Vistle

Tutorial

HLRS

May 2021

Authors:
Leyla Kern
Martin Aumüller

Content

1. About Vistle **3**

2. Getting started **4**

2.1. Download and installation 4

2.2. Starting Vistle 4

3. The Vistle GUI **5**

4. COVER **6**

5. Vistle modules **7**

6. Creating a module pipeline **8**

6.1. Pipeline steps 8

6.2. Examples 9

7. Basic tutorials **12**

7.1. Tutorial 1 12

7.2. Tutorial 2 14

8. Further links **16**

1. About Vistle

Vistle is a software environment for scientific visualization of large-scale data sets. It implements a data-parallel visualization pipeline and can distribute the pipeline steps onto several interconnected clusters. Workflows are configured in an explicit graphical user interface. Vistle has a special focus on working in immersive virtual environments such as CAVEs and head-mounted displays. In order to enable visualization of large-scale data sets in such environments, object and image based remote rendering can be configured within the visualization workflows.

This document gives an introduction to Vistle. Key components are explained and simple examples presented, such that the user can easily learn the basic usage of Vistle.

2. Getting started

2.1. Download and installation

You can obtain Vistle from the Github repository:

https://github.com/vistle/vistle

Make sure you have installed all build requirements and follow the installation guide for Vistle as depicted on Github.

On MacOS, you can also use Homebrew for installation:

```
> brew install hlrs-vis/tap/vistle
```

2.2. Starting Vistle

You can start Vistle from the command line using the command

```
> vistle
```

This will open the Vistle GUI. You can also append the path to an existing Vistle file (.vsl) to open it on startup.

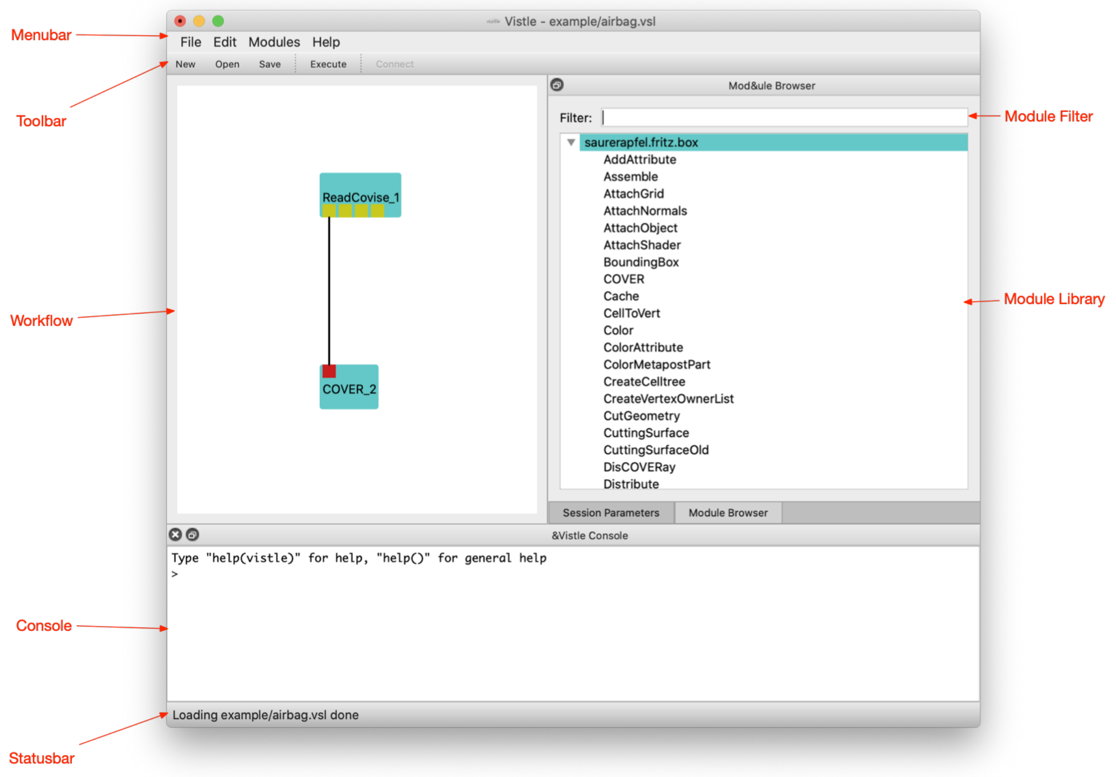
```
> vistle myFile.vsl
```

For a complete list of arguments, type

```
> vistle -h
```

3. The Vistle GUI

The Vistle user interface is composed of several components, as shown in Figure 1. At the top of the window, there is a menubar and a Toolbar for shortcuts to often used menu items. The left area below is the Workflow Area where the visualization pipeline is displayed graphically. At the right-hand side, there is the module window showing the Module Browser, composed of the Module Library, a list of available visualization modules, and the Module Filter. Using the tabs below, one can switch to the Module Parameters (not shown in Figure 1). The bottom pane is the Vistle Console, where you can enter Python commands and where modules can display messages. At the bottom of the window is a status bar which will occasionally display progress information.



```
Type "help(vistle)" for help, "help()" for general help
>
```

Loading example/airbag.vsl done

Figure 1: The Vistle GUI

For the following steps, it can be useful to follow the description in Vistle on a sample application. To do so, you can load a predefined module pipeline: Click on *Open* in the *Toolbar,* navigate to your Vistle directory and the *examples* folder. From there select to open *airbag.vsl.* The file browser window will vanish and modules as shown in Figure 1 will appear in the Workflow Area. Also, a new window, *COVER,* will open.

To execute the workflow, press the *Execute* button, in the Toolbar of the Vistle GUI. You should now see the visualized animation in the *COVER* window.

4. COVER

COVER is the module responsible for displaying the 3D scene. You can use it on a desktop, on immersive multi-screen projection system and with VR glasses. Here, we learn how to use it when configured for the desktop.

The COVER window consists of a menubar in the top, a Toolbar below and the render area, where the scene is shown. You can interact with the scene using your mouse. Moreover, there is the VR Menu, usually at the right-hand side of the render area. From there, as well as from the menu and Toolbar, you can configure several often used settings. In Figure 2, which shows the COVER window, the Toolbar buttons functions are indicated by labels. From left to right you can find settings for animations, views, navigation and workflow execution.

When an entry in the VR Menu is selected, a frame for its content opens up. Click the Menu entry once more to hide the frame again.

If COVER is started from within Vistle, a *Vistle* menu item is available in the menubar. From there it is possible to manipulate Module Parameters and enable interactors.

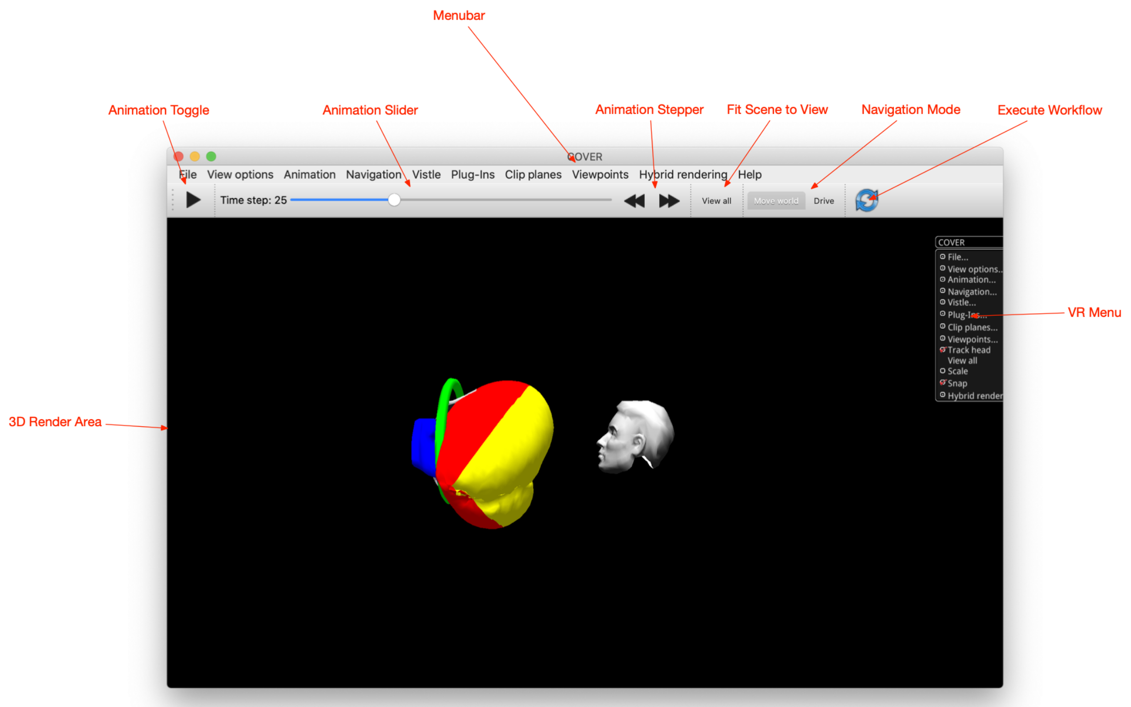


Figure 2: COVER renderer with desktop and VR user interface.

Further, COVER comes along with a tablet user interface (TabletUI), which allows the user to adjust many other parameters and in many cases offers more detailed settings. The TabletUI attaches to a running COVER and can be started using the command:

```
> tabletUI
```

You can also start COVER without Vistle, by typing:

```
> opencover
```

5. Vistle modules

Vistle Modules encapsulate processing steps of the workflow. They are represented by turquoise boxes. Modules can have input and output ports to exchange data with subsequent or preceding modules via shared memory. These ports are rendered as red and yellow squares for input and output, respectively. Links between two ports are depicted by lines. If you hover over a port using the mouse, information on the object type on this port is shown, e.g. *grid, scalar, vector*. This is useful, to know which two ports are compatible. Different types demand different processing. For instances, vector data is preferably handled by *Tracer* and *VectorField,* whereas scalar data is processed by *IsoSurface* or *DomainSurface,* among others. Examples for processing of the different types are given later on.

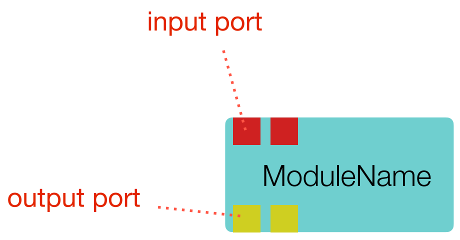


Figure 3: Vistle module

Modules can have a number of parameters that can be changed in the tab of the module window. To see the parameters for a different module, it needs to be selected by a click on the module in the Workflow Area. The selected module will be highlighted by a violet frame.

Double-click on a module triggers its execution, and also of subsequent modules. The execution state is indicated by a yellow frame. Right-click on a module opens a menu for additional options. If a module vanishes from the Workflow Area, this means it has crashed.

6. Creating a module pipeline

A workflow is assembled as a pipeline of modules in the Workflow Area. The modules can be chosen from the list in the module window and are added using drag-and-drop. A connection between two modules is established by linking their ports i.e. clicking and dragging the mouse from one port to the other. If the line does not remain after releasing the mouse, the connection was invalid. In case the input port is already linked to another port, it might not be possible for some modules to add a second port connection. The invalid link will appear as a grey line and does not function as a connector. A connection can be deleted by double-clicking on it. It is up to the user to ensure compatible connections. However, modules might print helpful information on execution.

When assembling a module pipeline, some modules are useful to extract information which can be used for validation or retrieving meta data: *BoundingBox, PrintMetadata, ObjectStatistics, Extrema.*

6.1. Pipeline steps

The necessary stages when creating a workflow strongly depend on the individual application. However, the following workflow can be used as a reference:

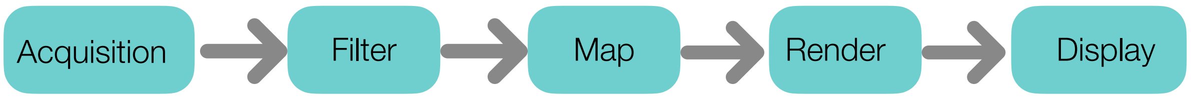


Figure 4: Pipeline steps

First, data is required, which is typically retrieved by importing data from a file or directory. Data is acquired using read modules. A number of these modules is available, each supporting the import from files of different formats. Examples are *ReadFoam, ReadCovise* or *ReadNek5000.*

Next, features of interest can be filtered from the data set. This can be done by cutting off redundant parts using the module *CutGeometry,* extracting the outer surface of an object using *DomainSurface* or filtering a specific grid layer using *IndexManifolds,* for instance.

However, the then obtained geometries itself do not provide information about data values, but only about the grid. Therefore, data values need to be mapped to the extracted objects. A *Color* module can be added to convert data values into color values which are mapped to the surfaces. Moreover, also geometries can also be created based on data values, as through the *IsoSurface* module.,

The generated objects are then forwarded to the renderer, *COVER* and displayed.

In summary, a map implementing the stages depicted in Figure 4, can look as follows:

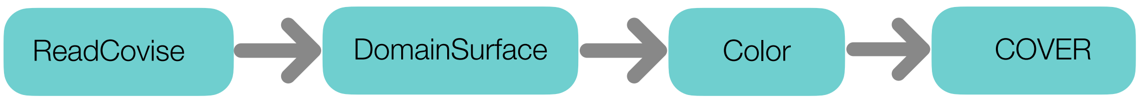


Figure 5: Example of a Vistle pipeline implementation

Note, that not all modules can be employed on all data sets, as some might require specific data structures, such as uniform grids or 3D vector data.

6.2. Examples

Some examples demonstrating the usage of basic modules are given in the following. Moreover, hints for addressing common issues are listed. The examples are separated into scalar data visualization and vector data visualization. The former can be applied when a scalar field such as temperature or pressure is present. The latter is used to handle vector fields like wind fields or flows.

Scalar data

Figure 6 shows three examples for scalar data visualisation. Sample data is generated using the *Gendat* module. *Gendat* can be replaced by a read module, providing it has the according output object types (i.e. scalar data).

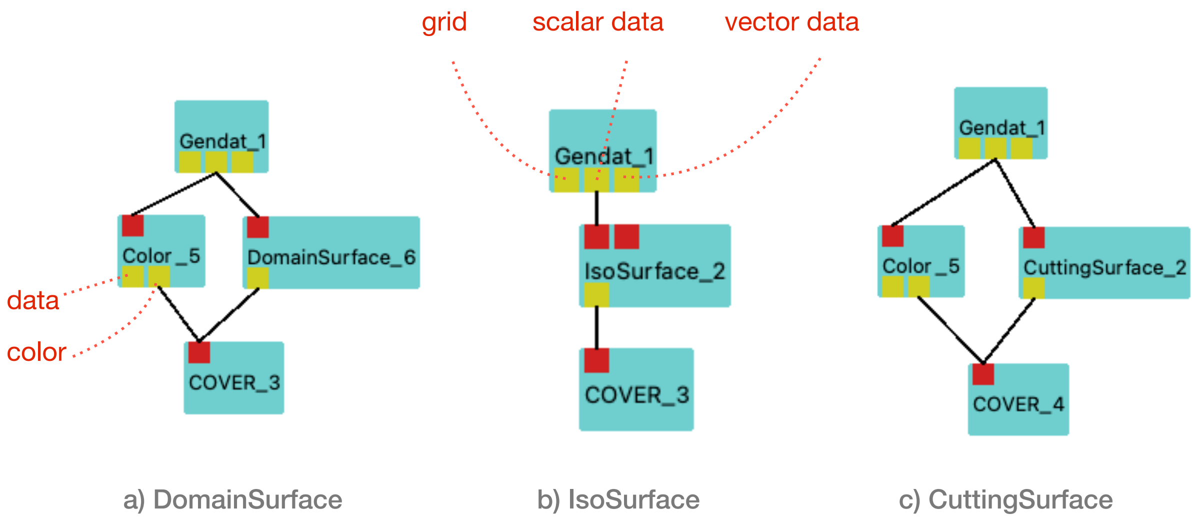


Figure 6: Example maps for scalar data processing

Gendat has three output ports. From left to right is the underlying grid, scalar and vector data available. In this example we will use the central port, for scalar data. This is then passed to the **Surface* modules, to extract and compute object surfaces. Example a) extracts the grid surface using *DomainSurface;* in b) the surface is computed based on an isovalue or -point; in c) the computation of an intersection area of the grid with a cutting surface is implemented.

Besides, in case a) and c), a *Color* module enables colorization of the extracted surface by setting a color value range to map the data values to.

The corresponding renderings for the three examples are shown in Figure 7.

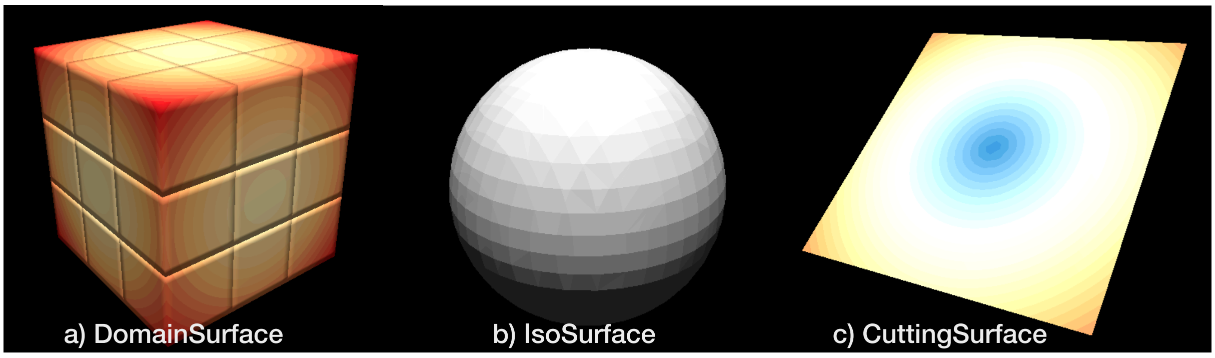


Figure 7: Renderings of scalar data

Vector data

For vector data, the modules *Tracer* and *VectorField* can be used. An example for the usage of *Tracer* is shown in Figure 8, computing streamlines from vector data. Sample vector data is generated using the *Gendat* module. It is then forwarded to the *Tracer*. The *Tracer* provides various output, however in this tutorial only the first port, containing data, is used. The data values are then mapped to a color map in the subsequent *Color* module. For refinement of the streamlines' display, the *Tubes* module is appended where geometry properties like thickness of the streamlines can be adjusted. Before passing the object to *COVER,* the geometry needs to be converted into Triangles.

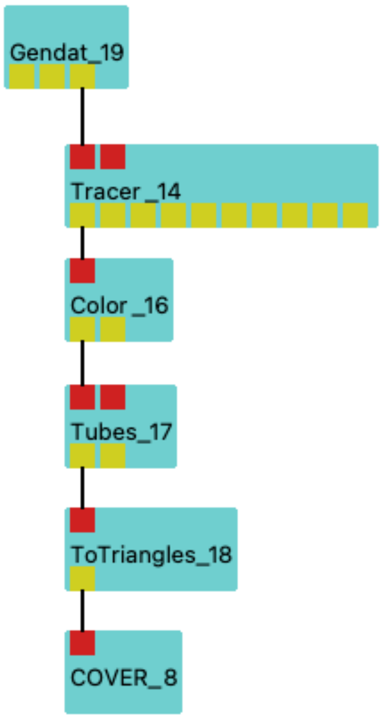


Figure 8: Example for vector data processing

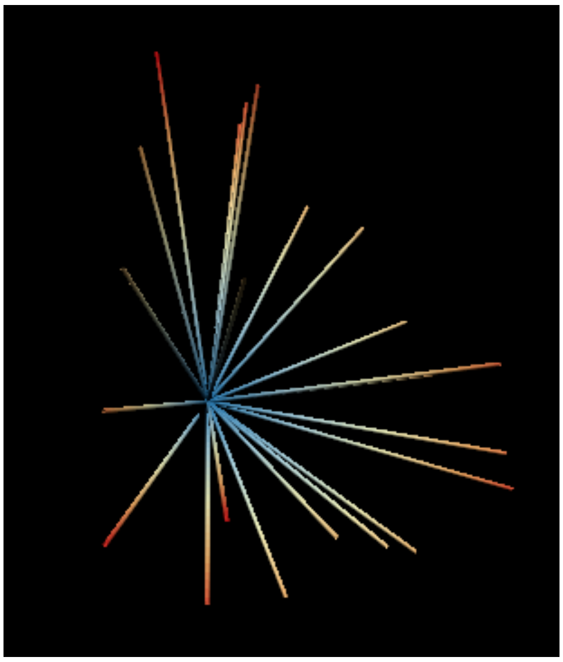


Figure 9: Example of vector data rendering

Hints

In case you observe difficulties when assembling the above shown examples yourself, here are some settings to check, in particular for the case that COVER just remains blank after execution.

COVER: Click on *View all* in the Toolbar of the COVER window (or press the key "V") to adjust the view such that the whole object is visible.

IsoSurface: Make sure, a reasonable isovalue (or -point) is set.

CuttingSurface: Use the PickInteractor in COVER to adjust the planes location and orientation
(In the COVER menubar: Vistle → CuttingSurface → PickInteractor)

Tracer: In the Module Parameters, increase the length, number of points. User COVER's PickInteractor to adjust the starting location

Tubes: Thicken the streamlines, by increasing the tube's radius in the Module Parameters.

Meta data and information

As mentioned above, some prior knowledge is needed to set up a processing pipeline. Different data structures demand for different processing, as shown for scalar and vector data. Also, data value ranges or geometric extends might have to be known to achieve adequate preparation of the data. Thus, the following supporting module should be kept in mind. Note, that these are just examples and there are plenty more modules that can be used.

BoundingBox: Obtain extrema of a grid, such as minimal and maximal coordinates, the ID of the partition containing these minima and maxima and the extremal indices.

Extrema: Retrieve minima and maxima of data values along with their partition ID and indices.

ObjectStatistics: Get information about the number of blocks, vertices and more

PrintMetaData: Get more elaborated information such as the object types and the number of ghosts cells

7. Basic tutorials

This chapter is a hands-on session in which you learn how to assemble a Vistle pipeline yourself. The following tutorials teach you the basic usage of Vistle, so you can create your own module maps for visualization.

Tutorial 1 demonstrates the very basics and gives detailed explanations. Scalar data obtained from a VTK file is visualized using isosurfaces and cutting planes.

Tutorial 2 is more advanced and less guidelines are provided. There, COVISE data, both in scalar and vector structure, is processed and besides surfaces, also streamlines are computed.

7.1. Tutorial 1

In this tutorial, data in VTK format is processed. Therefore, you need to download and unpack a sample data set from the VTK website: http://www.vtk.org/files/VTKTextbook/Data.tgz. A CT scan of the human heart will be visualized.

Assembling your first module map

1. **Start Vistle**:
   In your terminal, type
   ```
   > vistle
   ```
   and the Vistle GUI will show up.
2. **Start COVER:**
   From the Module Browser on the right hand side, add a *COVER* module to the workflow by clicking on *COVER* and dragging it to the Workflow Area. A module, rendered as a turquoise box, will appear there. Also, a second window should show up: COVER.
3. **Import data:** In the Vistle GUI, add the module *ReadVTK* from the Module Browser to the Workflow Area. After the module showed up there, click on it to find the Module Parameters in the frame to the right. There, click on the folder symbol next to *filename.* A window showing the file browser will show up. First, in the bottom, change the option *Files of type* to *Legacy VTK files.* Then navigate to directory where you saved and extracted the VTK data. Inside the directory, select *Heart.vtk.* Then click *Open.* If everything works fine, you should now be able to change the parameter *point field 0* to *scalars.*
4. **Process data:** Add an *IsoSurface* module from the list. Again, click on the module in the Workflow Area to see the parameters. There, change the *isovalue* to *90.*

   Note: If you hover over the parameter names (left column) you can get information on the parameter.
5. **Connect modules:** Link the modules by clicking and dragging the mouse from an output port to an input port. In detail, connect the second output port of *ReadVTK* with the first input port of *IsoSurface;* and the output of the latter with *COVER*.
6. **Execute the pipeline** by clicking on *Execute* in the Toolbar of the Vistle window.

   Change to the COVER window. You will see some grey faces. To view the rendering from a better perspective:
7. Click on ***View all*** in the Toolbar.

   You should now see a grey object, a heart.

Manipulation

Still in COVER, you can interact with the scene using your mouse to change the view.

Switch back to the Vistle window, then click on the *IsoSurface* module. In the Module Parameters, you can change the *isovalue* or choose to compute the surface based on a point rather than a given value (*point or value*). Adjust some of the parameters and execute the pipeline again, to see how they affect your scene.

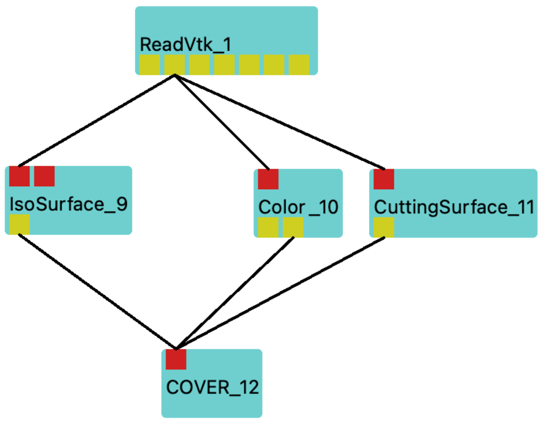
Colorize objects by data

Now, we will define a plane inside the object on which we would like to examine the data. Therefore, we will convert the data values into color values.

8. **Add** a *CuttingSurface* module and a *Color* module to the map.
9. **Link** the modules: click on the central output port of *ReadVTK* and drag the mouse to the input port of Cutting*Surface.* Then connect the output port of the latter with the input port of *COVER*. Also, connect the central port of the reader with the input port of *Color;* and the left output port of *Color* with *COVER.*
10. **Execute** the pipeline and switch to the COVER window.
    You will notice, that no new object is visible in the scene. This is because we haven't set a location for the cutting surface yet and default values are used instead (which might lie outside the object). Go to the Module Parameters to check the values. The cutting plane is define by a point on the plane (*point)* and the plane's normal (*vertex*). You also have the option to create other shapes of cutting surfaces *(option)* such as a sphere or a cylinder. For now, just set *point* to (*1,1,1*).

Interaction

11. To simplify the positioning of the cutting plane, one can also interact with COVER. Therefore, change to the COVER window and in the Toolbar click *Vistle* → *CuttingSurface* → *Pick Interactor.* Next to the object, the interactor will appear as an arrow. Use the mouse on its bottom for translation or on its tip for rotation. Drag the interactor somewhere inside the object to see the colored cutting plane. To see the coordinates of the current position, go back to the Module Parameters in the Vistle GUI.



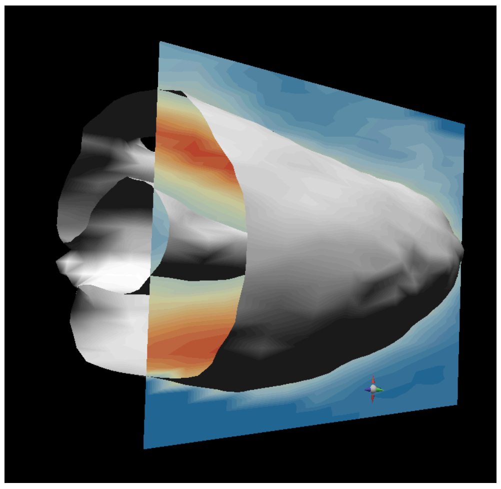


Figure 10: Module Map (left) and rendering (right) for the VTK data set

7.2. Tutorial 2

In the following example, we will assemble the visualization of a channel simulation to investigate the flow inside the channel. We make use of streamlines, isosurfaces and domain surfaces to do so. If you need help or want to validate your setup, you can find a solution in the *vistle* directory : example/tiny-covise.vsl.

Creating a new map

1. Open Vistle or create a new, empty workspace
2. Import data from a COVISE file: Add a *ReadCoviseDirectory* module and set the directory path in the parameters to *covise/share/covise/example-data/tutorial* which is located in your COVISE directory

   We would like to visualize geometry and pressure data: Select *tiny_geo* as *grid* and *tiny_p* for *field0.*
3. To frame the simulated domain: Create a *BoundingBox* from the *grid* available at one of the output ports of the reader.

   Also, create a *DomainSurface* based on *grid* to show the geometry of the object.

   Add a *COVER* module and connect it to the other modules

   Don't forget to check if everything is working as expected from time to time, by executing the pipeline and taking a look at the rendered scene in COVER. There should be the domain surface of the channel in grey and a wire box enclosing it.
4. Use the data of *field0* to compute Is*oSurfaces* for an *isovalue* of *0.0,* by connecting the correct port of the reader module to an *IsoSurface* module
5. Highlight the isosurface by colorization (you can use the *ColorAttribute* module to assign a uniform color by adding it between *IsoSurface* and *COVER*)

   After execution of the pipeline, you need to zoom to the inside of the object to see the isosurface. Alternatively, you can cut off some sections of the domain surface, as described in the following.

Cutting objects

6. Remove parts of the domain to get a better view inside the channel: Replace the connection between *DomainSurface* and *COVER* by a *CutGeometry* module
   Set its point and vertex parameters to *(0,0,0.1)* and *(0,0,1)*, respectively. And execute the pipeline.

   This will create a cut perpendicular to the z-Axis, removing the upper part of the channel.
7. Change the location of the cut. In the COVER window, there is a VR Menu. Select *Vistle → CutGeometry → PickInteractor.*
   An arrow-like interactor will appear in the scene. Use the mouse on its bottom to translate or on its tip to rotate the cutting plane
8. Proceed in a similar way, to add a *CuttingSurface,* which shows the data on a plane. Therefore, add a *CuttingSurface* and connect the input port to *field0.* Then add and connect *Color* in succession. Connect the left output of *Color* with *COVER.* Make use of the *PickInteractor* to create a surface across the channel.

Computing streamlines

9. Also, read velocity data form the file: In the parameters of *ReadCoviseDirectory,* change *field1* to *tiny_velocity*
10. Compute, colorize and thicken streamlines: Add a *Tracer* and connect it to the port *field1* of the reader. Starting from the first port (*data)* of the *Tracer,* add subsequently *Color, Tubes* and *ToTriangles.* Link the first port of each of them with the respective following module (for *ToTriangles,* this is *COVER*)

    Adjust the parameters as follows:

    Tubes: **radius** : 0.005
    Tracer: **no startp**: 12

    Execute. Switch to COVER, and activate the *PickInteractor* for *Tracer* from the VR Menu (*Vistle → Tracer → PickInteractor).* Grab the interactor and move it somewhere inside the object. You should now see the streamlines.

An example of the final rendering is shown in Figure 11. Note that colors and shapes can vary, depending on parameters and position of interactors.

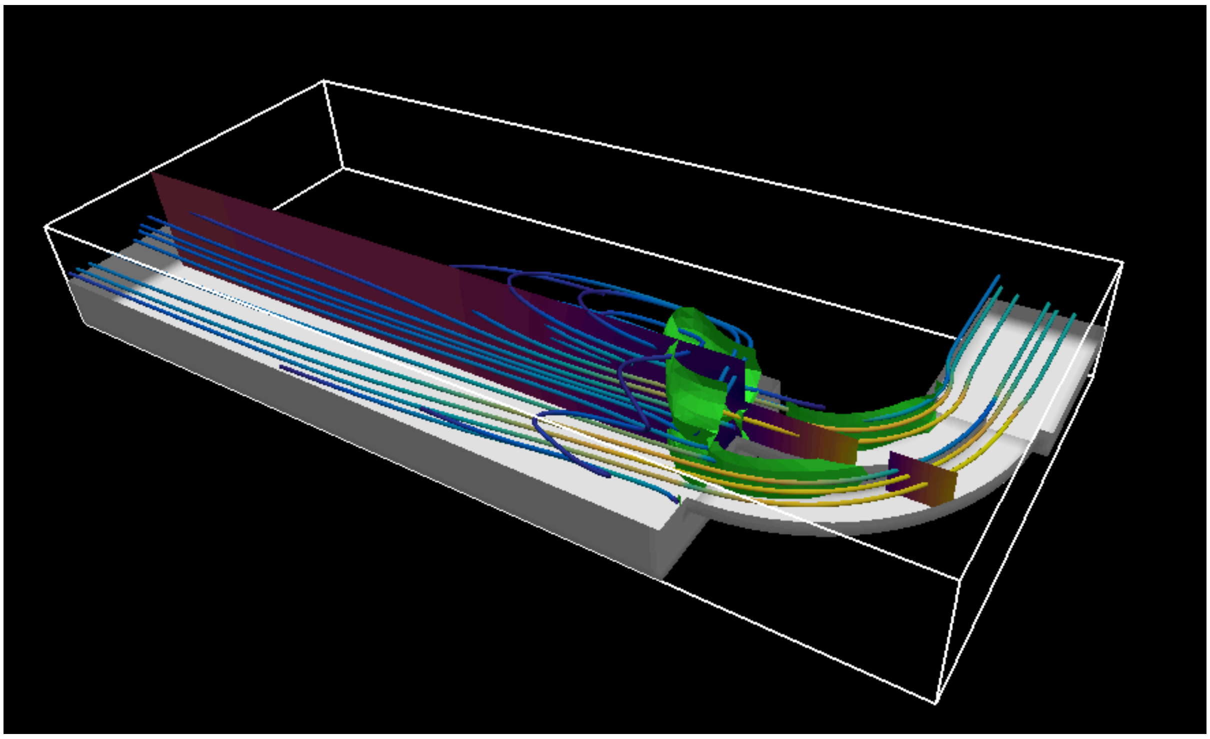


Figure 11: Rendering of the channel flow

8. Further links

More details on Vistle, its functionalities and architecture, can be found on the Vistle website:

https://vistle.io

Many Vistle modules and features also exist in COVISE. For more information, take a look at COVISE documentation and training material, available here:

https://www.hlrs.de/solutions-services/service-portfolio/visualization/covise/documentation/

There, you can also find documentation for COVER.