Public resources for visualization

Miloš Šrámek & Leonid Dimitrov

Credits

- Based on presentations of
  - Bill Schroeder
  - Erik Widholm
  - Luis Ibáñez
  - Martin Urschler

Agenda

- Data sets
- Software
  - VTK and VTK-based applications
  - ITK and ITK-based applications
  - SciRun

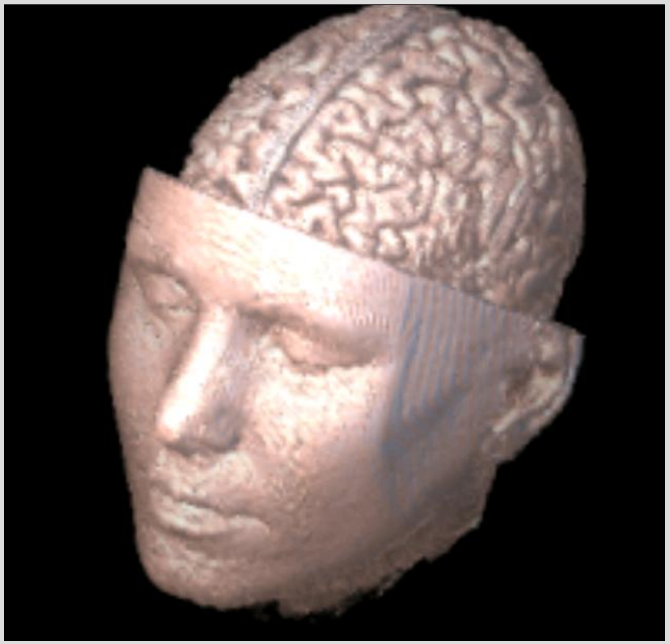
Publicly Available Data Sets

- Purpose:
  - Algorithm testing
  - Common reference for comparison

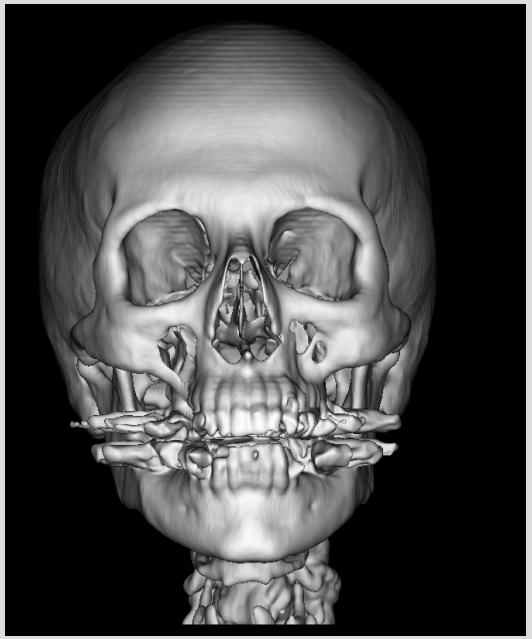
The Stanford stuff

- The Stanford volume data archive:

The Stanford bunny (photo)



MR Brain data



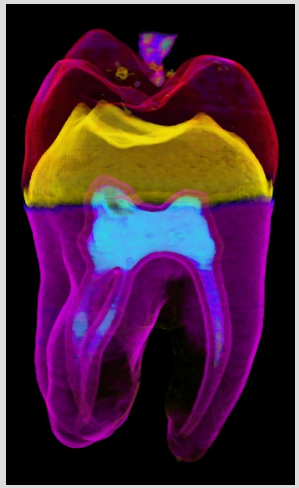
CT Head data

http://graphics.stanford.edu/data/voldata/

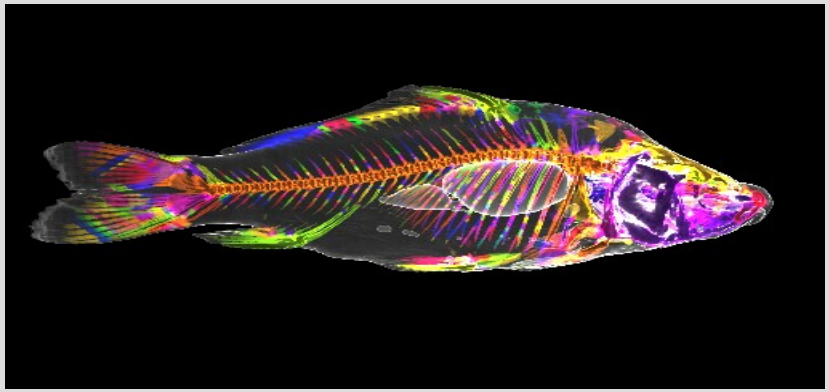
The Volume Library

http://www9.informatik.uni-erlangen.de/External/vollib/

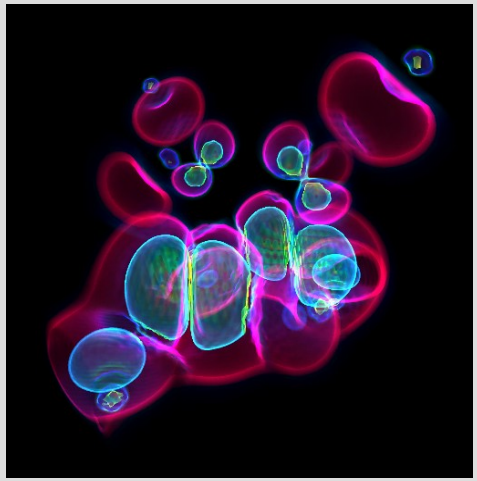
The lobster



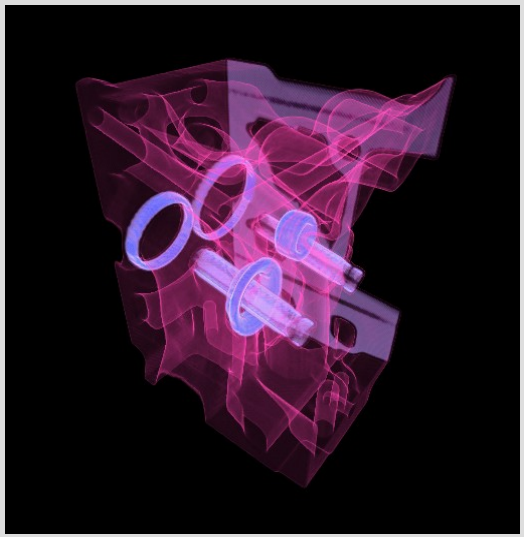
The tooth



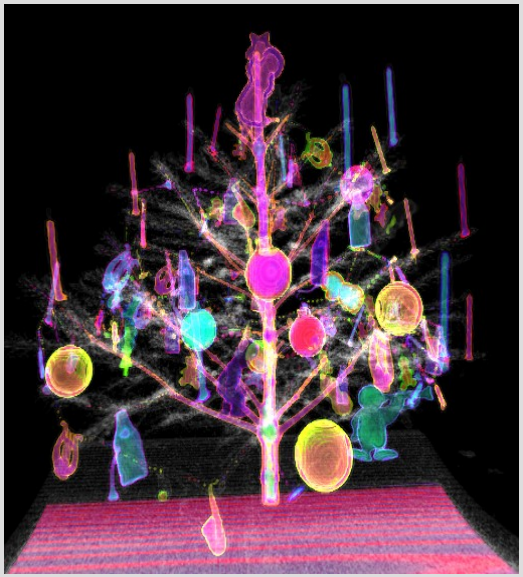
The carp



Hipip molecule



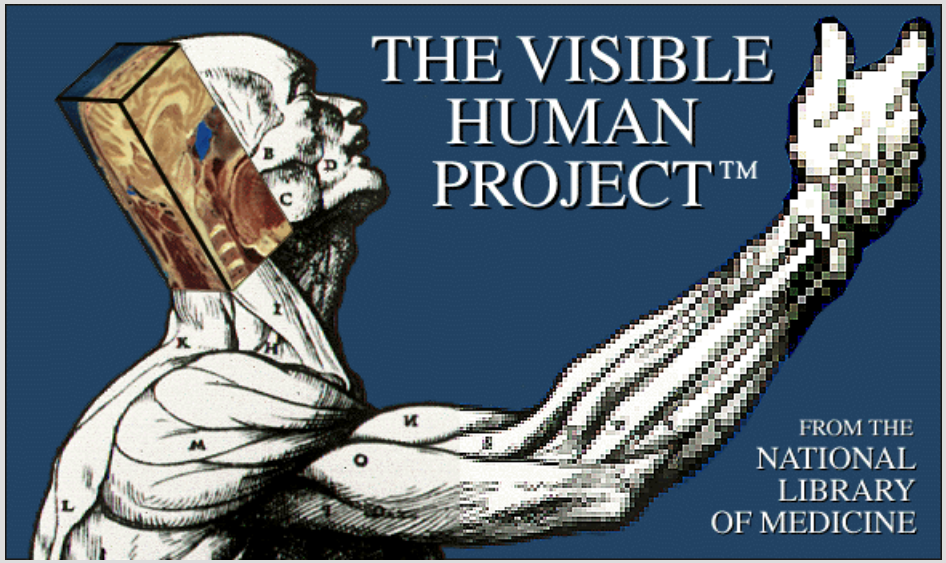
The engine

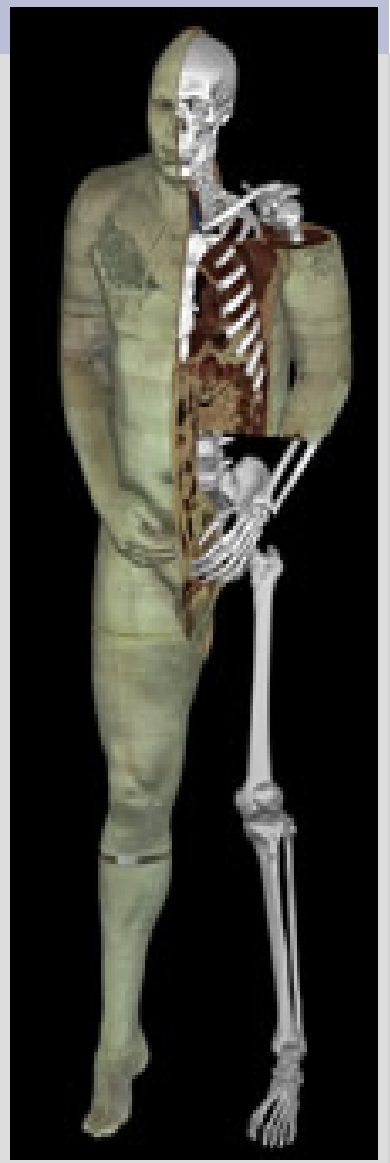


The XmasTree

The Visible Human project®

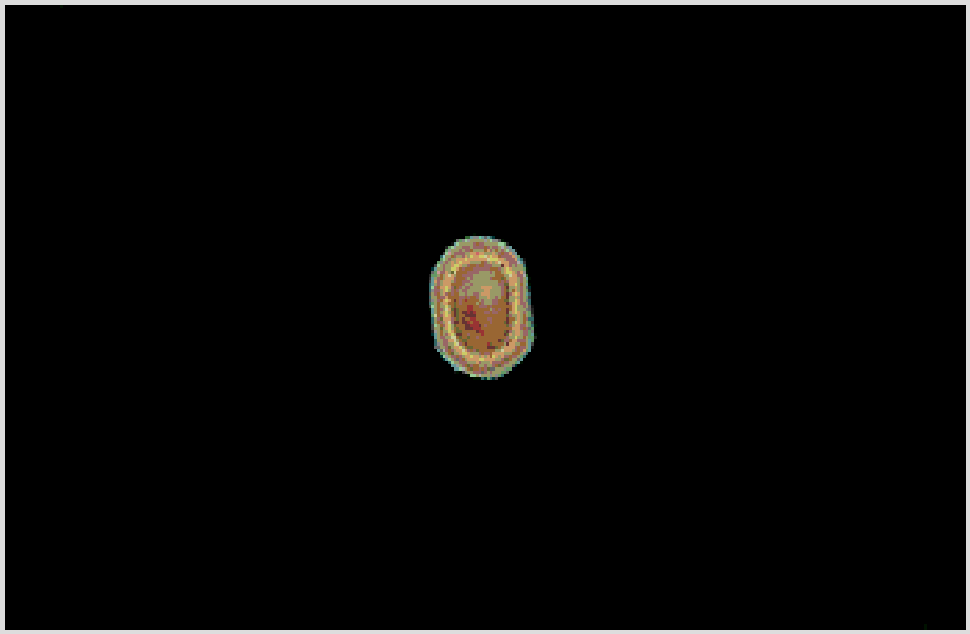
- Complete, anatomically detailed, three-dimensional representations of the normal male and female human bodies.
- Transverse CT, MR and cryosection images of representative male and female cadavers





The Visible Human project®

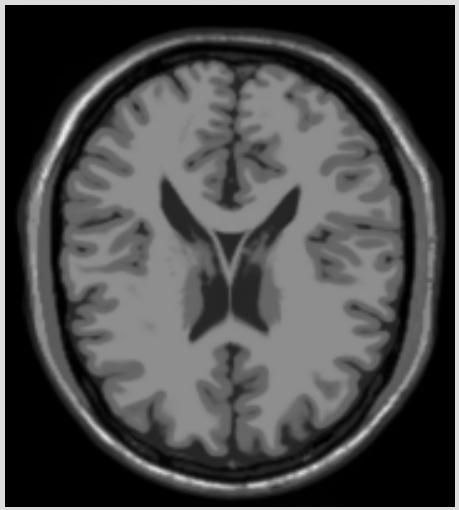
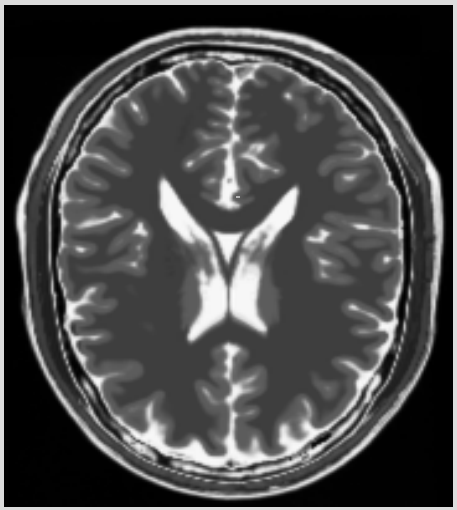
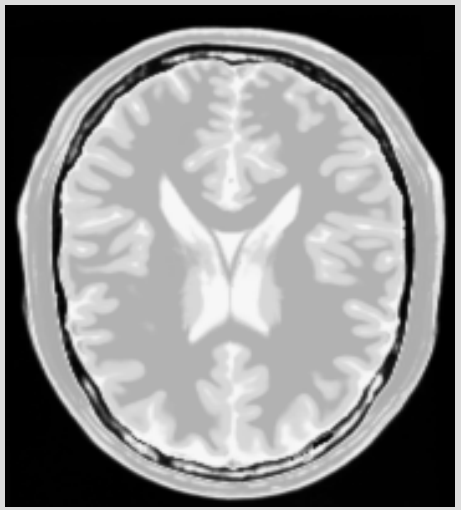
- Complete, anatomically detailed, three-dimensional representations of the normal male and female human bodies.
- Transverse CT, MR and cryosection images of representative male and female cadavers



Brainweb

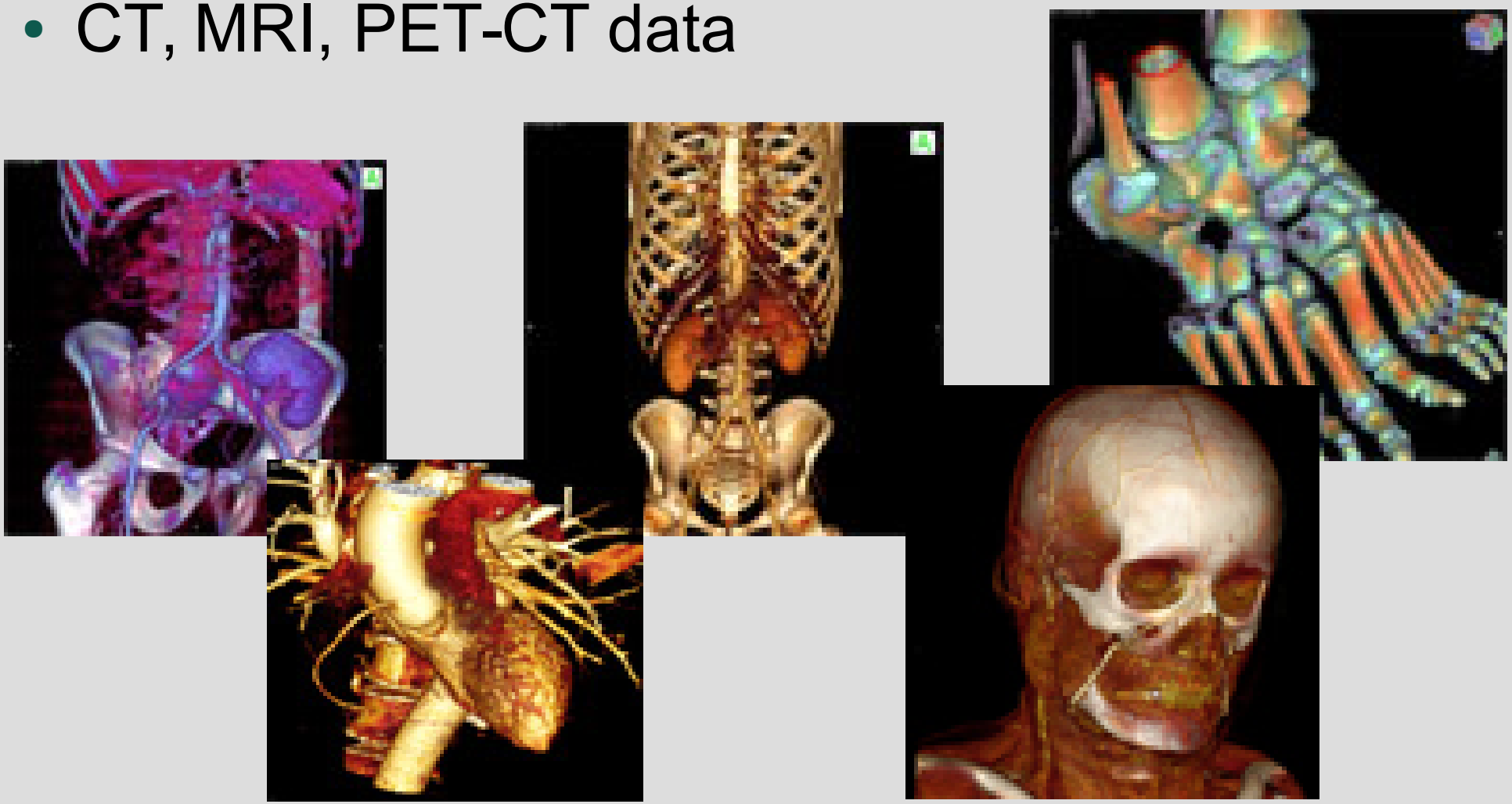
http://www.bic.mni.mcgill.ca/brainweb/

- Simulated Brain Database
  - Normal Brain Database
  - MS Lesion Brain Database
- Types
  - T1, T2, PD data
  - Noise 0 – X
  - Geometric distortion, segmented tissues ...

DICOM sample image sets

- http://www.osirix-viewer.com/datasets/
- CT, MRI, PET-CT data

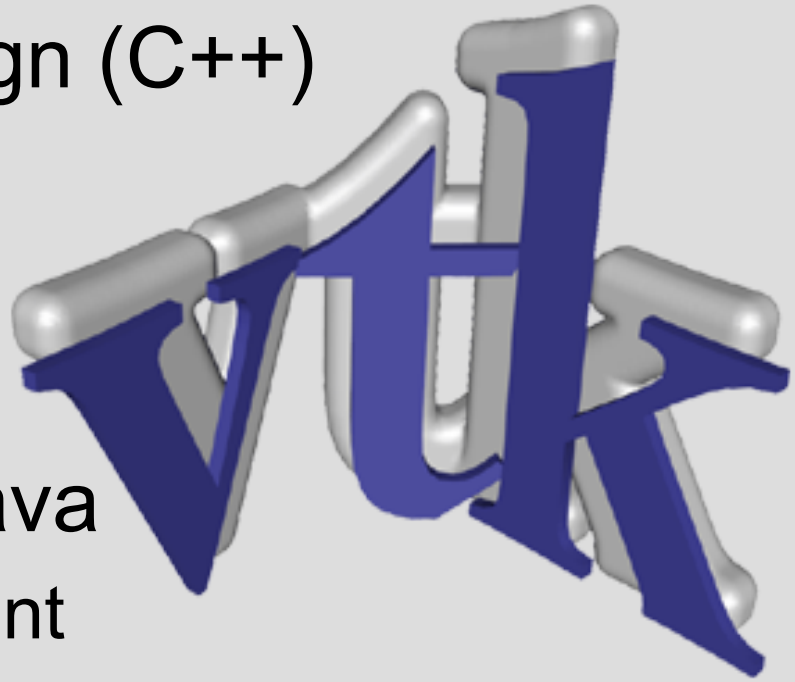


Freely available software

- Open source tools
- Commercial with free download

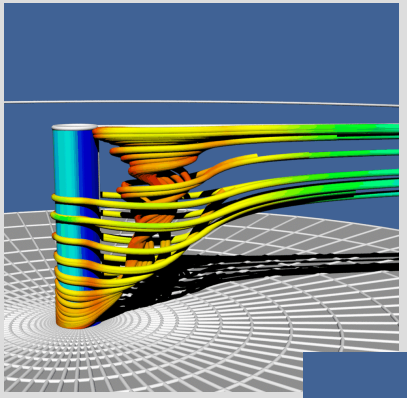
VTK – The Visualization ToolKit

- Open source, freely available software for 3D computer graphics, image processing, and visualization
- Managed by Kitware Inc.
- Strictly object-oriented design (C++)
  - 500.000 lines of code
  - Hundreds of classes
- High-level of abstraction
- Use C++, Tcl/Tk, Python, Java
  - Rapid application development

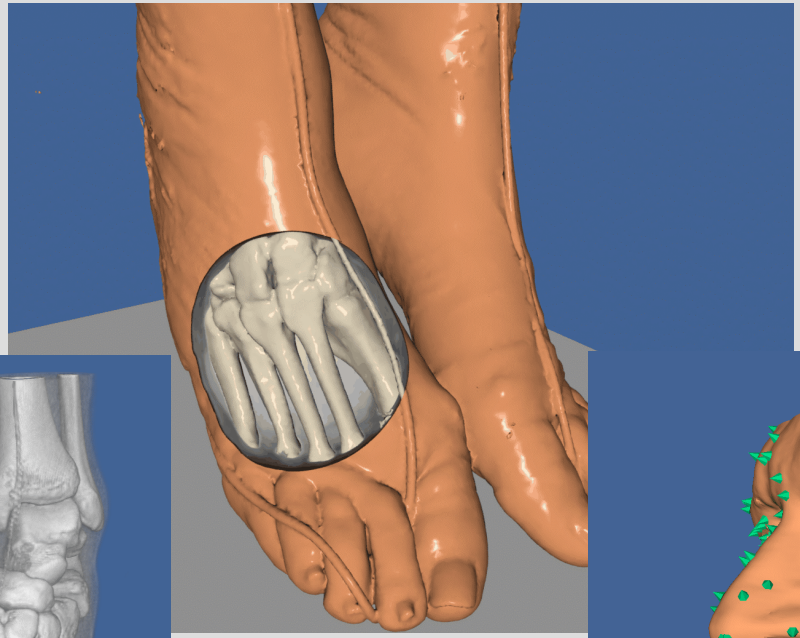


Example Applications

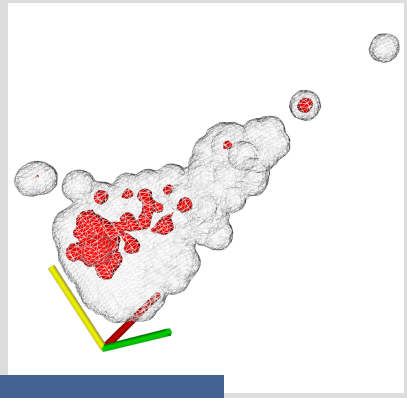
Scientific, medical, financial, information, ...



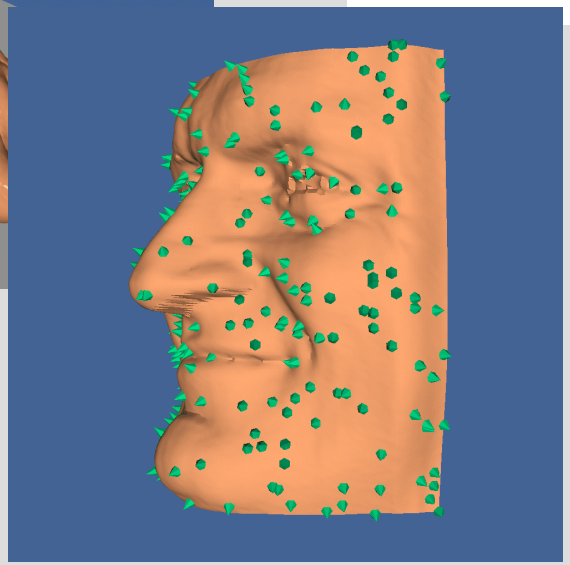
Simulation



Medical
CT / MRI / Ultrasound



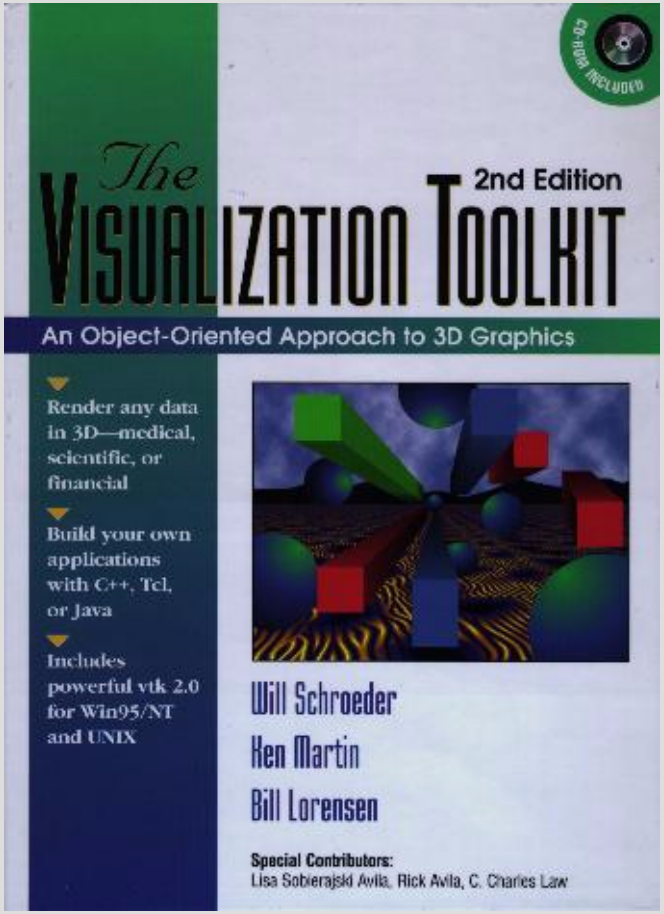
Business



Modeling

Textbook

VTK: accompanying SW for the book:



The Visualization Toolkit
An Object-Oriented Approach To 3D Graphics
Will Schroeder, Ken Martin, Bill Lorensen
ISBN 0-13-954694-4
Prentice Hall

Work on first edition began in 1994

VTK features

- Visualization techniques for visualizing
  - Scalar fields (medical, tomographic data)
  - Vector fields
  - Tensor fields
- Polygon reduction
- Mesh smoothing
- Image processing
- Volume rendering
- Custom algorithms

3D graphics

- Surface rendering
- Volume rendering
  - Ray casting
  - Texture mapping (2D)
  - Volume pro support
- Lights and cameras
- Textures
- Save render window to .png, .jpg, ... (useful for movie creation)

Visualization with VTK

- Scalar algorithms
  - Iso-contouring
  - Volume rendering with transfer functions
- Vector algorithms
  - Hedgehogs
  - Streamlines / streamtubes
- Tensor algorithms
  - Tensor ellipsoids

Image processing

- Supports streaming => huge datasets
- vtkImageToImageFilter
  - Diffusion
  - High-pass / Low-pass (Fourier)
  - Convolution
  - Gradient (magnitude)
  - Distance map
  - Morphology
  - Skeletons

Summary +

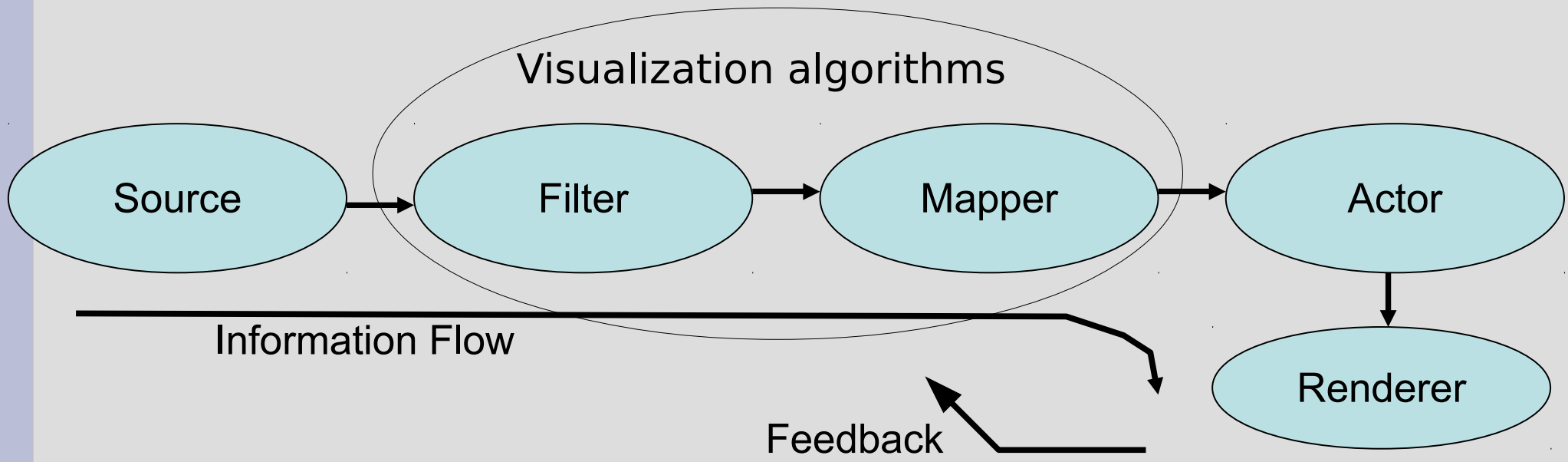
- Free and open source
- Create graphics/visualization applications fairly fast
- Object oriented - easy to derive new classes
- Build applications using "interpretive" languages Tcl, Python, and Java
- Many (state of the art) algorithms
- Heavily tested in real-world applications
- Large user base provides decent support
- Commercial support and consulting available

Summary -

- Not a super-fast graphics engine due to portability and C++ dynamic binding – a decent workstation is needed
- Large memory demands
- Very large class hierarchy => learning threshold might be steep

How does VTK work?

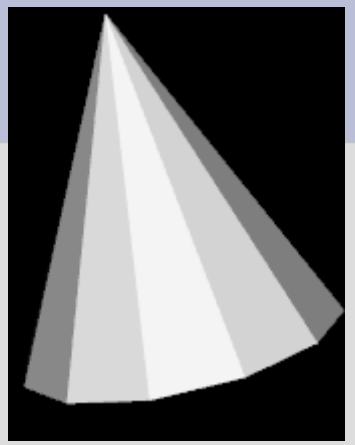
- Uses a pipeline execution to pass the data through various objects.
- Implements a lazy processing scheme (waits for data to be needed to process)

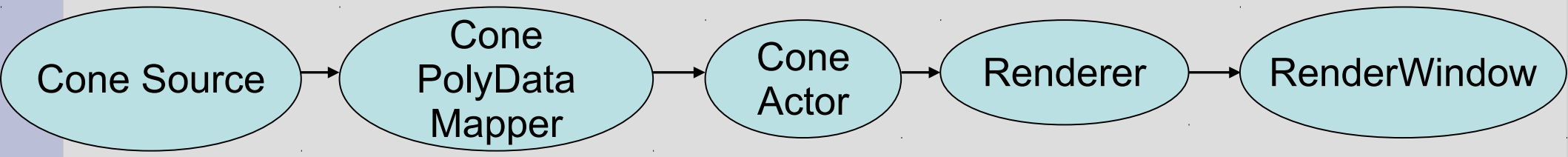


VTK objects and Abstraction

- Abstract VTK objects are used in order to make the subsequent inherited objects more uniform. Some inherited objects include
  - vtkData
  - VtkMapper
  - vtkProp3D (vtkActor, vtkVolume)
  - vtkCamera
  - vtkLight
  - vtkRenderer
  - vtkRenderWindow
  - vtkRenderWindowInteractor

VTK Pipeline example – a cone model





Some form of source data is required, in this case it is generated by a formula in the shape of a cone

The source data is mapped into 2D info and passed on

The actor is a global information storage unit for global manipulation and reference

The Rendered takes the mapped 3D data and adds in light, motion sound and camera position

The Render Window takes the current image and puts it into your monitor within a given frame

VTK by Example

Python Cone Example

```
#!/usr/bin/env python
from vtkpython import *

cone = vtkConeSource()
cone.SetHeight( 3.0 )
cone.SetRadius( 1.0 )
cone.SetResolution( 10 )

coneMapper = vtkPolyDataMapper()
coneMapper.SetInput( cone.GetOutput() )
coneActor = vtkActor()
coneActor.SetMapper( coneMapper )

ren1= vtkRenderer()
ren1.AddActor( coneActor )
ren1.SetBackground( 0.1, 0.2, 0.4 )
```

```
renWin = vtkRenderWindow()
renWin.AddRenderer( ren1 )
renWin.SetSize( 300, 300 )
for i in range(0,360):
    renWin.Render()
    ren1.GetActiveCamera().Azimuth( 1 )
```

VTK by Example

C++ Cone Example

```
#include "vtkConeSource.h"
#include "vtkPolyDataMapper.h"
#include "vtkRenderWindow.h"

int main( int argc, char *argv[] )
{
  vtkConeSource *cone = vtkConeSource::New();
  cone->SetHeight( 3.0 );
  cone->SetRadius( 1.0 );
  cone->SetResolution( 10 );

  vtkPolyDataMapper *coneMapper =
         vtkPolyDataMapper::New();
  coneMapper->SetInput( cone->GetOutput() );

  vtkActor *coneActor = vtkActor::New();
  coneActor->SetMapper( coneMapper );

  vtkRenderer *ren1= vtkRenderer::New();
  ren1->AddActor( coneActor );
  ren1->SetBackground( 0.1, 0.2, 0.4 );
```

```
  vtkRenderWindow *renWin = vtkRenderWindow::New();
  renWin->AddRenderer( ren1 );
  renWin->SetSize( 300, 300 );

  for (int i = 0; i < 360; ++i)
    {
    // render the image
    renWin->Render();
    // rotate the active camera by one degree
    ren1->GetActiveCamera()->Azimuth( 1 );
    }

  return 0;
}
```

VTK by Example

Java Cone Example

```
// we import the vtk wrapped classes first
import vtk.*;

// then we define our class
public class Cone {
  static {
    System.loadLibrary("vtkCommonJava");
    System.loadLibrary("vtkFilteringJava");
    System.loadLibrary("vtkIOJava");
    System.loadLibrary("vtkImagingJava");
    System.loadLibrary("vtkGraphicsJava");
    System.loadLibrary("vtkRenderingJava");
  }
  public static void main (String []args) {
    vtkConeSource cone = new vtkConeSource();
    cone.SetHeight( 3.0 );
    cone.SetRadius( 1.0 );
    cone.SetResolution( 10 );

    vtkPolyDataMapper coneMapper = new
vtkPolyDataMapper();
    coneMapper.SetInput( cone.GetOutput() );
```

```
  vtkActor coneActor = new vtkActor();
    coneActor.SetMapper( coneMapper );

    vtkRenderer ren1 = new vtkRenderer();
    ren1.AddActor( coneActor );
    ren1.SetBackground( 0.1, 0.2, 0.4 );

    vtkRenderWindow renWin = new vtkRenderWindow();
    renWin.AddRenderer( ren1 );
    renWin.SetSize( 300, 300 );

    int i;
    for (i = 0; i < 360; ++i)
      {
      // render the image
      renWin.Render();
      // rotate the active camera by one degree
      ren1.GetActiveCamera().Azimuth( 1 );
      }

    }
}
```

VTK by Example

TCL Cone Example

```
package require vtk

vtkConeSource cone
cone SetHeight 3.0
cone SetRadius 1.0
cone SetResolution 10

vtkPolyDataMapper coneMapper
coneMapper SetInput [cone GetOutput]

vtkActor coneActor
coneActor SetMapper coneMapper

vtkRenderer ren1
ren1 AddActor coneActor
ren1 SetBackground 0.1 0.2 0.4
```

```
 vtkRenderWindow renWin
renWin AddRenderer ren1
renWin SetSize 300 300

for {set i 0} {$i < 360} {incr i} {
   # render the image
   renWin Render
   # rotate the active camera by one
degree
   [ren1 GetActiveCamera] Azimuth 1
}

vtkCommand DeleteAllObjects

exit
```

Demo1 – the cone

- cone-rot.py
- cone-int.py

Surface rendering of Volume data: the MC algorithm

- Replace object definition by
  - Reading the data
  - Surface tilling by the MC algorithm

```
# data reader
reader = vtkImageReader()
reader.SetFileDimensionality(3)
reader.SetFileName("tot2-170x190x184.raw")
reader.SetDataScalarTypeToUnsignedChar()
reader.SetDataExtent(0,169,0,189,0,150)

# marching cubes
isosurface = vtkMarchingCubes()
isosurface.SetInput( reader.GetOutput() )
isosurface.GenerateValues(1, 90, 255 )
```

Demo2 – triangular model

- iso-mc.py
- iso-mc2.py

Volume rendering (1)

- Requires
  - Specification of transfer functions and setting of vtkVolumeProperty:

```
# Create transfer mapping scalar value to color
colorTransferFunction = vtkColorTransferFunction()
colorTransferFunction.AddRGBPoint(0.0, 0.0, 0.0, 0.0)
colorTransferFunction.AddRGBPoint(64.0, 1.0, 0.0, 0.0)
....
# Create transfer mapping scalar value to opacity
opacityTransferFunction = vtkPiecewiseFunction()
opacityTransferFunction.AddPoint(20, 0.0)
opacityTransferFunction.AddPoint(255, 0.2)

# The property describes how the data will look
volumeProperty = vtkVolumeProperty()
volumeProperty.SetColor(colorTransferFunction)
volumeProperty.SetScalarOpacity(opacityTransferFunction)
```

Volume rendering with texture mapping (2)

- Requires
  - Mapper: data -> texture mapping

```
volumeMapper = vtk.vtkVolumeTextureMapper2D()
volumeMapper.SetInputConnection(reader.GetOutputPort())
```

  - Create vtkVolume and add it the mapper and property

```
volume = vtk.vtkVolume()
volume.SetMapper(volumeMapper)
volume.SetProperty(volumeProperty)
```

  - Add volume to the renderer (instead of an actor)

```
ren.AddVolume(volume)
```

Modifications of volume rendering

- VR by ray casting – replace mapper

```
blendFunction = vtkVolumeRayCastCompositeFunction()
volumeMapper = vtkVolumeRayCastMapper()
volumeMapper.SetVolumeRayCastFunction(blendFunction)
volumeMapper.SetInputConnection(reader.GetOutputPort())
```

- MIP by ray casting – replace the composite function

```
blendFunction = vtkVolumeRayCastMipFunction()
```

Demo3 – volume rendering

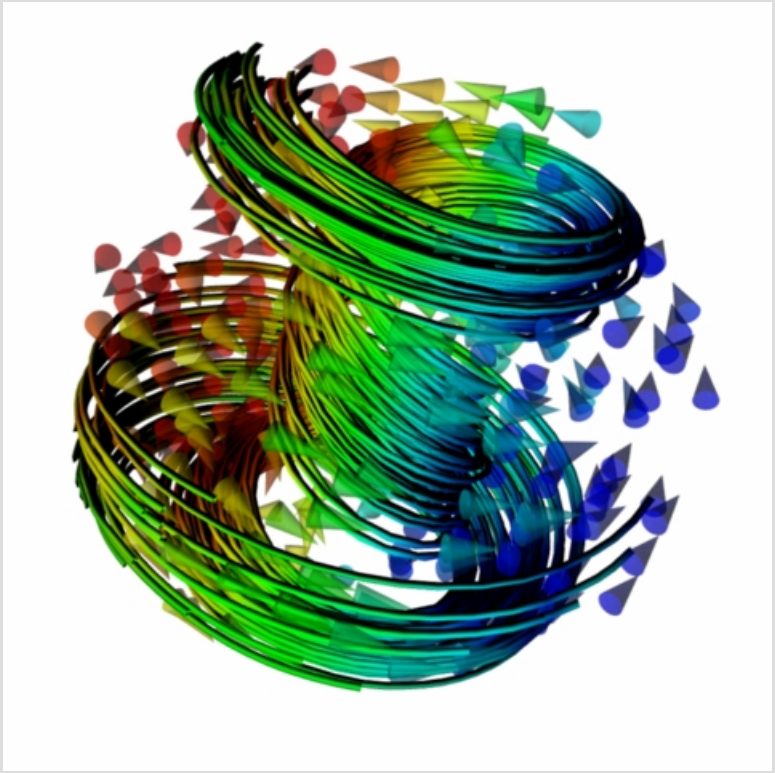
- With texture mapping
  - volrend-texture.py
- With ray casting
  - volrend-ray.py
- MIP by ray casting
  - volrend-ray-mip.py

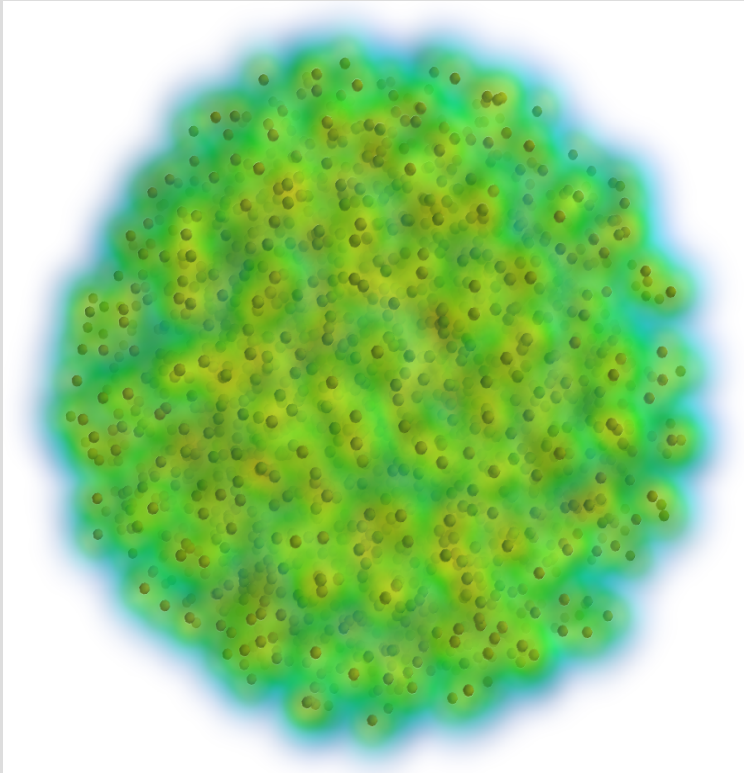
Applications using VTK

- As opensource SW it can be used by
  - Open source applications
  - Free software applications
  - Commercial applications
- Why to write applications:
  - Hide the complexity of programing by a GUI
  - Provide ready-to-use tools

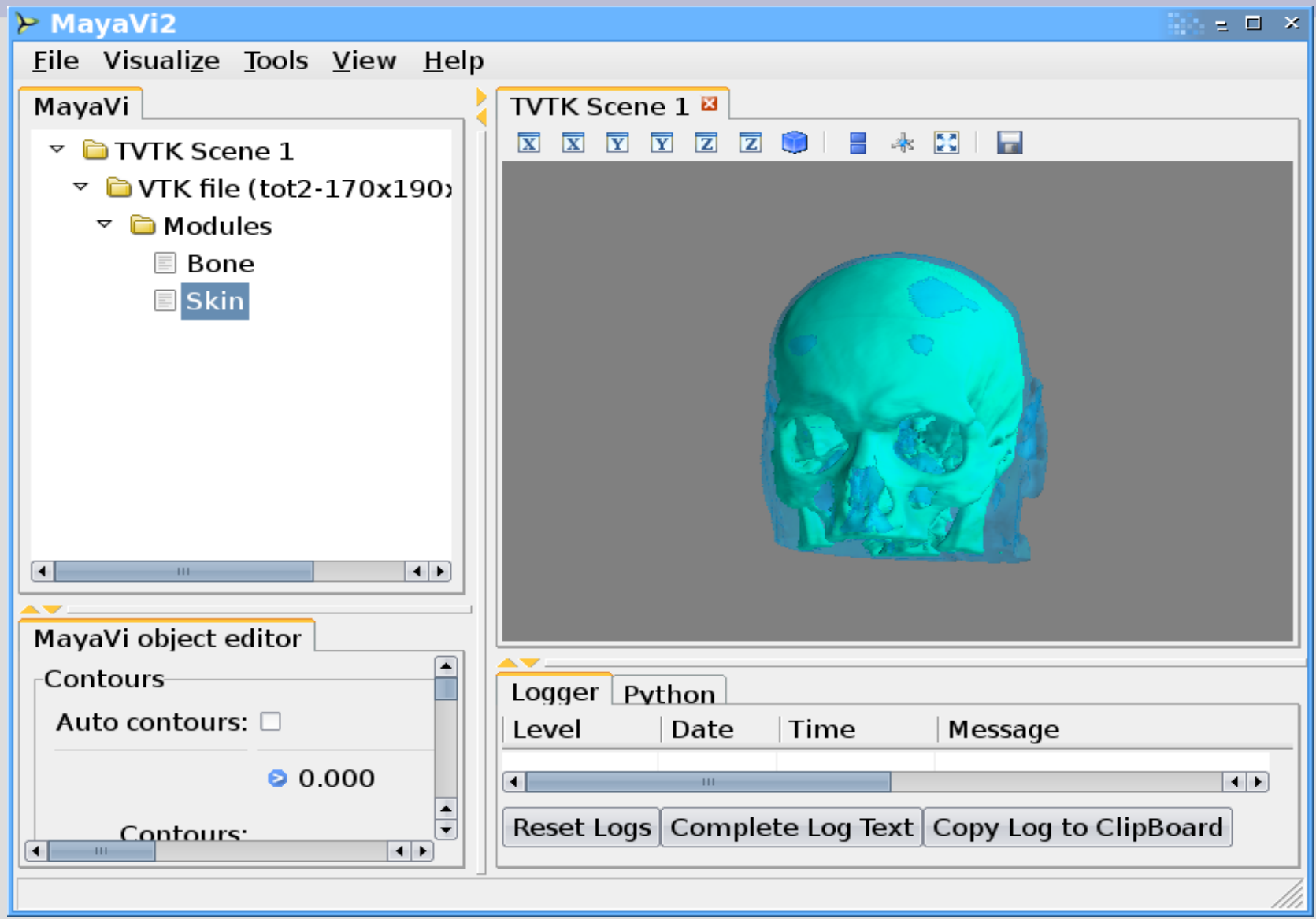
Mayavi2

- Easy and interactive visualization of 3D data
- A simple and clean scripting interface in Python
- Use the power of the VTK toolkit without forcing you to learn it.



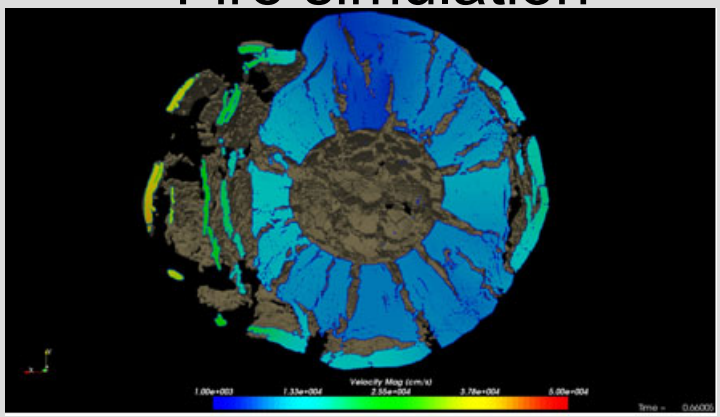


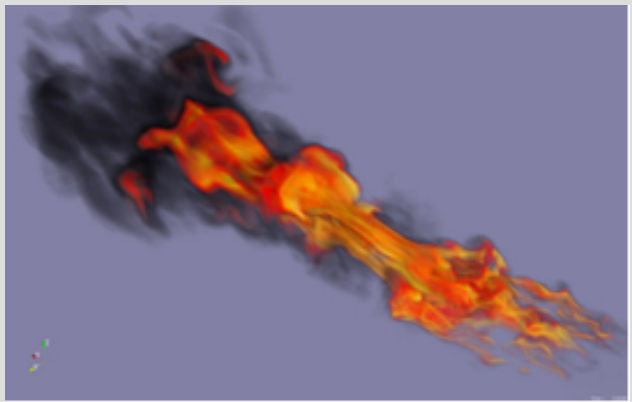
Demo


MayaVi2
File Visualize Tools View Help
MayaVi
TVTK Scene 1
VTK file (tot2-170x190
Modules
Bone
Skin
TVTK Scene 1
MayaVi object editor
Contours
Auto contours:
0.000
Contours:
Logger
Python
Level
Date
Time
Message
Reset Logs
Complete Log Text
Copy Log to ClipBoard

Paraview (1)

- VTK-based
- Visualization of large data sets in a parallel environment
- Typical usage: visualization of simulation results
- Example: Sandia National Lab
  - Visualization cluster 250+ nodes
  - Example tasks:
    - Simulated asteroid explosion
    - Fire simulation

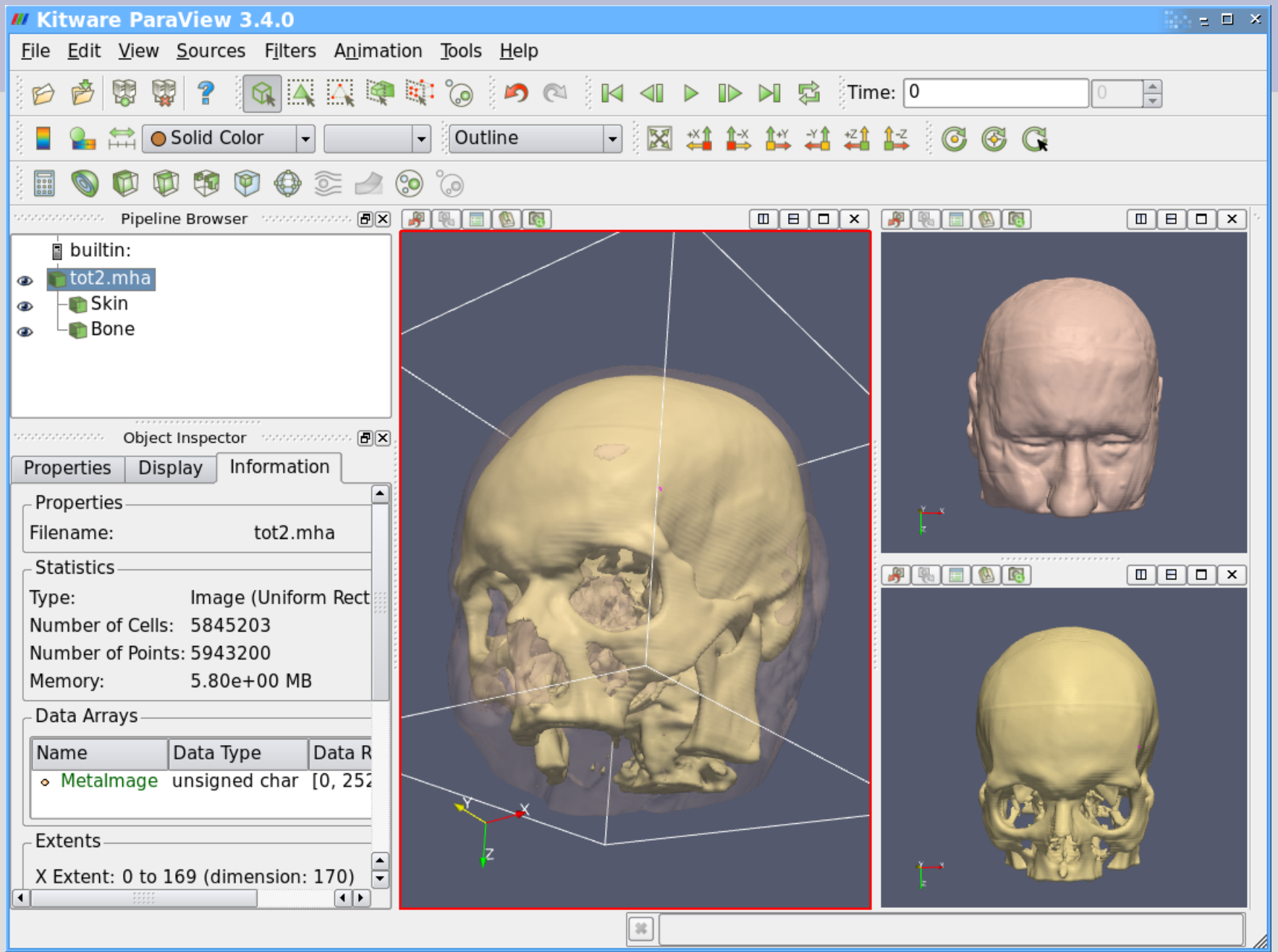




Paraview (2)

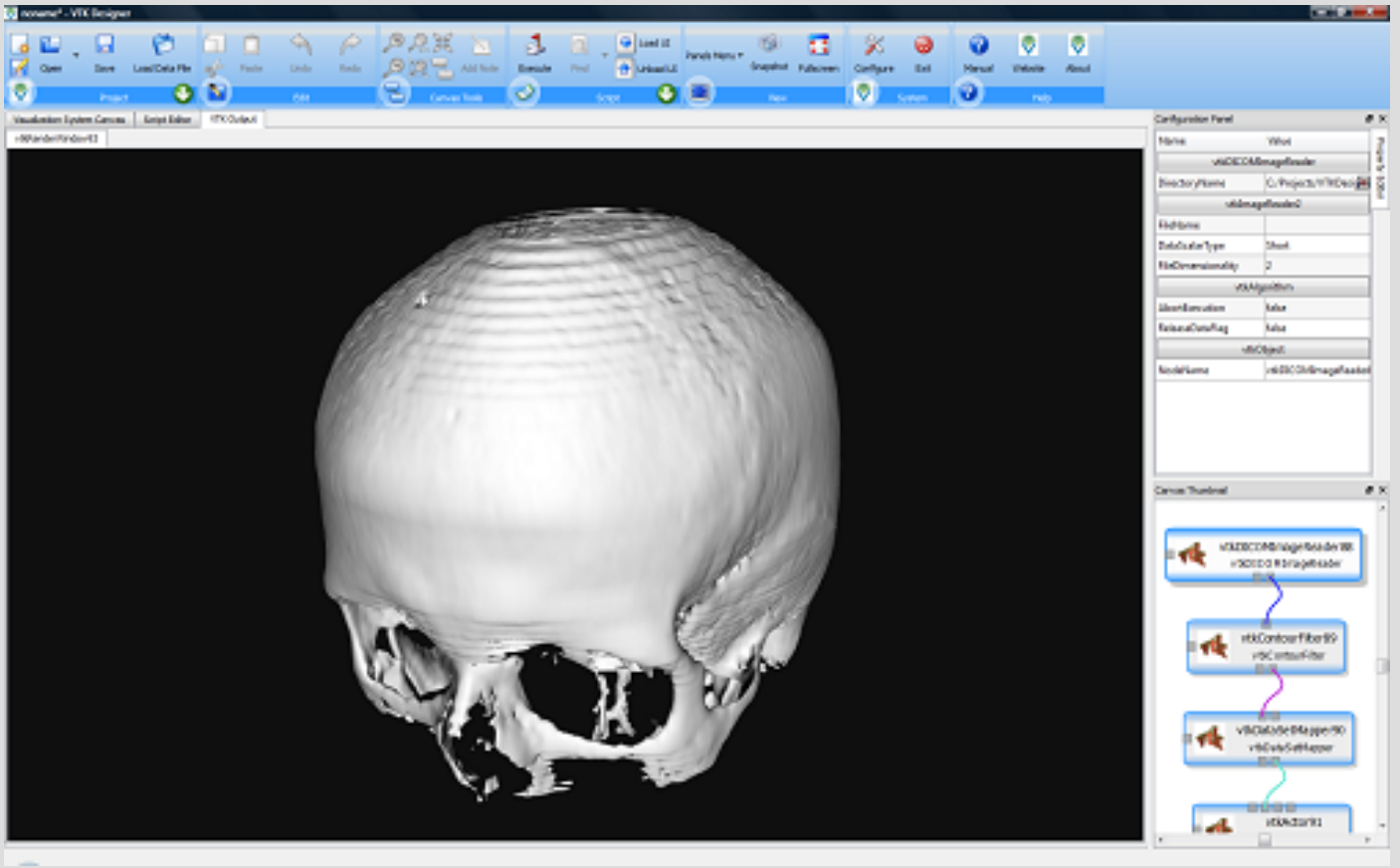
- Web:
  - Page: www.paraview.org
  - Download:
    http://www.paraview.org/paraview/resources/software.html
    (statically linked executables)

Demo

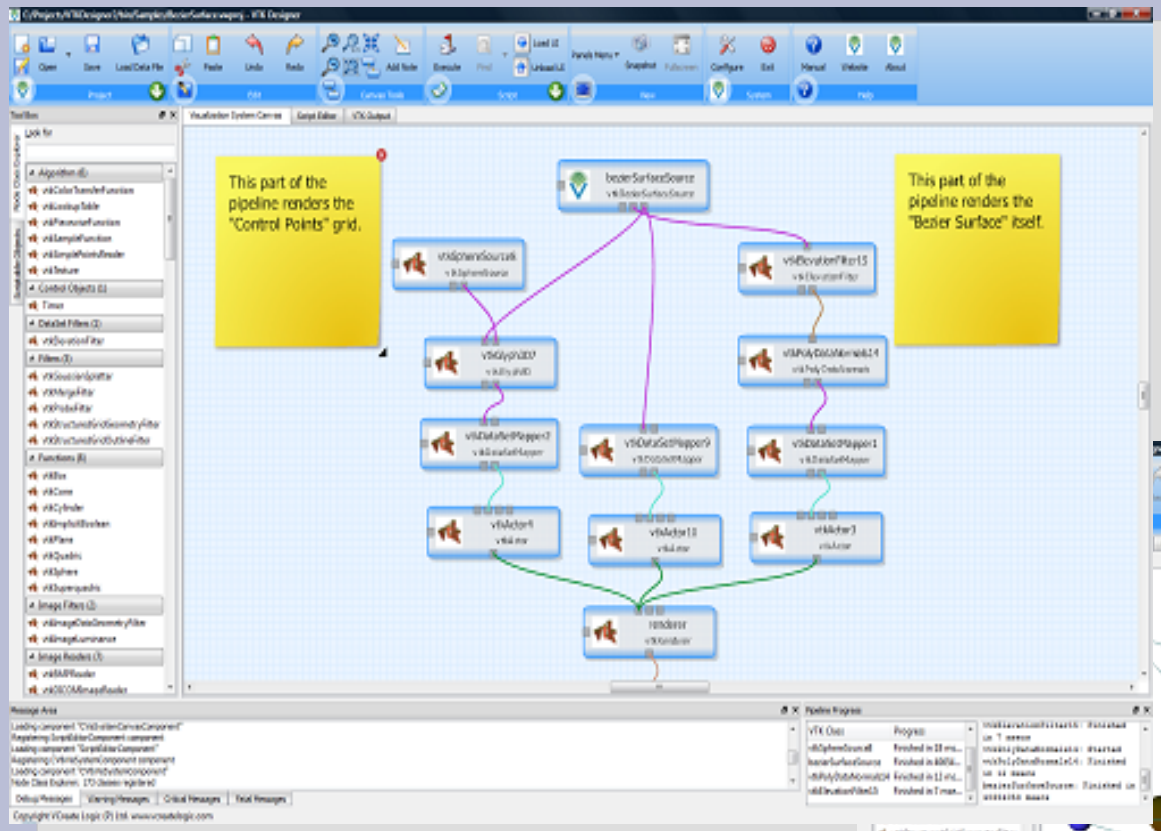

Kitware ParaView 3.4.0
File Edit View Sources Filters Animation Tools Help
Time: 0
Solid Color
Outline
Pipeline Browser
builtin:
tot2.mha
Skin
Bone
Object Inspector
Properties Display Information
Properties
Filename: tot2.mha
Statistics
Type: Image (Uniform Rect
Number of Cells: 5845203
Number of Points: 5943200
Memory: 5.80e+00 MB
Data Arrays
Name Data Type Data R
MetaImage unsigned char [0, 252
Extents
X Extent: 0 to 169 (dimension: 170)

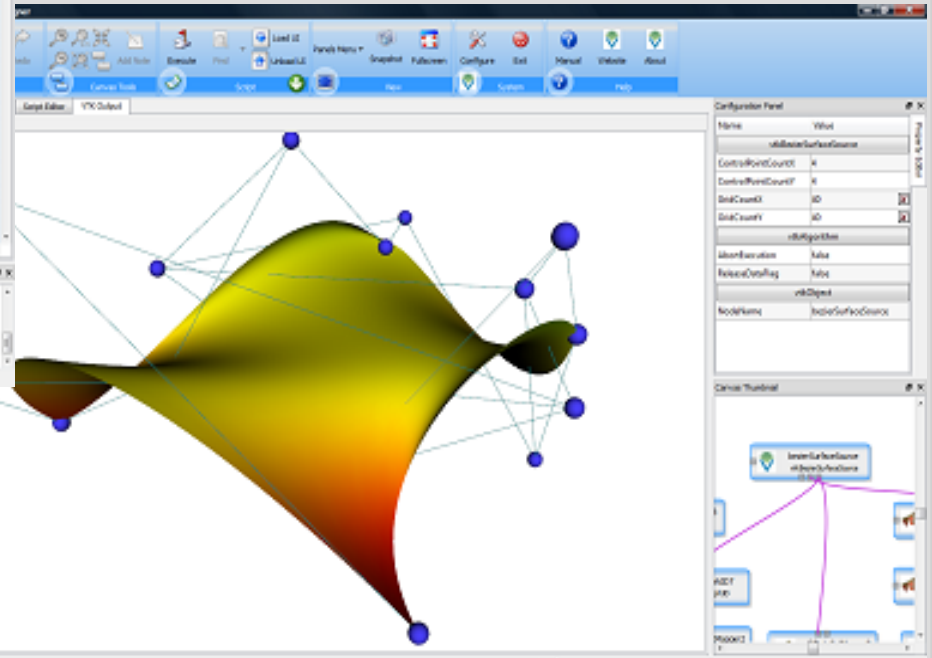
VTK Designer

- Graphical Tool for constructing VTK Pipelines
- Learning Tool to help understand pipelines
- RAD tool to rapidly construct VTK solutions



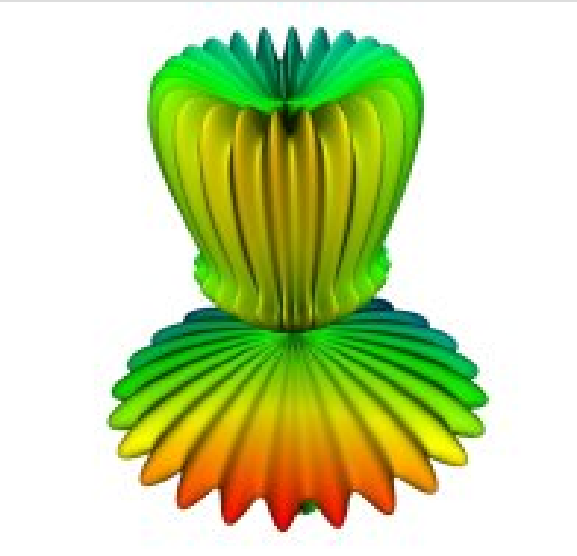
VTK-designer networks





OctaViz

- A visualization system for Octave.
- Wrapper making VTK classes accessible from within Octave
- High-level functions for 2D and 3D visualization.
  - no knowledge about VTK necessary



Spherical harmonics

```
[phi,theta] = meshgrid(0:pi/250:pi,0:pi/250:2*pi);
m0 = 4; m1 = 3; m2 = 2; m3 = 3;
m4 = 6; m5 = 2; m6 = 6; m7 = 4;
r = sin(m0*phi).^m1 + cos(m2*phi).^m3 +
            sin(m4*theta).^m5 + cos(m6*theta).^m7;
x = r .* sin(phi) .* cos(theta);
y = r .* cos(phi);
z = r .* sin(phi) .* sin(theta);
vtk_surf(x,y,z);
```

More SW using VTK

- http://www.vtk.org/Wiki/VTK_Tools

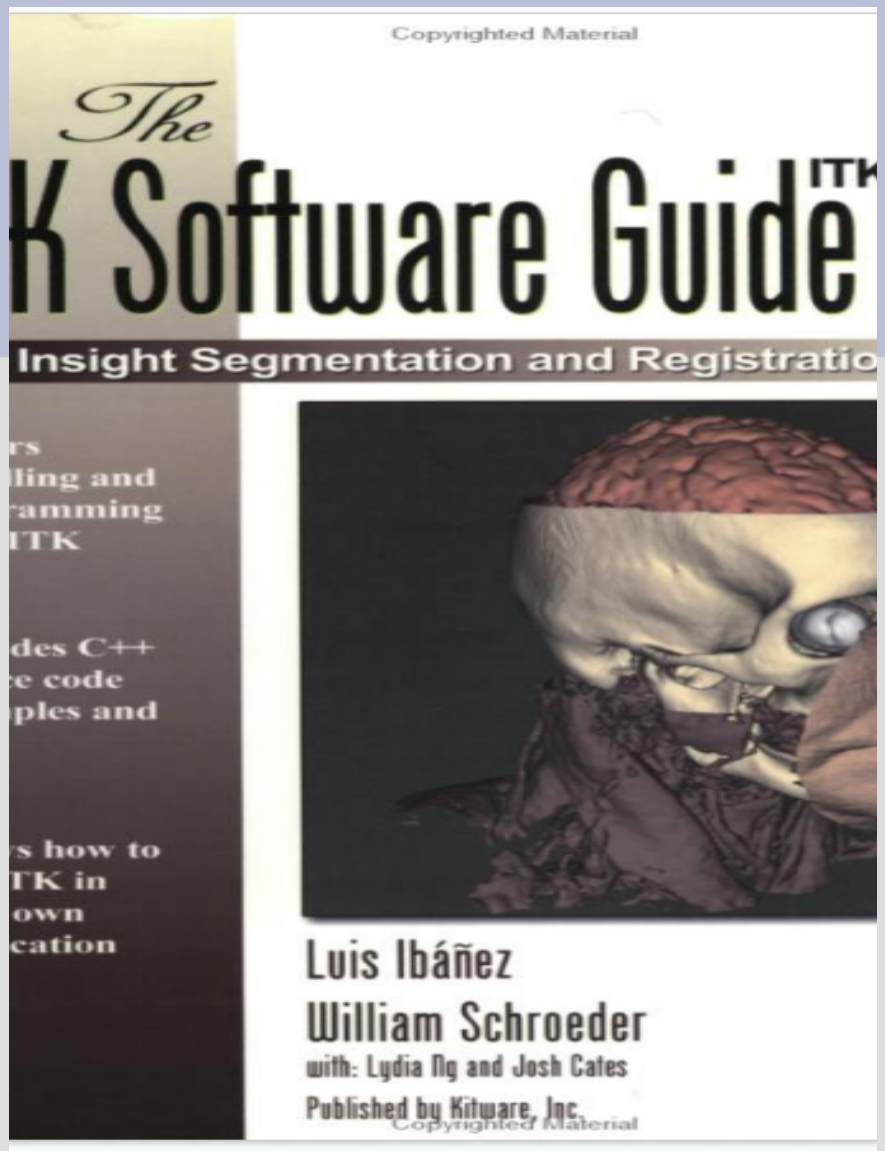
The Insight toolkit (ITK)

- Image Processing
- Segmentation
- Registration
- No Graphical User Interface (GUI)
- No Visualization (use VTK)

ITK by the Numbers

- Public Investment
  - $13 Million
- # of platforms (software + hardware): 42
- # of C++ classes: 1,647
- # of lines of code (Insight / Code directory): 136K
- # of lines of test code (Insight / Testing directory): 102K
- # of lines of examples (Insight / Examples directory): 27K
-

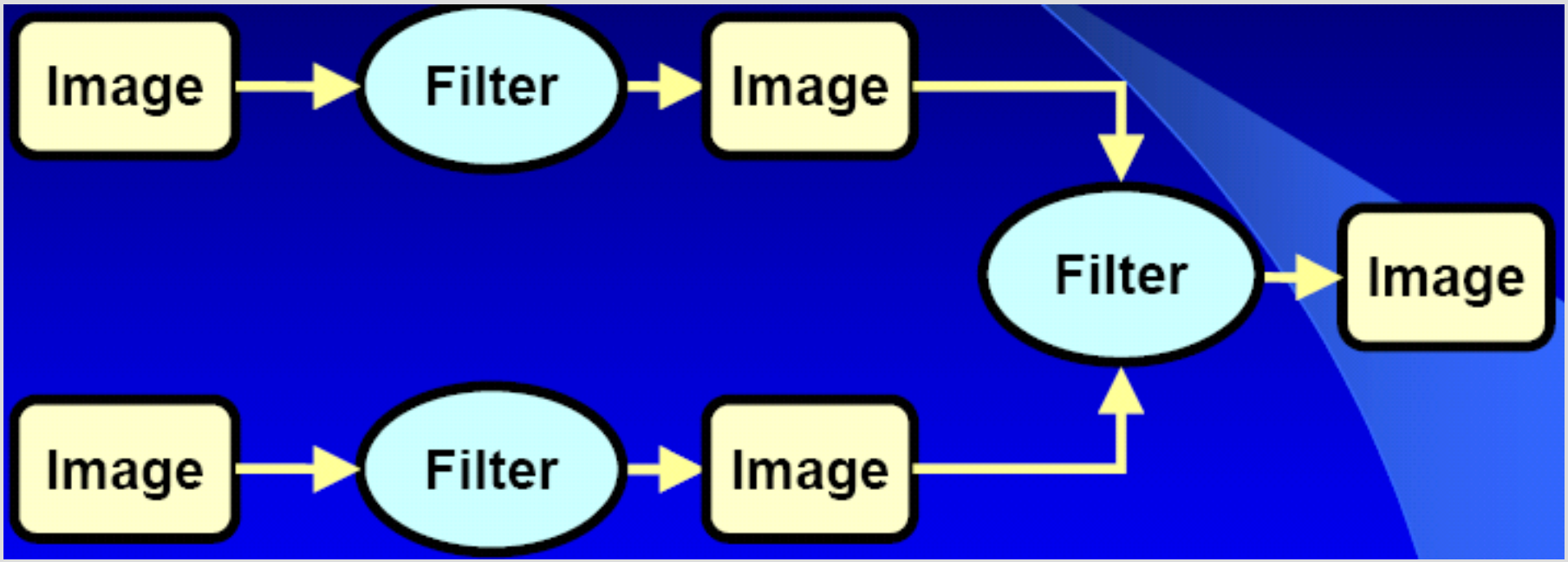
Documentation



- http://www.itk.org/ItkSoftwareGuide.pdf
  - 800 pages
- http://www.itk.org/Doxygen/html/index.html

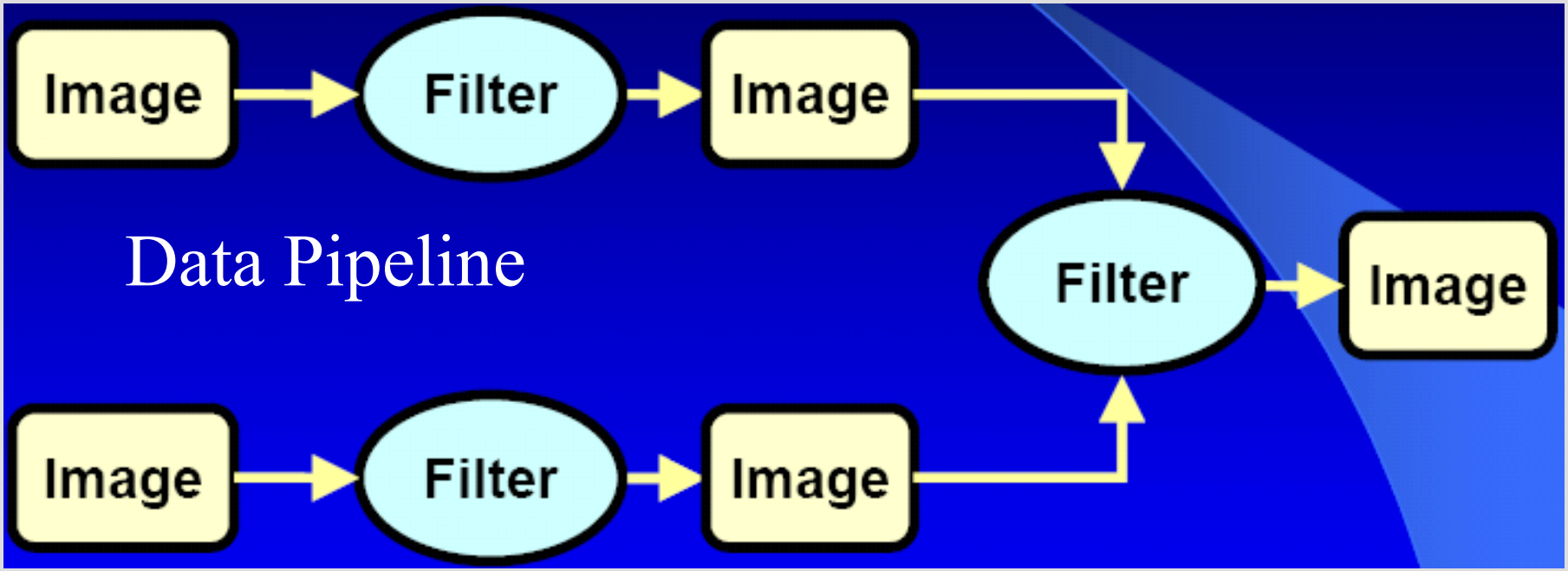
Concept

- Image: N-dimensional



Data Pipeline

Basic Image Processing

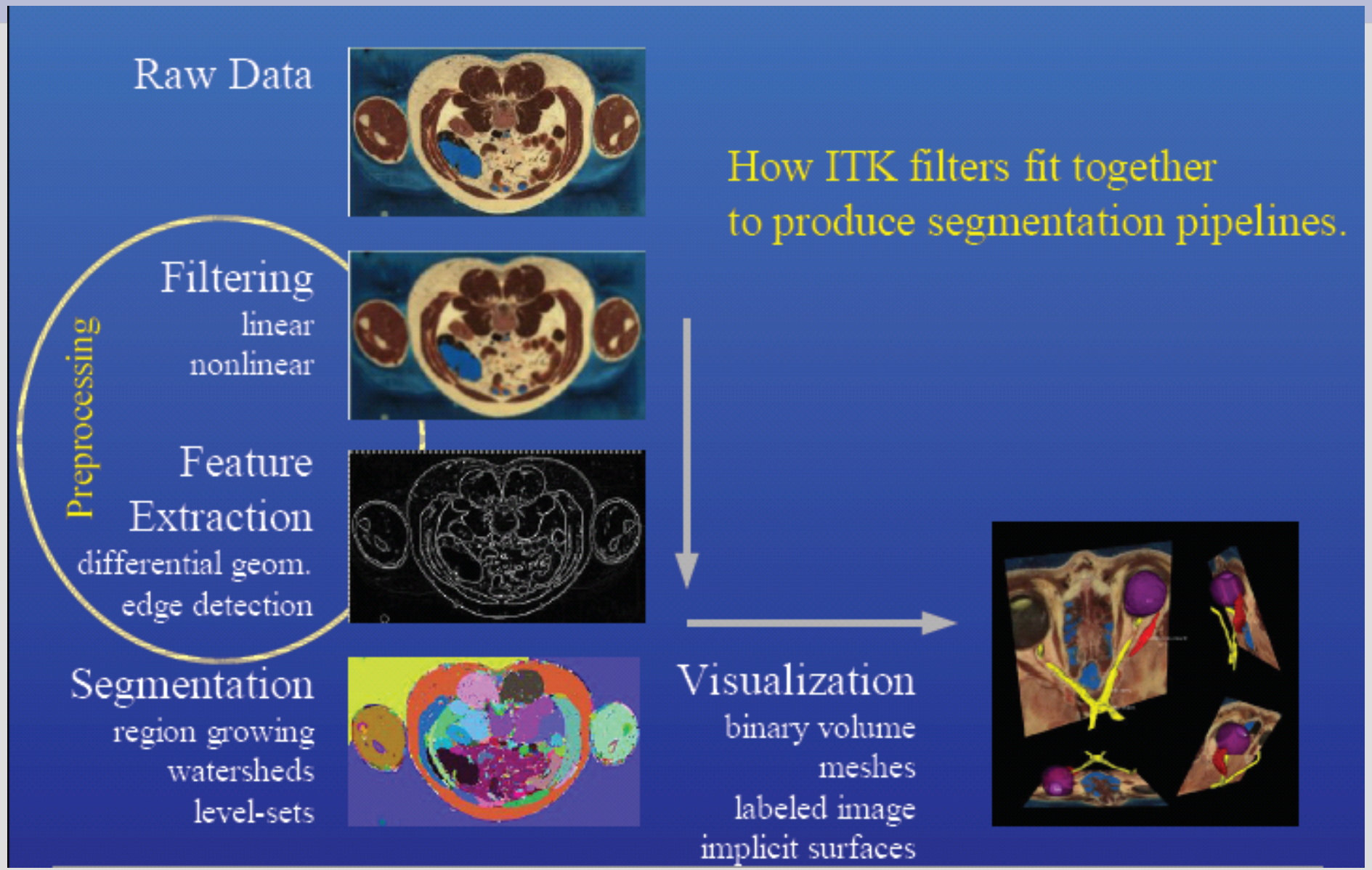


e.g. Threshold, Casting, Intensity Mapping, Gradient, Mean, Median, Binary & Grayscale Morphology, (Recursive) Gaussian-Blur, Canny Edge Detect, Laplacian, Anisotropic Diffusion, Bilateral Filtering, DistanceMap, Image Resampling, …

Segmentation

- ***Partitioning images into meaningful pieces, e.g. delineating regions of anatomical interest.***
  - Edge based – find boundaries between regions
  - Pixel Classification – metrics classify regions
  - Region based – similarity of pixels within a segment

Segmentation Pipeline

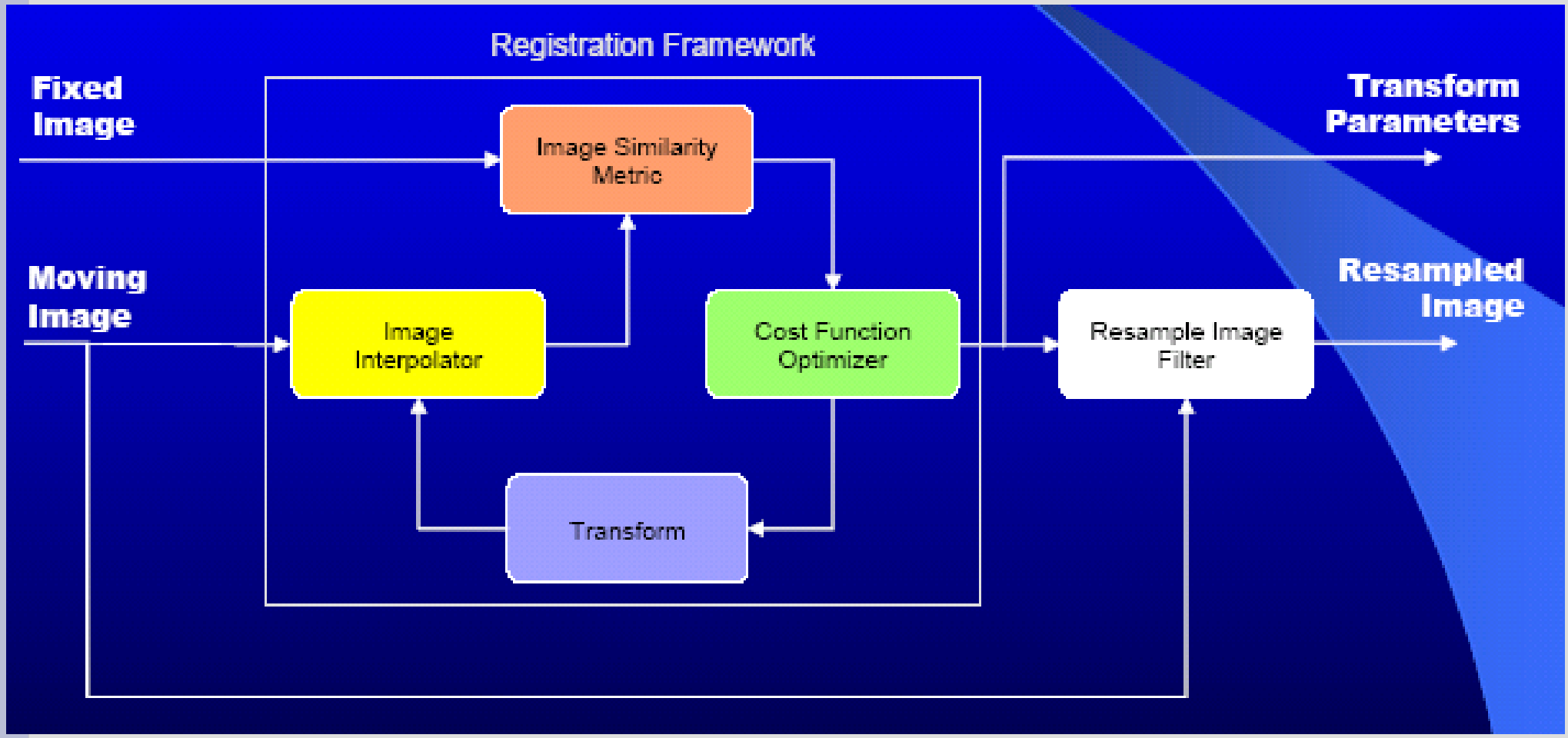


Registration Framework

- Find transformation mapping homologous points into each other
- Rigid & nonrigid
- Many medical applications:
  - Time series registration
  - Multi-modality image fusion (MR/CT – SPECT, …)
  - Atlas construction (for segmentation)

Registration Framework

- Components:



Conclusion

- Very useful for rapid prototyping
- Strongly growing community and code base
- Problems:
  - Very complex
  - Overhead -> higher run-times
  - Still under development
- Simplified version for Python: SimpleITK
  - Hides data type

Applications Using ITK

- Numerous
  - MITK (open source)
  - VolView (commercial)
  - MeVisLab (freeware)
  - SciRun (open source)