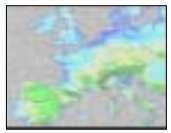
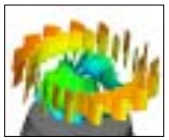
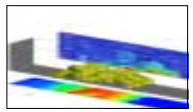
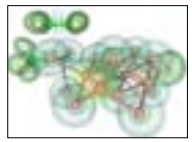
CSCS
Swiss National Supercomputing Centre





Utilisation de VTK/ParaView pour la visualization de gros volumes de donnees





Jean M. Favre
Responsable du groupe de visualisation

Outline

Some strategies to deal with large data

How do VTK and ParaView fit in our portfolio?

VTK and ParaView's best features for Large Data
- Data Streaming, data parallelism
- ParaView's Compositor
- VTK customization

Conclusion

History of visualization softwares at CSCS

In the last 10 years,
- IRIS Explorer
- AVS5, AVS/Express
- EnSight
- Amira
- CFXPost (ANSYS)
- In-house packages
- *COVISE*
- VTK
- ParaView

Examples of VTK/ParaView use

AMR patch-based support
- Parallel reader, contouring, cutting

Navier-Stokes Multi-Block solver
- Parallel reader
- Surface extractor

Astronomy, SPH, large molecules
- Particle rendering without polygonal representation

Some strategies to deal with large data

Data access

- on-demand
- streamed
- client-server
- parallel

Interaction techniques

- Efficient
- Progressive
- Interruptible

Rendering

- new paradigms (e.g. Vector field visualization)
- Open-GL renaissance
- multi-resolution
- distributed with compositing
- Levels of Details
- Distributed control, interrupts

VTK and ParaView

- www.vtk.org and www.paraview.org
- Open source
  - VTK is a C++ toolkit, modular components assembled in “pipelines”
  - ParaView is an end-user application integrating many VTK features
  - ParaView hides the complexity of pipeline editing
- Parallelism
- Client-server
- EnSight format readers
- And a lot more…
- See the books “Users Guide”

VTK/ParaView strengths

Very rich feature set (driven by graphics & viz conference publications)

3D direct-manipulations widgets

Level of Details

Composite datasets are first-class citizens

Client-server

- Pixel sub-sampling over slow connection

MPI-based parallelism

- Ghost-cells
- Load-balancing
- Piece invariance

Tiled-display

Missing features can be added by creating C++ classes

VTK contributions available in ParaView

in collaboration with CEA-DAM:

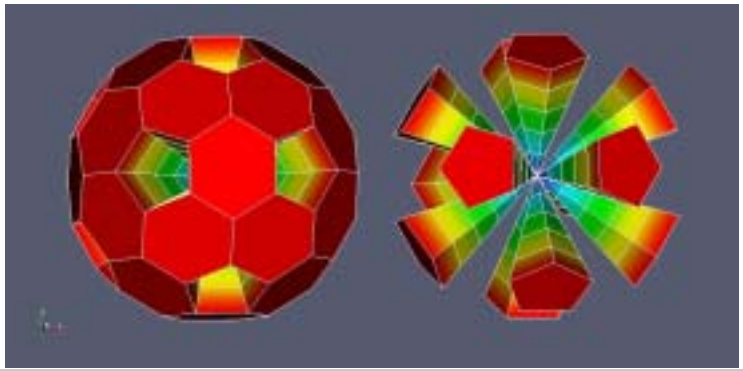
AVS UCD format reader

- Binary/ASCII
- Little-endian, big-endian aware
- Allow pre-selection of cell-data and node-data variable before reading
- Make variable’s scalar range available without reading
- Time-dependent support via multiple files only

VTK contributions available in ParaView

in collaboration with CEA-DAM:

PentagonalPrism and HexagonalPrism

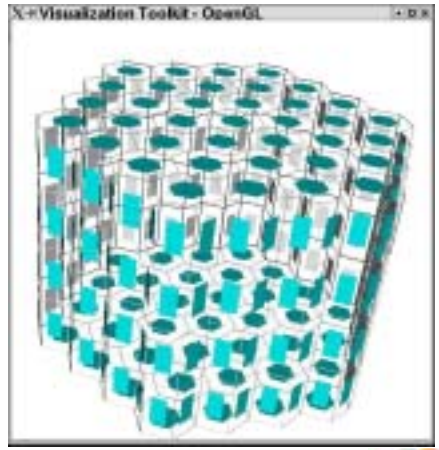


Data by CEA-DAM

PentagonalPrism and HexagonalPrism

Adding a new cell type encompasses many tasks:

- Iso-surfacing
- Slicing, cutting
- External surface

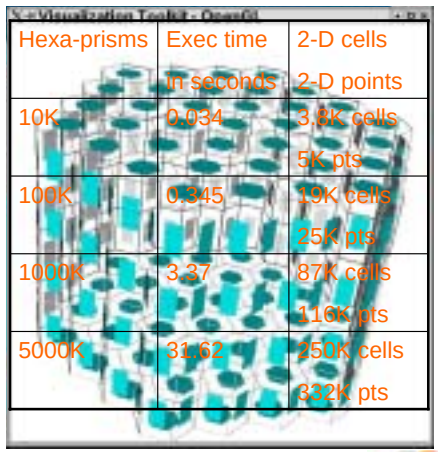


PentagonalPrism and HexagonalPrism

Adding a new cell type encompasses many tasks:

- Iso-surfacing
- Slicing, cutting
- External surface =>

- Timings on a 1.8Ghz Xeon with 2Gb RAM
- 10M cells crashed the application



Hexa-prisms	Exec time in seconds	2-D cells 2-D points
10K	0.034	3.3K cells 5K pts
100K	0.345	19K cells 25K pts
1000K	3.37	87K cells 116K pts
5000K	31.62	250K cells 332K pts

Data access

EnSight was the first to popularize *on-demand* data loading

=> ParaView lets us create GUI widgets to select variables, volumes, time steps to load

EnSight also had a robust client-server model

=> ParaView emulates it with more efficient options

VTK offers *piece-wise* visualization or *streaming* of data

ParaView render server

Pixel-subsampling

RLE and bit-compression

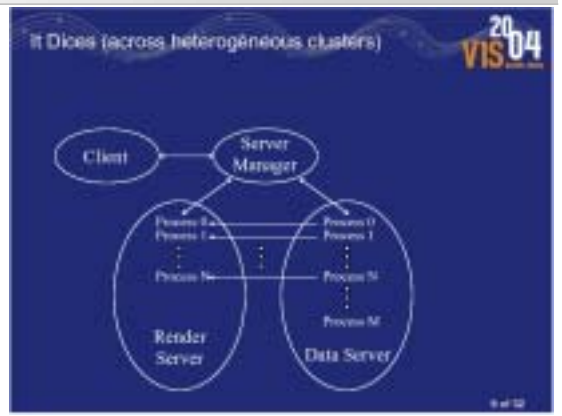
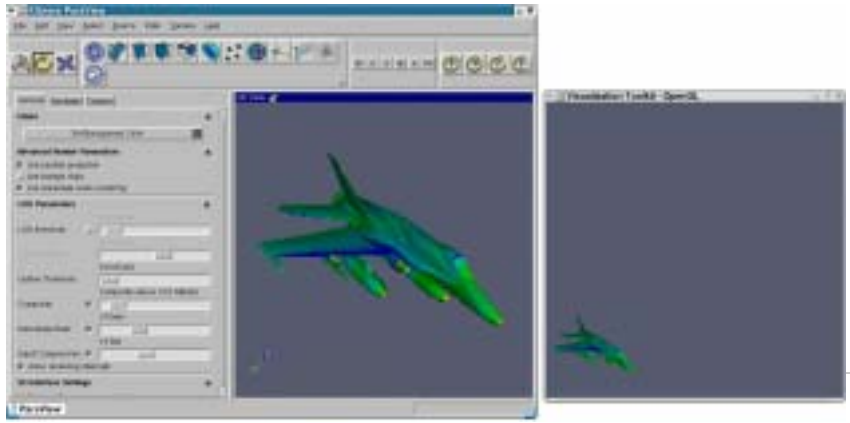


image borrowed from Ken Martin @ Kitware

Client-server bandwith efficient

Pixel-subsampling and RLE and bit-compression are used



Data Streaming: Data Flow or Event Flow?

AVS/Express is a data-flow environment. Any change in the data triggers a downstream chain of updates

VTK is an event-flow environment. The renderer drives the request for data updates.

Data Sources

↕

Data Filters

↕

Data Mappers

↕

Graphics Rendering

Event-driven Data Streaming

Data larger than memory is treated

- Piece by piece
- Releasing memory after each subset
- Optionally accumulating sub-object representations for the final image

Options are:

- Set Number Of Pieces
- Set Memory Limit (Kbytes)

Data Sources

↕

Data Filters

↕

Data Mappers

↕

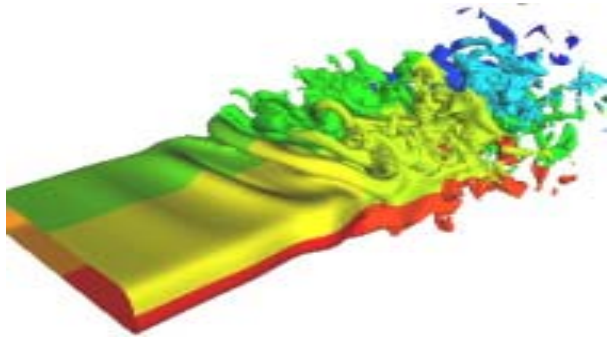
Graphics Rendering

3D Data Streaming example

Loop over all blocks

AND

Accumulate the results



Data Sources inherit a tough responsability

Support the two streaming options

- Set Number Of Pieces
- Set Memory Limit (Kbytes)

Read data by block, by part, by cells

Provide ghost-cells

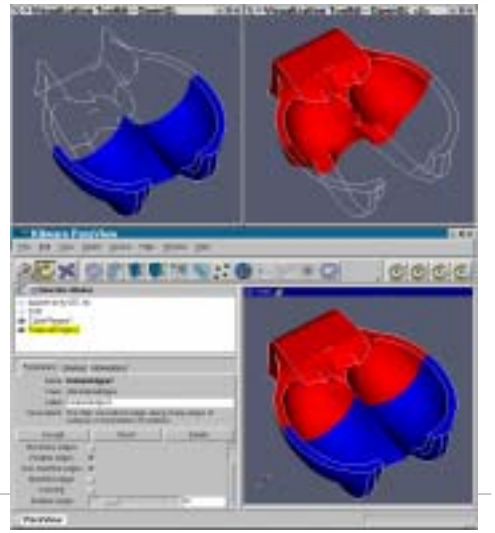
Data Sources
↕
Data Filters
↕
Data Mappers
↕
Graphics Rendering

Piece-invariance for distributed-data

Why is it important?

Features extraction filters which use more than the single point or cell to derive their information need neighboring regions.

If the data is divided in different ways among processors, the results need to be independent of the subdivision scheme.



Piece-wise data handling leads to parallelism

It is equivalent to treat individual pieces

- one at a time or,
- all at once to multiple handlers

⇒ Parallelism is supported

⇒ The application can be migrated

It will help if the data archiving supports

- Efficient piece-wise retrieval
- distributed storage hardware

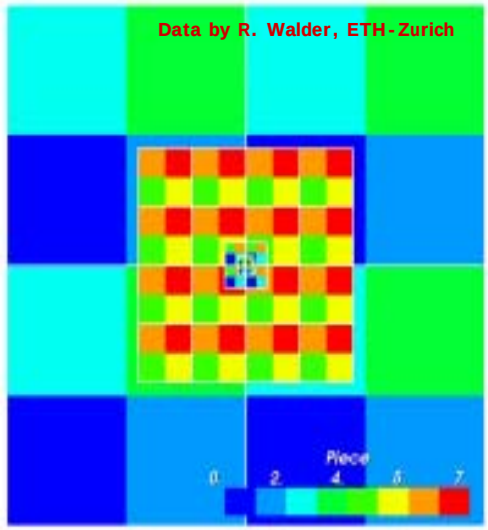
image borrowed from Ken Martin @ Kitware

AMR patch-based

Our reader is a subclass of vtkHierarchicalDataSetAlgorithm

- Set number of levels
- For each grid of each level, set the extents of the overlap box
- For each level, all grids are divided among all processes

In collaboration with Berk Geveci @ Kitware

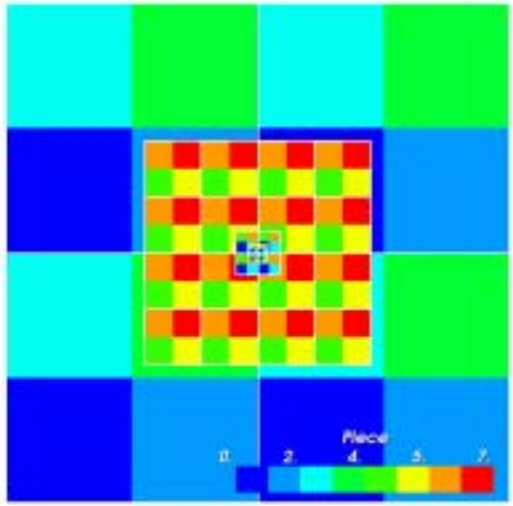


AMR patch-based

Based on Cell-blanking and visibility arrays

The data hierarchy is maintained throughout the operation

Thus, iso-contours lines from a hierarchical dataset are also stored as a hierarchical dataset

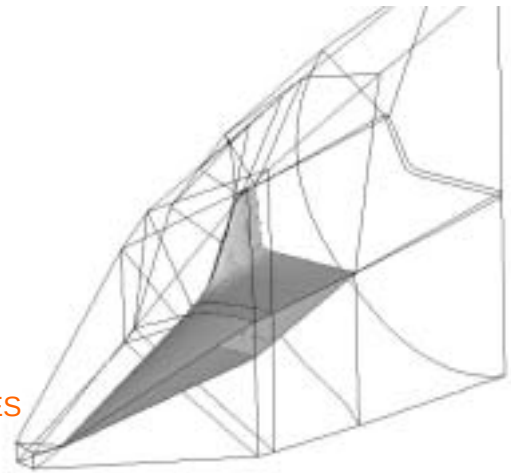


Navier Stokes Multi-Block Handler

Multi-block is now a vtkCompositeData of type vtkHierarchicalBoxDataSet with numberOfLevels = 1

RequestUpdateExtents() divides the load between processors based on the

- UPDATE_PIECE_NUMBER
- UPDATE_NUMBER_OF_PIECES



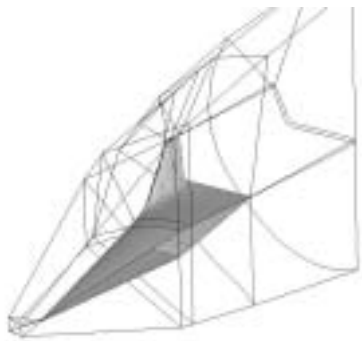
Data by P. Leyland, EPF-Lausanne

NSMB solid surface extractor

The solid surfaces are I/J/K planes in the curvilinear grids

Each vtkDataSet encodes its IJK ranges in a FieldData

The solid-surface extractor uses a CompositeDataIterator to walk through the structure and append the geometries together.



NSMB reader/cutter/contour & viewer

MPI-based runs

•mpirun –np 16 pvserver

•pvclient --server-host="host"

CPUs =>	1	2	4	8	16
Read	4.75	4.66	4.67	4.66	3.55
Solid Surface	9.81	5.10	3.52	2.61	2.04
Cutting planes	33.8	32.53	51.56	58.4	58.47
Iso-mach lines	1.20	1.20	2.33	2.86	3.02

ParaView render server

Full-parallelism involves doing also parallel rendering

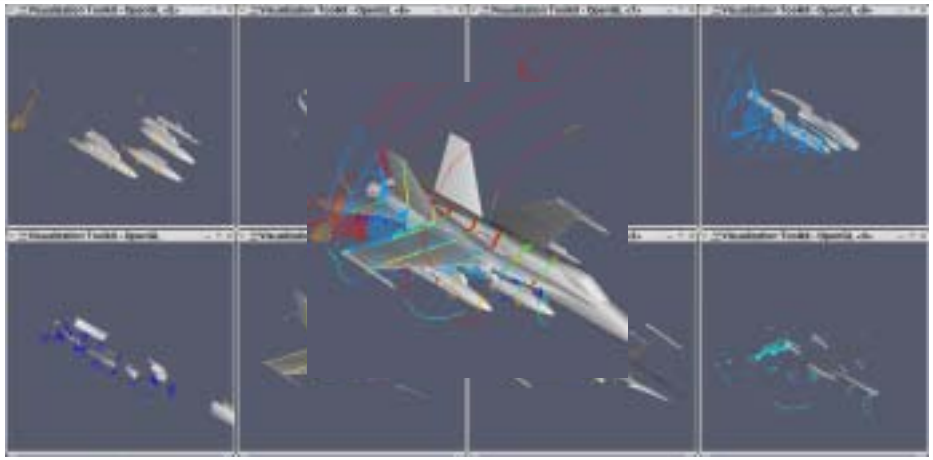
Use fast and cheap graphics cards

Sub-division can be

- Screen-space (tiled display)
- Object-space (sort-last)

image borrowed from Ken Martin @ Kitware

ParaView's compositor [sort-last]



Server-side rendering or client-rendering

All data processing happens on the server (including polygonal data generation)

If the geometry is small, or the client has high-performance rendering, let the client render data

ParaView has a user threshold to make that decision

1. The server can render geometry and send images back to the client
2. If the data is very large
   - It probably will not fit on the client
   - It is probably partitioned on the server

Send geometry or images to the client?

Example of time-dependent data:

- Threshold to deliver geometry to the client = 10 MB
- 400 time steps, 10 MB each = 4 GB
- 4GB on a 100T line = 6 minutes for data rendered once and then thrown away

- Better deliver the frame buffer contents after compositing

Example by B. Wylie (SNL)

ParaView image compositing

ParaView uses a binary-tree compositing of colors and Z-buffers

Image transfer uses RLE and bit-compression

Could be replaced by a hardware compositing platform

- We are now evaluating an HP platform which couples NVIDIA cards + Sepia cards doing the compositing in a hardware chain

VTK 4.x and VTK 5

Old technique

Readers produce many outputs

Dedicated pipelines are created and run applying filter to each output separetely.

Problems at boundaries

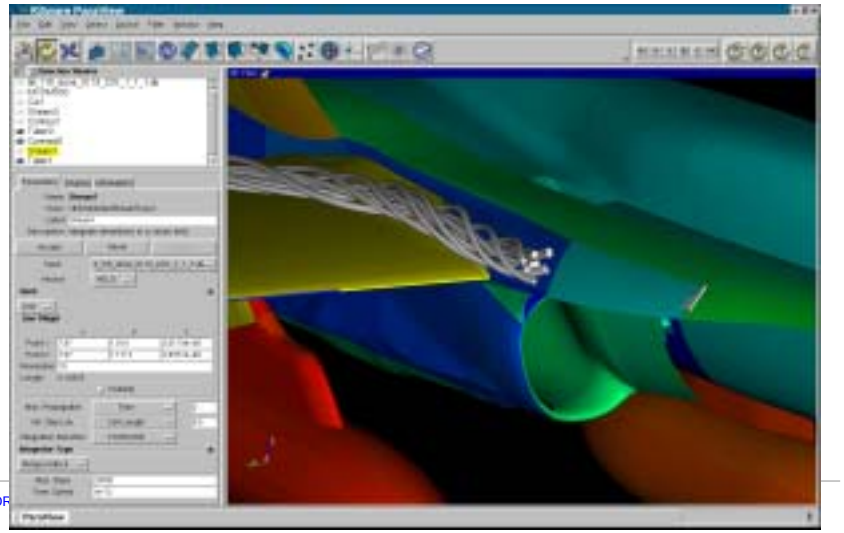
New composite support

Readers produce a single “composite” object

And the filter iterates thru the structure

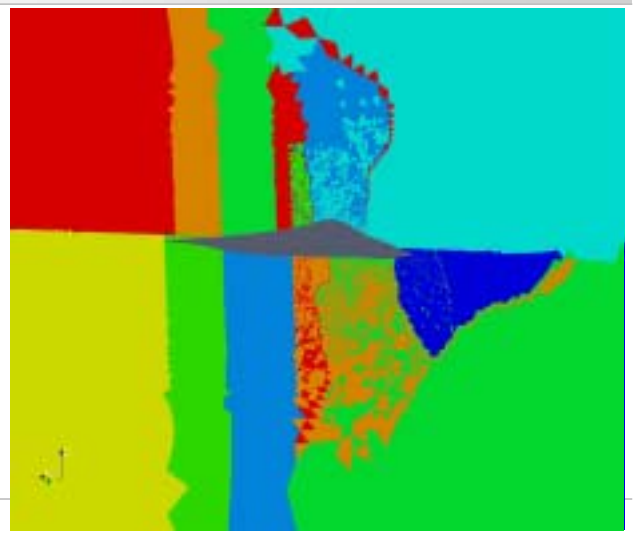
F18 data by RUAG Aerospace

A few remaining problems with distributed processing



AllToNRedistributePolyData

Before	After
83504 cells	39872 cells
29857	idem
63046	idem
22914	idem
34781	idem
39670	idem
16180	idem
29024	Idem
Total=318976	



New graphical representations

Current paradigms can be augmented:

- Vector field visualization
- Point-based visualization

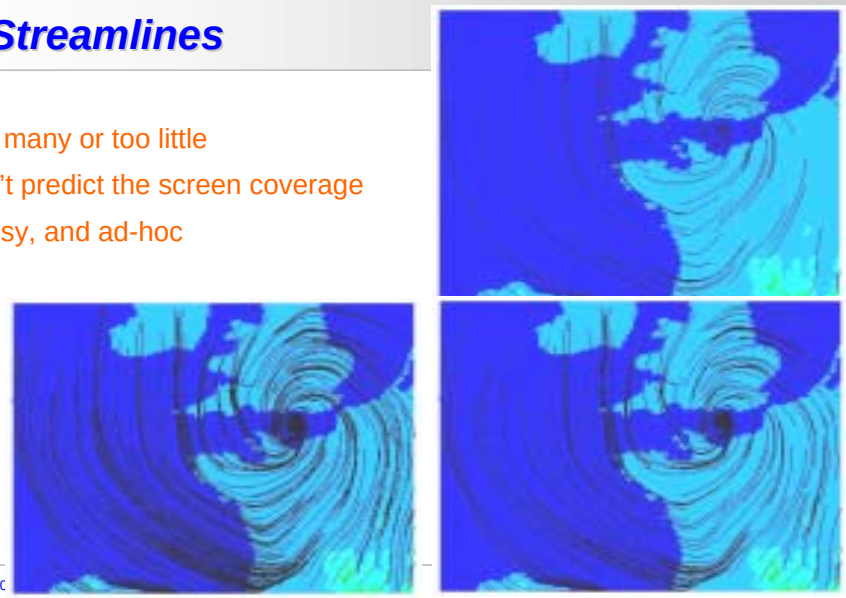
Visualization of flow vector fields

Goals:

- Produce the most effective forms of flow visualization for large-resolutions grids
- Control screen coverage accurately (density, uniformity)
- Avoid visual clutter and distracting effects
- Support transient flows
- Support multi-resolutions solutions

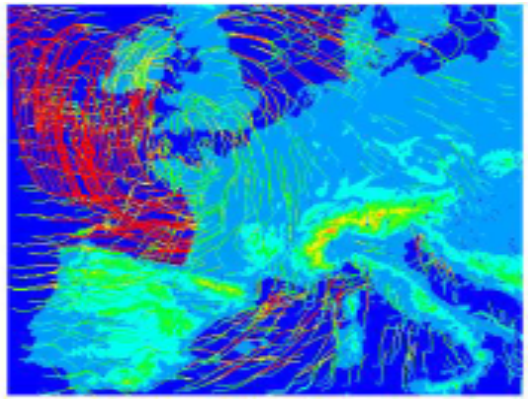
Streamlines

- Too many or too little
- Can't predict the screen coverage
- Messy, and ad-hoc



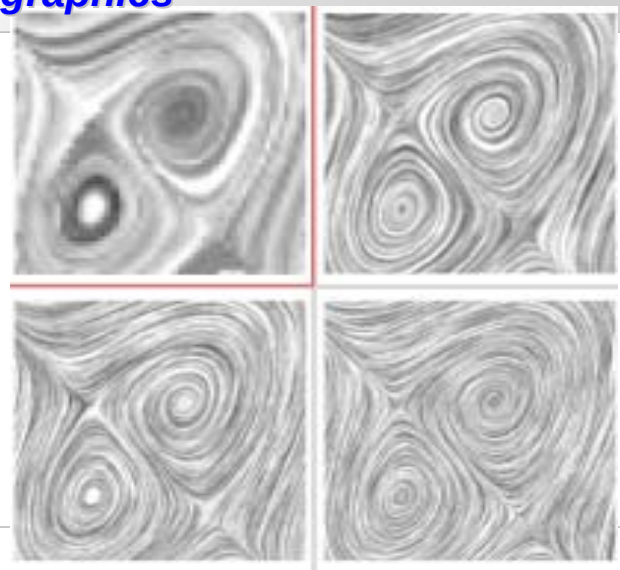
Pathlines

- Very compute- and memory-intensive
- Spaghetti-like
- Too much screen space is use

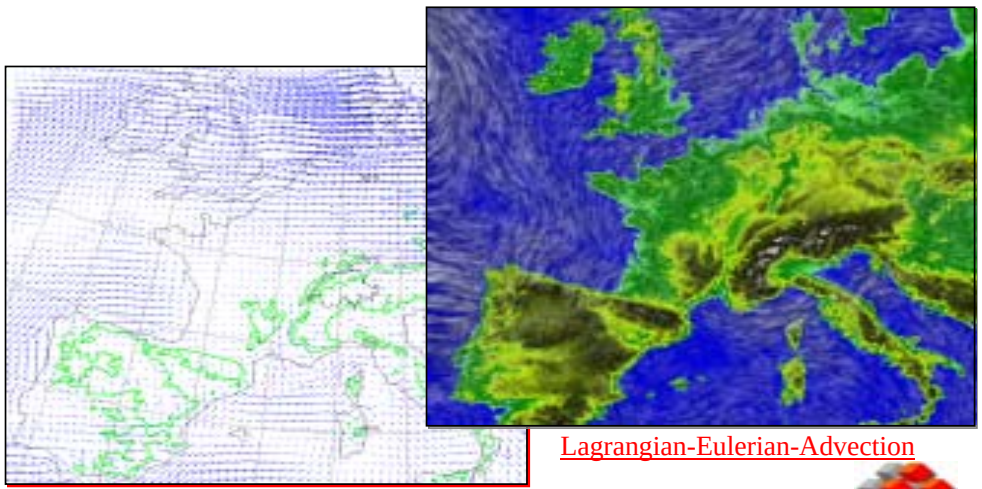


Texture-based graphics

- Independent of the computational grid resolution
- Scales linearly as a function of the number of pixels
- Can be OpenGL-accelerated
- Now available in GPU



Texture-based flow field visualization used in production mode at CSCS



Lagrangian-Eulerian-Advection

Particle Rendering using Point Sprites

Requirements

- Large number of point data (SPH, molecule, astronomy)
- Traditional (polygon-based) has a lot of overhead
  - Create a sphere-representation
  - Time-dependent number, scaling, coloring of geometries not graphics-memory efficient

Rendered spheres(NVidia Quadro FX3400)

Polygonal	10,000 spheres	8x8	25 frames/sec
Polygonal	150,000 spheres	8x8	0.5 frames/sec

Particle Rendering using Point Sprites

Point-Sprite use newer OpenGL extensions

- GL_ARB_point_sprite
- GL_NV_point_sprite

Two modes implemented so far

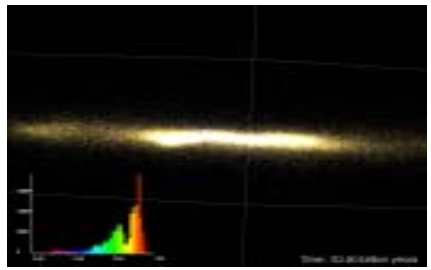
- Accumulative
  - Ideal for Galaxy rendering
- Occluding
  - Ideal for solid spheres

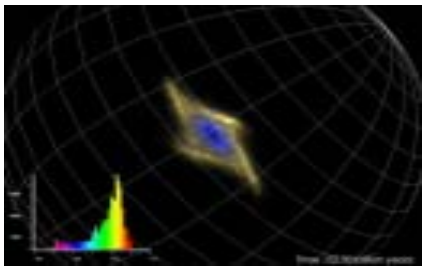
High frame rates compared to rendered spheres

- Sprites 10,000 sprites 64x64 texture 120+ fps
- Sprites 150,000 sprites 64x64 texture 15 fps

Accumulative Mode : Uses

Astronomical simulations with very large numbers of particles





Accumulate all RGB values of all overlapping points (Galaxy bright when edge on)

No attenuation of “hidden” points

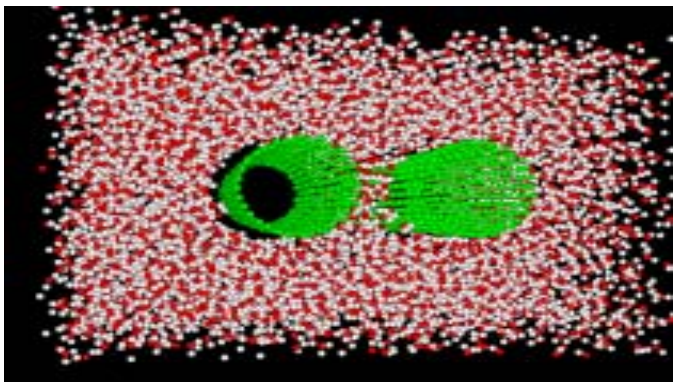
No depth sorting necessary for transparent particles

Depth testing turned off : glBlendFunc(GL_SRC_ALPHA, GL_ONE)

Occluding Mode : Uses

Any simulation using solid particles

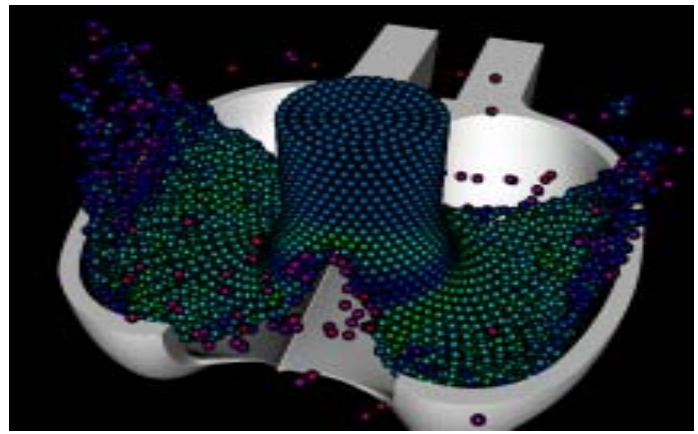
- No transparency
- No depth sorting
- Depth buffer testing on as usual.
- But…
- Need to keep particles small enough that they do not overlap - depth testing only done on a single point – popping effects and overlapping artefacts are visible when mixing with conventional geometry



Particle editing

Choice of particle

- Particle can be Gaussian blob, with parameters adjusted on screen.
- Particle can be a simple bitmap image loaded from disk, e.g. Sphere or other image.
- Or particle could be generated by rendering an image to a texture.
- Gaussian particle good for accumulative mode (Galaxy), but disappointing for solid particles as too much of the image is dark.



Conclusion

VTK/ParaView can be tailored for large-data visualization apps.

in-house data interfaces need to support the VTK model of *piece-wise* handling.

Once mastered, a lot of varied parallel data extraction and rendering features are available to handle large data.

Some new rendering techniques are being developed. They will eventually be part of the standard VTK library.