SIGGRAPH ASIA 2018 TOKYO
CONFERENCE 4 – 7 December 2018
EXHIBITION 5 – 7 December 2018
Tokyo International Forum, Japan
SA2018.SIGGRAPH.ORG

GPU-Based Large-Scale Scientific Visualization

Johanna Beyer, Harvard University

Markus Hadwiger, KAUST

Course Website:

http://johanna-b.github.io/LargeSciVis2018/index.html

SA '18 Courses, December 04-07, 2018, Tokyo, Japan
ACM 978-1-4503-6026-5/18/12.
10.1145/3277644.3277805

COURSE OVERVIEW (1)

Overview of modern GPU techniques for large-scale visualization

- Focus on volume data

Out-of-core techniques leveraging modern GPU features

- Virtual texturing approaches

Display-Aware, Remote and Web-Based Visualization

Course webpage (updated material):

http://johanna-b.github.io/LargeSciVis2018/index.html

State-of-the-Art in GPU-Based Large-Scale Volume Visualization

[J. Beyer, M. Hadwiger, H. Pfister; Computer Graphics Forum, 2015]

https://dl.acm.org/citation.cfm?id=3071497

COURSE OVERVIEW (2)

- Part 1 – Introduction & Basics of Scalable Volume Visualization
  Markus Hadwiger [60 min]
- Part 2 – Scalable Volume Visualization Architectures & Applications
  Johanna Beyer [45 min]
- Break

COURSE OVERVIEW (3)

- Part 3 – GPU-Based Ray-Guided Volume Rendering Algorithms
  Johanna Beyer [60 min]

- Part 4 – Display-Aware Visualization and Processing
  Markus Hadwiger [45 min]

- Part 5 – Outlook and Summary
  Johanna Beyer, Markus Hadwiger [15 min]

Part 1 -
Introduction & Basics of Scalable Volume Visualization

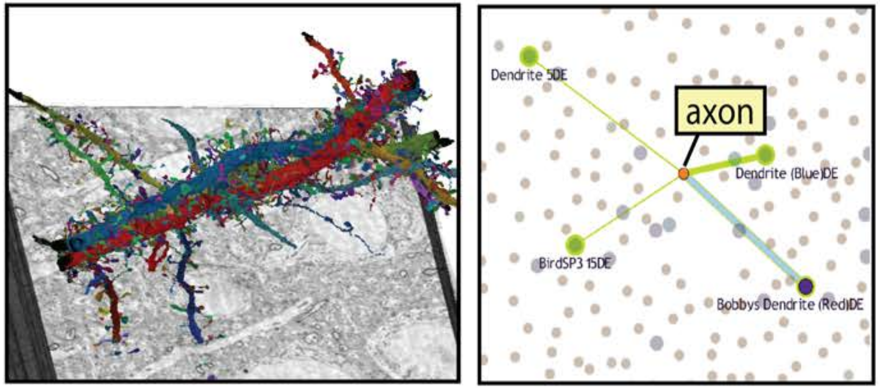
Motivation

BIG DATA

"In information technology, big data is a collection of data sets so large and complex that it becomes difficult to process using on-hand database management tools or traditional data processing applications. The challenges include capture, curation, storage, search, sharing, analysis, and visualization."

'Big Data' on wikipedia.org

Our interest:
Very large 3D volume data

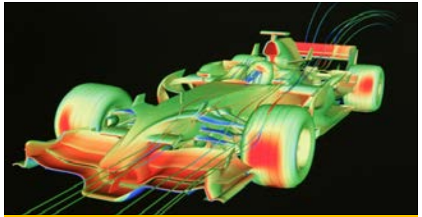


Example: Connectomics (neuroscience)

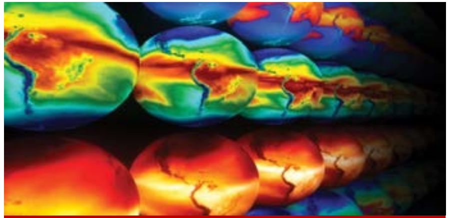
DATA-DRIVEN SCIENCE (E-SCIENCE)

MEDICINE
Digital Health Records

BIOLOGY
Connectomics



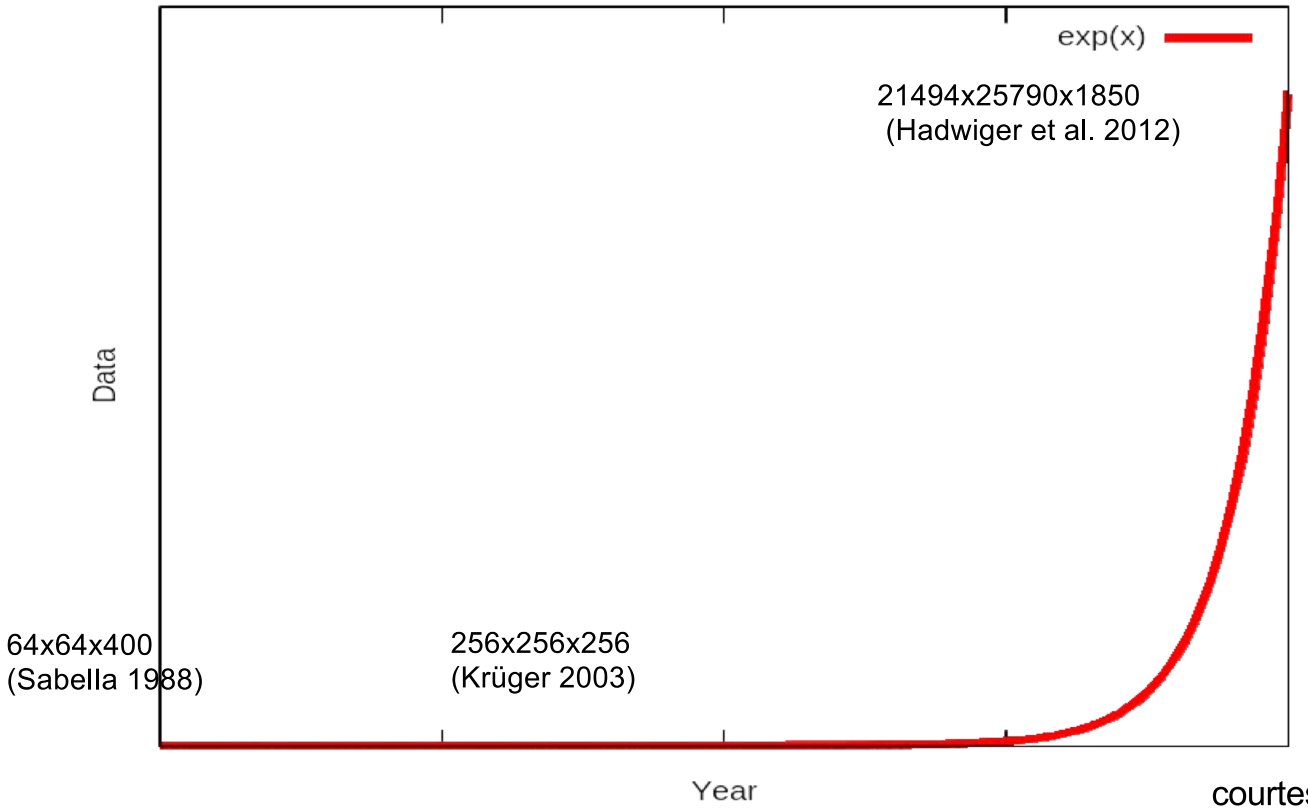
ENGINEERING
Large CFD Simulations



EARTH SCIENCES
Global Climate Models

courtesy Stefan Bruckner

VOLUME DATA GROWTH



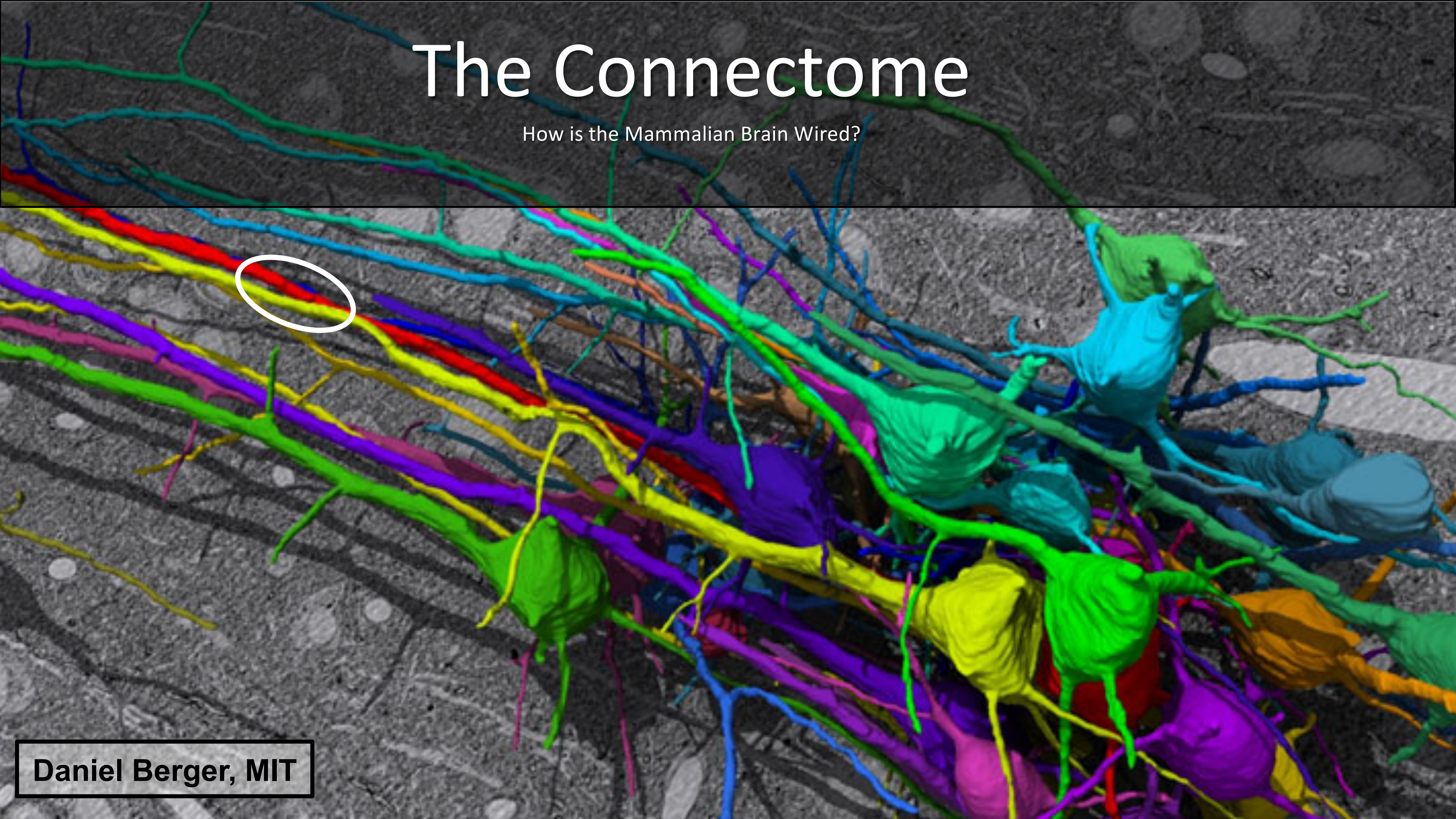
courtesy Jens Krüger

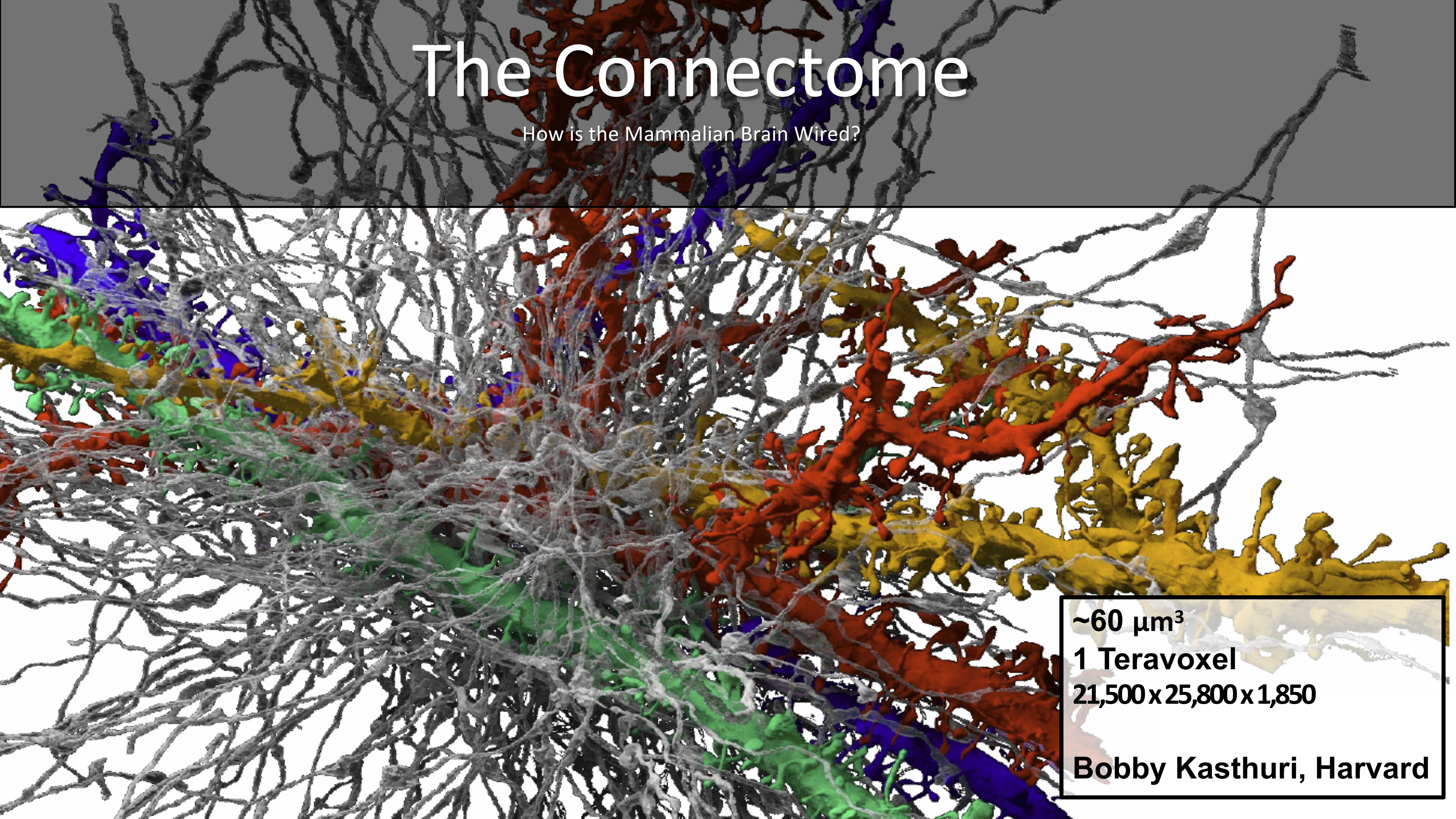
DATA SIZE EXAMPLES

year	paper	data set size	comments
2002	Guthe et al.	512 x 512 x 999 (500 MB) 2,048 x 1,216 x 1,877 (4.4 GB)	multi-pass, wavelet compression, streaming from disk
2003	Krüger & Westermann	256 x 256 x 256 (32 MB)	single-pass ray-casting
2005	Hadwiger et al.	576 x 352 x 1,536 (594 MB)	single-pass ray-casting (bricked)
2006	Ljung	512 x 512 x 628 (314 MB) 512 x 512 x 3396 (1.7 GB)	single-pass ray-casting, multi-resolution
2008	Gobbetti et al.	2,048 x 1,024 x 1,080 (4.2 GB)	'ray-guided' ray-casting with occlusion queries
2009	Crassin et al.	8,192 x 8,192 x 8,192 (512 GB)	ray-guided ray-casting
2011	Engel	8,192 x 8,192 x 16,384 (1 TB)	ray-guided ray-casting
2012	Hadwiger et al.	18,000 x 18,000 x 304 (92 GB) 21,494 x 25,790 x 1,850 (955 GB)	ray-guided ray-casting visualization-driven system
2013	Fogal et al.	1,728 x 1,008 x 1,878 (12.2 GB) 8,192 x 8,192 x 8,192 (512 GB)	ray-guided ray-casting

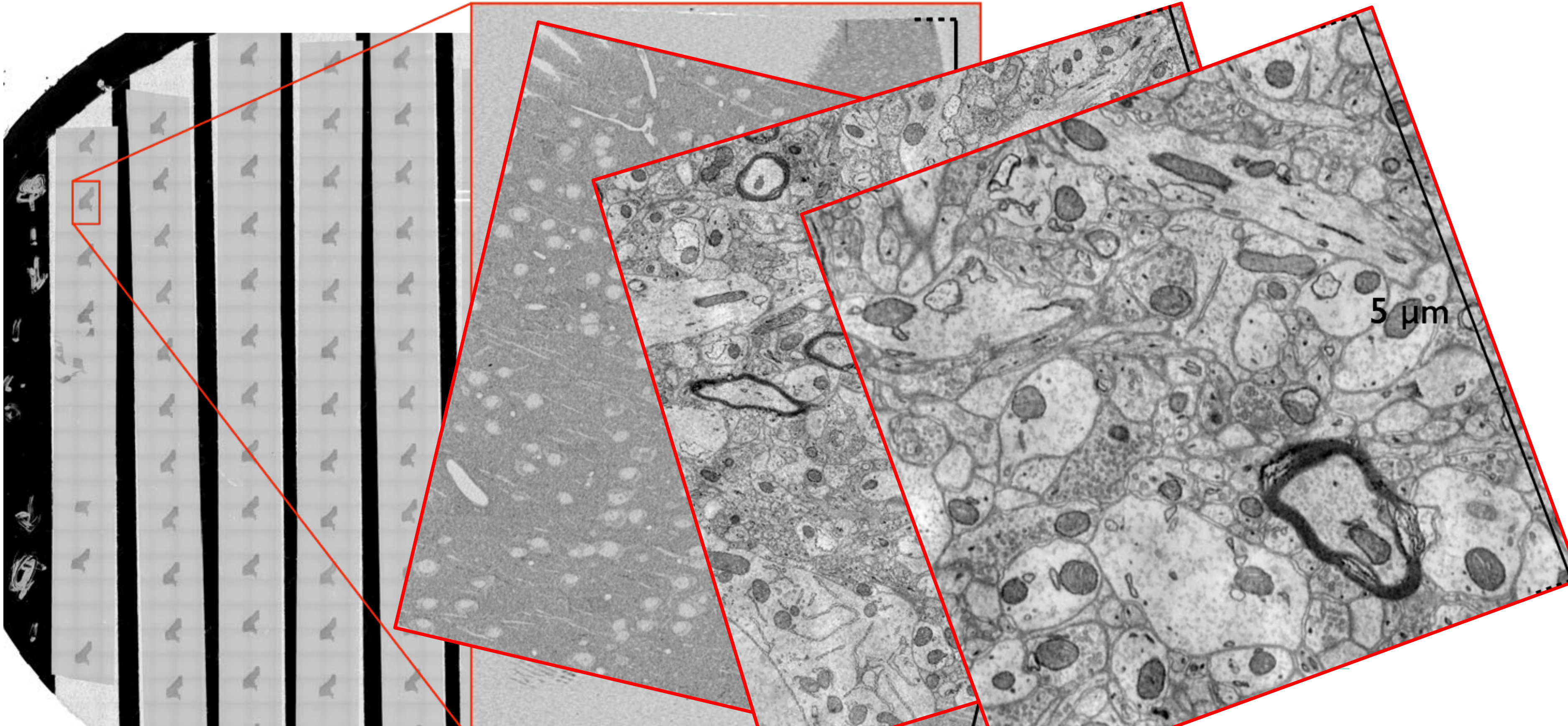
The Connectome

How is the Mammalian Brain Wired?



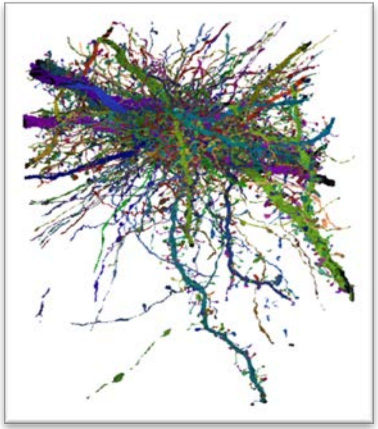

The Connectome
How is the Mammalian Brain Wired?
~60 μm³
1 Teravoxel
21,500 x 25,800 x 1,850
Bobby Kasthuri, Harvard

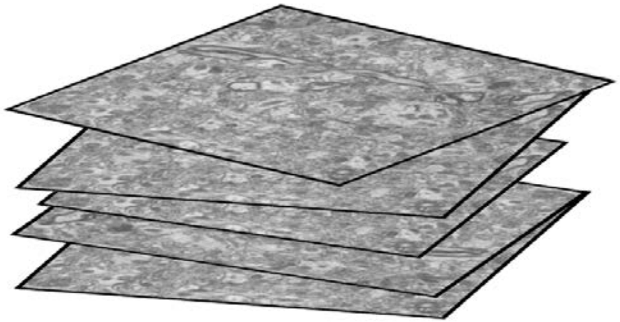
ELECTRON MICROSCOPY (EM) IMAGES

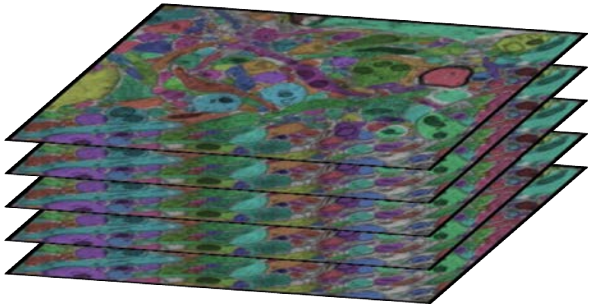


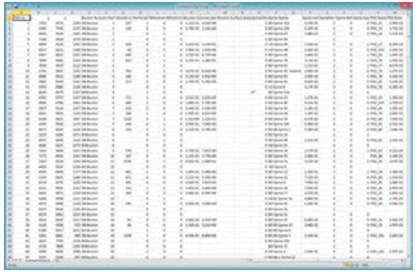
PETAVOXEL MICROSCOPY VOLUMES

- Huge amount of raw data (terabytes to petabytes)
- Takes months to years to scan, align, segment
- How to visualize and analyze this?









COURSE SCOPE

Course focus

- (Single) GPUs in standard workstations
- Scalar volume data; single time step
- But a lot applies to more general settings…

Orthogonal techniques (won't cover details)

- Parallel and distributed rendering, clusters, supercomputers, …
- Compression

RELATED BOOKS AND SURVEYS

Books

- Real-Time Volume Graphics, Engel et al., 2006
- High-Performance Visualization, Bethel et al., 2012

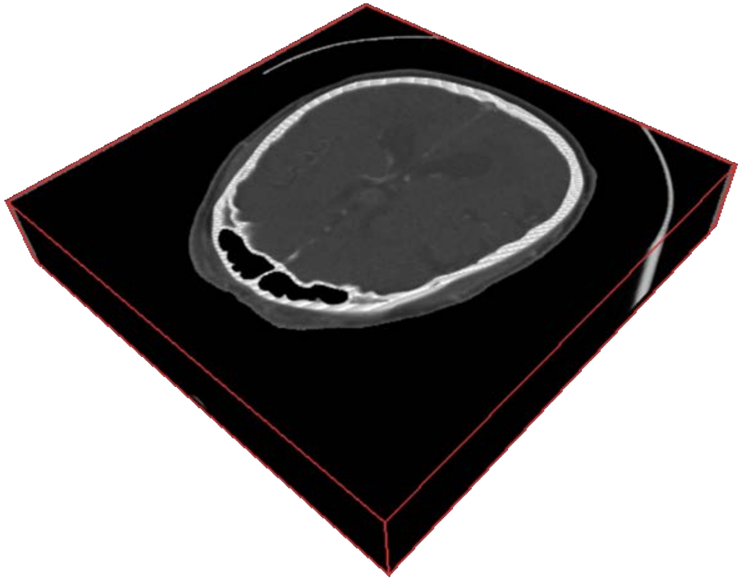
Surveys

- Parallel Visualization: Wittenbrink ’98, Bartz et al. ‘00, Zhang et al. ’05
- Real Time Interactive Massive Model Visualization: Kasik et al. ‘06
- Vis and Visual Analysis of Multifaceted Scientific Data: Kehrer and Hauser ‘13
- Compressed GPU-Based Volume Rendering: Rodriguez et al. ‘13

Fundamentals

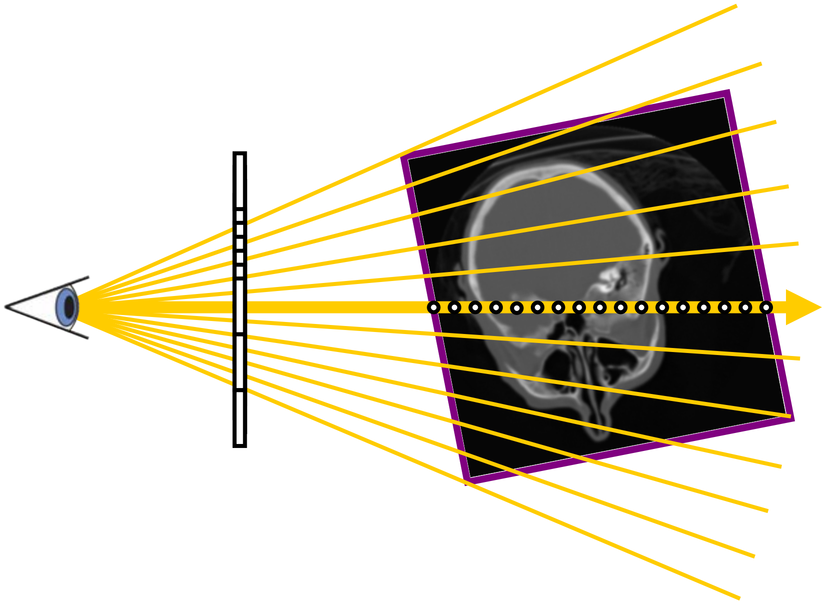
VOLUME RENDERING (1)

Assign optical properties (color, opacity) via *transfer function*



VOLUME RENDERING (2)

Ray-casting



SCALABILITY

Traditional HPC, parallel rendering definitions

- Strong scaling (“more nodes are faster for same data”)
- Weak scaling (“more nodes allow larger data”)

Our interest/definition: output sensitivity

- Running time/storage proportional to size of output instead of input
  - Computational effort scales with visible data and screen resolution
  - Working set independent of original data size

SOME TERMINOLOGY

Output-sensitive algorithms

- Standard term in (geometry) occlusion culling

Ray-guided volume rendering

- Determine working set via ray-casting
- Actual visibility; not approximate as in traditional occlusion culling

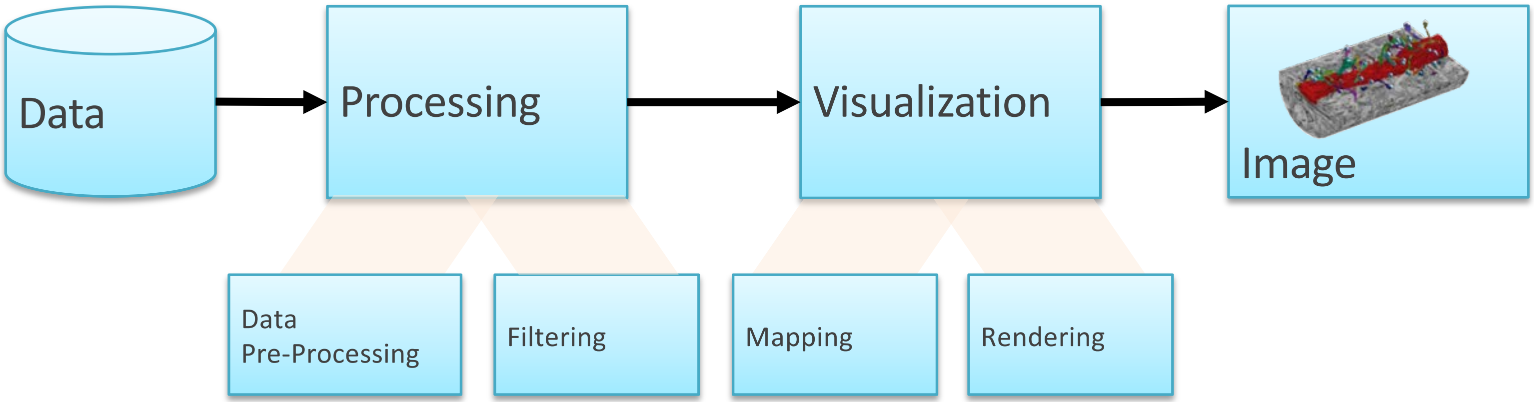
Visualization-driven pipeline

- Drive entire visualization pipeline by actual on-screen visibility

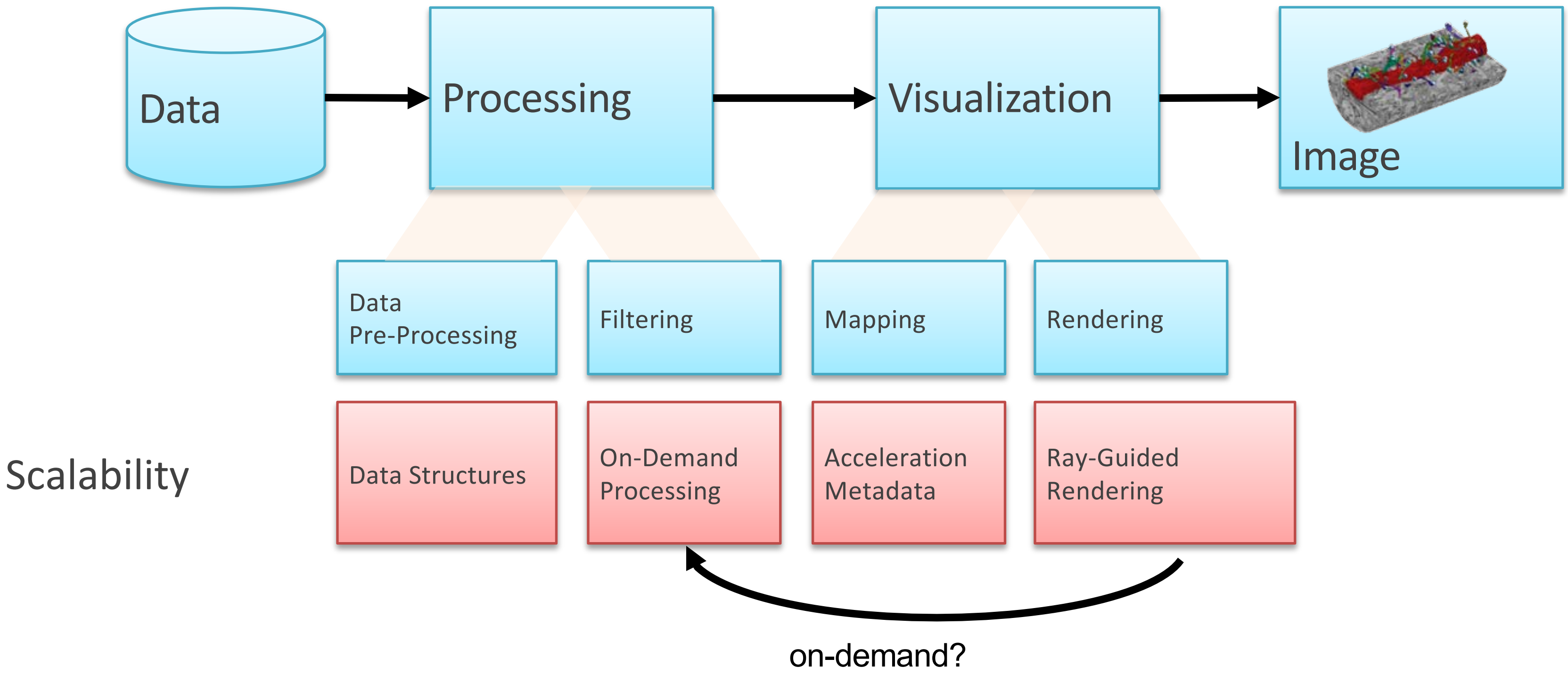
Display-aware techniques

  - Image processing, … for current on-screen resolution

LARGE-SCALE VISUALIZATION PIPELINE



LARGE-SCALE VISUALIZATION PIPELINE



Basic Scalability Issues

SCALABILITY ISSUES

Scalability issues	Scalable method
Data representation and storage	Multi-resolution data structures
	Data layout, compression
Work/data partitioning	In-core/out-of-core
	Parallel, distributed
Work/data reduction	Pre-processing
	On-demand processing
	Streaming
	In-situ visualization
	Query-based visualization

SCALABILITY ISSUES

Scalability issues	Scalable method
Data representation and storage	Multi-resolution data structures
	Data layout, compression
Work/data partitioning	In-core/out-of-core
	Parallel, distributed
Work/data reduction	Pre-processing
	On-demand processing
	Streaming
	In-situ visualization
	Query-based visualization

DATA REPRESENTATIONS

Data structure	Acceleration	Out-of-Core	Multi-Resolution
Mipmaps	-	Clipmaps	Yes
Uniform bricking	Cull bricks (linear)	Working set (bricks)	No
Hierarch. bricking	Cull bricks (hierarch.)	Working set (bricks)	Bricked mipmap
Octrees	Hierarchical traversal	Working set (subtree)	Yes (interior nodes)

Additional issues

- Data layout (linear order, Z order, …)
- Compression

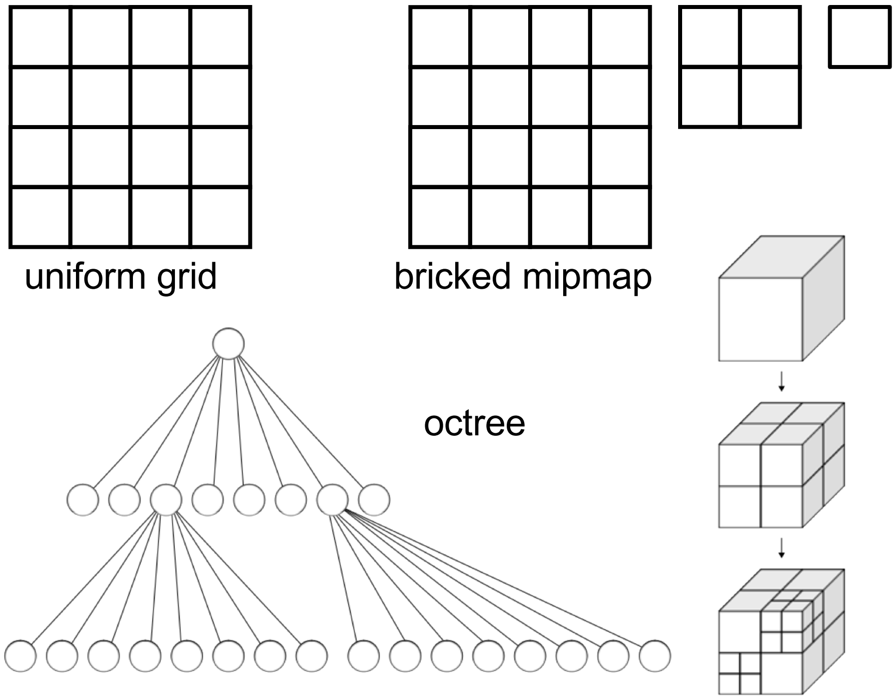
UNIFORM VS. HIERARCHICAL DECOMPOSITION

Grids

- Uniform or non-uniform

Hierarchical data structures

- Pyramid of uniform grids
  - Bricked 2D/3D mipmaps
- Tree structures
  - kd-tree, quadtree, octree

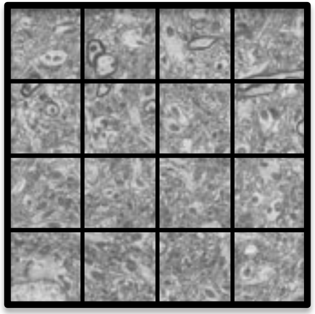


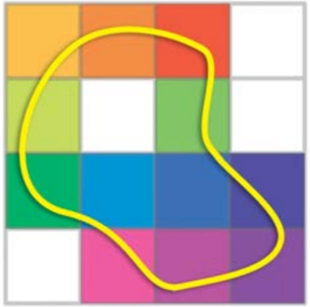
wikipedia.org

BRICKING (1)

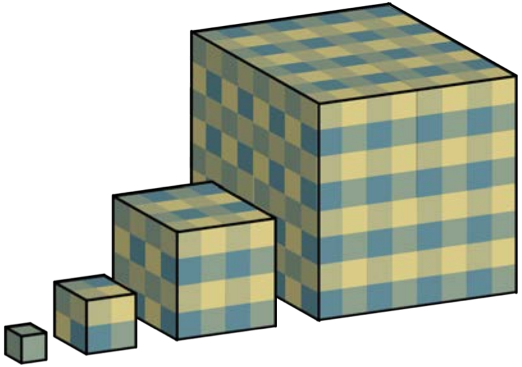
Object space (data) decomposition

- Subdivide data domain into small bricks
- Re-orders data for spatial locality
- Each brick is now one unit (culling, paging, loading, …)





BRICKING (2)



What brick size to use?

- Small bricks
  - \+ Good granularity
    (better culling efficiency, tighter working set, …)
  - \- More bricks to cull, more overhead for ghost voxels,
    one rendering pass per brick is infeasible
- Traditional out-of-core volume rendering: **large** bricks (e.g., 256^3)
- Modern out-of-core volume rendering: **small** bricks (e.g., 32^3)
  - Task-dependent brick sizes (small for rendering, large for disk/network storage)

Analysis of different brick sizes:
[Fogal et al. 2013]

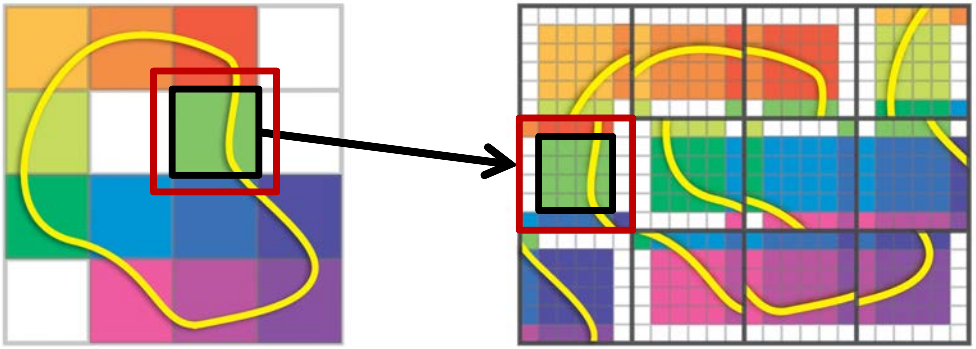
FILTERING AT BRICK BOUNDARIES

Duplicate voxels at border (ghost voxels)

- Need at least one voxel overlap
- Large overhead for small bricks

Otherwise costly filtering at brick boundary

- Except with new hardware support: *sparse textures*



PRE-COMPUTE ALL BRICKS?

Pre-computation might take very long

- Brick on demand? Brick in streaming fashion (e.g., during scanning)?

Different brick sizes for different tasks (storage, rendering)?

- Re-brick to different size on demand?
- Dynamically fix up ghost voxels?

Can also mix 2D and 3D

- E.g., 2D tiling pre-computed, but compute 3D bricks on demand

MULTI-RESOLUTION PYRAMIDS (1)

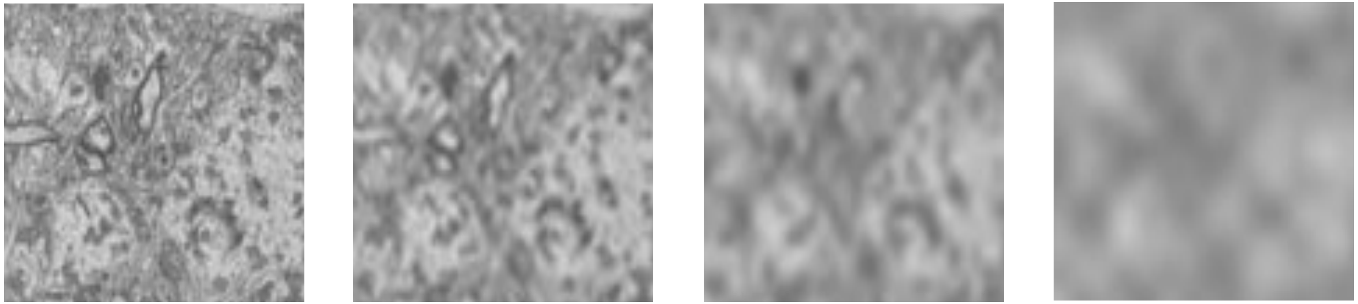
Collection of different resolution levels

- Standard: dyadic pyramids (2:1 resolution reduction)
- Can manually implement arbitrary reduction ratios

Mipmaps

- Isotropic

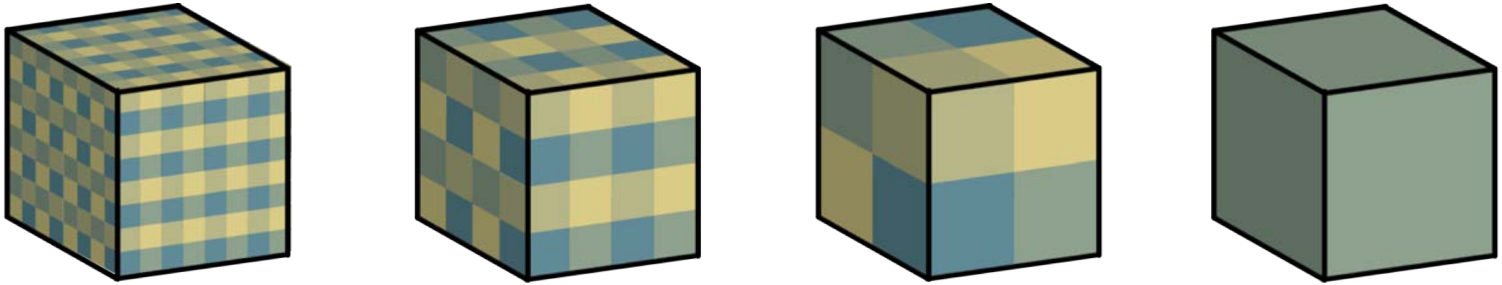
level 0 level 1 level 2 level 3



MULTI-RESOLUTION PYRAMIDS (2)

3D mipmaps

- Isotropic



level 0
(8x8x8)

level 1
(4x4x4)

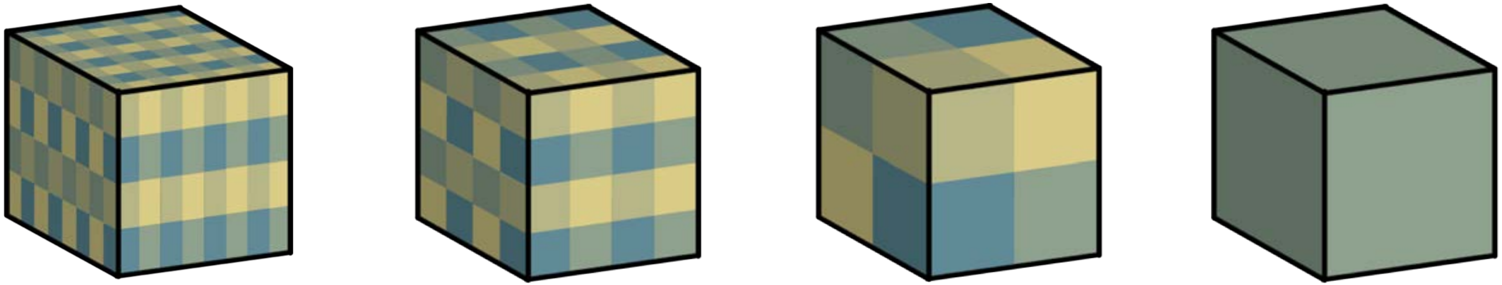
level 2
(2x2x2)

level 3
(1x1x1)

MULTI-RESOLUTION PYRAMIDS (3)

Scanned volume data are often anisotropic

- Reduce resolution anisotropically to reach isotropy



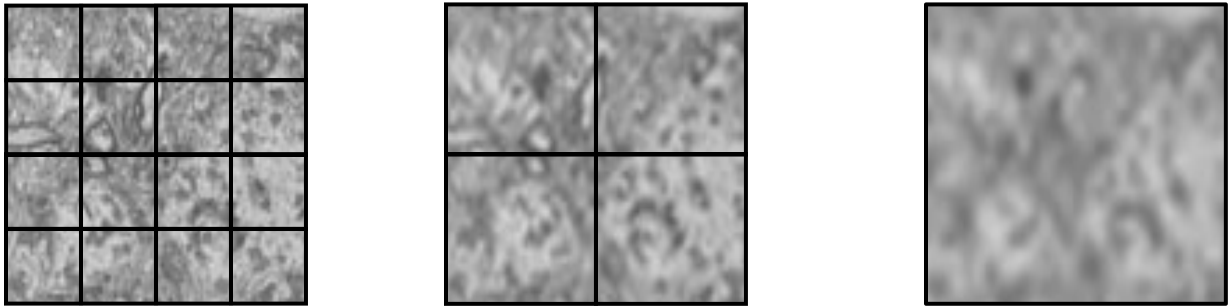
level 0 (8x8x4) | level 1 (4x4x4) | level 2 (2x2x2) | level 3 (1x1x1)

BRICKING MULTI-RESOLUTION PYRAMIDS (1)

Sponsored by

Each level is bricked individually

- Use same brick resolution (# voxels) in each level



level 0

level 1

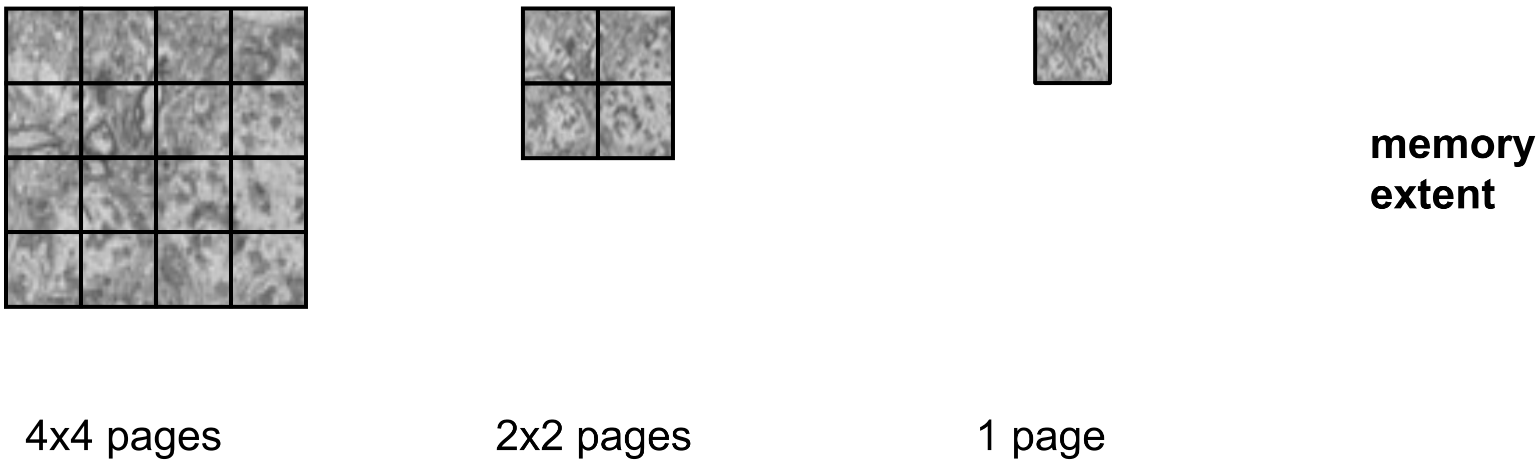
level 2

spatial extent

BRICKING MULTI-RESOLUTION PYRAMIDS (2)

Virtual memory: Each brick will be a "page"

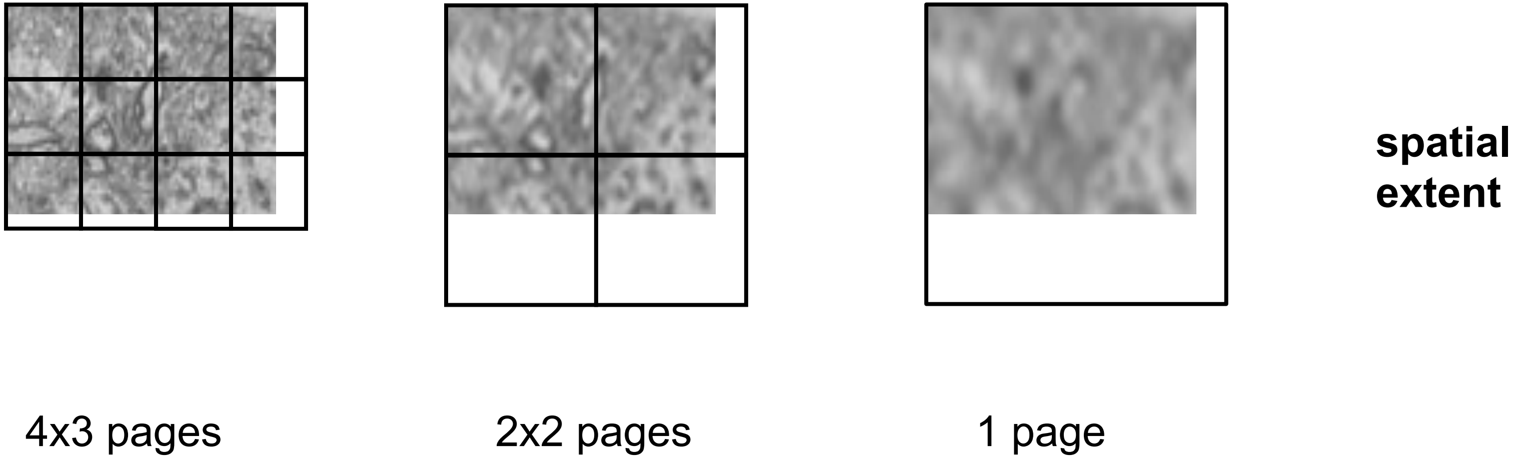
- "Multi-resolution virtual memory": every page lives in some resolution level



BRICKING MULTI-RESOLUTION PYRAMIDS (3)

Beware of aspect ratio and partially filled pages

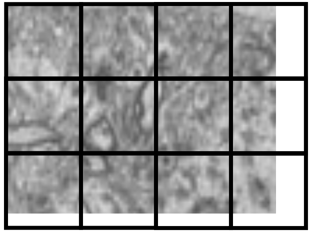
- Reduce total resolution in voxels; compute number of pages (ceil); iterate



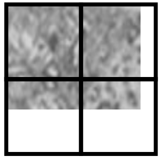
BRICKING MULTI-RESOLUTION PYRAMIDS (3)

Beware of aspect ratio and partially filled pages

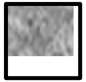
- Reduce total resolution in voxels; compute number of pages (ceil); iterate



4x3 pages



2x2 pages



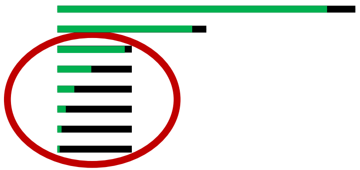
1 page

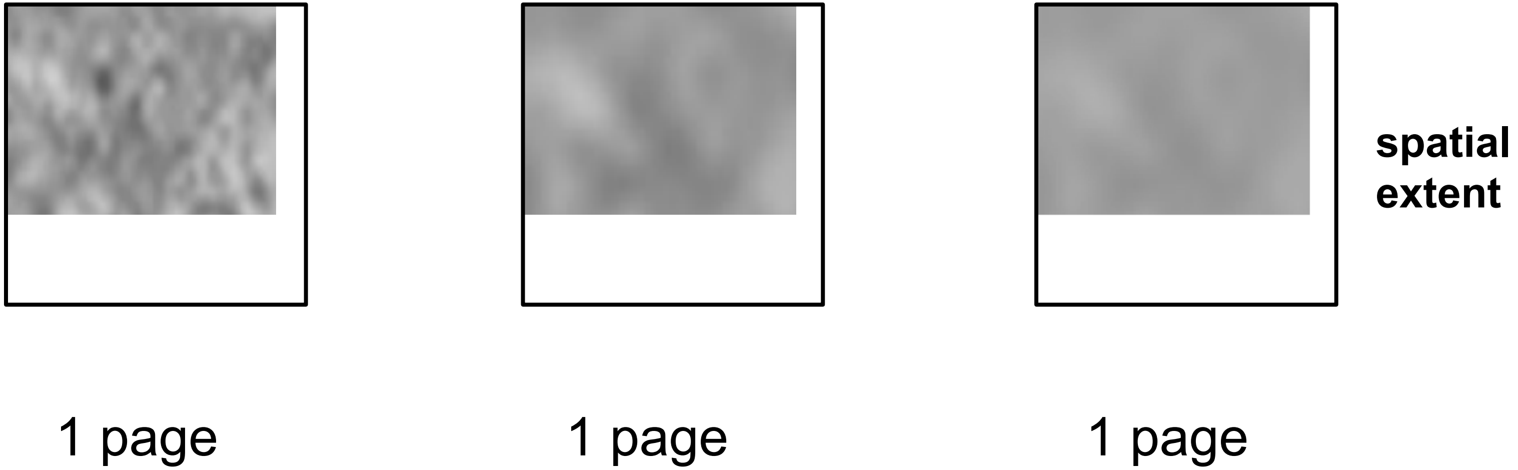
memory extent

BRICKING MULTI-RESOLUTION PYRAMIDS (4)

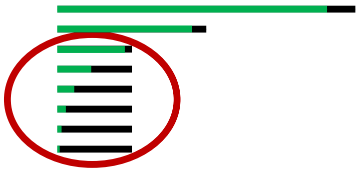
Tail of pyramid

- Below size of single page; can cut off early



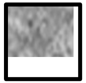


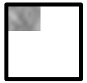
BRICKING MULTI-RESOLUTION PYRAMIDS (4)

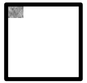


Tail of pyramid

- Below size of single page; can cut off early







- **`GL_ARB_sparse_texture`** treats tail as single unit of residency (implementation-dependent definition of tail !)

memory extent

1 page

1 page

1 page

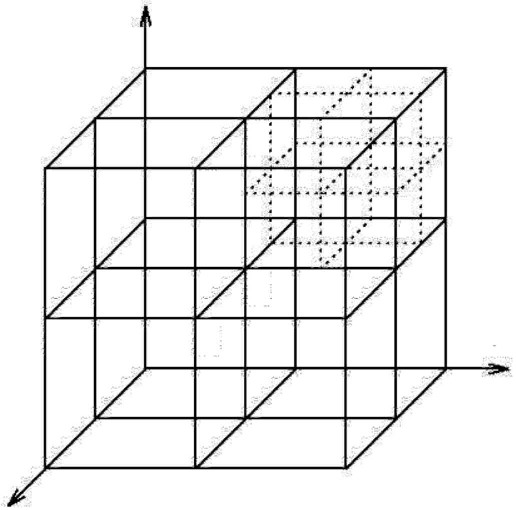
OCTREES FOR VOLUME RENDERING (1)

Multi-resolution

- Adapt resolution of data to screen resolution
    - Reduce aliasing
    - Limit amount of data needed

Acceleration

- Hierarchical empty space skipping
- Start traversal at root
  (but different optimized traversal algorithms:
  kd-restart, kd-shortstack, etc.)



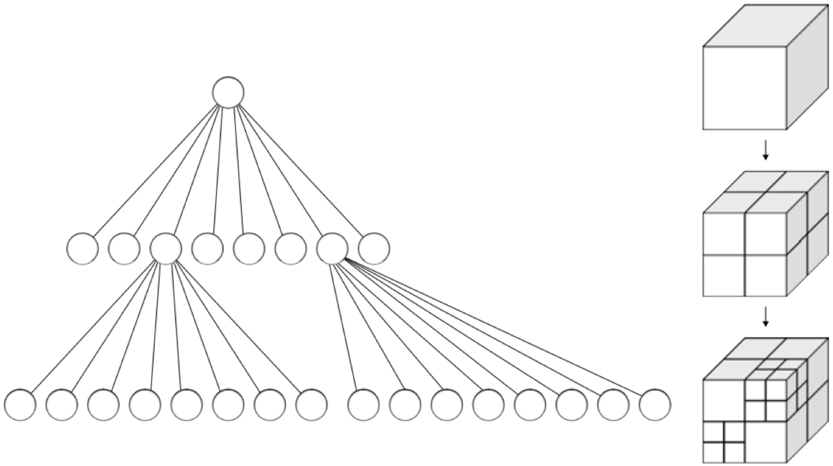
OCTREES FOR VOLUME RENDERING (2)

Representation

- Full octree
  - Every octant in every resolution level
- Sparse octree
  - Do not store voxel data of empty nodes

Data structure

- Pointer-based
  - Parent node stores pointer(s) to children
- Pointerless
  - Array to index full octree directly



wikipedia.org

SCALABILITY ISSUES

Scalability issues	Scalable method
Data representation and storage	Multi-resolution data structures
	Data layout, compression
Work/data partitioning	In-core/out-of-core
	Parallel, distributed
Work/data reduction	Pre-processing
	On-demand processing
	Streaming
	In-situ visualization
	Query-based visualization

WORK/DATA PARTITIONING

- Out-of-core techniques
- Domain decomposition
- Parallel and distributed rendering

OUT-OF-CORE TECHNIQUES (1)

Sponsored by

Data too large for GPU memory

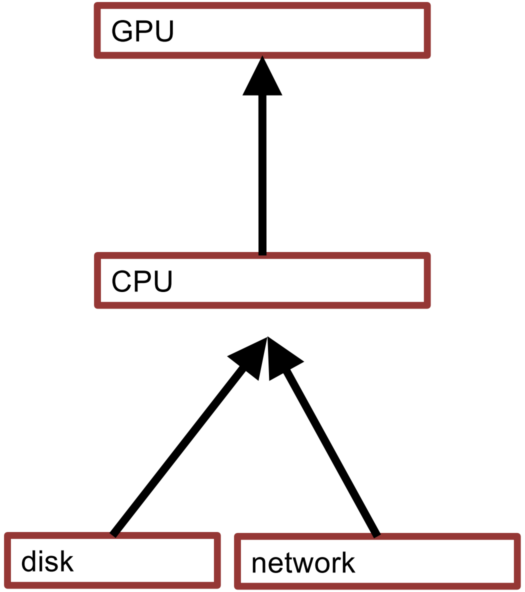
- Stream volume bricks from CPU to GPU on demand

Data too large for CPU memory

- Stream volume bricks from disk on demand

Data too large for local disk storage

- Stream volume bricks from network storage



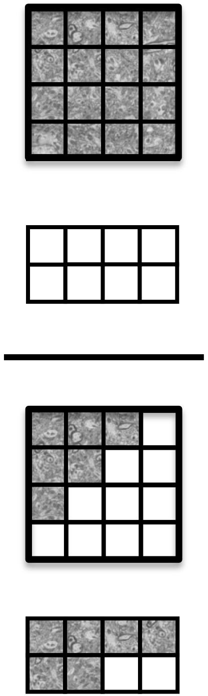
OUT-OF-CORE TECHNIQUES (2)

Preparation

- Subdivide spatial domain
  - May also be done “virtually”, i.e., data re-ordering may be delayed
- Allocate cache memory (e.g., large 3D cache texture)

Run-Time

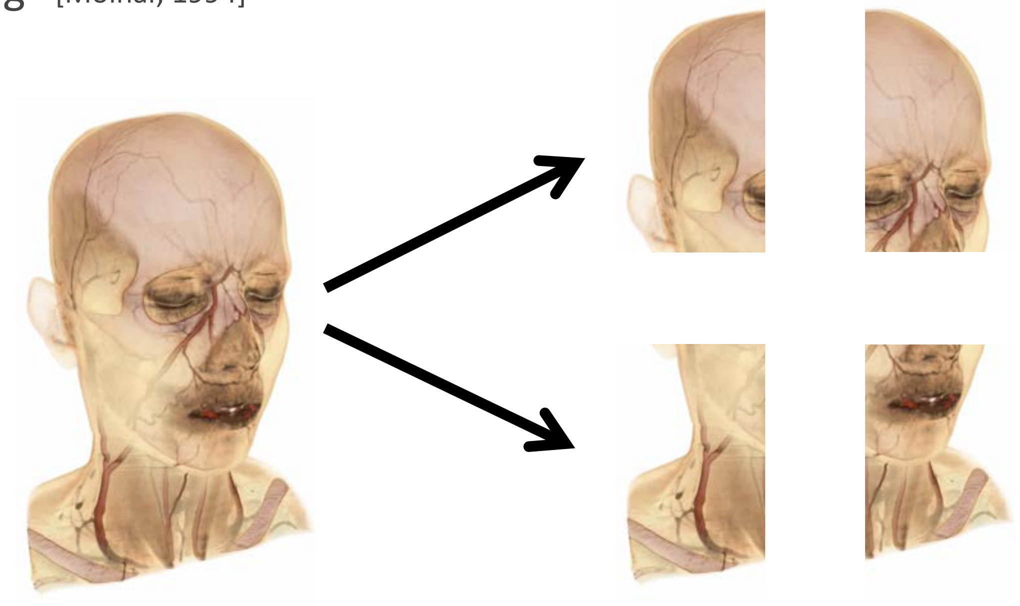
- Determine **working set**
- Page working set into cache memory
- Render from cache memory



DOMAIN DECOMPOSITION (1)

Subdivide image domain (image space)

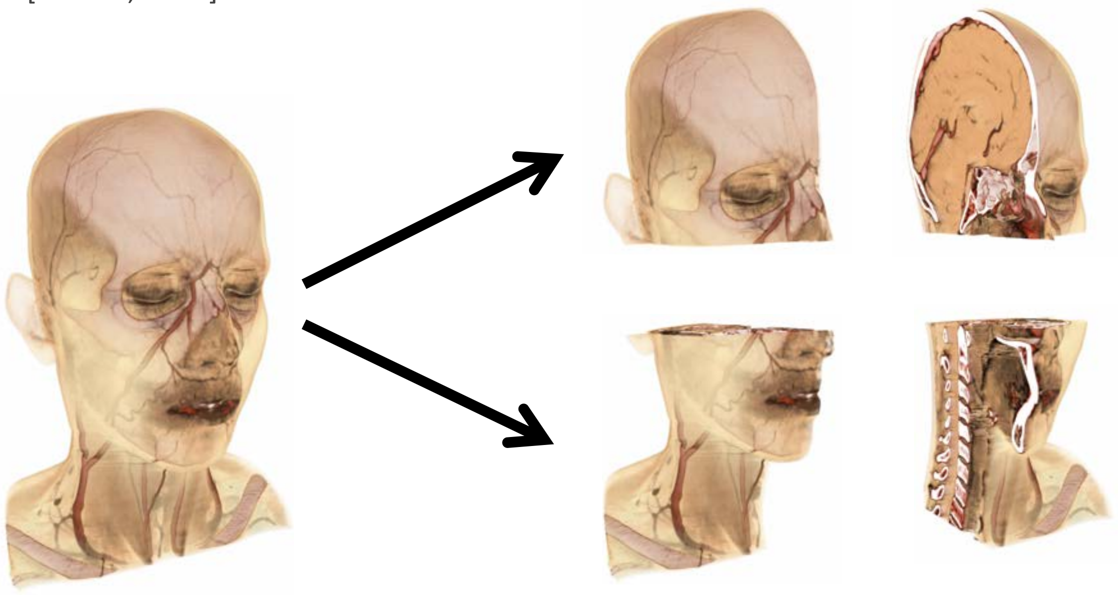
- “Sort-first rendering” [Molnar, 1994]
- View-dependent



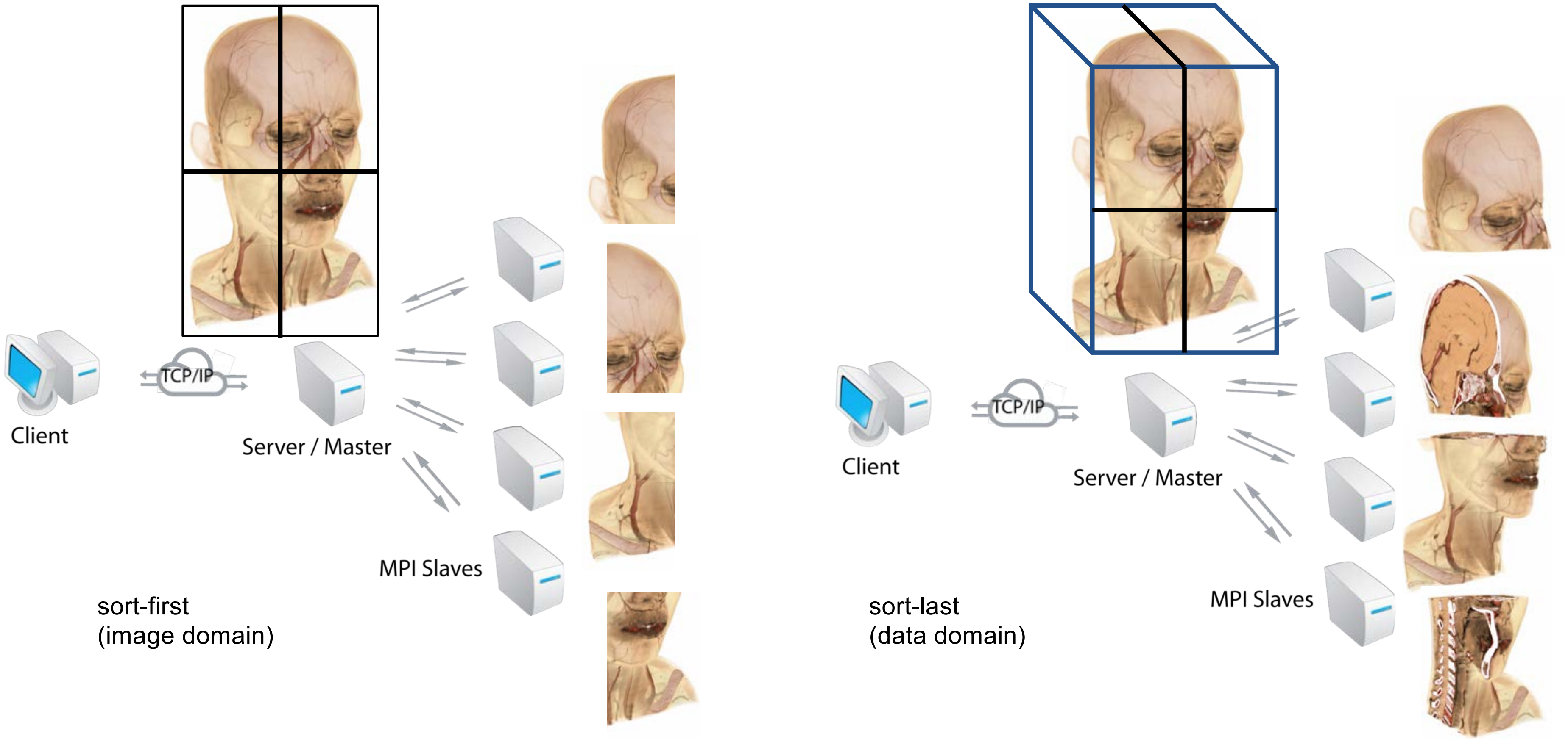
DOMAIN DECOMPOSITION (2)

Subdivide data domain (object space)

- "Sort-last rendering" [Molnar, 1994]
- View-independent



SORT-FIRST VS. SORT-LAST



SCALABILITY ISSUES

Scalability issues	Scalable method
Data representation and storage	Multi-resolution data structures
	Data layout, compression
Work/data partitioning	In-core/out-of-core
	Parallel, distributed
Work/data reduction	Pre-processing
	On-demand processing
	Streaming
	In-situ visualization
	Query-based visualization

ON-DEMAND PROCESSING

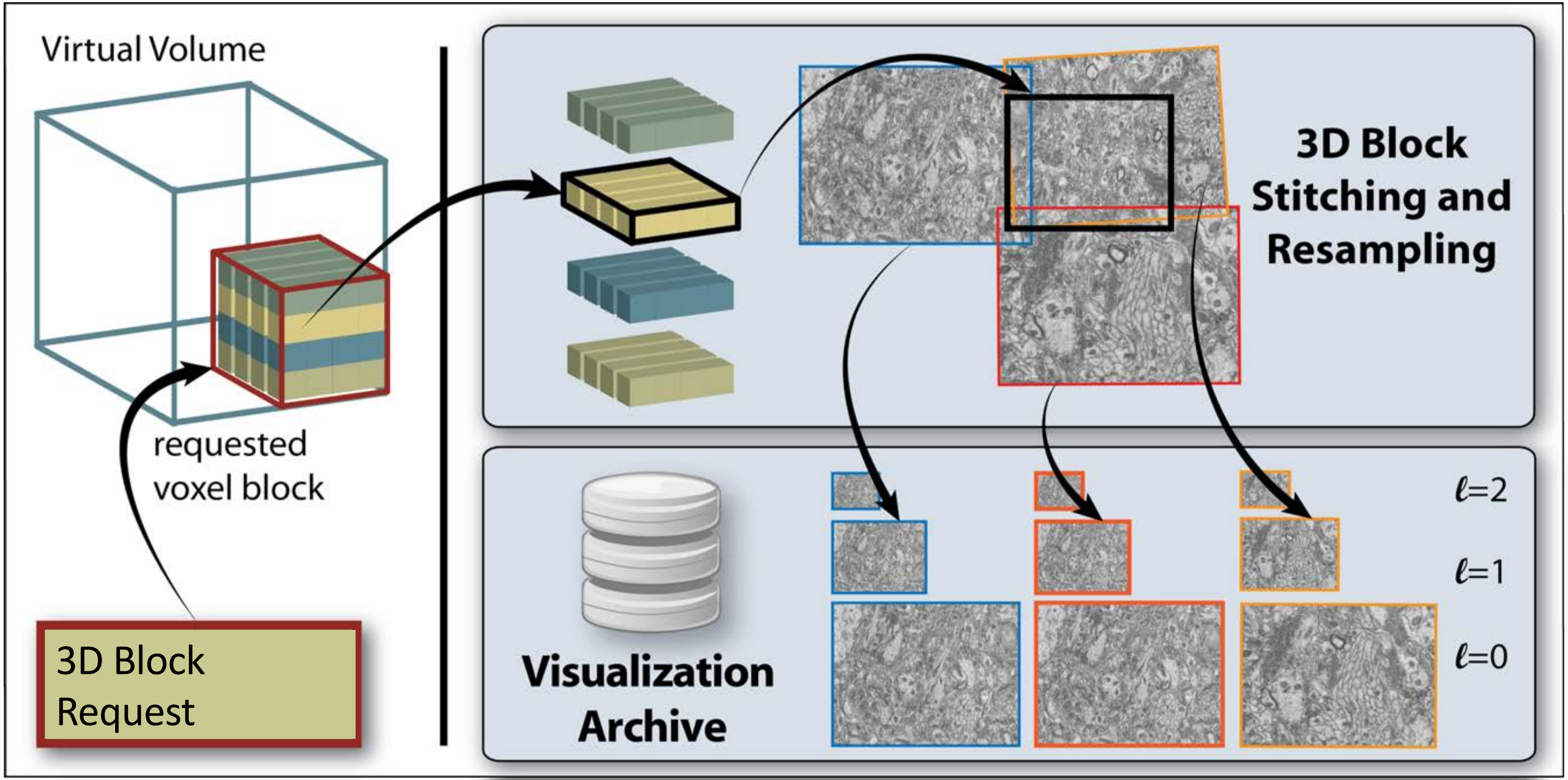
First determine what is visible / needed

Then process only this working set

- Basic processing
  - Noise removal and edge detection
  - Registration and alignment
  - Segmentation, …
- Basic data structure building
  - Construct pages/bricks/octree nodes only on demand?

EXAMPLE: 3D BRICK CONSTRUCTION FROM 2D EM STREAMS

[Hadwiger et al., IEEE Vis 2012]



EXAMPLE: DENOISING & EDGE ENHANCEMENT

Edge enhancement for EM data

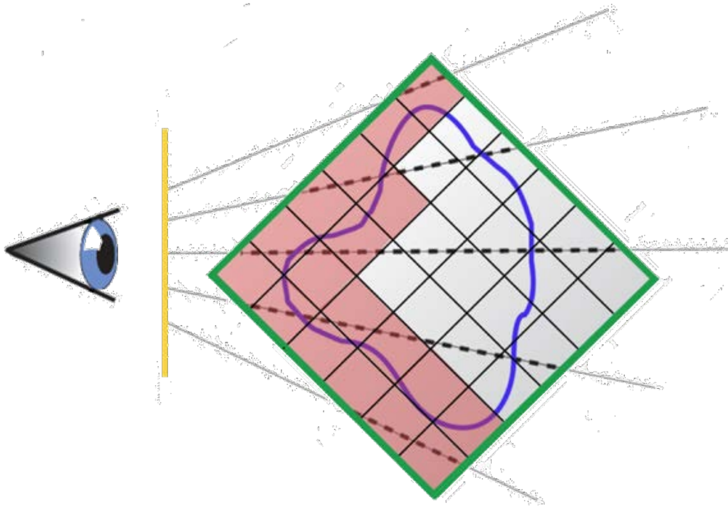
Caching scheme

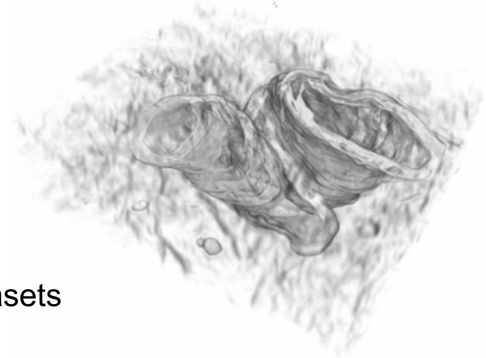
- Process only currently visible bricks
- Cache result for re-use

GPU Implementation

- CUDA and shared memory for fast computation

Different noise removal and filtering algorithms

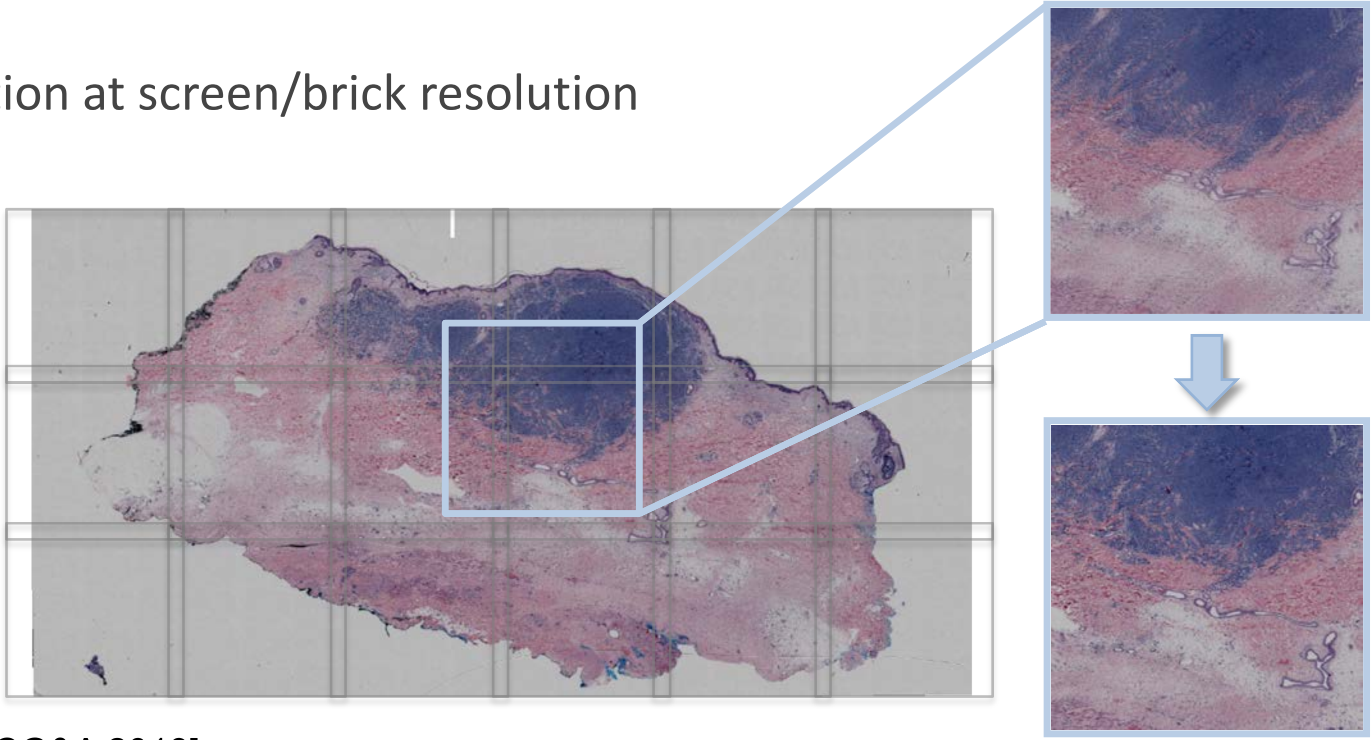




[Jeong et al., IEEE Vis 2009]
Scalable and Interactive Segmentation and Visualization of Neural Processes in EM Datasets

EXAMPLE: REGISTRATION & ALIGNMENT

Registration at screen/brick resolution



[Beyer et al., CG&A 2013]
Exploring the Connectome – Petascale Volume Visualization of Microscopy Data Streams

Part 2 - Scalable Volume Visualization Architectures and Applications

PART 2 – SCALABLE ARCHITECTURES & APPLICATIONS

History

Categorization

- Working Set Determination
- Working Set Storage & Access
- Rendering (Ray Traversal)

Ray-Guided Volume Rendering Examples

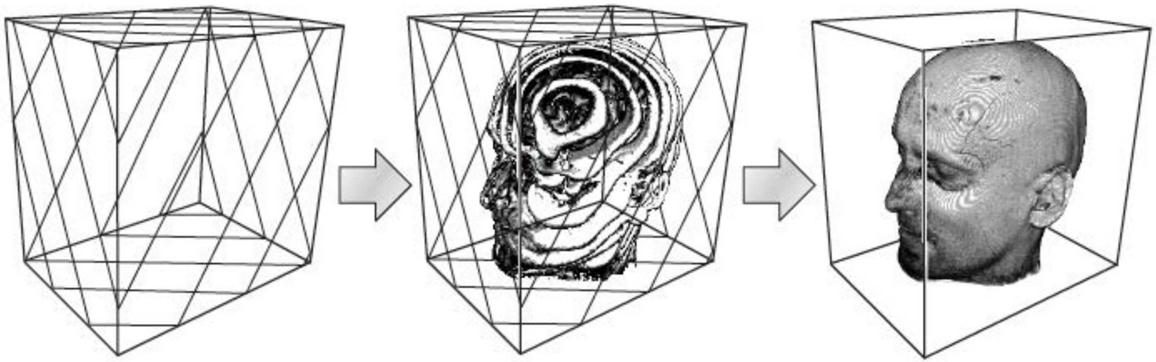
Summary

HISTORY (1)

Texture slicing [Cullip and Neumann ’93, Cabral et al. ’94, Rezk-Salama et al. ‘00]

+ Minimal hardware requirements (can run on WebGL)

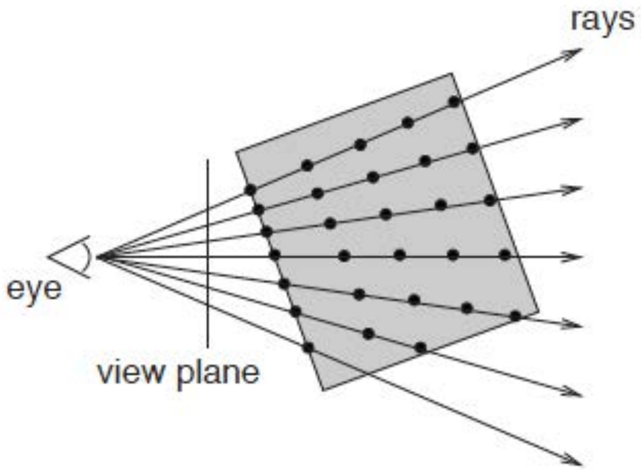
- Visual artifacts, less flexibility



HISTORY (2)

GPU ray-casting [Röttger et al. ‘03, Krüger and Westermann ‘03]

+ standard image order approach, embarrassingly parallel
+ supports many performance and quality enhancements



HISTORY (3)

Large data volume rendering

- Octree rendering based on texture-slicing
  [LaMar et al. '99, Weiler et al. '00, Guthe et al. '02]
- Bricked single-pass ray-casting
  [Hadwiger et al. '05, Beyer et al. '07]
- Bricked multi-resolution single-pass ray-casting
  [Ljung et al. '06, Beyer et al. '08, Jeong et al. '09]
- Optimized CPU ray-casting [Knoll et al. '11]

Examples

Octree Rendering and Texture Slicing

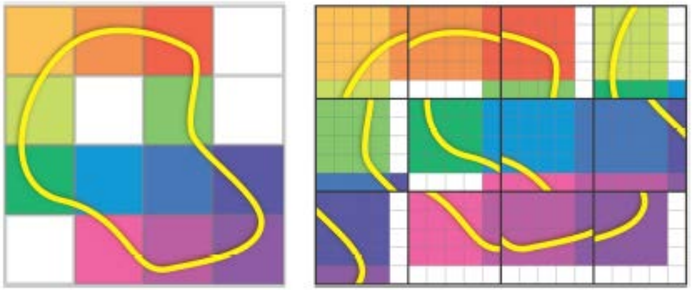
- GPU 3D texture mapping with arbitrary levels of detail
- Consistent interpolation between adjacent resolution levels
- Adapting slice distance with respect to desired LOD (needs opacity correction)
- LOD based on user-defined focus point

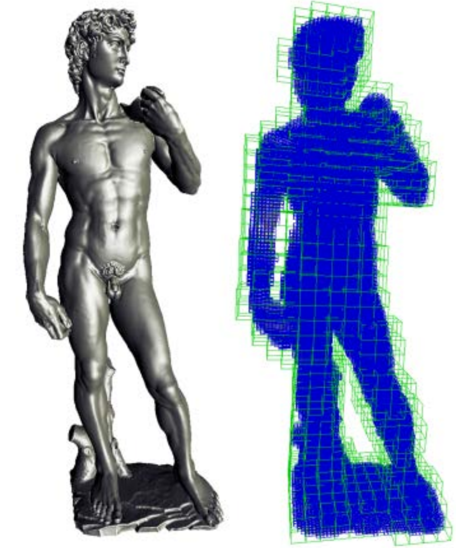
[Weiler et al., IEEE Symp. Vol Vis 2000]
Level-Of-Detail Volume Rendering via 3D Textures

Working set determination:	View frustum
Volume representation:	Octree
Rendering:	CPU octree traversal, texture slicing

Bricked Single-Pass Ray-Casting

- 3D brick cache for out-of-core volume rendering
- Object space culling and empty space skipping in ray setup step
- Correct tri-linear interpolation between bricks





[Hadwiger et al., Eurographics 2005]
Real-Time Ray-Casting and Advanced Shading of Discrete Isosurfaces

Working set determination:	Global, view frustum
Volume representation:	Single-resolution grid (page table)
Rendering:	Bricked single-pass ray-casting

Bricked Multi-Resolution Ray-Casting

- Adaptive object- and image-space sampling
  - Adaptive sampling density along ray
  - Adaptive image-space sampling, based on statistics for screen tiles
- Single-pass fragment program
  - Correct neighborhood samples for interpolation fetched in shader
- Transfer function-based LOD selection

[Ljung, Volume Graphics 2006]
Adaptive Sampling in Single Pass, GPU-based Raycasting of Multiresolution Volumes

Working set determination:	Global, view frustum
Volume representation:	Multi-resolution grid
Rendering:	Bricked single-pass ray-casting

CATEGORIZATION

Main questions

- Q1: How is the working set determined?
- Q2: How is the working set stored?
- Q3: How is the rendering done?

Huge difference between ‘traditional’ and ‘modern’ ray-guided approaches!

CATEGORIZATION

Working set determination	Full volume	Basic culling (global attributes, view frustum)		Ray-guided / visualization-driven
Volume data representation	- Linear (non-bricked)	- Single-resolution grid - Grid with octree per brick	- Octree - Kd-tree - Multi-resolution grid	- Octree - Multi-resolution grid
Rendering (ray traversal)	- Texture slicing - Non-bricked ray-casting	- CPU octree traversal (multi-pass) - CPU kd-tree traversal (multi-pass) - Bricked/virtual texture ray-casting (single-pass)		- GPU octree traversal (single-pass) - Multi-level virtual texture ray-casting (single-pass)
Scalability	Low	Medium		High

Q1: WORKING SET DETERMINATION – TRADITIONAL

Global attribute-based culling (view-independent)

- Cull against transfer function, iso value, enabled objects, etc.

View frustum culling (view-dependent)

- Cull bricks outside the view frustum

Occlusion culling?

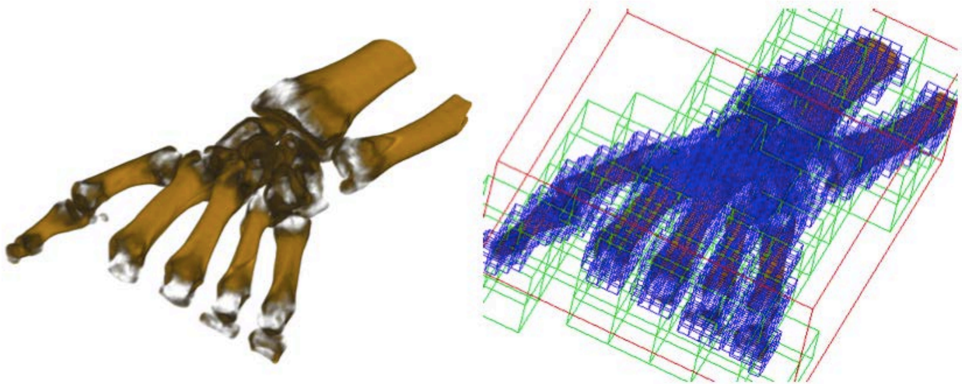
SIGGRAPH ASIA 2018 TOKYO

Sponsored by

GLOBAL ATTRIBUTE-BASED CULLING

Cull bricks based on attributes; view-independent

- Transfer function
- Iso value
- Enabled segmented objects

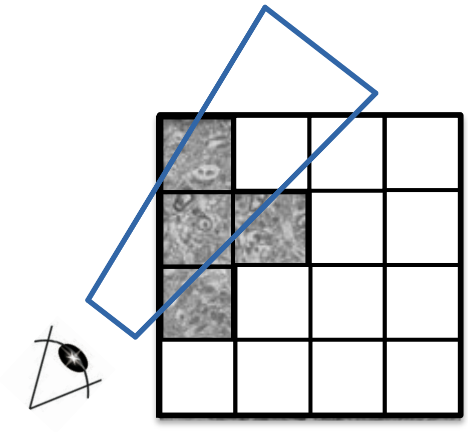


Often based on min/max bricks

- Empty space skipping
- Skip loading of 'empty' bricks
- Speed up on-demand spatial queries

VIEW FRUSTUM, OCCLUSION CULLING

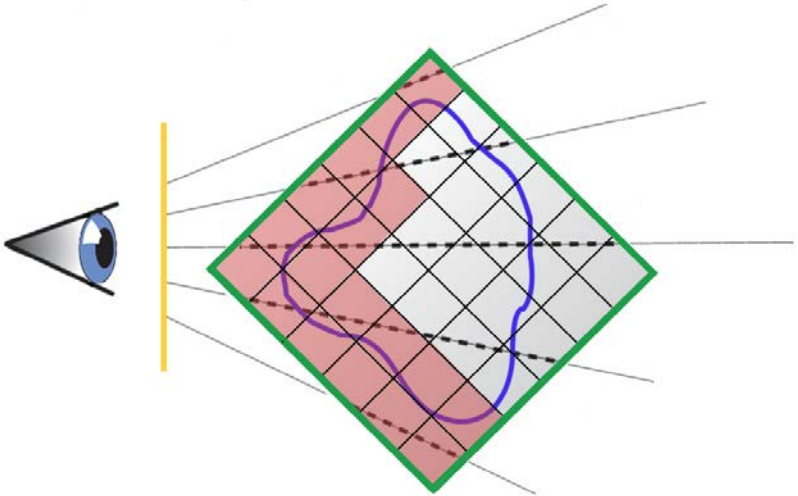
- Cull all bricks against view frustum
- Cull all occluded bricks



Q1: WORKING SET DETERMINATION – MODERN (1)

Visibility determined during ray traversal

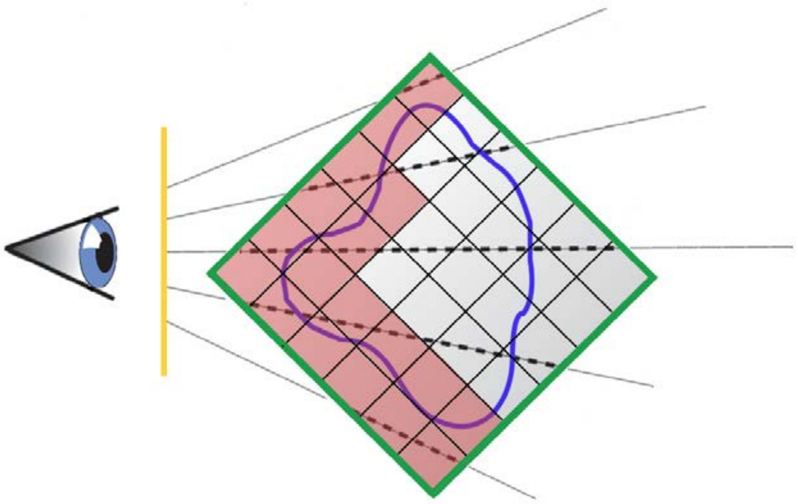
- Implicit view frustum culling (no extra step required)
- Implicit occlusion culling (no extra steps or occlusion buffers)



Q1: WORKING SET DETERMINATION – MODERN (2)

Rays determine working set directly

- Each ray writes out list of bricks it requires (intersects) front-to-back
- Use modern OpenGL extensions (`GL_ARB_shader_storage_buffer_object`, …)



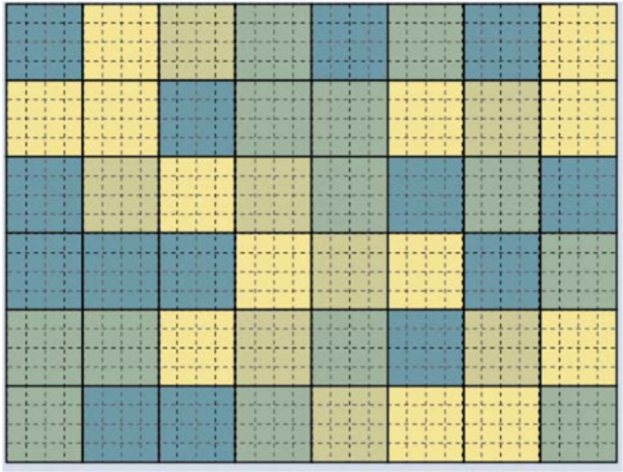
Q2: WORKING SET STORAGE - TRADITIONAL

Different possibilities:

- Individual texture for each brick
  - OpenGL-managed 3D textures (paging done by OpenGL)
  - Pool of brick textures (paging done manually)
- Multiple bricks combined into single texture
  - Need to adjust texture coordinates for each brick

Q2: WORKING SET STORAGE – MODERN (1)

Shared cache texture for all bricks ("brick pool")



Q2: WORKING SET STORAGE – MODERN (2)

Caching Strategies

- LRU, MRU

Handling missing bricks

- Skip or substitute lower resolution

Strategies if the working set is too large

- Switch from single-pass to multi-pass rendering
- Interrupt rendering on cache miss (“page fault handling”)

Q3: RENDERING - TRADITIONAL

Traverse bricks in front-to-back visibility order

- Order determined on CPU
- Easy to do for grids and trees (recursive)

Render each brick individually

- One rendering pass per brick

Traditional problems

- When to stop? (early ray termination vs. occlusion culling)
- Occlusion culling of each brick usually too conservative

Q3: RENDERING - MODERN

Preferably single-pass rendering

All rays traversed in front-to-back order

Rays perform dynamic address translation (virtual to physical)

Rays dynamically write out brick usage information

- Missing bricks (“cache misses”)
- Bricks in use (for replacement strategy: LRU/MRU)

Rays dynamically determine required resolution

- Per-sample or per-brick

VIRTUAL TEXTURING

Similar to CPU virtual memory but in 2D/3D texture space

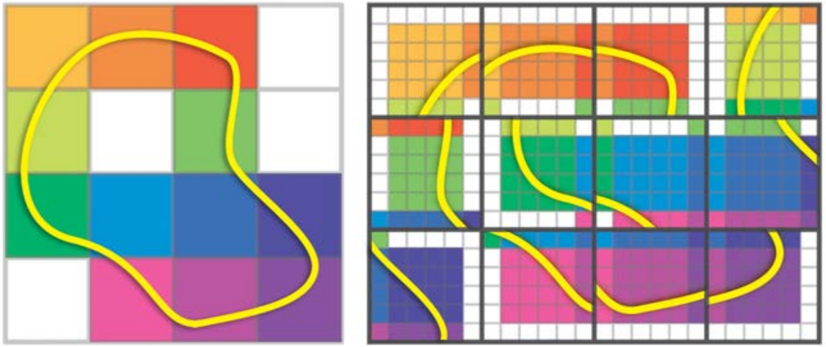
- Domain decomposition of virtual texture space: pages
- Page table maps from virtual pages to physical pages
- Working set of physical pages stored in cache texture

[Kraus and Ertl, Graphics Hardware ’02]
Adaptive Texture Maps

VIRTUAL TEXTURING

Similar to CPU virtual memory but in 2D/3D texture space

- Domain decomposition of virtual texture space: pages
- Page table maps from virtual pages to physical pages
- Working set of physical pages stored in cache texture

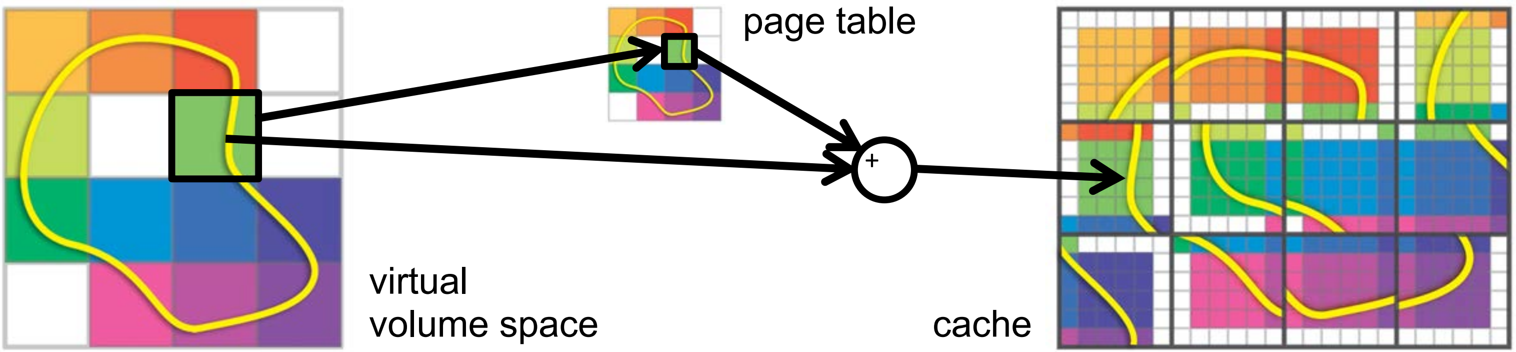


[Hadwiger et al., Eurographics ’05]
Real-Time Ray-Casting and Advanced Shading of Discrete Isosurfaces

ADDRESS TRANSLATION

Map virtual to physical address

```
pt_entry = pageTable[ virtAddx / brickSize ];
physAddx = pt_entry.physAddx + virtAddx % brickSize;
```



If cache is a texture, need to transform coordinates to texture domain (scale factor)!

ADDRESS TRANSLATION VARIANTS

Tree (quadtree/octree)

- Linked nodes; dynamic traversal

Uniform page tables

- Can do page table mipmap; uniform in each level

Multi-level page tables

- Recursive page structure decoupled from multi-resolution hierarchy

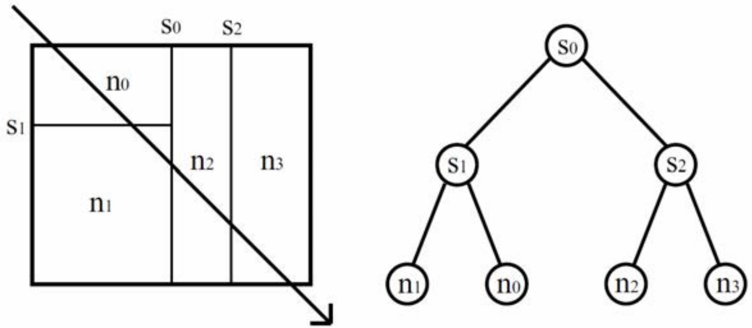
Spatial hashing

- Needs collision handling; hashing function must minimize collisions

TREE TRAVERSAL

Adapt tree traversal from ray tracing

- Standard traversal: recursive with stack
- GPU algorithms without or with limited stack
  - Use "ropes" between nodes [Havran et al. '98, Gobbetti et al. '08]
  - kd-restart, kd-shortstack [Foley and Sugerman '05]

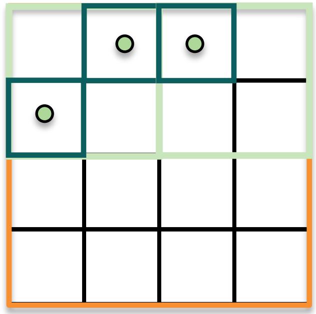


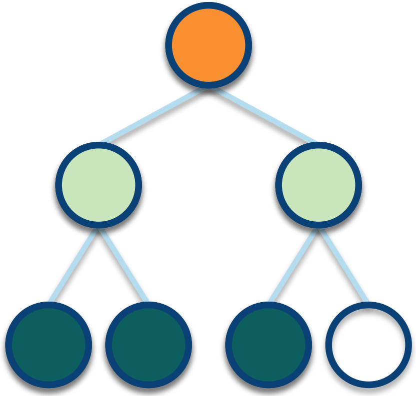
courtesy Foley and Sugerman

VARIANT 1: TREE TRAVERSAL

Tree can be seen as a 'page table'

- Linked nodes; dynamic traversal
- Nodes contain page table entries



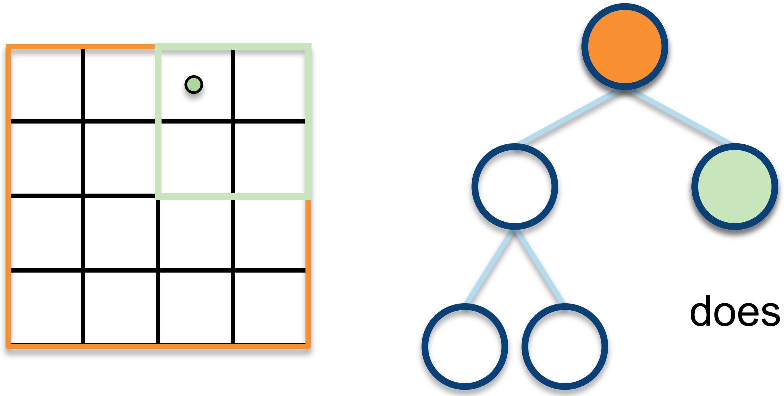


"page table hierarchy" (tree) coupled to resolution hierarchy!

VARIANT 1: TREE TRAVERSAL

Tree can be seen as a 'page table'

- Linked nodes; dynamic traversal
- Nodes contain page table entries

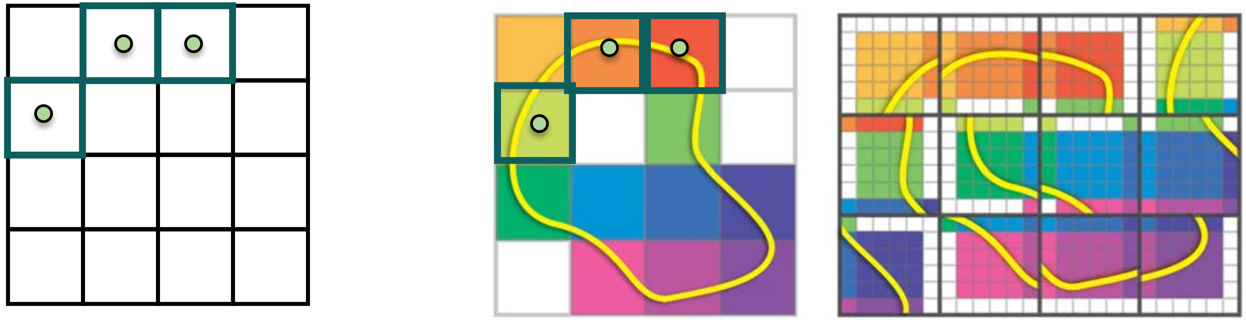


does not require full tree!

VARIANT 2: UNIFORM PAGE TABLES

Only feasible when page table is not too large (depends on brick size)

- For “medium-sized” volumes or “large” page/brick sizes



requires full-size page table!

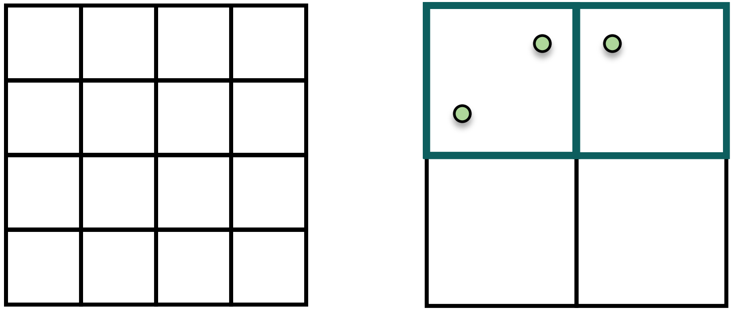
VARIANT 2: UNIFORM PAGE TABLES

Only feasible when page table is not too large (depends on brick size)

- For “medium-sized” volumes or “large” page/brick sizes

Can do page table for each resolution level (page table mipmap)

- Uniform in each level



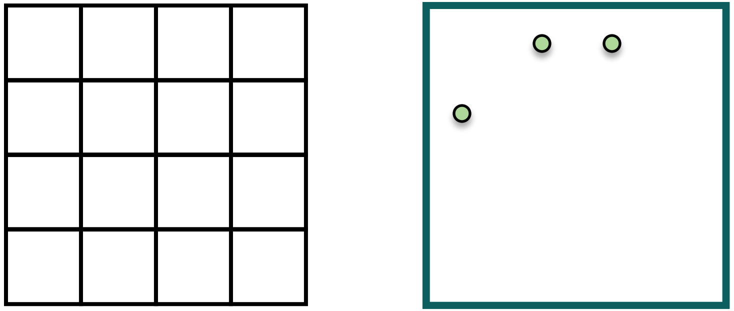
VARIANT 2: UNIFORM PAGE TABLES

Only feasible when page table is not too large (depends on brick size)

- For “medium-sized” volumes or “large” page/brick sizes

Can do page table for each resolution level (page table mipmap)

- Uniform in each level



VARIANT 2B: HARDWARE PAGE TABLES

- Uniform page tables (mipmaps) managed in hardware
  - Query for page residency in fragment shader
  - Fragment shader decides how to handle missing pages

- OpenGL sparse textures (`GL_ARB_sparse_texture`)

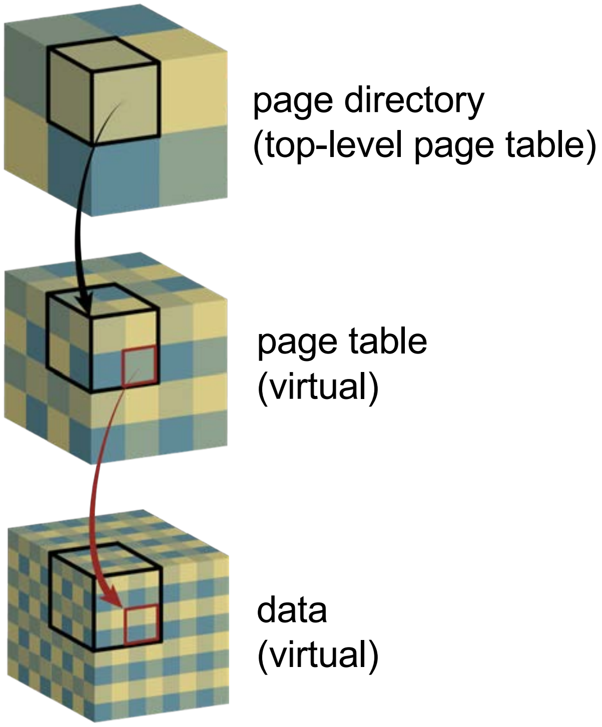
VARIANT 3: MULTI-LEVEL PAGE TABLES

Virtualize page tables recursively

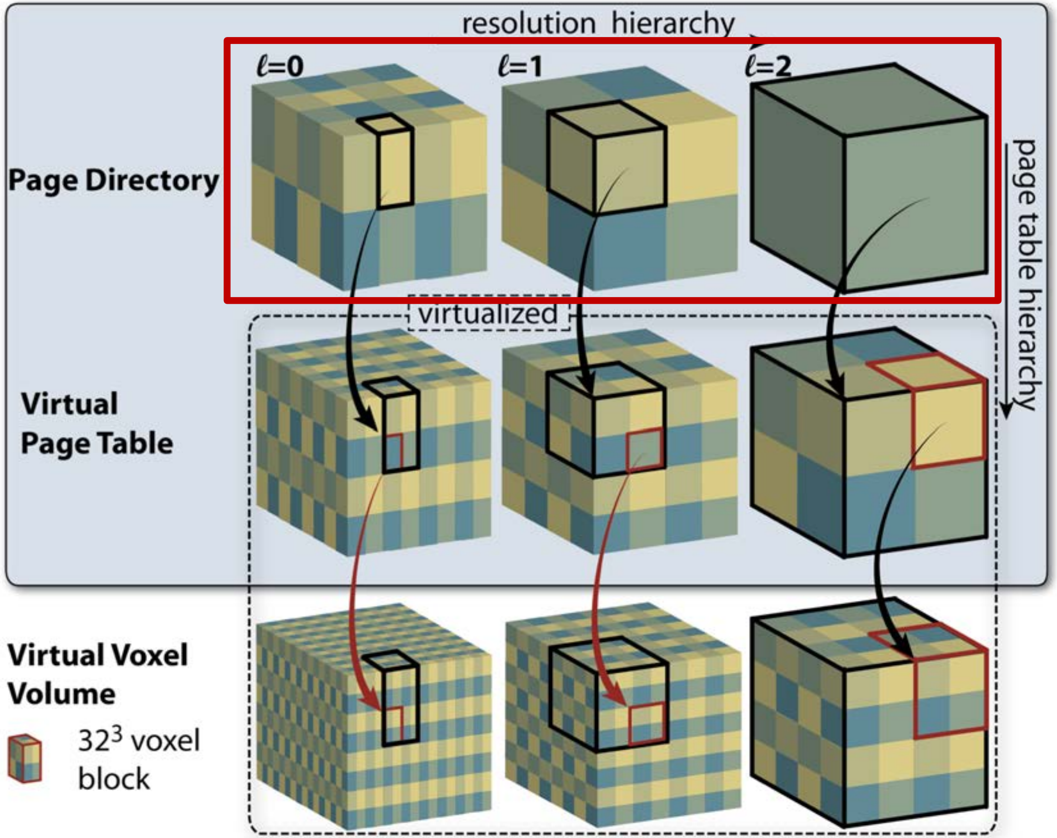
- Same idea as in CPU multi-level page tables
- Pages of page table entries like pages of voxels

Recursive page table hierarchy

- Decoupled from data resolution levels !
- # page table levels << # data resolution levels



MULTI-LEVEL PAGE TABLES: MULTI-RESOLUTION



multi-resolution
page directory

[Hadwiger et al., 2012]

MULTI-LEVEL PAGE TABLES: SCALABILITY

resolution	size	resolution hierarchy	page table hierarchy	page directory
32,000 x 32,000 x 4,000	4 TB	11 levels	2 levels	32 x 32 x 4
128,000 x 128,000 x 16,000	196 TB	13 levels	2 levels	128 x 128 x 16
512,000 x 512,000 x 64,000	15 PB	15 levels	3 levels	16 x 16 x 2
2,000,000 x 2,000,000 x 250,000	888 PB	17 levels	3 levels	64 x 64 x 8

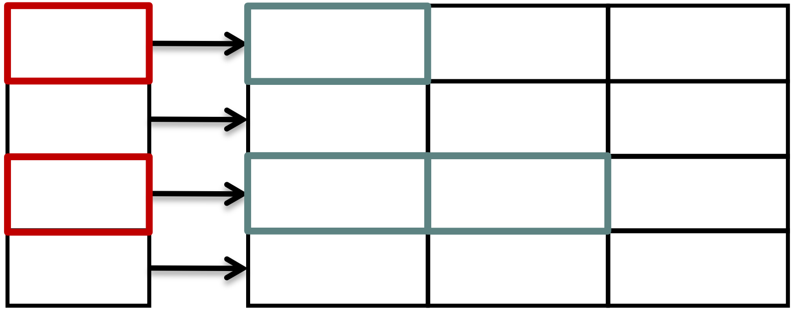
voxel blocks: 32^3 voxels

page table blocks: 32^3 page table entries

VARIANT 4: SPATIAL HASHING (1)

Instead of virtualizing page table, put entries into hash table

- Hashing function maps virtual brick to page table entry
- Hash table size is maximum working set size



working set

VARIANT 4: SPATIAL HASHING (2)

Hashing function

- Map (x, y, z) or (x, y, z, lod) of brick to 1D index
- $x*p_1$ xor $y*p_2$ xor $z*p_3$ modulo # hash table rows
- p_1, p_2, p_3 are large prime numbers

Hashing function must minimize collisions

- Collision handling expensive (linear search, link traversal)

Missing bricks: linear search through hash table row

RAY-GUIDED VOLUME RENDERING (1)

Working set determination on GPU

- Ray-guided / visualization-driven approaches

Prefer single-pass rendering

- Entire traversal on GPU
- Use small brick sizes
- Multi-pass only when working set too large for single pass

Virtual texturing

- Powerful paradigm with very good scalability

RAY-GUIDED VOLUME RENDERING (2)

With octree traversal (kd-restart)

- Gigavoxels [Crassin et al., 2009]
  - Gigavoxel isosurface and volume rendering
- Tera-CVR [Engel, 2011]
  - Teravoxel volume rendering with dynamic transfer functions

Virtual texturing instead of tree traversal

- Petascale volume exploration of microscopy streams [Hadwiger et al., 2012]
  - *Visualization-driven* pipeline, including data construction
- ImageVis3D [Fogal et al., 2013]
  - Analysis of different settings (brick size, …)

Examples

EARLY 'RAY-GUIDED' OCTREE RAY-CASTING (1)

Data structure:

- Octree with ropes
  - Pointers to 8 children, 6 neighbors and volume data
  - Active subtree stored in spatial index structure and texture pool on GPU

[Gobbetti et al., The Visual Computer, 2008]
A single-pass GPU ray casting framework for interactive out-of-core rendering of massive volumetric datasets

Working set determination:	Interleaved occlusion queries
Volume representation:	Octree
Rendering:	GPU octree traversal

EARLY 'RAY-GUIDED' OCTREE RAY-CASTING (2)

Rendering:

- Stackless GPU octree traversal (rope tree)

Culling:

- Culling on CPU (global transfer function, iso-value, view frustum)
  - Only nodes that were marked as visible in previous rendering pass refined
  - Occlusion queries to check bounding box of node against depth of last sample during raycasting

[Gobbetti et al., The Visual Computer, 2008]
A single-pass GPU ray casting framework for interactive out-of-core rendering of massive volumetric datasets

Working set determination:	Interleaved occlusion queries
Volume representation:	Octree
Rendering:	GPU octree traversal

RAY-GUIDED OCTREE RAY-CASTING (1)

Data structure:

- N^3 tree + multi-resolution volume
- Subtree stored on GPU in node/brick pool
  - Node: 1 pointer to children, 1 pointer to volume brick
  - Children stored together in node pool

[Crassin et al., ACM SIGGRAPH i3D, 2009]
GigaVoxels: Ray-Guided Streaming for Efficient and Detailed Voxel Rendering

Working set determination:	Ray-guided
Volume representation:	Octree
Rendering:	GPU octree traversal

RAY-GUIDED OCTREE RAY-CASTING (2)

Rendering:

- Stackless GPU octree traversal (Kd-restart)
- 3 mipmap levels for correct filtering
- Missing data substituted by lower-res data

Culling:

- Multiple render targets write out data usage
  - Exploits temporal and spatial coherence

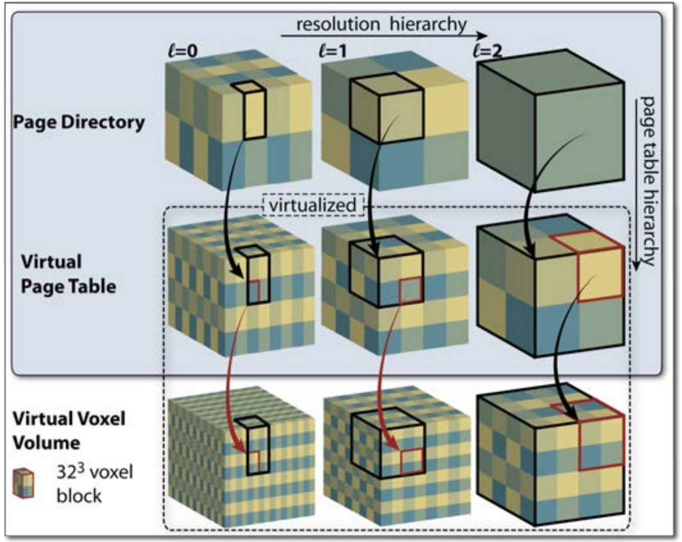
[Crassin et al., ACM SIGGRAPH i3D, 2009]
GigaVoxels: Ray-Guided Streaming for Efficient and Detailed Voxel Rendering

Working set determination:	Ray-guided
Volume representation:	Octree
Rendering:	GPU octree traversal

RAY-GUIDED MULTI-LEVEL PAGETABLE RAY-CASTING (1)

Data structure:

- On-the-fly reconstruction of bricks
- Stored on disk in 2D multi-resolution grid (supports highly anisotropic data)
- Multi-level multi-resolution page table on GPU
- Larger bricks for disk access, smaller bricks for rendering



[Hadwiger et al., IEEE SciVis 2012]
Interactive Volume Exploration of Petascale Microscopy Data Streams Using a Visualization-Driven Virtual Memory Approach

Working set determination:	Ray-guided
Volume representation:	Multi-resolution grid
Rendering:	Multi-level virtual texture ray-casting

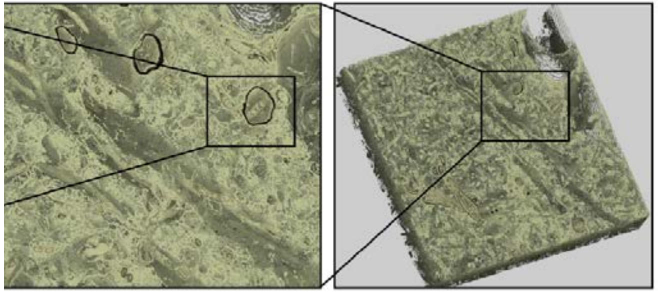
RAY-GUIDED MULTI-LEVEL PAGETABLE RAY-CASTING (2)

Rendering:

- Multi-level virtual texture ray-casting
- LOD chosen per individual sample
- Data reconstruction triggered by ray-caster

Culling:

- GPU hash table to report missing blocks
  - Exploits temporal and spatial coherence



[Hadwiger et al., IEEE SciVis 2012]
Interactive Volume Exploration of Petascale Microscopy Data Streams Using a Visualization-Driven Virtual Memory Approach

Working set determination:	Ray-guided
Volume representation:	Multi-resolution grid
Rendering:	Multi-level virtual texture ray-casting

RAY-GUIDED MULTI-LEVEL PAGETABLE RAY-CASTING - ANALYSIS

Implementation differences:

- Lock-free hash table, pagetable lookup only per brick
- Fallback for multi-pass rendering

Analysis:

- Many detailed performance numbers (see paper)
- Working set size: typically lower than GPU memory
- Brick size: larger on disk (>= 64^3), smaller for rendering (16^3, 32^3)

[Fogal et al., IEEE LDAV 2013]
An Analysis of Scalable GPU-Based Ray-Guided Volume Rendering

Working set determination:	Ray-guided
Volume representation:	Multi-resolution grid
Rendering:	(Multi-level) virtual texture ray-casting

SIGGRAPH ASIA 2018 TOKYO

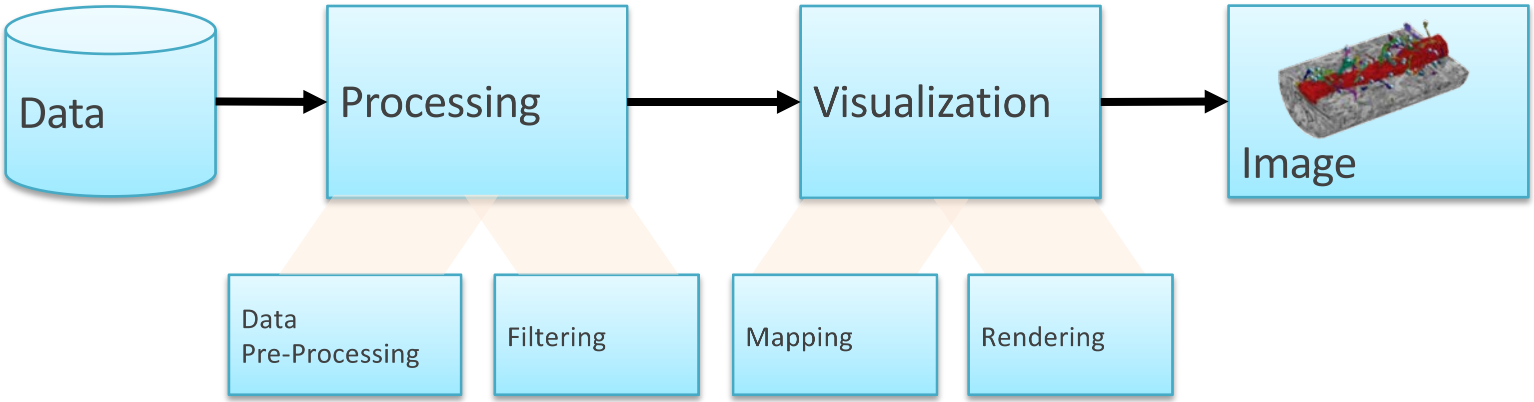
Sponsored by

Summary

SUMMARY (1)

Many volumes larger than GPU memory

- Determine, manage, and render working set of visible bricks efficiently



SUMMARY (2)

Traditional approaches

- Limited scalability
- Visibility determination on CPU
- Often had to use multi-pass approaches

Modern approaches

- High scalability (output sensitive)
- Visibility determination (working set) on GPU
- Dynamic traversal of multi-resolution structures on GPU

SUMMARY (3)

Orthogonal approaches

- Parallel and distributed visualization
- Clusters, in-situ setups, client/server systems

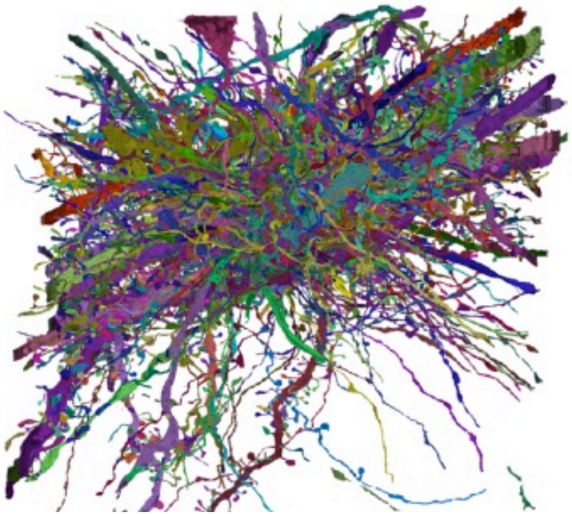
Future challenges

- Web-based visualization
- Raw data storage

Part 3 -
GPU-Based Ray-Guided Volume Rendering Algorithms & Efficient Empty Space Skipping

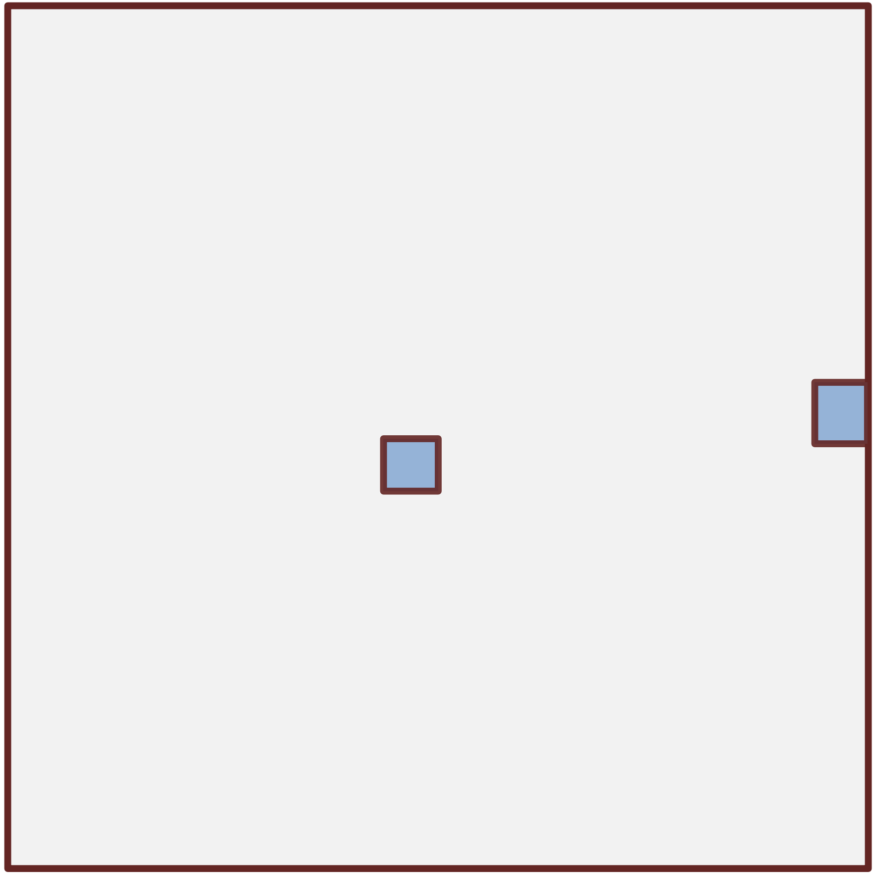
MOTIVATION

Large volumes, finely detailed structures, many segmented objects

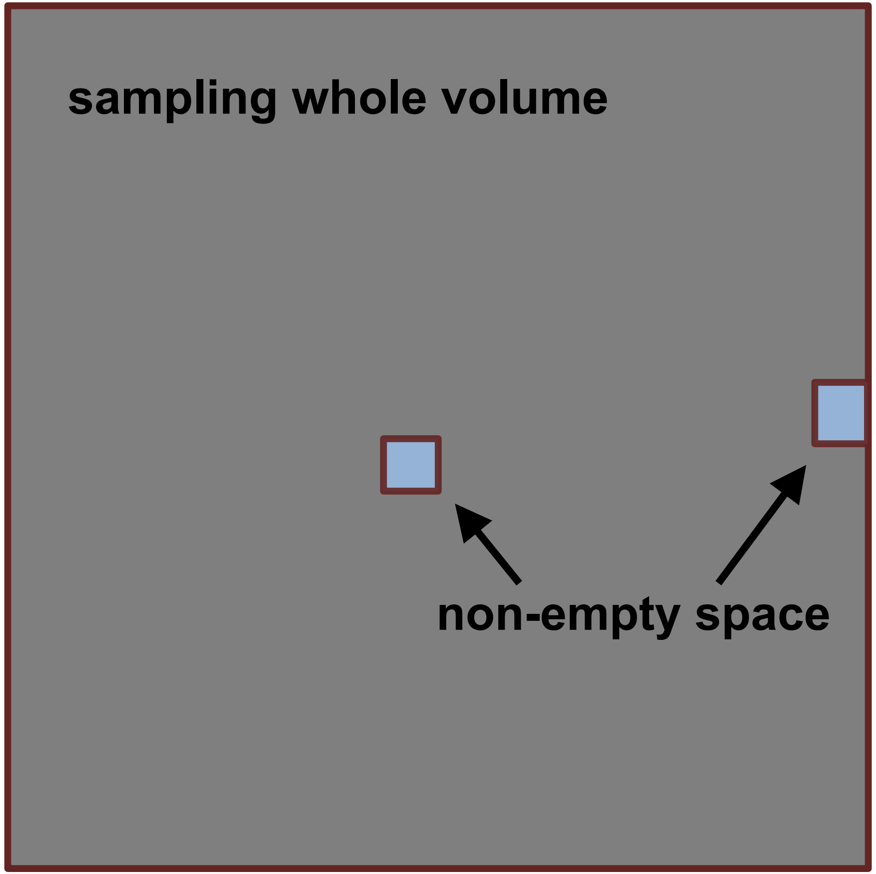


connectomics electron microscopy volume

21,000 x 25,000 x 2,000 > 1 teravoxels > 4,000 objects



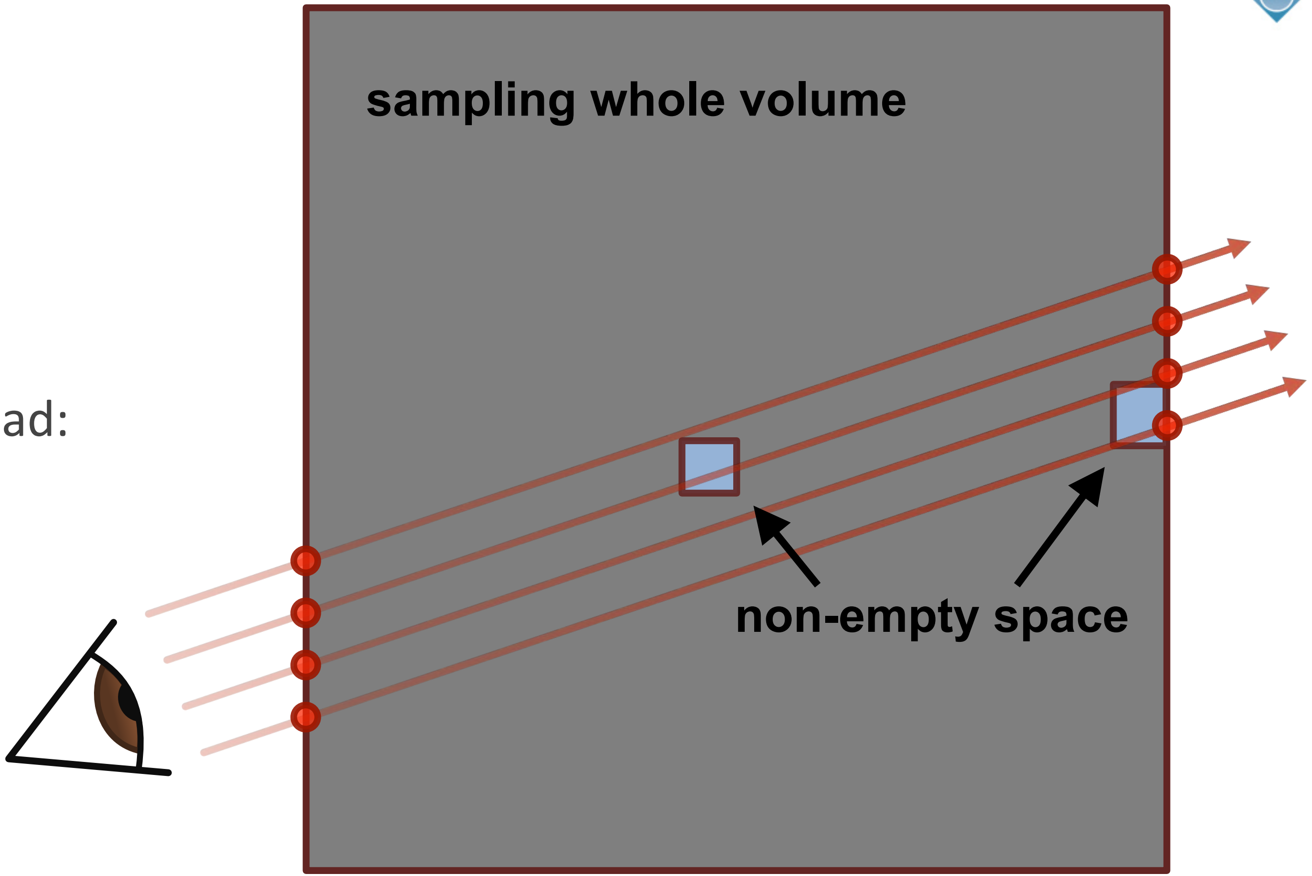
no skipping



no skipping

look-up overhead:

none



look-ups

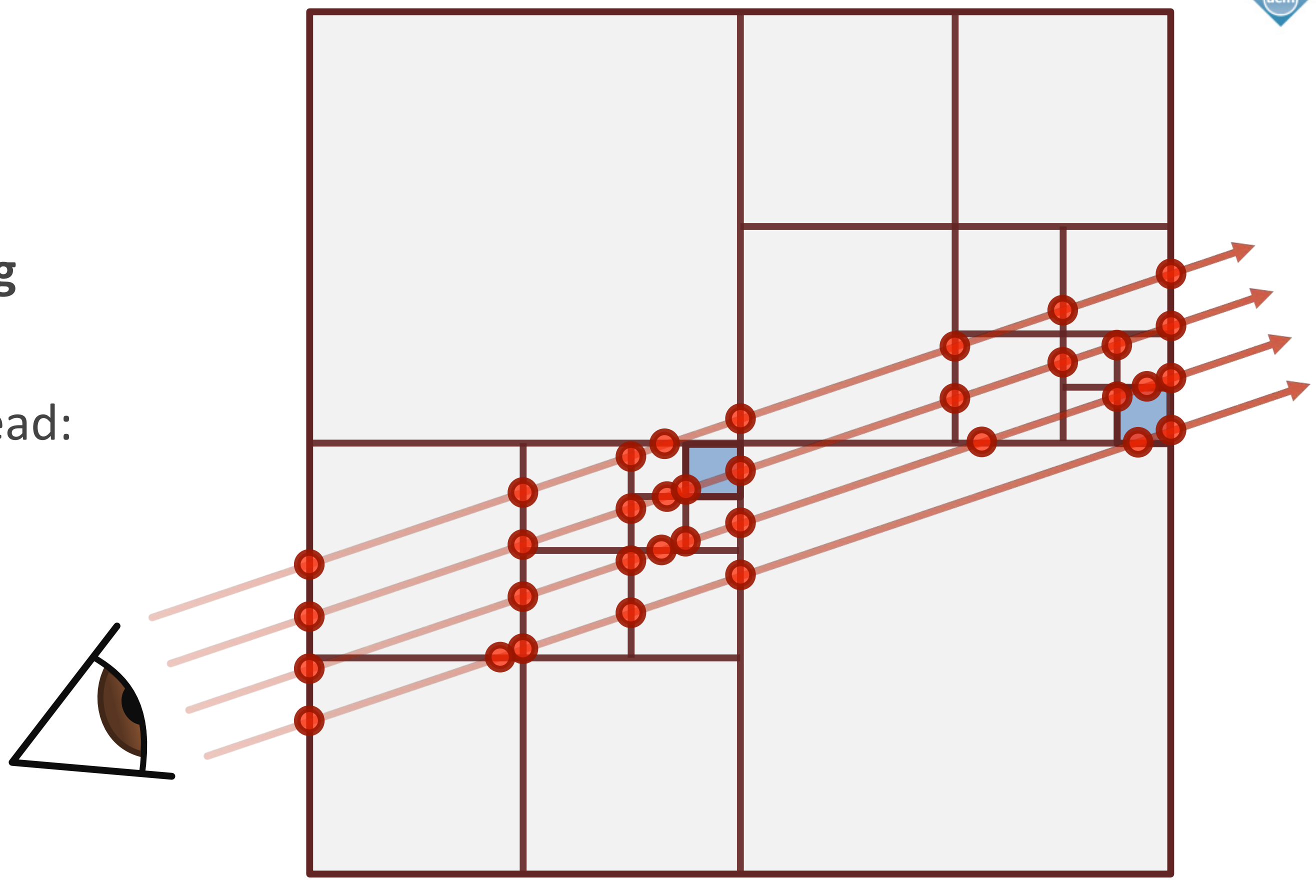
SIGGRAPH ASIA 2018 TOKYO

Sponsored by

octree skipping

look-up overhead:

high

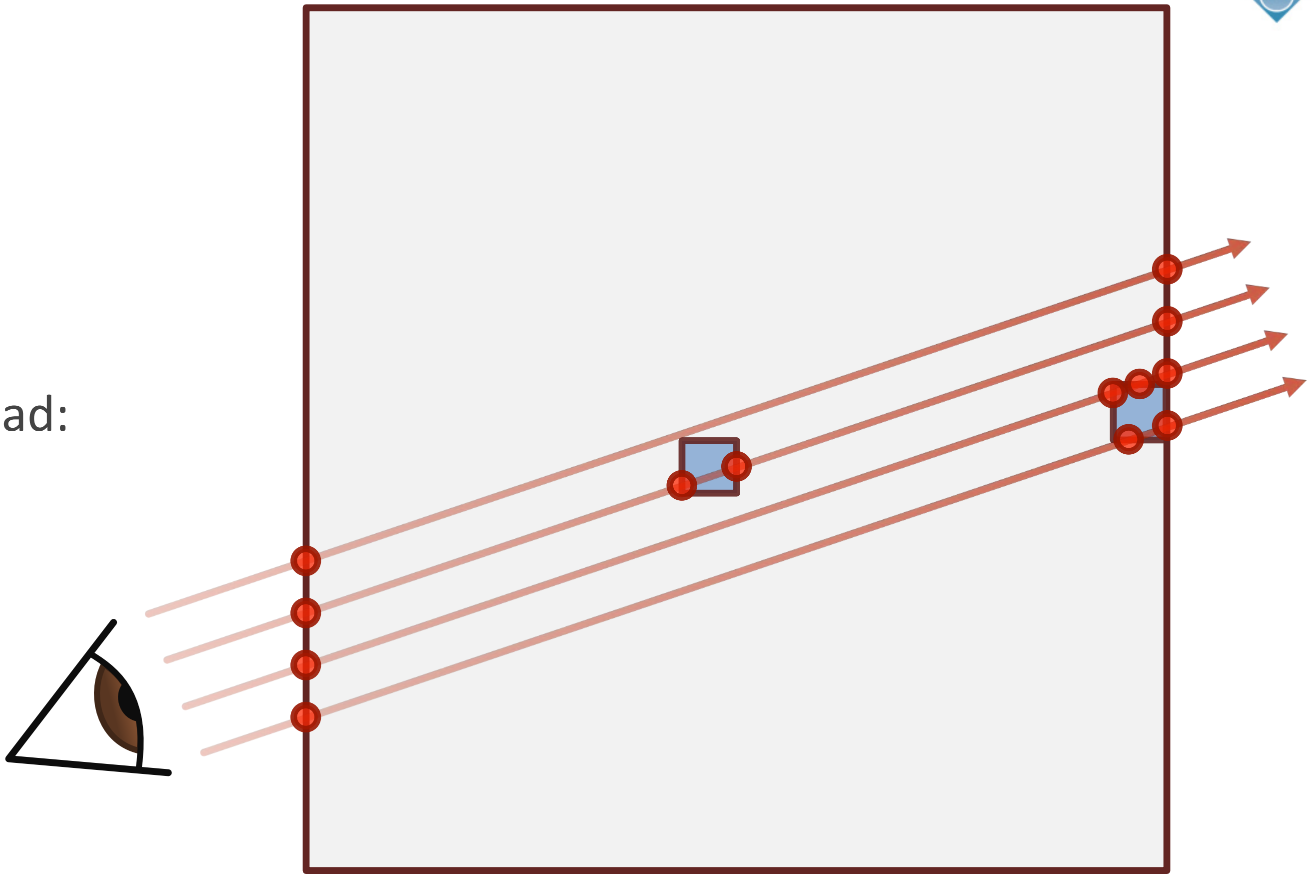


look-ups

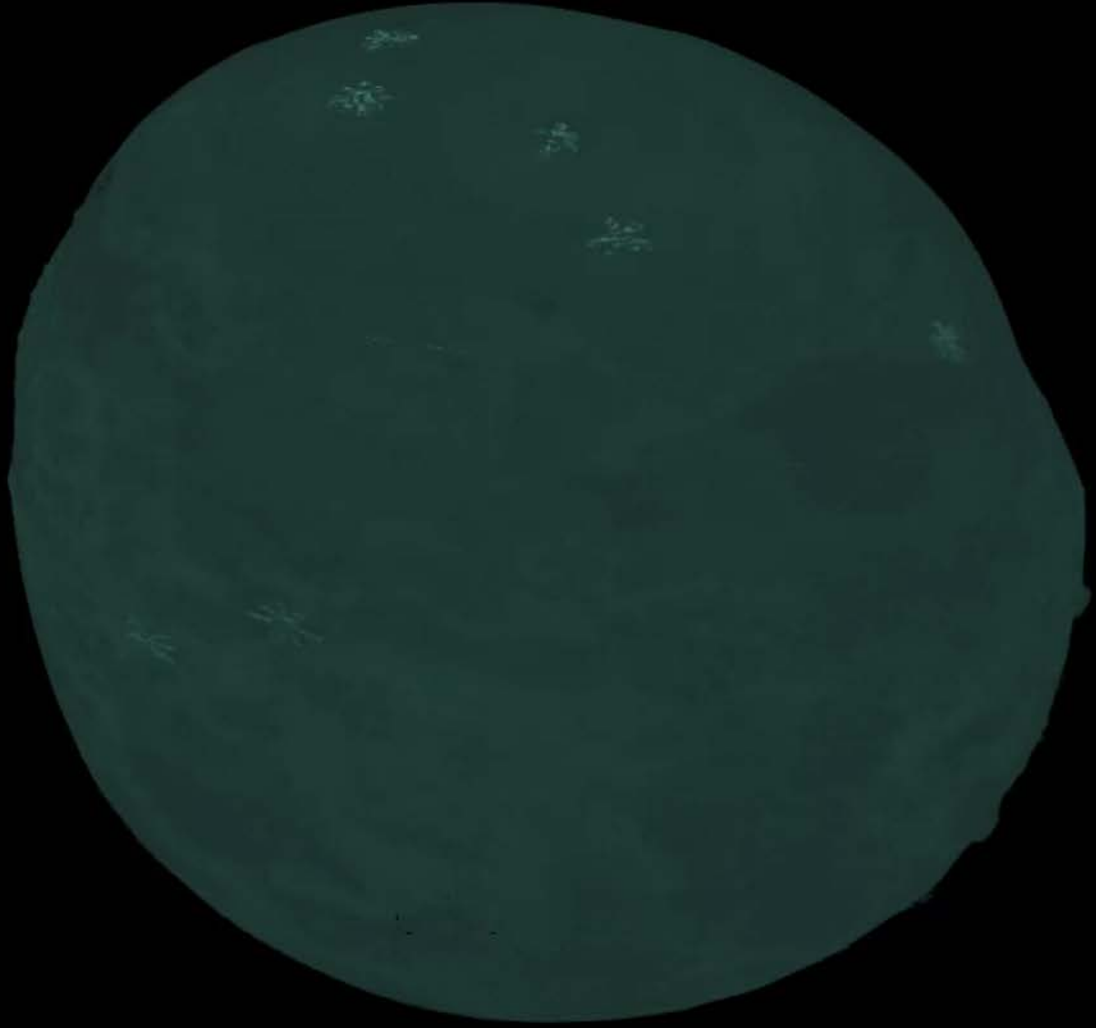
SparseLeap

look-up overhead:

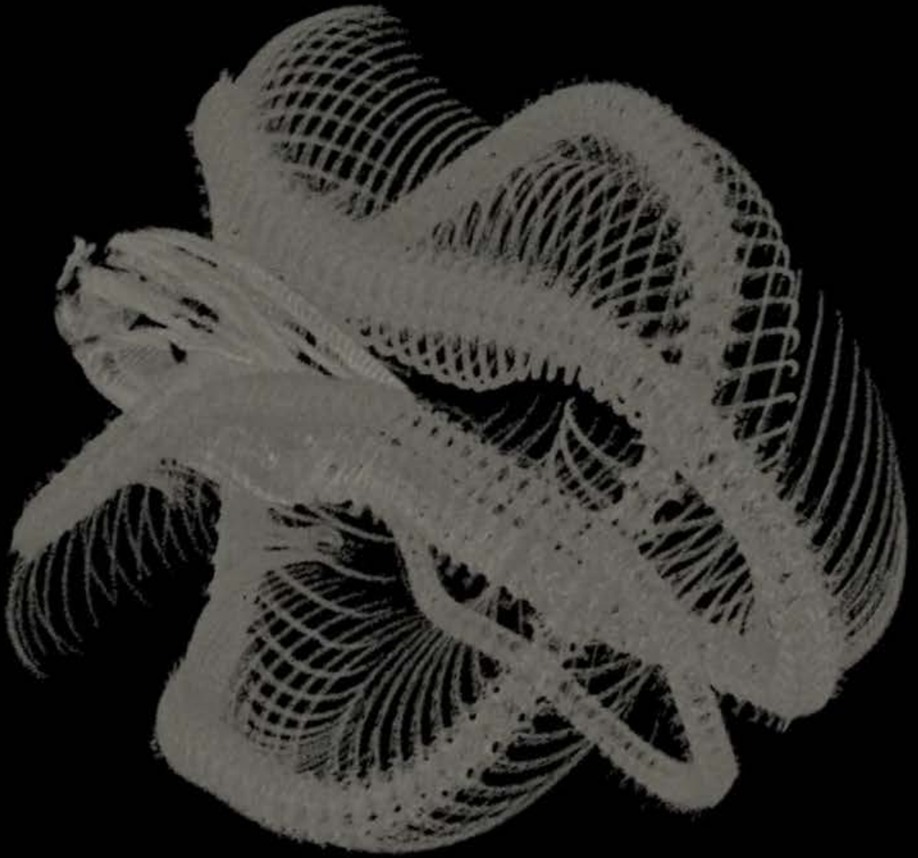
small



look-ups



Octree



depth complexity: # look-ups for space skipping


SparseLeap
depth complexity: # look-ups for space skipping

SPARSELEAP PIPELINE

Track volume occupancy

- Occupancy histogram tree

Extract nested occupancy

- Occupancy geometry

Rasterize occupancy

- Ray segment lists

Empty space skipping: Linear list traversal

SPARSELEAP PIPELINE

Track volume occupancy

- **Occupancy histogram tree**

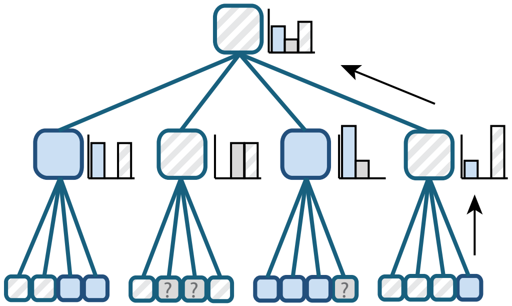
Extract nested occupancy

- Occupancy geometry

Rasterize occupancy

- Ray segment lists

Empty space skipping: Linear list traversal



SPARSELEAP PIPELINE

Track volume occupancy

- Occupancy histogram tree

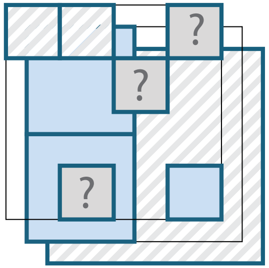
Extract nested occupancy

- **Occupancy geometry**

Rasterize occupancy

- Ray segment lists

Empty space skipping: Linear list traversal



SPARSELEAP PIPELINE

Track volume occupancy

- Occupancy histogram tree

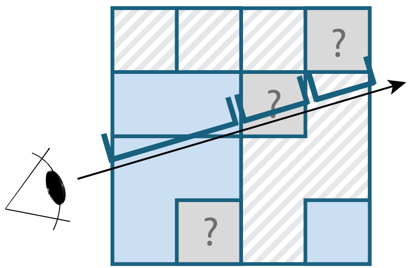
Extract nested occupancy

- Occupancy geometry

Rasterize occupancy

- **Ray segment lists**

Empty space skipping: Linear list traversal



SPARSELEAP PIPELINE

Track volume occupancy

- Occupancy histogram tree

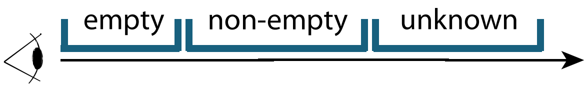
Extract nested occupancy

- Occupancy geometry

Rasterize occupancy

- Ray segment lists

Empty space skipping: Linear list traversal

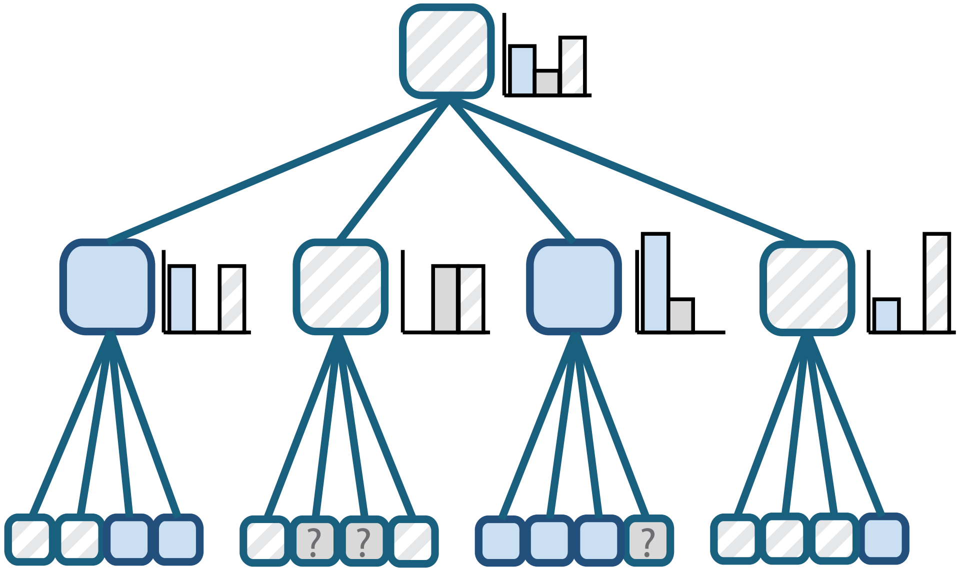


OCCUPANCY HISTOGRAM TREE

Occupancy classes

- non-empty
- empty
- ? unknown

Node count in each class over whole subtree

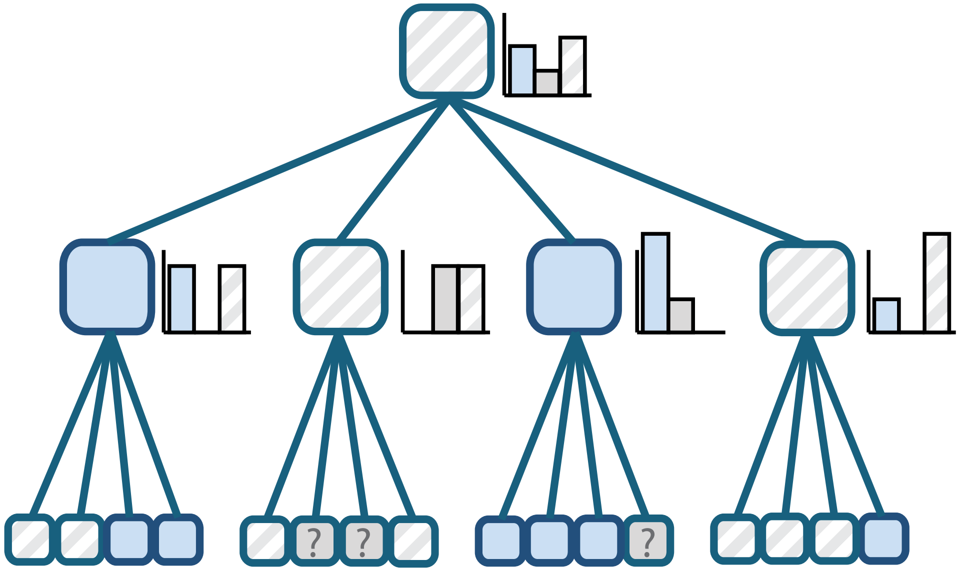


OCCUPANCY HISTOGRAM TREE

Occupancy classes

- non-empty
- empty
- ? unknown *

Node count in each class over whole subtree



* enables deferred culling

SIGGRAPH ASIA 2018 TOKYO

Sponsored by

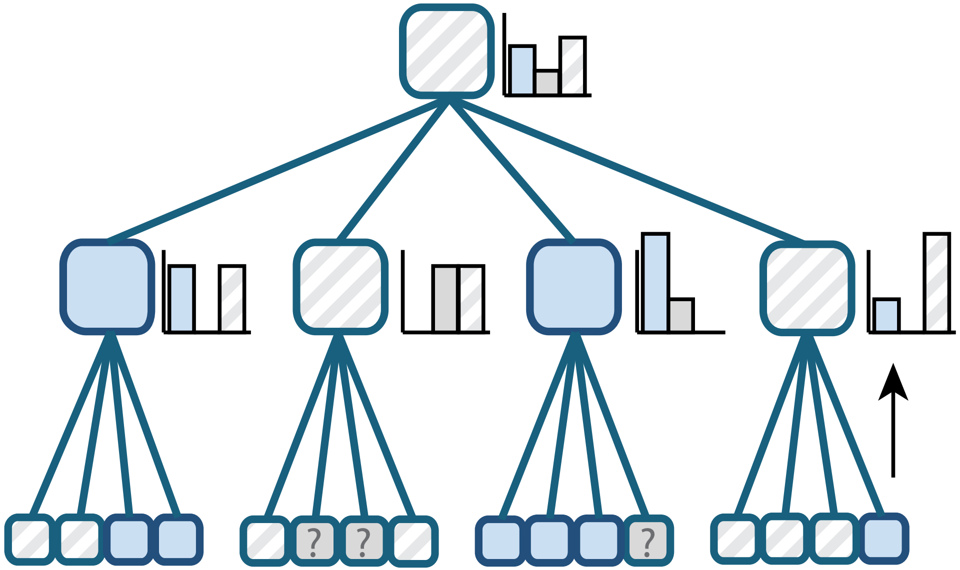
OCCUPANCY HISTOGRAM TREE

Occupancy classes

- non-empty
- empty
- unknown *

Node count in each class over whole subtree

* enables deferred culling



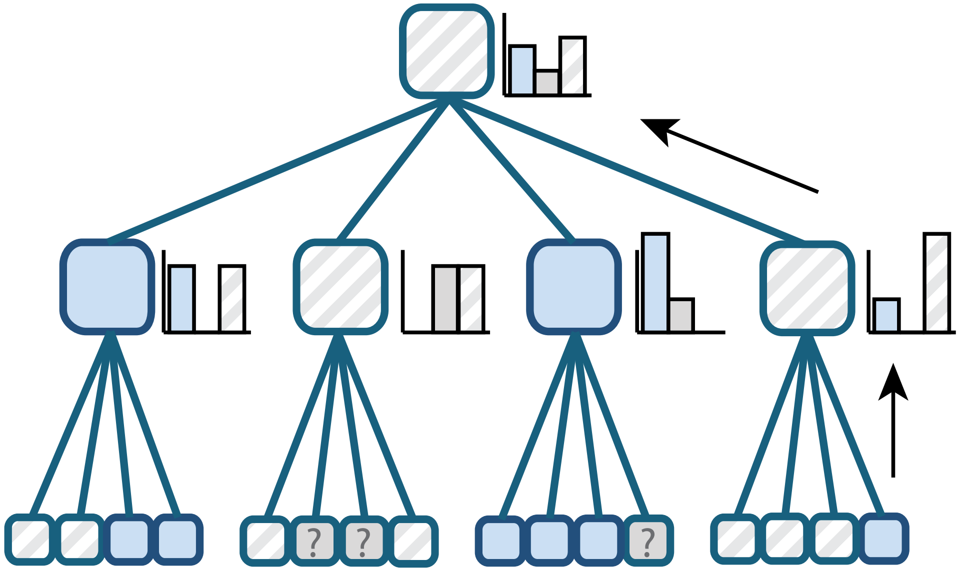
OCCUPANCY HISTOGRAM TREE

Occupancy classes

- non-empty
- empty
- ? unknown *

Node count in each class over whole subtree

* enables deferred culling

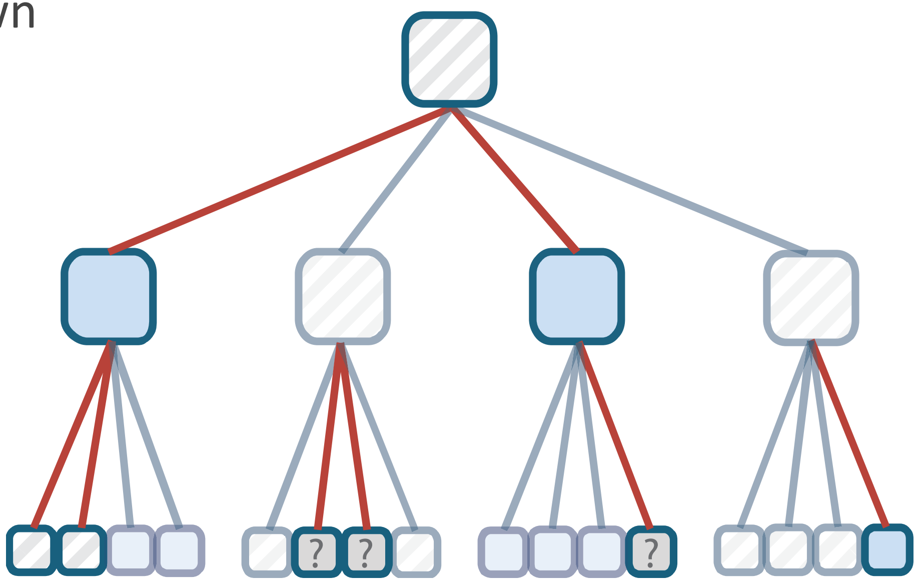


build bottom-up

OCCUPANCY GEOMETRY

Traverse histogram tree top-down

Pick majority class in each node

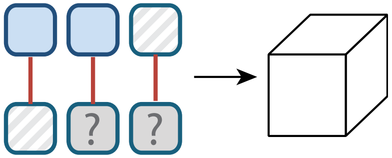


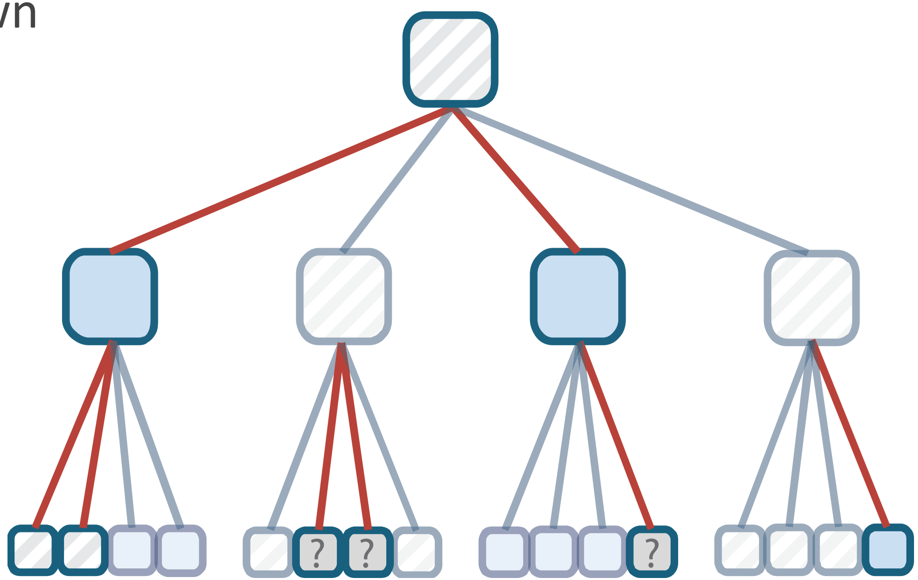
OCCUPANCY GEOMETRY

Traverse histogram tree top-down

Pick majority class in each node

Emit box on class change



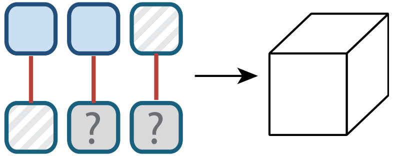


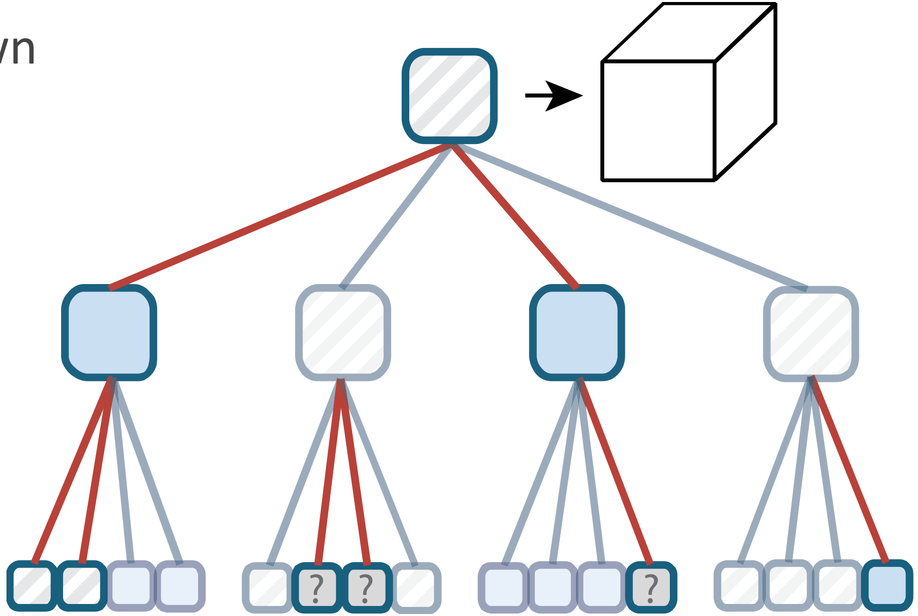
OCCUPANCY GEOMETRY

Traverse histogram tree top-down

Pick majority class in each node

Emit box on class change



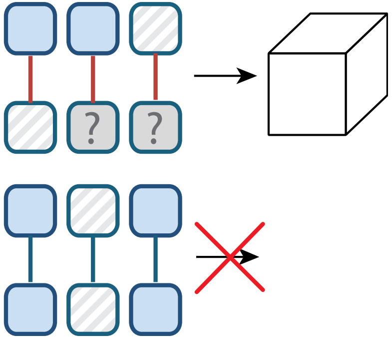


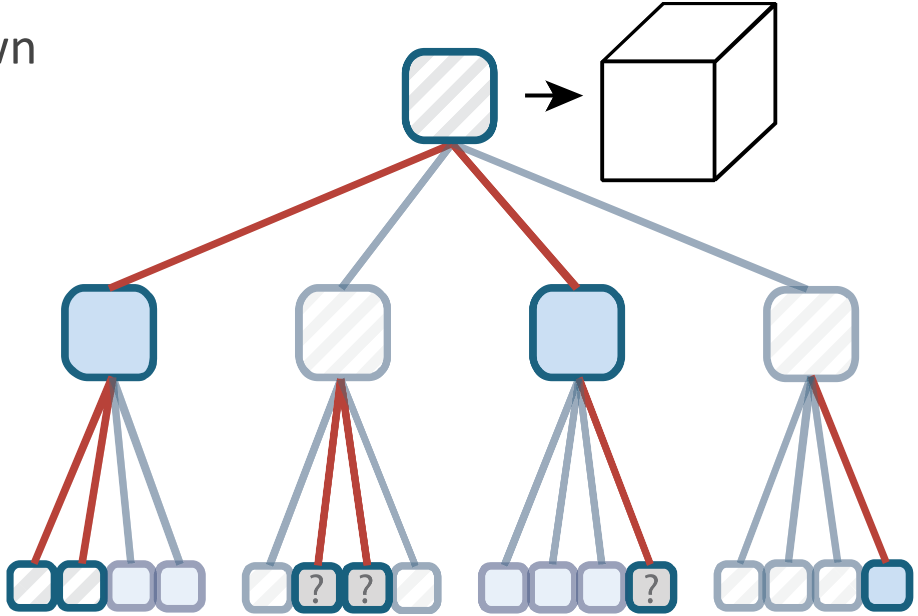
OCCUPANCY GEOMETRY

Traverse histogram tree top-down

Pick majority class in each node

Emit box on class change



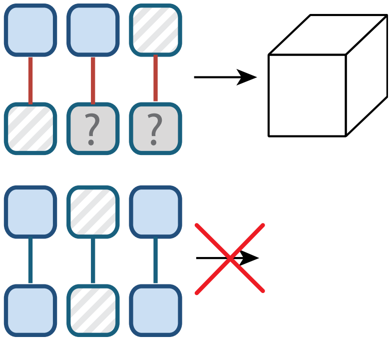


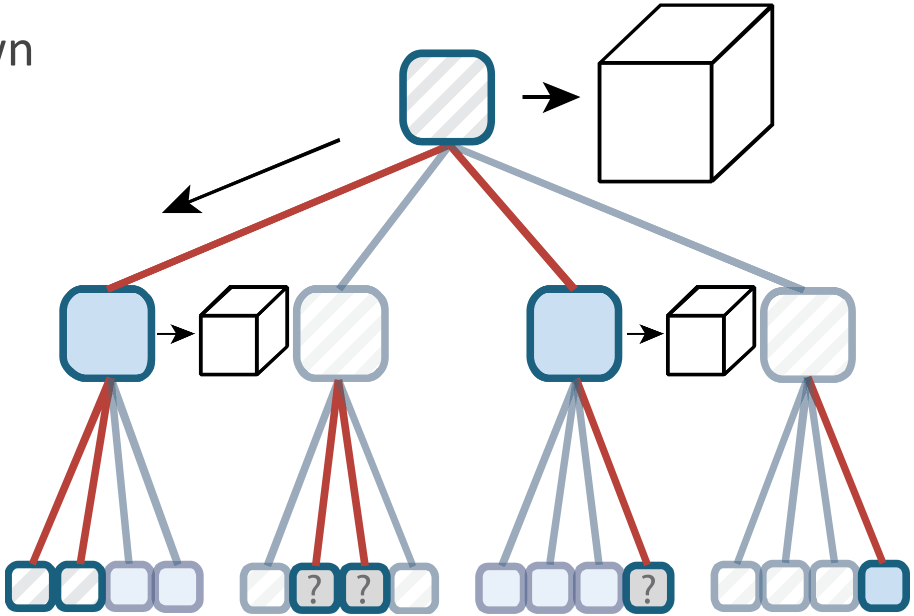
OCCUPANCY GEOMETRY

Traverse histogram tree top-down

Pick majority class in each node

Emit box on class change



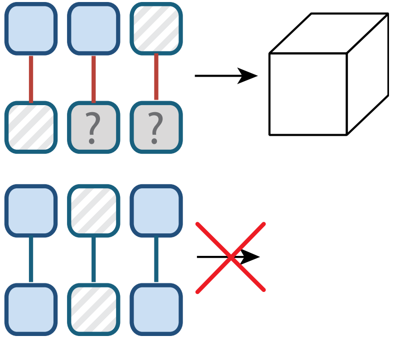


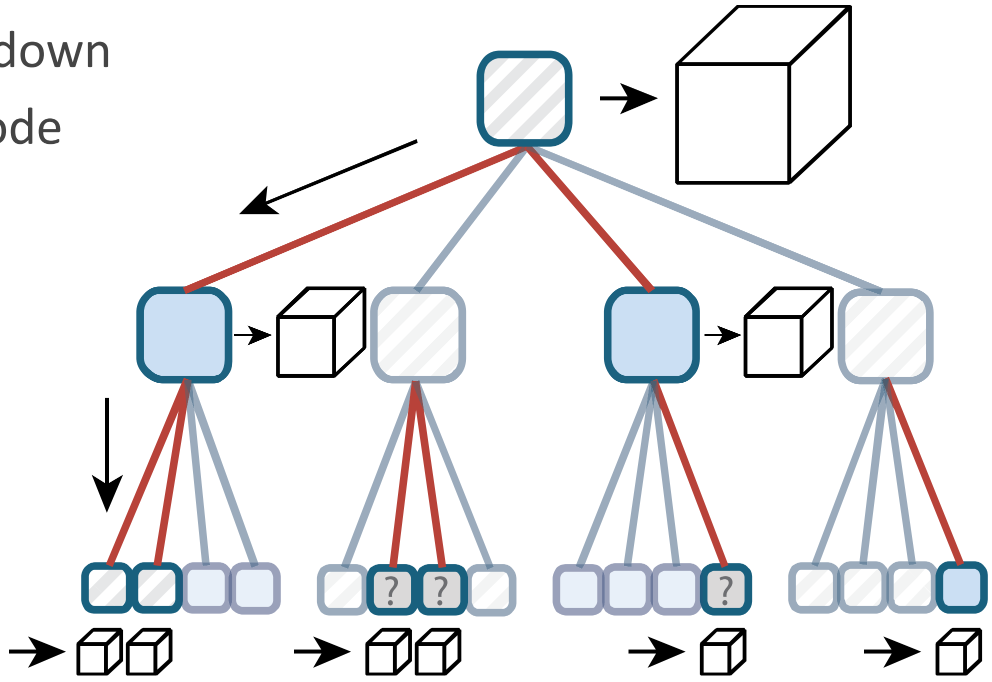
OCCUPANCY GEOMETRY

Traverse histogram tree top-down

Pick majority class in each node

Emit box on class change



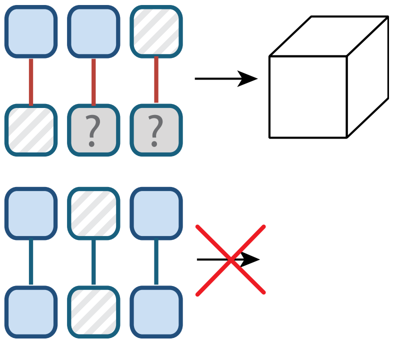


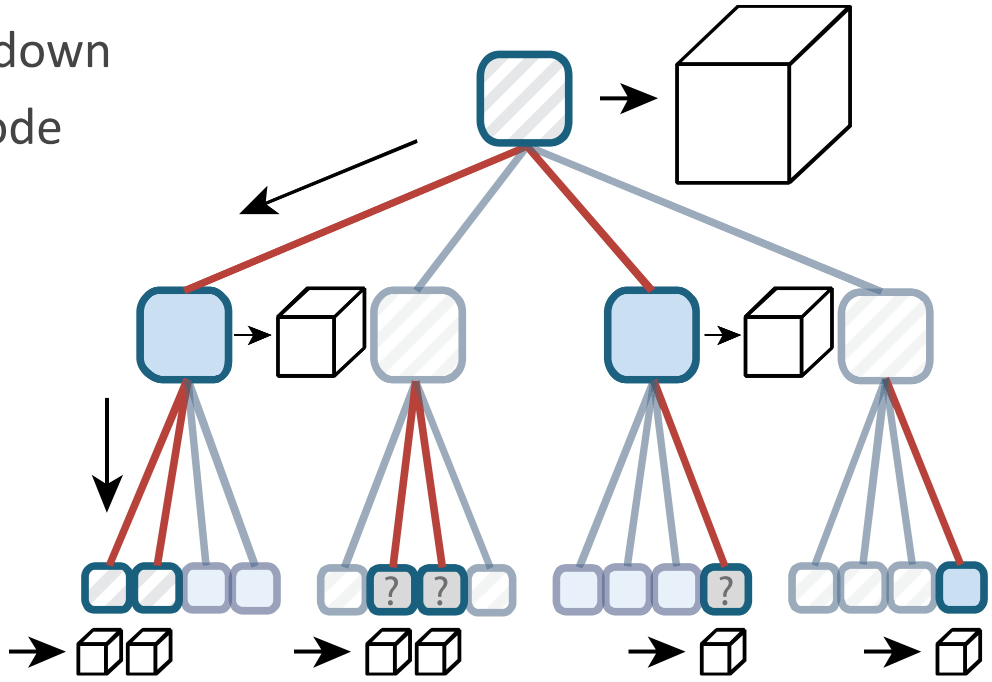
OCCUPANCY GEOMETRY

Traverse histogram tree top-down

Pick majority class in each node

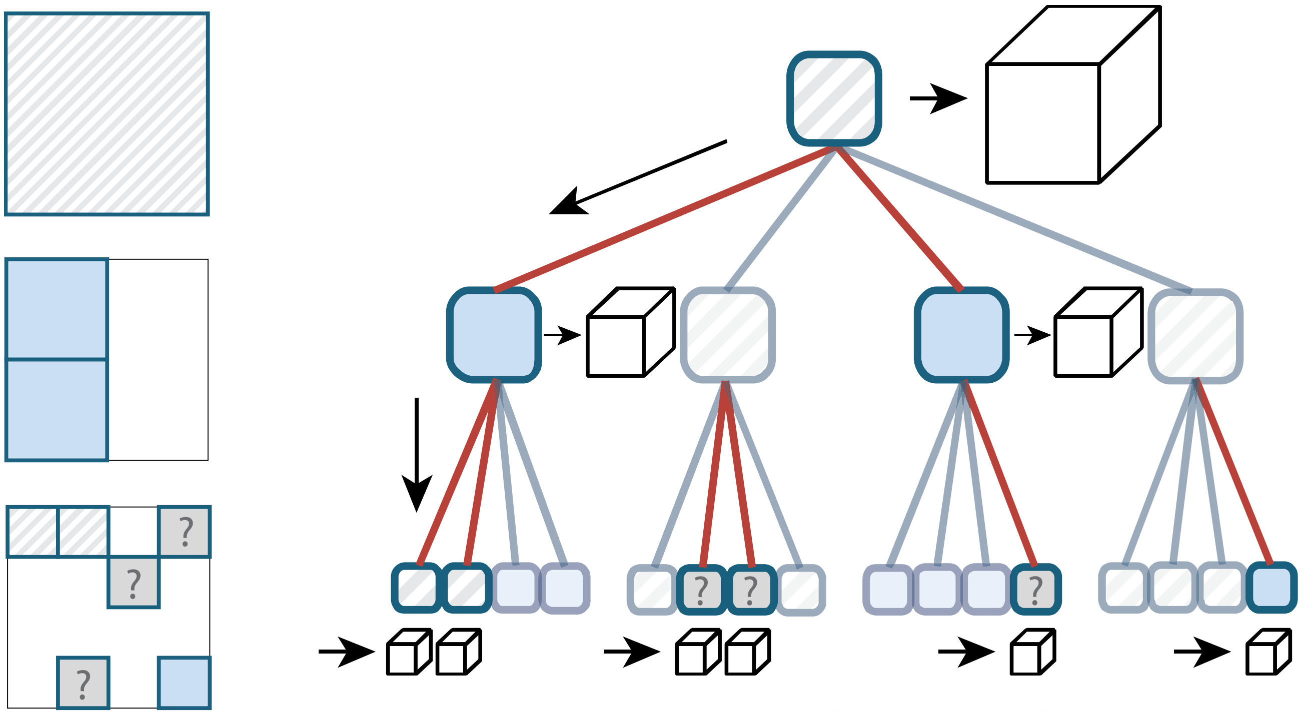
Emit box on class change





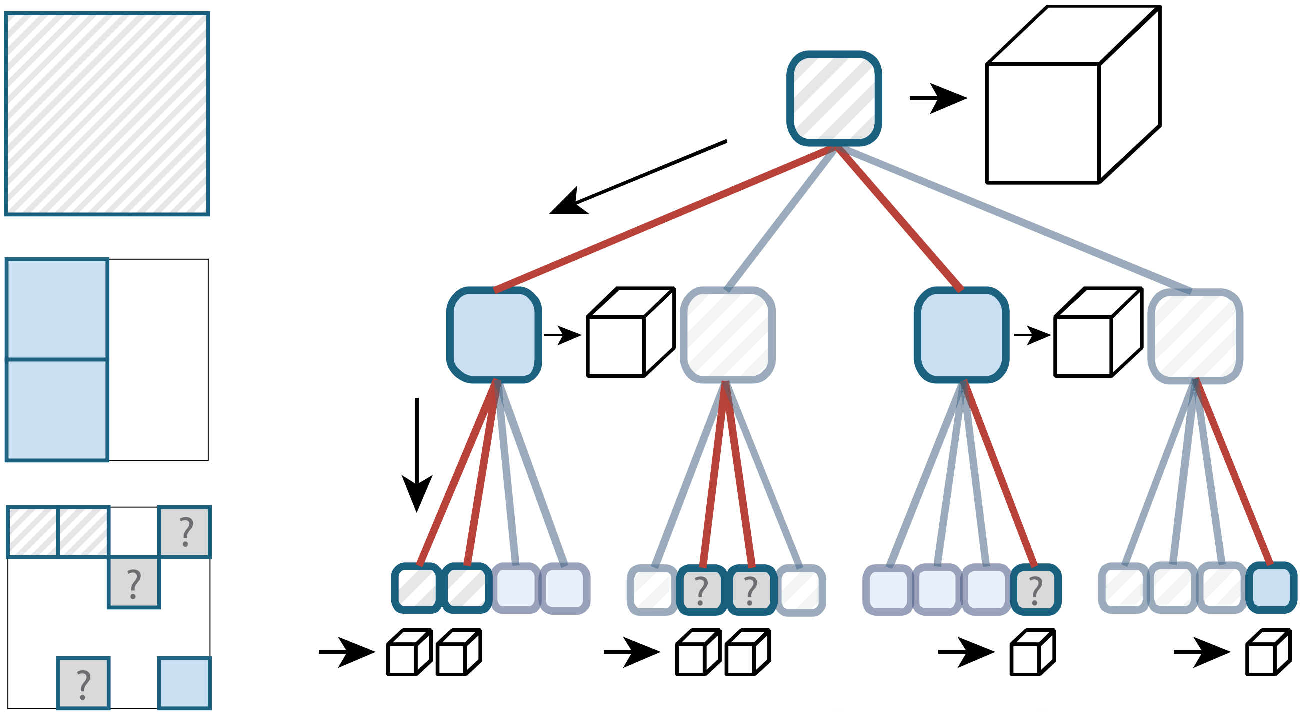
OCCUPANCY GEOMETRY

extracted geometry



OCCUPANCY GEOMETRY

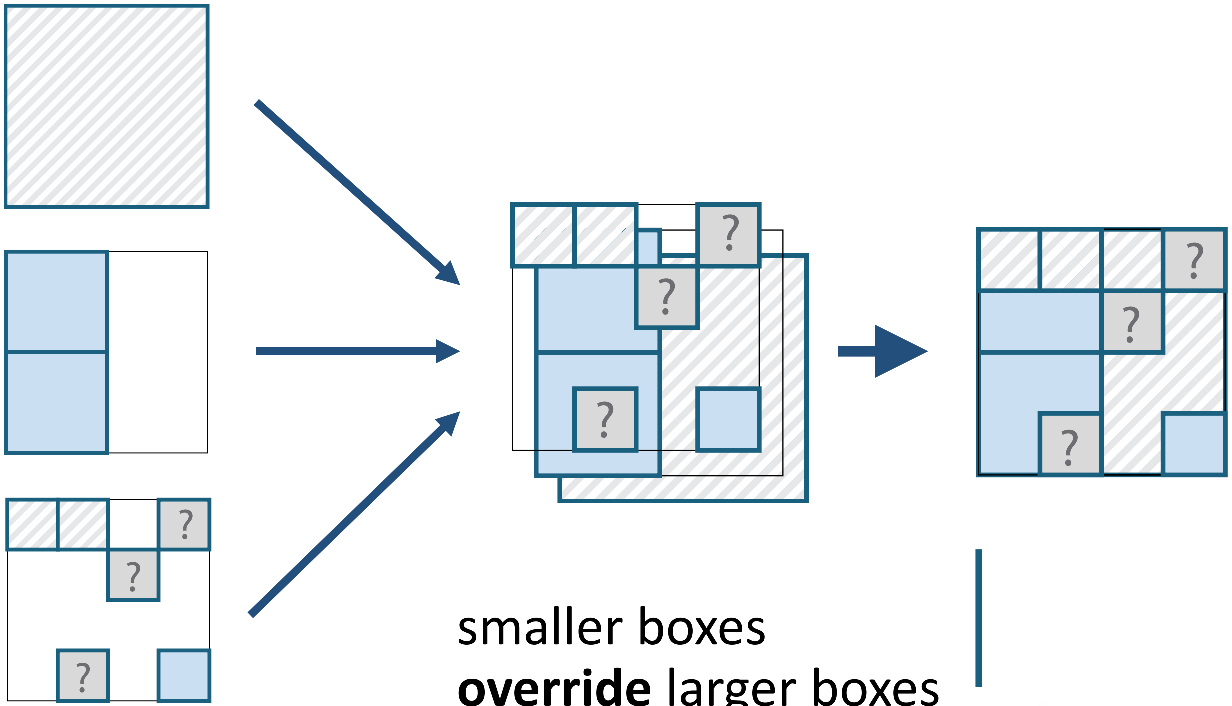
extracted geometry



OCCUPANCY GEOMETRY

extracted geometry

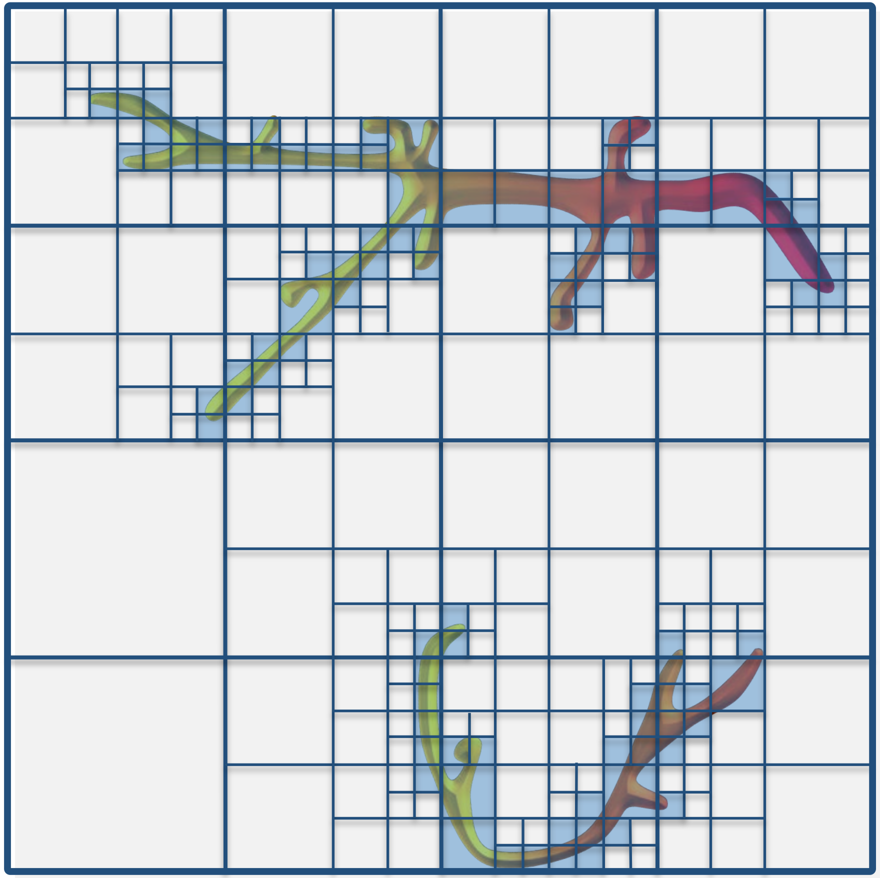
flattened occupancy



smaller boxes
override larger boxes

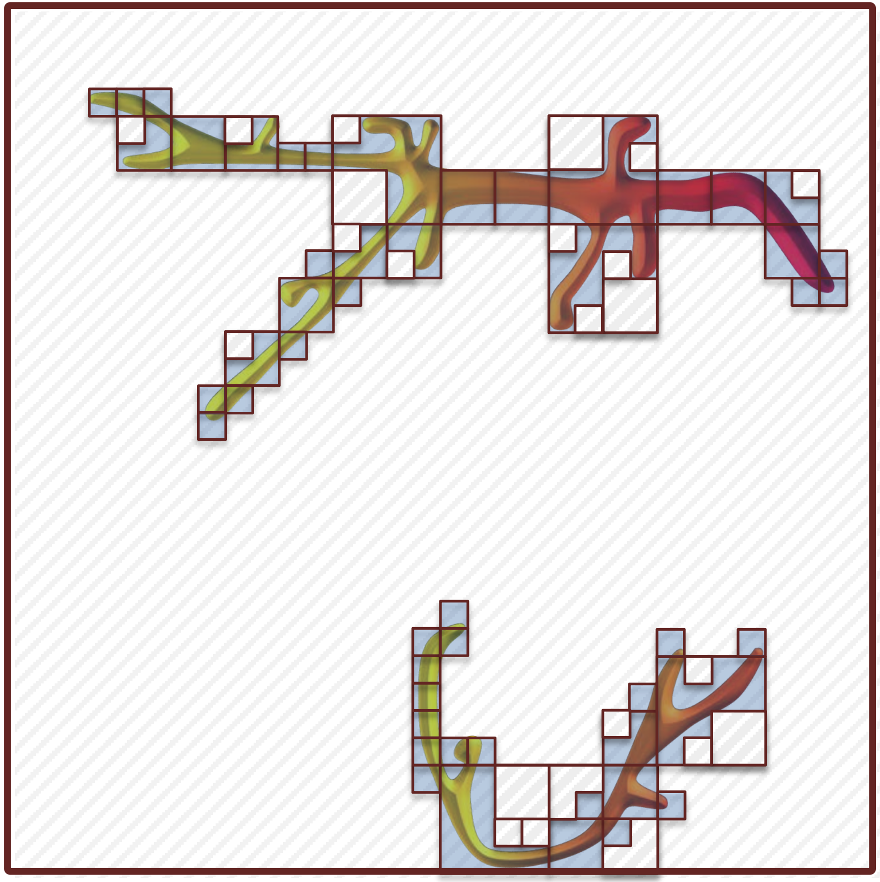
COMPARISON

octree
subdivision



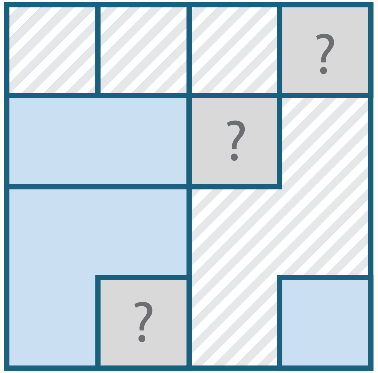
COMPARISON

occupancy
geometry



RASTERIZATION: OVERVIEW

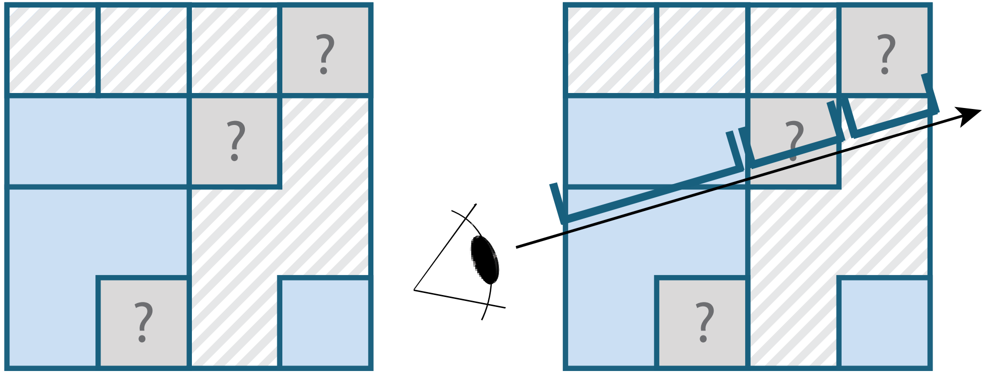
occupancy geometry



RASTERIZATION: OVERVIEW

occupancy geometry

rasterize front-to-back



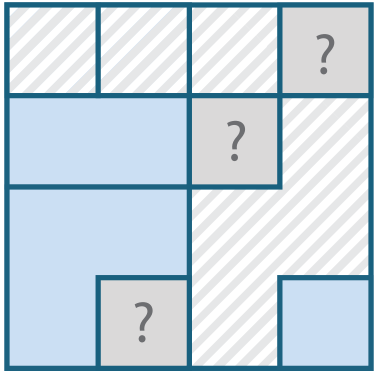
merge consecutive segments
of same occupancy class

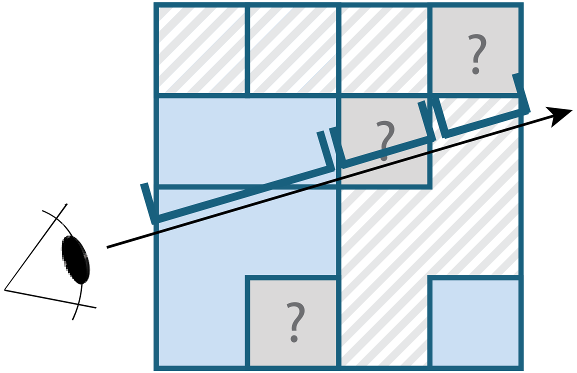
RASTERIZATION: OVERVIEW

occupancy geometry

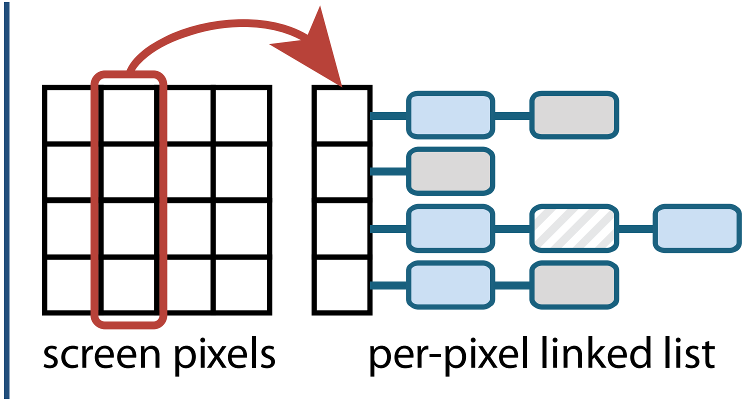
rasterize front-to-back

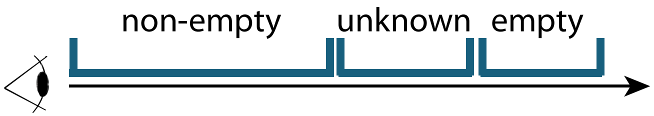
ray segment lists





merge consecutive segments
of same occupancy class





RAY-CASTING

Linear traversal of ray segment list

Deferred culling for large volumes:
Occupancy class *unknown*

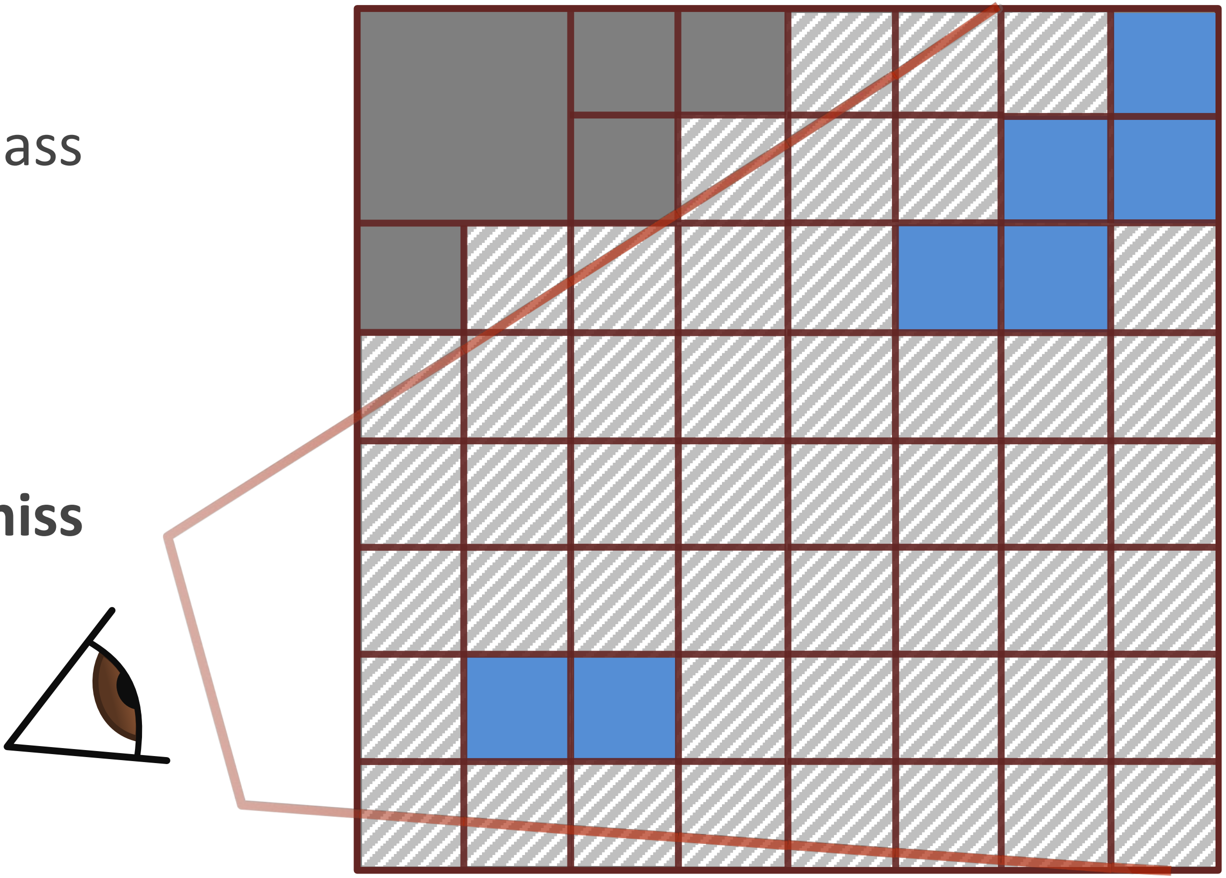
DEFERRED CULLING

occupancy class

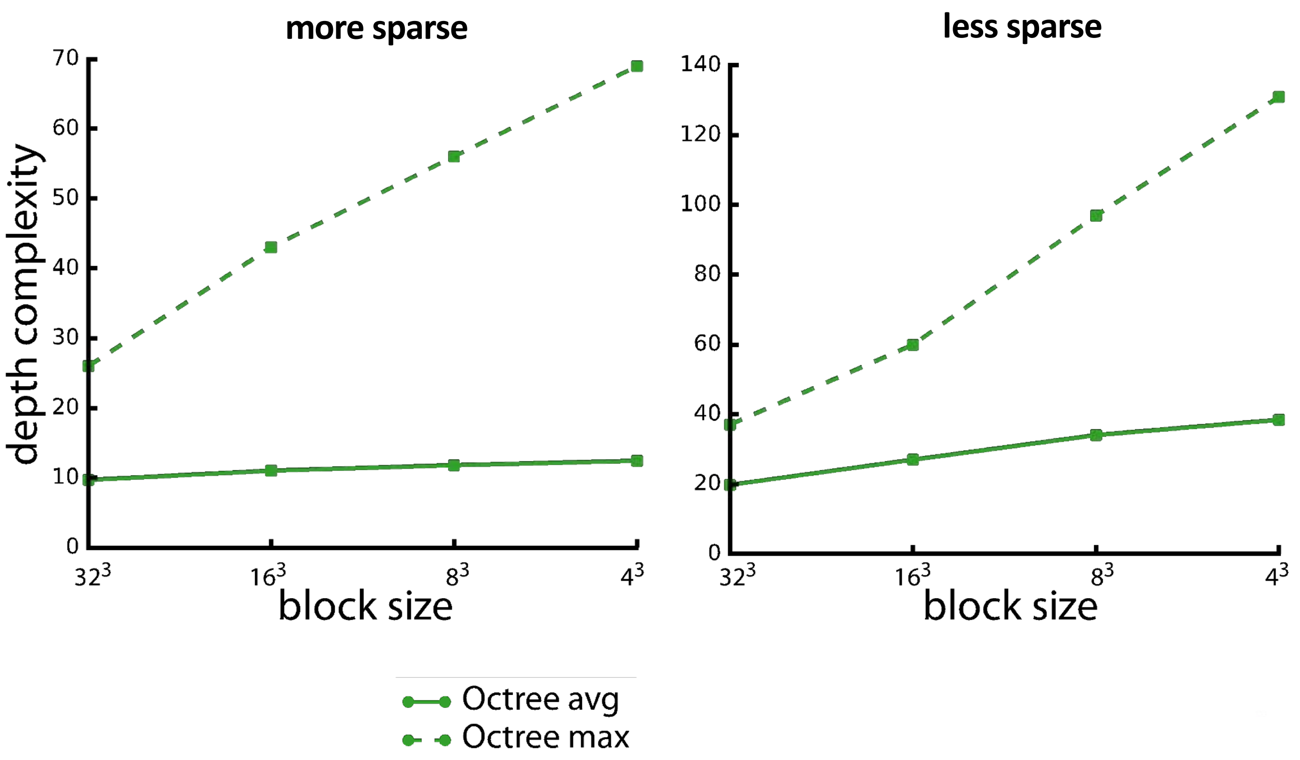
unknown

causes

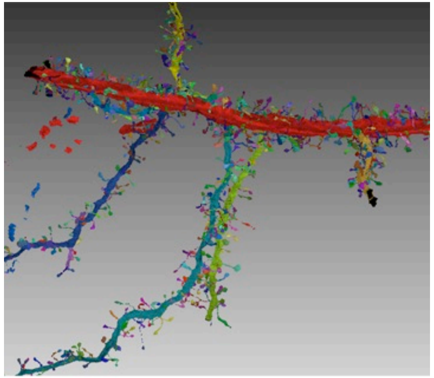
occupancy miss



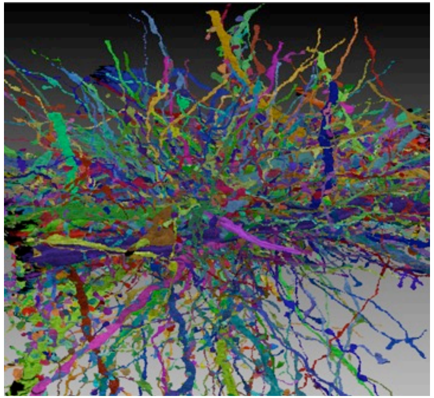
RESULTS: DEPTH COMPLEXITY



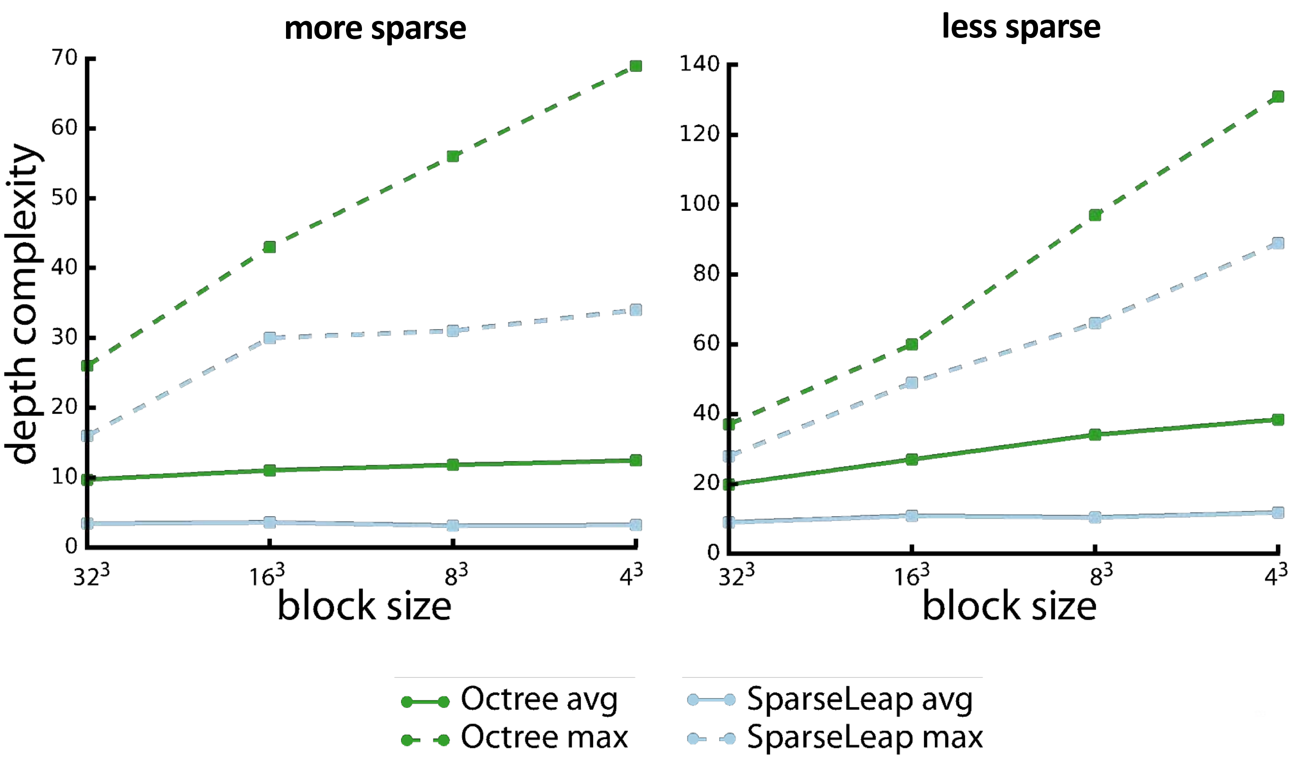
more sparse



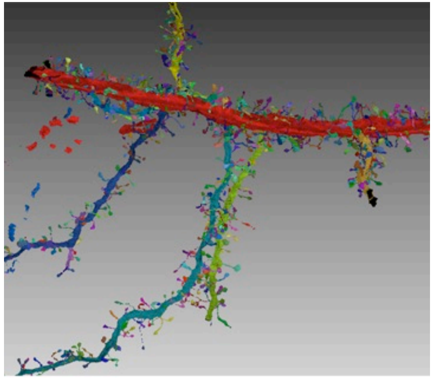
less sparse



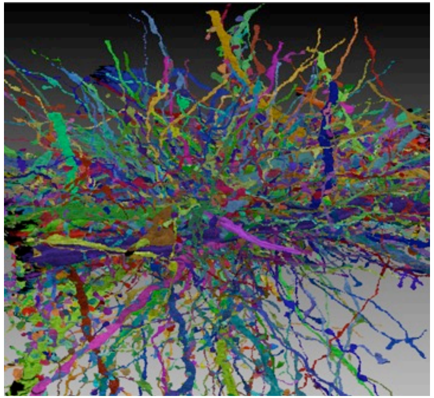
RESULTS: DEPTH COMPLEXITY



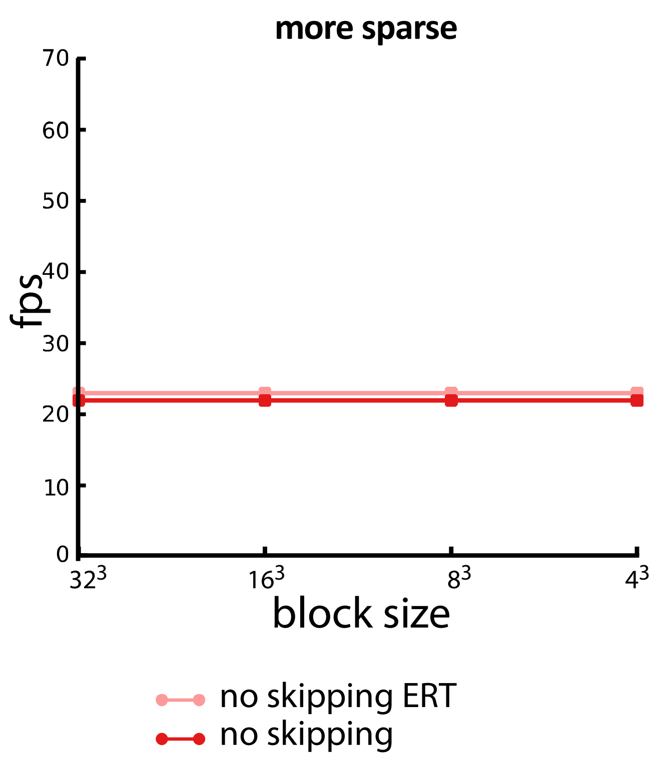
more sparse



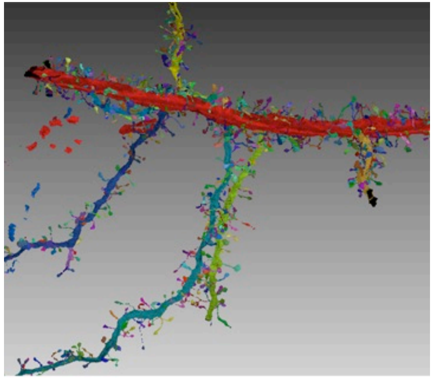
less sparse



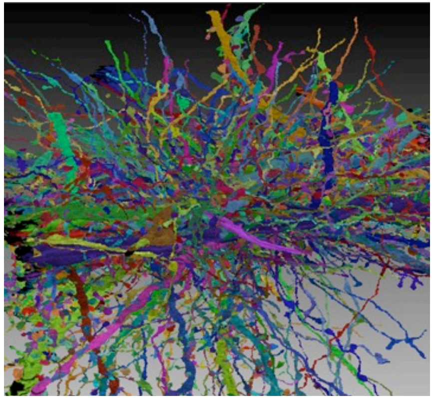
RESULTS: PERFORMANCE



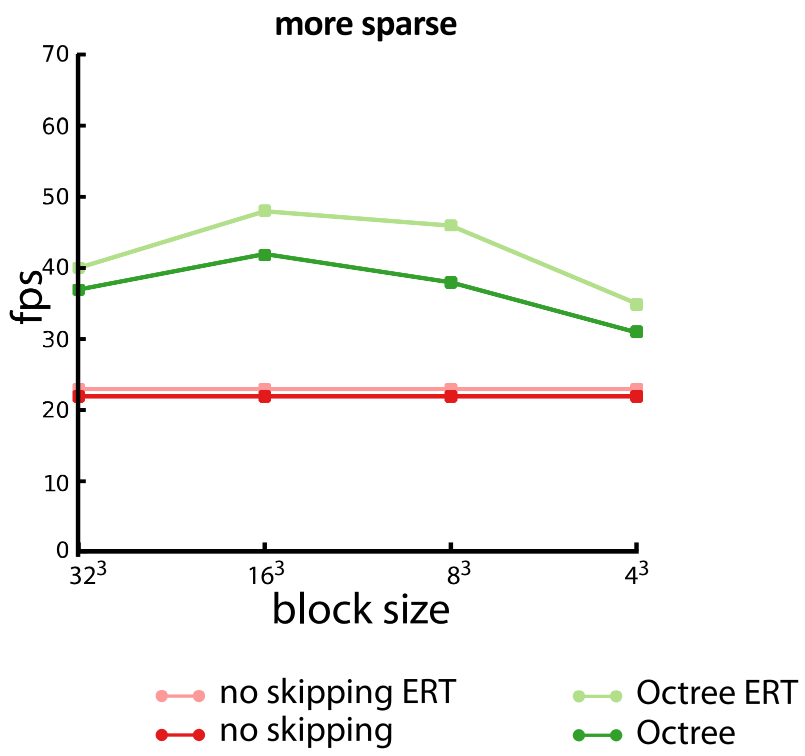
more sparse



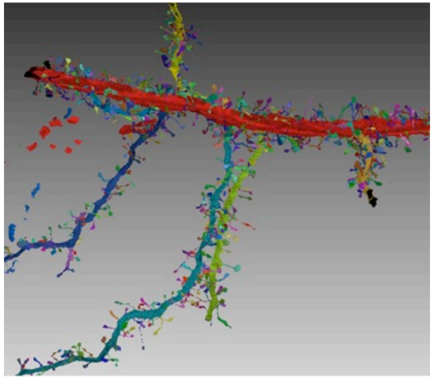
less sparse



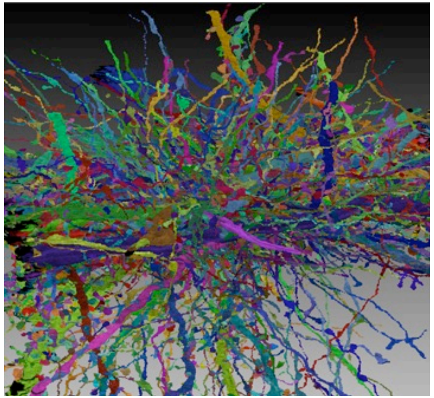
RESULTS: PERFORMANCE



more sparse



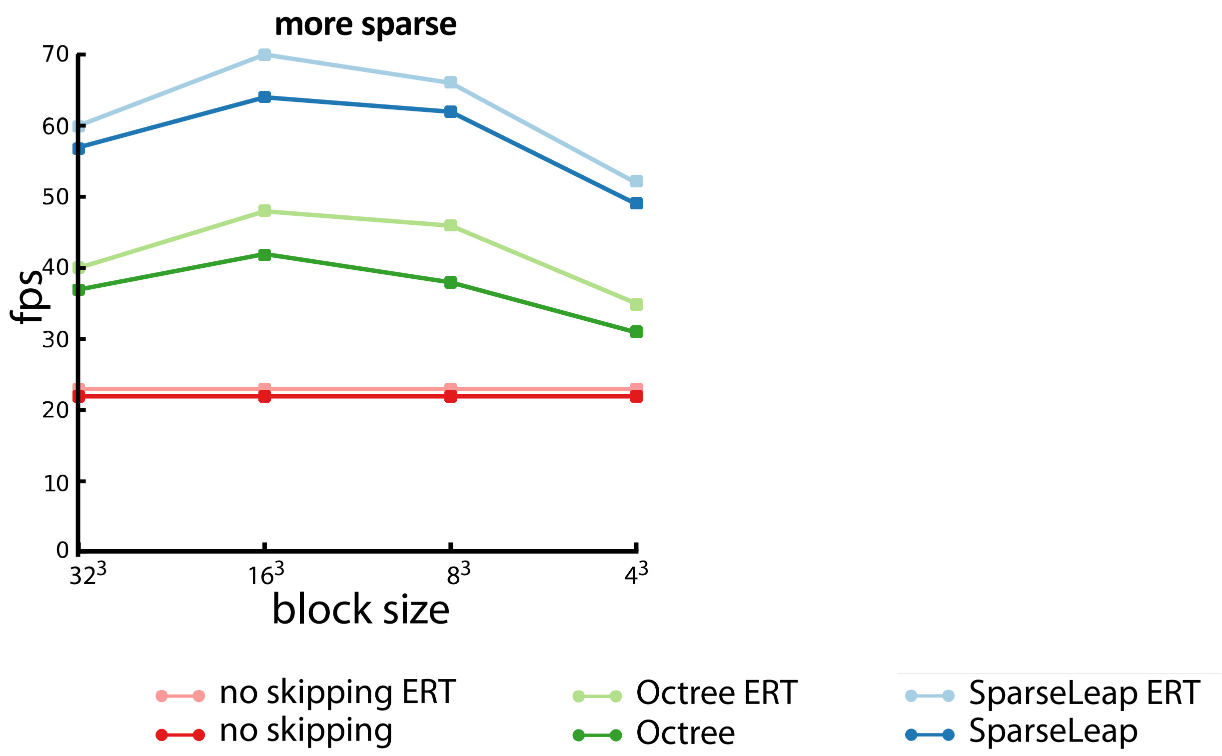
less sparse



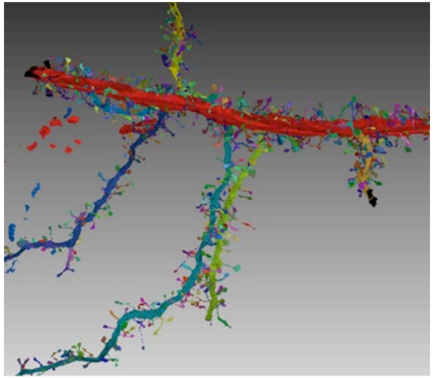
SIGGRAPH ASIA 2018 TOKYO

Sponsored by

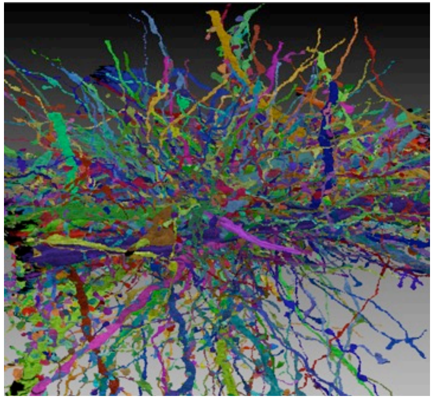
RESULTS: PERFORMANCE



more sparse



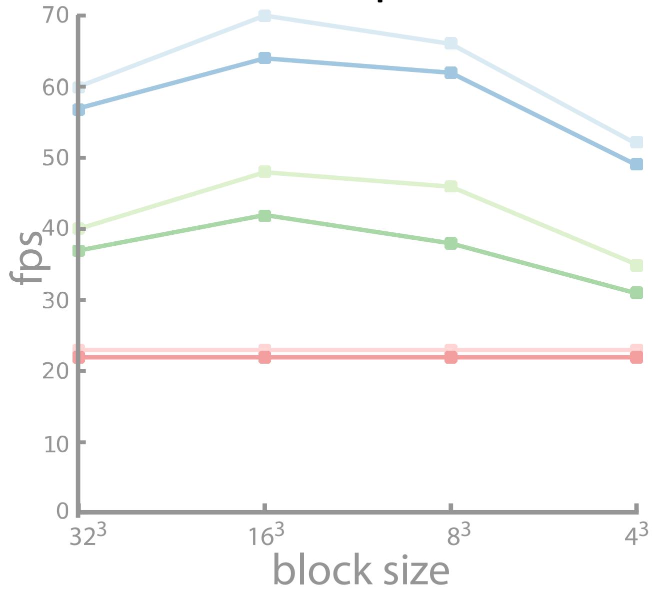
less sparse

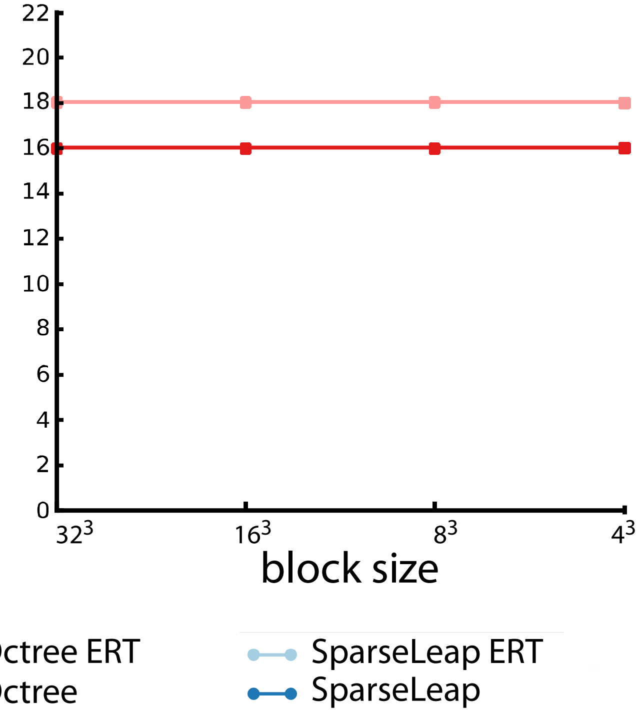


SIGGRAPH ASIA 2018 TOKYO

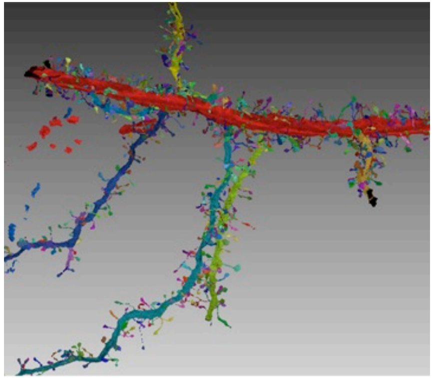
Sponsored by

RESULTS: PERFORMANCE

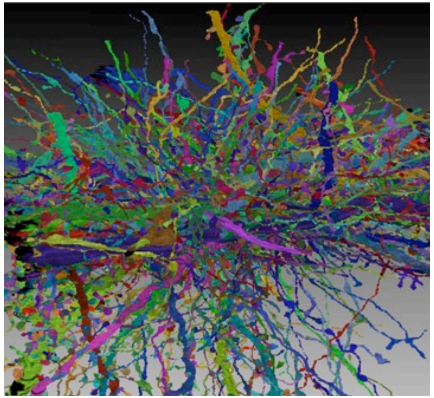




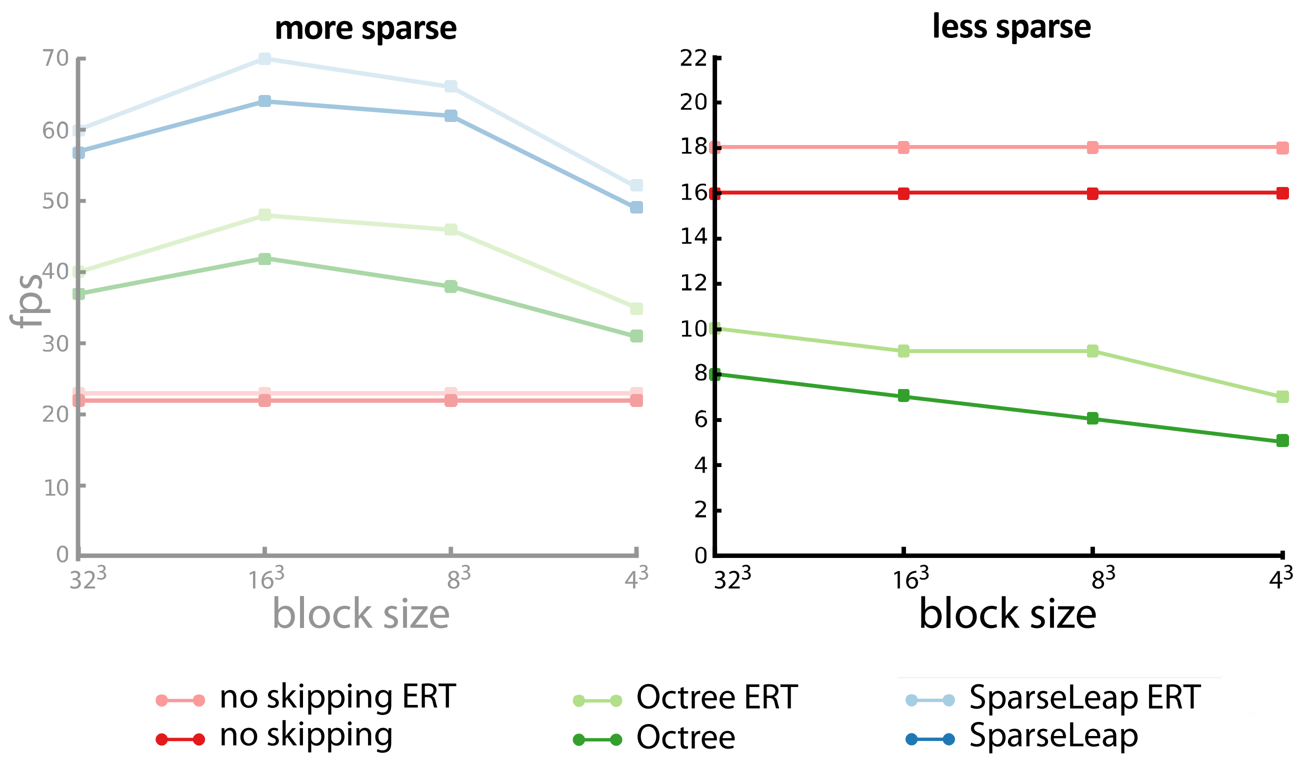
more sparse



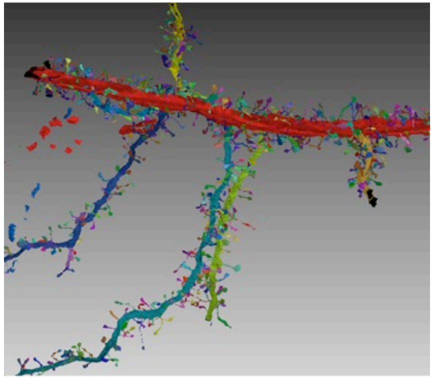
less sparse



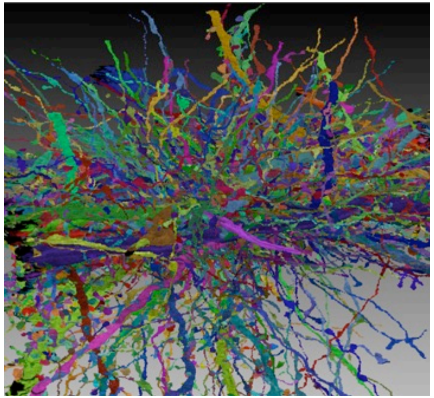
RESULTS: PERFORMANCE



more sparse



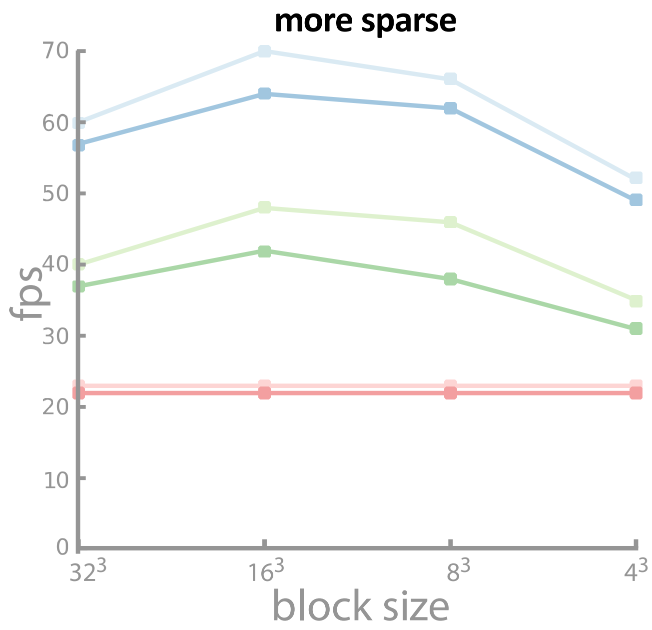
less sparse

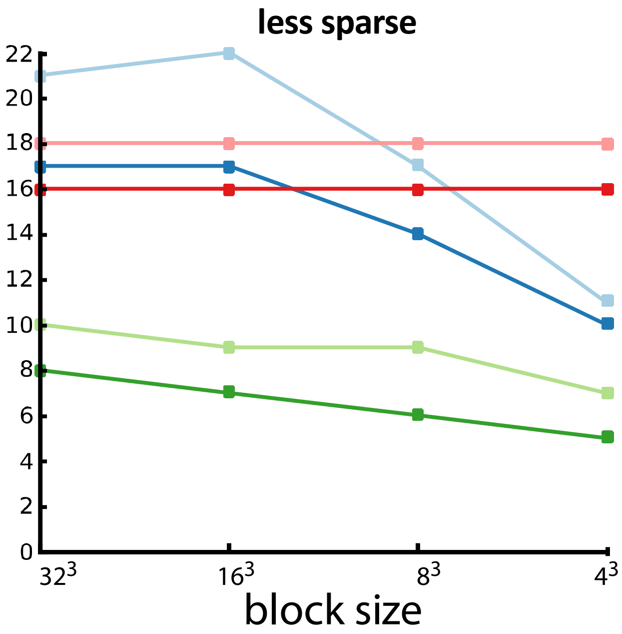


SIGGRAPH ASIA 2018 TOKYO

Sponsored by

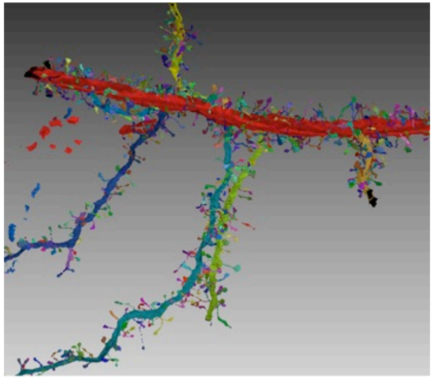
RESULTS: PERFORMANCE



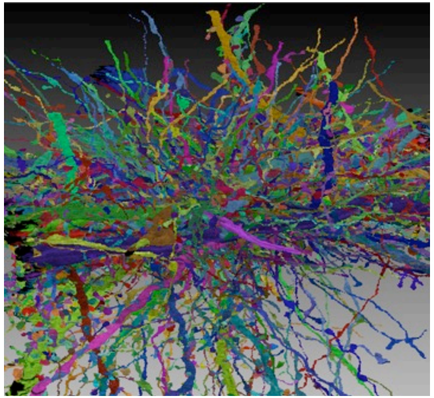


no skipping ERT
no skipping
Octree ERT
Octree
SparseLeap ERT
SparseLeap

more sparse



less sparse



SUMMARY

Cost of empty space skipping moved out of ray-casting loop

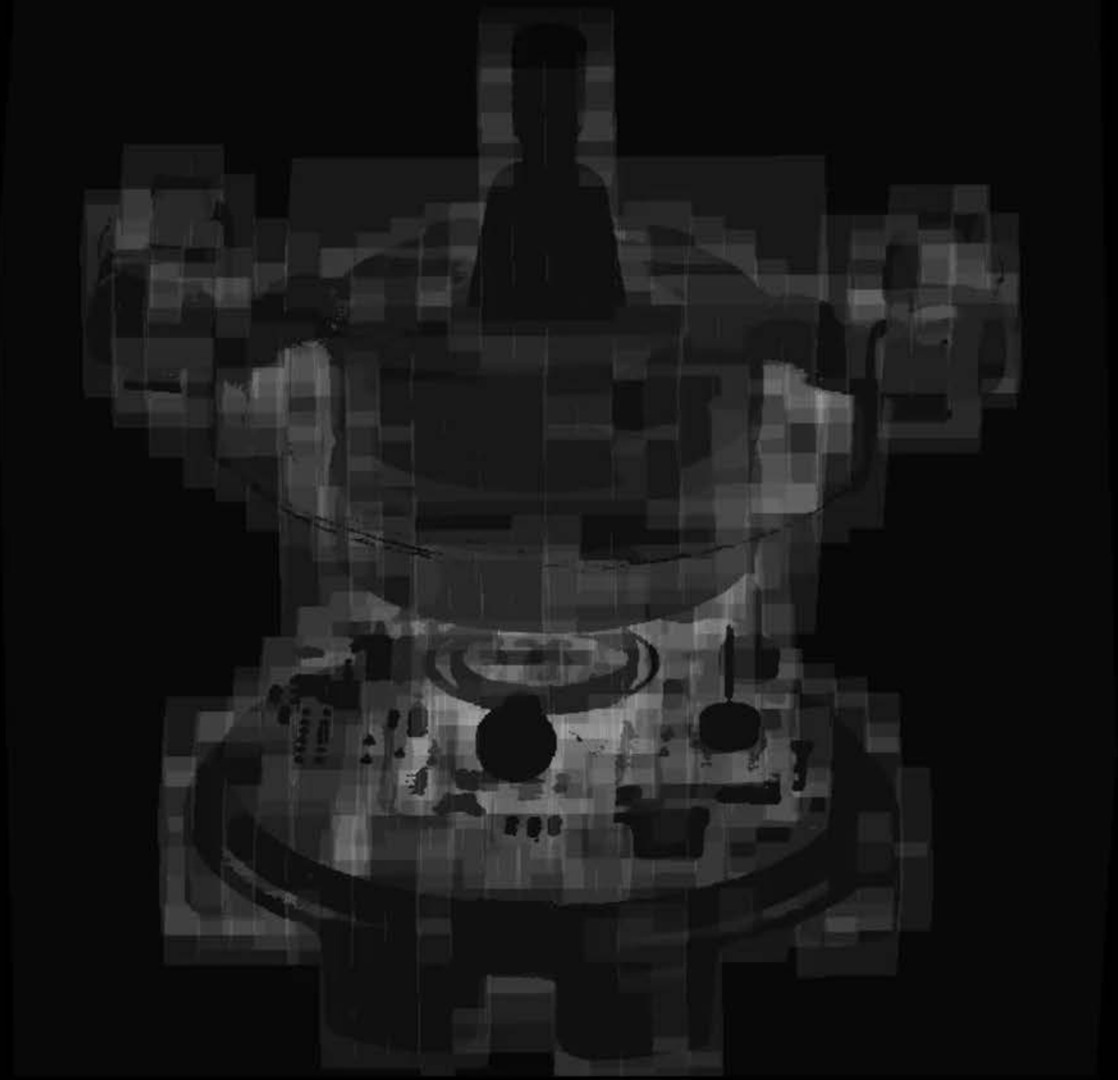
Attractive alternative for complex volumes

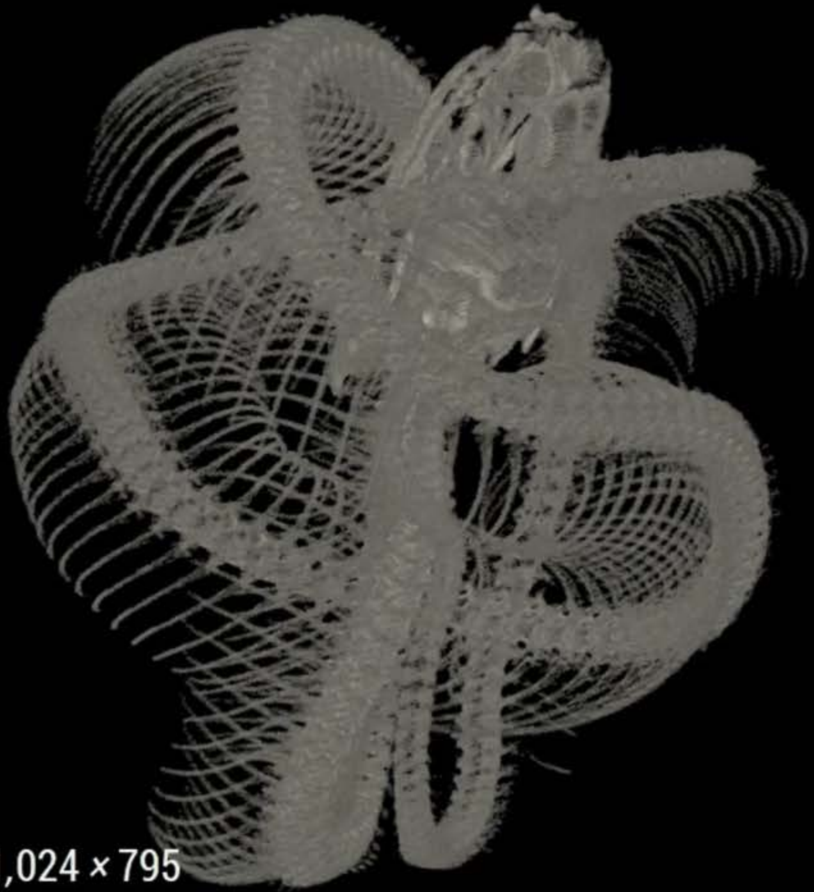
Memory consumption (GPU)

- Occupancy geometry: very low; much lower than octree storage
- Lists: depends on screen resolution and average depth complexity

Dreh Sensor data set: 2,048 × 2,048 × 2,048
85 segmented objects

SparseLeap
depth complexity





King Snake data set: 1,024 × 1,024 × 795
4 segmented objects

Part 4 -
Display-Aware Visualization and Processing

MOTIVATION


goal:
perform computations
at output resolution
250 megapixels
<1 megapixel visible


resolution level 0


resolution level 3

DISPLAY-AWARE IMAGE OPERATIONS

Input Resolution
(level 0)

Output Resolution
(level 3)

Display Region

Compute Resolution
(level 4)

Compute Region

IMAGE PYRAMIDS

- Dyadic image pyramids
  - Mipmaps [Williams 1983]: texture mapping (standard on GPUs)
  - Gaussian/Laplacian pyramids [Burt and Adelson 1983]: image processing/compression

level 0

level 1

level 2

level 3

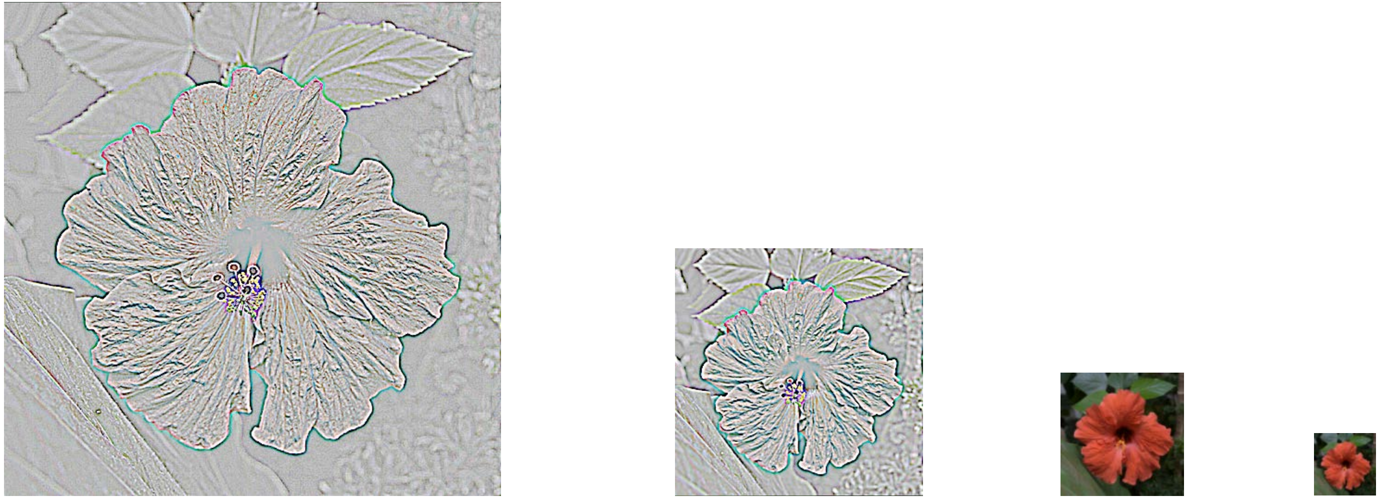
IMAGE PYRAMIDS

- Dyadic image pyramids
  - Mipmaps [Williams 1983]: texture mapping (standard on GPUs)
  - Gaussian/Laplacian pyramids [Burt and Adelson 1983]: image processing/compression

level 0 level 1 level 2 level 3

IMAGE PYRAMIDS

- Dyadic image pyramids
  - Mipmaps [Williams 1983]: texture mapping (standard on GPUs)
  - Gaussian/Laplacian pyramids [Burt and Adelson 1983]: image processing/compression

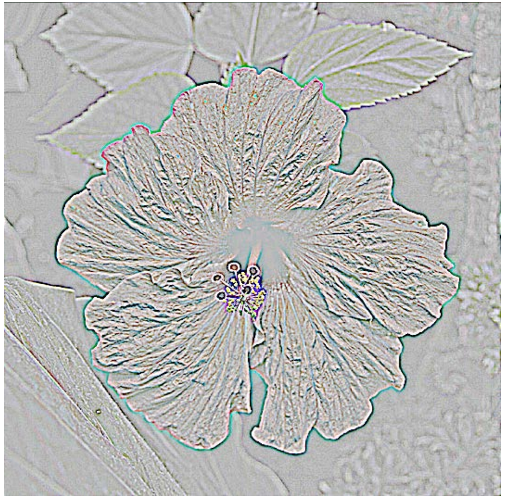


level 0 level 1 level 2 level 3

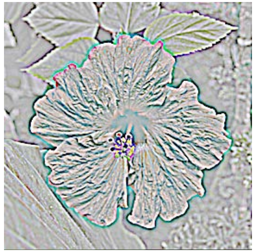
IMAGE PYRAMIDS

- Dyadic image pyramids
  - Mipmaps [Williams 1983]: texture mapping (standard on GPUs)
  - Gaussian/Laplacian pyramids [Burt and Adelson 1983]: image processing/compression
  - Sparse pdf maps [Hadwiger et al. 2012]

Laplacian pyramid



level 0



level 1

level 2

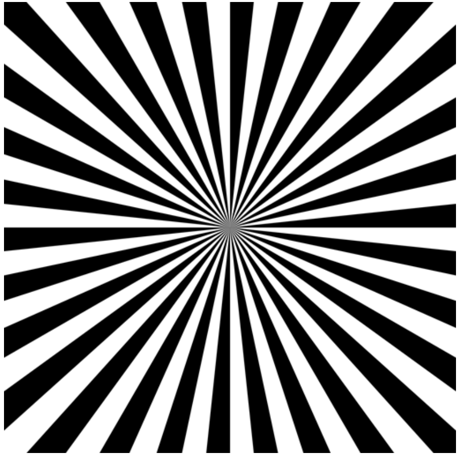
level 3

IMAGE PYRAMIDS

- Dyadic image pyramids
  - Mipmaps [Williams 1983]: texture mapping (standard on GPUs)
  - Gaussian/Laplacian pyramids [Burt and Adelson 1983]: image processing/compression
  - Sparse pdf maps [Hadwiger et al. 2012]

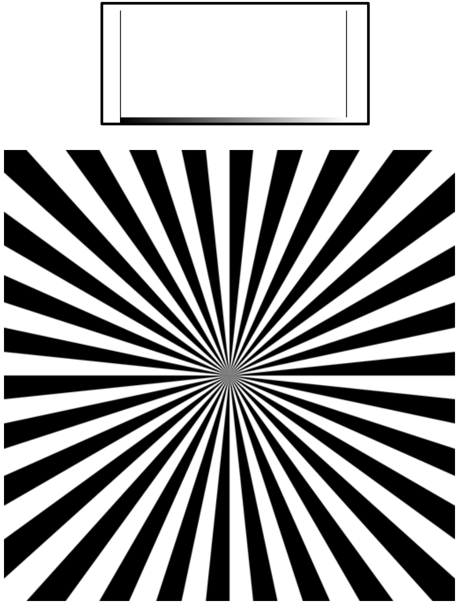
level 0 level 1 level 2 level 3

ANTI-ALIASING IN IMAGE PYRAMIDS

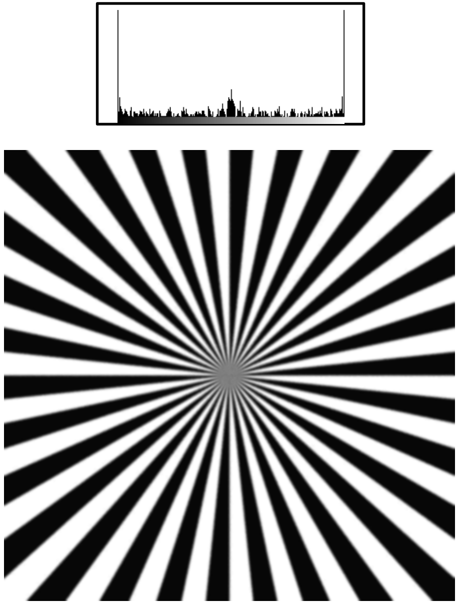


level 0

ANTI-ALIASING IN IMAGE PYRAMIDS

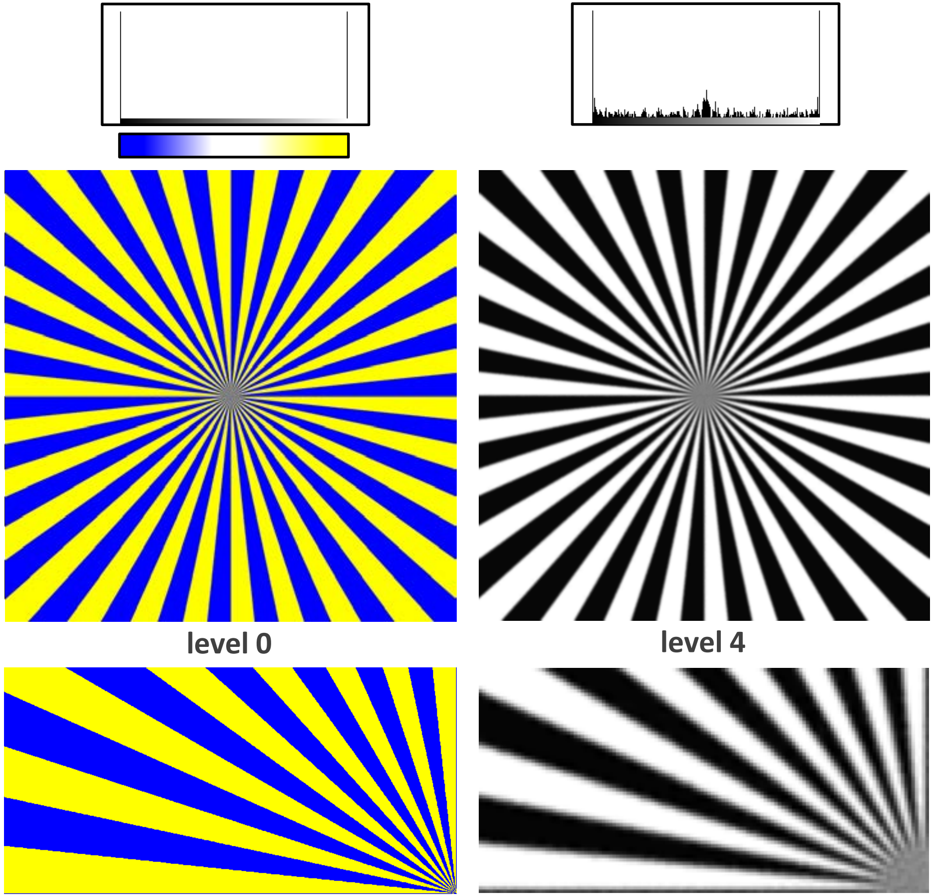


level 0

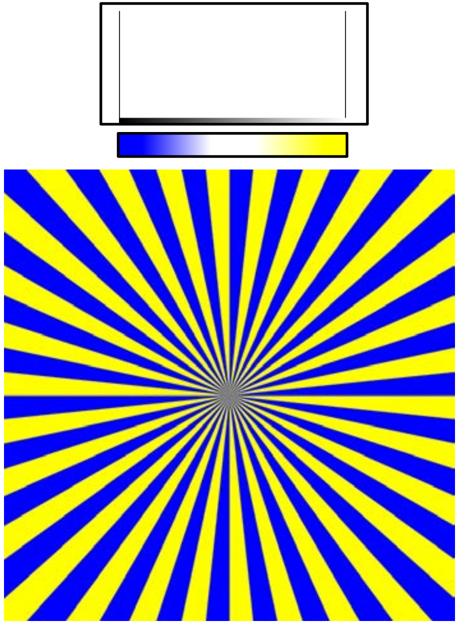


level 4

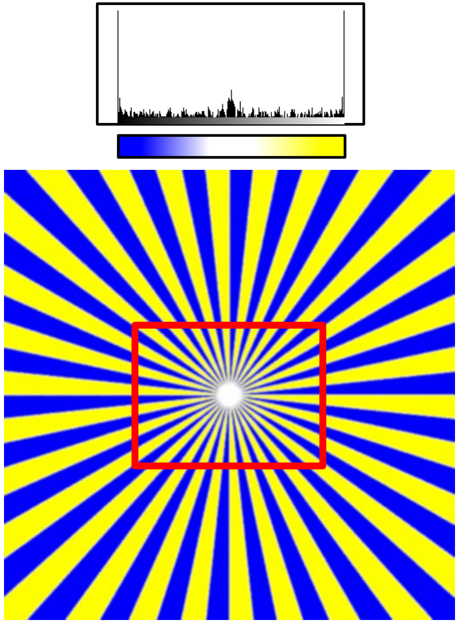
ANTI-ALIASING IN IMAGE PYRAMIDS



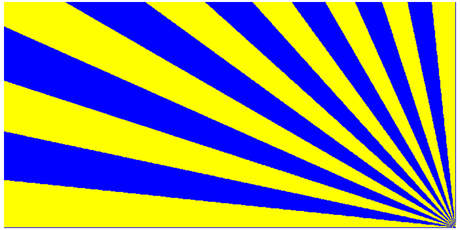
ANTI-ALIASING IN IMAGE PYRAMIDS

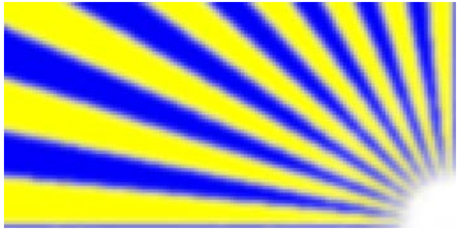


level 0

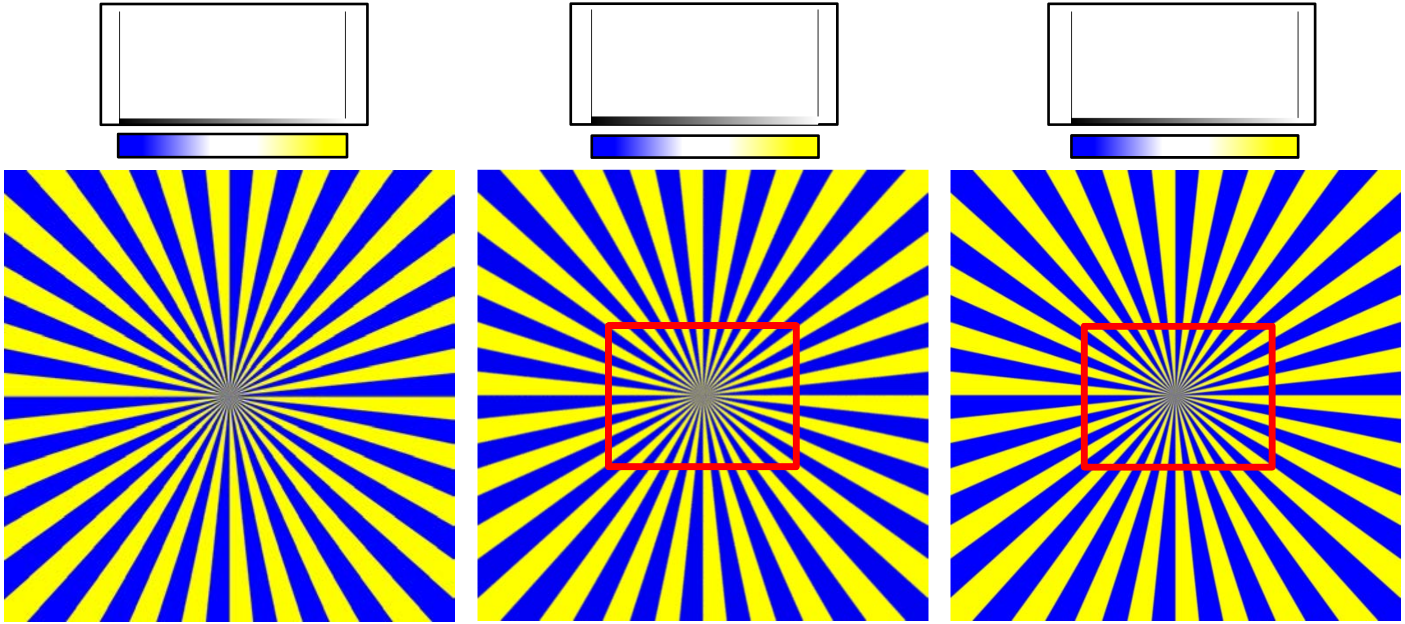


level 4, standard





ANTI-ALIASING IN IMAGE PYRAMIDS



level 0

level 4, sparse pdf maps

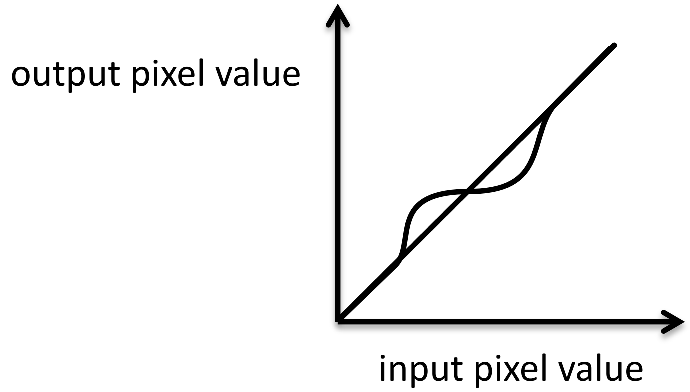
level 4, standard

level 4, ground truth

NON-LINEAR IMAGE OPERATORS

Apply non-linear operation to each pixel

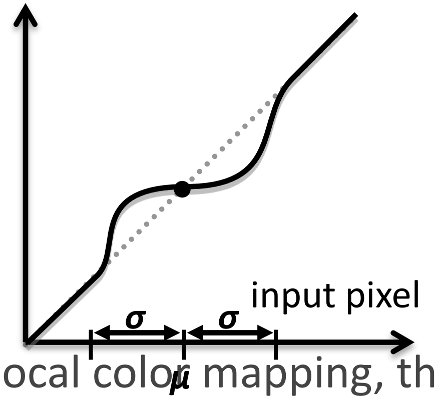
- Color map or non-linear contrast adjustment
- Bilateral filtering: range weight
- Smoothed local histogram filtering [Kass and Solomon 2010]
- Local Laplacian filtering [Paris et al. 2011]: point-wise, non-linear re-mapping

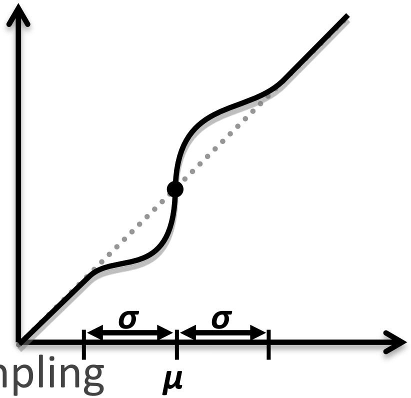


LOCAL LAPLACIAN FILTERING [PARIS ET AL. 2011]

Compute Laplacian pyramid coefficient

- Adjust local contrast via point-wise non-linearity; then downsample





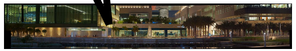
Same as local color mapping, then downsampling

- Cannot apply the re-mapping function to the downsampled image!
- Need to compute ground truth (pyramid!) or proper “anti-aliasing”

LOCAL LAPLACIAN FILTERING: SCALABILITY

- Night Scene Panorama: 47,908 x 7,531 pixels (361 Mpixels)

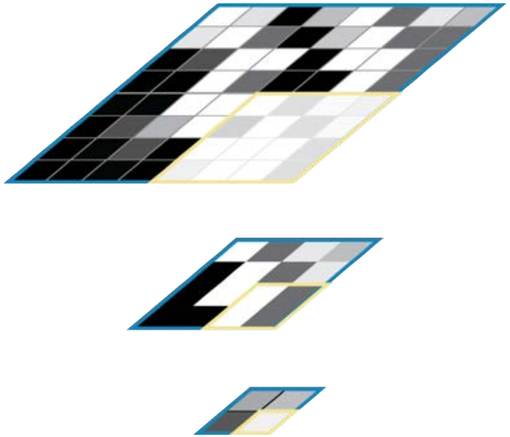
- Every downsampled pixel results from the entire pyramid above it
- Sparse PDF maps allow direct computation!



Sparse PDF Maps: Concept

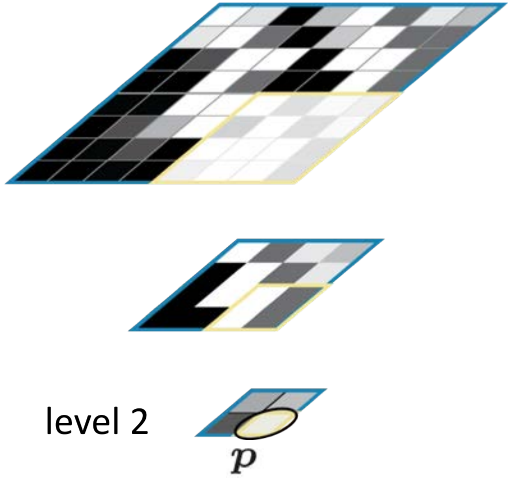
SPARSE PDF MAPS

Represent distribution of pixel values in footprint in original image



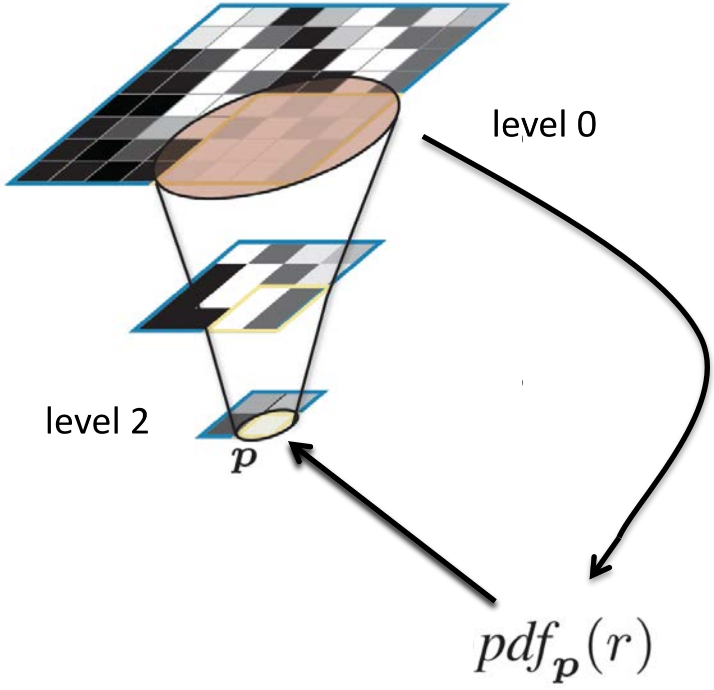
SPARSE PDF MAPS

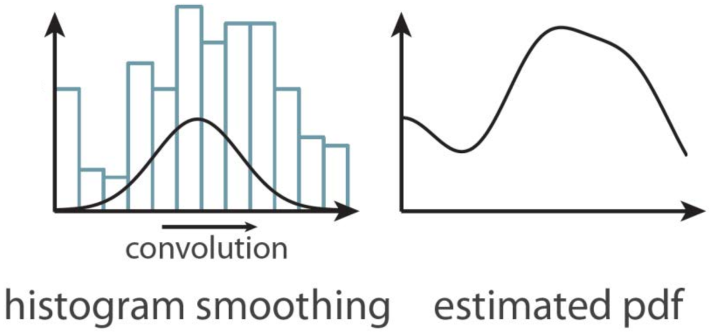
Represent distribution of pixel values in footprint in original image



SPARSE PDF MAPS

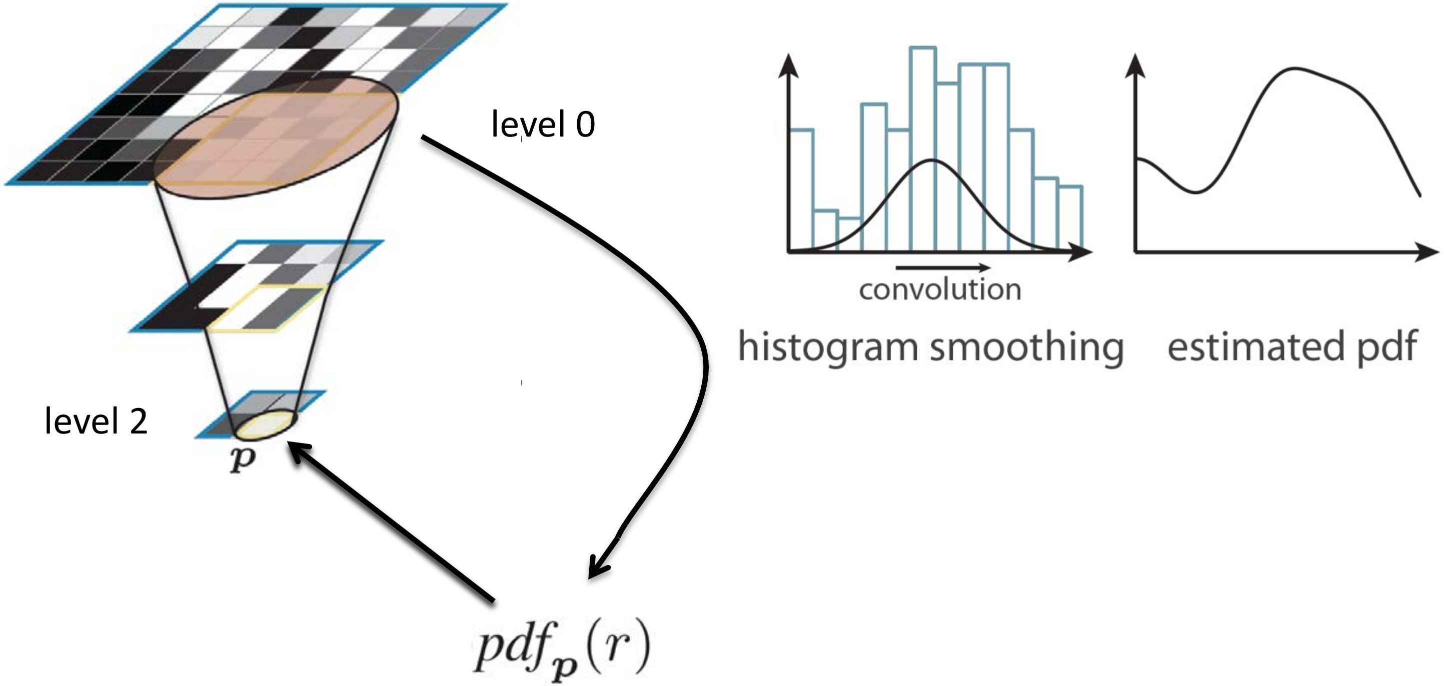
Represent distribution of pixel values in footprint in original image





SPARSE PDF MAPS

Represent distribution of pixel values in footprint in original image

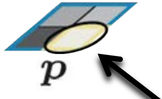


SPARSE PDF MAPS

Represent distribution of pixel values in footprint in original image

Apply non-linear operation

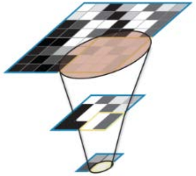
level 2



$$E\boxed{\left[t_{\boldsymbol{p}}\left(X_{\boldsymbol{p}}\right)\right]} = \frac{1}{w_{\boldsymbol{p}}}\int_0^1 \boxed{t_{\boldsymbol{p}}(r)}\,\boxed{pdf_{\boldsymbol{p}}(r)}\,dr$$

EXAMPLE 1: DOWNSAMPLED IMAGE

$$E\left[t_{\mathbf{p}}\left(X_{\mathbf{p}}\right)\right] = \frac{1}{w_{\mathbf{p}}} \int_0^1 t_{\mathbf{p}}(r) pdf_{\mathbf{p}}(r)\, dr$$

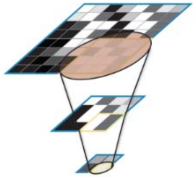


$$t_{\mathbf{p}}(r) = r$$

$$w_{\mathbf{p}} = 1$$

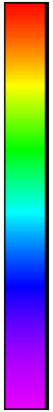
EXAMPLE 2: COLOR MAPPING

$$E\left[t_{\mathbf{p}}\left(X_{\mathbf{p}}\right)\right] = \frac{1}{w_{\mathbf{p}}} \int_0^1 t_{\mathbf{p}}(r) pdf_{\mathbf{p}}(r)\, dr$$



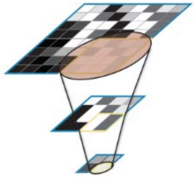
$t_{\mathbf{p}}(r) =$ color map

$w_{\mathbf{p}} = 1$



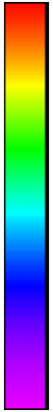
EXAMPLE 2: COLOR MAPPING

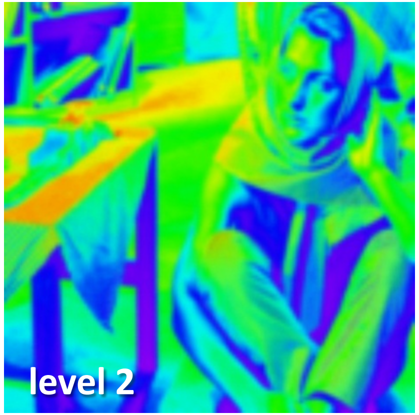
$$E\left[t_{\mathbf{p}}\left(X_{\mathbf{p}}\right)\right] = \frac{1}{w_{\mathbf{p}}} \int_{0}^{1} t_{\mathbf{p}}(r) pdf_{\mathbf{p}}(r)\, dr$$



$t_{\mathbf{p}}(r) =$ color map

$w_{\mathbf{p}} = 1$





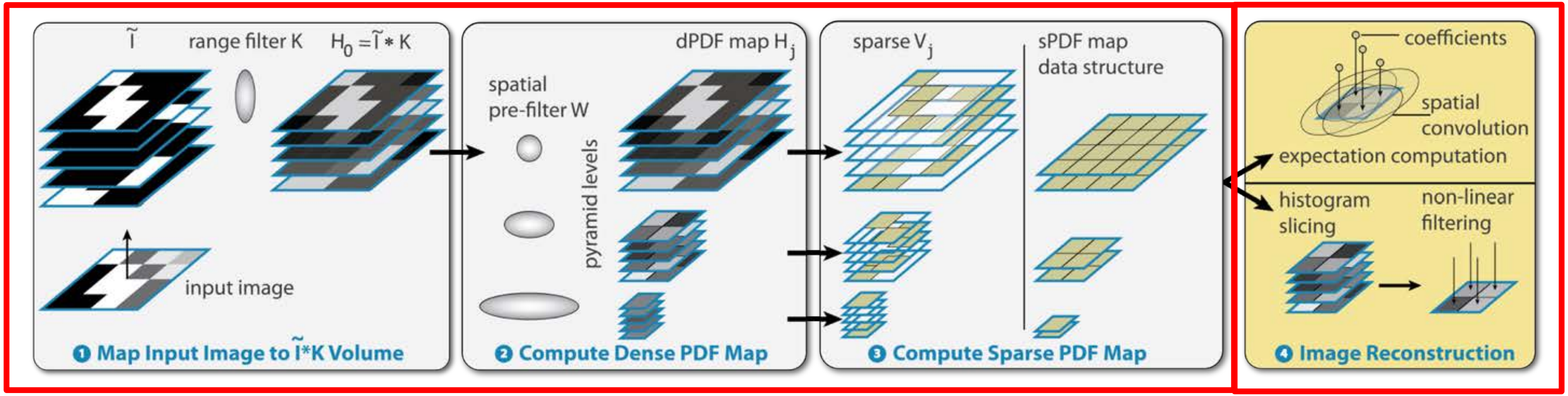
plus: bilateral filtering, local Laplacian filtering in linear time, …

INTERACTIVE GIGAPIXEL FILTERING

Sponsored by

Sparse PDF Map Computation

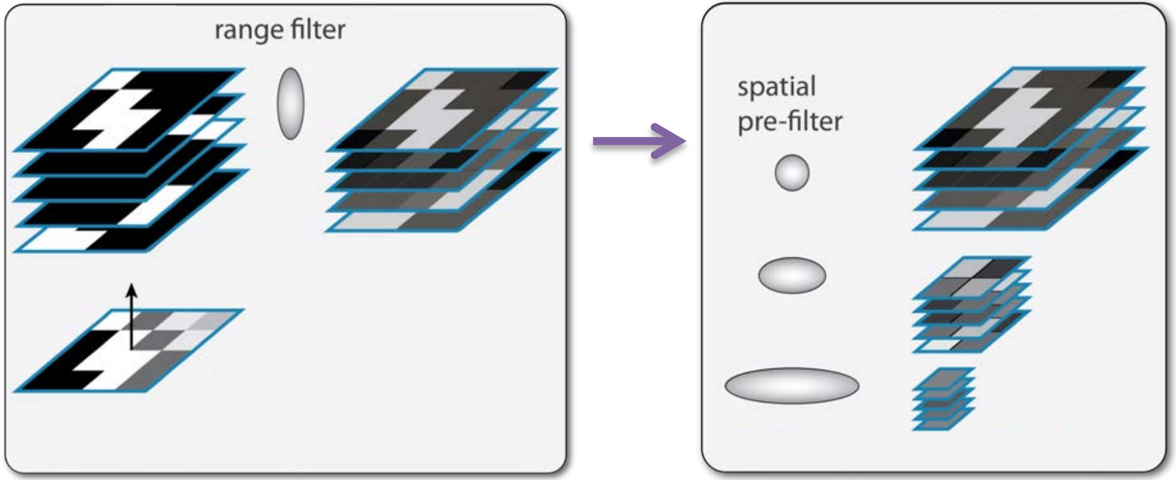
PIPELINE



Ĩ
range filter K
$H_0 = \tilde{I} * K$
input image
1 Map Input Image to Ĩ*K Volume
dPDF map H_j
spatial pre-filter W
pyramid levels
2 Compute Dense PDF Map
sparse V_j
sPDF map data structure
3 Compute Sparse PDF Map
coefficients
spatial convolution
expectation computation
histogram slicing
non-linear filtering
4 Image Reconstruction

STEP 1: DENSE PDF MAP

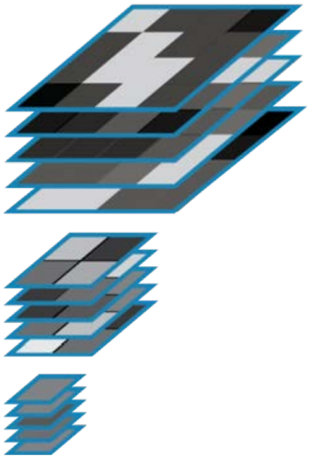
DENSE PDF MAP COMPUTATION



H is similar to a pyramid of bilateral grids [Chen et al. 2007]

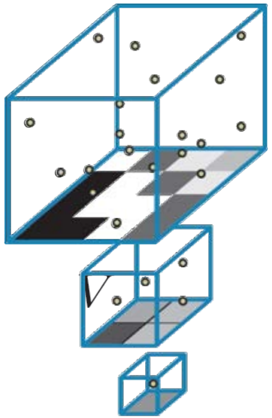
STEP 2: SPARSE PDF MAP

SPARSE REPRESENTATION

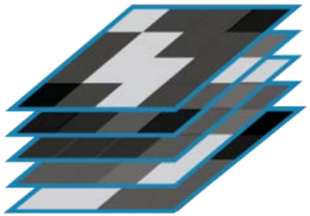


$$\boxed{H} \approx \boxed{V}\boxed{* \, (W \otimes K)}$$

$$G_{\boldsymbol{\sigma}_s} \otimes G_{\sigma_r}$$



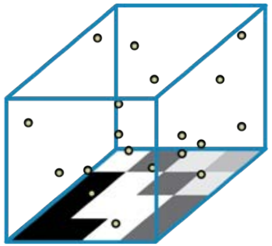
SPARSE REPRESENTATION



$$H \approx V * (W \otimes K)$$

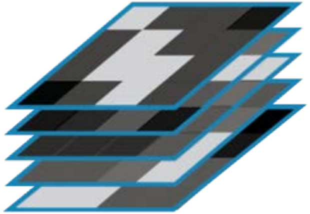
$$W \leftarrow G_{\boldsymbol{\sigma}_s}, \quad K \leftarrow G_{\sigma_r}$$

$$G_{\boldsymbol{\sigma}_s} \otimes G_{\sigma_r}$$

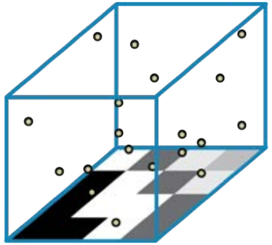


Compute via Matching Pursuit [Mallat and Zhang 1993]

SPARSE REPRESENTATION

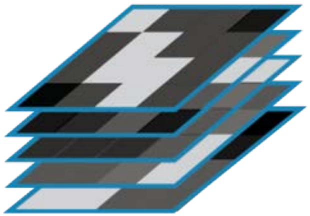


$$H \approx V * (W \otimes K)$$

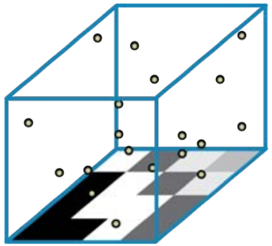


$$V(\boxed{\boldsymbol{p}_n}, \boxed{r_n}) = \boxed{c_n} \qquad c_n \neq 0$$

SPARSE REPRESENTATION



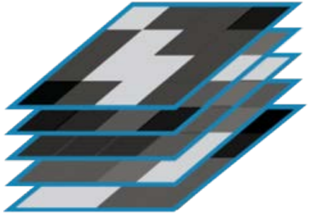
$$H \approx V * (W \otimes K)$$



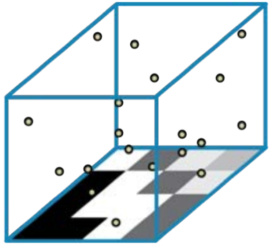
$$V(\boldsymbol{p}_n, r_n) = c_n \qquad c_n \neq 0$$

$$(\boldsymbol{p}_n, r_n, c_n)$$

SPARSE REPRESENTATION



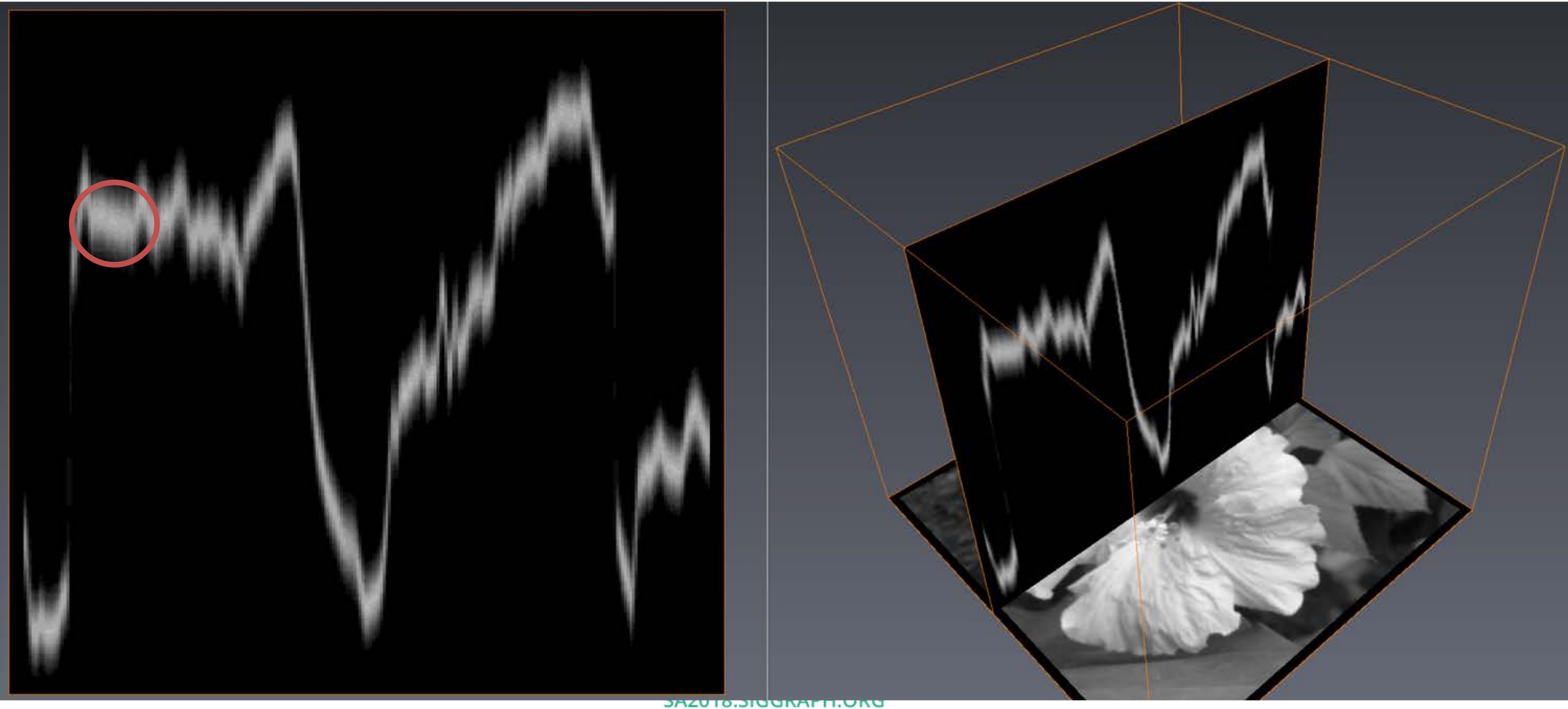
$$H \approx V * (W \otimes K)$$



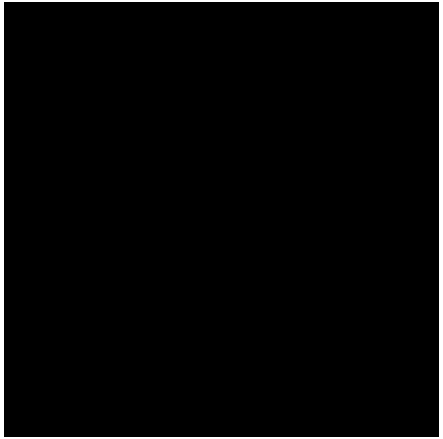
$$V(\boldsymbol{p}_n, r_n) = c_n \qquad c_n \neq 0$$

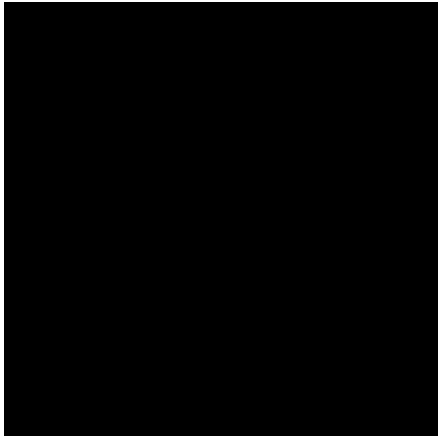
$$(\boldsymbol{p}_n, r_n, c_n)$$

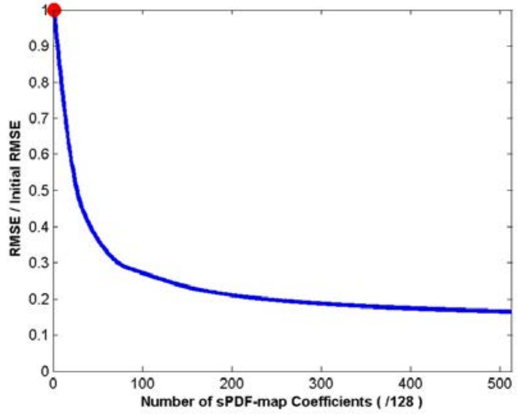
SPATIAL AND RANGE COHERENCE

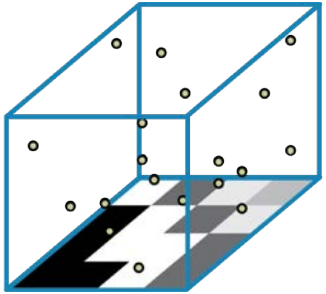


GREEDY APPROXIMATION







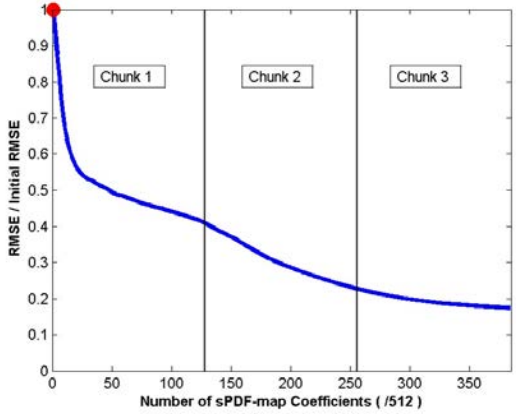


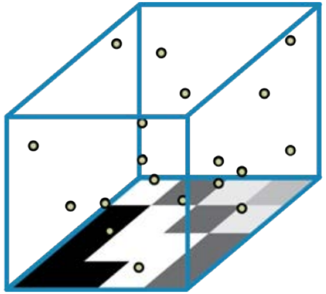
Spatial filter W : 5 x 5

1 coefficient chunk

(# coefficients == 1 * # pixels)

GREEDY APPROXIMATION



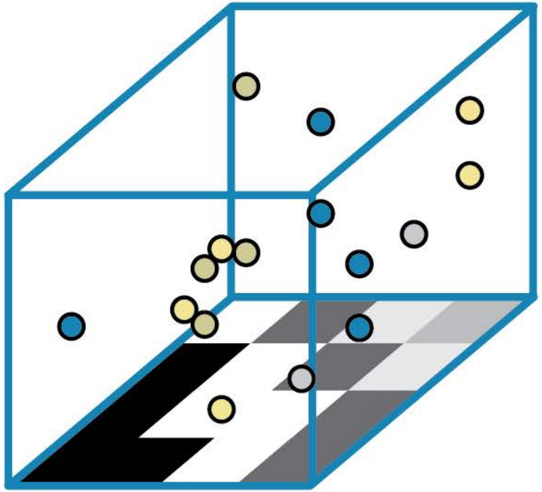


Spatial filter W : 3 x 3

1-3 coefficient chunks
(# coefficients == 1-3 * # pixels)

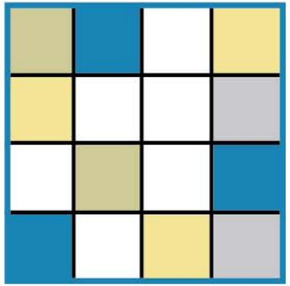
sPDF-Maps Data Structure

SPDF-MAPS DATA STRUCTURE



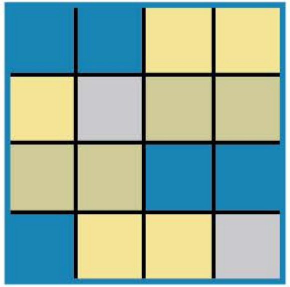
conceptual

$$V(\boldsymbol{p}_n, r_n) = c_n$$



index image

$$(index, count)_{\boldsymbol{p}}$$

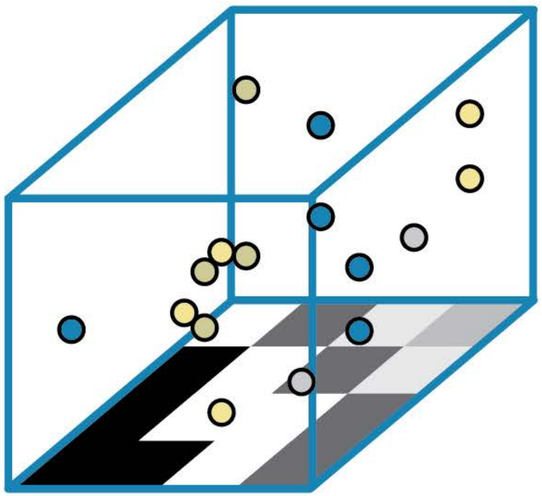


coefficient image

$$(r_n, c_n)$$

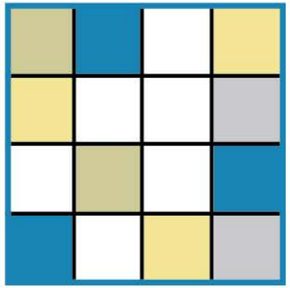
Sponsored by

SPDF-MAPS DATA STRUCTURE



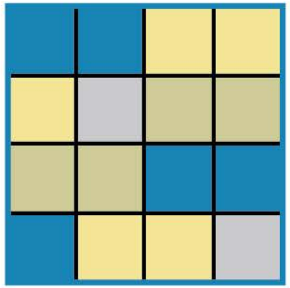
conceptual

$$V(\boldsymbol{p}_n, r_n) = c_n$$



index image

$$(index, count)_{\boldsymbol{p}}$$



coefficient image

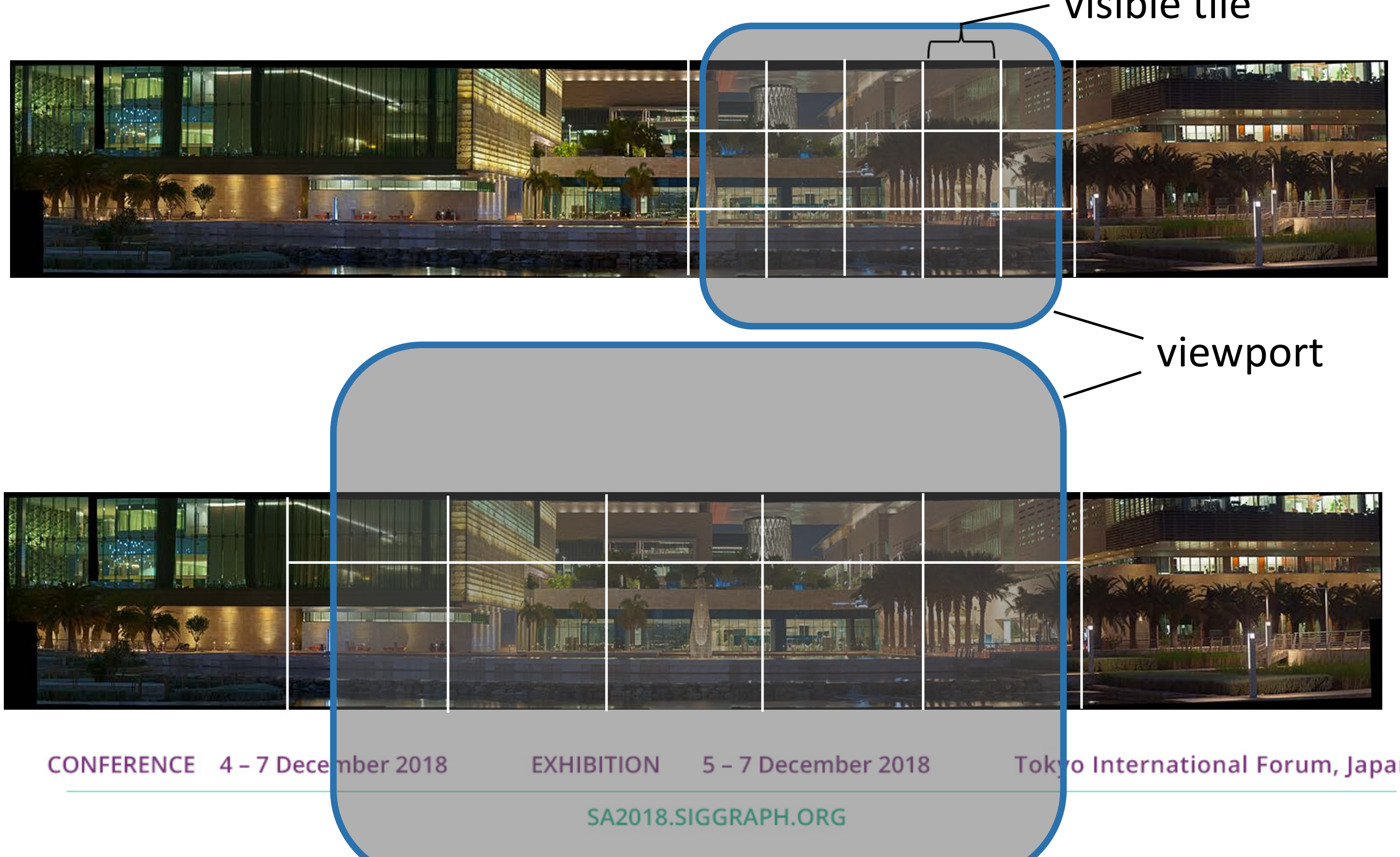
$$(r_n, c_n)$$

Display-Aware Gigapixel Image Processing

GIGAPIXEL IMAGE PROCESSING

- Out-of-Core Processing
  - Divide data into smaller tiles, process each tile independently (e.g., 256x256)
  - Image operations are performed only on requested sub-tiles (display-aware)
  - Rendering based on tiled data, using GPU-based virtual memory approach

GIGAPIXEL IMAGE PROCESSING

- 

GIGAPIXEL IMAGE PROCESSING

- GPU-based virtual memory architecture [Hadwiger et al. 2012]

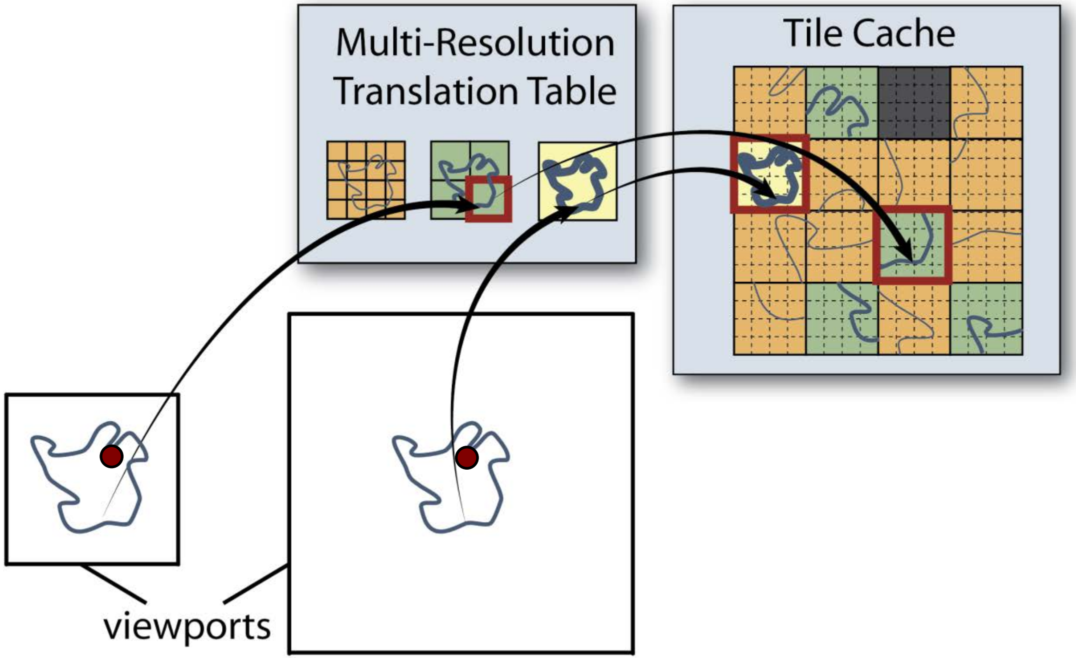


Image Reconstruction

IMAGE RECONSTRUCTION

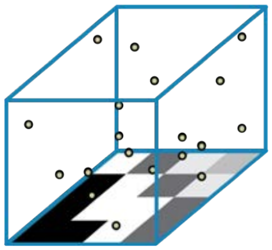
$$E\left[t_{\mathbf{p}}\left(X_{\mathbf{p}}\right)\right] = \frac{1}{w_{\mathbf{p}}} \int_0^1 t_{\mathbf{p}}(r) pdf_{\mathbf{p}}(r)\, dr$$

- Key idea # 1

  Use $V * (W \otimes K)$ instead of $pdf_{\mathbf{p}}(r)$

- Key idea # 2

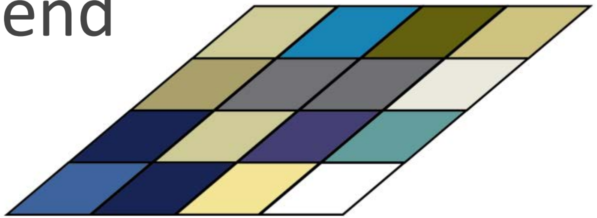
  Pre-convolve $t_{\mathbf{p}}(r)$ with K: $\tilde{t}_{\mathbf{p}} = t_{\mathbf{p}} * K$



V

COLOR MAPPING

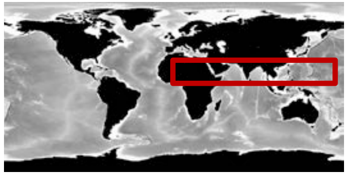
- Pre-convolve color map (with range kernel)
- For each pixel go over its coefficients
  - Apply color map to coefficient and sum up (not spatially convolved yet!)
- One spatial convolution in the end

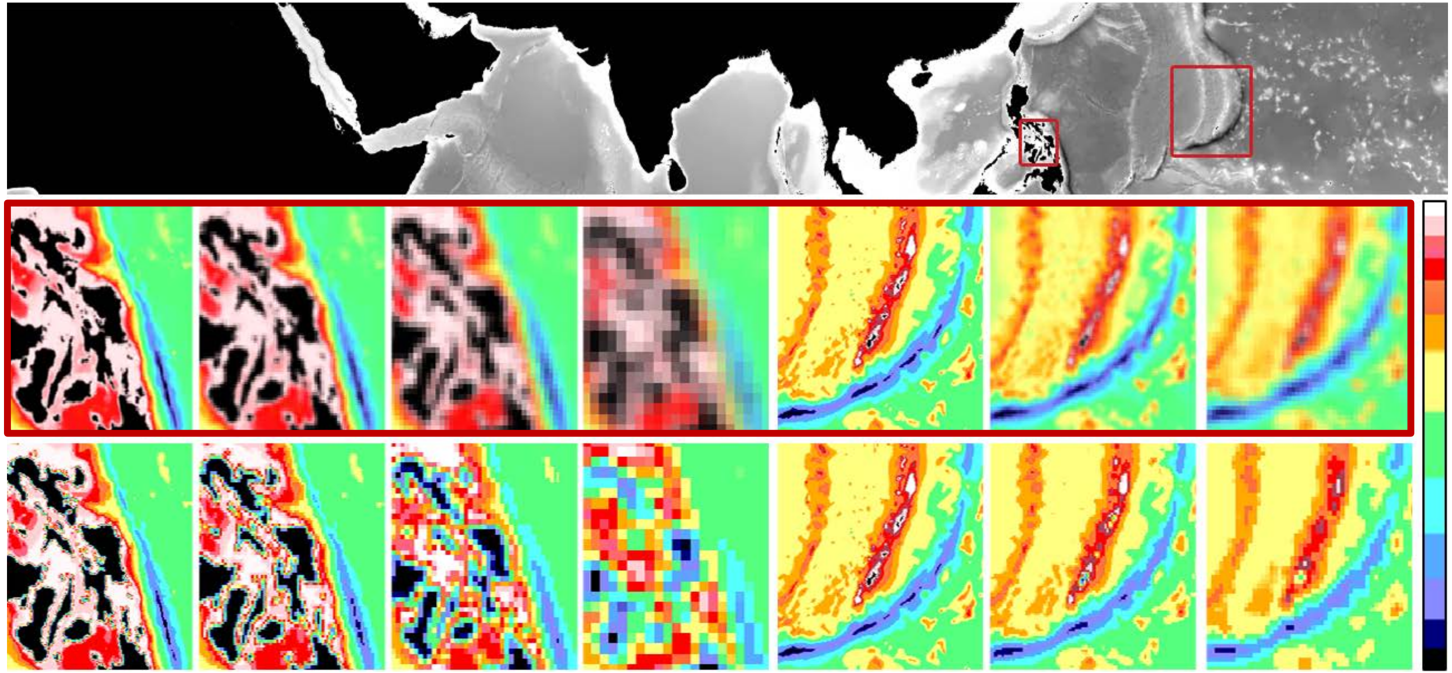


Results

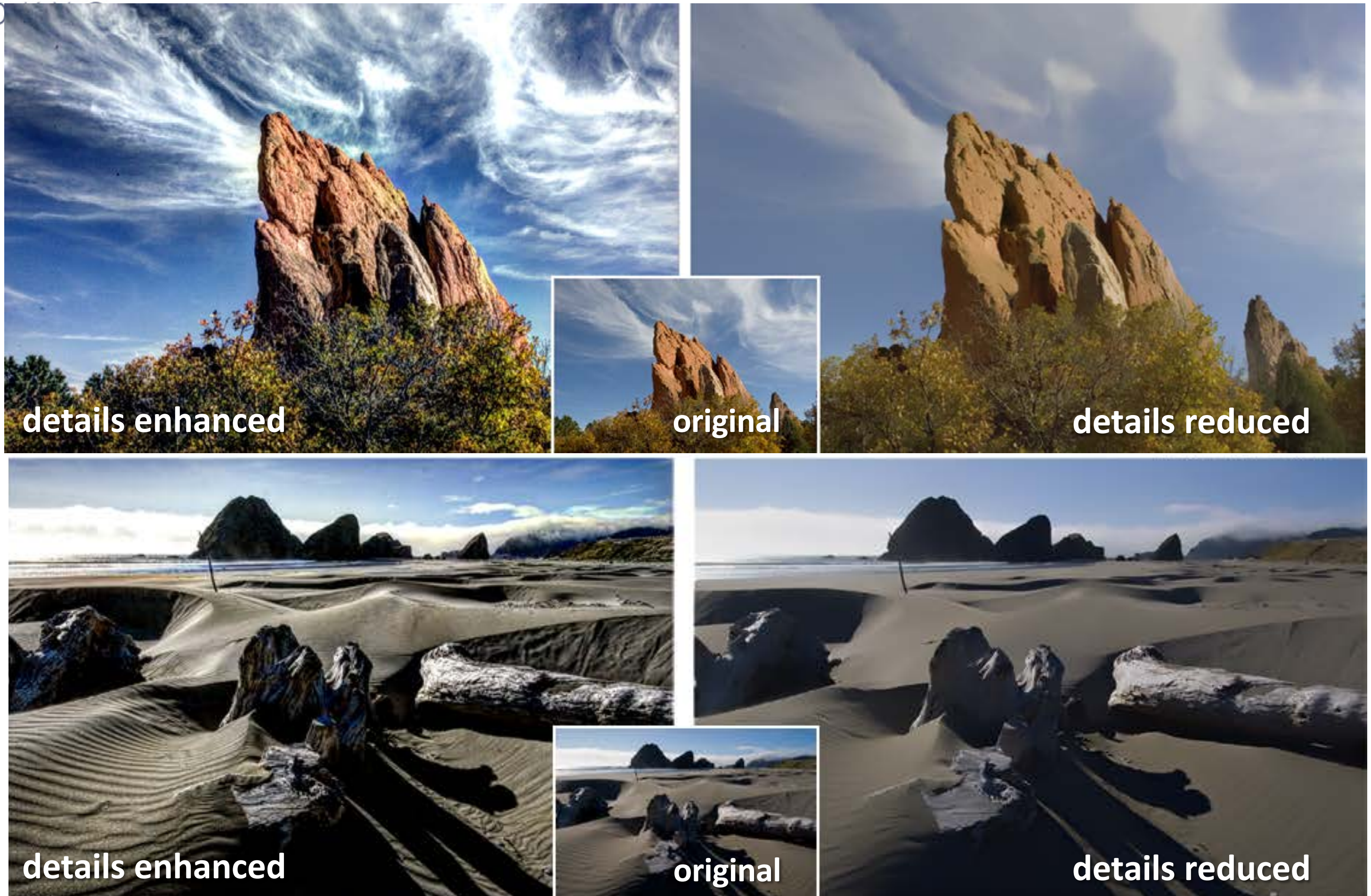
COLOR MAPPING GIGAPIXEL IMAGES

NASA Blue Marble bathymetry: 21,601 x 10,801 pixels (233 Mpixels)





Sponsored by



GIGAPIXEL LOCAL LAPLACIAN FILTERING


original


details reduced


details enhanced

SUMMARY

Display-aware processing with flexible new image pyramid (spdf map)
- Consistent, sparse representation of pixel footprint pdfs

Unified evaluation of many important non-linear image operations
- Local Laplacian filtering for gigapixel images

Efficient CUDA implementation

Pre-computation costly, but only performed once

Run time storage and computation similar to standard pyramids

Hadwiger, Sicat, Beyer, Krüger, Möller,
Sparse PDF Maps for Non-Linear Multi-Resolution Image Operations,
Siggraph Asia 2012

Sponsored by

SIGGRAPH ASIA 2018 TOKYO

CONFERENCE 4 – 7 December 2018
EXHIBITION 5 – 7 December 2018
Tokyo International Forum, Japan
SA2018.SIGGRAPH.ORG

THANK YOU!

Johanna Beyer, Harvard University
Markus Hadwiger, KAUST

Course Website:
http://johanna-b.github.io/LargeSciVis2018/index.html