Distributed Visualization
Parallel Visualization
Large data volumes

Dr. Jean M. Favre
Head of Scientific Visualisation

Outline

Historical perspective
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

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

Most examples in this class
Most practice in our Center

Large Data Volumes

Definition: What is considered “large”?

I’ll accept Jim Ahrens (LANL) own definition. “it is big if it does not fit on the computer you have access to”

Can be large because:
- it is time-dependent.
- It is partitioned over multiple domains.
- It is a very complex geometry.
Strategies to deal with large data

Data access
- on-demand
- client-server
- streamed
- parallel

Interaction techniques
- Efficient
- Progressive
- Interruptible

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

The Visualization Pipeline

Data Sources: reading from data files, or generating data on-line

Data Filters: construct objects used to understand the data
examples: warp scalar, contour, shrink, streamline, probe, gradient

Data Mappers: convert filters’s output to geometric primitives
examples: polygonal models (triangle strips)

Rendering: OpenGL or off-screen + interaction modes

The Visualization pipeline is an active collection of modules allowing users to create/delete derived quantities and objects to represent their data.
The Visualization Pipeline

The Visualization pipeline can most often be represented by a graph (DAG) of modules connected to each other with data “flowing” between them.

IRIS Explorer, AVS, MayaVi make this graph visible.
(see picture on the right) =>

EnSight, ParaView hide it from the user while maintaining its functionality.

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.

Why is this important?
Scaling to large data, and planning a distributed visualization should be driven bottom-to-top.
**Data access**

*on-demand* data loading: (was first popularized with EnSight). Variables are only read when accessed.

*client-server model*: share the memory cost between two machines

(The network bandwidth might become the bottleneck)

*streaming* data: VTK offers *piece-wise* data access for intermediate or accumulated representations.

*parallel data access*: piece-wise data + parallel processing

---

**VTK and ParaView**

- [www.vtk.org](http://www.vtk.org) and [www.paraview.org](http://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

<table>
<thead>
<tr>
<th>Feature</th>
</tr>
</thead>
<tbody>
<tr>
<td>Very rich feature set (driven by graphics &amp; viz conference publications)</td>
</tr>
<tr>
<td>3D direct-manipulations widgets</td>
</tr>
<tr>
<td>Level of Details</td>
</tr>
<tr>
<td>Composite datasets are first-class citizens</td>
</tr>
<tr>
<td>Client-server</td>
</tr>
<tr>
<td>- Pixel sub-sampling over slow connection</td>
</tr>
<tr>
<td>MPI-based parallelism</td>
</tr>
<tr>
<td>- Ghost-cells</td>
</tr>
<tr>
<td>- Load-balancing</td>
</tr>
<tr>
<td>- Piece invariance</td>
</tr>
<tr>
<td>Tiled-display</td>
</tr>
<tr>
<td>Missing features can be added by creating C++ classes</td>
</tr>
</tbody>
</table>

### VTK Event-driven Data Streaming

<table>
<thead>
<tr>
<th>Data Sources</th>
<th>Data Filters</th>
<th>Data Mappers</th>
<th>Graphics</th>
<th>Rendering</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data larger than memory is treated</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>- Piece by piece</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>- Releasing memory after each subset</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>- Optionally accumulating sub-object representations for the final image</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameter to set:</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Set Number Of Pieces</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
VTK Data Streaming

Access to tera-bytes of data on the desktop?
Load datasets of any size by splitting the volumes in pieces.
Process the split data.
There should be no overhead. It is the classic “number crunching” app.

2D Data Streaming example

Visualize the terrain from a high-density DEM

# vtk DataFile Version 3.0
European DEM File
BINARY
DATASET STRUCTURED_POINTS
DIMENSIONS 8319 7638 1
ORIGIN 0 0 0
SPACING 1 1 1
POINT_DATA 63540522
2D Data Streaming example

The Extent Translator will do a binary subdivision of the data and let the user access pieces one at a time.

Sub-sampling to 1 million points

The dataset is shrunk by a factor of 64 (8 in each dimension)

Problem is that it is first read in memory before being shrunk!
**Full resolution is possible**

Streaming, i.e. requesting 1 piece at a time from 64 pieces allow full resolution

---

**3D Data Streaming example**

Loop over all pieces

AND

accumulate the results
**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.

---

**Data Sources inherit a tough responsibility**

Streaming should be supported by the reader/producer of data

- Read data by block, by part, by cells, by levels
- Provide ghost-cells
  
or
- Use VTK Parallel data format

Data Sources

 Data Filters

 Data Mappers

 Graphics

 Rendering
A few considerations about data archiving

Don’t treat VISUALIZATION like a POST-PROCESSING

If you plan your data archiving methods after the fact, it might be very difficult to implement parallel and efficient data visualization.

Examples….

Choose a format which encodes your data efficiently

Example from Bio-electronics Laboratory, ETH-Zurich
A bone biopsy (4.4 millions triangles) encoded as UCD ASCII:

```plaintext
# UCD File Format
#
# File name: Whale_01_SWT59_S04_02P_THRES.INP
# Generator: ucd.exe
2201290 4400884 0 0 0
.....
4400883 1 tri 2147305 2141289 2147306
4400882 1 tri 2147306 2141289 2201290
4400884 1 tri 2147284 2147306 2201290
```

5-line HEADER

Last Lines of file
Example of a spectral element code (EPFL)

- Use TECPLLOT ASCII format.
- Number of block is unspecified. Must read until EOF. Very uncomfortable to allocate memory.
- Convert all blocks to a single UnstructuredGrid
- Block identity is maintained in CellId.

Distributed Data Archiving
Ghost cells

```xml
<VTKFile type="PStructuredGrid" version="0.1">  
    <PStructuredGrid WholeExtent="0 65 0 65 0 65" GhostLevel="1">  
        <Piece Extent="0 17 0 17 0 65" Source="d0372_00.vts"/>
        <Piece Extent="16 33 0 17 0 65" Source="d0372_01.vts"/>
        <Piece Extent="32 49 0 17 0 65" Source="d0372_02.vts"/>
        <Piece Extent="48 65 0 17 0 65" Source="d0372_03.vts"/>
        <Piece Extent="0 17 16 33 0 65" Source="d0372_04.vts"/>
        <Piece Extent="16 33 16 33 0 65" Source="d0372_05.vts"/>
        <Piece Extent="32 49 16 33 0 65" Source="d0372_06.vts"/>
        <Piece Extent="48 65 16 33 0 65" Source="d0372_07.vts"/>
        
        ....  
    </PStructuredGrid>  
</VTKFile>
```
Distributed Data Archiving
Ghost cells. Overlap of 1 cell is seen here.

Particle Data example. TECPLOT (272 Mbytes)

TITLE="resultats"
VARIABLES="Masse (kg)" "X(m)" "Z(m)" "Y(m)"
"Vx(m/s)" "Vz(m/s)" "Vy(m/s)" "rho (kg/m³)" "P(Pa)"
"kpar" "kent" "kfluid"
ZONE I= 39280 F=POINT
  #  25000
  0.000999999932944776234039
  0.01000000000000000208167
  0.01000000000000000208167
  0.000000000000000000000000
  0.000000000000000000000000
  0.000000000000000000000000
  1000.000000000000000000000000
  0.000000000000000000000000
  0.000000000000000000000000 0 0 1
Particle Data example. TECPLOT

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
ParaView’s support for remote rendering

Pixel-subsampling
RLE and bit-compression

Client-server bandwith efficient

Pixel-subsampling and RLE and bit-compression are used
AMR parallel viz and load balancing

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

CNRS 2005 Visualization Summer School

AMR parallel viz and load balancing

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

CNRS 2005 Visualization Summer School
**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

---

**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.

---

CNRS 2005 Visualization Summer School
**NSMB reader/cutter/contour & viewer**

<table>
<thead>
<tr>
<th>MPI-based runs</th>
<th>CPUs =&gt;</th>
<th>1</th>
<th>2</th>
<th>4</th>
<th>8</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read</td>
<td></td>
<td>4.75</td>
<td>4.66</td>
<td>4.67</td>
<td>4.66</td>
<td>3.55</td>
</tr>
<tr>
<td>Solid Surface</td>
<td></td>
<td>9.81</td>
<td>5.10</td>
<td>3.52</td>
<td>2.61</td>
<td>2.04</td>
</tr>
<tr>
<td>Cutting planes</td>
<td></td>
<td>33.8</td>
<td>32.5</td>
<td>32.5</td>
<td>51.56</td>
<td>58.4</td>
</tr>
<tr>
<td>Iso-mach lines</td>
<td></td>
<td>1.20</td>
<td>1.20</td>
<td>2.33</td>
<td>2.86</td>
<td>3.02</td>
</tr>
</tbody>
</table>

**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 compositors [sort-last]**

![ParaView's compositors](image)

**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
   - will probably not fit on the client
   - should be 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 separately.

New composite support
Readers produce a single “composite” object
Filters iterate through the composite structures.

A few remaining problems with distributed processing
Data redistributed among processors

<table>
<thead>
<tr>
<th>Before</th>
<th>After</th>
</tr>
</thead>
<tbody>
<tr>
<td>83504 cells</td>
<td>39872 cells</td>
</tr>
<tr>
<td>29857</td>
<td>idem</td>
</tr>
<tr>
<td>63046</td>
<td>idem</td>
</tr>
<tr>
<td>22914</td>
<td>idem</td>
</tr>
<tr>
<td>34781</td>
<td>idem</td>
</tr>
<tr>
<td>39670</td>
<td>idem</td>
</tr>
<tr>
<td>16180</td>
<td>idem</td>
</tr>
<tr>
<td>29024</td>
<td>idem</td>
</tr>
<tr>
<td>Total=318976</td>
<td></td>
</tr>
</tbody>
</table>

Conclusion

VTK/ParaView can be used for large-data visualization with VTK parallel data formats.

*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.
# Bibliography

<table>
<thead>
<tr>
<th>Resource</th>
</tr>
</thead>
<tbody>
<tr>
<td>VTK User Guide</td>
</tr>
<tr>
<td>ParaView User Guide</td>
</tr>
<tr>
<td>IEEE CG&amp;A</td>
</tr>
<tr>
<td>IEEE Visualization Conferences</td>
</tr>
</tbody>
</table>