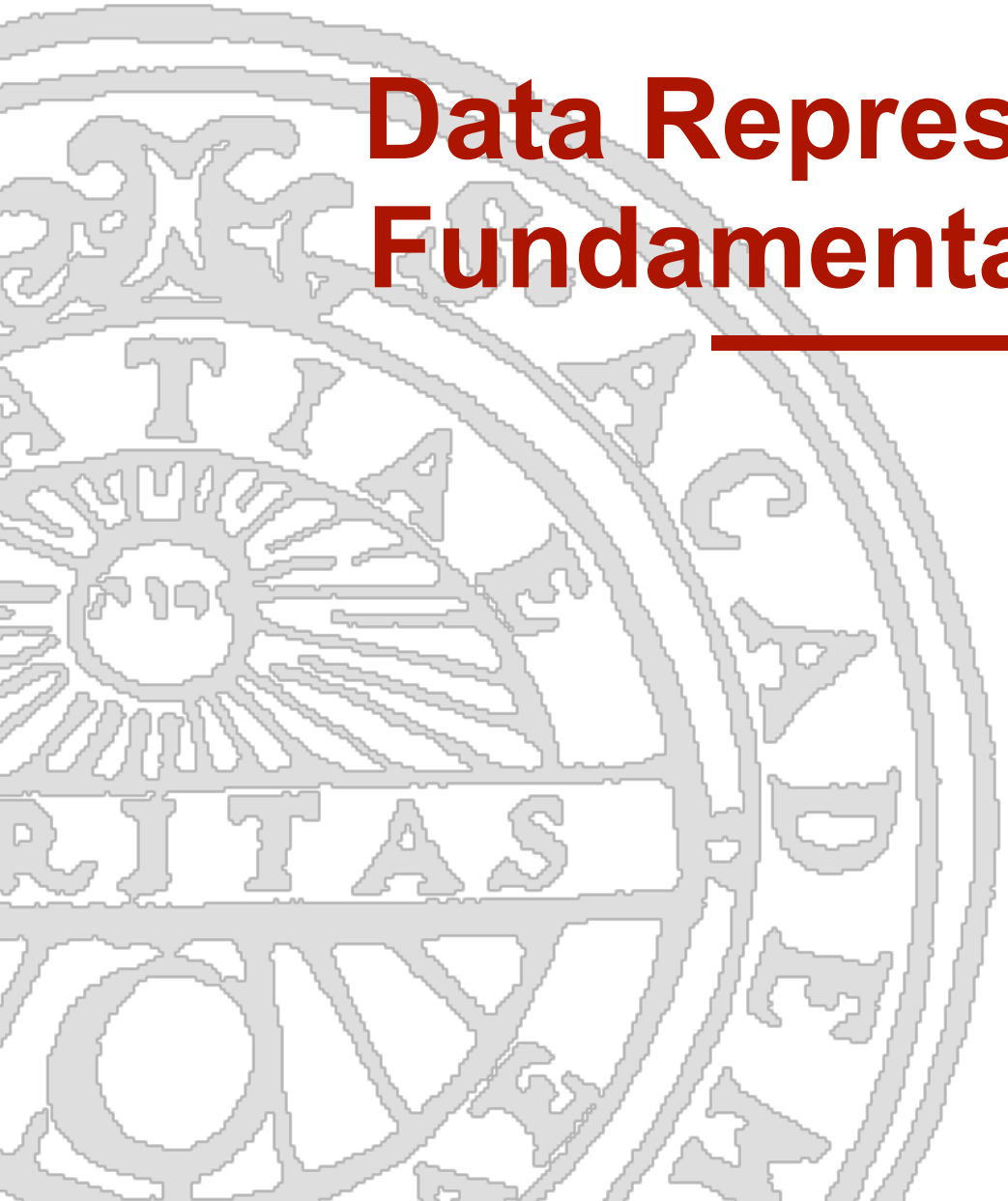


Data Representation and Fundamental Algorithms

Anders Hast

Data Representation

- Cells
  - Linear
  - Non linear
- Topolygy vs. Geometry
- Attribute Data
  - Scalar
  - Vectors
  - Normals
  - Texture Coordinates
  - Tensors

Cells

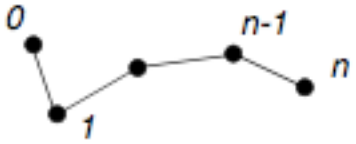
- A dataset consists of one or more *cells*
- They are the fundamental building block
- Cells are defined by specifying:
  - a type
  - an ordered list of points

Linear Cell Types

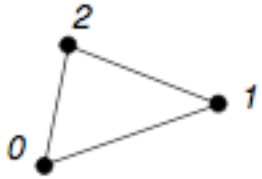
VTK_VERTEX (=1)

VTK_POLY_VERTEX (=2)

VTK_LINE (=3)

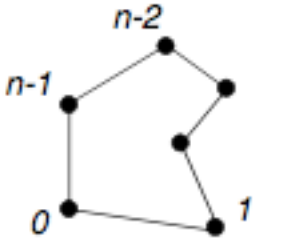


VTK_POLY_LINE (=4)

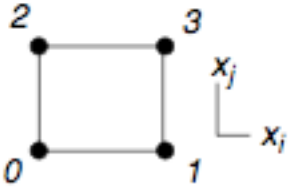


VTK_TRIANGLE(=5)

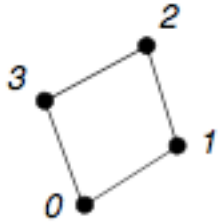
VTK_TRIANGLE_STRIP (=6)



VTK_POLYGON (=7)

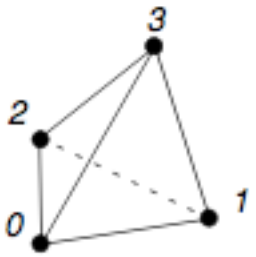


VTK_PIXEL (=8)

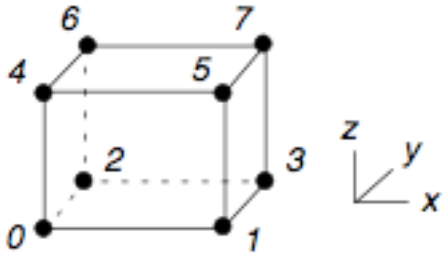


VTK_QUAD (=9)

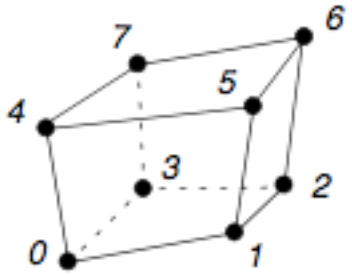
More Linear Cell Types



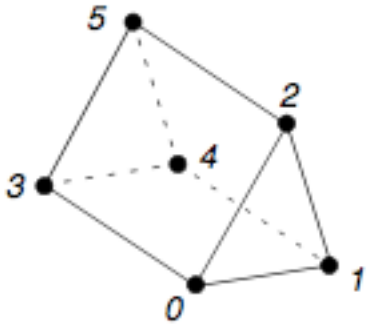
VTK_TETRA (=10)



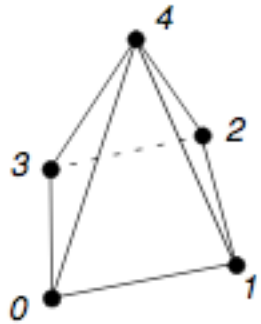
VTK_VOXEL (=11)



VTK_HEXAHEDRON (=12)



VTK_WEDGE (=13)

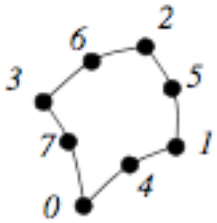


VTK_PYRAMID (=14)

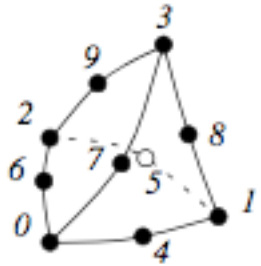
Non Linear Types

VTK_QUADRATIC_EDGE
(=21)

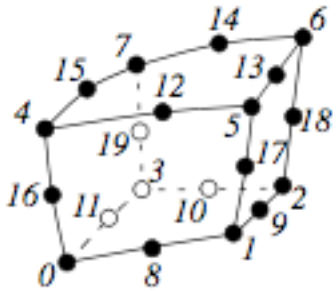
VTK_QUADRATIC_TRIANGLE
(=22)



VTK_QUADRATIC_QUAD
(=23)

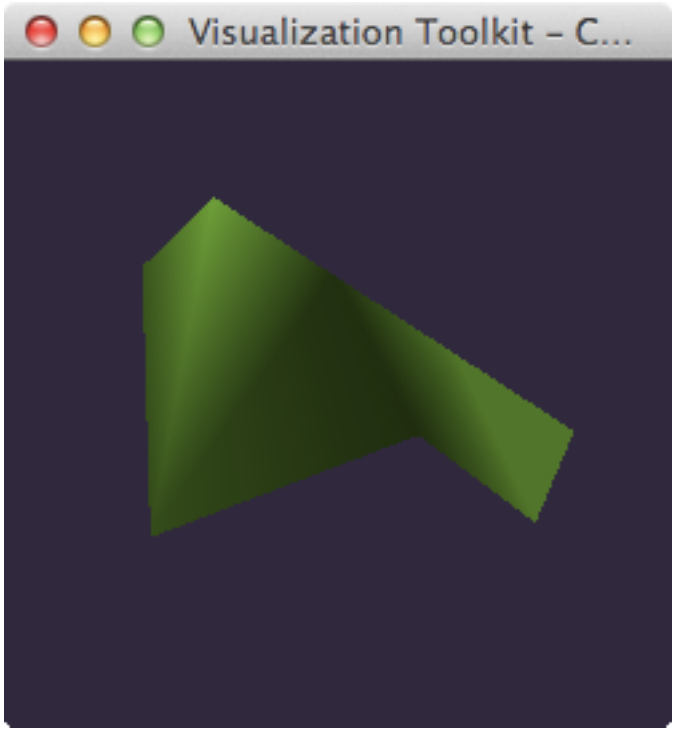


VTK_QUADRATIC_TETRA
(=24)



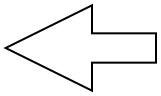
VTK_QUADRATIC_HEXAHEDRON
(=25)

Strips



```
from vtk import *

points = vtkPoints()
points.InsertPoint(0, 0.0, 0.0, 0.0)
points.InsertPoint(1, 0.0, 1.0, 0.0)
points.InsertPoint(2, 1.0, 0.0, 0.0)
points.InsertPoint(3, 1.0, 2.0, 1.0)
points.InsertPoint(4, 2.0, 0.0, 0.0)
points.InsertPoint(5, 2.0, 2.0, -1.0)
points.InsertPoint(6, 3.0, 0.0, 0.0)
points.InsertPoint(7, 3.0, 1.0, 0.0)
```

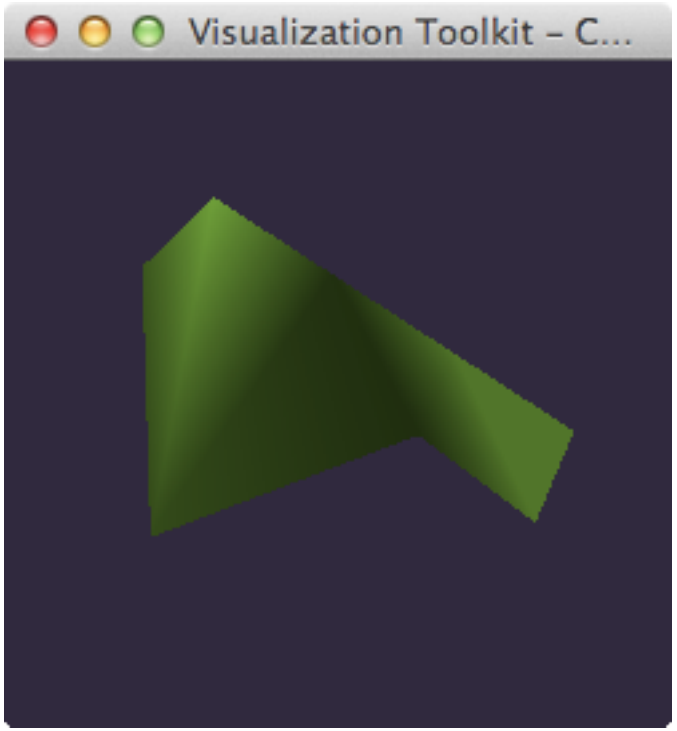
```
cells = vtkCellArray()
cells.InsertNextCell(8) # number of points
cells.InsertCellPoint(0)
cells.InsertCellPoint(1)
cells.InsertCellPoint(2)
cells.InsertCellPoint(3)
cells.InsertCellPoint(4)
cells.InsertCellPoint(5)
cells.InsertCellPoint(6)
cells.InsertCellPoint(7)

strip = vtkPolyData()
strip.SetPoints(points)
strip.SetStrips(cells)
```



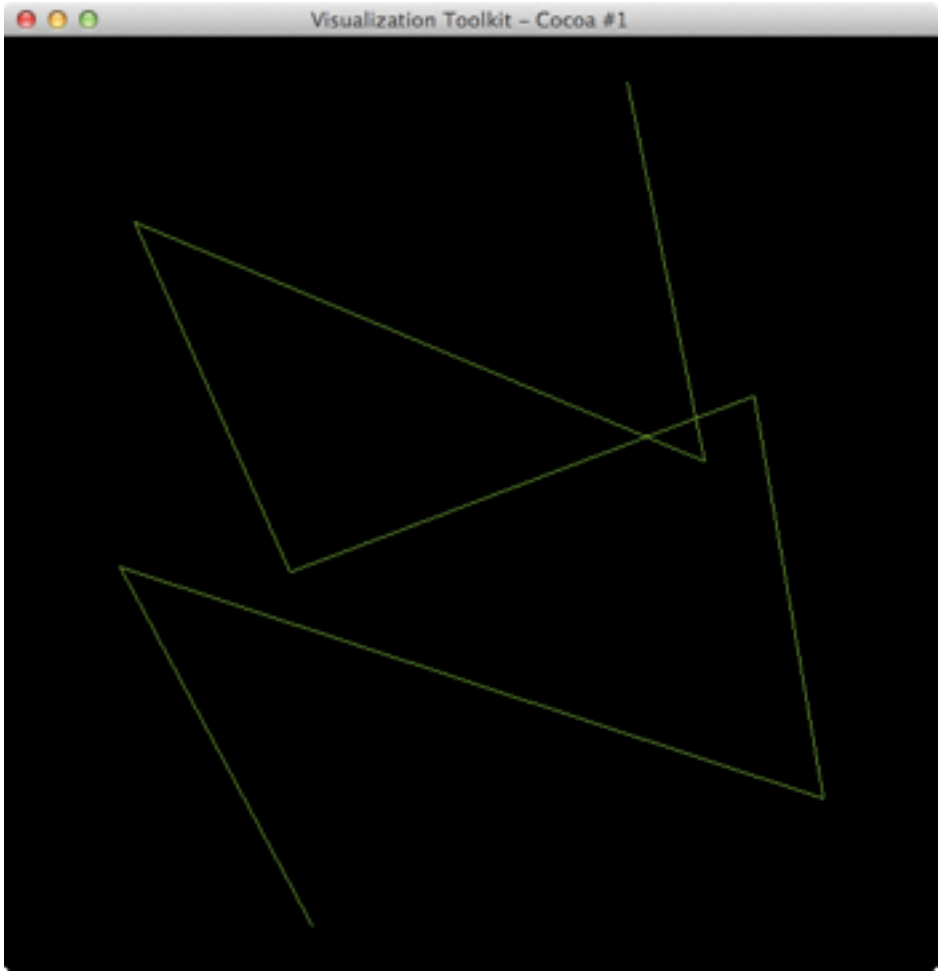
Specifies connection order

```
map = vtkPolyDataMapper()
map.SetInput(strip)

strip = vtkActor()
strip.SetMapper(map)
strip.GetProperty().SetColor(0.3800, 0.7000, 0.1600)

ren = vtkRenderer()
ren.AddActor(strip)
ren.SetBackground(0.2, 0.15, 0.25)

renWin = vtkRenderWindow()
renWin.AddRenderer(ren)
renWin.SetSize(250, 250)
renWin.Render()

iren = vtkRenderWindowInteractor()
iren.SetRenderWindow(renWin)
iren.Initialize()
iren.Start()
```

Result



PolyLine

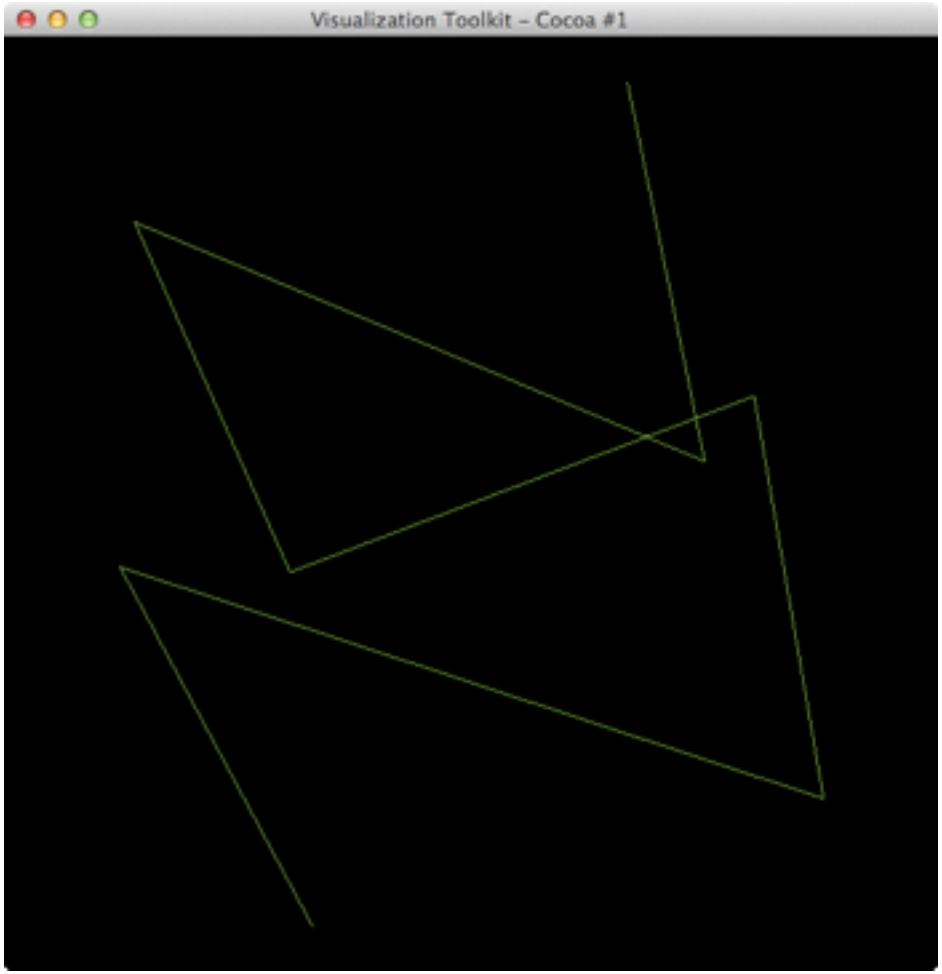


```
# load VTK
from vtk import *

lines   = vtkPolyData()
points = vtkPoints()
cells   = vtkCellArray()

points.InsertPoint(0,0,0,0)
points.InsertNextPoint(0,0,1)
points.InsertNextPoint(0,1,0)
points.InsertNextPoint(0,1,1)
points.InsertNextPoint(1,0,0)
points.InsertNextPoint(1,0,1)
points.InsertNextPoint(1,1,0)
points.InsertNextPoint(1,1,1)
```

```
cells.InsertNextCell(8)
cells.InsertCellPoint(0)
cells.InsertCellPoint(1)
cells.InsertCellPoint(2)
cells.InsertCellPoint(3)
cells.InsertCellPoint(4)
cells.InsertCellPoint(5)
cells.InsertCellPoint(6)
cells.InsertCellPoint(7)

lines.SetPoints(points)
lines.SetLines(cells)

lineMapper=vtkPolyDataMapper()
lineMapper.SetInput(lines)
lineActor=vtkActor()
lineActor.SetMapper(lineMapper)
lineActor.GetProperty().SetColor(0.3800, 0.7000, 0.1600)
```

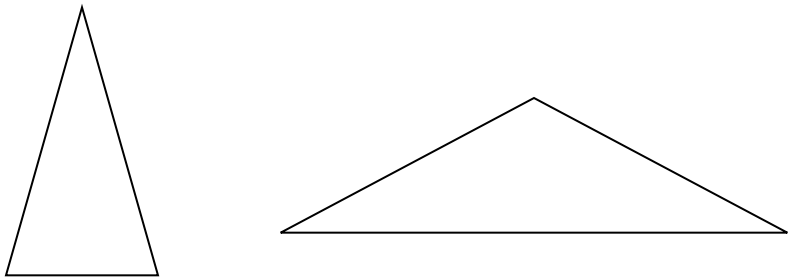
```
ren = vtk.vtkRenderer()
ren.AddActor(lineActor)

renWin = vtk.vtkRenderWindow()
renWin.AddRenderer(ren)
renWin.SetWindowName("Line")
renWin.SetSize(600,600)
renWin.Render()

iren = vtk.vtkRenderWindowInteractor()
iren.SetRenderWindow(renWin)
iren.Initialize()
iren.Start()
```

Result



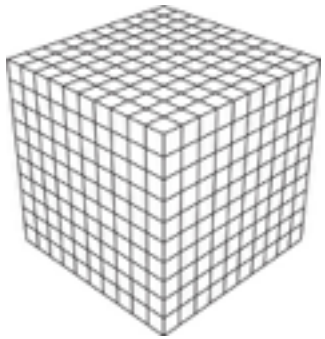
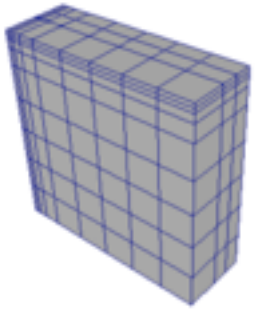
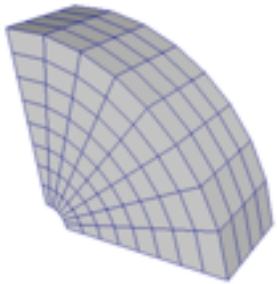
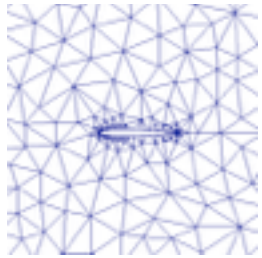
Topology vs. Geometry

- Each type has its topology
- Example:
  - A triangle has three vertices but a line has only two, etc
- Geometry
  - Can differ within the same type



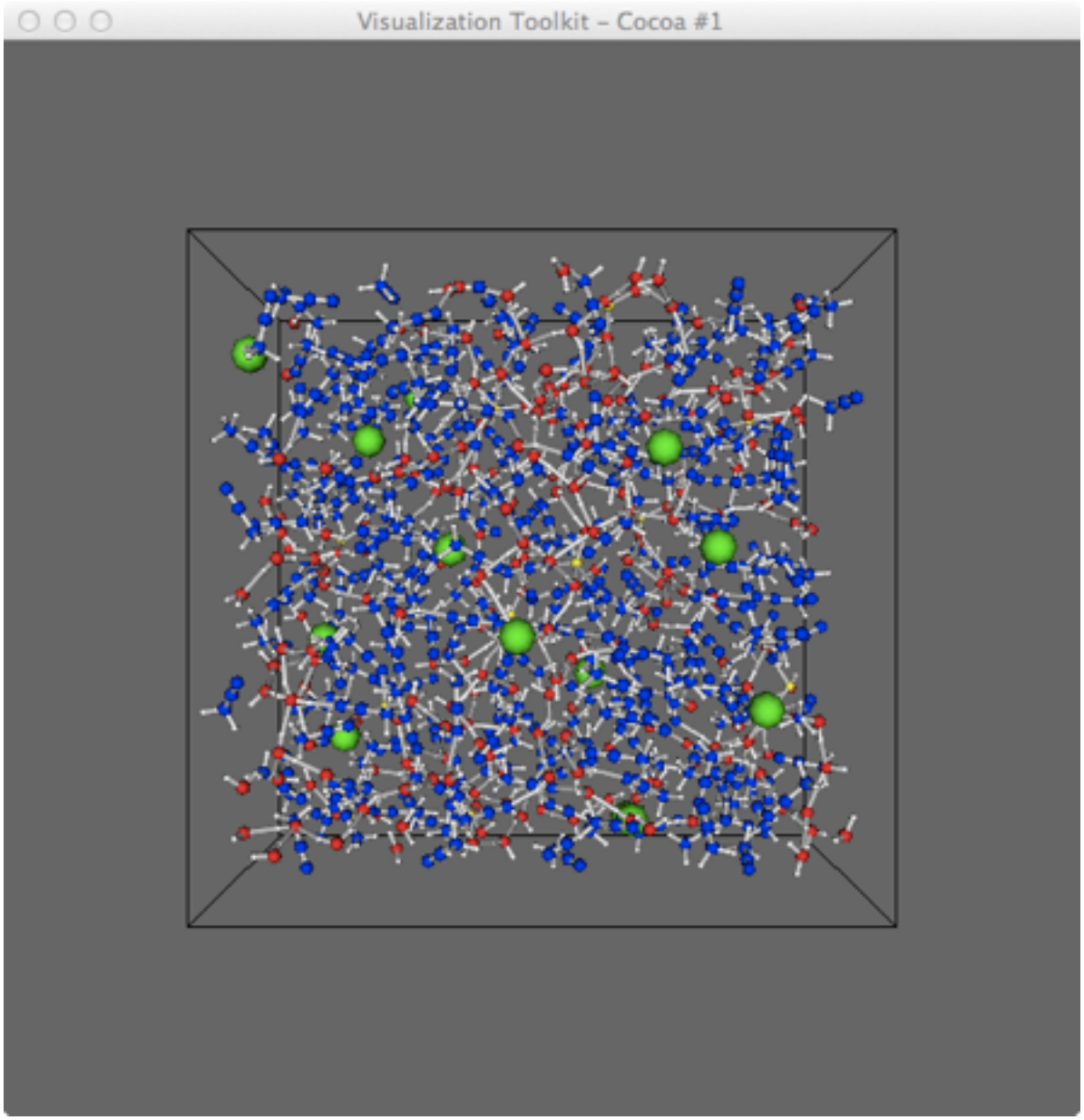
Dataset Types

- Uniform Grid
  - Regular Topology
  - regular Geometry
- Rectilininear Grid
  - Regular Topology
  - Partially Regular Geometry
- Structured grids
  - Regular Topology
  - Irregular Geometry
- Unstructured grids
  - Irregular Geometry
  - No Structure

UPPSALA UNIVERSITET

Informationsteknologi



Example

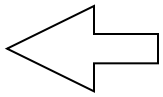
```
from vtk import *
from ReadAtoms import *

# Read the data into a vtkPolyData using the functions
# in ReadPoints.py
data = vtk.vtkPolyData()
data.SetPoints(readPoints("coordinates.txt"))
data.GetPointData().SetScalars(readScalars("radii.txt"))
data.SetLines(readConnections("connections.txt"))
.
.
.
```

ReadAtoms

```
# Reading Atom data

import string
import vtk

def readPoints(filename):
    points = vtk.vtkPoints()
    file = open(filename)

    line = file.readline()
    while line:
        data = string.split(line)

        if data and data[0] != '#':
            x, y, z = float(data[0]), float(data[1]), float(data[2])
            points.InsertNextPoint(x, y, z)
        line = file.readline()
    return points
```

```
def readScalars(filename):
    scalars = vtk.vtkFloatArray()
    file = open(filename)

    line = file.readline()
    while line:
        data = string.split(line)
        if data and data[0] != '#':
            x= float(data[0])
            scalars.InsertNextValue(x)
        line = file.readline()
    return scalars
```

```
def readConnections(filename):
    connections=vtk.vtkCellArray()
    file = open(filename)

    line = file.readline()
    while line:
        data = string.split(line)
        if data and data[0] != '#':
            a, b = int(data[0]), int(data[1])
            connections.InsertNextCell(2)
            connections.InsertCellPoint(a)
            connections.InsertCellPoint(b)
        line = file.readline()
    return connections
```

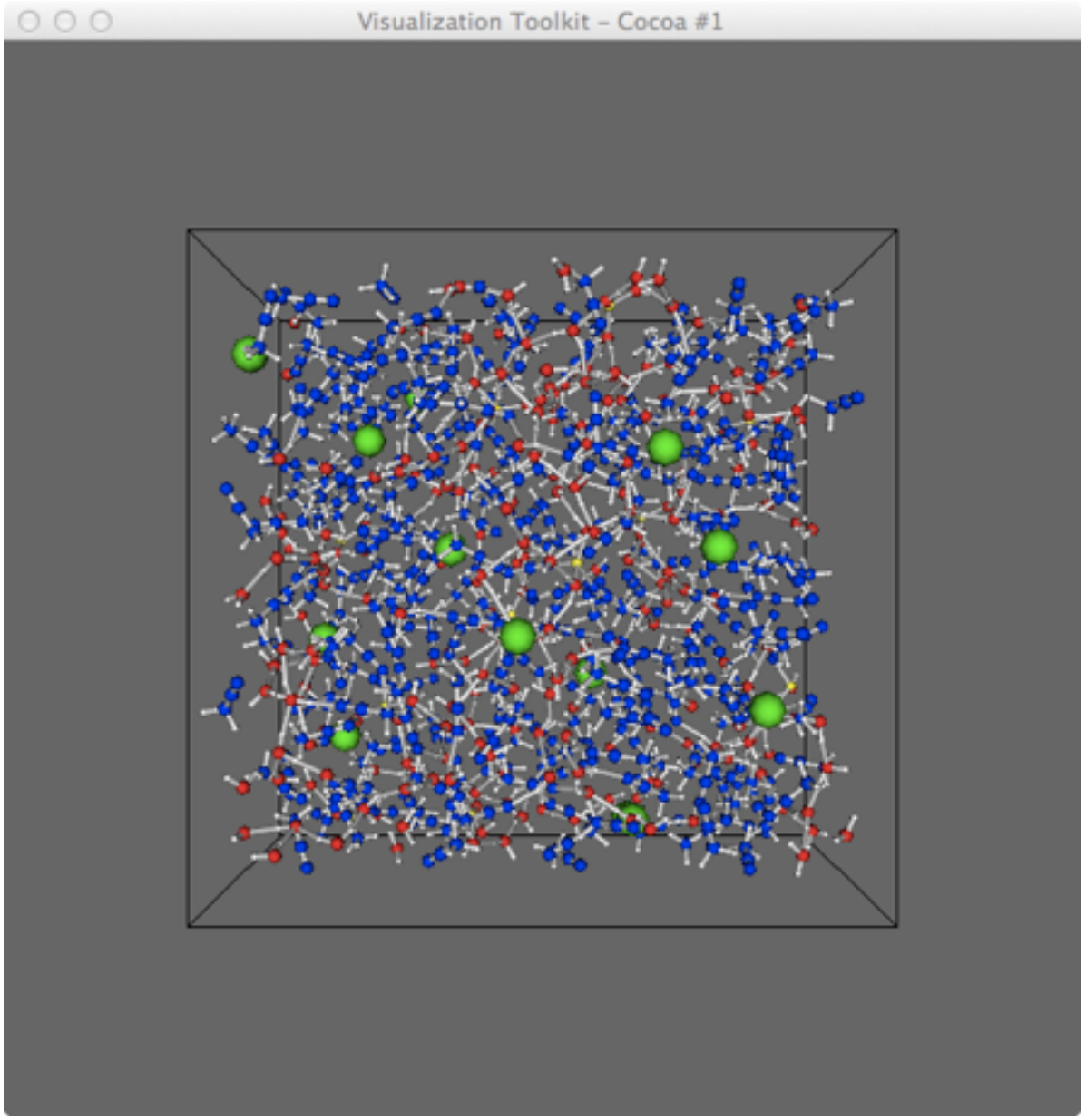


Specifies the connection

UPPSALA UNIVERSITET

Informationsteknologi

Result



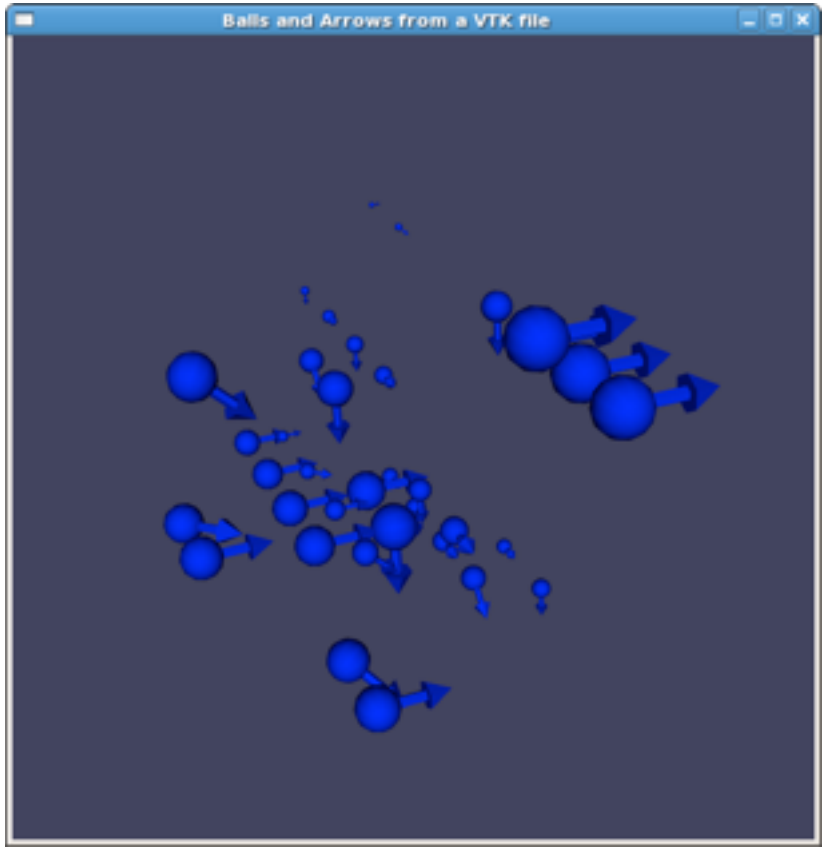
Summary

- A dataset consists of one or more *cells*
- Cells are defined by specifying:
  - a type
  - an ordered list of points
- The Topology describes the object
- The Geometry is defined by the coordinates of the object with a certain topology
- The Dataset types are defined by regular or irregular Toppology and Geometry

Fundamental Algorithms

- Colour Mapping
- Streamlines
- Cut Planes
- Glyphs
- Contouring
- Marching Cubes
- Hedgehogs

Glyphs



- Arrows
- Spheres
- etc

- How many attributes can be encoded here?

Attributes

- Ball
  - Radius
  - Colour/texture
- Arrow
  - Length of shaft
  - radius of tip
  - Colour/texture
    - Shaft
    - tip
- Etc… But don’t use too many!!

Glyphs

- vtkGlyph3D is a filter that copies a geometric representation (called a glyph) to every point in the input dataset. The glyph is defined with polygonal data from a source filter input.
- The glyph may be oriented along the input vectors or normals, and it may be scaled according to scalar data or vector magnitude.

Your own Files (Unstructured Grid)

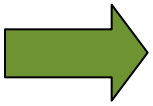
Informationsteknologi

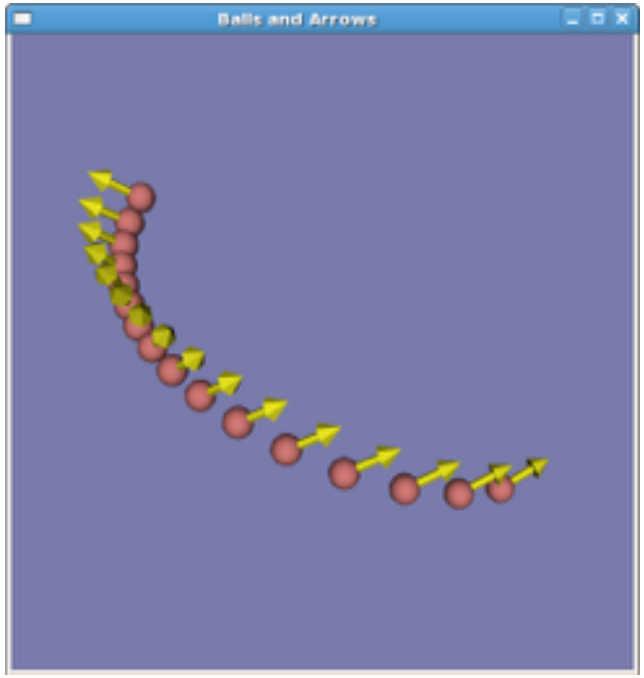
data1.txt

```
# nb =
   0.6744    0.1113    0.7300
   0.5362    0.2407    0.8091
   0.3671    0.3744    0.8515
   0.1866    0.4948    0.8487
   0.0175    0.5904    0.8069
  -0.1259    0.6597    0.7409
  -0.2399    0.7078    0.6644
  -0.3273    0.7412    0.5860
  -0.3929    0.7654    0.5096
  -0.4409    0.7843    0.4364
  -0.4746    0.8005    0.3661
  -0.4958    0.8159    0.2973
  -0.5053    0.8321    0.2287
  -0.5023    0.8501    0.1581
  -0.4845    0.8708    0.0830
  -0.4472    0.8944    0.0000
```

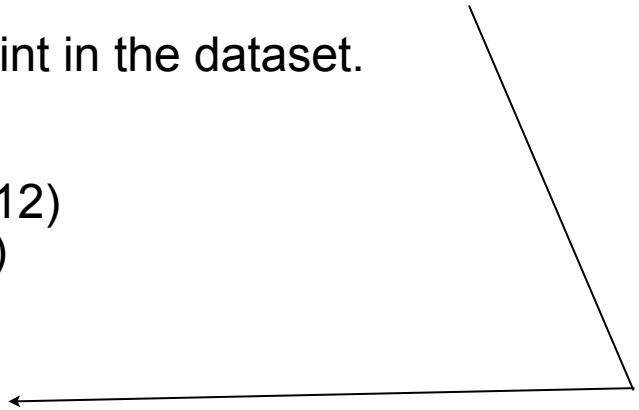
data2.txt

```
# c =
   0.8798    0.0573   -0.4718
   0.9105    0.1226   -0.3950
   0.9299    0.1968   -0.3107
   0.9345    0.2806   -0.2189
   0.9194    0.3746   -0.1200
   0.8781    0.4783   -0.0154
   0.8029    0.5890    0.0916
   0.6864    0.7007    0.1944
   0.5248    0.8028    0.2831
   0.3230    0.8813    0.3449
   0.0976    0.9243    0.3690
  -0.1271    0.9275    0.3515
  -0.3287    0.8966    0.2968
  -0.4943    0.8425    0.2141
  -0.6198    0.7766    0.1128
  -0.7071    0.7071   -0.0000
```





```
import sys
from vtk import *

# This import style makes it possible to write just readPoints
from ReadPoints import *

# Read the data into a vtkPolyData using the functions in ReadPoints.py
data=vtkUnstructuredGrid()

data.SetPoints(readPoints(sys.argv[1]))
data.GetPointData().SetVectors(readVectors(sys.argv[2]))

# Put spheres at each point in the dataset.
ball = vtkSphereSource()
ball.SetRadius(0.05)
ball.SetThetaResolution(12)
ball.SetPhiResolution(12)

ballGlyph = vtkGlyph3D()
ballGlyph.SetInput(data)
ballGlyph.SetSourceConnection(ball.GetOutputPort())
```



```
ballMapper = vtkPolyDataMapper()
ballMapper.SetInputConnection(ballGlyph.GetOutputPort())

ballActor = vtkActor()
ballActor.SetMapper(ballMapper)
ballActor.GetProperty().SetColor(0.8,0.4,0.4)

arrow = vtkArrowSource()
arrow.SetTipRadius(0.2)
arrow.SetShaftRadius(0.075)

arrowGlyph = vtkGlyph3D()
arrowGlyph.SetInput(data)
arrowGlyph.SetSourceConnection(arrow.GetOutputPort())
arrowGlyph.SetScaleFactor(0.2)

arrowMapper = vtkPolyDataMapper()
arrowMapper.SetInputConnection(arrowGlyph.GetOutputPort()

arrowActor = vtkActor()
arrowActor.SetMapper(arrowMapper)
arrowActor.GetProperty().SetColor(0.9,0.9,0.1)
```

```
ren = vtkRenderer()
ren.AddActor(ballActor)
ren.AddActor(arrowActor)
ren.SetBackground(0.4, 0.4, 0.6)

renWin = vtkRenderWindow()
renWin.AddRenderer(ren)
renWin.SetWindowName("Balls and Arrows")
renWin.SetSize(500,500)

iren = vtkRenderWindowInteractor()
iren.SetRenderWindow(renWin)
iren.Initialize()
iren.Start()
```

ReadPoints

```
import string
#Read Points
def readPoints(file):
    # Create an array of Points
    points = vtkPoints()

    #Open the file
    file = open(file)
    # Read one line
    line = file.readline()
    # Loop through lines
    while line:
        # Split the line into data
        data = string.split(line)

        # Skip the commented lines
        if data and data[0] != '#':
            # Convert data into floats
            x, y, z = float(data[0]), float(data[1]), float(data[2])

            # Insert floats into the point array
            points.InsertNextPoint(x, y, z)

        # read next line
        line = file.readline()

    return points;
```

UPPSALA UNIVERSITET

Informationsteknologi

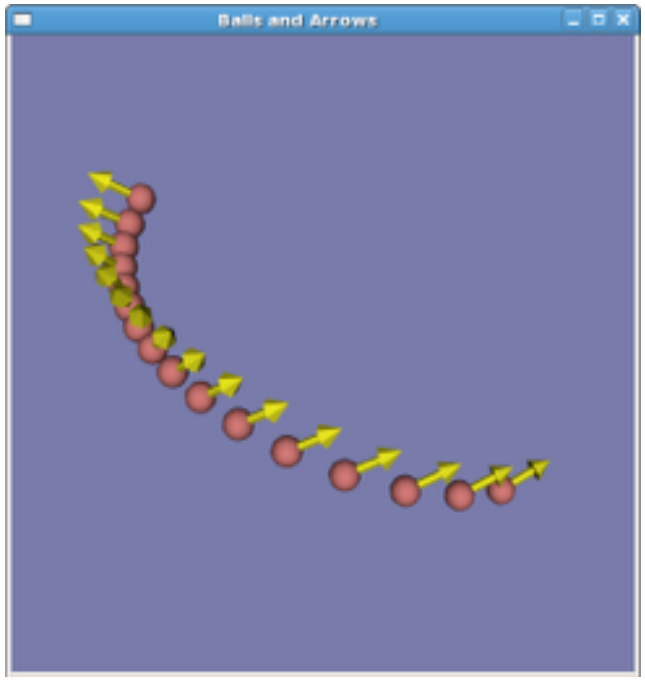
Read Vectors

```
# Read Vectors.
# This method works in the same way as readPoints but
returns a different type of array
def readVectors(file):
    # Create a Double array which represents the vectors
    vectors = vtkDoubleArray()

    # Define number of elements
    vectors.SetNumberOfComponents(3)

    file = open(file)
    line = file.readline()
    while line:
        data = string.split(line)
        if data and data[0] != '#':
            x, y, z = float(data[0]), float(data[1]), float(data[2])
            vectors.InsertNextTuple3(x, y, z)
        line = file.readline()
    return vectors
```

UPPSALA UNIVERSITET

Informationsteknologi

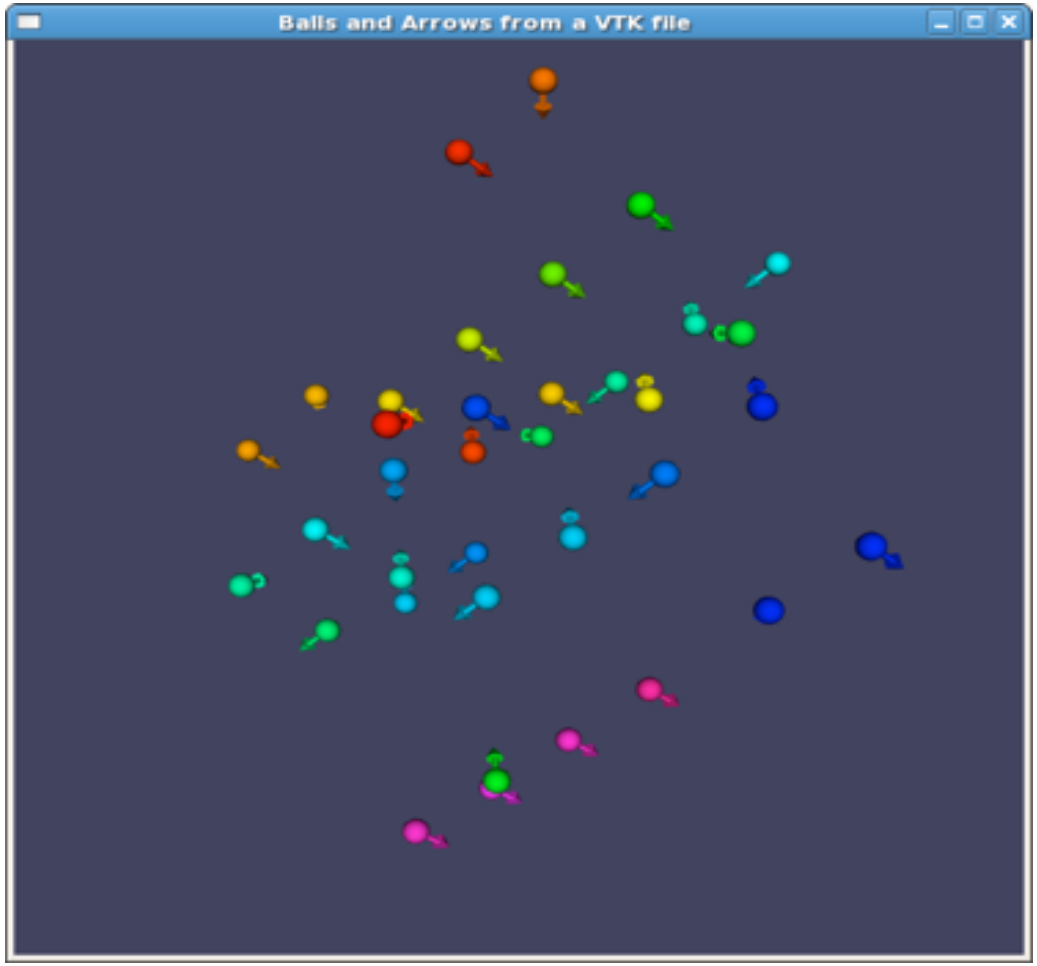
Result



UPPSALA UNIVERSITET

Informationsteknologi

Colour Mapping and Colour Transfer Functions



```
import sys
from vtk import *

# Use the VTK reader to read the vtk file
reader = vtkUnstructuredGridReader()

# Don't forget to give the file name as an argument: "python Vectors.py data.vtk"
reader.SetFileName(sys.argv[1])

# Put spheres at each point in the dataset
ball = vtkSphereSource()
ball.SetRadius(0.12)
ball.SetThetaResolution(12)
ball.SetPhiResolution(12)

ballGlyph = vtkGlyph3D()
ballGlyph.SetSourceConnection(ball.GetOutputPort())
ballGlyph.SetInputConnection(reader.GetOutputPort())

# We do not want the Ball to have the size depending on the Scalar
ballGlyph.SetScaleModeToDataScalingOff()

ballMapper = vtkPolyDataMapper()
ballMapper.SetInputConnection(ballGlyph.GetOutputPort())
```

UPPSALA UNIVERSITET

Informationsteknologi

Colour Transfer Function

```
# Create a color transfer function to be used for both the balls and arrows.
colorTransferFunction = vtkColorTransferFunction()
colorTransferFunction.AddRGBPoint(5.0 , 0.0, 0.0, 1.0)
colorTransferFunction.AddRGBPoint(10.0, 0.0, 1.0, 1.0)
colorTransferFunction.AddRGBPoint(15.0, 0.0, 1.0, 0.0)
colorTransferFunction.AddRGBPoint(20.0, 1.0, 1.0, 0.0)
colorTransferFunction.AddRGBPoint(25.0, 1.0, 0.0, 0.0)
colorTransferFunction.AddRGBPoint(30.0, 1.0, 0.0, 1.0)

# Set colors depending on the color transfer functions
ballMapper.SetLookupTable(colorTransferFunction)
```

```
ballActor = vtkActor()
ballActor.SetMapper(ballMapper)
```

```
#Put an arrow (vector) at each ball
arrow = vtkArrowSource()
arrow.SetTipRadius(0.2)
arrow.SetShaftRadius(0.075)

arrowGlyph = vtkGlyph3D()
arrowGlyph.SetInputConnection(reader.GetOutputPort())
arrowGlyph.SetSourceConnection(arrow.GetOutputPort())
arrowGlyph.SetScaleFactor(0.4)

# We do not want the Arrow's size to depend on the Scalar
arrowGlyph.SetScaleModeToDataScalingOff()

arrowMapper = vtkPolyDataMapper()
arrowMapper.SetInputConnection(arrowGlyph.GetOutputPort())

# Set colors depending on the color transfer functions
arrowMapper.SetLookupTable(colorTransferFunction)

arrowActor = vtkActor()
arrowActor.SetMapper(arrowMapper)
```

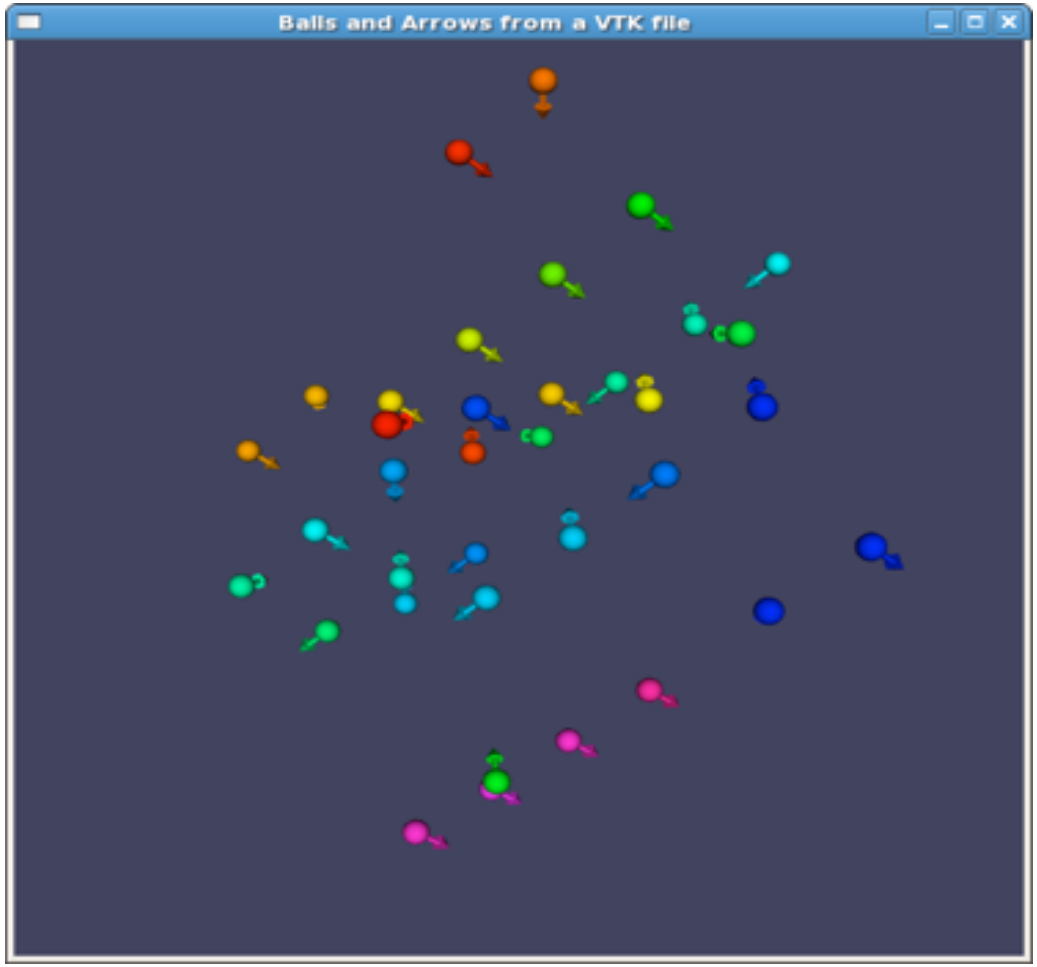
As usual...

```
# Create the RenderWindow, Renderer and Interactor
ren = vtkRenderer()
ren.AddActor(ballActor)
ren.AddActor(arrowActor)
ren.SetBackground(0.2, 0.2, 0.3)

renWin = vtkRenderWindow()
renWin.AddRenderer(ren)
renWin.SetWindowName("Balls and Arrows from a VTK file")
renWin.SetSize(600,600)

iren = vtkRenderWindowInteractor()
iren.SetRenderWindow(renWin)
iren.Initialize()
iren.Start()
```

UPPSALA UNIVERSITET

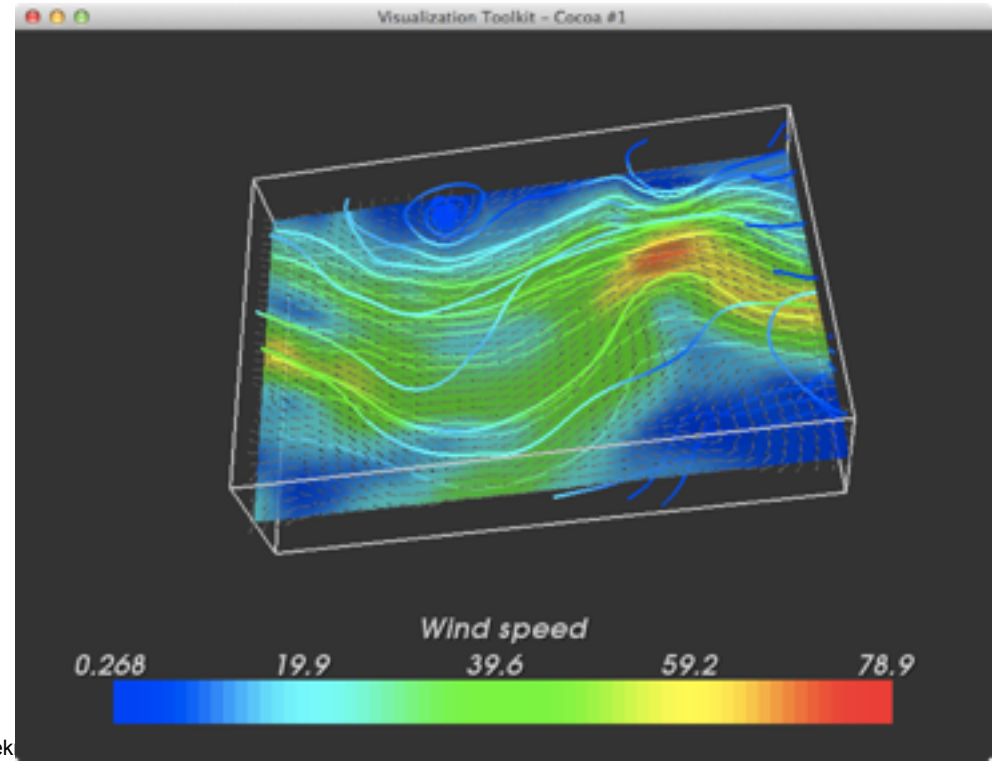
Informationsteknologi

Result



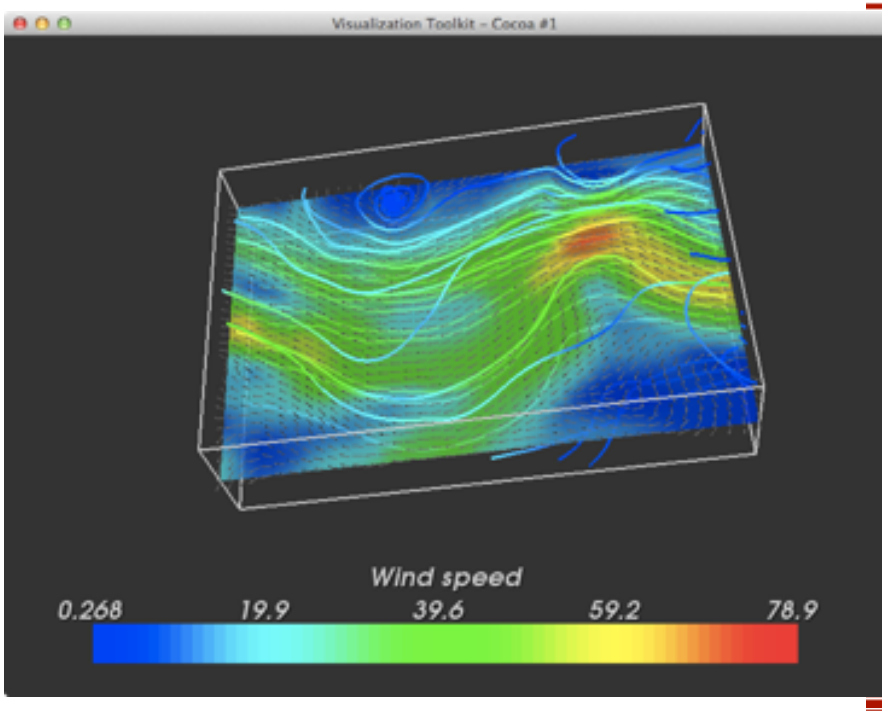
Colour Mapping

- Maps scalar data to colour
- Can be done by a colour look up table



```
from vtk import *

reader = vtkStructuredPointsReader()
reader.SetFileName("wind.vtk")
reader.Update()

a,b = reader.GetOutput().GetScalarRange()
W,H,D = reader.GetOutput().GetDimensions()

lut = vtkLookupTable()
lut.SetHueRange(0.667, 0.0)
lut.SetTableRange(a,b)
```

Streamlines

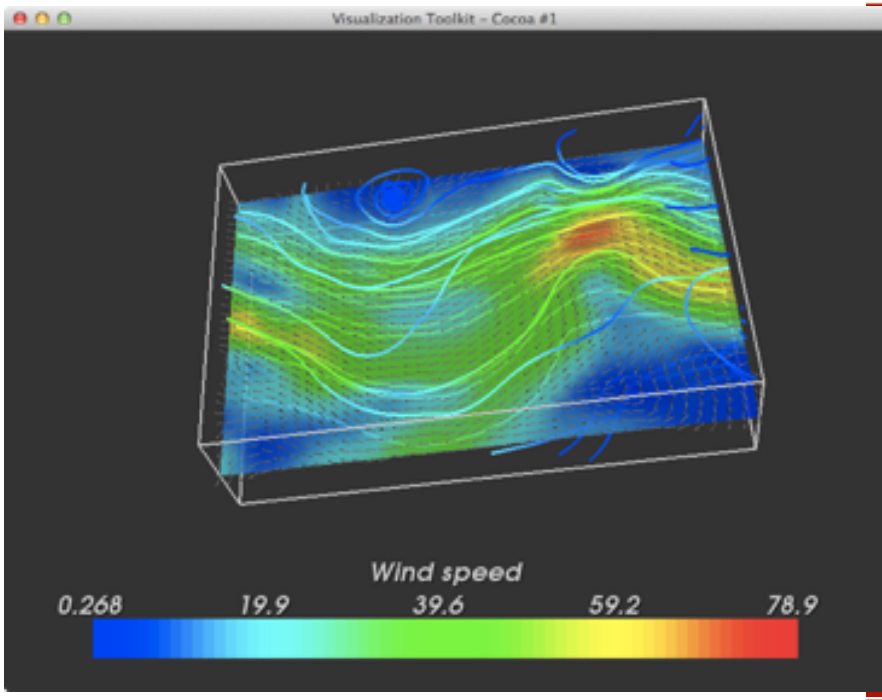


- Like driving a car by following the GPS
- If you do not turn when the GPS tells you to turn, you'll get out of course.
- Hence, it is important to check direction often = short sample step

Streamlines

```
streamer = vtkStreamLine()
plane = vtk.vtkPlaneSource()
.
.
.
streamer.SetSource(plane.GetOutput())
streamer.SetInput(reader.GetOutput())
streamer.SetIntegrationDirectionToForward()
streamer.SetIntegrator(vtk.vtkRungeKutta4())
streamer.SetStepLength(0.05)

# Connect to streamer to a mapper etc...
```

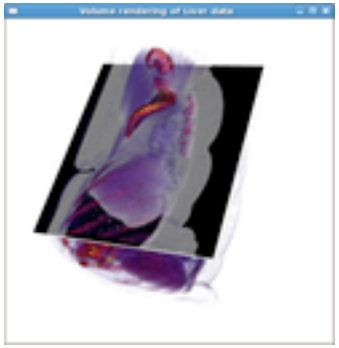
Cut Plane




```
sliceplane = vtkImageDataGeometryFilter()
sliceplane.SetInput(reader.GetOutput())
sliceplane.SetExtent(0,W,0,H,D/2,D/2)
sliceplaneMapper = vtkPolyDataMapper()
sliceplaneMapper.SetLookupTable(lut)
sliceplaneMapper.SetInput( sliceplane.GetOutput() )
sliceplaneMapper.SetScalarRange(a,b)
```

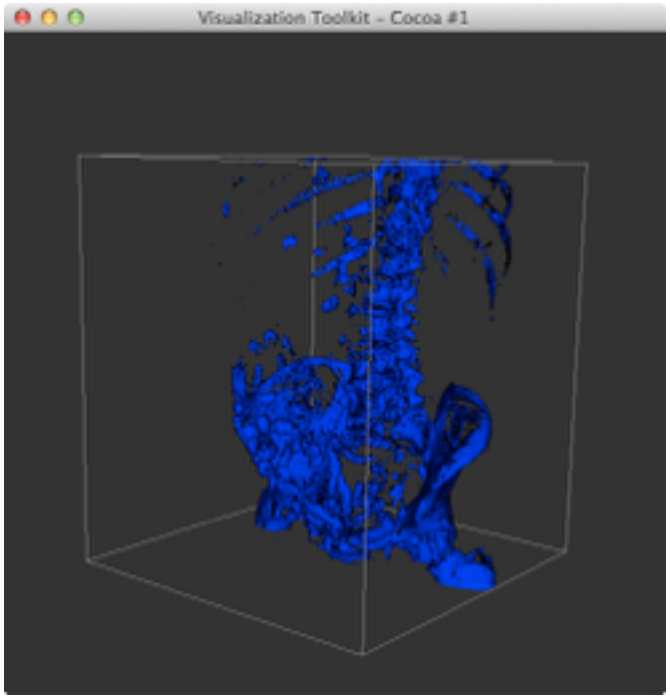
Cut Plane

```
plane=vtkImagePlaneWidget()
plane.SetInput(reader.GetOutput())
plane.SetSliceIndex(20)

plane.SetInteractor(iren)

plane.EnabledOn()


plane.SetPlaneOrientationToXAxes()
#plane.SetPlaneOrientationToYAxes()
#plane.SetPlaneOrientationToZAxes()
```



Contouring

- Creates an Isosurface
- That is: a surface where the volume data has the same value (iso = same)



Surface Rendering

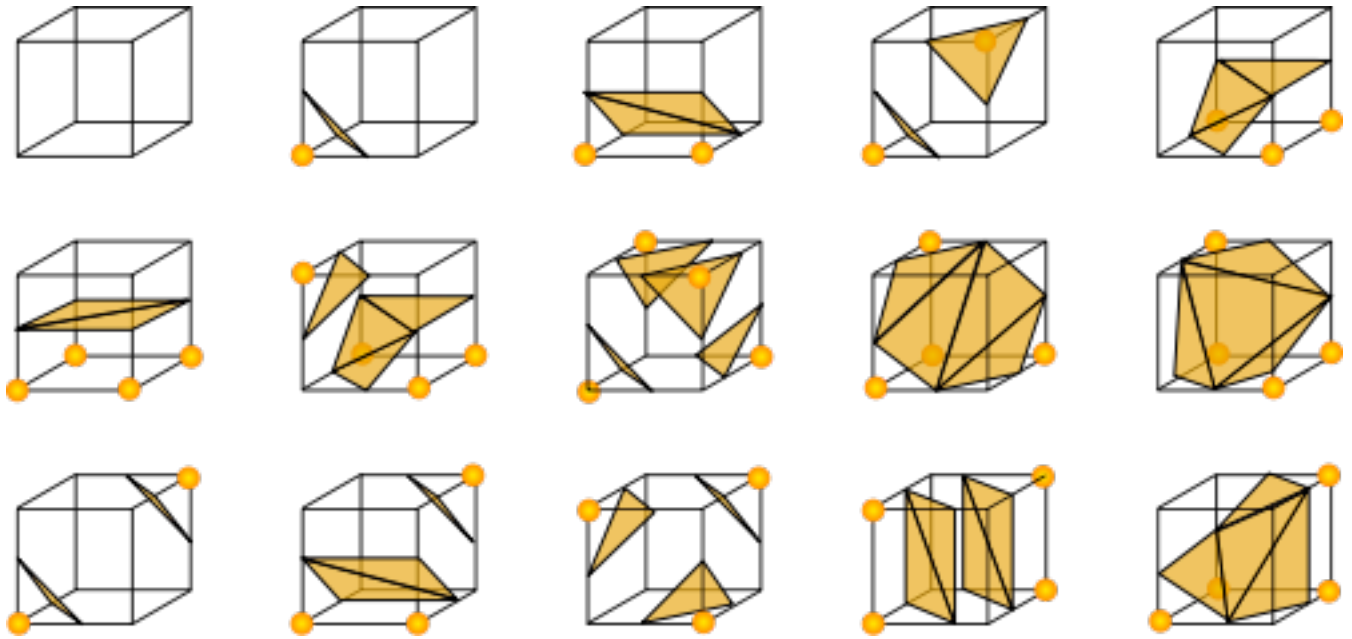
- Use Computer Graphics Techniques to render the surface (polygons)
- If triangles are used the surface is ”triangulated”
- These triangles are computed from the volume data using Marching Cubes

Isosurfaces

```
from vtk import *

# Read the volume
reader = vtkStructuredPointsReader()
reader.SetFileName("liver.vtk")

# Isosurface
isoSurface = vtkContourFilter()
isoSurface.SetInputConnection(reader.GetOutputPort())
isoSurface.SetValue(0, 160)
```

Marching Cubes

- 8 neighbouring voxels can be intersected in several ways.

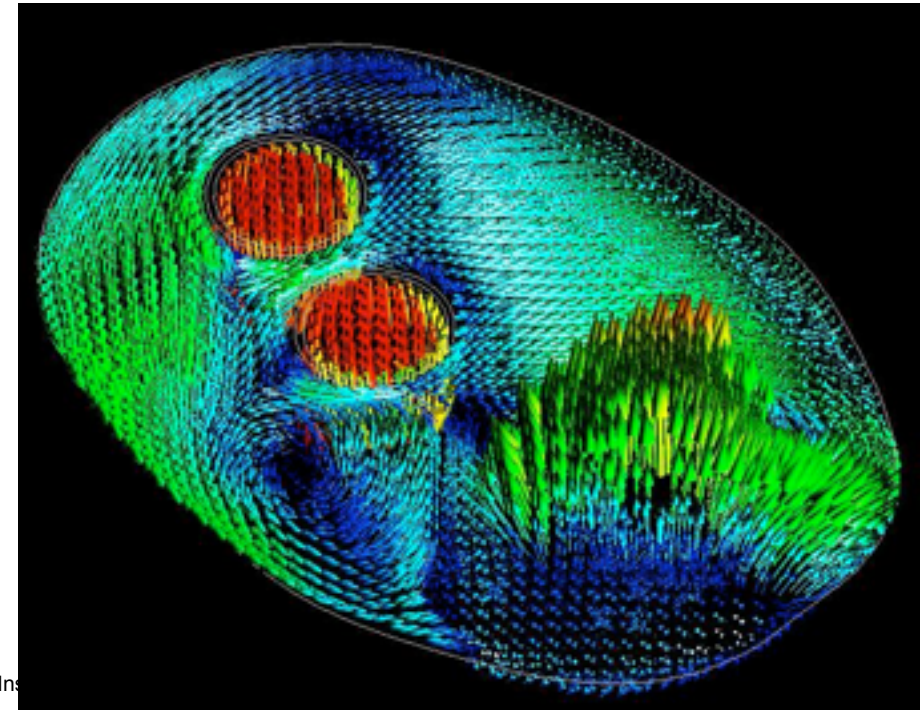


Marching Cubes

- These polygons must be stitched together.
- Lorensen et al. 1987
  - Ambiguities
- Similar techniques
  - Marching Squares 2D
  - Marching Tetrahedrons
    - No ambiguities

Hedgehogs

- Vector data is visualized as arrows at the point where they exist (looks like a hedgehog = sv. igelkott)



Conclusions

- Fundamental Algorithms
  - Colour Mapping
  - Streamlines
  - Cut Planes
  - Glyphs
  - Hedgehogs
  - Contouring
  - Marching Cubes