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
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, earth quake studies, Statistical Analysis - Info Vis (Financial Data ...
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

slices

write

Data file

Surface extraction → triangles

rendering → pixels

images

Raw data → Slices

write

Data file
DFD example (2)

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

1. Sampling \( F(x, y, z) \) → Point array
2. Iso-Surface extraction
   - Polygons
     - Display
3. Cutting plane
   - Meshed samples
     - Display
4. Line contouring
   - Lines
     - 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**
- **Cutting planes**
- **Contouring**

- Display
Looping Pipeline

Example: velocity field

Sampling points

Probe data with points for velocity $v$

Compute motion $p_{i+1} = p_i + v_i \cdot \Delta t$

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: $U(P_i) = \{C_i : P_i \ in \ C_i\}$

Cell types

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

Type of datasets (3)

- **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.
- \texttt{vtkActor}
  - Has polygon geometry
  - Positions and geometry in world coordinate system
- \texttt{Camera} and \texttt{Lights} can be defined (there are default light and camera parameters)
- Actors and camera can be transformed
- Actor transformation handled by \texttt{vtkProp} (a super class of \texttt{vtkActor})

VTK graphics model (2)

- Windows: \texttt{vtkRenderWindow}
  - Can have multiple windows (instances)
- Renderer: \texttt{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 rendered 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
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

```cpp
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);```
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

```c++
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();
```
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
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

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
Process Objects

```
Source
  ↓
1 or more outputs
  ↓
Filter
  ↓
1 or more outputs
  ↓
Mapper
```

1 or more inputs

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: # vtk DataFile Version 2.3
   - This is a sample datafile
2. Header line: ASCII (or BINARY)
3. Format line: DATASET type
   - ...
4. Dataset structure: POINT_DATA n
   - ...
5. Dataset attributes:

VTK Examples

http://www.vtk.org/Wiki/VTK/Examples/Cxx
```c++
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);
    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();
}
```
Rendering Volume Data

```cpp
tvtkSLCReader *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, 1.0, 0.0);
vColor->AddRGBPoint(196, 0.0, 0.0, 1.0);
```
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);