Introduction to Scientific Visualization

- Data Sources
- Scientific Visualization Pipelines
- VTK System
Scientific Data Sources

Common data sources:
✓ Scanning devices
✓ Computation (mathematical) processes
✓ Measuring

Application tools usually coupled with
✓ Haptic feedback devices
✓ Stereo output (glasses)
✓ Interactivity
---- demanding of the rendering algorithm
Scanning - Domains

- Biomedical scanners: MRI, CT, SPECT, PET, Ultrasound, confocal microscopes.
- Surface scanner (range data, surface details)
- Geospatial sensor
Surface Scanner

✓ Laser Scanner

✓ Structured Light Scanner
Scanning - Applications

- Medical education, illustration and training
- Biomedical research
Scanning – Applications (2)

✓ Surgical simulation for treatment planning
✓ Medical diagnosis and Tele-medicine
✓ Inter-operative visualization in brain surgery, biopsies, etc. (computer-aided surgery)
✓ Industrial purposes (quality control, security)
✓ Games with realistic 3D effects?
Scanning – Applications (3)

- Range data: Digital library, Virtual museum, etc.
- Geographical Information Systems

Terracotta army
Scientific Computation

- Data sources (domain): Mathematical analysis, ODE/PDE, Finite element analysis (FE), Supercomputer simulations, etc.
- Applications: Scientific simulation, Computational fluid dynamics (CFD), etc.
Measuring - Domains

- Data sources (domain): Orbiting satellites, Spacecraft, Seismic devices, Statistical Data.

- Applications: for military intelligence, weather and atmospheric studies, planetary and interplanetary exploration, oil exploitation, earthquake studies, Statistical Analysis - Info Vis (Financial Data ...)
Remote Runtime Steering of Integrated Terascale Simulation and Visualization

Hongfeng Yu
(Technical lead)
Department of Computer Science
University of California, Davis

Tiankai Tu
(Team lead)
Computer Science Department
Carnegie Mellon University

Jacobo Bielak (CMU)  Omar Ghattas (UT Austin)  Julio C. L´opez (CMU)
Kwan-Liu Ma (UC Davis)  David R. O’Hallaron (CMU)  Leonardo Ramirez-Guzman (CMU)
Nathan Stone (PSC)  Ricardo Taborda-Rios (CMU)  John Urbanic (PSC)
Visualization Functional Model

☑ Shows data flow, transformation, and functional dependencies between processes

☑ Data flow diagram (DFD)
  - **Data source**: creates data
  - **Data sink**: consumes data
  - **Filter**: data transformation
  - **Data store**: data storage
  - **Data synchronization**
DFD example

3D medical imaging system:

- **CT/MRI scanning** → **Raw data** → **Data reconstruction**
- **Raw data** → **slices** → **Surface extraction** → **triangles** → **rendering** → **pixels** → **images**
- **Raw data** → **write** → **Data file**
DFD example (2)

Visualizing function: $F(x, y, z) \in R$

Sampling $F(x, y, z)$ → Point array → Cutting plane

Iso-Surface extraction → Meshed samples → Line contouring

polys → display

display
Visualization pipeline

- **Data objects**: data and methods to create, access & delete data

- **Process objects**: processes to transform data
  - **Source object** (or read object): creates data.
  - **Filter object**: transform data
  - **Mapper object** (or write object): consume (display, output) data.

- **Pipeline connections require type and multiplicity consistency**
Pipeline example

Sampling $F(x,y,z)$

- Iso-surface extraction
  - Display

- Cutting planes
  - Display

- Contouring
  - Display
Looping Pipeline

Example: velocity field

Sampling points

Probe data with points for velocity $\vec{V}_i$

Compute motion

$P_{i+1} = P_i + \vec{V}_i \cdot \Delta t$

Display points

Input field data
Executing Pipeline

✓ Pipeline is executed only when there is a change to input data or process parameter

✓ Demand-driven approach: when output is requested (VTK style)

✓ Event-driven approach: every change leads to pipeline re-execution
Data Characteristics

- Visualization data is discrete
  - Digital technology
  - Analytic property unknown - sampling necessary
  - Interpolation necessary
- Regularity
  - Regular (structured) data
  - Irregular (unstructured) data
- Dimensionality
  - 1D (curve, line), 2D (surface), 3D (volume)
  - Multiple use of lower dimensional techniques for higher dimensional problems.
Dataset model

- Dataset = structure + attributes
- Structure: spatial info and relationships
  - Topology: cells
  - Geometry: points
- Attributes: properties
  - temperature, velocity, etc
  - Associated with the structure
- Example: height field (2D or 3D cells)
Cells and Points

✓ Cells
  - type
  - connectivity list $C_i = \{P_1, P_2, \ldots, P_n\}$
  - Primary cells: smallest unit in topology
  - Composite cells: groups of primary cells

✓ Points
  - Pointers to point storage
  - Use set: $U(P_i) = \{C_i: P_i \text{ in } C_i\}$
Cell types

- vertex
- line
- poly-line
- triangle
- triangle-strip
- quadrilateral
- pixel
- polygon
- tetrahedron
- pyramid
- hexahedron
- voxel
Attributes

✓ Information associated with structure (usually with points)

✓ Common attribute types
  – Scalar: temperature, density, pressure, etc
  – Vector: velocity, trajectory, gradient, etc.
  – Normals: direction only
  – Tensors: k-dimensional array (e.g. stress & strain)
  – User-defined data
  – Texture coordinates
Type of datasets

✓ Structured points
  - Regular, rectangular lattice of points
  - Cells: pixels or voxels
  - 2D (image, pixmap, bitmap), 3D (volume)
  - Simple data structures
Type of datasets (2)

✓ Rectilinear grid
  • Regular lattice
  • Non-uniform spacing

✓ Structured grid
  • Regular topology
  • Irregular geometry
Unstructured grid
- Irregular topology
- Irregular geometry
- Finite element grid
- tetrahedral grid, etc.

Unstructured points
- No topology
- Unstructured geometry
  - Particle systems
  - Smoothed Particle Hydrodynamics
VTK: A Scientific Visualization and Graphics System


✓ VTK Software

http://www.vtk.org/
VTK as a toolkit

✓ C++ Core
  – Each module is a C++ class
  – Type of connections enforced by compiler
  – Connect by `SetInput()` and `GetOutput()` methods

✓ Portable across platforms (e.g. built on openGL)
✓ Interface support through Tcl/Tk and Java interpreters
VTK Graphics Model

- Scene consists of objects, called *actors* in VTK, viewed by virtual camera.
- *vtkActor*
  - Has polygon geometry
  - Positions and geometry in world coordinate system
- *Camera* and *Lights* can be defined (there are default light and camera parameters)
- Actors and camera can be transformed
- Actor transformation handled by *vtkProp* (a super class of *vtkActor*)
VTK graphics model (2)

- **Windows**: `vtkRenderWindow`
  - Can have multiple windows (instances)

- **Renderer**: `vtkRenderer`
  - Coordinates rendering process, handle lights, camera and objects.
  - Can have multiple renderers in a window, each has its own viewports, objects, mappers and rendering properties.
VTK Graphics Model (3)

✓ Props (*vtkProp*):
  - super class of *vtkActor*, *vtkDataset*, *vtkVolume*, etc.
  - objects added to a renderer to create a scene
  - associated with a mapper and a property object
  - Actor: representing 3D geometric objects

✓ Mapper: *vtkMapper*, *vtkDataSetMapper*, *vtkPolyDataMapper*, etc.
  - Rendering methods for data and objects

✓ Properties: *vtkProperty*, *vtkVolumeProperty*, etc.
  - Rendering parameters
VTK Graphics Objects

Instances of render window

Renderer instances

Actor instances
Interact with rendering window

- **Interactor:** `vtkRenderWindowInteractor`
  - Attached to a window
  - Activating interactor for event-handling
  - Event-driven pipeline execution

- **Interactor functions:**
  - *Rotate* (left mouse button)
  - *Zoom* (right mouse button)
  - *Pan* (left mouse+shift)
  - “w”: draw wireframe
  - “s”: draw surface mesh
  - “r”: reset camera
  - “e”: exit
  - “p”: pick actor (underneath mouse pointer)
Example

```c++
vtkRenderWindow *renWin = vtkRenderWindow::New();
renWin->SetSize(600,300);
vtkRenderer *ren1 = vtkRenderer::New();
vtkRenderer *ren2 = vtkRenderer::New();
ren1->SetViewport(0.0,0.0,0.5,1.0);
ren1->SetBackground(0.8,0.4,0.2);
ren2->SetViewport(0.5,0.0,1.0,1.0);
ren2->SetBackground(0.1,0.2,0.4);
renWin->AddRenderer(ren1);
renWin->AddRenderer(ren2);
vtkRenderWindowInteractor *iren = vtkRenderWindowInteractor::New();
iren->SetRenderWindow(renWin);
renWin->Render();
iren->Start();```
Reader, Writer, Source, Filter, etc

- **Reader**: `vtkDataSetReader`, `vtkPolyDataReader`, `vtkVolumeReader`, `vtkStructuredGridReader`, `vtkPNMReader`, etc.
  - Reading from datafile

- **Writer**: `vtkDataSetWriter`, `vtkPolyDataWriter`, `vtkUnstructuredGridWriter`, `vtkPNMWriter`, etc.
  - Write to datafile

- **Source**: `vtkCubeSource`, `vtkStructuredGridSource`, etc
  - Source object

- **Filter**: `vtkContourFilter`, `vtkStreamLine`, `vtkHedgehog`, ...
  - Filter object

- **Misc**: `vtkCamera`, `vtkLight`, etc.
Example: rendering geometric objects

vtkCubeSource *cube = vtkCubeSource::new();
vtkPolyDataMapper *mapper = vtkPolyDataMapper::new();
mapper->SetInput(cube->GetOutput());

vtkActor *actor = vtkActor::new();
actor->SetMapper(mapper);
ren1->AddProp(actor);
actor->RotateX(30.0);  actor->RotateY(20.0);
actor->GetProperty()->SetColor(1.0,.07,.07);
ren1->ResetCamera();
renWin->Render();
The VTK Model - a visualization pipeline

Data Object → Process Object → Display

(Computational Methods, Measured Data)

Source → Filter → Mapper

Procedural, Reader
Transforms the data
Creates Graphics Primitives
Data Representation: Cells & Points

- **Topology**
  - Shape such as triangle, tetrahedron

- **Geometry**
  - Point Coordinates assigned to a topology

- **Data Attributes**
  - Data associated with topology or geometry
Cells specify Topology

- Vertex
- Polyvertex
- Line
- Polyline
- Triangle
- Triangle Strip
- Quadrilateral
- Polygon
- Tetrahedron
- Hexahedron
- Voxel
Cells

✓ Cell is defined by an ordered list of points
  - Triangle, quadrilateral points specified counter clockwise
  - Others as shown

Tetrahedron

Hexahedron
**Meshes are made of Cells**

- Cells can be many different shapes and sizes
  - 2D: Triangles, Quadrilaterals, etc
  - 3D: Tetrahedra, Hexahedra, Pyramids, etc.
- Meshes can consist of one or more types of cells

![Diagram of Meshes](image-url)
VTK Dataset Types

- vtkStructuredPoints
- vtkRectilinearGrid
- vtkStructuredGrid
- vtkPolyData
- vtkUnstructuredGrid
- Methods for reading and writing
Datasets

Organizing structure plus attributes

Structured points

Rectilinear Grid

Structured Grid
Unstructured Grid

A collection of vertices, edges, faces and cells whose connectivity information must be explicitly stored
Data Attributes Assigned to points (VTK) or cells

- Scalars
- Vector
  - Magnitude and direction
- Normal
  - A vector of magnitude 1
  - Used for lighting
- Texture Coordinate
  - Mapping data points into a texture space
- Tensor
Visualization of Attributes

✓ Scalar
  – Color Mapping
  – Contouring
    • 3D Isosurface

Contour Value of 5
Visualization of Attributes

✓ Vectors
  – Oriented Lines
  – Oriented Glyphs
  – Streamlines
Datasets

- \texttt{vtkDataSet}
  - \texttt{vtkStructuredPoints}
    - *implicit topology/geometry*
  - \texttt{vtkPointSet}
    - *explicit geometry*
  - \texttt{vtkRectilinearGrid}
    - *implicit topology, semi-explicit geometry*
  - \texttt{vtkStructuredGrid}
    - *implicit topology*
  - \texttt{vtkPolyData}
  - \texttt{vtkUnstructuredGrid}
Data Objects

- vtkPolyData
- vtkStructuredPoints
- vtkStructuredGrid
- vtkRectilinearGrid
- vtkUnstructuredGrid
Process Objects

Source

1 or more inputs

1 or more outputs

Filter

1 or more inputs

1 or more outputs

Mapper
Creating The Pipeline Topology

- `aFilter->SetInput( bFilter->GetOutput())`

- The Role of Type-Checking
  - `SetInput()` accepts dataset type or subclass
  - C++ compile-time checking
  - Interpreter run-time checking
VTK File Format

✓ Several standard file formats supported
✓ VTK file format:

1. Title line:
2. Header line:
3. Format line:
4. Dataset structure:
5. Dataset attributes:

```
# vtk DataFile Version 2.3
This is a sample datafile
ASCII (or BINARY)
DATASET type
……
POINT_DATA n
……
CELL_DATA n
……
```
VTK Examples

http://www.vtk.org/Wiki/VTK/Examples/Cxx
main ()
{
    vtkRenderer *renderer = vtkRenderer::New();
    vtkRenderWindow *renWin = vtkRenderWindow::New();
    renWin->AddRenderer(renderer);
    vtkRenderWindowInteractor *iren = vtkRenderWindowInteractor::New();
    iren->SetRenderWindow(renWin);
    vtkSphereSource *sphere = vtkSphereSource::New();
    sphere->SetPhiResolution(12);    sphere->SetThetaResolution(12);
    vtkElevationFilter *colorIt = vtkElevationFilter::New();
    colorIt->SetInput(sphere->GetOutput());
    colorIt->SetLowPoint(0,0,-1);
    colorIt->SetHighPoint(0,0,1);
    vtkDataSetMapper *mapper = vtkDataSetMapper::New();
    mapper->SetInput(colorIt->GetOutput());
}
vtkActor *actor = vtkActor::New();
actor->SetMapper(mapper);
renderer->AddActor(actor);
renderer->SetBackground(1,1,1);
renWin->SetSize(450,450);
renWin->Render();
iren->Start();

// Clean up
renderer->Delete();
renWin->Delete();
iren->Delete();
sphere->Delete();
colorIt->Delete();
mapper->Delete();
actor->Delete();
}
vtkRenderer *aren = vtkRenderer::New();
vtkRenderWindow *renWin = vtkRenderWindow::New();
  renWin->AddRenderer( aren);
vtkRenderWindowInteractor iren = vtkRenderWindowInteractor::New();
  iren->SetRenderWindow(renWin);
vtkSTLReader stl = vtkSTLReader::New();
  stl->SetFileName ("cad.stl");
vtkPolyDataNormals normals = vtkPolyDataNormals::New();
  normals->SetInput (stl.GetOutput ());
  normals->SetFeatureAngle (30);
vtkPolyDataMapper mapper = vtkPolyDataMapper::New();
  mapper->SetInput (normals.GetOutput ());
vtkActor actor1 = vtkActor::New(0);
  actor1->SetMapper (mapper);
  actor1->GetProperty () ->SetColor (.8, 1, .9);
aren->AddActors(&actor1);
renWin->Render ();
iren->Start ();
Rendering Volume Data

```
vtkSLCReader *vReader = vtkSLCReader::new();
vReader->SetFileName("hip.slc");
vtkVolumeTextureMapper2D *vMapper = vtkVolumeTextureMapper2D::new();
vMapper->SetInput(vReader->getOutput());

vtkPiecewiseFunction *vOpacity = vtkPiecewiseFunction::new();
vOpacity->AddPoint(0,0.0); vOpacity->AddPoint(255,0.2);
vtkColorTransferFunction *vColor = vtkColorTransferFunction::new();
vColor->AddRGBPoint(64,1.0,0.0,0.0);
vColor->AddRGBPoint(128,0.0,0.0,1.0);
vColor->AddRGBPoint(196,0.0,1.0,0.0);

vtkVolumeProperty *vProp = vtkVolumeProperty::new();
vProp->SetColor(vColor);
vProp->SetScalarOpacity(vOpacity);
```
vtkVolume *volume = vtkVolume::new();
volume->SetMapper(vMapper);
volume->SetProperty(vProp);
ren2->AddProp(volume);

vtkPolyDataReader *sReader = vtkPolyDataReader::new();
sReader->SetFileName("hipSurface.vtk");
vtkPolyDataMapper *sMapper = vtkPolyDataMapper::new();
sMapper->SetInput(sReader->GetOutput());
vtkActor *sActor = vtkActor::new();
sActor->SetMapper(sMapper);
Ren2->AddProp(sActor);

ren2->ResetCamera();
renWin->Render();