An Introduction to Visualization Using VTK

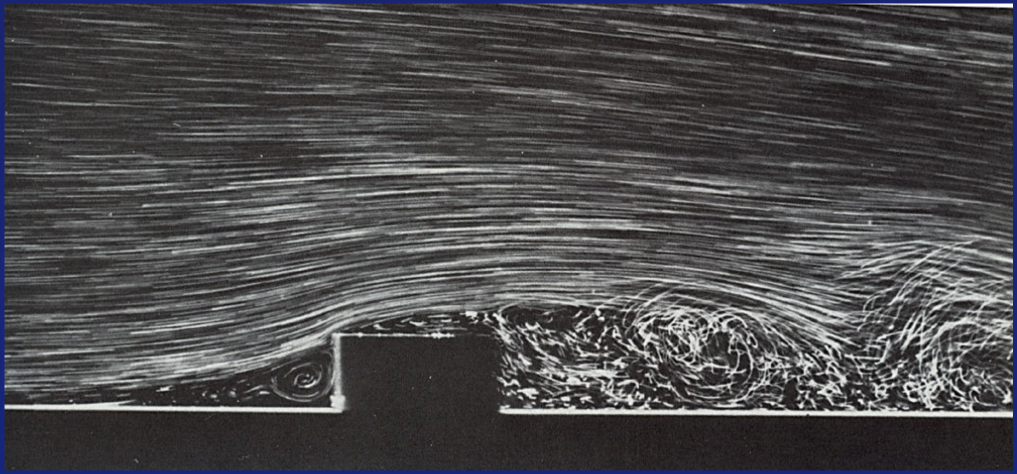
Vector Visualization

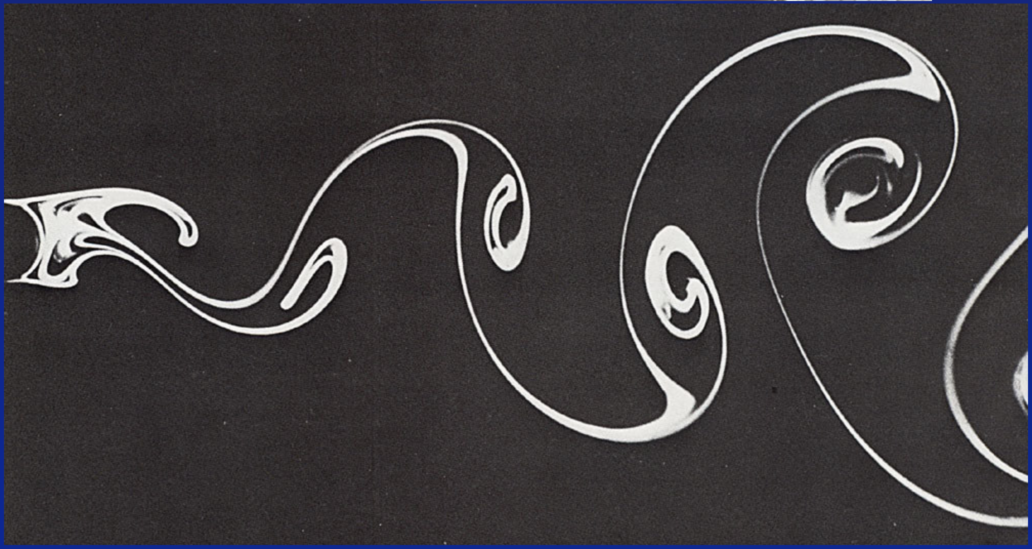
Berk Geveci
Kitware Inc.

Vector Visualization - Outline

- Key Concepts – Vector Calculus
- Generating and Processing Vector Data
- Extracting Geometry
- Hedgehogs and Glyphs
- Warping

Visualizing Vector Fields

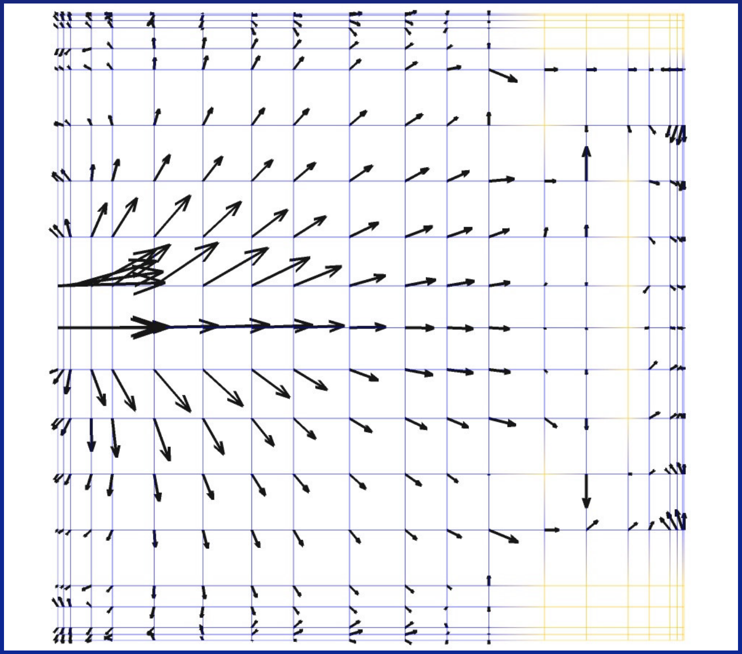




B&W photographs from An Album of Fluid Motion, Milton Van Dyke

Key Concepts – Vector Fields

- A vector represents a physical quantity that possesses magnitude described by a positive real number, and direction.
- A vector field is the ensemble of points in a region together with the corresponding vector point function.



VTK Fields

- vtkDataObject represents a “blob” of data
  - contain instance of vtkFieldData
  - an array of arrays
  - no geometric/topological structure
  - Superclass of all VTK data objects

- vtkDataSet has geometric/topological structure
  - Consists of geometry (points) and topology (cells)
  - Has associated “attribute data” (e.g., scalars, vectors, tensors, normals, tcoords) as well as field data

VTK Fields - Continued

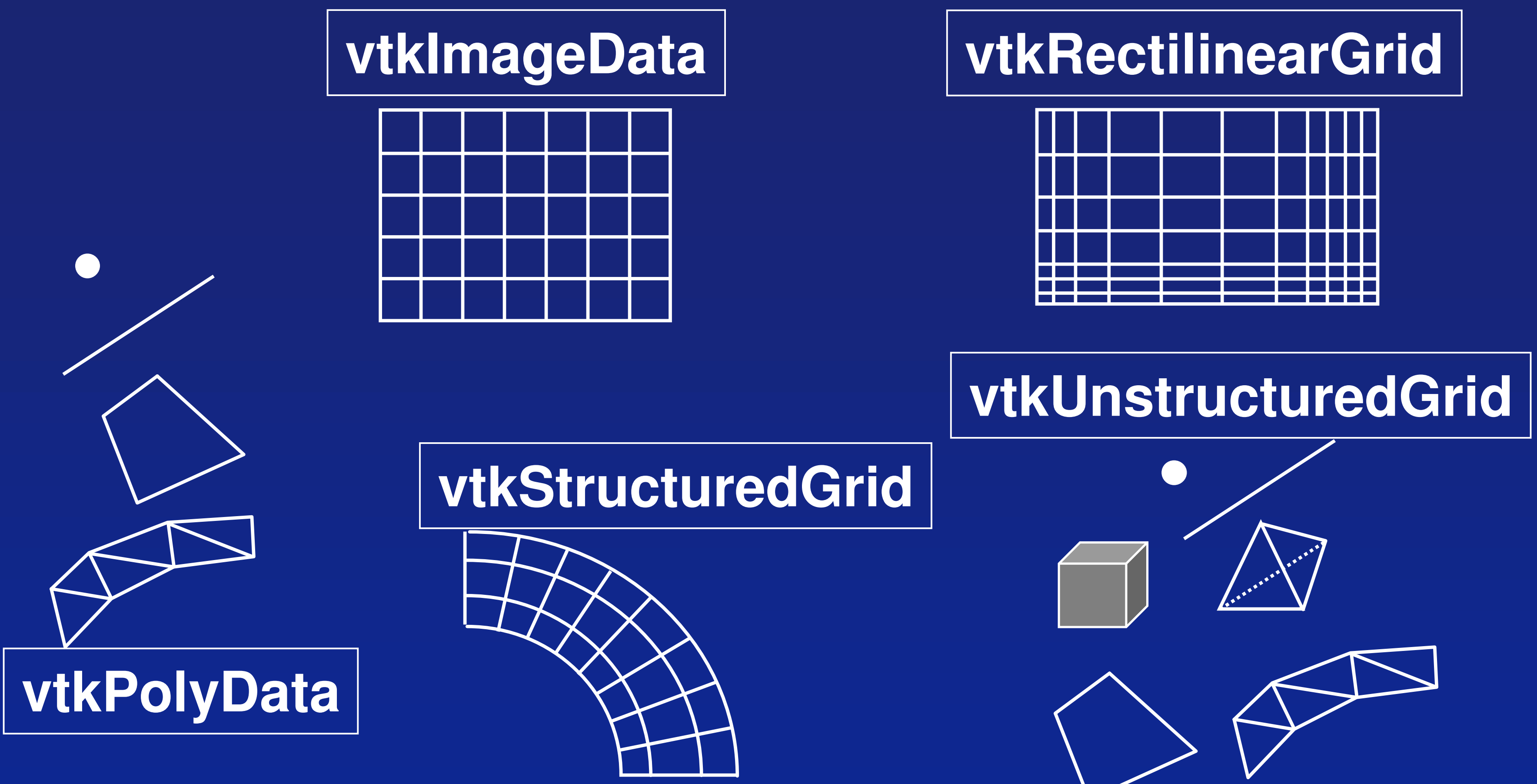
vtkImageData

vtkRectilinearGrid

vtkUnstructuredGrid

vtkStructuredGrid

vtkPolyData



VTK Fields - Continued

- **vtkDataSet also has point and cell attribute data:**
  - **Scalars** (1-4 components)
  - **Vectors** - 3-vector (3 components)
  - **Tensors** - 3x3 symmetric matrix (9 components)
  - **Normals** - unit vector (3 components)
  - **Texture Coordinates** 1-3D (1-3 components)
  - Array of arrays (I.e. FieldData) (m components)
- Attributes and other field data are represented by vtkDataArray.
- vtkDataArray is an array of n tuples. Each tuple has m components.

vtkAssignAttribute

- void Assign(const char* fieldName, int attributeType, int attributeLoc)

attributeType = {SCALARS, VECTORS, NORMALS, TCOORDS, TENSORS}

attributeLoc = {POINT_DATA, CELL_DATA}

```
vtkAssignAttribute* aa = vtkAssignAttribute::New();
 aa->SetInput(acalc->GetOutput());
 aa->Assign("displacement",
        vtkDataSetAttributes::VECTORS,
        vtkAssignAttribute::POINT_DATA);
```

Importing and Generating Data

- Load a file using a VTK reader
  - Native VTK formats
  - PLOT3D
  - EnSight
  - …
- Use a VTK source
  - vtkSphereSource
  - vtkCylinderSource
- Apply a filter
  - vtkBrownianPoints
  - vtkPolyDataNormals
  - vtkImageGradient
  - vtkArrayCalculator

Example – Using PLOT3D Reader

- void SetXYZFileName(const char* name)
- void SetQFileName(const char* name)
- void SetScalarFunctionNumber(int num)
- void SetVectorFunctionNumber(int num)
- void AddFunction(int num)
  - 200 – velocity
  - 201 – vorticity
  - 202 – momentum
  - 210 – pressure gradient

Example - Continued

```
vtkPLOT3DReader* reader = vtkPLOT3DReader::New();
 reader->SetXYZFileName(xyzname.str());
 reader->SetQFileName(qname.str());
 // The following will load/generate these arrays:
 // density (load), momentum (load),
 // stagnation energy (load) velocity (compute,
 // needed for vorticity), vorticity (compute)
 reader->SetScalarFunctionNumber(100); // density
 reader->SetVectorFunctionNumber(202); // momentum
 reader->AddFunction(201); // vorticity
```

Vector Algebra

$$\mathbf{u} = (u_x, u_y, u_z)$$

$$\mathbf{u} + \mathbf{v} = (u_x + v_x, u_y + v_y, u_z + v_z)$$ addition

$$|\mathbf{u}| = (u_x^2 + u_y^2 + u_z^2)^{1/2}$$ magnitude

$$\mathbf{u_n} = (u_x / |\mathbf{u}|, u_y / |\mathbf{u}|, u_z / |\mathbf{u}|)$$ normalization

$$\mathbf{u} \cdot \mathbf{v} = u_x v_x + u_y v_y + u_z v_z$$ dot product

$$\mathbf{u} \times \mathbf{v} = \begin{vmatrix} \hat{\mathbf{i}} & \hat{\mathbf{j}} & \hat{\mathbf{k}} \\ u_x & u_y & u_z \\ v_x & v_y & v_z \end{vmatrix}$$ cross product

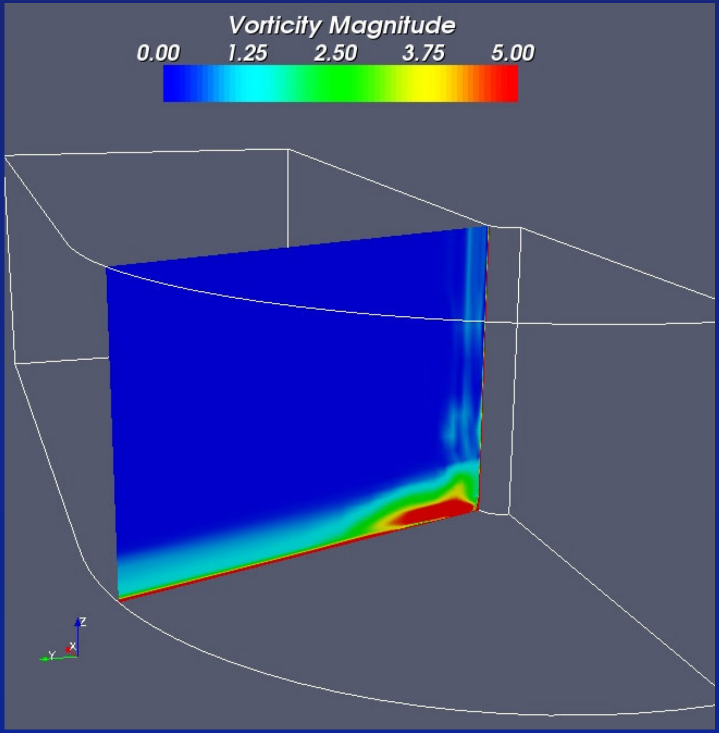
Array Calculator

- void AddScalarArrayName(const char* name, int comp)
- void AddVectorArrayName(const char* name,
  int comp0, int comp1, int comp2)
- void SetAttributeModeToUsePointData()
- void SetAttributeModeToUseCellData()
- void SetResultArrayName(const char* name)
- void SetFunction(const char* function)

- Some of supported operations are:
  - +, -, *, / ,^ , . (dot product), mag (magnitude), norm (normalize)
- Constants:
  - iHat, jHat, kHat

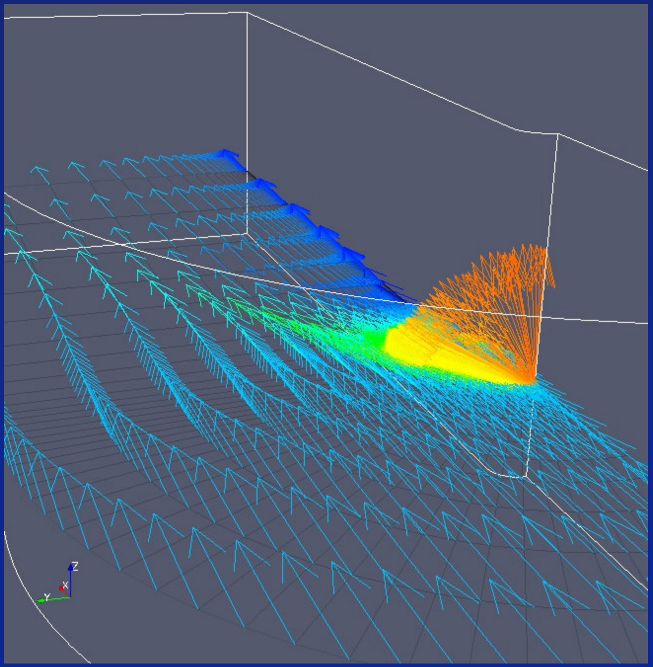
Array Calculator - Examples

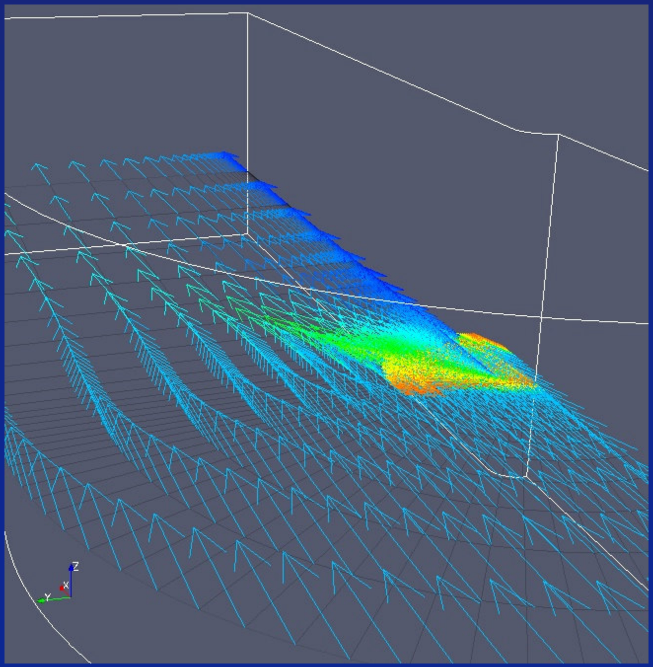
```
// Compute the magnitude of the vorticity vector
acalc->AddVectorArrayName("Vorticity");
acalc->SetResultArrayName("Vorticity Magnitude");
acalc->SetFunction("mag(Vorticity)");
```



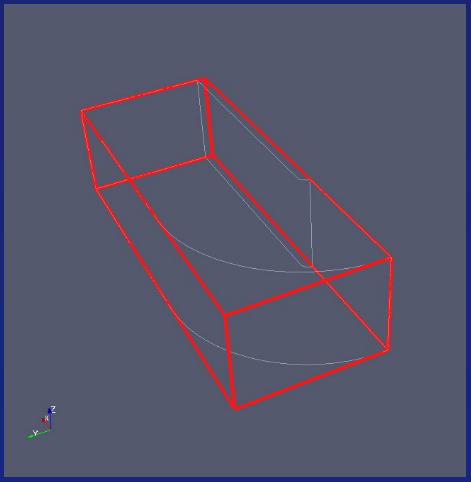
Array Calculator - Examples

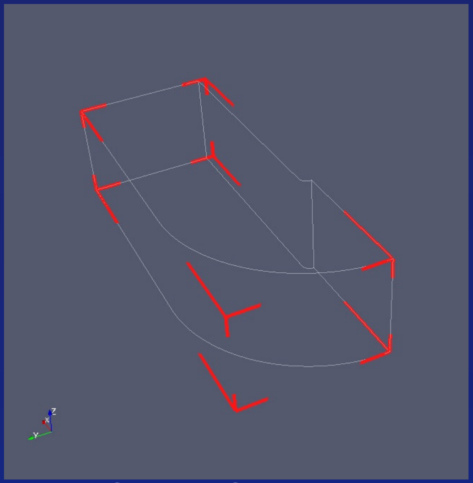
```
// Set the k component of velocity to 0
acalc->AddScalarVariable("u", "Velocity", 0);
acalc->AddScalarVariable("v", "Velocity", 1);
acalc->SetResultArrayName("2D Velocity");
acalc->SetFunction("u*iHat+v*jHat+0*kHat");
```

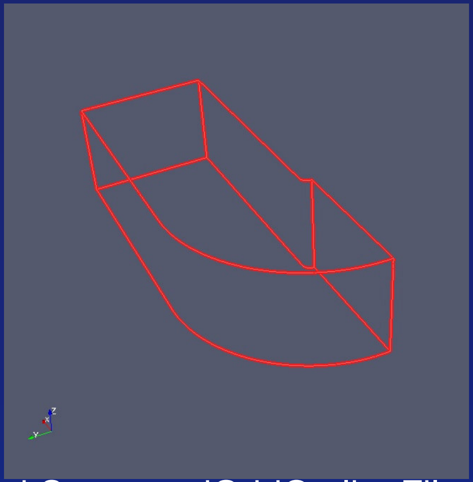


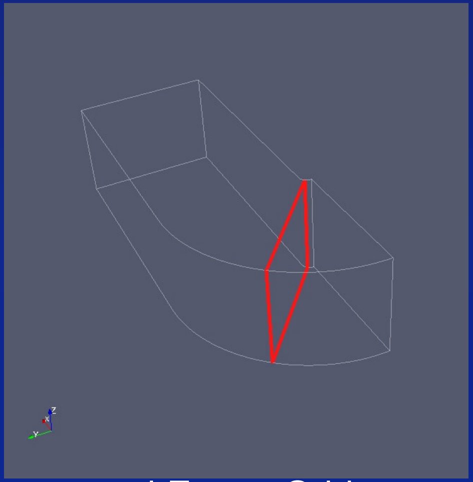


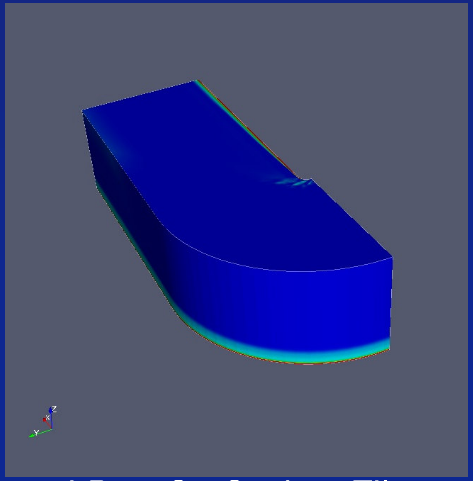
Extracting Geometry

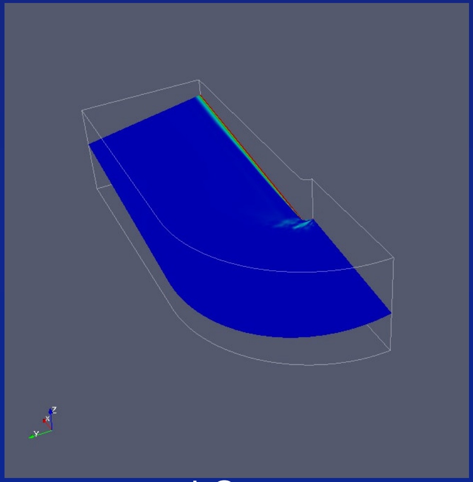

vtkOutlineFilter


vtkOutlineCornerFilter


vtkStructuredGridOutlineFilter


vtkExtractGrid
vtkExractVOI
vtkExtractRectilinearGrid


vtkDataSetSurfaceFilter


vtkCutter

Extracting Geometry - Continued

```
vtkExtractGrid* extg = vtkExtractGrid::New();
extg->SetInput(vtkStructuredGrid::SafeDownCast(
      acalc->GetOutput()));
extg->SetVOI(17, 17, 0, 31, 0, 31);

vtkLookupTable* lu = vtkLookupTable::New();
lu->SetNumberOfTableValues(256);
lu->SetHueRange(0.6667, 0);
lu->SetTableRange(0, 5);
lu->Build();

vtkDataSetMapper* mapper = vtkDataSetMapper::New();
mapper->SetInput(extg->GetOutput());
mapper->SetLookupTable(lu);
mapper->SetInterpolateScalarsBeforeMapping(1);
mapper->SetScalarMode(VTK_SCALAR_MODE_USE_POINT_FIELD_DATA);
mapper->SelectColorArray("Vorticity Magnitude");
mapper->SetUseLookupTableScalarRange(1);
```

Coloring By Vector Quantities

- Color by vector magnitude:

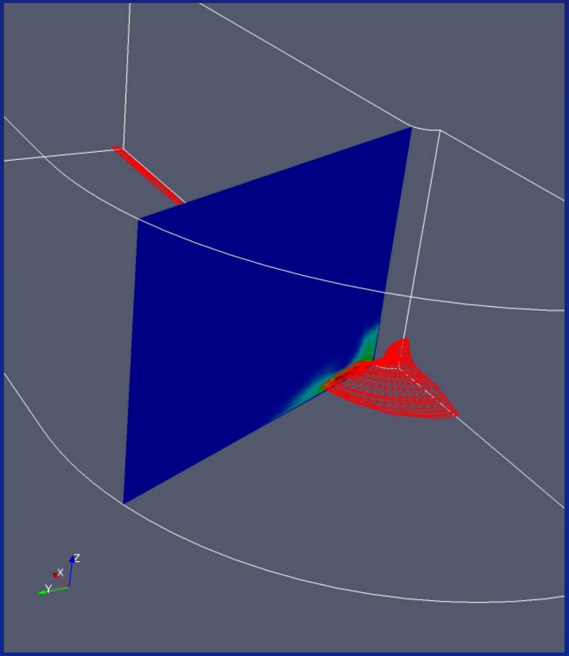
```
lu->SetVectorMode(vtkScalarsToColors::MAGNITUDE);

mapper->SelectColorArray("Vorticity");
```

- Color by vector component:

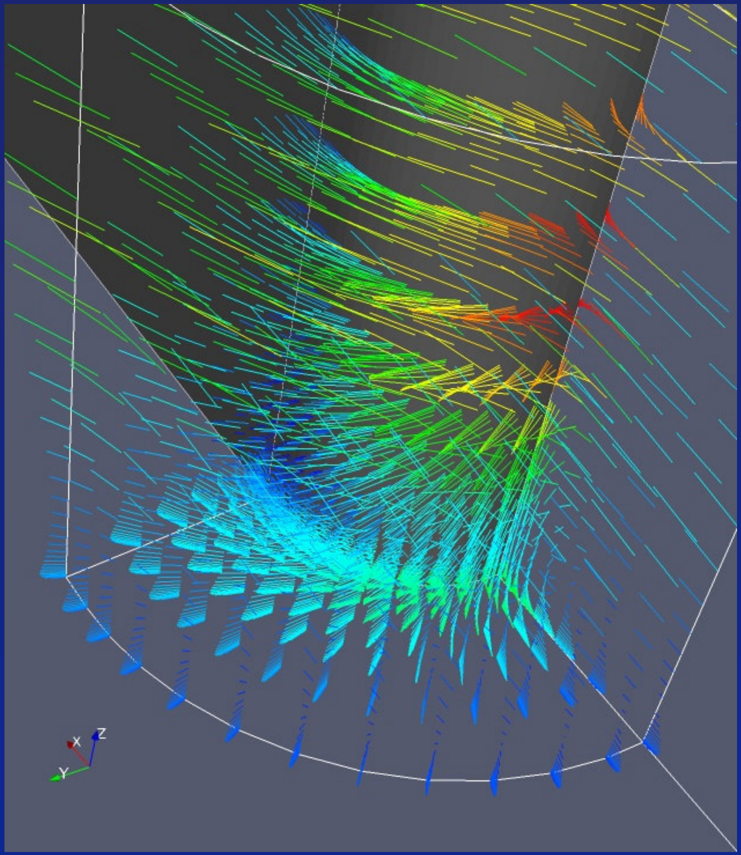
```
lu->SetVectorMode(vtkScalarsToColors::COMPONENT);
lu->SetVectorComponent(1);

mapper->SelectColorArray("Vorticity");
```

Isosurfaces

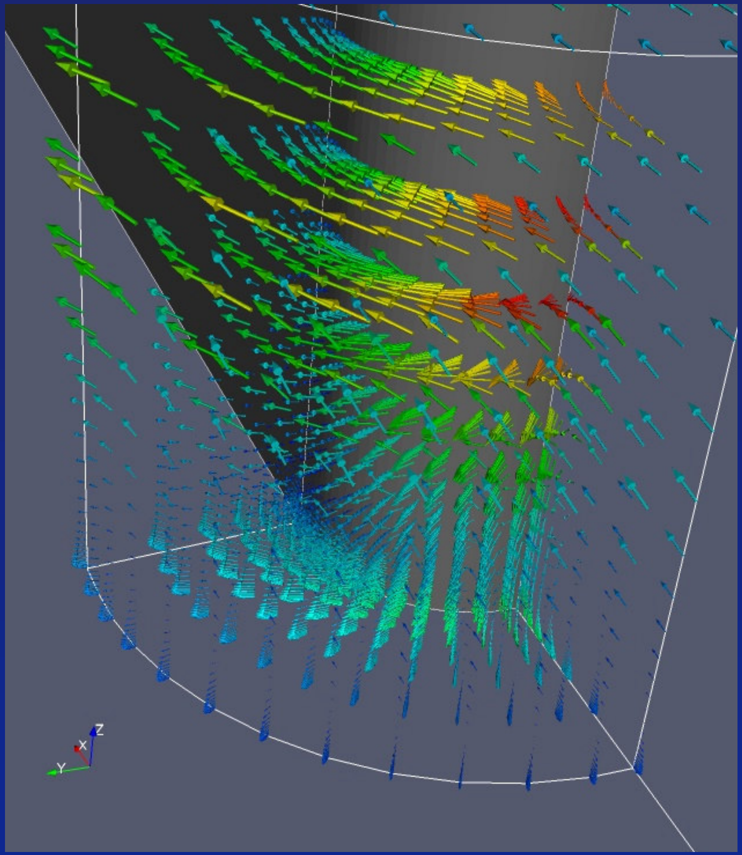
```
// Extract the 2nd component of vorticity
acalc->AddScalarVariable("w_1", "Vorticity", 1);
acalc->SetResultArrayName("Vorticity comp 2");
acalc->SetFunction("w_1");

vtkContourFilter* cf = vtkContourFilter::New();
cf->SetInput(acalc->GetOutput());
cf->SetValue(0, 5);
```



Hedgehogs and Oriented Glyphs



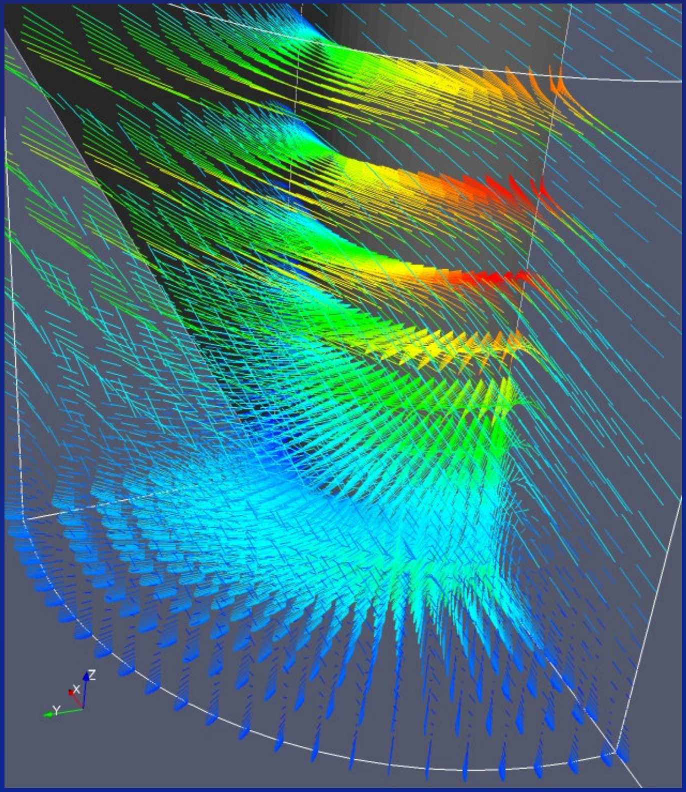
Hedgehog

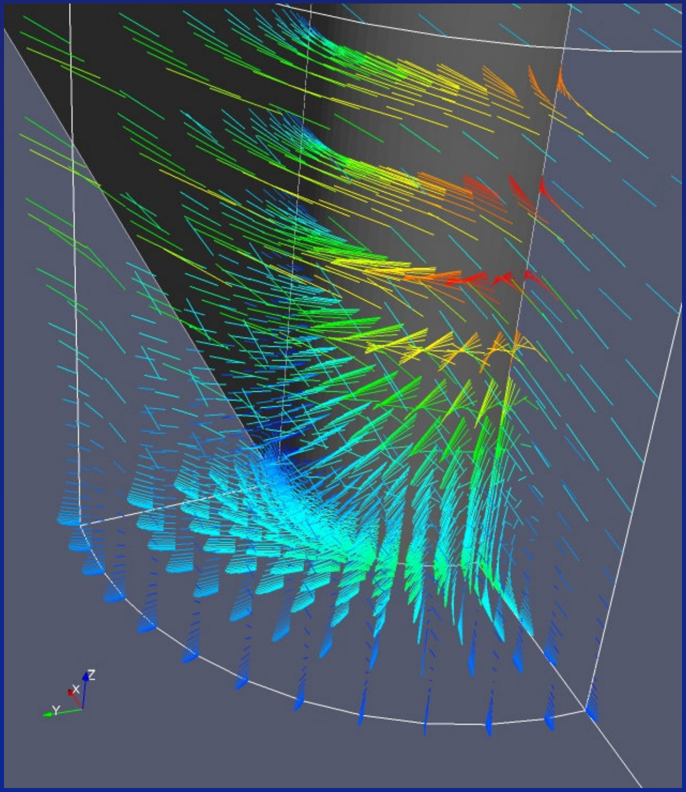


Oriented Glyph

Reducing the Number of Glyph Points

- Too many glyph points clutter the display and make it hard to interpret the plot





Downsampling – Extract Grid

- vtkExtractVOI, vtkExtractGrid, vtkExtractRectilinearGrid
- void SetVOI(int imin, int imax, int jmin, int jmax,
  int kmin, int kmax)
- void SetSampleRate(int isample, int jsample, int ksample)

```
vtkExtractGrid* extg = vtkExtractGrid::New();
extg->SetInput(vtkStructuredGrid::SafeDownCast(
                    acalc->GetOutput()));
extg->SetSampleRate(2, 2, 1);
```

Downsampling – Masking Points

- void SetOnRatio(int onpts)
- void SetMaximumNumberOfPoints(vtkIdType numPts)
- void SetRandomMode(int onoff)

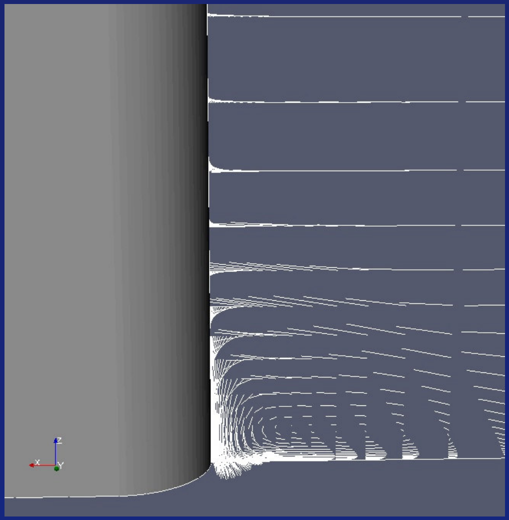
```
vtkMaskPoints* mp = vtkMaskPoints::New();
 mp->SetInput(acalc->GetOutput());
 mp->SetOnRatio(7);
 mp->SetRandomMode(1);
```

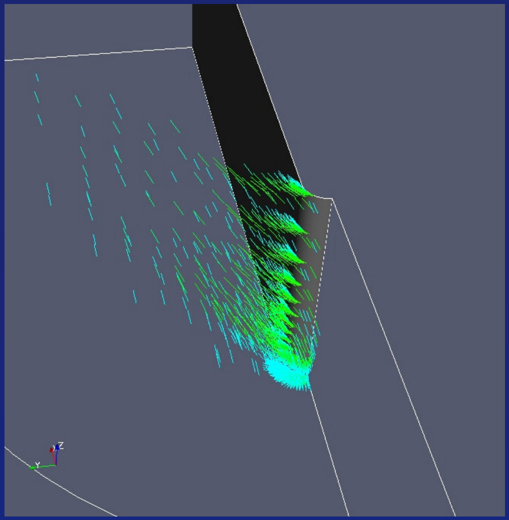
Downsampling - Threshold

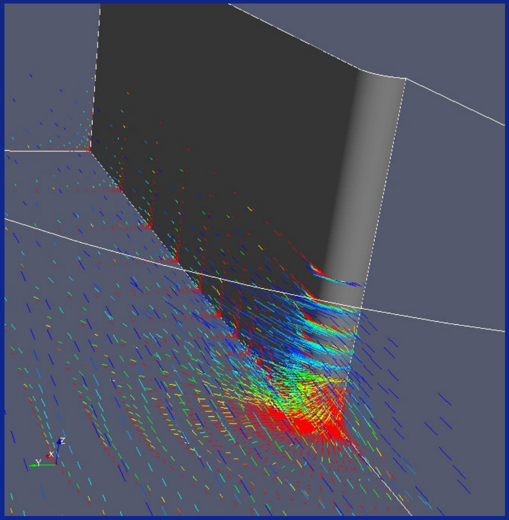
- void ThresholdBetween(double min, double max)
- void SetAttributeModeToUsePointData()
- void SetAttributeModeToUseCellData()

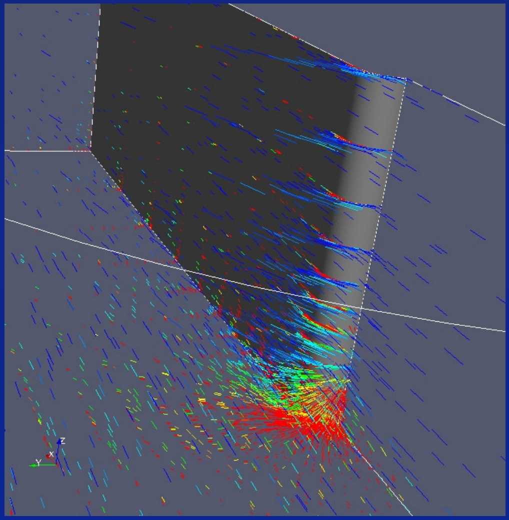
```
vtkThreshold* tr = vtkThreshold::New();
tr->SetInput(reader->GetOutput());
tr->ThresholdBetween(1.3, 3.0);
```

Downsampling - Examples


Extract Grid


Threshold (density)


Mask Points


Mask Points (random)

Hedgehogs

- Draw an oriented, scaled line for each vector
- void SetScaleFactor(double factor)

```
vtkHedgeHog* hh = vtkHedgeHog::New();
hh->SetInput(tr->GetOutput());
hh->SetScaleFactor(0.1);
```

Oriented Glyphs

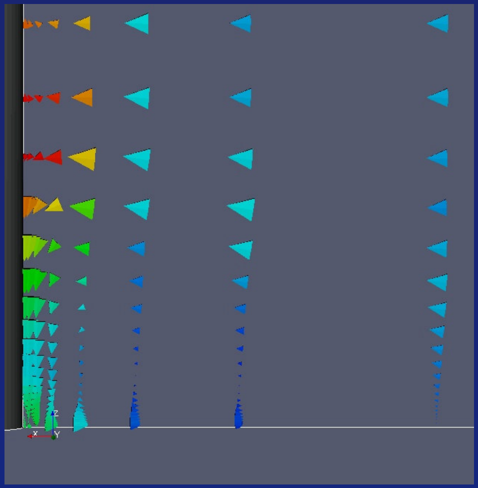
- Instead of a line, use any 2D or 3D geometric representation

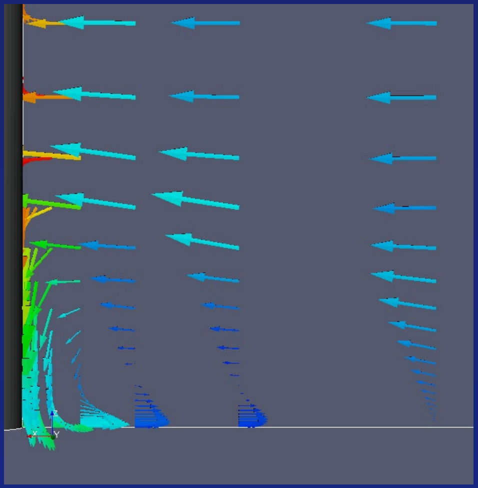
- void SetSource(vtkPolyData* source)
- void SetScaleFactor(double factor)
- void SetScaleModeToScaleByScalar()
- void SetScaleModeToScaleByVector()
- void SetScaleModeToScaleByVectorComponents()
- void SetOrient(int onoff)

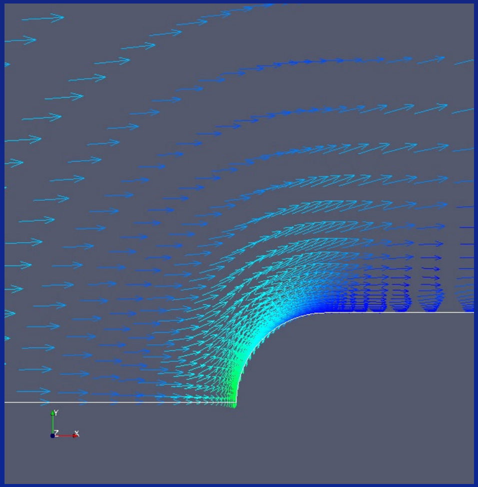
Oriented Glyphs - Continued

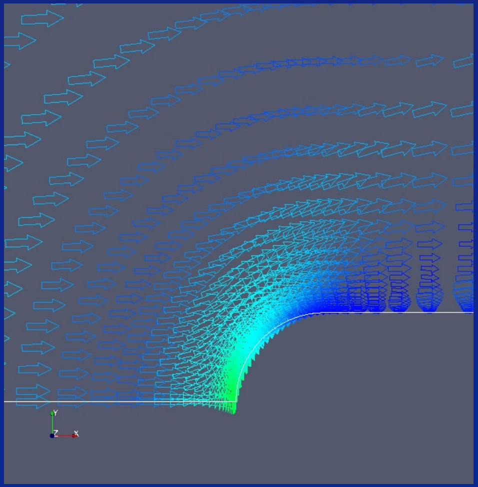
```
vtkArrowSource* as = vtkArrowSource::New();

vtkGlyph3D* gl = vtkGlyph3D::New();
gl->SetInput(tr->GetOutput());
gl->SetSource(as->GetOutput());
gl->SetScaleModeToScaleByVector();
gl->SetScaleFactor(0.1);
//gl->SetScaling(0); // turn scaling off
```

Oriented Glyphs - Sources


Cones


3D Arrows


2D Arrows


2D Thick Arrows

Warping

- Deform geometry according to a vector field:

$$\mathbf{x}_{\mathrm{i}} = \mathbf{x}_{\mathrm{i}} + s\mathbf{u}(\mathbf{x}_{\mathrm{i}})$$

- vtkWarpScalar
    - void SetScaleFactor(double factor)
    - void SetNormal(double x1, double x2, double x3)
- vtkWarpVector
    - void SetScaleFactor(double factor)

Warping - Continued

```
vtkArrayCalculator* acalc = vtkArrayCalculator::New();
acalc->SetResultArrayName("displacement");

vtkWarpVector* wv = vtkWarpVector::New();
wv->SetInput(vtkPolyData::SafeDownCast(acalc->GetOutput()));

for (int i=0; i<numSteps; i++)
  {
  double time = maxTime / numSteps * i;
  ostrstream func;
  func << "(" << sin(w1*time) << "*mode1) + ("
       << sin(w2*time) << "*mode2)" << ends;
  acalc->SetFunction(func.str());
  delete[] func.str();
  renWin->Render();
  }
```

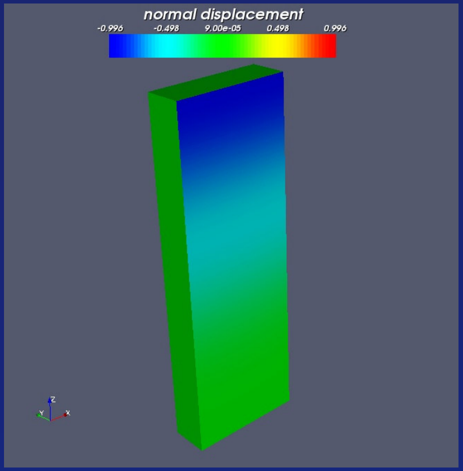
Displacement Plots

- Shows the motion of an object perpendicular to its surface

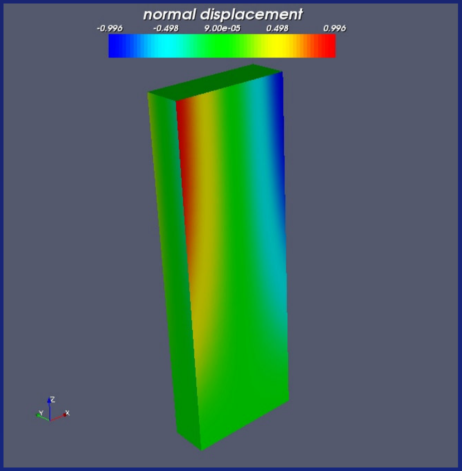
  $M(\mathbf{x}_\mathbf{i}) = \mathbf{u}(\mathbf{x}_\mathbf{i}) \cdot \mathbf{n}(\mathbf{x}_\mathbf{i})$ where **n** is the normal to the surface

```
vtkPolyDataNormals* pdn = vtkPolyDataNormals::New();
 pdn->SetInput(
   vtkPolyData::SafeDownCast(reader->GetOutput()));
 pdn->SetFlipNormals(1);

 vtkArrayCalculator* acalc =
          vtkArrayCalculator::New();
 acalc->SetInput(pdn->GetOutput());
 acalc->SetResultArrayName("Normal Displacement");
 acalc->SetFunction("Normals.mode8");
```

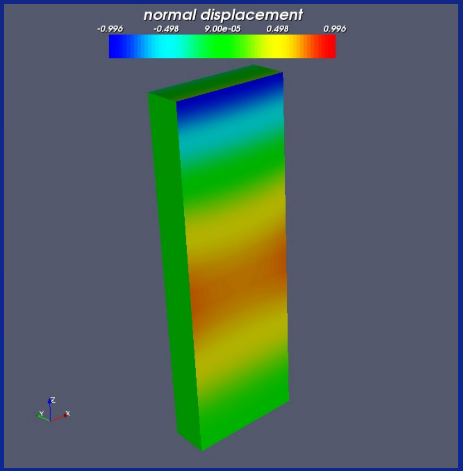
Displacement Plots - Continued



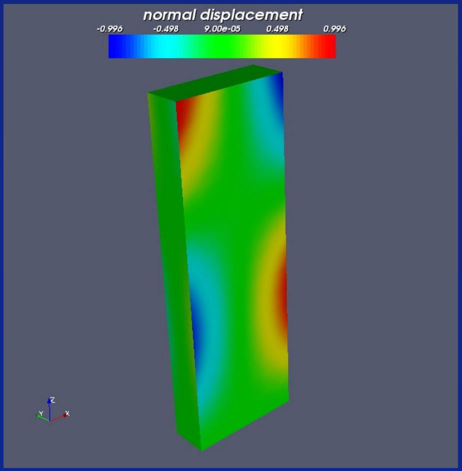
Mode 1



Mode 3



Mode 4



Mode 8

Streamlines - Outline

- Streamline Basics
- Streamtubes
- Streamribbons
- Streamsurfaces
- Integrators
- Caveats

Definitions

- Along a streamline the velocities of the fluid elements are everywhere tangent to it.
- A path line represents the path of a particular fluid element.
- A streak line is defined as a line formed by the fluid elements which pass through a given location in the flow field.
- In a steady flow, streamlines, path lines and streak lines are coincident.

Definitions - Continued

- A path line is defined as

$$\frac{dx_i}{dt} = u_i(\mathbf{x}, t)$$

- A streamline is defined as

$$\frac{dx_i}{ds} = u_i(\mathbf{x}, t_0)$$

Basic Streamline Algorithm (Euler)

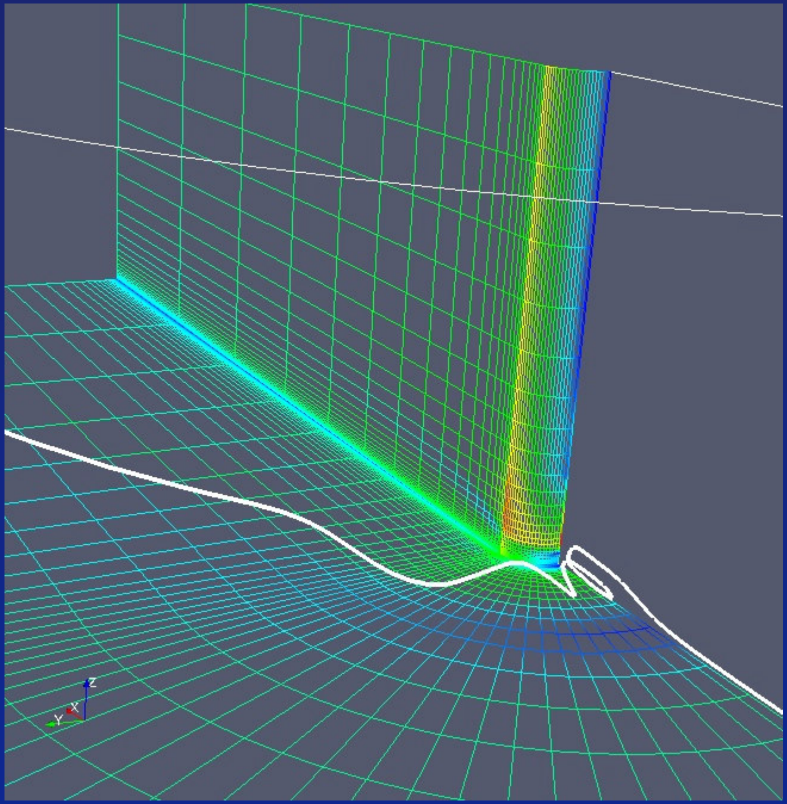
1. Start at x_0
2. Find cell containing current point
3. Interpolate the velocity field at current point
4. Compute next point with $x_{i+1}=x_i+u_i\,\Delta t$
5. If next point is in the data set, add it to the streamline (if requested, interpolate other field data)
6. If next point is in the data set, set current point to next point and go to 2
7. Associate the termination reason with the current streamline and move to the next one

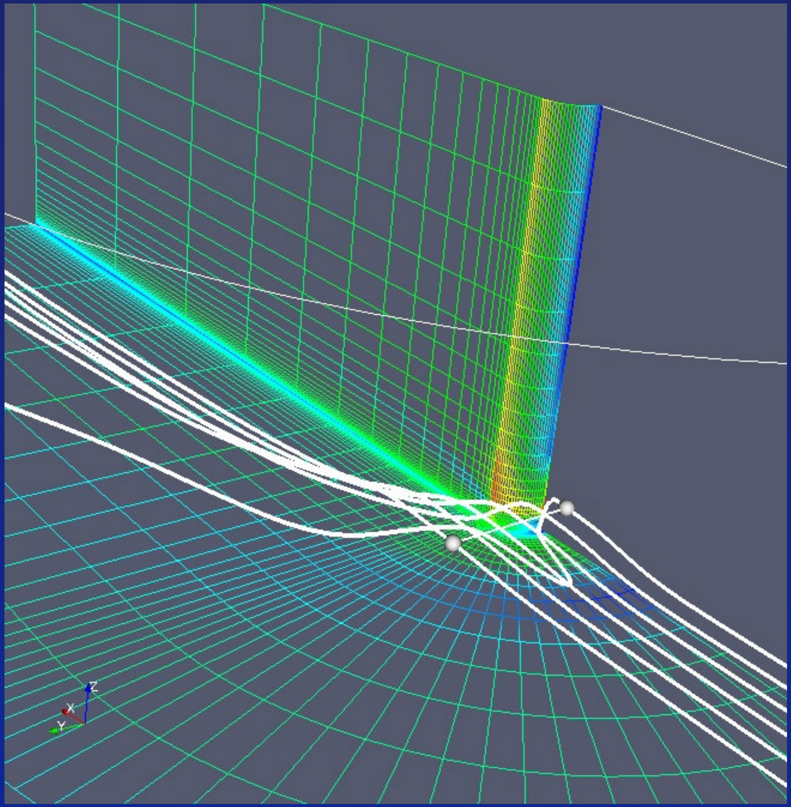
Note that the default integrator in VTK is 2nd order Runge-Kutta not Euler.

Basic Streamline Integration

```
vtkPointSource* ps = vtkPointSource::New();
ps->SetCenter(-0.44483, 0.18709, 0.25978);
ps->SetRadius(0);
ps->SetNumberOfPoints(1);

vtkStreamTracer* tracer = vtkStreamTracer::New();
tracer->SetInput(reader->GetOutput());
tracer->SetSource(ps->GetOutput());
tracer->SetMaximumPropagation(
          vtkStreamTracer::LENGTH_UNIT, 30);
tracer->SetIntegrationDirectionToBoth();
```

Basic Streamline Integration





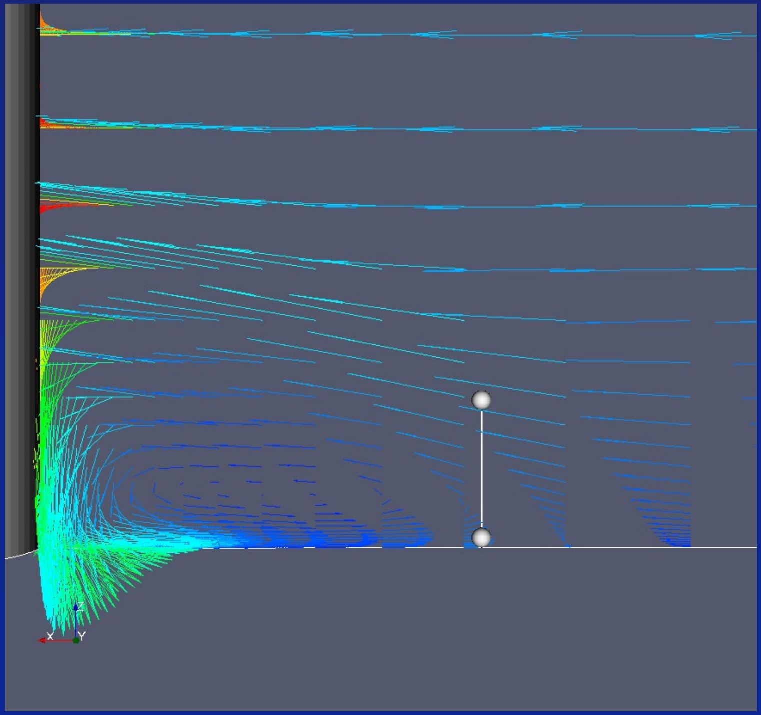
Basic Streamline Integration

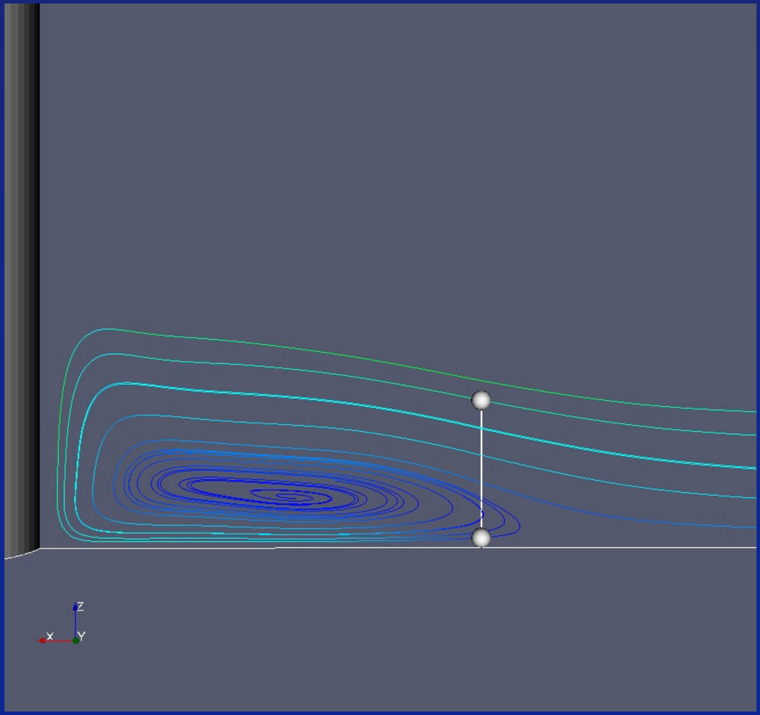
- void SetStartPosition(double x, double y, double z)
- void SetSource(vtkDataSet* source)
- void SetMaximumPropagation(int unit, double max)
- void SetInitialIntegrationStep(int unit, double step)
- void SetMaximumNumberOfSteps(vtkIdType maxSteps)
- void SetTerminalSpeed(double termSpeed)
- void SetIntegrationDirection()

units = { TIME_UNIT, LENGTH_UNIT, CELL_LENGTH_UNIT }

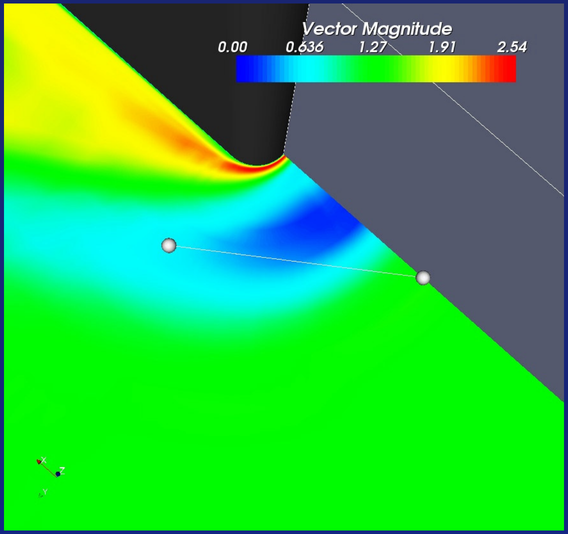
Seed Placement

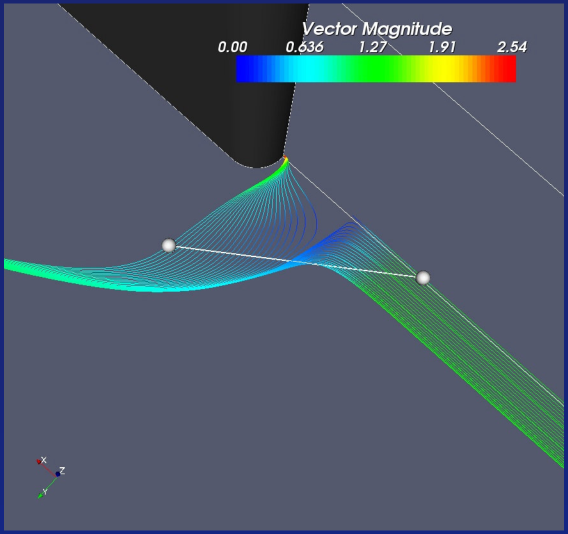
- Place streamline seeds near critical (stagnation) points
- Start with rakes and refine as necessary
- It is sometimes easier to work on 2D planes when placing seeds

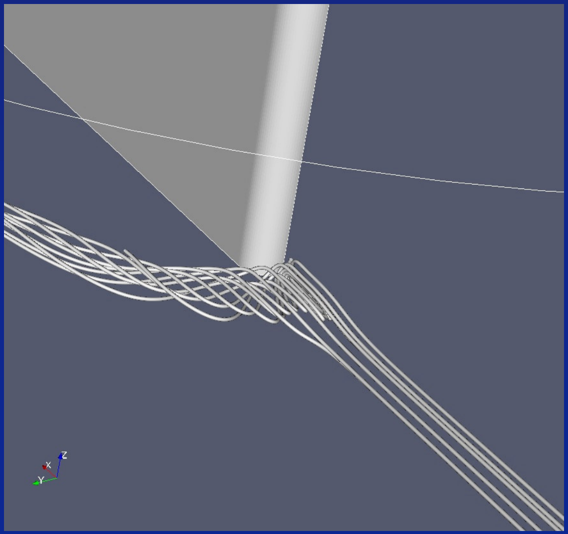




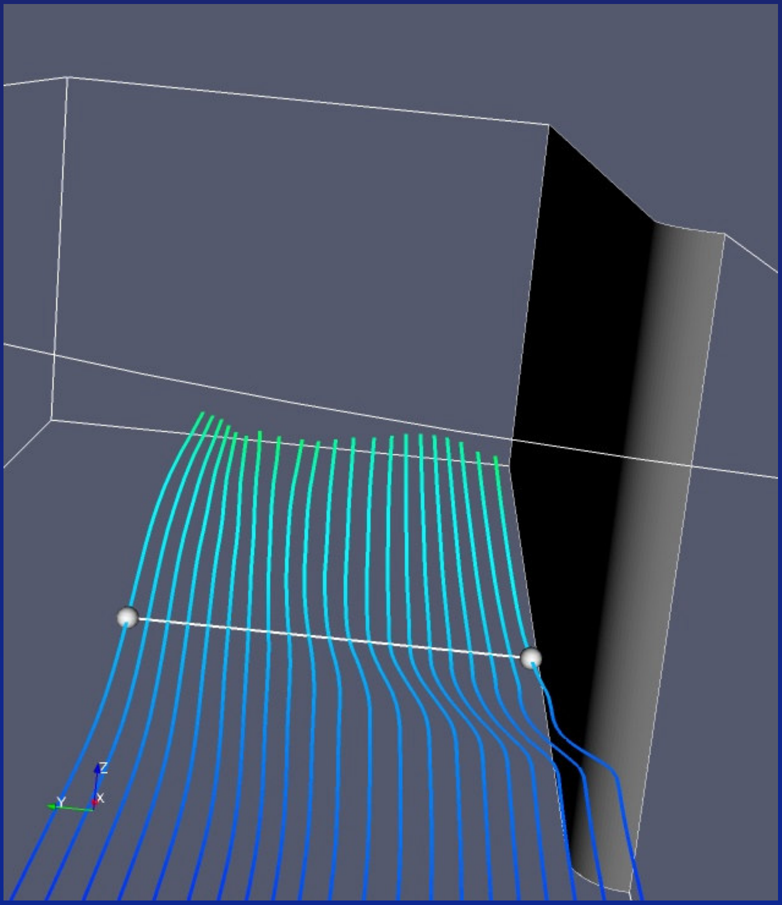
Seed Placement - Continued

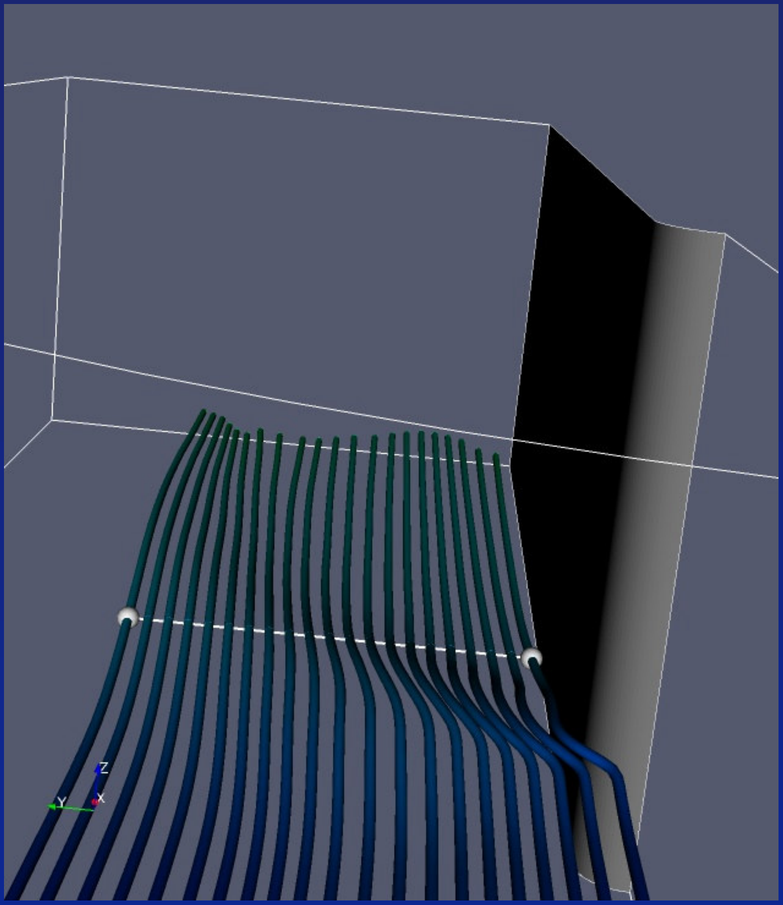






Streamtubes



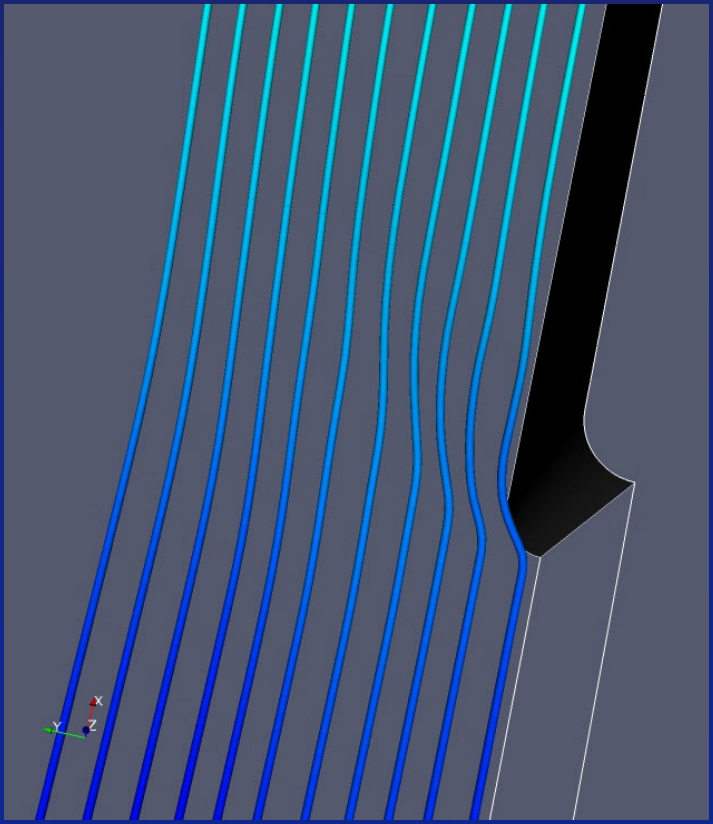


Streamtubes - Continued

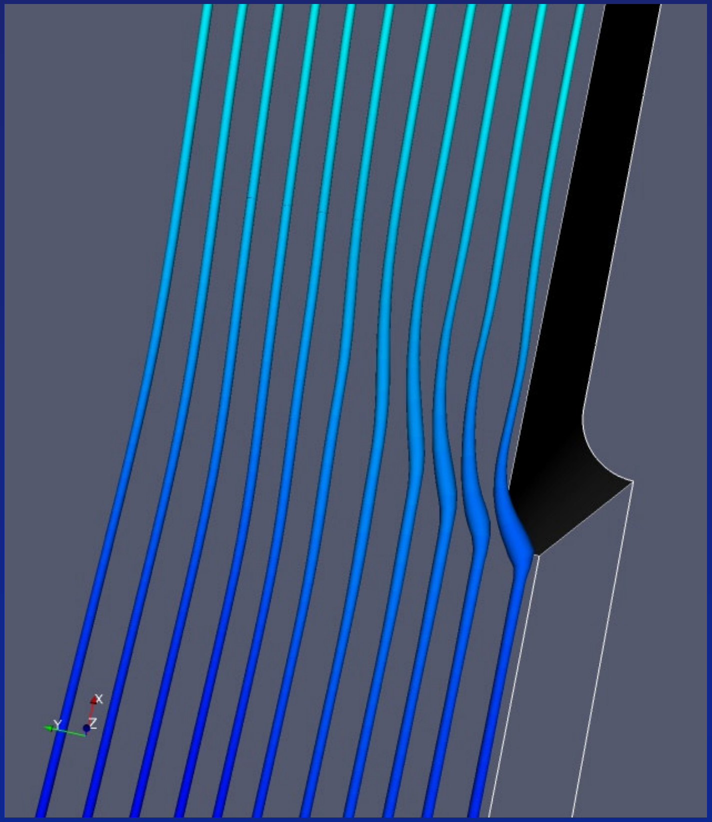
- void SetRadius(double radius)
- void SetVaryRadius()
- void SetRadiusFactor(double factor)
- void SetNumberOfSides(int numSides)

```
vtkTubeFilter* tube = vtkTubeFilter::New();
tube->SetInput(tracer->GetOutput());
tube->SetRadius(0.05);
tube->SetNumberOfSides(16);
tube->SetVaryRadiusToVaryRadiusByScalar();
tube->SetRadiusFactor(3);
```

Streamtubes - Continued

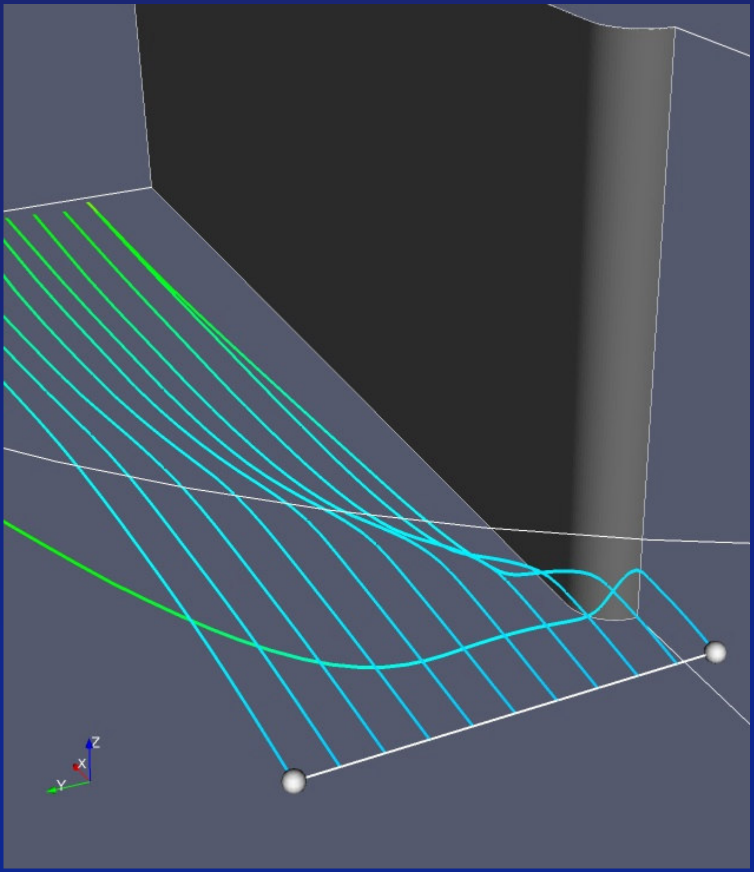


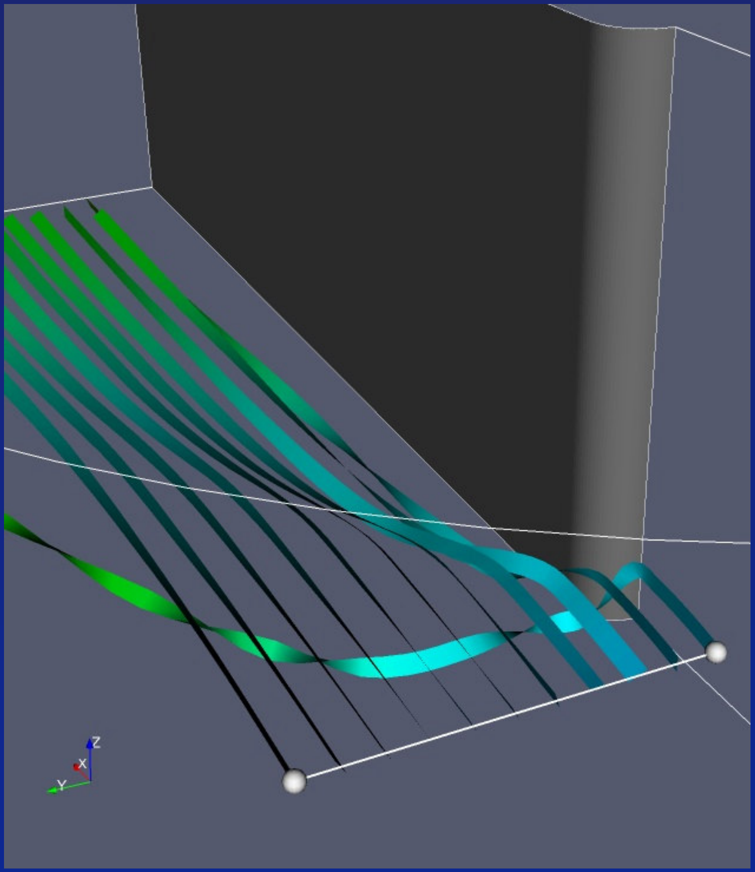
Constant radius



Radius varied based on density

Streamribbons





Streamribbons - Continued

- void vtkStreamTracer::SetComputeVorticity(int onoff)
- void SetWidth(double width)
- void SetAngle(double angle)
- void SetVaryWidth(int onoff)
- void SetWidthFactor(double factor)

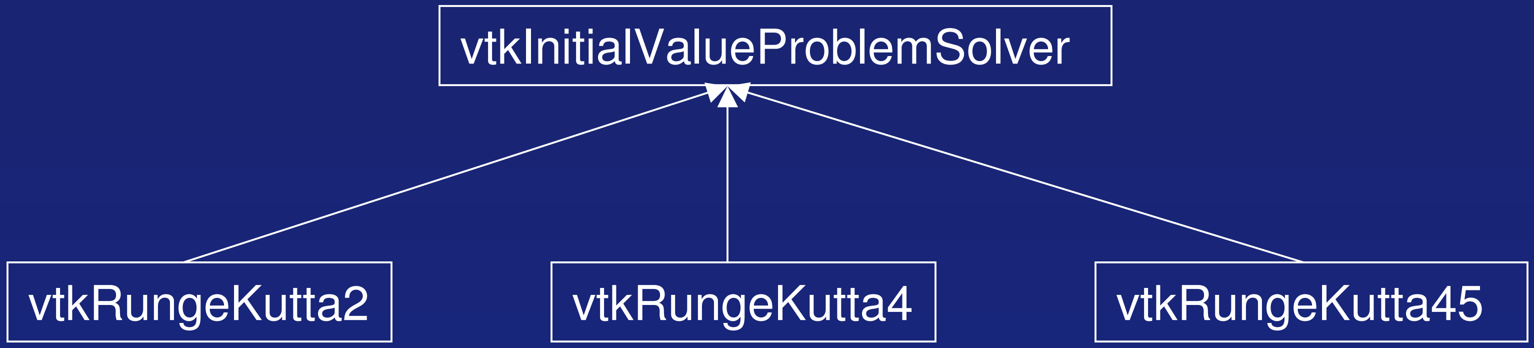
```
vtkRibbonFilter* ribbon = vtkRibbonFilter::New();
ribbon->SetInput(tracer->GetOutput());
ribbon->SetWidth(0.1);
```

Integrators – Higher Order

- By default, vtkStreamTracer uses a 2nd order integrator (2nd order Runge-Kutta)
- Higher order integrators do not require step sizes as small as lower order integrators
- Adaptive step size integrators are more flexible using smaller step sizes when necessary but switching to larger step sizes in regions of small gradient

- void SetIntegrator(vtkInitialValueProblemSolver* solver)

Integrators



- void SetFunctionSet(vtkFunctionSet* functionSet)
- All sub-classes of vtkInitialValueProblemSolver must define:

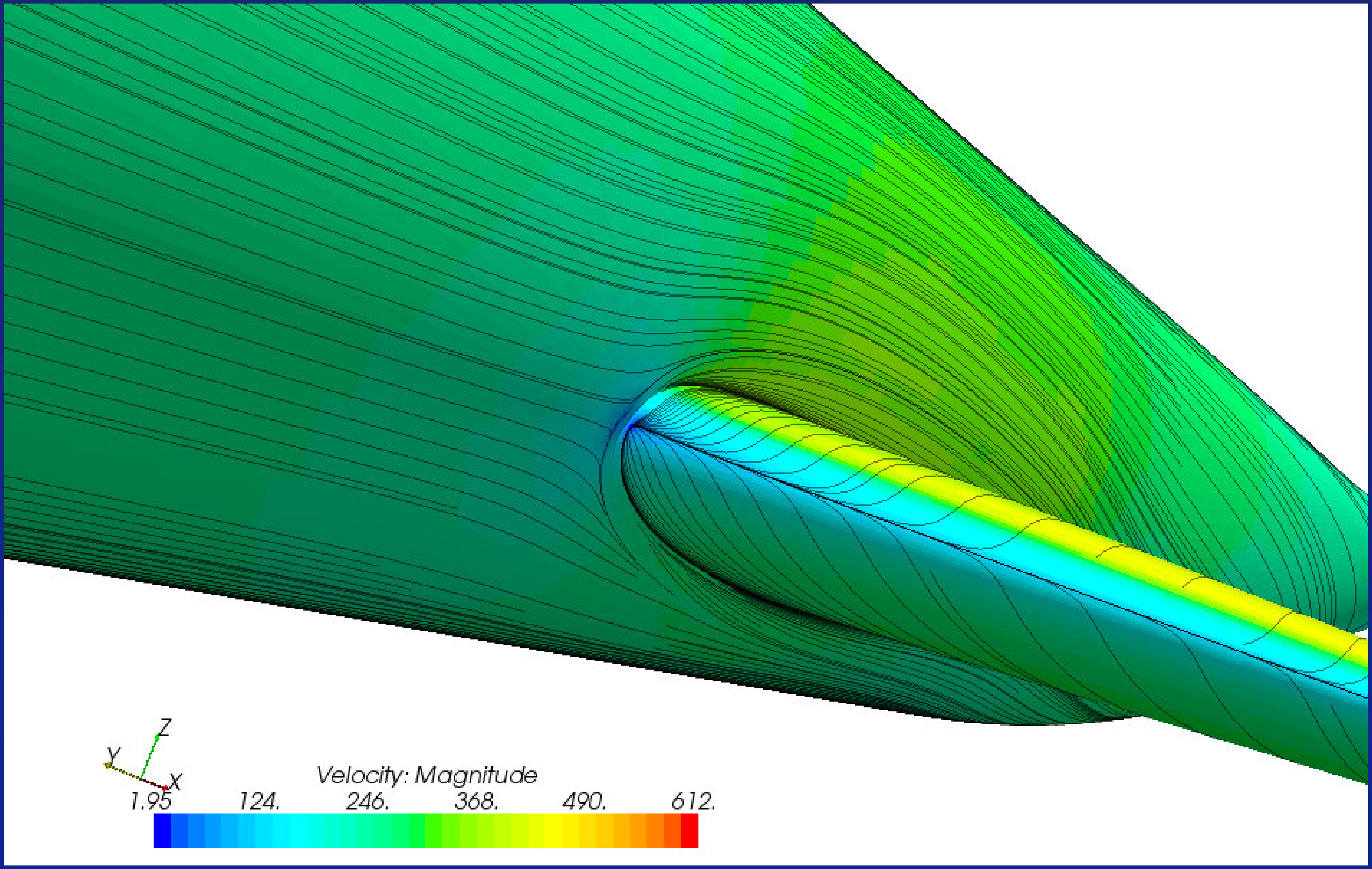
```
virtual int ComputeNextStep(double* xprev, double* dxprev, double* xnext,
                            double t, double delt, double& delTActual,
                            double minStep, double maxStep,
                            double maxStep, double error)
```

Adaptive Integration Parameters

- void SetMinimumIntegrationStep(int unit, double step)
- void SetMaximumIntegrationStep(int unit, double step)
- void SetMaximumError(double maxError)

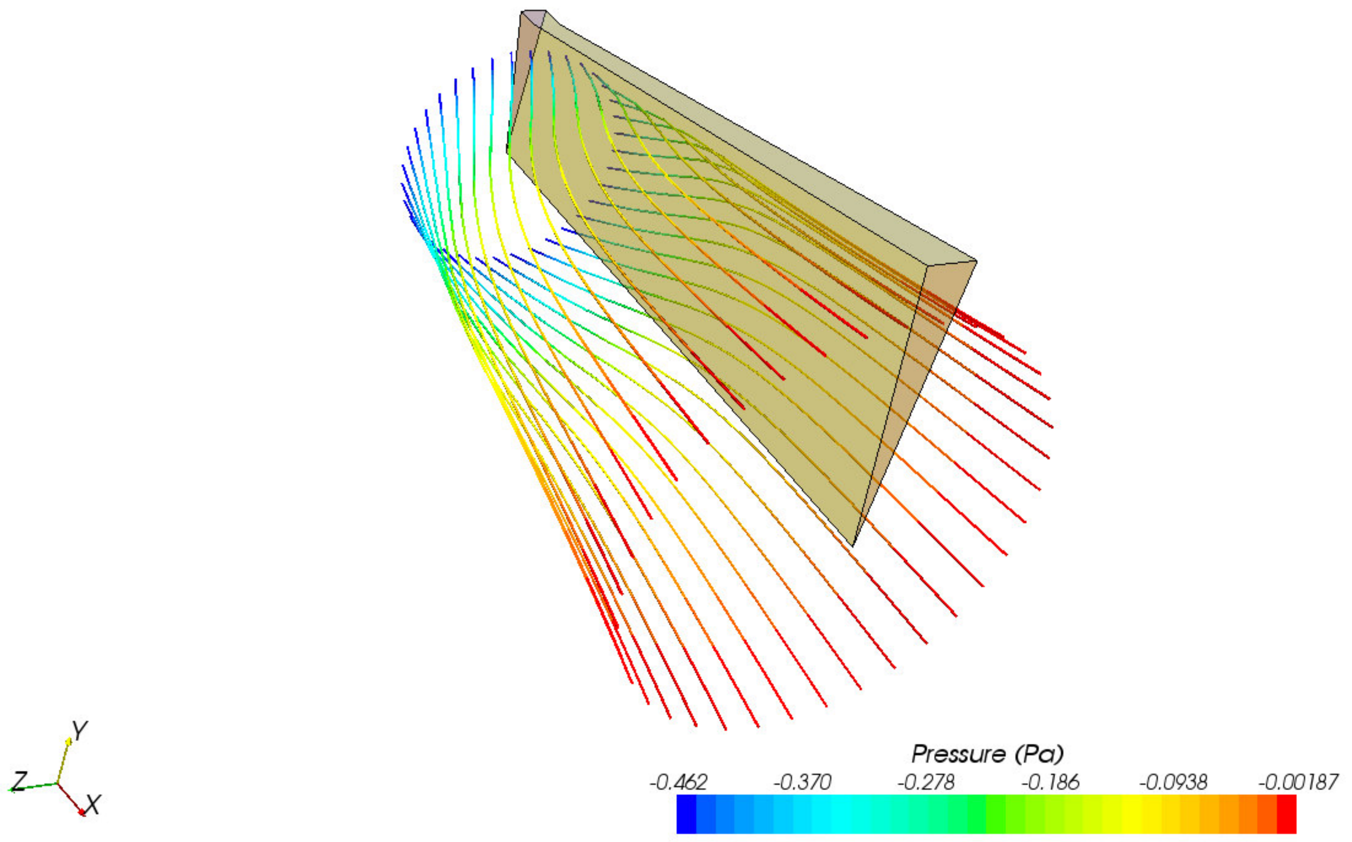
units = { TIME_UNIT, LENGTH_UNIT, CELL_LENGTH_UNIT }

Surface Constrained Streamlines



Courtesy of Jeff Lee, CD adapco Group

Streamlines Across Periodic Boundaries



Courtesy of Jeff Lee, CD adapco Group