Rensselaer

# ESCE 4960: Open Source Software Practice
# VTK Overview
# September 15, 2008

Dr. Will Schroeder, Kitware

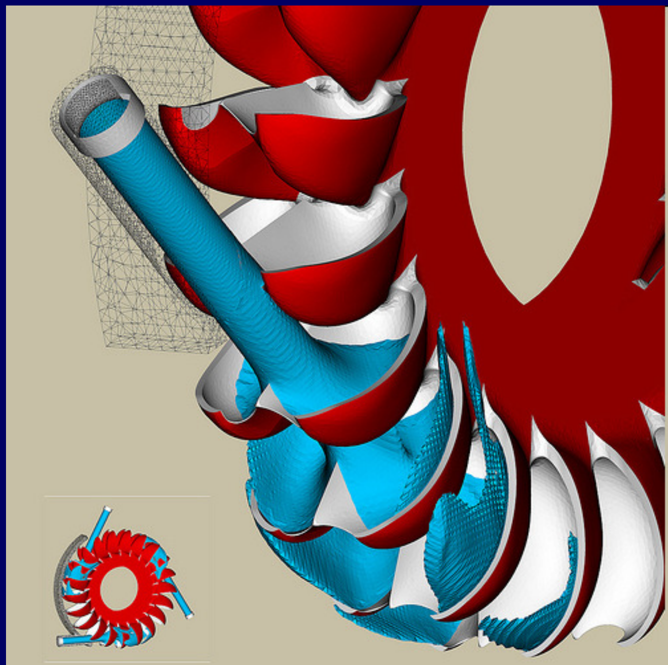
# Overview

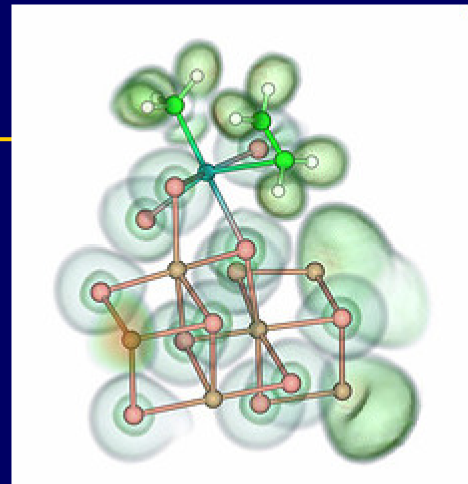
- Case Studies
  - CMake
  - VTK (today)
  - ITK
  - Others
- Basis for course projects

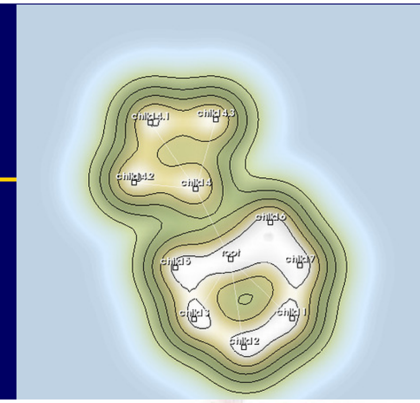
# Delving into Project Character

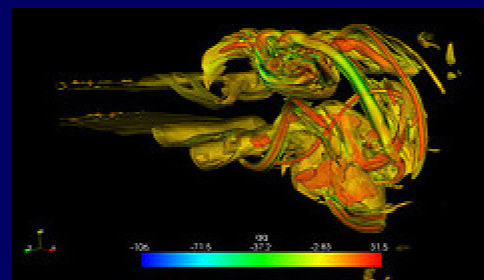
- Project goals
- Origins of the project
- History
- Community
  - Size
  - Constitution
  - Control issues
- Backward Compatibility

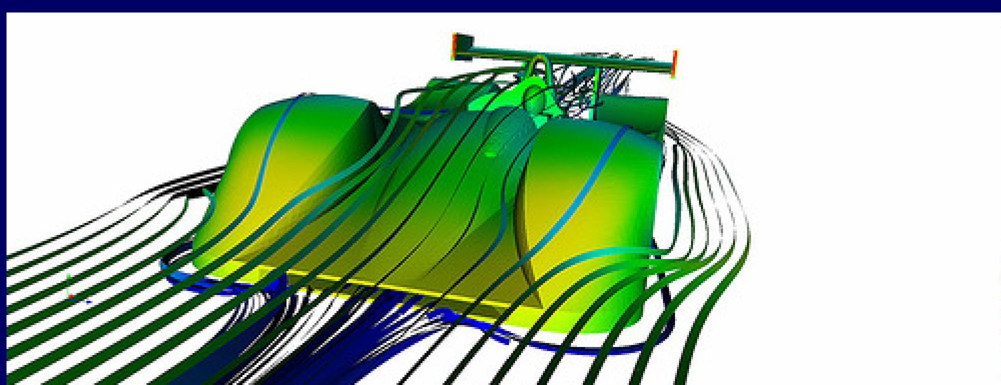
# Examples

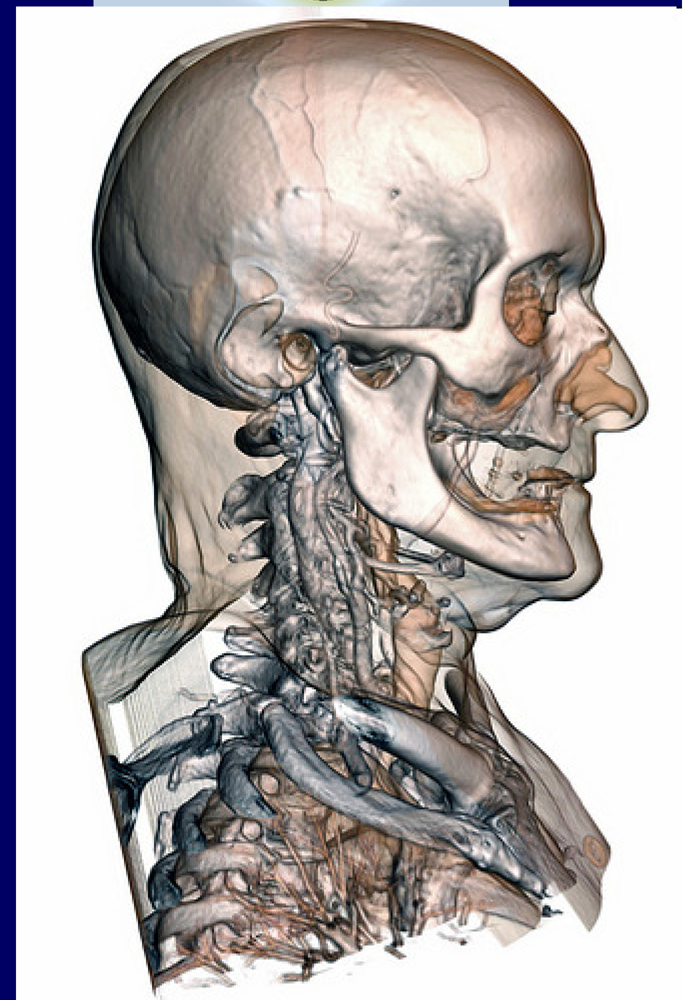












# VTK Goals

- Become the de facto standard for
  - Scientific Visualization
  - Information Visualization
  - Large data visualization
  - Volume rendering
  - 3D interaction

# Background: Visualization

- Definition
  - Map data or information into images or other sensory input (touch, sound, smell, taste)
  - Engages human perception system
  - Simple, effective powerful
    - Complex data
    - Voluminous data

# Background: Visualization

- Scientific Visualization
  - Spatial – temporal data
  - Easy to relate to

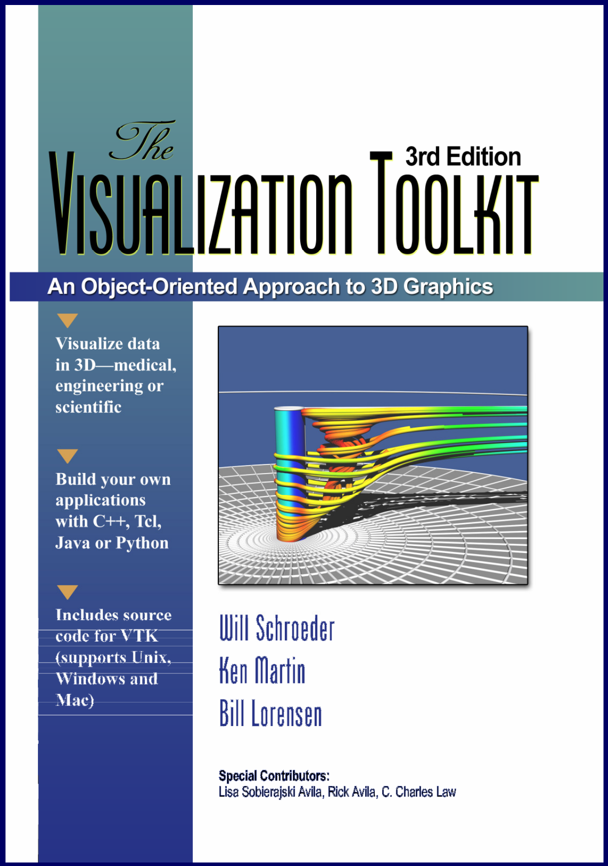
- Information Visualization
  - Non-spatial – temporal
  - Difficult perceptual context

# Visualization

- Related disciplines
  - 3D Graphics
  - Image processing
  - Modeling
  - Computational geometry
  - Numerical methods
  - Scientific and parallel computing
  - GUI and Computer/Human Interaction techniques
  - Perception / Human factors

# Origins of VTK: Textbook

Now in Fourth Edition




**The Visualization Toolkit**
**An Object-Oriented Approach To 3D Graphics**
**Will Schroeder, Ken Martin, Bill Lorensen**
**ISBN 1-930934-07-6**
**Kitware, Inc. Publishers**

***Work on first edition began in December 1993***

# What Is VTK?

A visualization toolkit

- Designed and implemented using object-oriented principles
- C++ class library (750,000 LOC)
- Automated Java, TCL, Python language bindings
- Portable across Unix, Windows XP, MacOSX
- Supports 3D/2D graphics, visualization, image processing, volume rendering

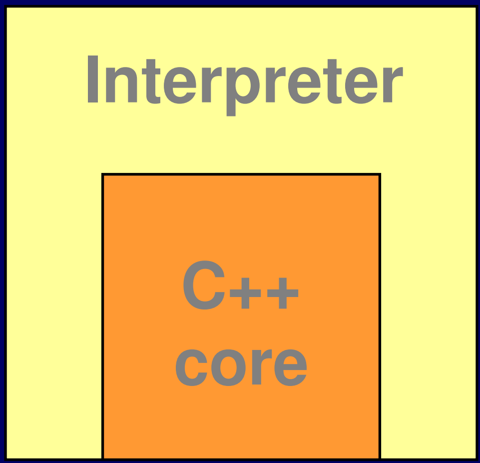
# Building Applications in VTK

- Applications can be created using
  - C++ (compiled language)
  - Tcl (interpreted language)
  - Java (interpreted language)
  - Python (interpreted language)
- Interpreted Languages
  - Have GUI support
  - Easy to prototype with
  - Slower than compiled C++

# VTK Architecture

Hybrid approach

- Compiled C++ core (faster algorithms)
- Interpreted applications (rapid development)
  (Java, Tcl, Python)




***Interpreted layer generated automatically by VTK wrapping process***

# How to Get The VTK Software

- VTK5.2 CD (Release version – most stable)
- Anonymous CVS (Development version – caution)
  - CVS is a source code control system
  - The command:
    ```
    cvs -d :pserver:anonymous@public.kitware.com:/cvsroot/VTK
    co VTK
    ```
    is used to checkout the source code (password "`vtk`")
  - Release branch:
    ```
    cvs -d […] co -r VTK-5-2 VTK
    ```

# VTK Directory Structure

**./VTK**

- **./VTK/Common** – *core VTK classes*
- **./VTK/Filtering** – *classes used to manage pipeline dataflow*
- **./VTK/Rendering** – *rendering, picking, image viewing, and interaction*
- **./VTK/VolumeRendering** – *volume rendering techniques*
- **./VTK/Graphics** – *3D geometry processing*
- **./VTK/GenericFiltering** – *none-linear 3D geometry processing*
- **./VTK/Imaging** – *imaging pipeline*
- **./VTK/Hybrid** – *classes requiring both Graphics and imaging functionality*
- **./VTK/Widgets** – *sophisticated interaction*
- **./VTK/IO** – *VTK input and output*
- **./VTK/Parallel** – *parallel processing (controllers and communicators)*
- **./VTK/Wrapping** – *support for Tcl, Python, and Java wrapping*
- **./VTK/Examples** – *extensive, well-documented examples*

# VTK Documentation

- *VTK User's Guide*
- *The Visualization Toolkit* textbook
- On the VTK5.0 CD (Doxygen):
  *`cdrom:/vtkhtml/html/index.html`*
- On the Web (Doxygen, current state):
  *http://www.vtk.org/doc/release/5.0/html/*
- Embedded documentation in `.h` header files
- Search *`VTK/Examples`*, *`VTK/*/Testing/Tcl`* and *`VTK/*/Testing/Cxx/`* directories for usage

# Doxygen

Alphabetical listing of classes

Method names to classes

Classes to examples; events to classes

Main Page | Class Hierarchy | Alphabetical List | Class List | Directories | File List | Class Members | File Members | Related Pages

VTK 5.0.2 Documentation

**Revision**
1.2196

**Date**
2005/09/01 09:01:29

**Useful links:**
- VTK Home: http://www.vtk.org
- VTK Users Mailing-list: http://public.kitware.com/mailman/listinfo/vtkusers
- VTK Developer Mailing List: http://www.vtk.org/mailman/listinfo/vtk-developers
- VTK FAQ: http://www.vtk.org/Wiki/VTK_FAQ
- VTK Wiki: http://www.vtk.org/Wiki/
- VTK Search: http://www.kitware.com/search.html
- VTK Dashboard: http://www.vtk.org/Testing/Dashboard/MostRecentResults-Nightly/Dashboard.html
- VTK-Doxygen scripts (Sebastien Barre): http://www.barre.nom.fr/vtk/doc/README
- Kitware Home: http://www.kitware.com
- Sebastien's VTK Links: http://www.barre.nom.fr/vtk/links.html
- Other Links: http://www.vtk.org/links.php

# Important File: `vtkSetGet.h`

- System-wide macro definitions
  - vtkTypeRevisionMacro – versioning information
    - GetClassName()
    - IsTypeOf()
    - IsA()
  - vtkSet / vtkGet Macros
  - vtkDebugMacro / vtkWarningMacro / vtkErrorMacro
  - vtkStandardNewMacro() – defines New() method
  - vtkCxxRevisionMacro() – versioning information for .cxx file

# Example Set/Get Macros

- vtkSetMacro(var,type) → void SetVar(type _arg);
  - E.g., SetDebug()
- vtkGetMacro(var,type) → type GetVar()
  - E.g., GetDebug()
- vtkBooleanMacro(var,type) →
  void VarOn(), void VarOff()
  - DebugOn(), DebugOff()
- vtkSetVectorMacro(var,type,count) →
  void SetVar(type data[count])
  - SetPosition(x[3])

# VTK Build Process

- Based on CMake
- Cross-platform
  - Linux, Windows, Mac OSX, others
  - Variety of compilers

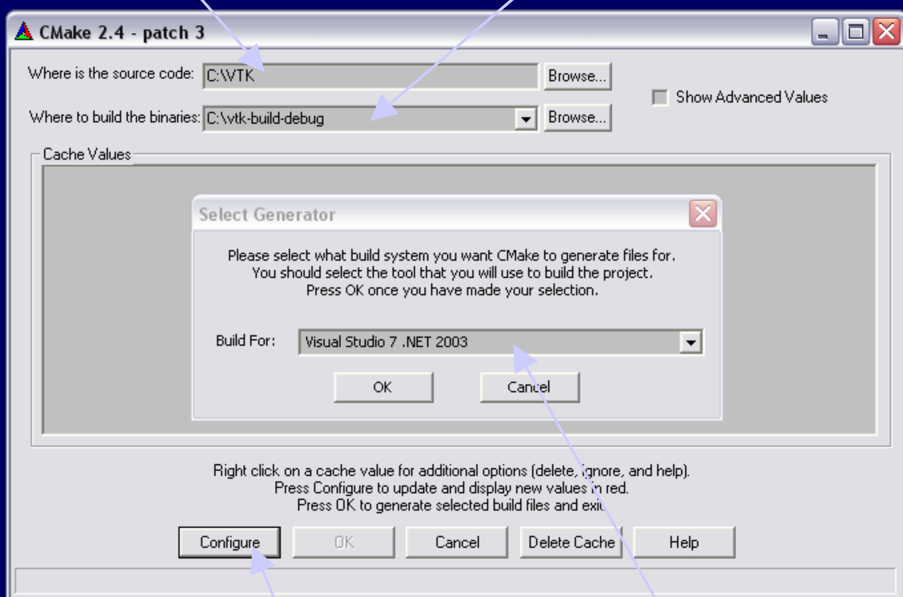
# Running CMake

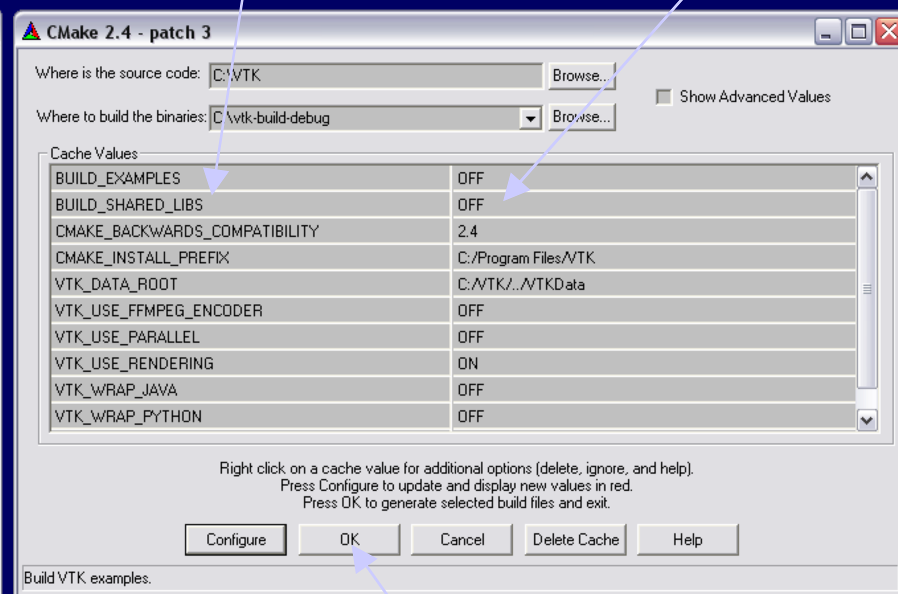
Source code location

Object code location

Build parameters

Build settings




Configure build

Compiler to use

Produce build files

# Reference Counting

- Object creation and destruction - objects shared between other objects requires careful memory management (when to delete memory?)

- In VTK: Instantiate & destroy objects with New() / Delete()
  - `vtkActor *anActor = vtkActor::New();`
  - `anActor->Delete();`

- Can not allocate on the stack due to reference counting
  - ~~`vtkActor anActor;`~~ ← *Compiler won't let you*

- If you need to hold onto an object use Register() / UnRegister() methods.
  - `anActor->Register(…);` ← *Increase reference count*
  - `anActor->UnRegister(…);` ← *Decrement reference count; same as Delete() method*

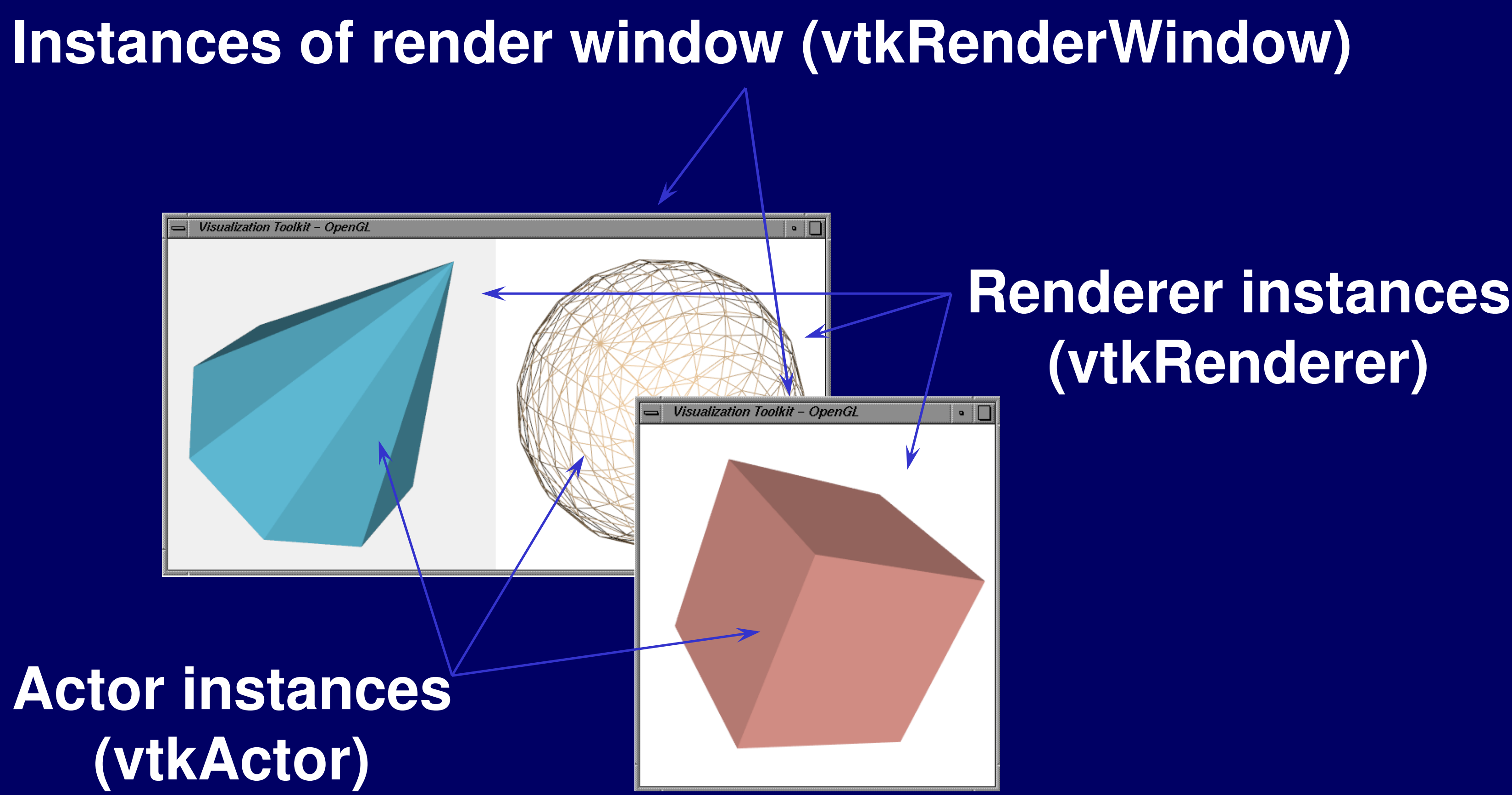
# Reference Counting Example

| *Operation* | *Result* | *this->ReferenceCount* |
|---|---|---|
| `*bar = vtkObject::New()` | ***instantiation*** | **1** |
| `bar->Register(…)` | ***Increment*** | **2** |
| `bar->UnRegister(…)` | ***Decrement*** | **1** |
| `bar->Delete()` | ***Destroy*** | **0** |

# Major VTK Subsystems

- Graphics
- Image processing pipeline
  - ImageData (data object)
  - Filters (process only vtkImageData)
- 3D geometry processing pipeline
  - DataSets (I.e., data objects)
  - Filters (I.e., process datasets or subclasses)
- Picking, Interaction, etc.

# The VTK Graphics Subsystem



# The VTK Graphics Subsystem

A VTK *scene* consists of:

- vtkRenderWindow - contains the final image
- vtkRenderer - draws into the render window
- vtkActor - combines properties / geometry
  - vtkProp, vtkProp3D are superclasses
  - vtkProperty
- vtkLights - illuminate actors
- vtkCamera - renders the scene
- vtkMapper - represents geometry
  - vtkPolyDataMapper, vtkDataSetMapper are subclasses
- vtkTransform - position actors

# Typical Application (in C++)

```
vtkSphereSource *sphere = vtkSphereSource()::New(); // create data pipeline
vtkPolyDataMapper *sphereMapper = vtkPolyDataMapper::New();
  sphereMapper→SetInputConnection(sphere→GetOutputPort());
vtkActor *sphereActor = vtkActor::New();
  sphereActor→SetMapper(sphereMapper); //mapper connects actor with pipeline

vtkRenderer *renderer = vtkRenderer::New();
vtkRenderWindow *renWin = vtkRenderWindow::New();
        renWin→AddRenderer(renderer);
vtkRenderWindowInteractor *iren = vtkRenderWindowInteractor::New();
        iren→SetRenderWindow(renWin);

renderer→AddViewProp(sphereActor);
renderer→SetBackground(1,1,1);
renWin→SetSize(300,300);

renWin→Render();
iren→Start(); //starts the event loop
```

# Graphics API Overview

- The following is a summary of instance variables & methods
- Remember there is typically a Set__() and Get___() method to set and get the instance variable values.
- Refer to Doxygen man pages, or class header files, for more information.

# vtkRenderWindow

- AddRenderer() – add another renderer which draws into this vtkRenderWindow
- SetSize() – set the size of the window
- SetPosition() – set the position of the window
- SetWindowName() – set the name (in the titlebar)
- AAFrames, FDFrames, SubFrames – used for anti-aliasing and focal depth
- StereoType, StereoRenderOn/Off – control stereo
- AbortRender, AbortCheckMethod – methods to interrupt the rendering process

# vtkRenderWindow (cont.)

- DesiredUpdateRate – a frame rate which is used to control LOD (level-of-detail) actors
- DoubleBuffer – turn double buffering on/off
- PixelData, RGBAPixelData, ZbufferData – set/get the color buffer and depth buffer for the window

# vtkRenderer

- AddViewProp (preferred), AddActor, AddVolume, AddActor2D – add objects to be rendered
- AddLight – add a light to illuminate the scene
- SetAmbient – set the intensity of the ambient lighting
- SetViewport – specify where to draw in the render window
- SetActiveCamera – specify the camera to use render the scene
- ResetCamera – reset the camera so that all actors are visible

# vtkCamera

- Position – where the camera is located
- FocalPoint – where the camera is pointing
- ViewUp – which direction is “up”
- ClippingRange – data outside of this range is clipped
- ViewAngle – the camera view angle controls perspective effects
- EyeAngle – the angle between eyes (for stereo)
- ViewPlaneNormal – the normal vector to the view plane

# vtkCamera (cont.)

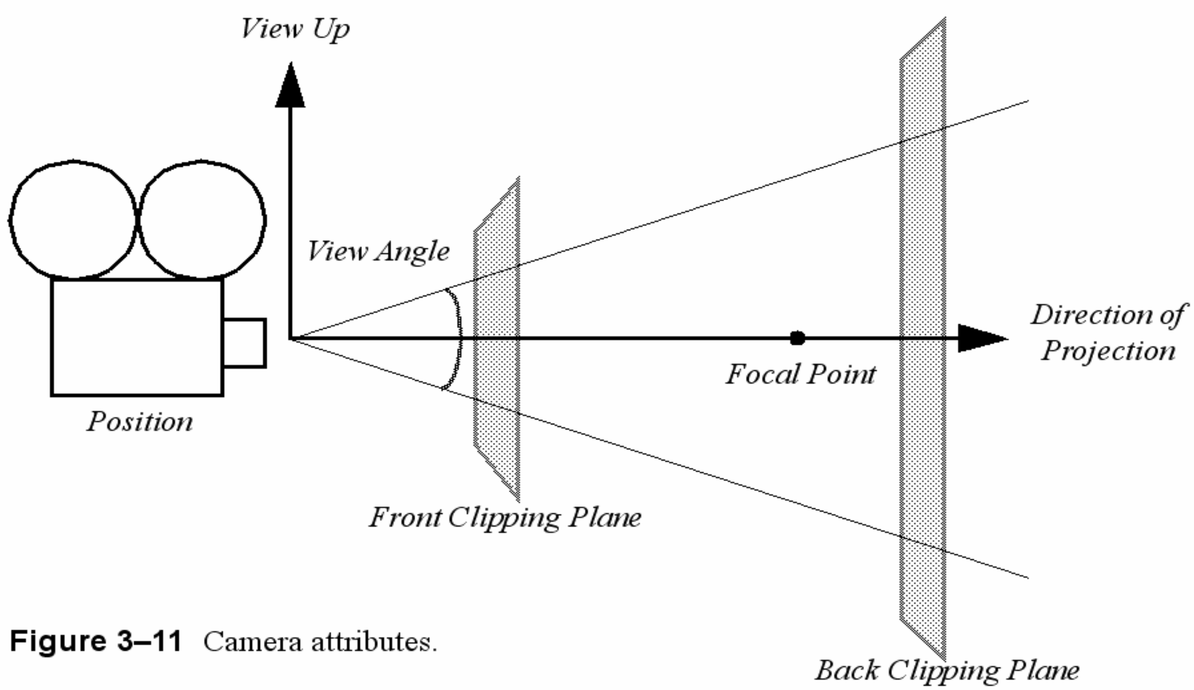
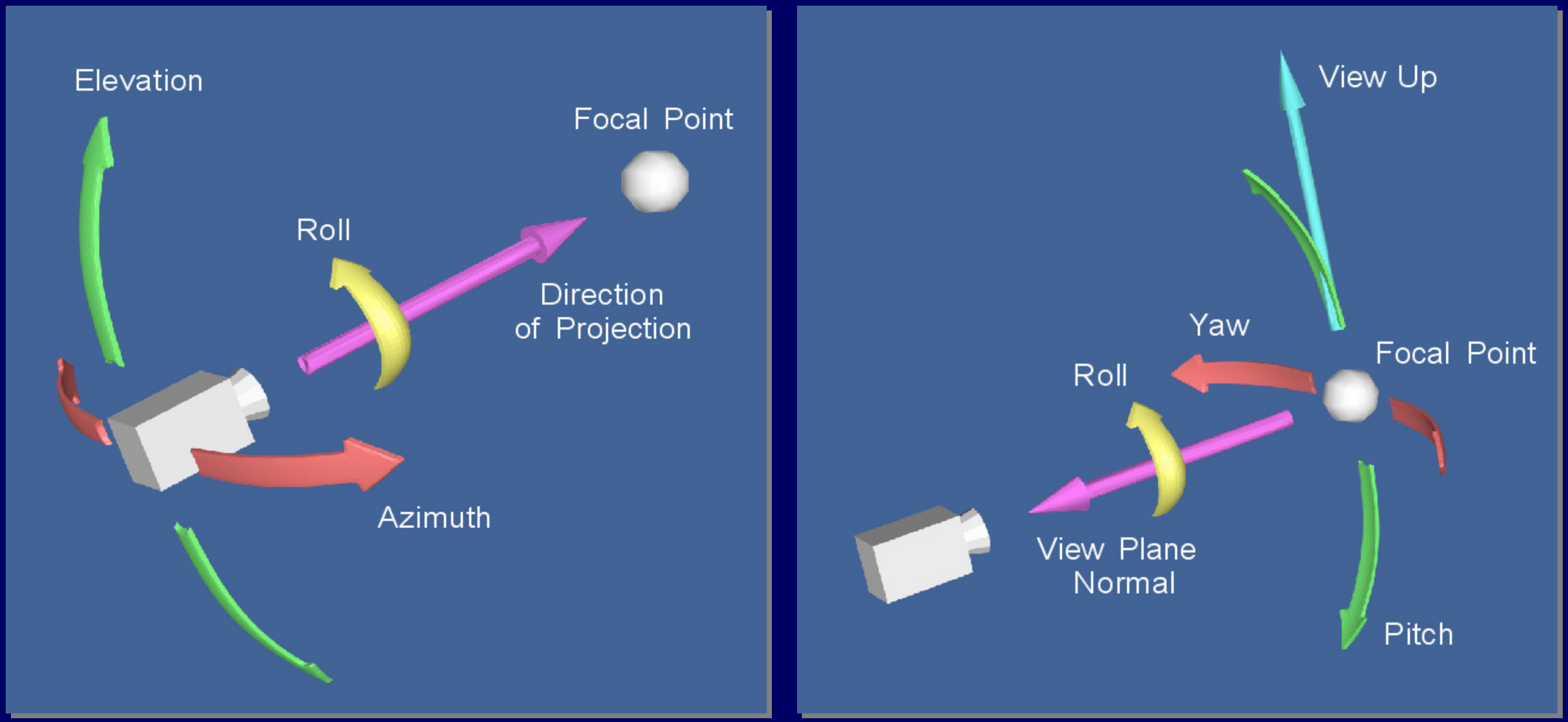



**Figure 3–11** Camera attributes.

# vtkCamera (cont.)

- ParallelProjection – turn parallel projection on/off (no perspective effects)
- ParallelScale – used to shrink or enlarge an image
- Roll, Pitch, Yaw, Elevation, Azimuth – move the camera in a variety of ways
- Zoom, Dolly – changes view angle (Zoom); move camera closer (Dolly)
- OrthogonalizeViewUp – make the view up vector perpendicular to the view plane normal

# vtkCamera (cont.)



# vtkLight

- Color – the light color
- Position – where the light is
- FocalPoint – where the light is pointing
- Intensity – the brightness of the light
- Switch – turn the light on or off
- Positional – is it an infinite or local (positional) light
- ConeAngle – the cone of rays leaving the light

# vtkActor (subclass of vtkProp)

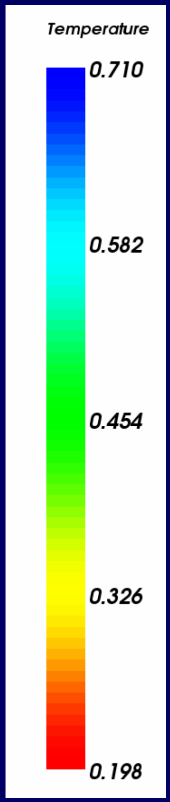
- Property – surface lighting properties
- Texture – a texture map associated with the actor
- Position – where it's located
- Origin – the origin of rotation
- Visibility – is the actor visible?
- Pickable – is the actor pickable?
- Dragable – is the actor dragable?
- RotateX, RotateY, RotateZ – rotate around different axes
- RotateWXYZ – rotate around a vector

# vtkProperty

- Interpolation - shading interpolation method *(Flat, Gouraud)*
- Representation – how to represent itself *(Points, Wireframe, Surface)*
- AmbientColor, DiffuseColor, SpecularColor – a different color for ambient, diffuse, and specular lighting
- Color – sets the three colors above to the same
- Ambient, Diffuse, Specular – coefficients for ambient, diffuse, and specular lighting
- Opacity – control transparency

# vtkLookupTable

- NumberOfColors – number of colors in the table
- TableRange – the min/max scalar value range to map
- If building a table from linear HSVA ramp:
  - HueRange – min/max hue range
  - SaturationRange – min/max saturation range
  - ValueRange – min/max value range
  - AlphaRange – min/max transparency range
- If manually building a table
  - Build (after setting NumberOfColors)
  - SetTableValue( idx, rgba) for each NumberOfColors entries



# Important vtkProp Subclasses

- vtkLODActor - automated LOD creation
- vtkLODProp3D - manual control of LOD’s including mixed volumes/surfaces
- vtkFollower - always face a camera
- vtkAssembly - groups of vtkProp3D’s, transformed together.

# vtkRenderWindowInteractor

<u>Key features:</u>

- SetRenderWindow – the single render window to interact with
- Key and mouse bindings (Interactor Style)
- Light Follow Camera (a headlight)
- Picking interaction

# Interactor Style(s)

- Button 1 – rotate
- Button 2 – translate (<Shift> Button 1 for 2-button mouse)
- Button 3 – zoom
- Keypress e or q – exit
- Keypress f – “fly-to” point under mouse
- Keypress s/w – surface/wireframe
- Keypress p – pick
- Keypress r – reset camera
- Keypress 3 – toggle stereo

***Switch styles: Keypress j – joystick; t - trackball style***

# Picking

- vtkPropPicker - *hardware-assisted* picking of vtkProps (returns vtkProp picked and x,y,z coordinate)
- vtkWorldPointPicker - get x-y-z coordinate; does not pick prop (*hardware assisted*, returns x,y,z coordinate)
- vtkPicker - pick based on prop3D’s bounding box (*software* ray cast – returns vtkProp)
- vtkPointPicker - pick points (closest point to camera within tolerance - *software* ray cast – returns point id & x,y,z coordinate)
- vtkCellPicker - pick cells (*software* ray cast – returns cell id & x,y,z)

# Example: Picking and Style

```
vtkRenderWindowInteractor *iren =
  vtkRenderWindowInteractor::New();
vtkInteractorStyleFlight *style =
  vtkInteractorStyleFlight::New();
vtkCellPicker *picker = vtkCellPicker::New();

iren->SetInteractorStyle(style);
iren->SetPicker(picker);
```

*(Note: defaults are automatically created, you rarely ever need to do this)*

# 3D Widgets

- Added since VTK 4.0 release
  - Requires version 4.2 or later

- Subclass of vtkInteractorObserver
  - Interactor observers watch events invoked on vtkRenderWindowInteractor
  - Events are caught and acted on
  - Events can be prioritized and ordered
  - The handling of a particular event can be aborted

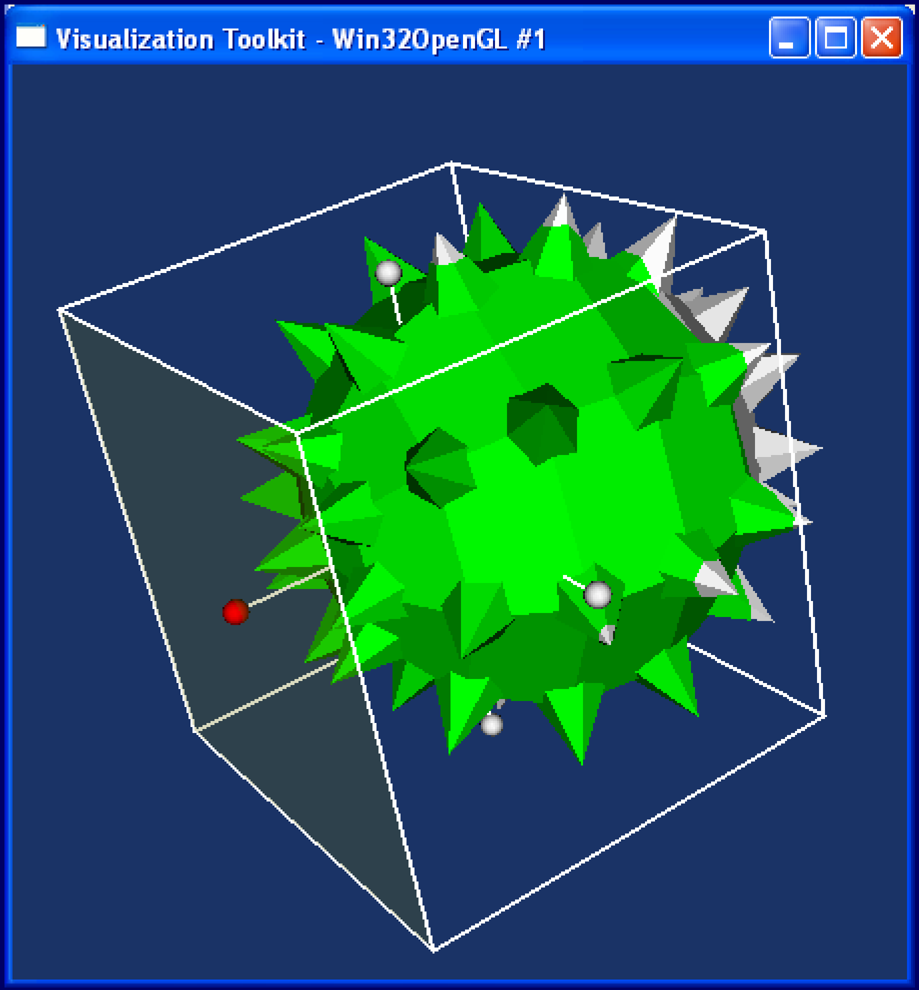
# Some 3D Widgets

- vtkPointWidget
- vtkLineWidget
- vtkPlaneWidget
- vtkImplicitPlaneWidget
- vtkImagePlaneWidget
- vtkBoxWidget
- vtkSphereWidget
- …..

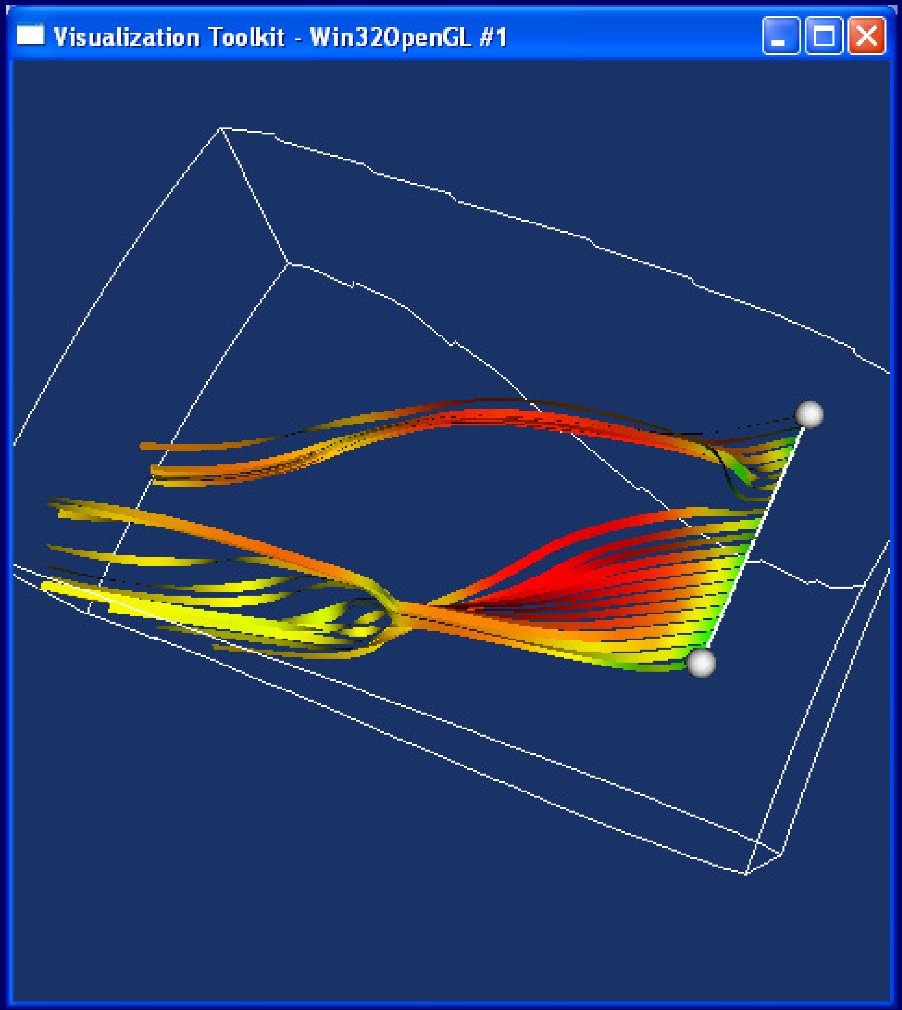
- Widgets often provide auxiliary functionality (e.g., obtaining transforms, polydata, implicit functions, etc.)

- More than one widget at a time can be used

# vtkBoxWidget



# vtkLineWidget



# Example Usage

```
vtkLineWidget lineWidget
vtkPolyData seeds
lineWidget SetInput [pl3d GetOutput]
lineWidget SetAlignToXAxis
lineWidget PlaceWidget
lineWidget GetPolyData seeds

…….
………

lineWidget SetInteractor iren
lineWidget AddObserver StartInteractionEvent
BeginInteraction
lineWidget AddObserver InteractionEvent
GenerateStreamlines
```

# Visualization Pipeline Topics

- Interpreters
- Visualization Model
- Pipeline Mechanics
- Data Management
- Start, End, & Progress Events
- Surface Rendering
- Volume Rendering

# Interpreters

VTK provides automatic wrapping for the following interpreted languages:

- Tcl
- Java
- Python

***Interpreters*** *provide faster turn-around (no compilation) but suffer from slower execution*

# Tcl Interpreter

To use VTK from Tcl, add the following line to the beginning of your script:

```
package require vtk
```

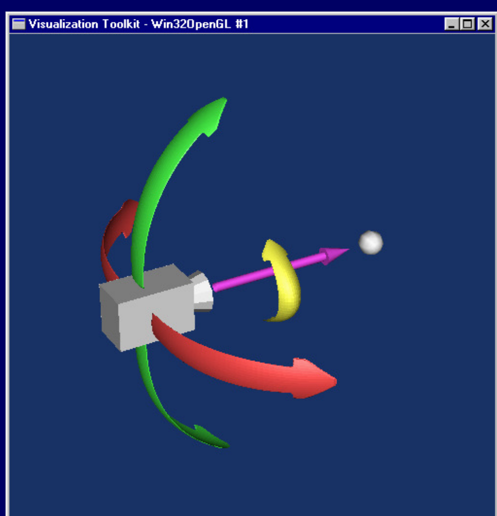
Create an actor in Tcl: `vtkActor actor`

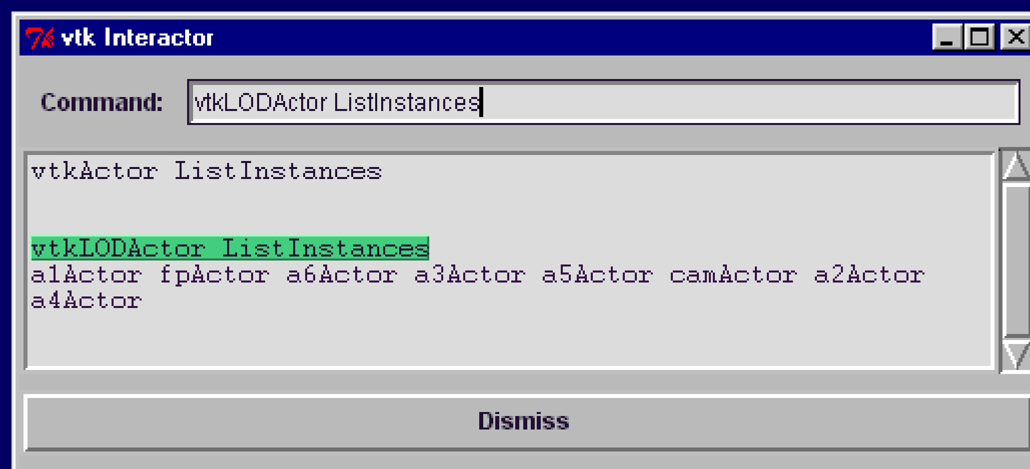
Invoke a method: `actor SetPosition 10 20 30`

# Tcl Interpreter

A special package provides a Tcl interpreter when the ‘u’ key is pressed in the render window:

```
package require vtkinteraction
iren AddObserver UserEvent {wm deiconify .vtkInteract}
```



# Tcl Interpreter

`vtkActor ListInstances`: list all vtkActor objects

`vtkActor ListMethods`: list all vtkActor methods

`anActor Print`: print internal state of anActor

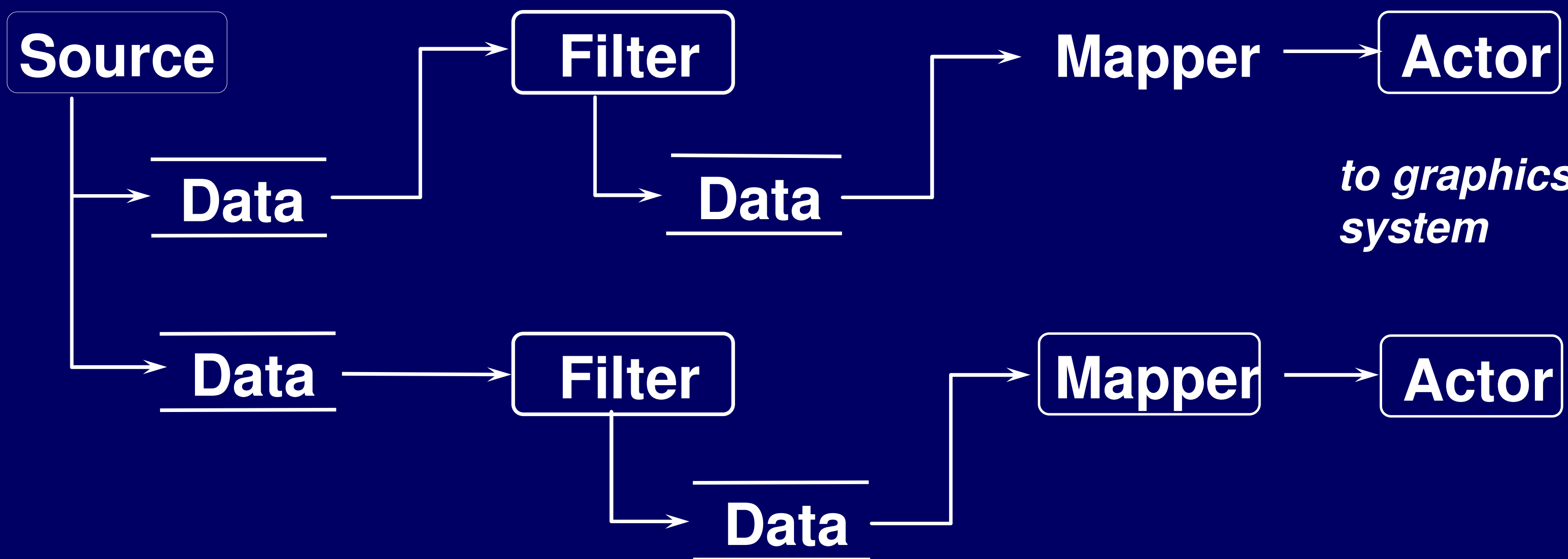
`vtkCommand DeleteAllObjects`: delete all VTK objects

`vtkTkRenderWidget`: embed a render window in Tk

`vtkTkImageViewerWidget`: embed an image window in Tk

# The Visualization Pipeline

A sequence of *algorithms* that operate on *data objects* to generate geometry that can be rendered by the graphics engine or written to a file



# Visualization Model

- Data Objects
  - represent data
  - provide access to data
  - compute information particular to data (e.g., bounding box, derivatives)

- Algorithms
  - Ingest, transform, and output data objects

# vtkDataObject / vtkDataSet

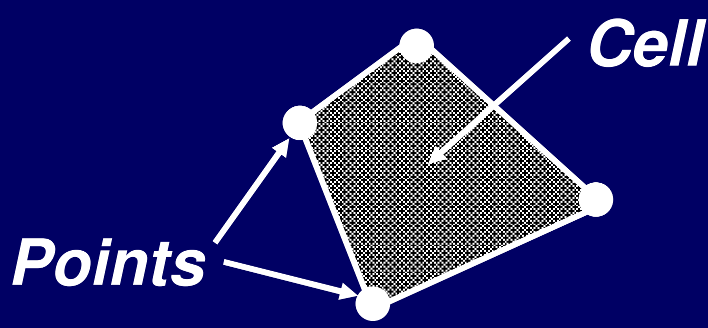
- vtkDataObject represents a “blob” of data
  - contain instance of vtkFieldData
  - an array of arrays
  - no geometric/topological structure
  - Superclass of all VTK data objects
- vtkDataSet has geometric/topological structure
  - Consists of geometry (points) and topology (cells)
  - Has associated point- and cell-centered data arrays
  - Convert data object to data set with vtkDataObjectToDataSetFilter

# vtkDataObject / vtkDataSet

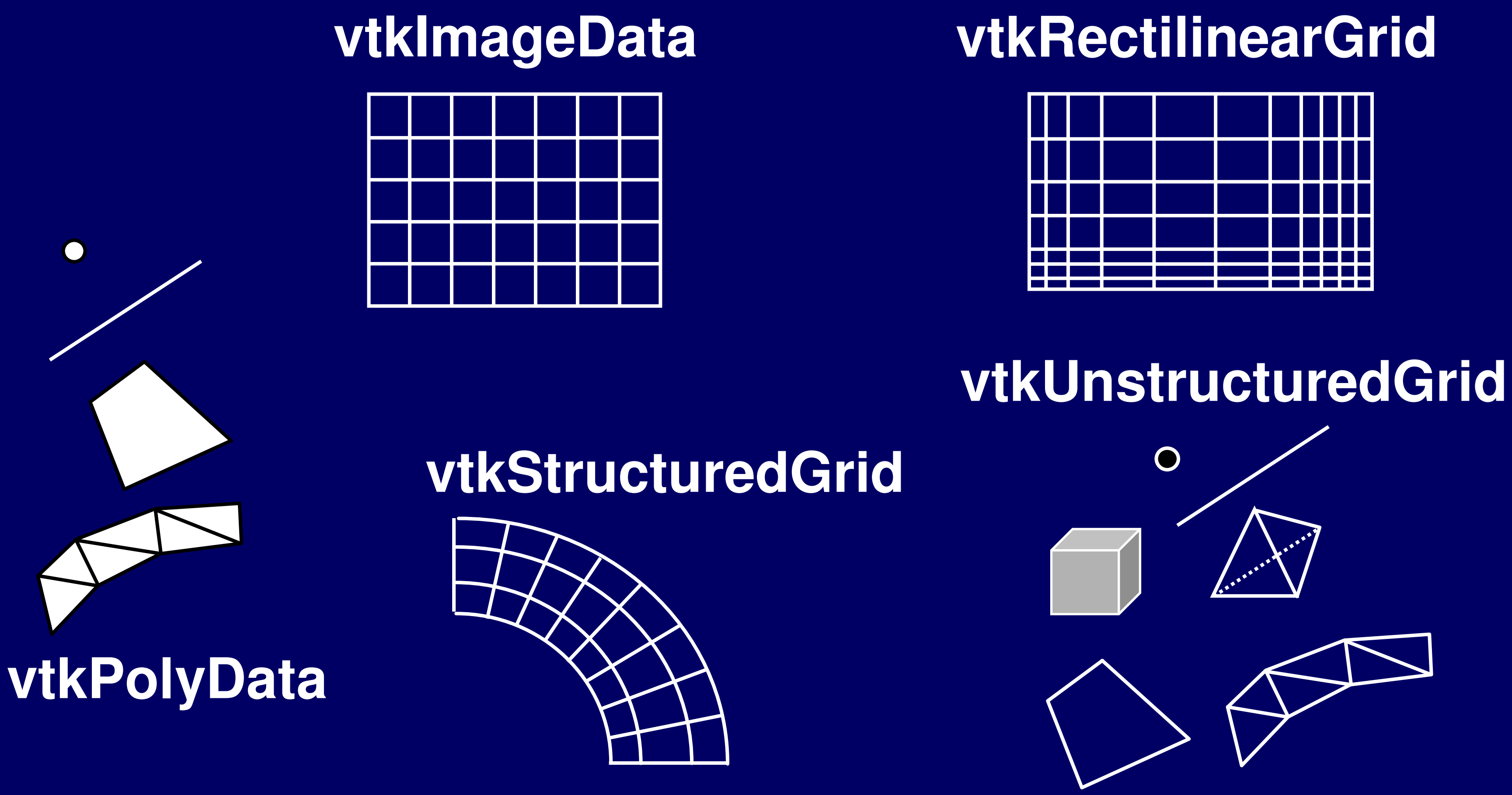
| | Concept | | Implementation |
|---|---|---|---|
| ***vtkDataObject*** | Array of arrays | → | Field Data |
| ***vtkDataSet*** | Geometry & Topology<br>Point- and cell-centered arrays | → | Points & Cells<br>Point Data Cell Data |

# Dataset Model

- A dataset is a data object with structure
- Dataset structure consists of
  - points (x-y-z coordinates)
  - cells (e.g., polygons, lines, voxels) which are defined by connectivity list referring to points ids
  - Access is via integer ID
  - implicit representations
  - explicit representations



# vtkDataSet Subclasses



# Data Set Attributes

- **vtkDataSet also has point and cell attribute data:**
  - **Scalars**
  - **Vectors** - 3-vector
  - **Tensors** - 3x3 symmetric matrix
  - **Normals** - unit vector
  - **Texture Coordinates** 1-3D
  - Array of arrays (I.e. FieldData)

# Data Set Attributes (cont.)

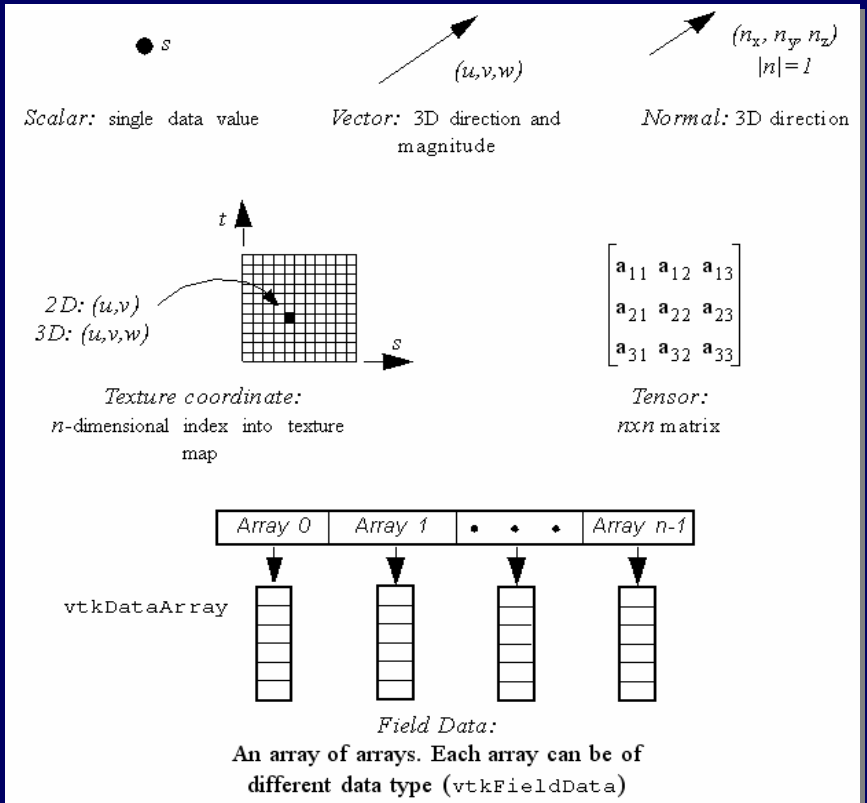
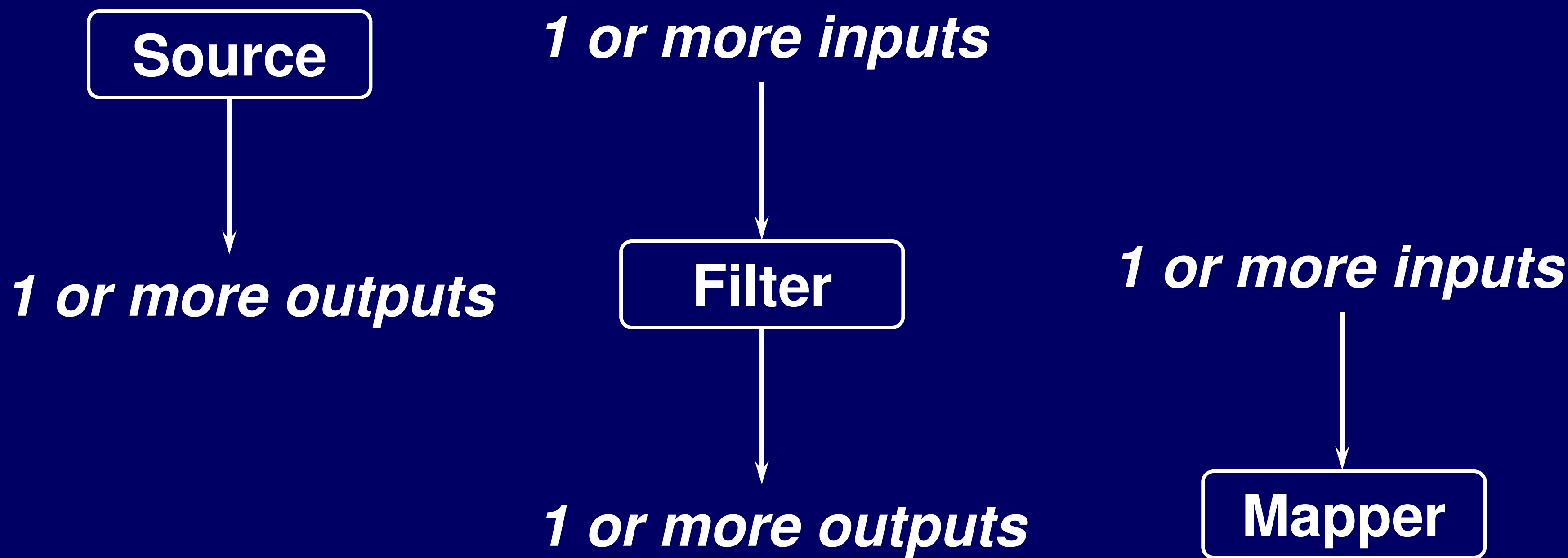



**Figure 3–3** Data attributes associated with the points and cells of a dataset.

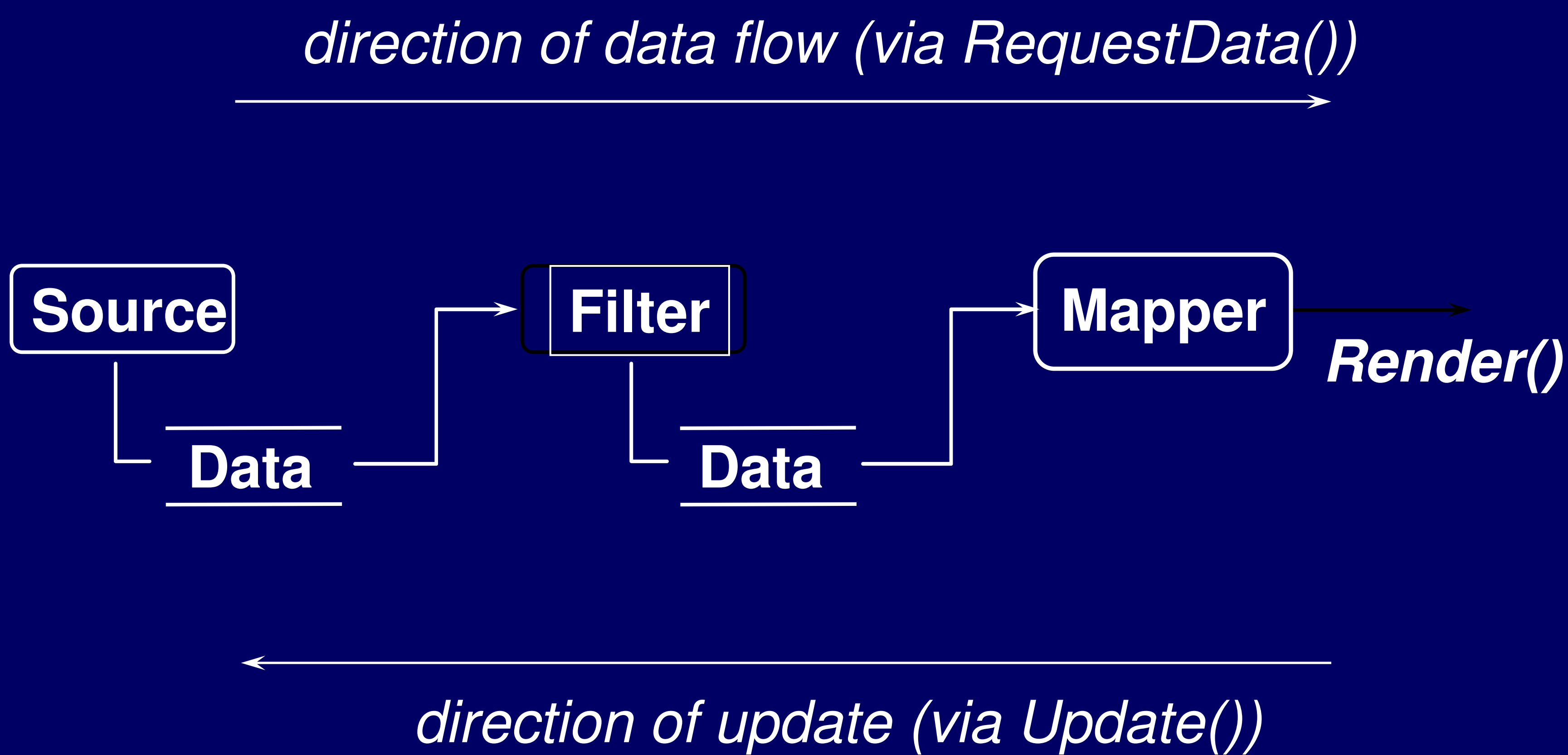
# Algorithms

- Algorithms operate on data objects



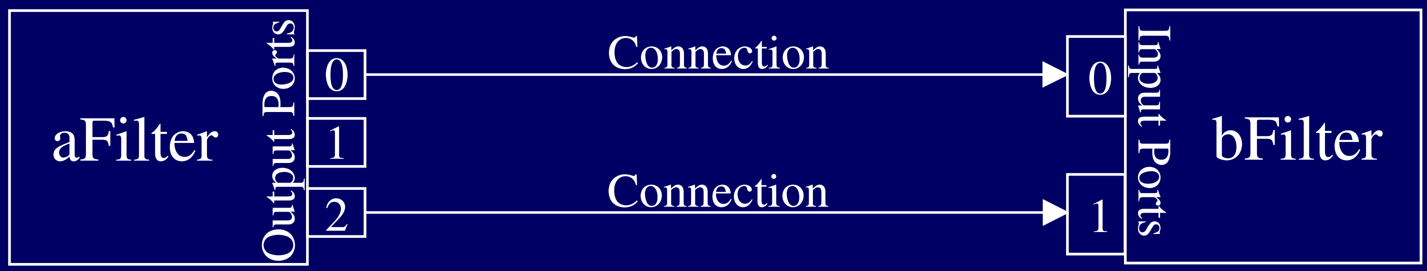
# Pipeline Execution Model

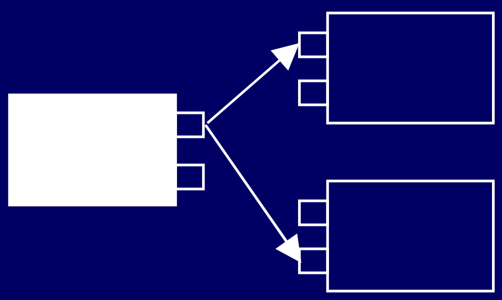
***(conceptual depiction)***



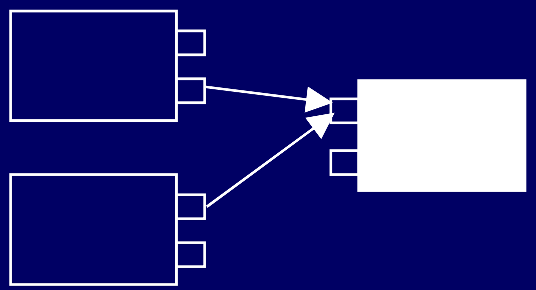
# Creating Pipeline Topology

- bFilter->SetInputConnection(aFilter->GetOutputPort());
- bFilter->SetInputConnection(1,aFilter->GetOutputPort(2));



Reuse an output port: OK

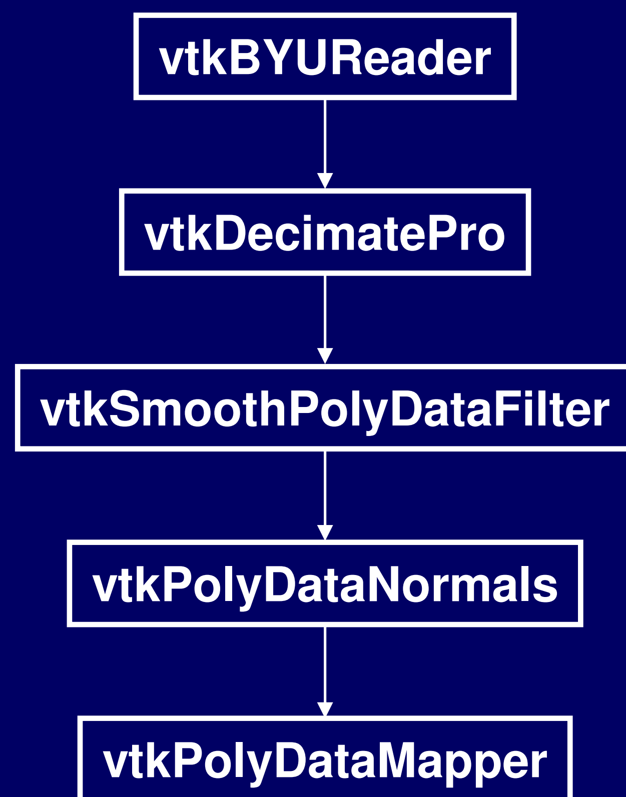


Several connections on an input port: AddInputConnection() if allowed by the filter (ex: vtkAppendFilter)

# Example Pipeline

- Decimation, smoothing, normals
- Implemented in C++

***Note: data objects are not shown → they are implied from the output type of the filter***



# Create Reader & Decimator

```
vtkBYUReader *byu = vtkBYUReader::New();
    byu->
SetGeometryFileName("../../vtkdata/fran_cut.g”);

vtkDecimatePro *deci = vtkDecimatePro::New();
  deci->SetInputConnection( byu->GetOutputPort() );
  deci->SetTargetReduction( 0.9 );
  deci->PreserveTopologyOn();
  deci->SetMaximumError( 0.0002 );
```

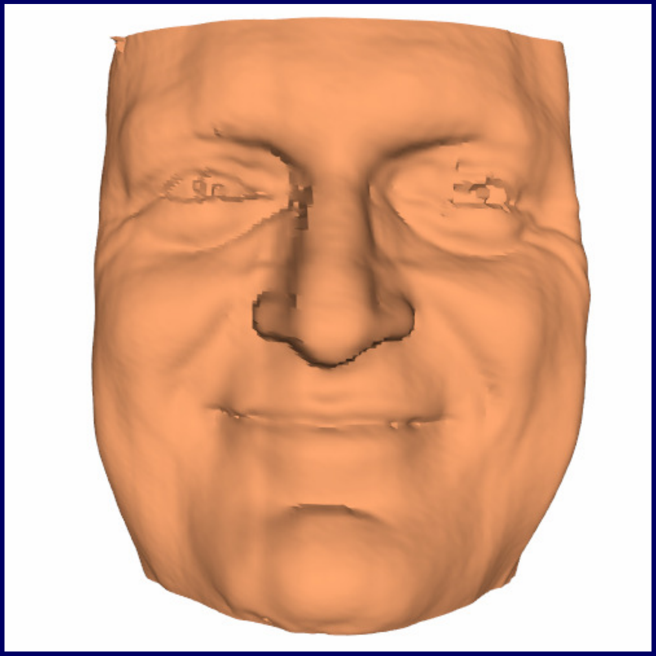
# Smoother & Graphics Objects

```
vtkSmoothPolyDataFilter *smooth = vtkSmoothPolyDataFilter::New();
    smooth->SetInputConnection(deci->GetOutputPort());
    smooth->SetNumberOfIterations( 20 );
    smooth->SetRelaxationFactor( 0.05 );

vtkPolyDataNormals *normals = vtkPolyDataNormals::New();
    normals->SetInputConnection( smooth->GetOutputPort() );

vtkPolyDataMapper *cyberMapper = vtkPolyDataMapper::New();
    cyberMapper->SetInputConnection( normals->GetOutputPort() );

vtkActor *cyberActor = vtkActor::New();
    cyberActor->SetMapper (cyberMapper);
    cyberActor->GetProperty()->SetColor ( 1.0, 0.49, 0.25 );
    cyberActor->GetProperty()->SetRepresentationToWireframe();
```

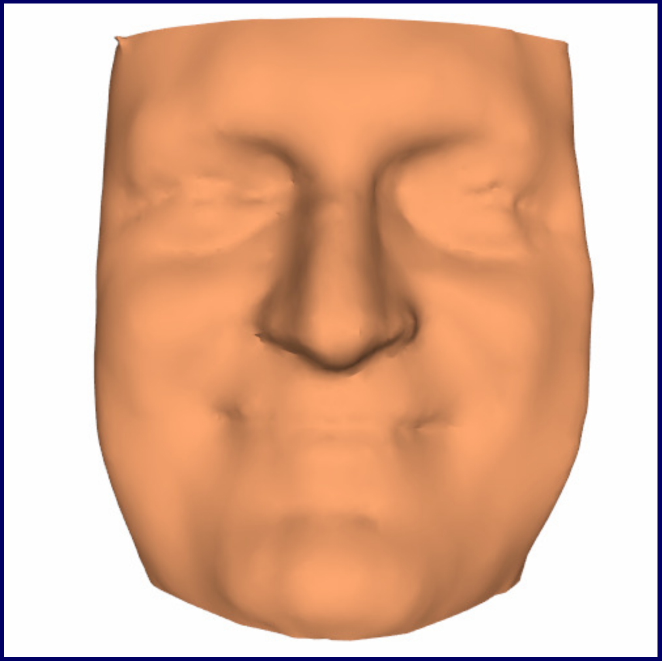
# More Graphics Objects

```
vtkRenderer *ren1 = vtkRenderer::New();

vtkRenderWindow *renWin = vtkRenderWindow::New();
renWin->AddRenderer( ren1 );

vtkRenderWindowInteractor *iren =
              vtkRenderWindowInteractor ::New();
iren->SetRenderWindow( renWin );

ren1->AddViewProp( cyberActor );
ren1->SetBackground( 1, 1, 1 );
renWin->SetSize( 500, 500 );

iren->Start();
```

# Results



Before
(52,260 triangles)



After Decimation
and Smoothing
(7,477 triangles)

# Volume Rendering

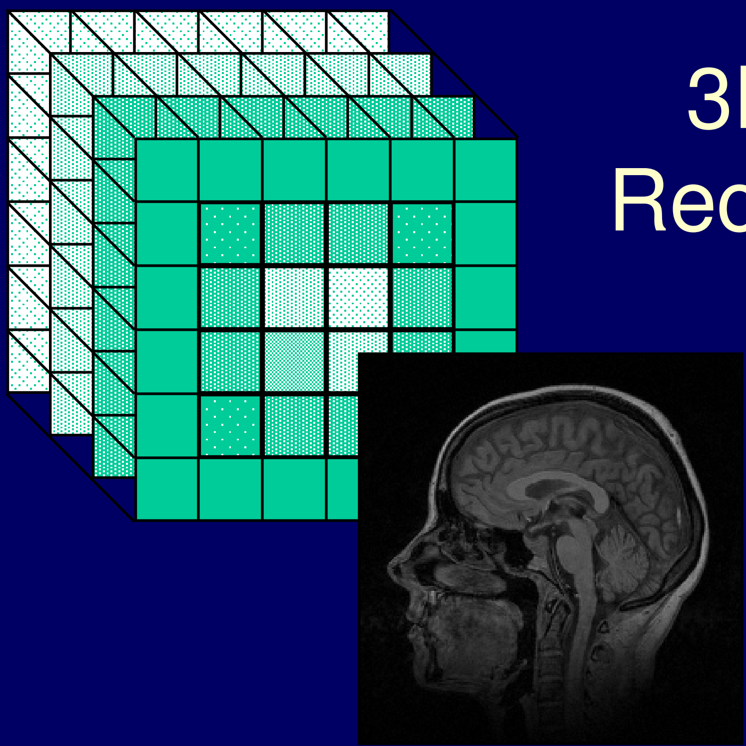
Volume rendering is the process of generating a 2D image from 3D data.

The line between volume rendering and geometric rendering is not always clear. Volume rendering may produce an image of an isosurface, or may employ geometric hardware for rendering.

# Volume Data Structures

- ImageData: 3D regular rectilinear grid
- UnstructuredGrid: explicit list of 3D cells

# 3D Image Data Structure

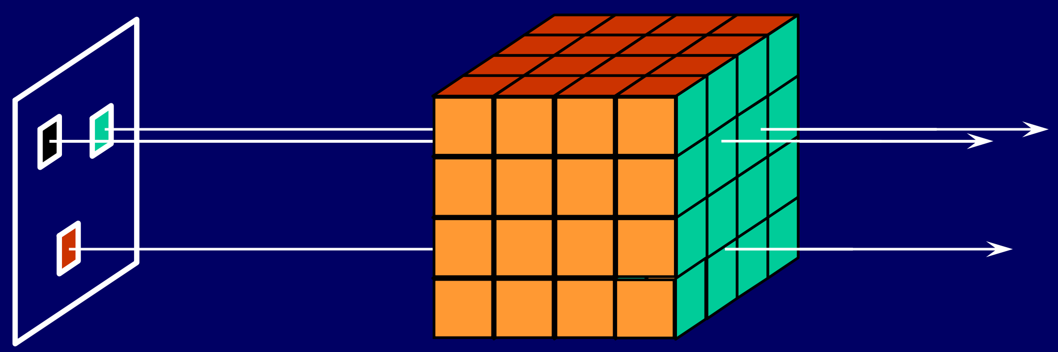


3D Regular
Rectilinear Grid

vtkImageData:
Dimensions = $(D_x, D_y, D_z)$
Spacing = $(S_x, S_y, S_z)$
Origin = $(O_x, O_y, O_z)$

# Volume Rendering Strategies

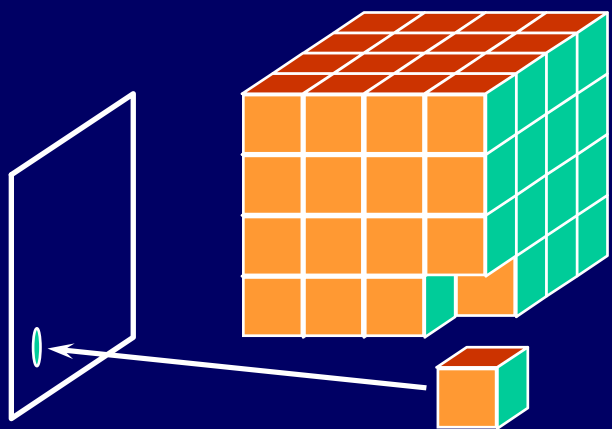
Image-Order Approach: Traverse the image pixel-by-pixel and sample the volume via ray-casting.



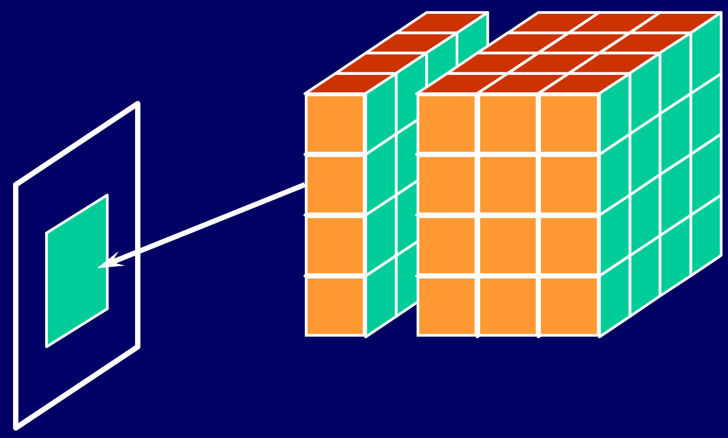
Ray Casting

# Volume Rendering Strategies

Object-Order Approach: Traverse the volume, and project to the image plane.

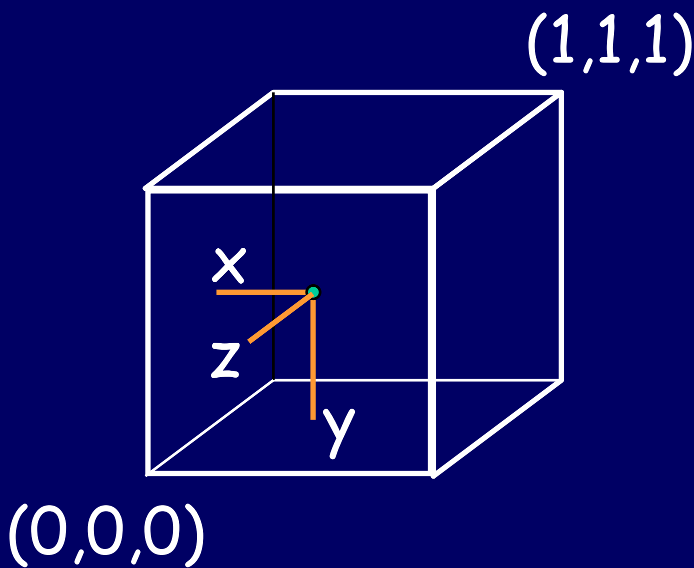


Splatting
cell-by-cell



Texture Mapping
plane-by-plane

# Scalar Value Interpolation



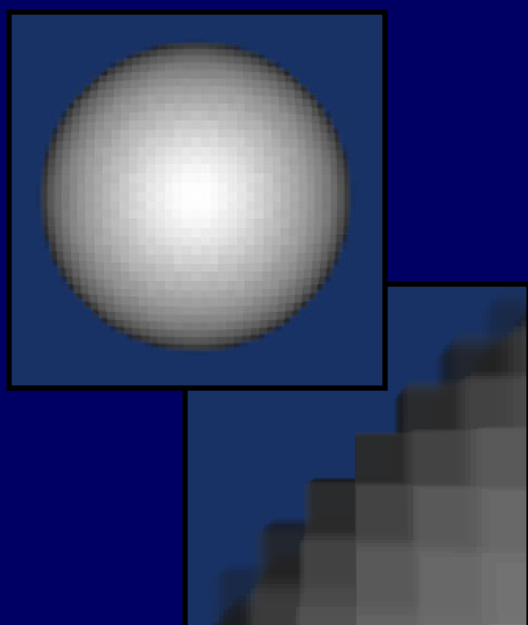

$v = S(rnd(x), rnd(y), rnd(z))$

Nearest Neighbor

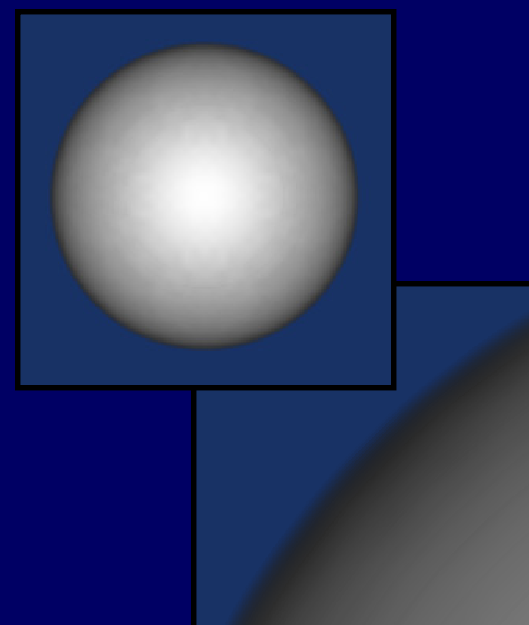
$$
\begin{aligned}
v = {} & (1-x)(1-y)(1-z)S(0,0,0) + \\
& (x)(1-y)(1-z)S(1,0,0) + \\
& (1-x)(y)(1-z)S(0,1,0) + \\
& (x)(y)(1-z)S(1,1,0) + \\
& (1-x)(1-y)(z)S(0,0,1) + \\
& (x)(1-y)(z)S(1,0,1) + \\
& (1-x)(y)(z)S(0,1,1) + \\
& (x)(y)(z)S(1,1,1)
\end{aligned}
$$

Trilinear

# Scalar Value Interpolation



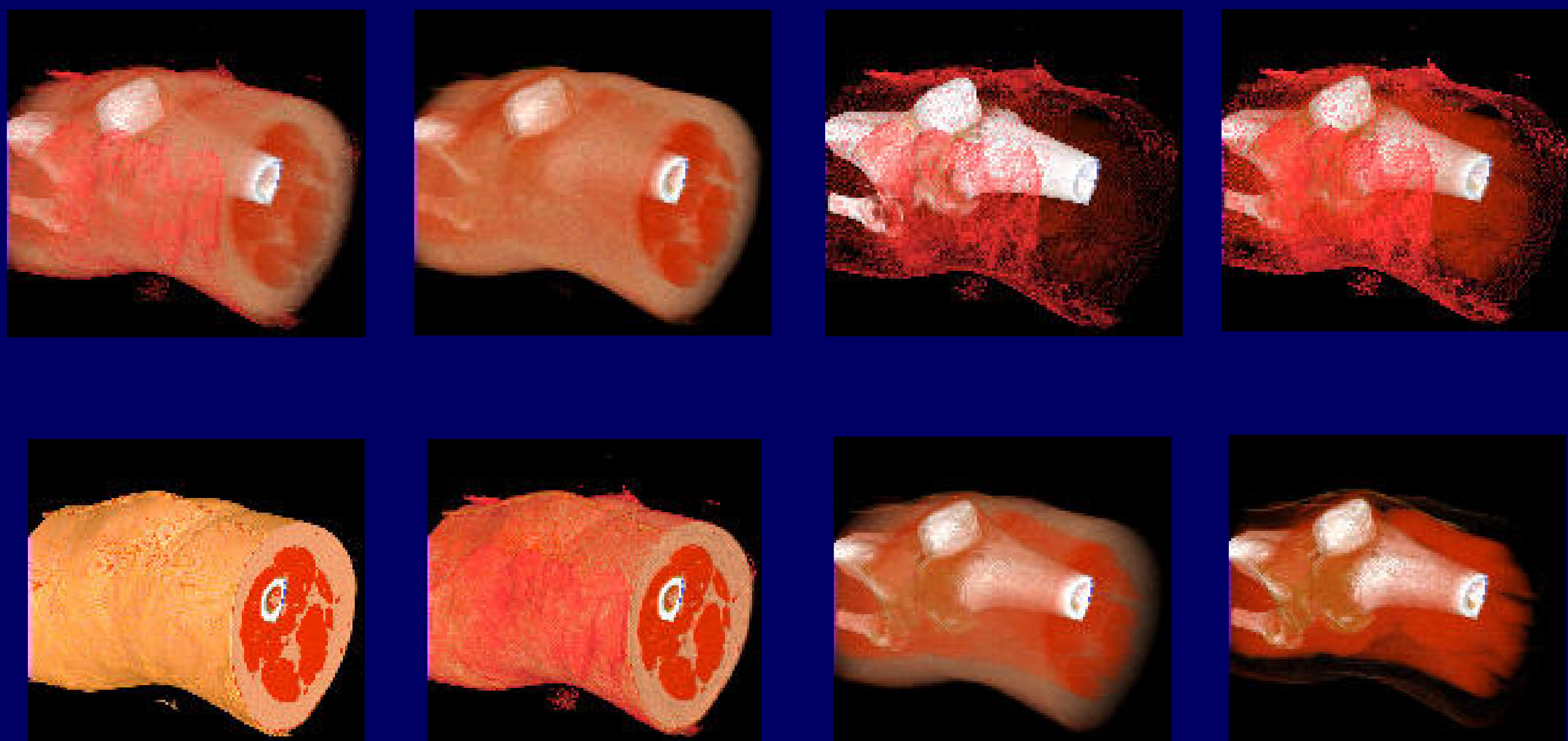
Nearest Neighbor Interpolation



Trilinear Interpolation

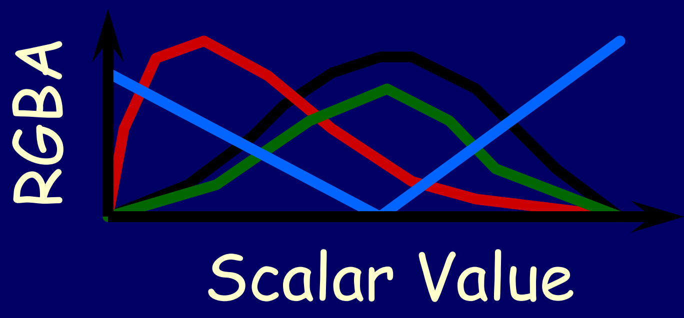
# Material Classification

Transfer functions are the key to volume renderings

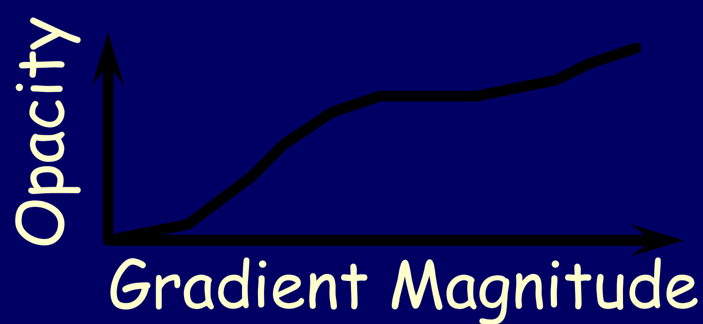


# Material Classification

Scalar value can be classified into color and opacity (RGBA)

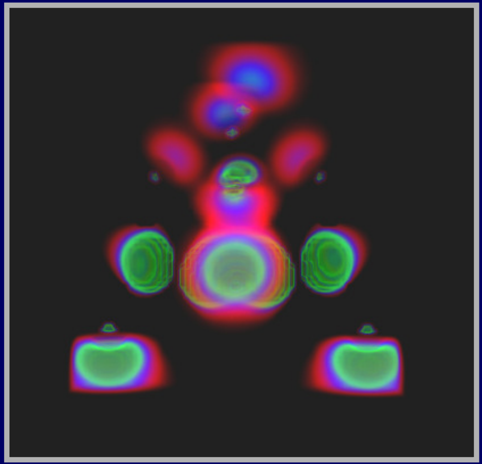


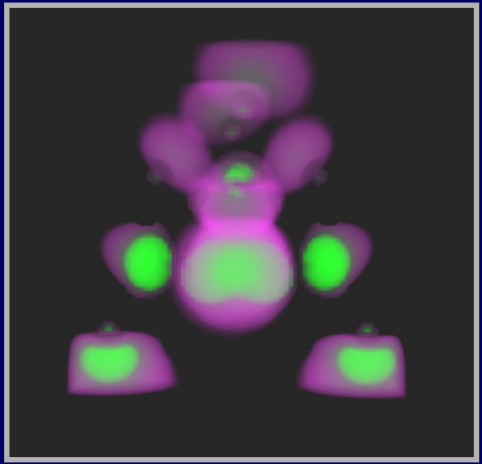

Gradient magnitude can be classified into opacity

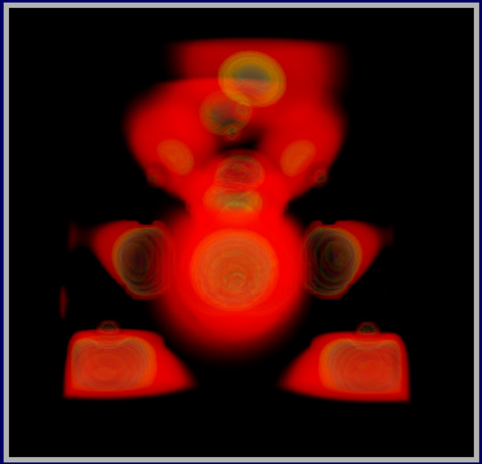


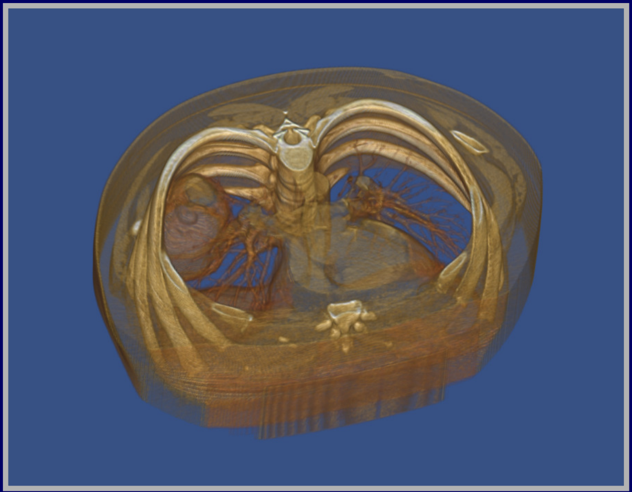

Final opacity is obtained by multiplying scalar value opacity by gradient magnitude opacity

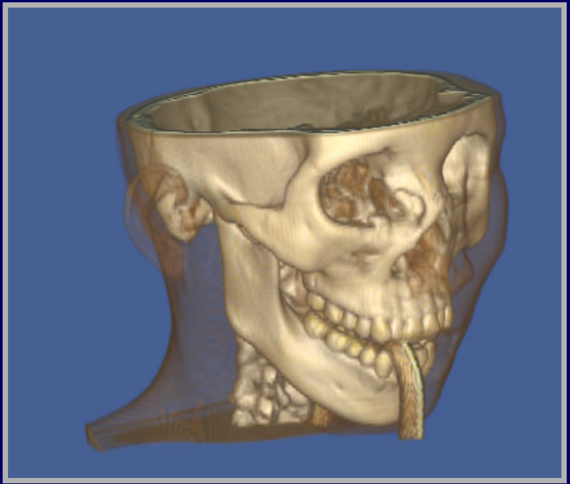
# Material Classification



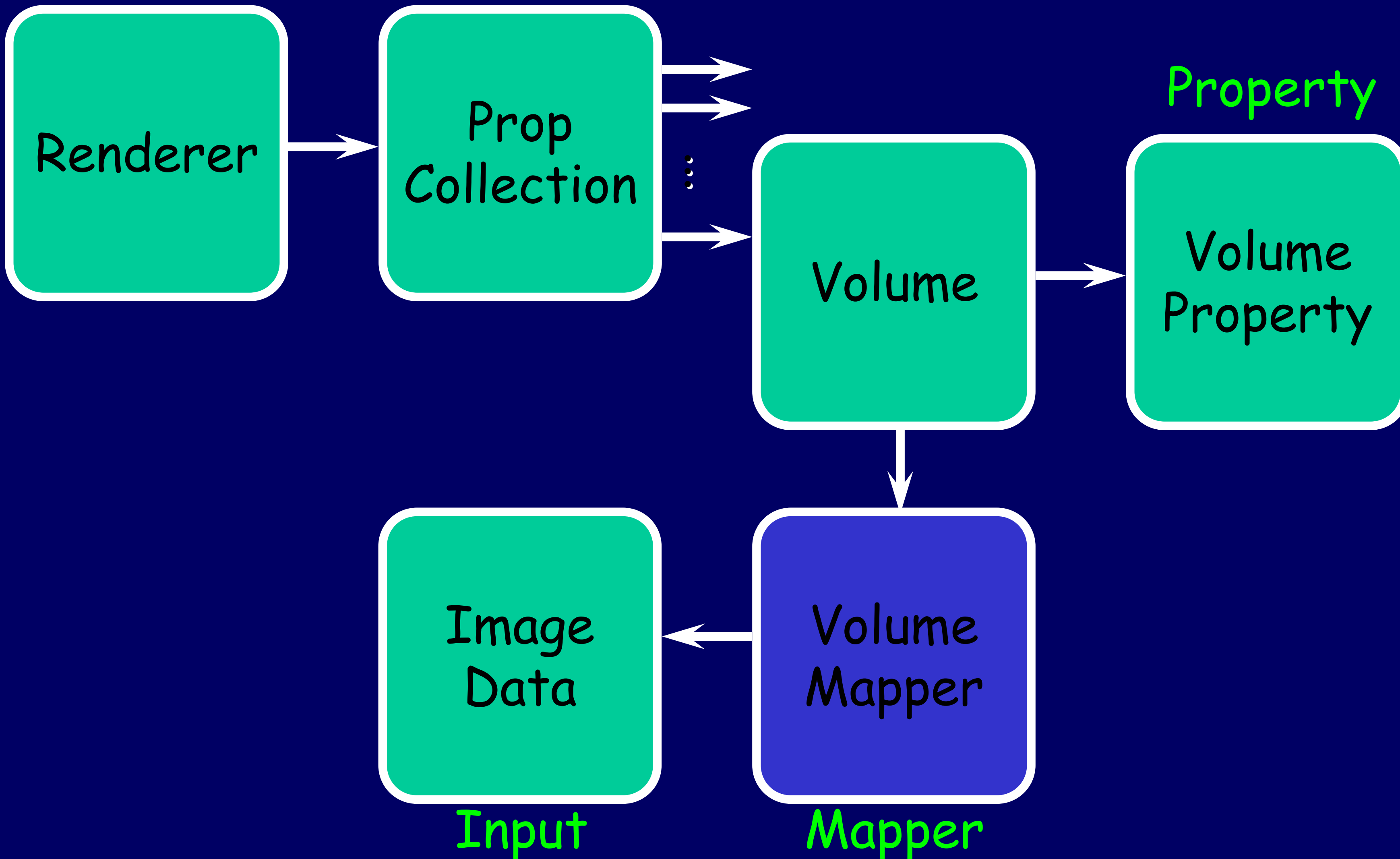




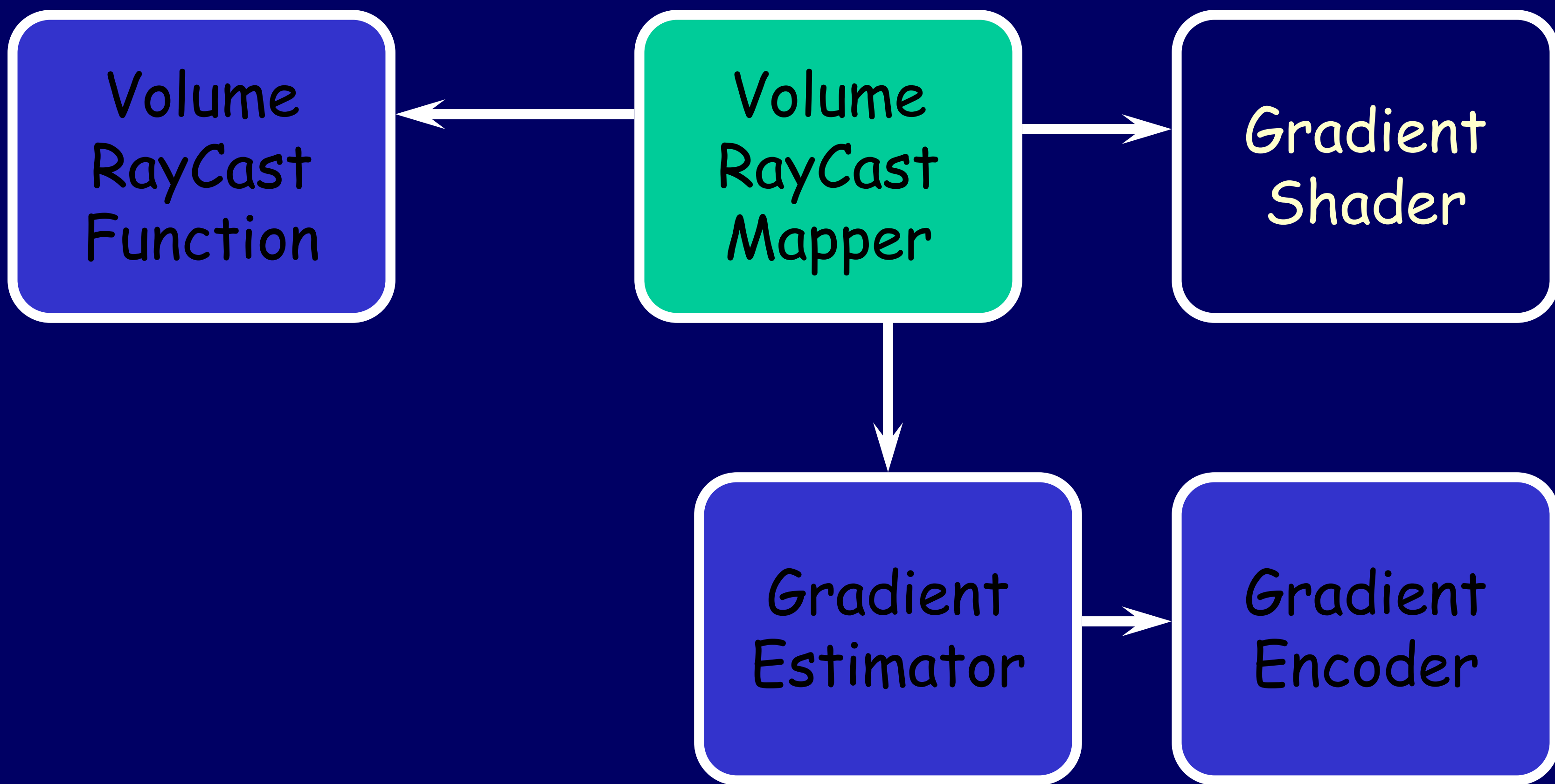




# Implementation

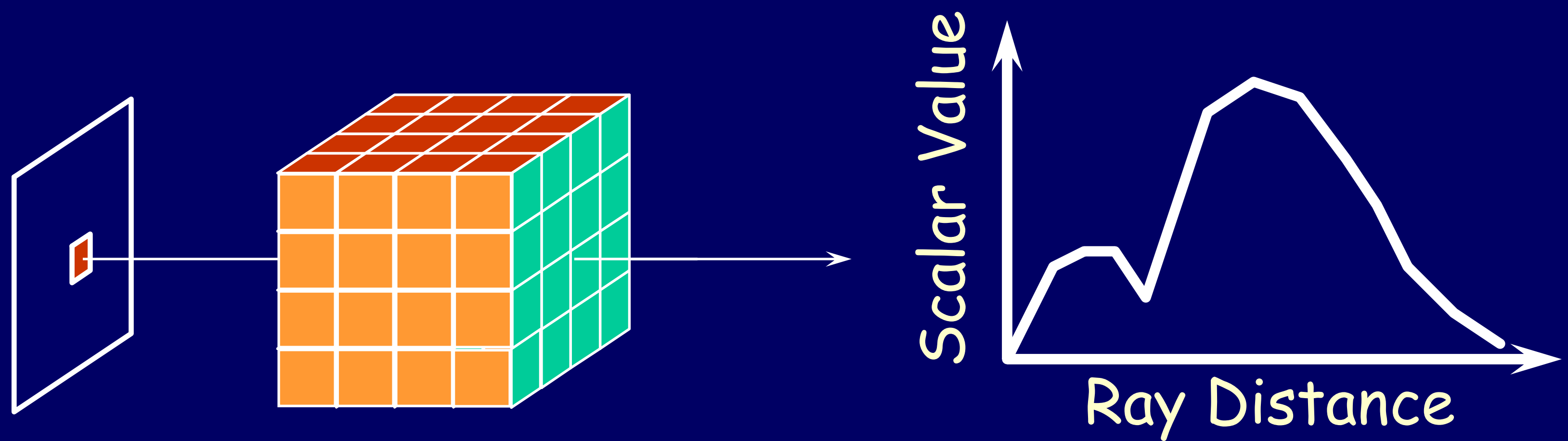


# Volume Ray Casting

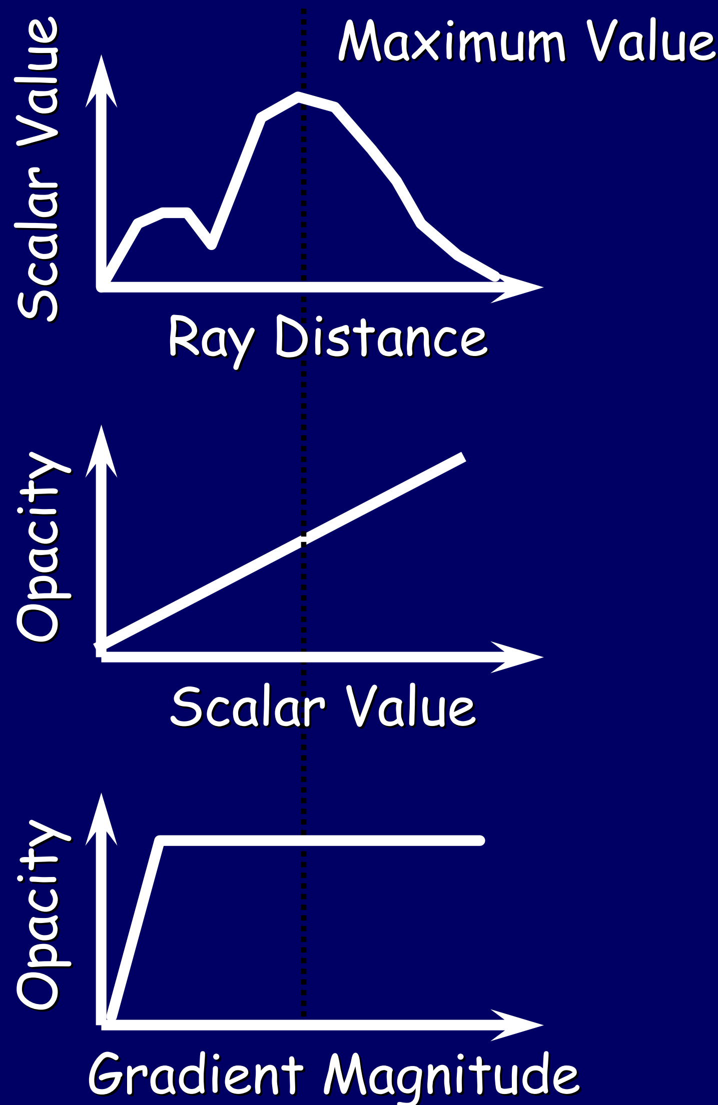


# Ray Cast Functions

A Ray Function examines the scalar values encountered along a ray and produces a final pixel value according to the volume properties and the specific function.

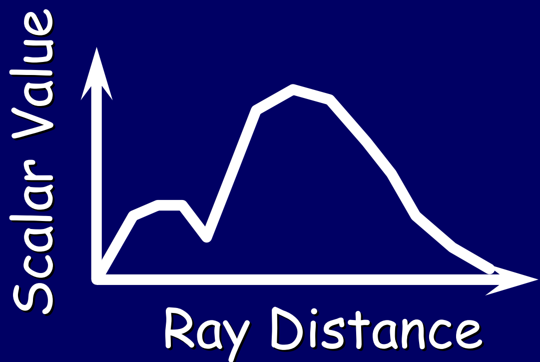


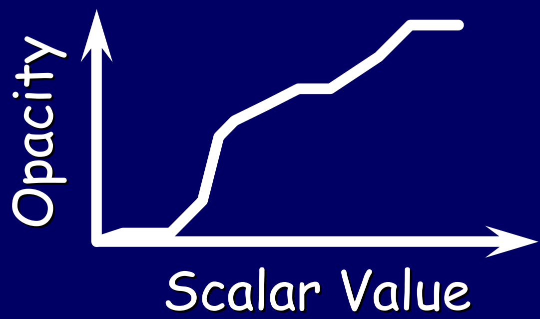
# Maximum Intensity Function

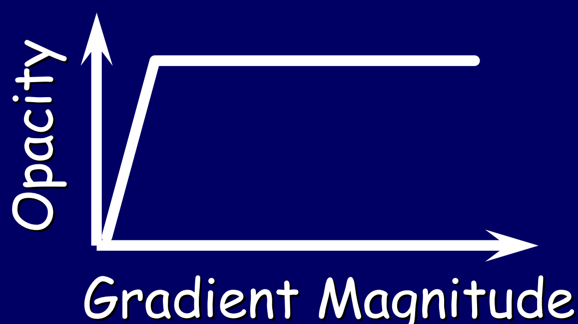



Maximize Scalar Value

# Composite Function



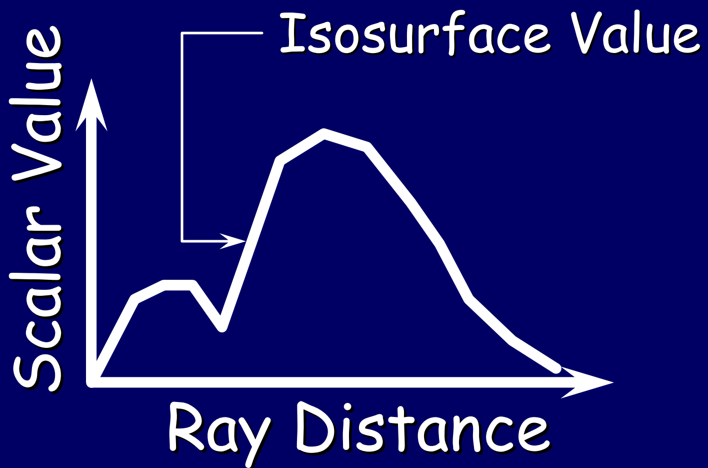



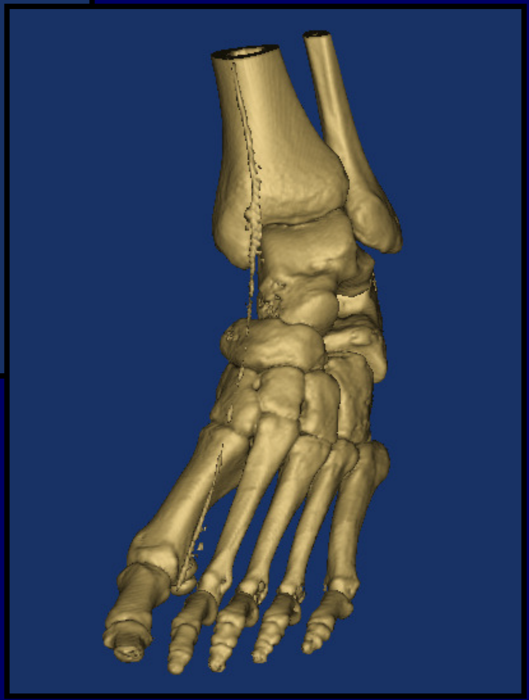

Use $\alpha$-blending along the ray to produce final RGBA value for each pixel.

$$I_i = c_i\, a_i + I_{i+1}\,(1-a_i)$$

# Isosurface Function

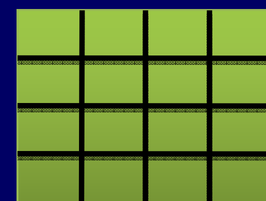


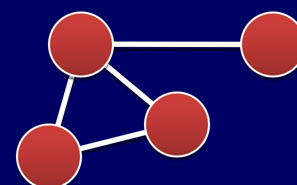

Stop ray traversal at isosurface value. Use cubic equation solver if interpolation is trilinear.

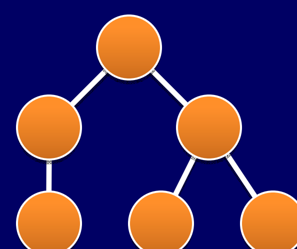


# Information Visualization

- Data Structures

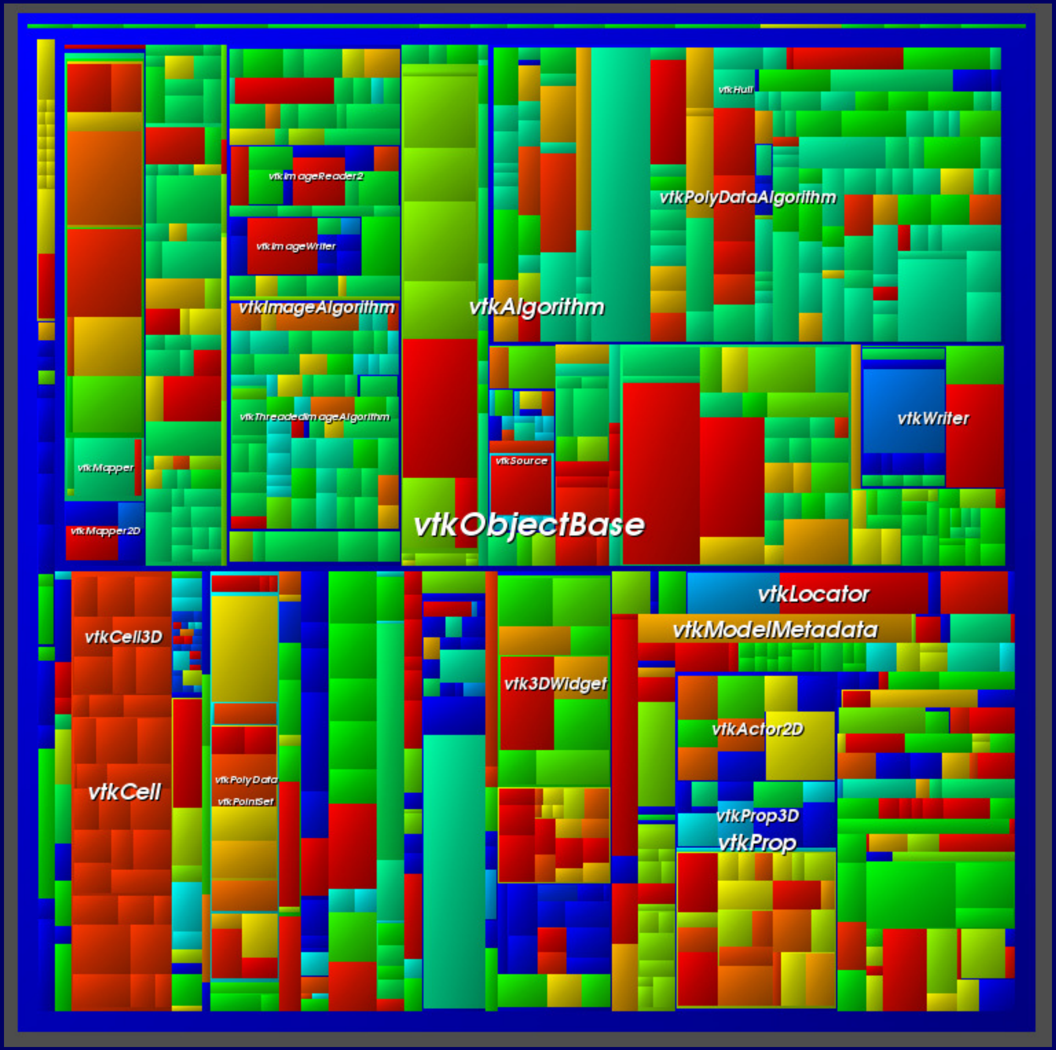
table 

graph (network) 

tree (hierarchy) 

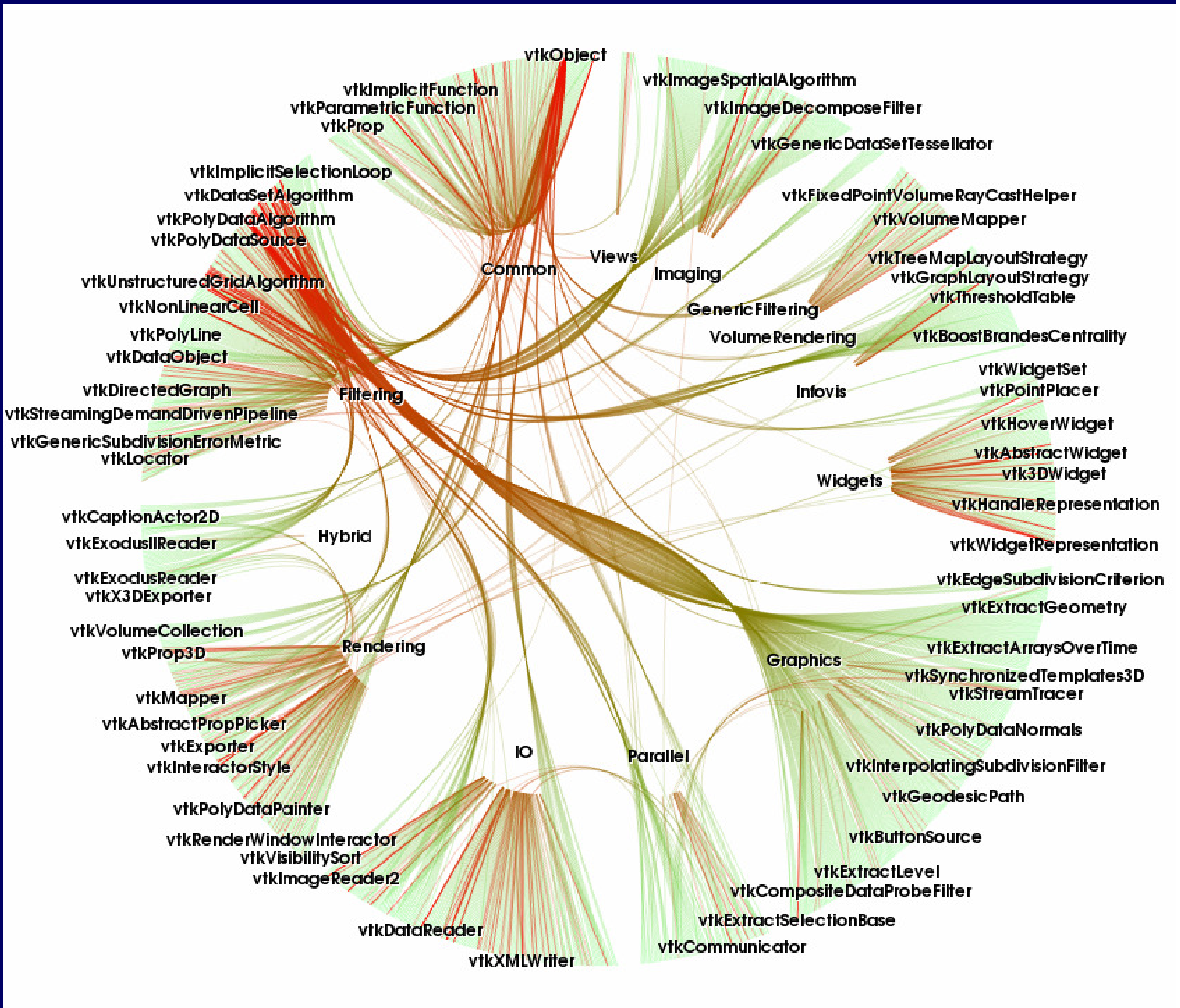
# Views

- Treemap



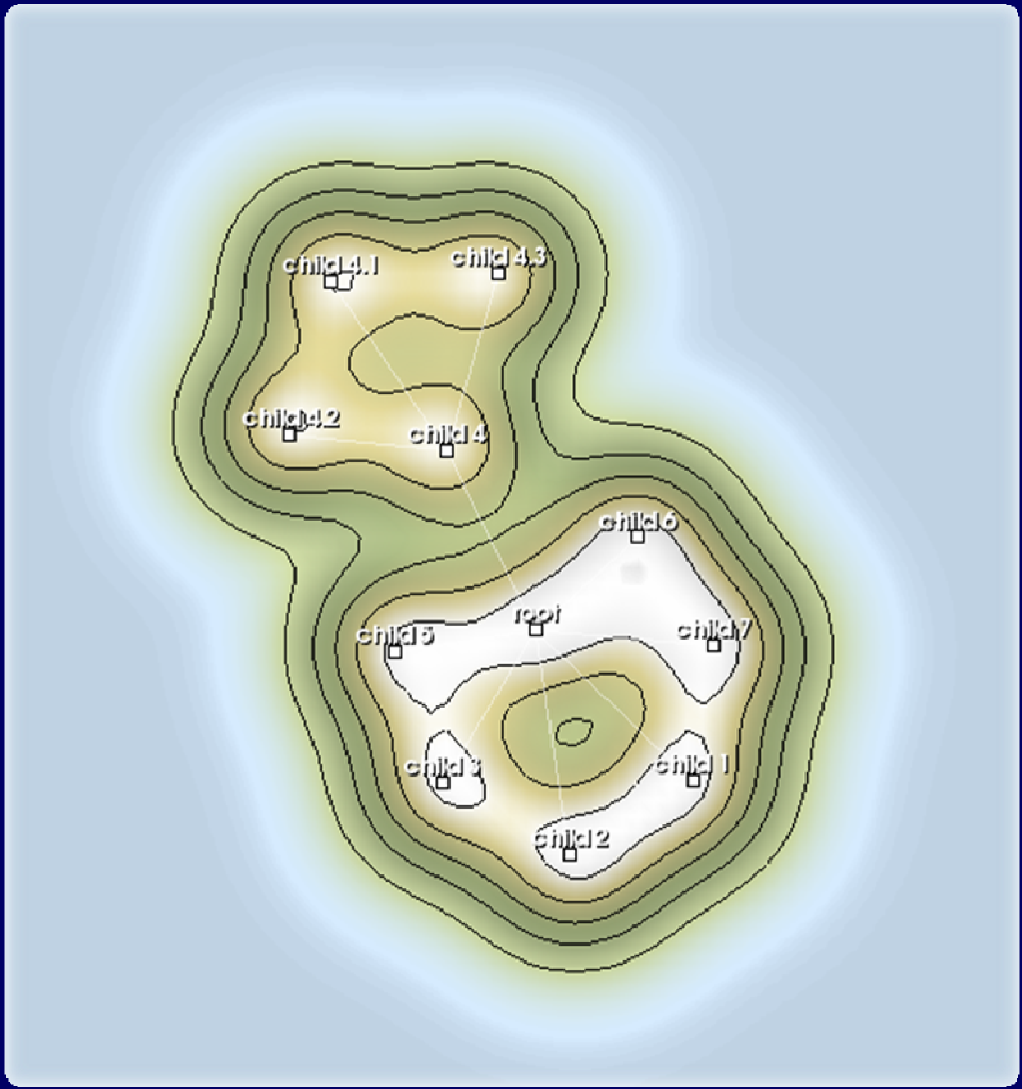
# Views

- Hierarchical bundles



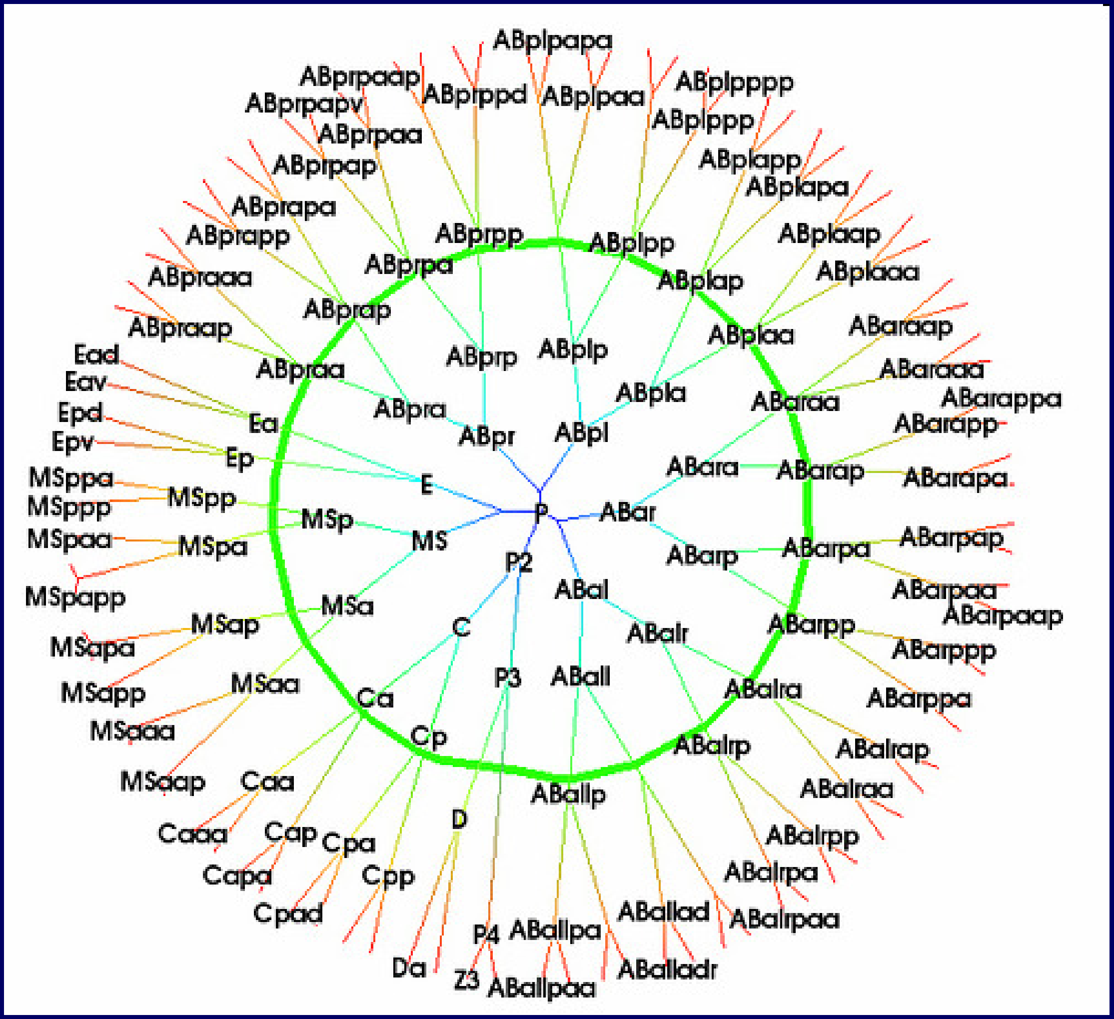
# Views

- Landscapes



# Views

- Radial Layout



# Parallel Graphing