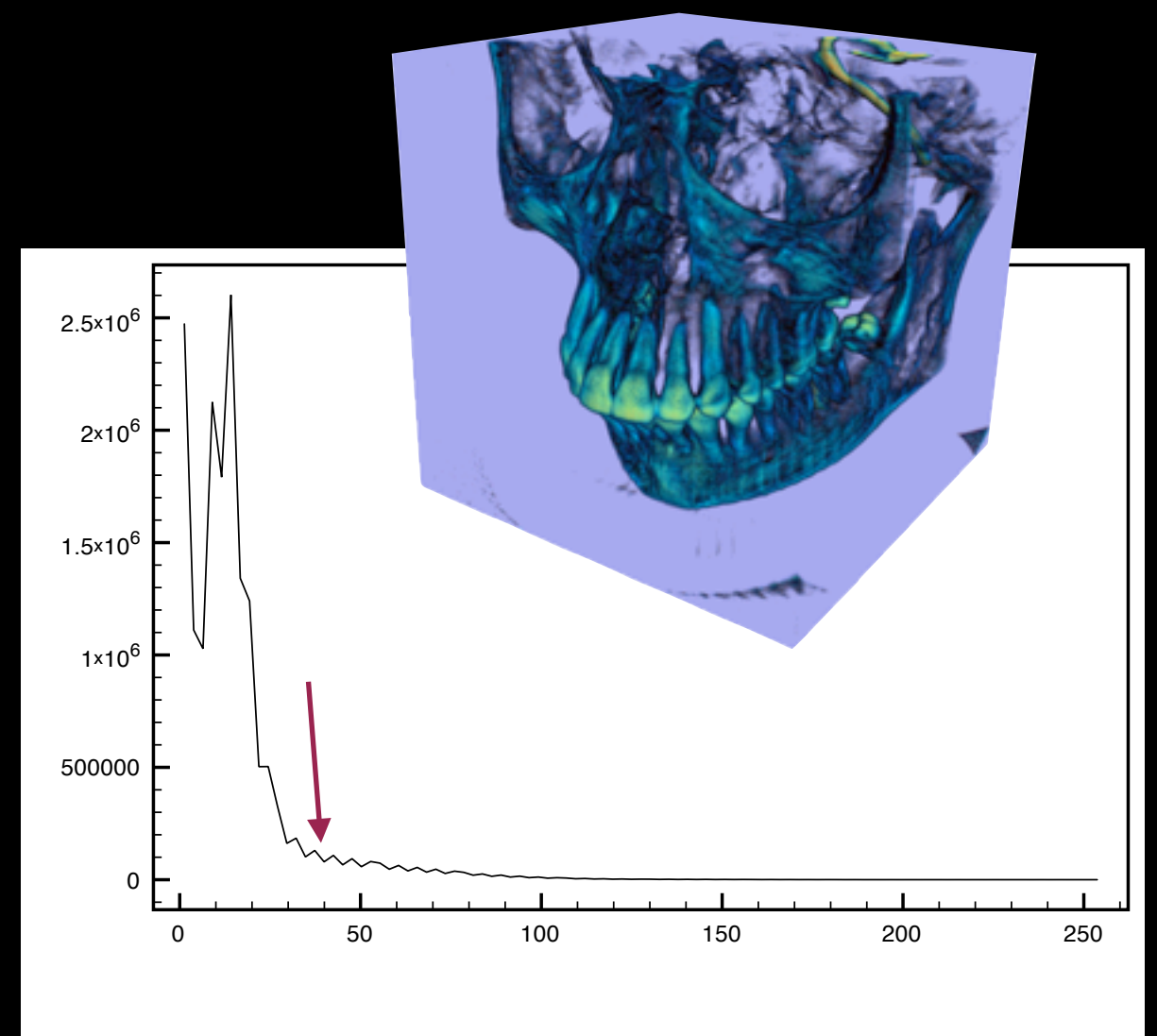
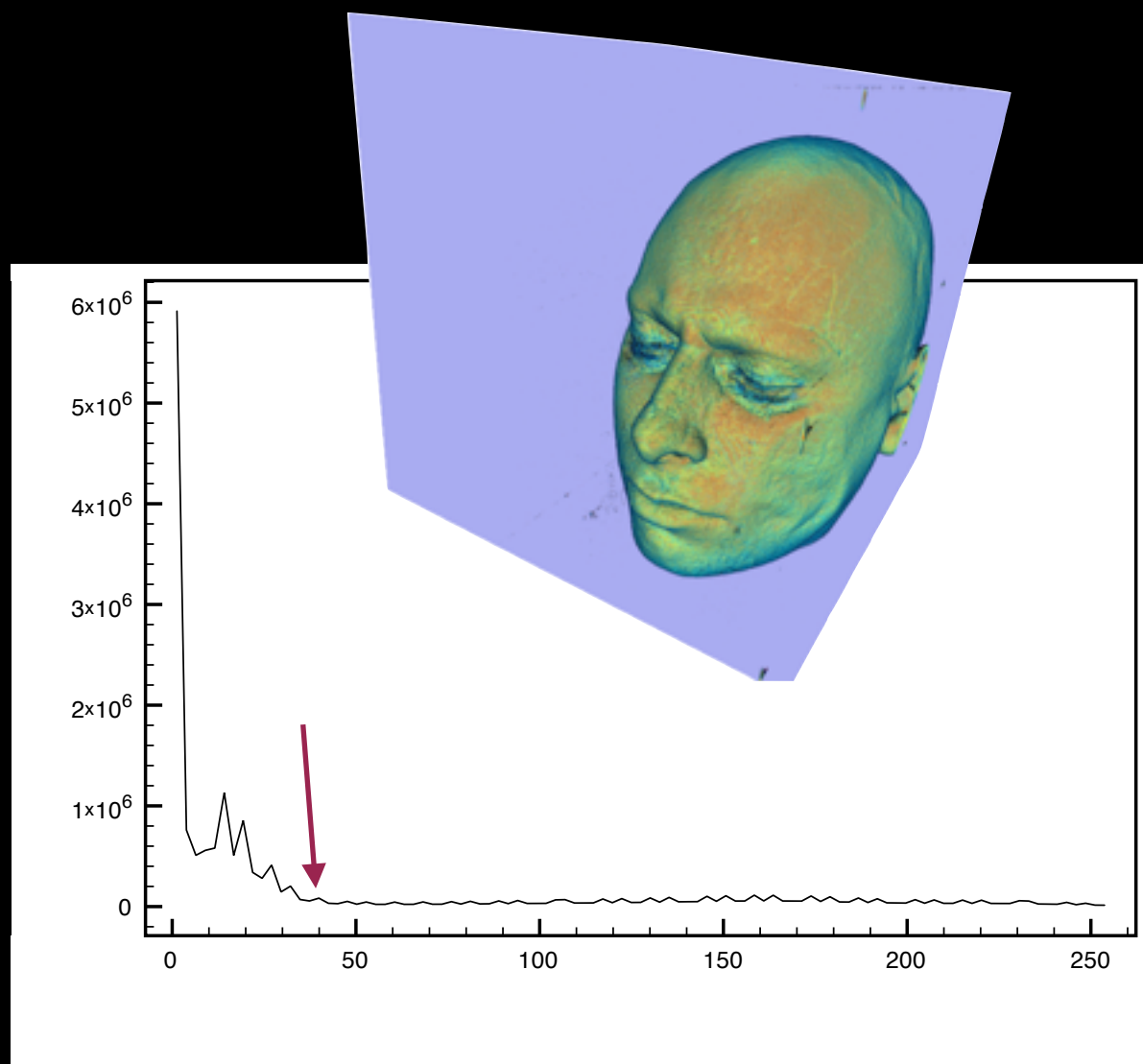
What's in a voxel?

- Volumetric data: Tabulated $f: R^3 \rightarrow R$
- What is a voxel?
 1. A grid point with an associated value?
 2. A cuboid spatial region?
 3. Both of the above!
- Volumetric data is usually very sparse



How Sparse?

- Sparse means that few voxels are “of interest”
- Red arrow indicates visibility threshold

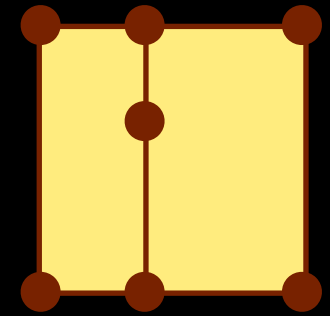


Dense Voxel Grids

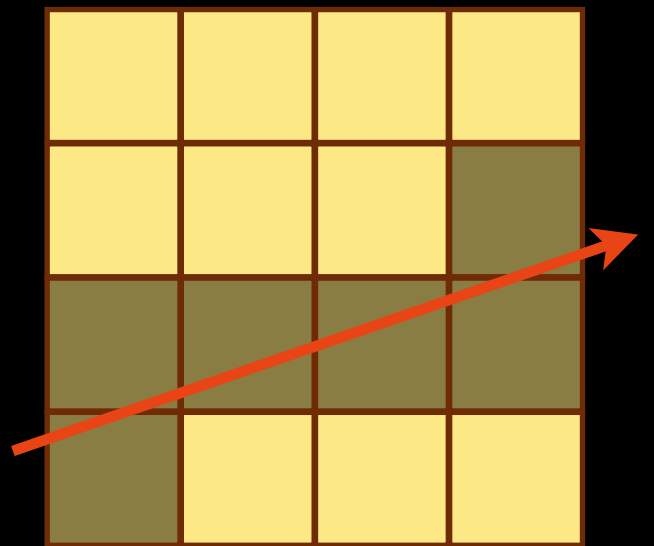
- Computer memory is linear ...
- Assume a volume
 - of dimensions: x_{dim} , y_{dim} , z_{dim}
 - stored in a linear array `data`
- Voxel at grid pos $[x,y,z]$ is stored in `data` at position:
 - `data[z*xdim*ydim+y*xdim+x]`

Coherence

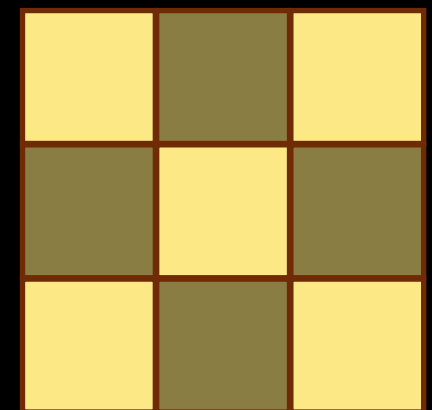
- We usually need to access voxels in a spatially coherent fashion
 - Interpolation (trilinear)
 - Walking along rays
 - Computing gradients, curvature, etc.
- Sometimes, we just need to visit all voxels:
E.g. when computing a histogram.
- Coherence is important on both CPU and GPU



Bilinear Interpolation



Voxels pierced by ray



Typical gradient stencil

Coherence:

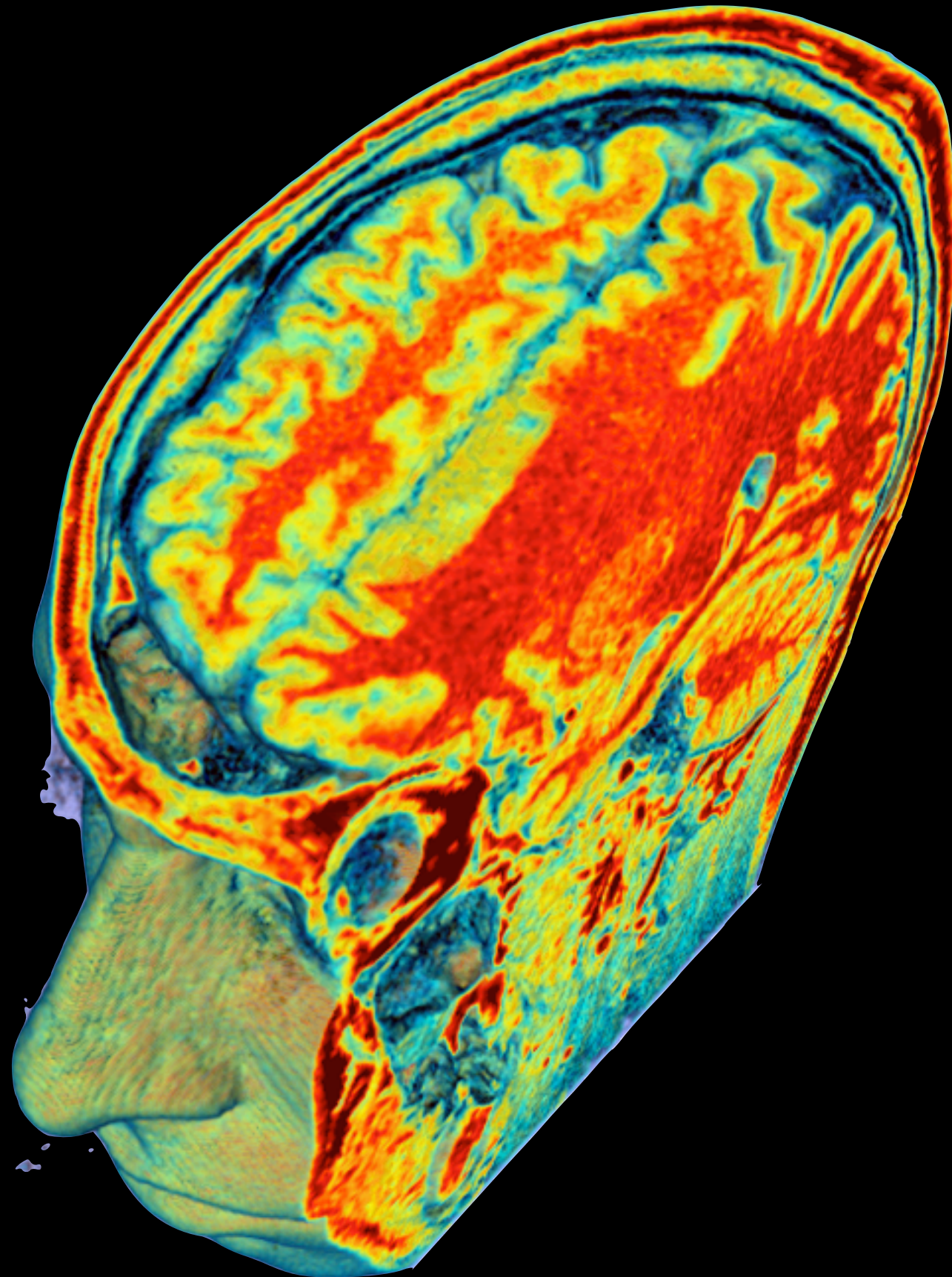
Importance on CPU

- An experiment: Compute mean voxel value of 256^3 volume by traversing the grid in a triple for loop
 - Order ZYX takes 0.22 seconds
 - Order XYZ takes 1.02 seconds
- So matching the linear order matters!
- This is good to bear in mind when thinking about data structures for volumes...

Visualization

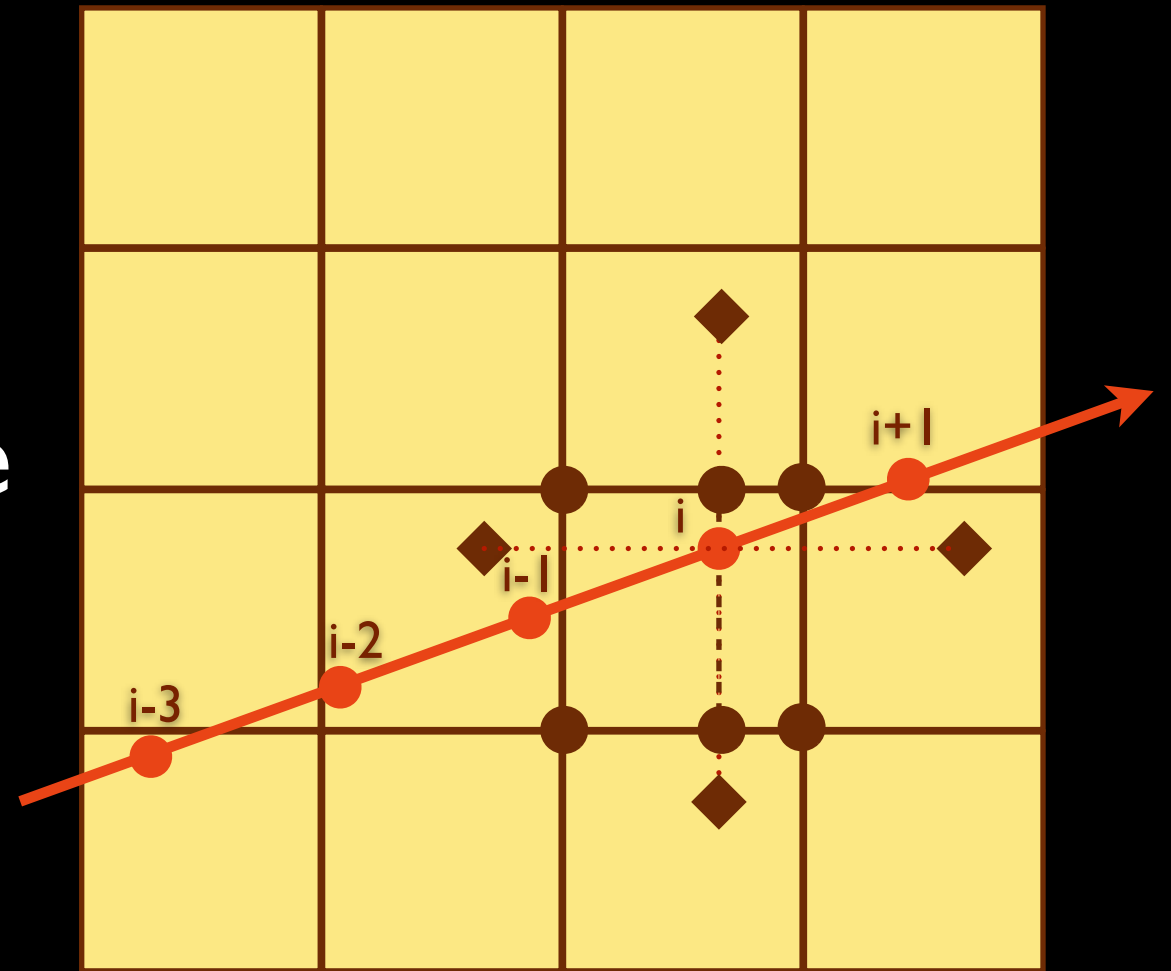
- We consider GPU based volume rendering
- Volume stored in GPU memory

MRI data set of Lars Pedersen rendered using VoxelRay - a tool from DTU Informatics



Volume Rendering

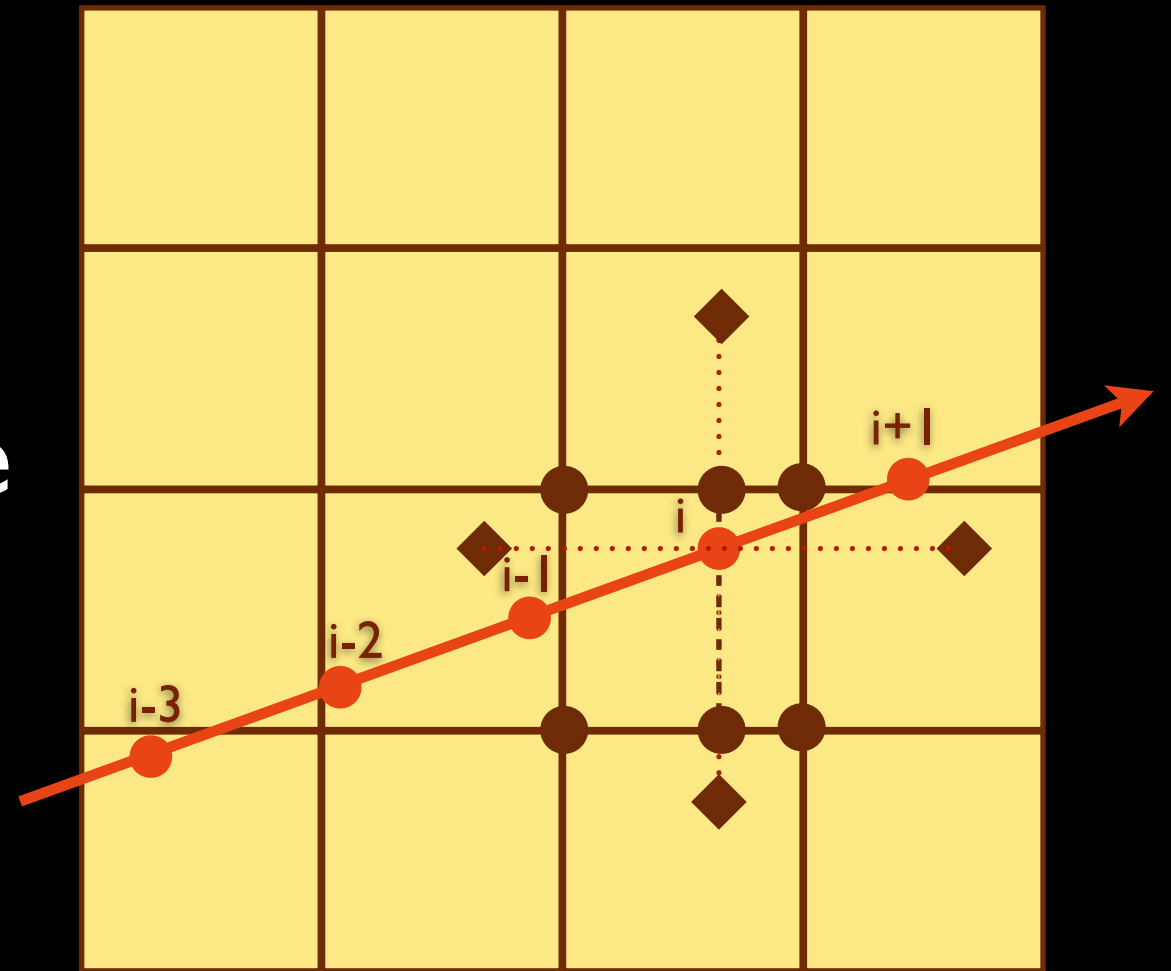
- Ray Casting: For each pixel walk along a ray through the volume and integrate



$$I \leftarrow I + T * col_i * alpha_i$$
$$T \leftarrow T * (1 - alpha_i)$$

Volume Rendering

- Ray Casting: For each pixel walk along a ray through the volume and integrate

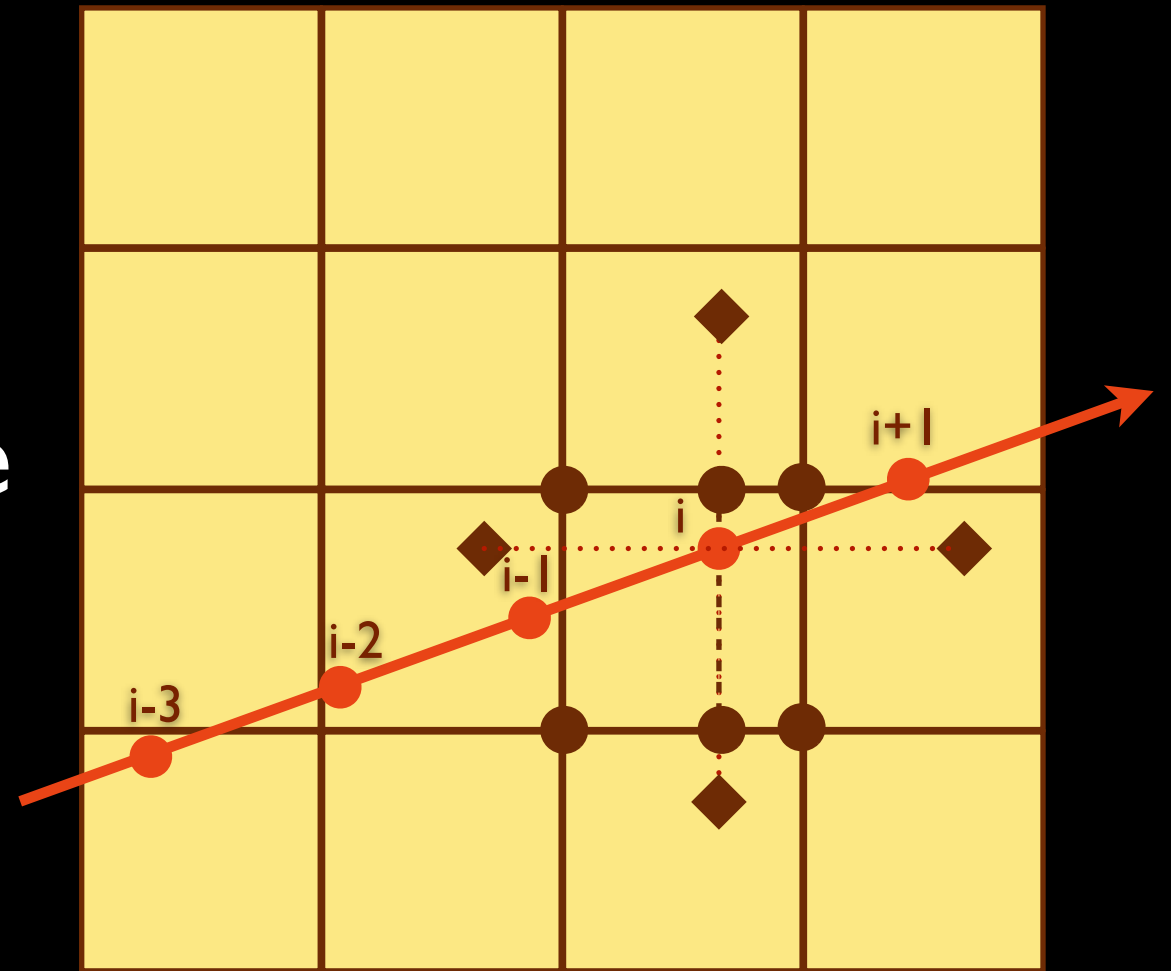


$$I \leftarrow I + T * col_i * alpha_i$$
$$T \leftarrow T * (1 - alpha_i)$$

Alpha at sample i

Volume Rendering

- Ray Casting: For each pixel walk along a ray through the volume and integrate



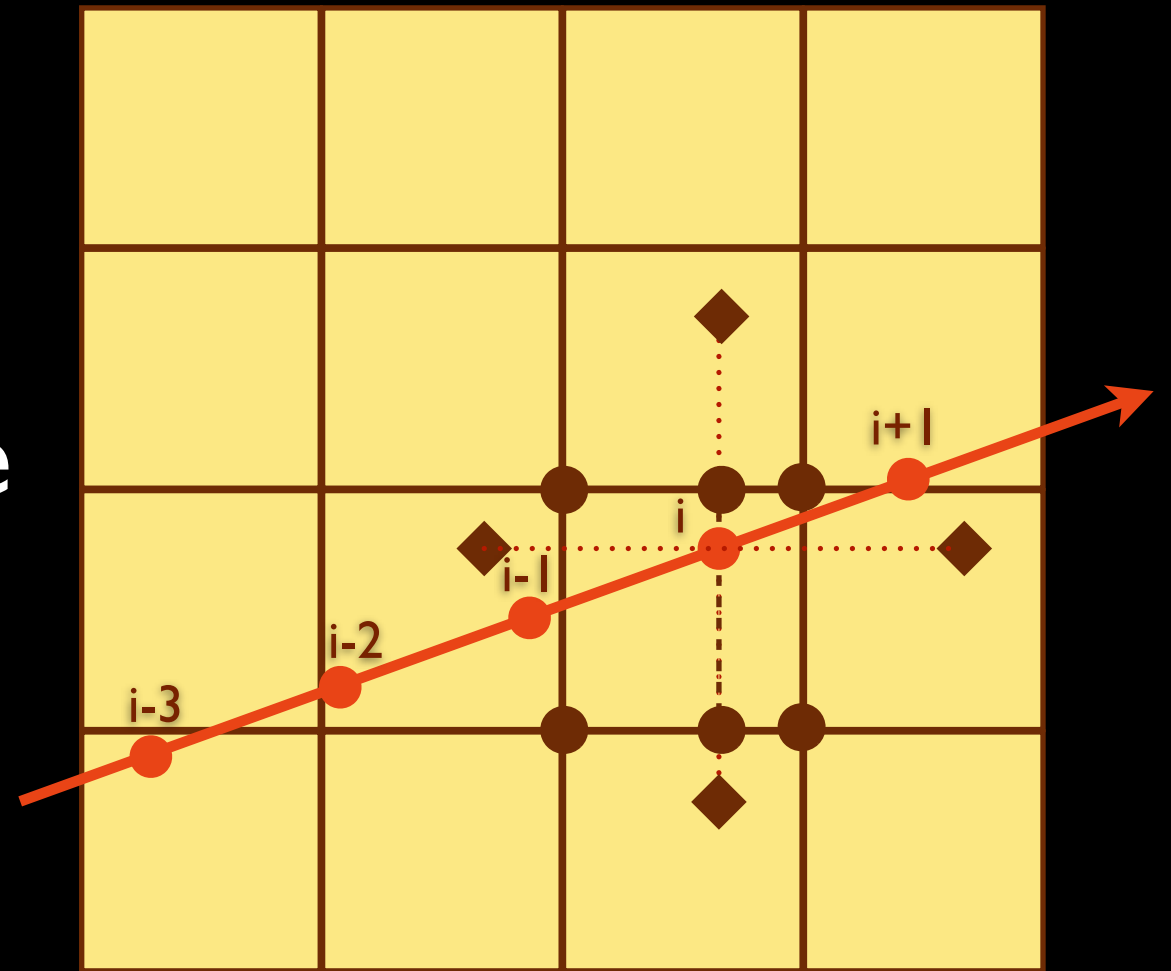
Color
Computed from
Shading at sample
 i

$$I \leftarrow I + T * col_i * alpha_i$$
$$T \leftarrow T * (1 - alpha_i)$$

Alpha at sample i

Volume Rendering

- Ray Casting: For each pixel walk along a ray through the volume and integrate



Color
Computed from
Shading at sample
 i

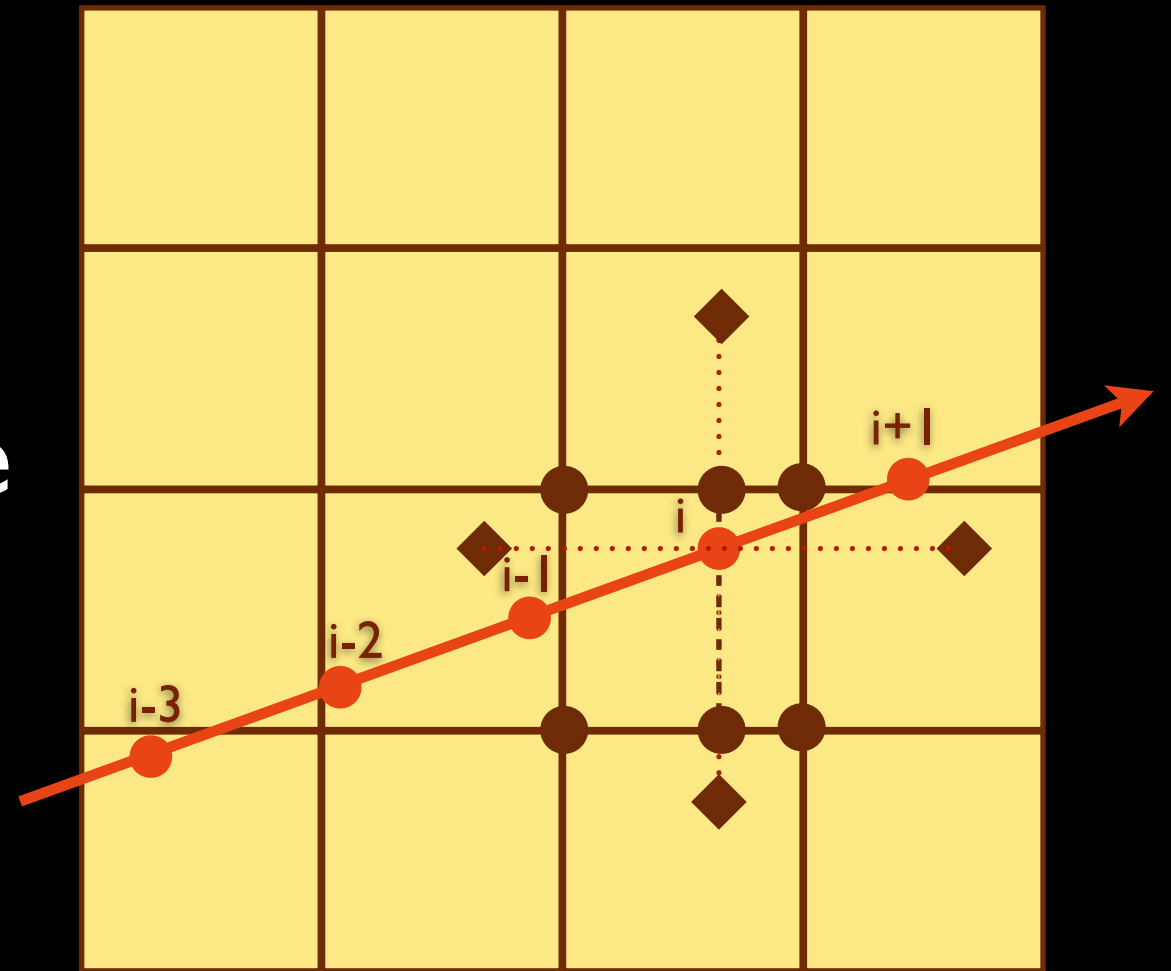
$$I \leftarrow I + T * col_i * alpha_i$$
$$T \leftarrow T * (1 - alpha_i)$$

Accumulated
Transparency

Alpha at sample i

Volume Rendering

- Ray Casting: For each pixel walk along a ray through the volume and integrate



Accumulated
Color

Color
Computed from
Shading at sample
 i

$$I \leftarrow I + T * col_i * alpha_i$$
$$T \leftarrow T * (1 - alpha_i)$$

Accumulated
Transparency

Alpha at sample i

Volume Rendering

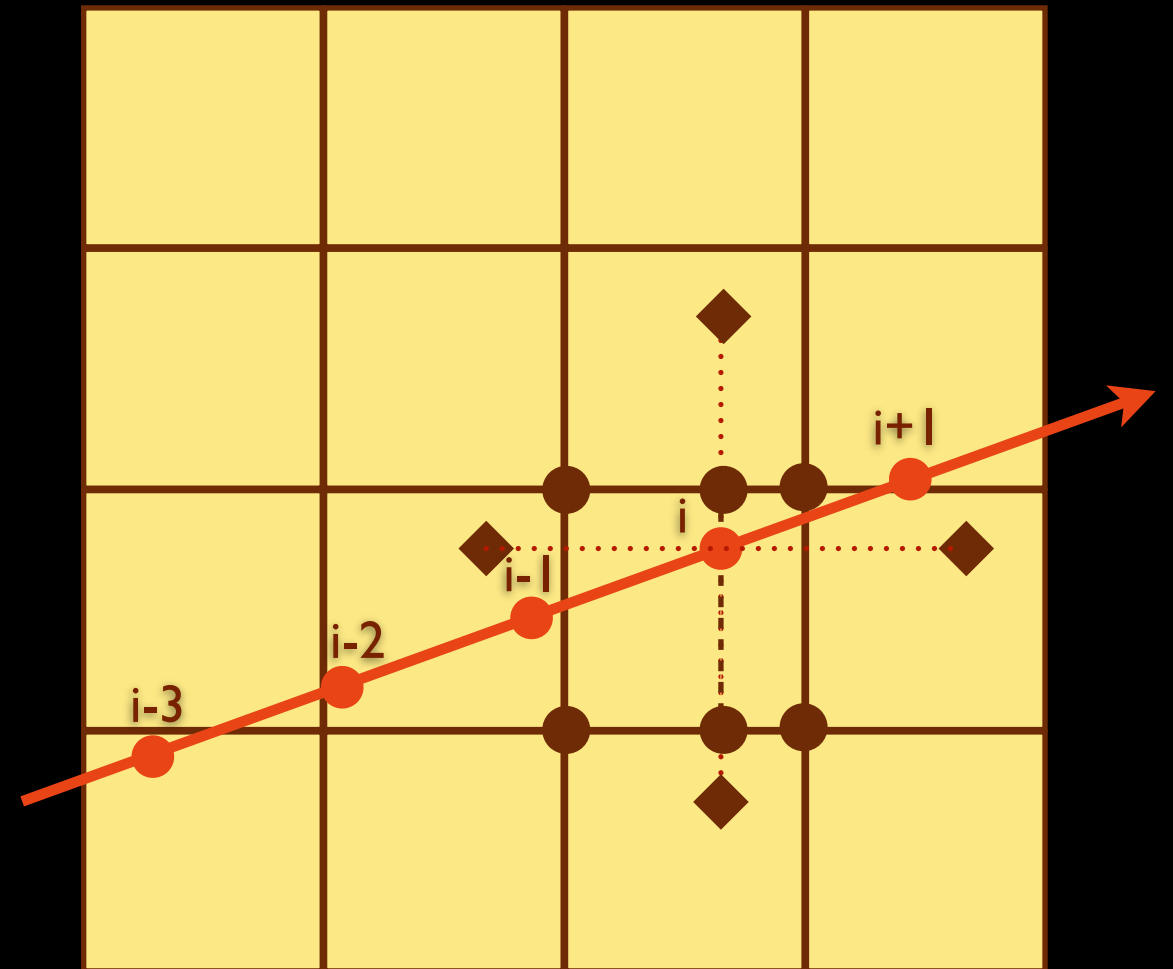
For each pixel:

For $i=1$ to N :

$$I \leftarrow I + T * col_i * alpha_i$$

$$T \leftarrow T * (1 - alpha_i)$$

- We need
 - Seven trilinearly interpolated texture look ups per ray sample.
 - Fast parallel ray traversal



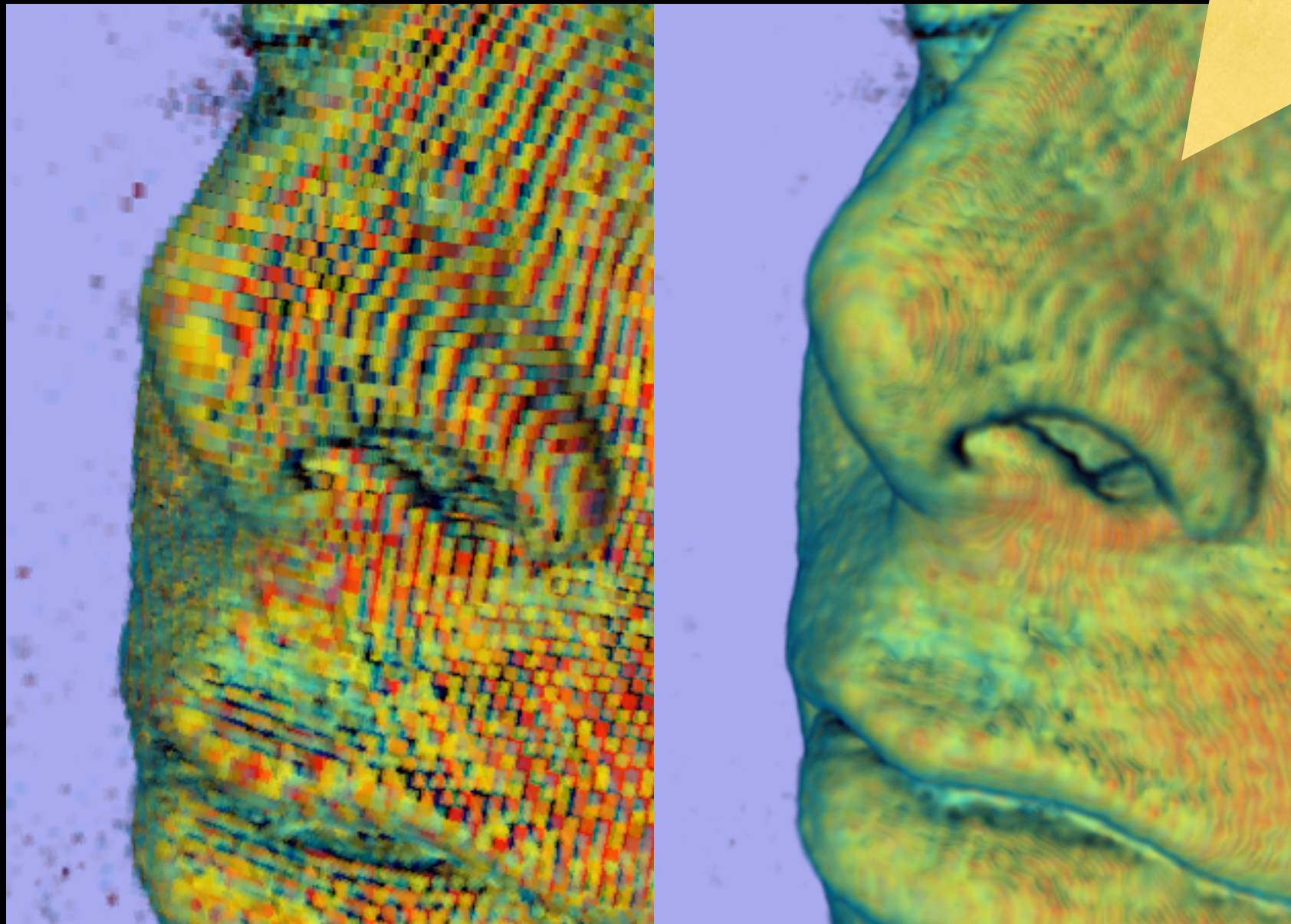
- GPU provides
 - Fast parallel execution of ray traversal
 - Volume as 3D texture
 - Built-in nearly FREE trilinear interpolation

Interpolation

nearest neighbor

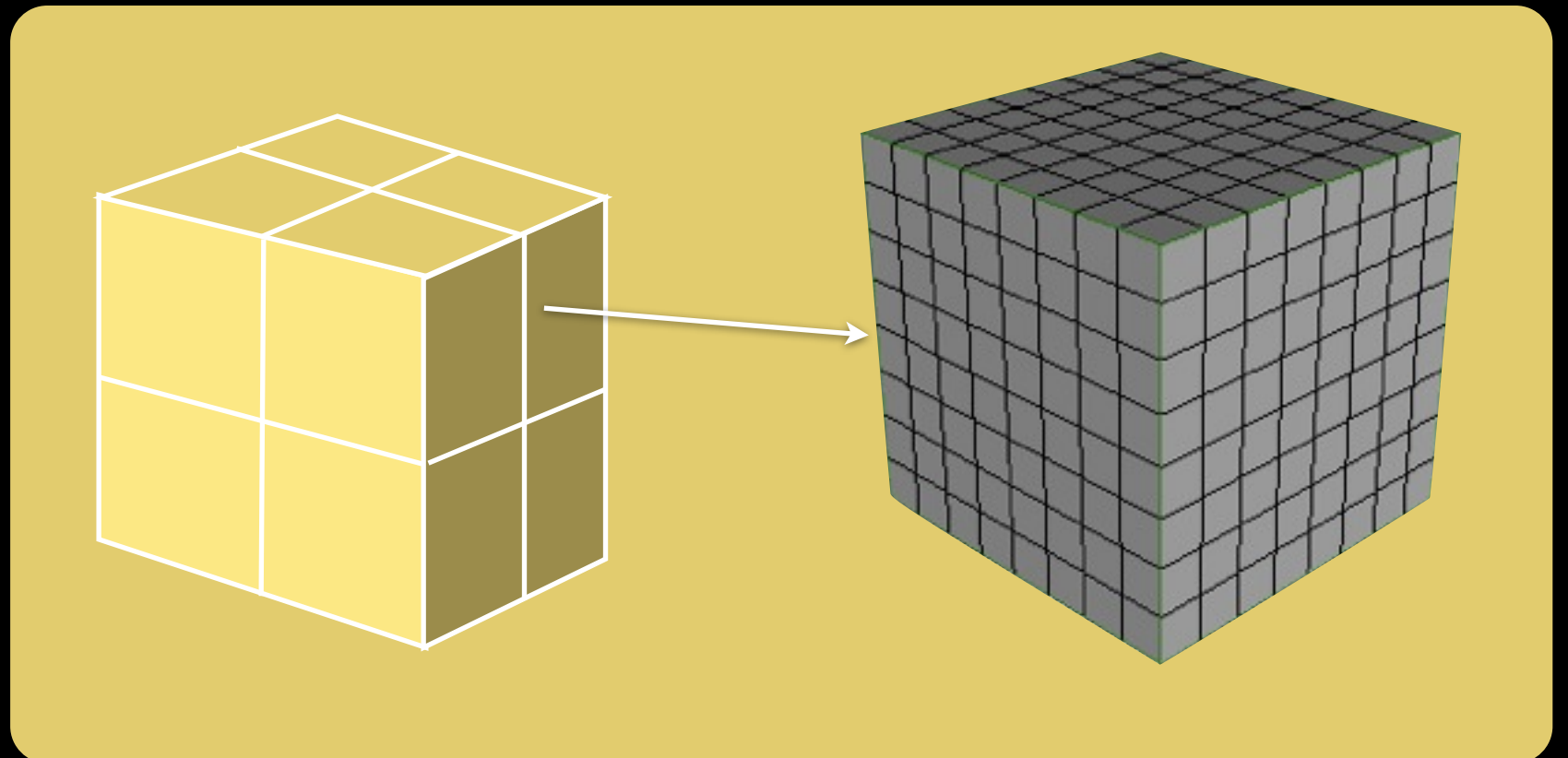
trilinear

FREE!!



No significant difference in rendering performance

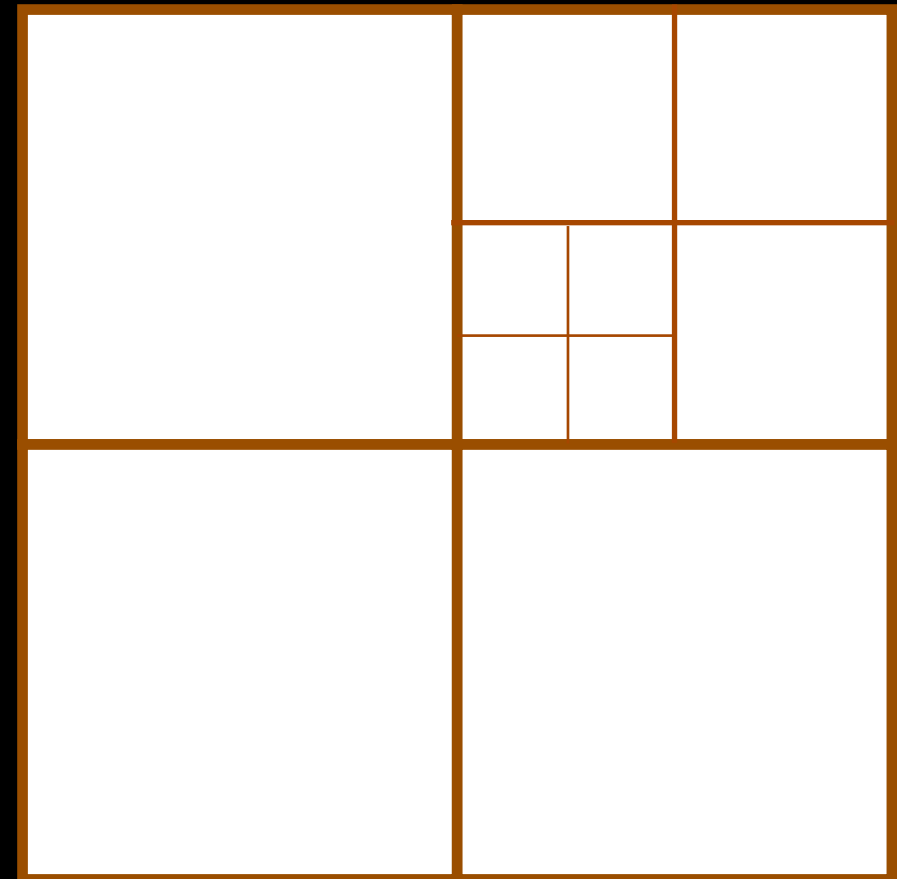
Bricking



- Divide volume into “bricks” - often **just to cope** with amount of data
- Bricks could be slices (full X,Y dimension)
- Alternatively, exploit sparsity:
 - Bricks simply divide volume into smaller pieces but empty bricks may be omitted

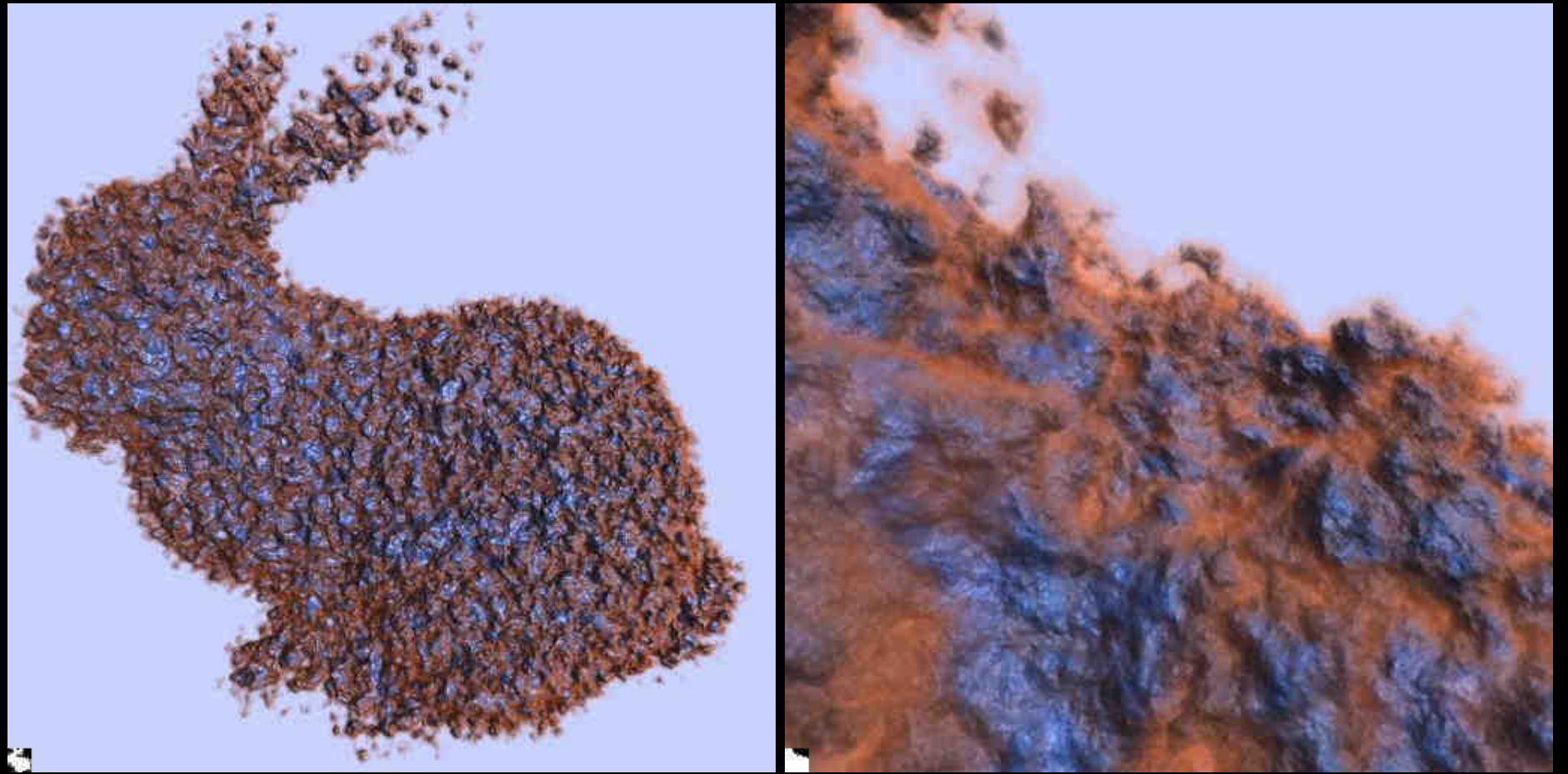
Recursive Decomposition

- Recursively divide space into 2^d cells where d is dimension ($2^3 = 8$)
- Each node contains:
 - Pointer to each child
 - Data associated with each child
- Save by culling empty cells
- Note leaves should be bricks not voxels!



this is a 2^2 aka “quad” tree

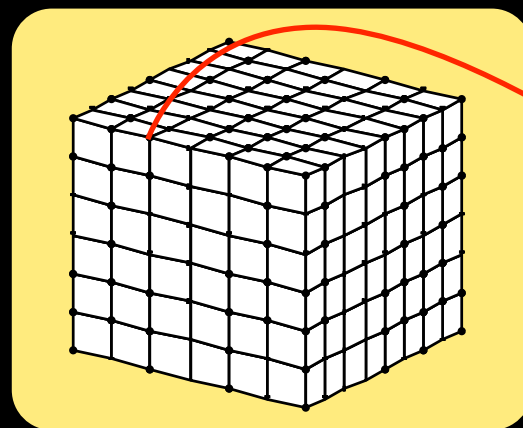
Giga Voxels



- Crassin et al. [2008] proposed Octree of identical 3D Texture blocks (typically 32^3)
- Blocks can be at various levels and also replaced by constant color values.
- Texture blocks support built-in *mipmapped* (multiresolution) interpolation.
- NOT SIMPLE: Blocks loaded onto GPU as needed

Hash Tables

Hashing is a mapping H from a key (e.g. $[x, y, z]$) to an index in a table of size $tsize$



#	key	data
0	(0,2,6)	23
1	(5,3,1)	232
2	(2,4,3)	67

H maps grid positions to array positions:
 $H(0, 2, 6) = 0$, $H(5, 3, 1) = 1$, $H(2, 4, 3) = 2$

In some cases we get collisions, i.e. maybe
 $H(1, 2, 3) = H(2, 4, 5)$

Hash Tables

What would be a good hash function H ?

How about

$$H(x, y, z) = z * xdim * ydim + y * xdim + x$$

No: of course, $tsize \ll xdim * ydim * zdim$.

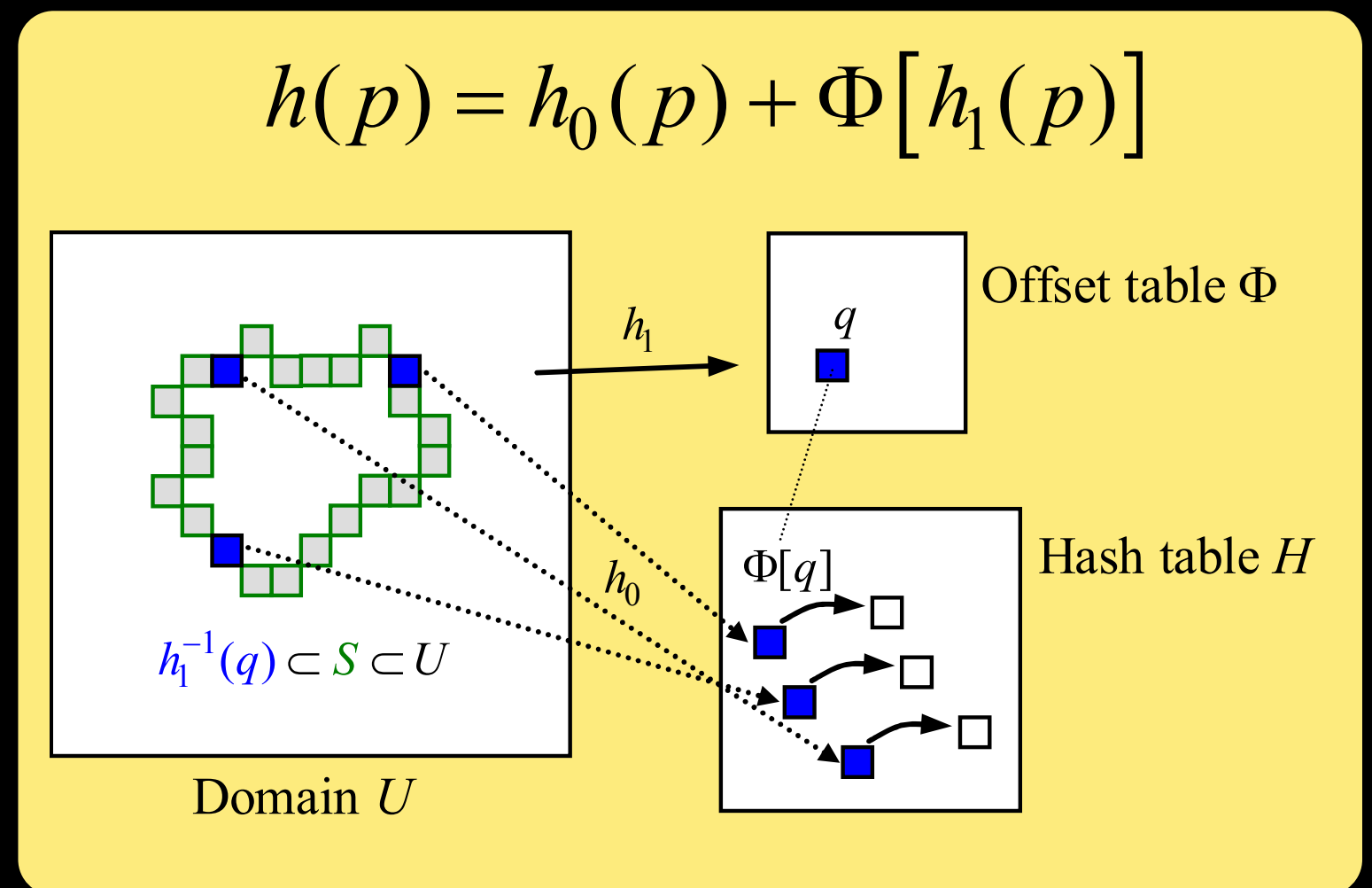
$$H(x, y, z) = (z * xdim * ydim + y * xdim + x) \% tsize$$

No: Values of H should be *uniformly distributed* for actual inputs to avoid collisions!

Collisions are resolved by *rehashing* or *chaining*

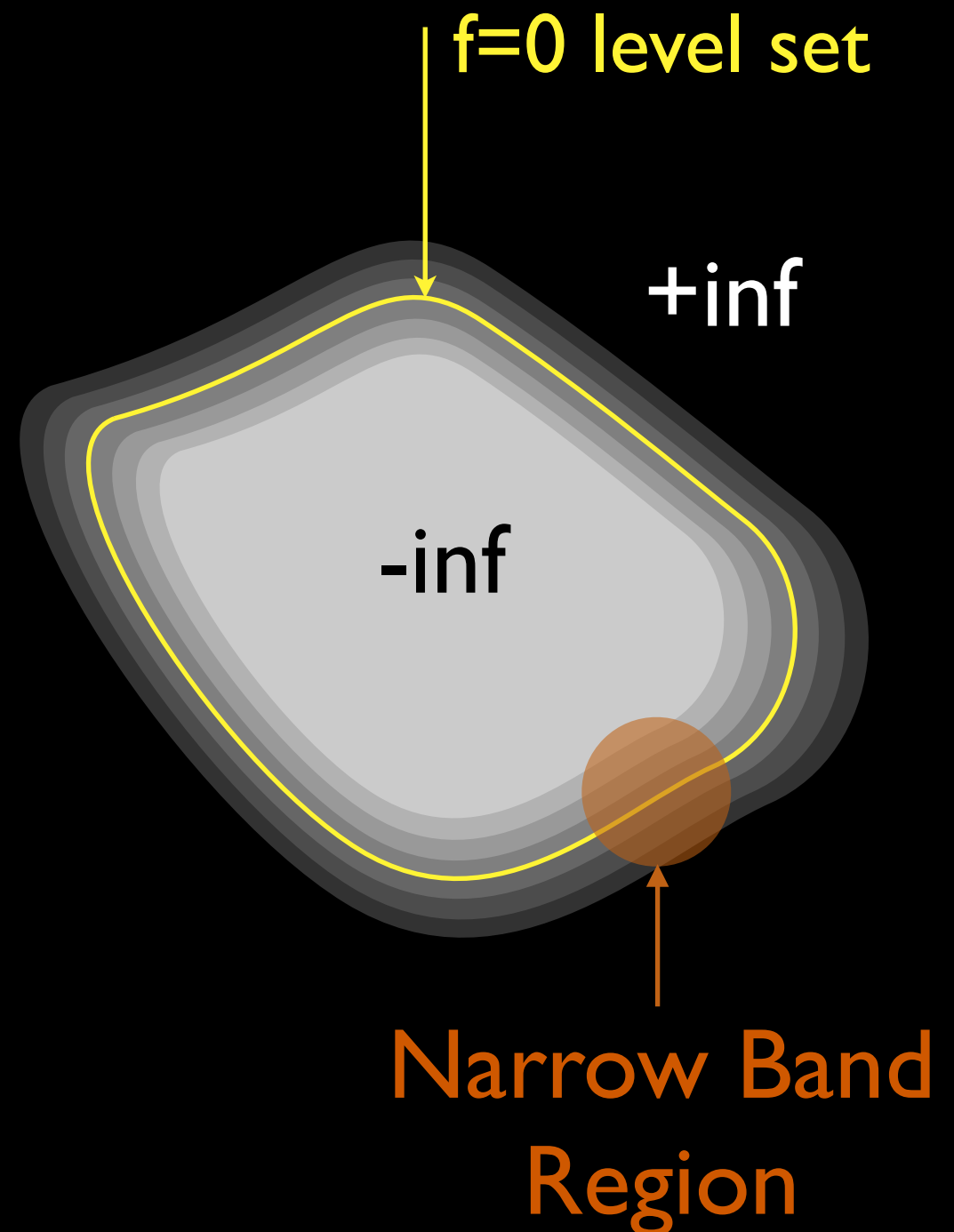
Perfect Hashing

- The need to deal with collisions makes hash tables GPU unfriendly
- Lefebvre et al. [2006] propose a scheme for perfect hashing
- However GPU built-in interpolation is still an issue



Distance Fields

- We can store complex shapes using *distance fields*
- A distance field stores the distance to the closest point on some surface, *the 0-level set*
- Often we do not care about the distance outside a *narrow band*
- Typically only $O(n^2)$ voxels are used.



Level Set Method

- The LSM represents surfaces using distance fields
- Surface is updated indirectly by updating distance field:

$$\phi^{n+1} = \phi^n - F|\nabla\phi|$$

- Changes to topology are trivial

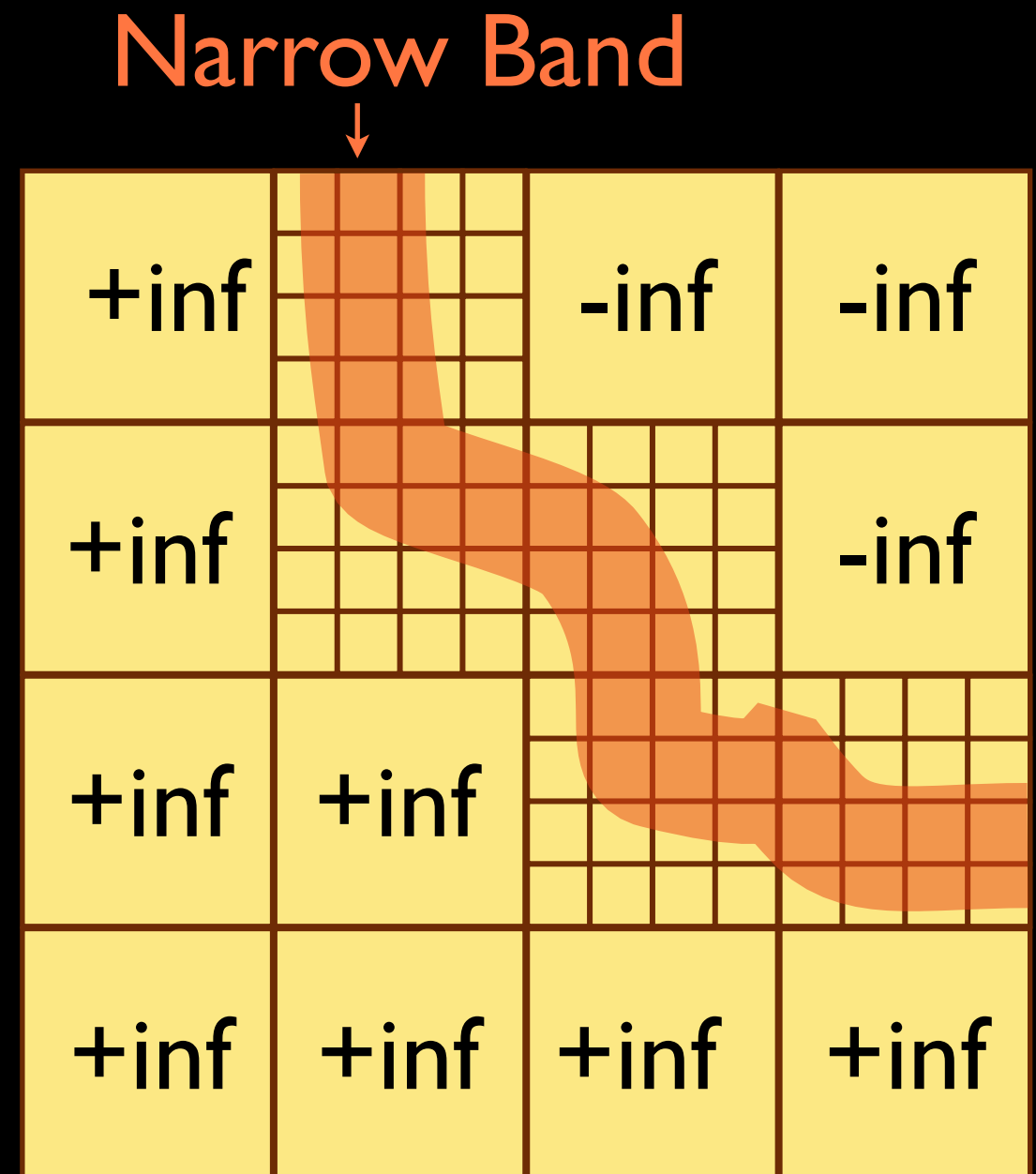


Recursive Decomposition

- Octrees can and have been used for distance fields but octrees exhibit
 - $O(\log n)$ access time not $O(1)$
 - poor data locality and too many indirections
 - Much computational overhead involved in maintaining data structure.

Hierarchical Grids

- Top-level grid contains pointers to fine grid OR +/- INF
- Fine grid contains precise distance values.
- Three or more levels could be used.
- Analyzed by Bridson (who called it *sparse block grids*)



Run Length Encoding

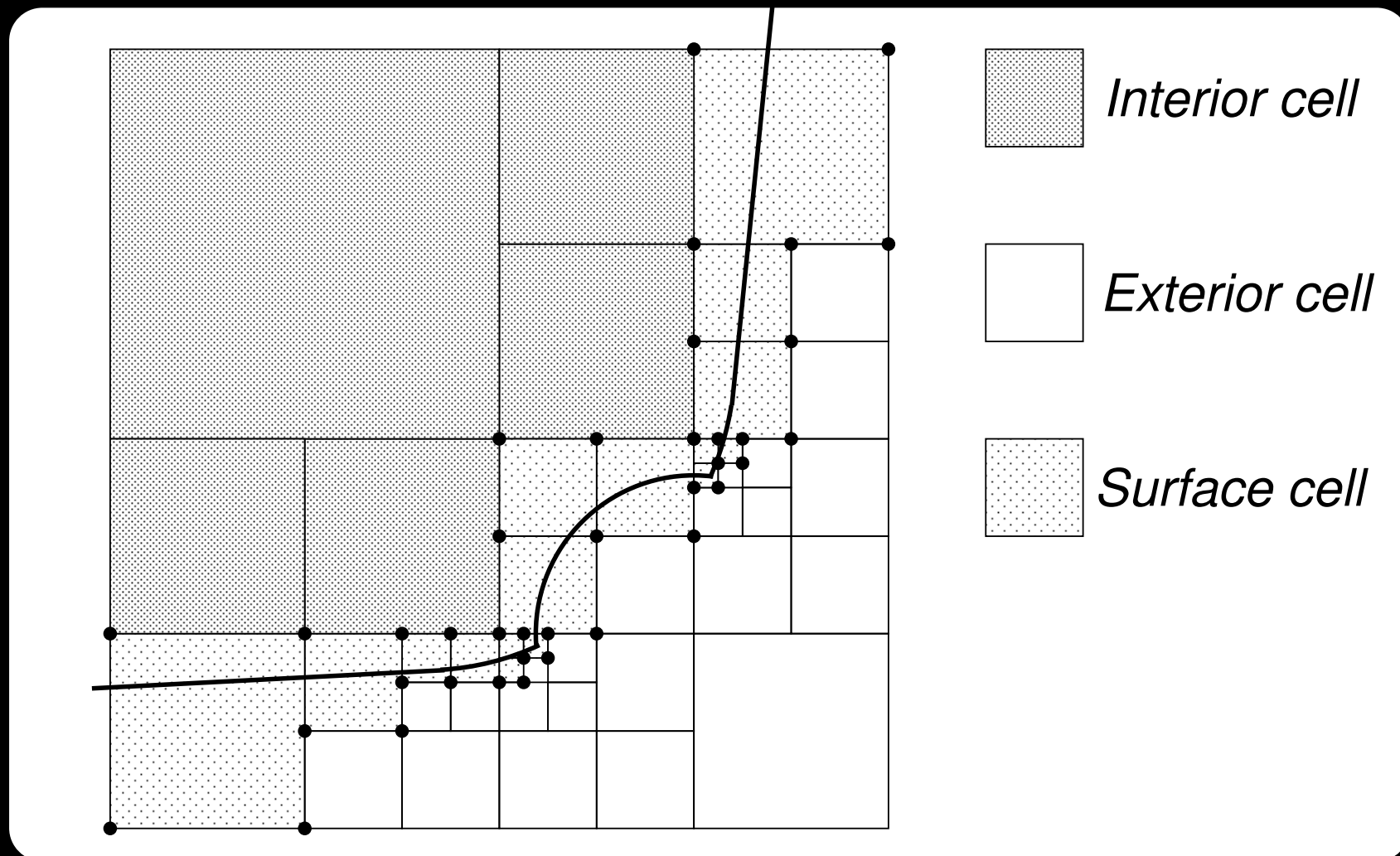
- A very simple way of compressing data. In ID:
 - Plain: AAAACCDEFFFFFFFGGGGGG
 - Coded: 4A2C1D1E6F6G
- Hierarchical Run-length encoding is a run length coding of a run length coding of a run length coding ...

Discussion

- Houston et al. [2006] claim better memory performance than octrees for H-RLE
 - Believable: Narrow band volumes ideal for run length coding.
 - Caveat: No comparison to hierarchical grids.
- H-RLE allows the volume to easily be extended in any direction.
- However: A hierarchical grid with a top level grid as hash table offers the same!

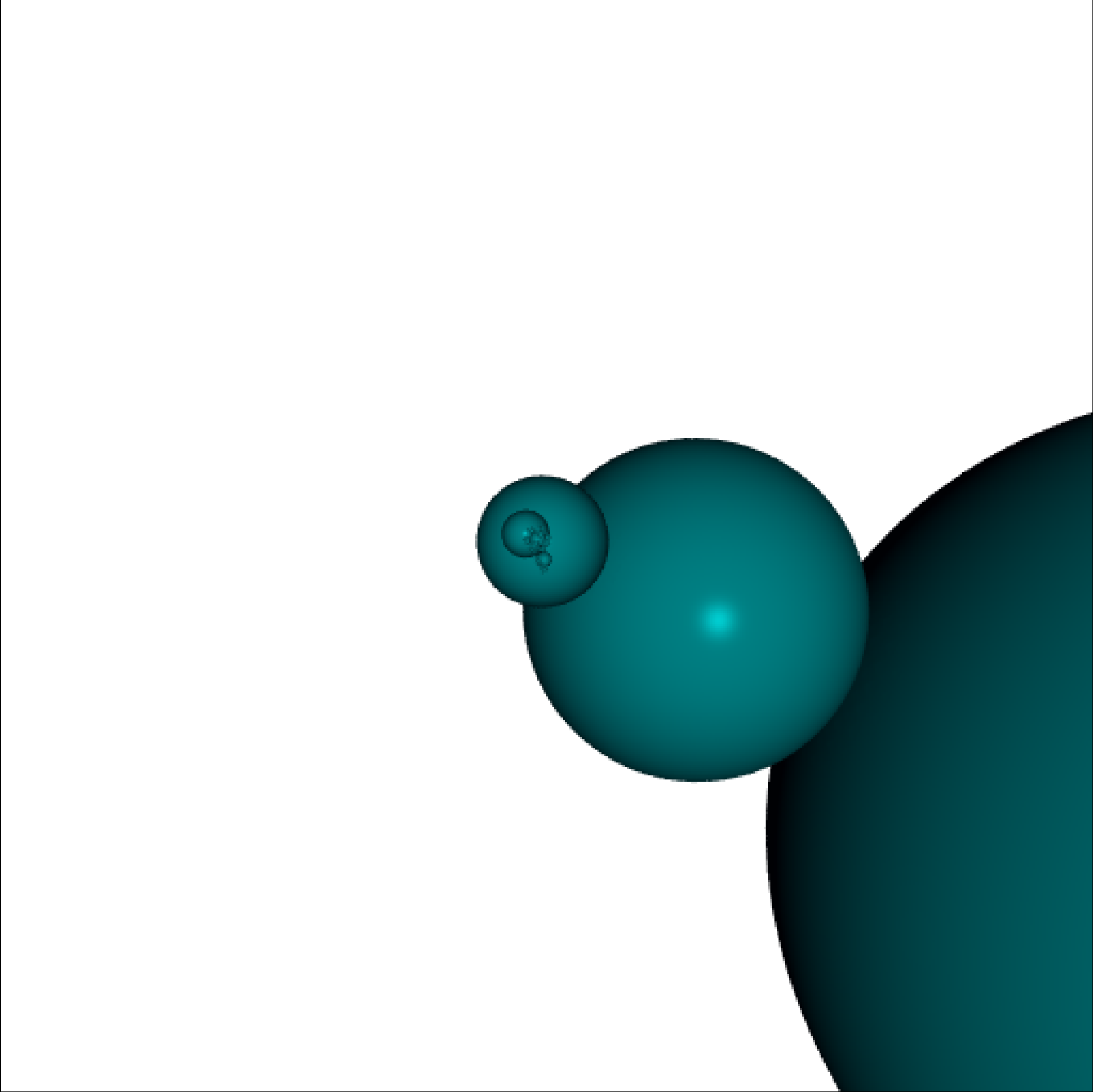
Adaptive Volumes

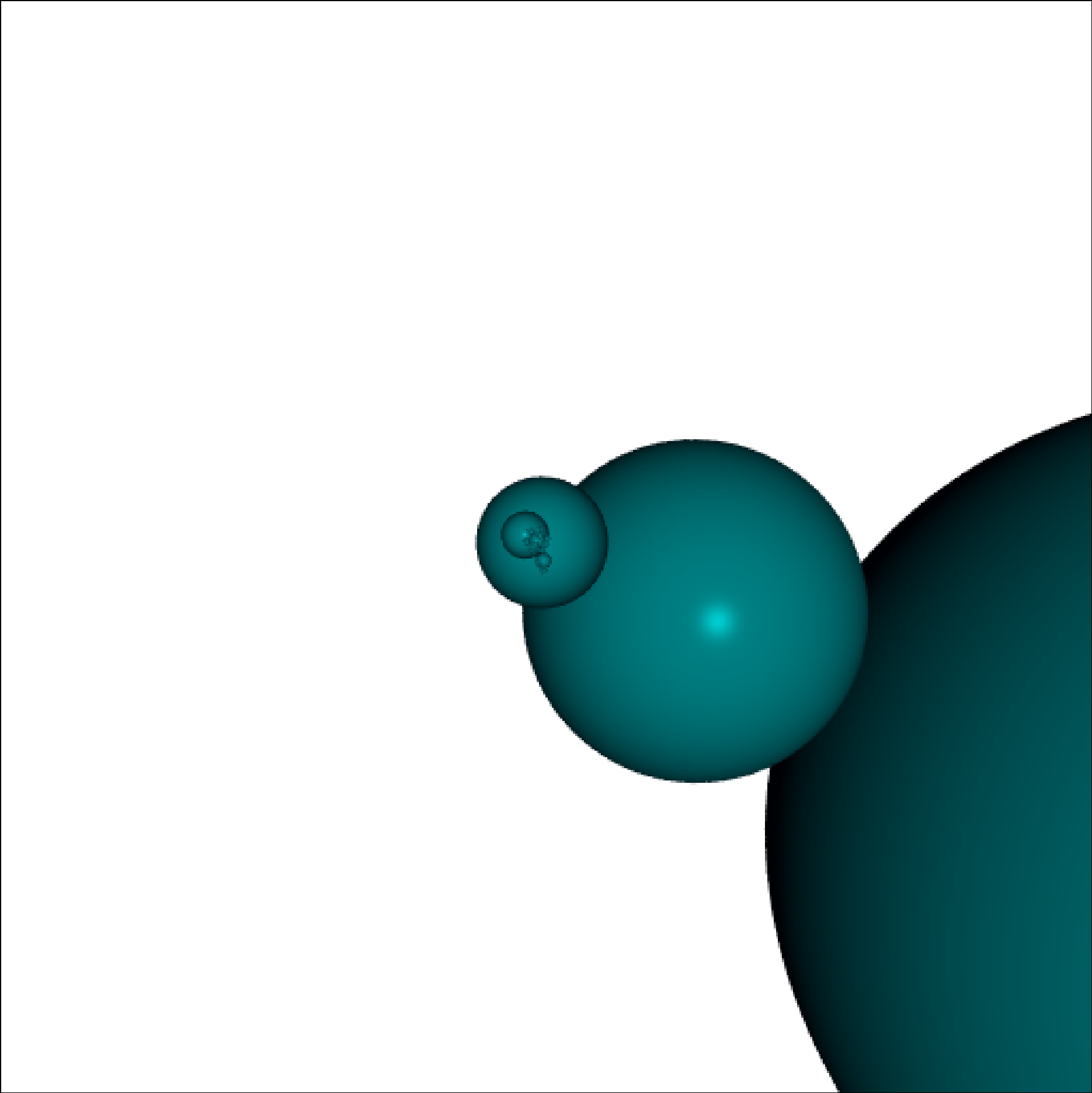
- Bærentzen used an octree to divide space into cells
- Voxels (grid points) are stored in a grid of hash tables.

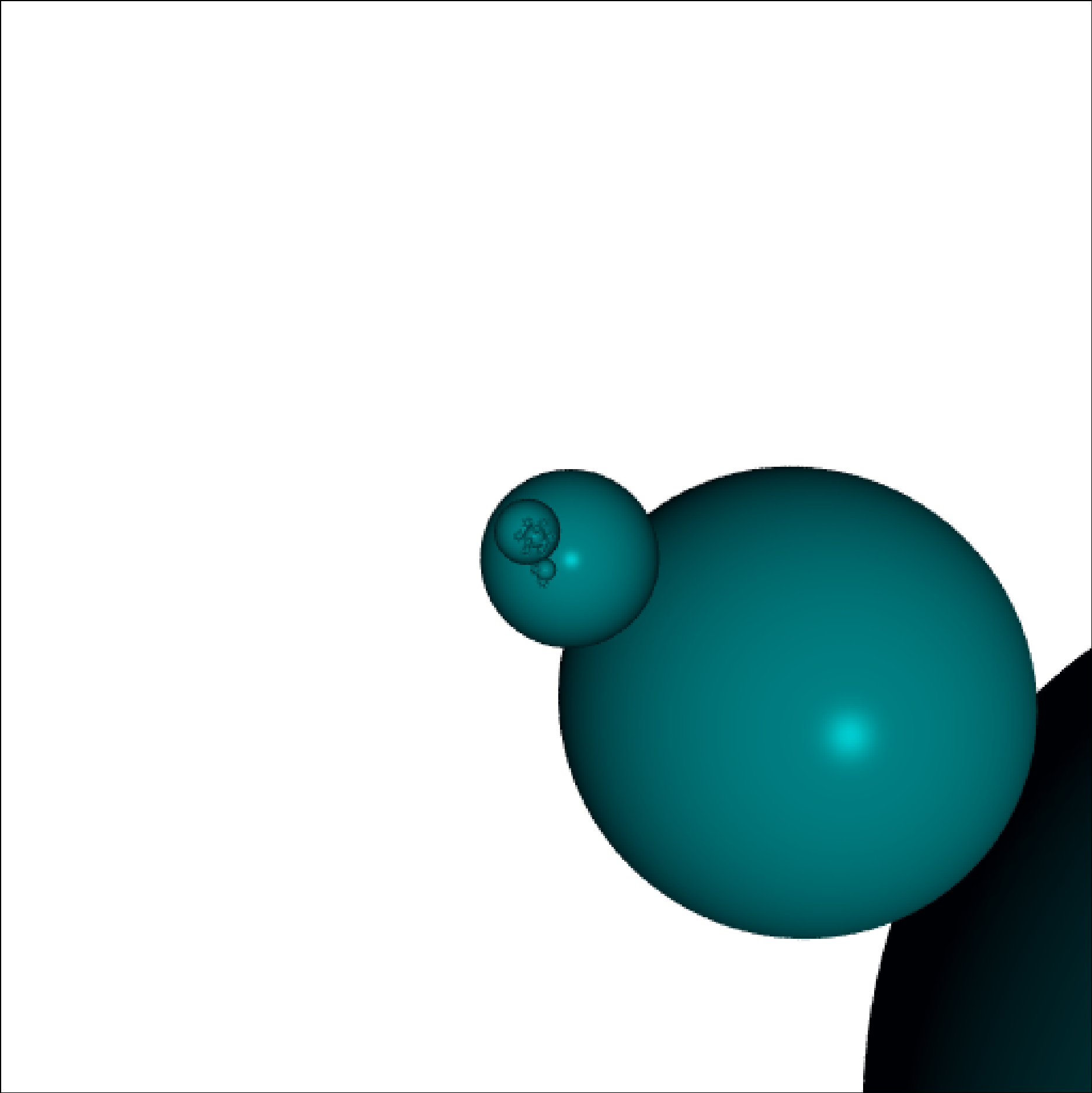


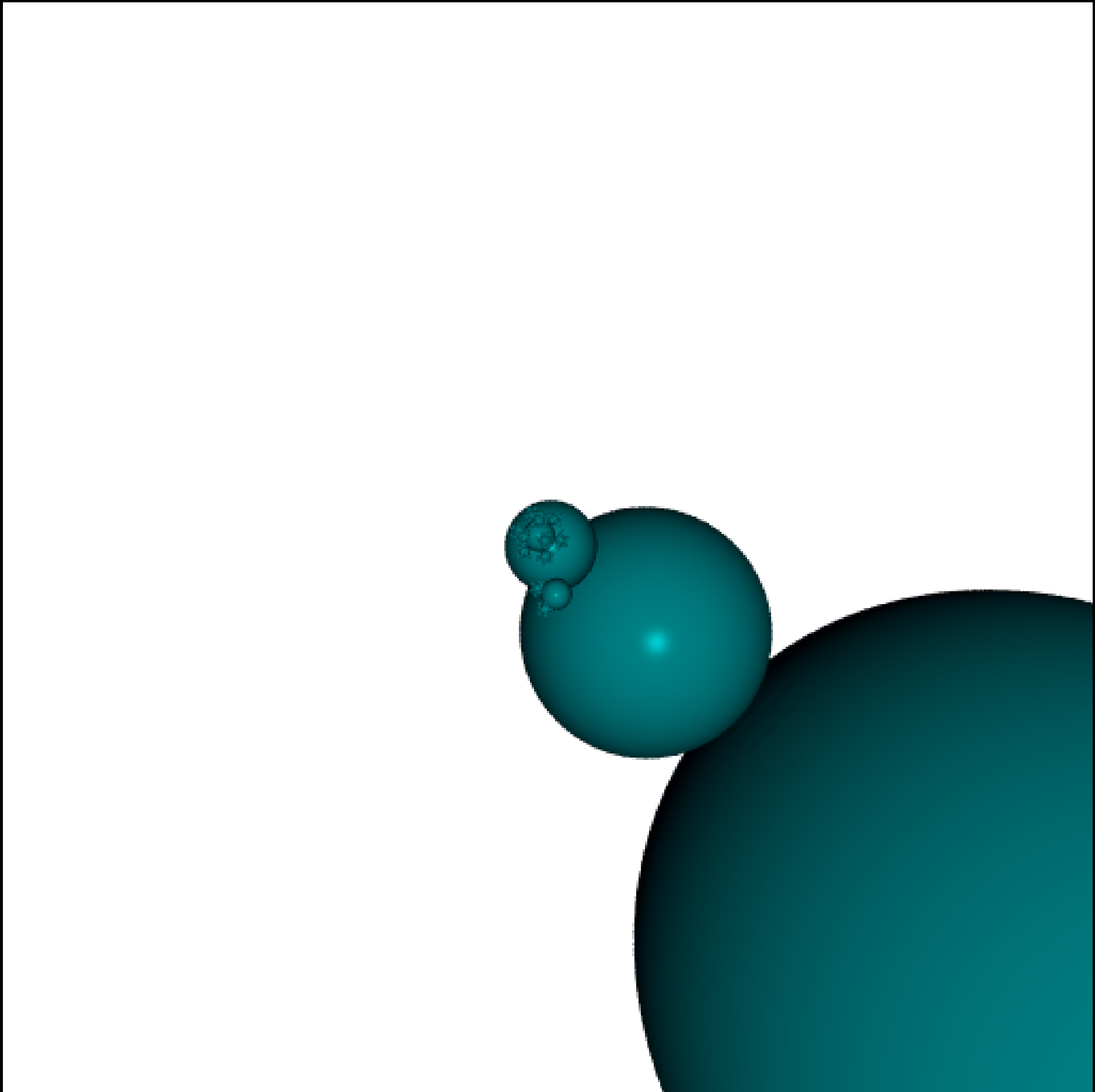
A 16384^3 Volume

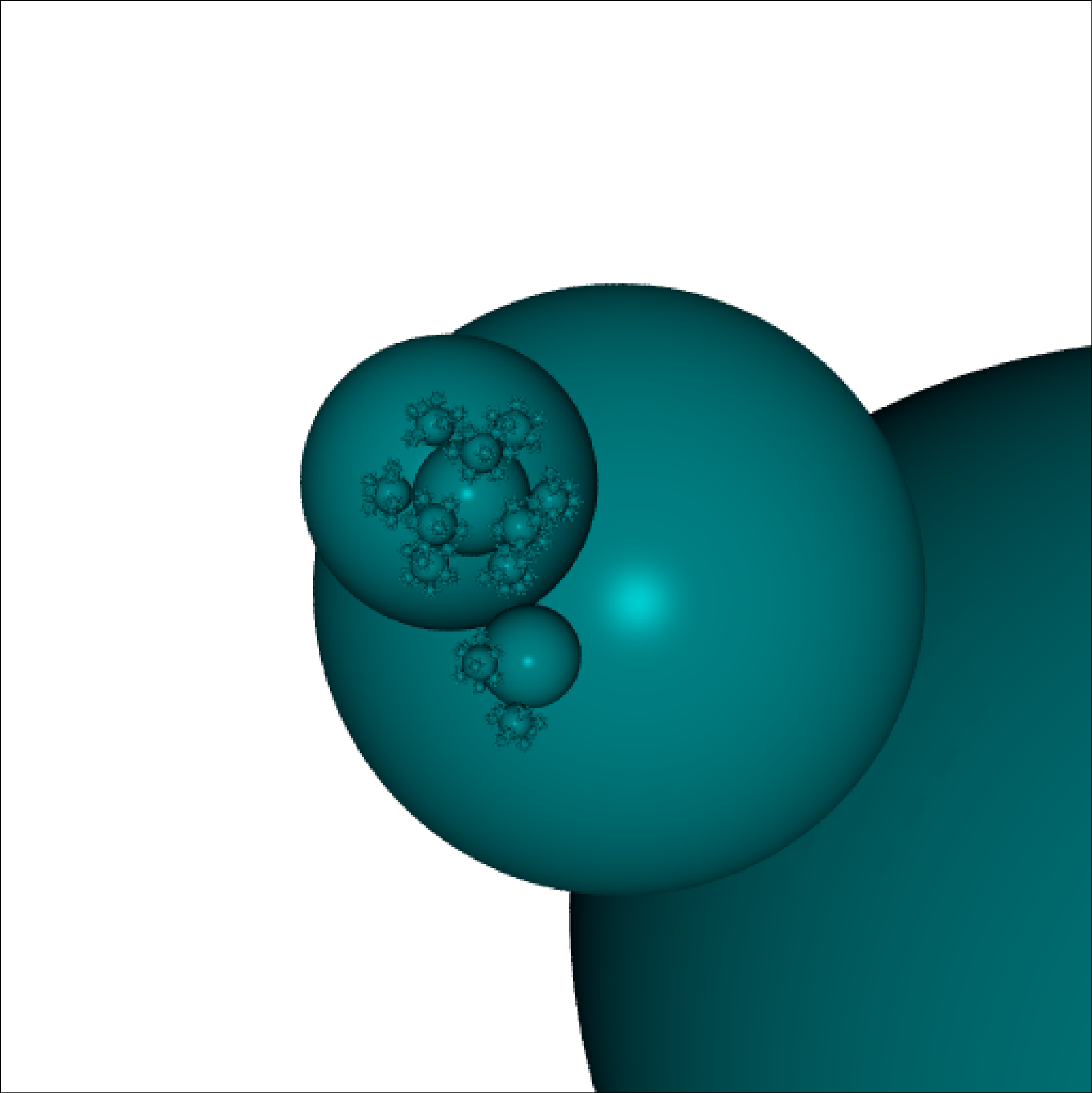
- Very high resolutions is possible if we only use high resolution where needed.
- Decoupling the storage of voxels from the octree, we can deal with huge volumes
- Similar to Frisken et al. [2000]

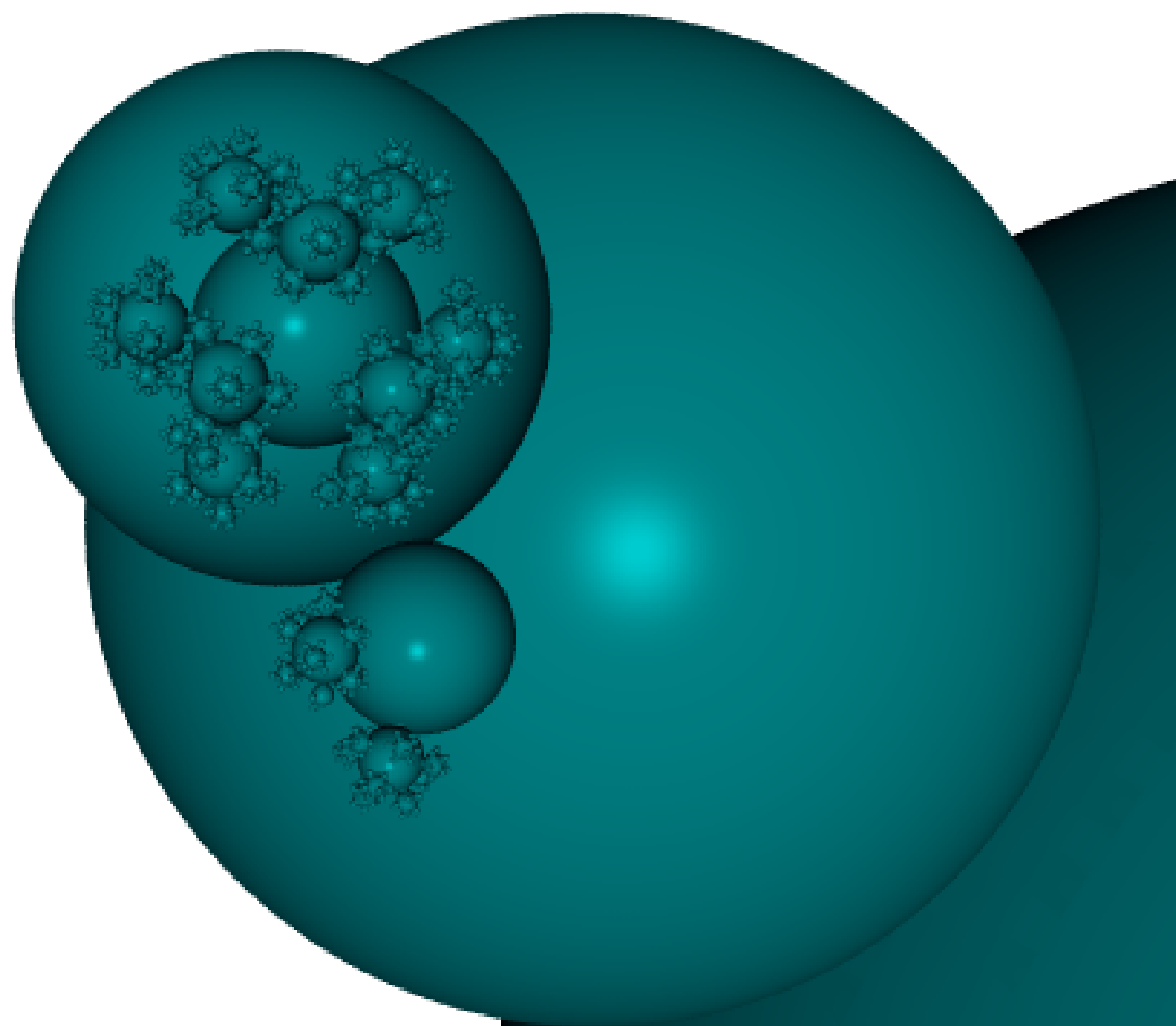


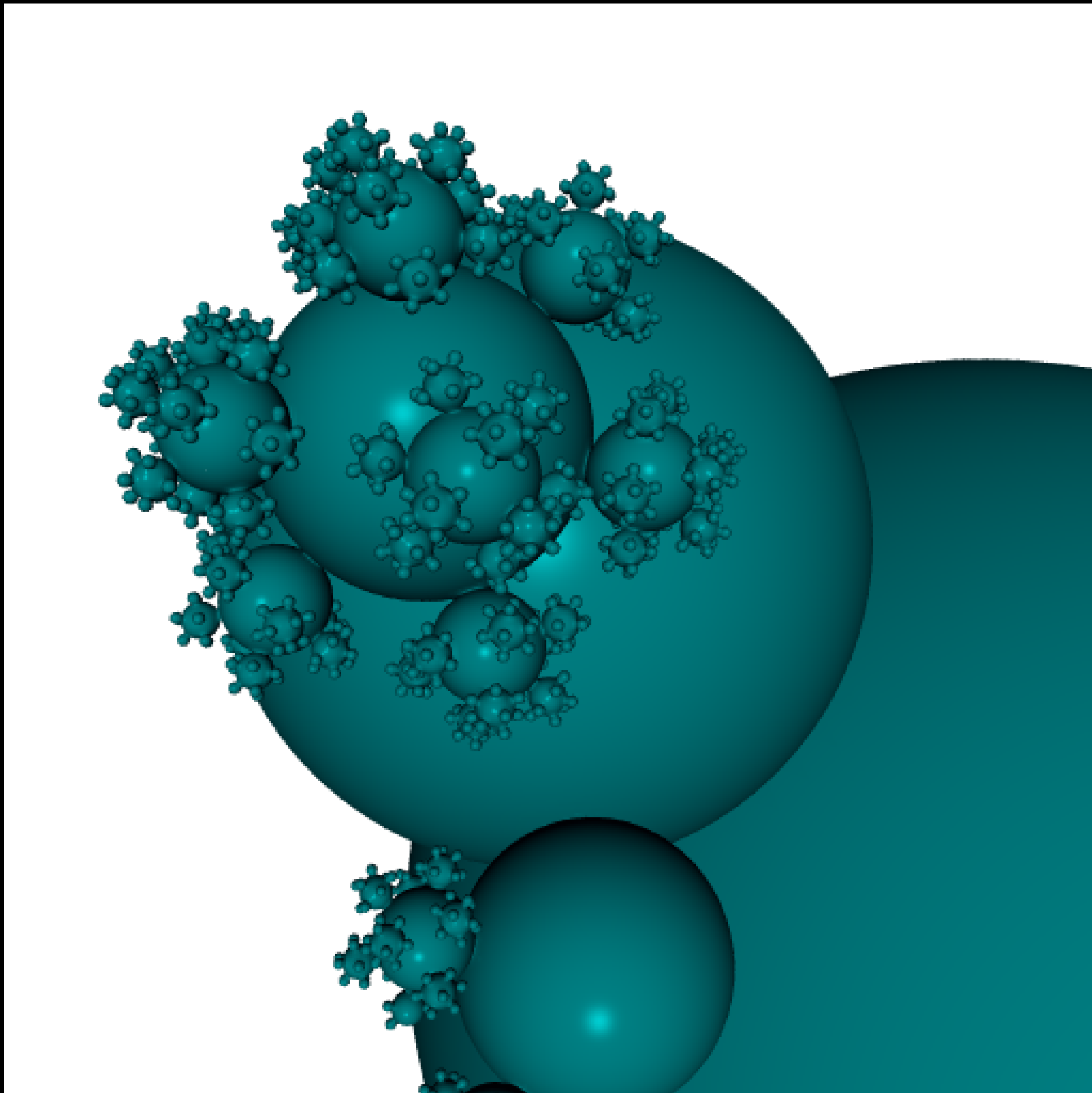


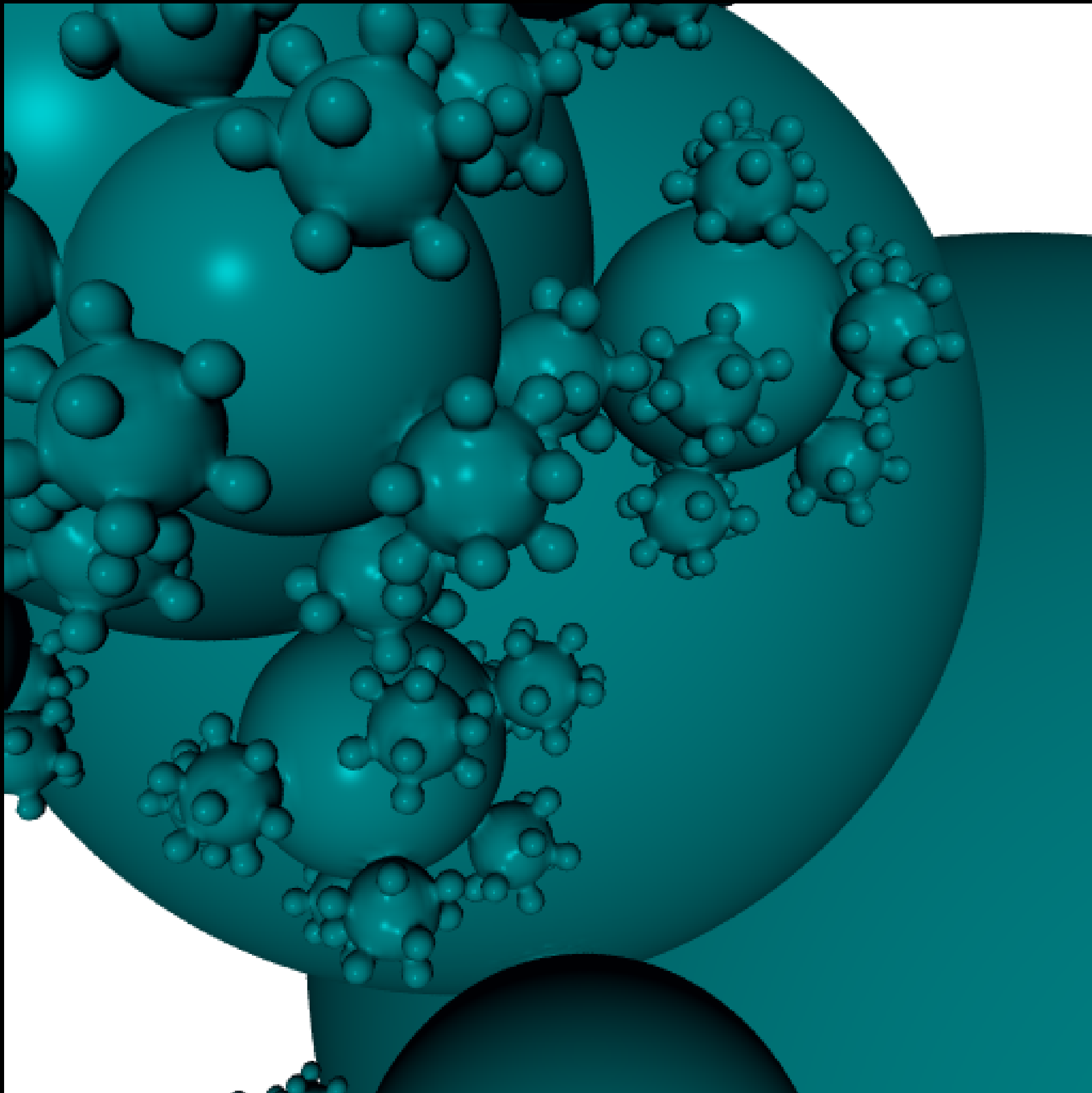












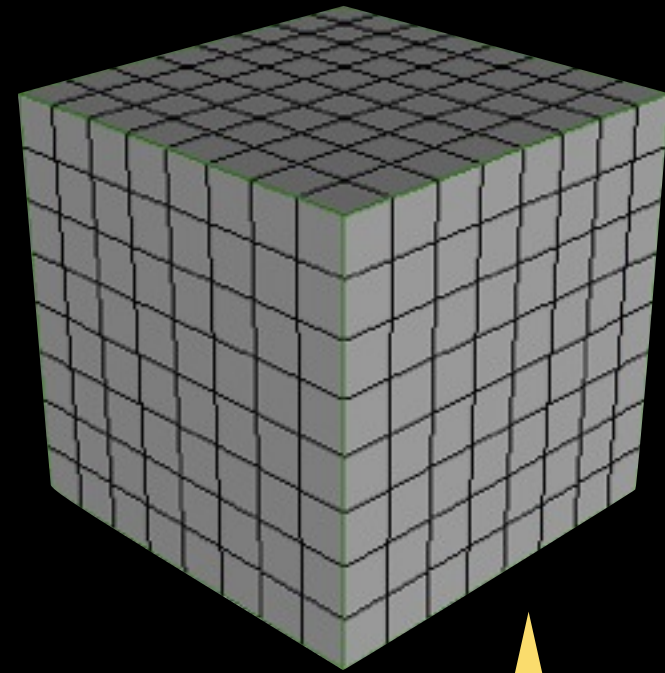
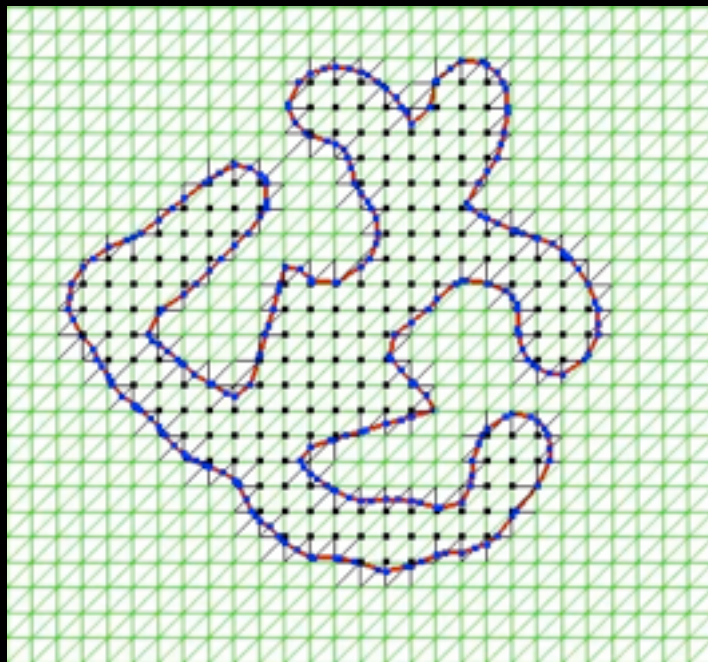
THE WALL

if you want to do the level set method with adaptive resolution

- Several plausible solutions for extremely high resolution volumes
- Some plausible solutions for adaptive resolution volumes - but these tend to be static
- Even if a adaptive dynamic volume rep was designed - the level set method has its flaws!

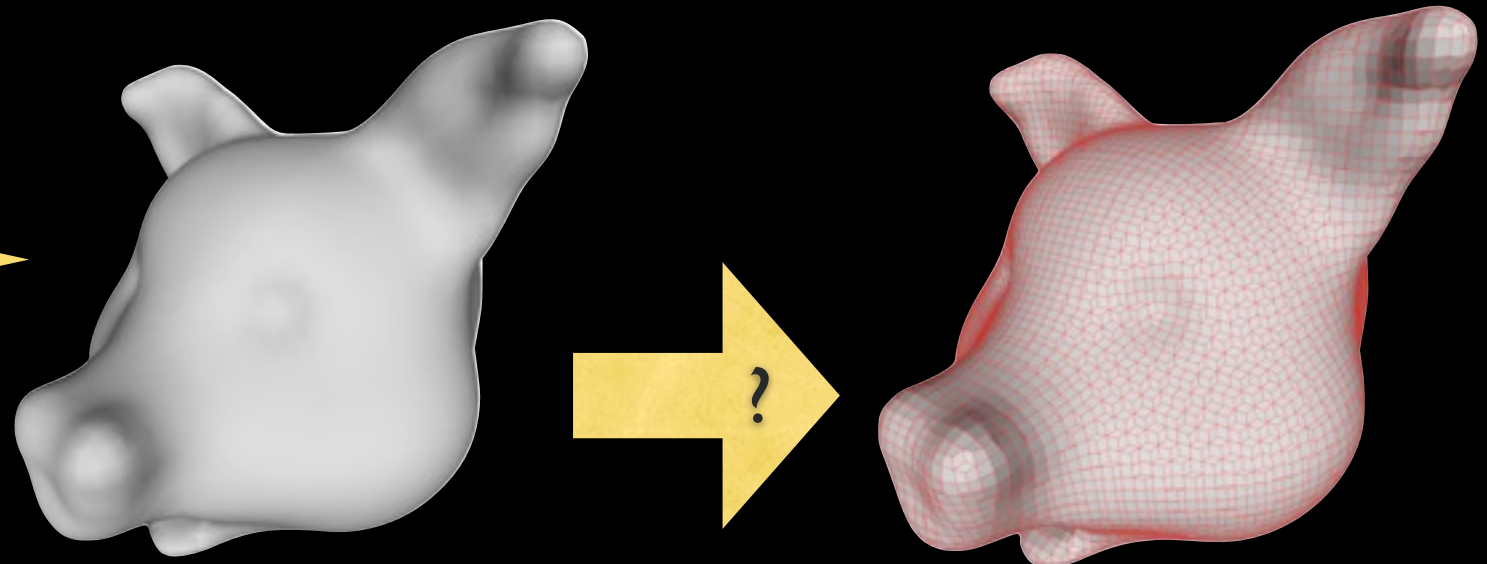
Top 3 Problems with the LSM

The LSM suffers from numerical diffusion



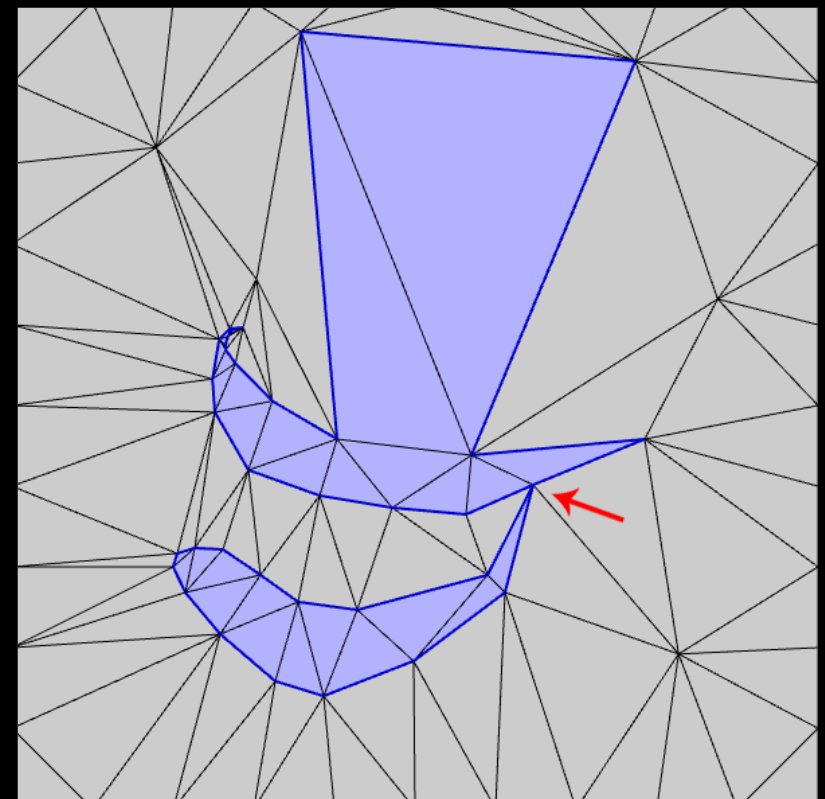
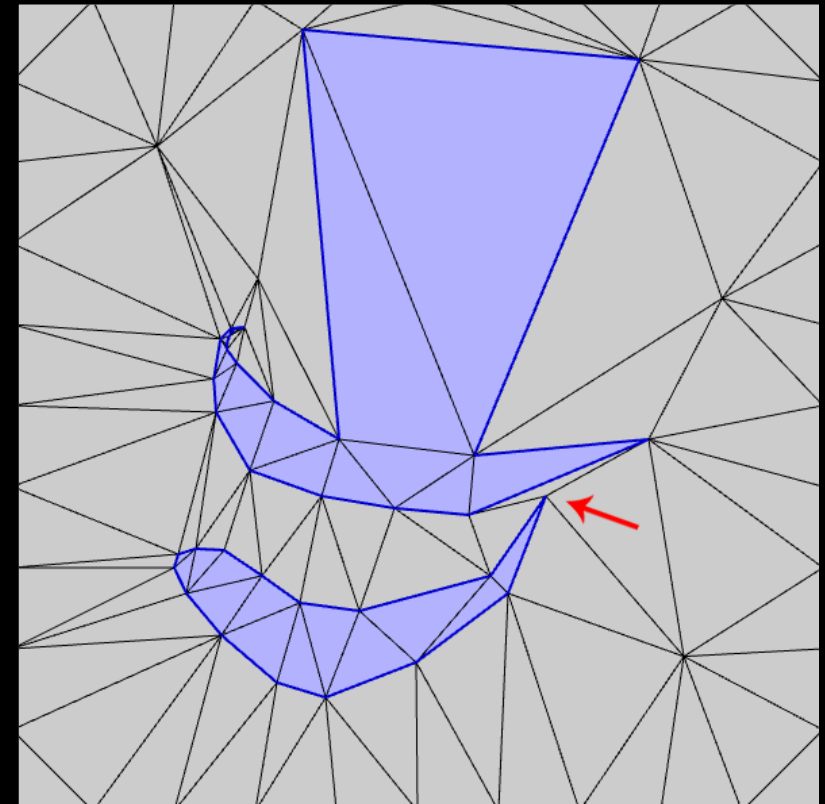
The LSM is bound to a scale interval dictated by the grid resolution

The LSM has no explicit interface representation

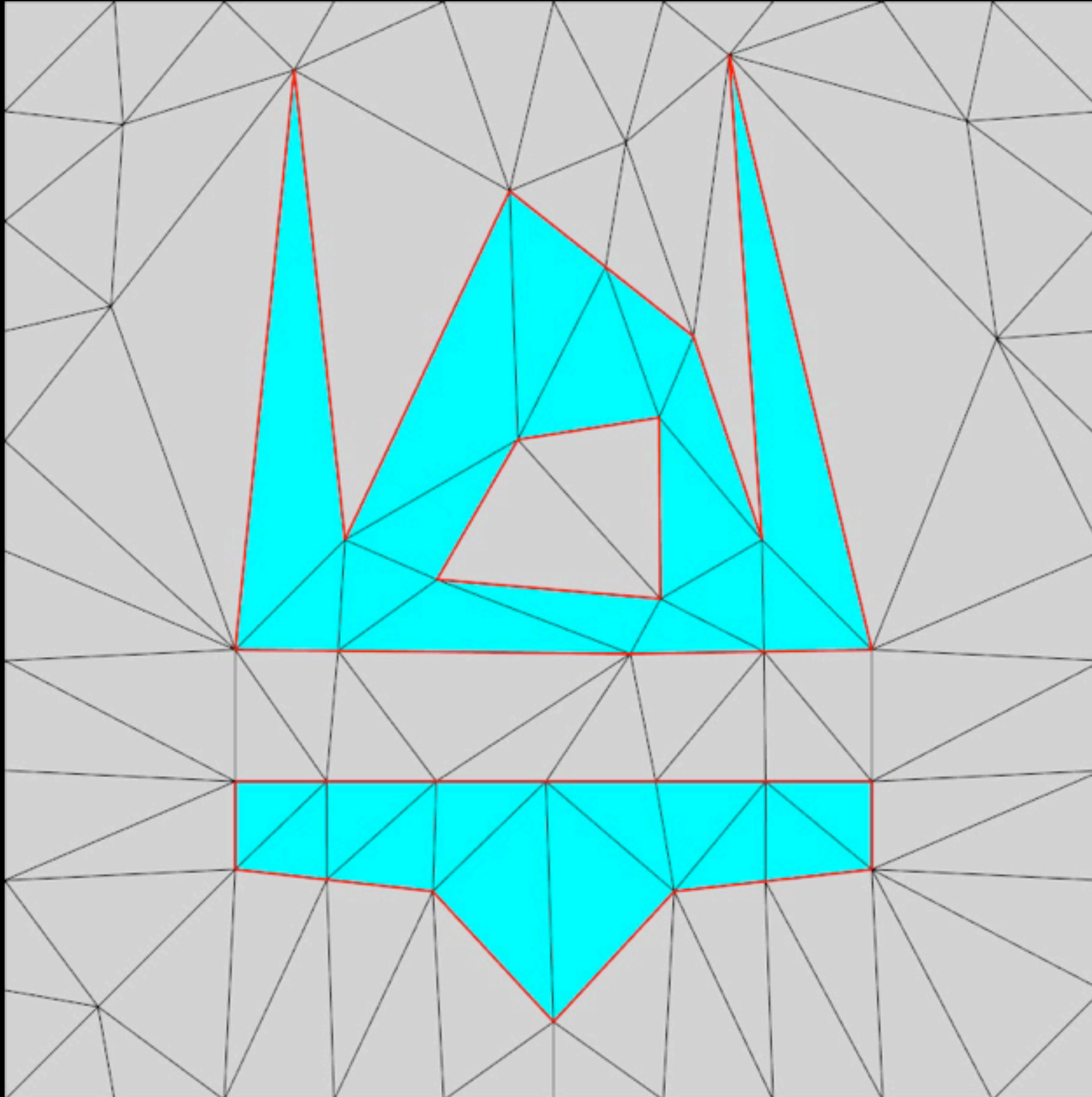


Deformable Simplicial Complexes

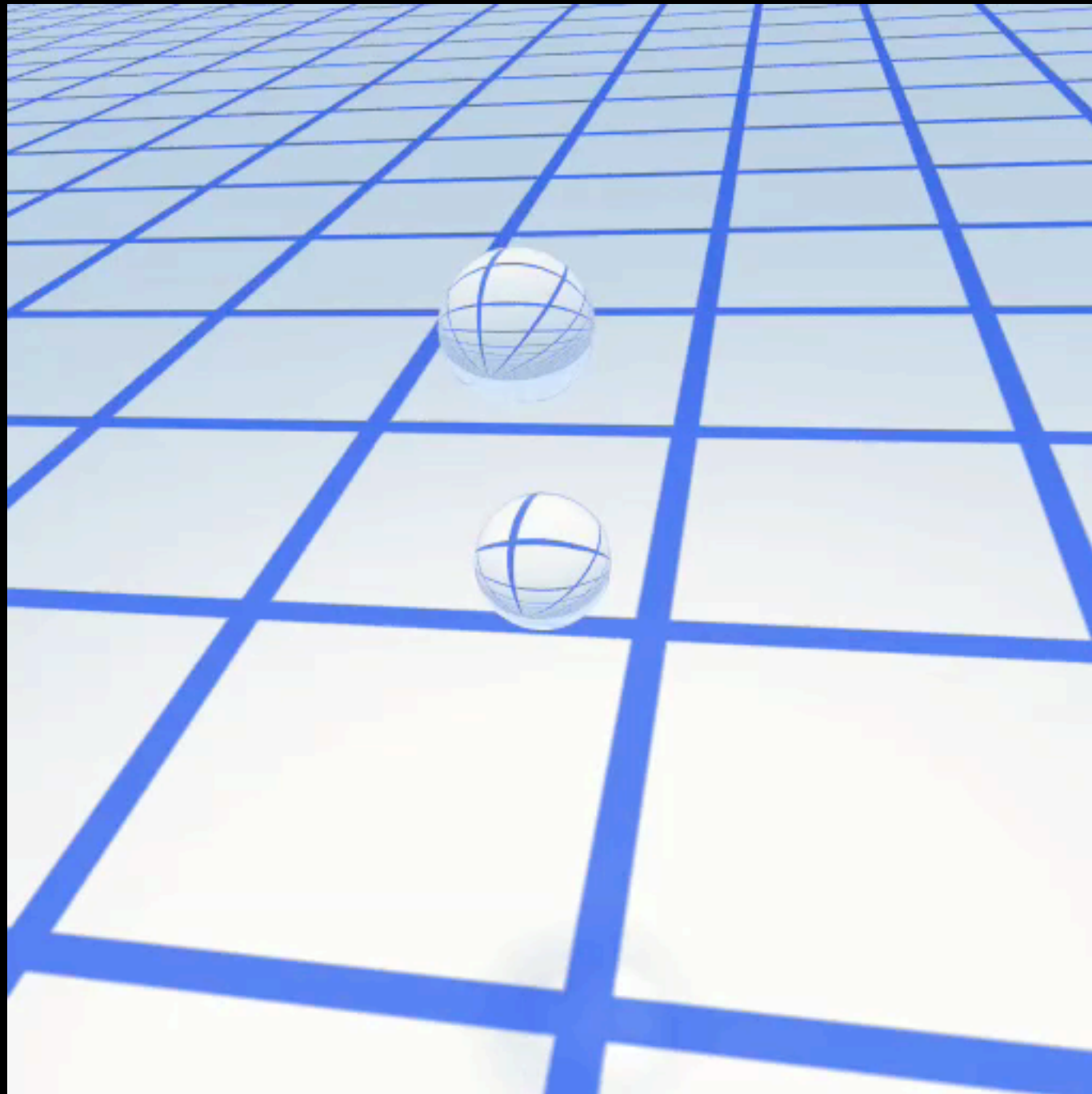
- The DSC method addresses all three “level set issues”
- Interface (boundary) is explicitly represented as a sub-complex of a simplicial complex
- which is irregular, hence may adapt to details.
- Interface vertices are moved directly which leads to very little diffusion.



In 2D



3D



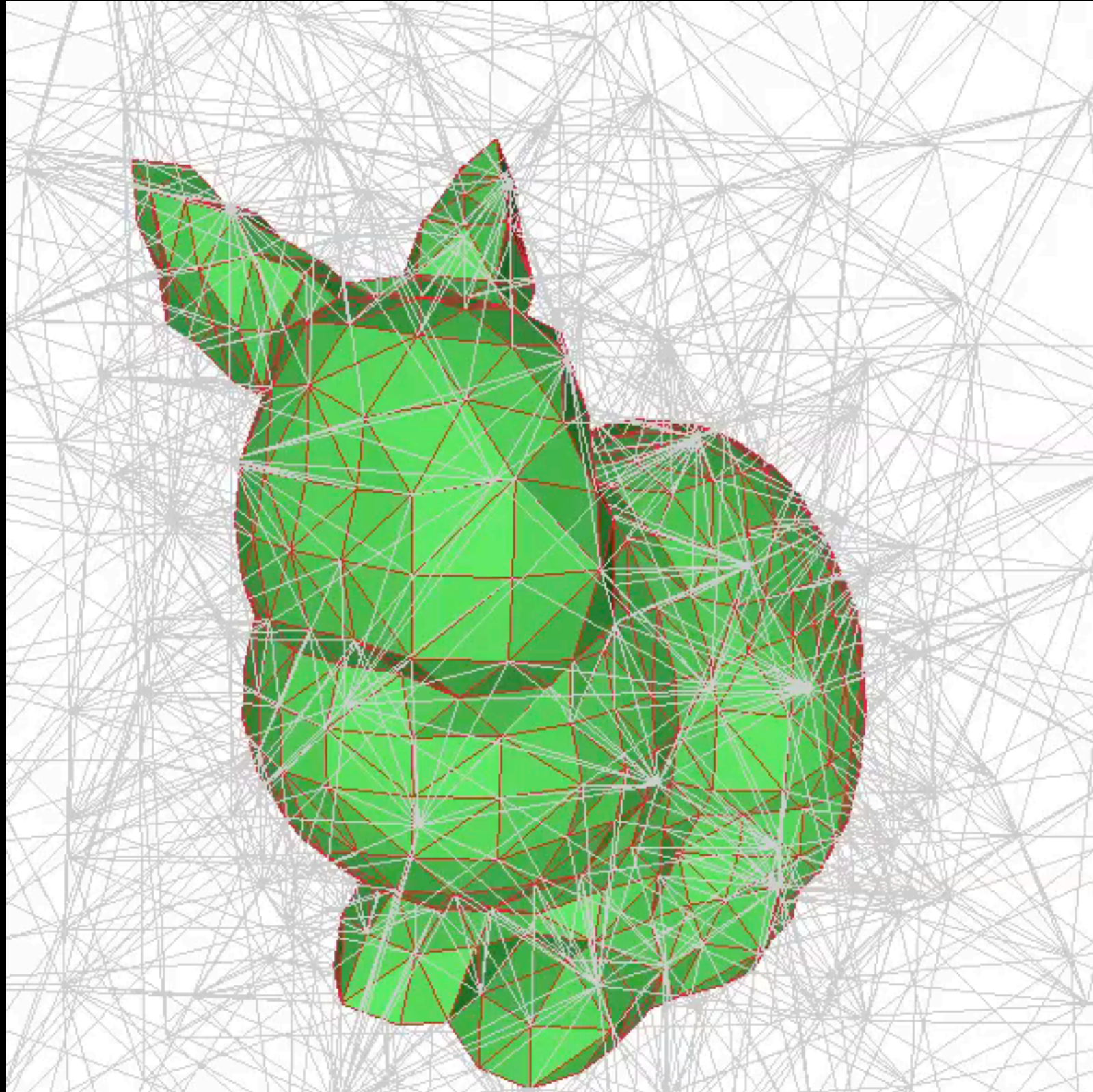
Steps

1. For each vertex find target positions
2. Move vertices as far as possible towards target without inversions.
3. Improve the mesh
 1. Smooth & improve connectivity of tetra mesh
 2. Remove degeneracies
 3. Improve surface mesh
4. Unless all vertices are at target go to 2

Advantages

1. Robust topological adaptivity
2. Little numerical diffusion
3. Highly scale-adaptive (thanks to the irregular grid) - this is where sparse comes in ...
4. Availability of both surface and volume representation
5. Explicit surface representation does not change gratuitously between time steps
6. Allows for topology control

Shrinking by Face Offsetting

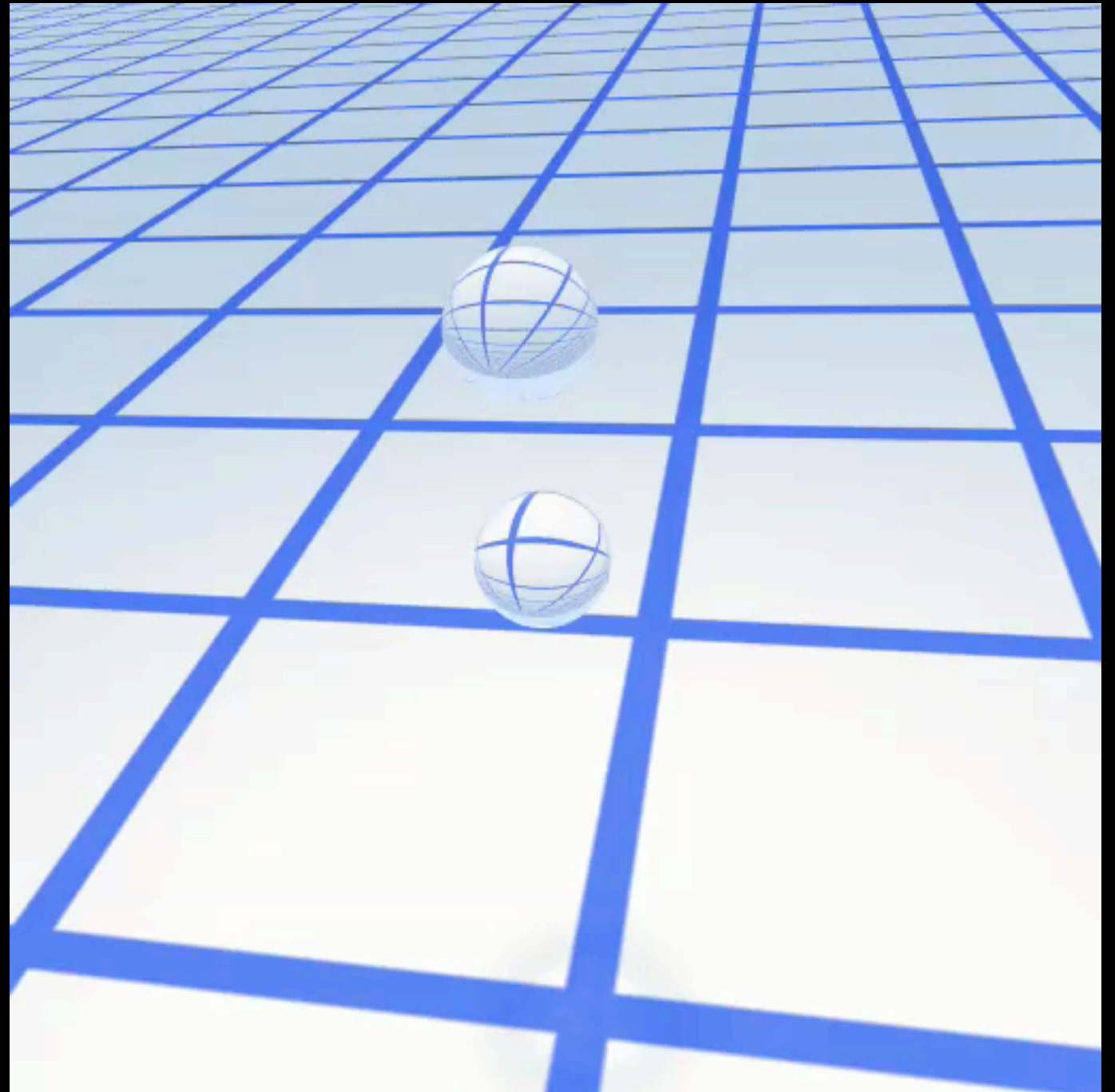


Conclusions

- Exploiting sparsity has its costs:
 - added complexity
 - lost coherence
- Hence chooses simple schemes:
hierarchical grids, run length coding.
- For problems where you need topological and spatial adaptivity use DSC (let me know :-)

Acknowledgements

- DSC is the work of Marek Misztal

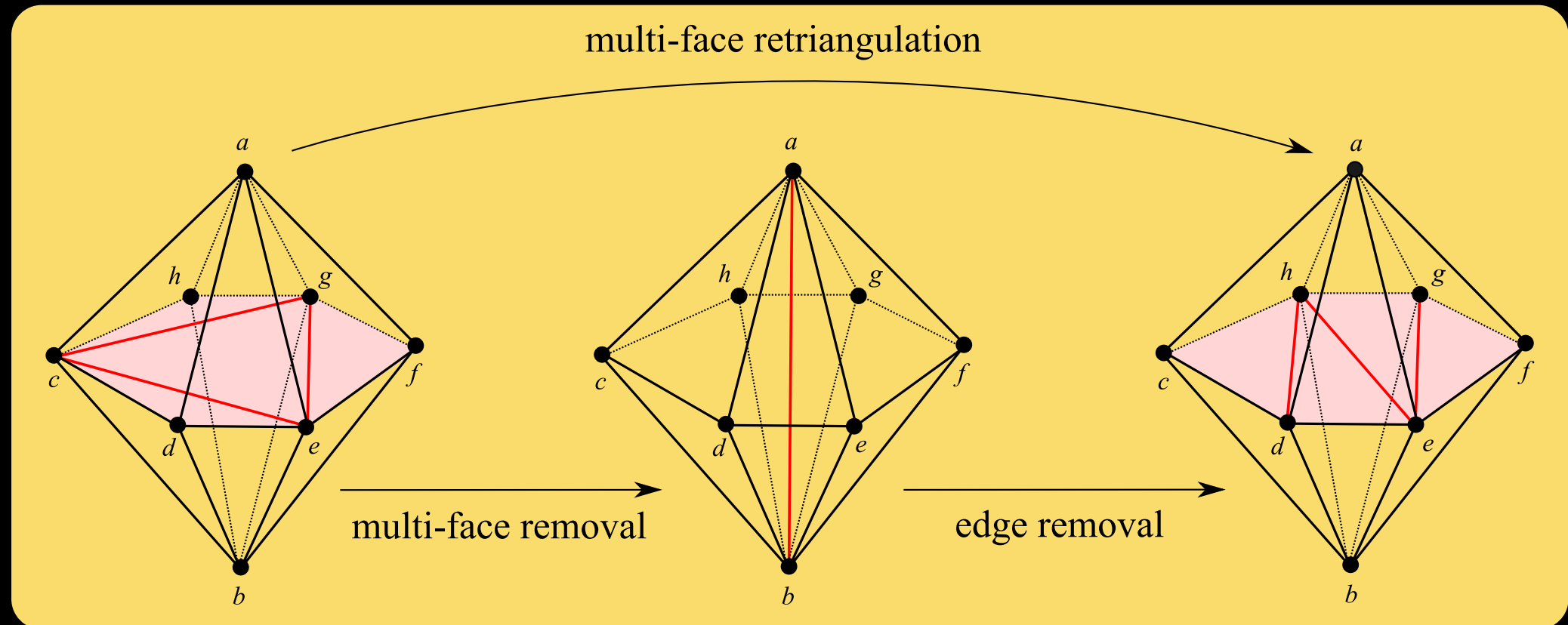


References

- Bridson, R. 2003. *Computational aspects of dynamic surfaces*. dissertation. Stanford University, Stanford, CA.
- Bærentzen J.A. *Manipulation of Volumetric Solides with Applications to Sculpting*, dissertation. Technical University of Denmark.
- Frisken, S. F., Perry, R. N., Rockwood, A. P., and Jones, T. R. 2000. *Adaptively sampled distance fields: A general representation of shape for computer graphics*. In Proceedings of SIGGRAPH 2000, pp. 249–254.
- Ben Houston Michael Bang Nielsen, Christopher Batty, Ola Nilsson and Ken Museth. *Hierarchical RLE Level Set: A Compact and Versatile Deformable Surface Representation*, ACM Transactions on Graphics 25(1), January 2006. Pages 151-175
- Lefebvre, S. and Hoppe H. *Perfect Spatial Hashing*, In Proceedings of SIGGRAPH 2006, pp. 579-588
- Schneider, J., and R. Westermann. 2003. *Compression Domain Volume Rendering*. In Proceedings of IEEE Visualization, pp. 293–300.

Thanks ... questions?

bonus



Hash Tables


- Observations:
 - We need to store the key
 - If the table is nearly full, we get many collisions, so we stay below 70% load factor
 - Cache coherence and hashing are at odds!
 - Note: we can traverse the hashed-to table linearly
 - A hash table could be used to store the top level grid in a two level hierarchy

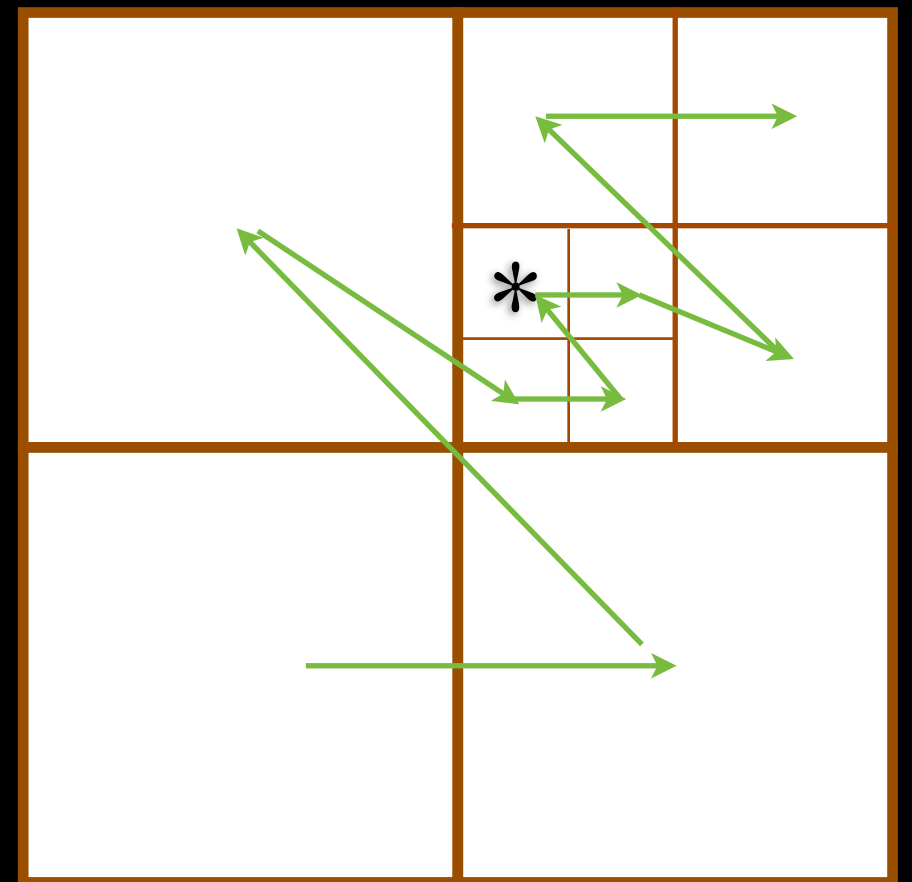
Efficient Sparse Voxel Octrees

- Laine et al. 2010 proposed a GPU based method
- Polygonal geometry converted to voxels - stored in octree
- Neighbour information is never used. Blockiness accepted and fixed in post process blurring




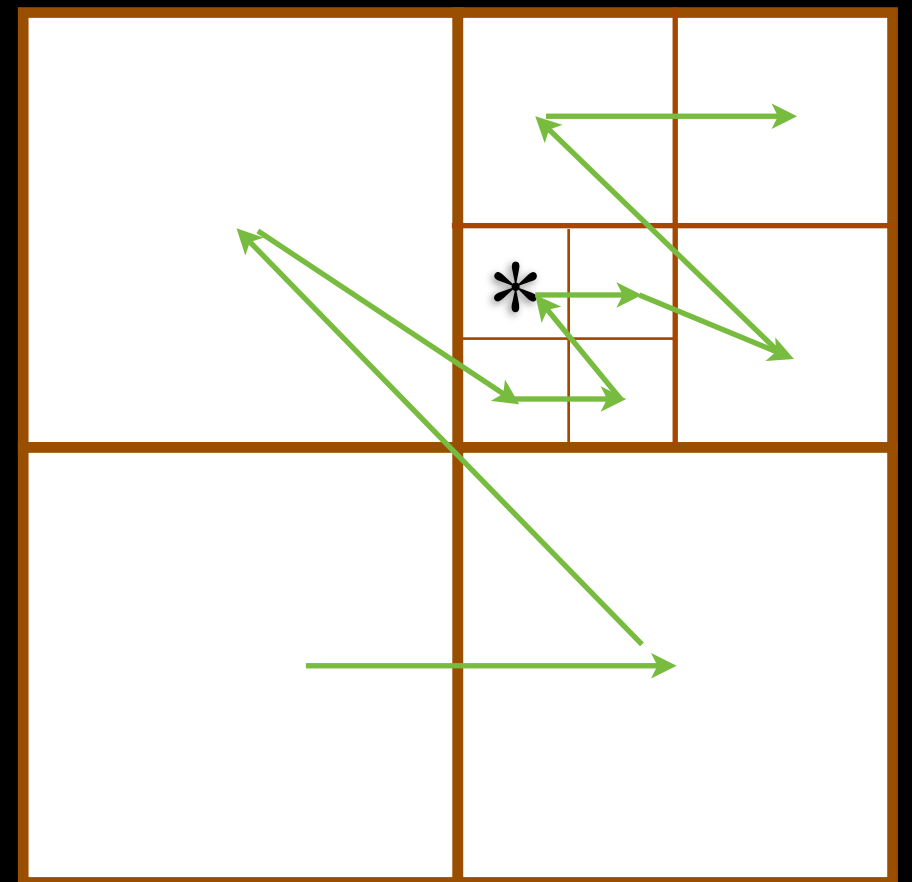
Linear Octrees

- Observe 
- $x=100, y=101$
- The octree path is



Linear Octrees

- Observe 
- $x=$, $y=$
- The octree path is 110010=302 (base 4)



Linear Octrees

- In the normal grid encoding, we shift concatenate $Z_0Z_1Z_2Y_0Y_1Y_2X_0X_1X_2$
- To compute the *Morton* code we interlace $Z_0Y_0Z_1Y_1X_1Z_2Y_2X_2$
- A linear octree is an array of Morton codes. Saves intermediate nodes+pointers!
- We could use b bits from Morton code as a simple hash function [Lewiner et al. 2010]