Introduction to Python and VTK

Scientific Visualization, HT 2014
Lecture 2

Johan Nysjö
Centre for Image analysis
Swedish University of Agricultural Sciences
Uppsala University

When you need more speed

- NumPy & SciPy
- Cython (supports parallel processing via OpenMP)
- PyCUDA
- PyOpenCL

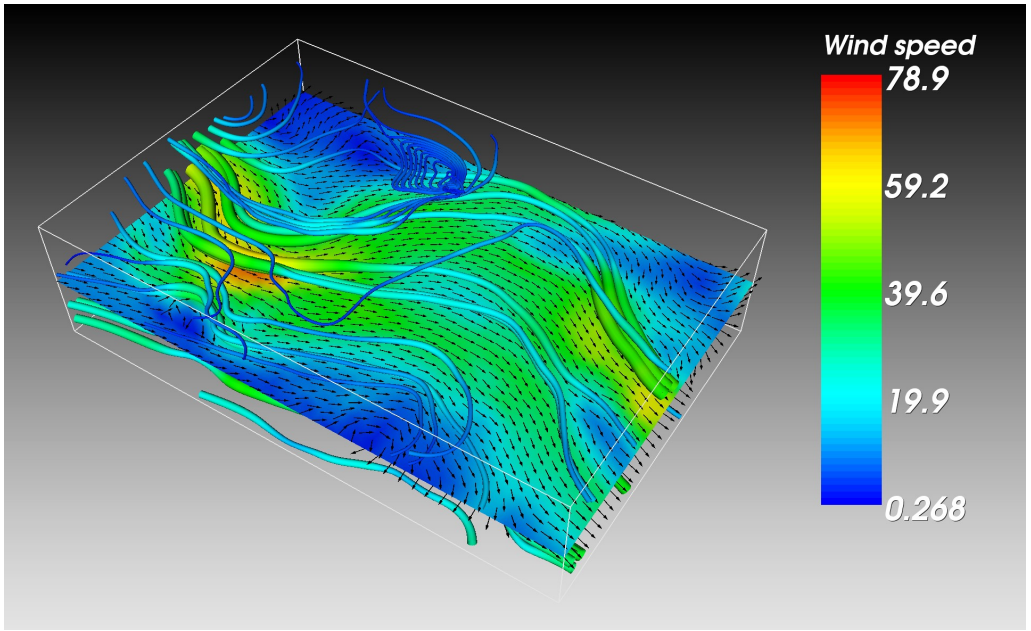
Other useful packages

- Graphics programming and visualization
  - PyOpenGL, VTK, Mayavi
- GUI programming
  - PyQt/PySide, wxPython, Tkinter
- Image analysis and processing
  - ITK, Pillow
- Computer vision
  - OpenCV
- Plotting
  - Matplotlib


vtk

The Visualization Toolkit (VTK)

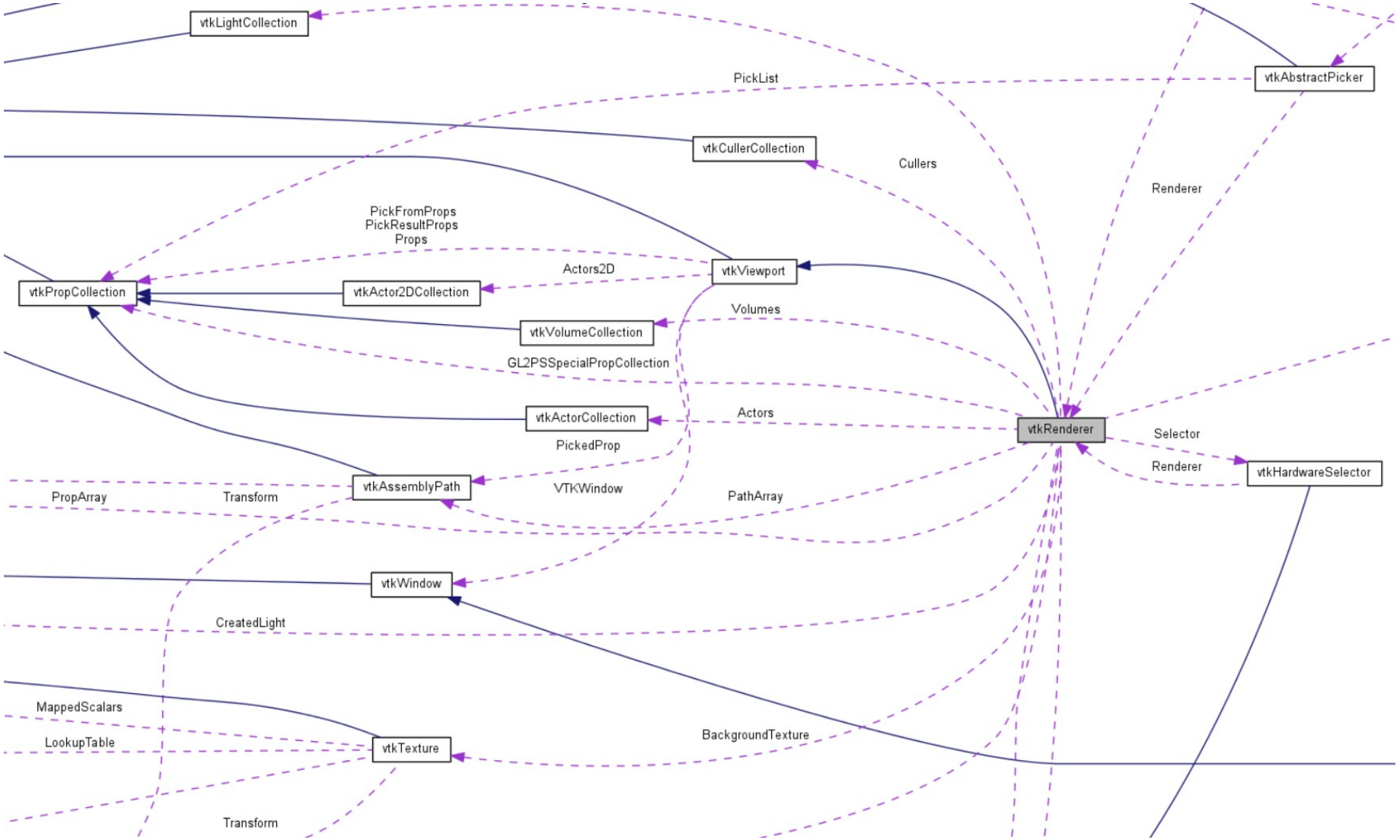
- Open source, freely available C++ toolkit for
  - scientific visualization
  - 3D computer graphics
  - mesh and image processing
- Managed by Kitware Inc.



VTK

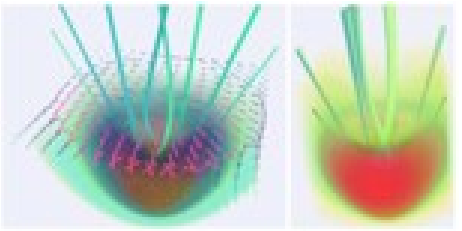
- Object-oriented design
- High level of abstraction (compared to graphics APIs like OpenGL or Direct3D)
- Provides bindings to Tcl/Tk, Python, and Java
- GUI bindings: Qt, wxWidgets, Tkinter, etc

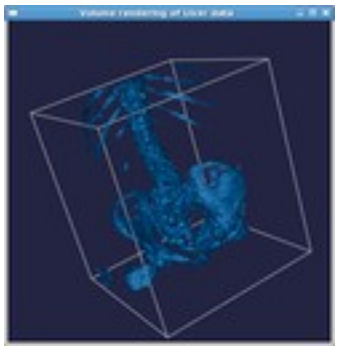
Heavily object-oriented
(and a bit over-designed...)

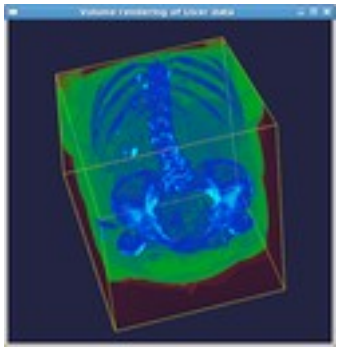


Some examples of what you can do with VTK

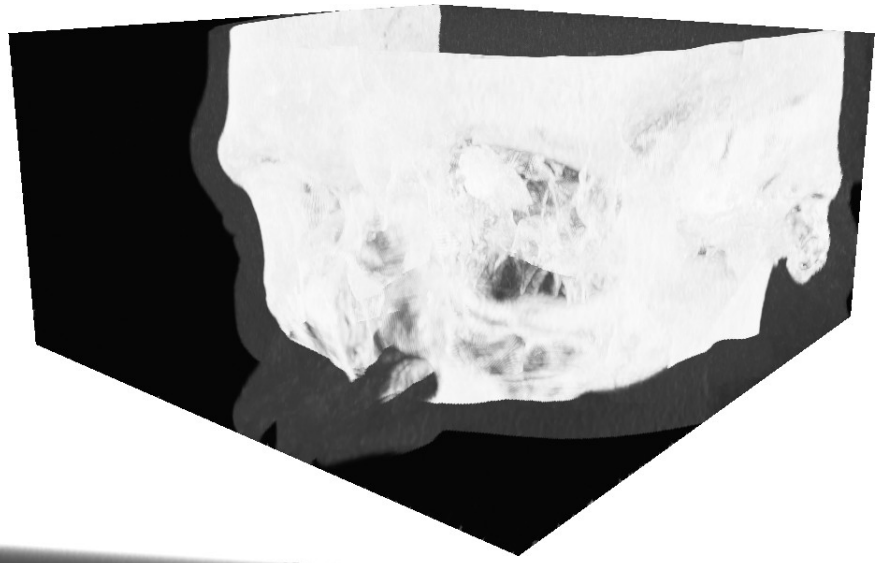
- Create visualizations of
  - scalar, vector, and tensor fields
  - volume data (e.g., 3D CT or MRI scans)
- Mesh and polygon processing
- Image analysis (2D and 3D images)
- Isosurface extraction
- Implementing your own algorithms

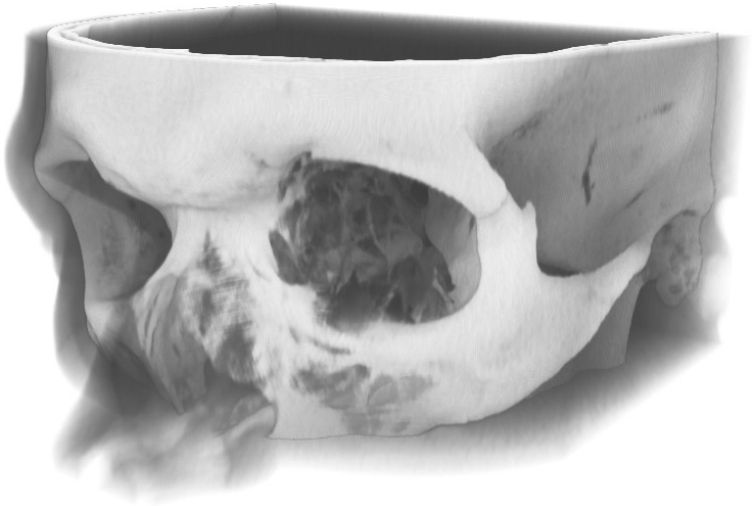


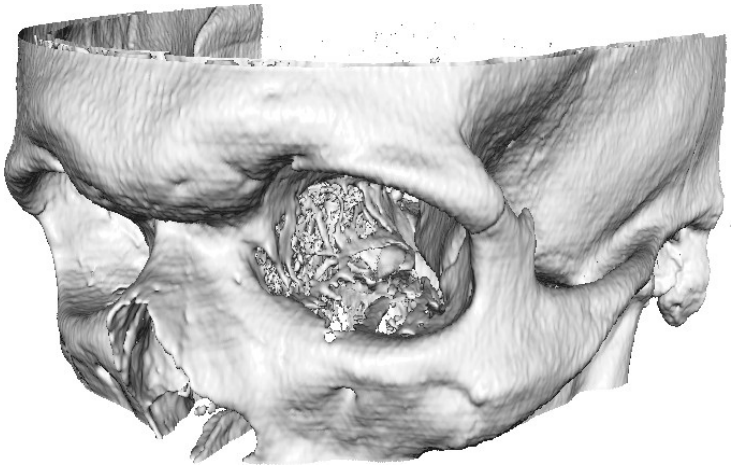




Volume rendering



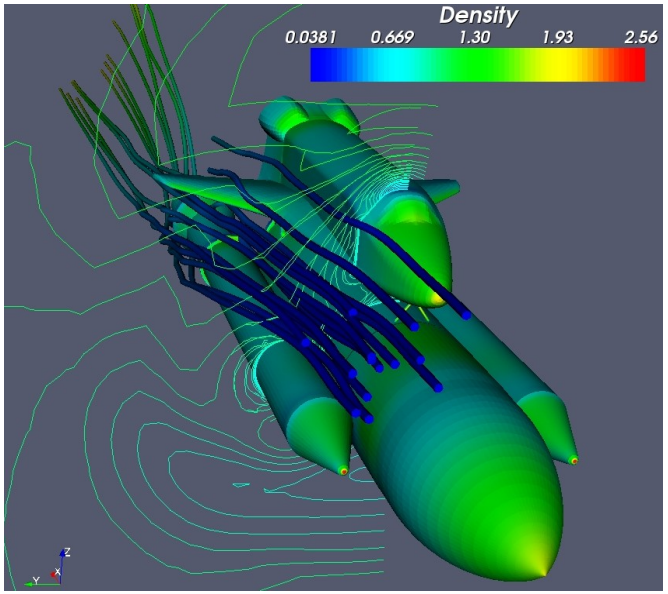




Rendering graphical 3D models (imported from .stl, .ply, .obj, etc)

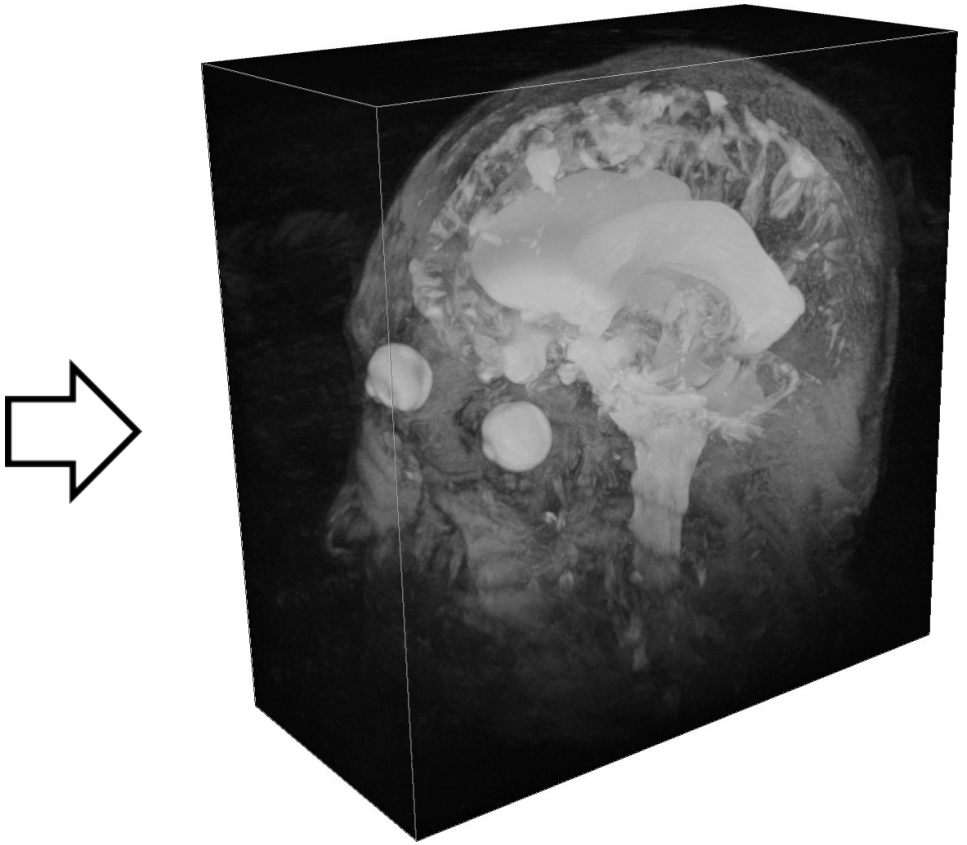
Rendering performance

- VTK has decent rendering performance and is good for rapid prototyping of 3D visualization tools
- Not suitable for rendering large realistic 3D scenes with lots of dynamic content (i.e., games)



The visualization pipeline

```
# vtk DataFile Version 3.0
vtk output
BINARY
DATASET STRUCTURED_POINTS
DIMENSIONS 256 256 124
SPACING 0.9 0.9 0.9
ORIGIN 0 0 0
CELL_DATA 7998075
POINT_DATA 8126464
COLOR_SCALARS ImageFile 1
^D^E^C^G^D^B^D^B^B^C^D^E^D^E^C^C
^D^C^C^C^C^E^D^C^A^B^B^B^F^A^C^E
^D^D^E^A^A^C^B^B^E^B^A^A^E^B^E^E
^A^C^C^G^C^D^F^B^D^E^@^G^C^D^D^C
^D^C^F^C^B^E^E^E^B^C^C^B^C^B^C^B
^C^C^F^E^F^C^D^A^A^C^F^D^D^E^E^B
```

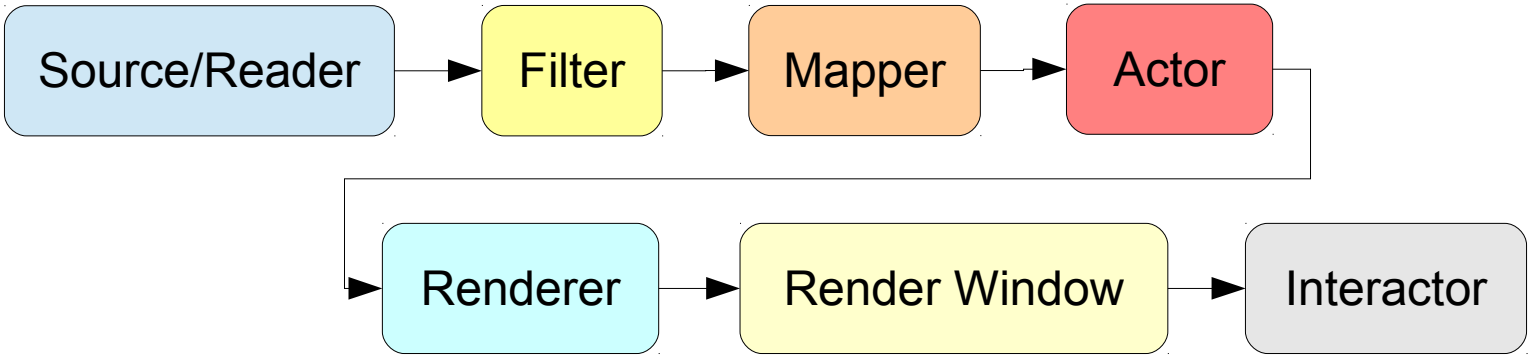


Input data

Visualization

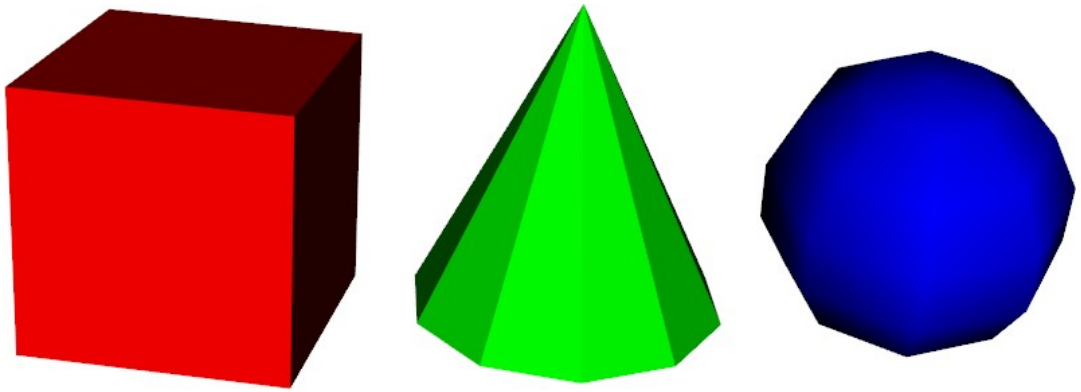
The visualization pipeline

- To visualize your data in VTK, you normally set up a pipeline like this:



Sources

- VTK provides various source classes that can be used to construct simple geometric objects like spheres, cubes, cones, cylinders, etc...
- Examples: **vtkSphereSource**, **vtkCubeSource**, **vtkConeSource**



source/reader → filter → mapper → actor → renderer → renderWindow → interactor

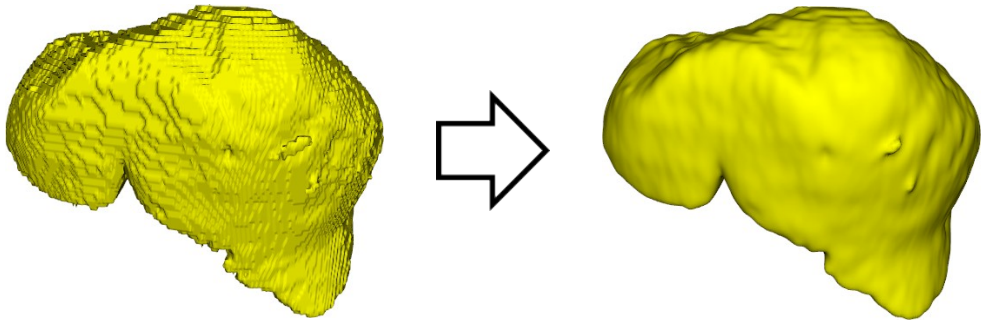
Readers

- Reads data from file
- You can use, e.g., **vtkStructuredPointsReader** to read a volumetric image from a .vtk file
- or **vtkSTLReader** to load a 3D polygon model from a .stl file
- If VTK cannot read your data, write your own reader!

source/reader → filter → mapper → actor → renderer → renderWindow → interactor

Filters

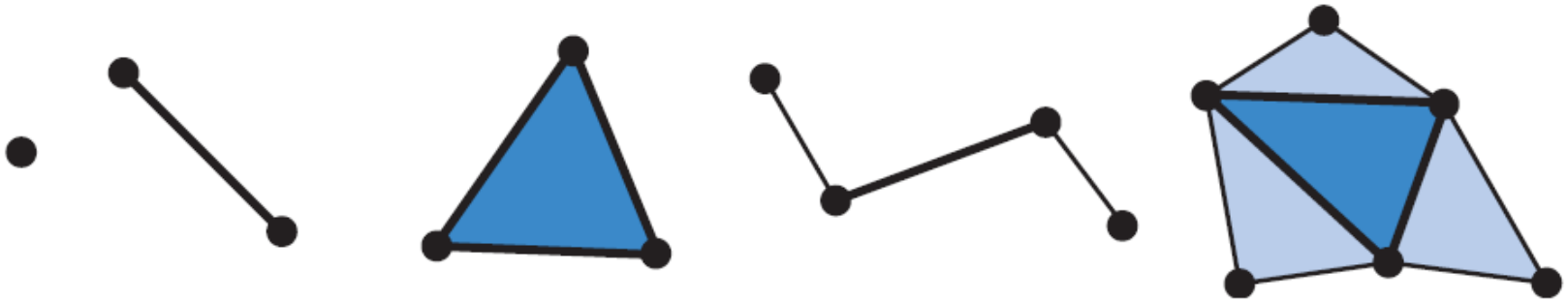
- Takes data as input, modifies it in some way, and returns the modified data
- Can be used to (for example)
  - select data of a particular size, strength, intensity, etc
  - process 2D/3D images or polygon meshes
  - generate geometric objects from data



source/reader → filter → mapper → actor → renderer → renderWindow → interactor

Mappers

- Maps data to graphics primitives (points, lines, or triangles) that can be displayed by the renderer
- The mapper you will use most in the labs is **vtkPolyDataMapper**
- **vtkPolyDataMapper** maps polygonal data (**vtkPolyData**) to graphics primitives

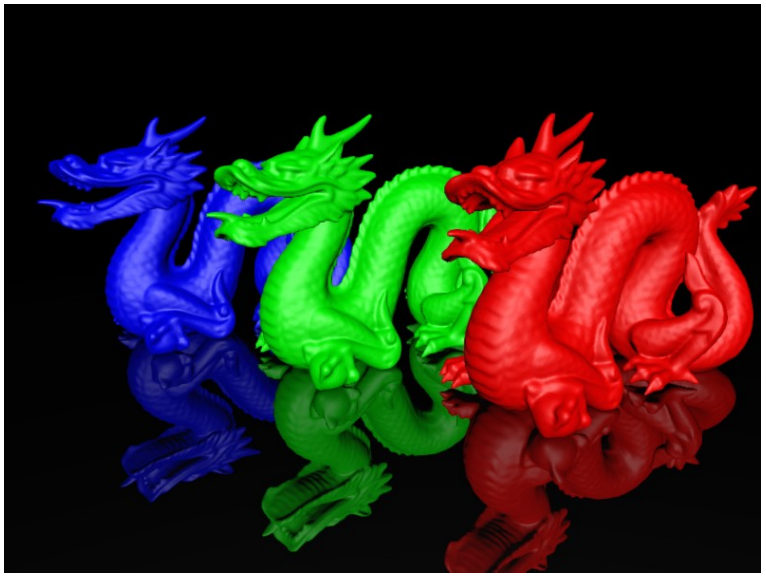


source/reader → filter → mapper → actor → renderer → renderWindow → interactor

Image source: http://www.realtimerendering.com

Actors

- **vtkActor** represents an object (geometry and properties) in a rendering scene
- Has position, scale, orientation, various rendering properties, textures, etc. Keeps a reference to the mapper.



source/reader → filter → mapper → actor →
renderer → renderWindow → interactor

Rendering

- The process of converting 3D graphics primitives (points, lines, triangles, etc), a specification for lights and materials, and a camera view into an 2D image that can be displayed on the screen

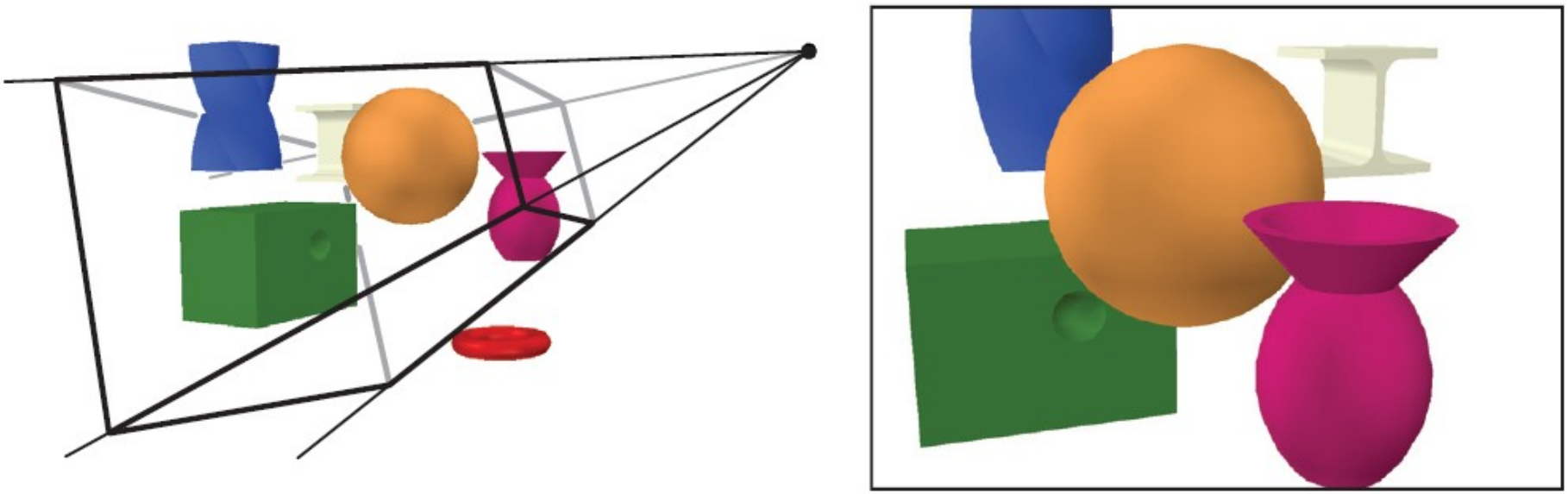
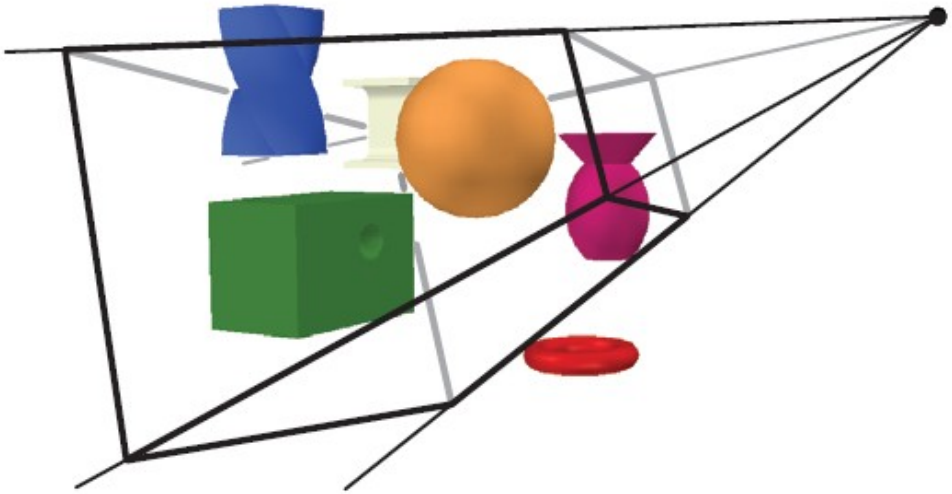
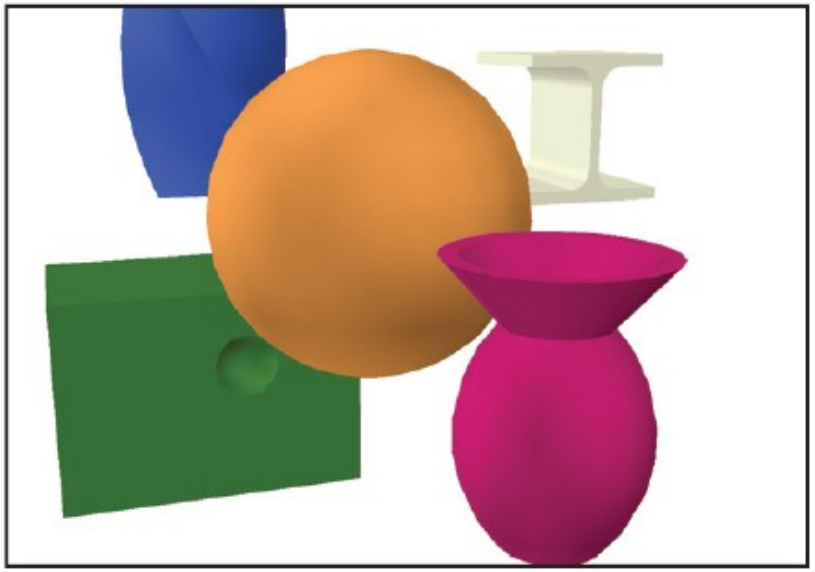


Image source: http://www.realtimerendering.com

Renderer

- **vtkRenderer** controls the rendering process for actors and scenes
- Under the hood, VTK uses OpenGL for rendering



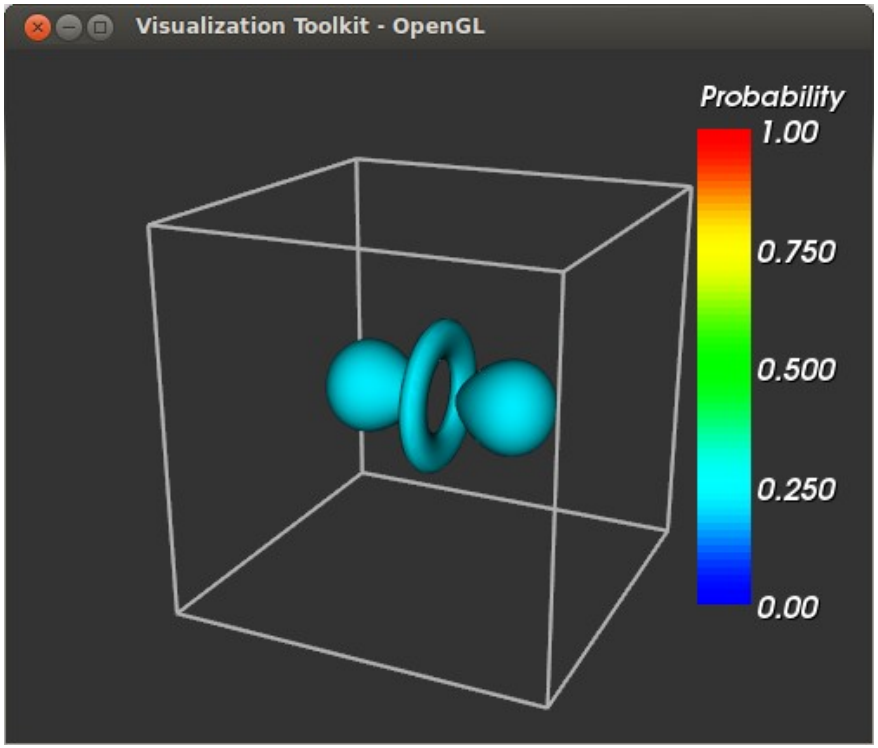


source/reader → filter → mapper → actor → renderer → renderWindow → interactor

Image source: http://www.realtimerendering.com

Render window

- The **vtkRenderWindow** class creates a window for renderers to draw into



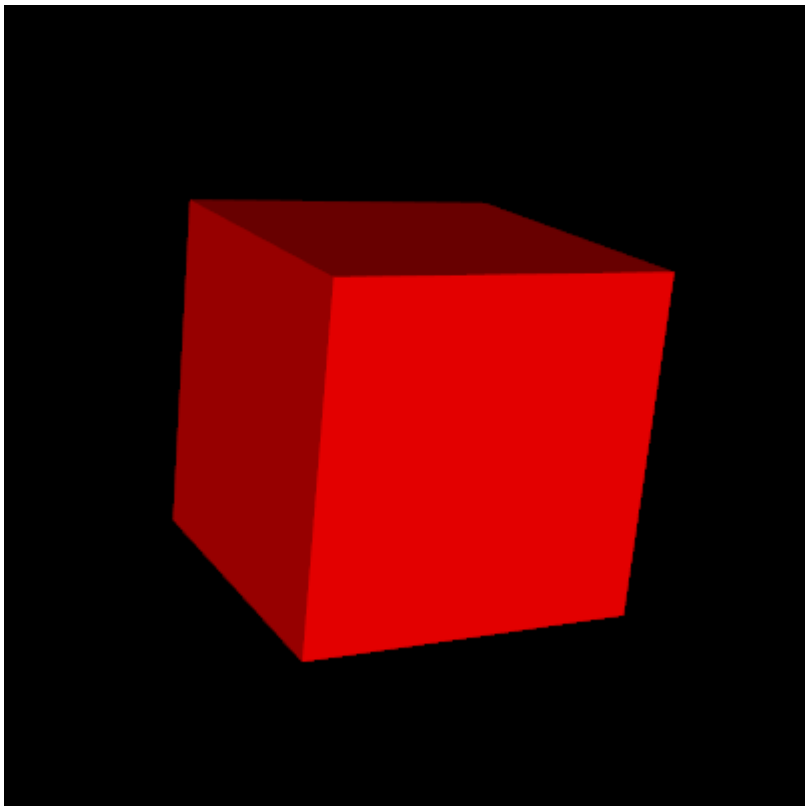
source/reader → filter → mapper → actor → renderer → renderWindow → interactor

Interactors

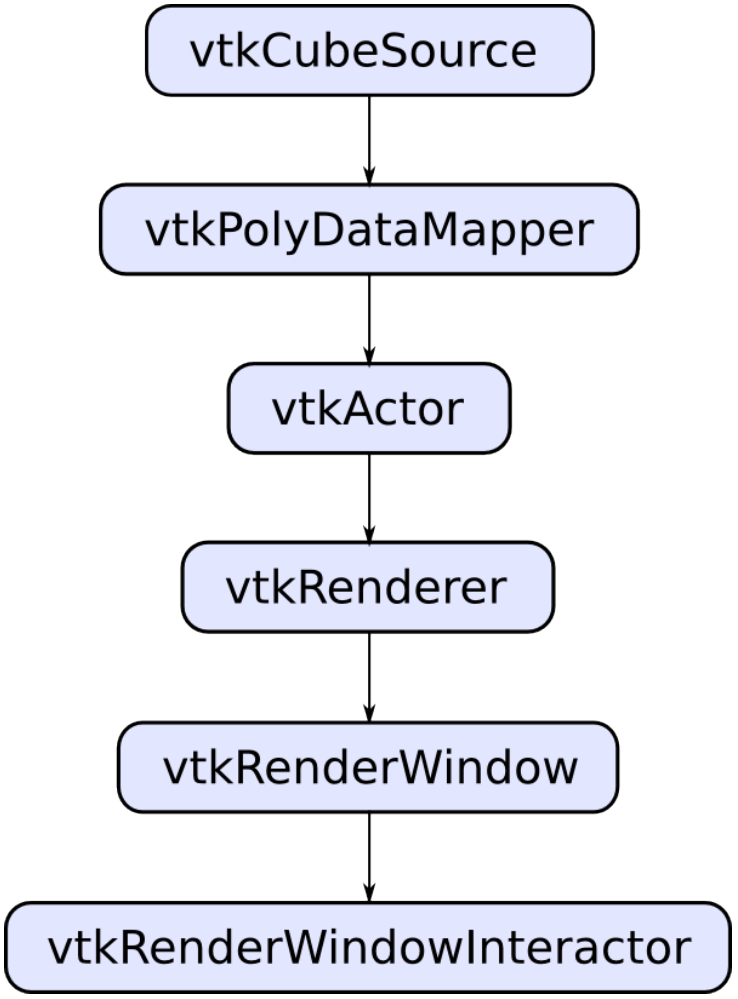
- The **vtkRenderWindowInteractor** class provides platform-independent window interaction via the mouse and keyboard
- Allows you to rotate/zoom/pan the camera, select and manipulate actors, etc
- Also handles time events

source/reader → filter → mapper → actor → renderer → renderWindow → interactor

Example 1:
Rendering a cube



Pipeline for the cube example



Source

```
import vtk

# Generate polygon data for a cube
cube = vtk.vtkCubeSource()
```

source/reader → filter → mapper → actor →
renderer → renderWindow → interactor

Mapper

```
# Create a mapper for the cube data
cube_mapper = vtk.vtkPolyDataMapper()
cube_mapper.SetInput(cube.GetOutput())
```

source/reader → filter → mapper → actor →
renderer → renderWindow → interactor

Actor

```
# Connect the mapper to an actor
cube_actor = vtk.vtkActor()
cube_actor.SetMapper(cube_mapper)
cube_actor.GetProperty().SetColor(1.0, 0.0, 0.0)
```

source/reader → filter → mapper → actor →
renderer → renderWindow → interactor

Renderer

```
# Create a renderer and add the cube actor to it
renderer = vtk.vtkRenderer()
renderer.SetBackground(0.0, 0.0, 0.0)
renderer.AddActor(cube_actor)
```

source/reader → filter → mapper → actor →
renderer → renderWindow → interactor

Render window

```
# Create a render window
render_window = vtk.vtkRenderWindow()
render_window.SetWindowName("Simple VTK scene")
render_window.SetSize(400, 400)
render_window.AddRenderer(renderer)
```

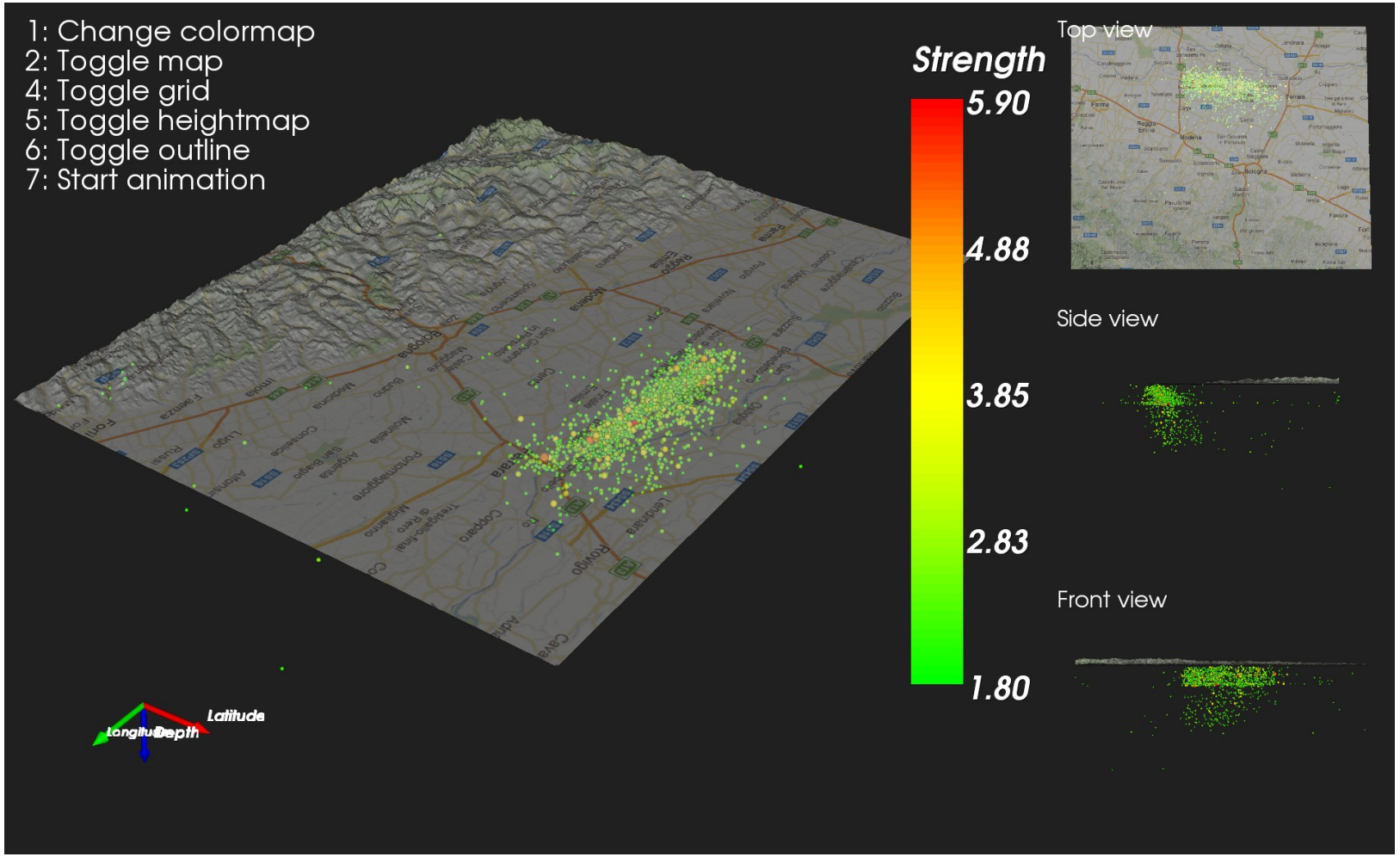
source/reader → filter → mapper → actor →
renderer → renderWindow → interactor

Interactor

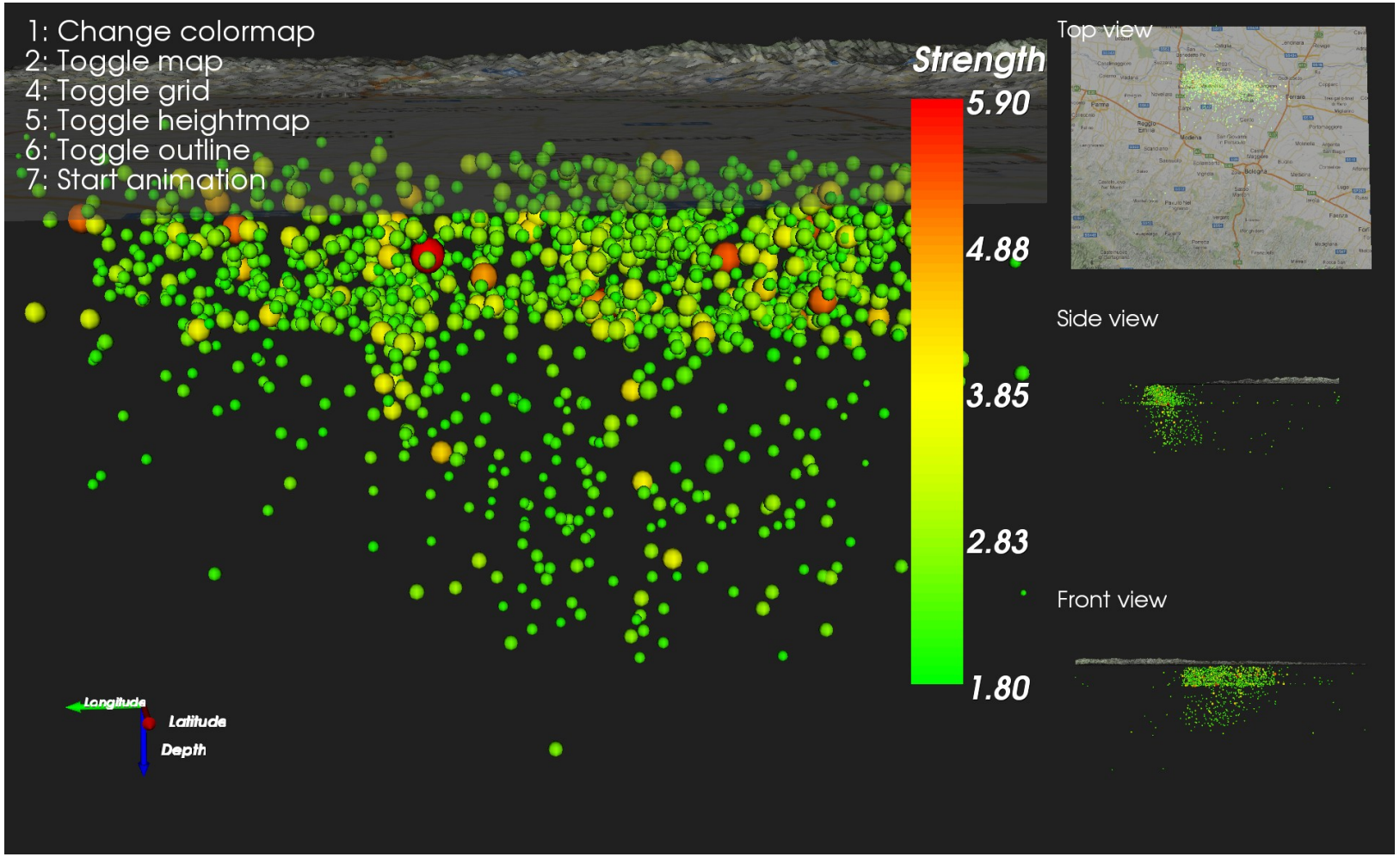
```
# Create an interactor
interactor = vtk.vtkRenderWindowInteractor()
interactor.SetRenderWindow(render_window)

# Initialize the interactor and start the
# rendering loop
interactor.Initialize()
render_window.Render()
interactor.Start()
```

source/reader → filter → mapper → actor →
renderer → renderWindow → interactor

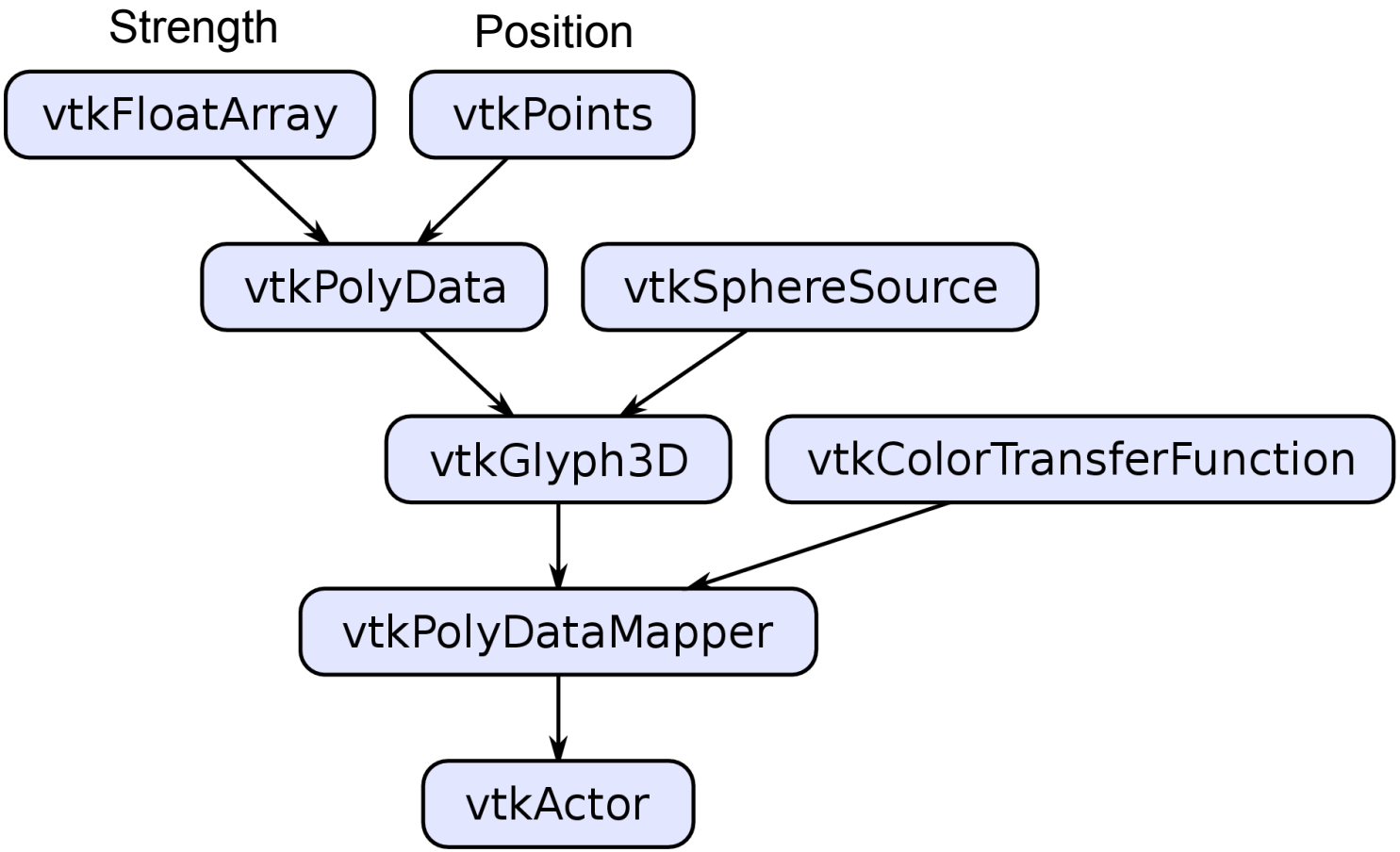
Example 2: Earthquake data



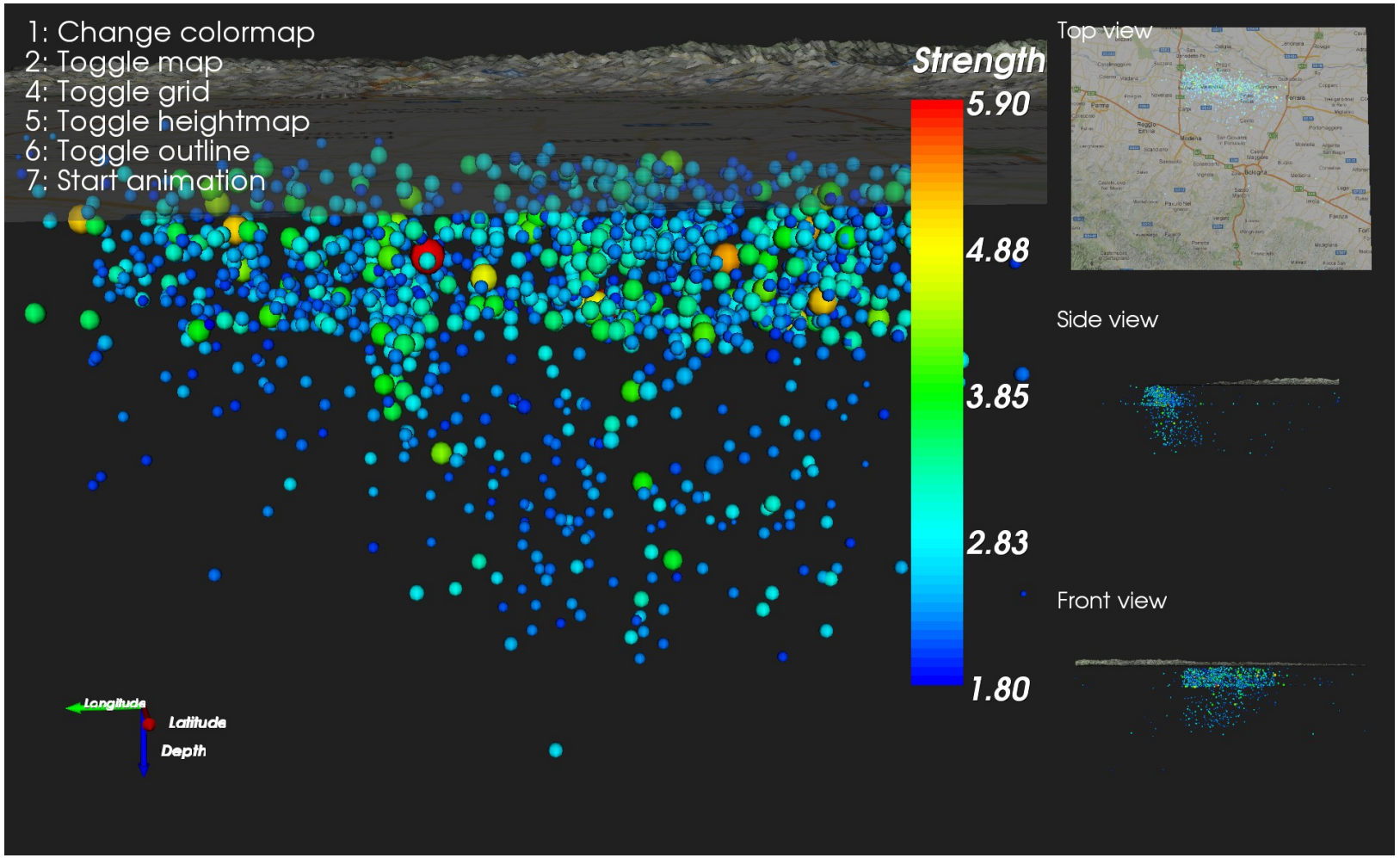
Visualizing the quakes with sphere glyphs



Sphere glyphs

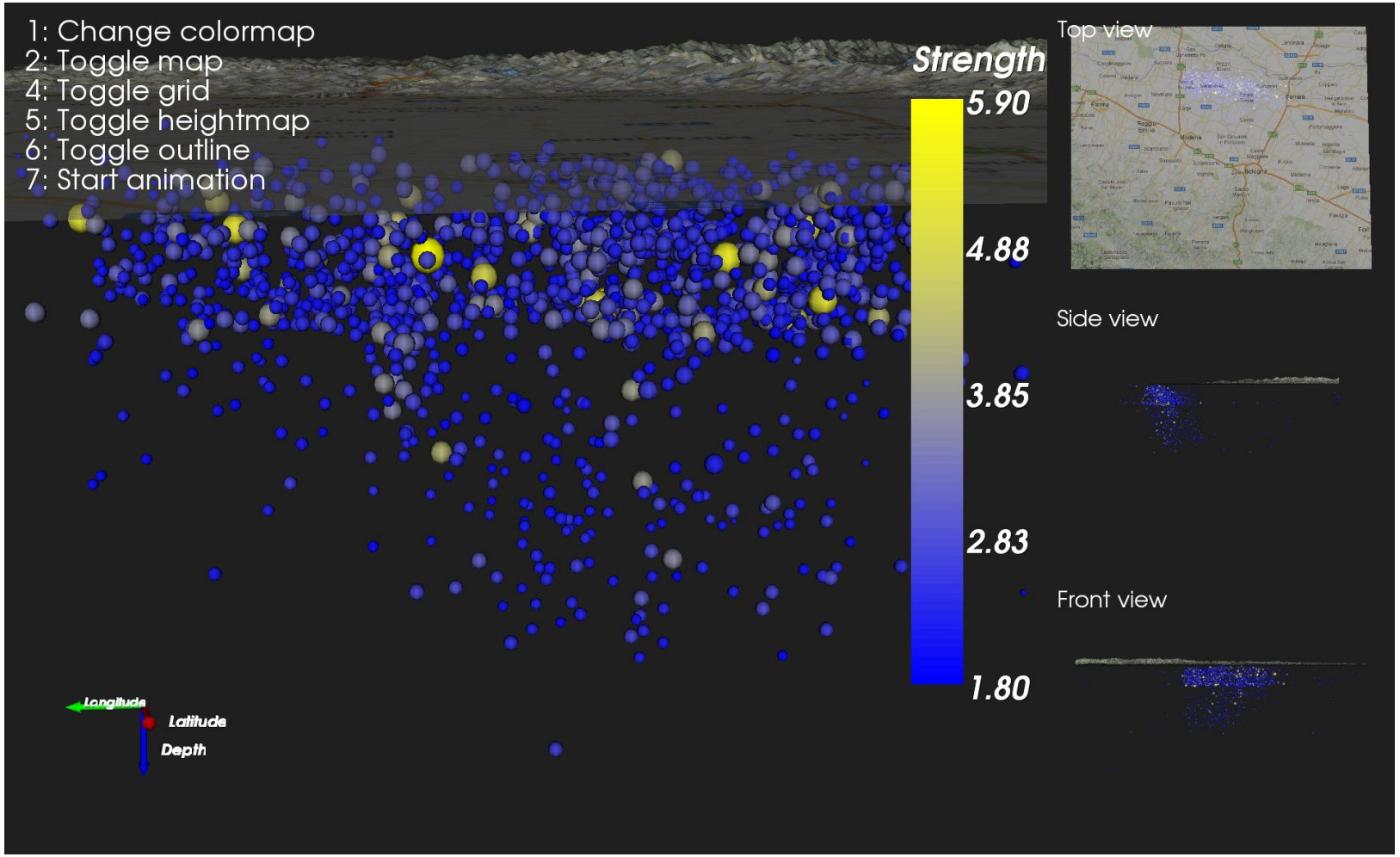


Colormaps

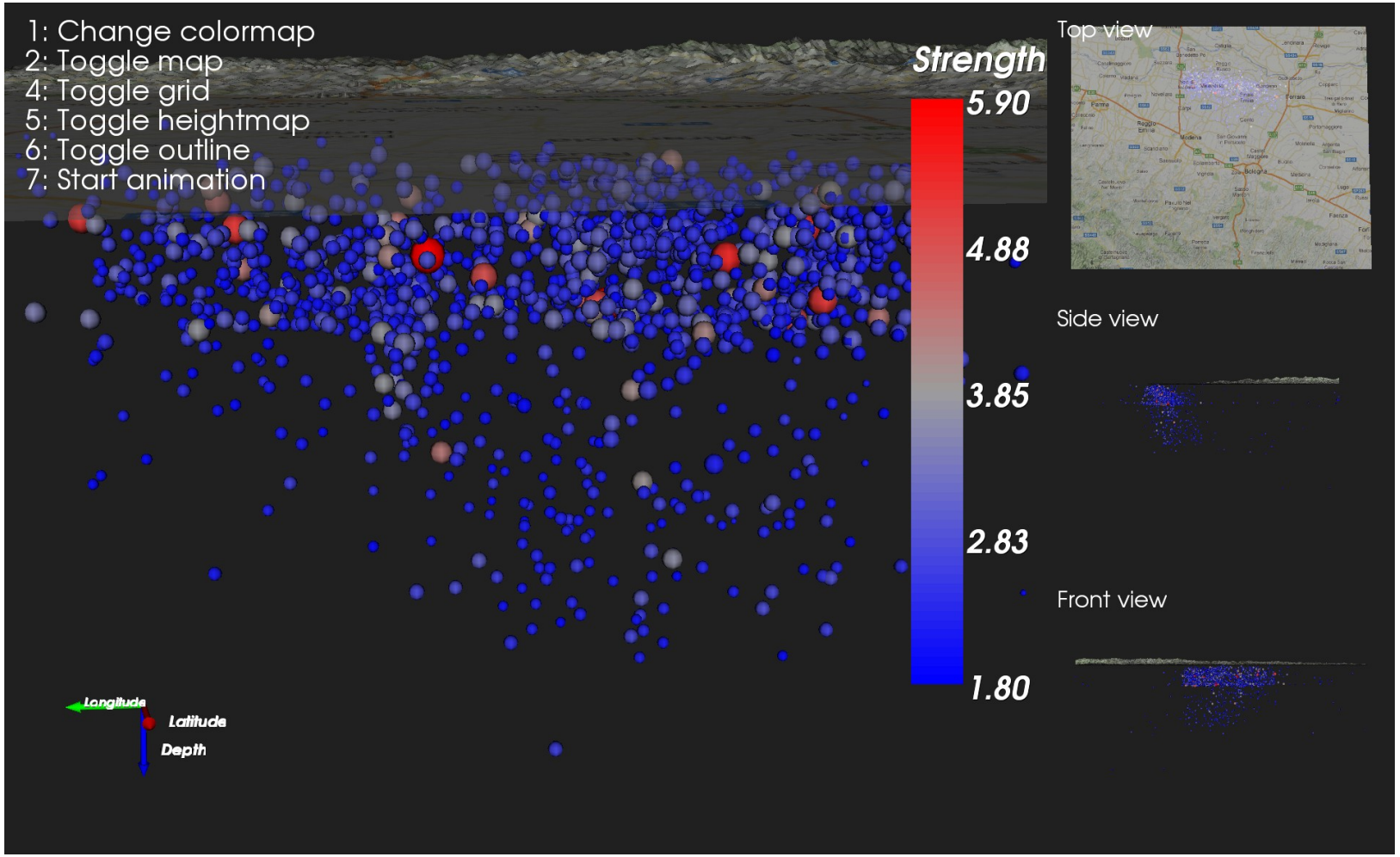


See this paper for a discussion on why the ”rainbow” colormap is a poor choice for most applications

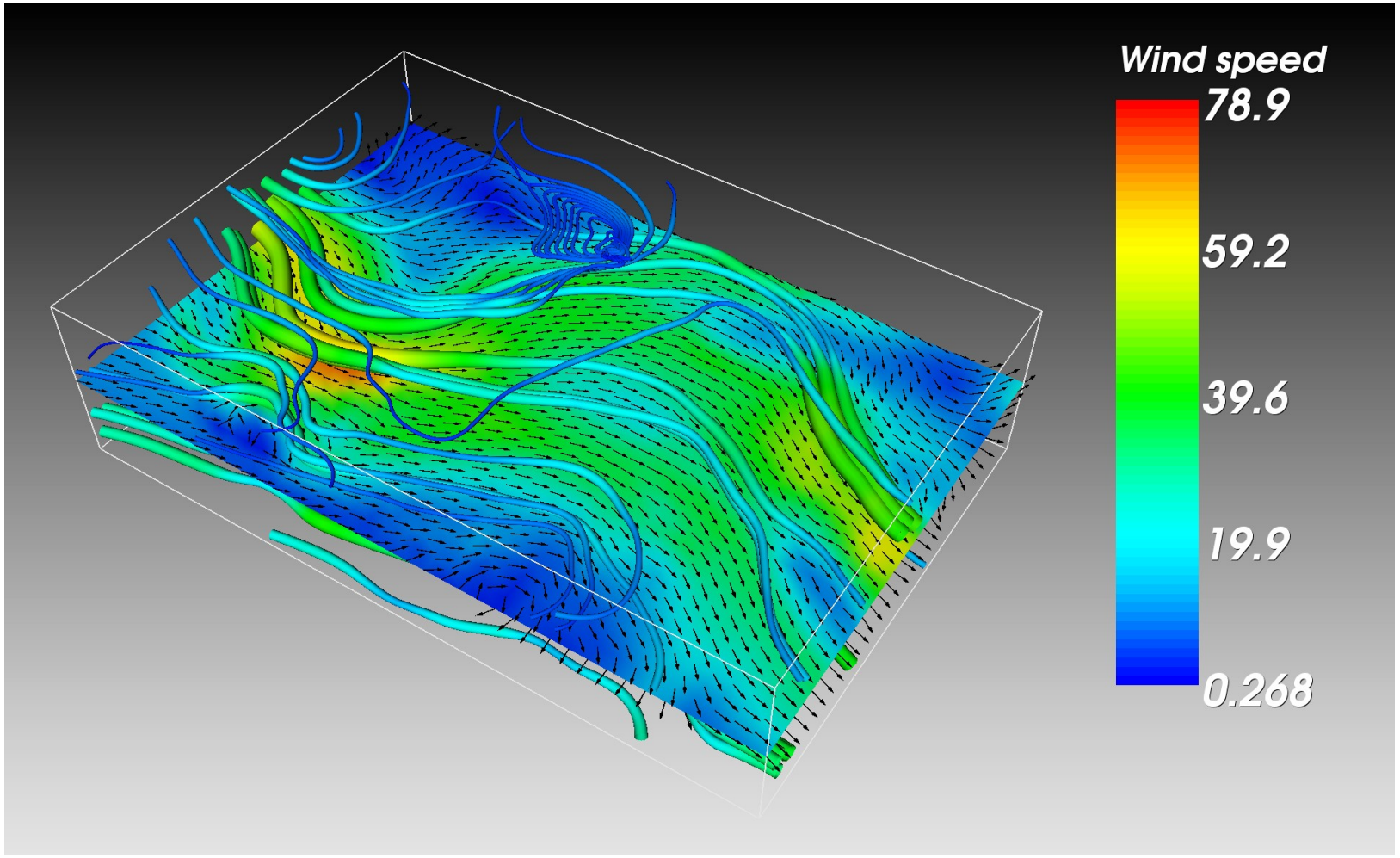
Colormaps


1: Change colormap
2: Toggle map
4: Toggle grid
5: Toggle heightmap
6: Toggle outline
7: Start animation
Strength
5.90
4.88
3.85
2.83
1.80
Top view
Side view
Front view
Longitude
Latitude
Depth

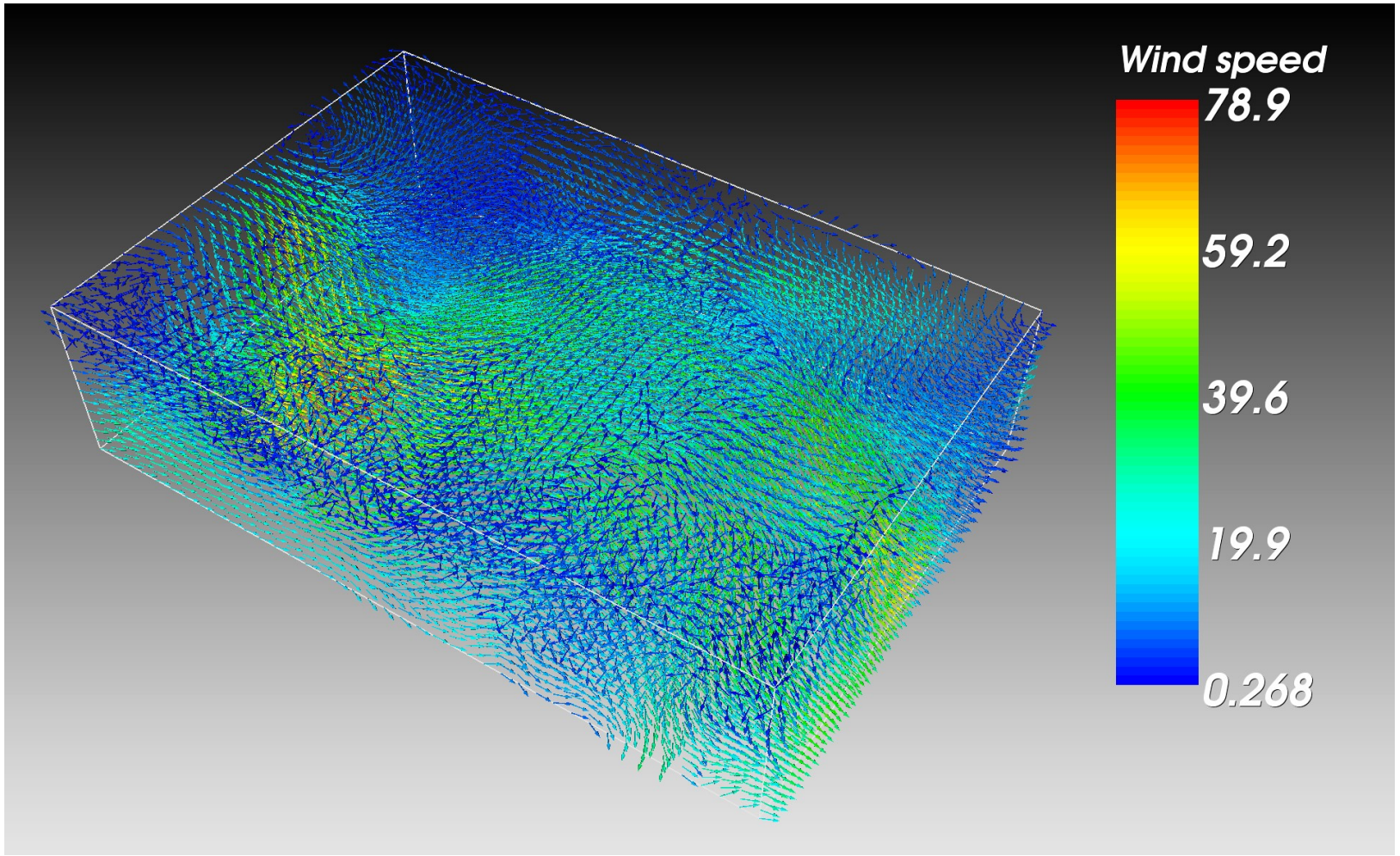
Colormaps


1: Change colormap
2: Toggle map
4: Toggle grid
5: Toggle heightmap
6: Toggle outline
7: Start animation
Strength
5.90
4.88
3.85
2.83
1.80
Top view
Side view
Front view
Longitude
Latitude
Depth

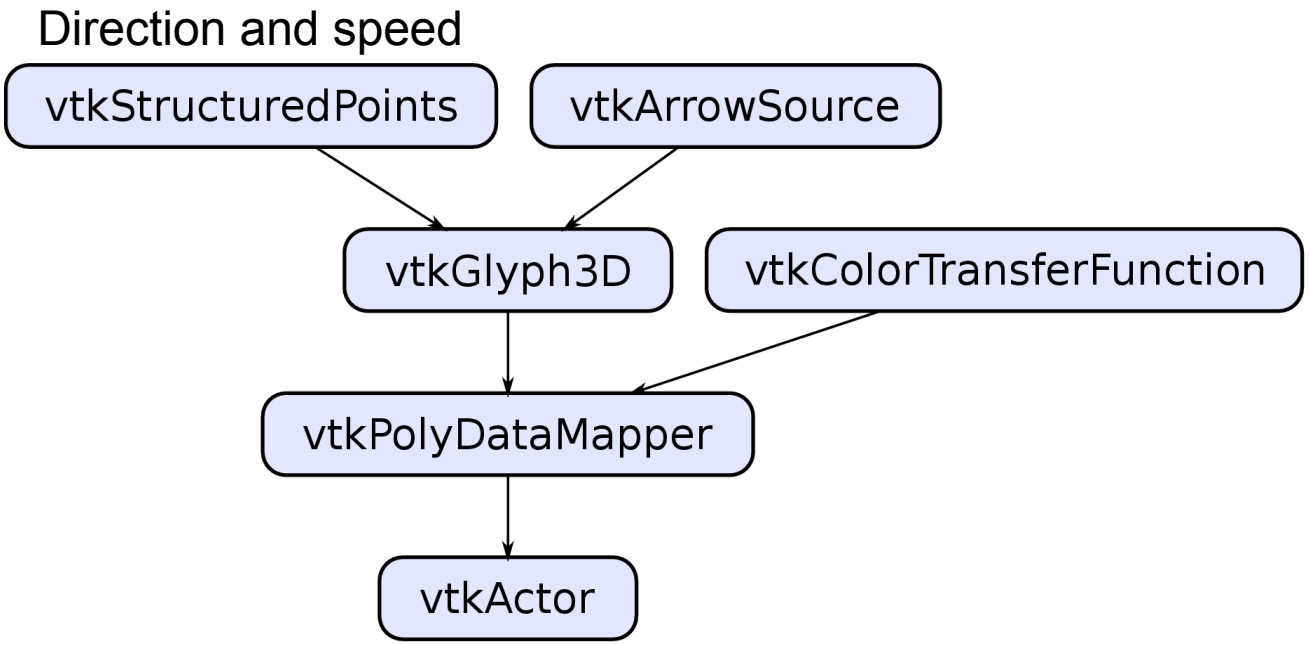
Example 2:
Air currents



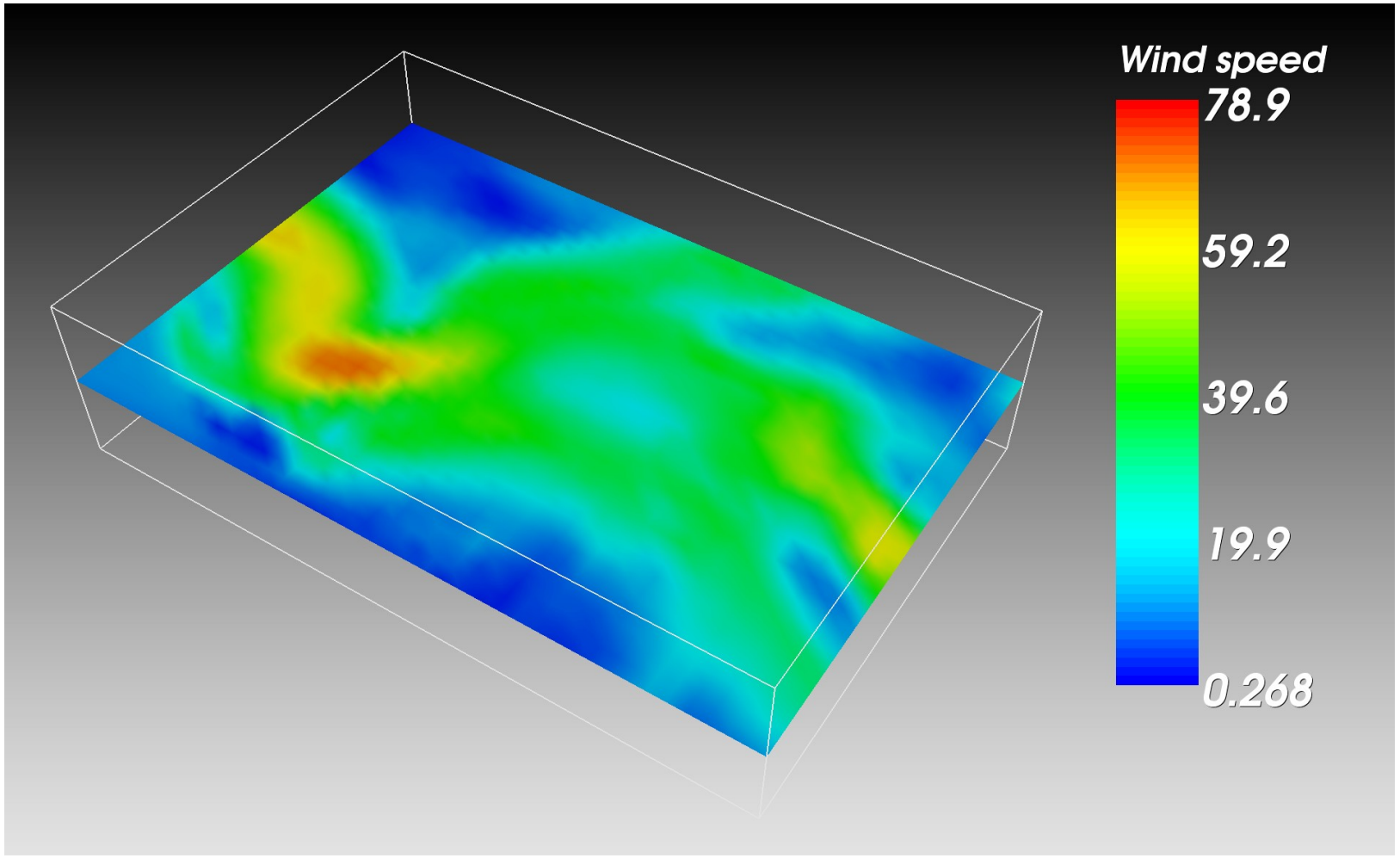
Arrow glyphs, first try



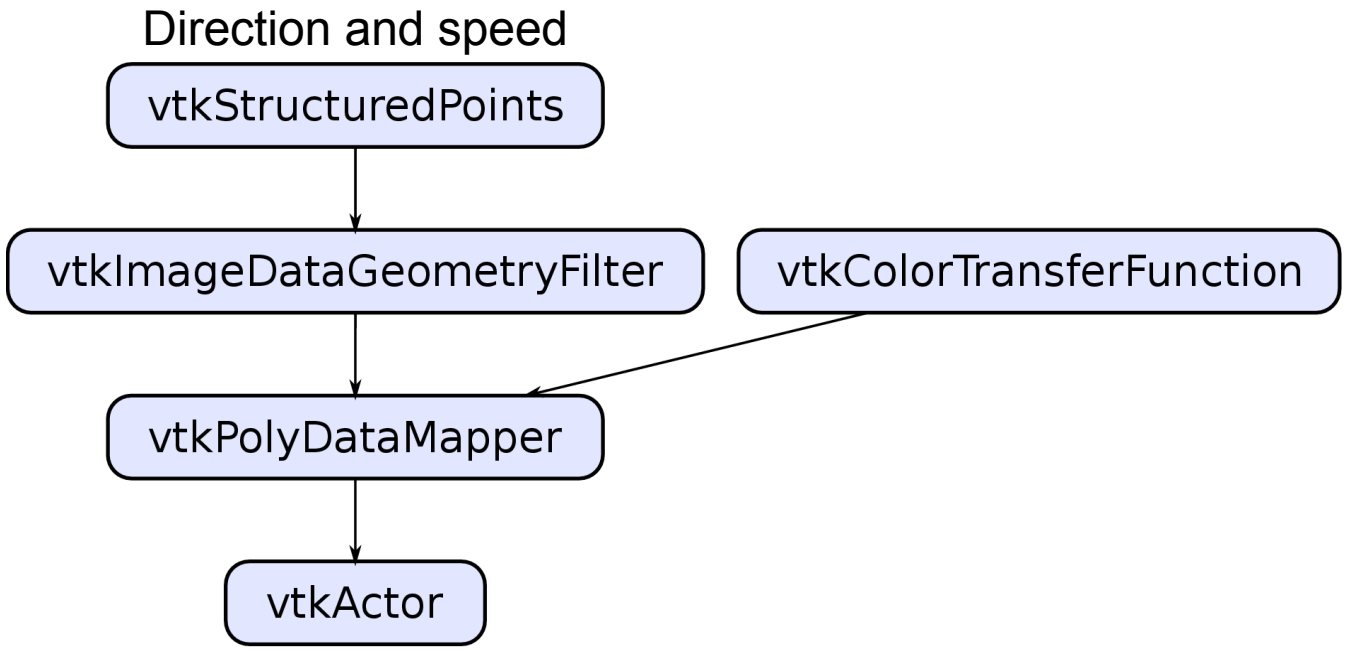
Arrow glyphs, first try



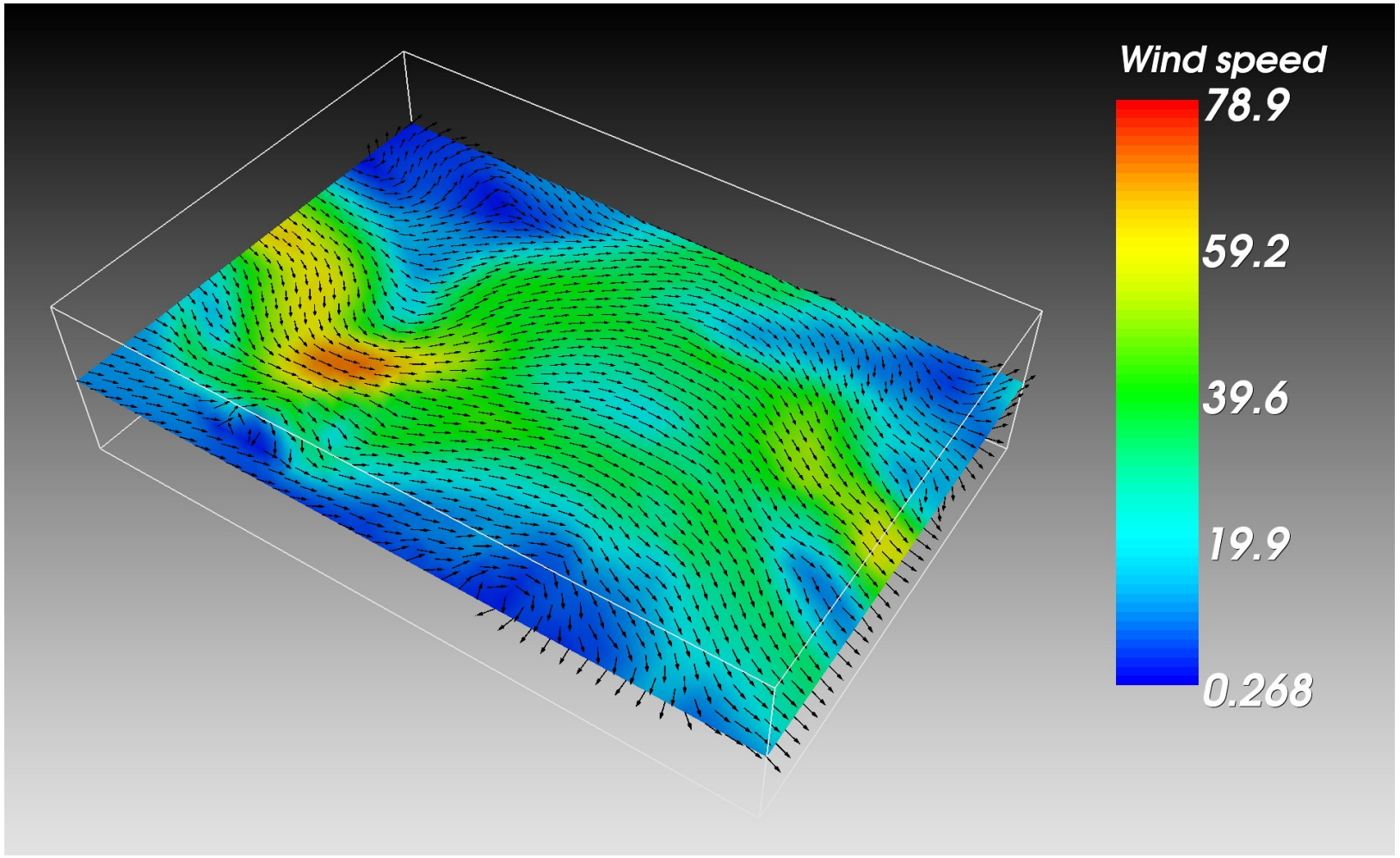
Cut planes



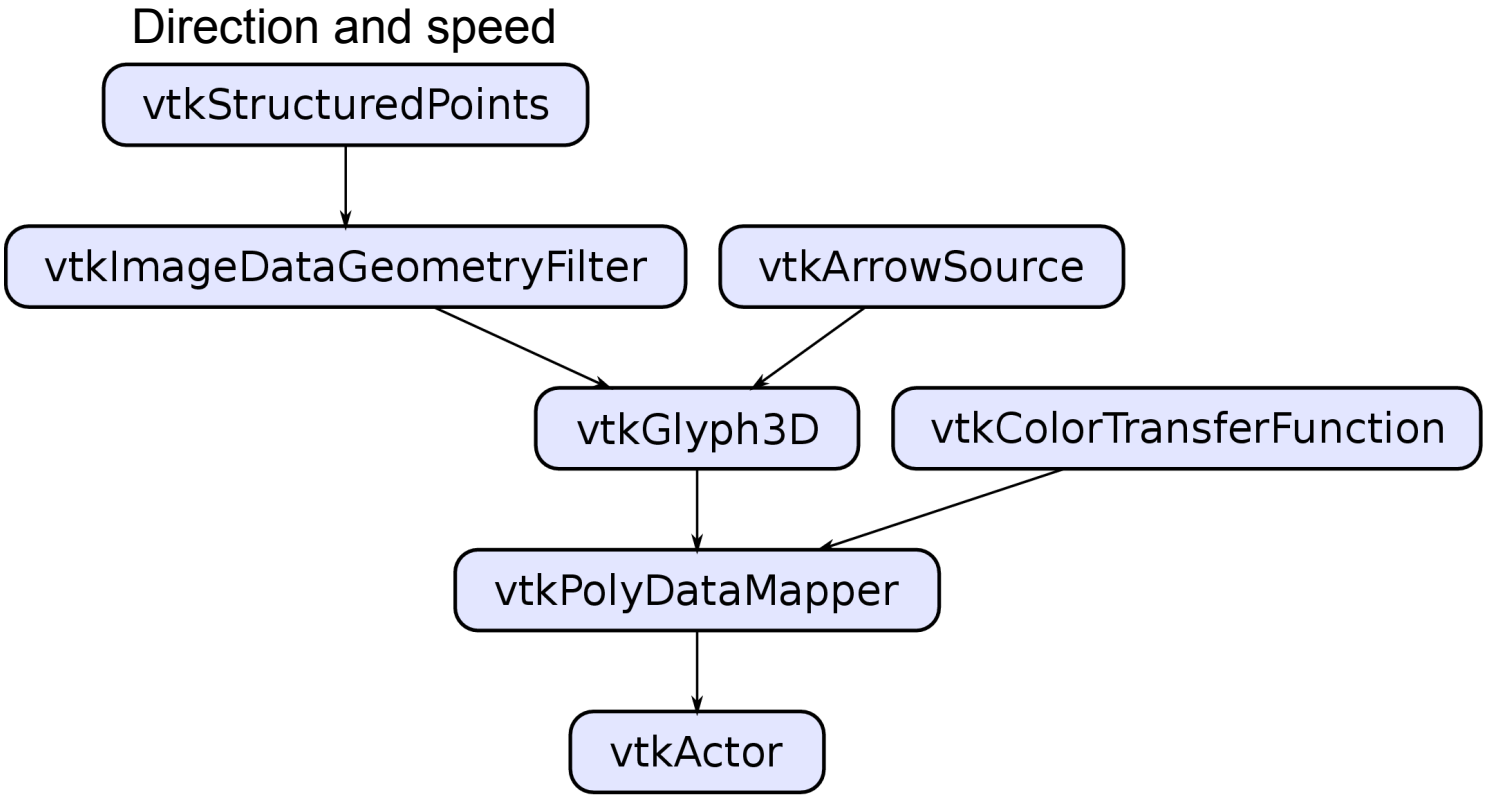
Cut planes


Direction and speed
vtkStructuredPoints
vtkImageDataGeometryFilter
vtkColorTransferFunction
vtkPolyDataMapper
vtkActor

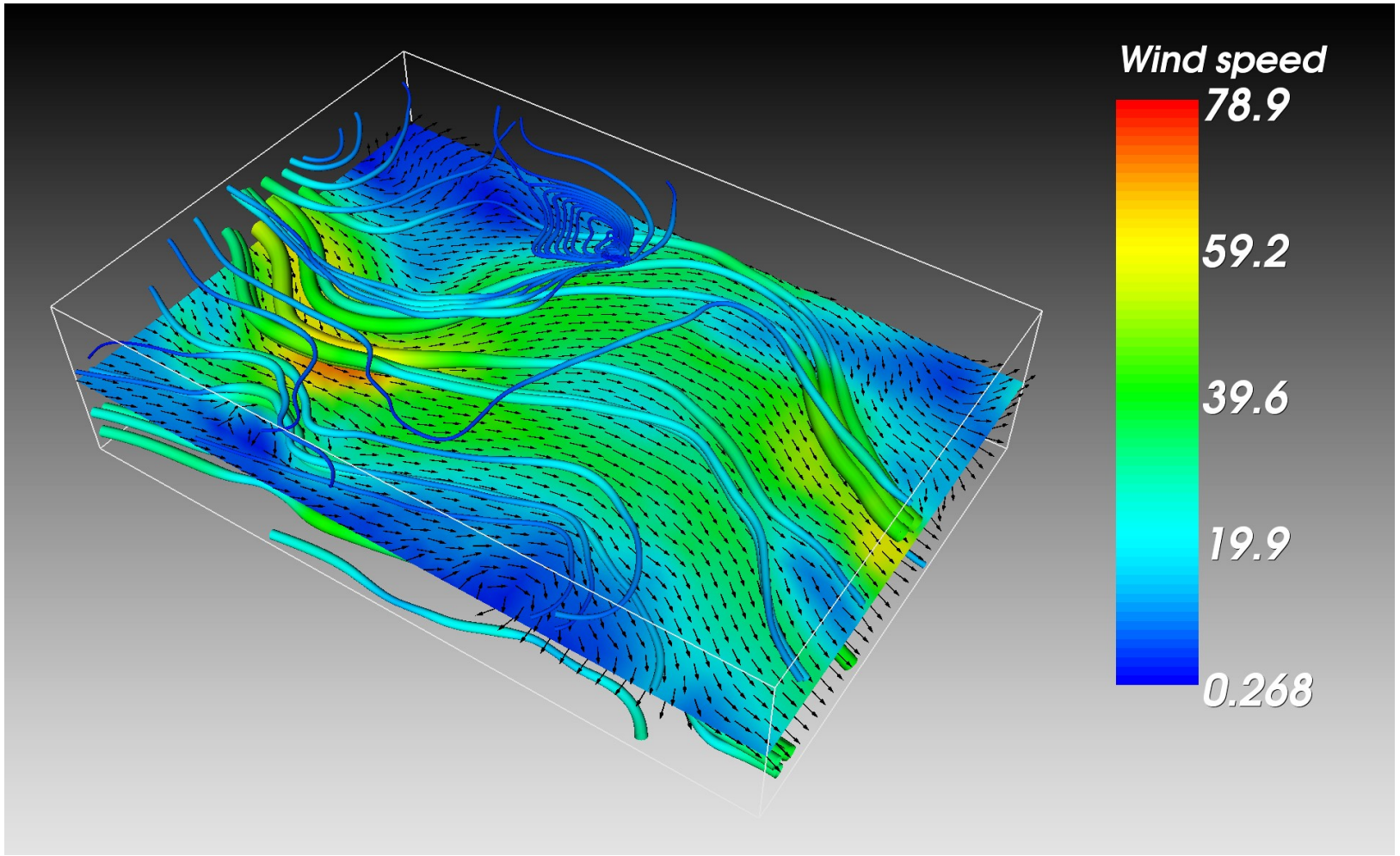
Arrow glyphs, second try



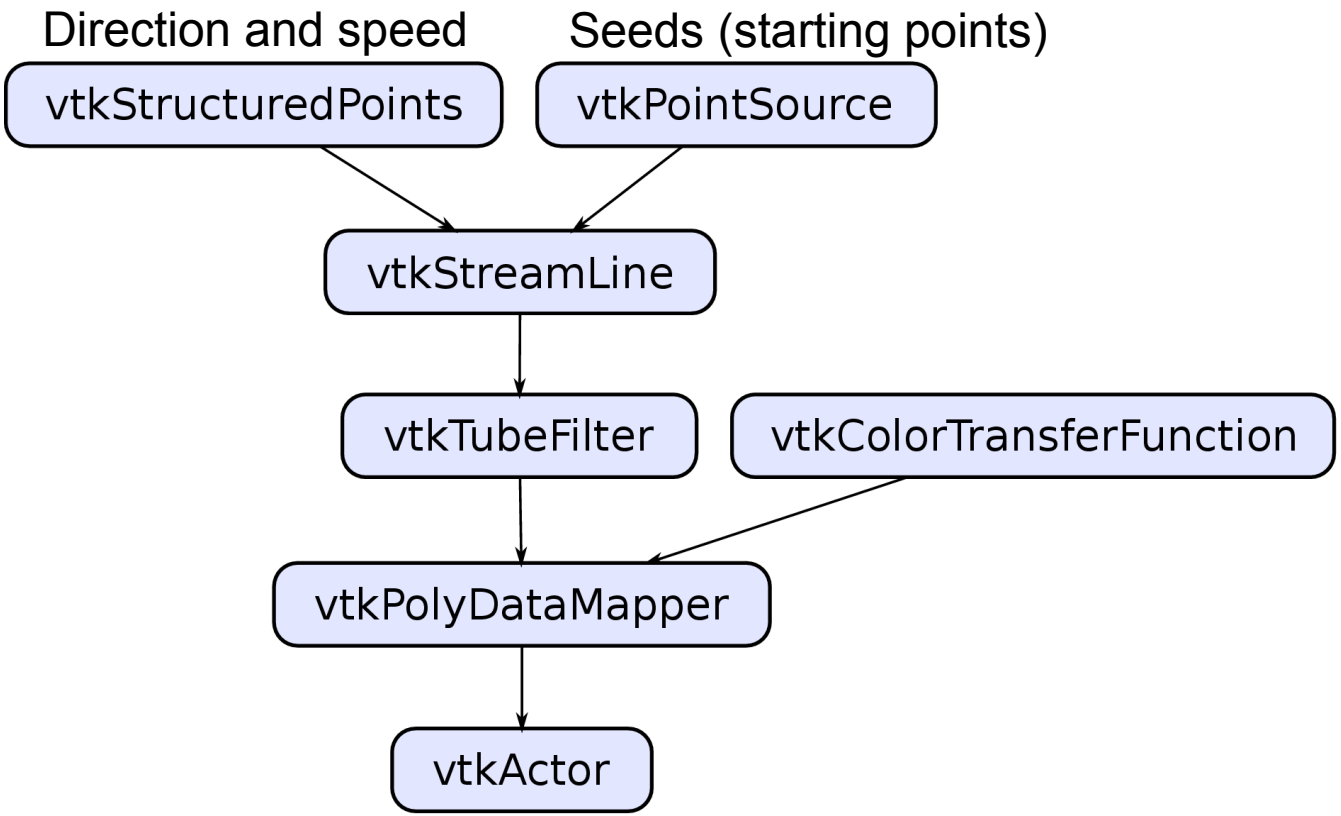
Arrow glyphs, second try



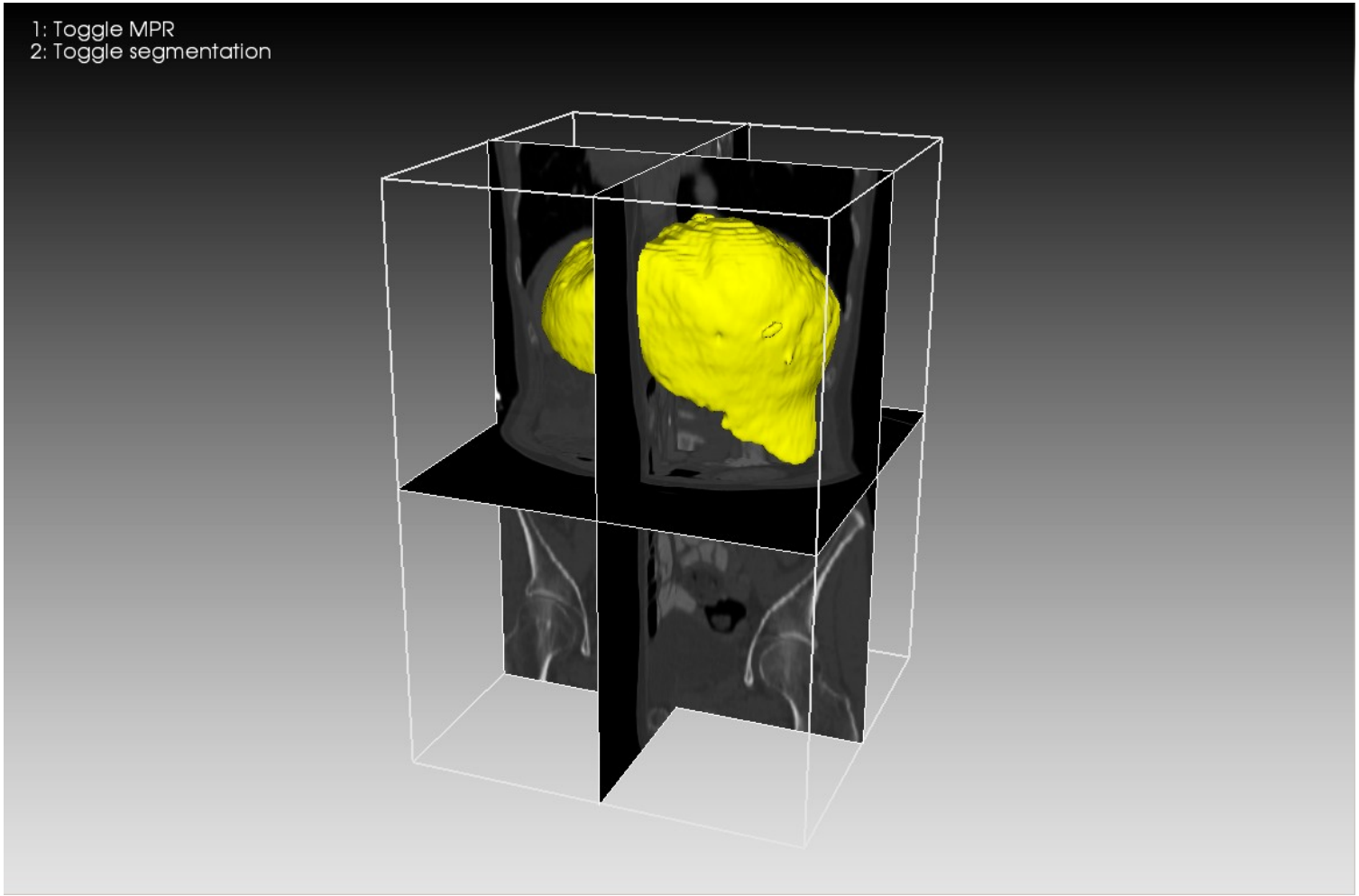
Streamtubes



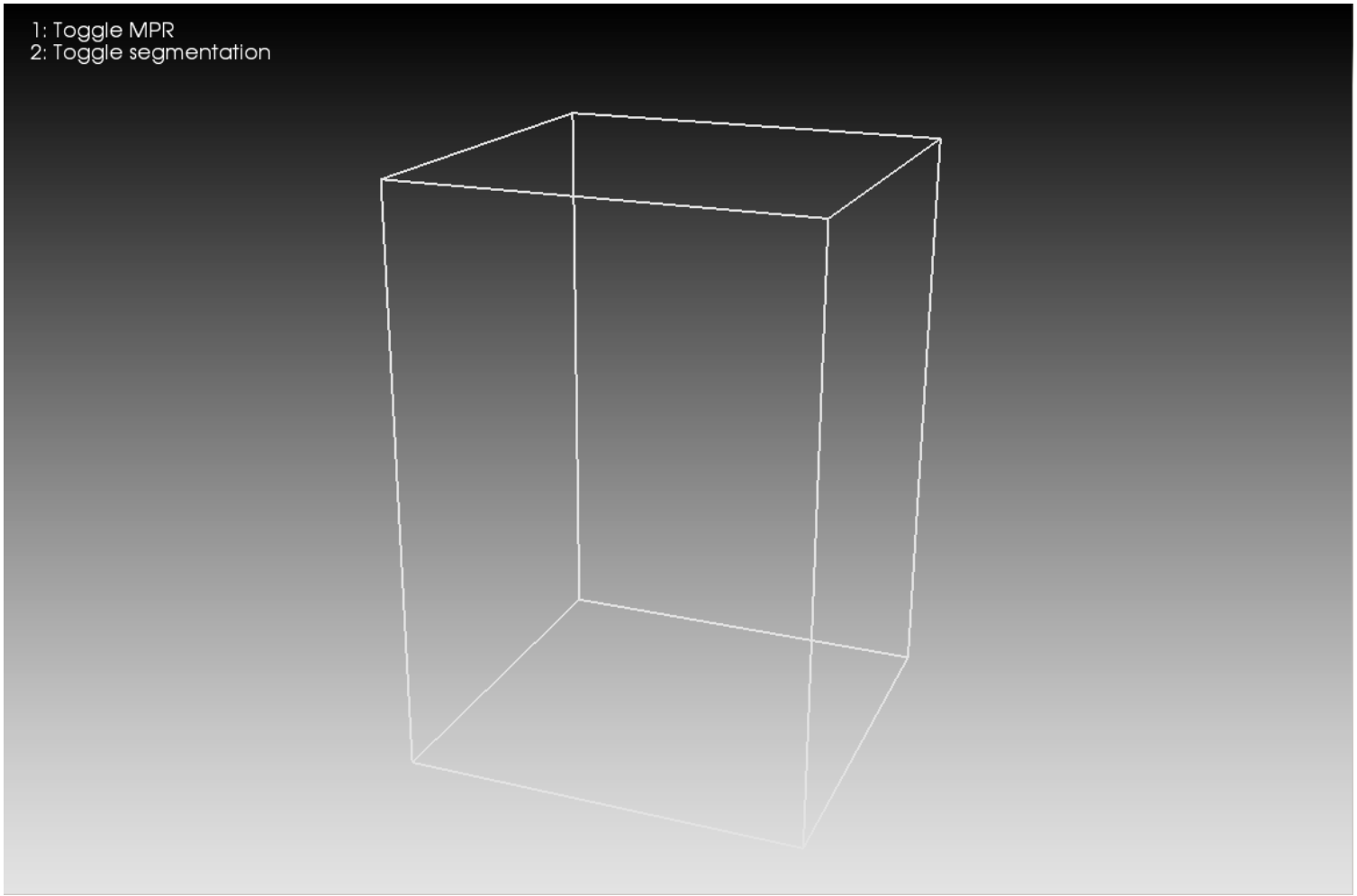
Streamtubes



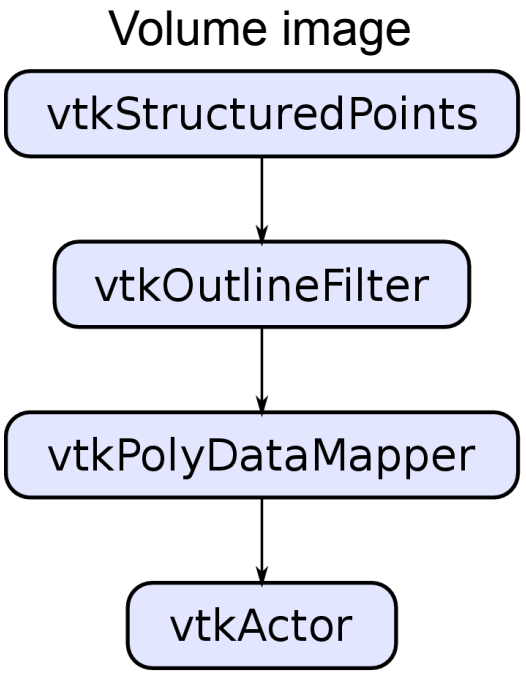
Example 3:
Medical 3D data



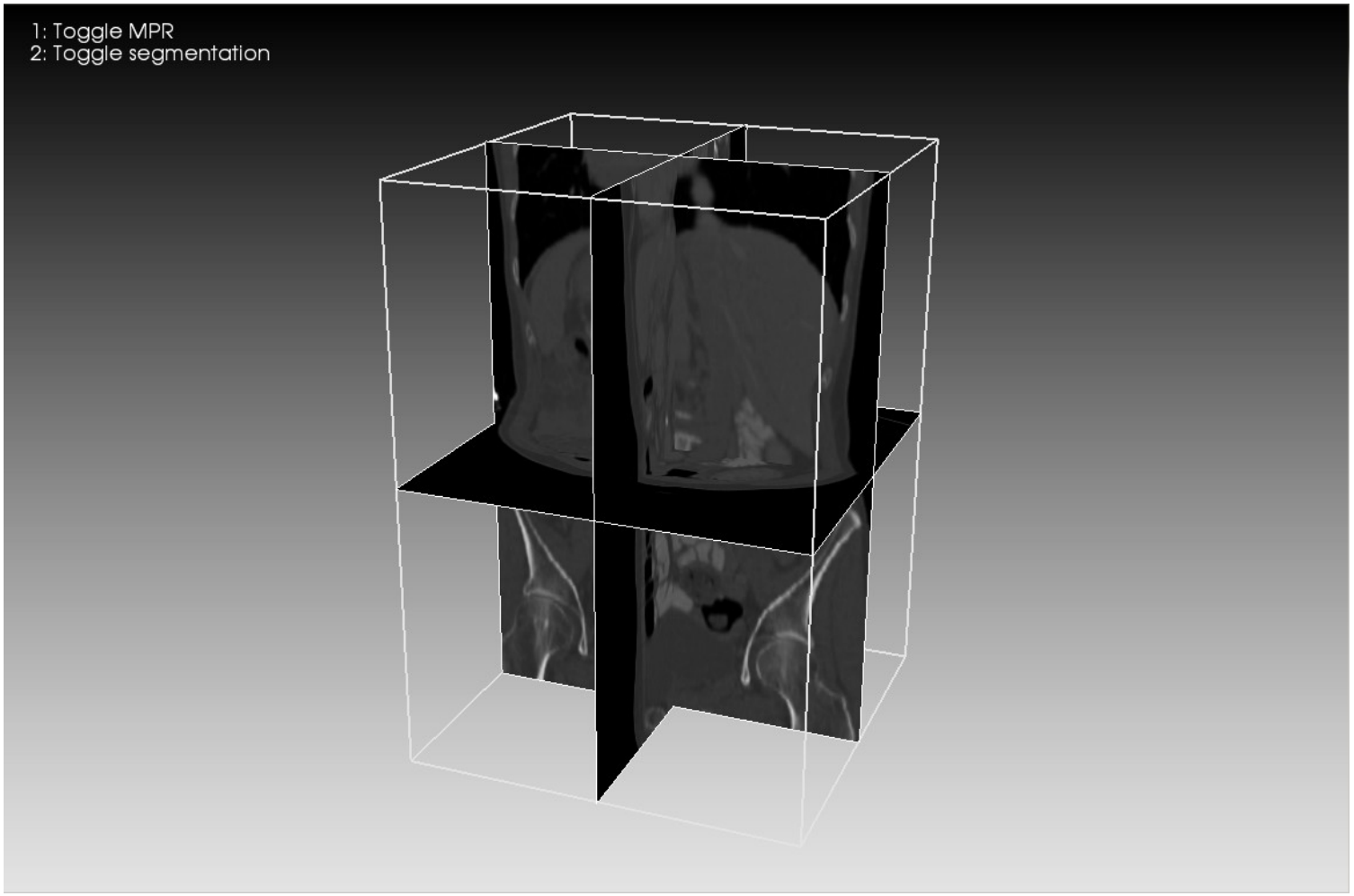
Outline



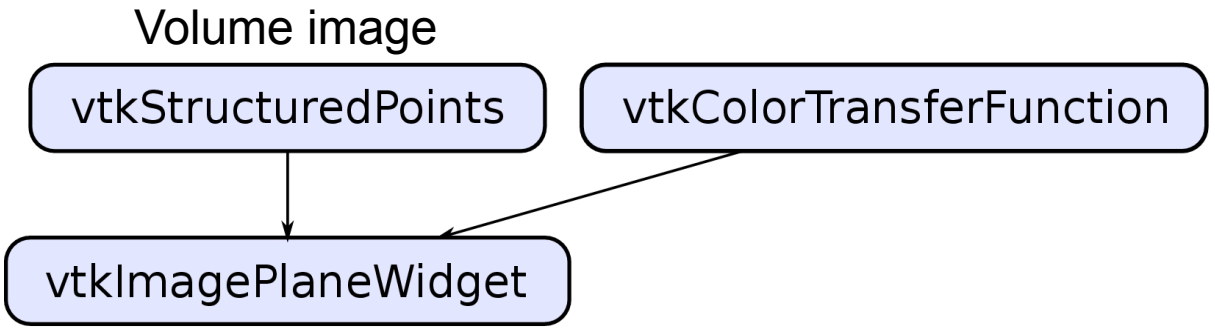
Outline


Volume image
vtkStructuredPoints
vtkOutlineFilter
vtkPolyDataMapper
vtkActor

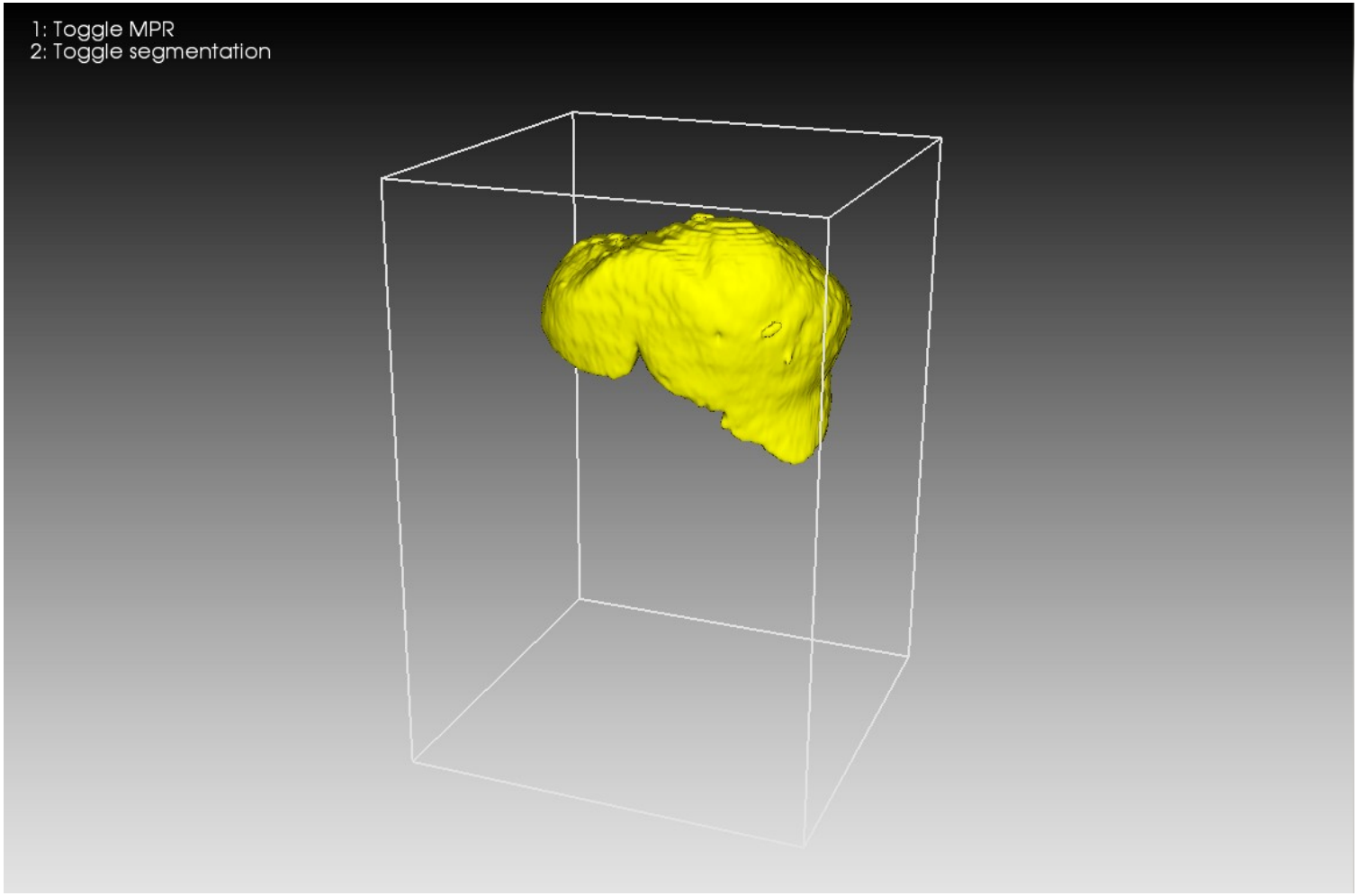
Multi-planar reformatting (MPR)



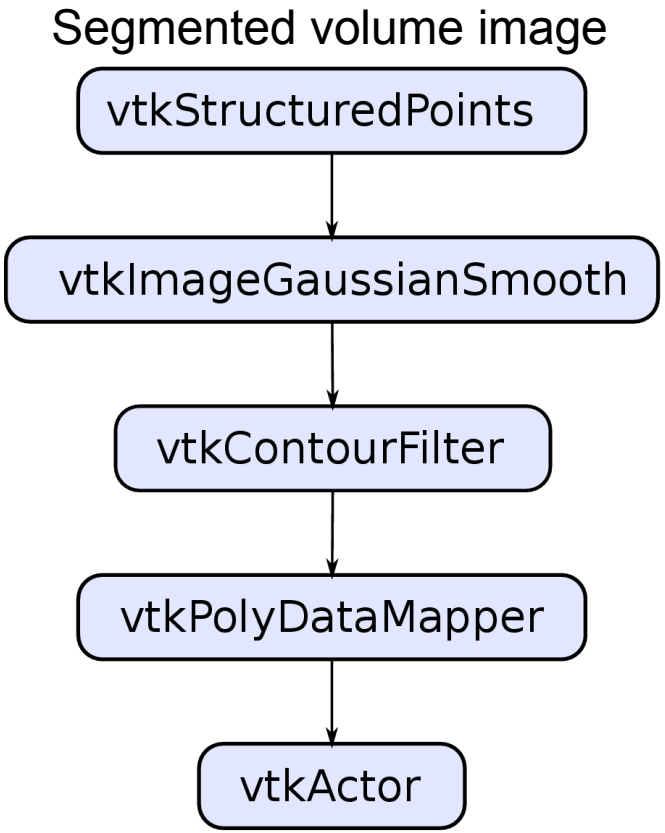
Multi-planar reformatting (MPR)



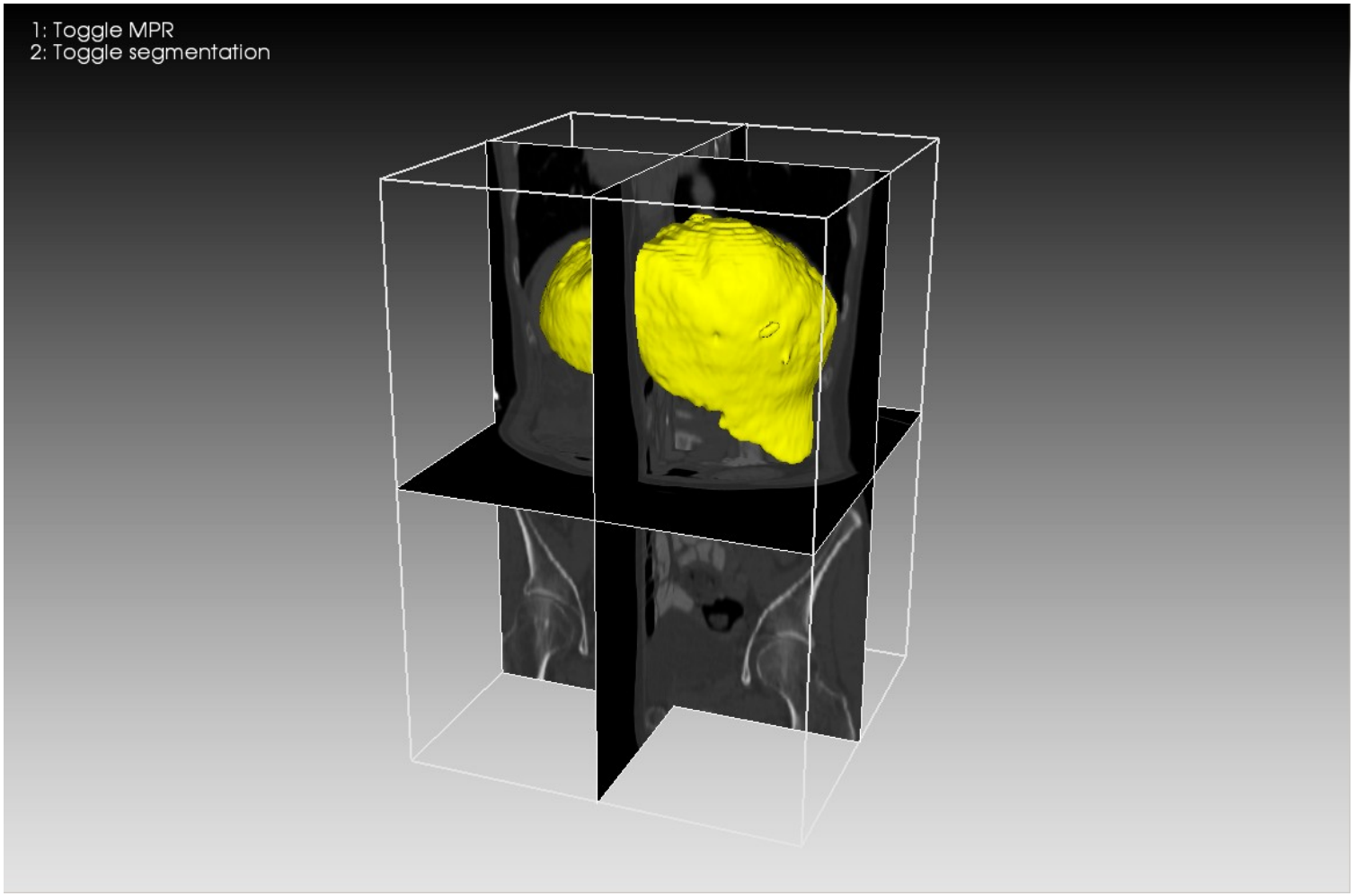
Surface rendering



Surface rendering



Combined visualization

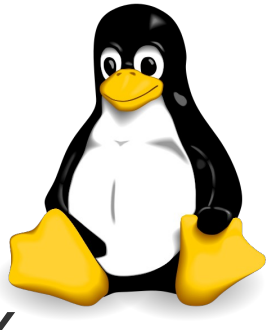


Summary

- VTK contains thousands of classes and might seem a bit intimidating at first...
  - however, one can create useful visualizations with just a few core classes
- The pipeline is typically

  source/reader → filter → mapper → actor → renderer → renderWindow → interactor
- Use VTK's example programs as templates when you write new programs!

Installing VTK on Linux



- Included in the package repository of most Linux distributions
- On Ubuntu 12.04 you can install VTK and the Python-wrapper with the command

```
sudo apt-get install libvtk5-dev python-vtk
```

- Also fairly easy to build VTK from source. You need GCC, CMake, + some extra dependencies
- Finally, you can install VTK via the Python distribution Anaconda (see next slide)

Installing VTK on Windows

- Don't bother compiling it yourself (unless you have plenty of time to spare)
- Install it via one of the following Python distributions:
  - Anaconda (VTK is available in the package repository)
  - pythonxy (Warning! will override existing Python installations)
- More detailed installation instructions can be found on the course webpage

Installing VTK on Mac

- Install it via Anaconda (see previous slide)
- Expect to spend several hours in front of the compiler if you try to build it yourself...

Paraview and Mayavi

- Free data visualizers built on VTK
- You can use them to try out different visualization techniques (without writing a single line of code)
- Links:
  - http://www.paraview.org/
  - http://docs.enthought.com/mayavi/mayavi/index.html