COVISE

Tutorial

July 2005

Title:
COVISE Tutorial
A short introduction in working with COVISE
January 26, 2022

Authors: Daniela Rainer

Contents

Preface

This document is a short introduction to working with COVISE. It is primarily a tutorial for COVISE novices. It doesn't cover advanced topics such as the development of new application modules or the installation and configuration process. We assume that you have a running COVISE on your machine. For installation guide read the files *README* and *INSTALL.TXT* which come with your COVISE distribution. For developing new application modules read the *COVISE Programming Guide*.

COVISE is a Collaborative Visualization and Simulation Environment developed at the Computing Center of the University of Stuttgart. It is an extendable distributed software environment to integrate supercomputer based simulations, postprocessing, and visualization functionality with cooperative working in a seamless manner.

The tutorial contains the five chapters:

In *Chapter 1 Starting COVISE* you learn how to initiate a single user session and load a saved session. The functionality of some basic modules (RWCovise, DomainSurface, CuttingSurface) is explained.

Chapter 2 Using the Inventor Renderer gives a short introduction to the typical work with the Renderer.

Chapter 3 Working with Modules covers module ports and module parameters.

Chapter 4 Analysis of 3D Data describes the basic steps in analyzing complex 3D data with the general COVISE modules. (This chapter has been reworked to make the user familiar with the 'Complex Modules' for building maps quicker and easier.)

Chapter 5 Advanced Topics covers distributed computing and multi user sessions.

This tutorial uses the following conventions:

File names, path names, and environment variables are printed *italic*, for example

$(COVISEDIR)/net/general/examples/ReadStar.net

Program names and COVISE module names are printed **bold**, for example the command to start COVISE: **covise**, or the name of the module that computes a cutting plane: **CuttingSurface**.

Output to the console window which COVISE produces is printed in fixed space font like

```
Starting COVISE...
Please be patient...
```

Some sections are included in a frame, indicating troubleshooting information.

1 Starting COVISE

1.1 Introduction

After having read this chapter you will be familiar with:

- how to start COVISE
- the parts of the MapEditor
- how to load a map

COVISE is a toolkit to integrate several stages of a scientific or technical application such as simulations, reading of data, postprocessing and visualization. Each step is implemented as a module. Connecting these module forms a data flow network that can be executed. The user interacts with COVISE and its modules through a Motif based user interface.

1.2 Starting COVISE

After the installation the path to the covise executable is appended to the environment variable $PATH. To start COVISE type **covise** in a cshell or tcshell window.

This initiates the following processes: the controller which is responsible for message handling, the request broker, responsible for data handling, and the main user interface called MapEditor.

If the command **covise** is not found make sure you use a tcshell or a cshell. Otherwise have a look at the files README and INSTALL.TXT and check the environment variable COVISE_PATH.
During the starting phase you might get the message

```
Timelimit in accept exceeded
```

Edit the file covise.config in the covise directory and increase the timelimit for your machine. You find more information about *covise.config* in Chapter 2 in the *User's Guide*.

COVISE can also be started with parameters. Typing `covise help` will show you the syntax.

The following 2 start options may be useful e.g. for demos with COVISE:

- **-i** start with MapEditor as icon
- **-e** execute immediately after loading

1.3 The Mapeditor

After the starting phase the MapEditor window (with Visual Programming as default) appears.

The MapEditor window consists of four main parts:

- the Menu Bar

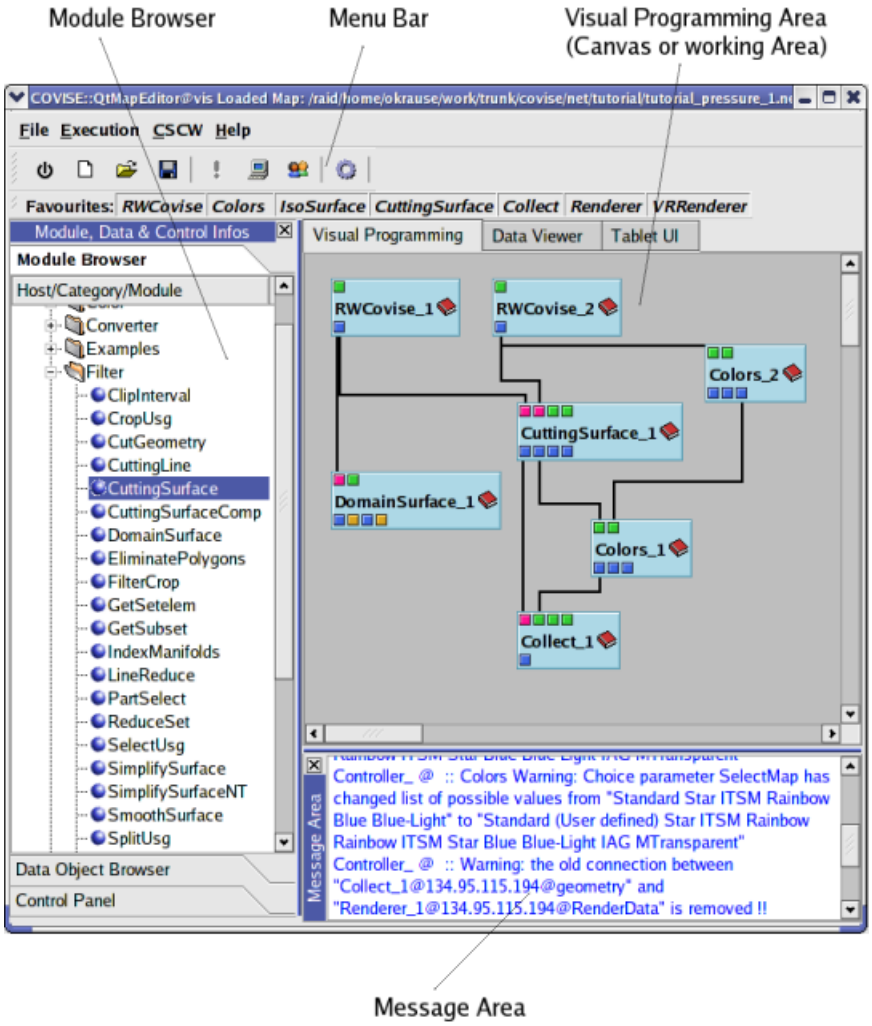

Figure 1.1: The Mapeditor

- the Tool Bar
- one of three index cards according to the selected function
  - Visual Programming - default: to build a map
  - COVISE Objects - to look at the data structures involved
  - Control Panel - to modify parameters
- the Message Area

A Chat Line - used for communication together with the Message Area - appears in case of two or more users only.

The Visual Programming index card contains the Module Browser and the Visual Programming Area (canvas or working area). The Menu Bar contains an item File. You can have a new file, in which you create a data flow network, you can save a file or you can open a file that contains a saved session. Such a file is called a map. It contains a list of the modules in the map and information about the connections.

Creating a new module network (map) is discussed in chapter 3. In this chapter you learn how to load and execute a saved map. COVISE comes with some example maps and the maps described in this tutorial. They are located in the following directories:
$COVISEDIR/net/general/example
$COVISEDIR/net/general/tutorial

1.4 Loading a Map

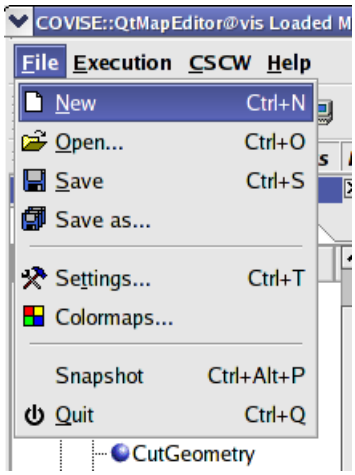

Figure 1.2: Menu Item File

The menu item File Open pops up a file browser (Figure 1.2). Select the file *covise/net/tutorial/tutorial_pressure_1.net*.

The loaded map appears in the working area. Each module is represented by an icon. A module has input data ports, represented by blue buttons on the top edge of the module icon and output data ports represented by blue buttons on the bottom of the icon. Green buttons mean optional data ports. Module icons are described in more detail in chapter 3.

When the Renderer module is started as part of the module network, its window is opened. Initially it contains only three coordinate axis. This is explained in more detail in chapter 2.

Now execute the network by selecting Execute in the Execution menu. One module icon after the other shows a red frame which indicates that it operates. Finally geometry objects appear in the render window (Figure 1.3).

Have a look at Figure 1.1: the modules on the top are named **RWCovise**. Their task is to read in data files in the COVISE data format. The files are the output of a flow simulation. In a flow simulation the geometric domain occupied by the fluid is covered by a grid of a certain type. Such a grid consists of grid points together with their connectivity information. The simulation results typically contain scalar or vector data such as pressure, temperature or velocity attached to the grid points. Grids and attached data are handled as separate objects in COVISE.

The left **RWCovise** module reads the grid file tiny_geo.covise. You select the grid via a file browser that pops up automatically.

Below **RWCovise** you find the module **DomainSurface**. Out of the whole 3D grid it extracts the outer surface or bounding lines of the grid. The **DomainSurface** module is directly connected to the Renderer module, which displays the results of **DomainSurface**. Figure 1.3 shows the content of the Renderer module window. The white lines are the bounding edges of the geometry. The displayed content is a flow channel with two inlets.

The second **RWCovise** module reads in the pressure distribution tiny_p.covise. **CuttingSurface** then computes a cutting plane in the pressure data. The module **Colors** converts the scalar data values to colors. The grid of the cutting plane and the colors are combined into a geometry object by the **Collect** module. **Collect** is directly connected to the **Renderer**, which displays the resulting colored plane.

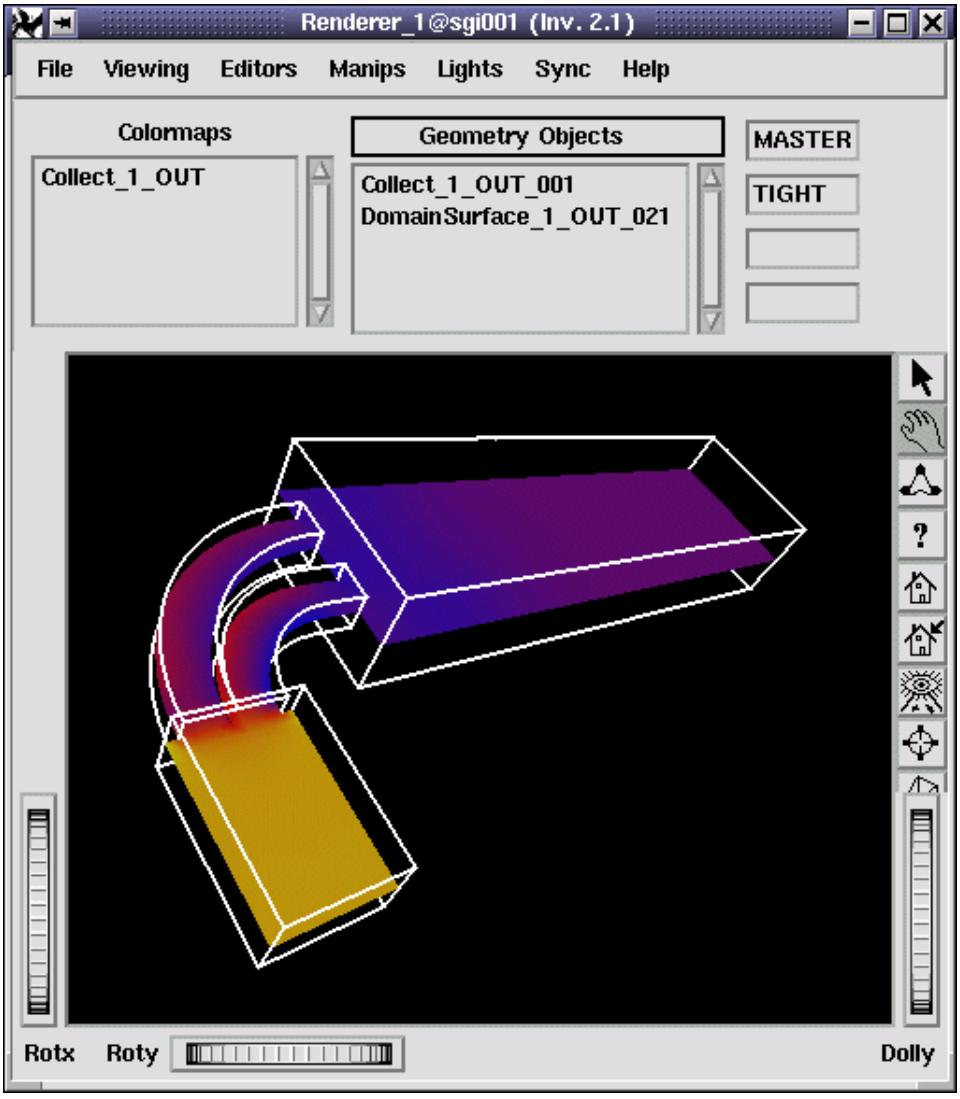

Figure 1.3: Geometry Objects in the Renderer

2 Using the OpenInventor Renderer

2.1 Introduction

After having read this chapter you will be familiar with:

- Renderer modules in general
- the user interface of the Renderer
- Mouse interaction modes

In the previous chapter you have read about some basic steps in working with COVISE. You have already learned how to load a prepared module network. In this chapter we introduce some functionalities of the OpenInventor **Renderer** and their main usages. COVISE comprises other render modules as well but the OpenInventor Renderer can be seen as the default desktop renderer. OpenInventor is a high level object oriented graphics toolkit developed by Silicon Graphics. It uses OpenGL, the 3D graphics standard, as its rendering interface.

You can regard the **Renderer** as a movable window to a virtual world. The displayed scenario contains objects which are typically visual representations of simulation data. Depending on the selected visualization method you can have different visual representations for the same dataset. This allows to explore the various aspects and characteristics of a dataset and detect features otherwise not seen. In addition to the mapping of data to visual representations supportive geometries, such as bounding surfaces or reference geometries of machineries provided via CAD models can also be included in combination with the data visualizations. A typical example would be the visualization of a pressure distribution in a combustion chamber of a car engine. Such scenarios can be further examined with all the possibilities that the Renderer provides.

2.2 The Render Module

Figure 2.1: The Renderer Module Icon

At the end of a module network processing chain there is an output module, which in most cases will be a render module to visualize resulting data. Other output modules are write modules for saving results to disk or a special plot module for xy- charting. In the **MapEditor** working area those output modules are typically placed at the bottom of the map. Output modules normally have input ports only (blue rectangle on the left top of the render module in Figure 2.1).

The module selection and the usage of ports to build module networks will be described in the following chapters

2.3 The User Interface

In Figure 2.2 you can see the OpenInventor Renderer as it will appear on your desktop after you have inserted the Renderer module in the module placing area.

The renderer window consists of three major parts:

- the menubar
- the information area
- the viewer area.

The menubar will only be explained shortly. It provides functionalities such as saving the current scene to a file or printing the scene. You can also adjust viewing parameters or set material and light properties using the menus. The information area provides information about the current state of the renderer. Beside some text fields for collaboration status there are two list boxes. The ColorMaps list box contains all the colormaps used in the visualization pipeline. The GeometryObjects list box contains all the geometry objects currently displayed. The viewer area holds the main viewer widget where the objects are displayed. The viewer area contains also other user interface elements.

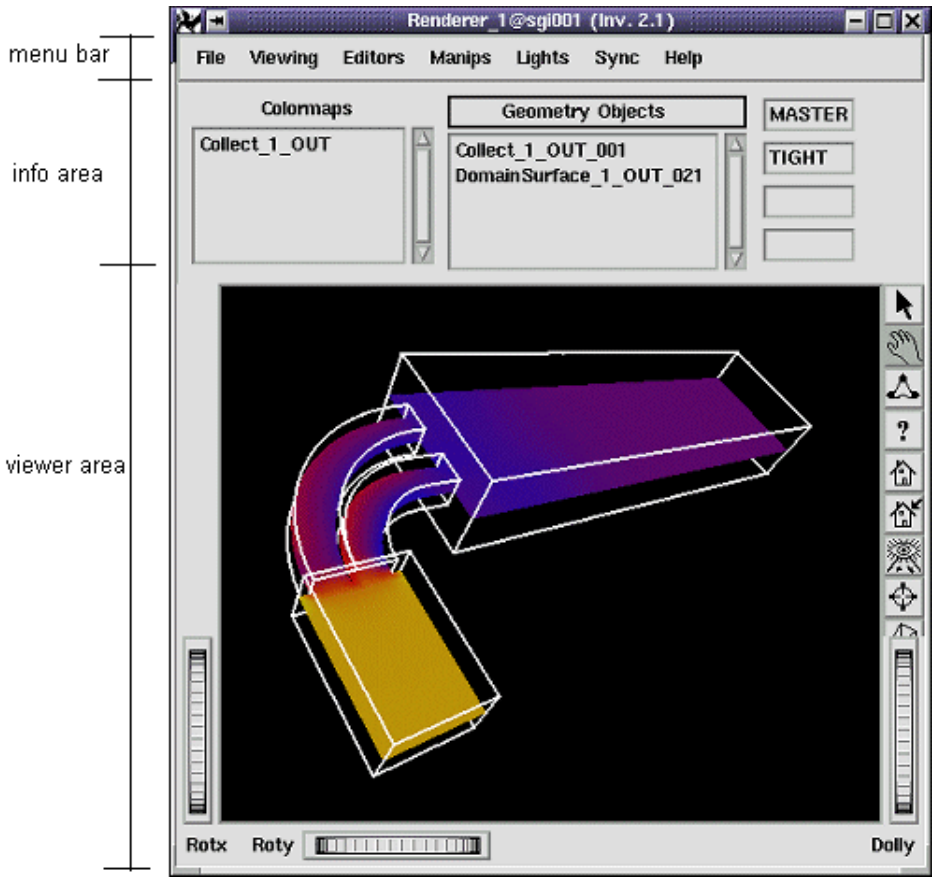

Figure 2.2: The Renderer User Interface

Figure 2.3 shows the toolbar at the right side of the viewer area in greater detail. One feature of the toolbar is the home position. It represents the special camera position and orientation of an initial view. By pressing the home position button in the toolbar the camera will switch to the position and orientation of the viewers home position. With the set home position button you can set these values at any time.

Another feature of the toolbar is the view all button. Pressing this button causes the **Renderer** to adjust the camera position and orientation to view all objects.

Figure 2.3: The Toolbar

Pressing the seek mode button changes the cursor to the seek mode symbol. Within this mode you can click on any part of your geometry. This will set the camera position in a way, that the selected point will be in the center of the window with a view orientation orthogonal to the object.

The button on the bottom of the toolbar selects a projection mode. By pressing this button you can switch between a perspective and an orthogonal projection. The default projection mode is the perspective one.

Arranged at the bottom of the viewer area are some user interface elements for rotating and zooming the camera. On the left side are two thumb wheels to rotate the camera about the x (Rotx) and the y (Roty) axis.

On the right side is a slider (Zoom) and a thumb wheel (Dolly) to zoom the camera.

2.4 Mouse Interaction Mode

The OpenInventor Renderer knows two fundamental interaction modes of the mouse:

The view mode transforms mouse interactions in viewpoint changes and the pick mode is used to change the properties of geometric objects. An active view mode is indicated by an active view mode button in the toolbar on the right side of the window. Also the cursor changes to a hand symbol. Within this mode you can rotate the whole scene about its center using the left mouse button. Move the mouse pointer over the geometry, press the button and move the mouse in that direction you want to rotate to. By releasing the button without moving the mouse simultaneously, the scene will be rendered from the new viewpoint. When you drag the mouse and release the button while you are moving, the scene will spin permanently until you click into to the viewer area. This continuation of the last transformation operation after mouse button release only works with rotations. Using the middle mouse button you can pan the scene. Using the combined left and middle button you can zoom the viewpoint in the same manner. Thus, dragging the mouse to the bottom of the window with the left and middle mouse button pressed, you will zoom into the scene. Dragging to the top will zoom out.

The pick mode is indicated by an active pick button in the toolbar and the cursor changes to an arrow. In this mode you can select individual geometries for further manipulations. By clicking with the left mouse button on a geometry object you select the geometry. The selection will be indicated by a bounding box (a red wireframed box) around that object. For the selection of multiple objects hold down the shift key while selecting the different objects. To remove the selection of the object just select the object again. To remove the whole selection click on any free part of the viewer widget. This working method is just the same you already know from popular drawing programs.

By clicking with the right mouse button into the viewer area in any of the two interaction modes a popup menu appears. This popup menu will not be further explained here. It provides many functionalities you find on the menubar too.

3 Working with Modules

3.1 Introduction

After having read this chapter you will be familiar with:

- Module Categories
- Input and Output Ports
- COVISE Data Object Types
- Parameters
- Module actions

3.2 Categories

COVISE separates the processing steps of an application into separate modules. A module realizes a certain functionality. Modules are grouped into categories. The general categories are:

- IO/Modules
- Simulation
- Mapper
- Color
- Tracer
- Filter
- Tools
- Renderer

These categories reflect the typical steps in the analysis of 3D data e.g. coming from flow simulations.

Categories and modules within them are shown in the Module Browser in the MapEditor window (see Figure 3.1).

If the hosts are colored (configurable in the file covise.config), modules running on this host appear in the same color. To start a module click on the category name and then drag the module name to the working area. To remove the module see the section *Actions with Modules* later in this chapter.

Filter modules typically extract data from larger datasets. The **CuttingSurface** module interpolates on a plane or cylinder surface data coming from a 3D data volume.

The computation of an isosurface belongs to the category Mapper, because data is mapped to a geometric object, that can be rendered.

The category Tools contains many useful functionalities such as the module **DomainSurface** that computes the outer surface of a computational grid or **CutGeometry**, which cuts off portions of polygonal geometric objects like the ones created by **DomainSurface**.

Modules in the category Renderer display data. The 3D Inventor based **Renderer** belongs to this category as well as the 2D Plot module.

3.3 Input and Output Ports

Figure 3.2 shows a typical module icon. The violet and green ports at the top edge are input data ports, the blue and yellow ports at the bottom edge are output data ports. Violet input ports mean that the port has to be connected to another port, green ones mean that the connection is optional.

Be aware that certain green input ports can become violet - and thus have to be connected - when the corresponding output port is connected.

Connection lines between the ports indicate the flow of data objects between the modules. You connect modules by clicking with the left mouse button onto the ports. To remove a connection between modules, double-click on the connection line.

3.4 Data Types

Data is exchanged between modules through COVISE data objects. If both modules which need to exchange data objects run on the same machine, the data objects are located in shared memory and thus allow an efficient exchange of data. If the modules run on different machines, the data objects are exchanged through tcp socket connections.

Input ports of one module and output ports of another module can only be connected if the data types are compatible. The most important data types are:

- Unstructured/Structured/Uniform/Rectilinear Grids
- Unstructured/Structured Scalar Data
- Unstructured/Structured Vector Data
- Lines, Points, Triangle Strips, Polygons
- Normals
- Colors
- Geometry
- Sets

Geometry is a container to hold geometric primitives (lines, points, triangle strips, polygons), colors, and normals. A set is used as a container for other objects. Sets can be used to group objects or to hold different time steps of an object.

If you click on the port with the right mouse button, the data type is displayed in the message window. Figure 3.3 shows the result of clicking on the output port of the RWCovise module.

3.5 Parameters

You see the parameter list when you click with the left mouse button on the book icon. The module info window pops up (Figure 3.4).

The first page of this notebook contains the parameters. Use the button with the parameter names on the left to attach interactors to this parameter which appear in the control panel. The next column contains the type of the parameters. After this you see an area which allows the user to change the parameter values. The content of each line depends on the parameter type.

If a module has a filebrowser as parameter, a filebrowser window automatically pops up as soon as the module is started.

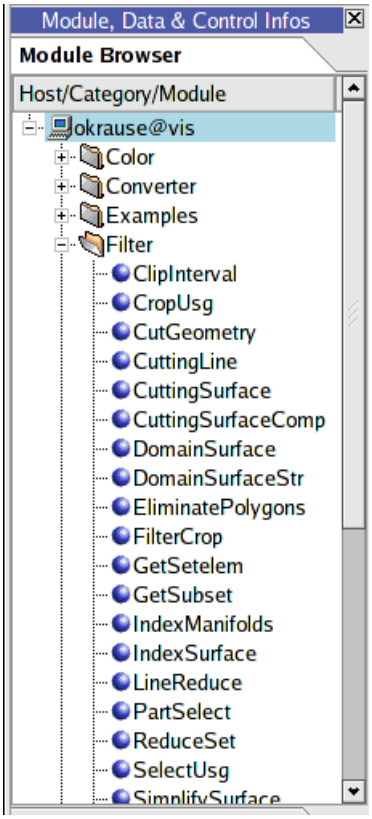

Figure 3.1: Category and Module Lists

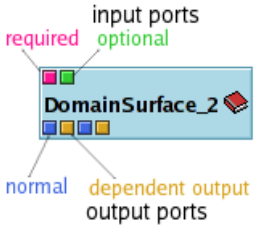

Figure 3.2: Module Icon

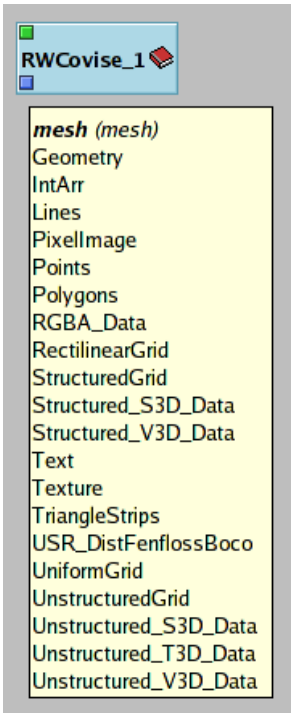

Figure 3.3: Port Data Type

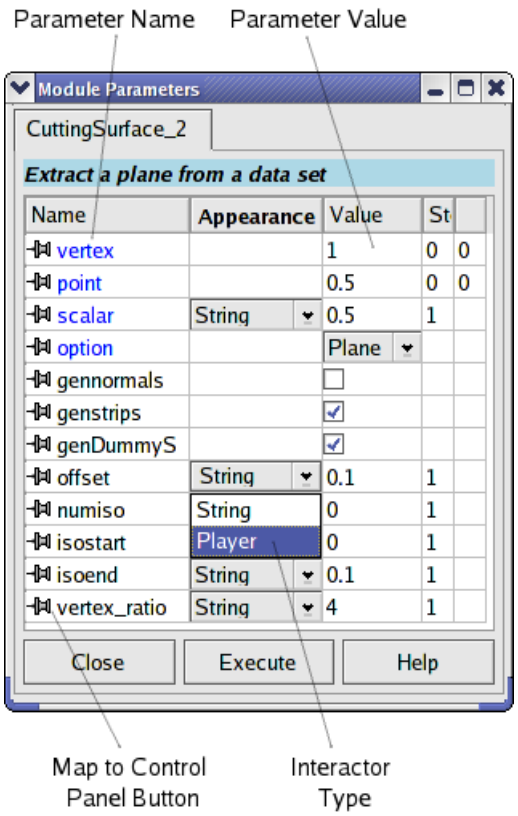

Figure 3.4: Module Info Window

When you add an interactor to the control panel, you usually have a choice between different types of interactors. To select the interactor click on the "Map to Control Panel Button" with the right mouse button. A small menu window pops up.

3.6 Module Actions

Clicking with the right mouse button on the module icon pops up a menu with the module actions (Figure 3.5)

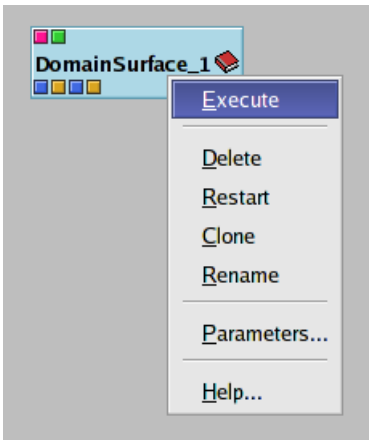

Figure 3.5: Module Actions

The most important actions are "Execute" and "Delete".

- Execute

executes the module pipeline starting from this module. It is typically more efficient to execute only a part of the module network after having changed some parameters instead of executing the whole network. I/O-modules often need a lot of time to read in large data files which is not necessary if you have just modified a module parameter further down the processing chain.

- Delete

To remove a module from the working area choose this item.

- Rename

To rename a module (can also be used to label a module group).

- Move

With this action an user can move this module to another host for execution. This is only possible, if an additional host was added to the session via "Add Host/Partner". All parameters and connection lines are moved too.

- Copy

With this action an user can copy this module to another host for execution. This is only possible, if an additional host was added to the session via "Add Host/Partner". The current parameters values and connection lines are copied too.

- Help

If you click on this item the online module documentation is loaded into the netscape browser.

All actions can also be used for a group of selected modules.

4 Analysis of 3D Data

Note for the experienced user:

This chapter has been reworked to demonstrate the new features for building maps quicker and easier. COVISE provides **Complex Modules** like TracerComp, CuttingSurfaceComp, or IsoSurfaceComp which include modules Colors, Collect etc.

For details please see Online Help, User's Guide, and Module Reference Guide.

4.1 Introduction

In this chapter you learn how to

- visualize the computational grid
- visualize vector data
- visualize scalar data

This section gives a very short overview how to use COVISE modules for the analysis of simulation data.

In the example the bounding geometry is a channel with two inlets. The grid was created with PROSTAR, the pre- and postprocessing program for STAR and the simulation was performed with STAR. In a previous COVISE session the grid and the simulation result files have been converted to COVISE format, as this format allows very compact files and fast reading by COVISE.

In the directory *covise/share/covise/example-data/tutorial* you find the files

- *tiny_geo.covise*
- *tiny_velocity.covise*
- *tiny_p.covise*
- *tiny_te.covise*
- *tiny_vis.covise*

tiny_geo.covise contains an unstructured grid while the other files contain data.

The following tutorial module pipelines (also called maps or nets) are located in the directory *covise/net/general/tutorial/*. In the next section only the most important module ports and parameters are explained. For detailed information about the modules a documentation in html format is available.

4.2 Visualizing the Computational Grid

Assuming we don't know what the files contain it may be useful to first make the simulation grid visible. The following modules have this or related functionality:

ShowGrid generates lines, points, hulls or bounding boxes for structured/rectilinear/uniform grids.

DomainSurface generates the outer surface of an unstructured grid.

SimplifySurface reduces the number of polygons representing a surface.

tutorial_grid_1.net

Start COVISE and open a new session. Choose the file *tutorial_grid_1.net* and execute the module network. Examining the module network, you see, that **RWCovise** is from the category IO-Module. It imports the data file *tiny_geo.covise* and generates an output data object containing an unstructured grid. The file name is specified as a file browser parameter.

The module **DomainSurface** from the category tools determines the outer surface of the grid or the edges of the outer surface. **DomainSurface** receives the unstructured grid as input data object. It produces polygons representing the outer surface which are available at the first output port.

This port is connected with the module **Renderer** which receives the poygons as input data objects. Figure 4.1 shows the map and the results displayed in the renderer window.

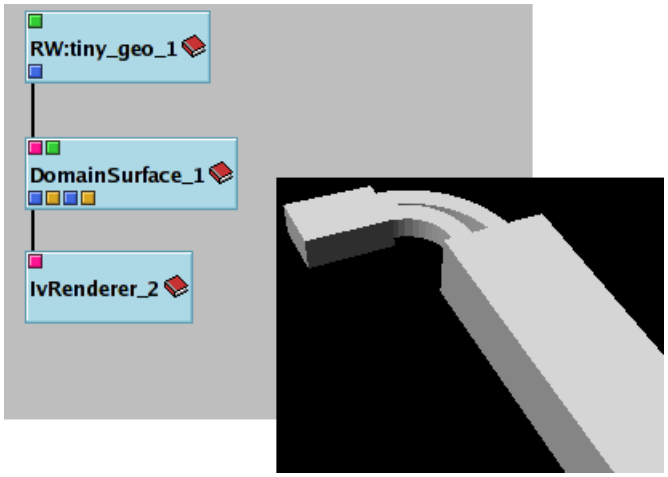

Figure 4.1: tutorial_grid_1

tutorial_grid_2.net

The disadvantage of the representation in Fig. 4.1 is that you can't see inside the channel.

Load the map *tutorial_grid_2.net*. The difference to the first map is that the third output port of DomainSurface is connected to the Renderer. This port outputs data objects which contain the edges of the outer surface.

tutorial_grid_3.net

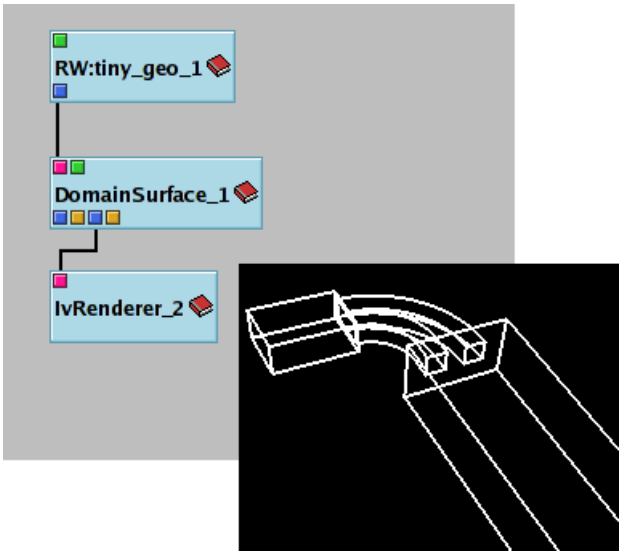

Figure 4.2: tutorial_grid_2

Another possibility to look inside the channel is to cut away parts of the geometry. This can be done by the module **CutGeometry**. Load the map *tutorial_grid_3.net*. The module **CutGeometry** cuts the geometry with a plane.

The plane is defined using the parameters scalar and vertex. Vertex defines the orientation of the plane and also the cutting direction. Scalar positions the plane along its normal direction. In this example vertex is [0.0 0.0 -0.1] and scalar is -0.12. This defines a plane which cuts away the upper surface of the channel.

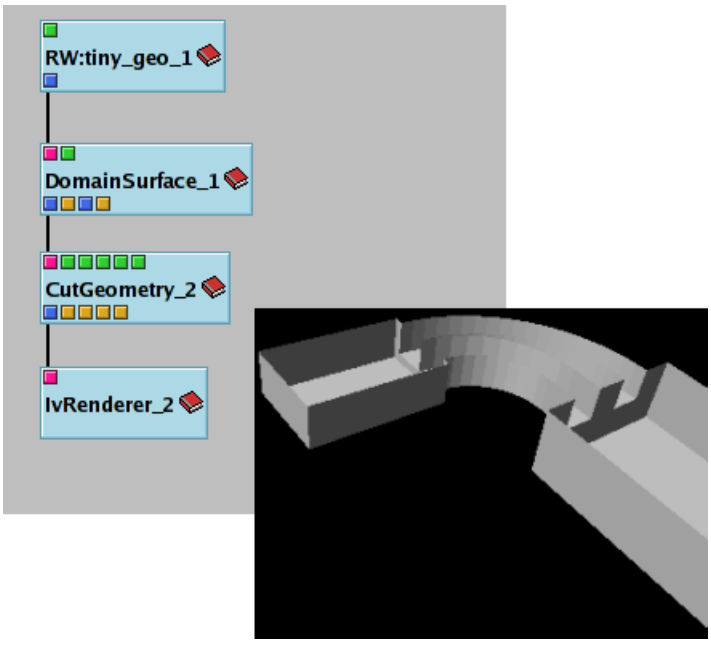

Figure 4.3: tutorial_grid_3

tutorial_grid_4.net

If a surface contains too many polygons, it can't be displayed with an acceptable frame rate (at least 12 frames per second).

For interactive handling it is necessary to reduce the number of polygons, which is done by the module **SimplifySurface** (see *tutorial_grid_4.net*).

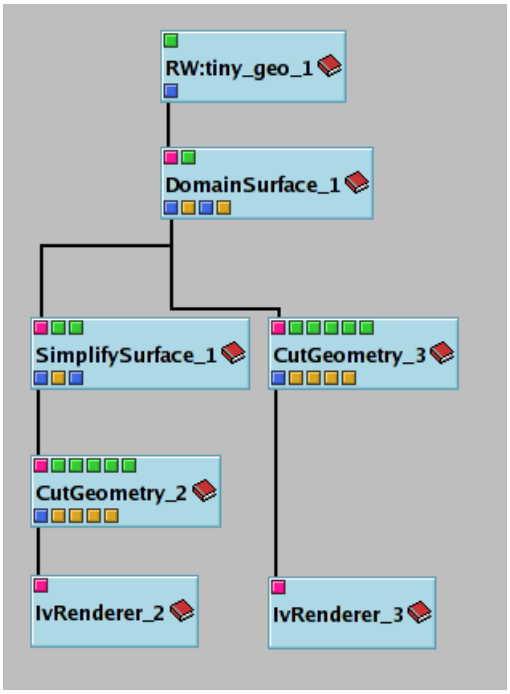

Figure 4.4: tutorial_grid_4.net

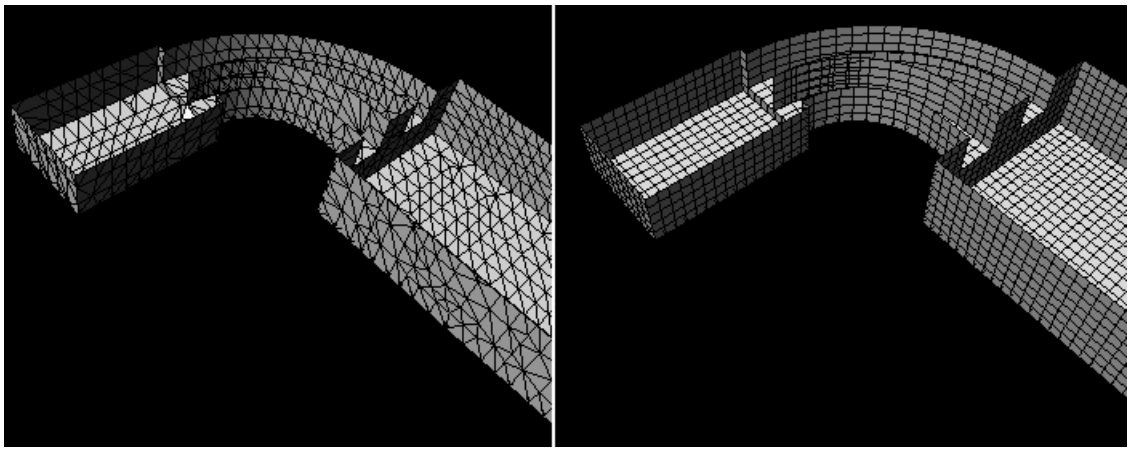

Figure 4.5: Original and Reduced Surface

Operating instructions for this example (to see how SimplifySurface works):

- Use the "Discrete Mesh" representation (click with the right mouse button on the renderer window and choose DrawStyle >> DiscreteMesh.

- Parameter "strategy": select "Vertex Removal" (click on the SimplifySurface icon to get the Module Information with the parameters)
- Parameter "percent": specify the percentage of remaining triangles after SimplifySurface.

4.3 Visualizing Vector Data

Typical vector data resulting from a simulation is the velocity field. The following modules have to deal with vector data:

TracerComp

- computes streamlines on unstructured grids
- computes streamlines on structured grids
- computes animated particle traces
- contains additional functions to map output data to colors

(Historical Notes:
- In former versions of COVISE these functions had been implemented in 3 different tracers - TracerUsg, TracerStr, and TetraTrace
- If you specify e.g. animated particle traces, TracerComp, on top of standard Tracer, contains functions to show the particles as spheres and map data as colors on that spheres - no need for additional modules Sphere, Colors, Collect)

VectorField (explicitly used in tutorial_vel3.net only - otherwise contained, implicitly,in CuttingSurface-Comp)

computes small lines at each grid point. The direction of the lines represents the direction of the velocity vector and the size the amount of the velocity.

tutorial_vel_1_new.net

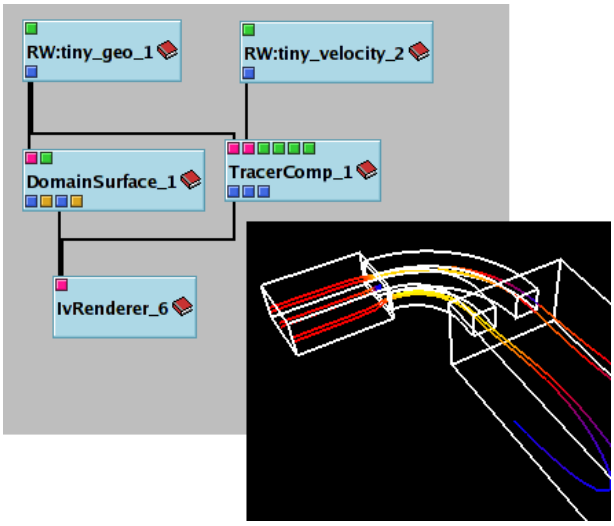

Figure 4.6: tutorial_vel_1_new

Load the map *tutorial_vel_1_new.net*. The second **RWCovise** module reads the data file *tiny_velocity.covise* which contains the velocity field.

The module **TracerComp** computes streamlines (Specify 'Streamlines' as parameter taskType in the Tracer). Its first input port receives the grid object and the second port the data object containing the velocity field.

The starting points of the streamlines and the number of streamlines are specified via module parameters.startpoint1 and startpoint2 define a line. no_of_startpoints defines how many streamlines start on this line. Usually it is difficult to determine the coordinates of the two points, if you don't know the size and position of the grid. In this example we choose [-0.4 0.5 0.02] for startpoint1 and [-0.4 0.3 0.02] for startpoint2 and draw 6 streamlines.

tutorial_vel_2_new.net

The map *tutorial_vel_2_new.net* shows an example of animated particle traces.

The layout of the map is the same as tutorial_vel_1_new.net, but you have to specify 'particle' in the parameter taskType in TracerComp. i The output of **TracerComp** contains sets of points where each set represents the position of the particles at a certain timestep, together with velocity data attached to the points. TracerComp integrates the function to make particles more visible by displaying small spheres instead of points (specify radius as parameter) and to map output data to colors.

As soon as the pipeline is executed, the geometry is displayed in the renderer window. In addition, the renderer contains a slider for selecting the timesteps of the animation. For a continuous animation press the play button.

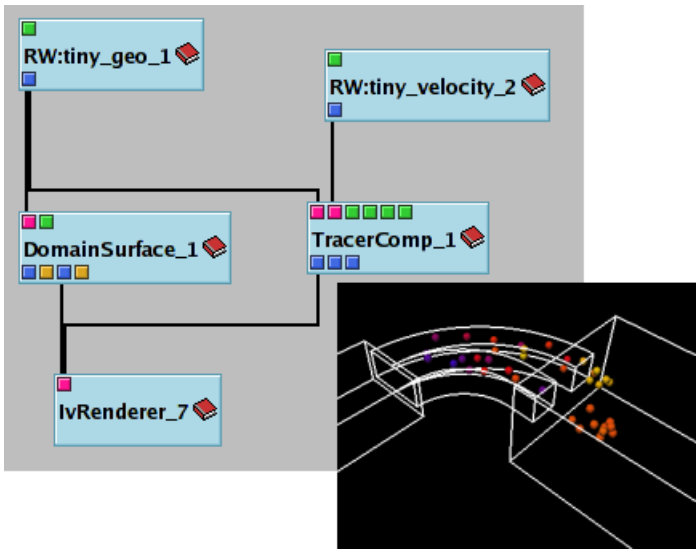

Figure 4.7: tutorial_vel_2_new

tutorial_vel_3.net / tutorial_vel_4.net

The map *tutorial_vel_3.net* is an 'old-fashioned' map - have a look at it for information only and don't use it as an example; it shows a flow visualization with small lines attached to the grid points which depict the magnitude and the direction of the respective velocities.

The module **VectorField** needs the grid and the velocity data as input data objects and computes lines

and data attached to the lines as output objects. The data are mapped to colors using **Colors** and **Collect**.

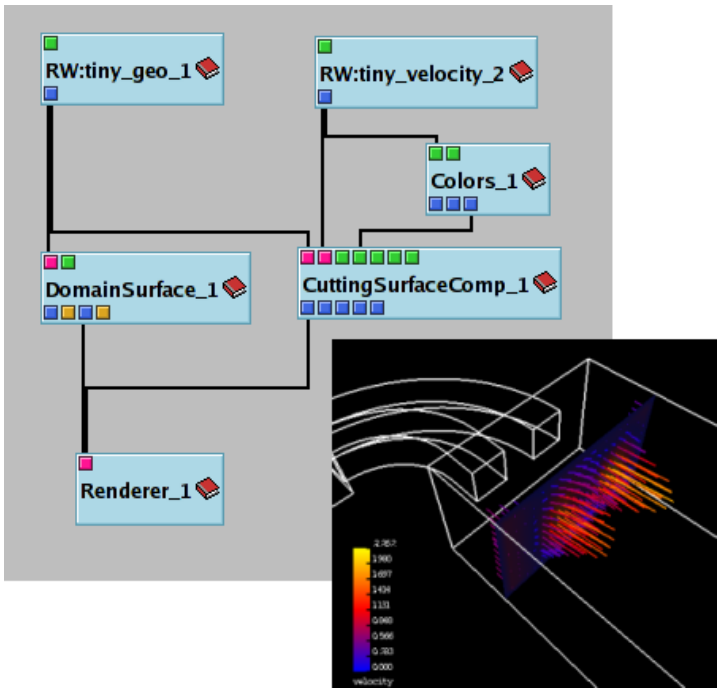

Figure 4.8: tutorial_vel_4_new

Displaying vector dashes at all grid points is not very useful. Instead a **CuttingSurfaceComp** module can extract a 2D subset of the grid of which the grid points are then used to attach the vector dashes (see *tutorial_vel_4.net*). Note that the data attached to the cutting plane is determined by interpolating in the original grid. Since displaying vector data as colored arrows on a cutting surface is a frequently used operation, this function has been integrated into CuttingSurfaceComp (see Fig. 4.8) and is automatically activated in case of vector data! **Colors** is used to select a colormap for the colors on the vector dashes.

4.4 Visualizing Scalar Data

Typical scalar data resulting from a simulation are e.g. pressure or temperature. Modules for visualizing scalar data are:

CuttingSurfaceComp computes a cutting plane, cylinder or sphere in unstructured grids. In addition, it can map scalar data to colors. **IsoSurfaceComp** computes a surface connecting all points in space, which have the same scalar value. In addition, it can map scalar data to colors. **Colors** is used in this example to select a colormap (including minimum, maximum, annotation)

tutorial_pressure_1_new.net

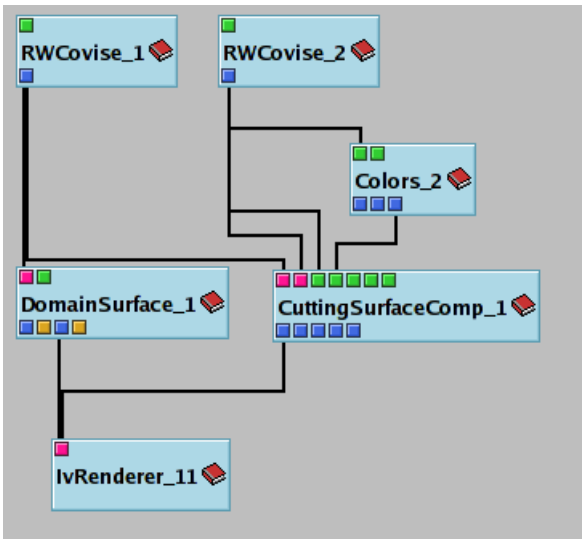

Figure 4.9: tutorial_pressure_1_new

Load the map *tutorial_pressure_1_new.net*. Additional to the **RWCovise** module which reads the grid we have a second **RWCovise** module which reads the pressure data file *tiny_p.covise*.

CuttingSurfaceComp from the category filter then extracts a surface (plane, cylinder, sphere). It needs the grid object and the data object containing scalar data as input and creates polygons or tristrips which form the cuttingsurface and data on the surface as output objects. In addition, the data are mapped to colors and combined into a geometry object together with the polygons/tristrips.

Using the parameter *option* you define the shape of the surface. Map the parameter to the ControlPanel by clicking on the button in front of the parameter. Now you can see that you have the options plane, cylinder and sphere. Using the parameters *vertex* and *scalar* you specify the orientation and position of the surface.

For a plane the parameter *vector* is the normal of the plane and the *scalar* is the distance of the plane from the origin. In this example the normal of the plane is (0 0 1) and the distance from the origin is 0.05.

A scalar parameter such as the distance is defined using three numbers: minimum, maximum and current value. Map the parameter distance to the ControlPanel. The default interactor in the control panel is a VCR type player. You can increase or decrease the distance via the player buttons applying a certain increment which is specified in an extra field. Modify the distance parameter to get an overview of the pressure distribution in the channel.

In the render window you can also see a color bar. Color bars are produced by the modules Colors and ColorEdit (explicitly or implicitly, if the function of Colors is integrated e.g. in the complex module CuttingSurfaceComp). They are only displayed in the render window if they are selected there. Click on the name of the data object to pop it up and click again to hide the colorbar.

Note: The module Colors_2 is used to select the colormap, the minimum and maximum values, and the annotation, e. g. "Pressure" (default: Colors).

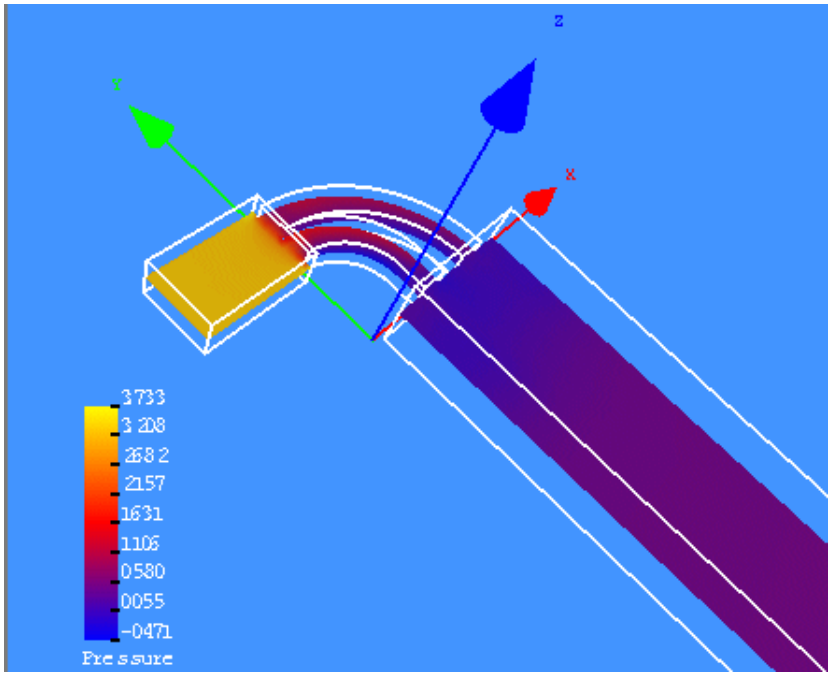

Figure 4.10: tutorial_pressure_1_new: Results in the Renderer

tutorial_pressure_2_new.net

Another method to visualize scalar data are isosurfaces. An isosurface is a surface consisting of points in space, where the scalar data has the isovalue. Load the map *tutorial_pressure_2.net* and execute it. The blue surface connects all grid points where the pressure has the value 0.05.

Have a look at the map in the MapEditor window. Again the first **RWCovise** reads the grid while the second one reads the pressure data.

IsoSurfaceComp needs the grid and the scalar data as input objects. The resulting polygons or tristrips are provided at the first output port and the data on the surface at the second port. The isovalue is defined using the input parameter isovalue. The parameter gennormals enables the generation of normals at each polygon vertex.

The normals are available at the third output port.. The normals are used for the computation of the surface lighting. If you do not provide normals for a surface the renderer computes one normal per polygon face. Normals per vertex result in smoother surfaces than normals per face.

Note: Like CuttingSurfaceComp in tutorial_pressure_1_new.net, IsoSurfaceComp maps the data to colors. The Module Colors_2 is used to select the colormap, the minimum and maximum values, and the annotation, e. g. "Pressure" (default: Colors).

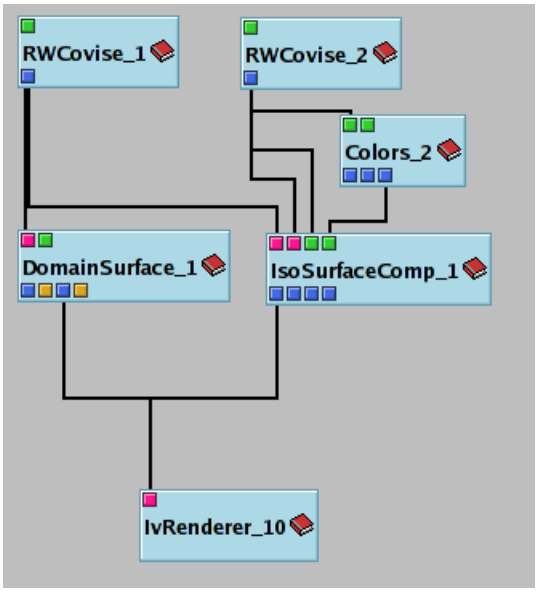

Figure 4.11: tutorial_pressure_2_new

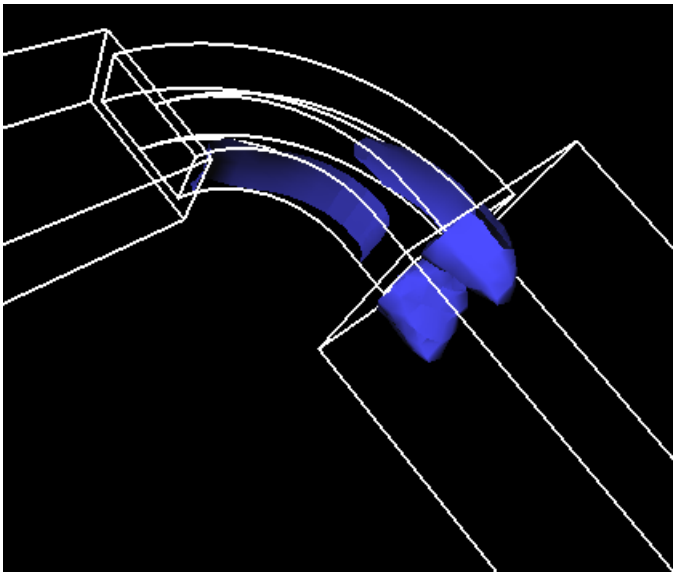

Figure 4.12: tutorial_pressure_2_new: Results in the Renderer

4.5 Summary (Example)

channel_new.net

The example **channel_new.net** below combines some of the features shown before in one map. In addition, it shows how to get the size of your geometry object using the BoundingBox module.

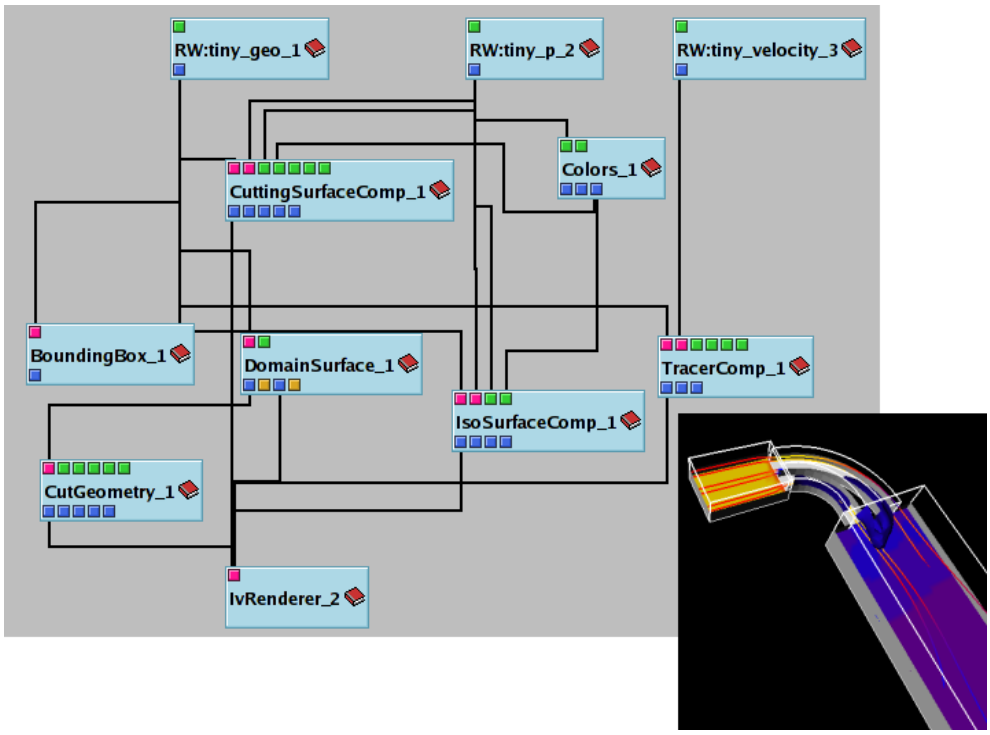

Figure 4.13: channel_new

5 Advanced Topics

5.1 Introduction

After having read this chapter you will be familiar with:

- The Architecture of COVISE
- how to prepare COVISE for distributed and collaborative working
- how to run COVISE across firewalls
- how to include a remote hosts or partner into a session
- Starting modules on remote computers
- Multiuser Sessions

In COVISE it is possible to run modules on remote computers. This is also known as "Distributed Computing". By distributing modules across a network one can make use of remote resources for example of a compute server with more CPUs or memory than on a local workstation or PC. The COVISE session is controlled from the Mapeditor on the local workstation. Remote hosts are included in the session via the menu item CSCW >> AddHost (CSCW = Computer Supported Collaborative Work).

In a multiuser session each participant has it's own Mapeditor and Renderer. The session has to be initiated by one partner who adds the other partners to the session (menu item CSCW >> AddPartner). The initiating partner plays the master role, which means that he has the control over the Mapeditor and the Renderer. If he e.g. changes the camera position in the Renderer all other partner's cameras are synchronised with the master camera. The master role can be exchanged between partners. This way of working together in a multiuser session is also known as "Collaborative Working" where COVISE is regarded as a "Shared Application" which is aware of the sharing.

As the hosts of the partners can also be used for distributed computing COVISE extends far beyond a "Shared Application" such as the ones based on X Windows sharing or a Windows application shared via Netmeeting.

5.2 Architecture

This section provides background information on the COVISE architecture and explains how distributed and collaborative working is implemented.

Distributed Working

Figure 5.1 shows the elements of distributed working in COVISE. The application consists of three modules: a module which reads in data (READ) a module which extracts a special feature (FILTER) a module which displays the extracted data (RENDER). As the filter module consumes much CPU time and memory it will be started on a remote compute server.

The first process which is started when covise is typed in is the Controller which in turn starts the user interface process (Mapeditor) and the data management process (CRB). As soon as another host is

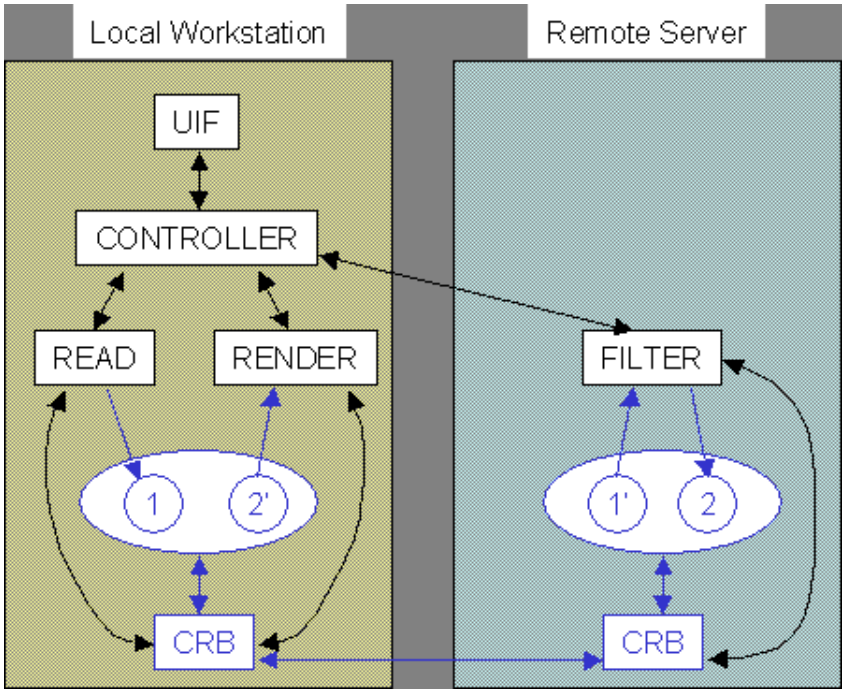

Figure 5.1: Distributed Working

included in the session a CRB is started on that computer. The read module is started on the local workstation, the filter module on the remote computer (see section "Starting a Module on a Remote Computer" later in this chapter) and the renderer on the local workstation. The black arrows between the processes Controller, UIF, CRB and the modules indicate TCP sockets, the blue arrows indicate shared memory access.

When the user executes the pipeline the Controller sends a start message to the read module. The read module reads in the data file and creates a COVISE data object (1) in shared memory and after processing tells the Controller that it has finished. The Controller informs the filter module on the remote computer to start. The filter module asks its data management process (CRB) for the data object (1). As this CRB doesn't have the object it asks the CRB on the workstation for the object and copies it to it's shared memory (1'). The filter module now reads that data object, computes something and puts the data object (2) into shared memory. It then tells the Controller that it has finished. The controller informs the Renderer module to start. The Renderer asks the CRB for object (2) and as this object is not available on the local workstations the CRB transfers it from the compute server into the shared memory of the workstation. Now the renderer can access this object and display the data.

Collaborative Working

In a collaborative session (Figure 5.2) a user interface process (Mapeditor) and a Renderer are started also on the remote machine. The Renderer module is the only module which is started on all computers in a session.

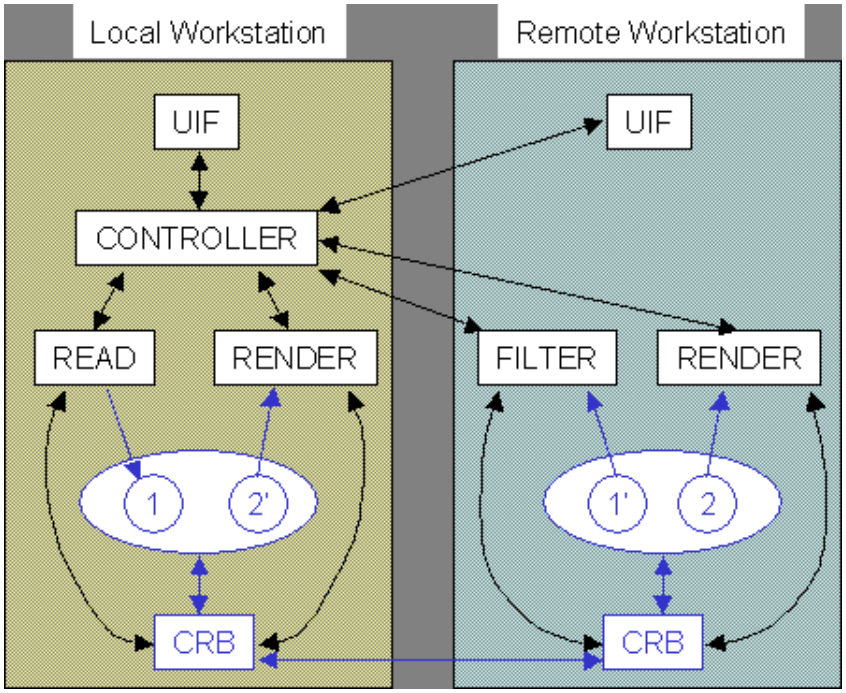

Figure 5.2: Collaborative Working

5.3 Preparing COVISE for distributed and Collaborative Working

Every computer that will participate in a distributed or collaborative session should be included in the section HostConfig in the file covise.config. For each host you have to specify the memory model for data exchange between modules on the local machine, the execution mode and a timeout for TCP connections.

```
HostConfig
{
    # Hostname MemoryModel ExecutionMode Timeout
    vista       shm         rexec       30
    visit       shm         rexec       30
}
```

For workstations and PCs the memory model is shm which stands for shared memory. There are other memory models like none specifically for machines such as CRAY Y-MP computers.

The execution mode specifies the command which should be used to start the CRB on the remote computer. Possible execution modes are:

- rexec
- rsh
- ssh
- nqs
- manual

For all execution modes besides manual one needs access to the account on the remote computer. For rexec one has to enter the hostname, the user id and the password on the remote machine (similar to logging in on the remote computer using telnet). rsh and ssh can only be used if they allow to log in without password specification (see man rsh and ssh for the files where allowed users are specified). nqs is not recommended, it can be used to put the CRB into a batch queue. Manual means that one has to start the CRB process manually on the remote machine. This can be useful for sessions across a firewall or if one doesn't have access to the remote account. In this case COVISE shows a command in the window where COVISE was started.

The time-out value specifies how many seconds a process will wait to be contacted by a new process that he initiated (e.g. the Controller waiting for a module). The default value is 5 seconds. For slow networks a time-out of 30 seconds is useful. For very slow networks even a higher value is recommended.

5.4 COVISE across Firewalls

As shown in Figure 5.2 COVISE uses TCP sockets for communication with remote hosts. A socket is defined by an IP address, a port number and the protocol (here tcp). COVISE port numbers start by default at 31000. One can configure the start number in the file covise.config using the keyword COVISE_PORT in the section network:

```
Network
{
    COVISE_PORT     5000
}
```

For collaborative or distributed sessions across firewalls the firewall has to allow tcp connections to ports in both directions starting with the number defined in covise.config. You need as many ports as modules started during the whole session +3 for distributed sessions or + 4 for collaborative sessions (if you load several maps in a session each map needs new ports). Depending on the execution mode the ports for rexec, rsh or ssh have to be allowed. For the execution mode manual no extra port is required.

Note:

If you use IP forwarding from your firewall to your local computer you have to make additional configurations. Every host that wants to connect to your session has to know that you are behind a firewall and use IP forwarding. Therefore you can tell COVISE not to connect to your machine but to your firewall instead. This is done by adding an IP_ALIAS entry on every client side. Assume your IP is 192.168.0.15 and your firewall has the IP 133.168.226.234 from the outside. Then you have to add

```
Network {
   IP_ALIAS     192.168.0.15     133.168.226.234
   #            <your IP>        <your firewall IP> =

}
```

to the config file on every host you want to connect to.

5.5 Including a remote host or partner in the session

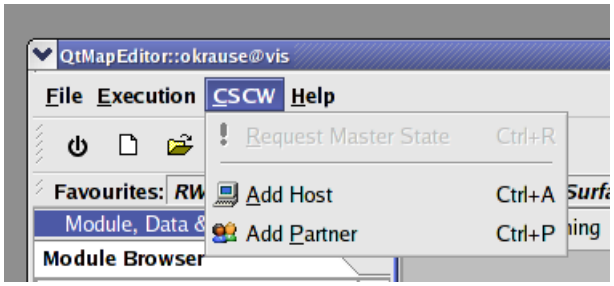

Figure 5.3: Hosts(CSCW) Menu

Figure 5.3 shows the menu item for adding a remote host or including a new partner into the session (CSCW = Computer Supported Collaborative Work).

The window in Figure 5.4 will pop up. First select a hostname or enter a new one.

If the selected hostname is available, the window in Figure 5.5 will appear. You can select the parameters for a connection.

Depending on the configuration parameters in *covise.config* the execution model and the time-out are adjusted. Now one can change the time-out and the execution mode if other values than the standard are required.

For execution mode rexec the user id and password on the remote computer has to be entered. For execution mode rsh or ssh only the user id is needed

In the manual execution mode COVISE writes a message in the window saying how COVISE should be started on the remote computer. It looks like

```
start "crb 31005 129.69.29.12 1005" on
visit.rus.uni-stuttgart.de
```

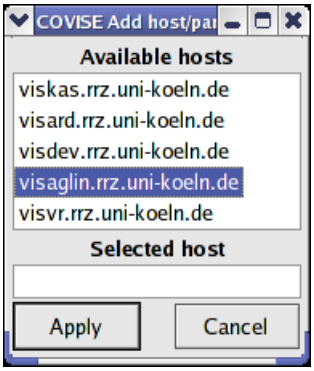

Figure 5.4: Host Selection Window

The collaboration partner has to type in the following command (which has to be provided to him by means such as phone, video conference or email):

```
crb 31005 129.69.29.12 1005
```

When the remote computer is added successfully the remote username and hostname will appear in the module browser list (see Figure 5.5). Here the option is used that hosts are shown colored.

In a multiuser session (CSCW >> AddPartner) a Mapeditor will pop up on the remote workstation.

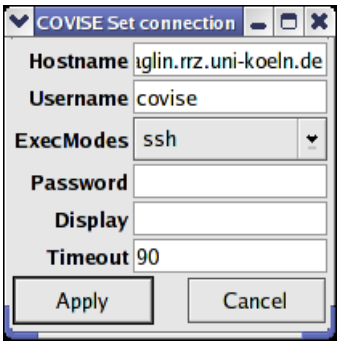

Figure 5.5: Set Connection Parameters

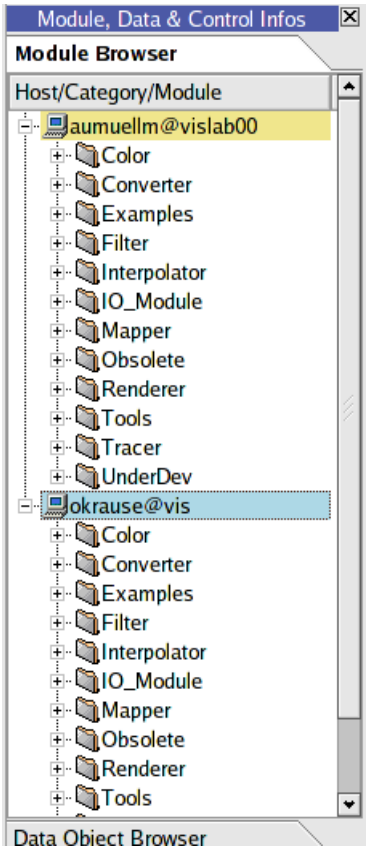

Figure 5.6: Remote Computer in the Module Browser

5.6 Starting a module on the remote computer

When selecting the remote computer in the hosts list the categories and modules available on this computer will be offered. Clicking on a module it is started on the remote computer. This is indicated by the hostname in front of the module name (Figure 5.7), if the hosts are not colored.

Figure 5.7: Icon for a Remote Module

Next the module ports have to be connected and parameters adjusted. It does not make any difference whether modules are executed locally or on a remote computer.

When a map is saved (menu File >> Save) the information about the hostname is saved, too. When a map is loaded which was saved including remote modules one is asked to add the remote hosts first

5.7 Collaborative Working

CSCW >> AddPartner includes the remote host in the session and starts a Mapeditor on the remote machine. Except for the renderers all other modules are started on the computer which was selected in the hosts list. Renderer modules are started on all workstations.

The partner who initiated the session initially has the master role. He can load maps or start modules and connect them. He also controls the renderers. The slave partners can watch all actions of the master but all menu items besides the menu master and interaction in the Mapeditor are deactivated. This is indicated by grey text on the menu buttons and in the modules. The slave partners can request the master role using the menu MasterControl >> Request (or use the corresponing item "MasterRequest" of the Viewer Popup Menu in the Renderer); thereupon a window (Figure 5.8) pops up on the master computer:

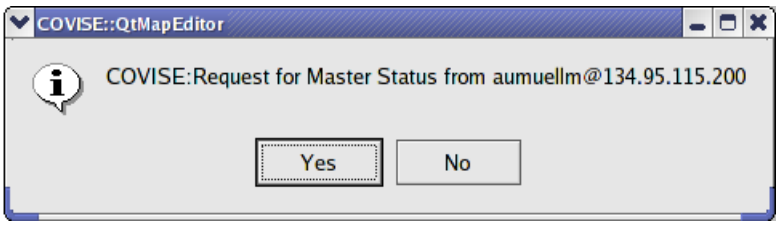

Figure 5.8: Master Request Window

The slave Renderers are synchronised with the master renderer which means that all manipulation actions like changing the camera position, zooming, selecting objects etc. are sent to the slave Renderers. As long as the master doesn't do anything in the Renderer the slave Renderers can be used independently.

One can make his own mouse pointer visible for the partners by pressing the SHIFT key and moving the mouse. This functionality is called Telepointer. In all remote renderers the originating hostname appears at the position pointed at (Figure 5.9). This also works for Renderer windows having different sizes as the position in 3D space is transmitted and not the 2D pixel coordinates.

Figure 5.9 shows a snapshot from a collaborative session. The user pw_te on host richard.vircinity uses the telepointer to show the other user(s) the backflow zone in a channel.

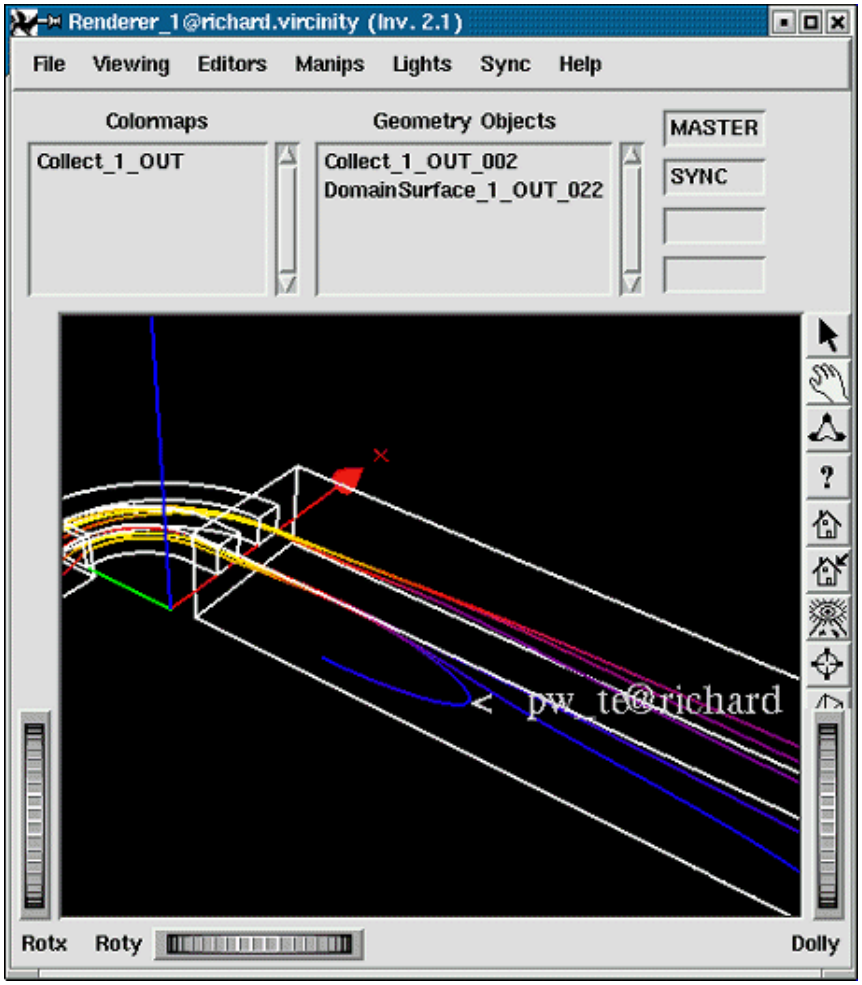

Figure 5.9: Telepointer in the Renderer