

Introduction to ParaView Plugins

March 21, 2013

Sonja Habbinga

ParaView Plugins - Introduction

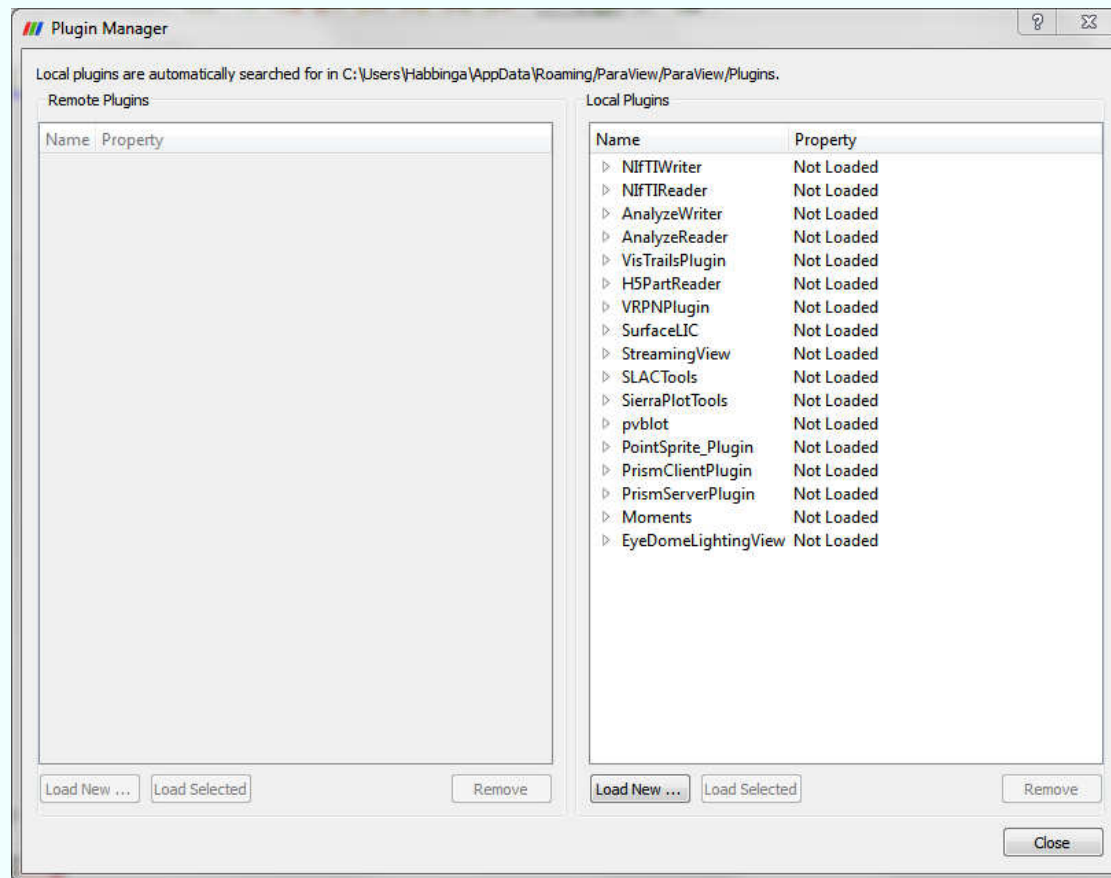
- ParaView comes with a large range of readers, filters and views providing a large range of functionality
- ParaView provides an extensive plugin mechanism which enables the user to
 - add new **readers**, writers and **filters**
 - add custom GUI components such as toolbar buttons to perform common tasks
 - add new views for displaying data
- The following slides cover
 - how to use existing plugins
 - how to write new plugins for ParaView.

Using ParaView Plugins

- plugins are distributed as shared libraries (*.so on Unix, *.dylib on Mac, *.dll on Windows)
- plugin must be built with the same version of ParaView
- ParaView has to be built with shared libs enabled
- three ways for loading plugins:
 - using environment variable PV_PLUGIN_PATH (Auto-loading plugins)
 - placing the plugins in a recognized location:
 - “plugins” subdirectory beneath the directory containing the paraview client or server executables
 - “plugins” subdirectory in the user's home area.
\$HOME/.config/ParaView/ParaView<version>/Plugins
on Unix/Linux/Mac
%APPDATA%\ParaView\ParaView<version>\Plugins
on Windows
 - using the GUI

Loading ParaView Plugins Using the GUI

- Plugin manager accessible from Tools | Manage Plugins menu:



Writing ParaView Plugins – Server Manager Configuration

- ParaView uses the VTK Server Manager for building distributed client-server applications by means of the Proxy design pattern
- Proxies are defined in the **Server Manager Configuration XML**
- Proxies are organized in **ProxyGroups**
 - Examples: sources, filters, views, ...
- Each **Proxy** stores a reference to the corresponding VTK object located on the server
 - Examples: SourceProxy, ViewProxy, ...
- The Proxy contains **Properties** to describe the part of the interface which is exposed to the client (function name and arguments)
 - Examples: IntVectorProperty, StringVectorProperty, InputProperty, ...
- The range of values of a property can be restricted by **Domains**
 - Examples: IntRangeDomain, FileListDomain, DataTypeDomain, ProxyGroupDomain, ...

Enabling existing VTK-Classes as Plugins

- To enable an existing VTK-class, write a Server Manager configuration xml-file and load it into the ParaView GUI
- example: enable the VTK-filter **vtkCellDerivatives**, write CellDerivatives.xml as follows

```
<?xml version="1.0"?>
<ServerManagerConfiguration>
  <ProxyGroup name="filters">
    <SourceProxy name="MyCellDerivatives" class="vtkCellDerivatives" label="My Cell Derivatives">
      <Documentation
        long_help="Create point attribute array by projecting points onto an elevation vector."
        short_help="Create a point array representing elevation.">
      </Documentation>
      <InputProperty
        name="Input"
        command="SetInputConnection">
        <ProxyGroupDomain name="groups">
          <Group name="sources"/>
          <Group name="filters"/>
        </ProxyGroupDomain>
        <DataTypeDomain name="input_type">
          <DataType value="vtkDataSet"/>
        </DataTypeDomain>
      </InputProperty>
    </SourceProxy>
  </ProxyGroup>
</ServerManagerConfiguration>
```

Writing New ParaView Plugins – Source Code

- compiling plugins requires own build of ParaView3 since binaries downloaded from www.paraview.org do not include necessary header files
- Example: simple reader called **vtkCSVImageReader** (example taken from
- <http://www.kitware.com/media/html/WritingAParaViewReaderPlugin.html>)
 - read a 2D image from a text file with simple delimited format
 - allow someone to export simple matrices of data from Excel and visualize them as heat maps or height fields in ParaView
 - assume the files consist of strictly numeric values separated by commas or some other delimiter
 - use a `vtkDelimitedTextReader` instance which will read delimited text into a `vtkTable`
 - methods to set and get the file name and field delimiter characters, which are delegated to the `vtkDelimitedTextReader` instance variable

Writing New ParaView Plugins

- Writing a new plugin requires generating the following files:
 - .cxx and .h files (the actual plugin source code)
 - Server manager configuration (xml-file)
 - configuration xml for the GUI (optional for filters)
 - CMakeLists.txt for compiling your source code into a shared library
- Recommended Tool: plugin Wizard
 - developed by MIRARCO
 - free and open source application which helps you to create a framework of the above mentioned files.
 - download at <http://pluginwizard.mirarco.org/>

Writing New ParaView Plugins – Source Code

- Step 1: create a framework for your plugin using the Plugin Wizard
- Step 2: override the following methods of the chosen vtk superclass to make it operate properly in a standard vtk pipeline:
 - tell the VTK pipeline information about the data before creating the output, e.g. extent, spacing and origin in case of vtkImageData

```
int vtkCSVImageReader::RequestInformation( vtkInformation *request,
                                           vtkInformationVector **inputVector,
                                           vtkInformationVector *outputVector)
```

- read data from the file and store it in the corresponding data object in the output port

```
int vtkCSVImageReader::RequestData( vtkInformation *request,
                                     vtkInformationVector **inputVector,
                                     vtkInformationVector *outputVector );
```

- Step 3: override other methods, e.g. **SetFileName**(const char* fname), **SetFieldDelimiterCharacters**(const char* delim), ...

Class vtkCSVImageReader

```
class vtkCSVImageReader : public vtkImageAlgorithm
{
public:
    static vtkCSVImageReader* New();
    vtkTypeRevisionMacro(vtkCSVImageReader,vtkImageAlgorithm);
    void PrintSelf(ostream& os, vtkIndent indent);
    virtual void SetFileName(const char* fname);
    virtual const char* GetFileName();
    virtual void SetFieldDelimiterCharacters(const char* delim);
    virtual const char* GetFieldDelimiterCharacters();
protected:
    vtkCSVImageReader();
    ~vtkCSVImageReader();

    int RequestInformation(vtkInformation*, vtkInformationVector**,
vtkInformationVector*);
    int RequestData(vtkInformation*, vtkInformationVector**, vtkInformationVector*);
    vtkDelimitedTextReader* Reader;
private:
    vtkCSVImageReader(const vtkCSVImageReader&);
    void operator=(const vtkCSVImageReader&);
};
```

RequestInformation(...)

```
int vtkCSVImageReader::RequestInformation (vtkInformation*, vtkInformationVector**,
    vtkInformationVector* outputVector)
{
    vtkInformation* outInfo =outputVector->GetInformationObject(0);
    this->Reader->Update();
    vtkTable* output = this->Reader->GetOutput();
    vtkIdType rows = output->GetNumberOfRows();
    vtkIdType columns = output->GetNumberOfColumns();

    int ext[6] = {0, rows, 0, columns, 0, 0};
    double spacing[3] = {1, 1, 1};
    double origin[3] = {0, 0, 0};

    outInfo->Set(vtkStreamingDemandDrivenPipeline::WHOLE_EXTENT(), ext, 6);
    outInfo->Set(vtkDataObject::SPACING(), spacing, 3);
    outInfo->Set(vtkDataObject::ORIGIN(), origin, 3);
    vtkDataObject::SetPointDataActiveScalarInfo(outInfo, VTK_FLOAT, 1);
    return 1;
}
```

RequestData(...)

```
int vtkCSVImageReader::RequestData(vtkInformation*, vtkInformationVector**,
    vtkInformationVector* outputVector)
{
    vtkImageData* image = vtkImageData::GetData(outputVector);
    vtkTable* output = this->Reader->GetOutput();
    vtkIdType rows = output->GetNumberOfRows();
    vtkIdType columns = output->GetNumberOfColumns();
    image->SetDimensions(rows, columns, 1);
    image->AllocateScalars();
    vtkDataArray* scalars = image->GetPointData()->GetScalars();
    scalars->SetName("Data");
    for (vtkIdType r = 0; r < rows; ++r)
    {
        for (vtkIdType c = 0; c < columns; ++c)
        {
            vtkVariant val = output->GetValue(r, c);
            float f = val.ToFloat();
            scalars->SetTuple1(c*rows + r, f);
        }
    }
    return 1;
}
```

Writing New ParaView Plugins – Server Manager Configuration

- Server Manager Configuration for the example CSVImageReader:

```
<?xml version="1.0"?>
  <ServerManagerConfiguration>
    <!-- Begin CSVImageReader -->
    <ProxyGroup name="sources">
      <SourceProxy name="CSVImageReader"
        class="vtkCSVImageReader">
        <StringVectorProperty name="FileName"
          number_of_elements="1"
          command="SetFileName">
          <FileListDomain name="files"/>
        </StringVectorProperty>
        <StringVectorProperty
          name="FieldDelimiterCharacters"
          command="SetFieldDelimiterCharacters"
          number_of_elements="1"
          default_values=","/>
      </SourceProxy>
    </ProxyGroup>
    <!-- End CSVImageReader -->
  </ServerManagerConfiguration>
```

Writing New ParaView Plugins – Configuration XML for the GUI

- Provides the higher-level information about where new elements will be placed in the user interface.
- Example: CSVImageReaderGUI.xml

```
<?xml version="1.0"?>
<ParaViewReaders>
  <Reader
    name=„CSVImageReader“
    extensions=„csvimg“
    file_description=„csv Image data“>
  </Reader>
</ParaViewReaders>
```

- tells ParaView to create GUI elements for “CSVImageReader”
- creates a new entry in the File Open dialog for CSV image data files
- associates files ending in “.csvimg” with the new reader

Writing ParaView plugins – CMakeLists.txt

- Use cmake to build the ParaView plugin
- CMakeLists.txt file ties sources and xml files together

```
CMAKE_MINIMUM_REQUIRED(VERSION 2.6)
PROJECT(CSVImageReader)

FIND_PACKAGE(ParaView REQUIRED)
INCLUDE(${PARAVIEW_USE_FILE})

ADD_PARAVIEW_PLUGIN(CSVImageReader "1.0"
    SERVER_MANAGER_XML CSVImageReader.xml
    SERVER_MANAGER_SOURCES vtkCSVImageReader.cxx
    GUI_RESOURCE_FILES CSVImageReaderGUI.xml)
```

- locate ParaView with FIND_PACKAGE
- import the CMake configuration parameters from ParaView by including PARAVIEW_USE_FILE.
- ADD_PARAVIEW_PLUGIN specify all the information needed to build the plugin.