CUDA-accelerated LOD Generation

Markus Schütz

Institute of Visual Computing & Human-Centered Technology

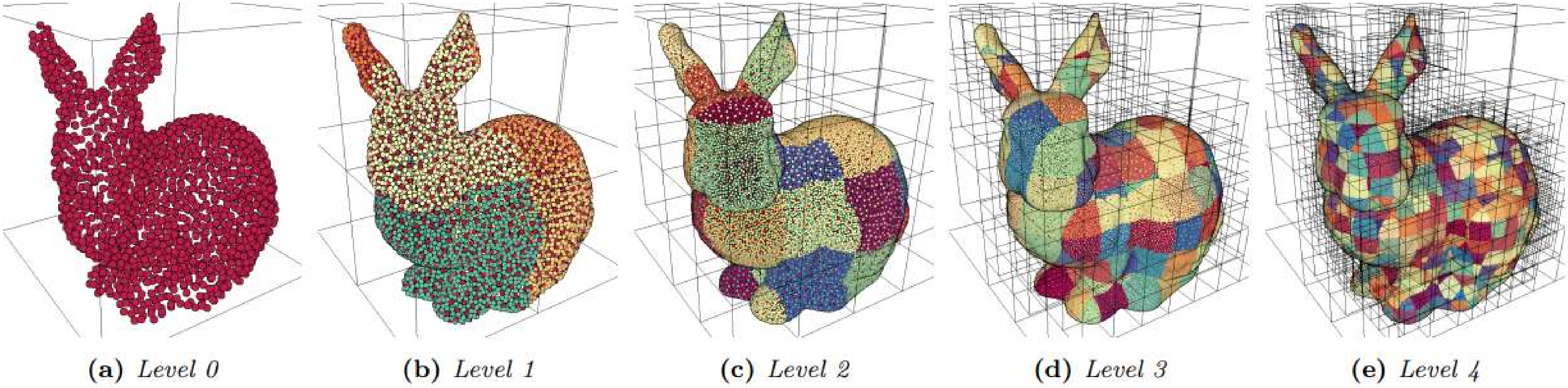
TU Wien, Austria

CUDA-accelerated LOD Generation

- Current peak LOD generation performance: 11M points/sec
  - On CPU
  - Out-of-core
- How much faster can we be in CUDA?
  - Pure GPU processing performance
  - Ignore disk I/O for now
  - In-core

CUDA-accelerated LOD Generation

- “Fast Out-of-Core Octree Generation for Massive Point Clouds”
    - Up to 11M points/sec
    - With randomly sampling lower LODs


(a) *Level 0* (b) *Level 1* (c) *Level 2* (d) *Level 3* (e) *Level 4*

CUDA-accelerated LOD Generation

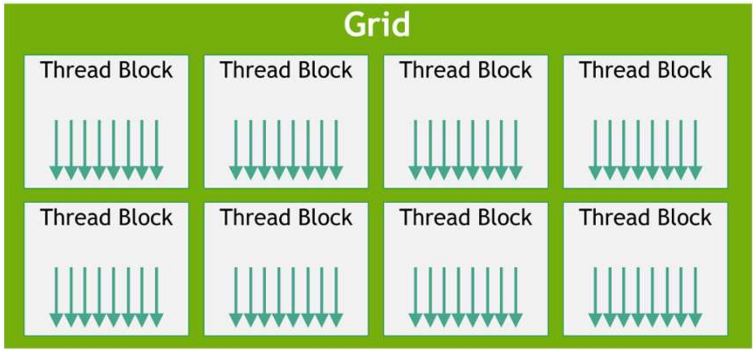
- CUDA Approach:
  - Largely the same process
  - About 100 times faster
  - Spend some of that extra performance on better quality
  - color filtering!

Algorithm overview

- Pass 1: Partition into octree leaf nodes with <10k points
  - Countsort. Pretty much the fastest way to sort into buckets.
  - O(n) complexity, O(2n) to be precise.
  - Hierarchical Countsort: Merge cells with few points
- Pass 2: Bottom-up Voxelization
  - Leaf-nodes contain original points
  - Lower LODs contain quantized points => voxels
  - Downsampling uses 128^3 voxel grids for each node

CPU vs. CUDA Approach differences

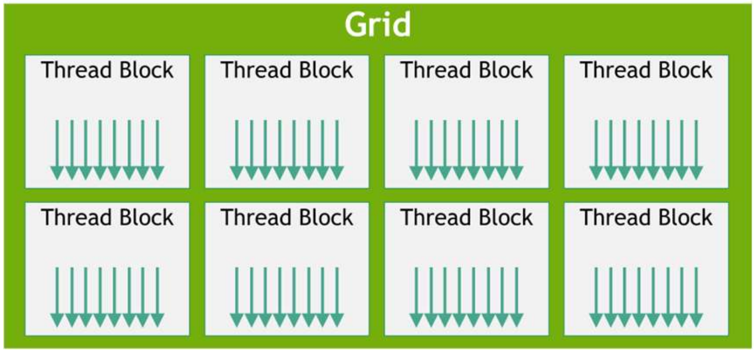
- Largely the same approach, but different parallelism
- CPU: ~16 threads, completely independent
- CUDA:
  - 10k CUDA cores / threads
  - Lot’s of threads, but not all are independant!
  - Threads grouped in blocks
  - Each thread in block operates in lockstep
  - 84 Streaming Multiprocessors (SMs)
  - SMs process blocks
  - Each SM operates independently!



https://docs.nvidia.com/cuda/cuda-c-programming-guide/

CUDA-accelerated LOD Generation

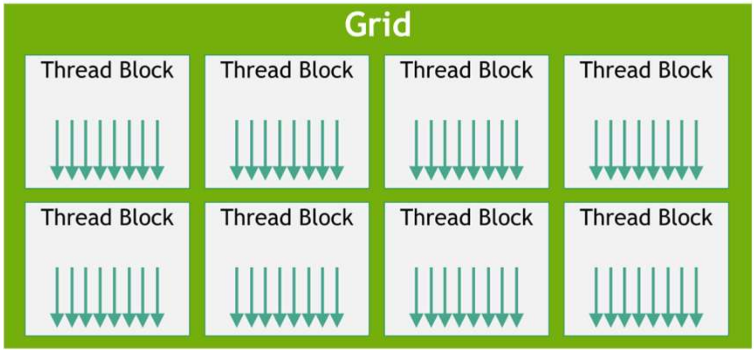
- My personal view of CUDAs model compared to a CPU
  - 84 threads (1 SMs = 1 thread)
  - Each SM can operate independently
  - Each SM has massive SIMD,
    simultaneously processing 1 instruction for 128 points at once
- Not entire truth but helps reason about things



https://docs.nvidia.com/cuda/cuda-c-programming-guide/

CUDA-accelerated LOD Generation

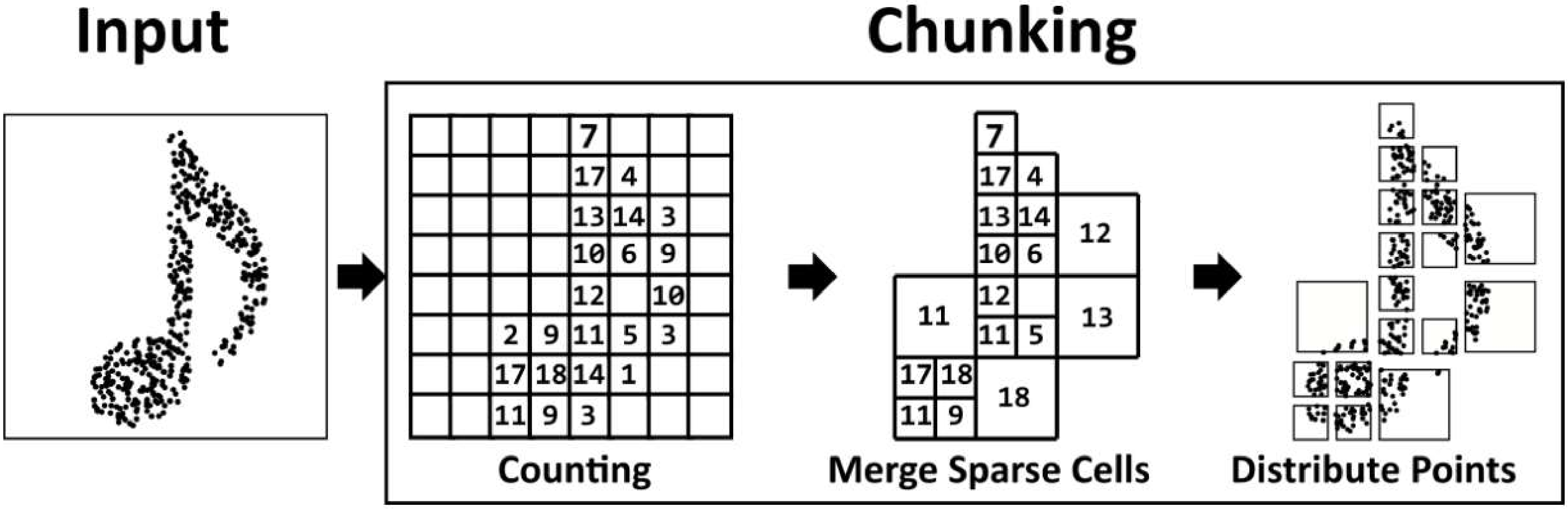
- CUDA port needs different approach to parallelism
- CPU approach utilizes 1 thread per chunk of 10M points
- CUDA approach varies parallelism:
    - For counting, each thread processes 10k points
    - For random sampling, each block processes 1 octree node
    - For filtered sampling, all 10k threads process 1 node, then next, …



https://docs.nvidia.com/cuda/cuda-c-programming-guide/

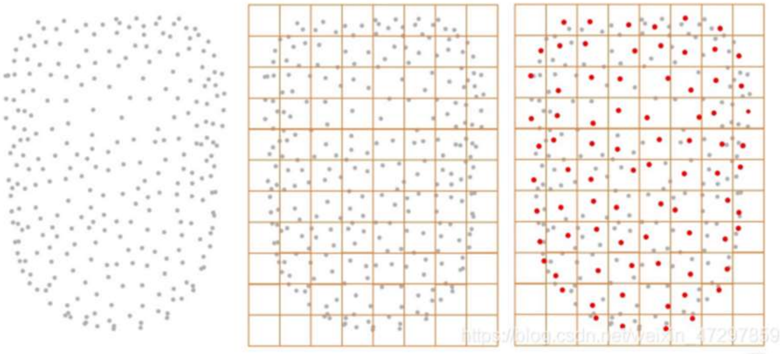
Why?

- “For counting, each thread processes 10k points”
  - Trivial case, each work item exact same. It’s what the GPU excels at
  - Just loop until all points were processed



Why?

- “For random sampling, each block processes 1 octree node”
  - Each SM has fast shared memory (L1 cache)
  - We use this for the sampling grid.
  - Sampling grid is 4 * 128^3 byte, but shared mem only 48kb
  - Use hash map as sampling grid.
  - Sufficient shared mem for one node, but not more



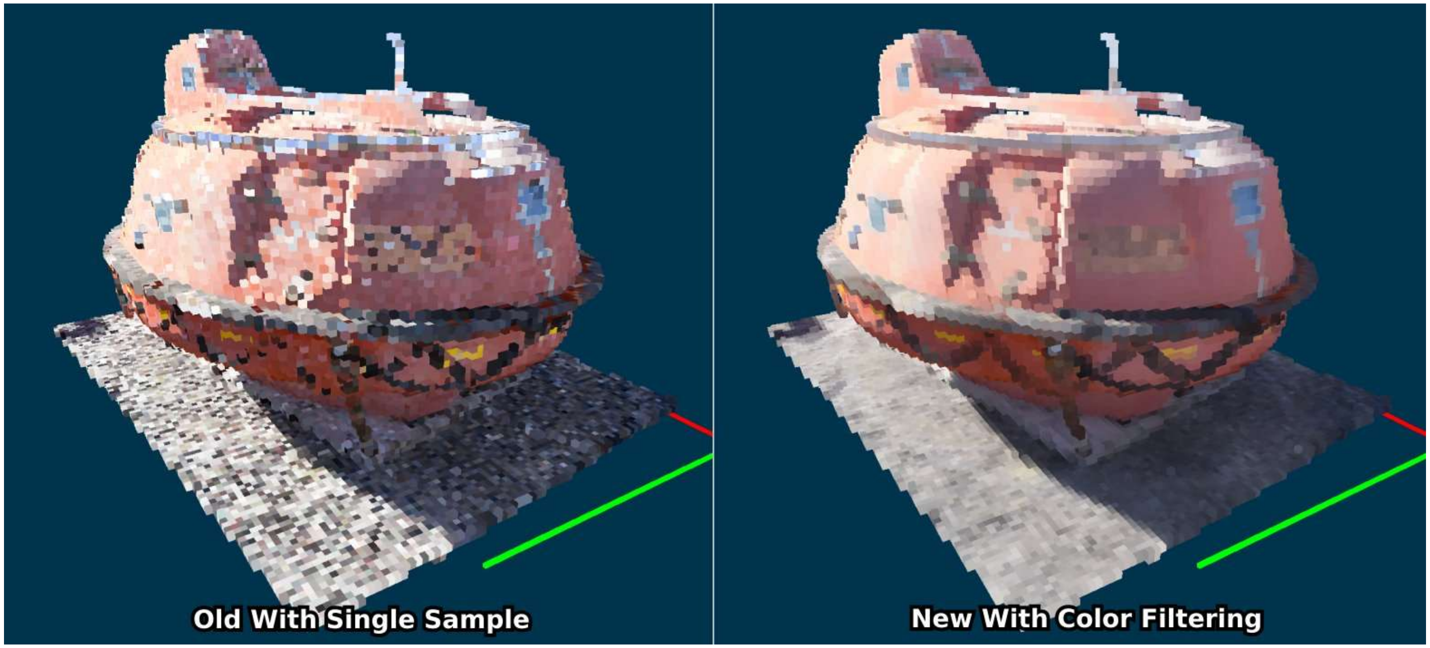
https://blog.csdn.net/weixin_47297859/article/details/117523118

Why?

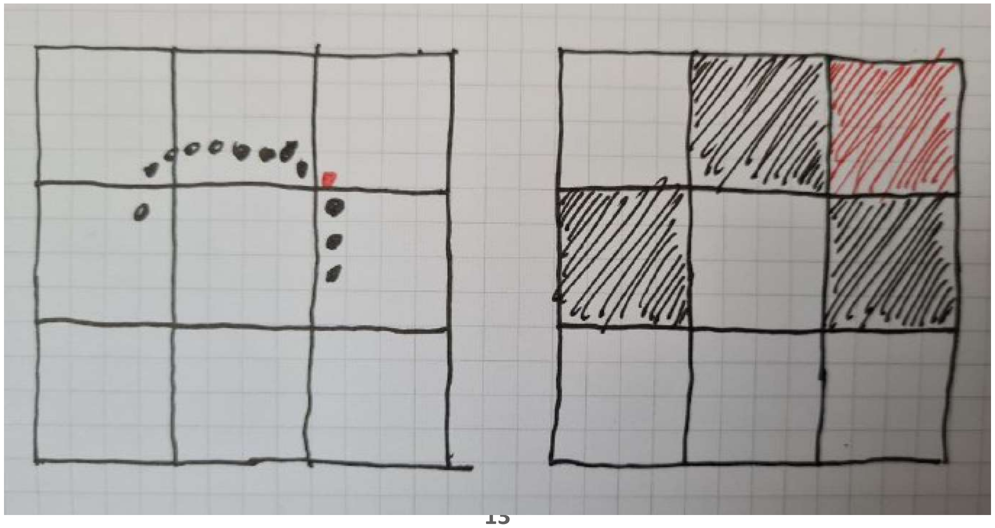
- “For filtered sampling, all 10k threads process 1 node, then next, …”
  - Now need 16 byte per sample grid cell
  - Also need fast access to 3^3 neighborhood
  - Hash map no longer suitable
  - Need to use global memory
  - Just use one single sample grid, and utilize all GPU threads to simultaneously process all points of a node
  - Use all threads to clear sampling grid, then process next node

Color Filtering

- Color Filtering / Anti-Aliasing
  - Picking random point for lower LOD => Bad color values
  - Compute representative color values instead
  - Lower LOD point = weighted average of points it represents

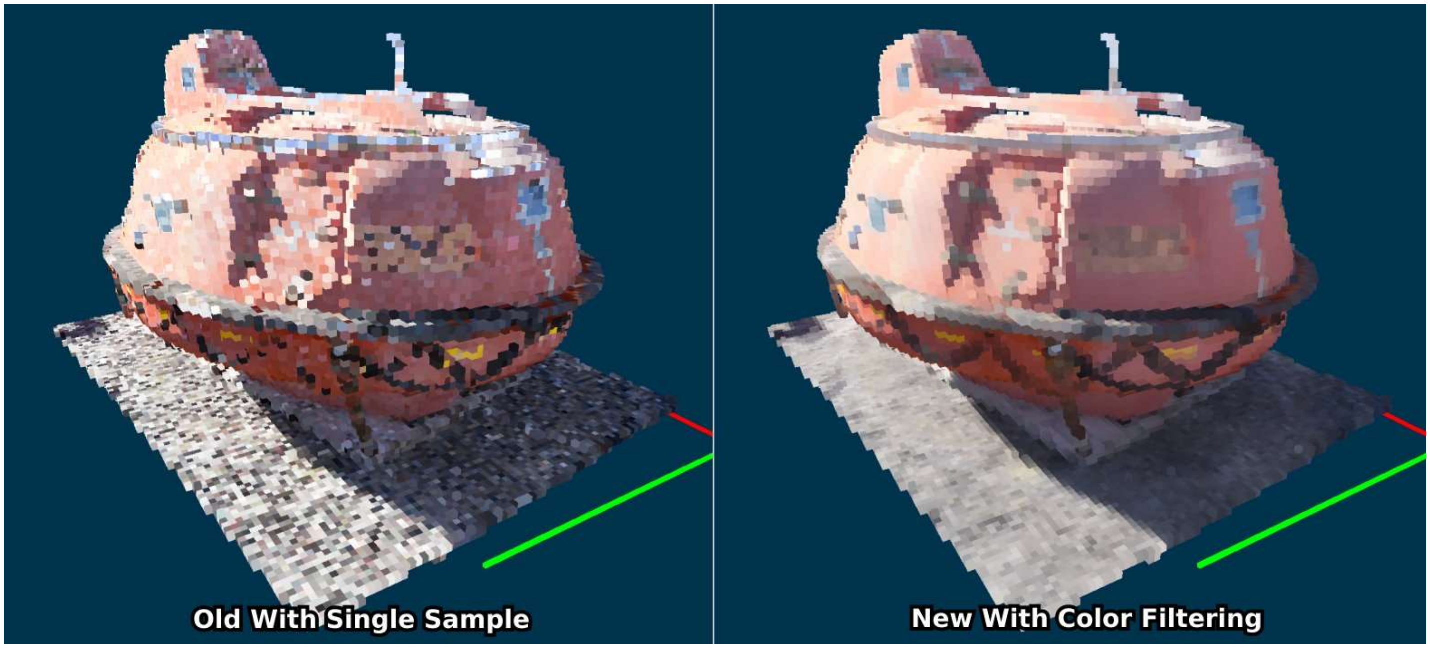


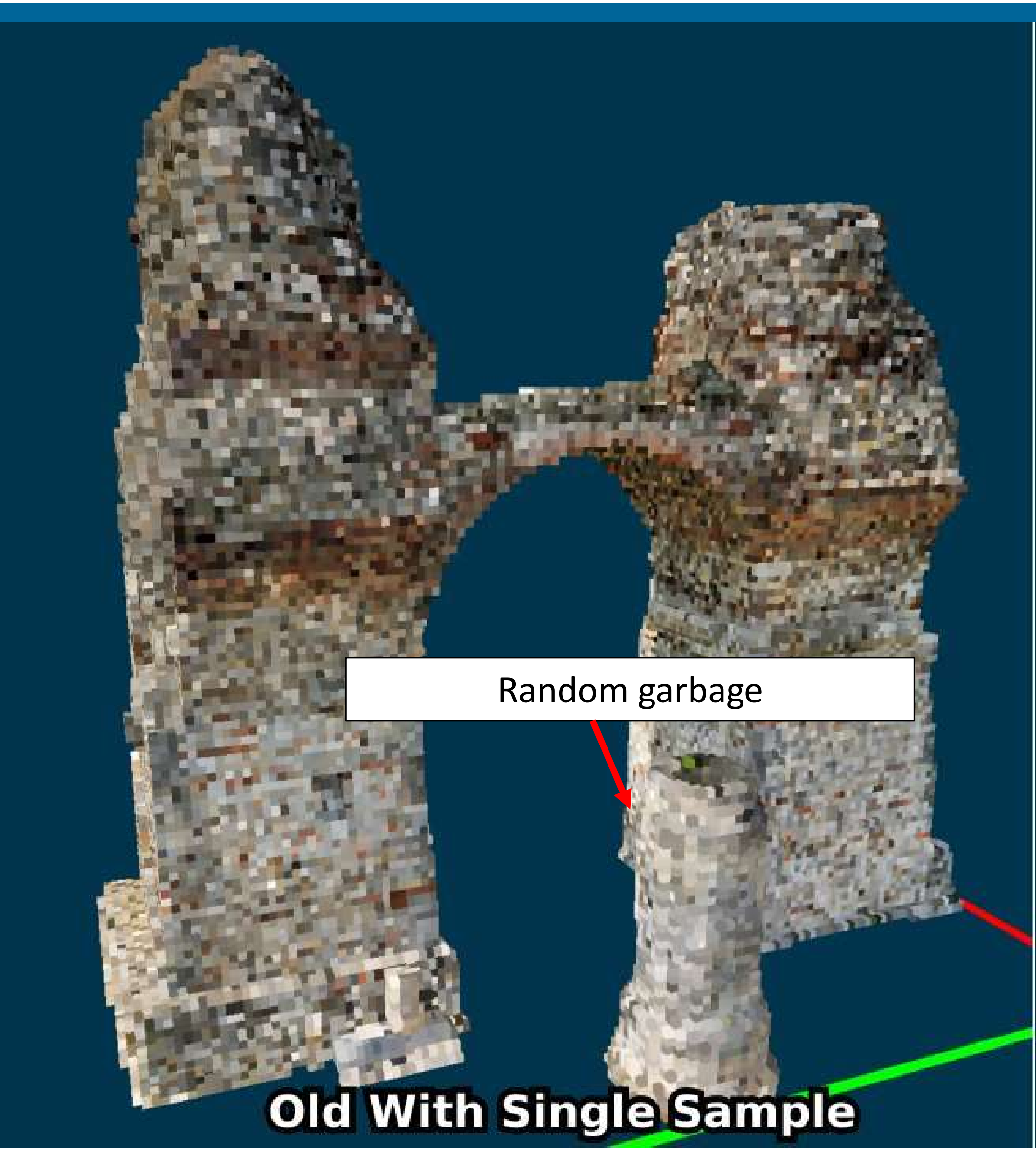
- Easy and fast way:
  - Compute average color of all points in sampling grid cell
  - Then color of point at LOD – 1 is color / count.
- But: Only considers points in cell
  - Should also consider neighborhood



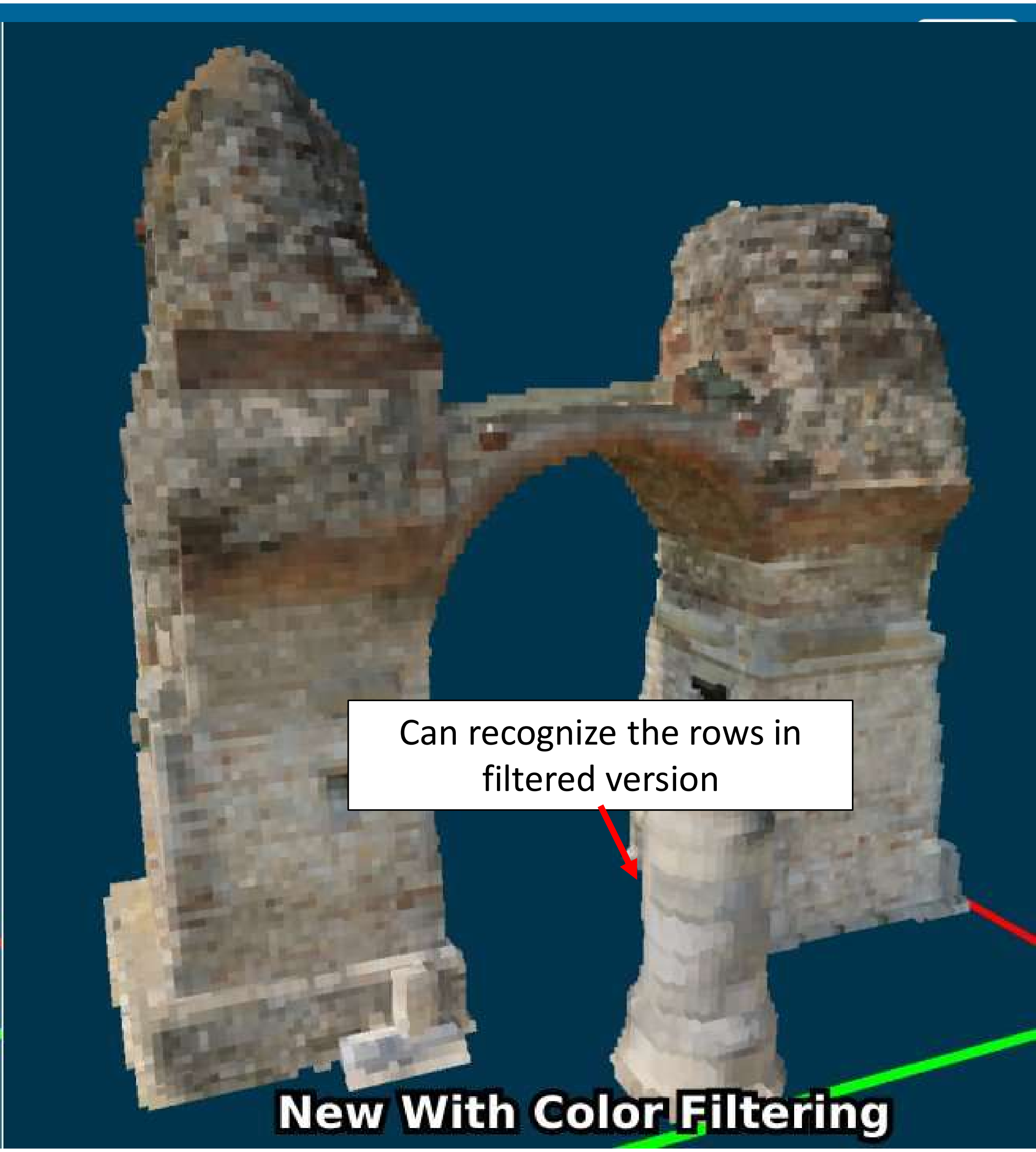
- Slower but nicer: Weighted average neighborhood
- For each point in all adjacent cells, compute weighted average
- The farther from center, the smaller the contribution
- Still fine-tuning, but much nicer appearance already

- Still fine-tuning, but much nicer appearance
- Especially when in motion!
  - Without filtering, results “sparkle”





Random garbage



Can recognize the rows in filtered version

Performance

- Random sampling
    - 145M points in 108ms (1.3 billion points/sec)
- Color Filtering (single-cell)
    - 145M points in 340ms (426M points/sec)
- Color Filtering (3x3x3 neighborhood)
    - 145M points in 946ms (152M points/sec)
- Same quality -> 100x faster than CPU
- Better quality -> still 10-40x faster than CPU

Thank you for your attention