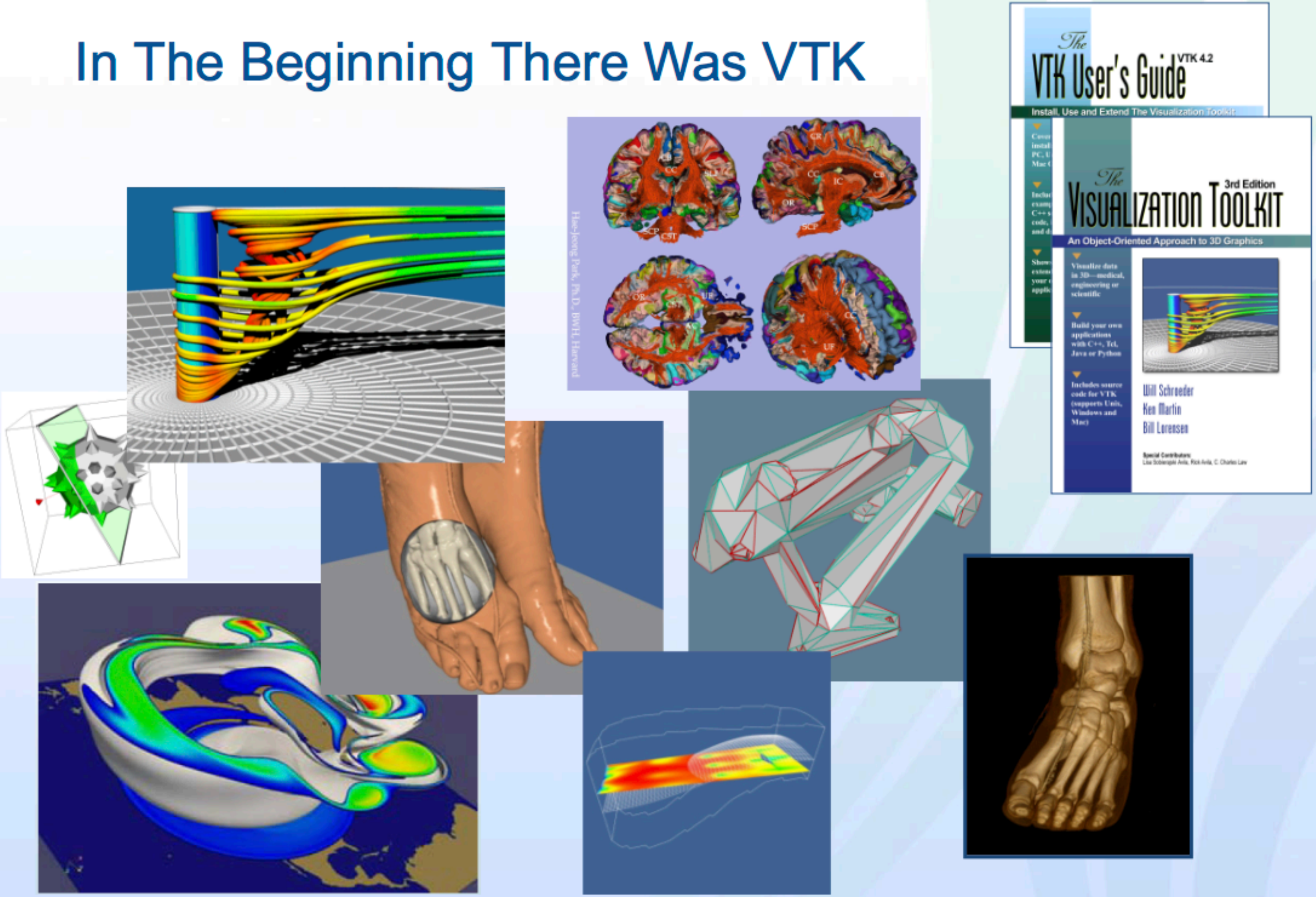
ParaView and VTK with OSPRay and OpenSWR

David DeMarle, Intel HPC DevCon 2016

In The Beginning There Was VTK



The VTK User's Guide VTK 4.2

Install, Use and Extend The Visualization Toolkit

The Visualization Toolkit 3rd Edition

An Object-Oriented Approach to 3D Graphics

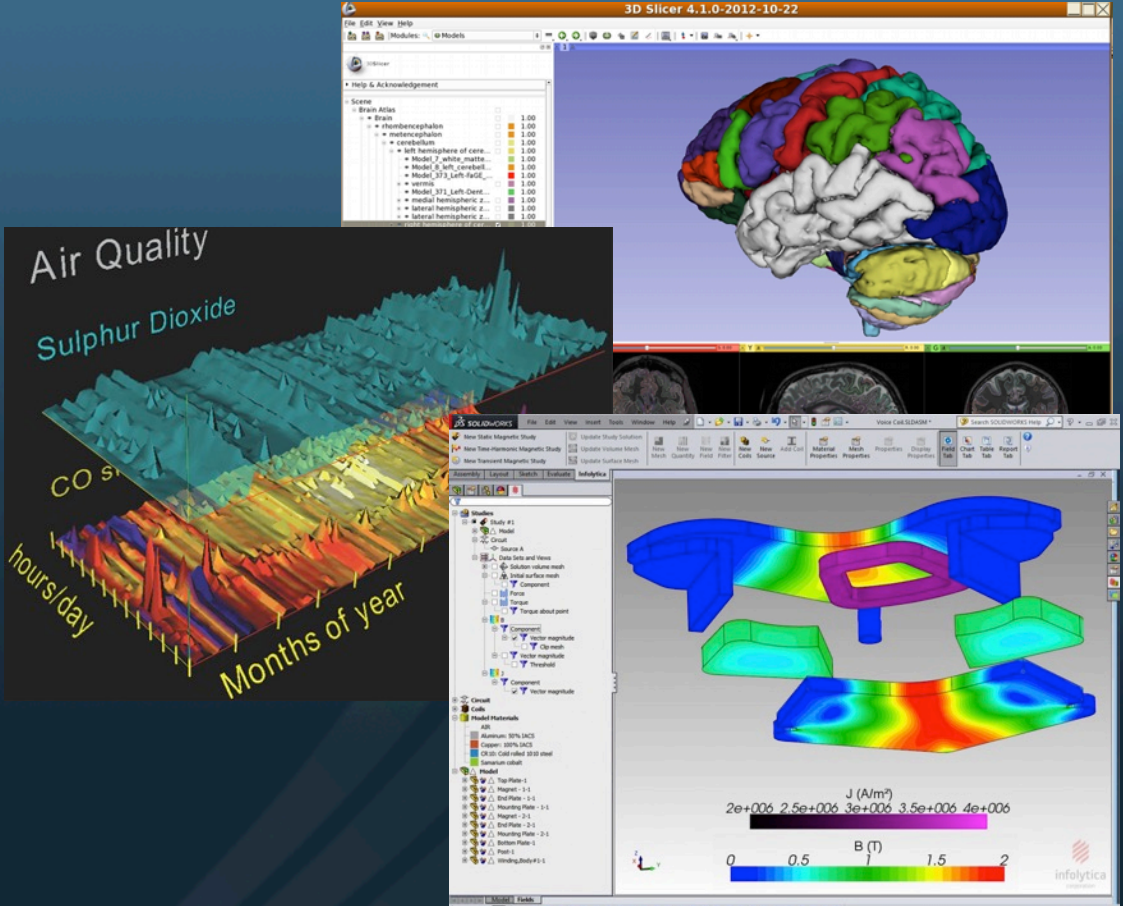
- Visualize data in 3D—medical, engineering or scientific
- Build your own applications with C++, Tcl, Java or Python
- Includes source code for VTK (supports Unix, Windows and Mac)

Will Schroeder
Ken Martin
Bill Lorensen

Special Contributors:
Lisa Sobierajski Avila, Rick Avila, C. Charles Law

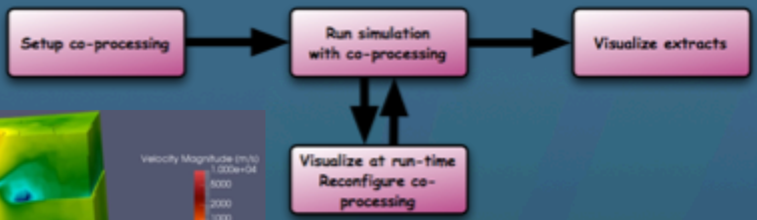
VTK - open source visualization library

- Visualization: Processing + ***Rendering*** + Interaction
- Desktop (win/mac/ linux), Mobile (iOS, android), HPC, Web
- Open Source BSD (commercially friendly)

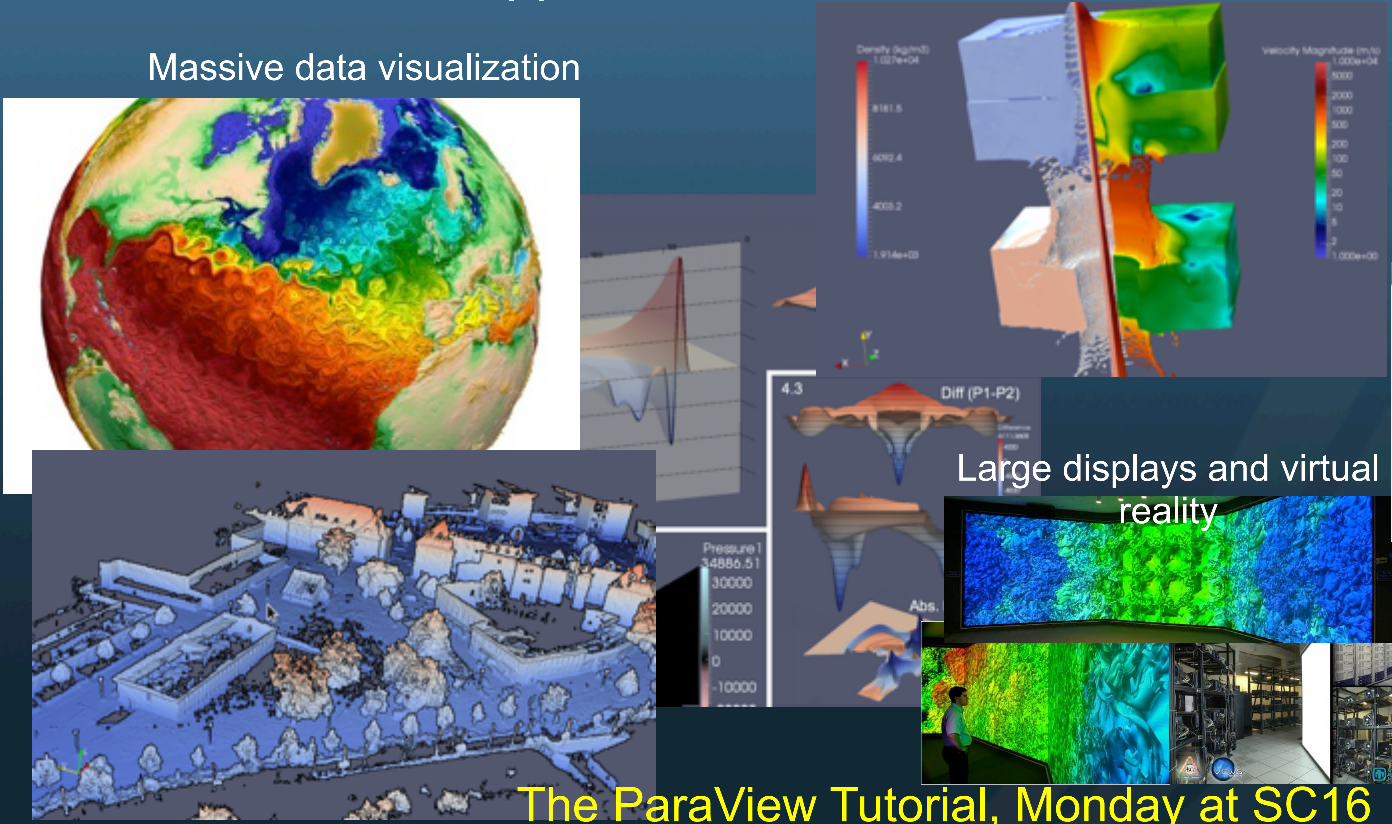


ParaView - scalable data analysis and visualization application

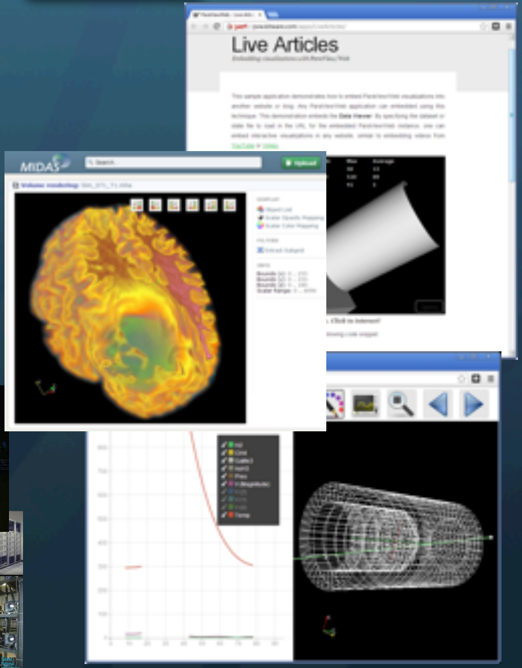
Catalyst In-Situ


Setup co-processing
Run simulation with co-processing
Visualize extracts
Visualize at run-time Reconfigure co-processing

Massive data visualization

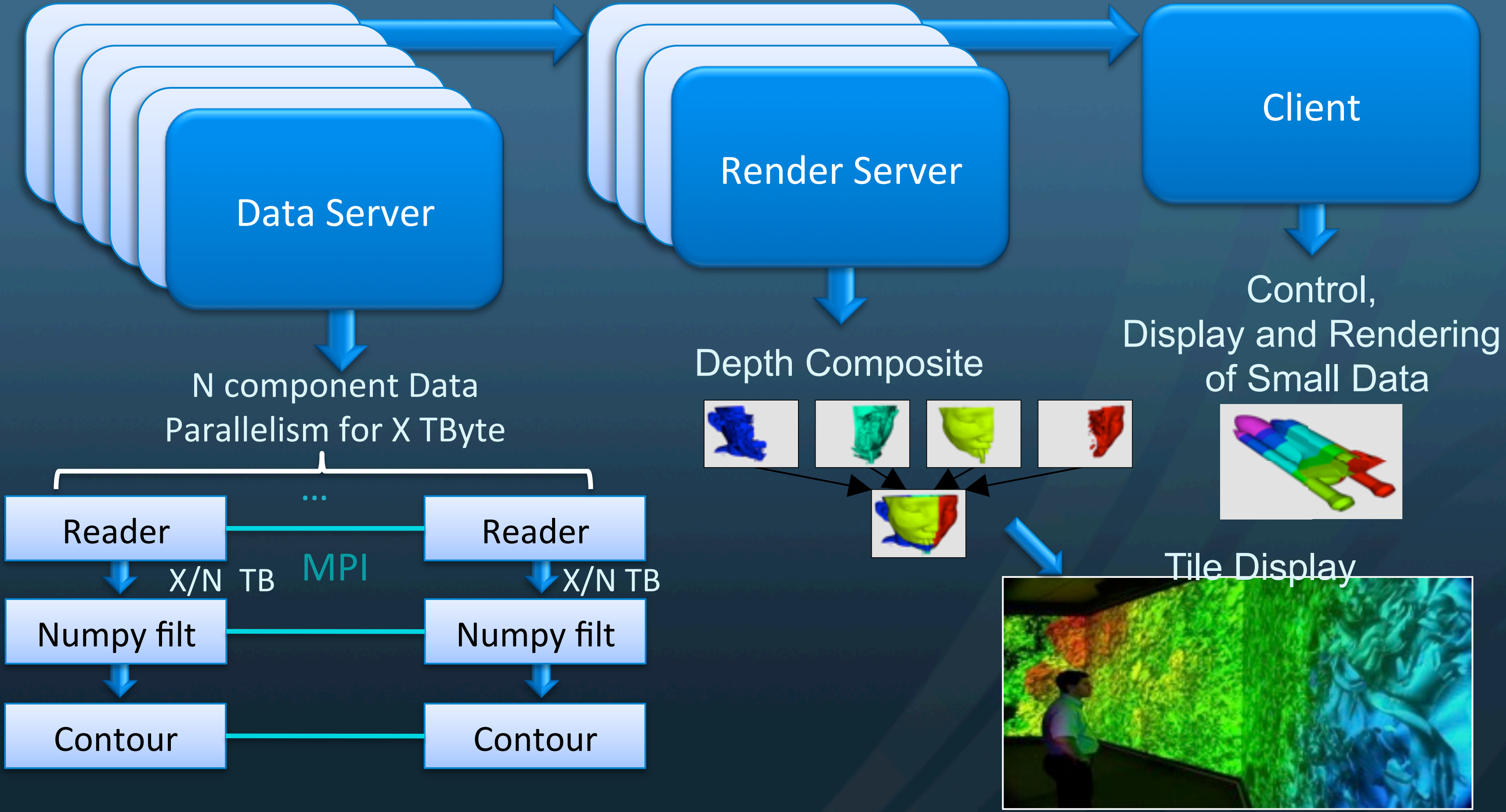


Large displays and virtual reality



The ParaView Tutorial, Monday at SC16

Web visualization



Data Server

Render Server

Client

N component Data
Parallelism for X TByte

Reader ... Reader

MPI

X/N TB

X/N TB

Numpy filt

Numpy filt

Contour

Contour

Depth Composite

Control,
Display and Rendering
of Small Data

Tile Display

Rendering on Supercomputers

2 Xeon E5-2699v3 @2.3GHz - 72 ht“cores”
GeForce GTX 750 Ti (~2014 model)
60 GB RAM

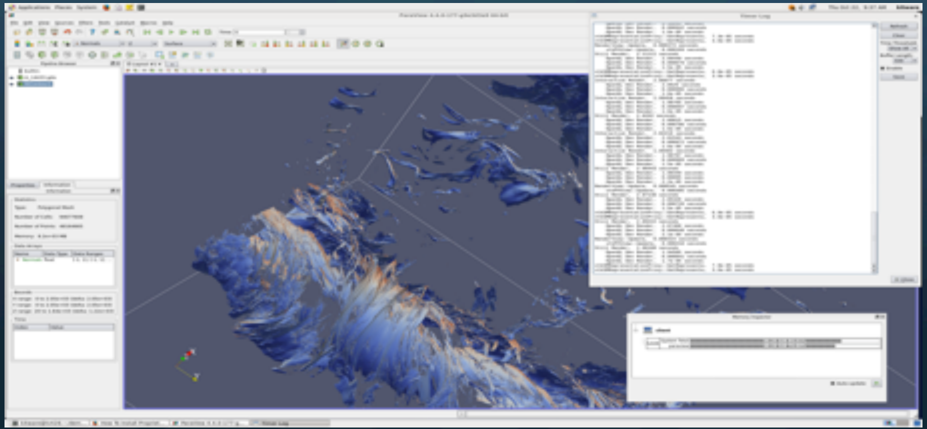
	GL1	GL2
SWR	.9 sec/.07 sec	.46 sec/.05 sec
GPU	2.6 sec/.1 sec	.25 sec/.02 sec
OSP	1.8 sec/.04 sec	1.7 sec/.04

Data too large to transfer

GPU: X11 or better EGL

Phi and CPU: OSMesa or better SWR

GPU avail and mem:	CPU [GB]	GPU [GB]
titan@ornl	32	6
rhea@ornl	128	0
maverick@tacc	256	12
stampede@tacc	32	8 (Phi)
cooley@anl	384	24
mira@anl	16	0



Magnetic reconnection data thanks to Bill Daughton
2k^3 float, 95mil cell (~8GB) iso

OpenSWR in ParaView

- SWR: A higher performance CPU only backend for Mesa GL
- Regression tested nightly on ParaView dashboard
- Available at TACC since 4.3
- Available in ParaView linux binaries since 5.0.0

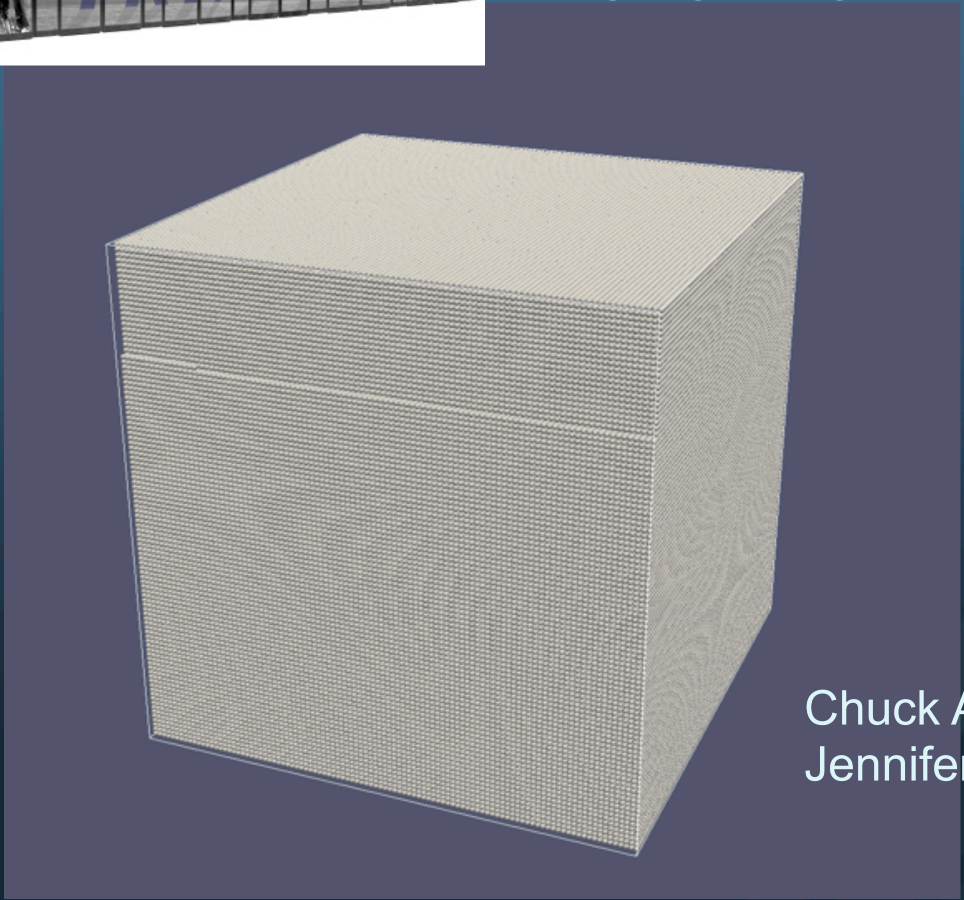
  https://blog.kitware.com/messing-with-mesa-for-paraview-5-0vtk-7-0/

```
# To use Mesa+llvmpipe
./paraview --mesa-llvm

# To use Mesa+openswr-avx
./paraview --mesa-swr-avx

# To use Mesa+openswr-avx2
./paraview --mesa-swr-avx2
```

Benchmark - to 1.1 Trillion Tris

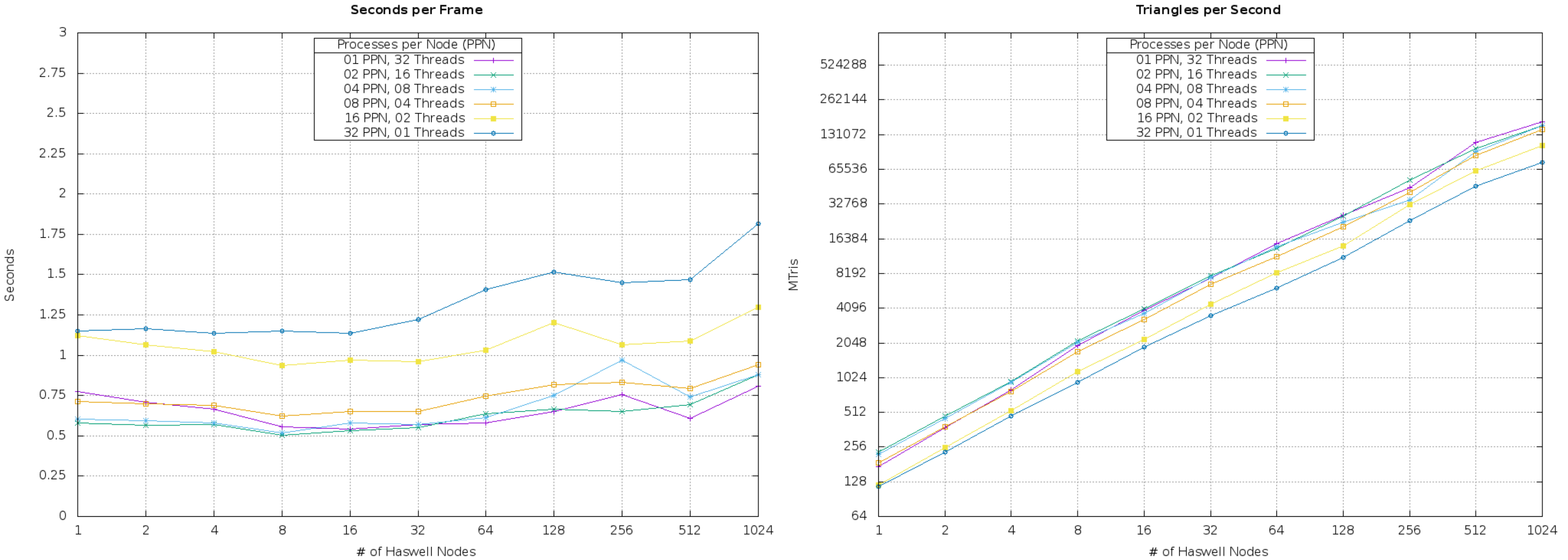


Chuck Atkins, Dave DeMarle @kitware
Jennifer Green @ lanl

unclassified LA-UR-16-23941

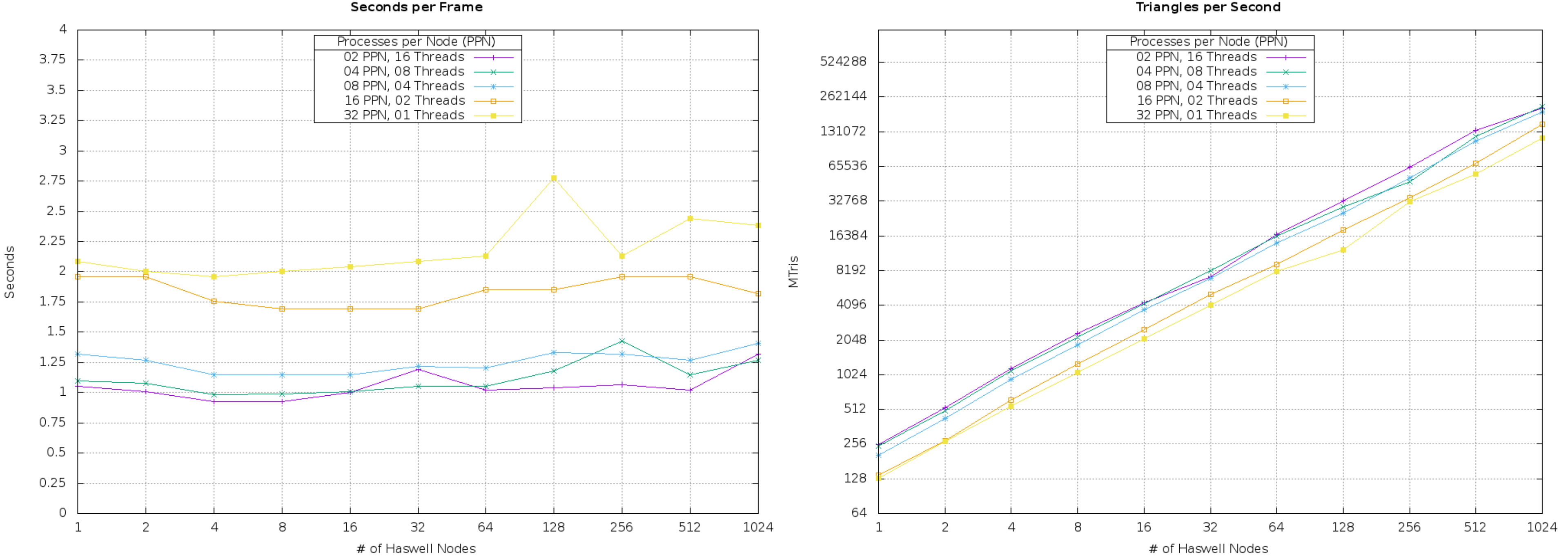
128 Million Tris per node

ParaView Benchmark on Trinity: 128 Spheres (128MTri) per Node

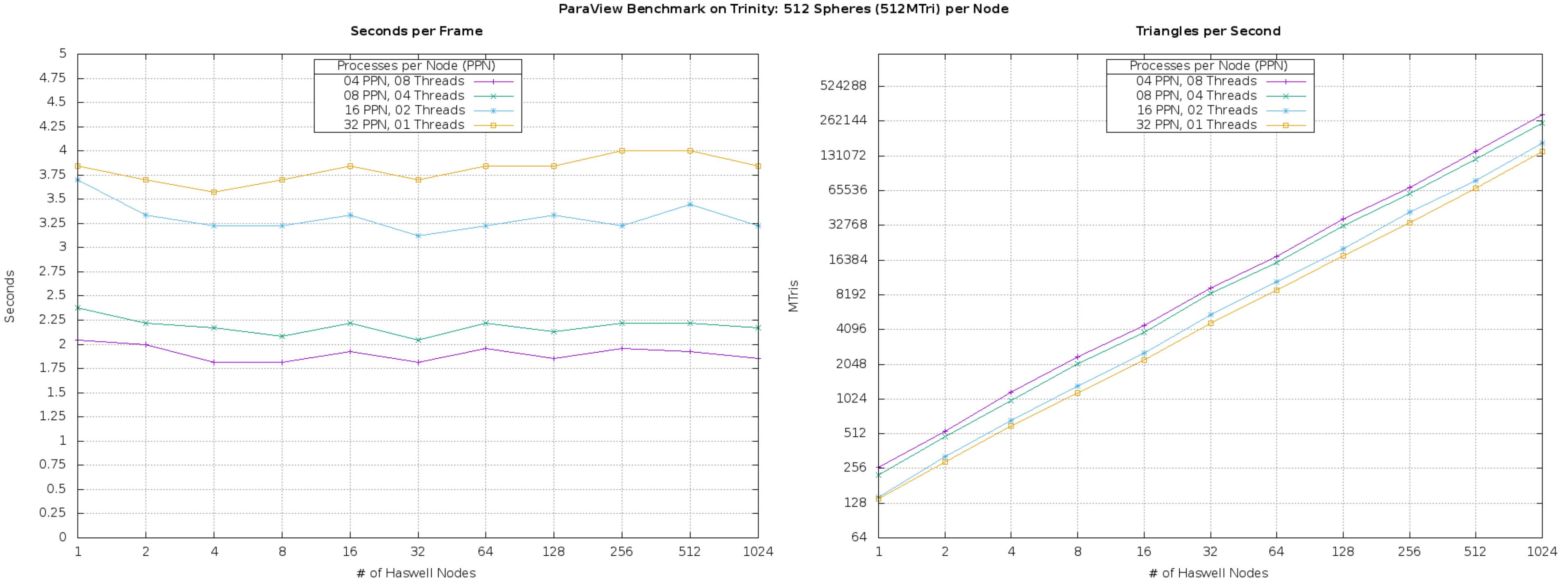


256 Million Tris per node

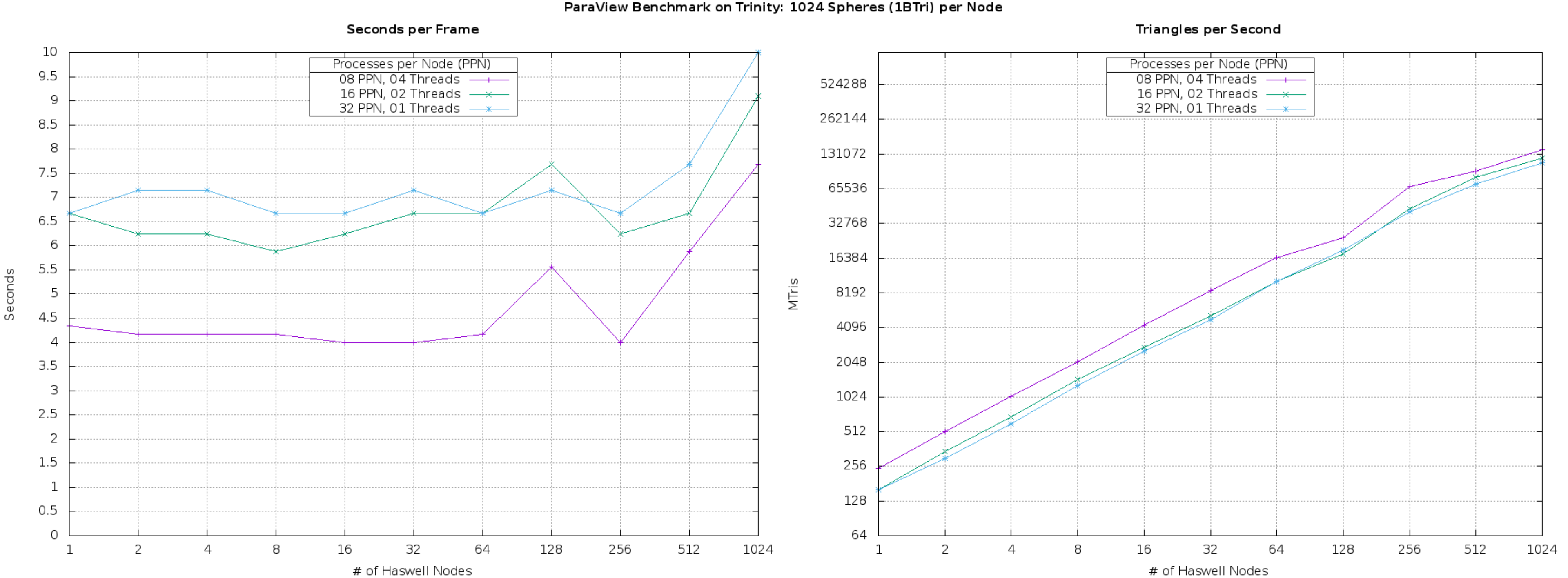
ParaView Benchmark on Trinity: 256 Spheres (256MTri) per Node



512 Million Tris per node



1 Billion Tris per node



Note: Only 1/19'th machine. Expect 10-20 trillion tris and about 1 minute per frame at pre KNL max.

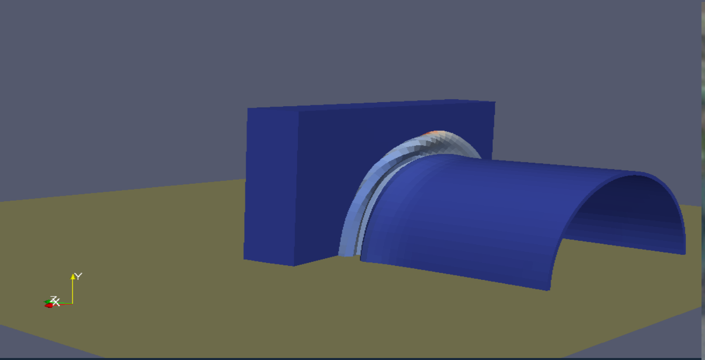
LA-UR-16-23941

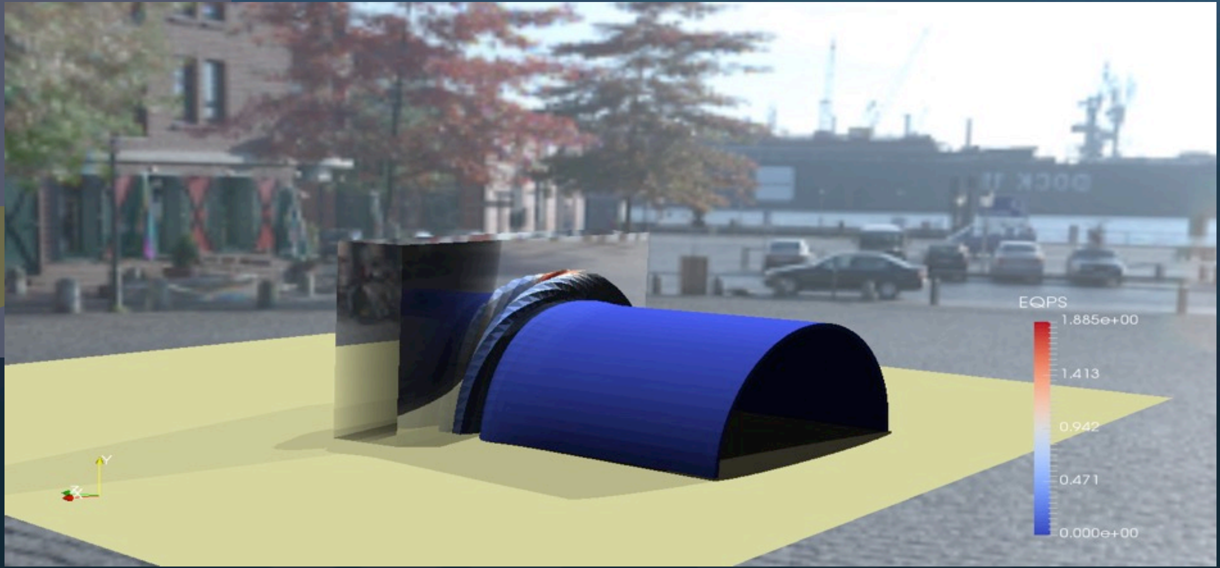
But most* of our images still look like they were made in 1985.

*Many notable exceptions, e.g. those shown throughout SC floor. Takes good data, expertise & time.

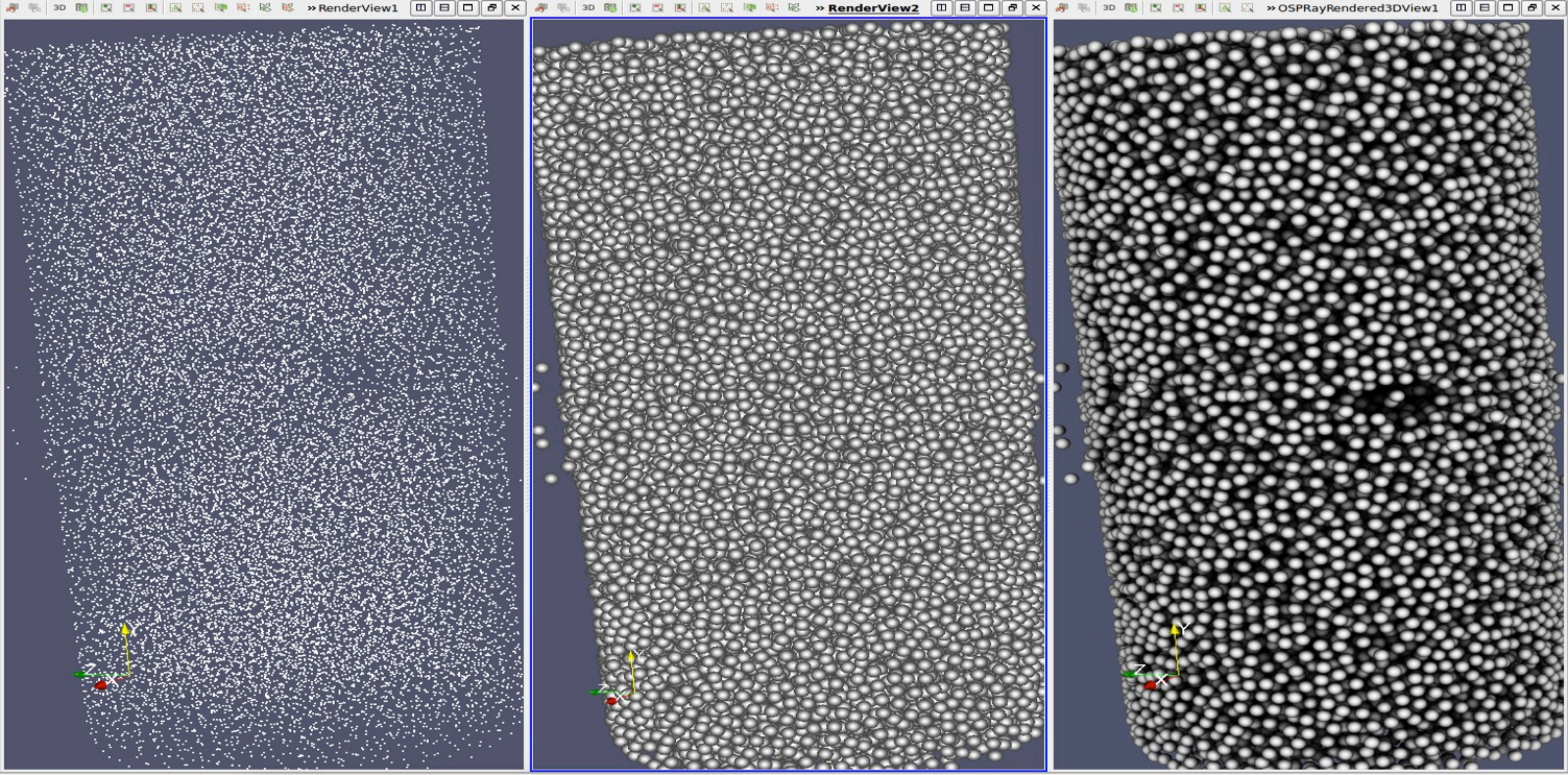
Ray tracing is an answer

- Transparency and Hard Shadows easy enough (today) with rasterization - depth peeling and shadow map passes
- Accurate translucency and reflection add complexity. Ray tracing makes it feasible to mix into a big complicated system like ParaView.





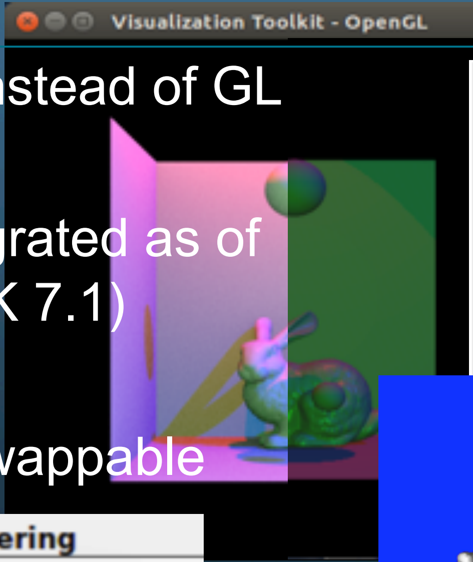
can.ex2 via GL (above)
and Manta plugin (right)

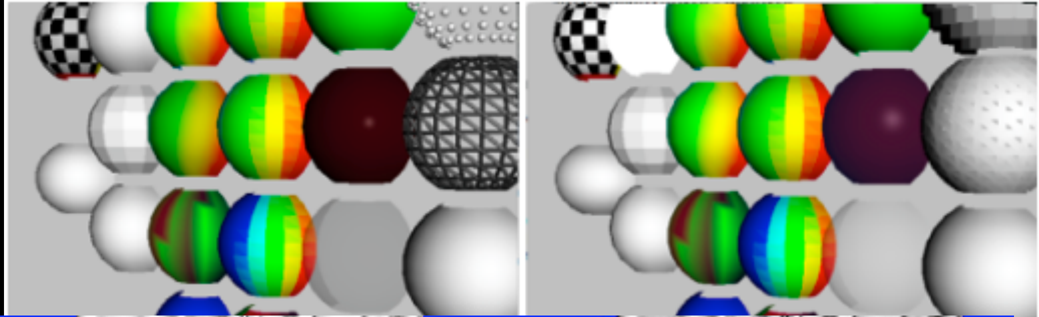


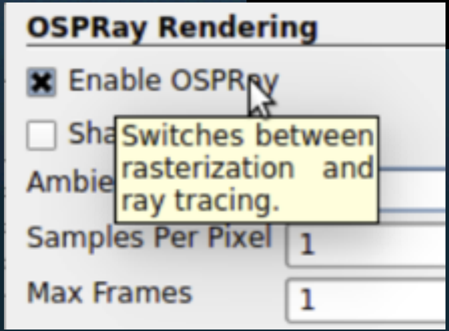
GL points (L) and sprites (C) lack the meso-scale clues that pOSPRay’s (R) ambient occlusion provides. Crack propagation data thanks Souchin Deng @ INL

OSPRay in VTK and ParaView

- Ray trace instead of GL
- Tightly integrated as of PV 5.1 (VTK 7.1)
- Run time swappable







OSPRay Rendering

☒ Enable OSPRay

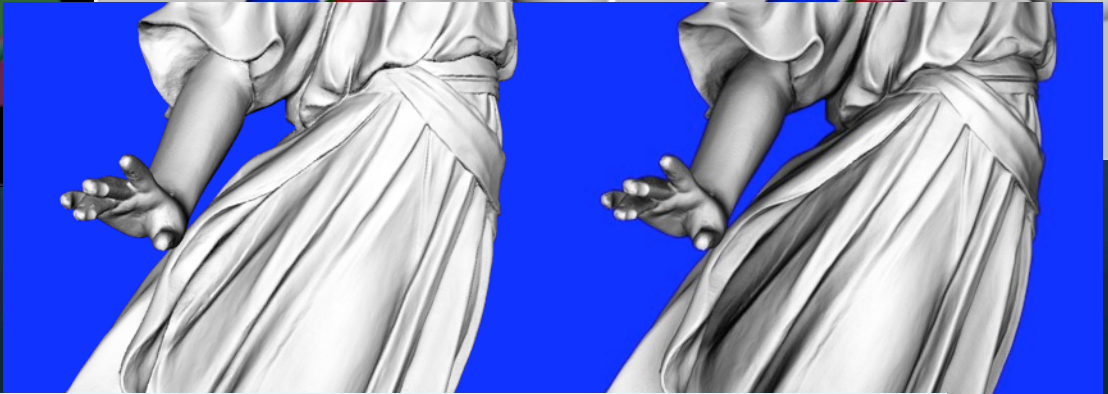
□ Sha

Switches between rasterization and ray tracing.

Ambie

Samples Per Pixel 1

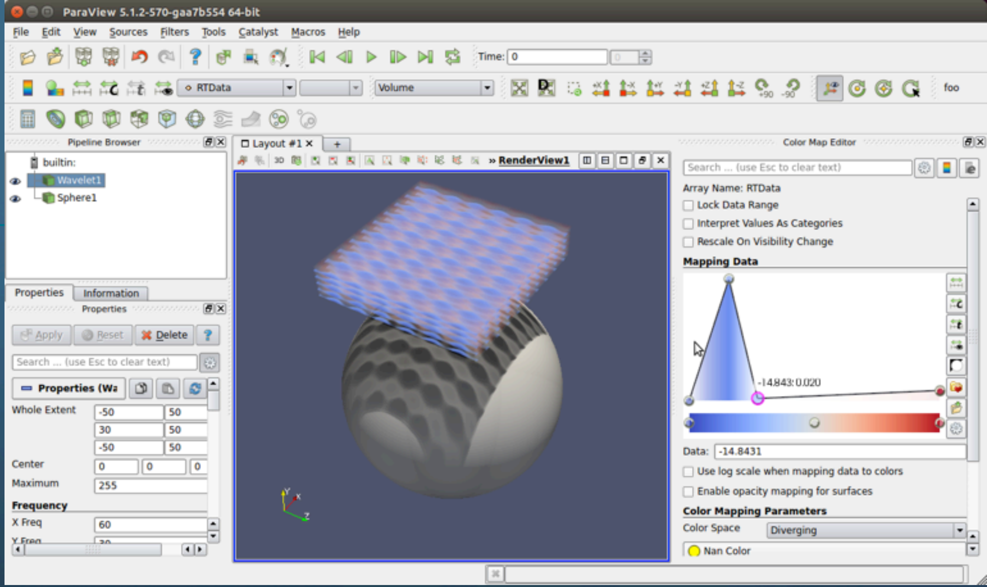
Max Frames 1



rasterization (left), ospray (right)
Simply hit ‘c’ to switch back and forth.

Potential Benefits

- Aesthetics (but only in SMP)
  - Ambient Occlusion
  - Shadows
  - 😢 No reflections/refractions yet
- Ray Space Transformations
  - Implicit Isosurfaces (soon)
  - Implicit Spheres/Cylinders



Pipeline Browser: builtin: / Wavelet1 / Sphere1

Properties (Wavelet1)

Whole Extent	-50	50
	30	50
	-50	50
Center	0	0 0
Maximum	255	
X Freq	60	

Array Name: RTData

□ Lock Data Range

□ Interpret Values As Categories

□ Rescale On Visibility Change

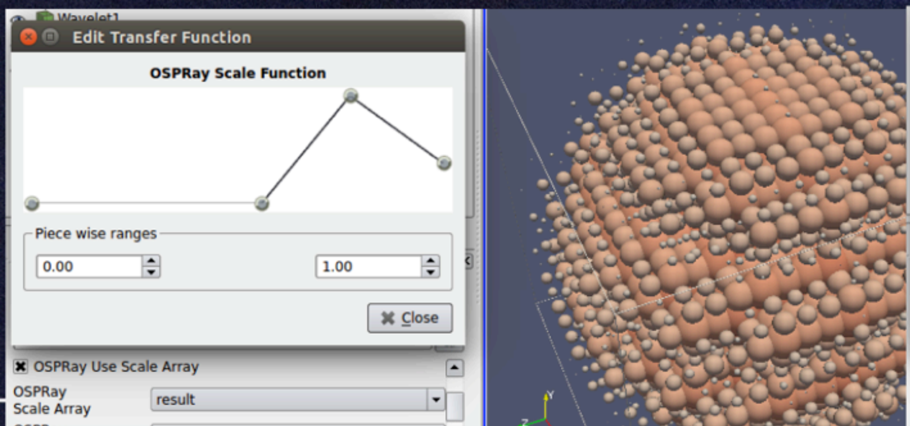
Data: -14.8431

□ Use log scale when mapping data to colors

□ Enable opacity mapping for surfaces

Color Space: Diverging

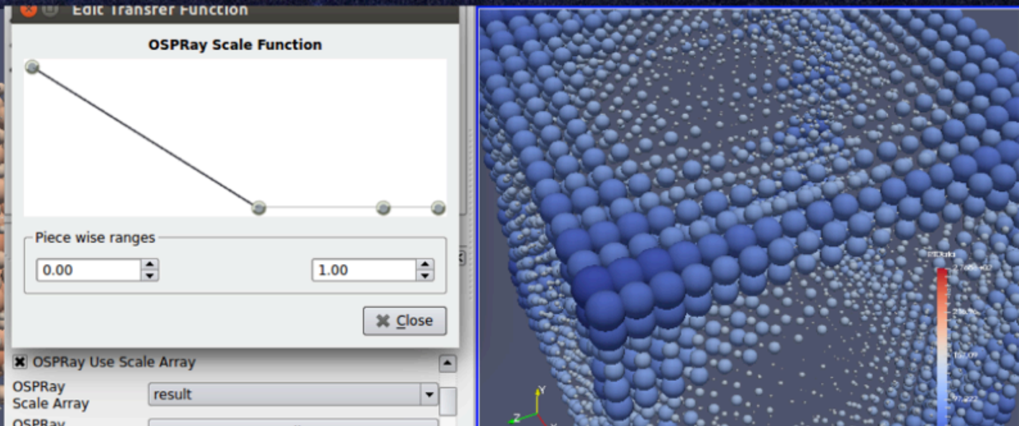
Nan Color



Piece wise ranges: 0.00 – 1.00

☒ OSPRay Use Scale Array

OSPRay Scale Array: result



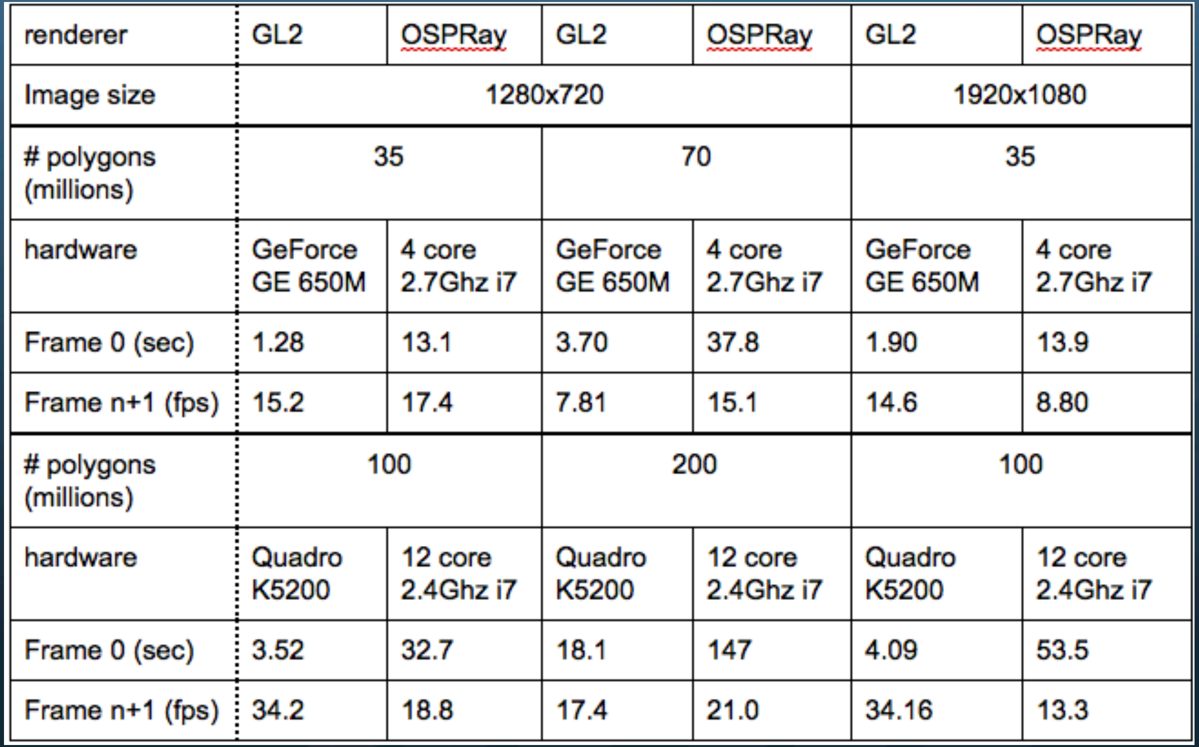
Piece wise ranges: 0.00 – 1.00

☒ OSPRay Use Scale Array

OSPRay Scale Array: result

Fast CPU Rendering

- Especially when #triangles dominate #pixels
- first frame is tolerable
- subsequent frames scream
- Ideal for Cinema use case



renderer	GL2	OSPRay	GL2	OSPRay	GL2	OSPRay
Image size	1280x720				1920x1080	
# polygons (millions)	35		70		35	
hardware	GeForce GE 650M	4 core 2.7Ghz i7	GeForce GE 650M	4 core 2.7Ghz i7	GeForce GE 650M	4 core 2.7Ghz i7
Frame 0 (sec)	1.28	13.1	3.70	37.8	1.90	13.9
Frame n+1 (fps)	15.2	17.4	7.81	15.1	14.6	8.80
# polygons (millions)	100		200		100	
hardware	Quadro K5200	12 core 2.4Ghz i7	Quadro K5200	12 core 2.4Ghz i7	Quadro K5200	12 core 2.4Ghz i7
Frame 0 (sec)	3.52	32.7	18.1	147	4.09	53.5
Frame n+1 (fps)	34.2	18.8	17.4	21.0	34.16	13.3

KNL Rendering first results

1 KNL node (256 ht cores, 1.6GHz), 94GB

all [frame/sec]

	llvm	swr-avx2	OSPRay	llvm	swr-avx2	OSPRay
mtris	720p = 1280x720			1080p = 1920x1080		
1	.84	9.57	14.96	0.76	6.24	8.19
10	.12	4.92	15.25	0.11	3.80	8.07
20	0.06	2.84	15.04	0.06	2.10	7.96
40		1.75	14.76		1.39	8.12
80		1.00	14.95		0.81	7.87
160		0.54	14.80		0.46	7.77
320		0.39	14.58		0.36	7.69

KNL Rendering first results

1 KNL node (256 ht cores, 1.6GHz), 94GB

all [frame/sec]

	llvm	swr-avx2	OSPRay	llvm	swr-avx2	OSPRay
mtris	720p = 1280x720			1080p = 1920x1080		
1	.84	9.57	14.96	0.76	6.24	8.19
10	.12	4.92	15.25	0.11	3.80	8.07
20	0.06	2.84	15.04	0.06	2.10	7.96
40		1.75	14.76		1.39	8.12
80		1.00	14.95		0.81	7.87
160		0.54	14.80		0.46	7.77
320		0.39	14.58		0.36	7.69

f0 = 32sec

KNL Rendering first results

1 KNL node (256 ht cores, 1.6GHz), 94GB

all [frame/sec]

	llvm	swr-avx2	OSPRay	llvm	swr-avx2	OSPRay
mtris	720p = 1280x720			1080p = 1920x1080		
1	.84	9.57	14.96	0.76	6.24	8.19
10	.12	4.92	15.25	0.11	3.80	8.07
20	0.06	2.84	15.04	0.06	2.10	7.96
40		1.75	14.76		1.39	8.12
80		1.00	14.95		0.81	7.87
160		0.54	14.80		0.46	7.77
320		0.39	14.58		0.36	7.69

f0 = 32sec

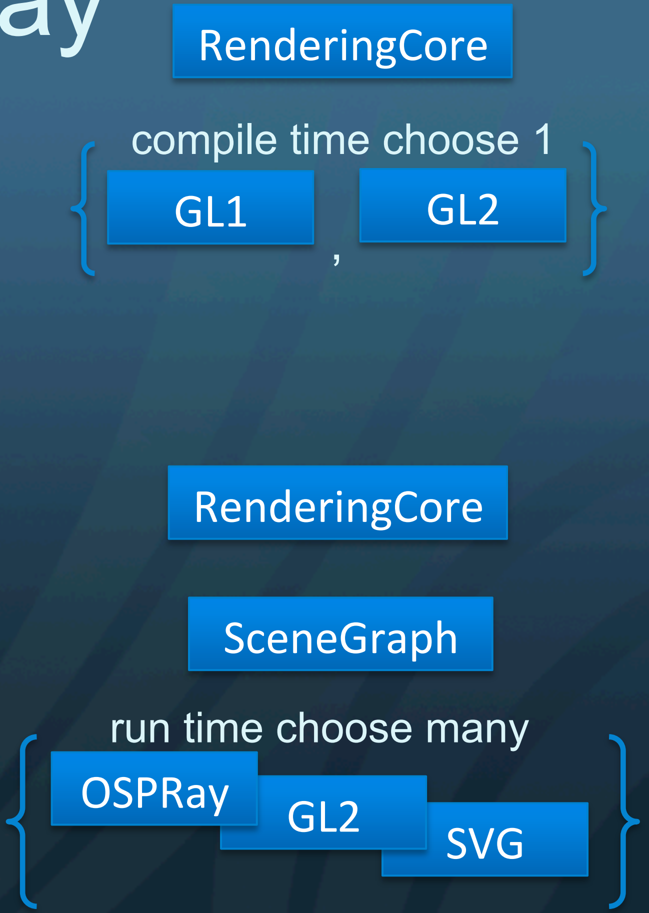
f0 = 71sec

VTK/Rendering/OSPRay

- New approach
  - separate render state from implementation
  - RenderingSceneGraph - render state
  - RenderingOSPRay - OSPRay rendering implementation

- Part of VTK

```
cmake -DvtkModuleRenderingOspray:BOOL=ON
FindPackage(OSPRay)
```



How to get it in your VTK app?

Use VTK 7.1
Enable Module
C++11
Point CMake to OSPRay lib

vtkRenderer->RenderPass
  mechanics of drawing.
  vtkOSPRayPass
    sends SceneGraph to OSPRay

Add to renderer and voila!

```
#include "vtkOSPRayPass.h"
...
vtkOSPRayPass* osprayPass = vtkOSPRayPass::New();
...
if (useOSPRay)
{
  renderer->SetPass(osprayPass);
}
else
{
  renderer->SetPass(NULL);
}
```

Ray traced visualization ready?

Yes!

Sort time drastically improved.

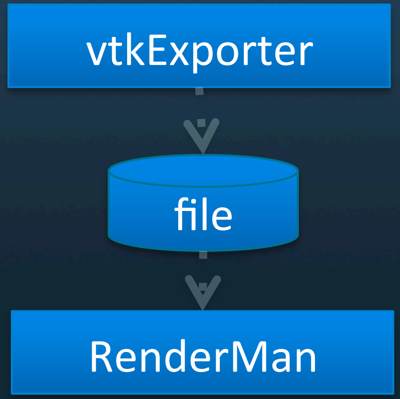
~45 min for 40 mil cells Manta

9.3 sec OSPRay

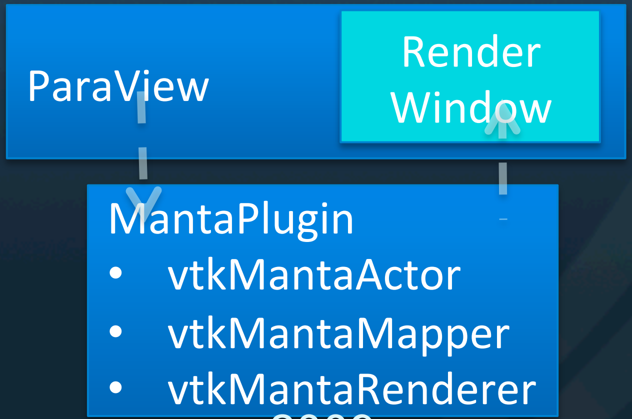
No! Someone please solve Distributed Memory 2ndary rays

Ray traced rendering in VTK

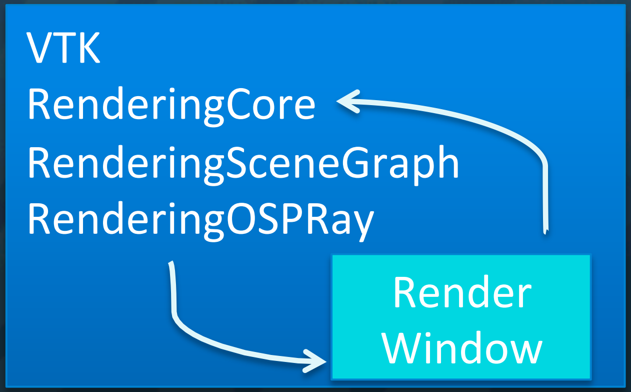
~1995	vtkVolumeRayCastMapper
1996	vtkRIBExporter (RenderMan)
~2003	vtkGPUVolumeRayCastMapper
2009	Manta ParaView plugin
2014	OSPRay ParaView plugin
2016	OSPRay VTK module



1996



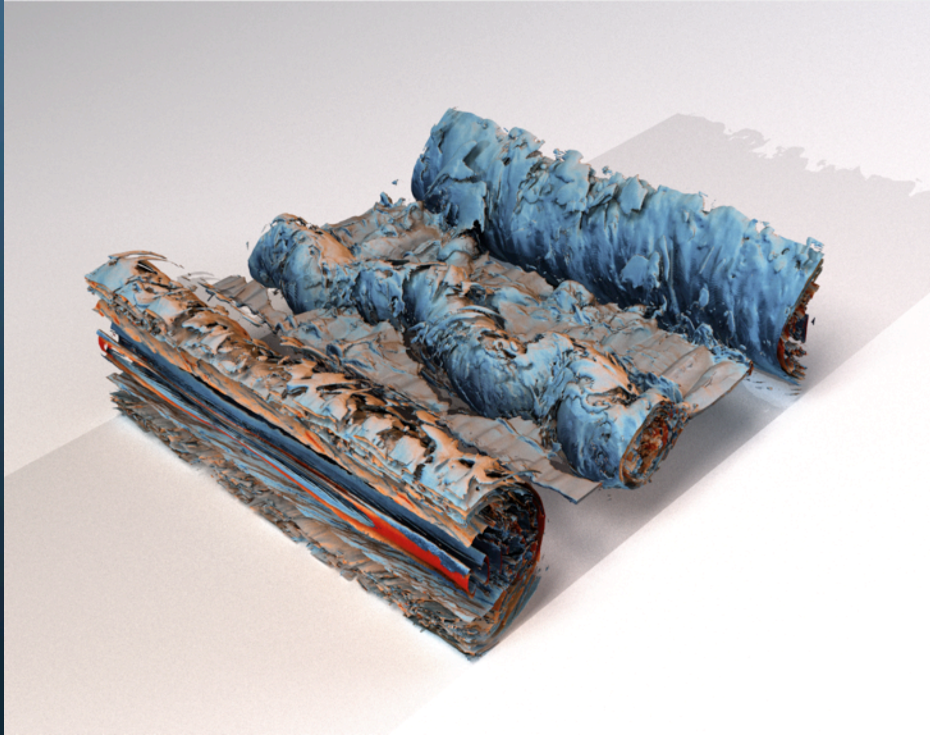
2009



2016

What’s coming up next?

- PathTracer
  - enable it (done)
  - (re)Enable Refinement (close)
  - Extend VTK lights
  - Extend VTK materials
  - Test and Prove out



What’s coming up next?

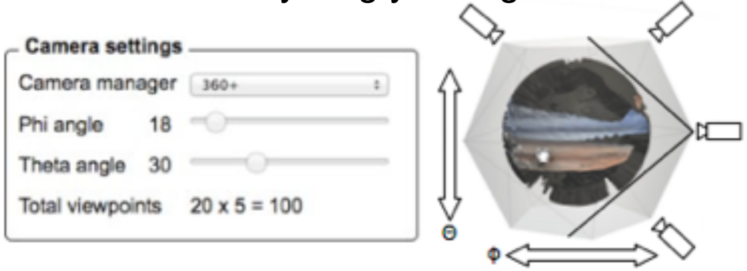
In Situ - Catalyst and Cinema

Data too large to save at every timestep

- In-Situ - render data at simulation time - images are tiny
- Keep data where produced
- Render as efficiently as possible

```
for all times:
  for all objects:
    for all options:
      for all arrays:
         for all camera_positions:
           render_into_database()
```

Cinema - render everything you might want to see



Camera settings

Camera manager: 360+

Phi angle: 18

Theta angle: 30

Total viewpoints: 20 x 5 = 100

- Sometime later, scientist browses and searches in a viewer

What's coming up next?

- Rendering:
  - is pretty close to done
- Interaction:
  - Will need more attention to widgets and interaction mechanisms
  - More VTK Applications besides just ParaView (and VisIt)
- Processing:
  - New opportunities for using within or instead of filters

Implicit Isosurfaces, Collision detection, Percent occlusion, …

Conclusion

- SWR and OSPRay incorporated into and enhance VTK
- Very useable in PV 5.2/VTK 7.1, will continue to refine

- Particularly beneficial for large simulation runs (ParaView/ VisIt use cases)

- New rendering algorithm (Ray tracing via OSPRay) for VTK opens up new possibilities