An Introduction to Visualization Using VTK

Geometric Modeling
3D Interaction / Widgets

William J. Schroeder, Kitware, Inc.

Overview

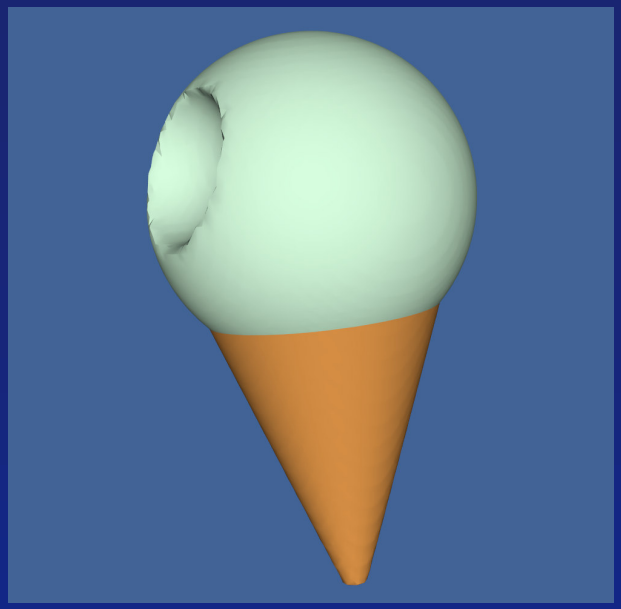
- Geometric Modeling
  - Implicit Functions / Implicit Modeling
  - Cutting / Clipping
  - Decimation
  - Surface smoothing
  - Surface normal generation
  - Triangle strip generation
  - Terrain
  - Other tricks (glyphing, tubing, shrinking, etc.)
- 3D Interaction / Widgets
  - The Role of Interaction
  - Command / Observer design pattern
  - 3D Widgets

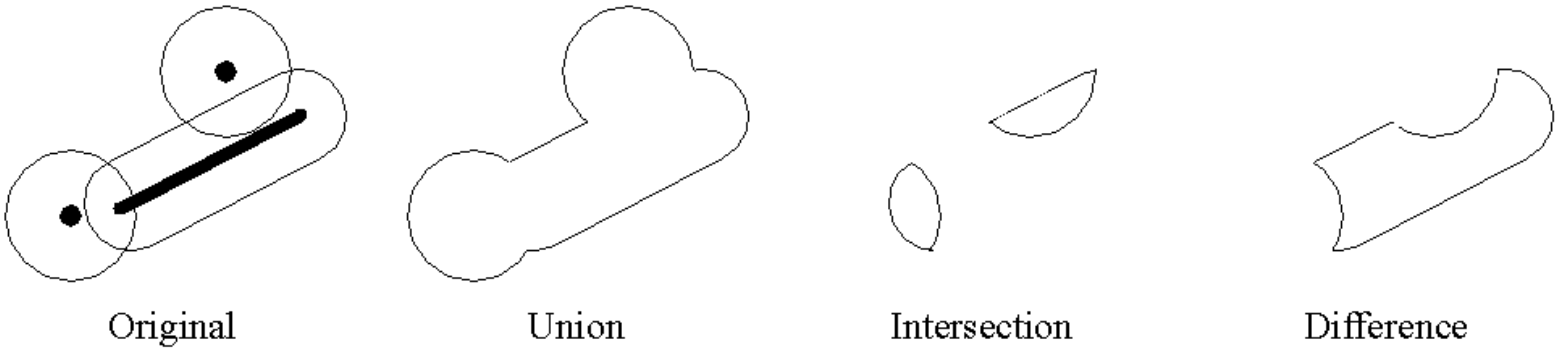
Implicit Functions

- Definition: F(x,y,z) = constant
  - Can generate scalar field with appropriate function F()
- Surface extracted by isocontouring
  (i.e., isocontour defined by F(x,y,z) = constant)
- Can represent complex shapes
  - Sphere, cone, plane, quadric functions, etc.
- Separate space: inside, on, and outside function
  - Clipping, cutting, thresholding
- Support boolean operations
  - Union, intersection, difference

Implicit Functions: Boolean Operations

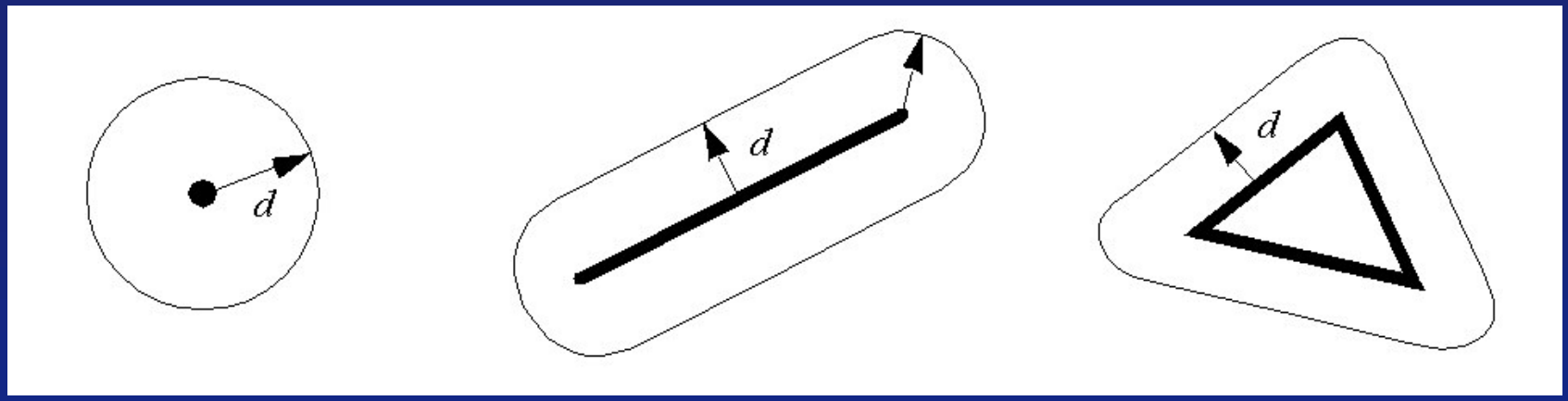
- F(x,y,z) union G(x,y,z):
  min(F,G) at each point (x,y,z)
- F(x,y,z) intersection G(x,y,z):
  max(F,G) at each point
- F(x,y,z) difference G(x,y,z):
  max(F, -G) at each point

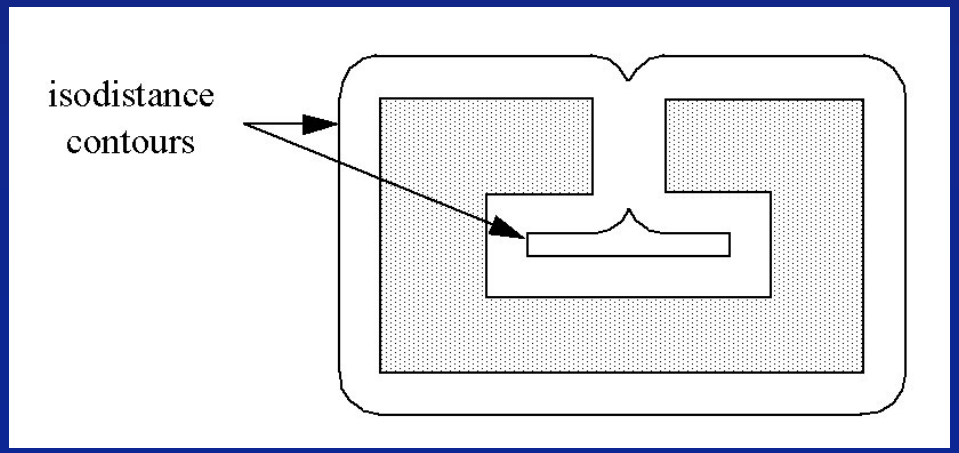


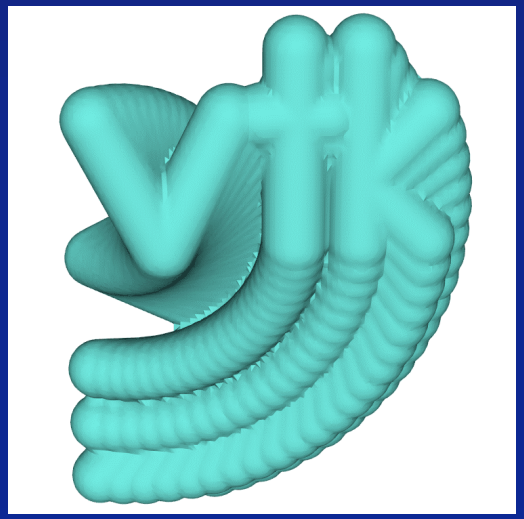


Implicit Modeling

- Define implicit function from generating primitives
  - Distance field





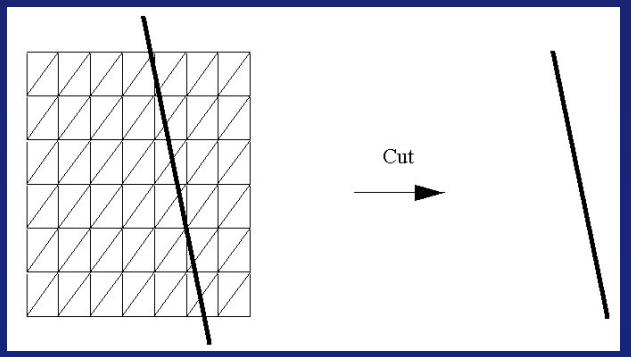


Implicit Modeling: VTK Example (in Tcl)

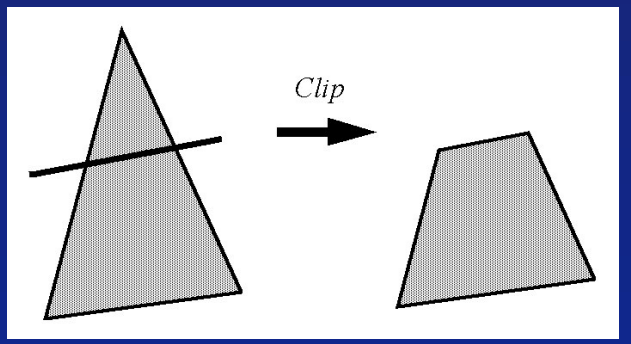

```
vtkSphere iceCream
    iceCream SetCenter 1.333 0 0
    iceCream SetRadius 0.5

vtkSphere bite
    bite SetCenter 1.5 0 0.5
    bite SetRadius 0.25

vtkImplicitBoolean theCream
    theCream SetOperationTypeToDifference
    theCream AddFunction iceCream
    theCream AddFunction bite

vtkSampleFunction theCreamSample
    theCreamSample SetImplicitFunction theCream
    theCreamSample SetModelBounds  0 2.5 -1.25 1.25 -1.25 1.25
    theCreamSample SetSampleDimensions 60 60 60
    theCreamSample ComputeNormalsOff

vtkMarchingContourFilter theCreamSurface
    theCreamSurface SetInput [theCreamSample GetOutput]
    theCreamSurface SetValue 0 0.0
```

Cutting and Clipping

- Cutting
  - Extract (n-1)-dimensional surface from n-dimensional data
  - Surface defined by F(x,y,z) = constant
  - Equivalent to iso-contouring



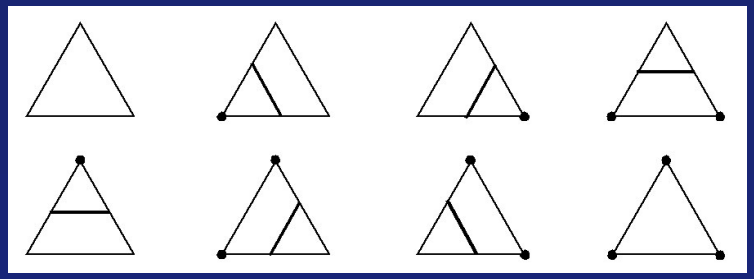
- Clipping
  - Extract n-dimensional region from n-dimensional data
  - Boundary defined by F(x,y,z) = constant
  - Boundary determined by iso-contouring operation



- In practice, clip and cut functions are either
  - Scalar values, or
  - Implicit function (equivalent to scalar field)

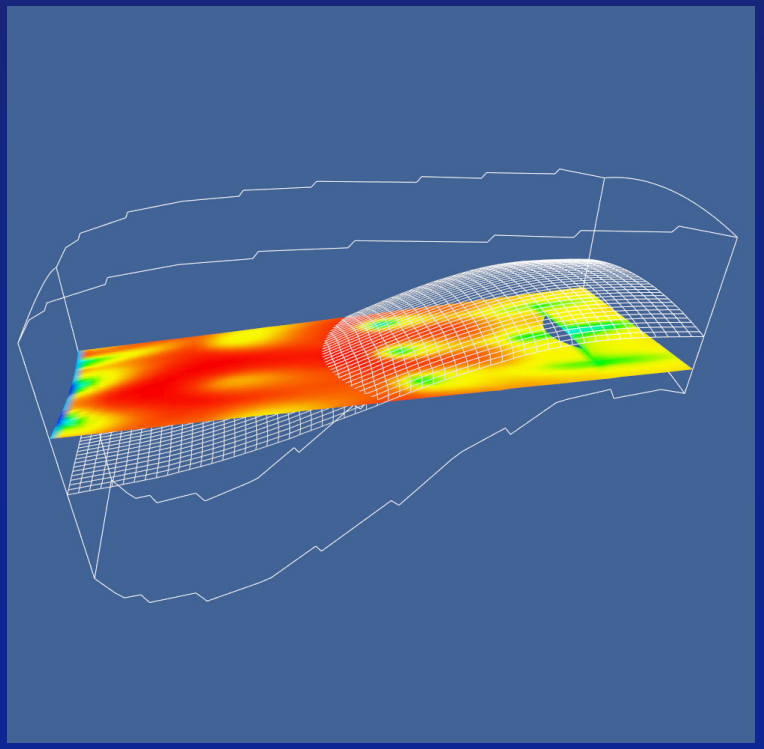
Cutting

- Implemented with case table (i.e., marching cubes)



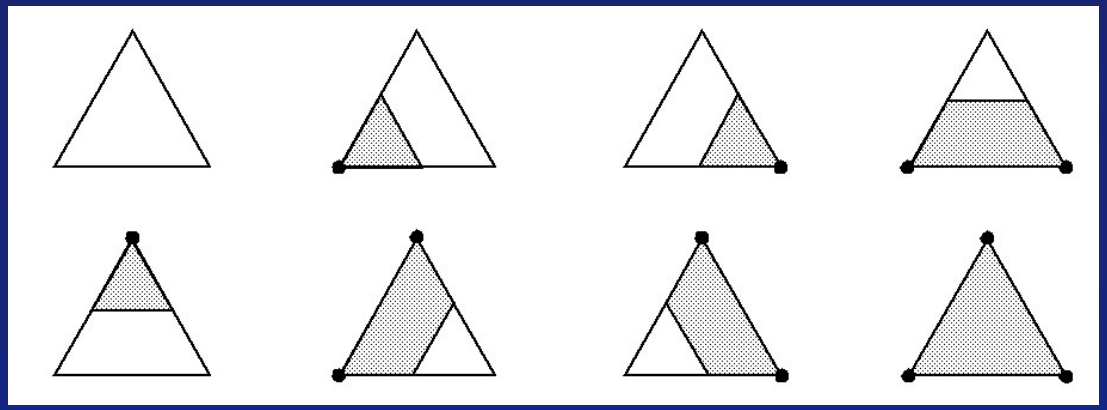
VTK Example (in C++)

```
vtkPlane *plane = vtkPlane::New();
   plane->SetOrigin( reader->GetOutput()->
                          GetCenter() );
   plane->SetNormal( -0.287, 0.0, 0.9579);

vtkCutter *planeCut = vtkCutter::New();
   planeCut->SetInput( reader->GetOutput() );
   planeCut->SetCutFunction(plane);
   planeCut->SetCutValue(0.0);
```

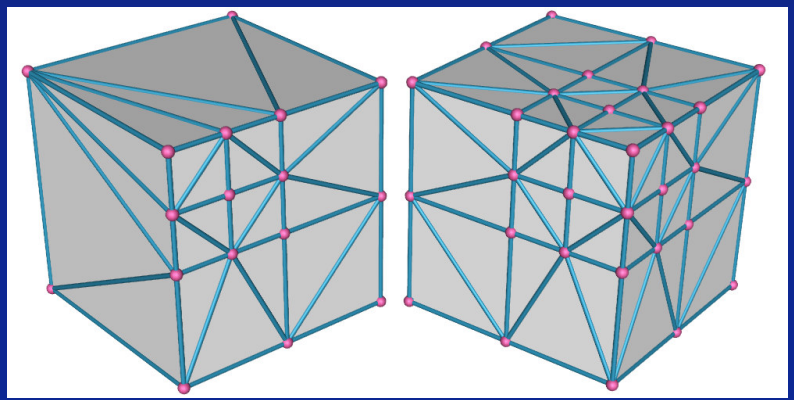


Clipping

- In 2D: implemented with case table

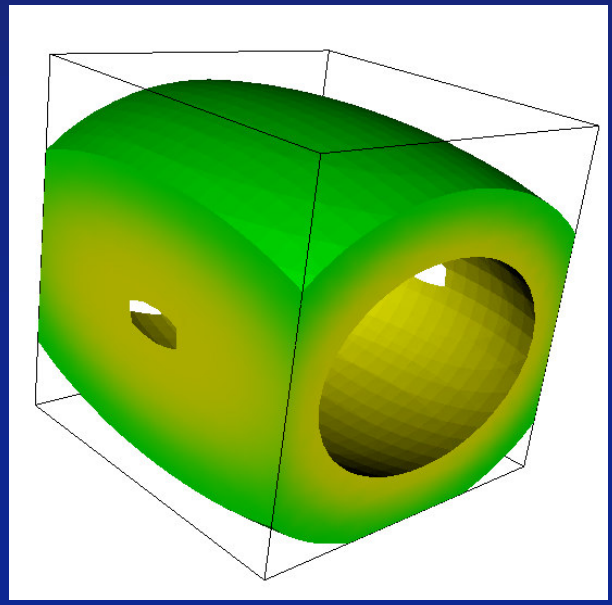


- In 3D: matching face diagonals pose a problem
  - In regular data (e.g. volume), use templates
  - Use ordered triangulator
  - Plug for Thursday morning paper "Compatible Triangulations of Spatial Decompositions "



Clipping: VTK Example (in Tcl)

```
vtkQuadric quadric
  quadric SetCoefficients .5 1 .2 0 .1 0 0 .2 0 0

vtkSampleFunction sample
  sample SetSampleDimensions 20 20 20
  sample SetImplicitFunction quadric
  sample ComputeNormalsOff

vtkClipVolume clip
  clip SetInput [sample GetOutput]
  clip SetValue 1.0
  clip GenerateClippedOutputOff
```



Mesh Operations

- Pragamatic view:
  - Visualization algorithms map data into graphics primitives
  - Primitives are typically represented by polygonal meshes
- Often require further processing
  - Improve appearance
  - Reduce data size
  - Remove noise / extraneous information
  - Highlight information

Decimation / Polygon Reduction

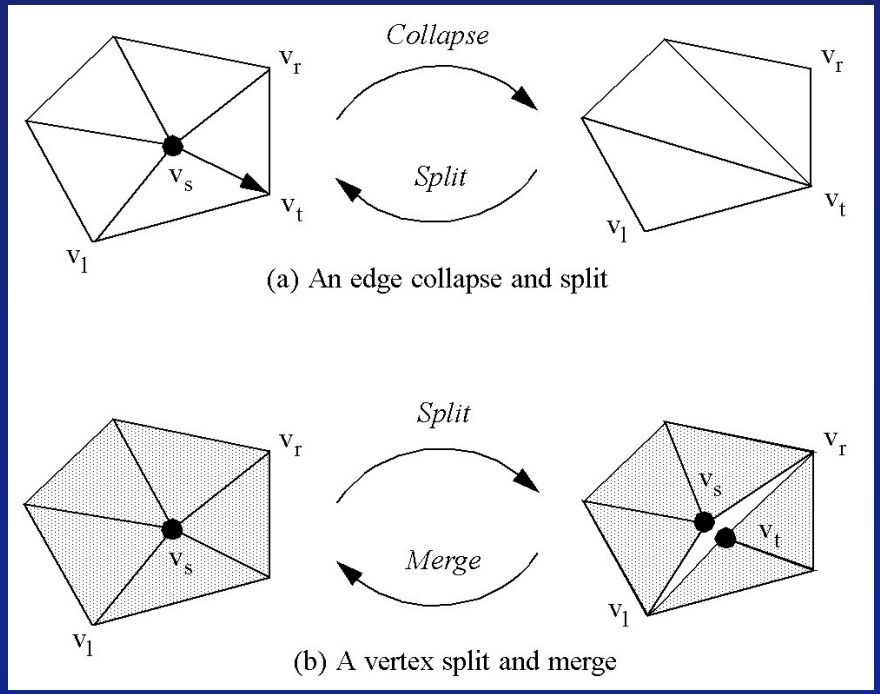
- Polygonal meshes can be large in number of polygons
  - Visualization algorithms: Isocontouring, cutting, implicit modeling
  - Laser digitizers
    (Levoy, Siggraph 2000)

- Rendering performance adversely affected
  - 480 million polygons

- Goal: reduce the number of polygons will retaining model fidelity

Error Metrics

- Different methods of measuring error
  - Object space
  - Image space
- Approaches
  - Vertex deletion followed by retriangulation
  - Edge collapse
  - Triangle deletion followed by retriangulation
  - Point merging (bucketing, distance)
  - Topological modification



VTK Decimation Algorithms

- vtkDecimate – Siggraph ’91 implementation (Schroeder et al, vertex deletion)
- vtkDecimatePro – Variant of Hoppe’s progressive meshes
- vtkQuadricDecimation – Garland and Heckbert’s quadric error measure
- vtkQuadricClustering – Lindstrom point merging and repositioning based on quadric error metric
- vtkGreedyTerrainDecimation – Garland & Heckbert’s top-down, Delaunay triangulation (points with maximum error are introduced first)

Example (in Tcl – general mesh)

```
vtkDecimatePro deci
  deci SetInput [fran GetOutput]
  deci SetTargetReduction .95
  deci PreserveTopologyOn
  deci AccumulateErrorOn
```

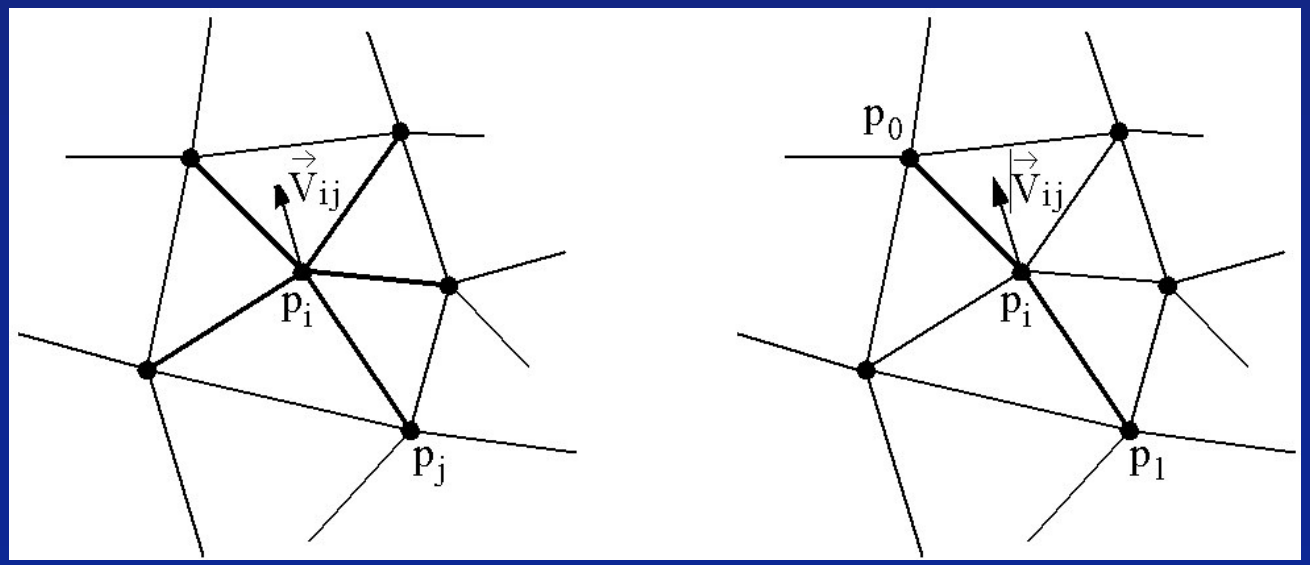
Example (in Tcl – Terrain)

```
vtkGreedyTerrainDecimation deci
  deci SetInput [demReader GetOutput]
  deci BoundaryVertexDeletionOn
  deci SetErrorMeasureToNumberOfTriangles
  deci SetNumberOfTriangles 20000
```

Surface Smoothing

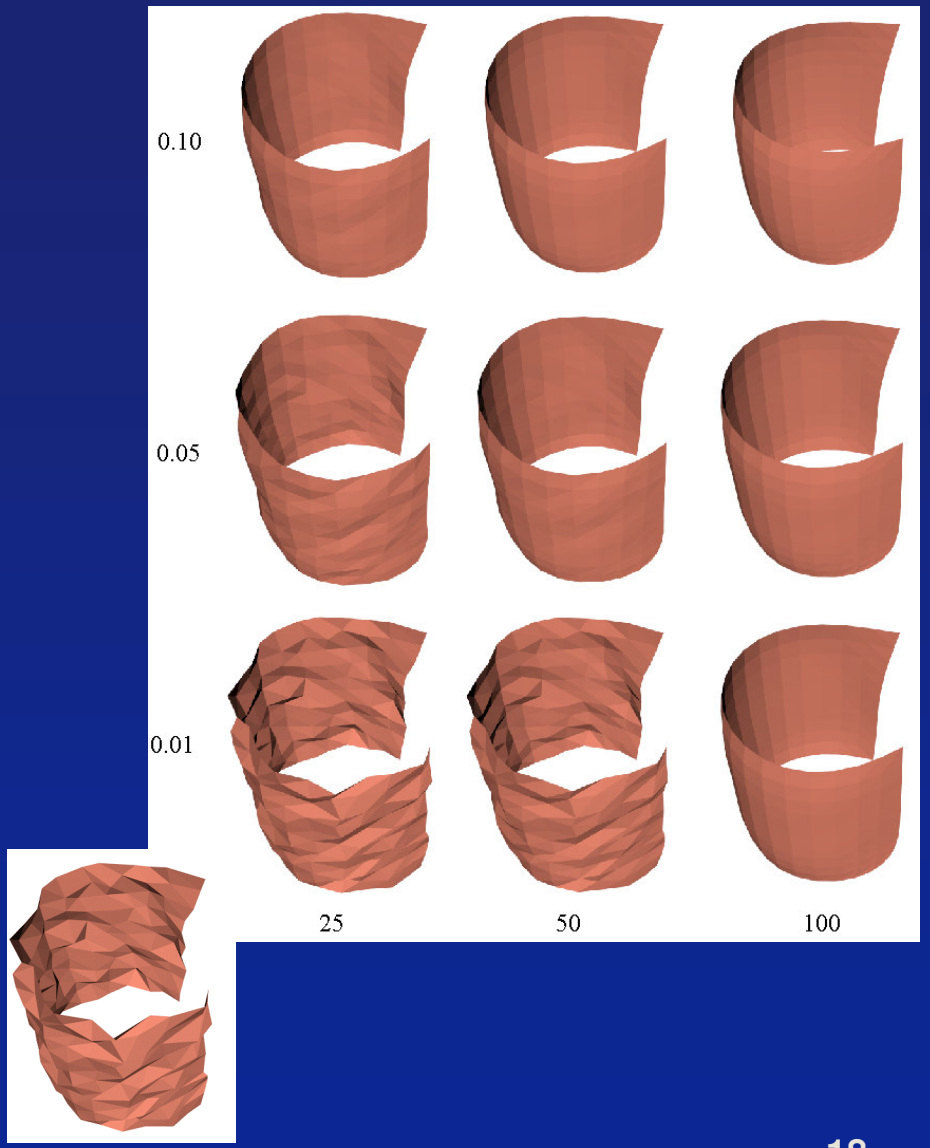
- Reposition mesh vertices to reduce high frequency noise
- Laplacian smoothing described by:
  - Relaxation factor
  - Multiple iterations
  - May be constrained along edges or boundary

$$\vec{x}_{i+1} = \vec{x}_i + \lambda \vec{V}_{ij} = \vec{x}_i + \lambda \sum (\vec{x}_j - \vec{x}_i) \quad \forall j : 0 \le j < n$$



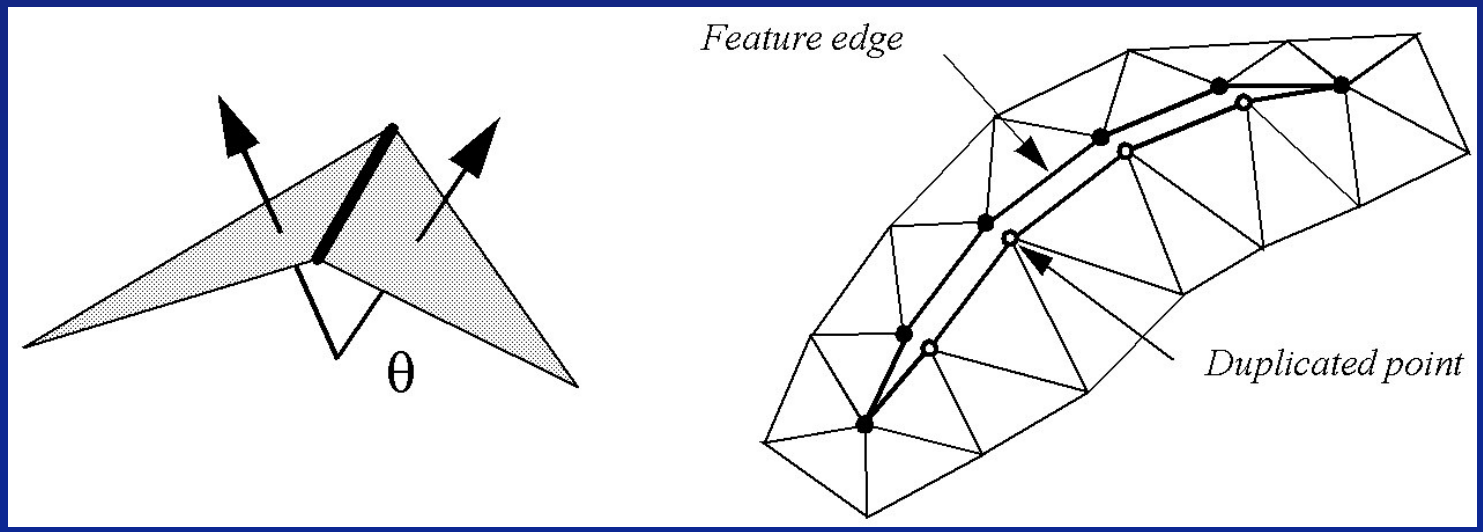
VTK Example (in Tcl)

```
vtkLineSource line
  line SetPoint1 0 1 0
  line SetPoint2 0 1 2
  line SetResolution 10

vtkRotationalExtrusionFilter lineSweeper
  lineSweeper SetResolution 20
  lineSweeper SetInput [line GetOutput]
  lineSweeper SetAngle 270

vtkBrownianPoints bump
  bump SetInput [lineSweeper GetOutput]

vtkWarpVector warp
  warp SetInput [bump GetPolyDataOutput]
  warp SetScaleFactor .2

vtkSmoothPolyDataFilter smooth
    smooth SetInput [warp GetPolyDataOutput]
    smooth SetNumberOfIterations 50
    smooth BoundarySmoothingOn
    smooth SetFeatureAngle 120
    smooth SetEdgeAngle 90
    smooth SetRelaxationFactor .025
```



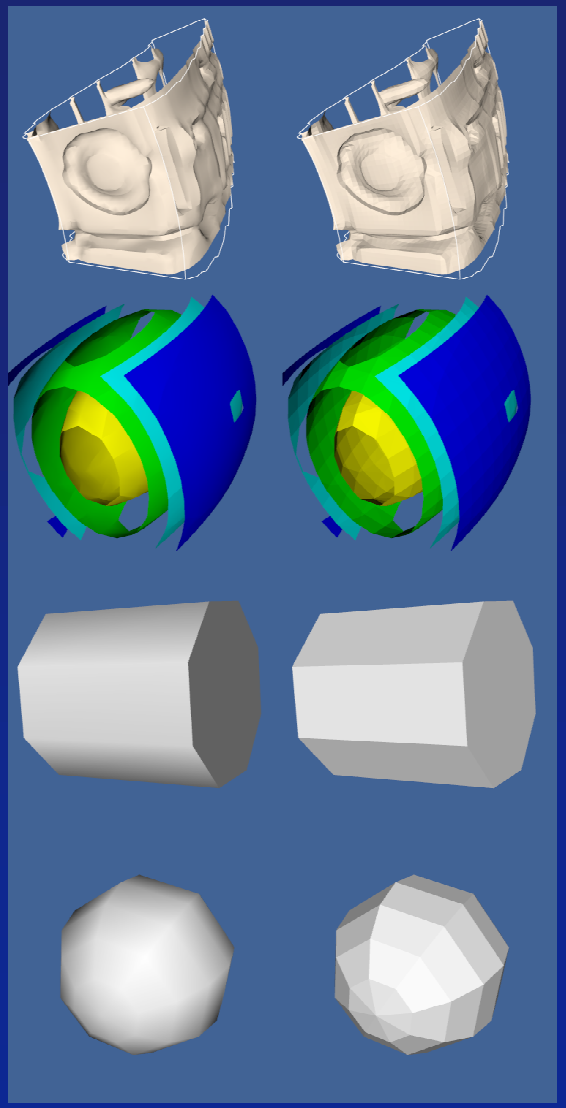
Surface Normal Generation

- Improve the appearance of objects by generating surface normals
  - Flat. Gouraud, Phong shading
- If rendering with a per vertex normal, vertices must be duplicated along sharp edges
- Sharp edges are defined w.r.t. feature angle



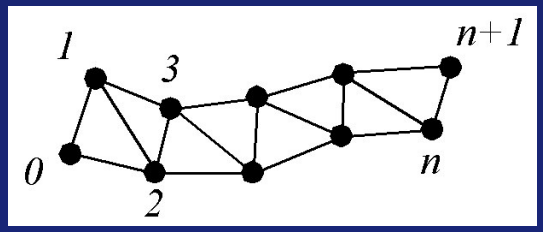
VTK Example (in Tcl)

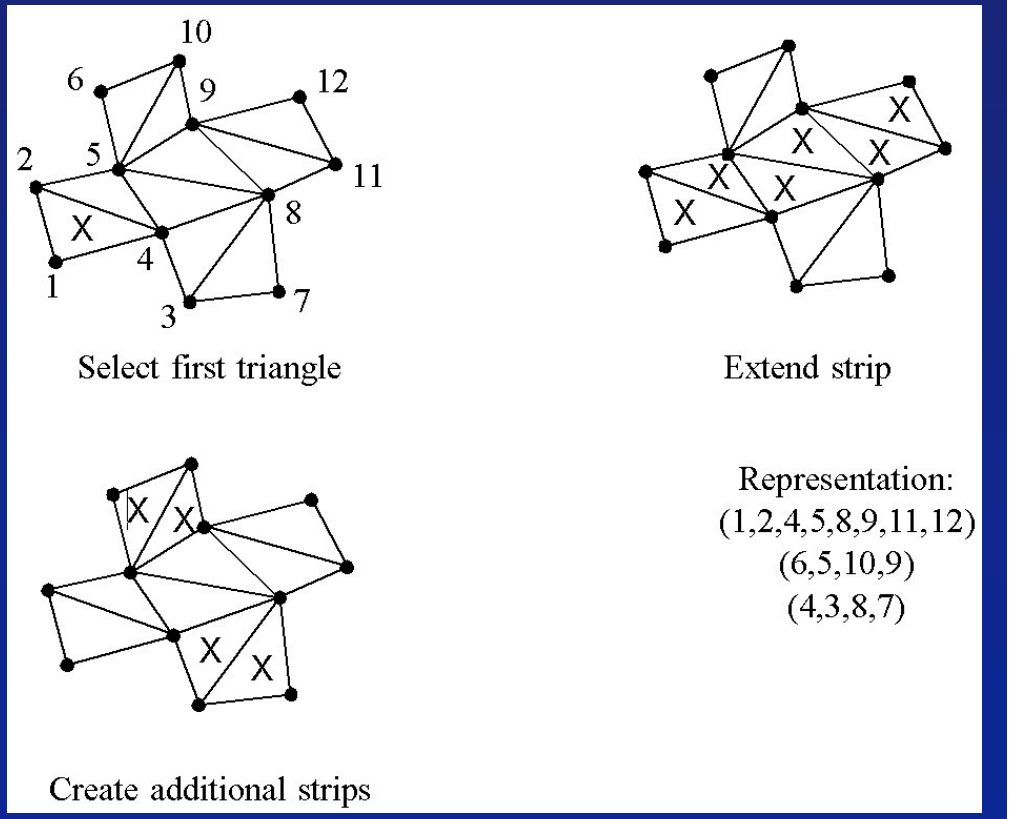
```
vtkDecimatePro deci
    deci SetInput [fran GetOutput]
    deci SetTargetReduction 0.9
    deci PreserveTopologyOn

vtkPolyDataNormals normals
    normals SetInput [fran GetOutput]
    normals FlipNormalsOn

vtkPolyDataMapper franMapper
    franMapper SetInput [normals GetOutput]

vtkActor franActor
    franActor SetMapper franMapper
```



Triangle Strip Generation

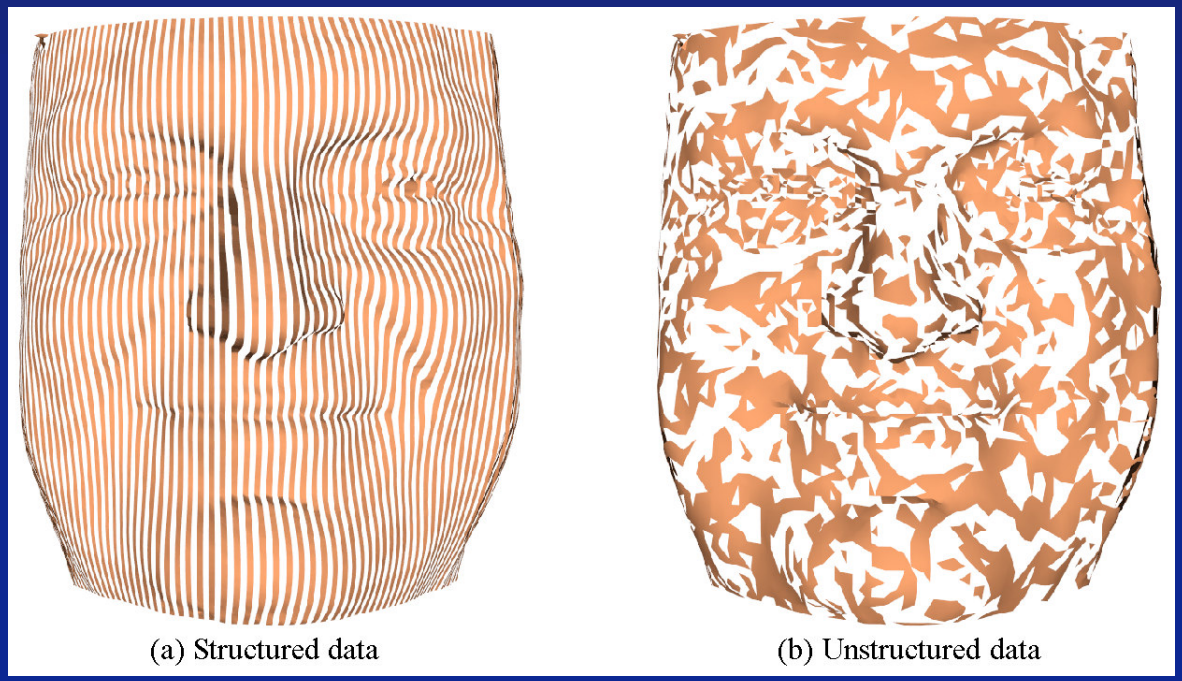
- Polygonal surfaces often consist of large collections of triangles
- Triangle strips are compact representations:
  - n+2 points can represent n triangles
- Graphics hardware renders strips fast
- Triangles may have to be generated from polygons by triangulation routines





Example (in C++)

```
vtkStripper *stripper vtkStripper::New();
    stripper->SetInput( reader->GetOutput() );
vtkMaskPolyData *mask = vtkMaskPolyData::New();
    mask->SetInput( stripper->GetOutput() );
    mask->SetOnRatio(2);
```



(a) Structured data

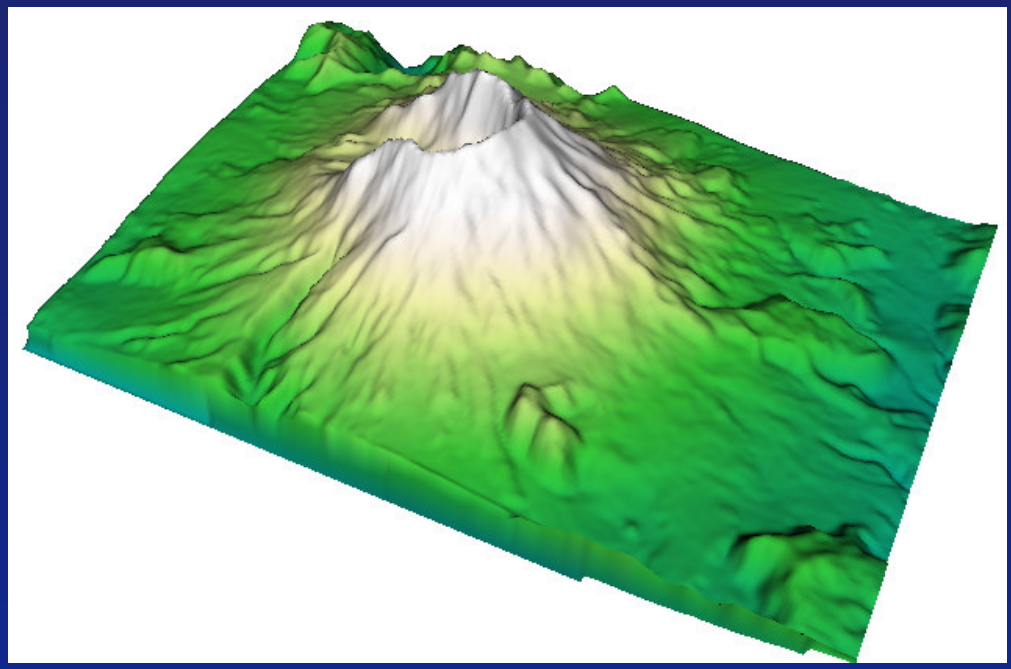
(b) Unstructured data

Terrain

- Terrain is often represented as elevation maps or height fields (i.e., images whose pixel values are height)
- 3D surface is created by using vtkWarpScalar
- Subsampling, decimation, special color maps are options

Example

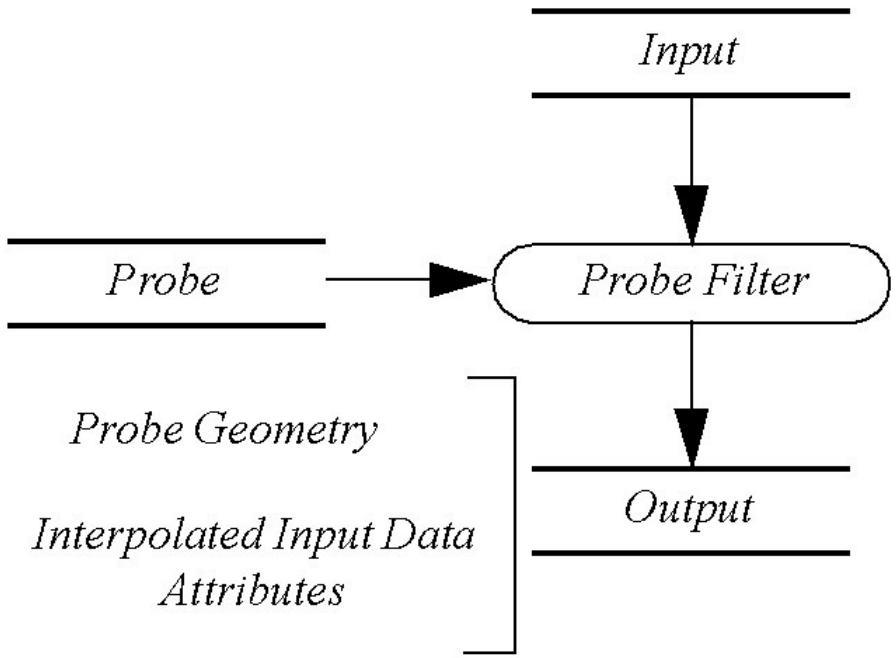
```
vtkImageShrink3D shrink
  shrink SetShrinkFactors   2 2 1
  shrink SetInput [demModel GetOutput]
  shrink AveragingOn

vtkImageDataGeometryFilter geom
  geom SetInput [shrink GetOutput]

vtkWarpScalar warp
  warp SetInput [geom GetOutput]
  warp SetNormal 0 0 1
  warp UseNormalOn
  warp SetScaleFactor $Scale

vtkElevationFilter elevation
  elevation SetInput [warp GetOutput]
  elevation SetLowPoint 0 0 $lo
  elevation SetHighPoint 0 0 $hi
  eval elevation SetScalarRange $lo $hi

vtkPolyDataNormals normals
  normals SetInput [elevation GetPolyDataOutput]
  normals SetFeatureAngle 60
  normals ConsistencyOff
  normals SplittingOff
```



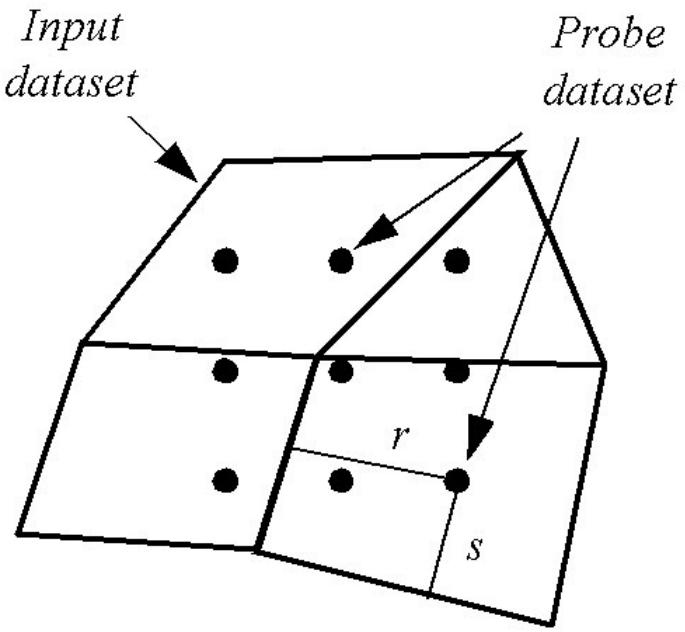
Probing

- Sample one dataset with another dataset
- Uses:
  - Obtain a value at a point
  - Plot along a line or curve
  - Transform one data form into another
  - Reduce the size of data
- Caveats
  - Probe resolution can be too high (false sense of security) or too low (lose important details)

Probing (cont.)



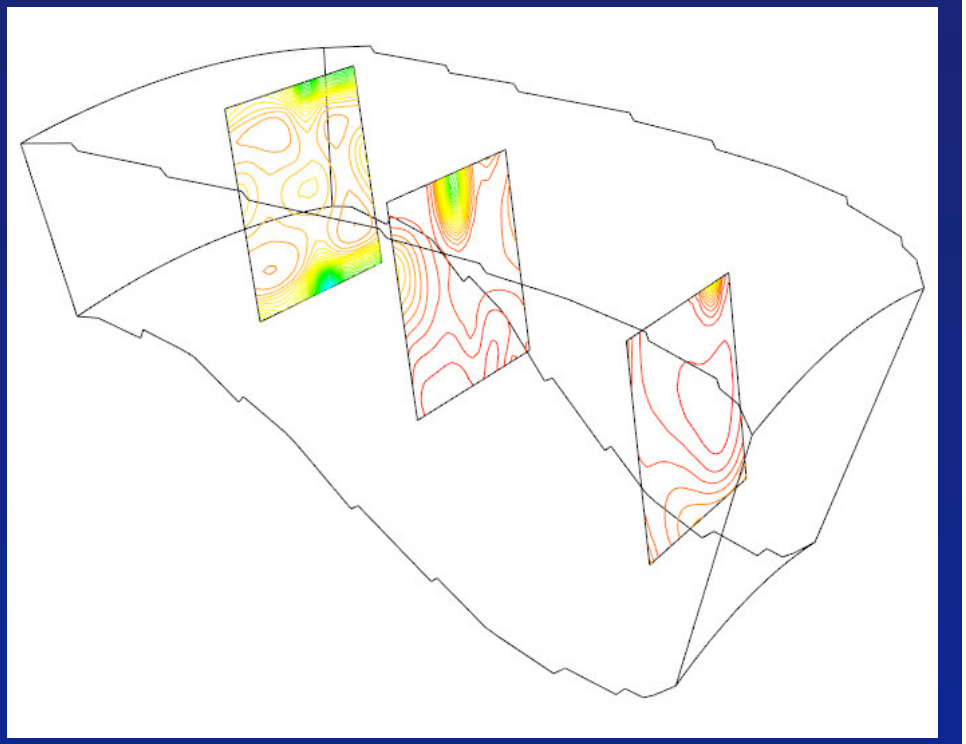
a) Probing process



b) Probe interpolation

Example (in Tcl)

```
vtkPlaneSource plane
   plane SetResolution 50 50
vtkTransform transP1
   transP1 Translate 3.7 0.0 28.37
   transP1 Scale 5 5 5
   transP1 RotateY 90
vtkTransformPolyDataFilter tpd1
   tpd1 SetInput [plane GetOutput]
   tpd1 SetTransform transP1
vtkOutlineFilter outTpd1
   outTpd1 SetInput [tpd1 GetOutput]

vtkAppendPolyData appendF
   appendF AddInput [tpd1 GetOutput]
   appendF AddInput [tpd2 GetOutput]
   appendF AddInput [tpd3 GetOutput]
vtkProbeFilter probe
   probe SetInput [appendF GetOutput]
   probe SetSource [pl3d GetOutput]
vtkContourFilter contour
   contour SetInput [probe GetOutput]
   eval contour GenerateValues 50 [[pl3d GetOutput] GetScalarRange]
```

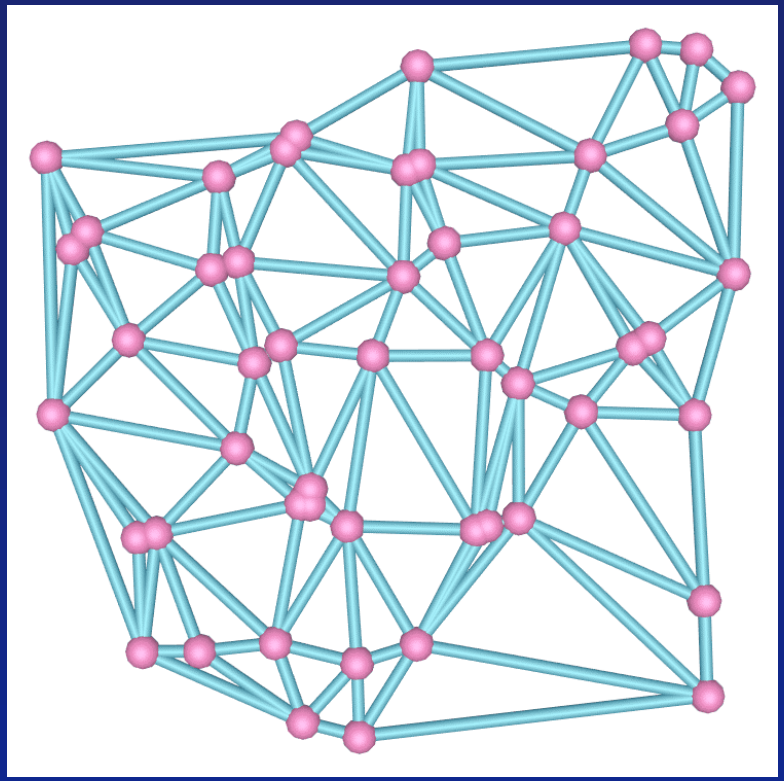


Other Techniques

- Tubing – wrap tubes around lines
  - Vary radius according to scalar value
  - Rotate tube with line normals
- Glyphing
  - Copy object to each point in input
  - Orient according to vector
  - Scale according to scalar value / vector magnitude
- Extrusion
  - Define generating profile
  - Linear or rotational extrusion

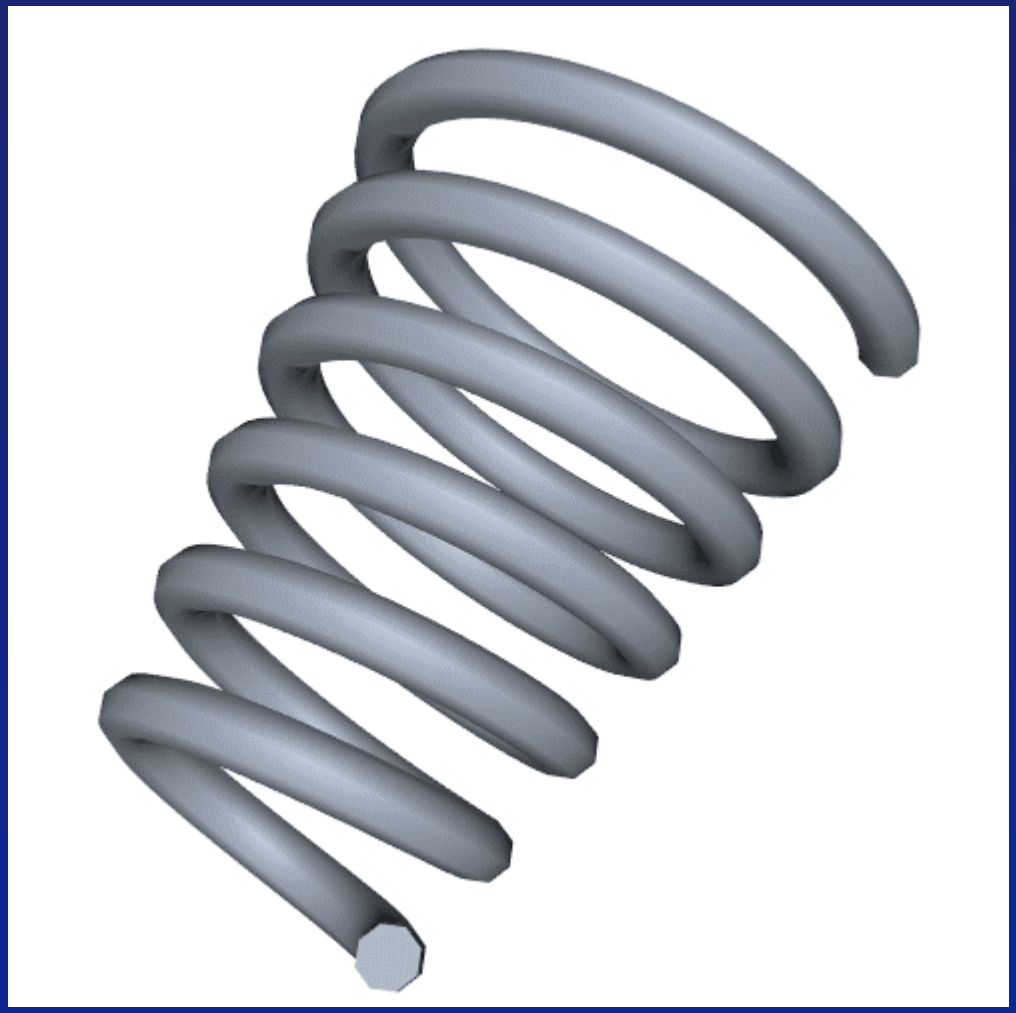
Example: Tubing & Glyphing

```
vtkDelaunay2D del
    del SetInput profile
    del SetTolerance 0.001

vtkExtractEdges extract
    extract SetInput [del GetOutput]
vtkTubeFilter tubes
    tubes SetInput [extract GetOutput]
    tubes SetRadius 0.01
    tubes SetNumberOfSides 6

vtkSphereSource ball
    ball SetRadius 0.025
    ball SetThetaResolution 12
    ball SetPhiResolution 12
vtkGlyph3D balls
    balls SetInput [del GetOutput]
    balls SetSource [ball GetOutput]
```



Example: Extrusion

```
vtkPoints points
    points InsertPoint 0 1.0 0.0 0.0
    points InsertPoint 1 1.0732 0.0 -0.1768
    points InsertPoint 2 1.25 0.0 -0.25
    ….

vtkCellArray poly
    poly InsertNextCell 8;#number of points
    poly InsertCellPoint 0
    poly InsertCellPoint 1
    …..

vtkPolyData profile
    profile SetPoints points
    profile SetPolys poly

vtkRotationalExtrusionFilter extrude
    extrude SetInput profile
    extrude SetResolution 360
    extrude SetTranslation 6
    extrude SetDeltaRadius 1.0
    extrude SetAngle 2160.0;#six revolutions

vtkPolyDataNormals normals
    normals SetInput [extrude GetOutput]
    normals SetFeatureAngle 60
```



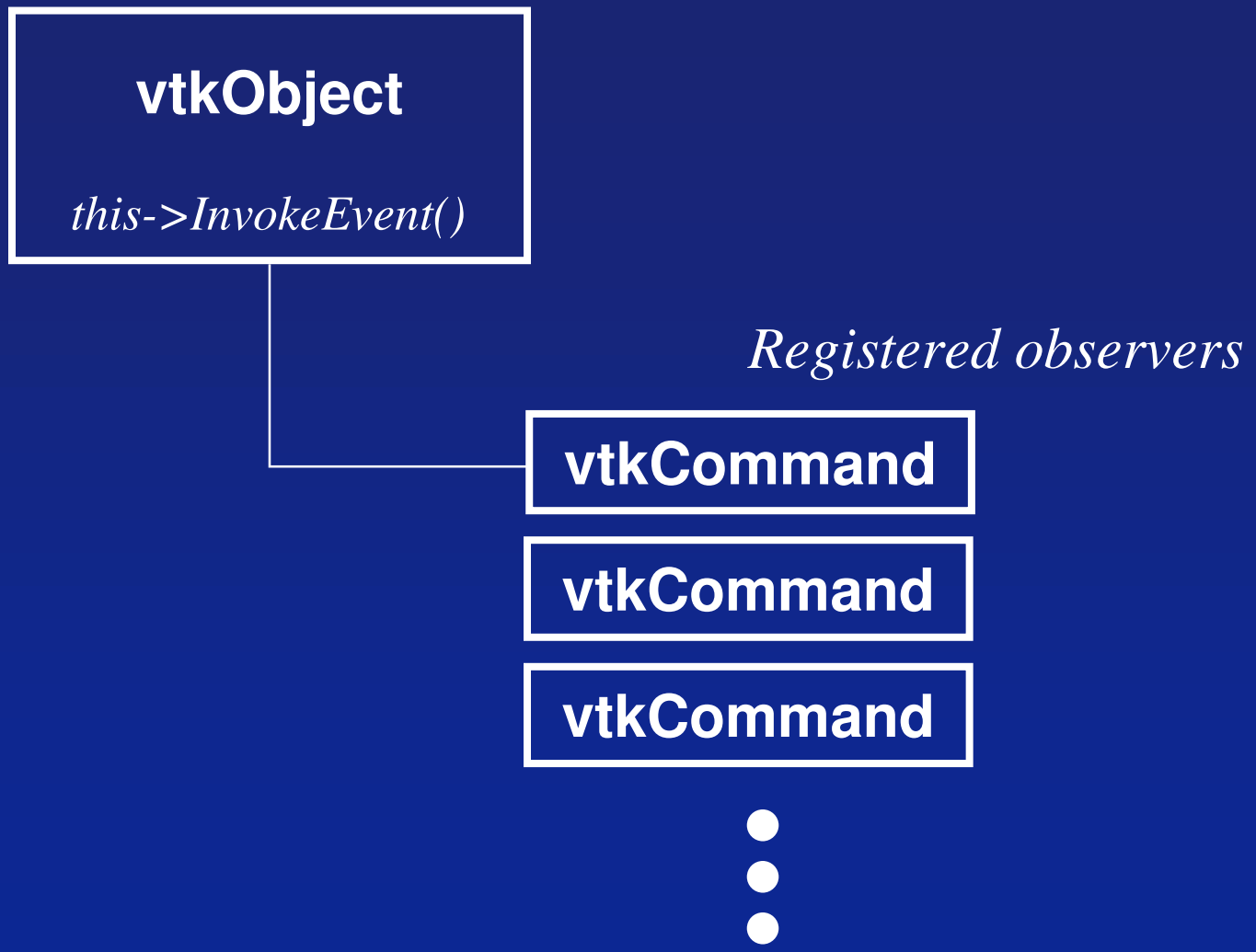
3D Interaction

- Interaction with data is the key to effective visualization
  - The user is in the loop
  - Think of visualization systems as providing probing instruments
- Event Handling
  - Translates user interaction events into actions
  - Command/Observer design pattern
- 3D Widgets
  - Define complex sets of interactions with visual display
  - Think of different types of widgets as “probes” into data

Command / Observer Event Handling

- *Observers* watch for particular event invocations on a particular instance
- When an observer sees the event it is interested in, it invokes an associated *Command*
- In VTK:
  - Register interest in an event; associate a command with the event
    ```
    renWin->AddObserver(unsigned long eventId, vtkCommand*);
    ```
  - Instances invoke an event on themselves:
    ```
    this->InvokeEvent(vtkCommand::ProgressEvent,NULL);
    ```

Command / Observer Example



Event Types (Some examples)

- Filter execution
  - Start, End, Progress events
- Rendering
  - Start, End
  - ResetCamera, ResetCamerClippingRange
- Picking
  - StartPick, EndPick, Pick events
- Mouse
  - MouseMove, MouseWheelForward, MouseWheelBackward
  - KeyPress, KeyRelease
- 3D Widgets
  - StartInteraction, EndInteraction, Interaction

VTK Example: Picking Callback (in C++)


```
class vtkMyCommand : public vtkCommand
{
public:
  static vtkMyCommand* New() { return new vtkMyCommand; }
  virtual void Execute(vtkObject *caller, unsigned long eventId,
                       void *callData)
    {vtkCellPicker *picker = vtkCellPicker::SafeDownCast(caller);
      cerr << "Picked cell id " << picker->GetCellId() << endl;  }
};

main ()
{
  vtkMyCommand *cmd = vtkMyCommand::New();
  vtkCellPicker *picker = vtkCellPicker::New();
  picker->AddObserver(vtkCommand::EndPickEvent,cmd);

  vtkRenderer *aren    = vtkRenderer::New();
  vtkRenderWindow *renWin  = vtkRenderWindow::New();
  renWin->AddRenderer(aren);
  iren->SetPicker(picker);
```

VTK Example: Progress Callback (in C++)


```
class vtkProgressCommand : public vtkCommand
{
public:
  static vtkProgressCommand *New() { return new vtkProgressCommand; }
  virtual void Execute(vtkObject *caller, unsigned long, void *callData)
   { double progress = *(static_cast<double*>(callData));
         std::cout << "Progress at " << progress<< std::endl; }
};

vtkCommand* sobserver = vtkStartCommand::New();
vtkCommand* eobserver = vtkEndCommand::New();
vtkCommand* pobserver = vtkProgressCommand::New();

vtkDecimatePro *deci = vtkDecimatePro::New();
deci->SetInput( byu->GetOutput() );
deci->SetTargetReduction( 0.75 );
deci->AddObserver( vtkCommand::StartEvent, sobserver );
deci->AddObserver( vtkCommand::EndEvent, eobserver );
deci->AddObserver( vtkCommand::ProgressEvent, pobserver );
```

Interaction Styles / 3D Widgets

- 3D Widgets typically consist of
  - Visual representation
  - Complex set of event definitions
- Interaction styles
  - Have no visual representation
  - Typically used to control cameras and actors

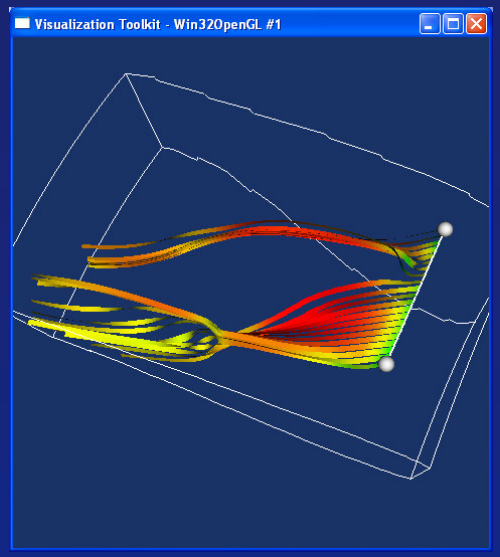
Interaction Styles

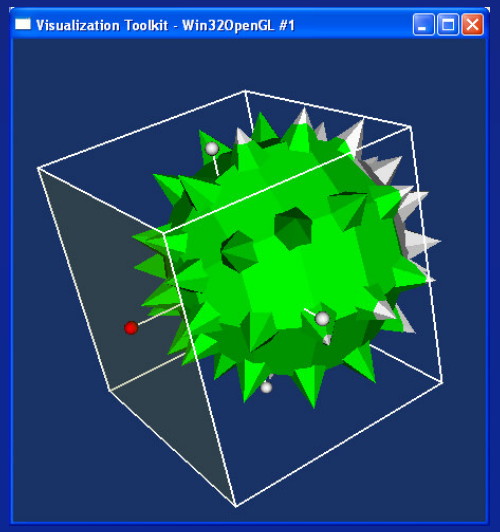
- Styles are associated with vtkRenderWindowInteractor
  - vtkInteractorObserver observes events in instances of vtkRenderWindowInteractor
- Example Styles (subclasses of vtkInteractorObserver):
  - vtkInteractorStyleJoystickCamera (also trackball)
  - vtkInteractorStyleJoystickCamera (also trackball)
  - vtkInteractorStyleFlight
  - vtkInteractorStyleTerrain
  - vtkInteractorStyleImage
- Example Usage (C++)

```
vtkRenderWindowInteractor *iren = vtkRenderWindowInteractor::New();
vtkInteractorStyleFlight *style = vtkInteractorStyleFlight::New();
iren->SetInteractorStyle(flight);
```

3D Widgets

- Some of the variety of widgets found in VTK
  - vtkPointWidget
  - vtkLineWidget
  - vtkPlaneWidget
  - vtkImplicitPlaneWidget
  - vtkBoxWidget
  - vtkSphereWidget
  - vtkScalarBarWidget
  - vtkImagePlaneWidget
  - vtkSplineWidget
- Often provide auxiliary functionality
  - Transformation
  - Output data (e.g., vtkPolyData)
  - Implicit functions





Using Widgets in VTK: Define Command (C++)

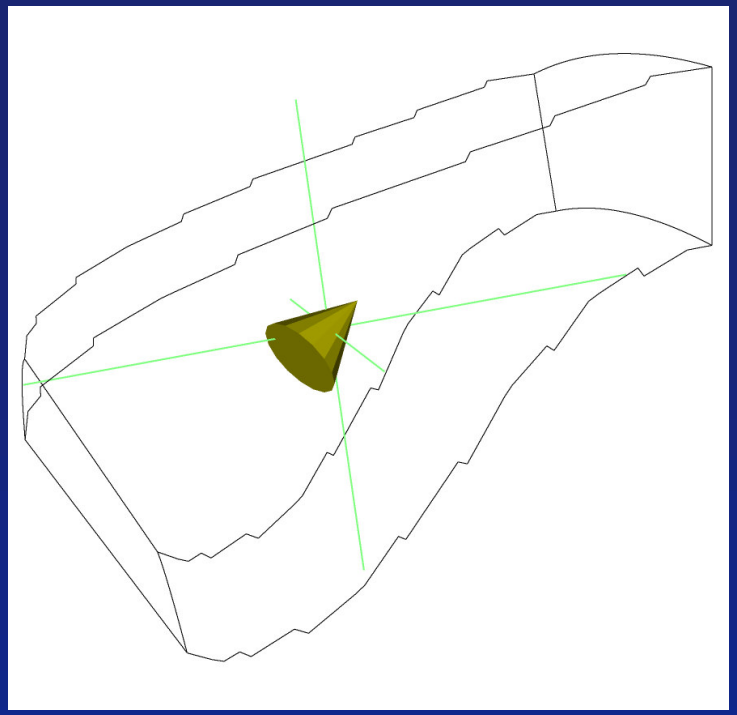

```
class vtkMyCallback : public vtkCommand
{
public:
  static vtkMyCallback *New()   { return new vtkMyCallback; }
  virtual void Execute(vtkObject *caller, unsigned long, void*)
    {
      vtkBoxWidget *boxWidget = reinterpret_cast<vtkBoxWidget*>(caller);
      boxWidget->GetTransform(this->Transform);
      this->Actor->SetUserTransform(this->Transform);
    }
  vtkMyCallback():Transform(0),Actor(0) {}
  vtkTransform *Transform;
  vtkActor     *Actor;
};
```

Using Widgets in VTK: Usage

```
  int main( int argc, char *argv[] )
{
…..
  vtkMyCallback *myCallback = vtkMyCallback::New();
  myCallback->Transform = t;
  myCallback->Actor = maceActor;

  vtkBoxWidget *boxWidget = vtkBoxWidget::New();
  boxWidget->SetInteractor( iren );
  boxWidget->SetPlaceFactor( 1.25 );
  boxWidget->AddObserver(
                    vtkCommand::InteractionEvent,myCallback);
…..
}
```

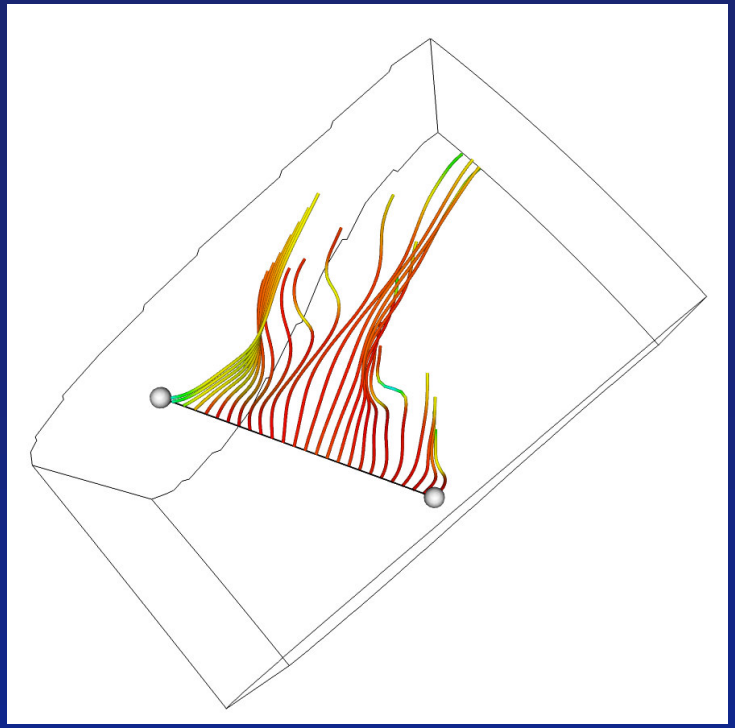
vtkPointWidget

- Position point in space
- 3D cursor, bounding box, cursor shadows
- Produces output vtkPolyData



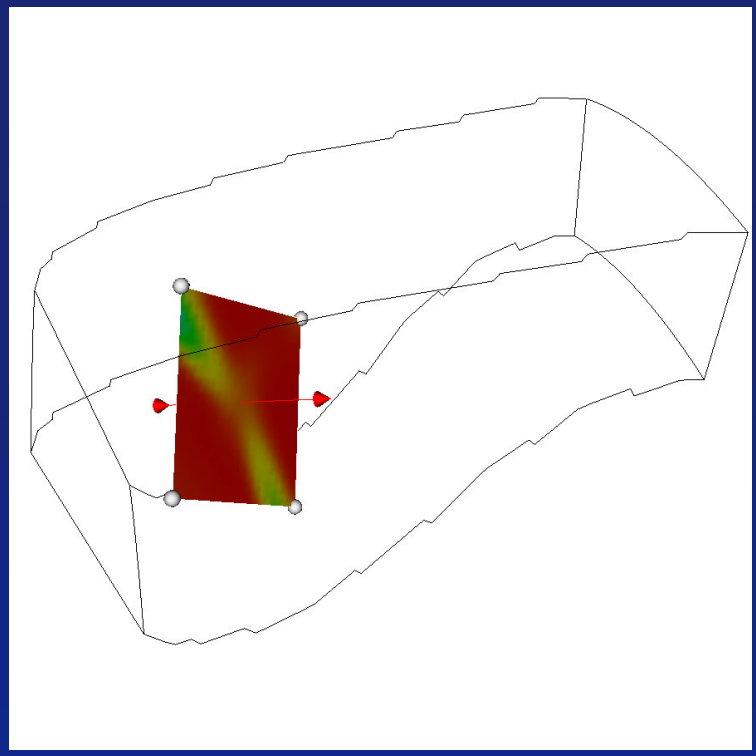
vtkLineWidget

- Position polyline in space (vtkLineSource)
- Line produces interior points (good for rakes)
- Two end points represented by spheres
- Produces vtkPolyData



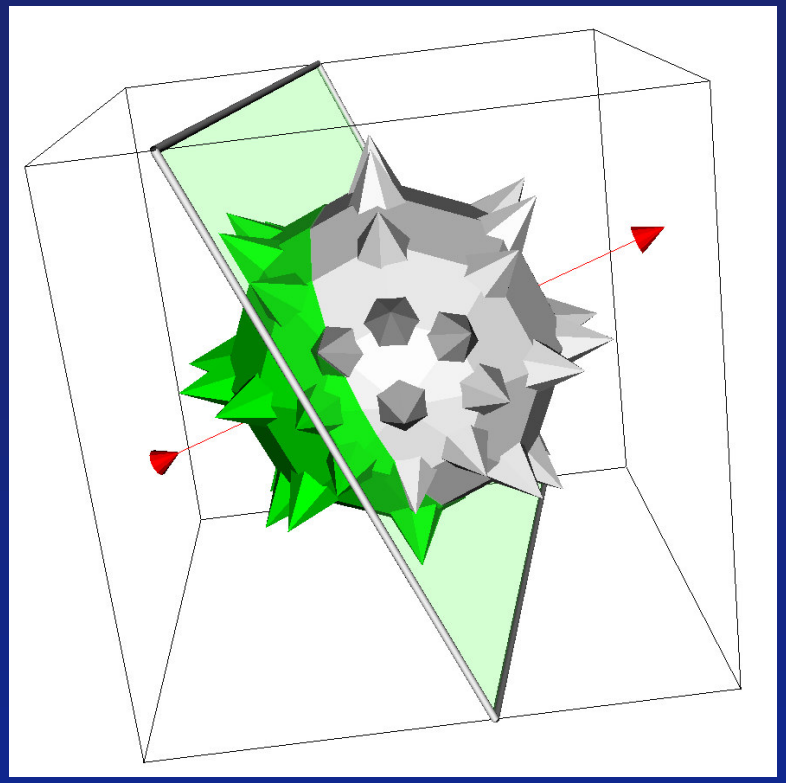
vtkPlaneWidget

- Position finite (bounded) plane
- Corner control points plus orientation vector
- Produces
  - vtkPolyData (at specified resolution)
  - vtkPlane implicit function



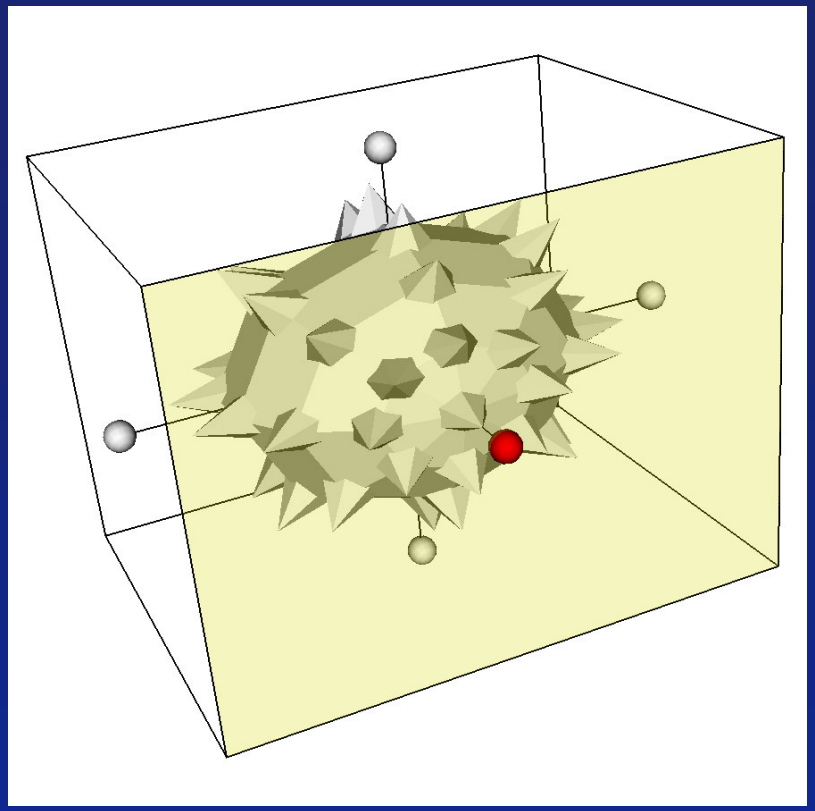
vtkImplicitPlaneWidget

- Position infinite (unbounded) plane
- Plane clipped by bounding box and orientation vector
- Produces
  - vtkPolyData (clipped polygon)
  - vtkPlane implicit function



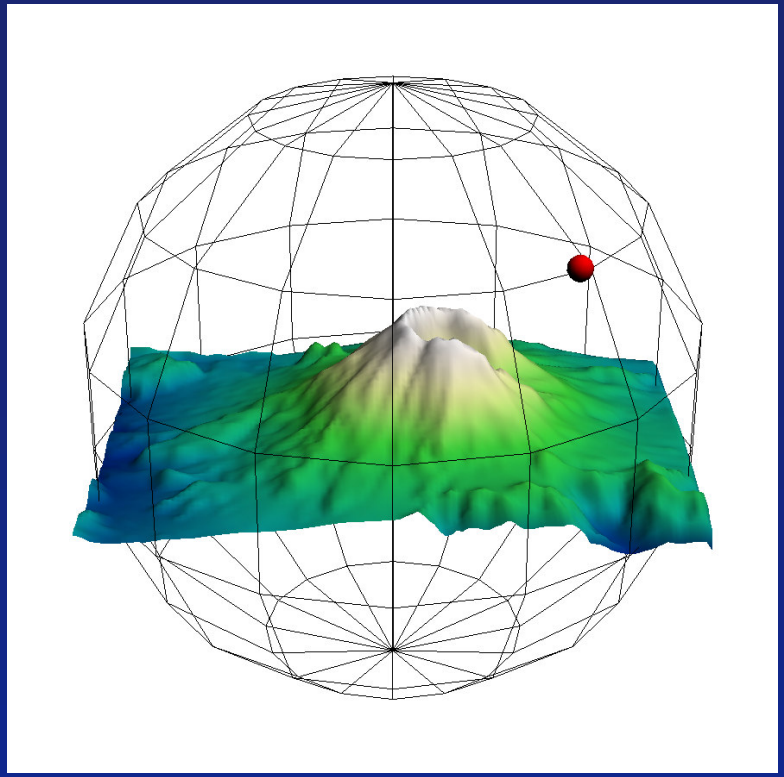
vtkBoxWidget

- Translate, scale, and orient a box
- Transparent box, face translation handles, axes
- Produces
  - vtkPolyData (box)
  - vtkPlanes implicit function
  - vtkTransform transformation matrix



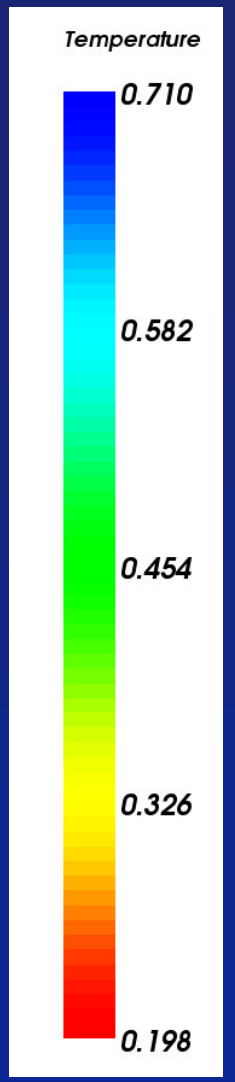
vtkSphereWidget

- Position a point constrained to the surface of a sphere
- Sphere plus position handle
- Produces
  - Center, radius of sphere
  - vtkPolyData (sphere surface)
  - vtkSphere implicit funtion



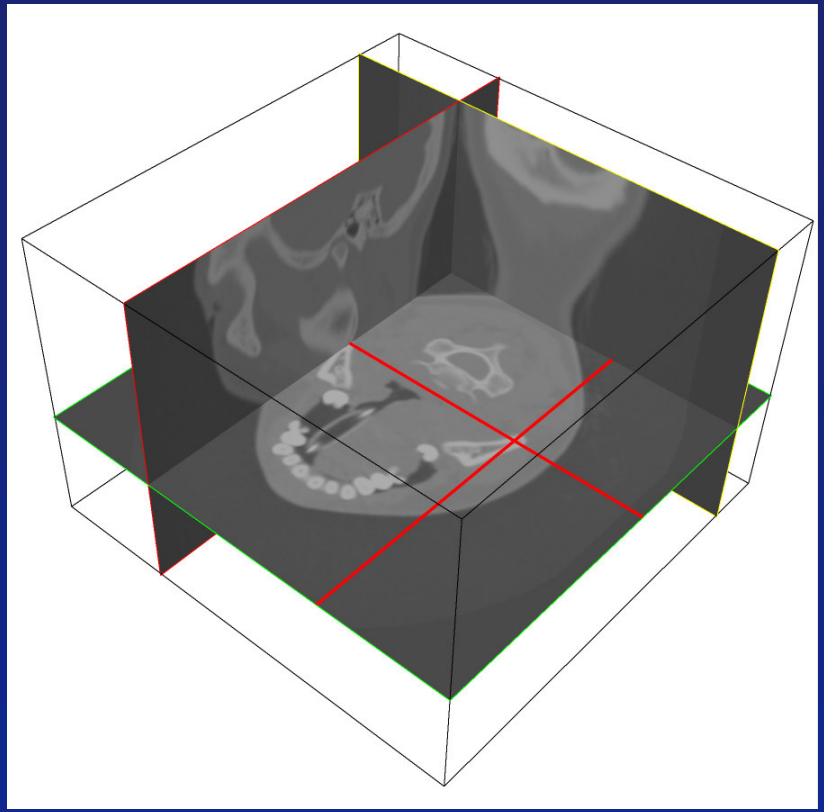
vtkScalarBarWidget

- Annotated lookup table
- Scalar bar, title, annotation along bar
- Dynamic placement (reorients depending on where it is in rendering window)
- Controls vtkScalarBarActor



vtkImagePlaneWidget

- Visualize volume on three orthogonal image planes; probe data values; orient arbitrary plane
- Outline, three axes-aligned planes, reslice plane, plus cursor jacks on plane surface
- Performs image reslice (resample)
- Produces
  - vtkPolyData (reslice plane)
  - vtkImageData (reslice data)



vtkSplineWidget

- Control interpolating spline
- Several handles plus spline
- Can be clamped to plane
- Produces
  - vtkPolyData (spline)