methods of computational science

visualization

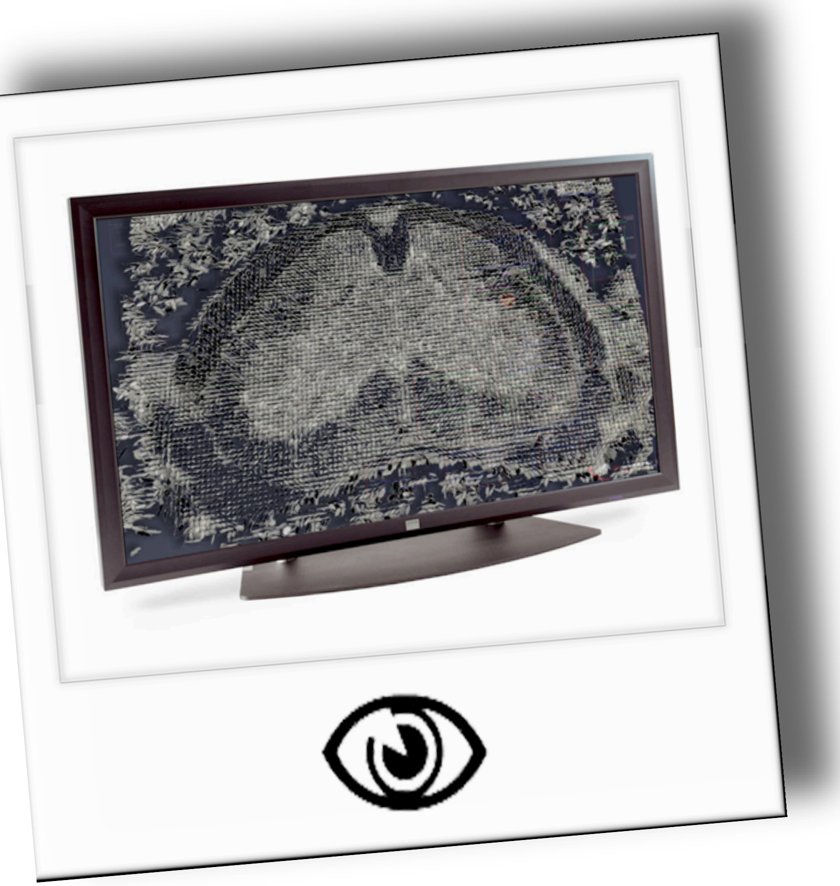
day i - intro/infoviz

santiago v lombeyda

center for advanced computing research

caltech

what *includes* visualization



data

pantom omni

more understandable representation

finding solution(s) via purpose

- for what purpose:
  - quick view/demonstration
    - we want to look at/show something particular
  - analysis
    - we know what we are looking for
  - research
    - we do not know what we are looking for
  - debugging
    - we want to assure there is nothing odd
  - …

visualization =

science

+

computer graphics/hci

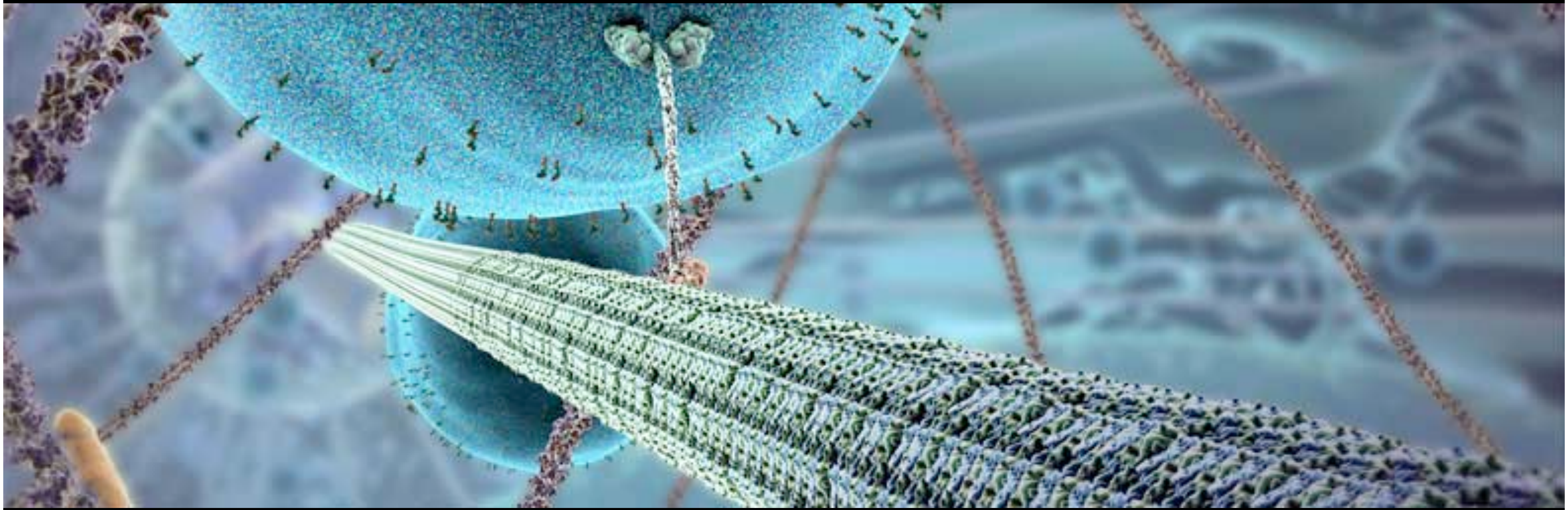
+

graphic design/(art?)

UNREALISTIC

sample:

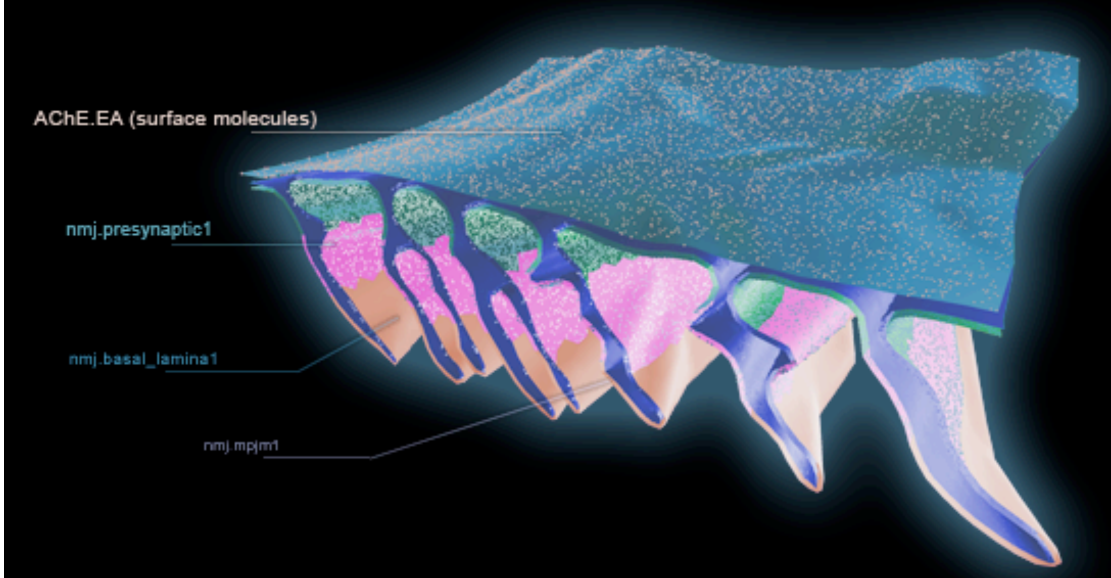
HARVARD’S BIO VISIONS

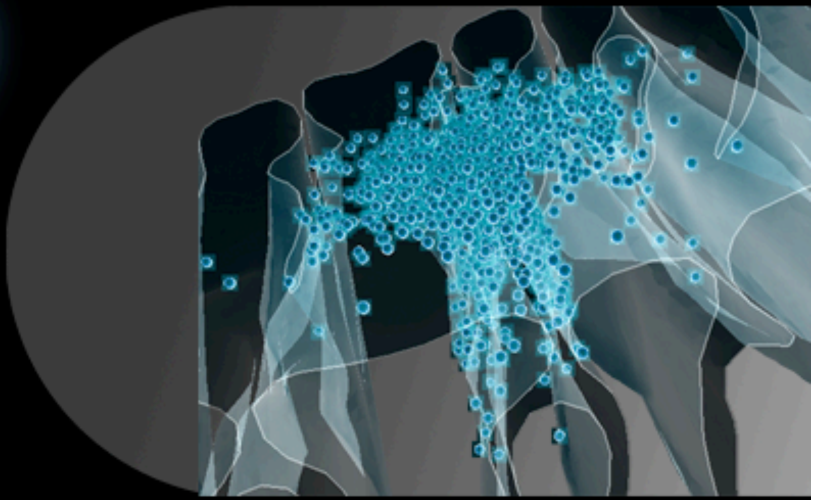


CALIFORNIA INSTITUTE OF TECHNOLOGY 1891 REALISTIC

sample:

NMJ MCELL SIMULATION

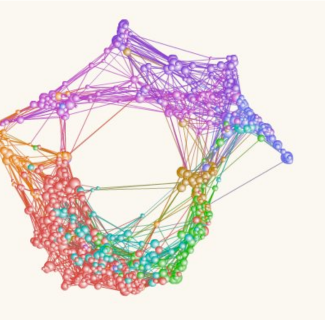

AChE.EA (surface molecules)
nmj.presynaptic1
nmj.basal_lamina1
nmj.mpjm1



process dictated by the data:

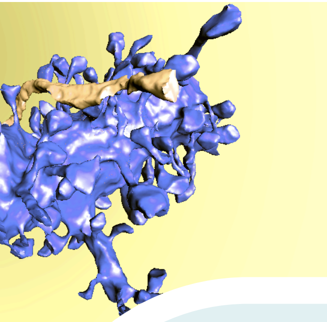
A CLOSER LOOK AT THE "DATA"

data: geometric structure



abstract multi-dimensional data records

- mapping + paradigms! …. -> *interaction*
- *infoviz*



2d/3d data + scalar/vector/tensor + time

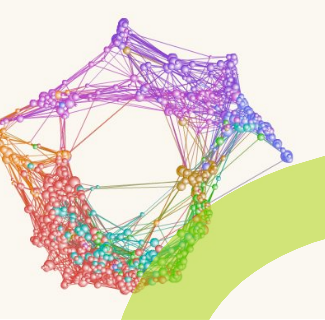
- paradigms …. -> *interaction*
- the more main stream viz

ieee vis
ieee infovis
siggraph

ieee xplore:
ieeexplore.ieee.org

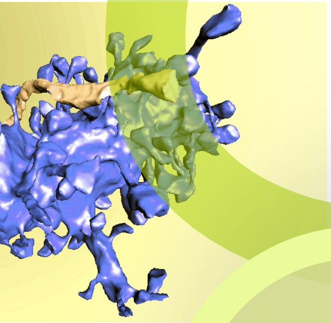
acm digital library:
portal.acm.org/dl.cfm

data: geometric structure

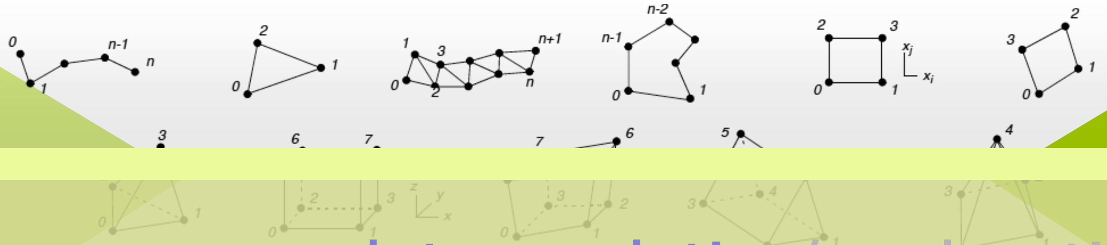


abstract multi-dimensional data records

```
MPG Cylinders Horsepower Weight Acceleration Year Origin
8. 50. 4 2.8 8.2 4 40. 250. 4 1500. 5500. 4 5. 30. 4 69.5 82.5 4 .8 3.2  3
18.000000 8.000000 130.000000 3504.000000 12.000000 70.000000 1.000000
15.000000 8.000000 165.000000 3693.000000 11.500000 70.000000 1.000000
18.000000 8.000000 150.000000 3436.000000 11.000000 70.000000 1.000000
16.000000 8.000000 150.000000 3433.000000 12.000000 70.000000 1.000000
17.000000 8.000000 140.000000 3449.000000 10.500000 70.000000 1.000000
... ... ... ... ... ... ... ... ...
```



2d/3d data + scalar/vector/tensor + time



FOCUS?
PURPOSE?

data analytics/exploration

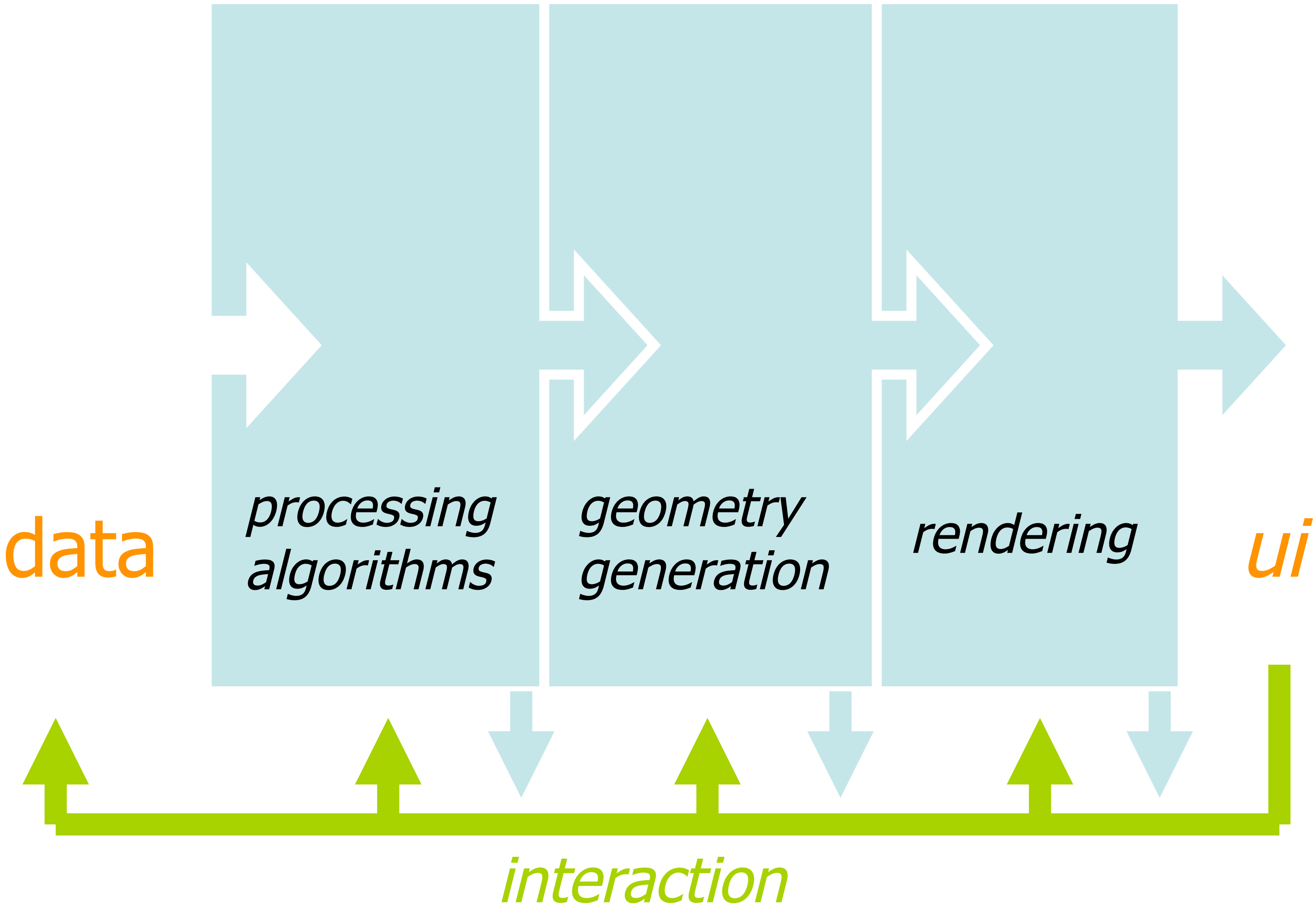
vs

physical analysis/exploration

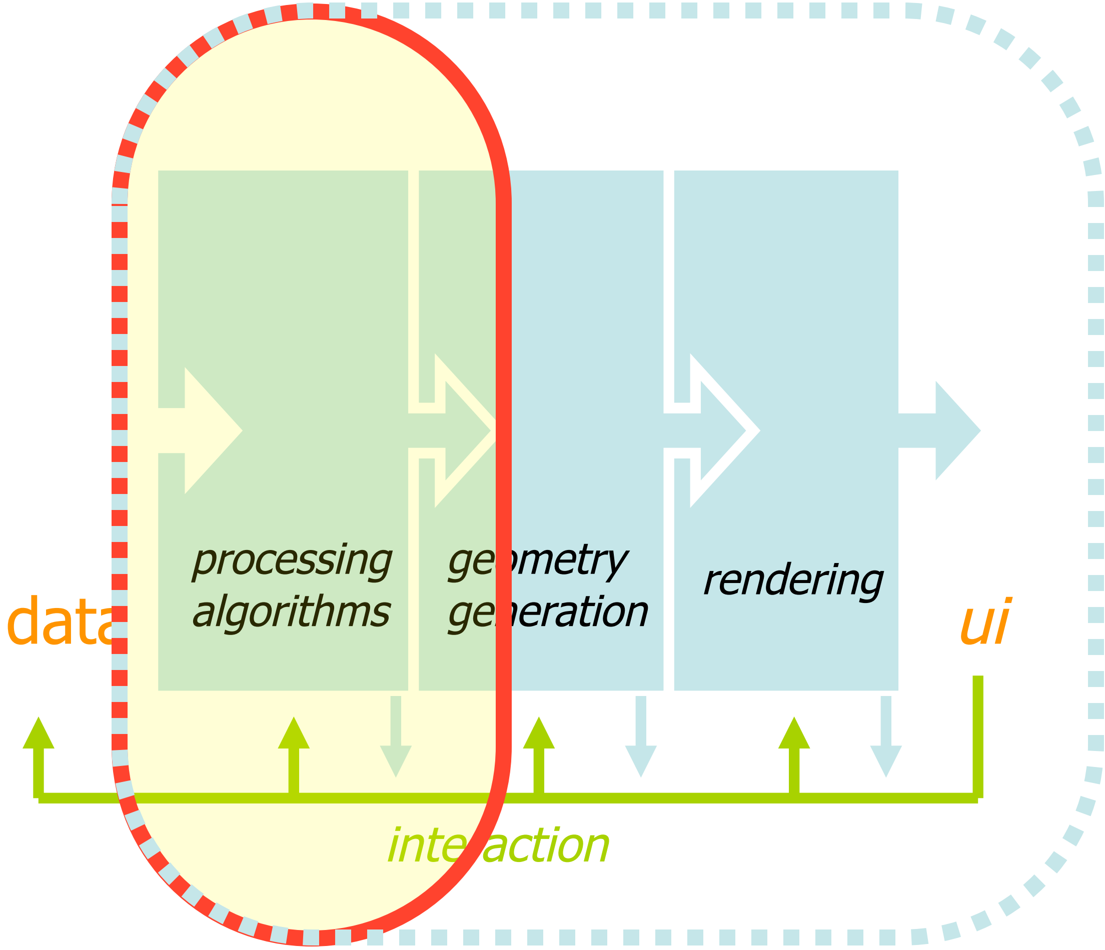
understanding:

THE VISUALIZATION PROCESS

usual visualization “engine”

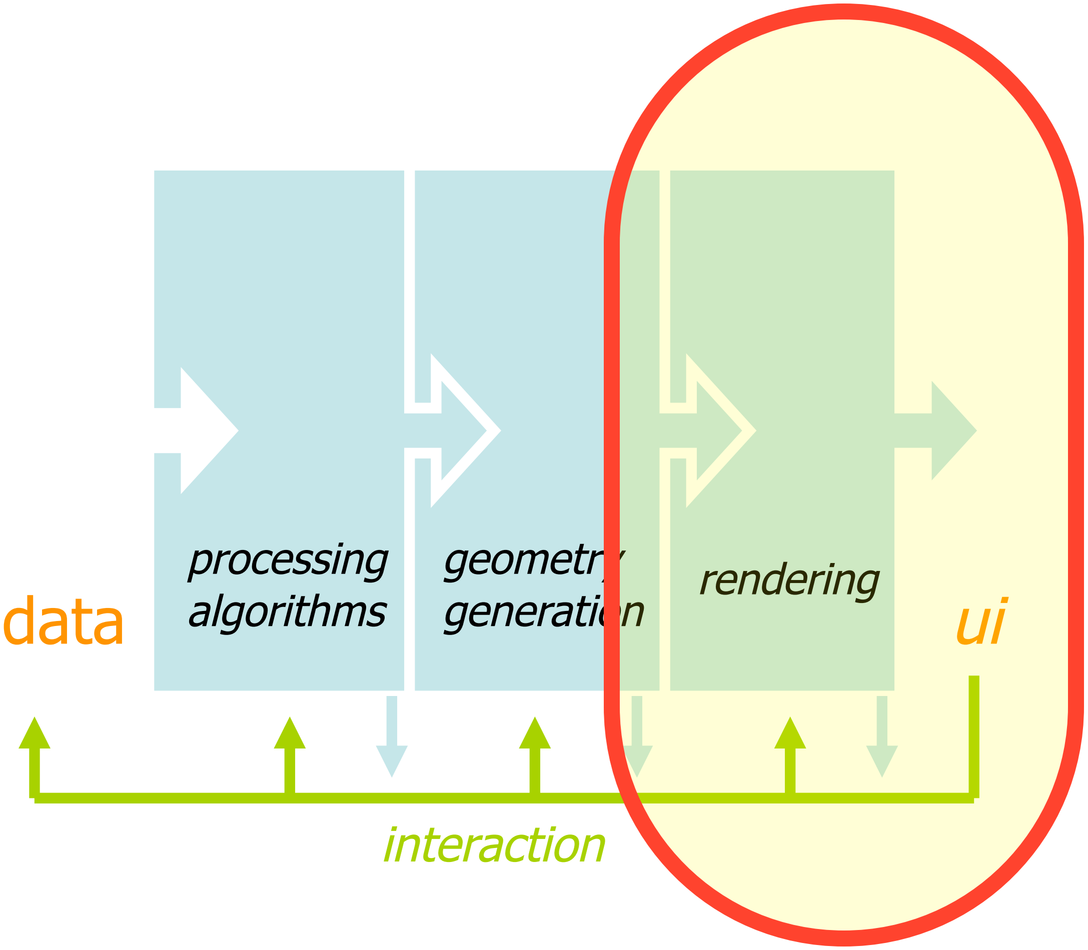


“the” visualization toolkit



- VTK ▢
  - c/c++
  - tcl/tk ⬚
  - python
  - java
- R

interactive renderers

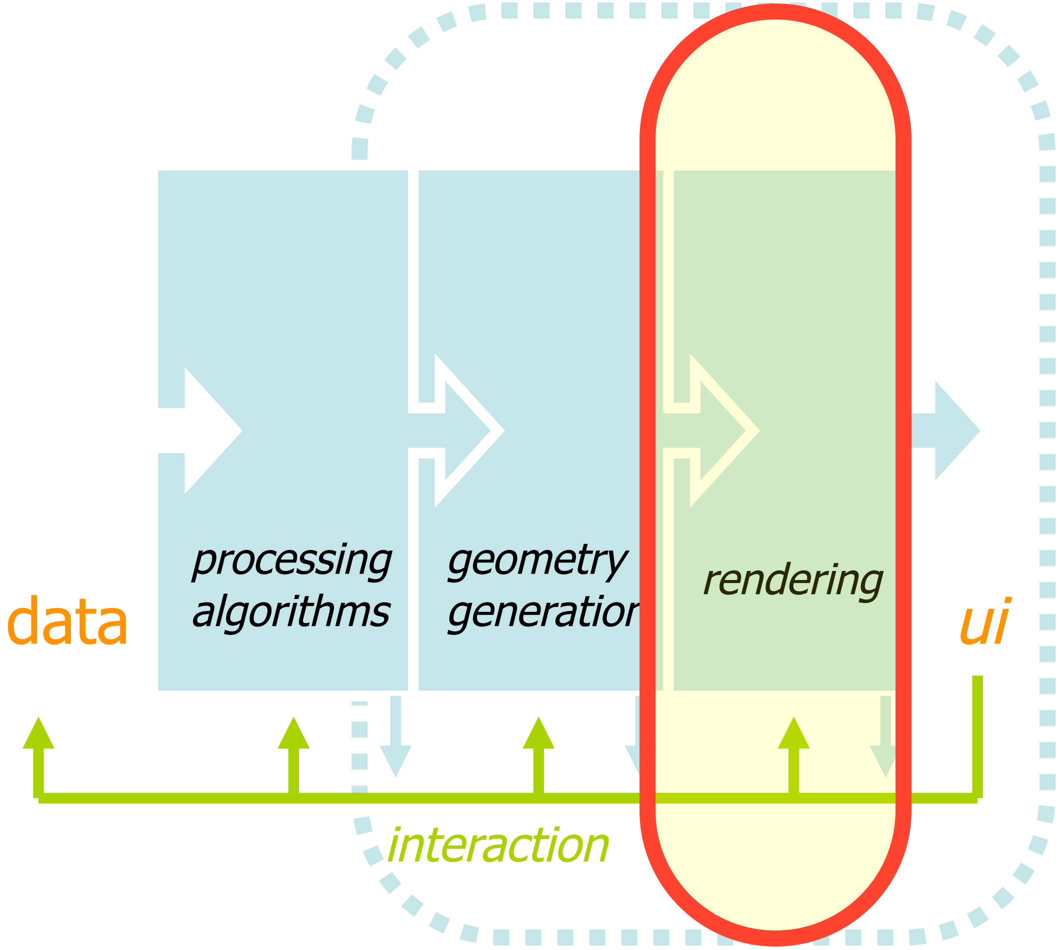


- OpenGL
  mesaGL
  directX

HIGHER LEVEL:

- OpenInventor
  - C++/GL
  - COIN
- Java3D
  - Java/GL

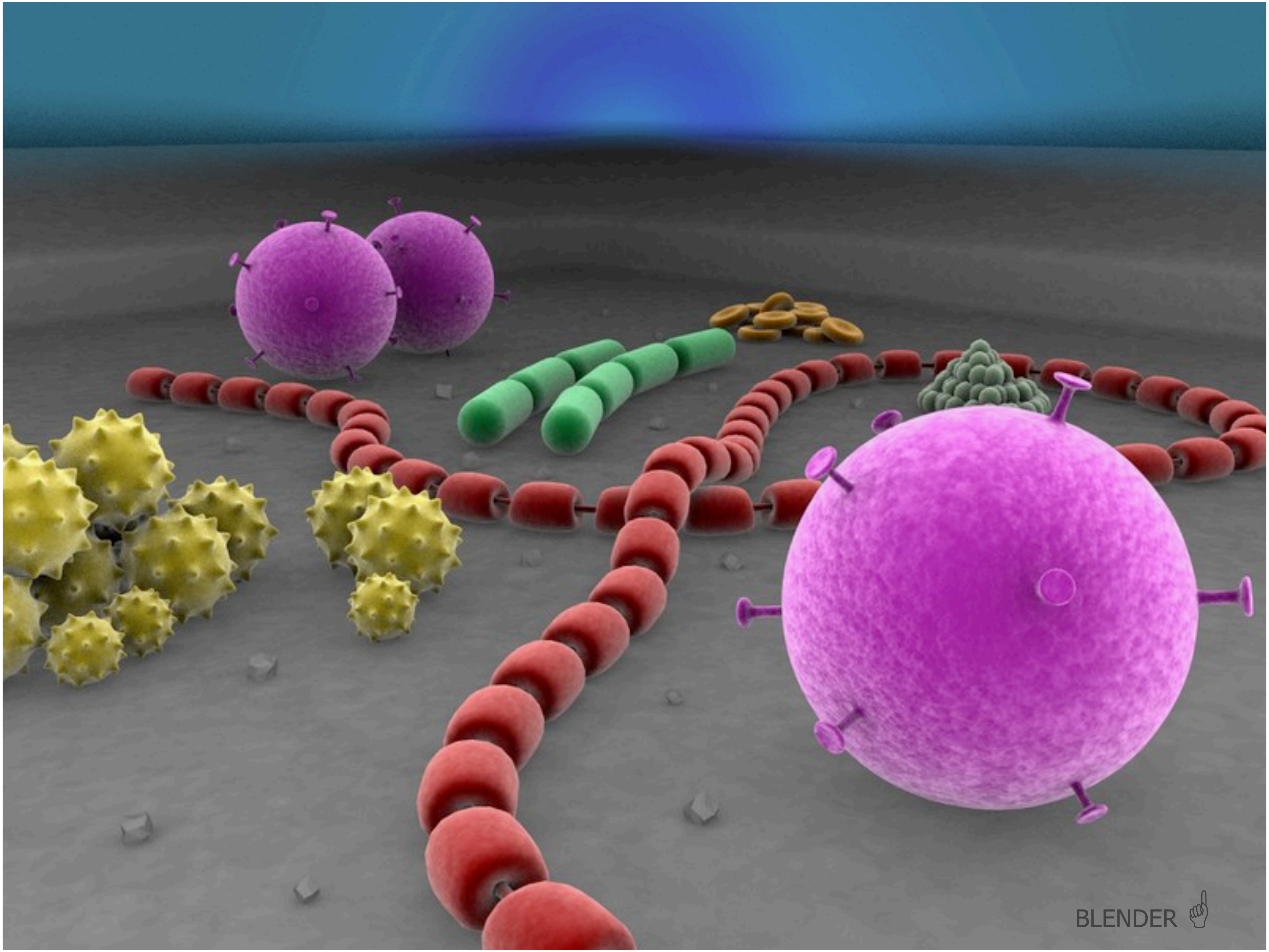
ray tracers

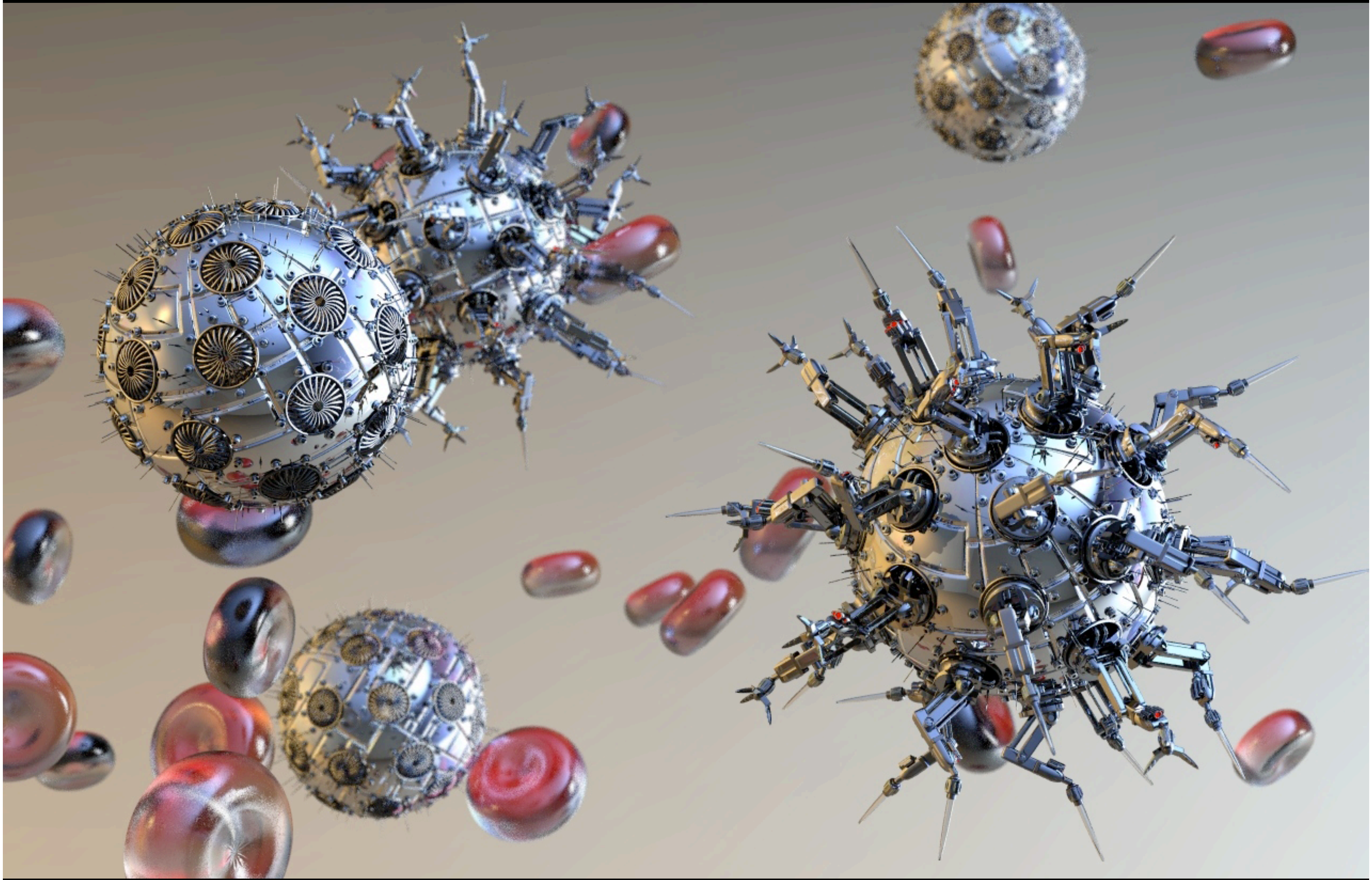


- POVRay
  - GELATO (GPU)
- RenderMan$^{\$\$}$

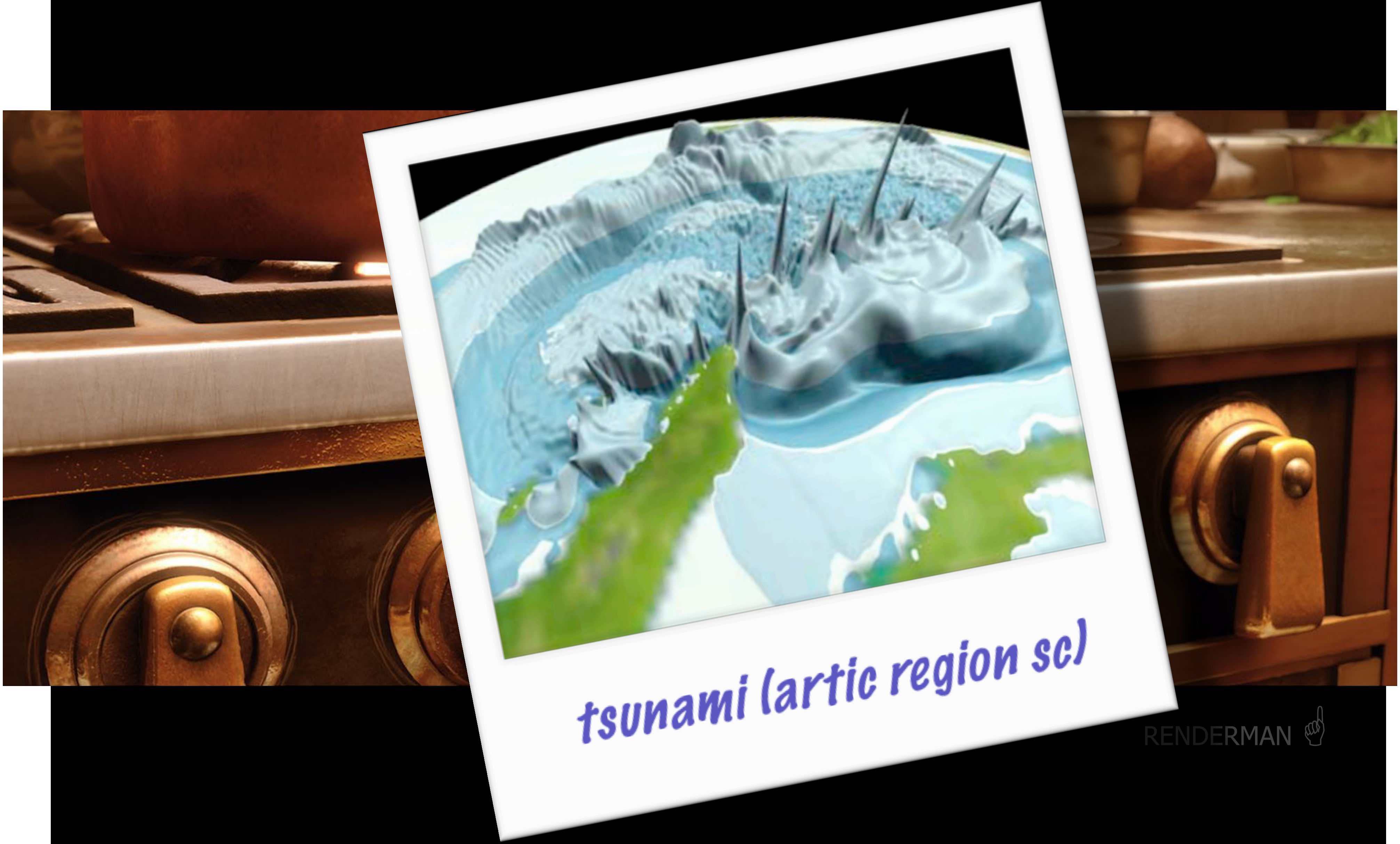
MODELLERS:

- Blender
- Maya$^{\$\$}$


BLENDER

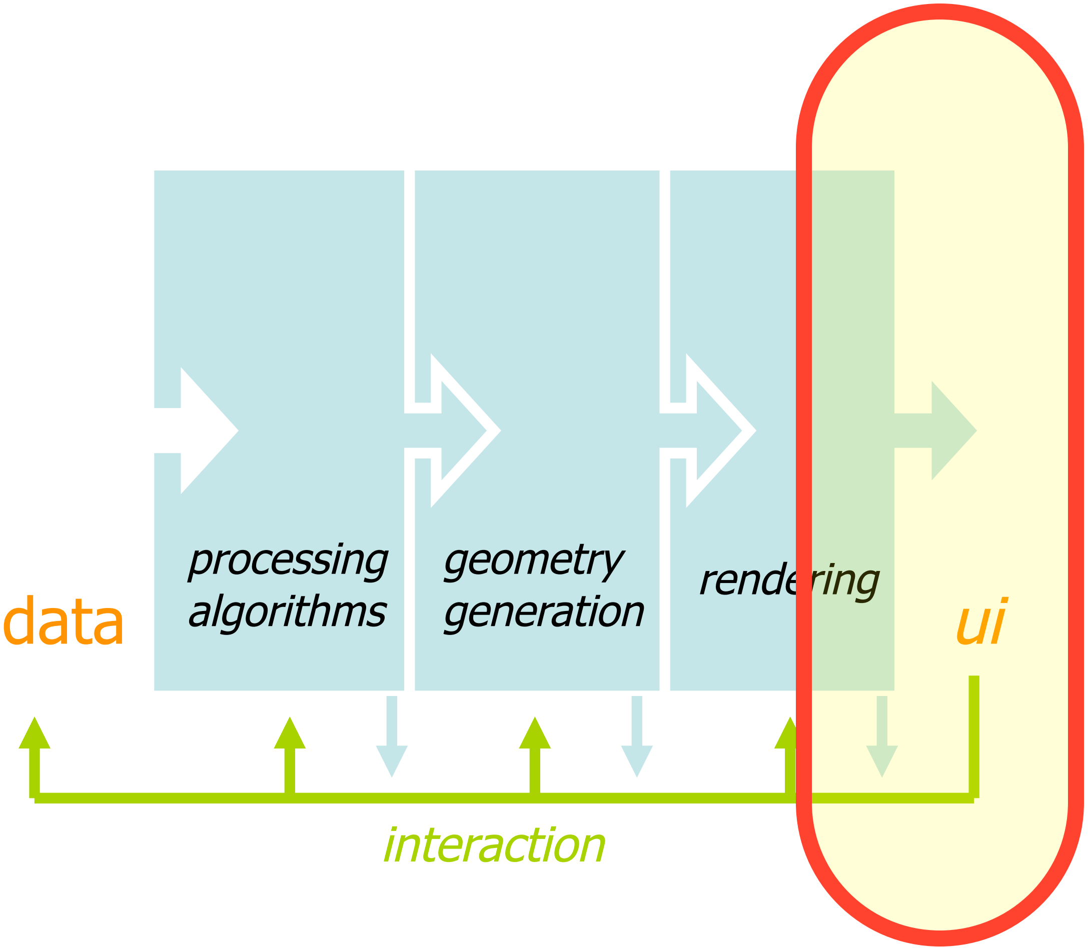


POVRAY



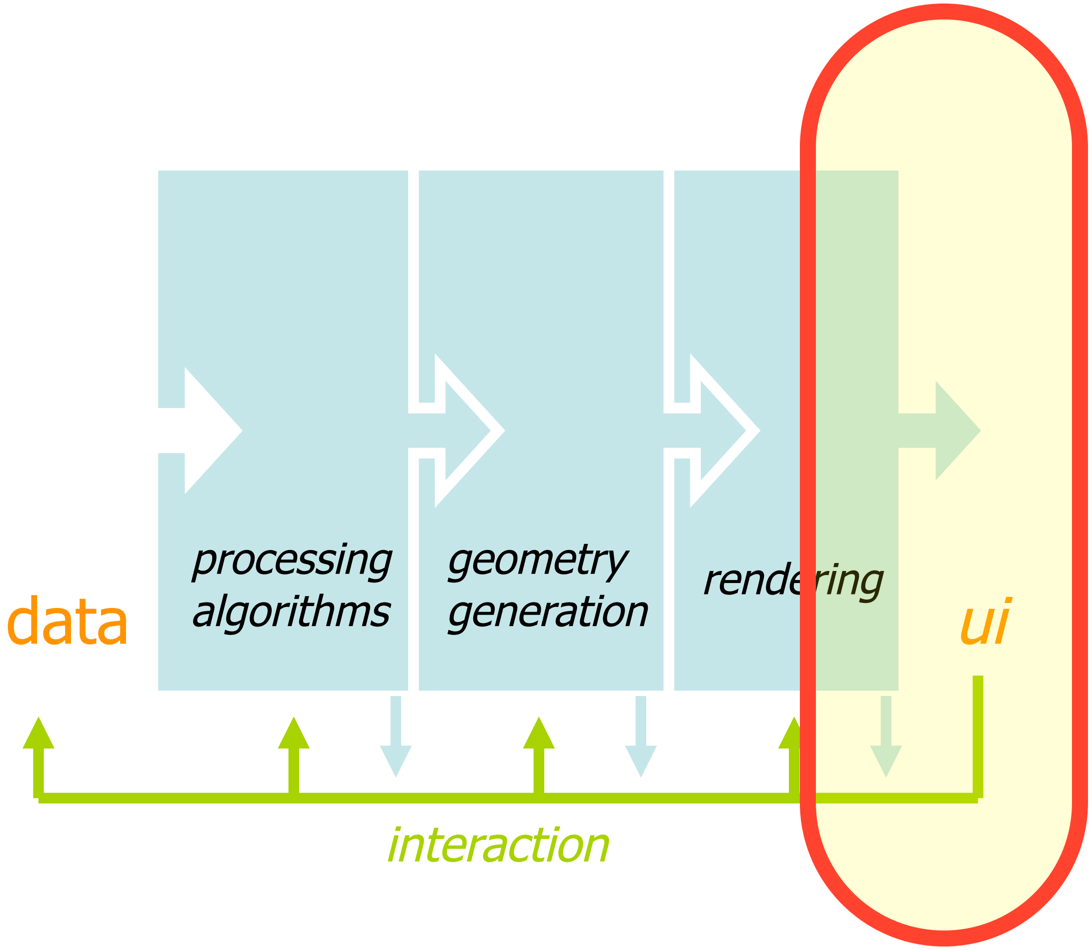
tsunami (artic region sc)

gui toolkits



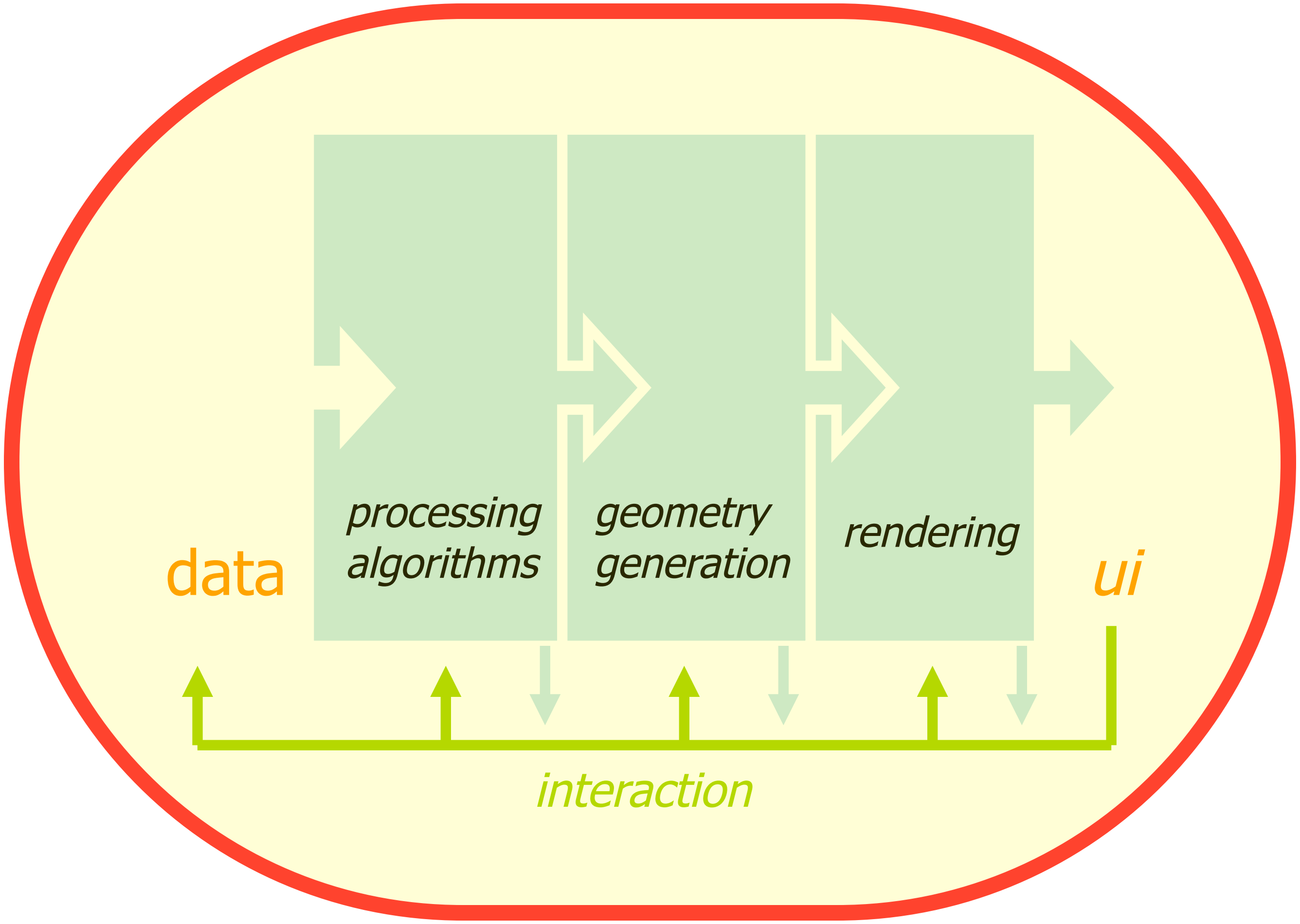
- QT
  - python
- GTK+
  - python
- *(blade)*
- TK (TCL/TK)
- Java Swing
- Motiff

web based ui



- dHTML/ JavaScript
- Processing
  - *Java*
- Flash

visualization system

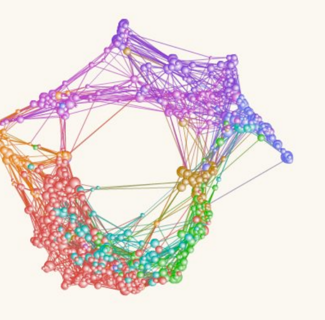


- Paraview^{VTK}
- LLNL VisIt^{VTK}
- EnSight$^{\$}$
- OpenDX
  - *IBM's DataExplorer*
- Mollegro, …

an overview: *tools & techniques*

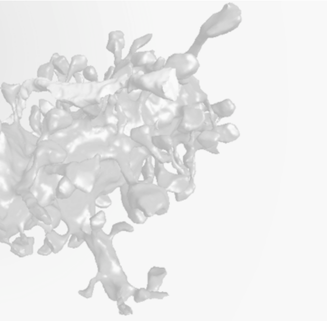
INFOVIZ

data: geometric structure



abstract multi-dimensional data records

- mapping + paradigms! .... -> *interaction*
- *infoviz*



2d/3d data + scalar/vector/tensor + time

- paradigms .... -> *interaction*
- the more main stream viz

basic infovis techniques:

N-DIMENSIONAL RECORD DATA

mapping data…

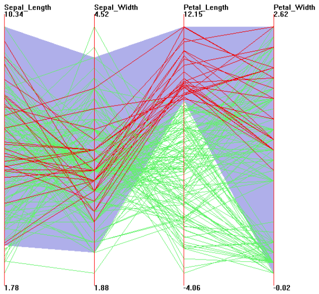
- visual analytics goal:

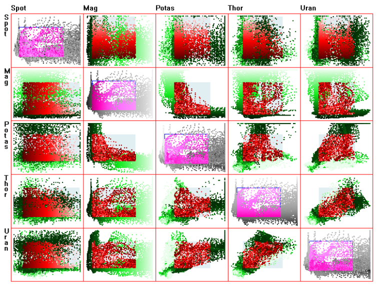
  detect, classify, and measure

  X

  trends, outliers, patterns, clusters, and correlations

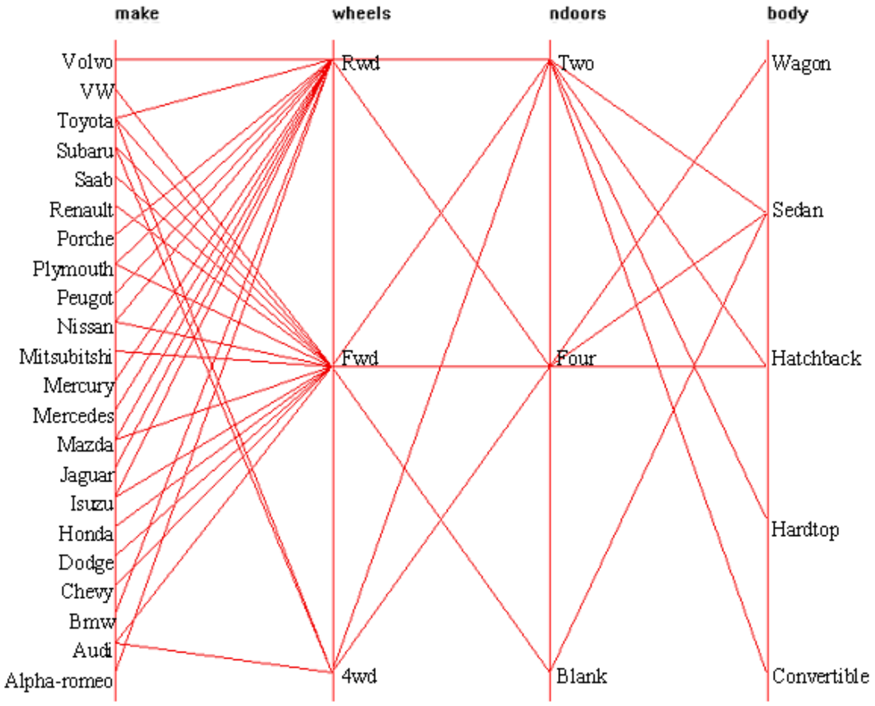
- *choose a layout strategy..*

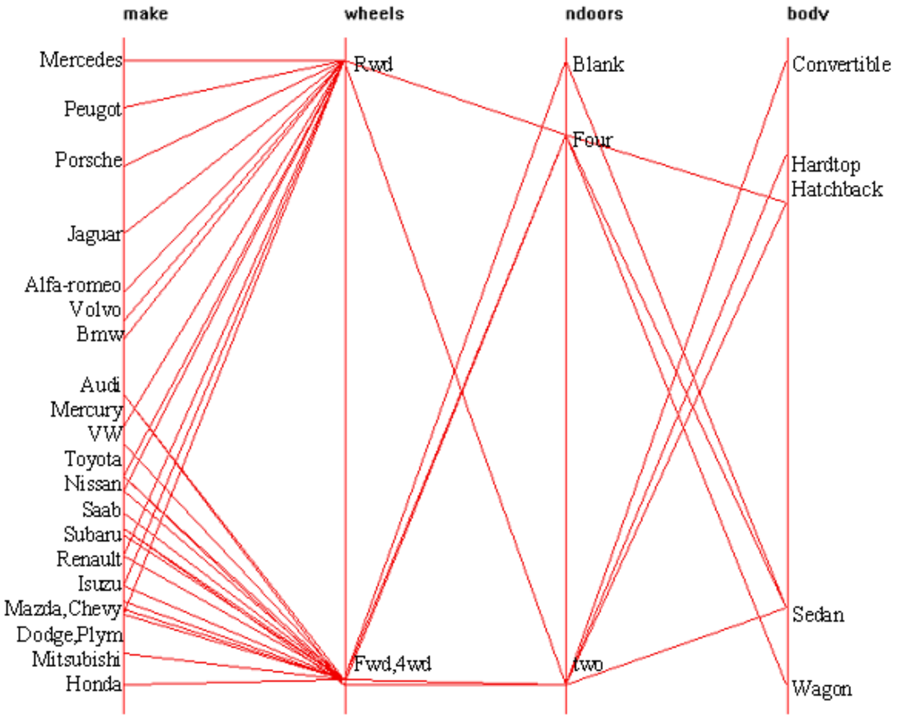


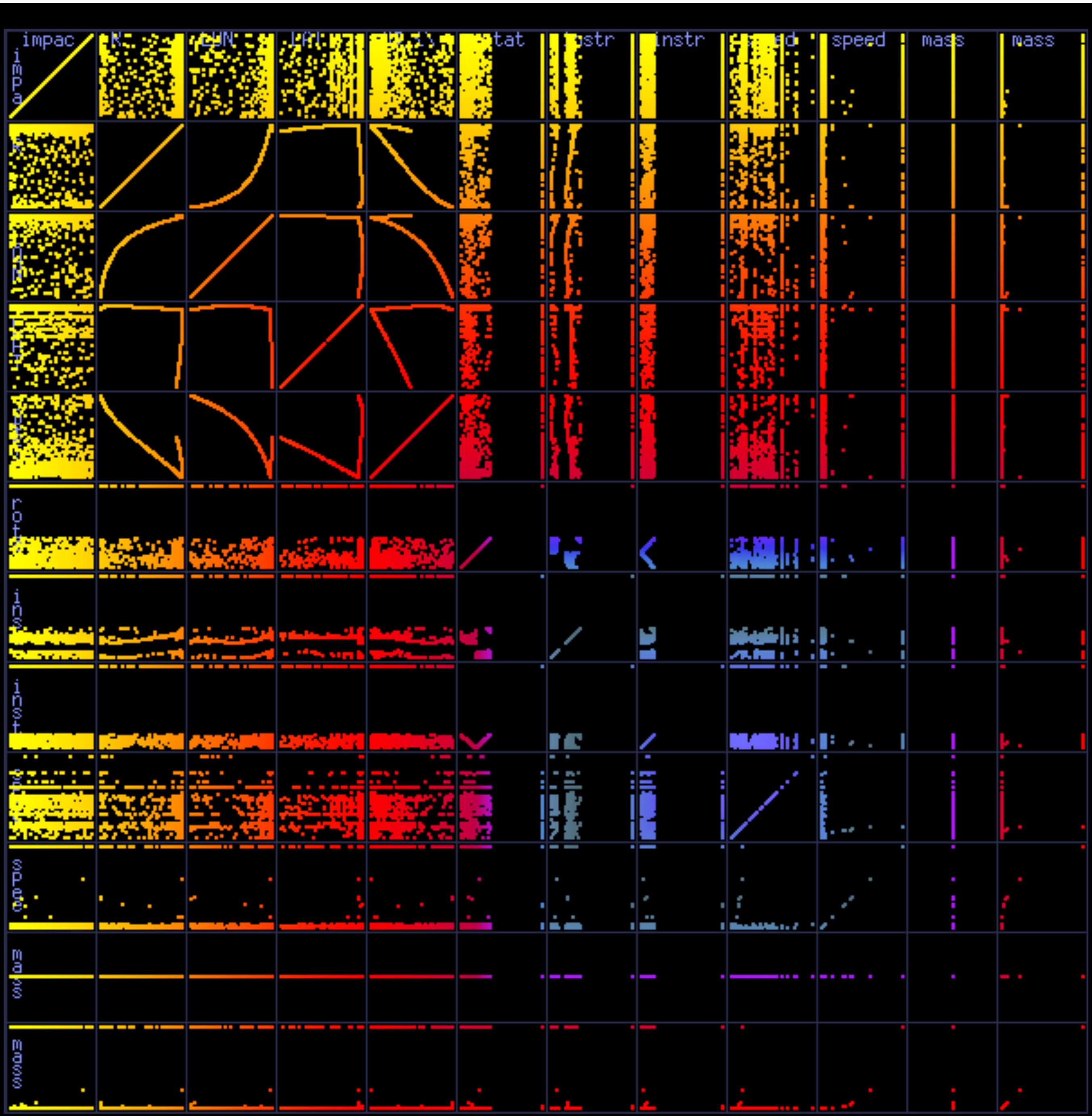


strategies

- parallel coordinates + clustering

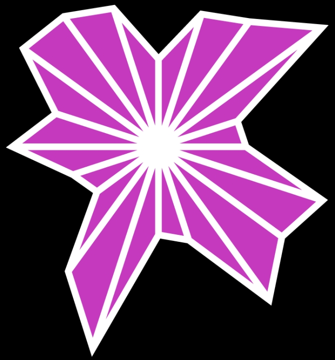




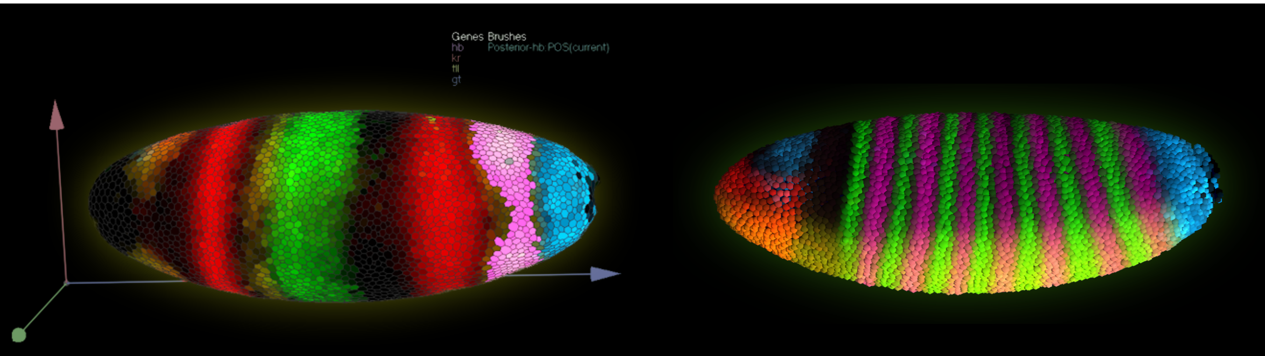


SCATTERPLOT
MATRIX

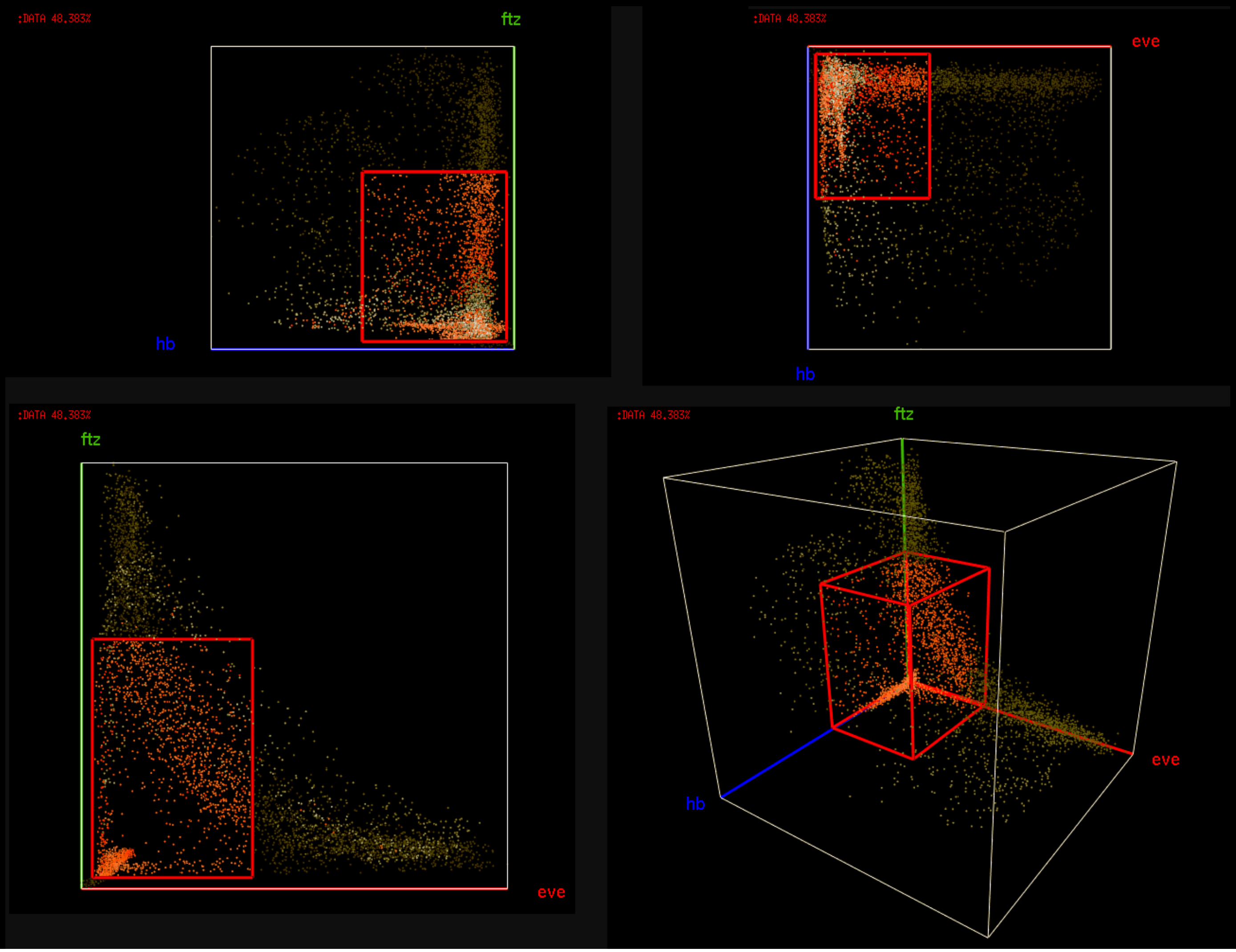
STAR PLOT GLYPHS

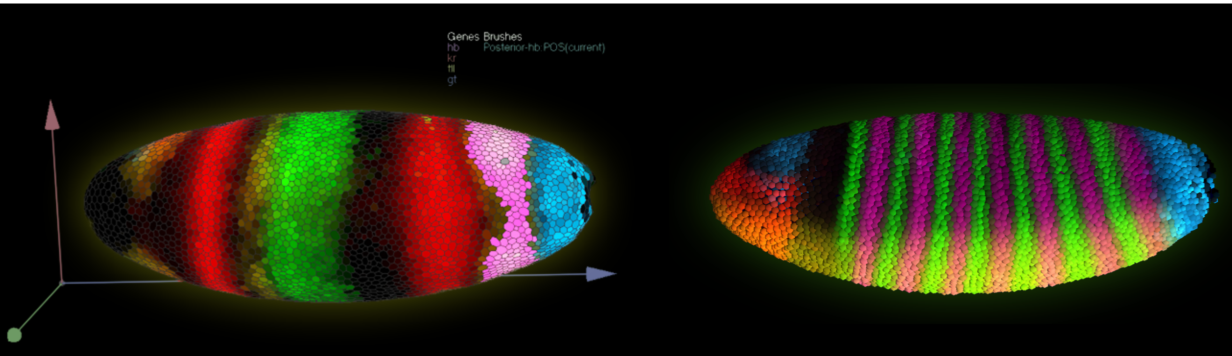


DROSOPHILIA



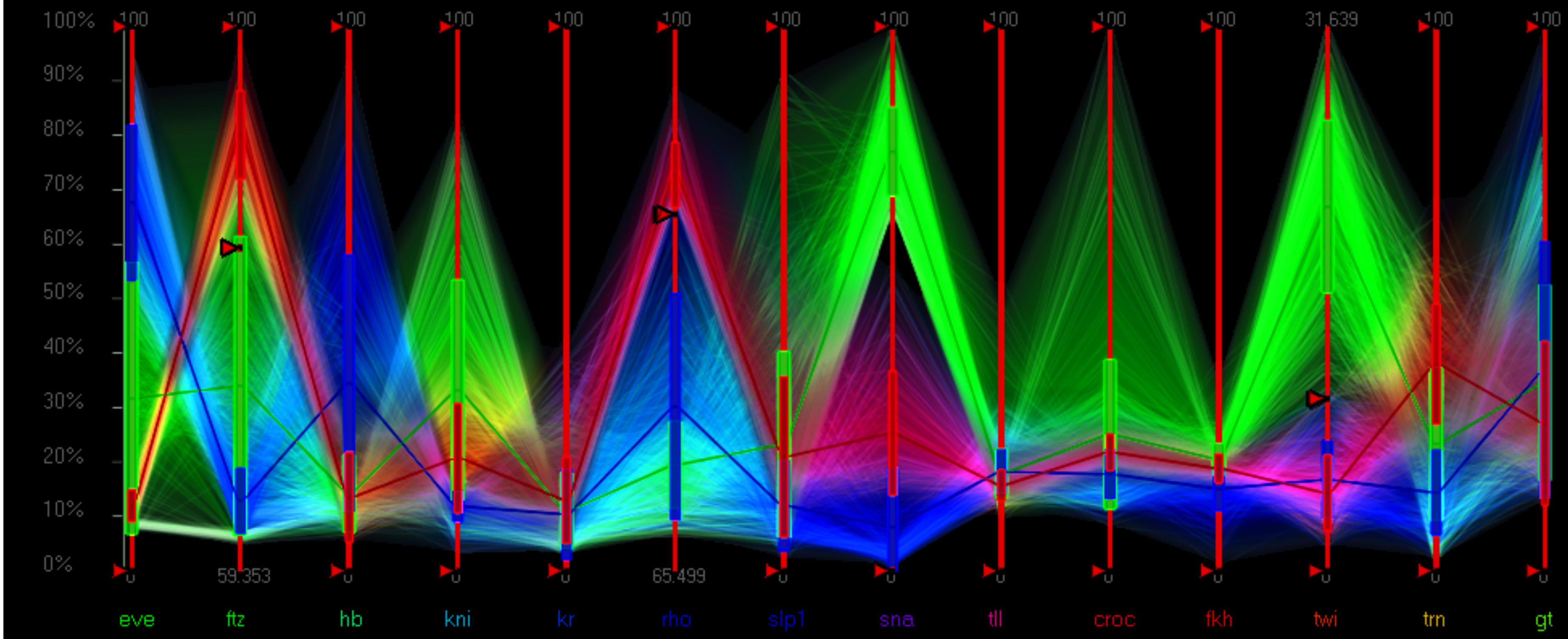
2D & 3D SCATTERPLOTS

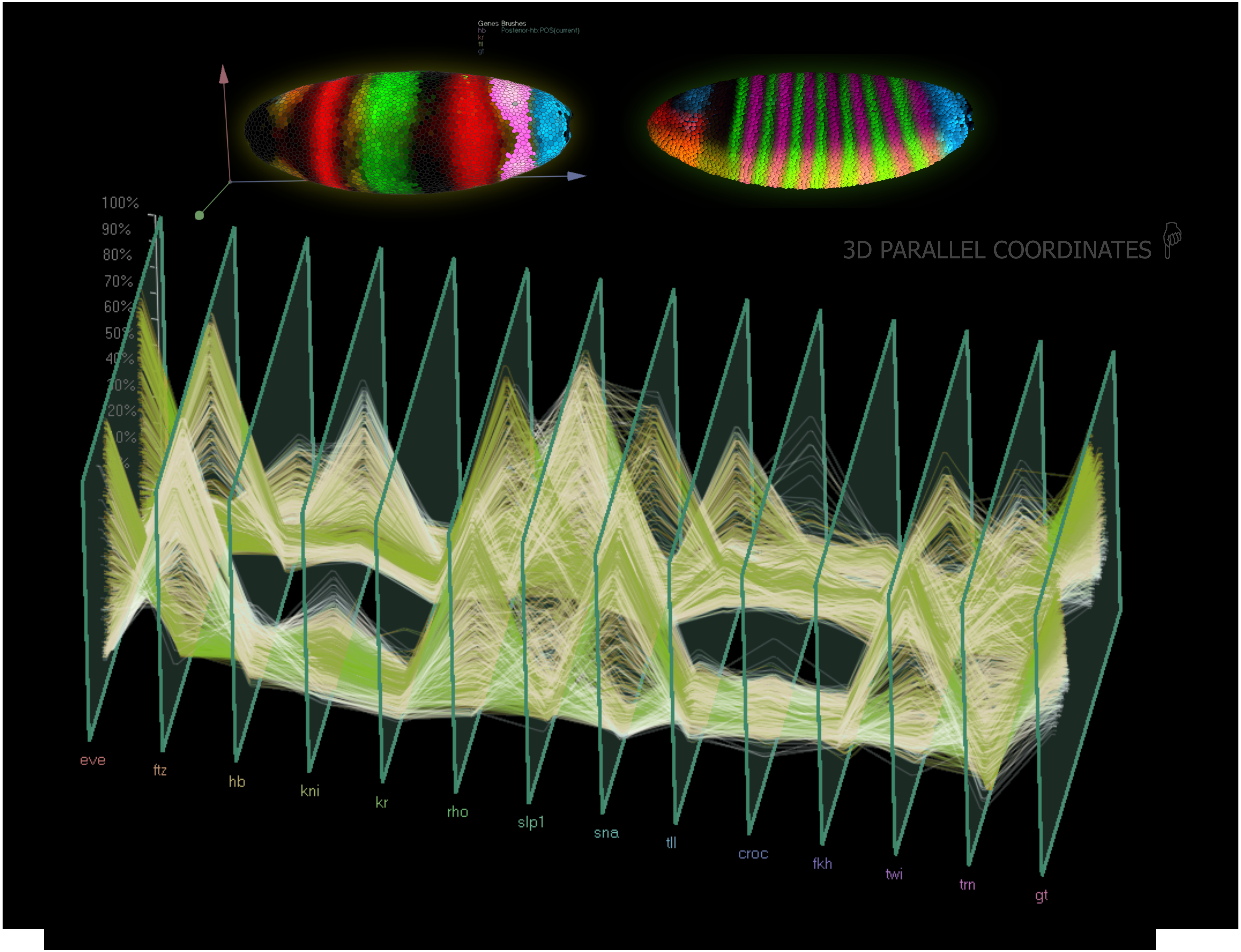




DROSOPHILIA

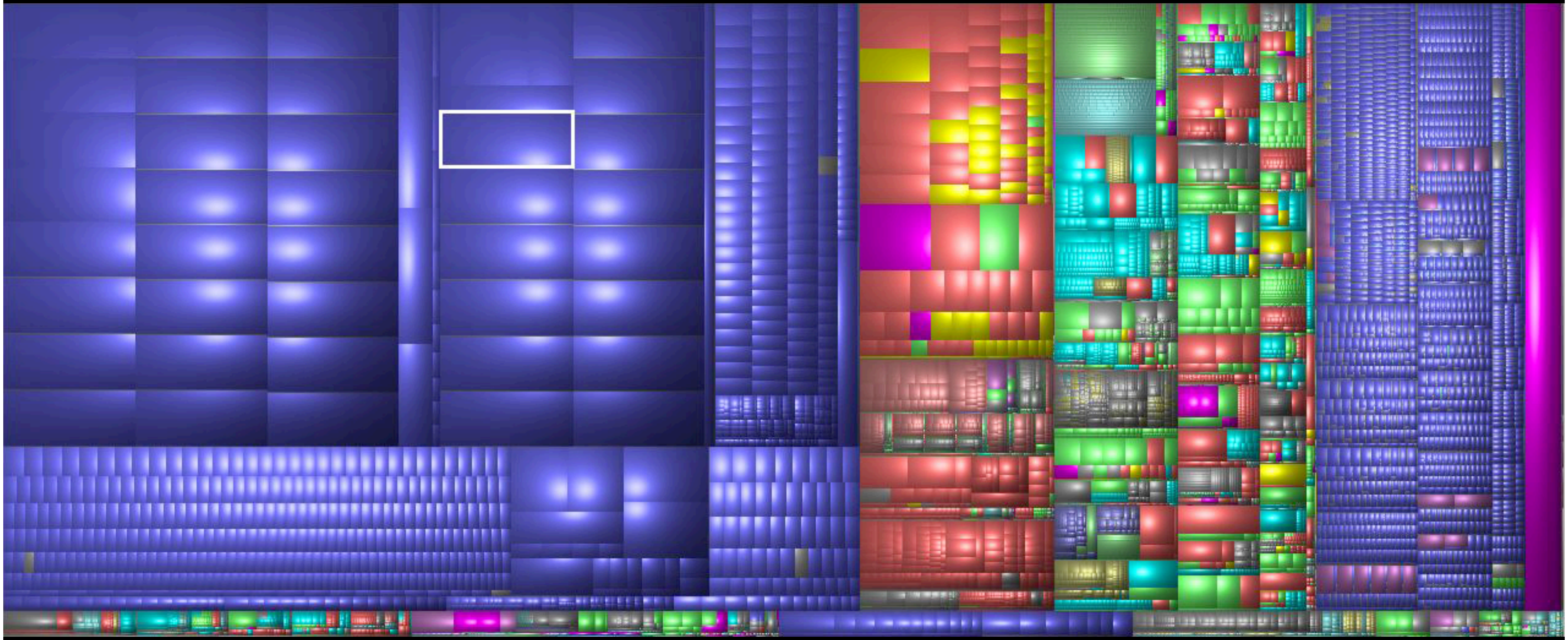
PARALLEL COORDINATES




Genes Brushes
hb Posterior-hb:POS(current)
kr
tll
gt
3D PARALLEL COORDINATES
100%
90%
80%
70%
60%
50%
40%
30%
20%
10%
0%
eve
ftz
hb
kni
kr
rho
slp1
sna
tll
croc
fkh
twi
trn
gt

basic infovis techniques:

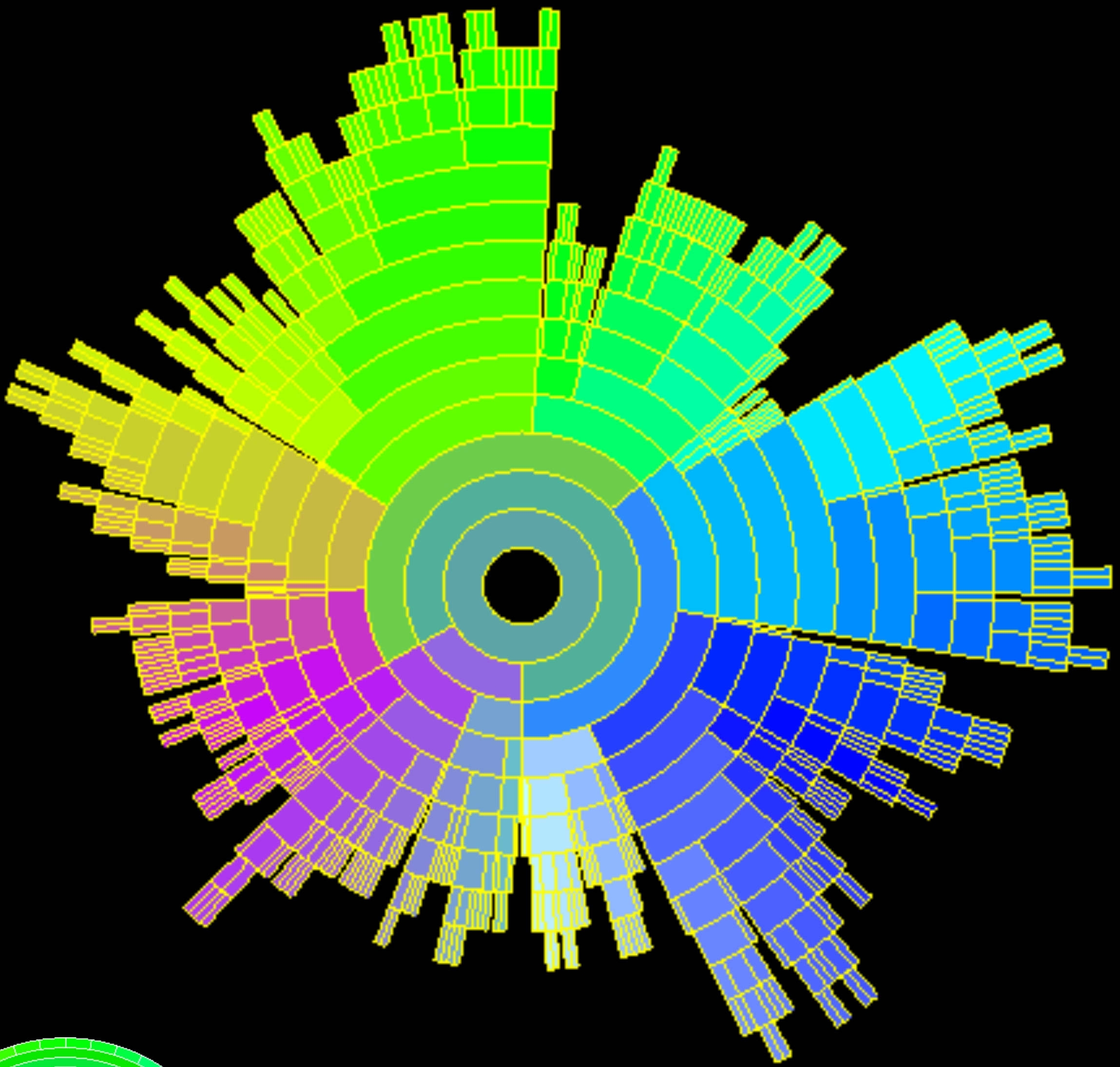
HEIRARCHICAL DATA

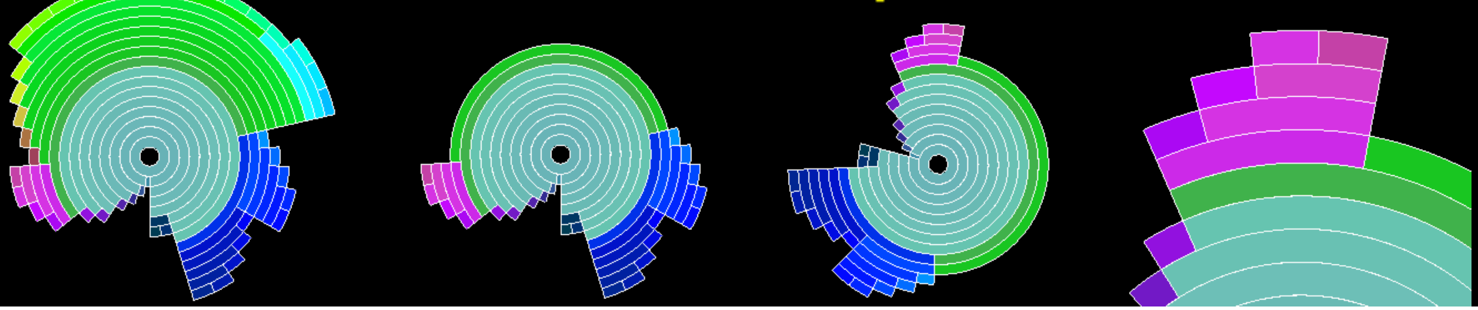


TREEMAPS

HEIRARCHICAL RINGS

☜





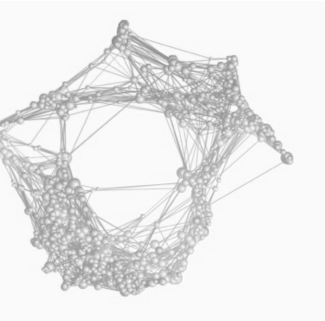
infovis packages

- mondrian (R based)
  - rosuda.org/Mondrian
- xmdv
  - davis.wpi.edu/~xmdv
- molegro data modeller *(bio)*
- topcat *(astro)*
- polaris (+datacubes)
  - now tableau.com
- www.infovis-wiki.net
  - *http://www.infovis-wiki.net/index.php/Software_Links_(InfoVis_Applications)*

demo:

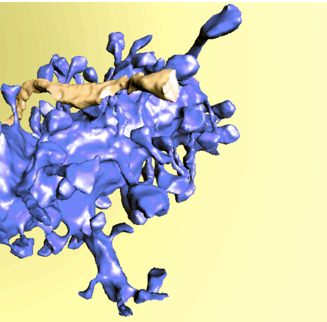
MOLEGRO & MONDRIAN{R}

data: geometric structure



abstract multi-dimensional data records

- mapping + paradigms! …. -> *interaction*
- *infoviz*



2d/3d data + scalar/vector/tensor + time

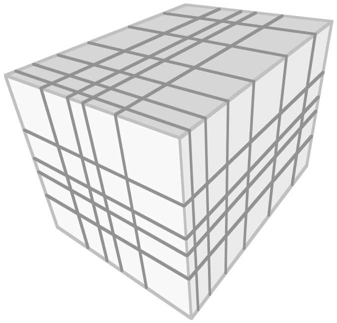
- paradigms …. -> *interaction*
- the more main stream viz

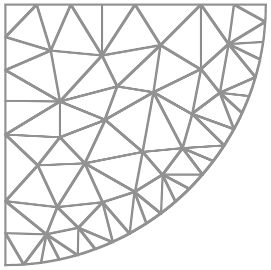
viz pipeline: *start*

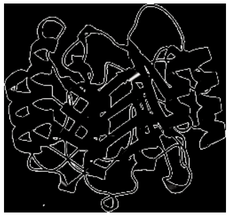
DATA FORMATS

basic data types

- structured grids…
  - *(raw binary)*
  - AMR (adptive mesh refinement)
- unstructured grids…
  - points
  - triangle meshes
  - tet meshes
- atomic coordinate files…
  - PDB (protein data bank)





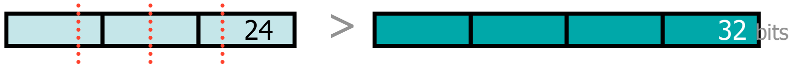


no standard formats

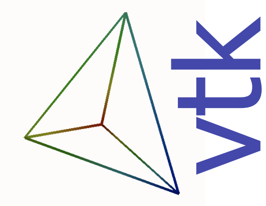
- raw
- vtk
    - ascii or binary
- *new* vtk
    - xml
    - ascii or base64 (mime) encoded binary
- amr
    - chombo(hdf5)/silo
- pdb

```
==   abcdefgh  ijklmnop
     qrstuvwx  yzABCDEF
     GHIJKLMN  OPQRSTUV
     WXYZ0123  456789+/
```

($64=2^6$) 6 bits -> 8 bit char (*size* * *4/3*)

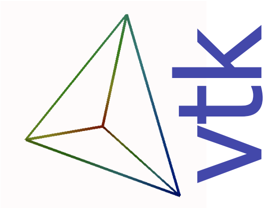


vtk sample: pyramid.vtk



```
# vtk DataFile Version 2.0
My Pyramid Example
ASCII
DATASET POLYDATA
POINTS 4 float
0.0 0.0 0.0
1.0 0.0 0.0
0.5 0.0 0.7
0.5 0.6 0.7
POLYGONS 4 16
3 0 2 1
3 0 1 3
3 0 3 2
3 1 2 3
POINT_DATA 4
SCALARS vertexData float 1
LOOKUP_TABLE default
0.1
0.2
0.3
0.4
CELL_DATA 4
SCALARS faceData int 1
LOOKUP_TABLE default
0
1
2
3
```

vtk sample: pyramid.vtp



```
<?xml version="1.0"?>
<VTKFile type="PolyData" version="0.1" byte_order="LittleEndian">
 <PolyData>
  <Piece NumberOfPoints="4" NumberOfPolys="4">
    <Points>
     <DataArray type="Float32" Name="coords" NumberOfComponents="3" form
      0.0 0.0 0.0
      1.0 0.0 0.0
      0.5 0.0 0.7
      0.5 0.6 0.7
     </DataArray>
    </Points>
    <Polys>
     <DataArray type="Int32" Name="connectivity" format="ascii">
      0 2 1
      0 1 3
      0 3 2
      1 2 3
     </DataArray>
     <DataArray type="Int32" Name="offsets" format="ascii">
      3 6 9 12
     </DataArray>
    </Polys>
    <PointData Scalars="vertexData">
     <DataArray type="Float32" Name="vertexData" format="ascii">
      0.1 0.2 0.3 0.4
     </DataArray>
    </PointData>
    <CellData Scalars="faceData">
     <DataArray type="Int32" Name="faceData" format="ascii">
      0 1 2 3
     </DataArray>
    </CellData>
</Piece>
 </PolyData>
</VTKFile>
```

vtk sample: volume.vti

```
<?xml version="1.0"?>
<VTKFile type="ImageData" version="0.1"
      byte_order="LittleEndian">
<ImageData WholeExtent="0 3 0 3 0 3" Origin="0 0 0"
      Spacing="1 1 1">
<Piece Extent="0 3 0 3 0 3">
<PointData Scalars="vertexData">
<DataArray type="Float32" Name="scalarData"
      format="ascii">
0 1 2 3 1 2 3 4 2 3 4 8 3 6 9 11
2 3 4 5 5 6 7 8 3 4 5 6 4 5 6 7
3 4 5 6 3 4 5 6 4 5 6 7 6 7 8 9
2 3 4 5 2 3 4 5 3 4 5 6 4 5 6 7
</DataArray>
</PointData>
<CellData Scalars="cellData" Normals="cell_ normals">
<DataArray type="Int32" Name="cellData" format="ascii">
1 3 9 2 8 16 3 9 27
2 3 4 6 7 8  6 9 10
0 1 2 0 2 4  1 2 3
</DataArray>
</CellData>
</Piece>
</ImageData>
</VTKFile>
```

viz pipeline: *the tools*

VTK

the visualization toolkit

- g.e. medical viz algorithms
  - -> kitware
- collection of *filters*
  - marching cubes (patented)
- educational
  - -> vtk book
- evolved/extended
  - object oriented
  - c++
  - GL + Tk (UI)
    - now python (+QT?), java

vtk.org

kitware.com

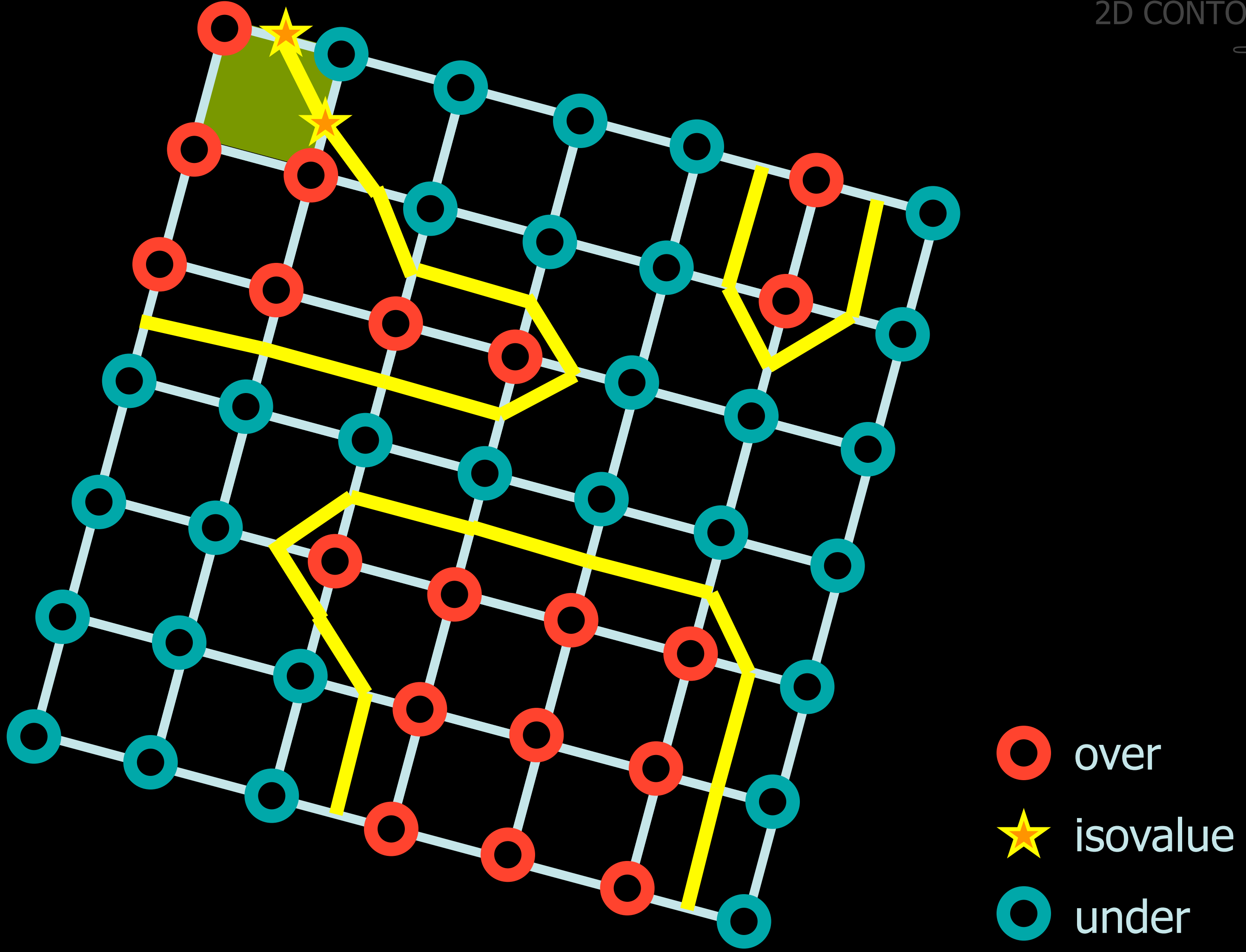
vtkpython

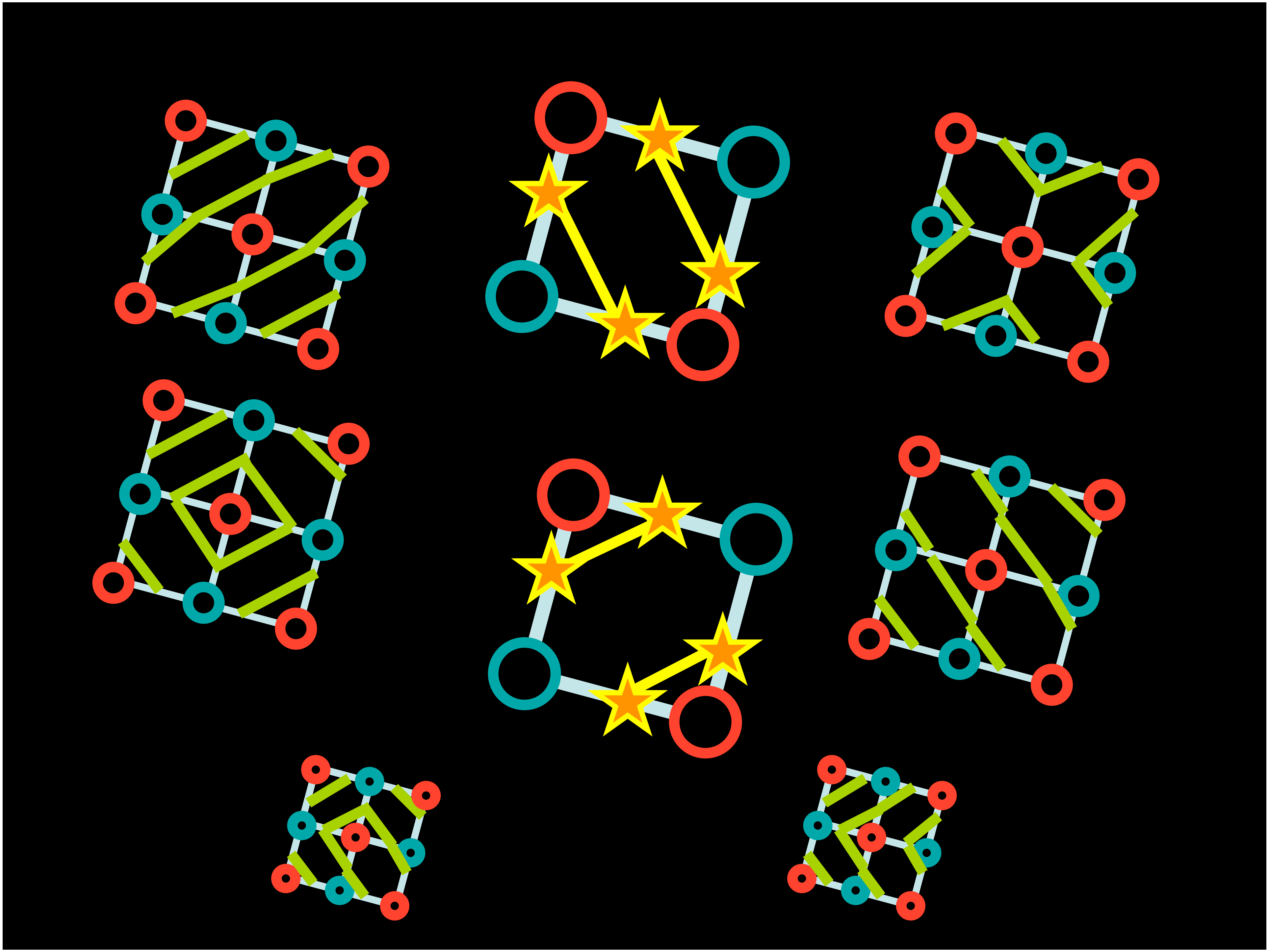
PYTHON MODULE

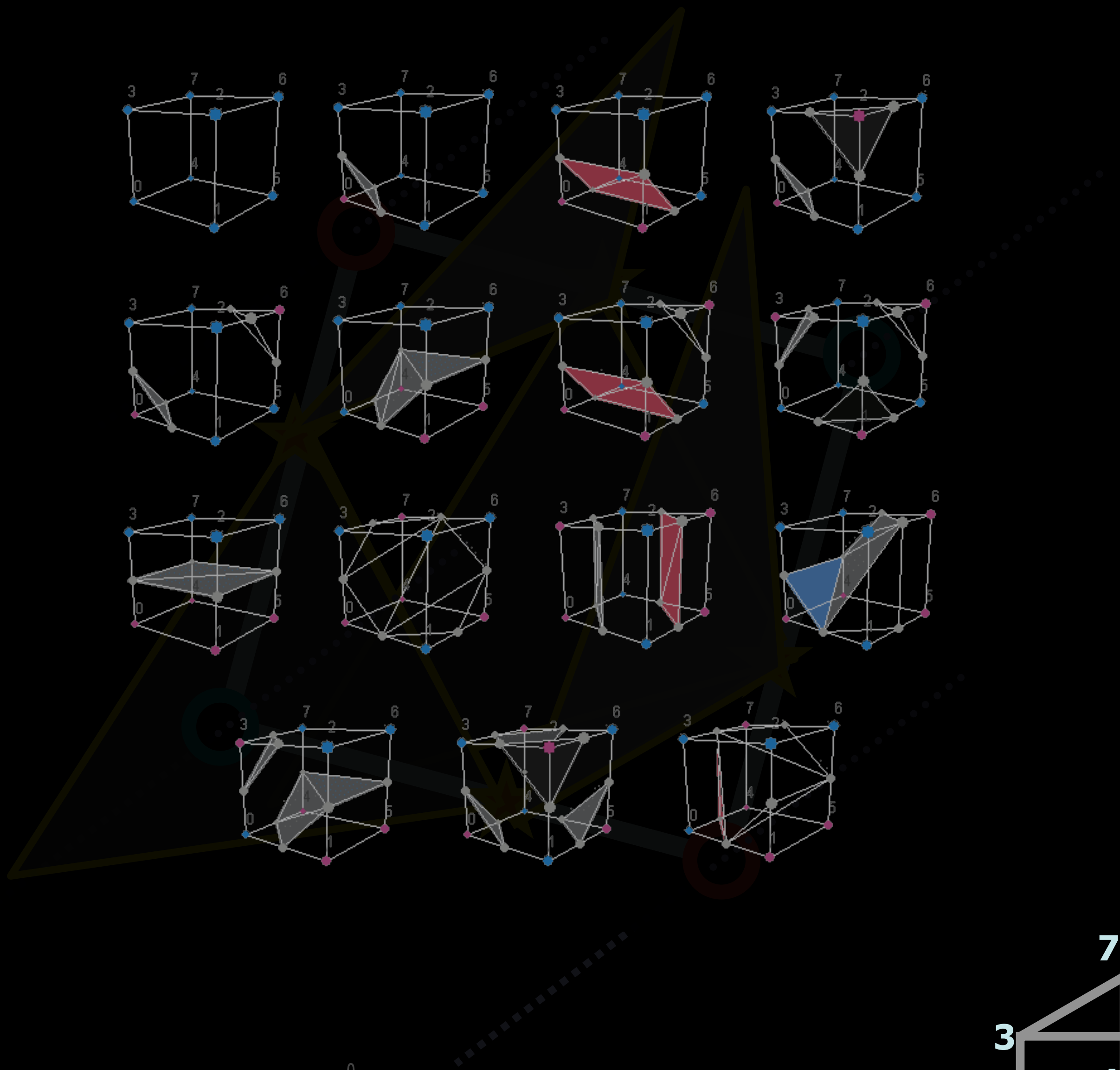
pyvtk

FILE MANIPULATION

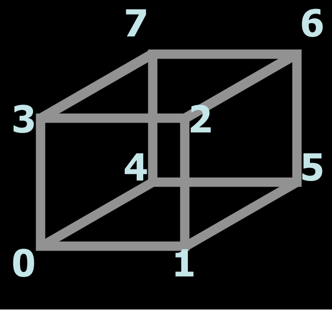
2D CONTOUR







INCONSISTENCY ERROR ☝ FIX: TABLE FORCING CONSISTENCY

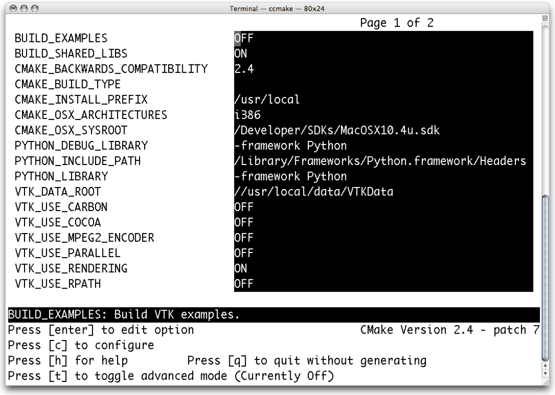


vtk

- visualization algorithms
  - scalar
  - vector
  - tensor
  - texture
  - volumetric
- imaging algorithms
  - directly integrated
  - mix 2D imaging/3D graphics
- modeling techniques
  - implicit modeling
  - polygon reduction
  - mesh smoothing
  - cutting
  - contouring
  - Delaunay triangulation

getting vtk

- windows -> binary
- unix
  - mac: macport/fink
  - debian/ubuntu: apt-get
  - redhat: rpm
- compile:
  - cmake! (binary from kitware)
  - “ccmake .”
    - toggle to advanced args
    - add python
    - use X



```
                                                     Page 1 of 2
 BUILD_EXAMPLES                    OFF
 BUILD_SHARED_LIBS                 ON
 CMAKE_BACKWARDS_COMPATIBILITY     2.4
 CMAKE_BUILD_TYPE
 CMAKE_INSTALL_PREFIX              /usr/local
 CMAKE_OSX_ARCHITECTURES           i386
 CMAKE_OSX_SYSROOT                 /Developer/SDKs/MacOSX10.4u.sdk
 PYTHON_DEBUG_LIBRARY              -framework Python
 PYTHON_INCLUDE_PATH               /Library/Frameworks/Python.framework/Headers
 PYTHON_LIBRARY                    -framework Python
 VTK_DATA_ROOT                     //usr/local/data/VTKData
 VTK_USE_CARBON                    OFF
 VTK_USE_COCOA                     OFF
 VTK_USE_MPEG2_ENCODER             OFF
 VTK_USE_PARALLEL                  OFF
 VTK_USE_RENDERING                 ON
 VTK_USE_RPATH                     OFF

BUILD_EXAMPLES: Build VTK examples.
Press [enter] to edit option                         CMake Version 2.4 - patch 7
Press [c] to configure
Press [h] for help           Press [q] to quit without generating
Press [t] to toggle advanced mode (Currently Off)
```

vtkpython: cone.py

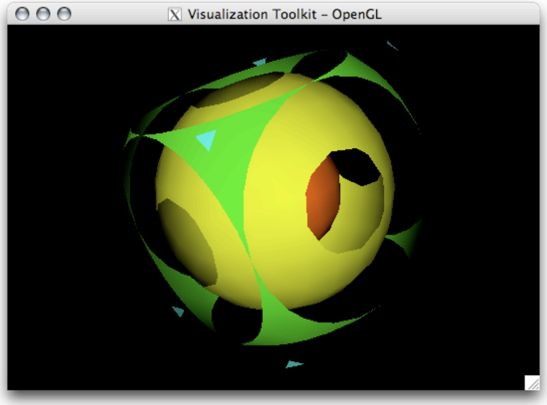
```
#! /usr/bin/python
# load VTK extensions
import vtk

# create a rendering window and renderer
renderer =   vtk.vtkRenderer()
myWindowRenderer = vtk.vtkRenderWindow()
myWindowRenderer.AddRenderer(renderer)
myWindowRenderer.SetSize(640,480)

myInteractiveWindow =  vtk.vtkRenderWindowInteractor()
myInteractiveWindow.SetRenderWindow(myWindowRenderer)

# create an actor and give it cone geometry
cone = vtk.vtkConeSource()
cone.SetResolution(8)
coneMapper = vtk.vtkPolyDataMapper()
coneMapper.SetInput(cone.GetOutput())
coneActor = vtk.vtkActor()
coneActor.SetMapper(coneMapper)

# assign our actor to the renderer
renderer.AddActor(coneActor)

# enable user interface interactor
myInteractiveWindow.Initialize()
myInteractiveWindow.Start()
```

demo:

PYVTK

vtkpython: iso.py

```
#! /usr/bin/python
# load VTK extensions
import vtk

# create a rendering window and renderer
renderer =   vtk.vtkRenderer()
myWindowRenderer = vtk.vtkRenderWindow()
myWindowRenderer.AddRenderer(renderer)
myWindowRenderer.SetSize(640,480)

myInteractiveWindow =   vtk.vtkRenderWindowInteractor()
myInteractiveWindow.SetRenderWindow(myWindowRenderer)

# read mydata from volume.vti file
mydata= vtk.vtkXMLImageDataReader()
mydata.SetFileName("volume.vti")

# create filter
mydataIso= vtk.vtkMarchingCubes()
mydataIso.SetInput(mydata.GetOutput())
mydataIso.SetValue(0,0.1)
mydataIso.SetValue(1,0.3)
mydataIso.SetValue(2,0.5)
mydataIso.SetValue(3,0.7)

# pipe resuts to polymapper, and add actor
mydataMapper= vtk.vtkPolyDataMapper()
mydataMapper.SetInput(mydataIso.GetOutput())
mydataActor = vtk.vtkActor()
mydataActor.SetMapper(mydataMapper)
# assign our actor to the renderer
renderer.AddActor(mydataActor)

# enable user interface interactor
myInteractiveWindow.Initialize()
myInteractiveWindow.Start()
```



demo:

PYVTK

exercises

- create your own ascii (tab, comma) data file
  - have at least 100 data records
  - have at least 4 data variables
  - if you have some ascii based data already
    - you can used awk/sed to filter data
  - get mondrian, molegro dm, xmdv, or tableau
    - make some screenshots!
- create your own vtk data file
  - have at least 100 data points/grid points
  - if you have some ascii based data already
    - you can used awk/sed to filter data
    - then you can manually add the vtk tags
- if you feel ambitious, get vtk, write a py program

thanks!

avyakta.caltech.edu:8888/esci101

methods of computational science

visualization

part i - jumpstart/tools

santiago v lombeyda

center for advanced computing research

caltech