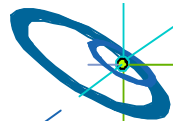




H L R I S

COVISE-Workshop 2008

# COVISE-Programmierung Grundlagen



1

HLRS  
Covise Workshop 2008 – COVISE VR

H L R I S 

## COVISE Programmierkurs

- Informationen zum Arbeiten auf den Rechnern:
  - Alle Kursteilnehmer arbeiten mit dem selben Account.
  - Login: -----
  - Passwort: \*\*\*\*\*
- Was wird benötigt, um mit COVISE zu arbeiten?
  - COVISE Development Distribution oder
  - COVISE Subversion-Entwickleraccount
  - externe Abhängigkeiten:
    - Von der Architektur abhängig;
    - diejenigen, für die es auf der Architektur keine fertigen Pakete gibt, liegen in \$EXTERNLIBS
    - Für existierende Pakete müssen die Entwicklungsbibliotheken installiert sein.
  - Alle nötigen Pakete sind hier schon installiert.

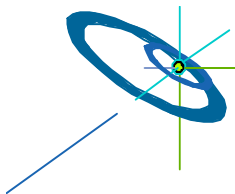


## COVISE Programmierkurs

- Folgende Verzeichnisse befinden sich u.a. im COVISE-Verzeichnis
  - *config* Standardverzeichnis für die COVISE-Konfigurationsfiles
  - *doc* Dokumentation & Hilfedateien
  - *net* Standard-Verzeichnis zum Abspeichern der COVISE-Netze
  - *src* Sourcen (application, kernel, renderer, sys, tools), jedes COVISE-Modul benötigt ein eigenes Verzeichnis
  - *gecko* ...oder \$ARCHSUFFIX. Hier liegen architekturabhängige Dateien (in *bin* und *lib*).
- Vor dem Compilieren müssen noch zwei Umgebungsvariablen gesetzt werden:
  - \$COVISEDIR: Verzeichnis, in das COVISE installiert wurde
  - \$COVISE\_BRANCH: Sollte auf HLRS gesetzt werden
- Danach muß *source \$COVISEDIR/.covise.sh* aufgerufen werden.

## COVISE Programmierkurs

- Zur plattformübergreifenden Projektverwaltung wird *qmake* verwendet
  - Eine *.pro*-Datei erzeugt aus einigen wenigen Einträgen ein vollwertiges Makefile oder eine Solution für Visual Studio.
- Handgestrickte Makefiles/Projektmappen sind natürlich auch möglich.
- Dokumentation: COVISE Programming Guide.
  - Befindet sich im COVISE-Verzeichnis unter *doc/pdf* im PDF-Format;
  - dient als Referenz;
  - enthält weitergehende Materialien.



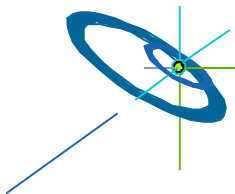
# COVISE-Module

- Jedes Modul ist ein eigener Prozeß
- Module sind in C++ geschrieben
- Module kommunizieren über Messages
- Daten werden über Shared Memory oder TCP/IP Sockets ausgetauscht
- Kommunikation geschieht transparent für den Programmierer.
- Drei Basisbibliotheken:
  - **libcoCore.so**: Data Management, Message Communication, Starten von Prozessen, etc.
  - **libcoAppl.so**: Application Library, enthält die Basisfunktionalität, um Anwendungen zu programmieren. Versteckt die Details der Kommunikation und bietet ein Gerüst, um Module zu strukturieren.
  - **libcoApi.so**: Application Programming Interface, das die Anwendungsprogrammierung einfacher und weniger fehleranfällig macht. Diese übergeordneten API-Funktionen sollten wann immer möglich benutzt werden, anstatt den direkten Weg über die Application Library zu gehen.



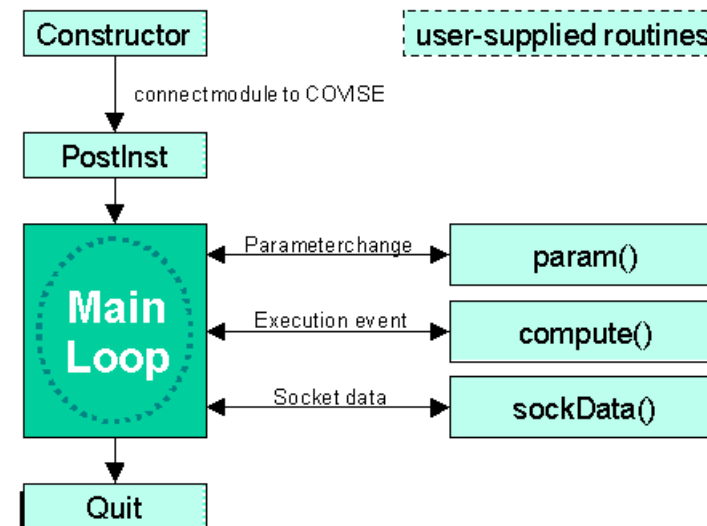
# Hello COVISE-World

- Modul befindet sich in `covise/src/application/examples/Hello`
- **AUFGABE**  
Wechseln sie mit `cd covise/src/application/examples` in das Beispielerverzeichnis  
Kopieren Sie das Verzeichnis `Hello` nach `Hello.<Nachname>` mittels `cp -r Hello Hello.<Nachname>`.  
Wechseln sie mit `cd Hello.<Nachname>` in das neue Verzeichnis.
- Ändern Sie in der Datei `Hello.pro` die Zeile `TARGET=Hello` in `TARGET=Hello.<Nachname>`
- Öffnen Sie bitte die Dateien `Hello.h` und `Hello.cpp` in einem Editor Ihrer Wahl (kdevelop, emacs, vi, nedit, o.ä.).
- Konstruktor `is` (noch) leer.
- Eine Methode implementiert: `compute()`
  - `compute` wird aufgerufen, wenn in der Oberfläche der *Execute*-Button gedrückt wird.
  - Hier findet die Arbeit des Moduls statt.
- **AUFGABE**  
Übersetzen Sie das Modul `Hello` und testen Sie es in COVISE.



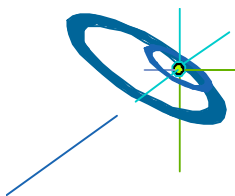
# Ausführungsablauf eines COVISE-Modules

- Beim Start des Modules wird der Konstruktor aufgerufen.
  - Zeitkritische Aktion;
  - hier müssen alle Ports und Parameter angelegt werden;
  - später können keine Ports und Parameter mehr erzeugt werden.
- Danach wird `postInst()` aufgerufen.
  - Zeit für längere Operationen
- Danach geht das Modul in den Main Loop und wartet auf Nachrichten.
- Über die `quit()`-Methode wird das Modul benachrichtigt, daß es beendet wird.



# coModule - Übersicht

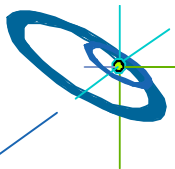
- Die Methoden, über die ein coModul mit COVISE kommuniziert, sind:
  - `void coModule::postInst(void)`
  - `void coModule::param(const char *paramName, bool inMapLoading)`
  - `int coModule::compute(void)`
  - `void coModule::addSocket(int fd)`
  - `virtual void coModule::sockData(int fd)`
  - `void coModule::removeSocket(int fd)`
  - `virtual void coModule::quit(void)`
  - `virtual float coModule::idle(void)`
  - `void coModule::feedback(int len, const char *data)`
  - `void feedbacksetInfo(int len, const char *datatext)`



# Modulparameter

- Modulparameter werden nicht direkt über ihren Konstruktor erzeugt sondern über die Methode `coModule::add*Param(const char * name, const char * description)`
- Beispiel:  

```
coBooleanParam * param =  
    addBooleanParam("Test", "Test");
```
- Folgende Parameter existieren in COVISE
  - `coBooleanParam`
  - `coFloatParam`
  - `coFloatSliderParam`
  - `coFloatVectorParam`
  - `coStringParam`
  - `coBrowserParam`
  - `coChoiceParam`
  - `coIntScalarParam`
  - `coIntSliderParam`
  - `coIntVectorParam`



## Modulparameter II

- Setzen eines Parameters: `setValue()`
- Auslesen eines Parameters: `getValue()`
- **AUFGABE**  
Erweitern Sie das „Hallo Welt“-Modul, so dass statt des statischen Textes ein Stringparameter ausgegeben werden kann.



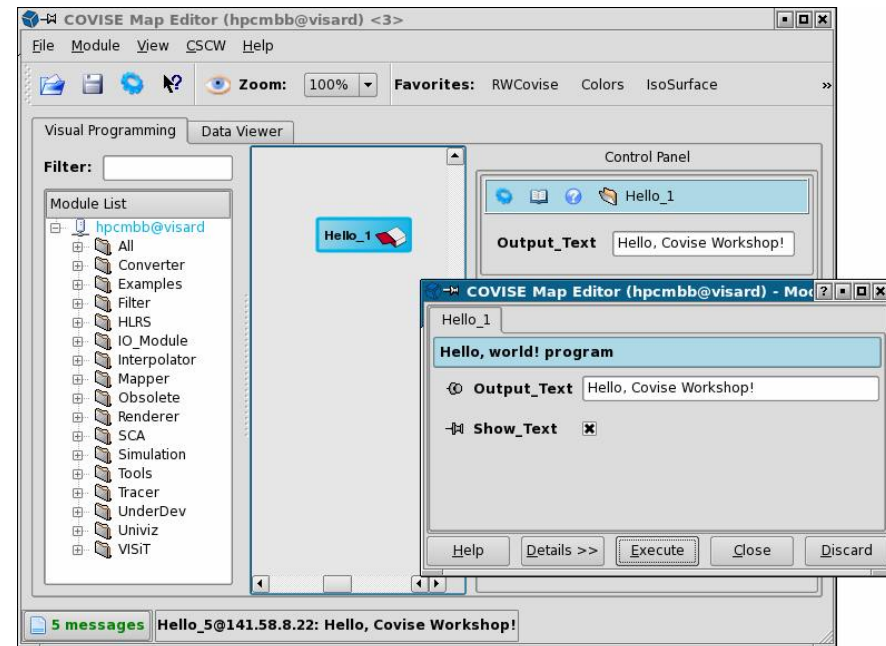
## Modulparameter III

- Parameter werden normalerweise gesetzt, bevor ein Modul ausgeführt wird.
- Das Modul erhält eine Nachricht, sobald ein Parameter neu gesetzt wurde:
  - Jedes mal, wenn der Parameter geändert wird, wird die Methode  
`void coModule::param(const char *paramName, bool inMapLoading)`  
mit dem Namen des geänderten Parameters aufgerufen.
  - Dieser kann dann mittels `getValue()` ausgelesen werden.



# Modulparameter III

- Parameter können, damit sie einfacher erreichbar sind, in das Control Panel eingeblendet werden.
- `co*Param::show()` blendet den Parameter ein.
- `co*Param::hide()` blendet den Parameter wieder aus.
- **AUFGABE:** Erweitern Sie das „Hello World“-Modul um einen Parameter, welcher den Text-Parameter im Control Panel ein- und ausblendet.



# Input/Output Ports I

- Ports dienen dem Austausch der Daten zwischen Modulen.
- Input-Ports nehmen Daten entgegen, Output-Ports geben diese weiter.
- Ports werden über spezielle Methoden von `coModule` erzeugt.
- ```
coInputPort * coModule::addInputPort (const char * name,  
    const char * typeList,  
    const char * description)  
coOutputPort * coModule::addOutputPort(const char * name,  
    const char * typeList,  
    const char * description)
```
- Die Parameter sind:
  - `name`            Eindeutiger Name des Ports (ohne Leerzeichen / Sonderzeichen!)
  - `typeList`            Liste der Datentypen, die ein Port zur Verfügung stellt / annehmen kann.
  - `description`        Beschreibung des Ports



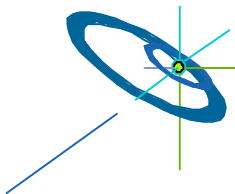
## Input/Output Ports II

- Beispiel:
- ```
gridIn =  
    addInPort("gridIn",  
              "UnstructuredGrid|Polygons|Lines",  
              "mesh");
```
- Wichtig: In der TypeList darf kein Leerzeichen vorkommen.
- Typen werden durch ' | ' abgetrennt.
- Die TypeList wird nur im MapEditor ausgewertet und verhindert die größten Fehlverbindungen. Die tatsächliche Typenüberprüfung muß vom Modul selbst durchgeführt werden.
- **AUFGABE:**  
Erzeugen Sie im „Hello World“-Modul einen Ausgabeport, welcher Linien (Lines) und Polygone (Polygons) zur Verfügung stellt.



# COVISE Datenobjekte I

- COVISE bietet alle Datenobjekte, welche für die Visualisierung von Daten wichtig sind.
  - Gitter (DO\_UniformGrid, DO\_RectilinearGrid, DO\_StructuredGrid, DO\_UnstructuredGrid),
  - Geometrien (DO\_Points, DO\_Lines, DO\_TriangleStrips, DO\_Polygons),
  - Vektor- und Skalarfelder (DO\_Unstructured\_S3D\_Data, DO\_Unstructured\_V3D\_Data, DO\_Unstructured\_T3D\_Data).  
Wichtig: seit COVISE 6.1 gibt es keine strukturierten Daten mehr.
  - Farben (DO\_RGBA\_Data)
  - Volumen (DO\_volumes)
- Diese Objekte können in Mengen (DO\_Set) gruppiert werden.
- Alle Datenobjekte sind von DistributedObject abgeleitet.
- Der Typ eines DistributedObject kann über einen dynamic\_cast ermittelt werden.

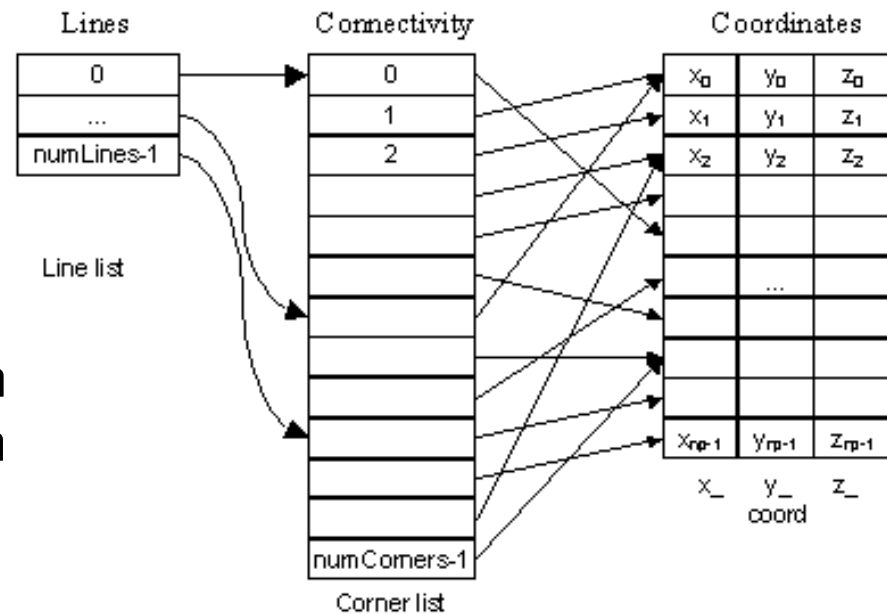


## COVISE Datenobjekte II

- Vier wichtige Dinge sind beim Arbeiten mit `DistributedObjects` zu beachten
  - Sie dürfen nur im `compute`-Callback verwendet werden.
  - DOs, welche an Ports gebunden sind, dürfen nicht gelöscht werden.
  - Schreibvorgänge im Shared Memory werden nicht auf Grenzen überprüft. Das Überschreiten von Array-Grenzen führt in der Regel zu Abstürzen.
  - Bei Objekten, die von einem Eingabeport kommend weitergereicht werden, muß ein Container mit dem Objektnamen des Ausgabeports erzeugt werden, in den das Objekt gelegt wird. Dabei muß der Referenzzähler des alten Objektes erhöht werden, damit dieses nicht zu früh gelöscht wird. Dies geschieht über die Methode  
`DistributedObject::increase_refcount()`
- Objekte werden über die Methode  
`coOutPort::setObj(DistributedObject * obj)` an einen Port gebunden.

# COVISE Datenobjekte – Beispiel Linien

- Beispiel: Linien (DO\_Lines)
- Abgeleitet von DO\_Geometry
- Allen Geometrien können direkt Farben, Normalen und Texturen zugewiesen werden.
- COVISE-Geometrien sind mehrfachindiziert:
  - Feld mit Koordinaten (x, y, z)
  - Feld mit den Kanten zwischen den Koordinaten (Index auf die Koordinaten)
  - Feld mit den Linienzügen (Index auf die einzelnen Linienzüge)



## COVISE Datenobjekte – Beispiel Linien

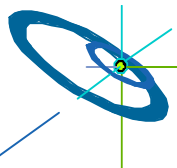
- Alle `DistributedObjects` besitzen zwei Arten von Konstruktoren:
  - Daten des Objektes werden direkt über den Konstruktor gesetzt

```
DO_Lines(const char * name,  
         int numPoints, float * x, float * y, float * z,  
         int numCorners, int * cornerList,  
         int numLines, int * lineList )
```

- Nur die Größe der Datenfelder wird angelegt. Diese werden vom Objekt erzeugt und können dann direkt befüllt werden.

```
DO_Lines(const char * name,  
         int numPoints, int numCorners, int numLines)
```

- Wenn ein Objekt an einen Ausgabeport gebunden wird, **muss** ihm der Objektname durch den Port zugewiesen werden. Den Objektnamen erhält man mittels `coPort::getObjName()`.

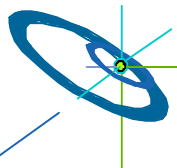


## COVISE Datenobjekte – Beispiel Linien

- Wird die zweite Form des Konstruktors verwendet, so kann man die Startadressen der angelegten Strukturen über

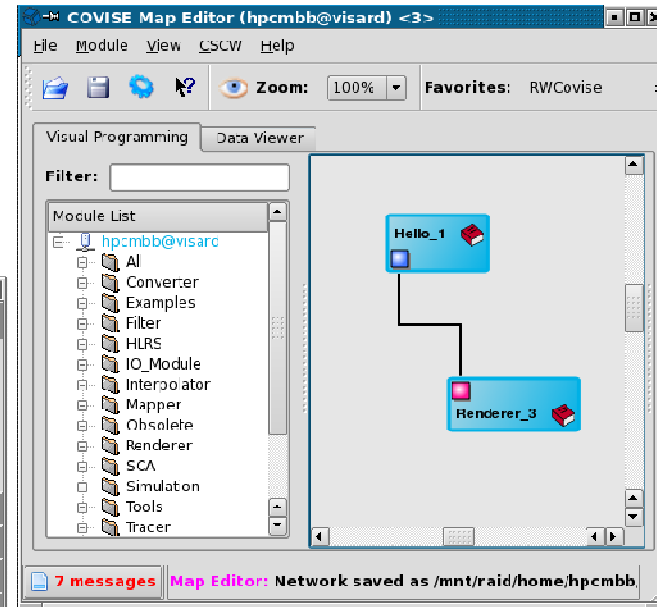
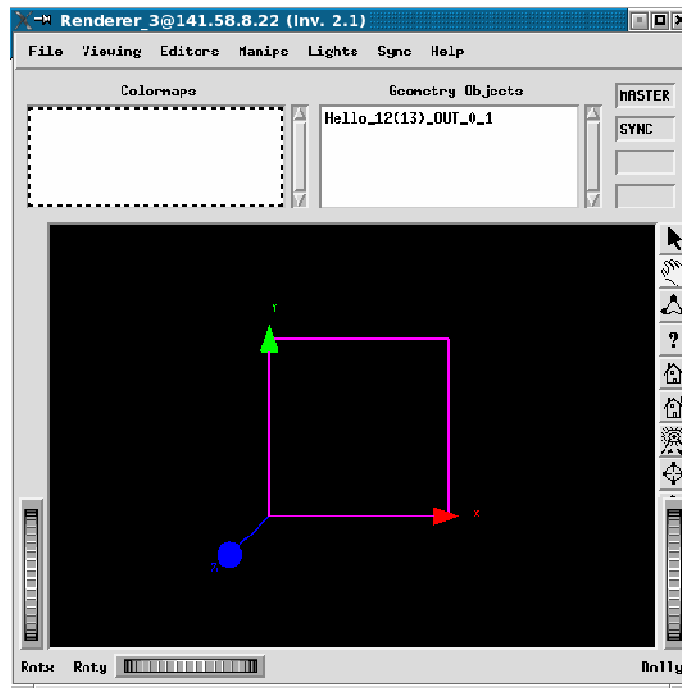
```
void get_addresses(float ** xStart,  
                 float ** yStart,  
                 float ** zStart,  
                 int ** cornerList,  
                 int ** lineList)
```

erhalten. Diese können dann direkt befüllt werden.



# COVISE Datenobjekte – Beispiel Linien

- **AUFGABE:**  
Erzeugen Sie ein Quadrat aus vier Linien. Stellen Sie diese Linien im Renderer dar.



## COVISE Datenobjekte - Skalarwerten

- Geometrien können direkt eingefärbt werden.
- Oft will man jedoch Farben dazu verwenden, um Ergebnisse zu visualisieren.
- Beispiel: Skalarwerten (DO\_Unstructured\_S3D\_Data)
- Konstruktoren (ähnlich denen der Linien)

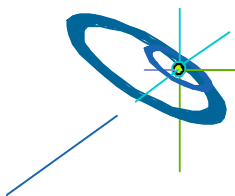
```
DO_Unstructured_S3D_Data(const char * name, int  
numValues)
```

```
DO_Unstructured_S3D_Data(const char * name, int  
numValues,  
float *scalarData)
```

- Zugriff auf die Datenarrays

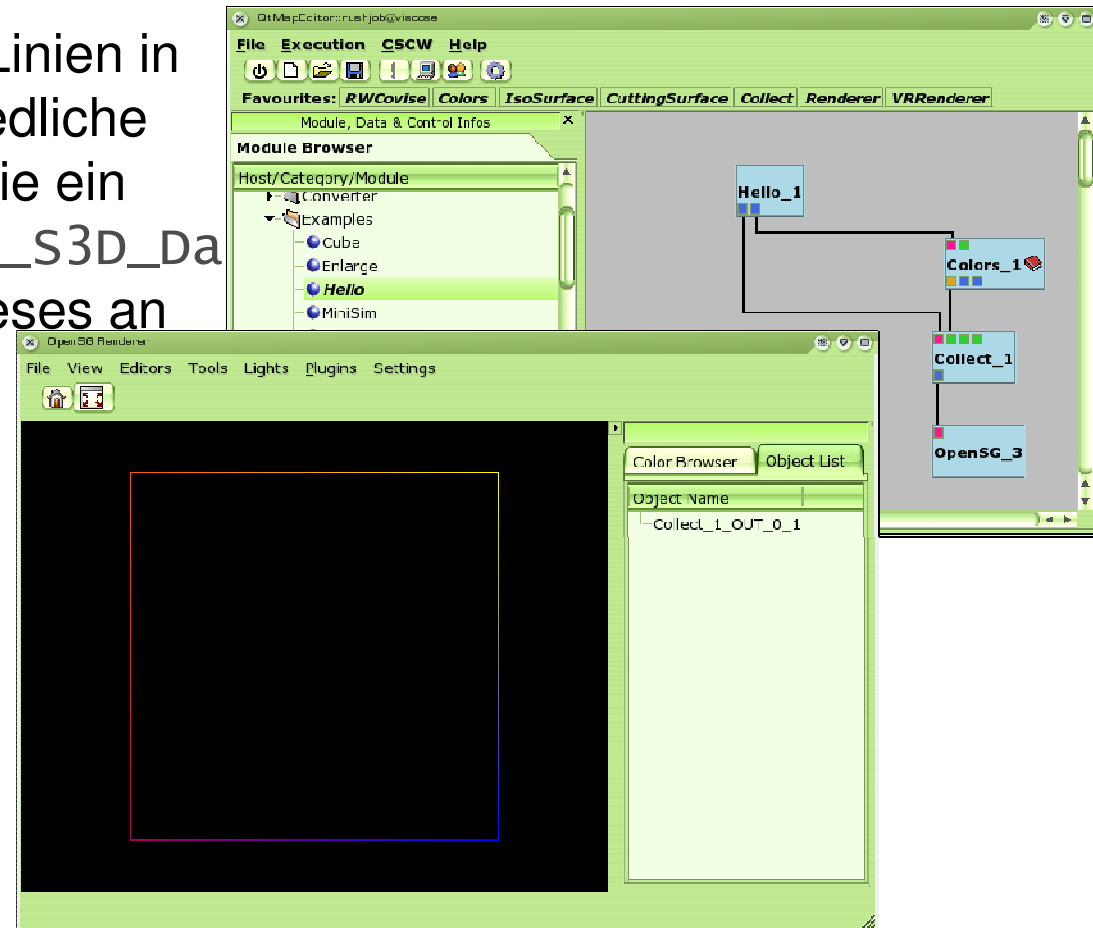
```
void DO_Unstructured_S3D_Data::get_address(float ** start)
```

- Die Daten können dann über das *Colors*-Modul eingefärbt werden.
- Wichtig: die Anzahl der Skalarwerten sollte genauso groß sein wie die der Koordinatenpunkte.



# COVISE Datenobjekte - Skalarwerte

- **AUFGABE:**
- Färben Sie die vier Linien in beliebige, unterschiedliche Farben ein, indem Sie ein `DO_Unstructured_S3D_Data` Objekt erzeugen, dieses an einen neuen Port binden und durch das Modul *Colors* post-processen lassen. Stellen Sie diese im *Renderer* dar.



# COVISE Datenobjekte

- **AUFGABE:**  
Finden Sie in der COVISE Programming Guide die Dokumentation zu Polygonen. Erzeugen Sie einen Würfel aus mehreren Polygonen. Benutzen Sie hier einen Konstruktor, der vorhandene Daten direkt in das Objekt kopiert.

