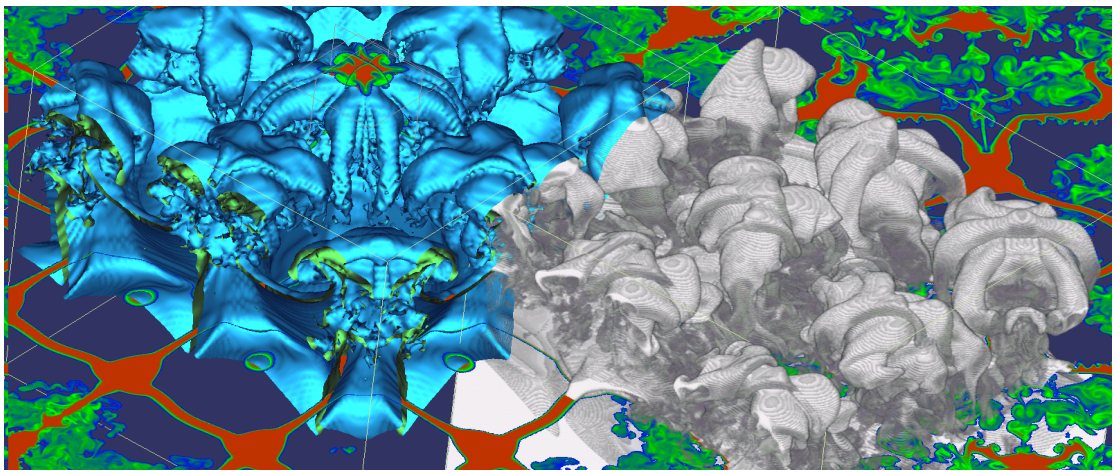
Real-time Rendering in External Memory.



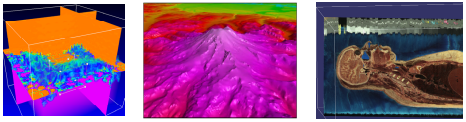
Valerio Pascucci

Lawrence Livermore National Laboratory

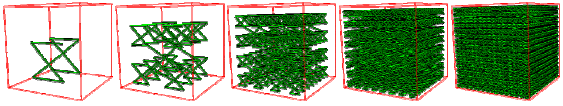
This work was performed under the auspices of the U.S. Department of Energy by the University of California, Lawrence Livermore National Laboratory under Contract No. W-7405-Eng-48. UCRL-PRES-149731

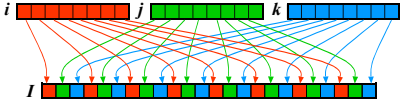
Outline

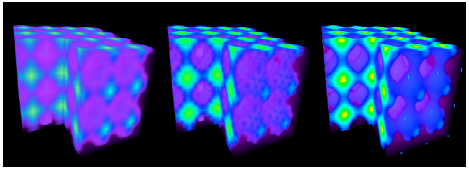
- Motivation
- ~~Previous work~~
- General data layout
- Rendering of large terrains
- 2^n tree indexing
- Slicing large grids
- Remote monitoring of running simulations
- Conclusions



(bla bla[vis`01], bla bla bla [sc`00] ……)

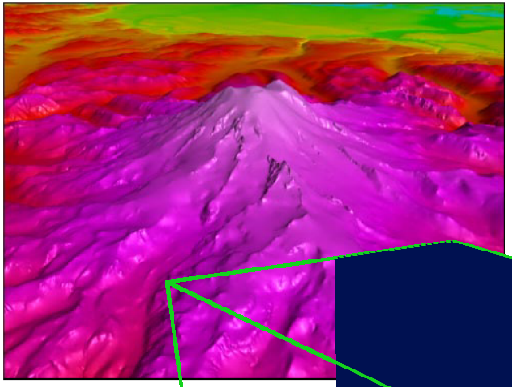




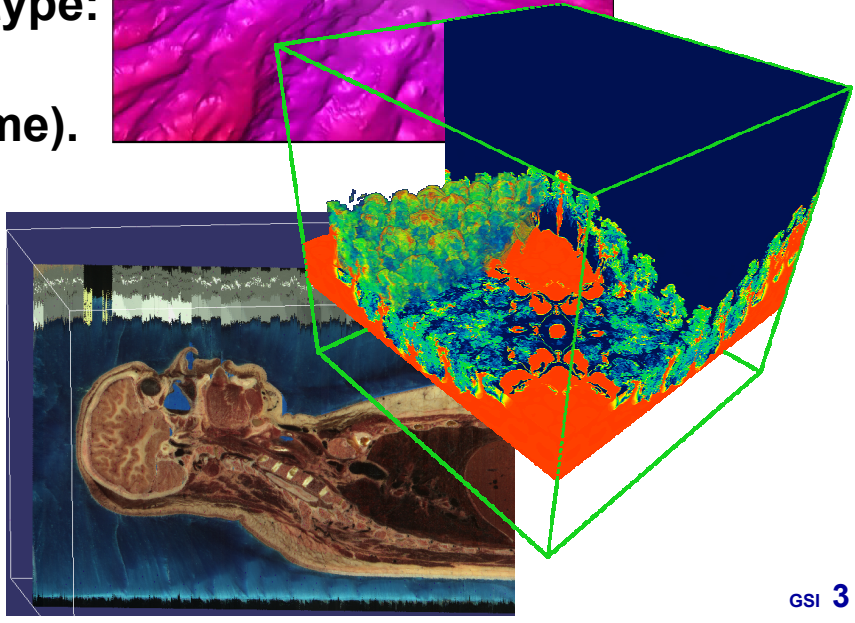


We must achieve real-time interaction with large datasets on a wide variety of platforms.

The problem



- Large datasets of different type: 16k^2 terrains, 8GB/timestep (2k^3 grids +time).
- Interactive rendering for real-time data exploration.
- Target platforms: desktop, parallel server, cluster.



Previous Work

- **Out-of-core geometric algorithms**

  [Goodrich, Tsay, Vengroff, Vitter '93]

  [Vitter '00][Matias, Segal, Vitter '00]

  [Asano, Ranjan, Roos, Welzl '95][Arge Miltersen '99]

- **Out-of-core visualization**

  [Chiang, Silva '97][Sutton, Hansen '99]

  [Livnat,Shen,Johnson '96][El-Sana, Chiang'00]

  [Bajaj, Pascucci, Thompson, Zhang '99]

- **Space filling curves**

  (image processing, multidimensional databse, geomtric datastructure …)

  [Bandou, Kamata.'99][Balmelli, Kovacevic, Vetterli '99]

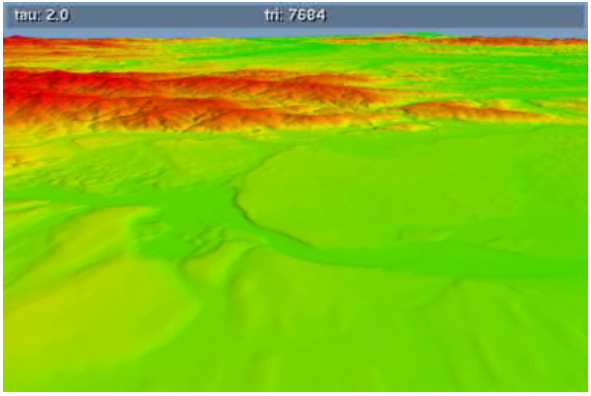
  [Parashar, Browne, Edwards, Klimkowski '97]

  [Niedermeier, Reinhardt, sanders '97][wise'00]

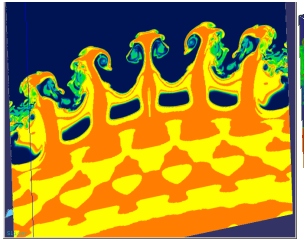
  [Hans Sagan '94] [Lawder king '00][Griebel Zumbusch '99]

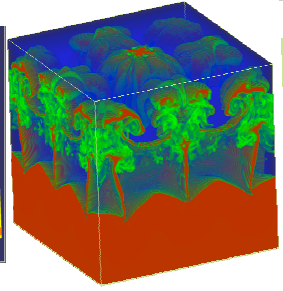
We apply three fundamental techniques to the visualization of large simulation data.

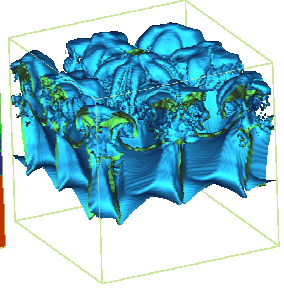
The general approach



- Multi-resolution geometric representation:
  - adaptive view-dependent refinement;
  - minimal geometric output for selected error tolerance.
- Cache oblivious external memory data layouts:
  - exploit spatial and resolution coherency;
  - no need for complicated paging techniques.
- Progressive processing:
  - continuously improved rendering;
  - scalability with the resources without budgeting.







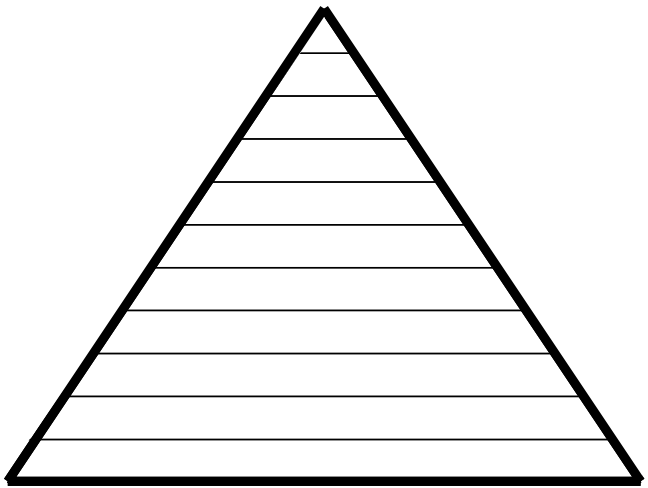
General Data Layout

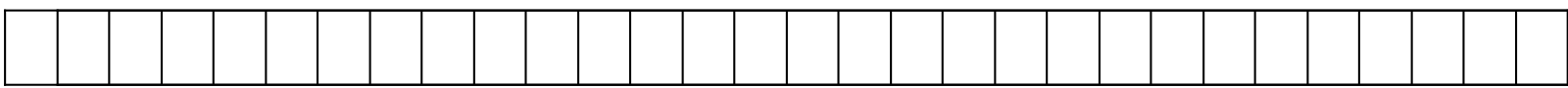
Data coherent Progressive refinement of a hierarchical geometric data-structure

Grouping the data by level of resolution

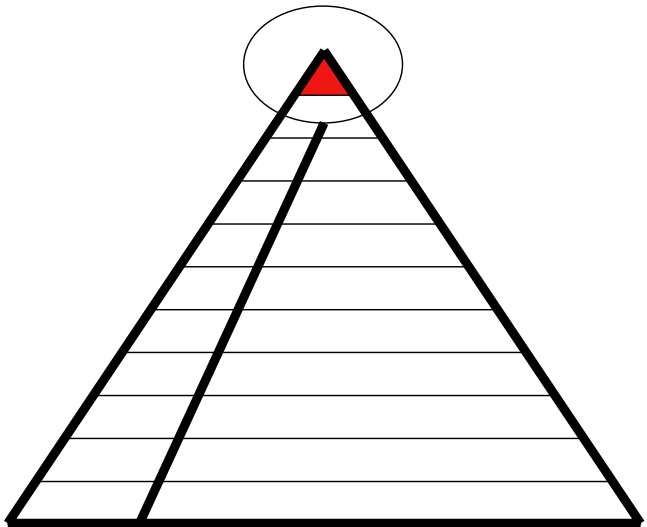
Grouping the data by geometric proximity

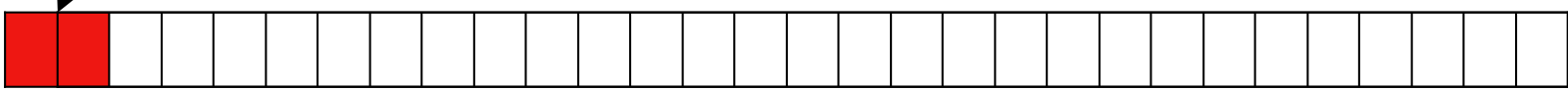
General Data Layout



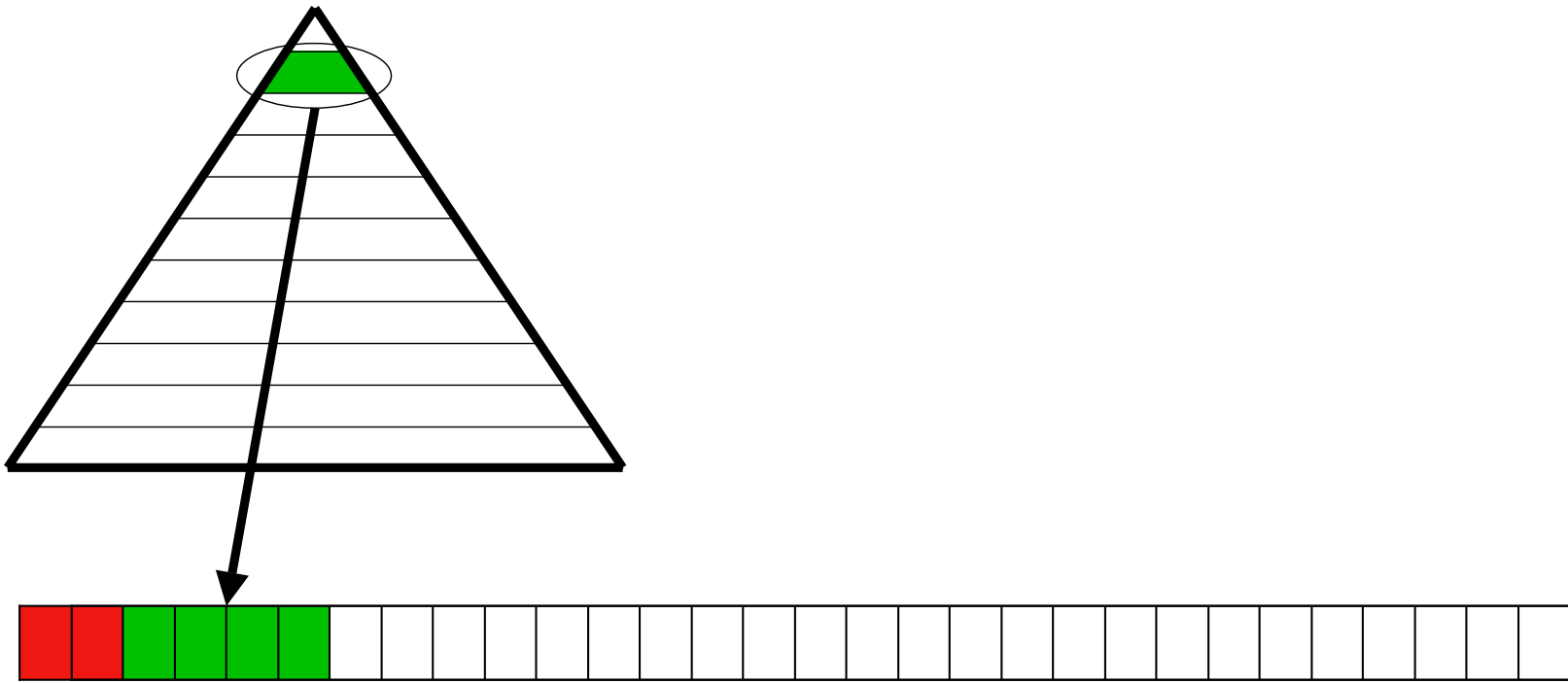


General Data Layout

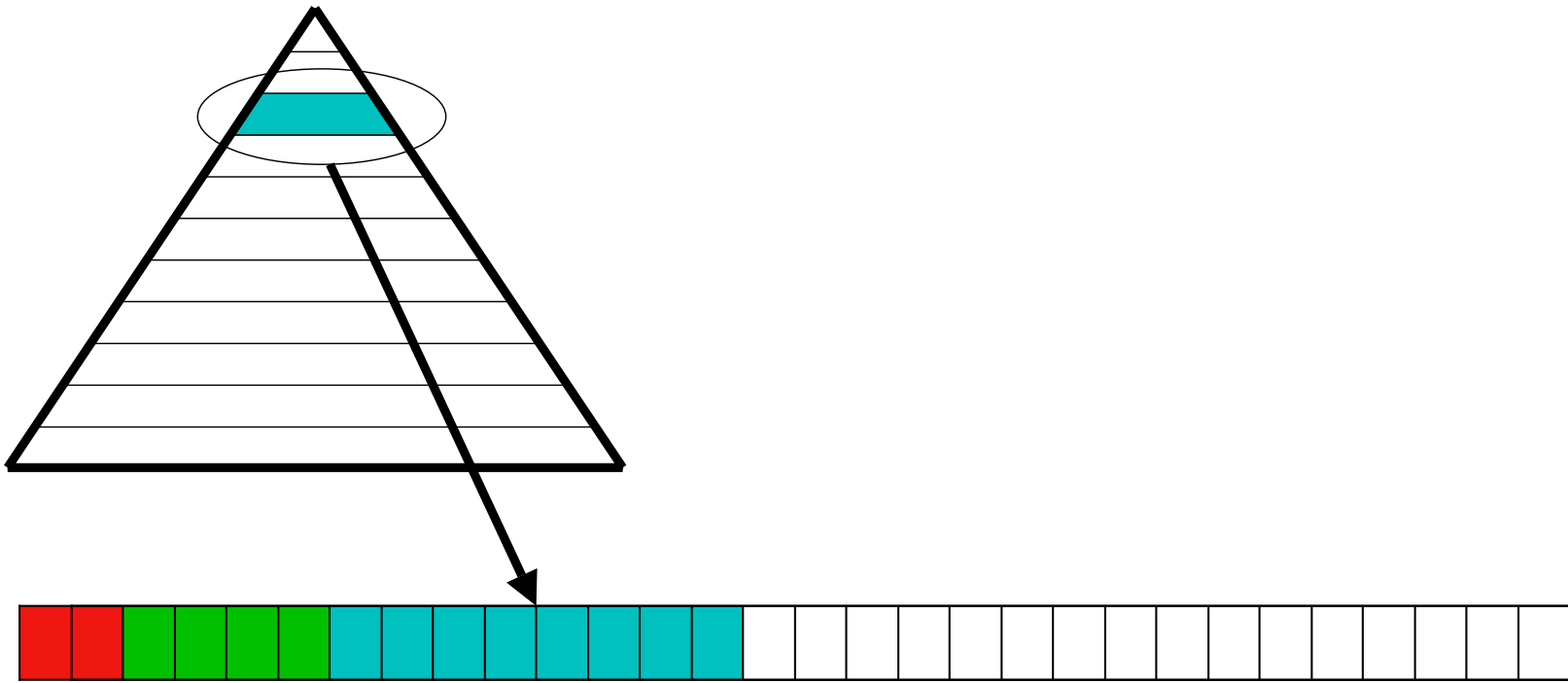




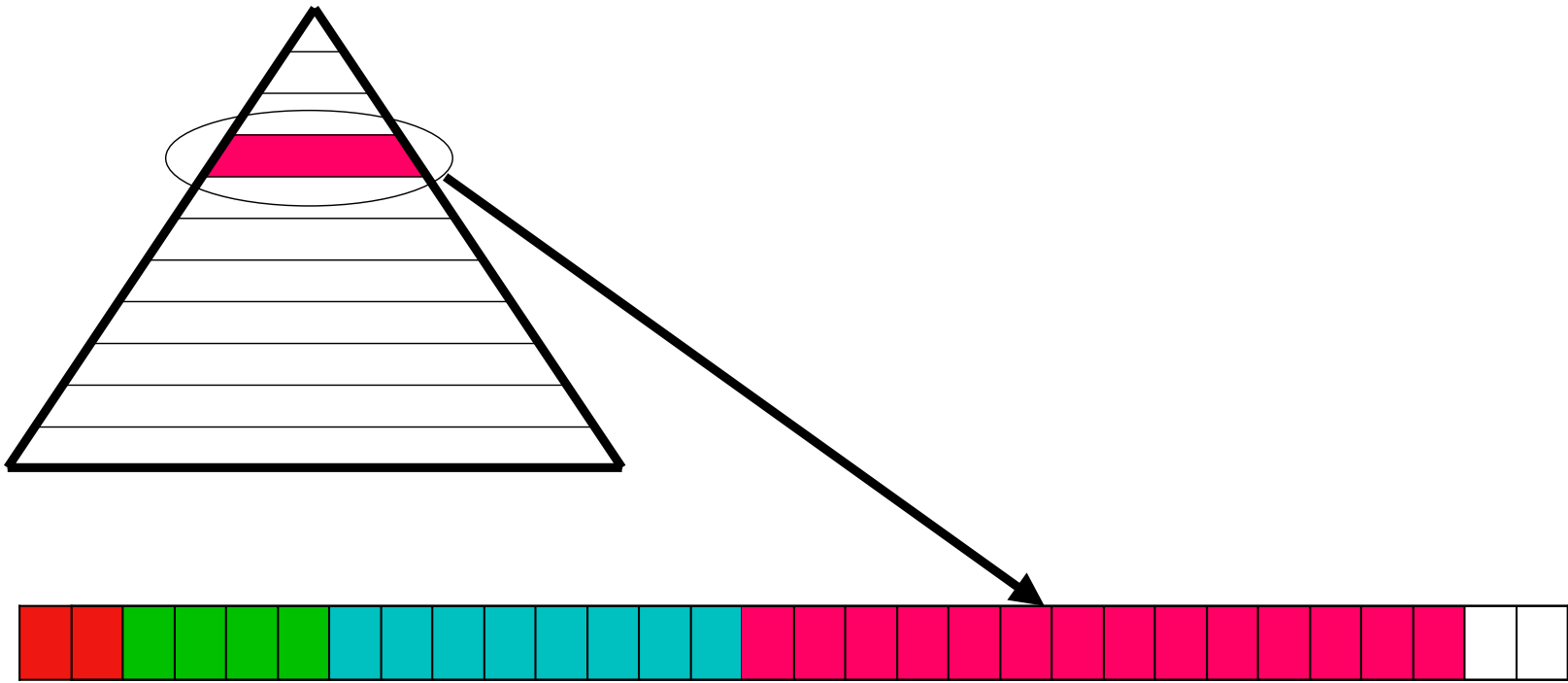
General Data Layout



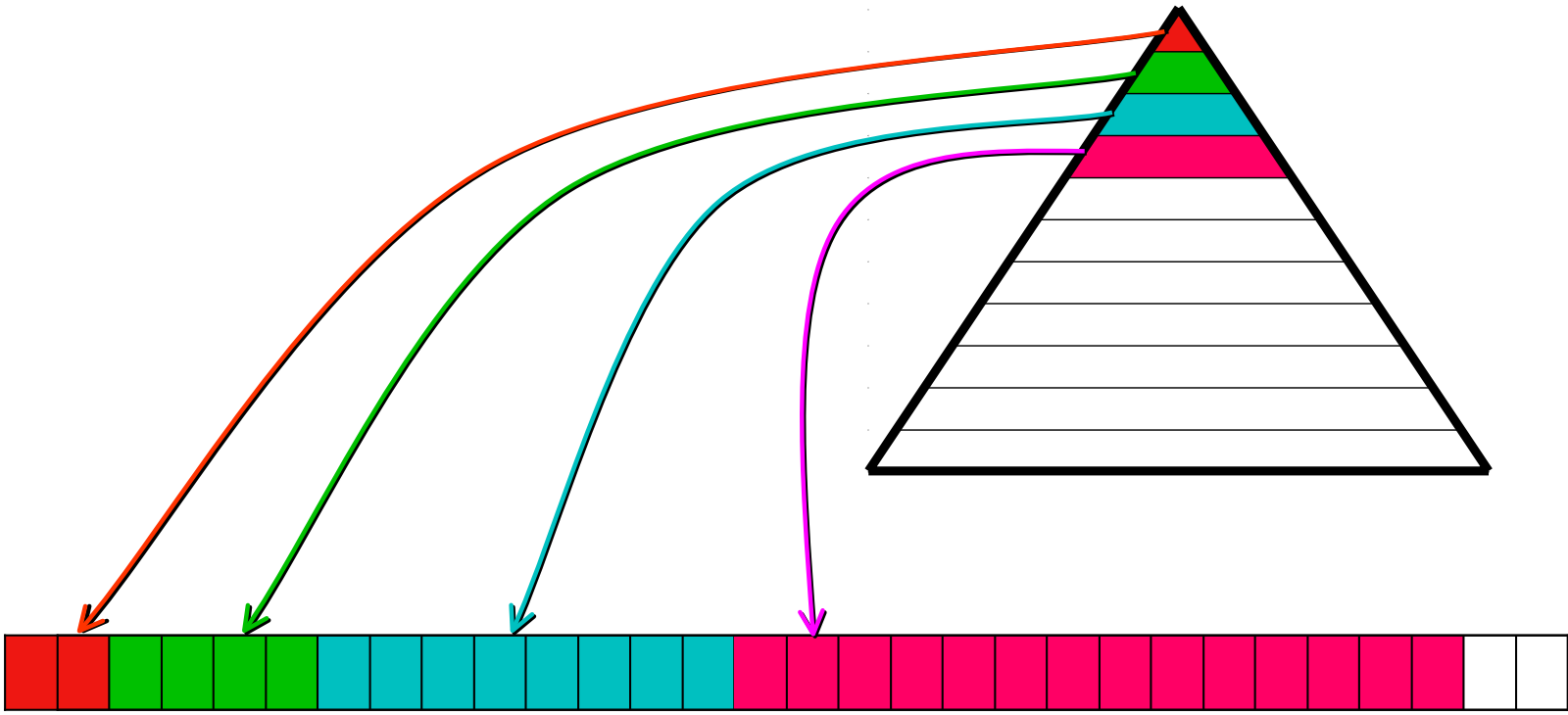
General Data Layout



General Data Layout



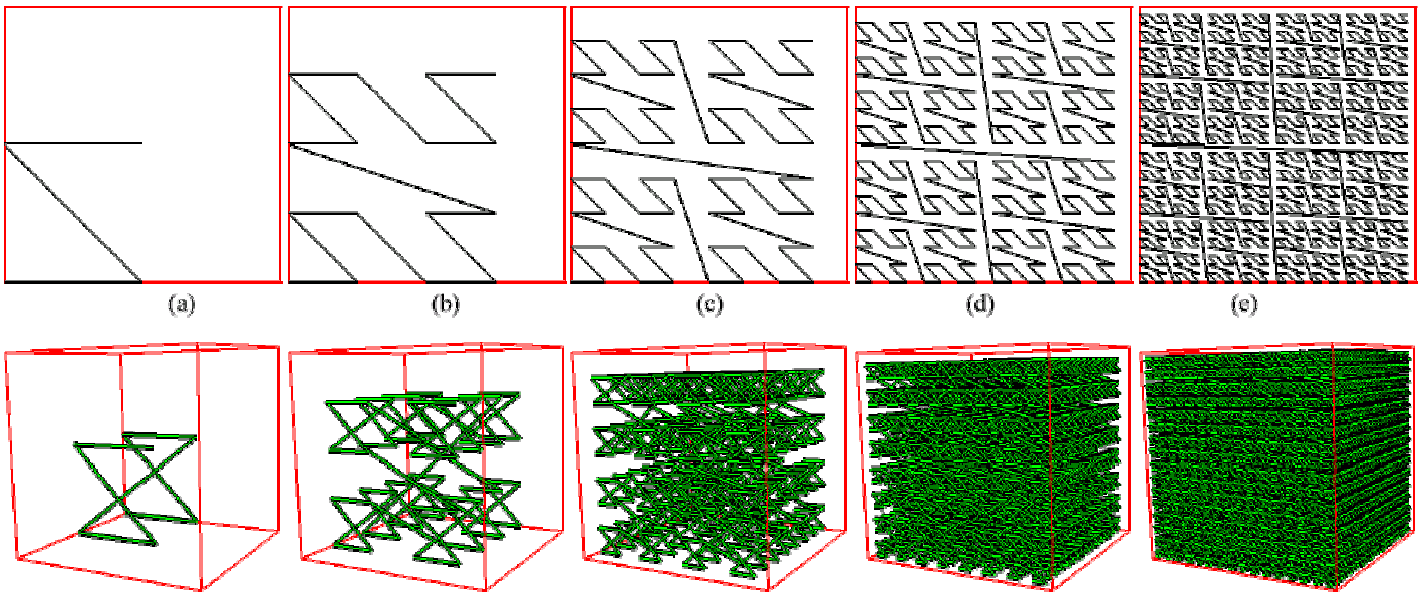
General Data Layout



We exploit the correlation of bin/quad/oct-trees with the Lebesgue space-filling curves.

The Lebesgue curve is also known as Z-order, Morton, …. Curve.

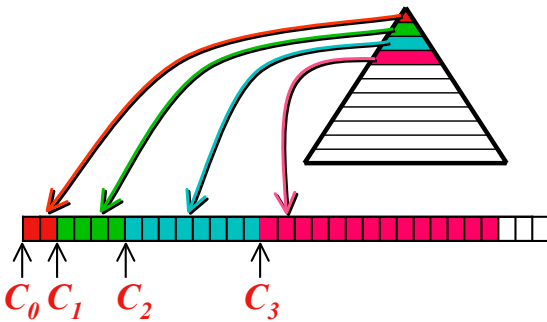
Special case of the general definition introduced by Guiseppe Peano in 1890.



General Data Layout

nD to 1D mapping:

$$I \rightarrow I^*$$



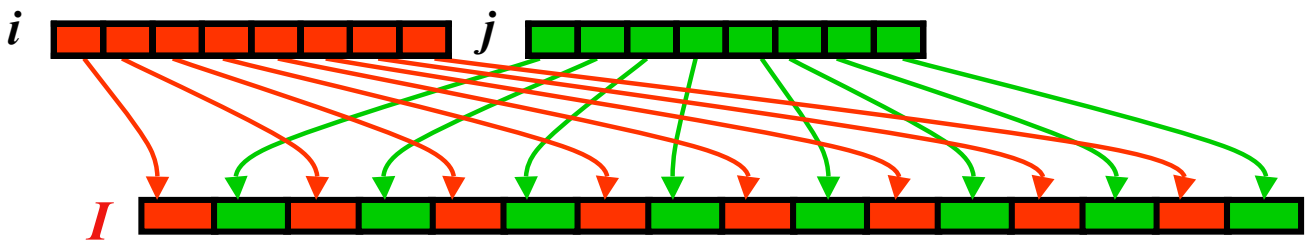
$I \rightarrow l$ find the level of resolution l

C_l (pre)compute the number of elements in the levels coarser than l

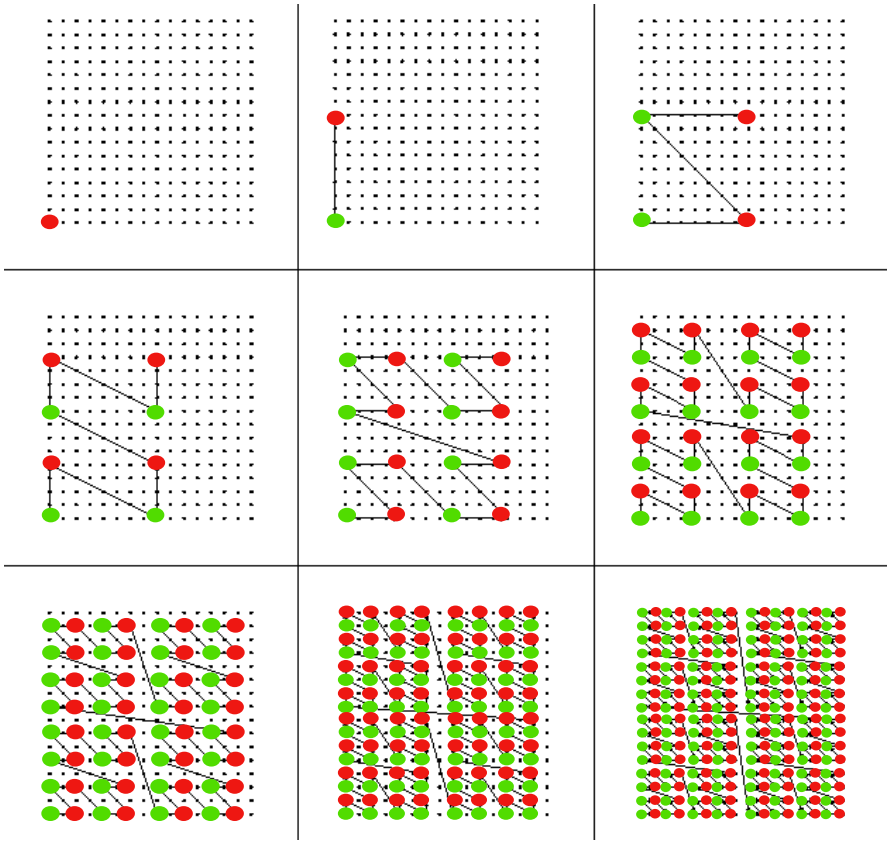
$I \rightarrow I'$ index of the element within its level of resolution

$$I^* = C_l + I'$$

The single resolution Z-order address is obtained with a simple a bit interleaving.



We turn the recursive definition of the Z-order curve into a binary sub-sampling hierarchy.



- Coarser data (even samples)
- New level data (odd samples)

Only one division is needed to convert from single resolution to hierarchical Z-order.

Index mapping for

		Level of resolution															
		0	1			2				3							
	Z-order 0	0															
	Z-order 1	0	1														
I	Z-order 2	0	2	1	3												
	Z-order 3	0	4	2	6	1	3	5	7								
	Z-order 4	0	8	4	12	2	6	10	14	1	3	5	7	9	11	13	15
*I**		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Indexing remapping

h trailing zeros

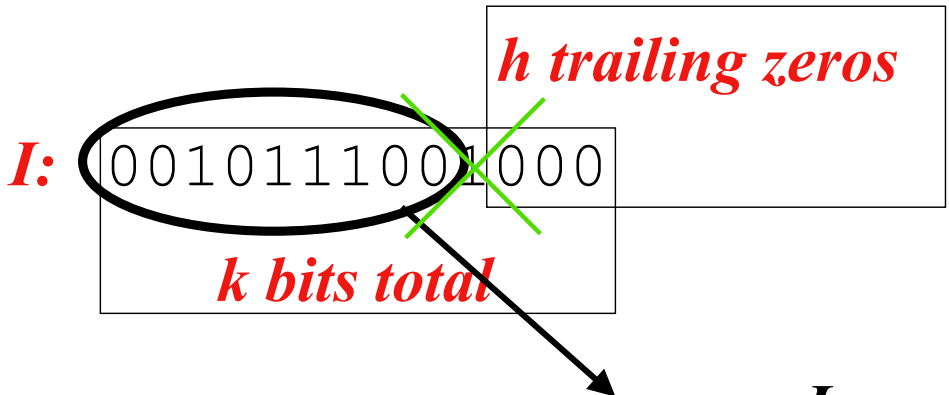
I: 00101110010 00

k bits total

$$l = k - h \qquad C_l = 2^{l-1} \qquad I' = \left\lfloor \frac{I}{2^{l+1}} \right\rfloor$$

$$I^* = C_l + I'$$

Indexing remapping



$$l = k - h \qquad C_l = 2^{l-1} \qquad I' = \left\lfloor \frac{I}{2^{l+1}} \right\rfloor$$

$$I^* = C_l + I'$$

Indexing remapping

h trailing zeros

I: 0010111001000

1000000000

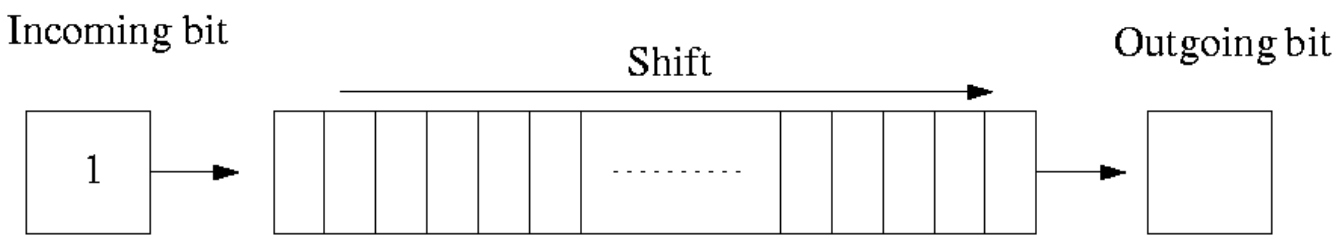
$$l = k - h \qquad C_l = 2^{l-1} \qquad I' = \left\lfloor \frac{I}{2^{l+1}} \right\rfloor$$

$$I^* = C_l + I'$$

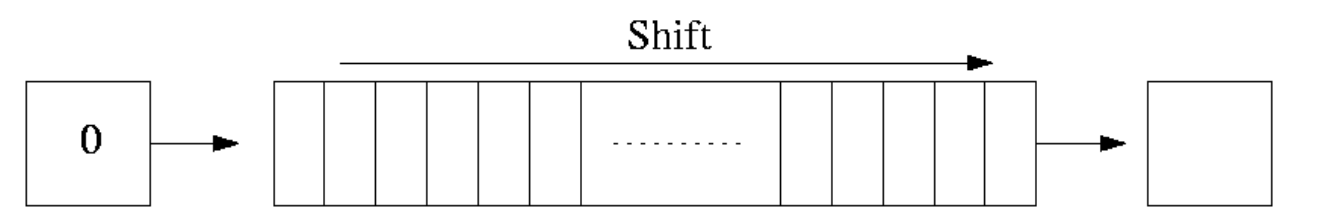
The division can be implemented as an adaptive bitwise shift operation.

Algorithm

Step 1: shift right with incoming bit set to 1

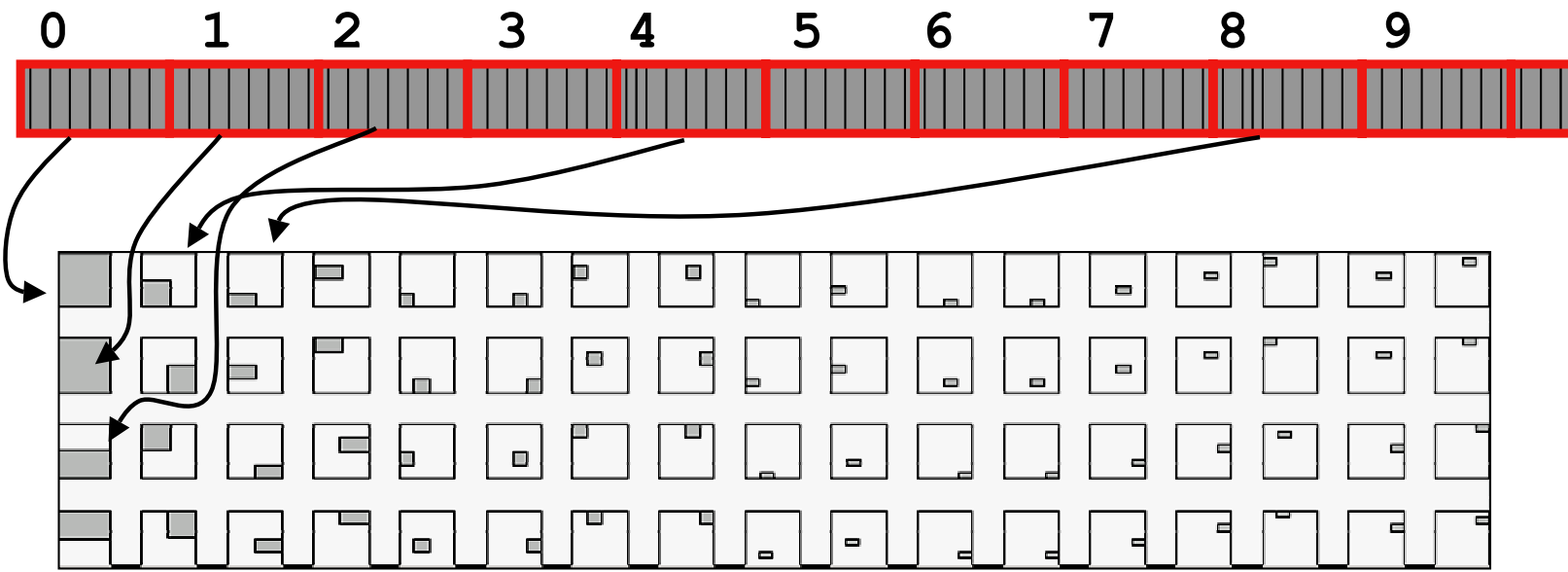


Loop: While the outgoing bit is zero
shift right with incoming bit set to 0

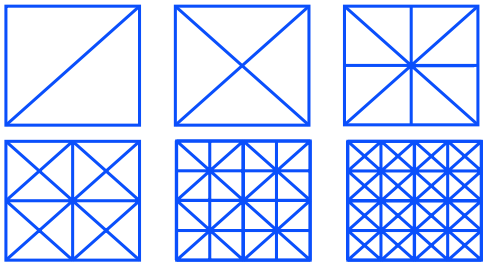


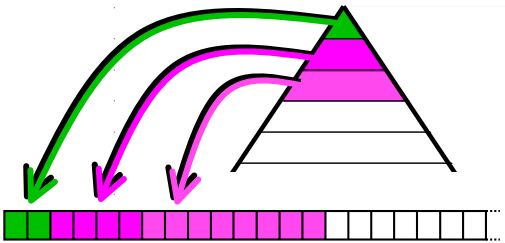
Overall the hierarchical Z-order yields a cache oblivious hierarchical data layout.

Distribution in the grid of each constant size block of data



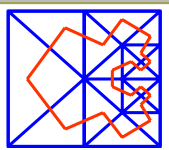
We implemented this combined design for elevation data-sets.

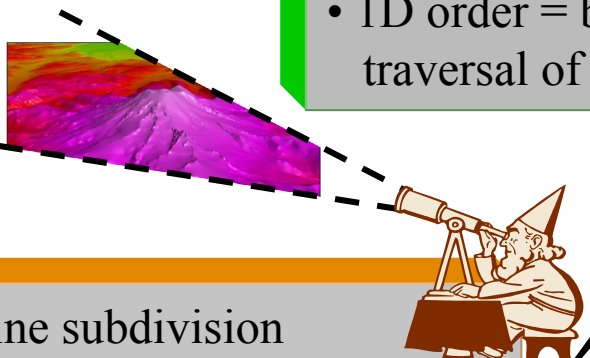




- Rectilinear grid
- Longest edge bisection

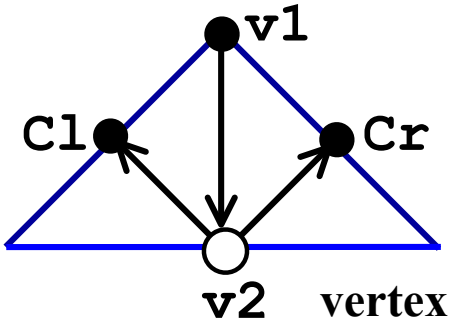
- 1D order = breadth first traversal of the hierarchy





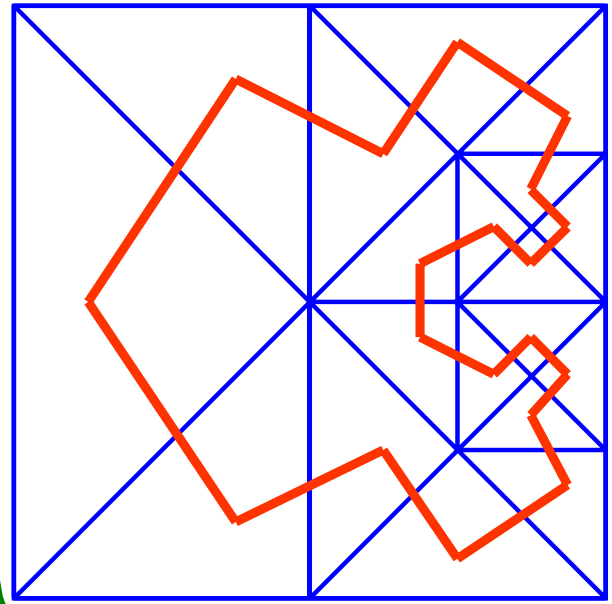
- Coarse to fine subdivision
- View dependent error adaptation
- Frustum culling and stripping

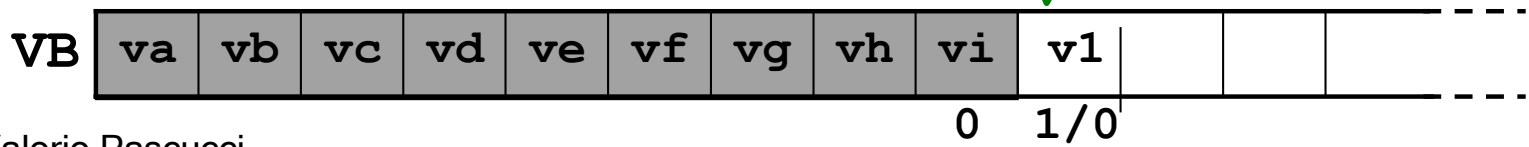
A hierarchical error metric simplifies the mesh construction and stripping.



vertex buffer level

```
mesh-refine(VB,v1,v2,l)
 if l>0 and δ*(v1)≥ε then
  mesh-refine (VB,v2,Cl,l-1)
  strip-append(VB,v1,l mod 2)
  mesh-refine (VB,v2,Cr,l-1)
```

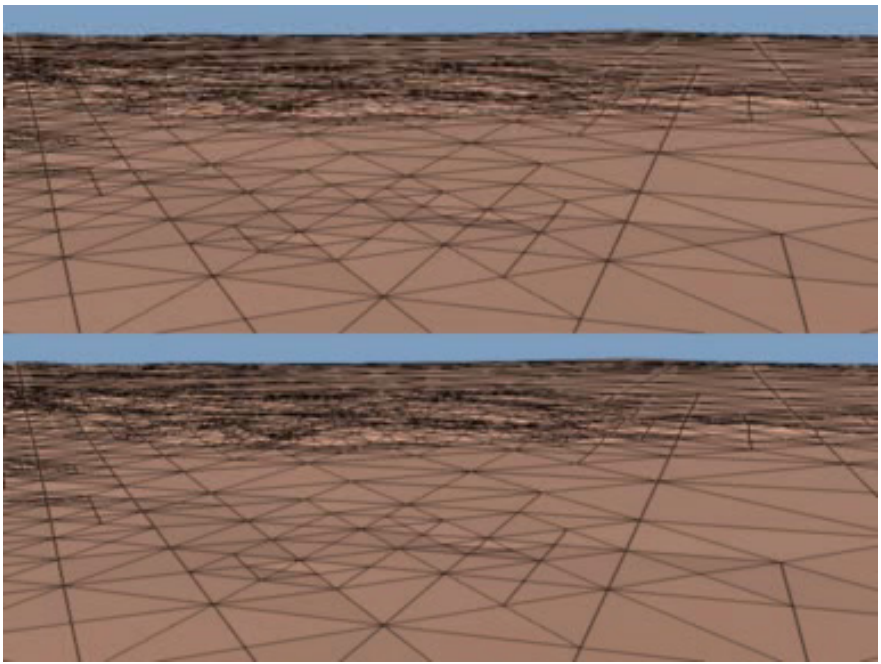




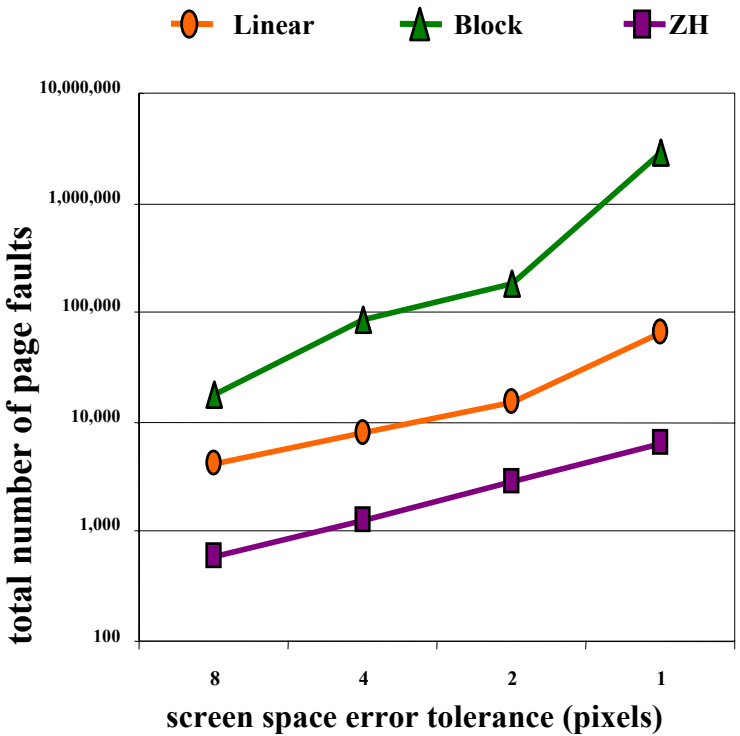
Substantial speedup is obtained due to the cache oblivious data layout.

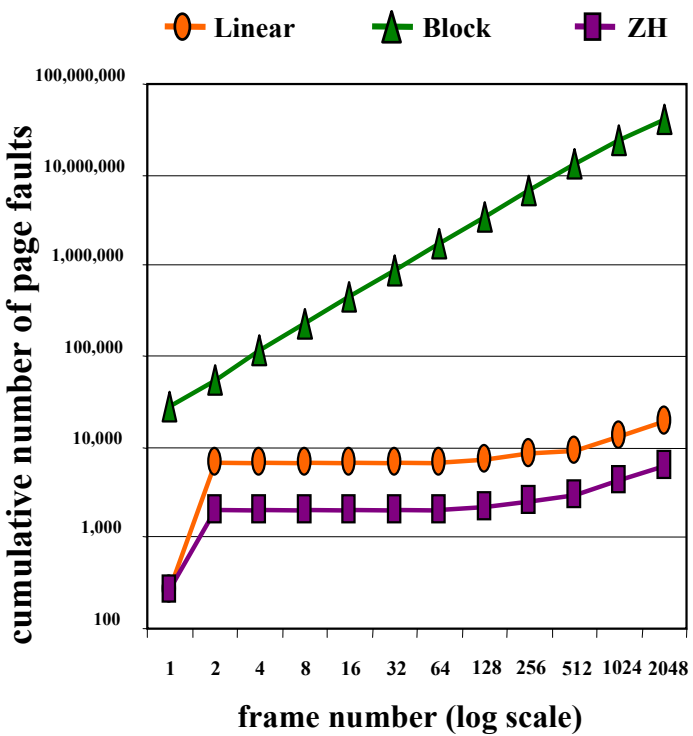
Practical comparison:

- Linear →
- ZH-order →

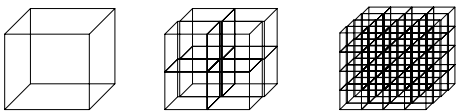


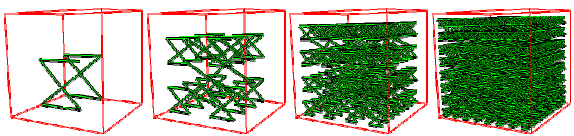
Speedup matches the theoretical expectations without special paging.





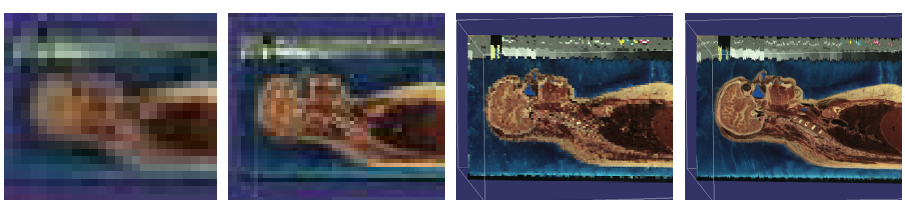
We focus on the progressive slicing (any orientation) of large 3D rectilinear grids.





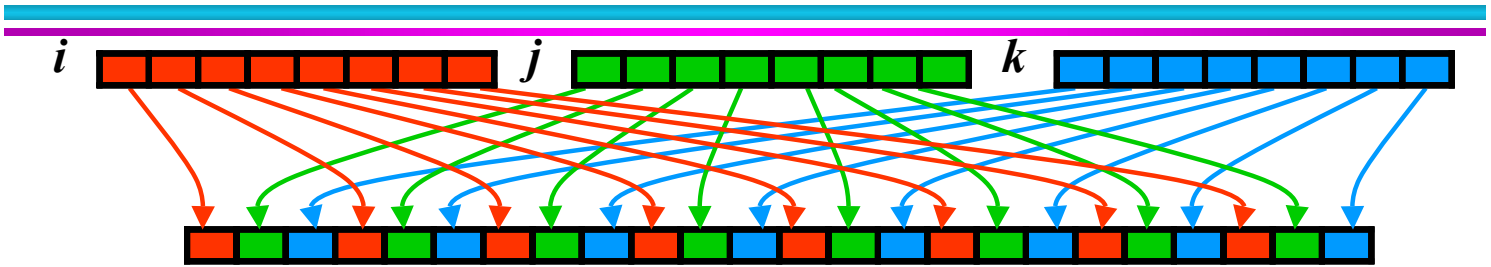
- Rectilinear grid
- Sub-sampling octtree

- 1D order = hierarchical 3D Z-order curve

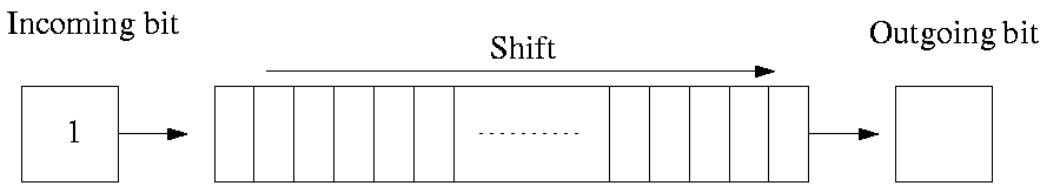


- Coarse to fine slice refinement

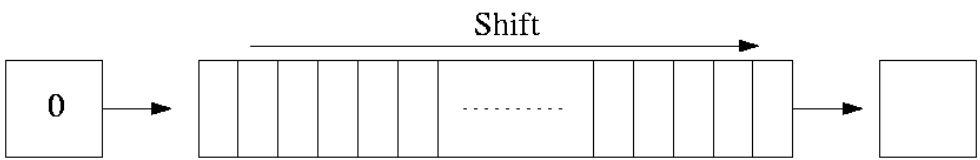
The 1D index I^* can be computed in a simple and efficient way in any dimension.



Step 1: shift right with incoming bit set to 1



Loop: While the outgoing bit is zero
shift right with incoming bit set to 0



The implementation of the bit-shift can be as simple as the following code.

```
inline adhocidex remap(register adhocindex i){
    i |= last_bit_mask; // set rightmost one
    i /= i&-i;          // remove right zeros
    return (i>>1);      // remove right one
}
```

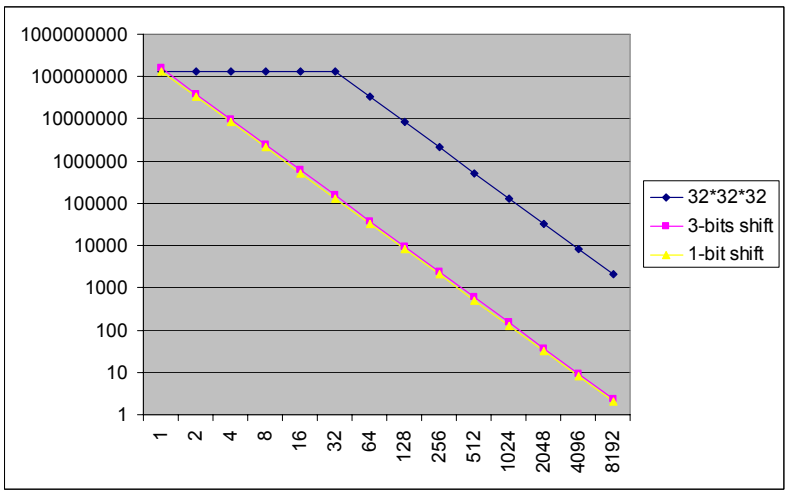
But lookup tables perform better in practice.

Progressive slicing queries designed for coherent data access.

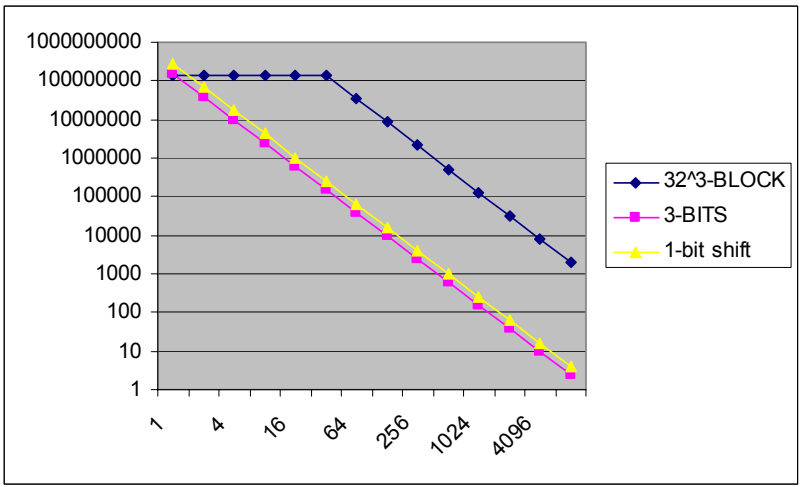
- Increase concurrently slice and sampling resolution.

```
Mask = 111100000; // initial sampling resolution
Size = 32       ; // initial slice resolution
for (l = min_l; l<max_l; l++) do //refinement loop
     for_each (i,j) in image(size) do
          image[i,j] = get_sample(i&Mask,j&Mask)
     draw(image);
     Mask = Mask | (Mask/2);
     Size = Size * 2;
```

Theoretical analysis shows a gain of orders of magnitude independently of the block size.



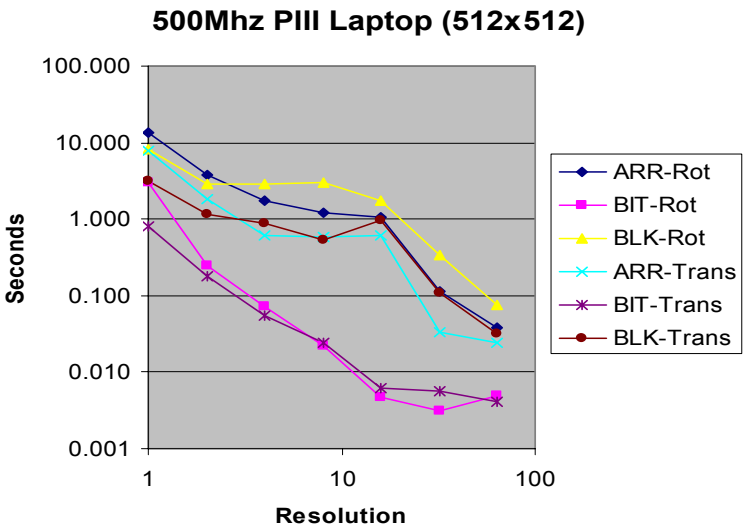
32K blocks

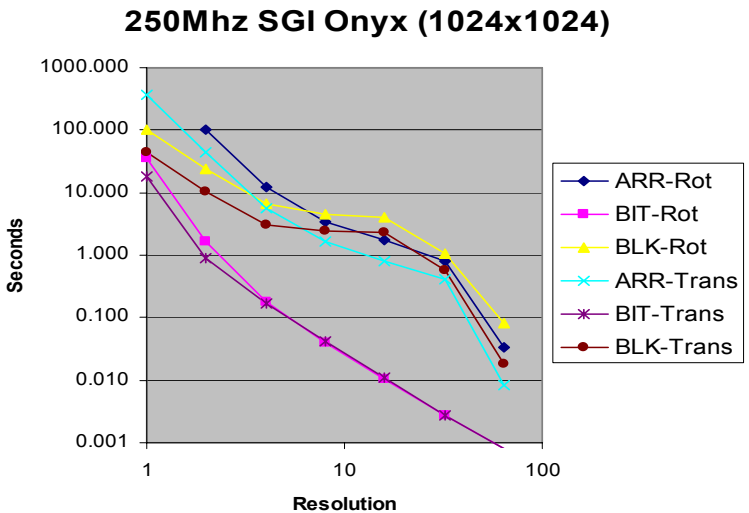


64K blocks

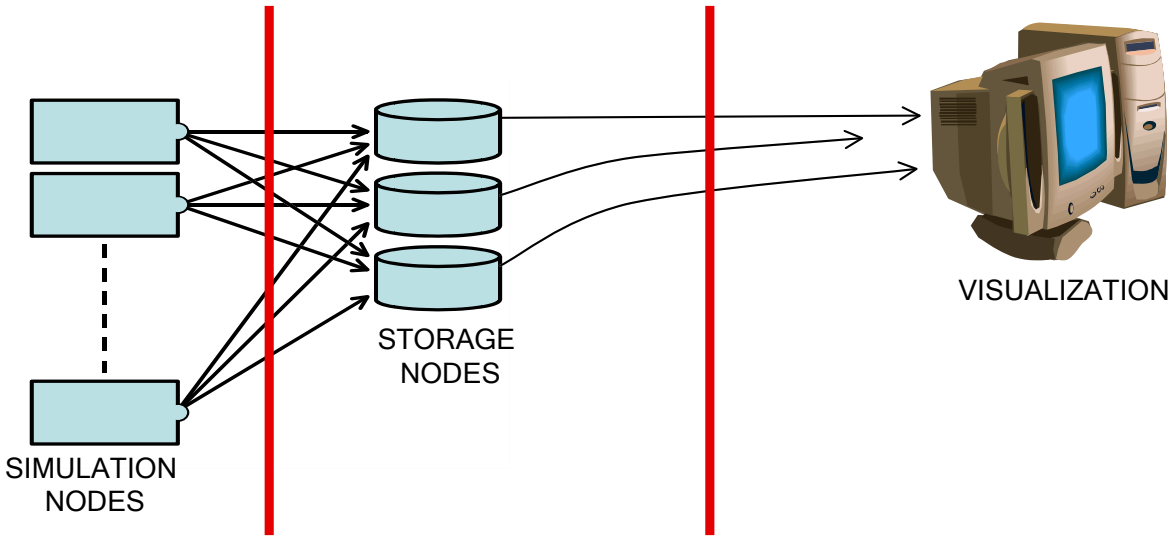
Real speedup matches theoretical expectations: more than 10x improvement, platform scalable.

- 2048x2048x1920 dataset (we have run up to 8192x8192x7680)
- 20MB memory cache
- Translation and rotation tests (average over 3 primary axis)





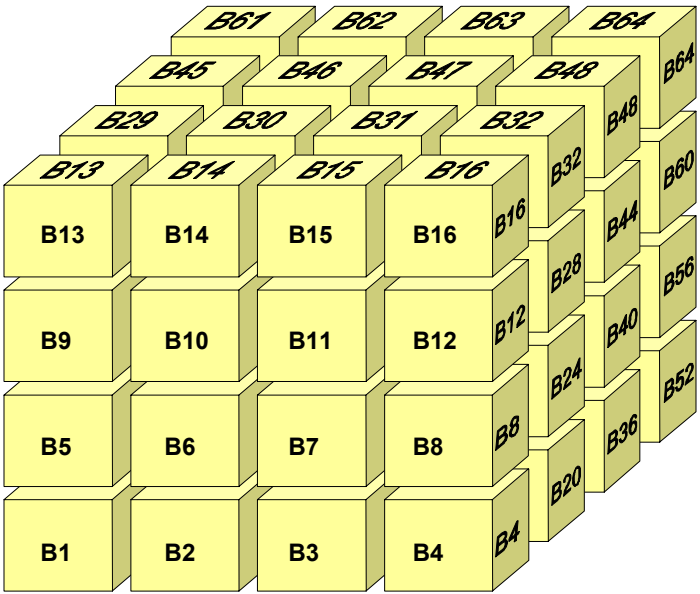
Preprocessing time and load balancing are problematic for distributed streaming.



Delay in data availability depends on preprocessing time

Load balancing is important because last available sample determines performance

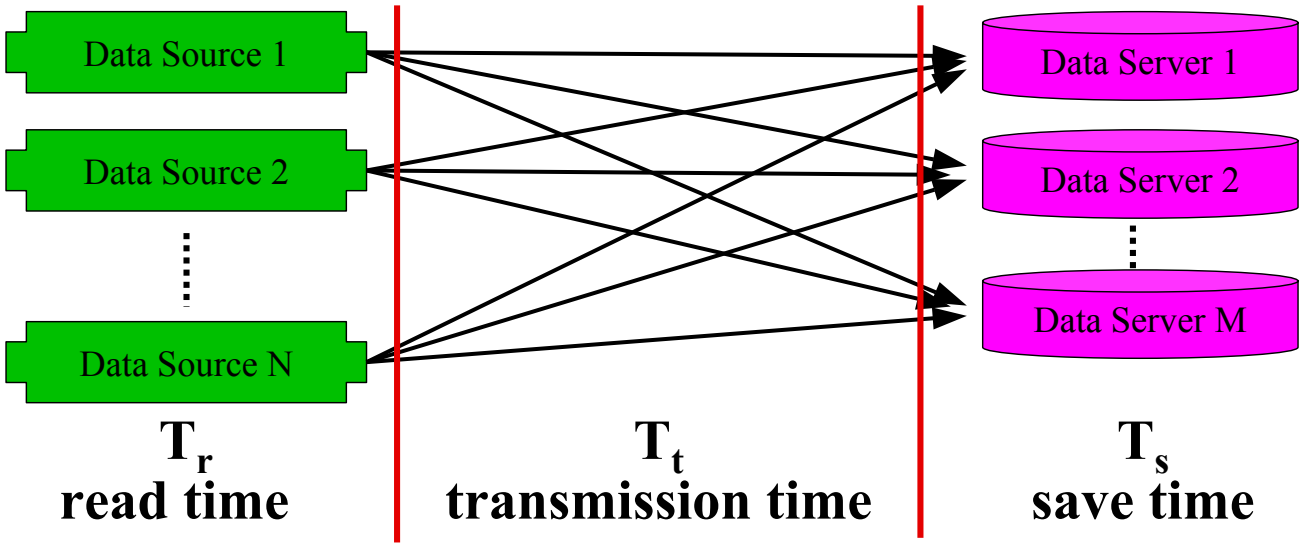
The preprocessing time can be nearly completely hidden in the transmission.



- Take advantage of simulation load balancing
- One Data Source per compute node
- Each compute node connected to all Data Servers

The data reordering can be done in parallel before the transmission with minor overhead.

$$T_{total}=T_r+T_t+T_s$$

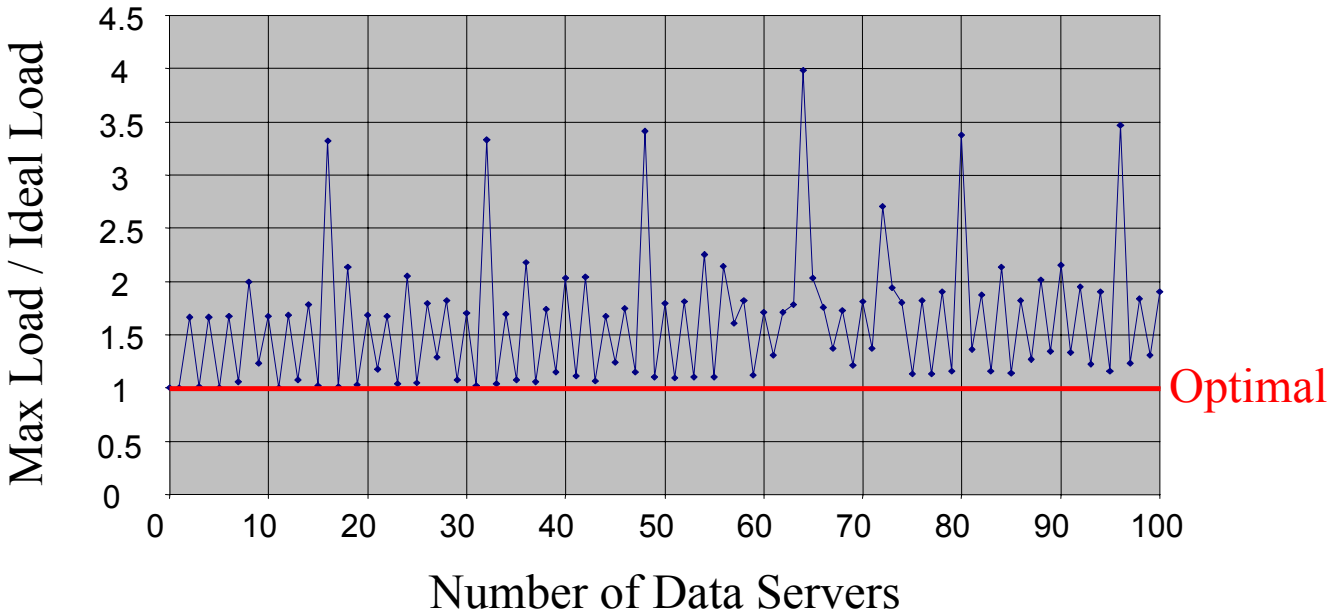


$2k^3$ grid on 64 nodes	T_r	T_s
Reorder first	0	20min
Reorder last	25s	0

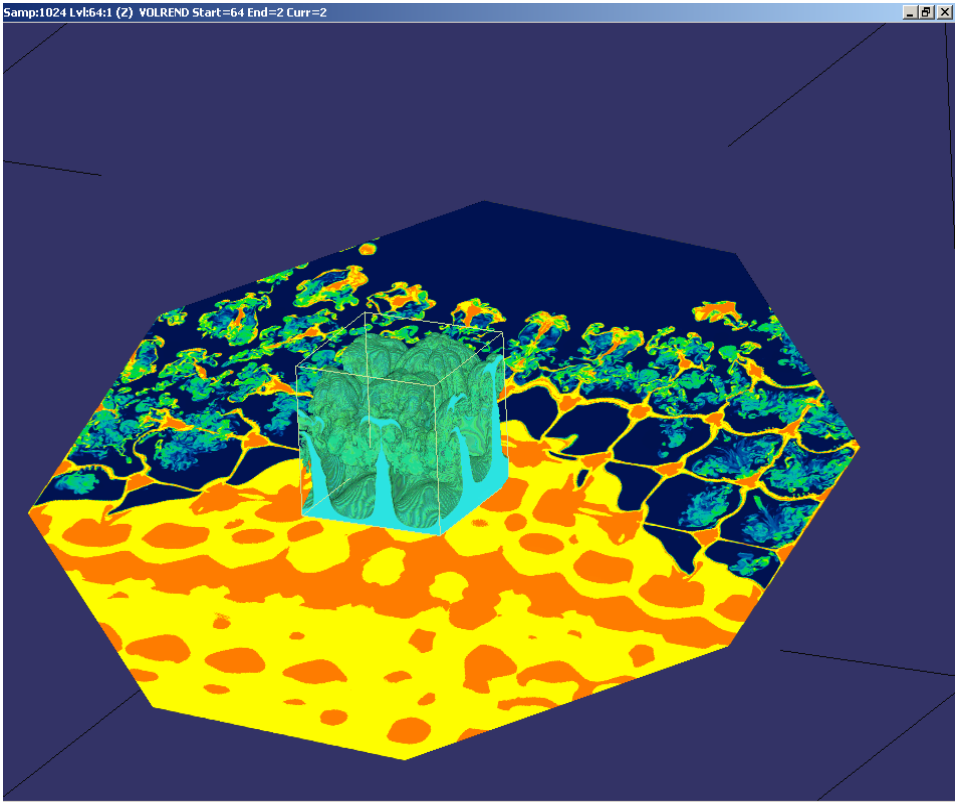
We achieve load balancing with a simple static data decomposition.

- **No contention**
- **No data replication**
- **Streaming sample I from server D:**
  - **D = I/b mod N**
    - D is Data server,
    - I is HZ index of sample,
    - b is data blocking factor,
    - N = number of servers.
- **If N is chosen properly the system provides good load balancing.**

The load balancing depend on the number of server among which the data is distributed.



Questions?



UCRL-PRES-149731

This work was performed under the auspices of the U.S. Department of Energy by University of California Lawrence Livermore National Laboratory under contract No. W-7405-ENG-48.