Institut für Visualisierung und Interaktive Systeme

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Fachstudie Nr. 131

Visualisierungssoftware für Daten aus numerischen Simulationen

- Software for Scientific Visualization -

Michael Metzger
Philipp Scholz
Philip Schneider

Studiengang:	Softwaretechnik
Prüfer:	Prof. Dr. Thomas Ertl
Betreuer:	Dipl.-Inf. Martin Falk Dipl.-Inf. Markus Üffinger
begonnen am:	23.11.2010
beendet am:	04.07.2012
CR-Klassifikation:	I.3.3, I.3.7

Inhaltsverzeichnis

1 Einleitung

1.1 Vorwort

Diese Fachstudie ist in enger Zusammenarbeit mit unseren Betreuern vom Visualisierungsinstitut der Universität Stuttgart, Dipl.-Inf. Martin Falk und Dipl.-Inf. Markus Üffinger, entstanden. Der praktischen Arbeit mit den zu untersuchenden Visualisierungswerkzeugen ging eine selektive Evaluation voraus um aus dem großen Pool an verfügbaren Programmen einen Teil herauszufiltern der durch uns genauer untersucht werden sollte. Wir formulierten dabei eine Reihe von Ausschlusskriterien. Es sollten unter anderem quelloffene, erweiterbare, auf wissenschaftliche Visualisierungen spezialisierte, aber nicht auf spezielle Anwendungsgebiete wie zum Beispiel die medizinische Visualisierung beschränkte Programme sein.

1.2 Abgrenzung

Das Ziel dieser Fachstudie ist der Vergleich der ausgewählten Programme im Hinblick auf eine wissenschaftliche Arbeitspraxis. Es ist dabei nicht vorgesehen die Programme bis ins Kleinste aufzuschlüsseln und ein zweites User-Manual zu schreiben.

1.3 Leserkreis

Wir gehen davon aus, dass der Großteil unserer Leser ein fundiertes Wissen auf dem Gebiet der Visualisierung besitzt und mit der entsprechenden Terminologie vertraut ist. Für alle anderen bietet das Kapitel 1.4 einen Einstieg in das Thema und Erklärungen zu Fachbegriffen die im weiteren Verlauf dieser Arbeit Verwendung finden und die für das Verständnis des Textes von Bedeutung sind.

1.4 Grundlegendes zur Visualisierung

Unter Visualisierung versteht man die Überführung von Daten in Bilder oder Animationen. Ziel dieser Umwandlung ist es, die meist schwer verständlichen Rohdaten, z.B. Messwerte, in eine für die menschliche Wahrnehmung geeignete Form zu bringen. Beispiele für die Ziele der Visualisierung können sein:

Zeigen von Zusammenhängen und Abhängigkeiten. Segmentierung, Differenzierung, Klassifizierung von einzelnen Objekten, Strukturen und Eigenschaften. Vereinfachung, um einen Überblick zu geben, wenn die Datensätze sehr groß sind.

Kernstück der Visualisierung ist die *Visualisierungspipeline*. Sie unterteilt die Umwandlung der Daten in mehrere Schritte, in denen alle Operationen und Interaktionen eingeteilt werden können. In Abb. 1.1 wird die Visualisierungspipeline dargestellt.

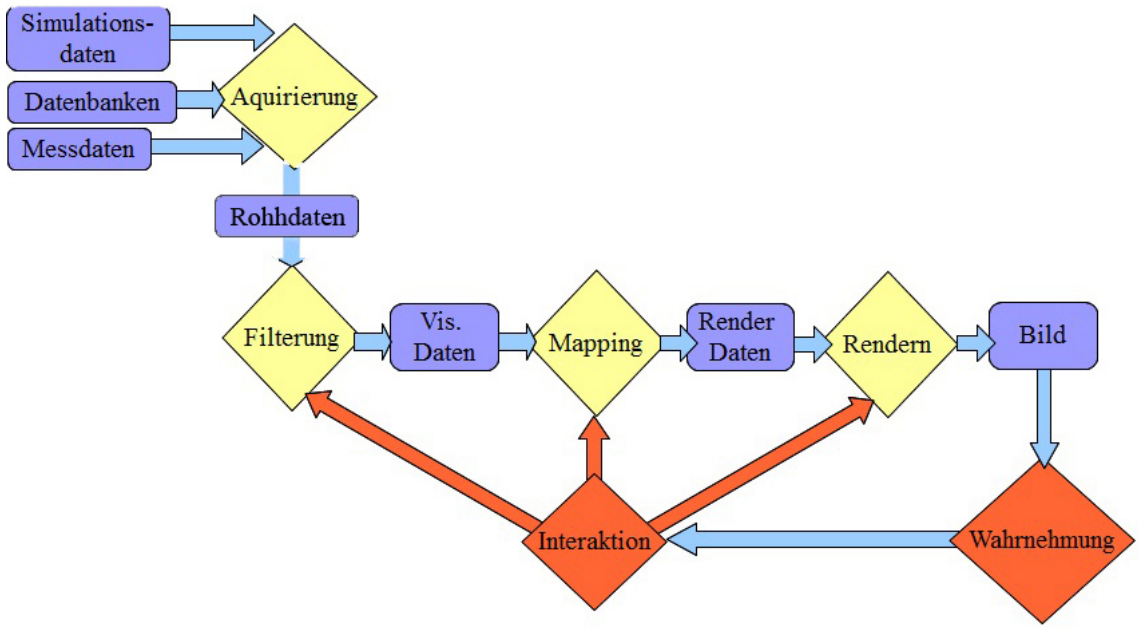


Abbildung 1.1: Visualisierungspipeline. Daten werden blau dargestellt, die Schritte der Pipeline gelb. Aktionen des Endanwenders werden rot gekennzeichnet.

1.4.1 Datenakquirierung

Im ersten Schritt werden die Rohdaten gewonnen. Grundlage können Simulationsdaten, Messdaten oder Daten aus Datenbanken sein. Dieser Schritt wird von Visualisierungsprogrammen meist übergangen, da die Daten in der Regel bereits vorliegen. Dieser Schritt wird vor allem durch die Integration der Software realisiert und hängt somit mit der Erweiterbarkeit des Programms zusammen. Soll zum Beispiel die Software in der Medizin eingesetzt werden, könnte für eine Visualisierungssoftware ein Plugin geschrieben werden, welches einen Computertomographen steuert. Die Bedienung dieses CT-Gerätes wäre dann die Datenakquirierung.

1.4.2 Datenfilterung

Im zweiten Schritt werden die Rohdaten gefiltert. Hier findet die Segmentierung statt, also die Abgrenzung verschiedener Bereiche. Ebenfalls werden beim Filtern Daten konvertiert und Fehler erkannt sowie beseitigt. Ergebnis der Filterung sind die Visualisierungsdaten, die Grundlage des Darstellungsprozesses. Die wichtigsten Operationen der Datenfilterung sollen im Folgenden kurz erläutert werden.

Clipping Das *Clipping* dient in der Praxis oft als Hilfsmittel für viele Probleme. Es werden lediglich Messwerte genommen die sich innerhalb eines geometrischen Körpers (z.B.

Bounding Box) befinden. Das spart Systemressourcen, da unwichtige Teile der Rohdaten entfernet werden können. Man kann jedoch auch die Werte selbst *clippen*, zum Beispiel um grobe Messfehler zu entfernen.

Schnittebenen Hat man ein Skalarfeld, das man als Farben darstellt, wird man immer nur den Rand des Skalarfeldes sehen, da er den Rest überdeckt. Abhilfe schafft eine Schnittebene die beliebig im Skalarfeld positioniert werden kann. Es wird dann für jeden Pixel der Schnittebene über die Skalare des Feldes interpoliert, die dem Pixel am nächsten sind.

Glätten Ist der eingelesene Datensatz zu groß oder waren die Messwerte ungenau, kann man durch Interpolation die Daten aneinander angleichen. Dies reduziert Messfehler, aber auch Details auf die es vielleicht ankommt.

Isolinien und -flächen In einem 2D Skalarfeld ist es möglich, alle (interpolierten) Punkte, die einem bestimmten Wert entsprechen, durch eine Linie zu verbinden. Das einfachste Beispiel sind Höhenlinien in einer Landkarte. Anstatt Höhenangaben für jeden Punkt darzustellen, werden lediglich für bestimmte Höhen Isolinien gezeichnet. Dasselbe ist auch bei 3D Skalarfeldern möglich. Bei einem Skalarfeld mit Werten zur Dichte im Inneren eines menschlichen Körpers wird wenig zu erkennen sein, da viel Überdeckung auftritt. Stellt man den Wert für eine Isofläche zum Beispiel knapp unter die Dichte von Knochen, wird die Umrandung des Skeletts als Isofläche dargestellt.

1.4.3 Mapping

Ein wichtiger Schritt in der *Visualisierungspipeline* ist das *Mapping*. Hierbei werden die gefilterten Daten auf geometrische bzw. grafische Objekte abgebildet. Grafische Objekte sind sehr oft Polygone - meist Dreiecke, aber auch Linien oder Punkte können hier genutzt werden. Man kann verschiedene Arten des *Mappings* unterscheiden. Die drei wichtigsten sollen hier kurz erläutert werden.

1.4.3.1 Mapping nach Position

Bildet man Zahlen auf eine Position in der Visualisierung ab, nutzt man diese Daten als Koordinaten. Das einfachste Beispiel sind Kurvendiagramme, *Scatterplots* oder Höhenfelder.

1.4.3.2 Mapping nach Farben

Die zur Verfügung stehenden Daten können in Farbwerte umwandelt und diese dann dargestellt werden. Meist wird eine Zahl auf eine bestimmte Farbe abgebildet. Dafür kann zum Beispiel einen Regenbogenverlauf, oder einen Verlauf zwischen zwei Farben verwendet werden. In Wetterberichten werden beispielsweise die Temperaturen in Grad Celsius auf Farben zwischen blau und rot *abgebildet*. Weitere Beispiele sind die Abbildung von Höhe in einen Farbwert einer Landkarte wie in Abb. 1.2 oder die Abbildung nach Position wie in Abb. 1.3.

Abbildung 1.2: Mapping der Höhendaten auf einen Farbwert. (http://de.academic.ru)

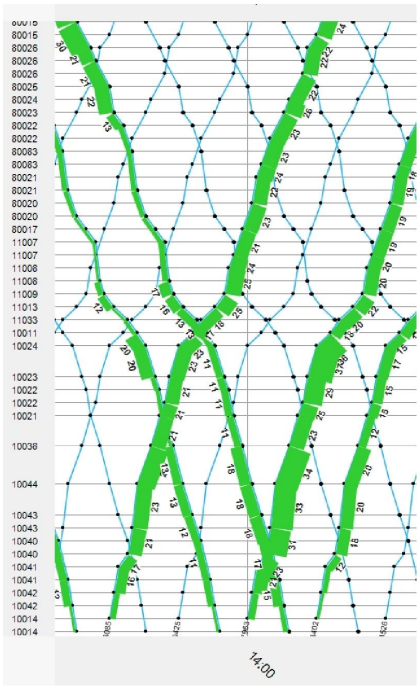


Abbildung 1.3: Mapping nach Position in einem Zeit-Weg-Diagramm (http://www.ptv.de)

1.4.3.3 Mapping auf Symbole/Icons

Eine weitere Möglichkeit besteht darin, die Visualisierungsdaten als Symbole darzustellen. Bei Wetterkarten werden zum Beispiel bestimmte Wettervorhersagen als Wolken- oder Sonnensymbole dargestellt, oder Temperaturen als Zahlen. Auch das sind Symbole, die wir als Wert interpretieren.

1.4.4 Rendern

Im *Renderprozess* werden die vorher erzeugten *Renderdaten* in ein Bild umgewandelt. Wie dies geschieht, ist vor allem von der Wahl der Grafikengine abhängig. Oft erfolgt das *Rendering* mit Hilfe der Grafikkarte und meist mit einer Grafik API wie *DirectX* bzw. *OpenGL*. Es können Beleuchtung und andere Effekte verwendet werden, um die Visualisierung besser verständlich oder schöner zu machen. Ein weiterer wichtiger Faktor beim *Rendern* ist die virtuelle Kamera, aus deren Perspektive man die Daten visualisiert. Wichtig sind dabei die Position und der Öffnungswinkel.

1.5 Programme

Eine Visualisierungssoftware muss für viele verschiedene Datenarten möglichst viele Ziele erreichbar machen, jedoch noch weitere Kriterien erfüllen. So sollte eine Software an eigene

Bedürfnisse anpassbar sein. Ebenso spielt die Interaktion eine wichtige Rolle. Durch Interaktion im Visualisierungsprozess kann die Qualität der Visualisierung gesteigert werden, da man verschiedene Ziele erreichen und mehrere Dinge in der Visualisierung zeigen kann. Vor allem für die Interaktion, sehr große Datensätze oder Visualisierung in Echtzeit bei Veränderung der Rohdaten, benötigt man sehr große Rechenleistung. Die Software sollte demnach ressourcenschonend arbeiten.

Einen ersten Ausblick auf die untersuchten Programme bieten die folgenden Abschnitte.

1.5.1 Paraview

Paraview ist ein quelloffenes und plattformunabhängiges Visualisierungswerkzeug, das seit dem Jahr 2000 als Kollaborationsprojekt von Kitware Inc. und dem Los Alamos National Laboratory entwickelt wurde. Der erste offizielle Release, Paraview 0.6, wurde im Oktober 2002 veröffentlicht. Zum Zeitpunkt des Verfassens dieser Arbeit war die Version 3.8 aktuell.

1.5.2 VisIt

VisIt ist ein frei verfügbares und plattformunabhängiges Visualisierungstool, das verteiltes und paralles Visualisieren von Daten ermöglicht. Es wurde vom *Department of Energy* (DOE) im Rahmen der *Advanced Simulation and Computing Initiative* (ASCI) entwickelt. Der initiale Release wurde Ende 2002 veröffentlicht. Seitdem wird es ständig weiterentwickelt und befand sich zum Zeitpunkt des Verfassens dieser Arbeit in der Version 2.3.2.

1.5.3 Covise

Covise ist eine Abkürzung für Collaborative Visualization and Simulation Environment. Entwickelt wurde die Software vom HLRS, dem High Performance Computing Center Stuttgart. Ziel war, es eine leicht erweiterbare, verteilte Software zu entwickeln, die Visualisierung und Nachbearbeitung vereint. Aktuell liegt Covise in der Version 6.5 vor, die auch in dieser Arbeit untersucht wurde.

1.6 Überblick über die Arbeit

Die folgenden Kapitel behandeln den Vergleich der Programme Paraview, VisIt und Covise bezüglich Benutzerfreundlichkeit, Architektur, Datenformate, Erweiterbarkeit, Remote-Visualisierung und Performance. Außerdem wird für ein Standardszenario das Vorgehen in allen drei Programmen beschrieben. Abschließend wird ein Fazit gezogen und eine persönliche Empfehlung gegeben.

2 Benutzerfreundlichkeit

2.1 Paraview

2.1.1 Aufbau der GUI

Auf den ersten Blick erscheint Paraview einfach und intuitiv. Die Oberfläche besteht standardmäßig aus drei großen Bereichen, die in einem Fenster vereint sind: Die Menüleiste mit mehreren Werkzeugleisten, das Fenster für die eigentliche Visualisierung und der *Pipeline Browser* mit dem *Object Inspector*. Jede Operation, die auf den Datensatz angewandt wird, erscheint im *Pipeline Browser* und kann im *Object Inspector* konfiguriert werden. So ist auf einen Blick sichtbar, durch welche Kombination von Filtern und Operationen das aktuell visualisierte Bild entstanden ist.

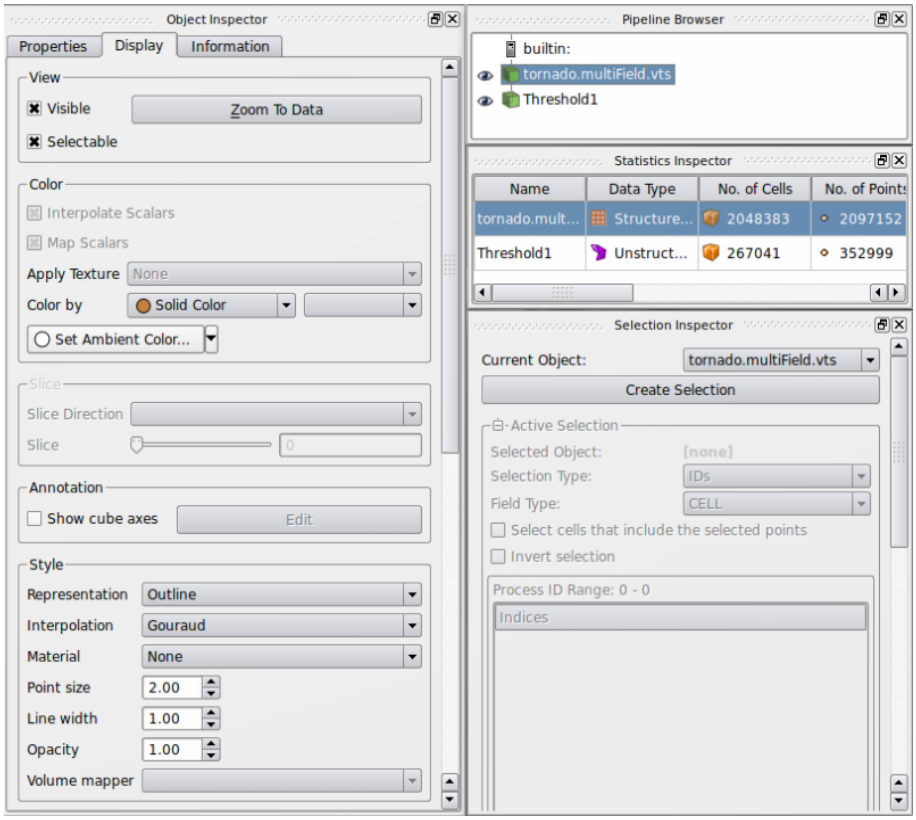


Abbildung 2.1: Verschiedene GUI-Elemente von Paraview.

2.1.2 Drag-and-Drop

Die einzelnen Bereiche sind per Drag-and-Drop verschiebbar. Auf diese Weise kann die grafische Oberfläche leicht angepasst werden. Außerdem ist jedes Element in seiner Größe variabel. Dies ermöglicht es, den vorhandenen Platz optimal auszunutzen. Zusätzlich lassen sich noch weitere Werkzeug-Oberflächen einblenden.

- *Animation View* Werkzeuge zum Erstellen einer Animation
- *Statistics Inspector* Zeigt globale, visualisierungsunabhängige Informationen über den Datensatz an, wie Größe des Datensatzes, Anzahl der Gitterelemente oder Art des Datensatzes
- *Selection Inspector* Konfiguration der verschiedenen Auswahlwerkzeuge
- *Comparative View Inspector*

Jede Werkzeugoberfläche, also auch der *Pipeline Browser* und der *Object Inspector* kann mittels Drag-and-Drop von der Haupt-Oberfläche entkoppelt werden. Es können auch mehrere Werkzeugoberflächen zusammen in eine Spalte gruppiert werden.

2.1.3 Erste Schritte

Wie aus vielen anderen Programmen bekannt, lässt sich mit File → Open ein Datensatz laden, der mit den Standard-Einstellungen visualisiert wird. Paraview erfordert für die ersten Schritte also wenig Einarbeitung. *Tooltips* auf den Symbolen der Werkzeugleisten sorgen dafür, dass sich der Benutzer mit Paraview-spezifischen Elementen schneller zurechtfindet.

In dem Bereich, in dem die Visualisierung erscheint, lässt sich mit der Maus die Ansicht konfigurieren. Durch Halten der linken Maustaste und Bewegen der Maus verändert sich der Blickwinkel. Über das Mausrad lässt sich, wie aus vielen anderen Programmen bekannt, der Zoom-Faktor einstellen. Ein kleines Symbol in der linken unteren Ecke zeigt die aktuellen Richtungen der 3 Koordinatenachsen und hilft dabei, den Überblick zu behalten. Diese Navigation mit der Maus reagiert auch auf schnelle Benutzereingaben prompt. Bei aufwändigen Render-Verfahren wie Volumenrendering wird beim Drehen der Ansicht zunächst eine durchsichtige Hülle gezeichnet, was deutlich weniger Rechenleistung benötigt. Diese wird mit Daten ausgefüllt, sobald die linke Maustaste wieder losgelassen wird. Es ist möglich, die Anzeige für die Visualisierung zu teilen, um eine zweite Ansicht mit anderen Visualisierungs-Parametern zu erzeugen. Wahlweise kann der Blickwinkel der Kamera bei verschiedenen Ansichten gleich gehalten oder unterschiedlich gewählt werden.

2.1.4 Weitere Funktionen

Neben der Drag-and-Drop Funktionalität für Werkzeugoberflächen bietet Paraview auch die Möglichkeit, einzelne Anzeigefenster vom Hauptfenster zu entkoppeln. Leider zeigen sich hier Einschränkungen. Entkoppelte Fenster können nicht aneinandergeheftet oder zusammengefasst werden. Außerdem werden entkoppelte Fenster immer im Vordergrund der Paraview-Arbeitsfläche angezeigt, bekommen in der Fensterliste des Betriebssystems keinen eigenen Eintrag und können nur gemeinsam mit dem Hauptfenster minimiert werden. Der einzige

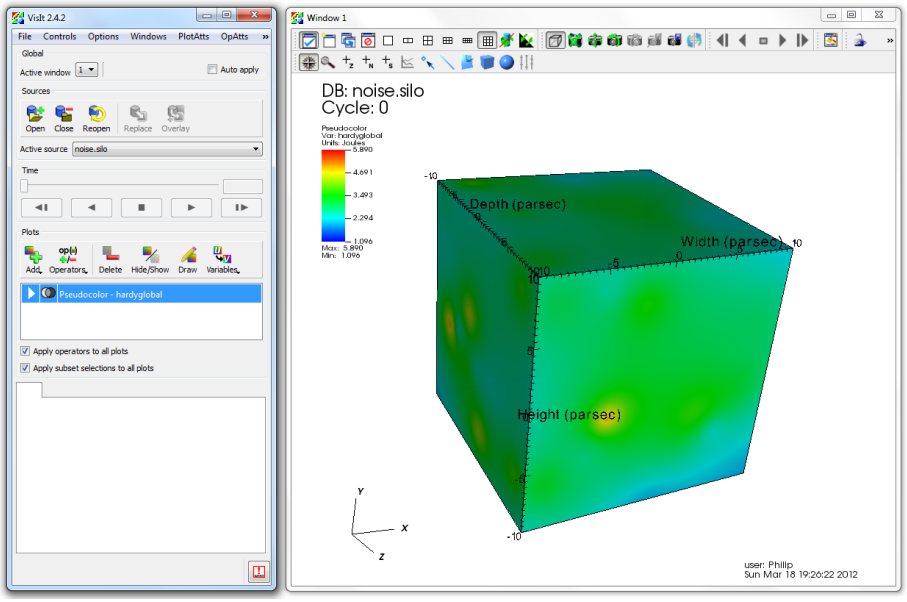


Abbildung 2.2: GUI Elemente von VisIt

Weg, sie auszublenden, ist sie zu schließen. Im Menü *View* können sie wiederhergestellt werden. Dabei nehmen sie ihre vorherige Position wieder ein. Hotkeys für die einzelnen GUI-Elemente gibt es nicht.

2.2 VisIt

VisIt besitzt eine geteilte Benutzeroberfläche. Beim Starten der Anwendung werden zwei Fenster geöffnet. Zum einen das Hauptfenster und zum anderen ein Visualisierungsfenster (siehe Abbildung 2.2). Im Hauptfenster sind alle von VisIt bereitgestellten Menüs erreichbar, vom Dateiauswahl Dialog über die Options- und Konfigurations- bis zum obligatorischen Hilfemenü. Im Visualisierungsfenster werden gerenderte Visualisierungsdaten (in VisIt Plots genannt) dargestellt. Dieses Fenster hat in der Menüleiste nochmal ein Schnellzugriff auf einige Menüs die während der Betrachtung der Daten nützlich sind.

VisIt bietet die Möglichkeit mit mehreren Visualisierungsfenstern gleichzeitig zu arbeiten. Unter dem Menüpunkt *Windows* oder im Visualisierungsfenster selber kann zwischen den Layouts 1x1, 1x2, 2x2, 2x3, 2x4 und 3x3 gewählt werden. Abbildung 2.3 zeigt ein 3x3 Gitter-Layout bei dem jedes der neun Fenster einen anderen Plot des selben Datensatzes darstellt.

Weitere Fenster können bei Bedarf erstellt werden, sodass keine Begrenzung nach oben gegeben ist. Durch die Entkopplung der Visualisierungsfenster vom Rest der Anwendung ist mit VisIt auf unkomplizierte Weise ein paralleles Arbeiten möglich und in den verschiedenen Fenstern kann unabhängig voneinander auf unterschiedliche Weise visualisiert werden. Im Hauptfenster kann entsprechend unter dem Menüpunkt *Global* das gewünschte aktive Fenster ausgewählt werden. Jede Operation verändert dann nur die Visualisierung im aktiven Fenster. Gleiches gilt für die Auswahl des aktiven Datensatzes. Alle Filter und Operationen, die

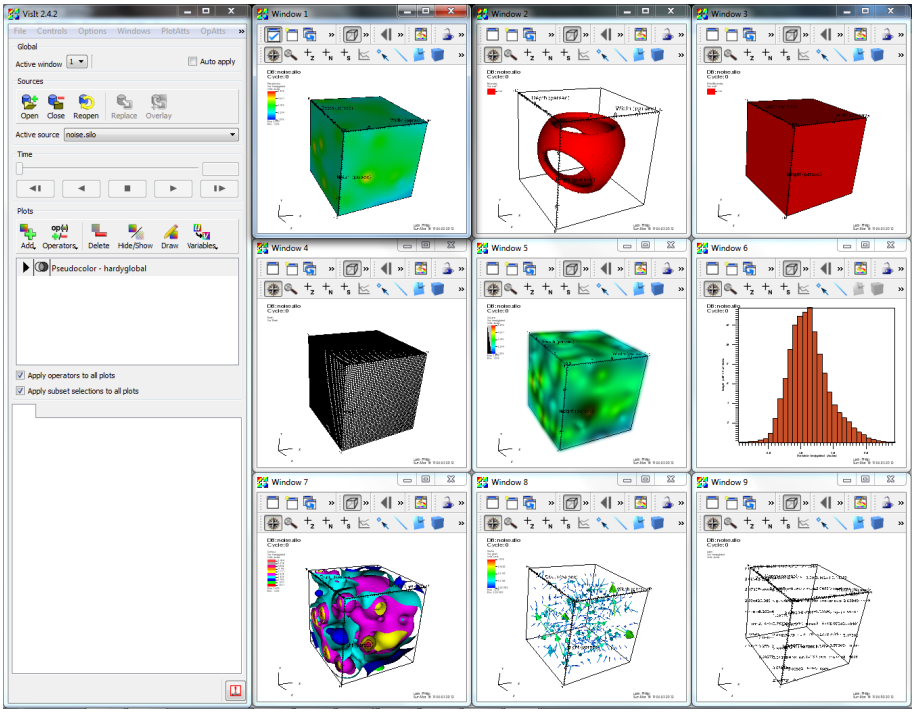


Abbildung 2.3: Visit GUI mit 3x3 Visualisierungsfenstern

danach angewandt werden, gelten somit nur für den aktiven Datensatz innerhalb des als aktiv gewählten Fensters. So ist nicht nur der Vergleich von Visualisierungen des selben Datensatzes möglich, sondern auch der Vergleich und das Verschmelzen von mehreren Datensätzen untereinander.

VisIt legt viel Wert auf Benutzerfreundlichkeit. Es bietet die Möglichkeit, die Gestaltung des Menüs an die eigenen Bedürfnisse anzupassen. Der untere Bereich des Hauptfensters kann dafür verwendet werden, so gut wie alle zur Verfügung stehenden Menüs als Schnellzugriff erreichbar zu machen. Abbildung 2.4 zeigt beispielhaft, wie das *Lighting* Menü als Schnellzugriff an den Dockbereich angehängt wird. Bei mehreren Menüs werden diese in einer Tab-Ansicht dargestellt. Mit dem Button *Dismiss* können sie wieder entfernt werden.

Bei VisIt werden die Arten der Visualisierung als Plots bezeichnet. Entsprechend ist der wichtigste Menüpunkt zur Auswahl und Parametrisierung der Filter auch als Plots gekennzeichnet. In einer Liste werden hier alle erstellten Plots angezeigt.

2.3 Covise

Der größte Unterschied zwischen Covise und den anderen untersuchten Programmen ist die strikte Aufteilung aller Funktionalitäten in Module. Diese Module lassen sich wie Bausteine zusammensetzen und verbinden, um so Visualisierungen zu erstellen.

Dies beginnt bei Loadern, die unterschiedliche Datenformate einlesen, über Mapper und Filter bis hin zu Renderern. Jedes Modul hat Ein- und Ausgänge mit speziellen Datentypen. Auf ei-

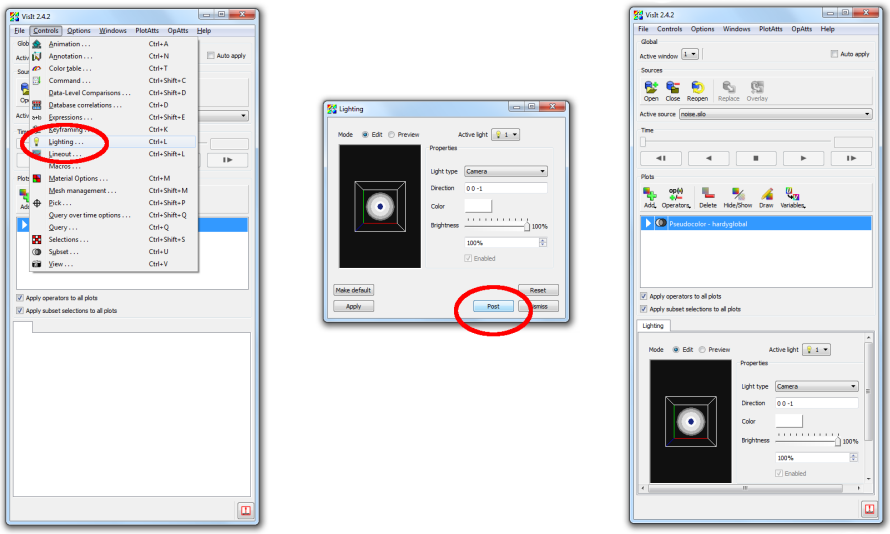


Abbildung 2.4: Ein Menü in den Schnellzugriff-Bereich einfügen

ner Zeichenebene lassen sich die Module mit Hilfe einer Auswahlliste platzieren und die Ein- und Ausgänge mit Linien verbinden, um sie zusammenzuschalten. Die Datentypen werden mit Farben an den Ein- und Ausgängen dargestellt und zieht man eine Linie, werden alle passenden Ein- bzw. Ausgänge farblich hervorgehoben. Zusätzlich gibt es ein Eigenschaftsfenster, in dem man Parameter ausgewählter Module ändern kann. Dies ist bei Loadern zum Beispiel der Dateipfad. Der Vorteil dieser Einteilung in Module ist vor allem eine sehr aufgeräumte Benutzeroberfläche. Es gibt kaum Buttons, da man Bedienfelder für eine Funktionalität erst nach dem Setzen eines Moduls erhält.

Die erste Visualisierung in Covise ist nicht etwa in einem Renderer zu sehen, sondern in der Benutzeroberfläche von Covise selbst. Denn die Darstellung der Module auf der Zeichenfläche, und die Verbindung der Module mit Linien, entspricht einer Visualisierung. Diese ist sehr übersichtlich und korrespondiert sehr gut mit der Visualisierungspipeline. Natürlich erfordert es eine gewisse Einarbeitungszeit und ein Kennenlernen der Module. Leider sind die Module nicht immer intuitiv benannt, was ein längeres Suchen einer bestimmten Funktionalität zur Folge hat.

Feedback über Aktionen erhält der Benutzer über eine Ausgabe, *Message Area* genannt. Diese Ausgabe ist wie alles andere in Covise in englischer Sprache. Eine deutsche Version ist zur Zeit nicht verfügbar.

Die Benutzeroberfläche besteht somit aus fünf Bereichen. Oben ist eine Menüleiste mit Funktionen wie Öffnen, Speichern, Rückgängig und Buttons für die am häufigsten verwendeten Module. Am linken Rand befindet sich eine Liste aller Module, sortiert nach Kategorien. Am rechten Rand ist das Eigenschaftsfenster zu finden, in dem man für ein selektiertes Modul Parameter verändern kann. Unten ist das bereits angesprochene Ausgabefeld für Meldungen von Covise und in der Mitte die Zeichenebene in der man Module platzieren und verbinden kann.

All diese Bereiche lassen sich, dank QT als Oberflächenengine, intuitiv in der Größe ändern

oder ent- bzw. andocken und zum Beispiel auf einem zweiten Bildschirm platzieren. Manche Module öffnen ein neues Fenster in der Taskleiste, zum Beispiel Renderer. Dort ist dann die eigentliche Visualisierung zusammen mit Bedienelementen des Renderers zu sehen. Im Folgenden werden zwei wichtige Renderer beschrieben. Der Standardrenderer und das Rendermodul *Cover*.

Der Standardrenderer wird über das Modul *Renderer* aufgerufen. Der Renderer wirkt nicht überladen mit Funktionen, der Hauptbereich mit der eigentlichen Visualisierung nimmt einen Großteil des Bildschirms ein.
Am rechten Rand findet man eine Liste mit allen darstellbaren Objekten (alle Module die am Eingangsport des Renderers angeschlossen sind). Über die rechte Maustaste im Visualisierungsbereich erscheint ein Menü mit weiteren Darstellungsoptionen. Hier kann man zwischen Drahtgitteransicht, Gouraud-Shading oder einer Darstellung mit reduzierter Polygonanzahl umschalten. Beleuchtung, Transparenz und Texturierung lassen sich hier ebenso einstellen, wie verschiedene Modi für Stereoprojektionen. Leider funktionieren die Darstellungsmodi in der getesteten Version nicht. Die Menüeinträge sind weder ausgegraut, noch erscheint eine Meldung in der Ausgabe des Hauptfensters, wieso die Funktion nicht verfügbar ist.

Der Renderer *Cover* geht einen anderen Weg. Die gesamte Fensterfläche ist der Visualisierung vorbehalten, es gibt keine angedockten Bereiche. Das Menü ist in der Visualisierung im Hintergrund und lässt sich dort verschieben. *Cover* wurde speziell für *Virtual Environments* entwickelt und lässt sich mit Plugins an spezielle Bedürfnisse anpassen. Mit entsprechenden Plugins für Headtracking, Datenhandschuhe und andere Eingabegeräte ist Covise somit die ideale Basis für eine *Virtual Environment*.

3 Beispielszenario

Um die drei Programme im Einsatz vergleichen zu können, haben wir ein Beispielszenario entwickelt. In drei Schritten sollen grundlegende Programmfunktionen und einige Visualisierungstechniken angewandt werden.

Der erste Schritt dient rein zur Darstellung des Datenbereichs.

Im zweiten Schritt werden Funktionen getestet, die einen Einblick in die Skalarfelder des Datenbereichs geben. Durch Farbvariation auf einer Schnittfläche, Isoflächen und Volumenvisualisierung sollen die Programme eine Repräsentation der Skalardaten liefern.

Abschließend soll im dritten Schritt ein Vektorfeld visualisiert werden. Durch Pfeile auf einer Schnittebene wird ein kleiner Teil davon genauer untersucht. Das komplette Vektorfeld wird unter der Verwendung von *Stromlinien* dargestellt.

- Schritt 1: Darstellung des Drahtgitters
  - Darstellung als fester Körper
  - Darstellung als Wireframe
  - Schnitt durch den Körper
  - Erstellen einer Schnittebene
- Schritt 2: Darstellung von Skalardaten
  - Darstellen des Skalarfeld für Druck als Farben auf der Schnittebene
  - Variation der Farbkodierung
  - Darstellen des Betrags der Geschwindigkeit als Farben auf der Schnittebene
  - Darstellen des Skalarfelds mit einer Isofläche
  - Darstellung des Skalarfelds mittels Volumenvisualisierung
- Schritt 3: Darstellung von Vektordaten
  - Darstellen der Geschwindigkeit als Pfeile auf der Schnittebene
  - Darstellen der Geschwindigkeit als Stromlinie
  - Kodieren des Geschwindigkeitsbetrags als Farben der Stromlinien

3.1 Datensätze

Für die Darstellung von Gitter und Skalardaten sowie zur Volumenvisualisierung wird der Datensatz *bluntfin.vts* verwendet. Er repräsentiert den Luftstrom um ein stumpfes Flugzeugteil bei einem Flug mit Überschallgeschwindigkeit. Durch seine relativ geringe Größe von 1.1 MB lassen sich die meisten Operationen ohne längere Wartezeit durchführen. Somit eignet er sich sehr gut, um Grundfunktionen von Paraview, VisIt und Covise zu testen.

Für die Darstellung von Vektordaten wurde uns ein Datensatz zur Verfügung gestellt, der den Luftstrom in einem Tornado enthält. Er enthält die Vektorfelder *Vorticity* (Wirbelvektor) und *Velocity* (Geschwindigkeitsvektor).

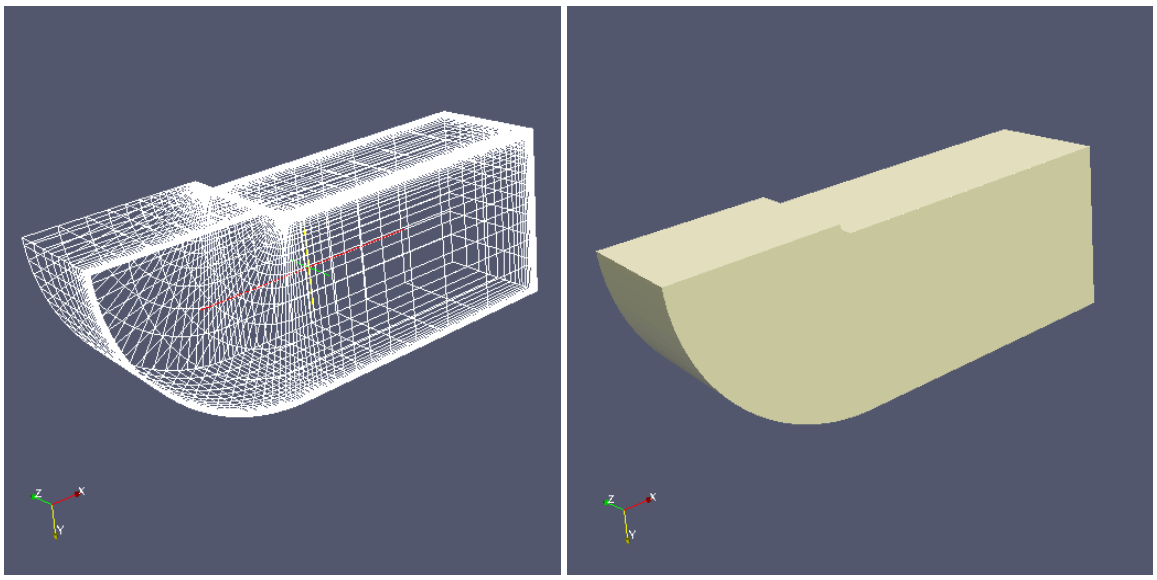


Abbildung 3.1: Links: Darstellung des Beispieldatensatzes *bluntfin.vts* in Paraview als Wireframe. Rechts: Darstellung als Festkörper

3.2 Szenario in Paraview

3.2.1 Gitter

Über das Menü *File -> Open* wird die Datei *bluntfin.vts* aus einem Paraview Tutorial ausgewählt. Im *Selection Inspector* kann der Benutzer wählen, welche Daten mit dem Gitter geladen werden sollen. In diesem Beispiel sind die Daten für *Density*, *Momentum* und *Stagnation Energy* verfügbar. Mit einem Klick auf *Apply* wird das Gitter angezeigt.

In Paraview wird ein neu gelesenes Gitter im Outline-Modus dargestellt. Im *Object Inspector* kann im Bereich *Style* unter Representation der Anzeige-Modus für das Gitter gewählt werden. Die Abbildung 3.1 zeigen die Beispieldaten als Drahtgitter (Wireframe) und als Festkörper (Surface).

Als nächstes soll ein Schnitt durch den Körper erfolgen. Dazu wird im Menü *Filters -> Common* der Menüpunkt *Clip* ausgewählt. Alternativ kann die Clip-Operation auch über die Toolbar *Common* erfolgen, sofern diese aktiviert ist. Mit einem Klick auf *Clip*, erscheint im *Pipeline Browser* ein neues Element Clip2 und im Bildfenster ein Rahmen um das Objekt (wie im Darstellungs-Modus *Outline*) sowie eine rot markierte Ebene, welche die Schnittebene darstellt. Die Linie senkrecht zur Ebene stellt den zugehörigen Normalenvektor dar. Mit der Maus lässt sich der Normalenvektor verschieben und drehen. Hat die Schnittebene die gewünschte Position, reicht ein Klick auf *Apply* im *Object Inspector* und der 3D-Körper wird entlang der markierten Ebene geteilt. Der Teil, in den der Normalenvektor zeigt, bleibt bestehen, der andere Teil, wird abgeschnitten. Im *Pipeline Browser* stehen jetzt die Elemente *bluntfin.vts* und *Clip2*. Ein Klick auf das Augensymbol neben einem Element bewirkt, dass das Objekt ein- oder ausgeblendet wird. Für das Element *Clip2* lassen sich jetzt im *Object Inspector* ebenfalls verschiedene Anzeigeoptionen konfigurieren.

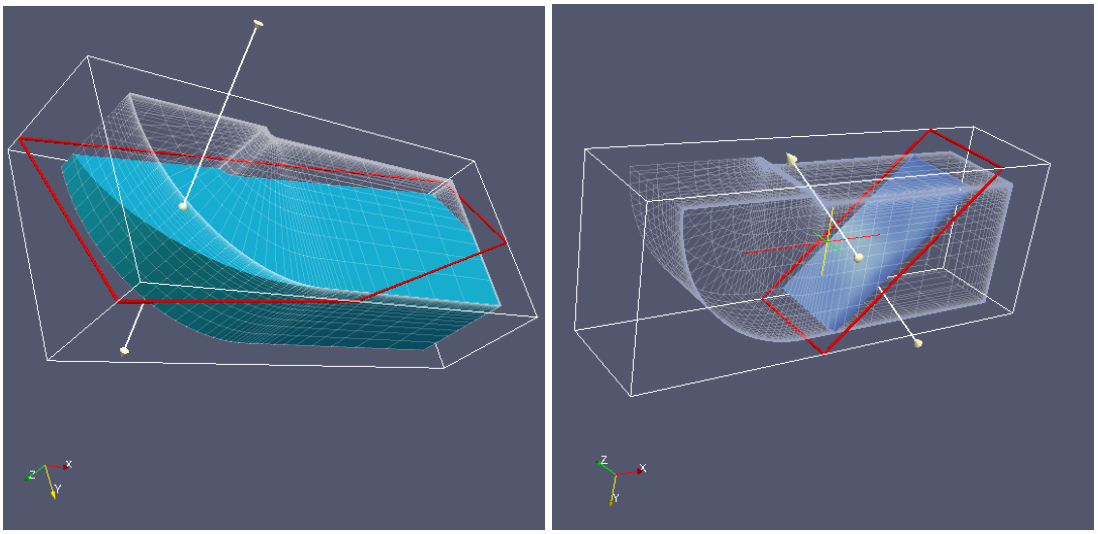


Abbildung 3.2: Links: Durch den Operator *Clip* wird der Körper entlang der Ebene geschnitten. Rechts: Die Anwendung des Operators *Slice* ergibt eine Schnittfläche.

- Um das Ergebnis der Schnittoperation deutlicher darzustellen wurden in diesem Beispiel einige Anpassungen vorgenommen:
  - Anzeige-Modus für *bluntfin.vts* ist *Wireframe*
  - *Opacity* wird im *Object Inspector* unter *Style* auf 0.2 gesetzt. So werden die Gitterlinien semi-transparent
  - Anzeige-Modus für Clip2 ist *Surface*
  - *Solid Color* wird im *Object Inspector* unter Color auf Blau gesetzt

Analog zu *Clip*, steht die Operation *Slice* zur Verfügung. Der Unterschied ist, dass bei *Slice* nicht einer von zwei Teilen des Objekts stehen bleibt, sondern nur die Schnittebene selbst, die wiederum als *Wireframe*, *Outline* oder *Surface* angezeigt werden kann.

3.2.2 Skalarfeld

Der bestehende Datensatz enthält auch Skalardaten, die im Folgenden visualisiert werden sollen. Hierzu ist sicherzustellen, dass mindestens einer der bestehenden Skalardatensätze *Density*, *Momentum* und *Stagnation Energy* im *Object Inspector* markiert ist. Wie im vorherigen Beispiel wird die Operation *Slice* dazu benutzt, um eine Schnittebene zu visualisieren.

Im *Object Inspector*, unter *Display* → *Color* → *Color by* wird festgelegt, welcher Datensatz dazu verwendet werden soll, um die Schnittfläche einzufärben. Die beiden Bilder zeigen die Visualisierungen für die Datensätze *Density* und *Stagnation Energy*. Im rechten Bild wurde zusätzlich die Farbpalette von blau→rot auf schwarz→gelb geändert.

Um einen Eindruck von dem Skalarfeld aus dem gesamten Datensatz zu gewinnen, gibt es die Möglichkeit, Punkte mit gleichen Skalarwerten zu einem Gitter zusammenzufassen, welches meist als geschlossene Oberfläche, eine *Isofläche* dargestellt wird.

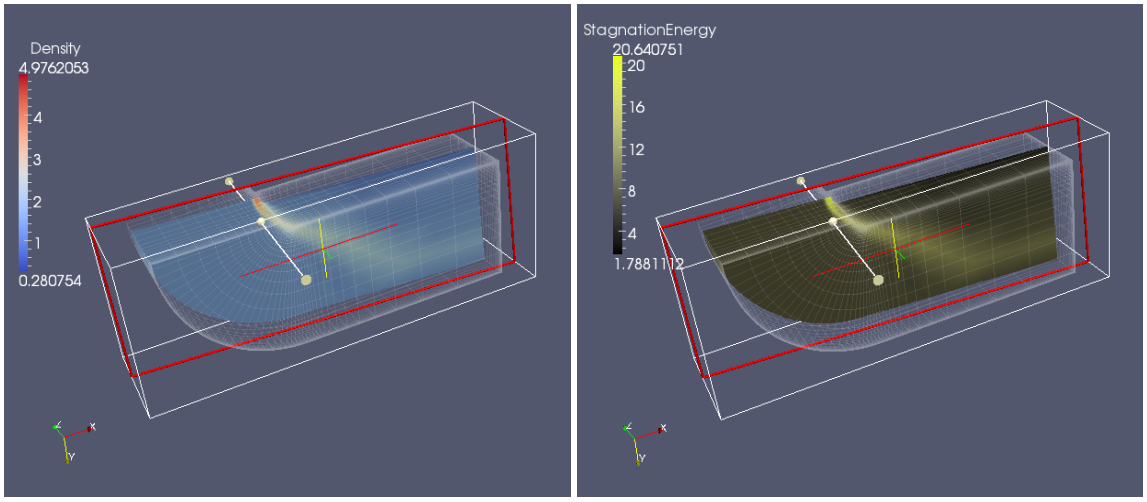


Abbildung 3.3: Links: Darstellung der Skalarwerte für *Density* als Farben auf der Schnittfläche. Rechts: Darstellung der Skalarwerte für *Density* mit geänderter Farbpalette.

In Paraview lassen sich *Isoflächen* durch den Filter *Contour* erstellen. Im *Object Inspector* lassen sich alle notwendigen Einstellungen vornehmen. Unter *Properties* → *Isosurfaces* befindet sich eine Liste für die Zahlenwerte, entlang derer die *Isoflächen* entstehen sollen. Bei mehreren *Isoflächen* empfiehlt es sich, den Wert für *Opacity* kleiner als eins zu wählen, um überdeckte kleinere Flächen sichtbar zu machen.

Für die Volumenvisualisierung wählt der Benutzer im *Object Inspector* unter Style → Representation die Option *Volume*. Unter der Option *Volume Mapper* lässt sich die Methode festlegen, mit der eine Volumenvisualisierung erzeugt wird. In der getesteten Version 3.8.1 sind folgende Algorithmen verfügbar:

- *Projected Tetra* (GPU - beschleunigt)
- *HAVS* (GPU - beschleunigt)
- *Z-Sweep*
- *Bunyk Ray Cast*

Der voreingestellte Algorithmus *Projected Tetra* ist auch gleichzeitig der schnellste.

3.2.3 Vektordaten

Paraview bietet auch die Möglichkeit, Vektorfelder zu visualisieren. Als Beispiel dient der Datensatz eines Tornados. Darin enthalten ist unter anderem ein Vektorfeld, das die Geschwindigkeit der Luft an jedem Punkt darstellt.

Um diese Daten sichtbar zu machen, wird zunächst die Operation *Slice* angewandt. Wie in Abbildung 3.6 zu sehen ist, verläuft die Schnittebene parallel zur X-Y-Ebene. Auf dieser Schnittebene werden durch die Operation *Glyph* Pfeile angezeigt. Diese ist entweder über die Toolbar *Common* oder über das Menü *Filters* → *Common* → *Glyph* erreichbar.

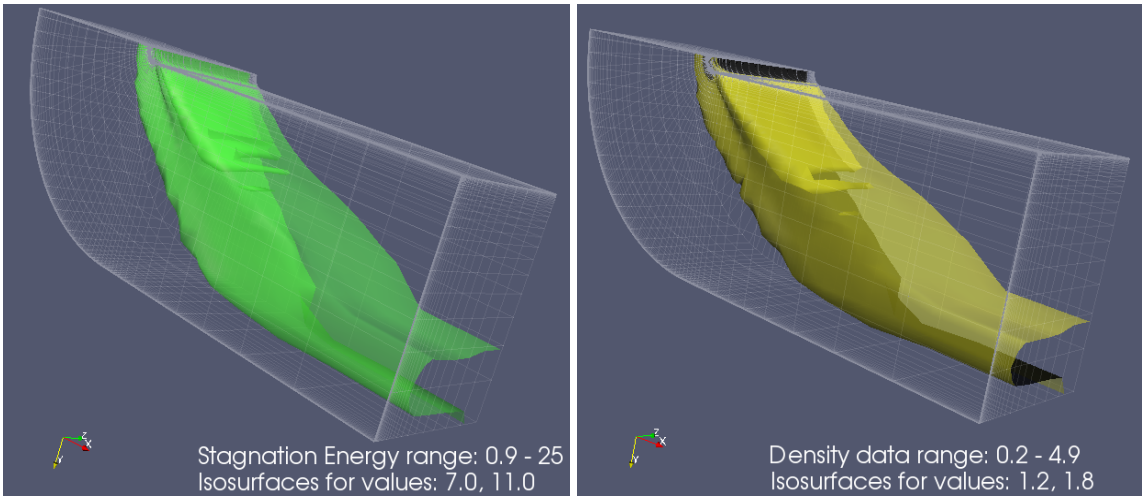


Abbildung 3.4: Links: Isoflächen der Werte 7.0 und 11.0 im Datensatz *Stagnation Energy* als geschlossene Oberfläche. Rechts: Zwei *Isoflächen* des Datensatzes *Density* zu sehen. Um die innere Fläche sichtbarer zu machen, wurden über die Einstellung *Display* → *Backface Style* die Innenflächen schwarz gezeichnet.

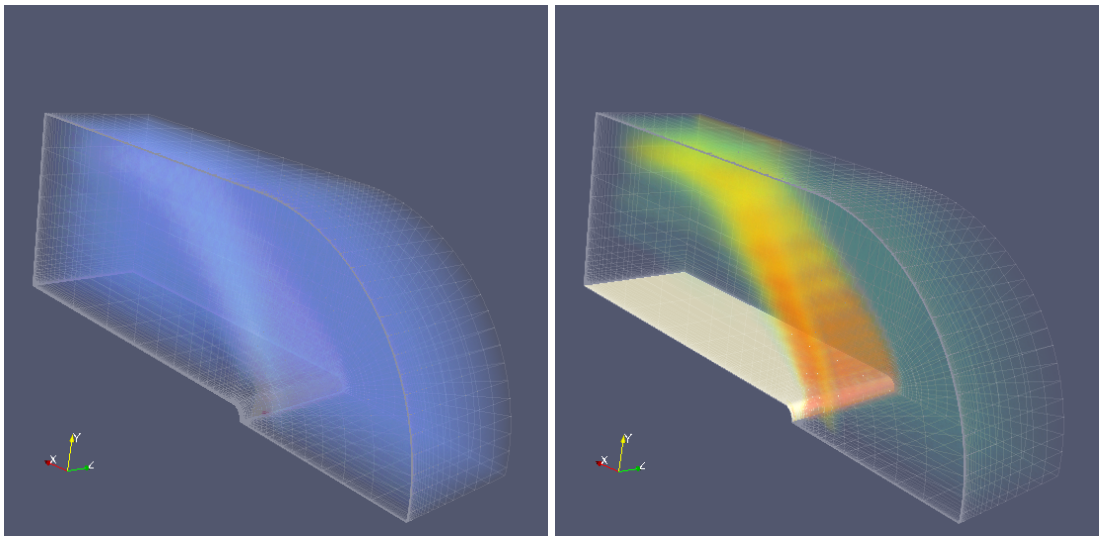


Abbildung 3.5: Volumenvisualisierung des Luftstroms. Im rechten Bild wird zusätzlich noch Das Flugzeugteil als Oberfläche dargestellt und die Transferfunktion geändert. So wird der Kern des Luftstroms deutlich sichtbar.

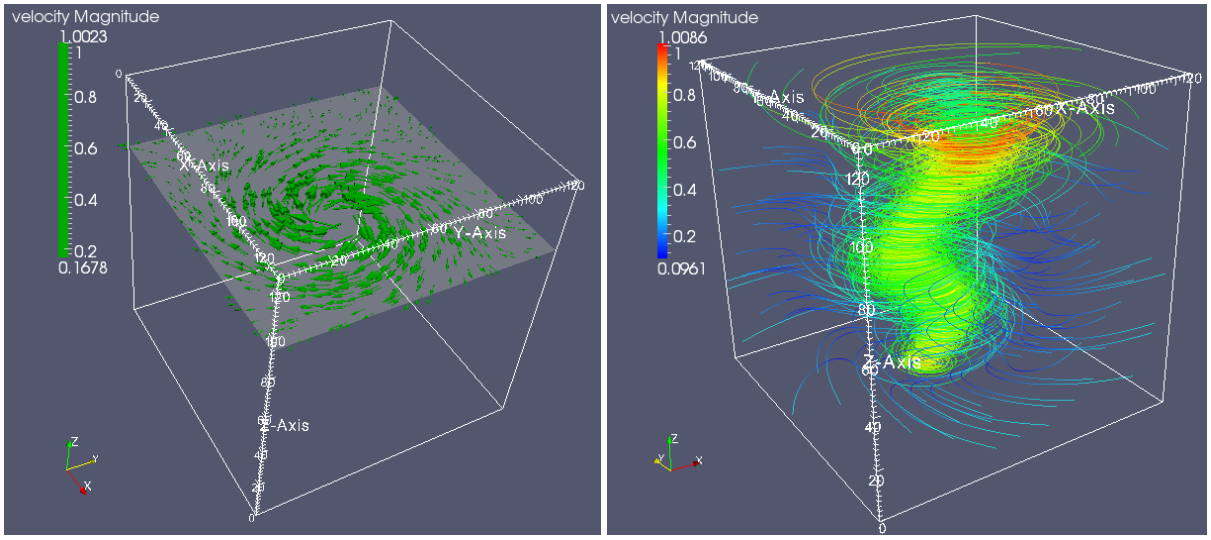


Abbildung 3.6: Links: Pfeile auf der Schnittebene geben die Richtung des Vektorfeldes an. Rechts: Stromlinien zur Visualisierung des kompletten Vektorfeldes. Farben repräsentieren die Geschwindigkeit

Paraview enthält auch Funktionen, um das komplette Vektorfeld zu visualisieren. Der *Stream Tracer* eignet sich hierzu hervorragend. Durch einen Klick auf das Symbol in der Toolbar *Common* oder die Auswahl des Menüpunktes *Filters* → *Common* → *Stream Tracer* wird im *Object Inspector* der Bereich sichtbar, wo die Einstellungen vorgenommen werden können.

- *Seed Mode*: Point
- *Seed Start*: Center of Domain
- *Seed Radius*: 80
- *Seed Points*: 200

3.3 Szenario in VisIt

3.3.1 Gitter

Da die beiden Programme, VisIt und Paraview, auf der VTK (Visualization Toolkit) Bibliothek aufsetzen, ist es möglich, den im vorherigen Abschnitt verwendeten Datensatz mit der Export-Funktion von Paraview als .vtk File zu speichern, welches dann von VisIt problemlos eingelesen werden kann. Wir werden versuchen, mit VisIt die gleichen Ergebnisse zu erzielen, dabei wird sich zeigen, worin sich die Programme unterscheiden und welche Gemeinsamkeiten sie aufweisen. VisIt arbeitet mit Plots. Plots sind sichtbare Objekte, die von einem Datensatz erstellt wurden, und von VisIt in einem Visualisierungs-Fenster angezeigt werden können. Um eine Darstellung des Objekts als Drahtgitter (Wireframe) zu realisieren, benutzen wir den *Mesh-Plot*. In den Plot-Eigenschaften sollte der Opaque-Modus deaktiviert werden um das Gitter auf der gesamten äußeren Hülle des Objekts anzuzeigen. Festkörper bzw. Oberflächendarstellungen werden, wenn es nur um das Visualisieren der Silhouette bzw. der Form

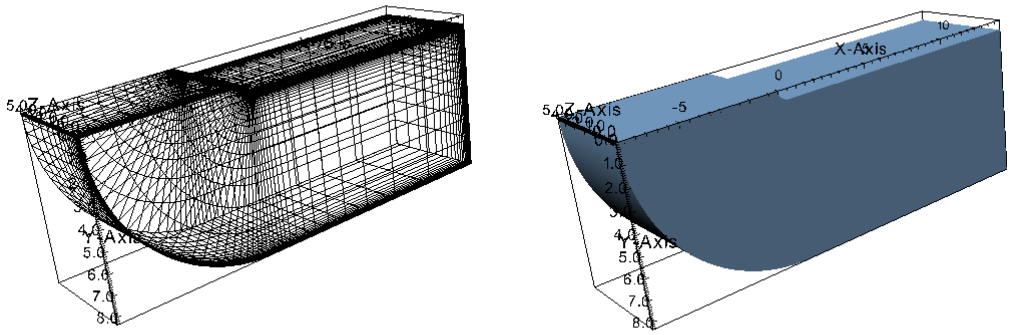


Abbildung 3.7: Links: *MeshPlot* des Datensatzes ohne interne Gitterelemente. Rechts: *SubsetPlot* für das Subset *Mesh*

des Objekts geht, am besten mit dem Subset-Plot umgesetzt. Die Abbildung 3.7 zeigt beide Plots.

Ein Schnitt durch den Körper lässt sich in VisIt mit dem Clip-Operator durchführen. Er ist bei den Operatoren im Untermenü *Selection* zu finden. Hat man einen Datensatz mit einem beliebigen Plot über den Button *Draw* zeichnen lassen, kann die Darstellung mit Operatoren noch angepasst werden. Der Clip Operator ist standardmäßig so eingestellt, dass er eine Ebene durch das Objekt legt und alles was sich ober- oder unterhalb der Ebene (definiert durch die Ebenennormale) befindet, aus der Darstellung entfernt. Es muss erst ein initiales Anwenden des Operators geschehen, bevor mit dem *Plane Tool* des Visualisierungs-Fensters die Ebene interaktiv in der Domäne verschoben werden kann. Darin unterscheidet sich VisIt von den anderen Programmen, da diese schon vor dem ersten Anwenden des Schnitts eine semi-transparente Ebene anzeigen, die dann schon direkt an die richtige Stelle verschoben werden kann.

Zum Erstellen von Schnittebenen wählt man im Operatoren-Menü unter *Slicing* > *Slice* aus. Standardmäßig wird eine Schnittebene orthogonal zu einer der Hauptachsen und eine Projektion in eine niederdimensionale Domäne durchgeführt. Mit dem *Plane Tool* kann auch hier die Schnittebene beliebig verschoben werden. Abbildung 3.8 zeigt das Objekt nach Anwenden der beiden Operatoren.

3.3.2 Skalardaten

Um nun die Eigenschaften der im Datensatz vorhandenen Skalarfelder zu visualisieren, müssen wir einen anderen Plot wählen. Ein Plot, der unterschiedliche Skalarwerte/Bereiche auf verschiedene Farben abbildet, um die ortsabhängige Änderung der Skalarwerte zu erkennen. Dafür ist der Pseudocolor-Plot gedacht. Wir ersetzen also den Subset- durch den Pseudocolor-Plot und wenden wieder wie im vorherigen Schritt den Slice-Operator an um das Skalarfeld der Dichte als Farben auf einer Schnittebene darzustellen.

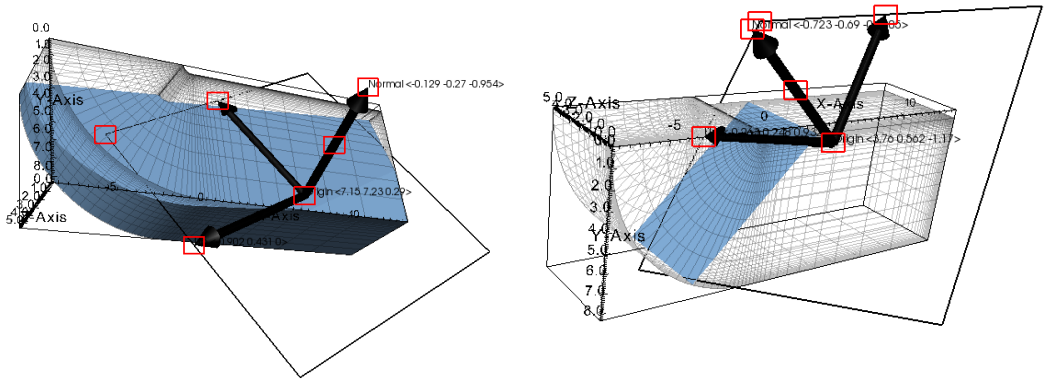


Abbildung 3.8: Links: *Clip*-Operator, der das Objekt entlang einer Ebene schneidet. Rechts: *Slice*-Operator. Die drei Pfeile sind die Vektoren, die die Ebene aufspannen und der zugehörige Normalenvektor

Zum Variieren das Farbkodierung kann leicht über die *Pseudocolor Plot Attributes* eine passende Farbtabelle nach Belieben gewählt werden. VisIt bietet hier mehr als 15 verschiedene Tabellen an. Zusätzlich kann über die *Scale*-Funktion das Mapping der Farben auf die Skalarwerte beeinflusst werden. Ändert man nichts an der Einstellung wird ein lineares Mapping angewendet. Das logarithmische wird verwendet um wenige Skalarwerte (Bereiche) auf viele Farben abzubilden. Das *Skew*-Mapping verwendet eine exponentielle Funktion die auf einem *Skew*-Faktor basiert, der vom Benutzer eingestellt werden kann. Wählt man einen Faktor größer oder kleiner eins, wird das untere bzw. obere Ende der Skalarwerte einem großen Bereich an Farben zugeordnet. In Abbildung 3.9 wurde die Farbtabelle von *hot* auf *hot desaturated* und der *Skew*-Faktor auf 0.5 geändert.

Das Darstellen von Isoflächen ist mit dem *Contour*-Plot möglich. Die wichtigsten Optionen in den *Contour Plot Attributes* sind das Festlegen der Selektion über den Punkt *Select by*. *N levels* legt die Anzahl an darzustellenden Isoflächen fest, die dann standardmäßig equidistant über den Skalarwertebereich verteilt werden. Mit *Value(s)* können alle gewünschten Isowerte von Hand festgelegt werden und mit *Percent(s)* kann die Auswahl der Isowerte prozentual an den Skalarwerten bestimmt werden. Für jede Isofläche können die Farb- und Transparenzwerte vom Benutzer gewählt werden. In Abbildung 3.10 wurden zwei Skalarfelder des Datensatzes mit dem *Contour-Plot* visualisiert. Im linken Bild wurde das *Stagnation Energy* Feld mit den Isowerten 7.0 und 11.0 verwendet. Das rechte Bild zeigt das Skalarfeld *Density* gerendert als Isofläche für Isowerte 1.2 und 1.8. In beiden Plots wurde eine semi-transparente Darstellung verwendet um die überlagerten Flächen besser erkennen zu können.

Für die Volumenvisualisierung bietet VisIt den gleichnamigen Plot *Volume* an. Es kann hierbei in den Plot-Eigenschaften aus mehreren Render-Methoden ausgewählt werden zu denen Splatting, Ray-Casting, 3D-Texturing und SLIVR zählen. Jede dieser Methoden kann von der Genauigkeit her im Unterpunkt Sampling eingestellt werden. Beispielweise die Anzahl der

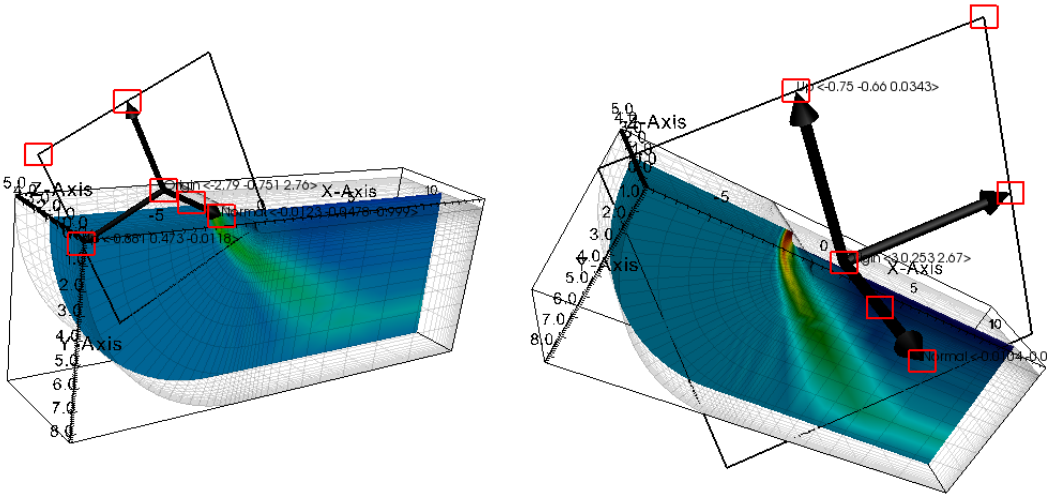


Abbildung 3.9: Links: Skalarwerte des Arrays *Density* als Farben auf der Schnittfläche. Rechts: Geänderte Farbtabelle durch das *Skew*-Mapping

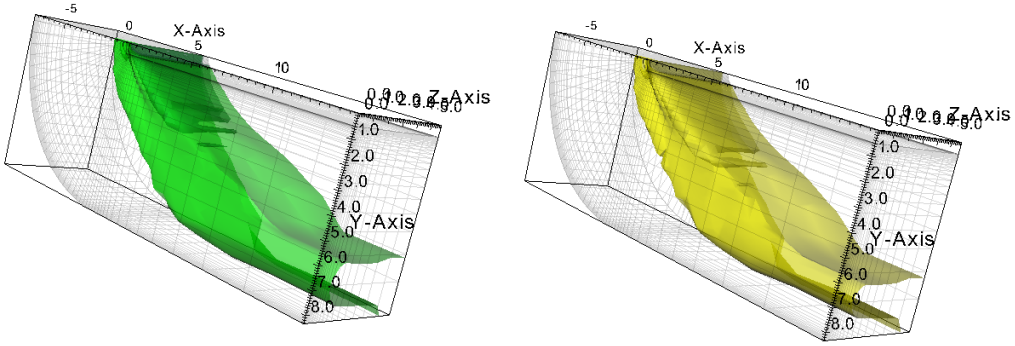


Abbildung 3.10: Links: *Contour*-Plot mit zwei Isowerten, dargestellt in hell- und dunkelgrün. Rechts: Zwei Isoflächen über das Array *Density*

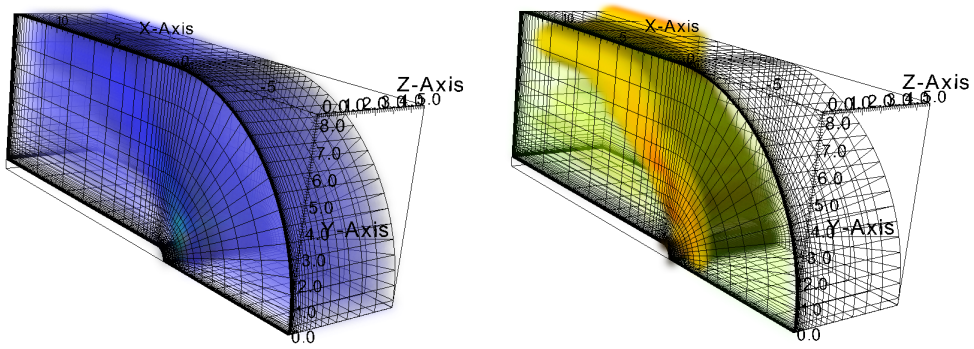


Abbildung 3.11: Links: Volumenvisualisierung mittels Splatting mit 50000 Samples , Rechts: gleiches Renderingverfahren mit angepasster Transferfunktion

Slices eines 3D-Texturing-Vorgangs. Der wichtigste Punkt ist das Auswählen und Einstellen einer geeigneten Transfer-Funktion mit Farbtabellen, Skalierungsfaktoren und Opazitätswerten.

3.3.3 Vektordaten

Für die Darstellung eines Vektorfeldes verwenden wir einen Datensatz der das Geschwindigkeitsfeld eines modellierten Tornados enthält. Will man die Geschwindigkeit einfach als Pfeile darstellen, bietet sich der Vector-Plot an. Das Visualisieren des Unterschiedes in den Beträgen der Geschwindigkeiten der Vektoren ist mit dem *Vector-Plot* auf zwei kombinierbare Arten möglich. Zum einen kann ein Skalierungsfaktor in Abhängigkeit des Geschwindigkeitsbetrages sowohl für Pfeillänge als auch Kopfgröße eingestellt werden, zum anderen können die unterschiedlichen Geschwindigkeiten auch in Farben der Pfeile kodiert werden, wofür VisIt seine Standard-Farbtabellen zur Verfügung stellt.

Wir wollen das Vektorfeld nur auf einer Schnittebene darstellen und wenden daher wie in den vorherigen Schritten den Slice-Operator an und verschieben die Schnittebene mit dem Plane-Tool das Visualisierungsfenster an die gewünschte Position. Um den visuellen Eindruck noch zu verbessern, wählen wir in den Plot-Eigenschaften den *Origin* für alle Vektoren als *Middle*, dann verlaufen alle Vektoren mittig durch die Gitterpunkte.

Abbildung 3.12 zeigt das Vektorfeld auf einer zur Grundfläche parallelen Schnittebene. Die Vektoren haben eine einheitliche grüne Farbe, es wurde nur ein Skalierung anhand des Betrags gewählt.

Stromlinien werden mit VisIt mit dem gleichnamigen Plot realisiert. Man kann dabei die *Seedpoints* (die Startpunkte für die Berechnung der Stromlinien) auf vielen verschiedenen Geometrien verteilen. Ein einzelner Punkt, eine Ebene, eine Sphäre oder auch die ganze Bounding-Box und noch mehr sind als Auswahl vorhanden. Eine zufällige Verteilung innerhalb der Geometrien ist ebenso wie eine äquidistante entlang der Achsen möglich. Für die Farbkodierung der Stromlinien stehen auch hier VisIts Standard-Farbtabellen zur Verfügung.

VisIt gibt an, erkennen zu können, wenn sich Stromlinien immer wieder um einen kritischen Punkt bewegen (ein Punkt, an dem die Geschwindigkeit Null wird) und terminiert das Ver-

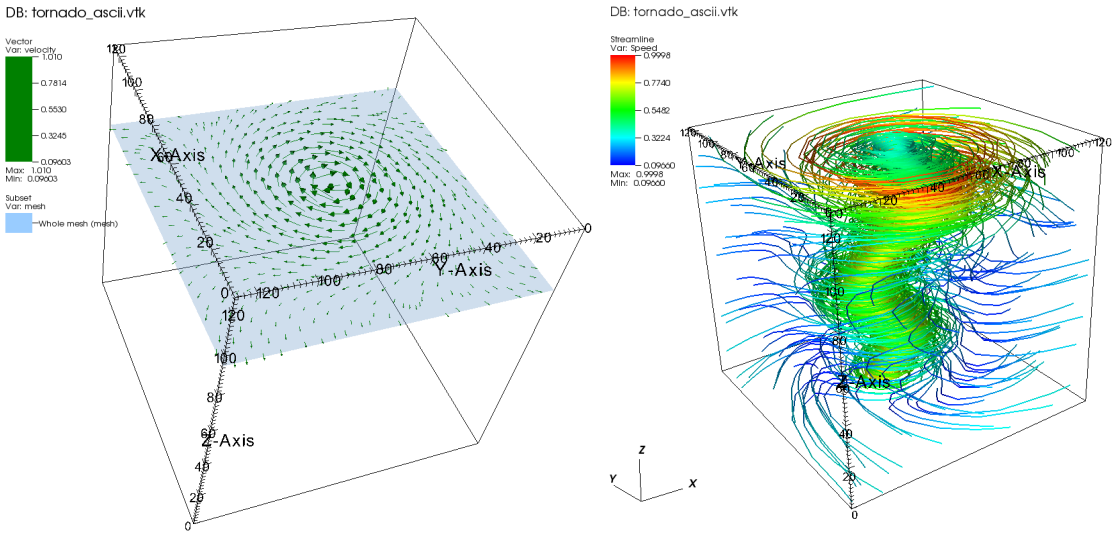


Abbildung 3.12: Links: Hellblauer *Subset*-Plot auf Ebene reduziert. *Vector*-Plot mit grünen Pfeilen auf derselben Ebene. Rechts: *Streamline*-Plot für einen Tornado-Datensatz mit farbiger Kodierung für den Betrag der Geschwindigkeit

fahren für die entsprechende Stromlinien. Auch *Stiffness* soll erkannt werden (Wenn sich eine Komponente eines Vektors sehr stark ändert, während die Anderen relativ konstant bleiben, so dass Toleranzkriterien nicht eingehalten werden können). Mit dem vorliegenden Tornado-Datensatz, der auch viele kritische Punkte enthält, hat dies sehr gut funktioniert. Es wurden entsprechende Warnungen angezeigt, aber die Visualisierung konnte trotzdem ausgeführt werden. Abbildung 3.12 zeigt rechts den Tornado-Datensatz mit Stromlinien für das Velocity-Feld (Seedpoints: 200, Streamline-Direction: both, Max. Nr. of Steps: 1000)

3.4 Szenario in Covise

3.4.1 Datensätze

Als Datensatz für das Szenario wurde in Covise ein Gitter, ein Skalar- und ein Vektorfeld einer Turbine verwendet. Das Skalarfeld stellt den Druck in der Turbine dar, das Vektorfeld die Geschwindigkeit.

3.4.2 Schritt 1: Darstellung des Drahtgitters

Um das Gitter als Körper darzustellen, braucht man drei Module. Neben einem *Loader*, der das Gitter einliest, und einem *Renderer* ist das Modul *Domainsurface* notwendig, welches zwischen *Loader* und *Renderer* geschaltet wird.

Im nächsten Schritt soll der Körper geschnitten werden. Dazu wird zwischen *Domainsurface* und dem Renderer das Modul *Cutgeometry* eingebaut. *Domainsurface* besitzt einen Ausgangsport, der es ermöglicht das Gitter in Linienform darzustellen. Dieser Port wird ebenfalls mit

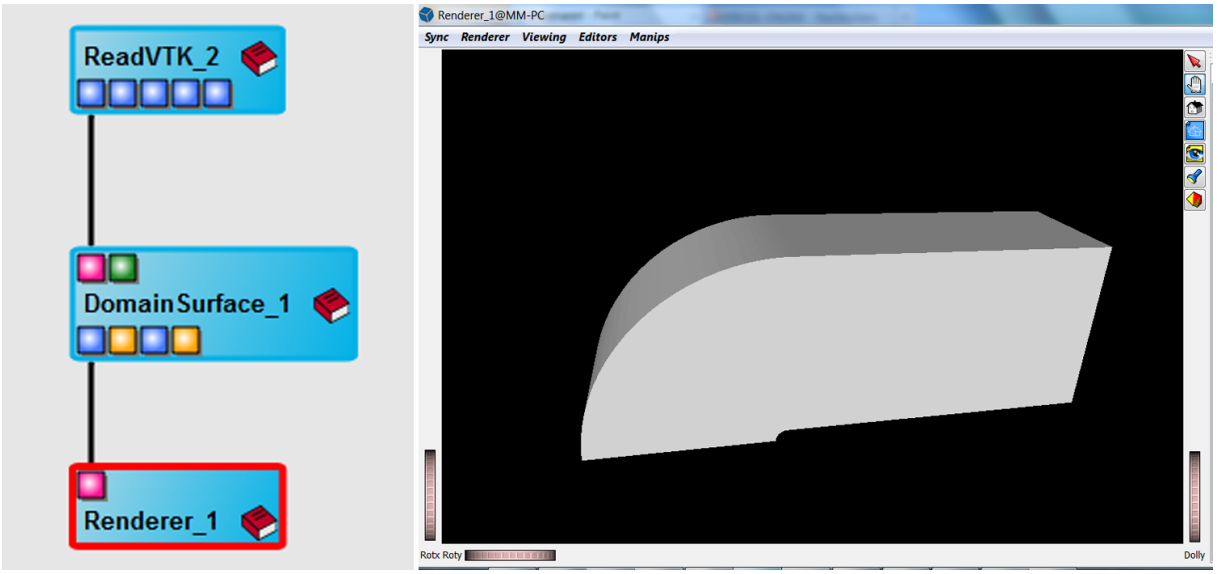


Abbildung 3.13: Graph und Visualisierung von Schritt 1 in Covise

dem Renderer verbunden, um beides zu visualisieren. Um die Schnittebene zu verändern, muss die Normale dieser Ebene verändert werden. Dies geschieht im Parameterfenster des *Cutgeometry-Moduls*.

Die Darstellung als Drahtgittermodell ist leider in der vorliegenden Version nicht möglich. Die Option gibt es zwar im Renderer, sie hat jedoch keine Auswirkung auf die Darstellung.

3.4.3 Schritt 2: Darstellung von Skalardaten

Zuerst soll das Skalarfeld für den Luftdruck auf einer Schnittebene dargestellt werden. Dazu wird das Modul *Cuttingsurface* benötigt. Neben dem Gitter aus Schritt 1, wird zusätzlich das Skalarfeld über einen Coviseloader mit *Cuttingsurface* verbunden. Die Ausgabe von *Cuttingsurface* besteht aus der Geometrie der Schnittebene und den Daten die darauf dargestellt werden. Diese Daten müssen nun noch auf Farben abgebildet werden, dies ermöglicht das Modul *Colors*. Werte können dabei auf einen Farbverlauf abgebildet werden.

Geometrie und Farben müssen nun wieder zusammengeführt werden. Das erledigt das Modul *Collect*. Der Ausgang von *Collect* wird nun mit dem Renderer verbunden. Zusätzlich wird wieder die geschnittene Geometrie des Gitters dargestellt.

Als nächstes soll der Betrag des Vektorfelds auf derselben Ebene gezeigt werden. Ersetzt man einfach das Skalarfeld im *Loader* durch das Vektorfeld wird man einen Fehler erhalten. Man muss zuerst das Vektorfeld in ein Skalarfeld für den Betrag umwandeln. Dies erledigt in Covise das Modul *Vectorscal*.

Das Skalarfeld kann jedoch nicht nur auf einer Schnittebene, sondern auch als Isofläche mit dem gleichnamigen Modul visualisiert werden. Wir behalten den Renderer und die *Loader* und löschen die anderen Module. Mittels *Domainsurface* erzeugen wir aus der Domäne ein Drahtgittermodell. Das Modul *Isosurface* verbindet die *Loader* für das Gitter und den Druck

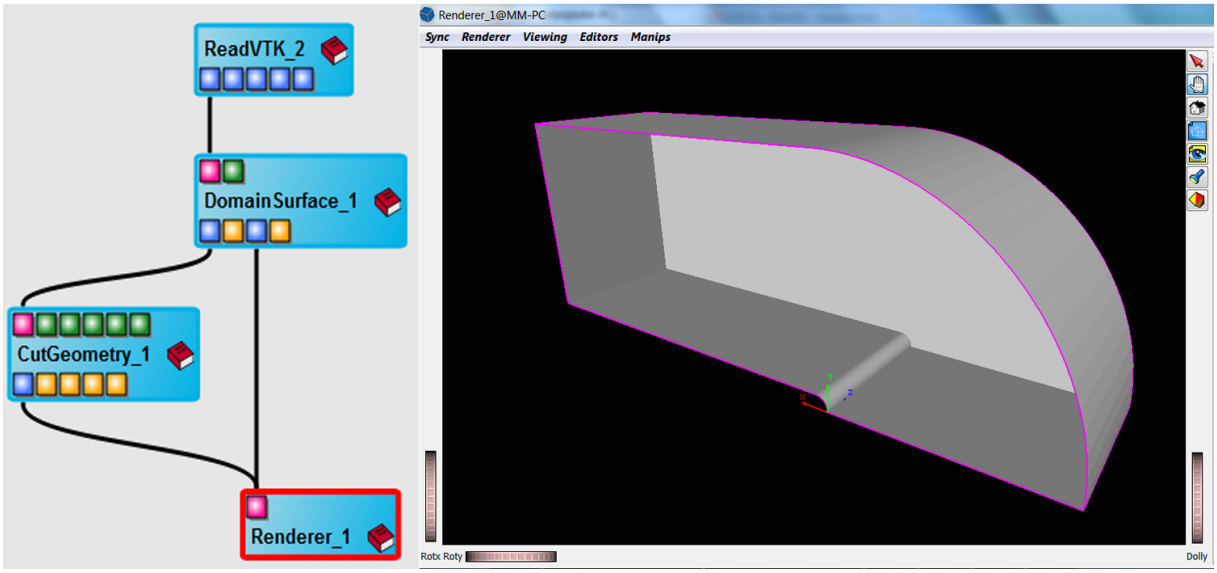


Abbildung 3.14: Schnitt durch einen Körper

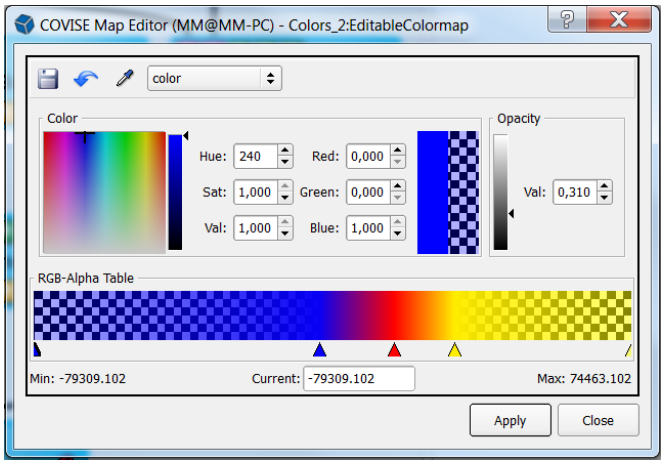


Abbildung 3.15: Das Modul *Color*

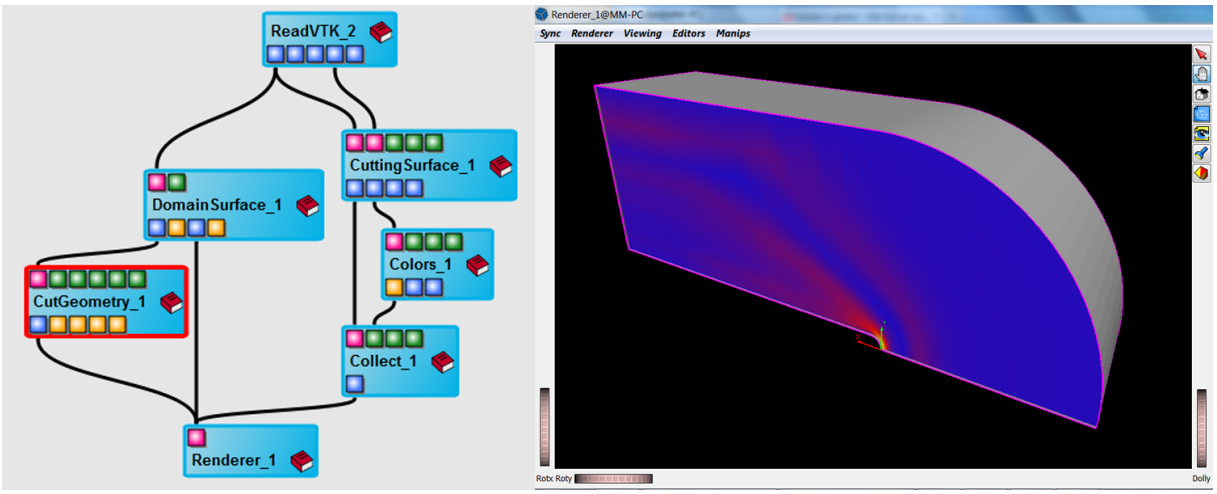


Abbildung 3.16: Visualisierung des Skalarfelds für den Luftdruck

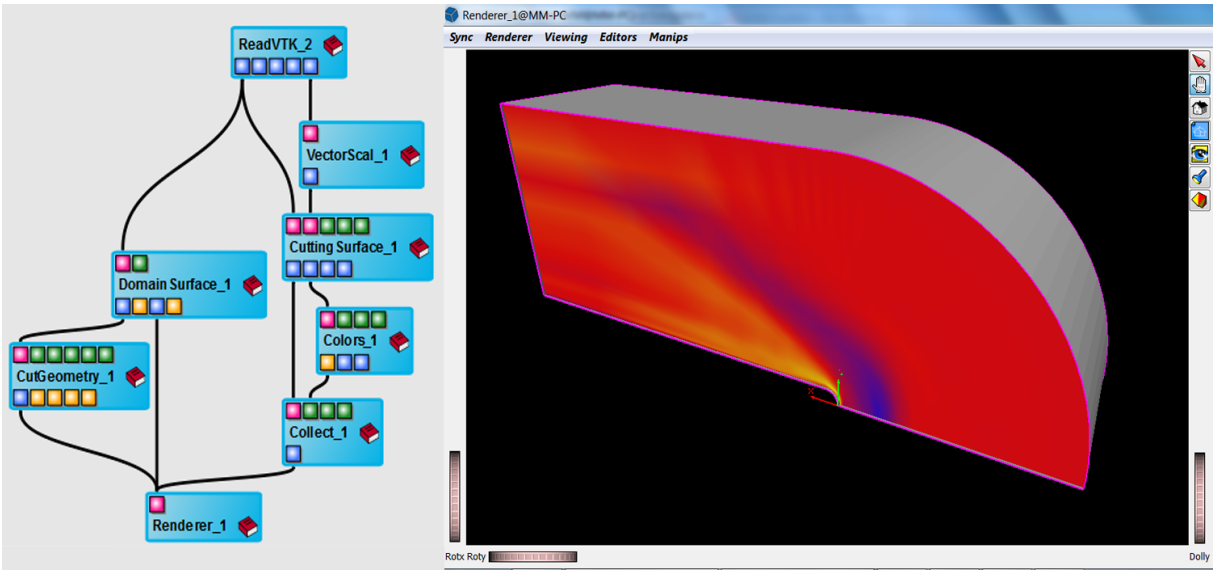


Abbildung 3.17: Darstellung der Beträge eines Vektorfelds in Covise

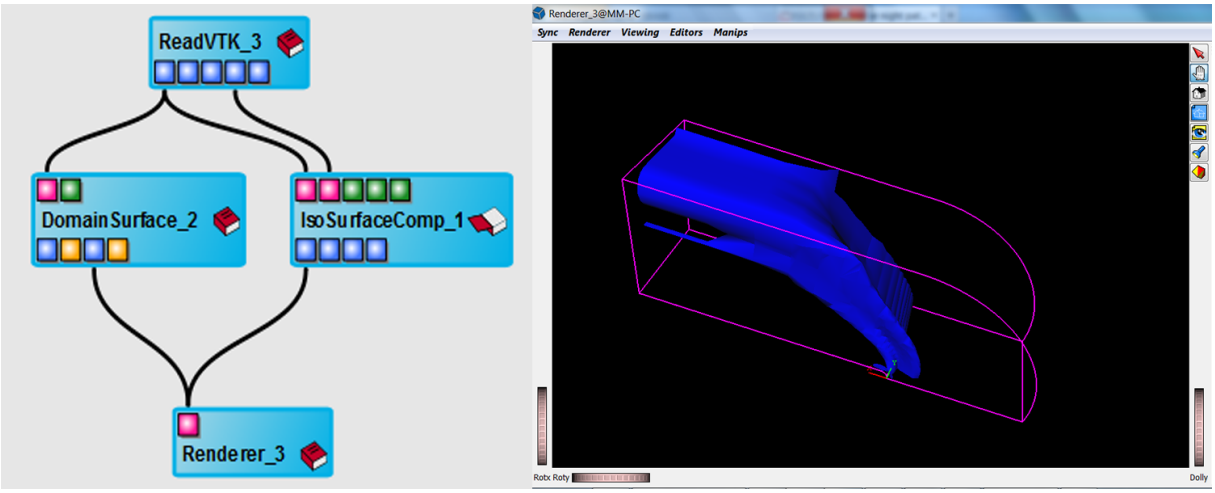


Abbildung 3.18: Darstellung der Beträge eines Vektorfelds mit Isoflächen

mit dem Renderer. Den Grenzwert der Isofläche kann man in den Parametern von *Isosurface* einstellen.

3.4.4 Schritt 3: Darstellung von Vektordaten

Leider war es trotz des entsprechenden Loaders nicht möglich, die Daten des Tornados in Covise zu laden. Darum verwenden wir hier dieselben Vektordaten wie aus den vorherigen Schritten. Die Vektordaten sollen als erstes als Pfeile auf einer Schnittebene gezeigt werden. Wir erweitern dazu die *Pipeline* aus dem entsprechenden Modell von Schritt 2, in welchem die Schnittebene mit dem Betrag des Vektorfelds als Farbkodierung visualisiert wird. Das Modul *Cuttingsurface* wird verwendet um das Vektorfeld auf die Schnittebene zu begrenzen und das Modul *Vectorfield* um es als Pfeile darzustellen. *Vectorfield* bietet sehr viele Parameter, unter anderem für die Länge der Pfeile.

Als nächstes sollen *Stromlinien* erzeugt werden. Mit dem Modul *Tracer* gelingt dies, allerdings mit erhöhtem Aufwand. Startpunkte, Längen usw. müssen in die entsprechenden Textfelder eingegeben werden, eine interaktive Bedienung mit der Maus wäre hier hilfreich. Da kein bemaßtes Koordinatensystem existiert, müssen die Werte erraten werden. Oft ist also ein größerer Aufwand nötig, um *Stromlinien* innerhalb des sichtbaren Bereichs zu erzeugen. Um die Stromlinien einzufärben, ist wieder ein *Collect-Modul* erforderlich, das die Geometrie der Linien und die Farben aus dem Modul *Color* wieder zusammenfasst und an den Renderer weitergibt. Das Modul *Color* erhält als Eingabe die Magnitude vom Modul *Tracer*.

3.5 Fazit

Tabelle 3.1 zeigt die Auswertung der einzelnen Schritte bzw. Visualisierungsmethoden des Szenarios. Die ausgewählten Operationen, die sich auf das **Anzeigen des Gitters** beziehen, werden in allen drei Programmen angeboten. In Paraview sind die Bedienelemente zu diesen

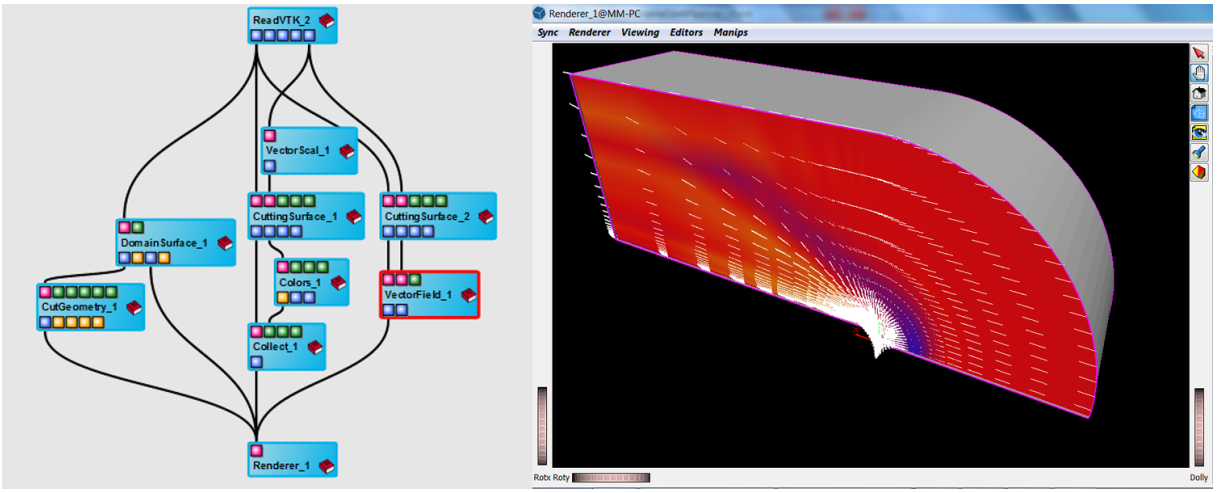


Abbildung 3.19: Darstellung eines Vektorfelds mit Pfeilen

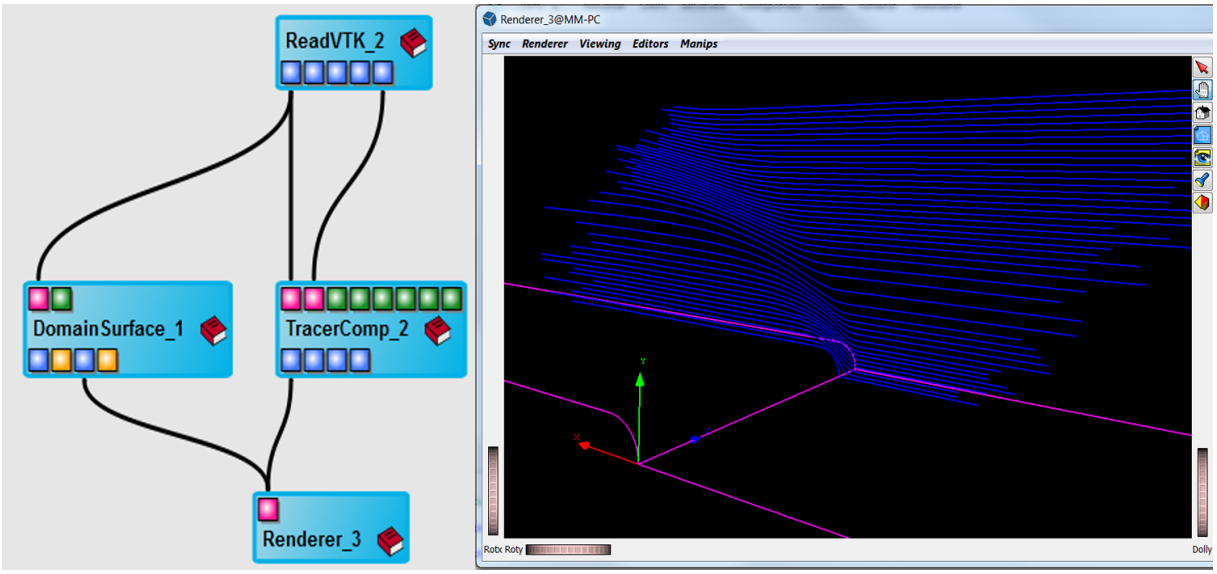


Abbildung 3.20: Darstellung von Stromlinien

Operationen direkt in der Hauptansicht zu finden, die Gitteroperationen sind also mit wenigen Mausklicks durchgeführt. Bei VisIt muss der Plot *Subset* benutzt werden, um eine Darstellung als Festkörper zu erreichen, was wir als keine intuitive Bezeichnung für diesen Anwendungsfall ansehen. Auch ist beim *Cut-* oder *Slice-Operator* das *Plane Tool* zum interaktiven Verschieben der Schnittebene nicht in den Schnittoperator integriert. Es muss im Kontextmenü des Visualisierungsfensters ausgewählt werden. Covise hat einen Bug im Renderer, der bewirkt, dass die Darstellung als Drahtgittermodell nicht direkt im Render-Modul ausgewählt werden kann. Es muss ein Umweg über ein spezielles Modul (*DomainSurfaces - Port: HighlightLines*) benutzt werden um eine Darstellung als Drahtgittermodell zu erreichen.

Die **Visualisierung von Skalardaten** wurde mittels Farbkodierung, Isoflächen und Volumenrendering durchgeführt. Paraview hatte keine Probleme mit der Visualisierung von Skalarda-

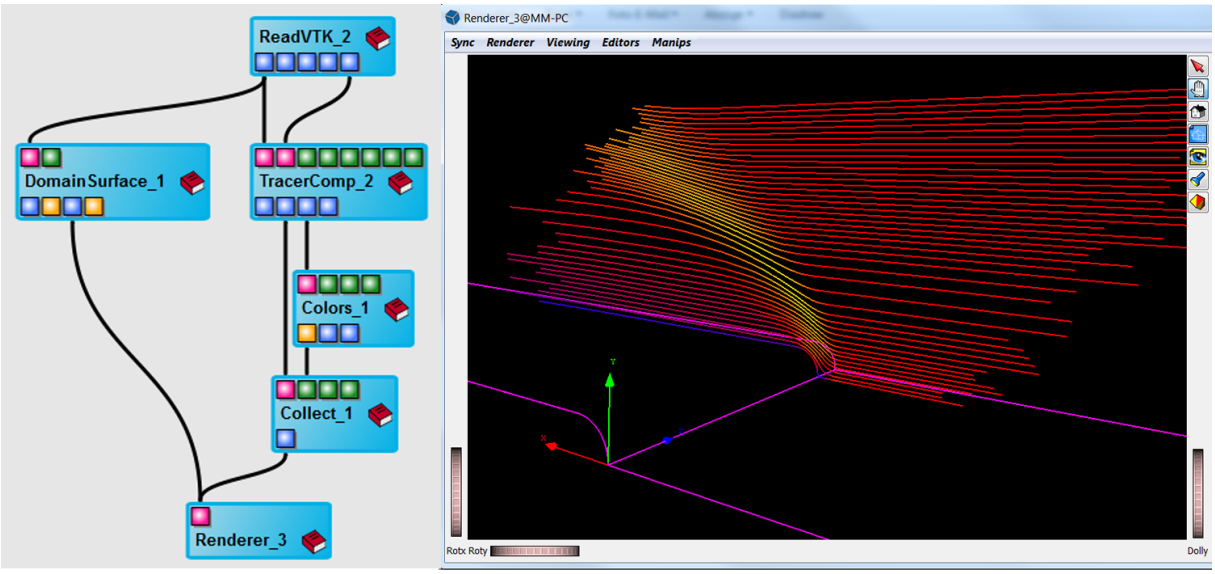


Abbildung 3.21: Visualisierung von Stromlinien mit Farben für die Stärke

ten. Lediglich bei den Isoflächen zeigte sich eine kleine Schwäche. In Paraview haben Isoflächen, die aus einer Operation stammen, standardmäßig die gleiche Farbe. Um mehrere Isoflächen farblich voneinander zu trennen, muss der Operator mehrmals angewandt werden, oder die Transferfunktion mühsam angepasst werden.

Bei VisIt ist die Bezeichnung *Pseudocolor* für einen Plot von Skalardaten irreführend. Außerdem unterstützt VisIt keine speziellen Darstellungsformen für *Backfaces*. Covise bietet Module für alle getesteten Funktionen. Auch hält sich die Zahl der benötigten Module für unsere Anwendungsfälle in Grenzen. Der einzig störende Faktor war, dass die Benutzeroberfläche von Covise gelegentlich nicht reagiert und nur mit einem Neustart des entsprechenden Moduls wieder bedienbar wird.

Die **Darstellung von Vektordaten** wurde über Pfeile auf einer Schnittebene sowie mittels *Stromlinien* durchgeführt. Vektordaten als Pfeile darzustellen, ist über die Funktionen *Glyph* (Paraview), *VectorPlot* (Visit) und das Modul *VectorField* (Covise) problemlos durchführbar.

Tabelle 3.1: Bewertung der Verfahren. Die Skala reicht von - - (unmöglich) bis ++ (intuitiv).

Anwendungsfall	Paraview	Visit	Covise
01 - Gitter: Darstellung als Festkörper	++	+	++
02 - Gitter: Darstellung als Drahtgittermodell	++	++	o
03 - Gitter: Schnitt durch den Körper	+	o	+
04 - Gitter: Erstellen einer Schnittebene	+	o	+
05 - Skalardaten: Skalarfeld als Farben auf der Schnittebene	++	++	++
06 - Skalardaten: Variation der Farbkodierung	++	++	++
07 - Skalardaten: Darstellung als Isofläche	+	++	o
08 - Skalardaten: Darstellung als Volumen	++	++	o
09 - Vektordaten: Geschwindigkeit als Pfeile auf der Schnittebene	+	+	+
10 - Vektordaten: Geschwindigkeit als *Stromlinien*	+	o	+
11 - Vektordaten: Betrag der Geschwindigkeit als farbige *Stromlinien*	+	+	++

Mit *Stromlinien* gab es in Paraview und Covise keine Probleme. In Visit ist der Stramline-Plot instabil. Es kann sein, dass der Prozess einfriert und VisIt neu gestartet werden muss.

4 Software-Architektur

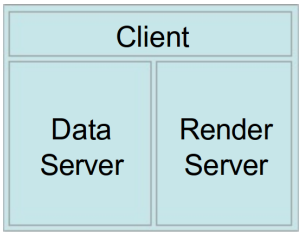
4.1 Paraview

Paraview ist als vorkompiliertes Paket für Windows (32 und 64 Bit), Linux (32 und 64 Bit) und Mac-Betriebssysteme (64 Bit) verfügbar. Das Gesamtsystem ist nach dem Client-Server-Prinzip implementiert. Es setzt sich aus drei Komponenten zusammen.

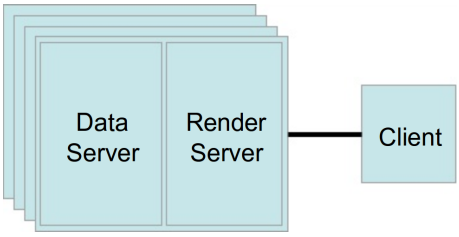
- **Daten-Server**: Diese Komponente stellt die Daten für die Visualisierung bereit. Abhängig vom geladenen Datensatz kann es sich hier beispielsweise um Gitterpunkte, Skalarfelder oder Vektorfelder handeln. Es können auch Kombinationen daraus auftreten. Außerdem ist der Daten-Server verantwortlich für die Filterung und Speicherung von Datensätzen. Der Daten-Server kann parallel betrieben werden.
- **Render-Server**: Der Render-Server kümmert sich ausschließlich um das Rendern der bereitgestellten Daten. Er kann wie der Daten-Server parallel betrieben werden. Dazu muss der Quellcode des Render-Servers mit der Option $PARAVIEW\_USE\_MPI = 1$ kompiliert werden.
- **Client**: Der Paraview-Client beinhaltet die GUI und die Kontrolle über die Objekte auf dem Daten-Server und Render-Server. Ohne selbst über Visualisierungsdaten zu verfügen, kann der Client Objekte auf den beiden Servern erzeugen, verarbeiten und löschen. So wird sichergestellt, dass die Skalierbarkeit auch bei großen Rechnernetzen erhalten bleibt, obwohl der Paraview-Client eine rein serielle Anwendung ist.

Diese drei Komponenten können sowohl verteilt als auch auf einer einzigen Maschine arbeiten. Somit ergeben sich drei Betriebsmodi für Paraview:

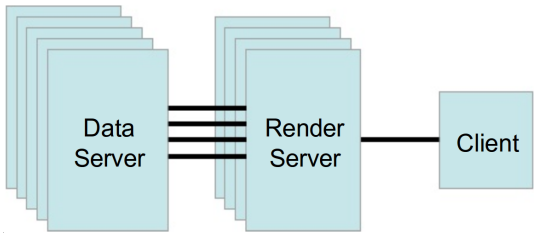
- **Standalone-Modus**: Der standardmäßige Betriebsmodus, wenn Paraview als vorkompiliertes Paket heruntergeladen und ausgeführt wird. Die Anwendung *paraview* wird gestartet. Darin ist sowohl der Client als auch ein Server enthalten.



- **Client-Server-Modus**: Client und Server werden auf separaten Rechnern ausgeführt. Auf dem einem Rechner wird die Anwendung *pvserver* gestartet. Dazu wird *mpirun* benötigt. Der Befehl $mpirun\ -np\ 4\ ./pvserver$ startet die Anwendung mit 4 Prozessen. Sie beinhaltet den Daten-Server und den Render-Server. Auf dem Client-Rechner kann nun *paraview* gestartet und eine Verbindung zu dem Server-Rechner hergestellt werden.



- **Client, Render Server, Data Server - Modus**: Alle Komponenten werden von eigenen Anwendungen realisiert und können auf verschiedenen Hardwareressourcen gestartet werden. Der Client hat eine einzige Socket-Verbindung zum Render-Server. Zwischen Render-Server und Daten-Server besteht eine Socket-Verbindung für jeden parallel laufenden Render-Prozess.
  Den Daten- und den Render-Server als getrennte Prozesse laufen zu lassen, lohnt sich in den seltensten Fällen. Dieser Modus war für Szenarien gedacht, in denen ein sehr leistungsstarkes Rechner-Cluster mit einem davon getrennten, kleineren Grafik-Rechner zusammenarbeitet. Der aufwändige Datenaustausch zwischen Daten-Server und Render-Server mindert jedoch stark die Vorteile einer solchen Konfiguration.



4.2 VisIt

VisIts Architektur besteht im Wesentlichen aus vier Hauptkomponenten. Die GUI stellt Menüs bereit, über die der Benutzer die zu visualisierenden Daten auswählen kann.

Der *Viewer* zeigt das Visualisierte in seinen *Vis-Fenstern* an und ist für das Überwachen von VisIts Zustand sowie die Kommunikation mit den anderen Komponenten verantwortlich.

Beide Komponenten, GUI und Viewer, sind dafür vorgesehen, lokal auf dem Client-PC ausgeführt zu werden, um von einer schnellen Grafik-Hardware zu profitieren.
Die anderen beiden Komponenten, *Database Server* und *Compute Engine*, können auch auf dem Client-PC laufen, werden aber meist remote auf einem leistungsstarken Parallelcomputer oder Cluster ausgeführt, auf dem die Daten auch erzeugt wurden. Der *Database Server* liest das Dateisystem des entfernten Rechners und gibt die Informationen an die GUI auf dem lokalen Rechner weiter, damit diese die Daten für den Benutzer zur Auswahl zur Verfügung stellen kann. Zusätzlich öffnet der *Database Server* auch die Dateien um die enthaltenen Variablen und andere Metadaten zu ermitteln.

Die *Compute Engine* ist die Komponente, die die Visualisierungsdaten ausliest, sie verarbeitet und anschließend Bilder oder Geometrien an den *Viewer* schickt, damit dieser sie unter Ver-

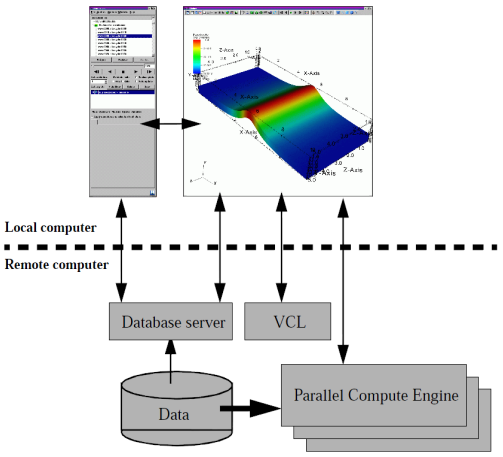


Abbildung 4.1: VisIts Architektur mit den wesentlichen Komponenten

wendung der schnellen Grafik-Hardware des lokalen Rechners darstellen kann. Abbildung 4.1 zeigt einen Überblick über VisIts Architektur-Komponenten.

4.3 Covise

Die Verteilung aller Funktionalitäten in Module ist auch in der Softwarearchitektur wieder zu finden. Es sind alle Module gekapselt, dies entspricht einer Plugin-Architektur. Covise ist für Linux und Windows in 32- und 64-Bit-Versionen, sowie für Mac OS X verfügbar. Die Unterteilung in Module geht bis hin zu den Prozessen im Betriebssystem. Jedes Modul, das verwendet wird, erhält einen eigenen Prozess. Ein Absturz eines Moduls führt somit nicht zwangsläufig zum Absturz von Covise. Einzelne Prozesse die nicht mehr reagieren lassen sich im Task Manager beenden ohne die gesamte Covise-Umgebung zu schließen. Verschiedene Module kommunizieren über TCP/IP, es spielt also prinzipiell keine Rolle, ob die Module lokal oder auf einem anderen Rechner im Netzwerk laufen.

5 Datenformate

Ein wichtiger Aspekt der wissenschaftlichen Arbeit ist der Austausch von Informationen mit Kollegen auch über sprachliche Grenzen hinweg. Gleiches gilt auch für den Austausch von digitalen Informationen. Hierfür sollten etablierte und weit verbreitete Daten-Formate verwendet werden, um einen mühelosen und auch plattformunabhängigen Austausch zu gewährleisten.

In diesem Abschnitt wollen wir auf die Datenschnittstellen und unterstützten Datenformate der jeweiligen Programme eingehen.

5.1 Rohdaten-Formate

Das Lesen von Rohdaten ist eine schnelle Methode, um beliebige Datensätze zu laden.

5.1.1 Paraview

5.1.1.1 RAW-Binary

Paraview beherrscht das Einlesen von Daten im RAW-Format. Dazu müssen einige Angaben dazu gemacht werden, wie die Datei interpretiert werden soll. Es muss unter anderem der Datentyp (short, int, usw.), die *byte order* (little oder big endian), die Anzahl der Dimensionen und der Definitionsbereich des Datensatzes in jeder Dimension angegeben werden.

5.1.1.2 RAW-ASCII

Eine Reihe von Zahlen, die durch ein Leerzeichen getrennt sind, wird als Skalarfeld interpretiert. Die Position der Zahl in der Datei bestimmt die Koordinate im 1D-, 2D oder 3D-Raum. Ein Beispiel:

Eine Datei mit dem Inhalt

```
01 02 03 04 05 06 07 .. .. 20 21 22 23 24
```

Interpretation als 2D-Feld 8x3:

```
01 02 03 04 05 06 07 08
09 10 11 12 13 14 15 16
17 18 19 20 21 22 23 24
```

Interpretation als 3D-Feld 4x3x2, also 2 *Slices* mit der Auflösung 4x3:

```
01 02 03 04        13 14 15 16
05 06 07 08        17 18 19 20
09 10 11 12        21 22 23 24
```

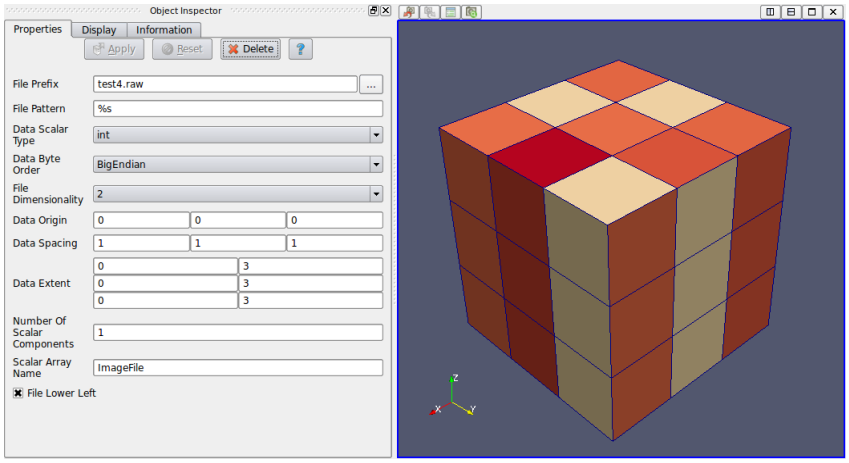


Abbildung 5.1: Das Bild zeigt die 3x3x3-Repräsentation eines Datensatzes, die aus einer Reihe von 64 Zahlen besteht. Die Gouraud-Interpolation über die Skalarwerte wurde deaktiviert, um die Flächen in einer einheitlichen Farbe darzustellen.

Um die Datei korrekt interpretieren zu können, muss beim Laden die Dimension und die Größe des resultierenden Feldes korrekt angegeben werden. Über die Einstellung *Dimensionality* lässt sich festlegen, ob es sich um *Slices* oder Volumendaten handelt.

5.1.1.3 CSV

Durch Komma (oder ein anderes definiertes Zeichen) getrennte Zahlen werden als Tabelle interpretiert. Eine neue Zeile in der Datei bedeutet eine neue Zeile in der Tabelle. Die Werte für die X-,Y-,Z-Koordinaten müssen in verschiedenen Spalten stehen.

Wird in Paraview eine Datei mit der Endung *.csv geladen, wird zunächst die resultierende Tabelle angezeigt. Anschließend kann der Benutzer entscheiden, was mit den Daten geschehen soll. Über die Filter *Table-To-Structured-Grid* oder *Table-To-Points* werden die Daten als Gitter oder Punkte dargestellt.

5.1.2 VisIt

5.1.2.1 RAW-ASCII

VisIt unterstützt einige simple ASCII Dateiformate um ein schnelles Speichern und Einlesen von einfachsten Visualisierungsdaten zu ermöglichen. Es gibt dabei nur wenige Dinge die beachtet werden müssen. Um sicherzustellen, dass VisIt die Daten richtig interpretiert, können bei den entsprechenden Reader-Plugins die *Default Options* einfach angepasst werden. Zur Zeit sind folgende Plugins implementiert:

Curve2D Interpretiert eine Liste von X-Y-Paaren als Kurve und erstellt einen passenden Plot

Lines Liest eine Liste von jeweils drei kommaseparierten Floating-Point Werten (X-,Y- und Z-Koordinate eines Punkts) pro Zeile ein und erstellt daraus eine 3D Kurve

PlainText Erlaubt das Einlesen von duch Komma oder Leerzeichen getrennten Werten die als 2D-Gitter, 1D-Kurve oder als 2- oder 3D-Punktnetz interpretiert werden können, einstellbar über die *Default Reader Options*

Point3D Reader für einfach strukturierte Dateien mit 3D Punktdaten. Jede Zeile enthält der Reihe nach X-,Y-,Z-Koordinate und den entsprechenden Skalarwert

Am Beispiel des *PlainText-Reader* Plugins soll demonstriert werden, wie schnell und einfach der Benutzer seine selbst erstellten Daten in VisIt importieren kann.

Die Formel $f(x,y) = e^{-r}cos(6 \cdot r)$, $r = \sqrt{x^2 + y^2}$ soll als Höhenfeld dargestellt werden.

Dafür wird ein wenige Zeilen langes Python Skript verwendet, das die Funktionswerte in eine CSV-Datei schreibt. Zu beachten ist hier, dass die Koordinaten nicht explizit angegeben werden können. Es ergibt sich immer ein reguläres Gitter, bei dem die Koordinaten durch die Anzahl der Spalten und Zeilen begrenzt wird.

Das folgende Skript wertet die oben genannte Funktion in den Intervallen -5 <= x <= 5, -5 <= y <= 5 mit einer Schrittweite von 0.1 aus und schreibt die Werte in die angegebene Datei.

```
import math
n = 100
f = open("heightfield.csv", "wt")
f.write("value");
for i in range(n):
    f.write("\n");
    for j in range(n):
        x = -5 + (i * 0.1)
        y = -5 + (j * 0.1)
        r = math.sqrt(math.pow(x,2) + math.pow(y,2))
        fvalue = math.exp(-r) * math.cos(6*r)
        f.write("%g " % (fvalue ))
f.close()
```

Die so erstellte CSV-Datei kann nun in VisIt mit dem Plugin *PlainText* geöffnet werden, indem in den *Default Open Options* das Data Layout auf *2D Array* eingestellt wird. Zum Rendern eines Höhenfeldes bietet VisIt den surface-Plot, der 2D Daten erwartet. Abbildung 5.2 zeigt das Resultat des Rendervorgangs.

5.1.2.2 RAW-Binary

Es gibt zwar einige Binär-Dateiformate die VisIt einlesen kann, beispielsweise *BOV* (Brick of Values), die Bezeichnung Rohdaten-Formate trifft hier aber nicht zu, denn der Benutzer kann bei ihnen nicht selbst festlegen, wie die Daten zu interpretieren sind. Die Byte-Reihenfolge, die Anzahl an Bytes für einen Skalarwert etc., all das ist schon vorher durch das Dateiformat spezifiziert und nicht variierbar.

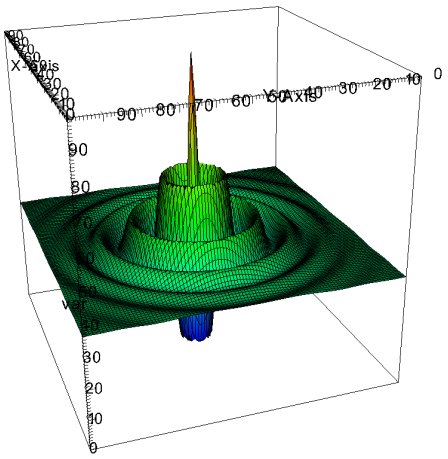


Abbildung 5.2: Höhenfeld der Funktion $f(x,y) = e^{-r}cos(6 \cdot r)$, $r = \sqrt{x^2 + y^2}$

5.1.3 Covise

5.1.3.1 RAW-Binary

Für ein binäres RAW-Format konnte kein passendes Modul gefunden werden.

5.1.3.2 RAW-ASCII

Mit dem Modul *ReadASCII* ist es möglich, aus Textdateien Daten zu lesen. Dabei kann das Schema, in dem Werte in der Datei stehen, angegeben werden. Man wählt aus, was das erste, zweite, dritte Element, usw. in der Datei darstellt und gibt das Trennzeichen an. Die Datei wird danach entsprechend eingelesen.

5.2 Standard-Formate

5.3 Programmspezifische Formate

5.3.1 Paraview

Das programmeigene Format für Paraview heißt *ParaView Data Format* und hat die Erweiterung **.pvd*. In diesem Format ist jedes Gitter möglich, das in Paraview unterstützt wird: Polygongitter, Rechtecksgitter, gekrümmte Gitter und unstrukturierte Gitter.

Dateiformat	Paraview	Visit	Covise
VTK	ja	ja	eingeschränkt
RAW	ja	ja	ja, sogar konfigurierbar

5.3.2 VisIt

VisIt arbeitet mit Daten im Silo-Format. Daher ist es bei der Arbeit mit VisIt eine gute Entscheidung, seine Visualisierungsdaten in diesem Format zu speichern. Silo ist eine Library, welche eine API für das Lesen und Schreiben von einer großen Breite an wissenschaftlichen Daten bereitstellt. Silo unterstützt dabei u.a. die Verarbeitung von gitterlosen Punktwolken, strukturierten Gittern, unstrukturierten Gittern, deren Elemente aus beliebigen Polygonen aufgebaut sind, lokal verfeinerte Netze und die Aufspaltung der Gitter in mehrere Teilmengen (*Subsets*) für die Kodierung von unterschiedlichen Materialien im Datensatz. Silos Architektur ist in zwei große Bereiche aufgeteilt. Das high-level API und eine lower-level I/O Implementierung, die *Driver* genannt wird. Silo unterstützt mehrere *I/O Driver*. Zu den gängigsten gehören *HDF5* (Hierarchical Data Format 5) und *PDB* (Portable Data Base). Silo setzt somit auf andere low-level Datei-Formate auf. Die Silo Library kann mit den Programmiersprachen C, Fortran und Python verwendet werden.

5.3.3 Covise

Covise besitzt eine Vielzahl von Loadermodulen, um verschiedene Datenformate für Punktmengen, Vektorfelder etc. und für die Volumendarstellung einzulesen. Es ist auch möglich, eigene Module für andere Datenformate zu schreiben. Die wichtigsten *Loadermodule* sind Folgende:

ReadITK Lesen der Formate .png, .jpg und .tiff zur Erstellung von Volumendaten

ReadObj Lesen von Dateien im *Wavefront OBJ* Format, inklusive Materialdateien im Format .mtl

ReadVolume Lesen von Volumendaten in den Raw-Formaten .dat und .rvf, dem Format .xvf und .tif (3D TIF)

ReadXYZ Lesen von Punktdaten im XYZ-Format

6 Erweiterbarkeit

Dieser Abschnitt behandelt die Erweiterbarkeit der Programme. Alle untersuchten Programme lassen sich durch Plugins oder Module erweitern. Typische Bereiche für Erweiterungen sind Reader für neue Dateiformate oder die Implementierung von neuen Filtern.

6.1 Plugins

6.1.1 Paraview

Paraview bietet die Möglichkeit, den Funktionsumfang mittels Plugins zu erweitern. So können beispielsweise neue Reader, Writer, Filter, GUI-Komponenten, Ansichten oder Repräsentationen hinzugefügt werden. Da Paraview eine Client-Server-Architektur besitzt, gibt es zwei Arten von Plugins. Server-Plugins sind typischerweise neue Reader, Writer oder Filter. Client-Plugins erweitern die GUI. Zum Beispiel durch neue Panels oder Toolbars.

Es kommt oft vor, dass Plugins für Client und Server bestimmt sind, wie etwa ein Filter mit einer zugehörigen Toolbar, mit der sich Attribute des Filters konfigurieren lassen. Diese Plugins müssen auf dem Client und auf dem Server installiert werden.

6.1.2 VisIt

VisIt legt Wert auf Erweiterbarkeit und verfolgt daher ein Plugin-Konzept. So kann eine installierte Version von VisIt mit neuen Funktionalitäten aufgewertet werden. *Database Reader und Writer*, *Plots* und Operatoren sind im aktuellen Release als Plugins umgesetzt:

Mit *Database Readern* kann VisIt um die Unterstützung bisher nicht lesbarer Dateiformate erweitert werden. Gleiches gilt für die *Writer* und das Exportieren von Daten.

Mit Operatoren können weitere Arten der Daten-Manipulation implementiert werden.

Plots sind die Umsetzung einer Rendering-Methode. Zu beachten ist hier, dass Plots oft einen impliziten Operator besitzen. Das beste Beispiel ist der *Contour Plot*. Er rendert Isoflächen, muss diese aber erst extrahieren um das zu tun. Das Extrahieren der Isoflächen ist also seine implizite Operation. VisIt bietet aber noch mehr Möglichkeiten. Mit den *Expressions* wird dem Benutzer ein umfangreiches Werkzeug bereitgestellt.

Sie ermöglichen es zum Beispiel, aus den vorhandenen Daten abgeleitete Größen zu erzeugen. Variablen die aus *Expressions* erstellt wurden, werden von VisIt gleich behandelt wie die Variablen eines eingelesenen Datensatzes. Sie werden im Plot-Menü aufgelistet und können mit ihnen visualisert werden.

VisIt stellt zum Definieren von *Expressions* einen sehr großen Satz an Funktionen bereit. Von mathematischen Funktionen wie *abs, min, max, sqrt, ...* über spezielle Vektor-Funktionen wie *magnitude, normalize, etc.* bis hin zu logischen Funktionen wie *and, not, or*.
Abbildung 6.1 zeigt ein Beispiel für die Anwendung von *Expressions*.

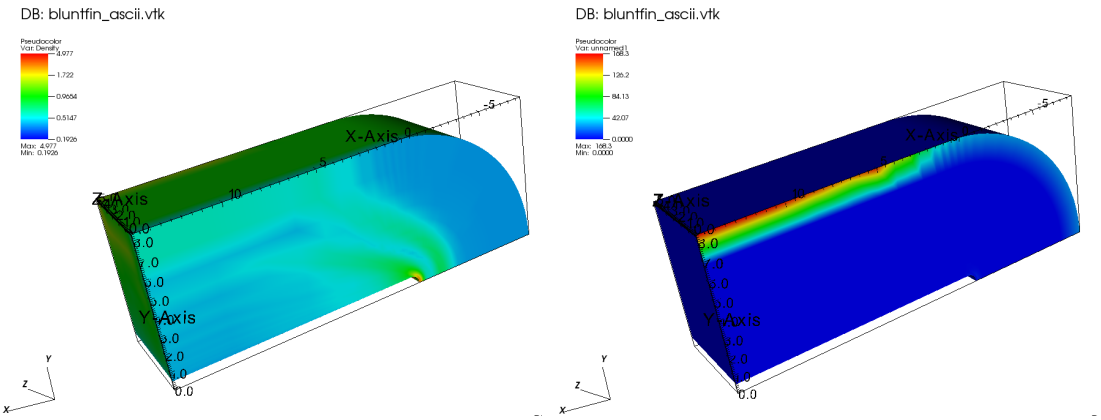


Abbildung 6.1: Links: Pseudocolor-Plot für Variable *density* Rechts: Mit *Expressions* abgeleitete Variable - Betrag des Dichtegradienten - mit Pseudocolor-Plot visualisiert

Wir verwenden hierfür den Datensatz des Beispielszenarios. Im linken Teil des Bildes wurde ein *Pseudocolor-Plot* für die Variable *density* verwendet. Im rechten Bild soll die abgeleitete Variable - Betrag des Dichtegradients - visualisiert werden, also die Bereiche der größten Änderung der Dichte.

Das lässt sich mit dem Ausdruck, *magnitude(gradient(Density))*, erreichen.

6.1.3 Covise

Die Trennung jeglicher Funktionalität in Module kommt der Erweiterbarkeit zugute. Die Kapselung, sowohl auf funktionaler Ebene, als auch in der Architektur, kann von bestehenden Modulen auf neue, eigene Module, übertragen werden. Unternehmen, die Covise einsetzen, können die Funktionalität sehr einfach auf ihre eigenen Bedürfnisse anpassen. Durch die Ausrichtung der Software auf Erweiterbarkeit und die Modularisierung mit definierten Schnittstellen lassen sich entwickelte Module auch in neuen Versionen von Covise mit hoher Wahrscheinlichkeit einsetzen.

Die Covise Development Distribution enthält alle Bibliotheken und Headerfiles um Module zu entwickeln. Zwar muss die Mainmethode in C++ geschrieben werden, allerdings lässt sich auch *C* und *FORTRAN* Code einbinden. Die *libcoCore.lib* in der Covise API stellt die Basisbibliothek für Covisemodule dar, zuständig für Datenmanagement, Kommunikation und Prozessverwaltung. In *libcoAppl.lib* findet man Grundfunktionen für ein Modul zum Versenden von Nachrichten etc. Die libcoApi.lib baut auf diesen Grundbibliotheken auf und macht das Entwickeln einfacher und sicherer.

Die Ausführungssequenz eines Moduls ist immer identisch. Nach dem Durchlaufen des Konstruktors und einer Initialisierungsmethode beginnt das Hauptprogramm, bestehend aus drei Teilen, die durch *Eventhandler* realisiert sind. *Param()* wird aufgerufen wenn Parameter des Moduls geändert werden. *SockData()* verwaltet die Verbindungen zu anderen Modulen und in *compute()* werden Eingangsdaten eingelesen, verarbeitet und in den Speicher geschrieben. Der Zugriff auf die Daten ist lediglich in dieser Methode möglich.

7 Remote-Visualisierung und verteilte Visualisierung

Im Bereich der numerischen Simulation können die zu verarbeitenden Daten schnell eine schwer zu handhabende Größe erreichen. Datensätze im mehrstelligen Terabyte-Bereich sind hier keine Seltenheit. Was aber nicht verwundert, wenn man sich eines der bekanntesten Anwendungsgebiete der numerischen Simulation vor Augen hält: die Wetterprognose. Dabei handelt es sich um ein globales Phänomen für dessen halbwegs genaue Vorhersage unzählige Einflüsse und Unmengen von Daten aus Messungen aus allen Teilen der Welt berücksichtigt werden müssen. Eine effiziente und für den schnellen Zugriff ausgewählte Datenstruktur wie Octrees und dergleichen reichen hier nicht mehr aus. Ein gutes Visualierungsprogramm sollte daher die Möglichkeit zur Verfügung stellen, die Verarbeitung der Daten über eine Remote-Visualisierungsschnittstelle auf einen leistungsfähigeren Computer auszulagern. Auch das Aufteilen und Verteilen der Daten auf mehrere beteiligte Computer sollte möglich sein. Die Bewertung dieser Fähigkeiten steht in diesem Abschnitt im Vordergrund.

7.1 Paraview

Um Paraview im verteilten Modus laufen zu lassen, muss die Software auf den jeweiligen Systemen kompiliert werden. Aufgrund der großen Vielfalt an Hardware, Betriebssystemen und MPI-Implementierungen gibt es keine *Binaries* für den Client-Teil und Server-Teil von Paraview. Eine schrittweise Anleitung, wie Paraview kompiliert werden muss, ist unter [1] zu finden. Für den parallelen Betrieb muss das CMake-Flag PARAVIEW_USE_MPI auf ON gesetzt werden.

Ist die Software kompiliert, kann der Server gestartet werden. Je nach MPI-Version gibt es unterschiedliche Befehle, um parallele Programme auszuführen. Bei *OpenMPI* geschieht das durch den Befehl:

```
mpirun –np 4 ./pvserver
```

Diese Anwendung beinhaltet sowohl Daten-Server als auch Render-Server. Um diese auch noch zu trennen müssen die Anwendungen *pvdataserver* und *pvrenderserver* separat gestartet werden. Wie schon im Kapitel 4 erwähnt, lohnt sich dieser Betriebsmodus in den seltensten Fällen, da die Kommunikation zwischen Daten- und Renderserver sehr datenintensiv ist. Der Client ist eine rein serielle Anwendung und wird daher nicht mit *mpirun* gestartet, sondern mit

```
./paraview
```

Über das Menü *File* → *Connect* kann nun eine Verbindung zum dem Rechner hergestellt werden, auf dem die Anwendung `pvserver` läuft.

[1] http://www.itk.org/Wiki/ParaView:Build_And_Install

7.2 VisIt

Das Ausführen von VisIt im verteilten (distributed) Modus unterscheidet sich nicht groß davon, es lokal auf dem User-PC auszuführen. Man beginnt auch damit, dass *File Selection Window* zu öffnen, nur gibt man nun den Namen oder die IP-Adresse des Computers, auf dem die zu visualisierenden Daten liegen, in das Textfeld *Host* ein. VisIt wird dann versuchen, auf dem entfernten Rechner den *VisIt Component Launcher* (VCL) zu starten. Dieser ist dafür verantwortlich, andere VisIt Komponenten wie den *Database-Server* und die *Compute Engine* zu starten (siehe Kapitel 4). Sobald man mit dem entfernten Rechner verbunden ist und der VCL läuft, werden die zur Verfügung stehenden Daten normal im *File Selection Window* aufgelistet. Zur Authentifizierung auf dem entfernten Rechner verwendet VisIt *SSH*, es kann aber auch so konfiguriert werden, dass kein Passwort benötigt wird, was als *passwordless SSH* bezeichnet wird.

Wenn VisIt eine Komponente auf einem entfernten Rechner startet, sucht es nach einem sogenannten *Host Profile*. Dies enthält Informationen darüber, wie VisIt Komponenten startet. Man kann damit Informationen spezifizieren wie Remote-Username, die Anzahl der Prozessoren oder das Programm, das zum Ausführen der parallelen Compute Engine verwendet wird. Über das *Host Profile Window* kann der Benutzer seine verschiedenen Profile verwalten. Eines der Hauptziele des *Host Profile* ist es, das Ausführen der Compute Engines leichter zu machen. Das trifft besonders dann zu, wenn es um parallele Compute-Engines auf Großrechnern geht. Diese Rechner haben oft komplexe Scheduling-Programme die definieren wann ein paralleler Job ausgeführt werden kann. VisIt verbirgt die Details des *Scheduling* vor dem Benutzer und vereinfacht so das Ausführen. Es erlaubt stattdessen das Festlegen einiger allgemeiner Optionen für Parallelität und kümmert sich dann darum, wie dies auf dem entfernten Rechner umgesetzt werden kann.

7.3 Covise

Trotz einigen Versuchen war es nicht möglich Covise verteilt zu nutzen. Mit Zwei Instanzen von Covise auf einem Rechner stürzte das Programm ab, sobald versucht wurde, eine Verbindung aufzubauen.

7.4 Fazit

Bei Paraview und VisIt gab es keinerlei Probleme bei der parallelen Inbetriebnahme. Covise stürzte beim Versuch einer Verbindung ab.

8 Performance-Vergleich

Da sich Covise schlecht durch Skripte bedienen lässt und auch keine interne Zeitmessung besitzt, beschränken wir uns beim Performance-Vergleich auf *Paraview* und *VisIt*.

8.1 Referenzumgebung

- Prozessor: Intel(R) Core(TM)2 Duo CPU T5800 2.00GHz
- Grafikkarte: nVidia GeForce 9500M GS
- Hauptspeicher: 4 GB RAM
- *bluntfin.vts*: Datensatz für Skalarfelder aus Kap. 3. Gekrümmtes Gitter, 40.960 Punkte
- *tornado.vtk*: Datensatz für Vektorfelder aus Kap. 3. Gekrümmtes Gitter, 2.097.152 Punkte

8.2 Durchführung

Für den Performance-Vergleich wurden Stromlinien, Isoflächen und Vektorfelder mit unterschiedlichen Datensätzen und Konfigurationen visualisiert. In Paraview wird der Zeitbedarf mit dem internen *Timer Log* gemessen. VisIt wird für die Zeitmessung mit der Option *-timing* gestartet. Diese bewirkt, dass in der Datei *viewer.timings* Angaben zum Zeitbedarf für jede Operation abgelegt werden.

Vergleich 1: 2.000 Stromlinien im Datensatz bluntfin.vts. Schrittweite: 0.2, maximale Länge der Stromlinien: 22, Verfahren: Runge-Kutta 4, zufällige Startpunkte auf einem Kreis um den Punkt (3.2, 4.1, 2.8)

Vergleich 2: Isoflächenextraktion im Datensatz bluntfin.vts. Eine Isofläche für *StagnationEnergy = 6*

Vergleich 3: Isoflächenextraktion im Datensatz tornado.vts. Eine Isofläche für $|velocity| = 0.6$

Vergleich 4: 100.000 Glyphen im Datensatz tornado.vtk.

8.3 Ergebnisse

Tabelle 8.1: Performance-Vergleich zwischen Paraview und VisIt

Vergleich Nr.	Vergleich	Paraview	Visit
1	2.000 Stromlinien	2,24 Sekunden	10,28 Sekunden
2	Isoflächen	0.013 Sekunden	0.039 Sekunden
3	Isoflächen	0.54 Sekunden	0.25 Sekunden
4	100.000 Glyphen	5,68 Sekunden	1,70 Sekunden

9 Fazit

9.1 Paraview

Paraview ist eine ausgereifte Visualisierungs- und Analysesoftware. Sie läuft unter Windows, MacOS und Linux auf Einzel- oder Mehrkernprozessoren. Paraview wird auch auf großen Rechnerclustern und Supercomputern für Visualisierungsaufgaben aller Art eingesetzt. Obwohl die Benutzeroberfläche für einen Einsteiger zunächst überladen wirkt, ist das Programm benutzerfreundlich und leicht zu erlernen. Gleichzeitig bietet Paraview durch die vielen eingebauten Filter und durch die Entwicklung von eigenen Plugins genug Spielraum für den fortgeschrittenen Anwender. Es kann daher für jede Art von Anwender empfohlen werden.

9.2 VisIt

VisIt scheint ein ausgereiftes Programm zu sein. Die Benutzeroberfläche ist ansprechend gestaltet und strukturiert. Die Software ist mit sehr vielen Visualisierungstechniken ausgestattet und unterstützt alle gängigen Dateiformate. Das durchdachte Plugin-Konzept ermöglicht es, in VisIt neue Visualisierungstechniken oder Reader für neue Dateiformate unkompliziert zu integrieren. Die Software macht insgesamt einen stabilen Eindruck. Nur wenige Programm-Abstürze sind bei der Arbeit aufgetreten, der Auslöser war aber stets nachvollziehbar. Die gute Dokumentation und Online-Hilfen werten das Programm noch zusätzlich auf.

9.3 Covise

Covise ist eher eine Plattform als eine fertige Visualisierungssoftware. Das Modulkonzept lässt dem Nutzer alle Möglichkeiten offen. Ein Nutzer von Covise ist jedoch nicht jemand der schnell etwas visualisieren möchte, sondern jemand der eine individuelle Visualisierungssoftware für einen bestimmten Zweck entwickeln will. Es gibt bereits Unternehmen, die auf Covise aufbauend, Lösungen anbieten. Sie verkaufen das Basissystem mit selbst entwickelten Modulen für verschiedenste Einsatzzwecke in Medizin, Ingenieursanwendungen und für die Forschung. Die vorhandenen Module leiden unter einigen gravierenden Bugs. Durch die mangelnde Dokumentation und vor allem durch die kaum vorhandenen Rückmeldungen weiß der Benutzer oft nicht, warum ein bestimmter Aufruf nicht funktioniert. Covise kann daher nur für Entwickler als Framework empfohlen werden.

9.4 Abschließende Empfehlung

Während der Arbeit mit Paraview, VisIt und Covise haben sich einige Stärken und Schwächen der Programme gezeigt. Besonders Covise scheint nicht für einen schnellen Einstieg in Visualisierungsaufgaben geeignet zu sein. Zwischen Paraview und VisIt gibt es in Bezug auf den Funktionsumfang keine signifikanten Unterschiede. Bei VisIt traten in der Windows- und Linux-Version gelegentlich Abstürze auf. Trotzdem lassen sich beide Programme gleichermaßen empfehlen. Die Auswahl kann daher eher nach persönlichen Vorlieben in Bezug auf Bedienoberfläche, unterstützte Dateiformate und Bedienbarkeit erfolgen.