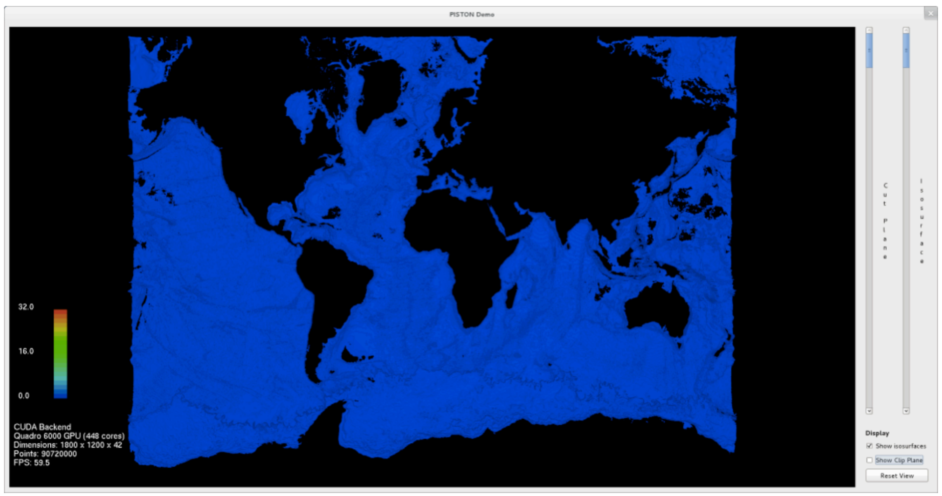
Portability and Performance for Visualization and Analysis Operators Using the Data-Parallel PISTON Framework

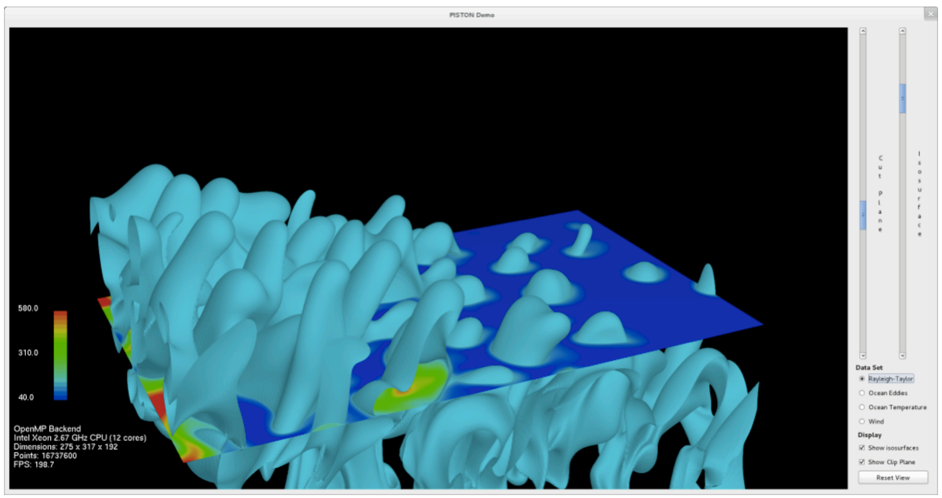
Chris Sewell

Li-Ta Lo

James Ahrens

Los Alamos National Laboratory





Operated by Los Alamos National Security, LLC for the U.S. Department of Energy's NNSA

LA-UR-11-11980

Outline

- Motivation
  - Portability and performance of visualization and analysis operations on current and next-generation supercomputers
- Introduction to data-parallel programming and the Thrust library
- Implementation of visualization operators
  - Isosurface, Cut Surfaces, Threshold
- Current target architectures and performance
  - CUDA/Nvidia GPU & OpenMP/Multi-core machines
- Recent Development
  - Curvilinear coordinates, unstructured grids, VTK/ParaView integration, multi-node parallelism

Motivation / Related Work

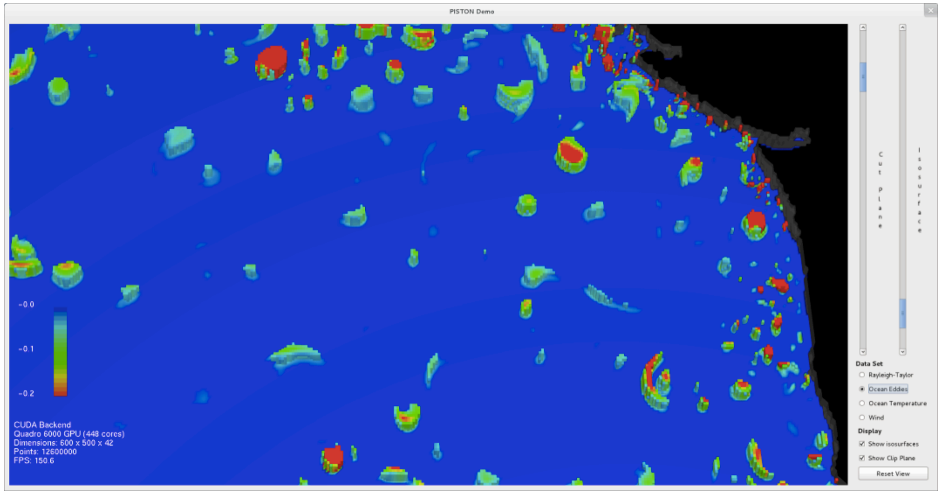
- Current production visualization software does not take full advantage of acceleration hardware and/or multi-core architecture
  - Vtk, ParaView, VisIt
- Research on accelerating visualization operations are mostly hardware-specific; few were integrated in visualization software
  - CUDA SDK demo
  - Dyken, Ziegler, “High-speed Marching Cubes using Histogram Pyramids”, Eurographics 2007.
- Most work in portability and abstraction layers/languages are not ready (yet)…
  - Scout, DAX, Liszt
- OpenCL: code is portable but performance is not
- Can we accelerate our visualization software with something that is based on “proven” technology and portable across different architectures?
  - NVidia Thrust library

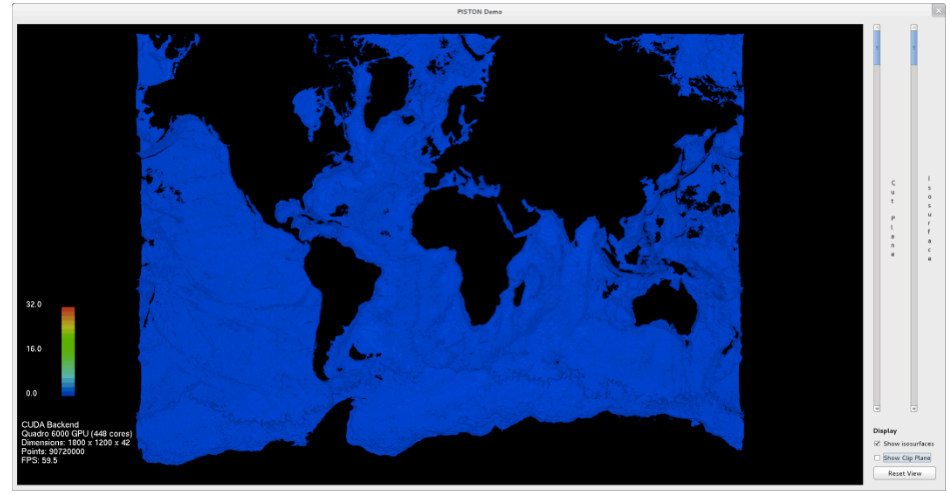
Brief Introduction to Data-Parallel Programming and Thrust

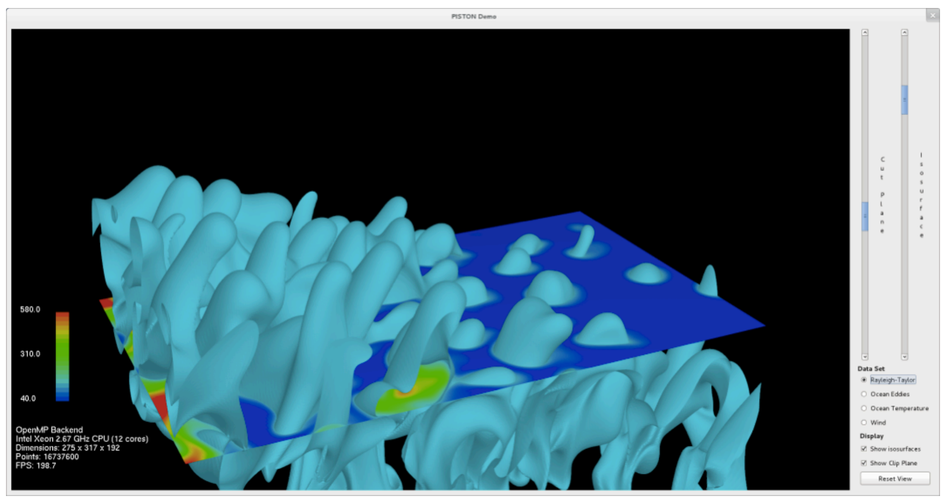
- What is data parallelism?
  - When independent processors performs the same task on different pieces of data
  - Due to the massive data sizes we expect to be simulating we expect data parallelism to be a good way to exploit parallelism on current and next generation architectures
  - "The data parallel bible" - Blelloch, "Vector Models for Data Parallel Computing"

- What is Thrust?
  - Thrust is a NVidia C++ template library for CUDA. It can also target OpenMP and we are creating new backends to target other architectures
  - Thrust allows you to program using an interface similar the C++ Standard Template Library (STL)
  - Most of the STL algorithms in Thrust are data parallel
  - Provided device_vector and host_vector data structures simplify memory management and host/device memory transfers

Videos of PISTON in Action







Brief Introduction to Data-Parallel Programming and Thrust

What algorithms does Thrust provide?

- Sorts
- Transforms
- Reductions
- Scans
- Binary searches
- Stream compactions
- Scatters / gathers

Challenge: Write operators in terms of these primitives only

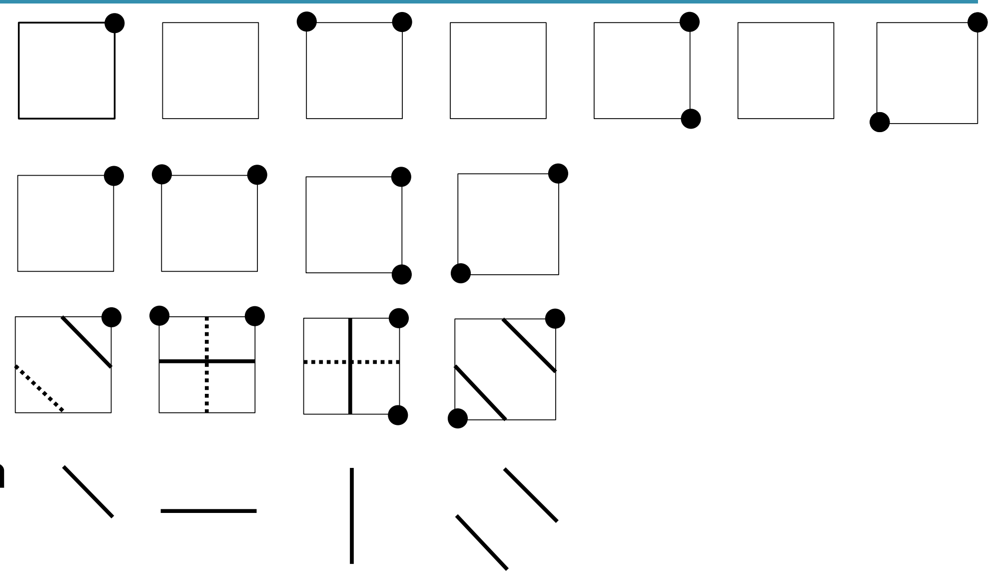
Reward: Efficient, portable code

```
input                    4   5   2   1   3
--------------------------------------------
transform(+1)            5   6   3   2   4
inclusive_scan(+)        4   9  11  12  15
exclusive_scan(+)        0   4   9  11  12
exclusive_scan(max)      0   4   5   5   5
transform_inscan(*2,+)   8  18  22  24  30
for_each(-1)             3   4   1   0   2
sort                     1   2   3   4   5
copy_if(n % 2 == 1)      5   1   3
reduce(+)                               15

input1                   0   0   2   4   8
input2                   3   4   1   0   2
--------------------------------------------
upper_bound              3   4   2   2   3
permutation_iterator     4   8   0   0   2
```

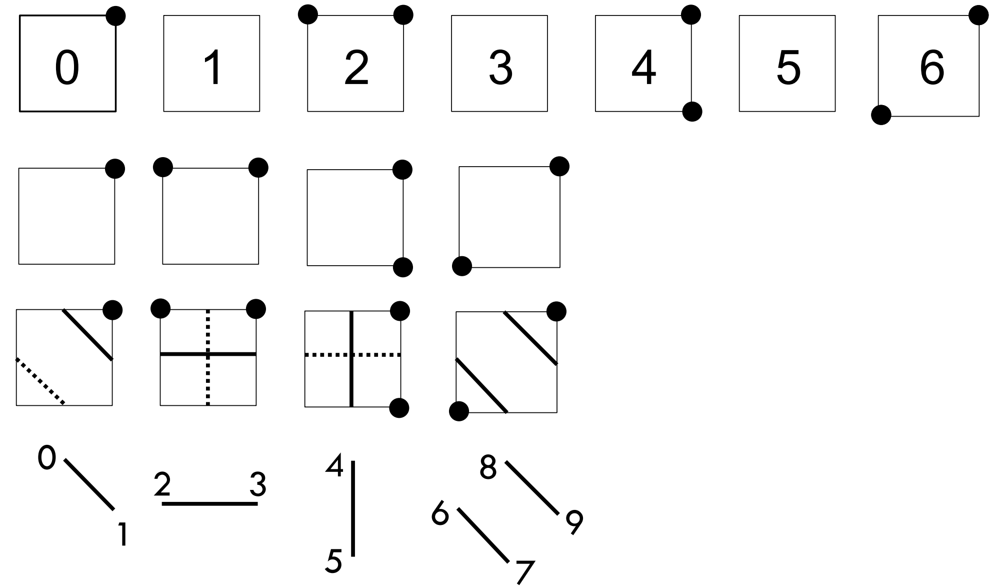
Isosurface with Marching Cube – the Naive Way

- Classify all cells by *transform*
- Use *copy_if* to compact valid cells.
- For each valid cell, generate same number of geometries with flags.
- Use *copy_if* to do stream compaction on vertices.
- This approach is too slow, more than 50% of time was spent moving huge amount of data in global memory.
- Can we avoid calling *copy_if* and eliminate global memory movement?



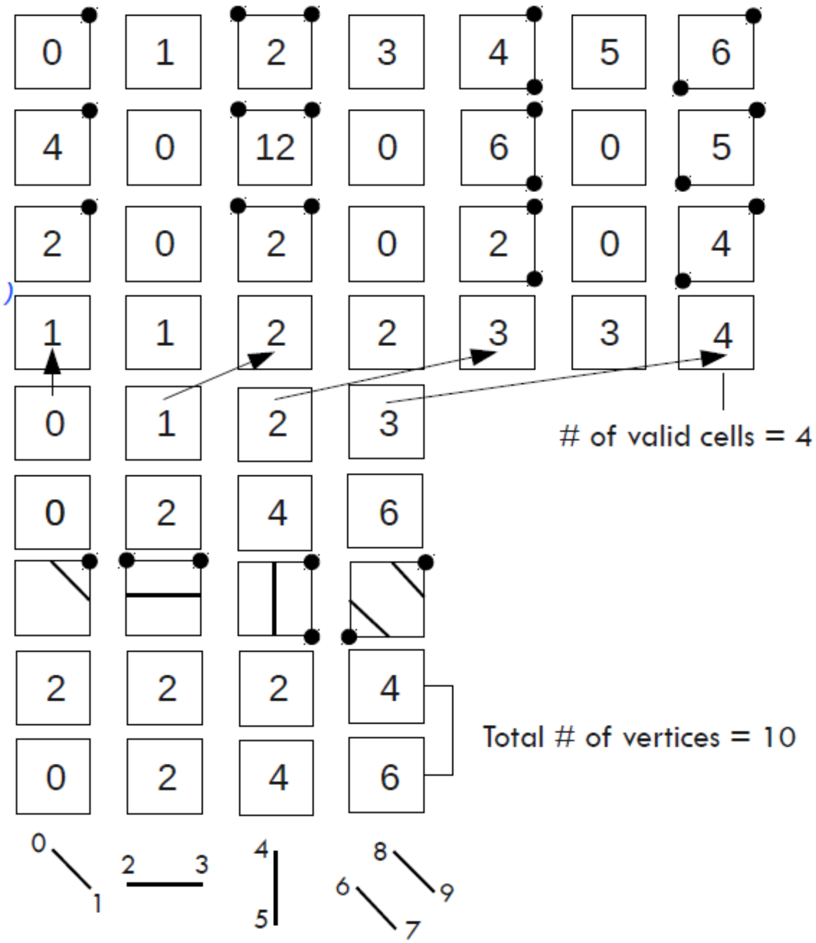
Isosurface with Marching Cube – Optimization

- Inspired by HistoPyramid
- The filter is essentially a mapping from input cell id to output vertex id
- Is there a "reverse" mapping?
- If there is a reverse mapping, the filter can be very "lazy"
- Given an output vertex id, we *only* apply operations on the cell that would generate the vertex
- Actually for a range of output vertex ids



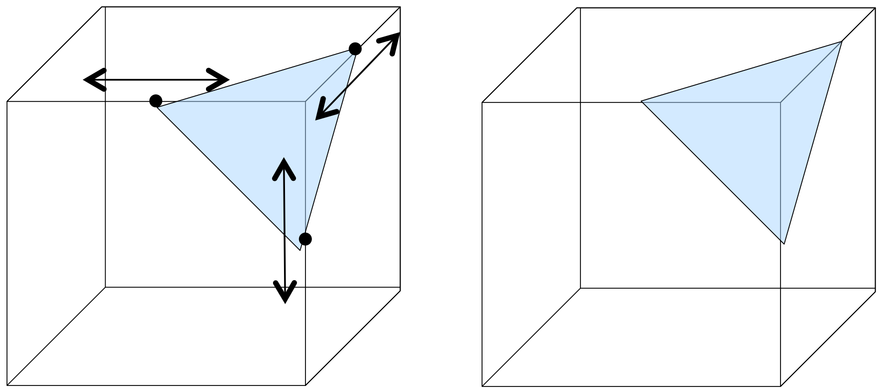
Isosurface with Marching Cubes Algorithm

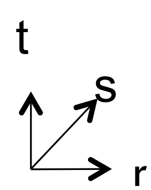
1. input
   transform(classify_cell)
2. caseNums
3. numVertices
   transform_inclusive_scan(is_valid_cell)
4. validCellEnum
5. CountingIterator
   upper_bound
6. validCellIndices
   make_permutation_iterator
7. numVerticesCompacted
   exclusive_scan
8. numVerticesEnum
   for_each(isosurface_functor)
9. outputVertices



Cut Surfaces

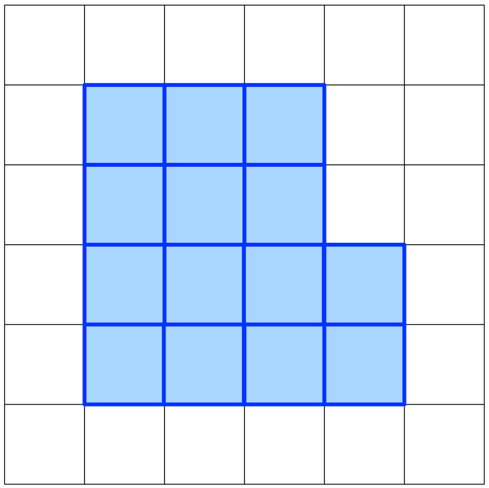
- All the vertices generated by marching cube are on the cell edges.
- They have only one degree of freedom, not three.
- 1D interpolation only, no need to do trilinear interpolation on scalar field.
- Two scalar fields, one for generating geometry (cut surface) the other for scalar interpolation.
- Less than 10 LOC change, negligible performance impact to isosurface.

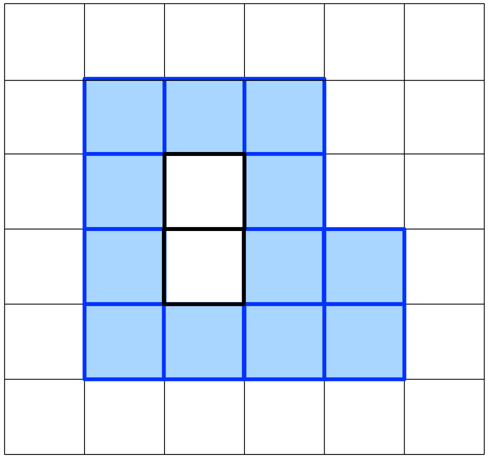




Threshold

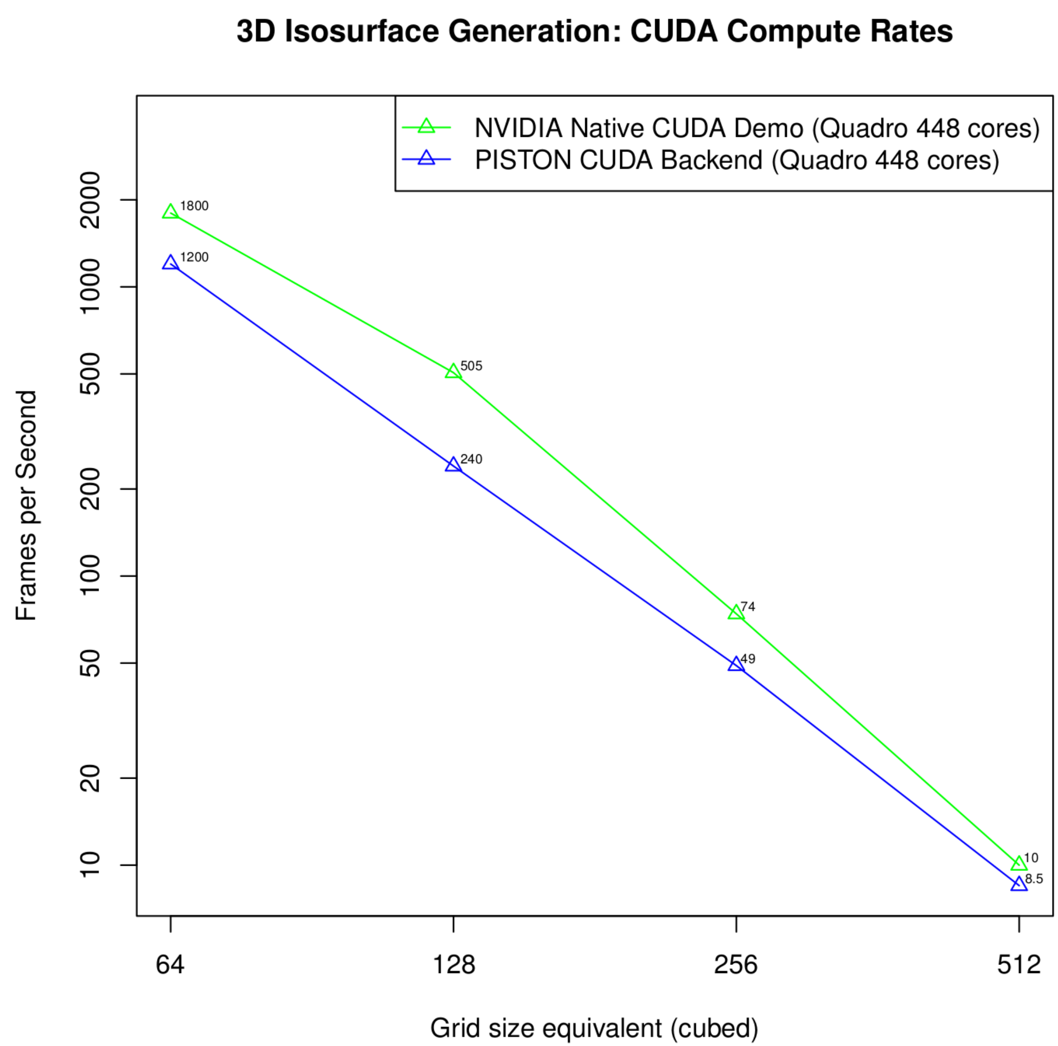
- Again, very similar to marching cube
  - Classify cells, stream compact valid cells and generate geometries for valid cells.
  - Optimization: what does the "inside" of a brick look like? Do we even care?
- Additional passes of cell classification and stream compaction to remove "interior cells"





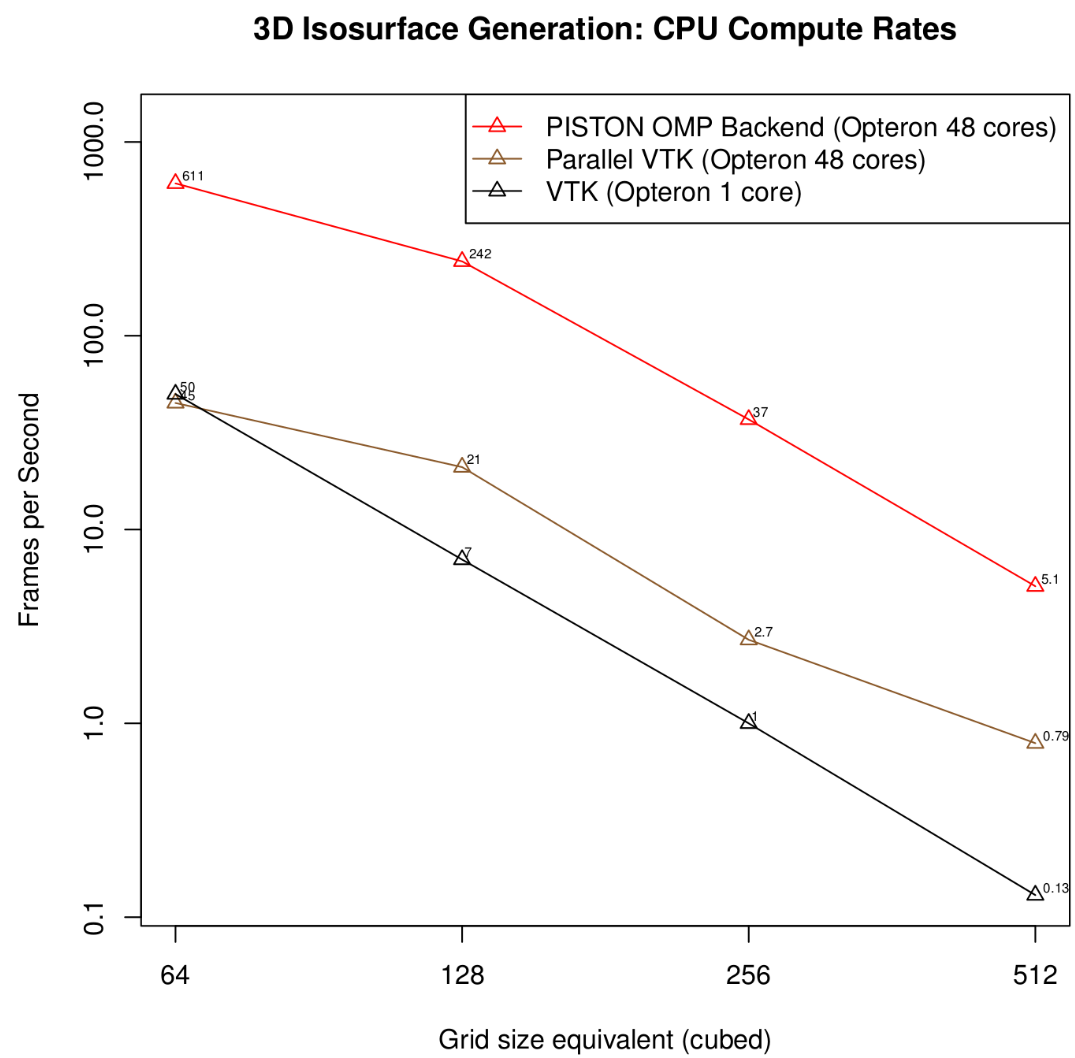
PISTON CUDA Backend Performance

- Limited performance degradation relative to native CUDA optimized code
- PISTON
  - Limited use of shared/texture memory due to portability
- NVIDIA CUDA Demo
  - Works only with data set with power of 2 per dimension, allowing use of shift instead of integer division
  - Memory inefficient; runs out of texture/global memory when data size is larger than 512^3



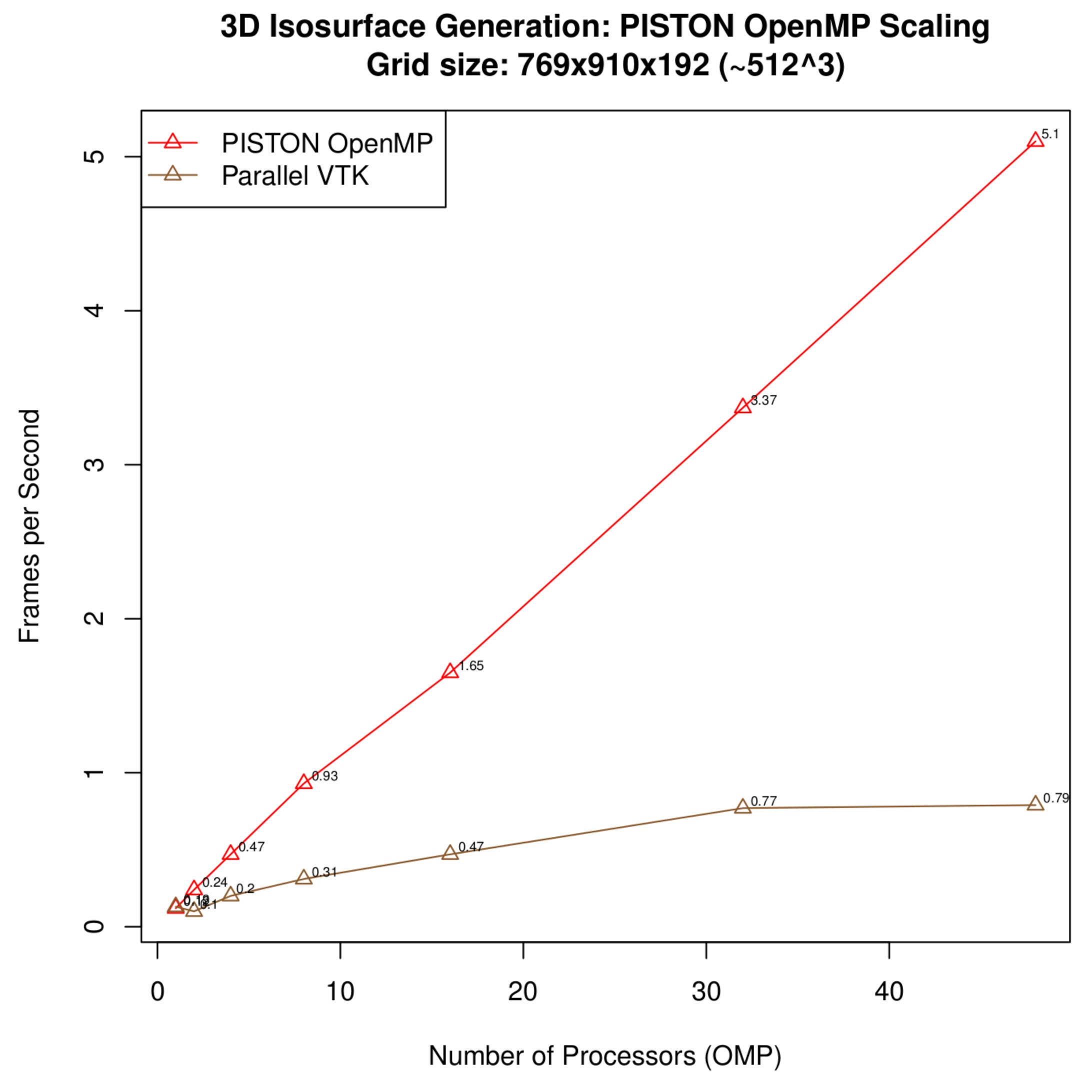
PISTON OpenMP Backend Performance

- Compile time #define/-D switches between backends
- Wrote our own parallel scan implementation for Thrust OpenMP backend
- Significantly better performance than both single process and parallel VTK



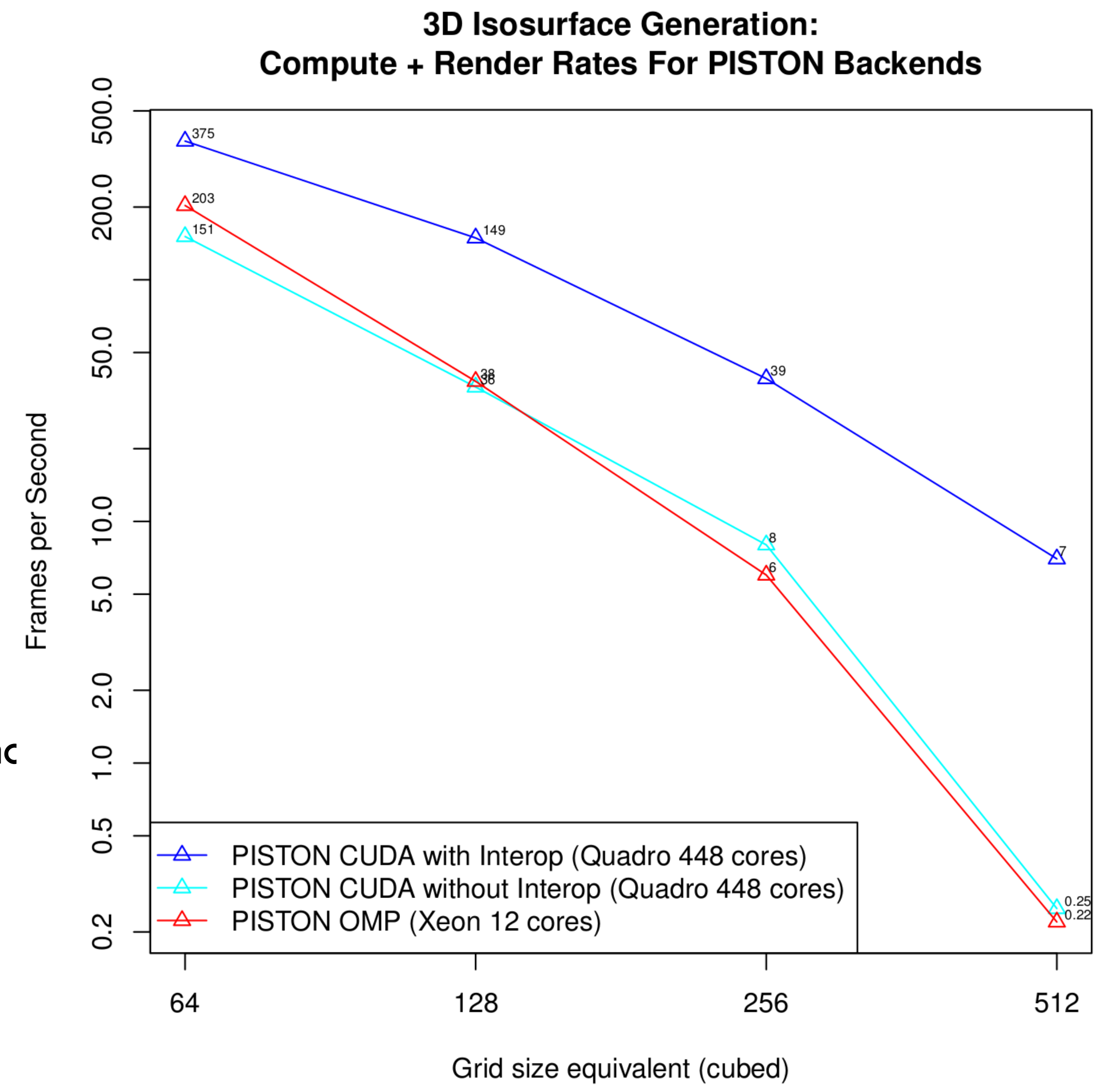
PISTON OpenMP Scaling Performance

- Significantly better scalability in term of # of cores than parallel VTK



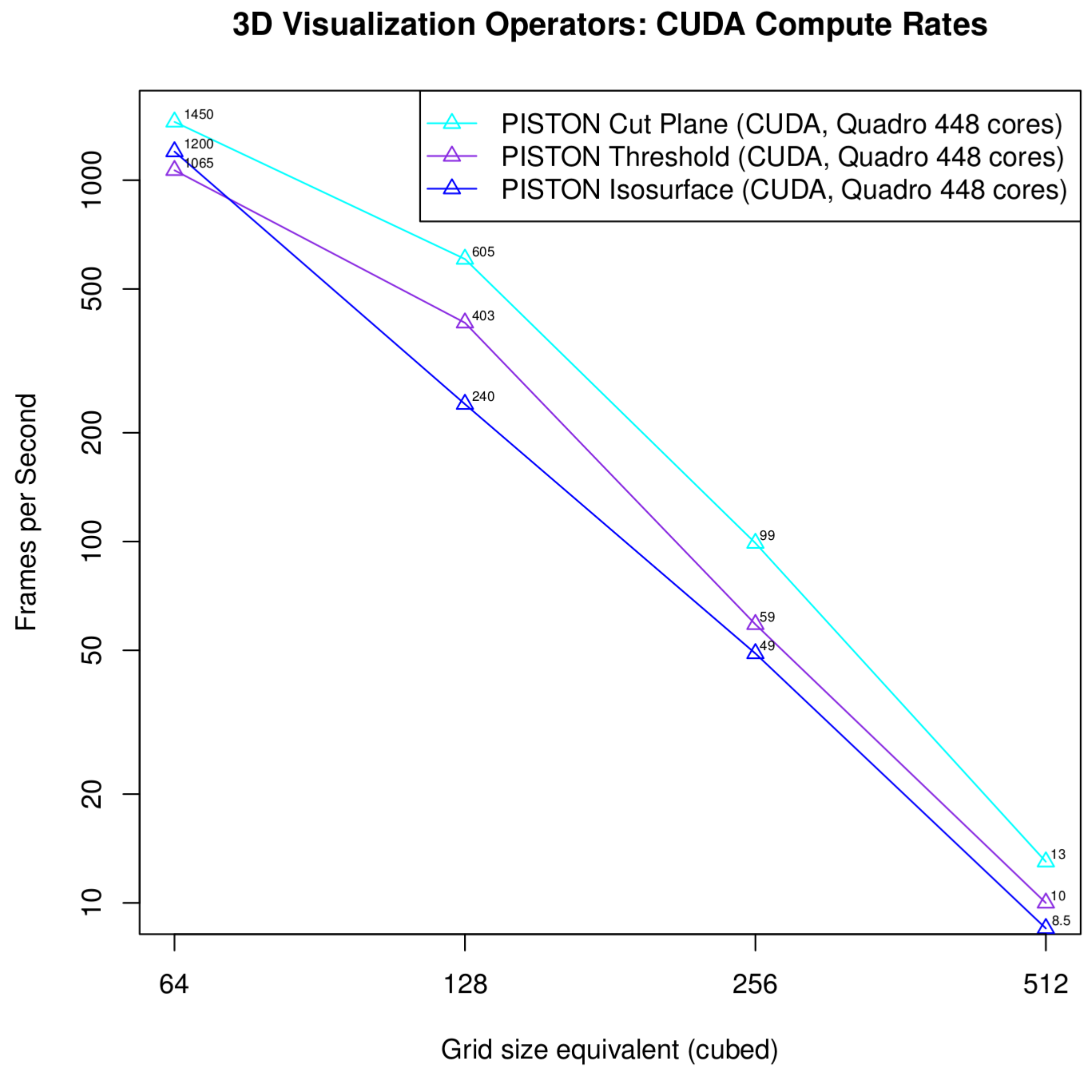
PISTON Compute and Render Results

- Compute and render results
    - CUDA and OpenMP backends
- CUDA/OpenGL interop
    - Platform specific, non-portable
    - Output geometries directly into OpenGL VBO
    - Avoid round trip between device and host memory movement
    - Vastly improves rendering performanc and reduces memory footprint



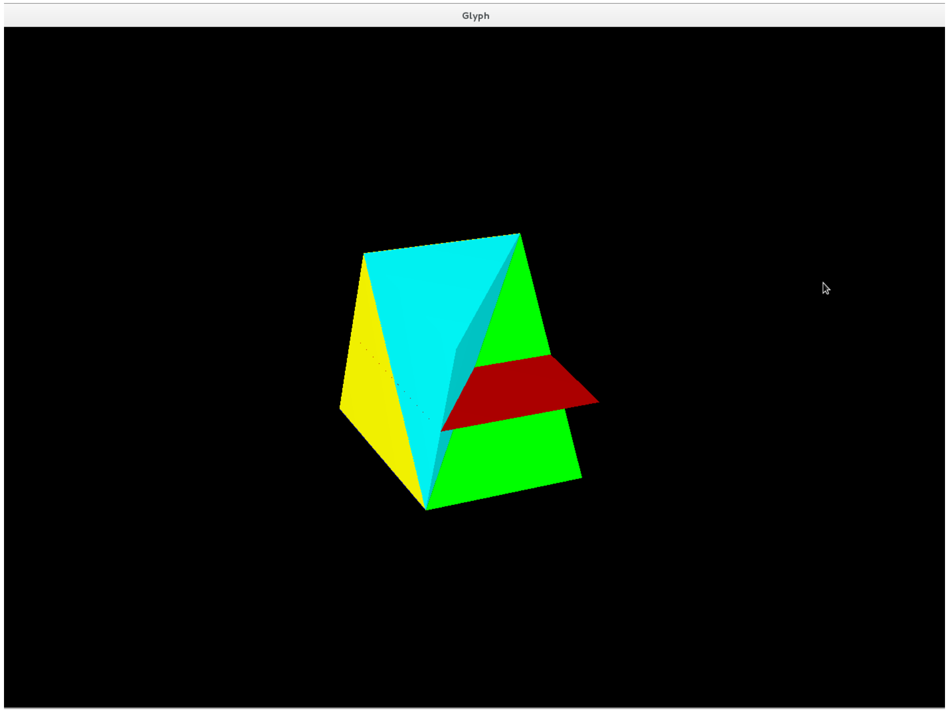
PISTON Visualization Operators

- Three fundamental visualization operations
- All based on the same basic data-parallelism
- Very similar performance characteristics
  - Cut plane is the fastest since it generates 2D planes
  - Threshold comes next because there is no interpolation for scalar nor position
  - Isosurface is actually the most complicated operator



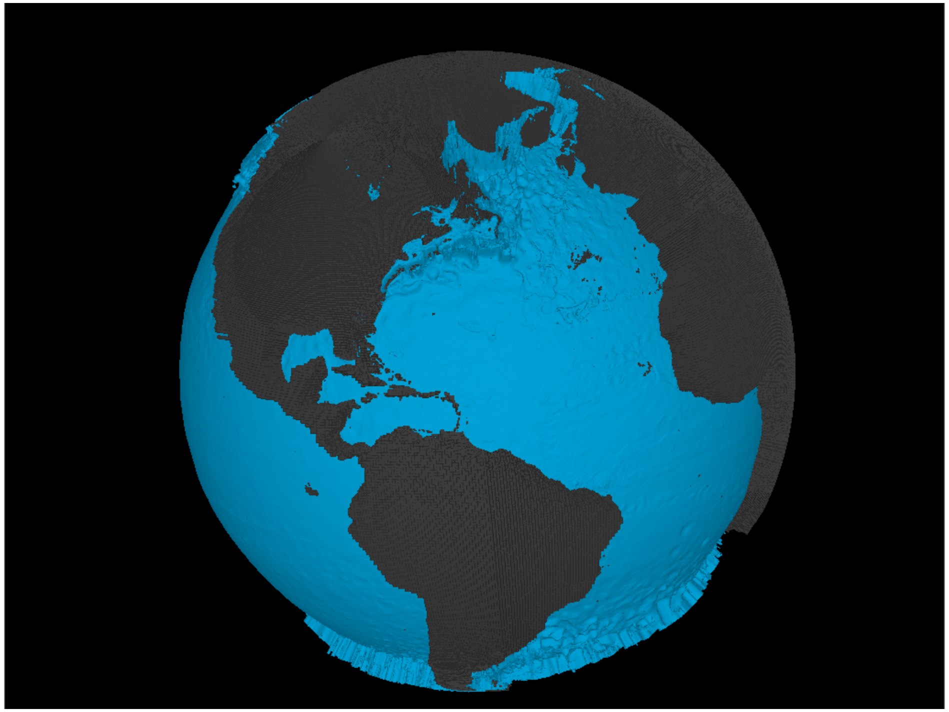
Recent Development – Marching Tetrahedra

- Current procedure
  - Tetrahedralize uniform grid
  - Generate isosurface geometry based on look-up table for tetrahedral cells
- Next step: tetrahedralize unstructured grids
- Polytypic algorithm design



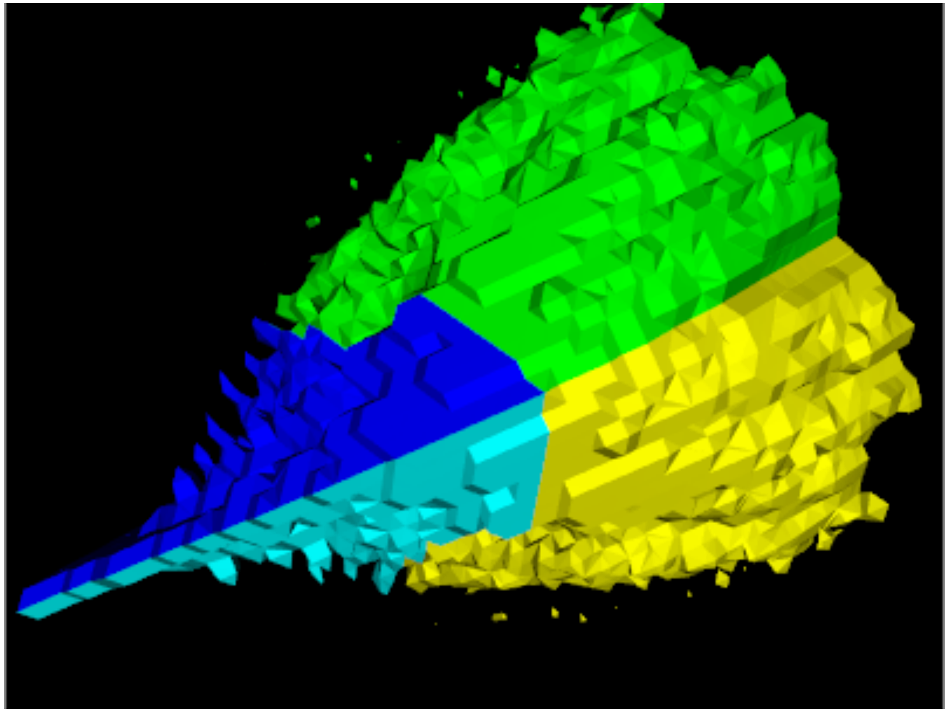
Recent Development – Curvilinear Coordinates

- Generic algorithm design
  - High-level algorithms are independent of coordinate systems
  - Implemented by multiple layers of coordinate transformations
  - Due to kernel fusion, very little performance impact



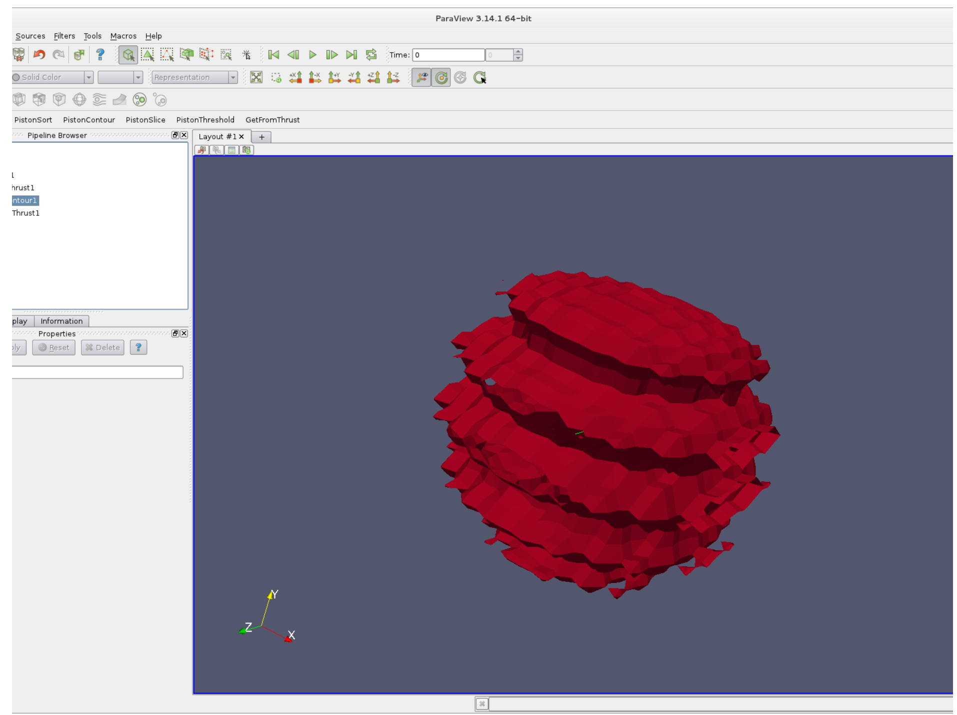
Recent Development – VTK Integration

- Done by Dave DeMarle and Aashish Chaudhary
- VTK/PISTON Data Transfer
- PISTON filters can be chained to keep intermediate data on GPU
- Domain decomposition and parallel rendering by VTK's parallel infrastructure
- In VTK Master repository



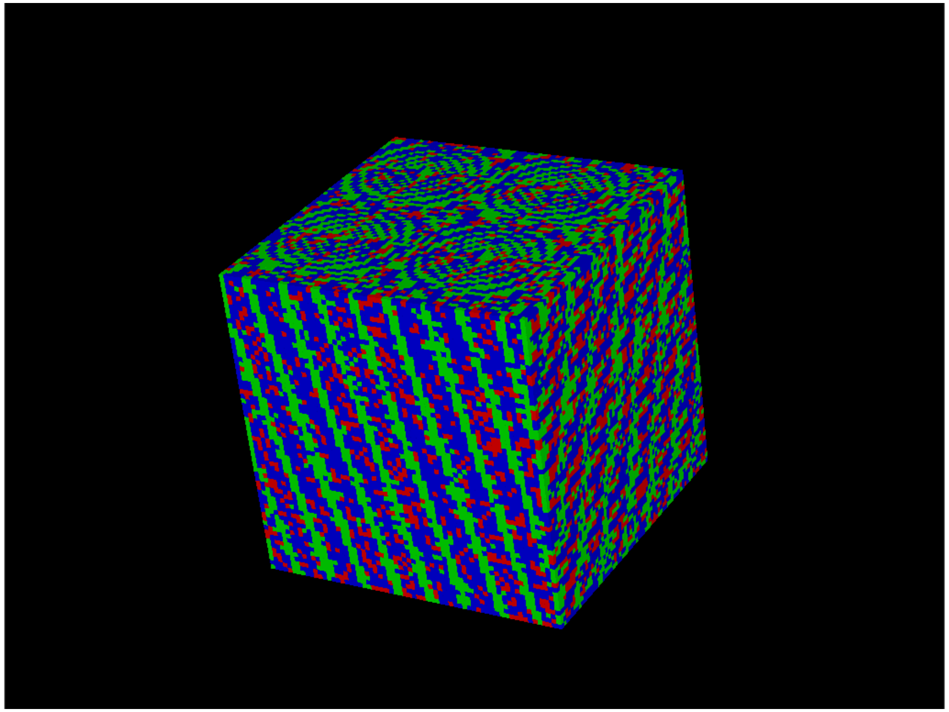
Recent Development – ParaView Intergration

- Uses VTK/PISTON filters in ParaView
- Supports OpenGL Interop, no data movement between GPU/CPU for rendering
- Parallel rendering is not yet supported
- In ParaView Master repository



Work in Progress

- ExMatEx In-situ simulation and visualization – Chris Sewell
- Data Parallel Halo Finder using KD-Tree – Wathsala Widanagamaachchi
- Improvement in the render operator - Jeffrey Sukharev
- Complementary work on physics simulation funded by LDRD ER



Open-Source Release

- Open-source release
  - Tarball: http://viz.lanl.gov/projects/PISTON.html
  - Repository: https://github.com/losalamos/PISTON
- VTK/ParaView Integration
  - VTK/PV git repository: http://paraview.org/ParaView.git

Acknowledgments and Resources

- The work on PISTON was funded by the NNSA ASC CCSE Program, Thuc Hoang, national program manager, Bob Webster and David Daniel, Los Alamos program managers

- For more information, see
  http://viz.lanl.gov/projects/PISTON.html