What is visualization? Why is it important?
What does visualization do?
What is the difference between scientific data and information data
Visualization Pipeline
Visualization Pipeline Overview

- **Data acquisition**: Data are generated.
- **Data enhancement**: Data are processed.
- **Visualization mapping**: Data are mapped to visual primitives, e.g. colors, geometry, etc.
- **Rendering (ND->2D)**: Images are generated.
Visualization Pipeline – Step 1

• **Data acquisition**
  - Real world (measured)
    - Measurements and observations, e.g., CT/MRI, GIS (MB), seismic data (GB), Hubble Space Telescope (TB)...
  - Theoretical world (computed)
    - Mathematical and technical models -> e.g., Structural mechanics (MB), CFD simulation (GB-TB/steady, TB and beyond for time-series)
  - Artificial world (documented/log...)
    - Data that is designed, e.g. drawing (MB), game industry (GB)
Visualization Pipeline – Step 2

- **Data enhancement**
  - Filtering (e.g., smoothing), denoising
  - Resampling (e.g., on a different-resolution grid)
    - Reduce data size
    - Calibrate different data sources
  - Data interpolation (e.g., linear, cubic, basis,...)
    - For continuous data only

- **Analysis** *(may be separated into different steps)*
  - Feature identification
  - Data derivation (e.g., gradients, limits, curvature, closed sub-sets, structure,...)
Visualization Pipeline – Step 3

- **Visualization mapping** = map data to something that is renderable
  - Choose proper geometric elements
    - Grids (e.g., the original meshes, images, etc.) – *typically come with the data.*
    - Iso-contour/surface calculation (create continuity or discontinuity) – *need to be extracted*
    - Glyphs, Icons determination – *need to be constructed, map data to the shape, orientation, size, boundary of the glyphs*
    - Graph-layout calculation (determine geometric locations)

- Choose proper optical attributes for the geometric elements as above
  - color, transparency, texture...
Visualization Pipeline – Step 4

- **Rendering** = image generation with Computer Graphics techniques
  - View point selection
  - Visibility calculation
  - Illumination (determine pixel colors)
  - **Compositing** (combine transparent objects,...)
  - Animation *(a sequence of static images)*
Visualization Pipeline Overview

- Data acquisition
- Data enhancement
- Visualization mapping
- Rendering (ND->2D)

**User interactions**
- Changing view point/parameters
- Selecting different attributes

......
Mike’s Visualization Pipeline—Seven Steps to Creating a Visualization

• 1. Get the data. Find out something about it. Units? Spatial dimension? Data dimension?

• 2. Formulate a scientific strategy. What do you want to show? How do you want to show it? What final format/medium do you need it in?

• 3. Import the data. (This often takes more time than you would ever expect...)

• 4. Create a simple program, network, script, etc.

• 5. Incrementally embellish it. Save it often!

• 6. Choose what quantities you want to interact with. Change the interaction styles to match the quantities being modified.

• 7. Create the final output. Pay attention to your color choices!
Some Useful Principles

Know the content of the application, and characteristics of the data

Understand or translate what the user is looking for

Visualization mapping, User Interface

Data owners
Domain experts
Expected Effects

Simulation (Data Acquisition)

Data Enhancement

Domain expert interpretations

Visual mapping Rendering

Various Visualizations
• Robert S. Laramee, How to Read a Visualization Research Paper: Extracting the Essentials
  http://cs.swan.ac.uk/~csbob/research/how2read/laramee09how2read.pdf

• A great tutorial for getting into this area
Data and Its Representation
General Data Types I

• **Data Dimensionality**
  
  – **1D**: slider bars for scalar value range
  
  – **2D**: geographical data, images, maps, 2D slices of 3D data...
  
  – **3D**: CAD, architecture, medical, biological, scientific computing, ...
  
  – **N-D**: records in logs, data entries in database, social media ...

• **Time-series**: things that are changing over time
General Data Types II

• Data organization
  – Sequential: lists (linear relation)
  – Rational: tables
  – Tree: hierarchical data, nested data
  – Graphs: computer network, social network
LET US FOCUS ON SCIENTIFIC DATA
Scientific Data

- Characteristics of datasets:
  - dimension of domain: number of coordinates or parameters
  - dimension of values
  - static vs. time-dependent

- discretized data
  - type of discretization: (un-)structured grid, scattered data, ...
  - deterministic vs. stochastic (uncertain)
In many cases, scientific data describe certain functions with the input as the spatial coordinates and time, and the output as the data values.

\[
\mathbb{R}^n \times \mathbb{R}^m \overset{f}{\rightarrow} \mathbb{R}^{n+m}
\]

**Scientific data**

Source: VIS, University of Stuttgart

<table>
<thead>
<tr>
<th>Dimension</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>1D</td>
<td>scalar</td>
</tr>
<tr>
<td>2D</td>
<td>vector</td>
</tr>
<tr>
<td>3D</td>
<td>tensor</td>
</tr>
<tr>
<td>+time</td>
<td></td>
</tr>
</tbody>
</table>
A Complexity Space of Data

- **Data sizes**
  - Megabytes
  - Gigabytes
  - Terabytes
  - Petabytes

- **Data Values**
  - Scalar
  - Vector
  - Tensor
  - Higher-order tensor

- **Domain**
  - 2D
  - 3D
  - 2D+time
  - 3D+time
  - Higher dimensional
Data Representation

• In the scientific data regime, the data typically describe continuous physical events in the continuous physical space+time.

• This is contradict to the discrete natural of the digital machine.

• In order to represent these data in the machine with finite space, **discrete representation** is necessary
  
  • how to sample — structured vs. unstructured
    
    • Sampling case — Cartesian grids, images, spacing, etc.
    • Unstructured, connectivity, spacing, primitives (e.g. simplexes)
    • Lists

• Continuous — basis, e.g. polynomials, spectral, wavelet
Data Discretizations

Discretization strategy is determined by the types of data sources:

• Measurement data:
  – typically scattered (no grid)

• Numerical simulation data:
  – structured, block-structured, unstructured grids
  – adaptively refined meshes
  – etc.

• Imaging methods:
  – uniform grids

• Mathematical functions:
  – uniform/adaptive sampling on demand
Scattered Data

• Scattered data means: only nodes, no cells
• Typical data sources: measurement data, e.g. meteorological

• Options for visualization:
  – point-based methods (relatively few algorithms)
  – triangulation, e.g. constrained Delaunay, difficult in 3D
  – resampling on uniform grid
Voronoi Diagram

Source: http://strongriley.github.com/d3/
Delaunay Triangulation

Source: http://en.nicoptere.net/?p=10
Data Stored on Grids – Unstructured

• Typical data sources: simulation data, e.g. CFD

• 2D (plane or surfaces) unstructured
  – cells are triangles and/or quadrangles
  – domain can be a surface embedded in 3-space
Arbitrary Surfaces

- **Mesh (geometry)**
  - Discrete representation

Parametric surfaces

Triangular mesh
Arbitrary Surfaces

• Shape visualization

• **Attributes** that can be used by visualization:
  – Shading/lighting
  – Silhouette
  – Feature curves
  – Colors
  – Transparency
What do we need to represent an unstructured grid?

• For 2-manifold surfaces:
  – Vertex (0D)
  – Edge (1D)
  – Face (2D)
  – Corner
  – Polyhedron (contains the above member variables)
Polyhedron (Container)

- **Vertex list**
  - vertex **vlist;
  - int num_verts, max_verts;
- **Face flist**
  - face **flist;
  - int num_faces, max_faces;
- **Edge elist**
  - face **elist;
  - int num_edges, max_edges;
- **Corner clist**
  - corner **clist;
  - int num_corners, max_corners;
Vertex (required)

- **Basic**
  - $(x, y, z)$ – coordinates, necessary
  - index – almost always needed, automatic based on ordering
  - Attributes or data, like $(nx, ny, nz)$ – normal, optional

- **Derived**
  - List of faces incident to the vertex - almost always needed, constructed later
  - List of edges incident to the vertex – almost always needed, constructed later
Face (required)

- **Basic**
  - List of vertices - necessary, typically the indices of the vertices
  - index – almost always needed, automatic based on ordering
  - Attributes and data, optional

- **Derived**
  - List of edges – almost always necessary, constructed later
Unstructured Grids

• 3D (volume) unstructured
  – cells are tetrahedra or hexahedra
  – mixed grids ("zoo meshes") require additional types:
    • wedge (3-sided prism), and pyramid (4-sided)
What Should be Stored?

- **Tetrahedral**
  - Vertex list
  - 3D cell list with four vertices for each cell

- **Hexahedra**
  - Vertex list
  - 3D cell list with edges (pairs of vertices) that form a cell

- **Hybrid**
  - Vertex list
  - 2D cell list
  - 3D cell list
Structured Grids

• General case: *curvilinear* grid
  – nodes given in array $N_i \times N_j \times N_k$
  – cells are implicit

• Special case: *rectilinear* grid
  – simpler coordinate functions:
    $x = x(i)$, $y = y(j)$, $z = z(k)$

• More special: *uniform* grid
  – coordinates defined by axis-aligned bounding box ($2$ points)
Data Value Storage – where are my data stored?

Point-based

Cell-based

Dual
PLY format

• Header
  – Elements
  – Properties

• Data
PLY format of a Cube

```
ply
format ascii 1.0
comment created by platoply
element vertex 8
property float32 x
property float32 y
property float32 z
element face 6
property list uint8 int32 vertex_indices
end_header
-1 -1 -1
1 -1 -1
1 1 -1
-1 1 -1
-1 -1 1
1 -1 1
1 1 1
-1 1 1
4 0 1 2 3
4 5 4 7 6
4 6 2 1 5
4 3 7 4 0
4 7 3 2 6
4 5 1 0 4
```
PLY format (header)

ply
format ascii 1.0
comment created by platoply
element vertex 8
property float32 x
property float32 y
property float32 z
element face 6
property list uint8 int32 vertex_indices
end_header
PLY format

-1 -1 -1
1 -1 -1
1 1 -1
-1 1 -1
-1 -1 1
1 -1 1
1 1 1
-1 1 1
4 0 1 2 3
4 5 4 7 6
4 6 2 1 5
4 3 7 4 0
4 7 3 2 6
4 5 1 0 4

Vertex list

Face list
PLY format

x y z
-1 -1 -1
1 -1 -1
1 1 -1
-1 1 -1
-1 -1 1
1 -1 1
1 1 1
-1 1 1
4 0 1 2 3
4 5 4 7 6
4 6 2 1 5
4 3 7 4 0
4 7 3 2 6
4 5 1 0 4
PLY format

-1 -1 -1
1 -1 -1
1 1 -1
-1 1 -1
-1 -1 1
1 -1 1
1 1 1
-1 1 1
4 0 1 2 3
4 5 4 7 6
4 6 2 1 5
4 3 7 4 0
4 7 3 2 6
4 5 1 0 4

# of vertices

Vertex indices

Face list

Face list:

0 1 2 3
0 1 2 4
4 1 5 6
1 5 7 2
2 3 6 7
3 0 7 4

# of vertices: 8

Example of PLY format:

-1 -1 -1 1 -1 -1 1 1 -1 -1 -1 -1 4 0 1 2 3 4 5 4 7 6 4 6 2 1 5 4 3 7 4 0 4 7 3 2 6 4 5 1 0 4

Diagram of a cube with vertices and face list.
PLY format

-1 -1 -1
1 -1 -1
1 1 -1
-1 1 -1
-1 -1 1
1 -1 1
1 1 1
-1 1 1
4 0 1 2 3
4 5 4 7 6
4 6 2 1 5
4 3 7 4 0
4 7 3 2 6
4 5 1 0 4

# of vertices

Vertex indices

Face list

0 1 5
1 0 4
4 0 5
5 1 2

PLY format

-1 -1 -1
1 -1 -1
1 1 -1
-1 1 -1
-1 -1 1
1 -1 1
1 1 1
-1 1 1
4 0 1 2 3
4 5 4 7 6
4 6 2 1 5
4 3 7 4 0
4 7 3 2 6
4 5 1 0 4

# of vertices

Vertex indices

Face list

-1
1
5
5
4
4
5
0
1
2
0
3
6
1
2
7

x
y
z
Store Values in PLY Files

X Y Z v1 v2

-1 -1 -1 0 0
1 -1 -1 0.1 -0.1
1 1 -1 -0.12 0.6
-1 1 -1 0.31 0
...

... element vertex 8
property float32 x
property float32 y
property float32 z
property float32 v1
property float32 v2
...
VISUALIZATION LIBRARY
• Free open source software for 3D computer graphics, image processing and visualization

• Consists of a C++ class library
  • VTK classes implemented with .h and .cxx files
  • Several interpreted interface layers, including Python, Tcl/Tk, and Java

• Supports a wide variety of visualization algorithms
  • including scalar, vector, tensor, texture, and volumetric

• Advanced modeling techniques
  • implicit modeling, polygon reduction, mesh smoothing, cutting, contouring and Delaunay triangulation...

• Design and implementation influenced by object-oriented principles...
VTK-Visualization Toolkit Resources

• Source distribution (source and binaries) www.vtk.org, Kitware, Inc.
  Distribution comes with many examples (http://www.vtk.org/Wiki/VTK/Examples)

• Documentation
  Online help, HTML based (http://www.vtk.org/doc/release/6.2/html/)

• Companion Text Books
  – The Visualization Toolkit
  – The VTK User’s Guide
VTK Architecture
Object Model

- Dataset types found in VTK
  - Image data
  - Rectilinear grid
  - Structured grid
  - Unstructured points
  - Polygonal Data
  - Unstructured grid

- Data objects have geometric and topological structure (points and cells)

- Cells are topological arrangements of points
VTK Architecture Object Model

- Associated with the points and cells of a dataset
  - scalar
  - vector
  - normal
  - texture coordinate
  - tensor
  - field data
VTK Architecture
Rendering Engine

A VTK scene consists of:

- **vtkRenderWindowInteractor**
  - window interaction
- **vtkRenderWindow**
  - contains the final image
- **vtkRenderer**
  - draws into the render window
- **vtkActor**
  - combines properties/geometry
- **vtkMapper**
  - represents geometry
- **vtkProp, vtkProp3D**
  - Superclasses
- **vtkProperty**

- **vtkLights**
  - illuminate actors
- **vtkCamera**
  - renders the scene
- **vtkTransform**
  - position actors

Carlos A. Vinhais
VTK Architecture
Visualization Pipeline

- **data objects** combined with **process object** to create the visualization pipeline

- **Pipeline execution**
VTK

ParaView

VisIt

Many Others

Image Gallery
Acknowledgment

• Thanks for materials from
• Dr. Eugene Zhang
• Dr. Ronald Peikert
• Carlos A. Vinhais