External Control of ParaView Visualization

Oleg Kovalevskiy [1], Marwan Charara [1] and Michel Cancelliere [2]

[1] Aramco Research Center, Aramco Innovations LLC, Leninskie Gory 1-75b, Moscow, 119234, Russia
[2] Saudi Aramco, Dhahran, Saudi Arabia

Abstract

3D visualization is essential for many industrial applications. For instance, in the oil and gas industry, the visualization of reservoirs and their associated wells helps petroleum engineers to develop exploration and production strategies. For that purpose, there are integrated domain-oriented commercial software solutions. However, their 3D visualization part has significant limitations due to their closed source nature and dedicated domain. An alternative is to use general purpose open-source visualization platform such as ParaView. The challenge is to be used seamlessly in combination with in-house data management system which needs an integration effort. Embedding one application into another requires significant effort of programing, especially, when the technology stack of the data management application is different from the one ParaView is built on. In this study we present the solution of using ParaView along with a hypothetical corporate data-management application without embedding one into another. The inter-process communication protocol is based on ParaView Python API. The proposed approach allows using the full power of ParaView and data management application with minimal integration efforts.

Keywords

Scientific visualization, 3D, ParaView, GUI, inter-process communication, python.

1. Introduction

In many cases 3D visualization is essential to understand data. For example, in oil and gas domain visualization of reservoir and wells helps petroleum engineers to develop exploration and production strategy. The market offers integrated domain-oriented software suits for those purposes. However, commercial solutions might not cover all the demands of the users in terms of data management. It is quite common when there is custom data management system covering critical business processes of the company. If this system lacks 3D visualization, commercial visualization software most likely will not help. First, it might have proprietary data format which is enclosed, meaning that it is impossible to create datafiles without using that software. Second, even if loading data is not an issue, the visualization feature set is limited and most likely not extendable due to closeness nature of those applications. Third is performance: 3D visualization is usually not a feature of primary focus for the business-domain oriented commercial applications, so it is unlikely that they will have visualization part designed for scalability and high performance in case of big datasets.

ParaView is an open-source cross-platform general purpose visualization suite (Figure 1). In some cases, it can be a good alternative to visualizers of commercial packages. It has been developed by Kitware Inc. for more than since 2000. The core is VTK (visualization toolkit) – open-source SDK for high-performance cross-platform 3D visualization applications from the same vendor [1]. Desktop version of ParaView written with C++ is the most feature rich application of the whole variety [2]. It supports quite large number of formats – more than 100. The user can load datasets from files or create new objects using means of the application. Once the dataset is loaded or created it can be further modified in the application and saved later into a new datafile. New dataset can also be created by applying a filter. There are more than 200 filters in ParaView including slicers, thresholds, isosurfaces and many others. If standard filters or sources are not sufficient, it is possible to extend them by using

GraphiCon 2022: 32nd International Conference on Computer Graphics and Vision, September 19-22, 2022, Ryazan State Radio Engineering University named after V.F. Utkin, Ryazan, Russia
EMAIL: oleg.kovalevskiy@aramcoinnovations.com (O. Kovalevskiy); marwan.charara@aramcoinnovations.com (O. Charara)
ORCID: 0000-0003-2537-0035 (O. Kovalevskiy); 0000-0002-8343-5135 (M. Charara); 0000-0002-0612-0148 (M. Cancelliere)

python plugins [3]. Such extension does not even require recompiling ParaView. Another major feature of ParaView is that it is designed for performance and scalability. It can be used in client-server mode where the server part can run in multi-processing mode on a powerful node or even on a cluster, while the client part displays the resulting visualization on the ordinary user's machine [4]. This feature is quite important when business domain deals with large datasets and oil reservoirs are good example of those [5]. The last, but not least, a very useful functionality of ParaView is its Python Shell which allows script commands as a more flexible alternative to GUI controls [6]. The python API scripting is highly facilitated by the tracing functionality of GUI controls during manipulation.

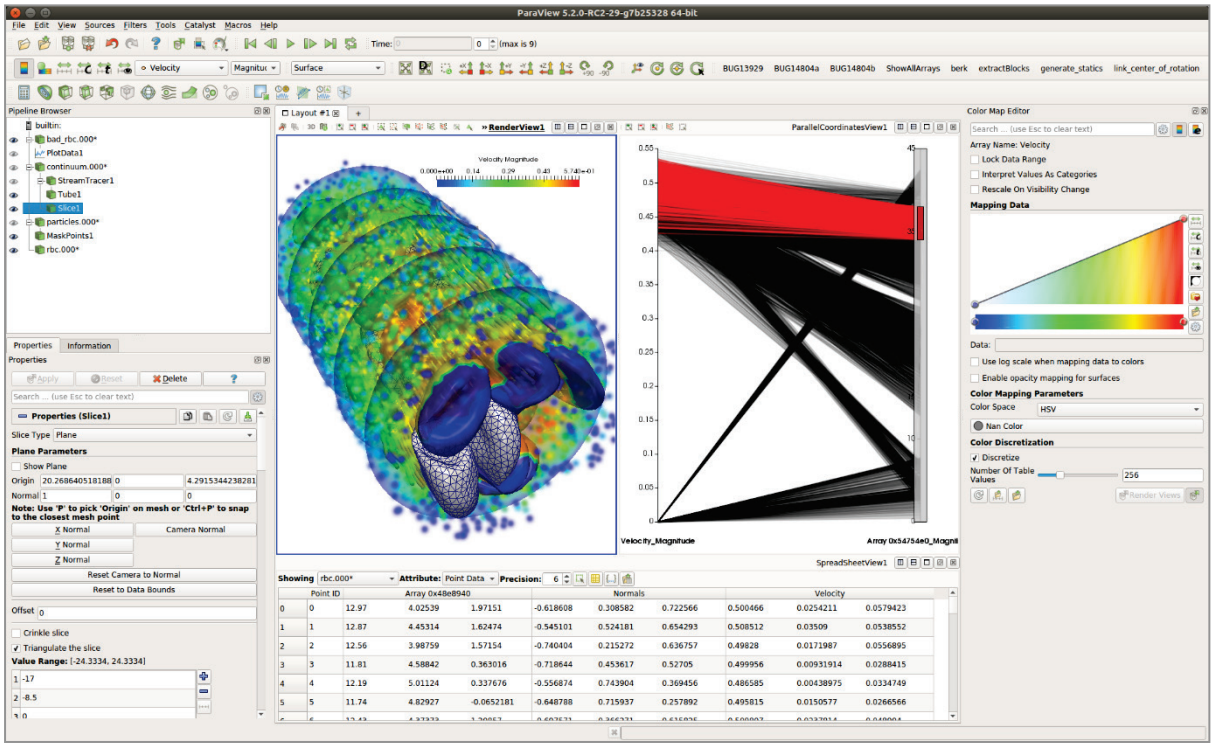

Figure 1: ParaView desktop general overview

If it is required to have a 3D visualization in a Data Management Application (DMA), ParaView can be a good choice. However, the strategy for integration is not straightforward. If the DMA is created using the same technology stack as ParaView, it is theoretically possible to embed one into another. However, it will require to understand the architecture of ParaView codebase (which is huge), to implement and to test the integrated solution, and finally, to ensure handling versions upgrade. In most cases, such approach will require too much effort to be practical. In case when DMA is written with something different from C++, the difficulty of such embedment increases even more. Another option which is the main topic of this paper is to use both existing Data Management Application and ParaView as two different executables and to enable communication between them, so that the visualizer can receive commands from DMA, execute them, display the result, and, optionally, send the reply back to the calling application.

2. Visualization in ParaView with GUI and Python Shell

For better understanding we will illustrate the common visualization use-cases which can be handled in ParaView GUI and its Python Shell by using a case example of the Johansen Dataset reservoir model which is publicly available [7].

Loading a dataset in ParaView is straightforward: *Main Menu → File → Open → (select file)*. When it is loaded, making it visible in *Pipeline Browser* will invoke rendering (Figure 2).

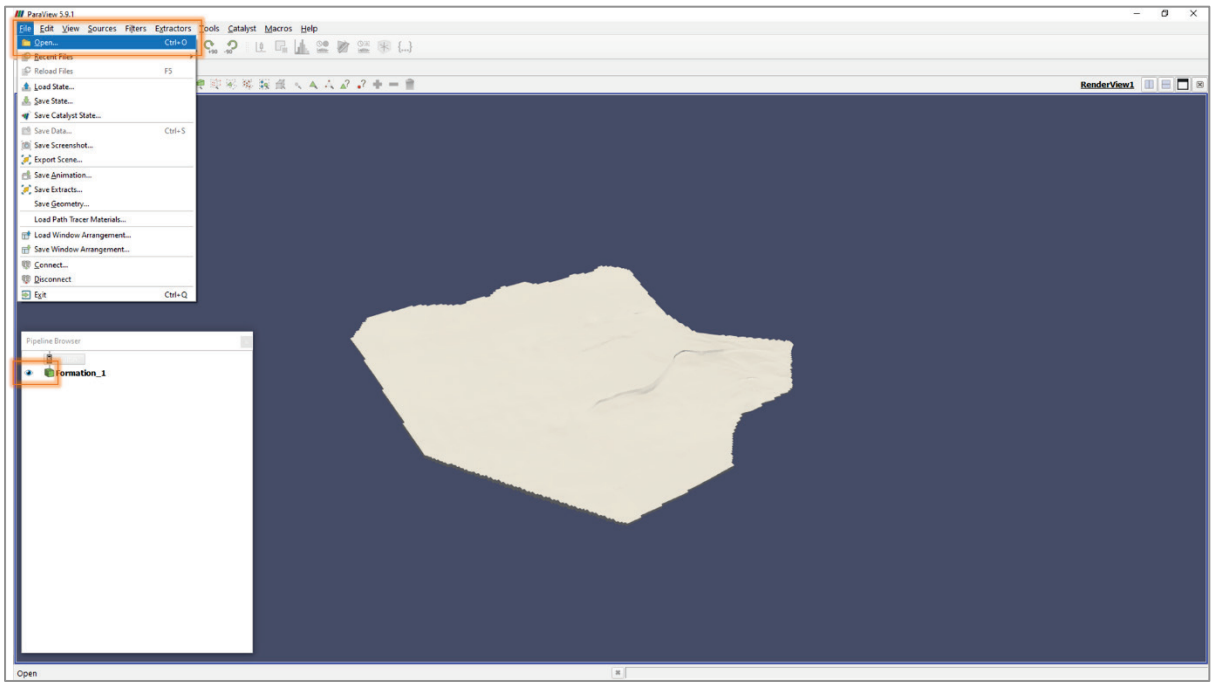

Figure 2: Loading a dataset in ParaView using GUI controls

It is also possible to load a dataset using *ParaView Python Shell* (Figure 3). The script to do this looks as follows:

```
from paraview.simple import *
formation_source = XMLUnstructuredGridReader(registrationName='…', FileName=[…])
formation_display = Show(formation_source)
GetActiveView().ResetCamera()
Render()
```

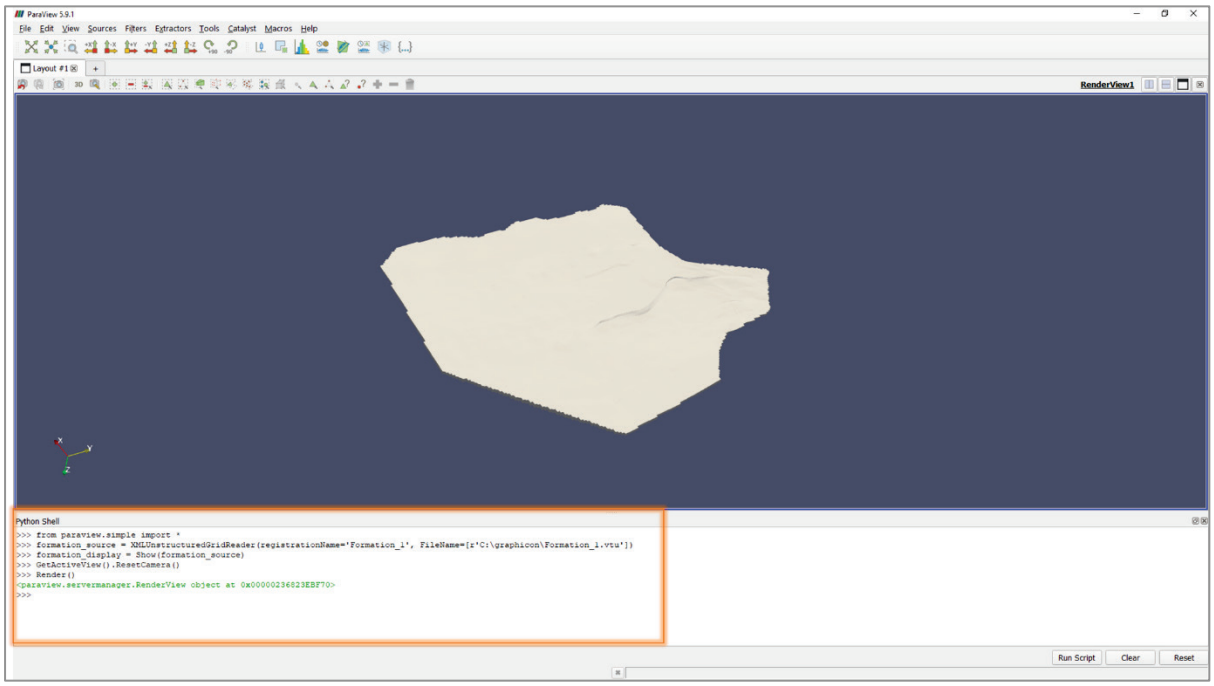

Figure 3: Loading a dataset in ParaView using Python Shell

To find these python commands one can go to online documentation [6] or use GUI actions tracing from *Main Menu* → *Tools* → *Start Trace*.
Default solid coloring can be easily edited in *Properties* panel and changed to the coloring related to one of the arrays in the dataset, for example *POROSITY* in our case. Also, it is possible to change scale in Transforming part. As our dataset has rather flat structure – the size in X and Y dimensions is much larger than in Z dimension, it is worth to change Z scale, for example, to 20 to have better insight into the layers of this formation (Figure 4).

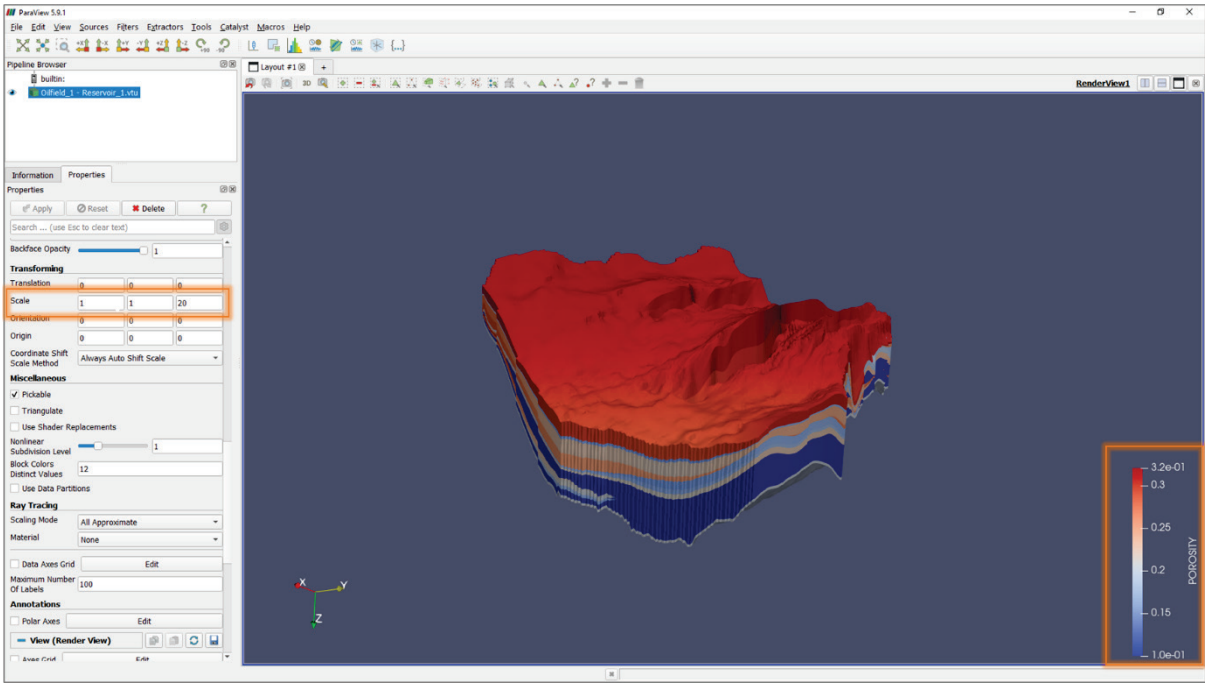

Figure 4: Changing scale and coloring of the dataset

Python equivalent of these actions is:

```
ColorBy(formation_display, ('CELLS', 'POROSITY'))
formation_display.Scale = [1.0, 1.0, 20.0]
```

For further enhancements of formation representation in terms of interior structure visibility it is possible to make the object semi-transparent. Again, it can be changed in the *Properties* panel or with short python command:

```
formation_display.Opacity = 0.33
```

Also, to distinguish the layered structure of the formation dataset it is possible to use *Threshold* filter. We'll use *POROSITY* as a base property for threshold and make it inverted with range from 0.2 to 0.3. It will make highly permeable middle layers hidden while keeping less permeable layers visible, indicating the barriers. Displaying grid by using *'Surface With Edges'* representation type will help to underline the curvatures of the layers. And original semi-transparent dataset shown on top of the threshold will help to display initial borders (Figure 5).

To apply a threshold filter using Python Shell:

```
threshold_source = Threshold()
threshold_source.Scalars = ['CELLS','POROSITY']
threshold_source.ThresholdRange = [0.2, 0.3]
threshold_source.Invert = True
threshold_display = Show(threshold_source)
threshold_display.Scale = formation_display.Scale
```

```
Show(formation_display)
threshold_display.SetRepresentationType('Surface With Edges')
```

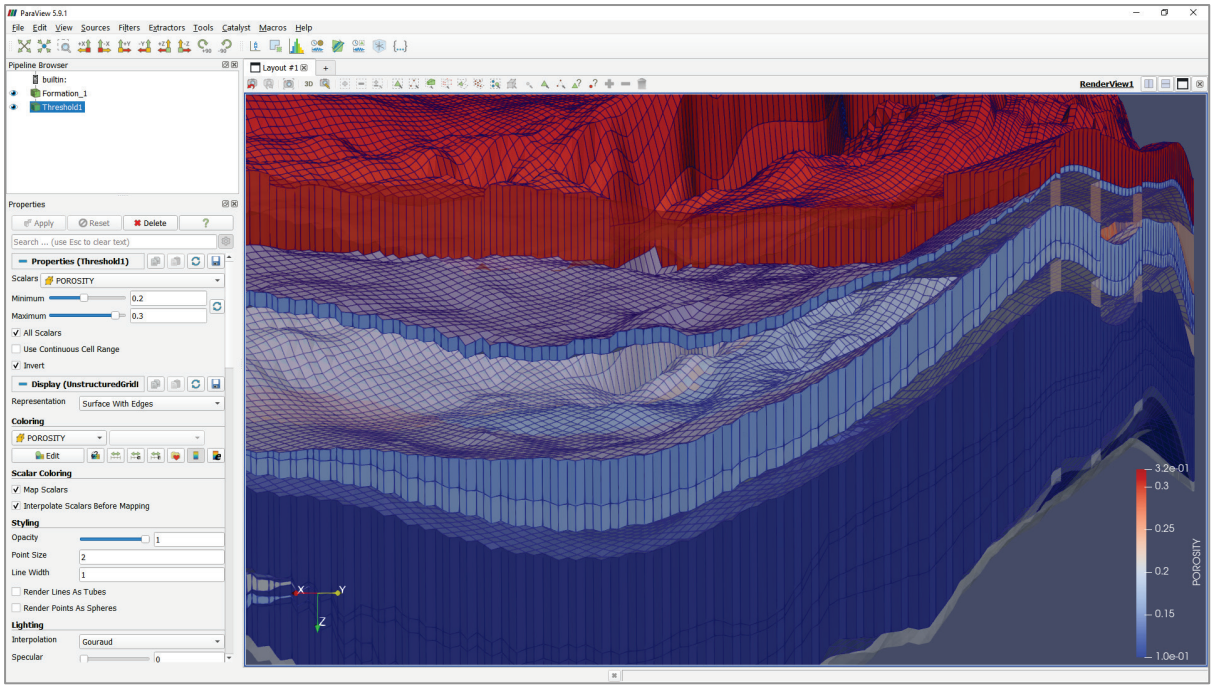

Figure 5: Changing opacity and adding property-based threshold filter

Now when the formation is loaded and visualized properly it's time to add wells. It is possible to load them from ParaView-compatible file, but they can also be created right in the app as a new *Geometric Shapes*. For this, we need to use *Main Menu* → *Sources* → *Geometric Shapes* → *Poly Line Source*. Then it is required to input XYZ coordinates of the points into the table in *Properties* panel. To make it look like a pipe, need to enable *Render Lines As Tubes* setting and set *Line Width* = 10.

To add labels with well names, we create new *Text* objects from *Sources* → *Annotation*. It is required to define the content – well name, and the position - at the top of the well. Also, it is possible to adjust font size and other parameters for more adequate representation (Figure 6).

Equivalent python code is:

```
well_source = PolyLineSource(registrationName = 'Well_1')
well_source.Points = [513000,6690000,0,513000,6690000,3200]
well_display = Show(well_source)
well_display.Scale = formation_display.Scale
well_display.RenderLinesAsTubes = True
well_display.LineWidth = 10


label_source = Text(registrationName='Label Well_1')
label_source.Text = 'Well_1'
label_display = Show(label_source)
label_display.TextPropMode = 'Billboard 3D Text'
label_display.BillboardPosition = [513000.0, 6690000.0, 0.0]
label_display.FontSize = 50
label_display.Bold = 1
label_display.Justification = 'Center'
```

For the second well the code is very similar with the only difference in its name and geometry.

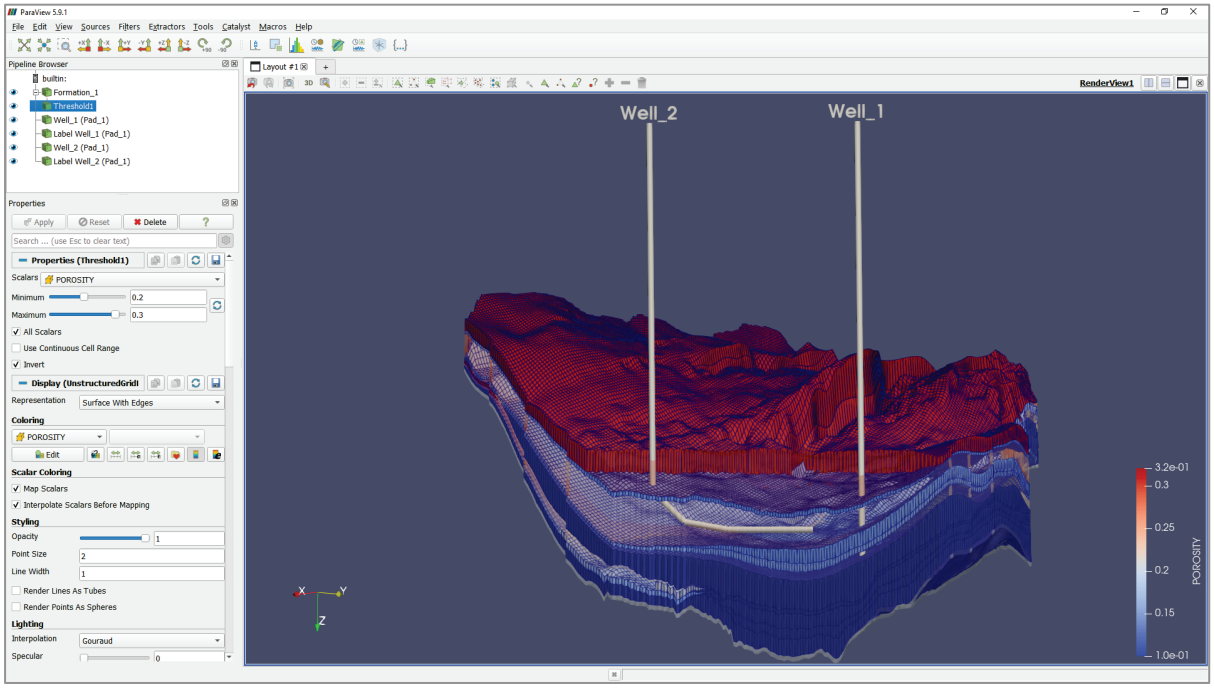

Figure 6: Adding wells as *Poly Line* sources and their labels as *Text* sources

3. Integration with Data Management Application

We have just reviewed how to load a formation dataset form file in ParaView-compatible format and how to create wells objects with manual input of properties. In real situation data may be stored in a database or in the files of the format oriented to the business systems, most likely different from what is required by ParaView. So, to make a visualization of the real data it needs to be converted into visualization-friendly format and stored on the same machine where ParaView is. If we think about wells, there can be hundreds of them in one formation. Of course, manual creation and data input as demonstrated above is not practical at all. I order to automate this it is possible to create python macros based on the scripts demonstrated above and then call them.

```
def load_dataset(file_path):
    import os
    dataset_name = os.path.splitext(os.path.basename(file_path))[0]
    formation_source = XMLUnstructuredGridReader(dataset_name, [file_path])
    formation_display = Show(formation_source)
    GetActiveView().ResetCamera()
    Render()

def create_well(pad_name, well_name, trajectory):
    well_source = PolyLineSource(registrationName = well_name + ' (' + pad_name +')')
    well_source.Points = trajectory
    ...

def change_z_scale(value):
    ...
...
```

However, this approach also has some disadvantages. Manual input of the input parameters to those macros will lead to regular errors. Also, petroleum engineers might feel uncomfortable working with python code as they are usually not professional programmers. On the other hand, if there is some data management system with at least basic GUI it is possible to manage those python macros using that app. Let's consider for definiteness that we have such system and it's a Windows desktop application created with C# WinForms [8] (Figure 7).

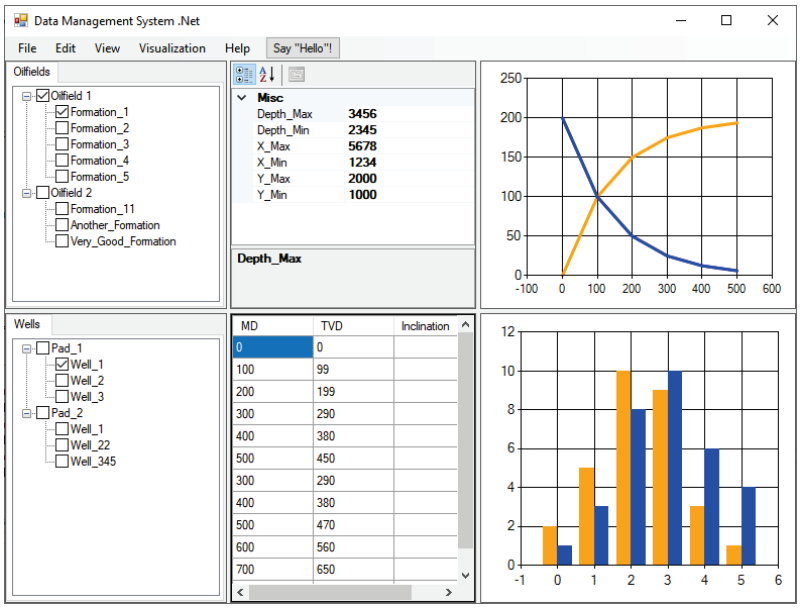

Figure 7: Simplified example of hypothetical oilfield Data Management Application (DMA)

When user interacts with this application, he selects some objects, reviews or edits their properties, makes some statistical analysis, or even runs a simulation. However, it would be nice to enable 3D visualization that can be quickly invoked from this application. It will be possible if we find a way to send python script from the DMA to ParaView.

There are different ways to do that and the simplest one is using means of operating system to capture the ParaView Python Shell window handle and then send keys to it as if user input those keys there directly. To implement *Python Shell* window capture we employ *user32.dll*– a Windows system library containing operating system management functions for message handling, timers, menus, and communications [9]. Though *user32.dll* is a native library it can be used from C# code if its methods are decorated with *DLLImportAttribute* [10]. Methods from that DLL help to find a ParaView application in the list of running processes and then identity Python Shell window and get its handle. Once window handle is captured, we can use *SendKeys.Send* method from *System.Windows.Forms* namespace to transfer messages from C# application to that *Python Shell* window [11] (Figure 8).

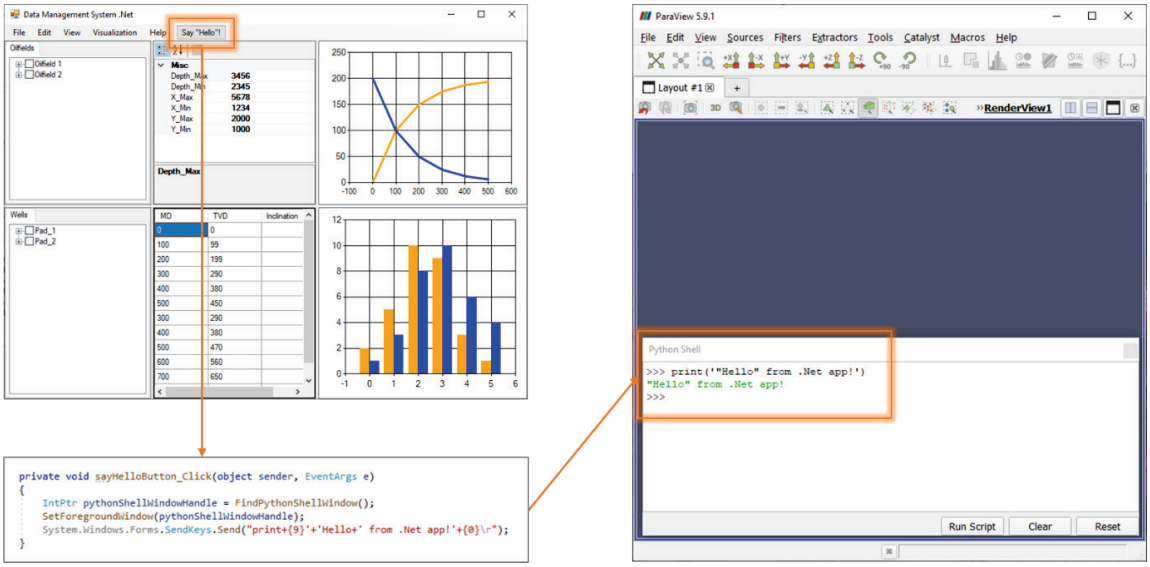

Figure 8: Example of inter-process message exchange

Using this mechanism now we can send our oilfield visualization commands. For example, when user ticks *Formation* checkbox, ParaView loads it from file.

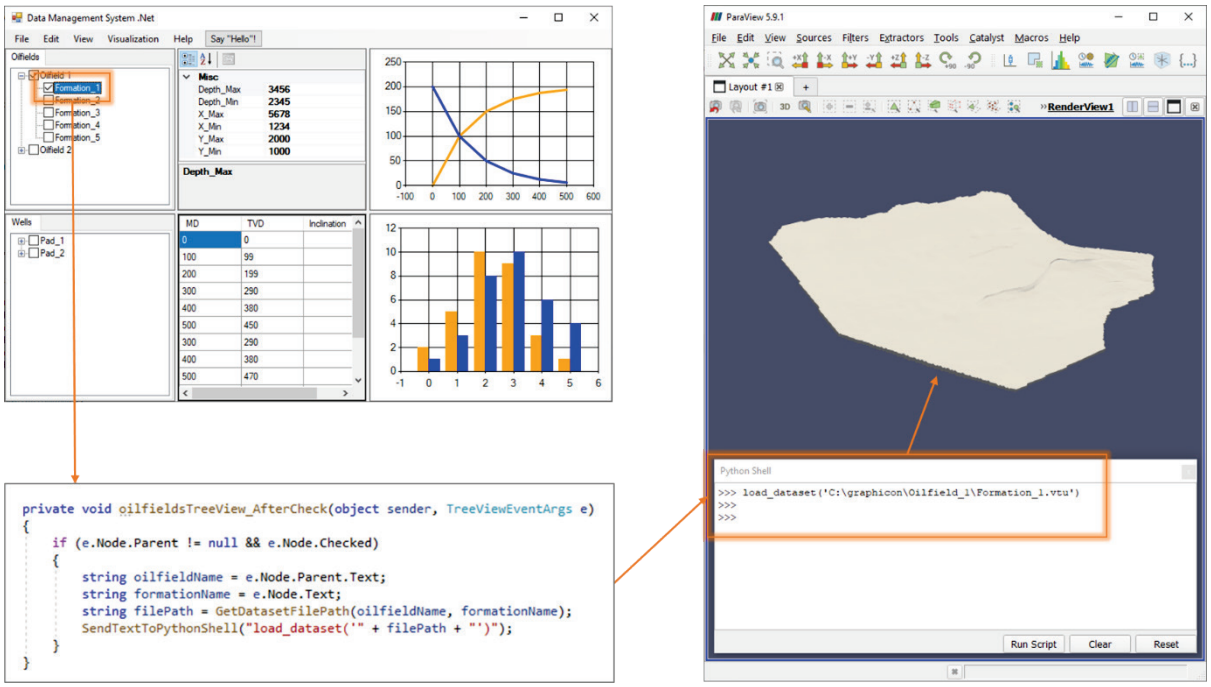

Figure 9: Invocation of dataset visualization from DMA GUI

In this example it is assumed that datasets are stored in **.vtu* files – *ParaView Unstructured Grid* format. But if they are not, one can add an action to convert the data into this format and store it in temp file. The C# code in this example is simplified. In real implementation it would first check if the dataset is already loaded. Checking/Unchecking will be corresponding to *Show()* and *Hide()* actions in visualizer.

Wells can be visualized using similar approach (Figure 10).

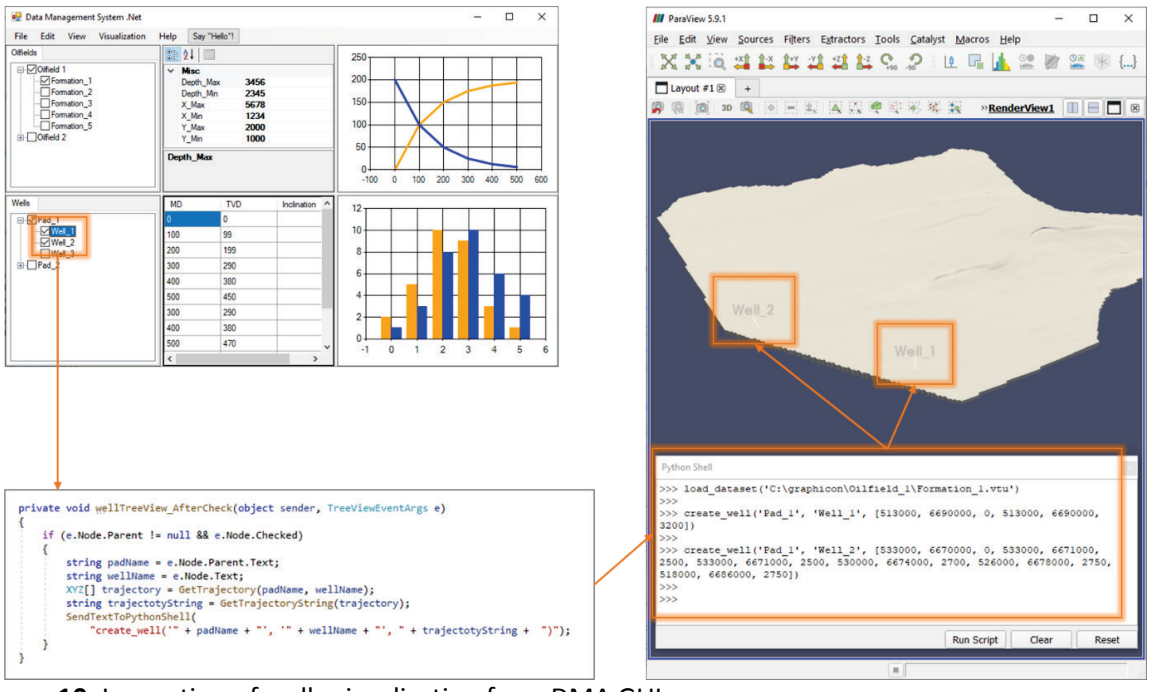

Figure 10: Invocation of wells visualization from DMA GUI

The default visualization of the formation and wells here is not very representative, so we might want to change it as before by adjusting Z-scale, using property-based coloring, transparency, threshold, line width and font size. We can do it in ParaView GUI, but in this case we'll need to edit several properties of each object individually. If there are few formations, and few hundreds of wells it will be very time consuming and error prone. As an alternative we can configure manipulation of those parameters from our Data Management App. We'll add visualization control panel (Figure 11).

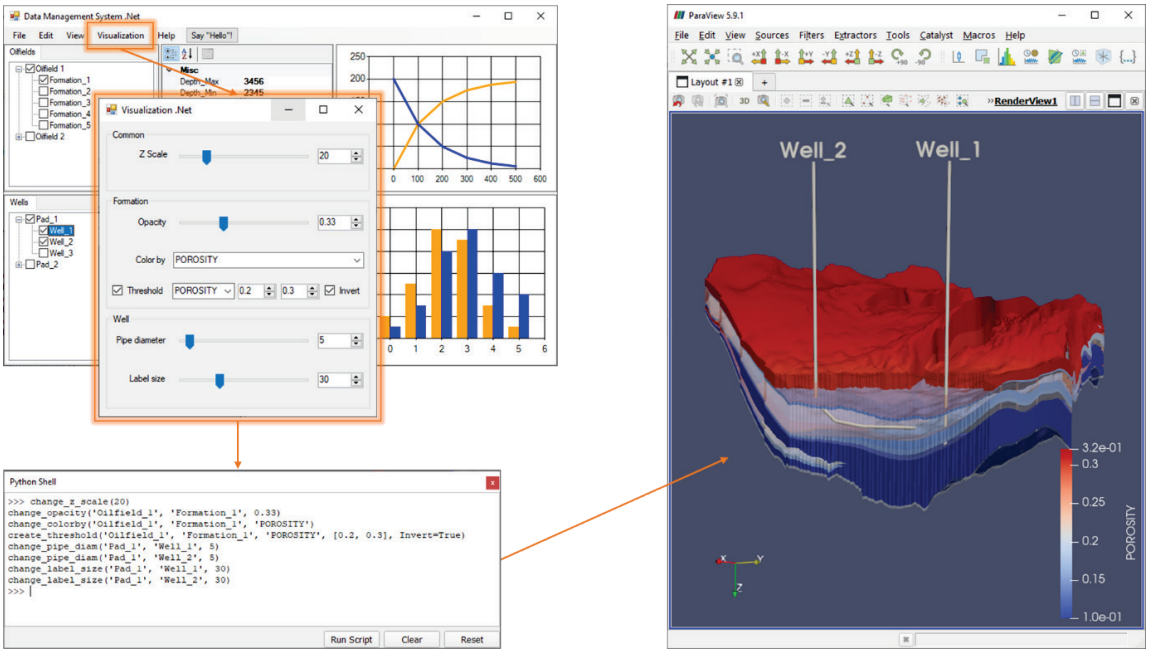

Figure 11: Invocation of visualization parameters adjustment from DMA GUI

At this point we stop with visualization commands examples. But there is one important thing left. Sometimes getting response from the visualizer into the GUI might be required (Figure 12). And there is a way to enable this in current configuration. The simplest will be to transfer backward messages using files. Python scripts sent to ParaView should contain a code writing those messages to the file, while DMA should run and infinite loop monitoring for that file change.

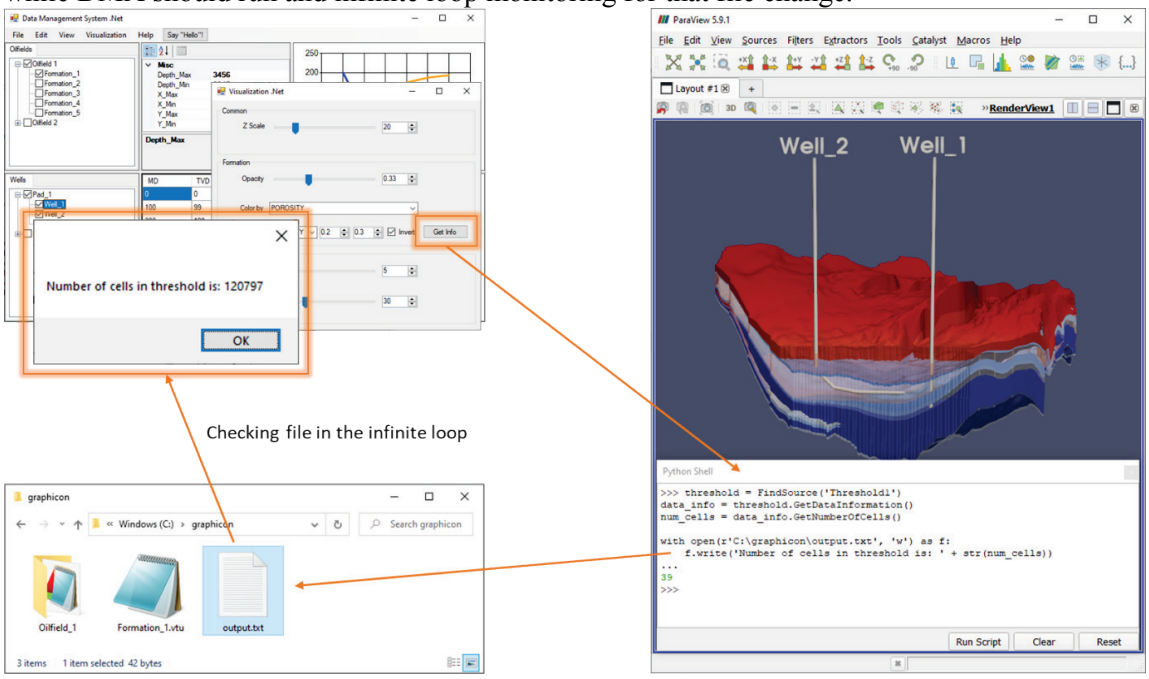

Figure 12: File-based mechanism of getting visualizer response

4. Conclusion

3D visualization can significantly improve data management in different business processes. Though it might seem difficult to implement a feature-rich visualization into in-house system or even impossible to efficiently integrate with existing commercial visualizers, there still is at least one solution. In this study we have demonstrated an easy way of using such a powerful 3D visualization system as ParaView in conjunction with hypothetical custom Data Management Application. Python API of ParaView makes it possible to program visualization scenarios as python scripts. To pass those scripts here we use manipulation of windows handles and keystrokes simulations. However, this is not the only method, and it might have some disadvantages, it allows using ParaView as it is, without introducing any modifications in its source code and compiling it. We have also demonstrated a way to receive a reply form visualizer using shared file.

5. References

[1] Kitware Inc. VTK User's Guide. paperback edition, 3 2010. ISBN 978-1930934238.
[2] Kitware Inc., ParaView guide. URL: https://docs.paraview.org/en/latest.
[3] The ParaView Community, ParaView / Plugin HowTo. URL: https://www.paraview.org/Wiki/ParaView/Plugin_HowTo.
[4] The ParaView Community. Setting up a ParaView server. URL: http://www.paraview.org/Wiki/Setting_up_a_ParaView_Server.
[5] P. Novikov, D Sabitov, N. Bukhanov, M. Charara, M. Cancelliere, F. Rashed, & A. Baiz. (2022). Efficient Visualization Methods for Large Scale Reservoir Models. Conference Proceedings, 83rd EAGE Annual Conference & Exhibition, 2022(1), 1–5. https://doi.org/10.3997/2214-4609.202210139.
[6] Kitware Inc., ParaView python online documentation. URL: https://kitware.github.io/paraview-docs/latest/python/paraview.simple.html.
[7] G.T. Eigestad, H. K. Dahle, B. Hellevang, F. Riis, W.T. Johansen, and E. Øian. Geological modeling and simulation of CO2 injection in the Johansen formation. URL: https://doi.org/10.1007/s10596-009-9153-y.
[8] Microsoft corp., Create a Windows Forms app in Visual Studio with C# URL: https://docs.microsoft.com/en-us/visualstudio/ide/create-csharp-winform-visual-studio?view=vs-2022.
[9] Microsoft corp. Identifying Functions in DLLs. UrL: https://docs.microsoft.com/en-us/dotnet/framework/interop/identifying-functions-in-dlls.
[10] Microsoft corp. Consuming Unmanaged DLL. URL: Functionshttps://docs.microsoft.com/en-us/dotnet/framework/interop/consuming-unmanaged-dll-functions.
[11] Microsoft corp., System.Windows.Forms Namespace. URL: https://docs.microsoft.com/en-us/dotnet/api/system.windows.forms?view=windowsdesktop-6.0.