SIGGRAPH ASIA 2018 TOKYO

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

Sponsored by

GPU-Based Large-Scale Scientific Visualization

Johanna Beyer, Harvard University

Markus Hadwiger, KAUST

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

Part 2 -
Scalable Volume Visualization Architectures and Applications

SIGGRAPH ASIA 2018 TOKYO

PART 2 – SCALABLE ARCHITECTURES & APPLICATIONS

History

Categorization

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

Ray-Guided Volume Rendering Examples

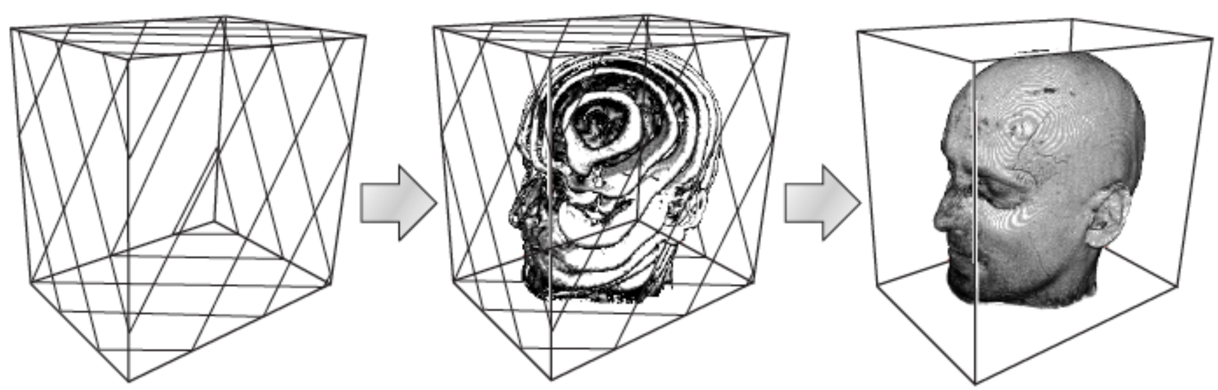
Summary

HISTORY (1)

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

+ Minimal hardware requirements

- Visual artifacts, less flexibility

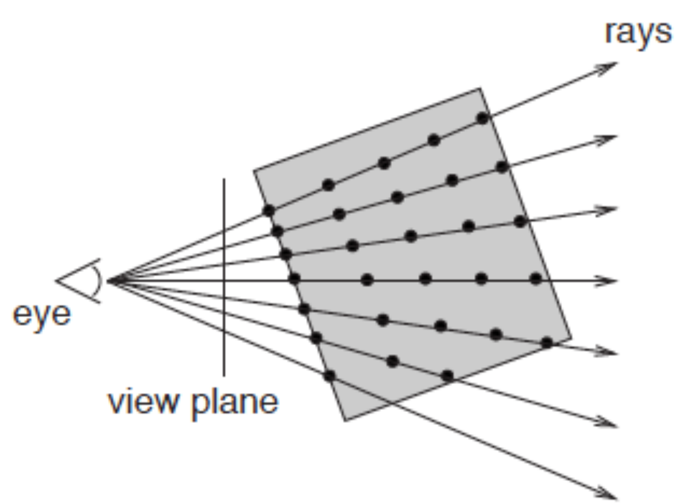


HISTORY (2)

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]
- Ray-guided volume rendering [Crassin et al. ‘09]
- Optimized CPU ray-casting [Knoll et al. ’11]
- Multi-level page tables [Hadwiger et al. ‘12]

Examples

OCTREE RENDERING AND TEXTURE SLICING

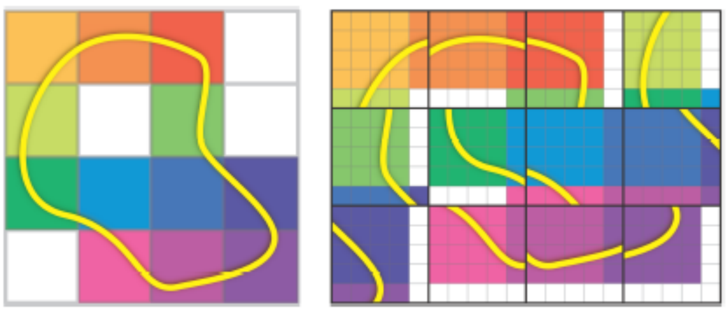
- GPU 3D texture mapping with arbitrary levels of detail
- Consistent interpolation between adjacent resolution levels
- Adapting slice distance with respect to desired LOD (needs opacity correction)
- LOD based on user-defined focus point

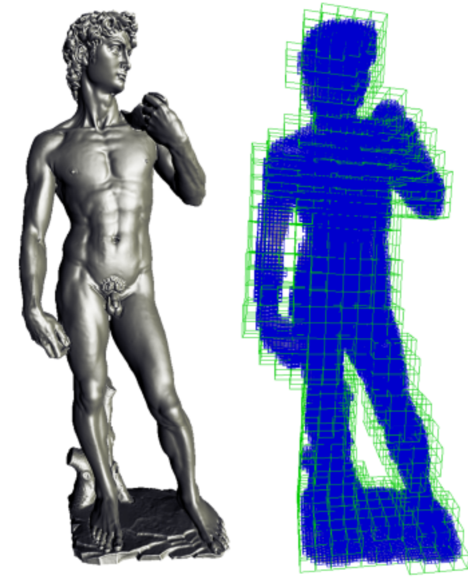
[Weiler et al., IEEE Symp. Vol Vis 2000]
Level-Of-Detail Volume Rendering via 3D Textures

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

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

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

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

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

CATEGORIZATION OF SCALABLE VOLUME RENDERING APPROACHES

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!

SIGGRAPH ASIA 2018 TOKYO

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?

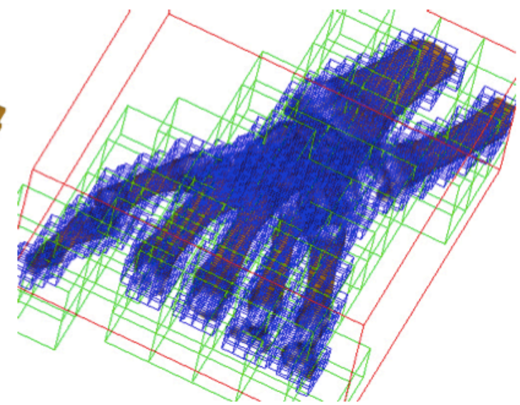
GLOBAL ATTRIBUTE-BASED CULLING

Cull bricks based on attributes; view-independent

- Transfer function
- Iso value
- Enabled segmented objects

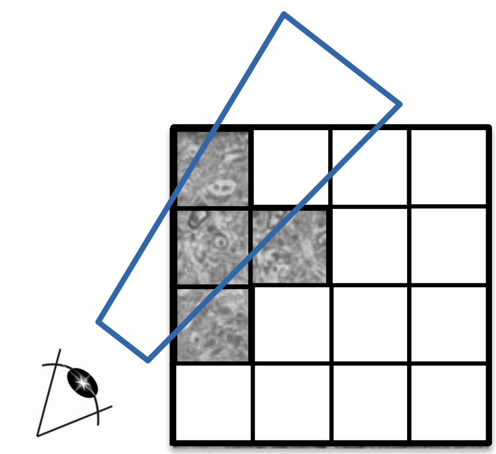
Often based on min/max bricks

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



VIEW FRUSTUM, OCCLUSION CULLING

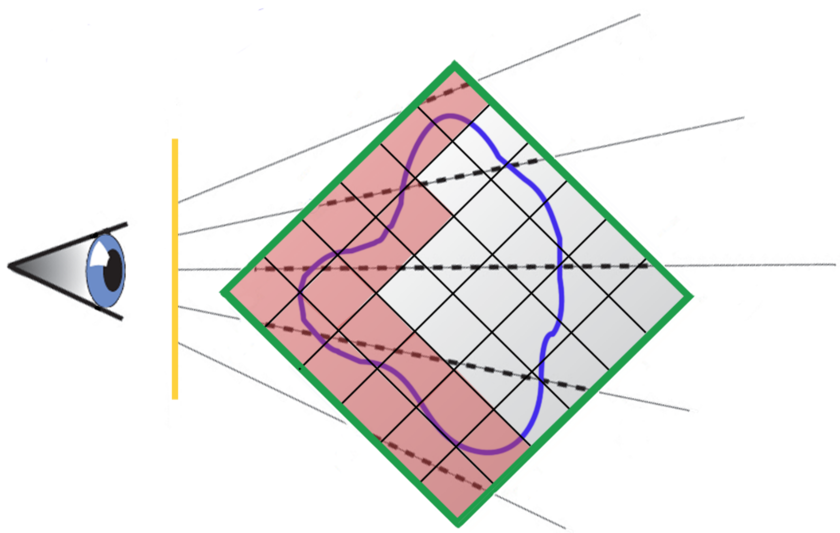
- Cull all bricks against view frustum
- Cull all occluded bricks



Q1: WORKING SET DETERMINATION – MODERN (1)

Visibility determined during ray traversal

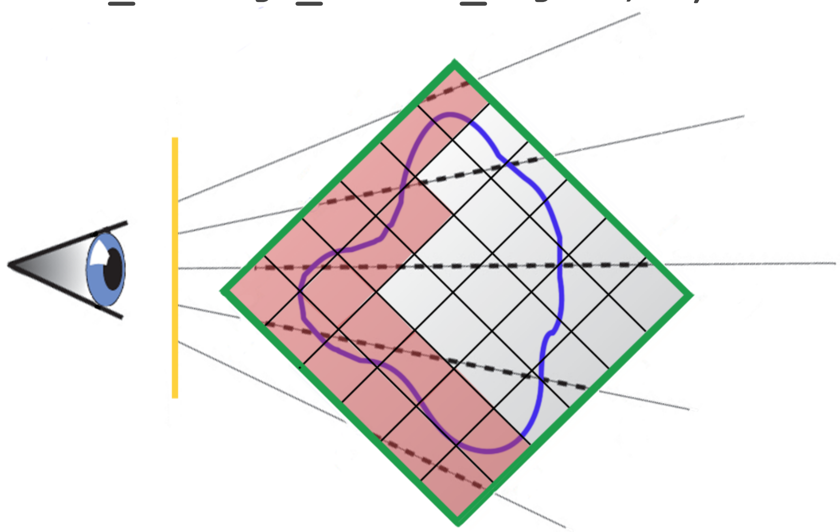
- Implicit view frustum culling (no extra step required)
- Implicit occlusion culling (no extra steps or occlusion buffers)



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`, ...)



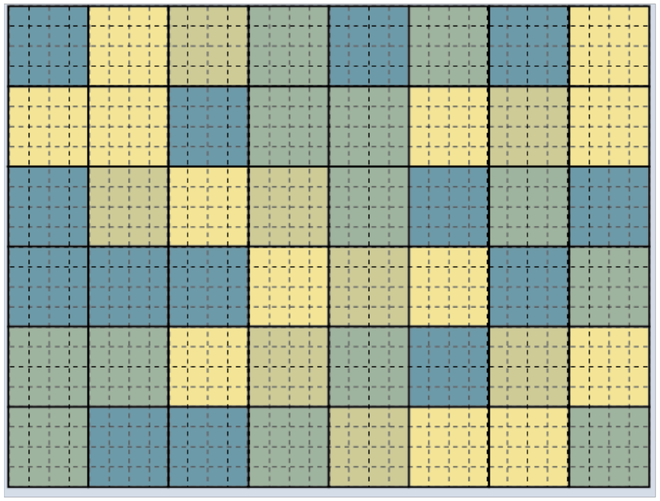
Q2: WORKING SET STORAGE - TRADITIONAL

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

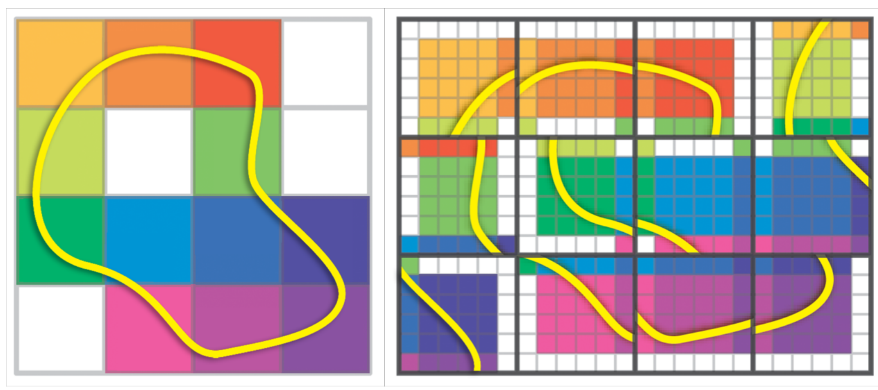
- Preferably single-pass rendering
- All rays traversed in front-to-back order
- Rays perform dynamic address translation (virtual to physical)
- Rays dynamically write out brick usage information
    - Missing bricks (“cache misses”)
    - Bricks in use (for replacement strategy: LRU/MRU)
- Rays dynamically determine required resolution
    - Per-sample or per-brick

VIRTUAL TEXTURING

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

- Virtual image or volume (extent of original data)
- Domain decomposition of virtual texture space: pages
- Working set of physical pages stored in cache texture
- Page table maps from virtual pages to physical pages

virtual image or volume space



texture cache

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

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

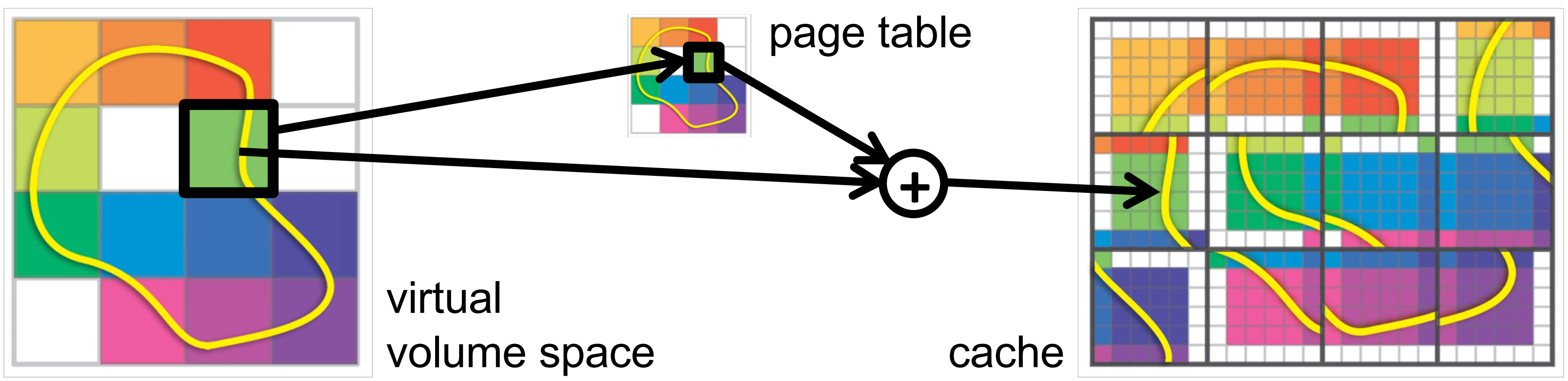
HARDWARE VIRTUAL TEXTURES

- OpenGL
  - Sparse textures (**ARB_sparse_texture, ARB_sparse_texture2**)
- Vulkan
  - Sparse partially-resident images (**VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT**)
- CUDA
  - Unified memory with on-demand page migration
  - Only for regular (global) memory, not for textures

ADDRESS TRANSLATION

Map virtual to physical address

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



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

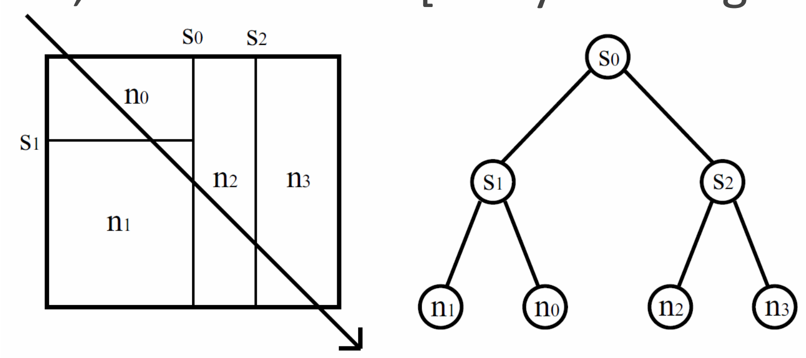
Spatial hashing

- Needs collision handling; hashing function must minimize collisions

TREE TRAVERSAL

Example: Volume rendering octrees or kd-trees

- Similar to tree traversal in 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]

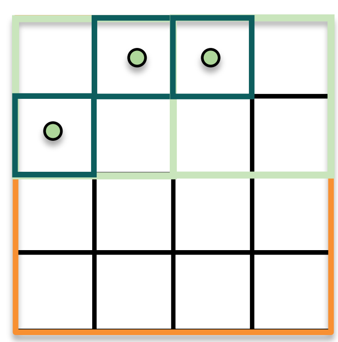


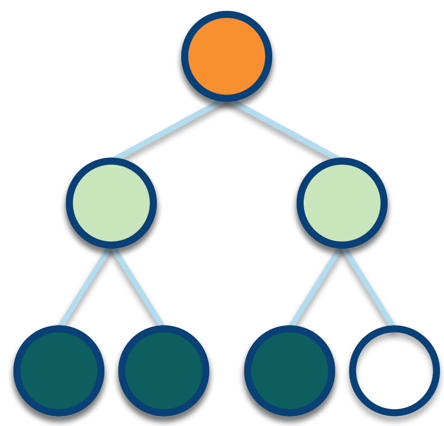
courtesy Foley and Sugerman

ADDRESS TRANSLATION – VARIANT 1: TREE TRAVERSAL

Tree can be seen as a 'page table'

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



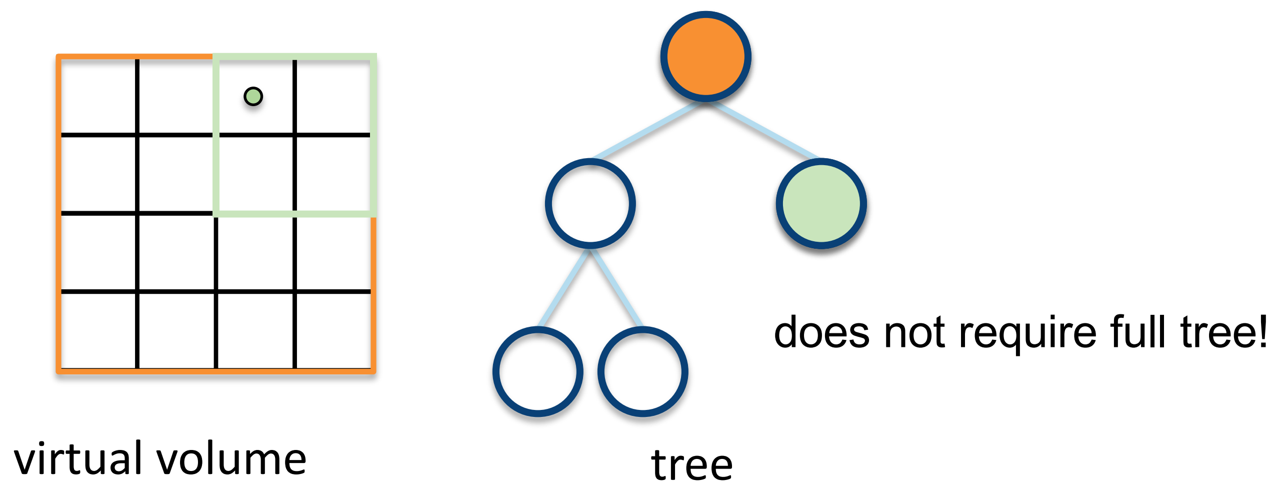


"page table hierarchy" (tree) coupled to resolution hierarchy!

ADDRESS TRANSLATION – VARIANT 1: TREE TRAVERSAL

Tree can be seen as a ‘page table’

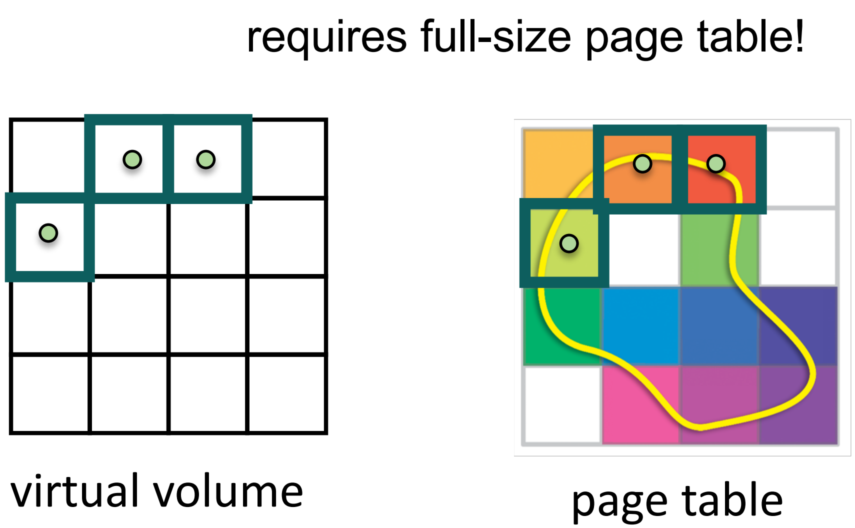
- Linked nodes; dynamic traversal
- Nodes contain page table entries



ADDRESS TRANSLATION –
VARIANT 2: UNIFORM PAGE TABLES

Only feasible when page table is not too large

- For "medium-sized" volumes or "large" page/brick sizes



ADDRESS TRANSLATION – VARIANT 2: UNIFORM PAGE TABLES

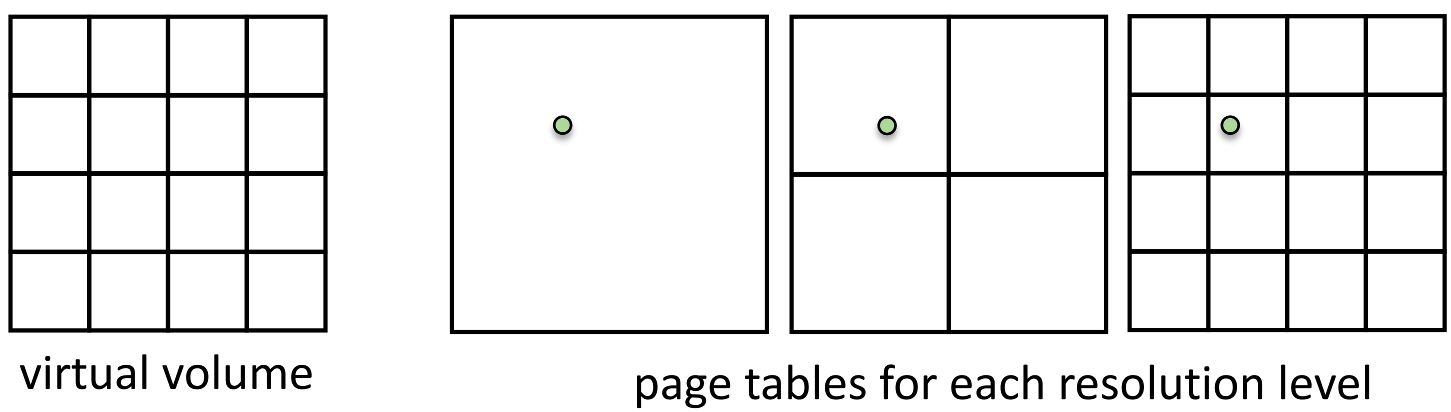
Only feasible when page table is not too large

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

Can do page table for each resolution level

-> page table mipmap

- Uniform in each level



ADDRESS TRANSLATION –
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`, `GL_ARB_sparse_texture2`)
- Vulkan sparse partially-resident images
- Maximum size limitations apply (e.g., 32k for 2D, 16k for 3D)

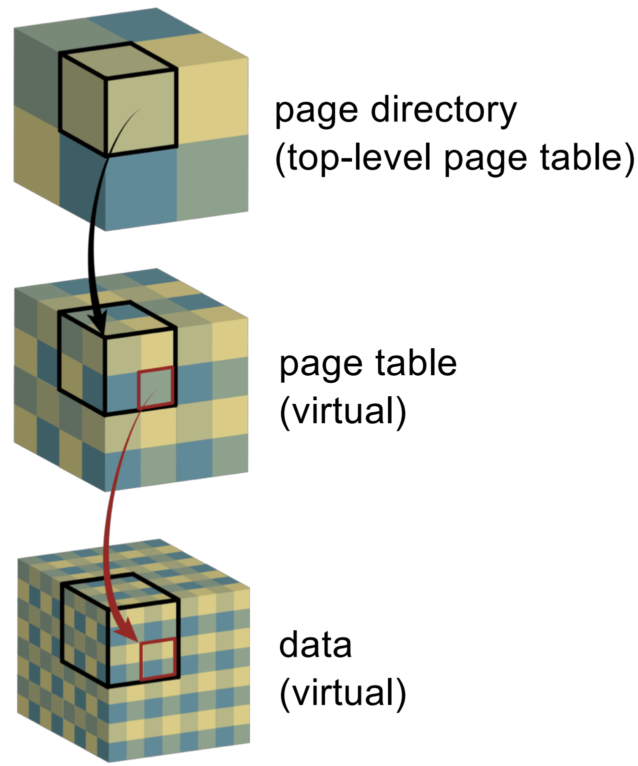
ADDRESS TRANSLATION –
VARIANT 3: MULTI-LEVEL PAGE TABLES

Virtualize page tables recursively

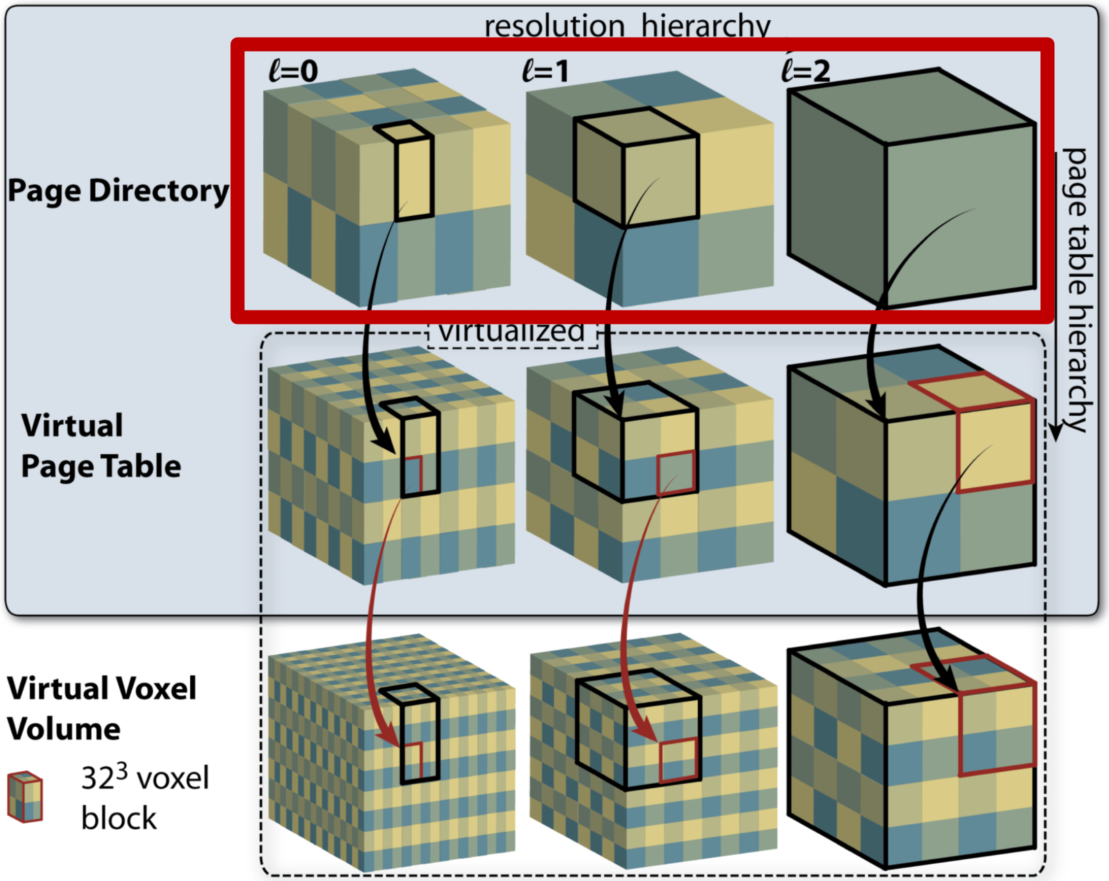
- Same idea as in CPU multi-level page tables
- Pages of page table entries like pages of voxels

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

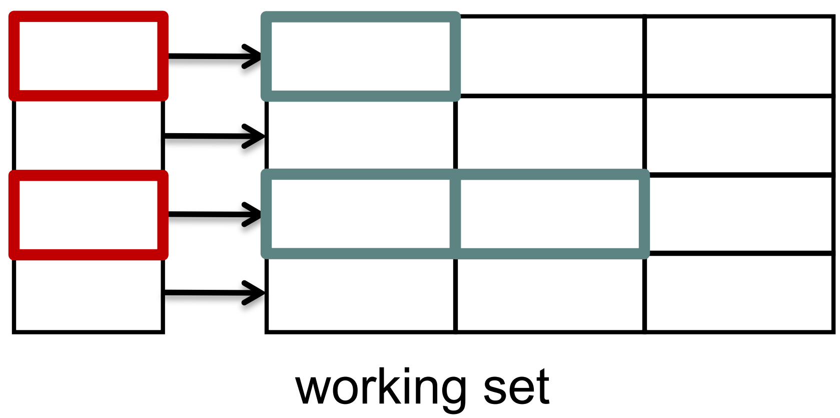
voxel blocks: 32^3 voxels

page table blocks: 32^3 page table entries

ADDRESS TRANSLATION – 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



ADDRESS TRANSLATION – 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)

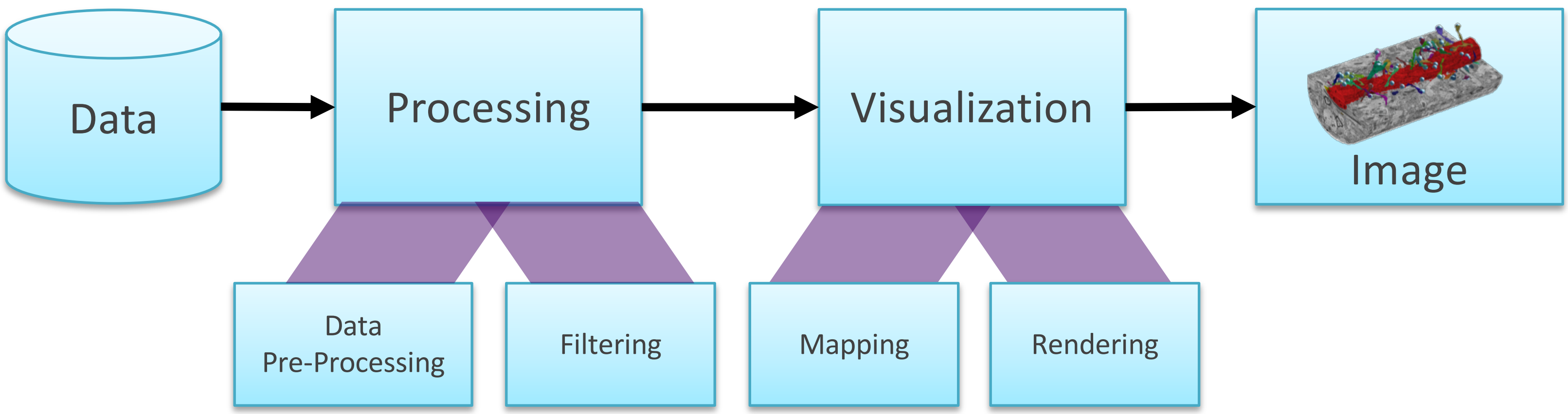
Missing bricks: linear search through hash table row

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

SUMMARY - RAY-GUIDED VOLUME RENDERING

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

Questions?

Break (15 min)

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