Performance of ParaView Grids for Reservoir Models Visualization

Oleg Kovalevskiy [1,2], Michel Cancelliere [2]

[1] *Aramco Research Center, Aramco Innovations LLC, Varshavskoe highway, 9 bld. 1, 117105 Moscow, Russia*
[2] *Saudi Aramco, Dhahran, Saudi Arabia*

Abstract
Interactive visualization of subterranean formations models is essential for reservoir engineering. ParaView is a good option - it is feature rich, performance oriented, cross-platform and free and open-source. One of the popular formats for reservoir model storage is GRDECL, which fits well with the nature of the reservoir data and yet is memory efficient. ParaView deals with more generic data formats such as image, structured and unstructured grids, polygonal data. GRDECL reservoir data format has features of both structured and unstructured ParaView grids. However, none of them ideally fit. Using structured grid is very economical, but it can lead to losing some information. Unstructured grid provides precise display, however, requires much more resources. This study illustrates quantitative difference between visualizations based on these two grid types and gives considerations for their application.

Keywords
Scientific visualization, 3D, ParaView, VTK, python, structured grid, unstructured grid, memory, computational efficiency.

1. Introduction

Visualization of reservoirs models is essential for developing engineering strategies of oil and gas production. High-resolution models provide better opportunities for high quality analysis, however, they have larger sizes. From a certain point the size can make visualization to be challenging, processing and rendering will take more time and require more memory. There can be numerous approaches to visualize such models. Using ParaView is one of them [1]. This tool is feature rich, yet extendable with plugins [2]. It is performance oriented – it's back end can run in multi-process environment, while even further separated to data-server and render-server [3]. This feature is quite important when business domain deals with large datasets and oil reservoirs are good example of those [4]. It is cross-platform – can run on Windows or Linux, or even on both at the same time if used in client-server mode. It is available for free together with its codebase should one require any customization. The core is VTK (visualization toolkit) – open-source SDK for high-performance cross-platform 3D visualization applications from the same vendor [5]. Reservoir models usually come in specific formats that are traditionally used in the industry and formed by the nature of the reservoir structures. They comprise continuous media organized by layers. The data comes usually from seismic survey and model is further used in production simulations. One of such formats is GRDECL [6]. It represents the data in a layered way, however, supports raptures and faults that may occur in reservoir models. ParaView formats are more generic. Two of them that are the closest to handle GRDECL data structure are *Curvilinear or Structured Grid* (aka *.vts) and *Unstructured Grid* (aka *.vtu) [1]. Structured Grid for 3D is a variant of rectilinear grid, where the cell configuration is relaxed to be just a hexahedron. Unstructured grid is the most flexible option. It supports cells of various types. Number of cells and their position can be any. So, it can reproduce the configuration of any structured grid, however, it will require significantly more memory. Detailed description of all the formats is provided below. In this study we'll compare the data configuration, memory consumption, processing and rendering time for one reservoir model

GraphiCon 2023: 33rd International Conference on Computer Graphics and Vision, September 19-21, 2023
V.A. Trapeznikov Institute of Control Sciences of Russian Academy of Sciences, Moscow, Russia
EMAIL: oleg.kovalevskiy@aramcoinnovations.com (O. Kovalevskiy); cancelma@sauptech.com
ORCID: 0000-0003-2537-0035 (O. Kovalevskiy); 0000-0002-0612-0148 (M. Cancelliere)

example when visualized in ParaView using structured and unstructured grids with different configurations.

2. Data formats

2.1. GRDECL

It is short of “Grid ECLIPSE”, where ECLIPSE is a commercial reservoir simulator by Schlumberger (now SLB). The grid is a 3D matrix of hexahedron cells with fixed N_x, N_y and N_z dimensions [6]. Vertices of the cells lay on the “vertical spokes” as shown on figure 1.

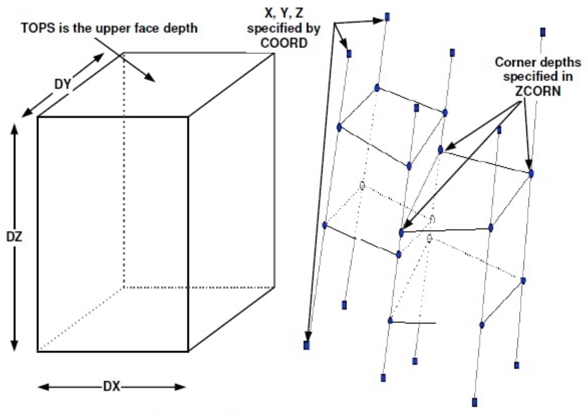
TOPS is the upper face depth
X, Y, Z specified by COORD
Corner depths specified in ZCORN
DY
DZ
DX

Figure 1: Block centered and corner-point geometries

Tops of those spokes are described with two 2D matrices of points in the data file part called “COORD” and the positions of the vertices on the spokes are provided in 3D matrix of z-coordinates in the part called ZCORN.

The dimensions of *COORD* and ZCORN are the following:

$\mathrm{COORD}: 2 \times N_x^p \times N_y^p \times 3, \quad where\ N_x^p, N_y^p - number\ of\ points\ in\ x\ and\ y\ directions$

$\mathrm{ZCORN}: N_x^c \times N_y^c \times N_z^c \times 8, \quad where\ N_x^c, N_y^c, N_z^c - number\ of\ cells\ in\ x, y\ and\ z\ directions$

$N_{x(y,z)}^p = N_{x(y,z)}^c + 1$

This helps to understand how much memory is required to store the dataset of specific dimensions in binary format, considering *float* or *double* representation of real numbers.

2.2. ParaView Structured Grid

A curvilinear grid, as illustrated on figure 2 defines its topology implicitly and point coordinates explicitly [1].

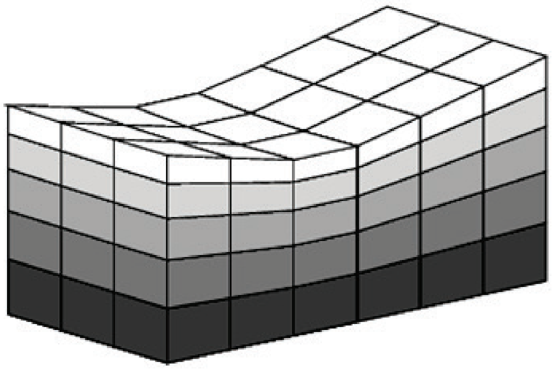

Figure 2: Curvilinear or structured grid

To fully define the mesh for a curvilinear grid, VTK uses the following:
Extents - These define the minimum and maximum indices in each direction. For example, a curvilinear grid of extents (0,9), (0,19), (0,29) has 10×20×30 points regularly defined over a curvilinear mesh.
An array of point coordinates - This array stores the position of each vertex explicitly.

A curvilinear grid consists of cells of the same type. This type is determined by the dimensionality of the dataset (based on the extents) and can either be vertex (0D), line (1D), quad (2D), or hexahedron (3D).So, the dataset is fully described by $N_x^p \times N_y^p \times N_z^p$ “doubles”.

2.3. ParaView Unstructured Grid

An unstructured grid, as illustrated on figure 3, is the most general primitive dataset type. It stores topology and point coordinates explicitly.

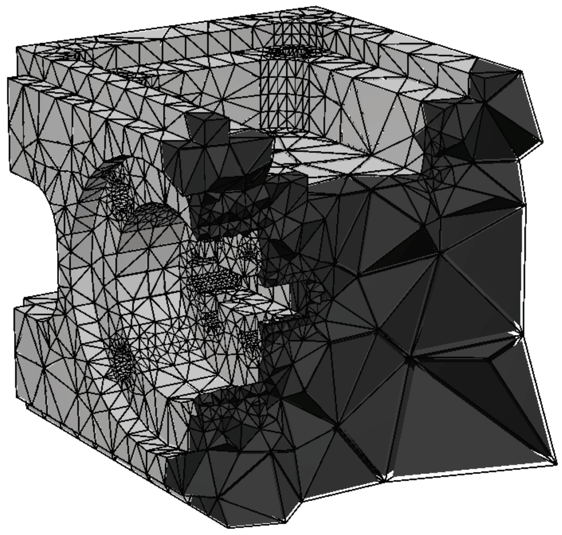

Figure 3*: Unstructured grid

Even though VTK uses a memory-efficient data structure to store the topology, an unstructured grid uses significantly more memory to represent its mesh. Therefore, use an unstructured grid only when you cannot represent your dataset as one of the above datasets. VTK supports a large number of cell types, all of which can exist (heterogeneously) within one unstructured grid.
The dataset is defined by the array of points and *connectivity* – array of cells. Each type of cells has its own format of connectivity record. In this study we deal with hexahedron cells. Each cell of this type is defined by the set of nine 8-byte integers – eight vertices indices + one service value.

$\text{POINTS:}\, N_x^p \times N_y^p \times N_z^p \times 3\ of\ doubles$

$\text{CONNECTIVITY:}\, N_x^c \times N_y^c \times N_z^c \times 8\ of\ longs$

*picture source - https://discourse.paraview.org/ - Conversion of unstructured points to a connected unstructured grid (auto add edges))

3. Dataset

The Johansen formation is a candidate site for large-scale CO2 storage offshore the south-west coast of Norway. It is made available under the Open Database License [7]. Any rights in individual contents of the database are licensed under the Database Contents License [8].
The simulation grids are made available in corner-point format. This format is an extension of the logical Cartesian grid format. Due to the vertical faulting of geological zones, the corners of grid cells are in general not conforming from one grid block to the neighboring grid block.
The full-field model is discretized by a 149x189x16 grid. This grid describes all zones and the entire lateral domain (figure 4).

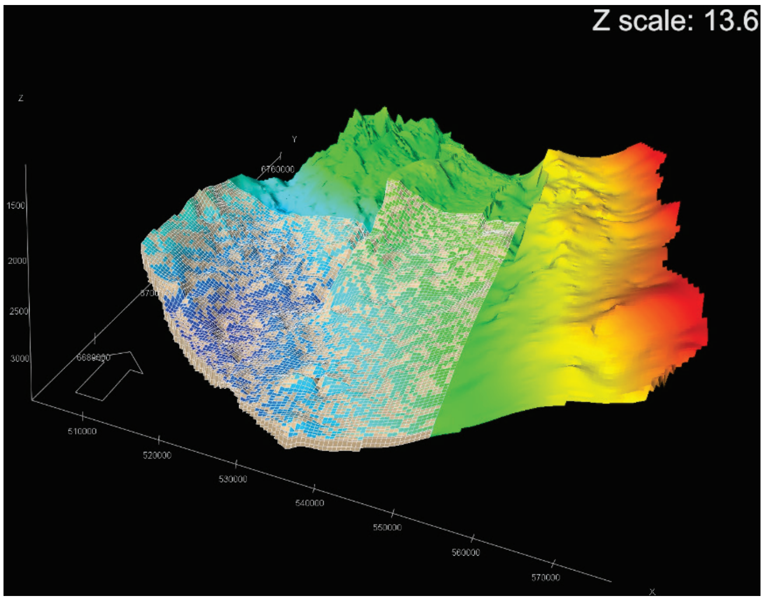

Figure 4: Johansen dataset visualization from the initial source

The Johansen formation has interbedded shale layers. The full field model is based directly on the geomodel where these shales are represented as pinched out, separate grid layers. [9]

4. Experiments and Analysis

In order to visualize Johansen dataset we have developed python scripts that read and parse GRDECL file, then convert it to either structured or unstructured ParaView grid format, create corresponding object and invoke its rendering. Python scripts are executed using ParaView *Python Shell* [10].

4.1. Unstructured grid with all points

The structure of data in GRDECL file describes every cell by separately defined vertices. Even if z coordinates of corresponding vertices of adjacent cells are the same, they will be two duplicated values in GRDECL file. This is close to the definition of unstructured grid, where each cell is defined by it's vertices individually. The result of visualization of this case using this type of grid where 8 points are created for each cell is shown on figure 5 below.

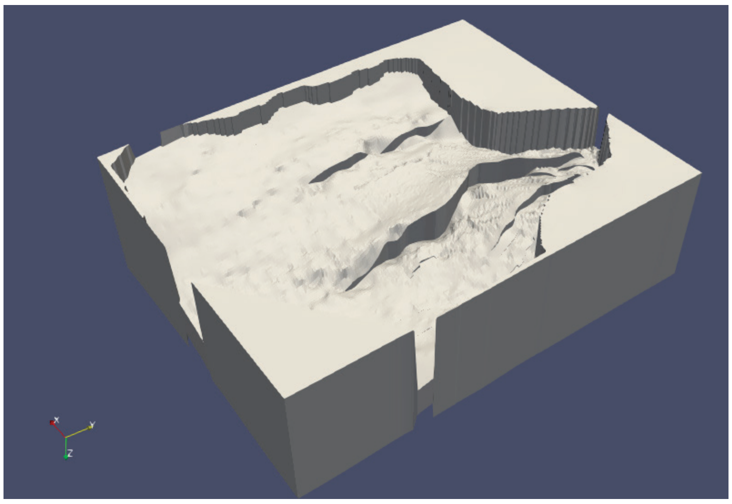

Figure 5: Visualization using unstructured grid with duplicated points

ParaView reports that the resulting grid contains 450 576 cells, 3 604 608 points and takes 113.873 MB of memory. However, huge consumption of memory for this type of representation is not the only problem. It also creates a very high load on rendering. When adjacent cells have vertices that are identical, but defined as different points, ParaView will create visible edges and faces for those vertices. This leads to graphical artifacts when transparency or *wiremesh* representation type is used as shown on figure 6.

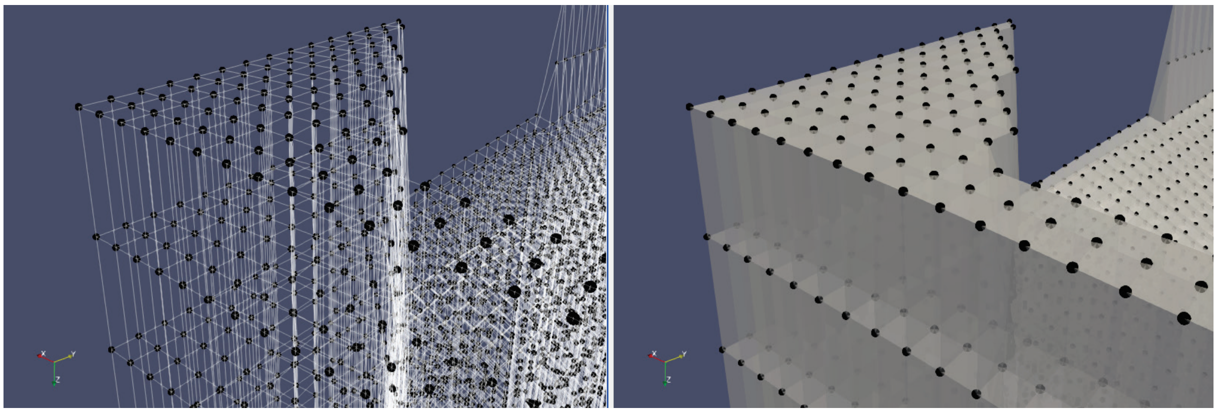

Figure 6: Edges and faces of unstructured grid cells with duplicated points

So, for more efficient processing and more representative graphical representation it is better not to duplicate identical vertices.

4.2. Unstructured grid with merged points

Now we will visualize the same dataset but with less points. Duplicated points are merged, only unique vertices are defined as unstructured grid points. The result is below on figure 7.

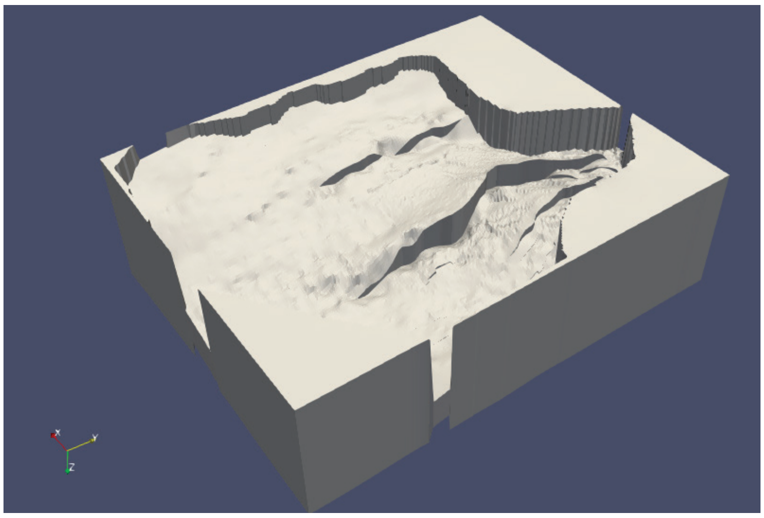

Figure 7: Visualization using unstructured grid with unique points

The representation is identical. However, way less resources were utilized to get it. The number of cells remains the same - 450 576, but the number of points is significantly reduced – to 495 601, resulting total memory usage of 42.7139 MB only. Comparison is illustrated on the plot on figure 8.

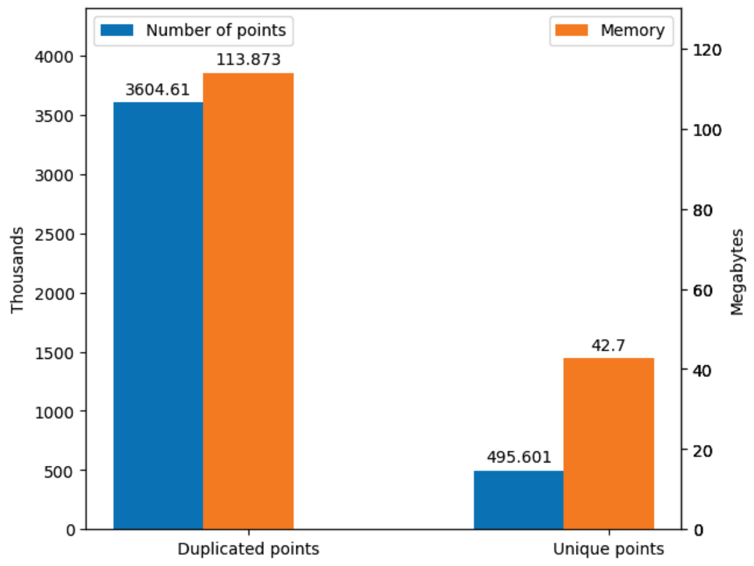

Figure 8: Unstructured grids configuration and memory consumption

Also, with this approach the representation in the form of wireframe (figure 9) or transparent surface (figure 10) is better, because only outer edges and faces are visualized.

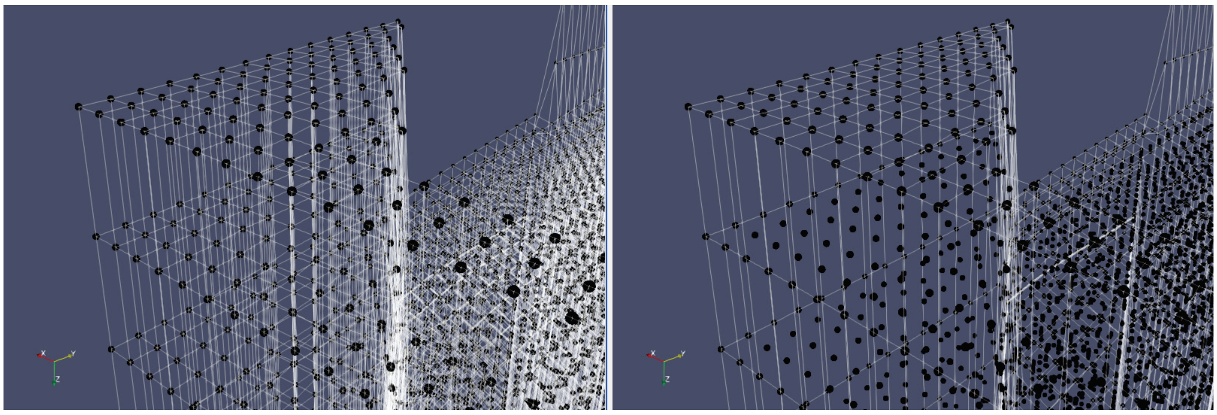

Figure 9: Edges of unstructured grid cells with duplicated points(left) and with unique points(right)

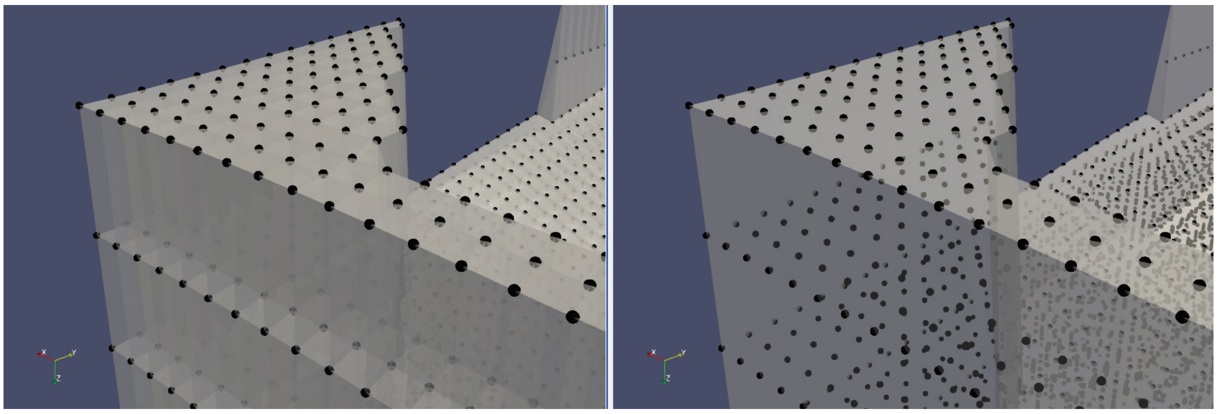

Figure 10: Faces of unstructured grid cells with duplicated points(left) and with unique points(right)

The effect of this difference is demonstrated on figure 11 and figure 12 below. Redundant edges and faces make it difficult to observe geometrical features of the model.

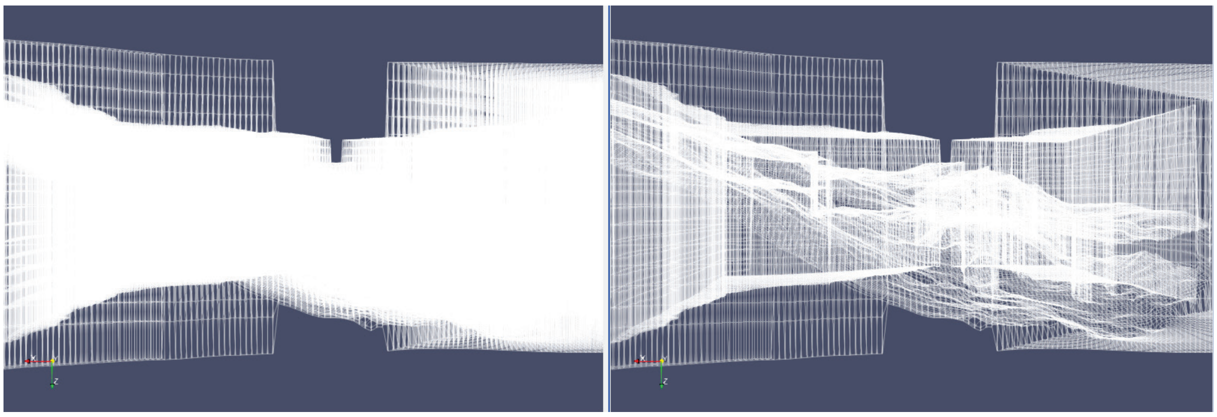

Figure 11: High-level difference between visualization using unstructured grid cells with duplicated points(left) and with unique points(right). Representation - wireframe.

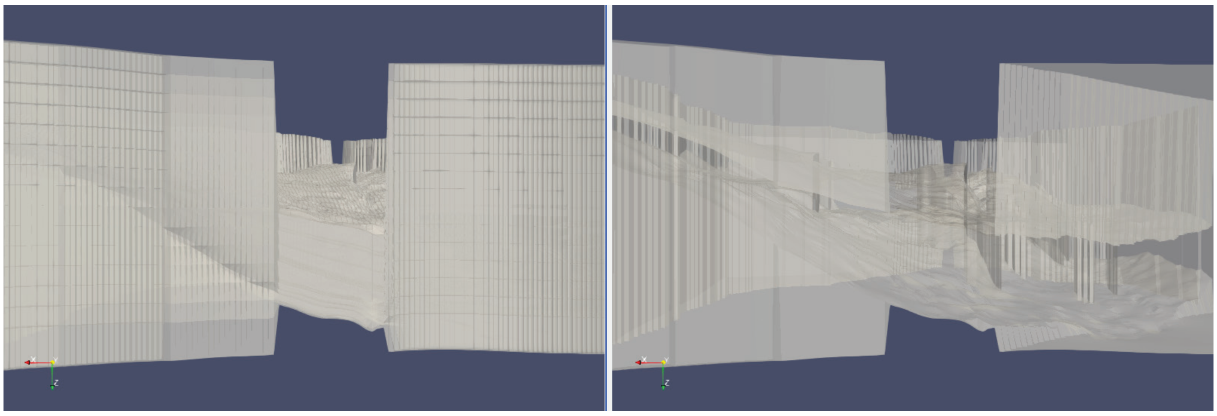

Figure 12: High-level difference between visualization using unstructured grid cells with duplicated points(left) and with unique points(right). Representation - transparent surface.

Building grid with duplicated points is quite straightforward. However, identifying unique points is trickier. We do it in three steps.

1. **Building base grid.**
   This is a set of vertices as if all adjacent cells had common vertices. In other words, we take every other point from the set of duplicated points in every of x, y and z directions.
2. **Identify the nodes with non-matching points.**
   Some of the adjacent cells might not conform, we need to find the corners with such vertices.
3. **Add new points and change cell references**
   When nodes with non-matching vertices are identified, we create those additional vertices and add them to the set of base grid points. Also, we need to put the reference to those new points as vertices for the corresponding cells in the connectivity array.

In our case base grid consists of 484 500 points. The number of additional points identified is 11 101. They comprise 2.2%. So, we can notice that almost all points (~98%) come from the base grid. Additional points are located only at faults – local shifts in dataset geometry. Our base grid falls exactly under the definition of ParaView structured grid. Let’s visualize just base grid using it.

4.3. Base grid as structured grid

To get this visualization ParaView created structured grid with 450 576 cells and 484 500 points. Such grid requires only 11.1 MB of memory. The result looks very similar to what we got using unstructured grid with precise points processing as shown on figure 13.

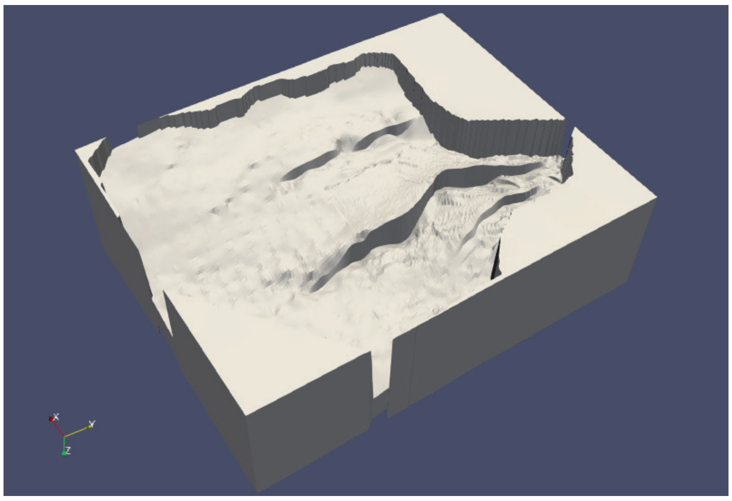

Figure 13: Visualization using structured grid with base points

We can notice slight geometrical difference on figure 14 only when two grids are visualized with the same settings and camera position.

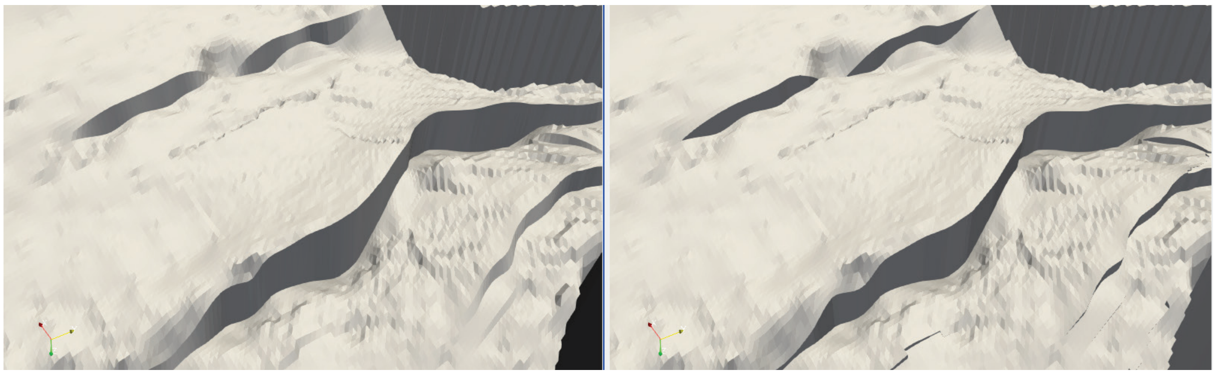

Figure 14: Visual difference between the image obtained using structured grid with base points(left) and unstructured grid with all unique points(right)

We can also see the difference in grid geometry at faults when two grids overlap as figure 15 shows.

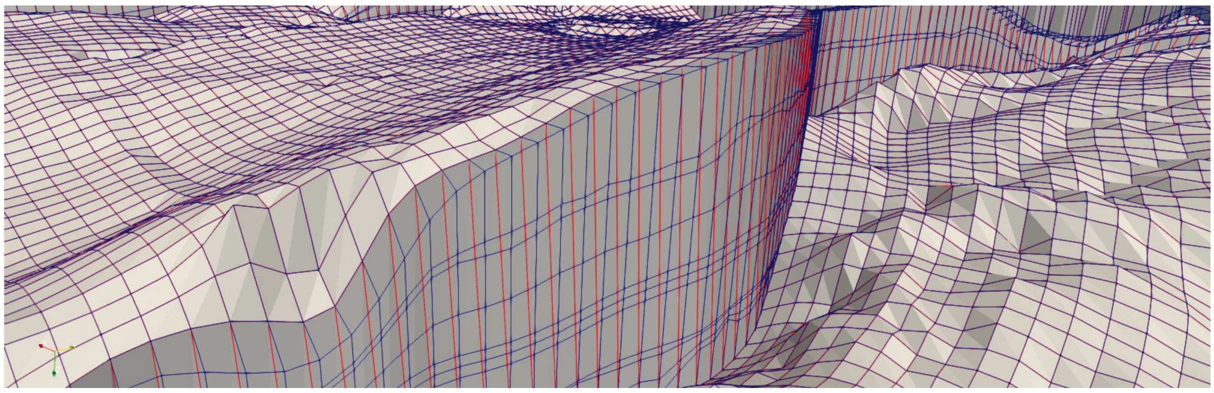

Figure 15: Difference in geometry between structured grid with base points(red) and unstructured grid with all unique points(blue)

Plot on figure 16 shows that structured grid is much more efficient in terms of memory. It is also faster in processing, we'll show this later on a larger scale case. But we know that the representation is not exact, some information is lost. But can we achieve precise representation with ParaView structured grid?

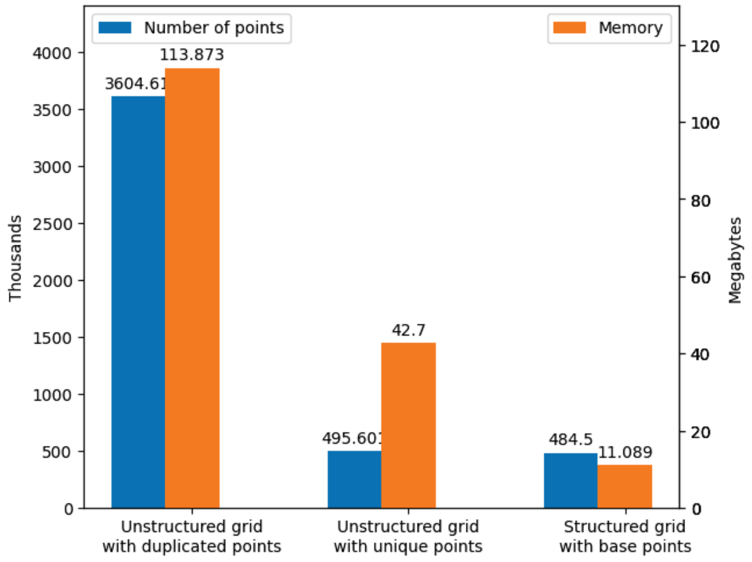

Figure 16: Structured and unstructured grids configuration and memory consumption

4.4. Faults as additional slices

To achieve precise representation, we need to add the points defining faults to the base grid. But to add new point we'll need to add the whole new slice. It can be i-, j- or k-slice. However, the structure of GRDECL file tells us that if two cells that are adjacent in z-direction have non matching vertices, there will be a discontinuity and this case cannot be represented by structured grid. Luckily in our dataset we don't have such cells/points, so we stick to i-, and j- cells only. We need to identify all unique i- and j-indices of new points. In our case there are 57 and 73 unique indices correspondingly. So, we'll add 57 new i-slices with new points where needed and duplicated points where we don't have any difference. Similarly, we'll add 73 new j-slices. Below on figures 17 and figure 18 are the results of our visualization comparing to original unstructured grid.

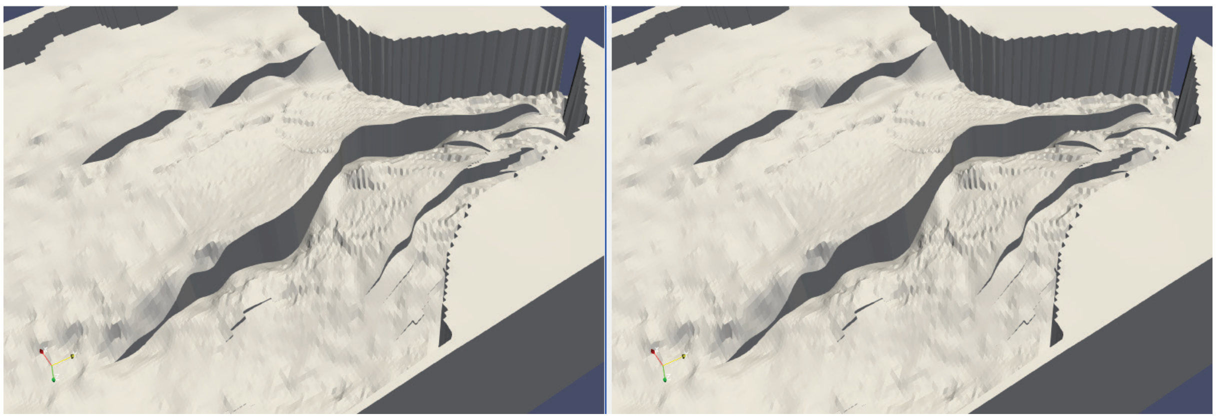

Figure 17: Visualization using structured grid with additional slices (left) and unstructured grid with all additional points (right). They are identical

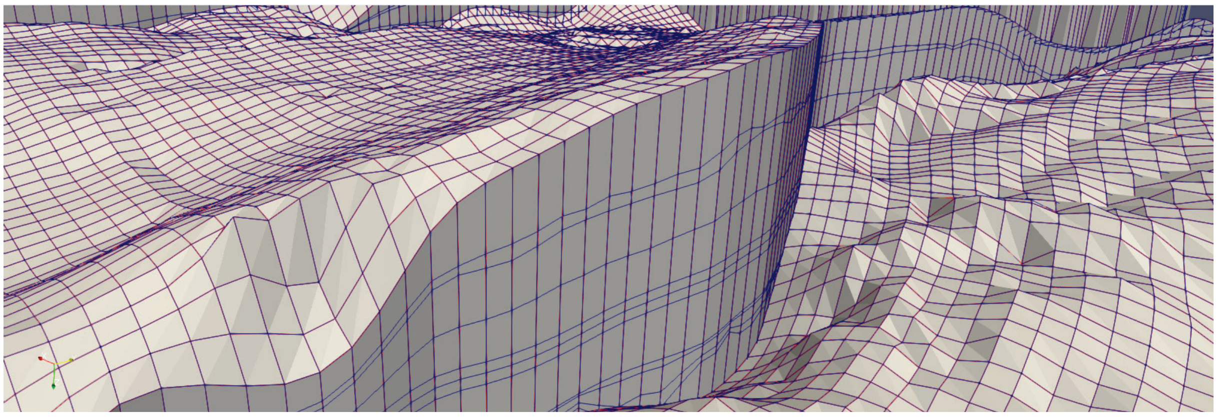

Figure 18: Geometries of structured grid with additional slices(red) and unstructured grid with all unique points (blue). They are identical

Now the representation is precise and it matches exactly with that created using unstructured grid. However, we added several additional slices, that means a lot of additional points – 440 997. This is much more (+91%) than was in case of structured grid (+2%). But even with those additional points structured grid takes only 21 MB of memory, which is still twice as low as in case of unstructured grid as demonstrated on the plot on figure 19 below.

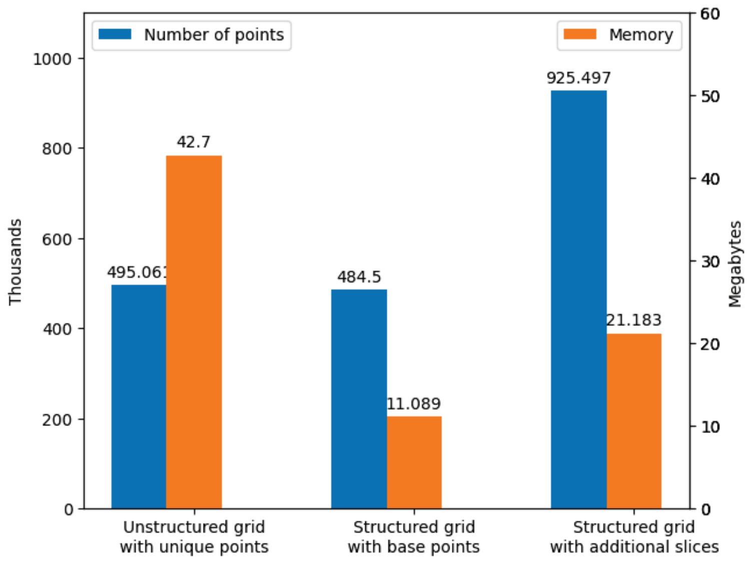

Figure 19: Configuration and memory consumption of structured and unstructured grids

4.5. Index mapping

There are few benefits of ParaView structured grid against unstructured grid. One of them is that structured grid is indexed with *i*, *j* and *k* be default. This allows us to create slices and subsets and identify probed cells. Such indexation is also a feature of GRDECL structure. However, if it is represented by unstructured grid this indexation needs to be added explicitly as scalar properties. When we use structured grid, but with additional slices, such index properties will be necessary as well. It will be required to reconstruct initial grid mapping which is distorted by added slices. There will be three 3D matrices of 4-byte integers for cells and the same for points. Exact values of base and additional memory required for indexation is plotted on figure 20 below.

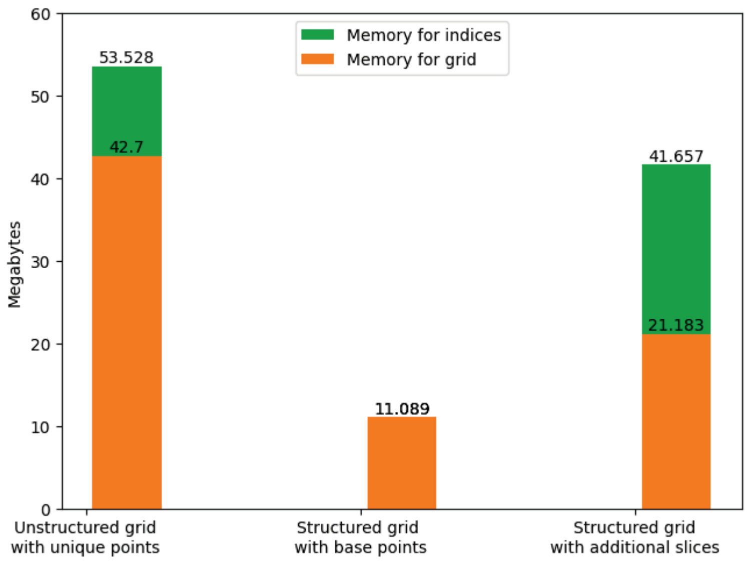

Figure 20: Memory consumption with and without additional (i, j, k)-indexation

There is another very important positive feature of structured grid. It shows in multi-processing mode of ParaView visualization.

4.6. Ghost cells

When using multiple processes and transparency, unstructured grid causes new problems. Each partition has it's own outer faces, and when subgrids processed by each thread are combined together some those subgrids faces comprise unpleasant graphical artifacts observed in the interior of the dataset as figure 21 demonstrates.

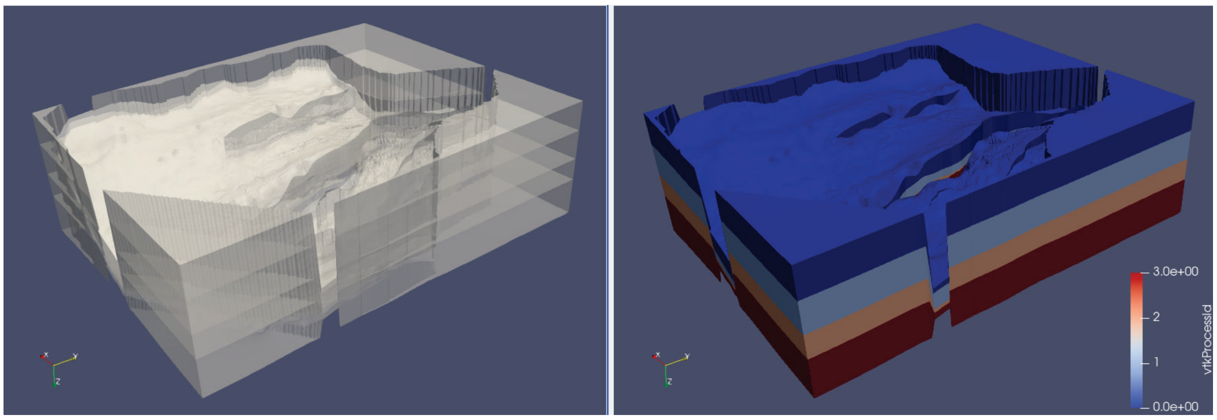

Figure 21: Multi-processing visualization artifacts on unstructured grid

To eliminate this artifacts need to define so called "ghost cells". These are additional cells that are declared invisible. Also, any common face of visible cell and ghost cell will be invisible. The estimation of ghost cells overhead depends on the partitioning strategy. If the dataset is partitioned by layers the estimation of worst case (when the number of process equals the number of layers) is that the size of connectivity array can almost triple. In contrast, structured grids are free from this problem. No ghost cells need be created, no additional memory and processing is required. Results of those visualizations are on figure 22 below, images are identical.

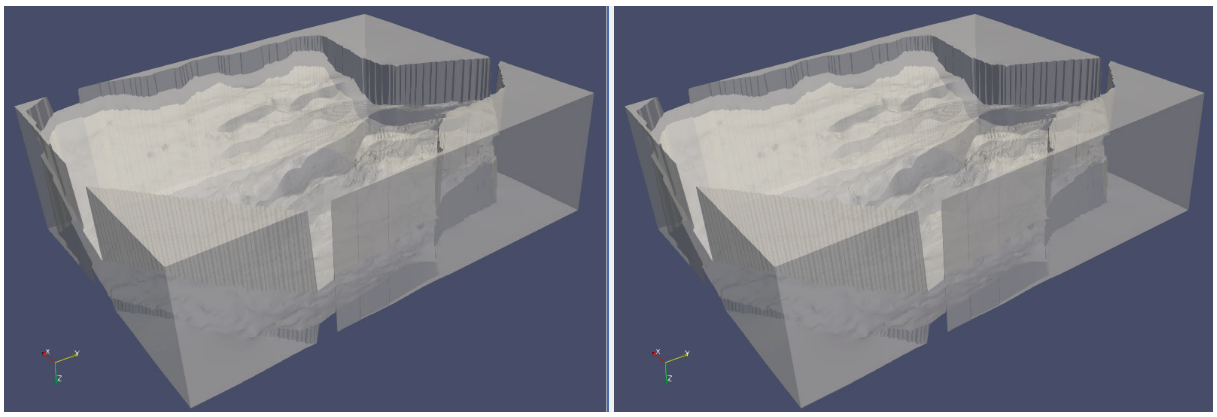

Figure 22: Unstructured grid with ghost cells (left) and structured grid without ghost cells (right)

On figure 23 we can see how much memory is required to get rid of the partitioning artifacts in case of unstructured grid. The more processes are used - the greater is overhead.

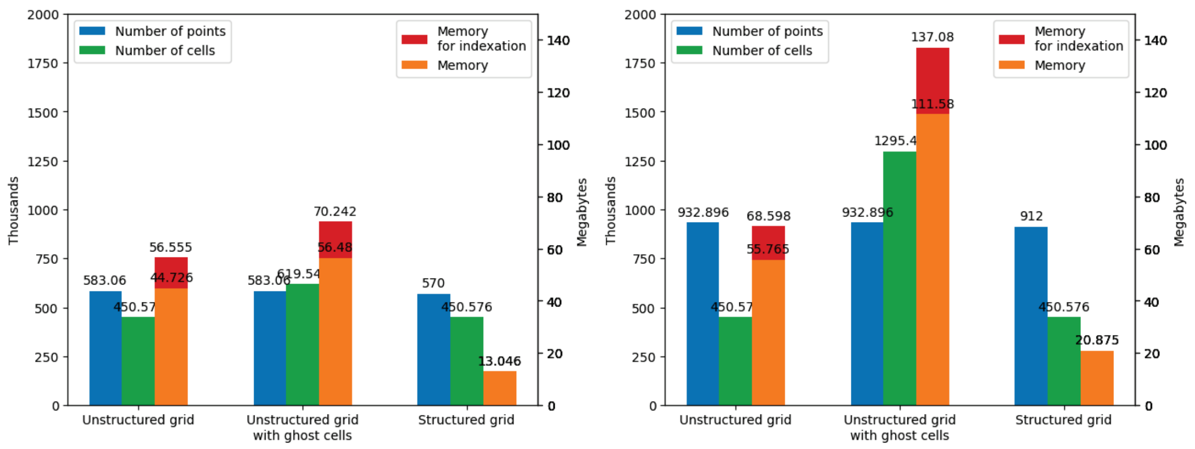

Figure 23: Grid configuration and memory consumption for 4 processes (left) and 16 processes (right)

4.7. Large scale visualization

As we mentioned before, of course memory and computational efficiency is critical when datasets are big. All the memory estimations scale well, but computational efficiency is better to check on noticeable scale. Let's review a refined case, where original dimensions were increased by factor of 4 in each direction (figure 24). This results in increasing total amount of cells by 64 up to ~29 mln. For the sake of analysis simplicity, the visualization is done using single process, may the multi-processing be a topic for a separate study.

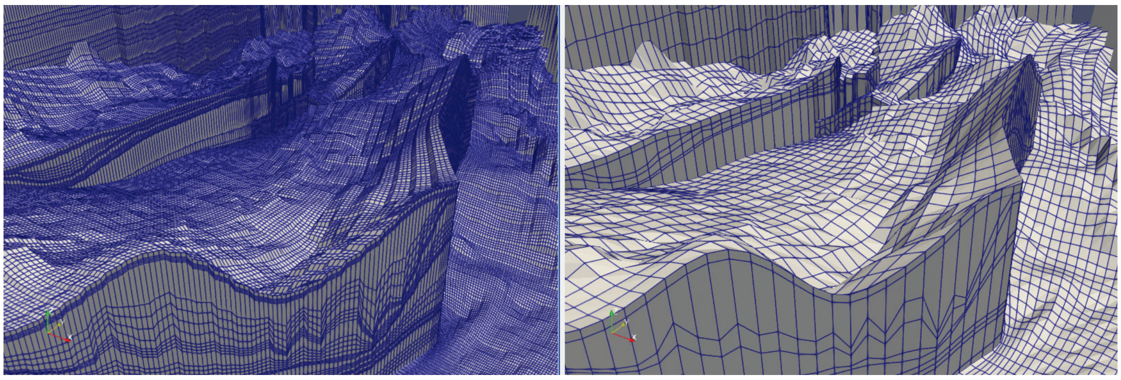

Figure 24: Refined (left) and original (right) datasets

The plot on figure 25 below shows the difference in memory consumption and computational time required for different types of grids.

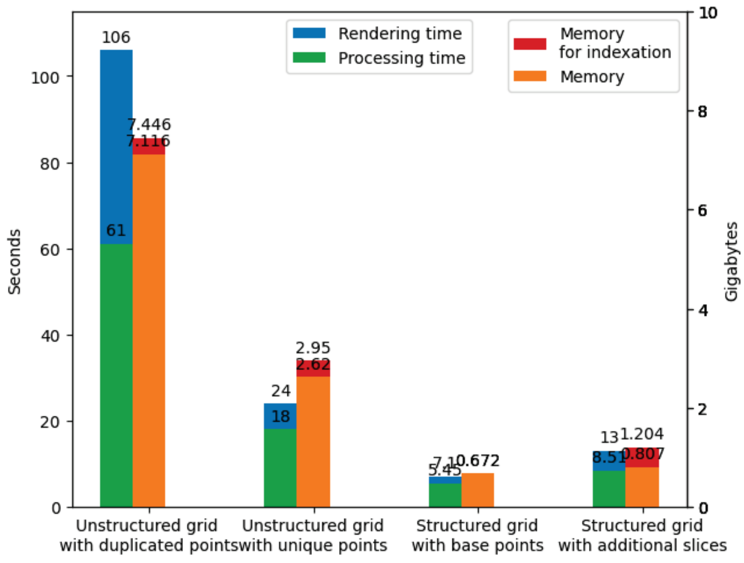

Figure 25: Processing and memory efficiency of different grids

First, we can notice that the first configuration outlines with huge resource consumption. It is an unstructured grid with duplicated points that also have rendering artifacts, which by the way cause the biggest fraction of rendering time required. However, the main observation from this plot should be the difference between the second and third series of columns. Precise visualization with unstructured grid takes 3.5 longer than draft image based on structured grid and takes 4.5 times more memory. At the same time, advanced structured grid can provide exact representation yet twice faster and saving almost 60% of memory.

5. Conclusion

In this study we have demonstrated different approaches to customized visualization of reservoir-specific format on the Johansen dataset example. These approaches showed highly diverse values of required memory and computational time. Obviously, the configuration of unstructured grid with duplicated points is the worst way of visualization and should be avoided as it is very resources hungry yet provides unpleasant image. An unstructured grid based on unique points is a universal option that provides precise and stable results. However, if some deviation from the exact representation is acceptable structured grid built on base (nodal) points can provide a high-quality draft image several times faster and with multiple memory savings. In case when dataset has no faults (geometry shifts) unstructured and structured grids will result in identical images. Sometimes, when the number of faults is not too high advanced structured grid can be considered. Additional slices will recreated the faults and precise representation can be achieved with structured grid, which will be still more resource-efficient than unstructured one. However, such grid type has very narrow applicability and will be more difficult to handle especially when additional properties are loaded because the original indexation is distorted. Resources efficiency is especially important when sizes of the datasets challenge the limits of the available hardware. In this case the choice can fall to the option that will produce any result rather than to one that will require more resources that are available.

6. Acknowledgements

I'm grateful to Marwan Charara – head of Geopetrophysical Modeling and Instrumentation department of Aramco Moscow Research Center, Denis Sabitov – senior researcher of Aramco Moscow

Research Center, and Pavel Novikov – former developer of Aramco Moscow Research Center who inspired, encouraged and assisted me in high-performance visualization development.

7. References

[1] Kitware Inc., ParaView guide, Understanding Data URL: https://docs.paraview.org/en/latest/UsersGuide/understandingData.html.
[2] The ParaView Community, ParaView/Plugin HowTo, URL: https://www.paraview.org/Wiki/ParaView/Plugin_HowTo.
The ParaView Community. Setting up a ParaView server. URL: http://www.paraview.org/Wiki/Setting_up_a_ParaView_Server.
[3] Novikov, P., Sabitov, D., Bukhanov, N., Charara, M., Cancelliere, M., Rashed, F., & Baiz, A. (2022). Efficient Visualization Methods for Large Scale Reservoir Models. Conference Proceedings, 83rd EAGE Annual Conference & Exhibition, 2022(1), 1–5. URL: https://doi.org/10.3997/2214-4609.202210139/
[4] Kitware Inc. VTK User's Guide. paperback edition, 3 2010. ISBN 978-1930934238.
[5] OpenGoSim Limited, GRDECL grids. URL: https://docs.opengosim.com/manual/input_deck/grid/grdecl.
[6] Open Data Commons Open Database License (ODbL) v1.0. URL: http://opendatacommons.org/licenses/odbl/1.0/.
[7] Database Contents License (DbCL) v1.0. URL: http://opendatacommons.org/licenses/dbcl/1.0/.
[8] 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. https://doi.org/10.1007/s10596-009-9153-y.
[9] Kitware Inc., ParaView python online documentation. URL: https://kitware.github.io/paraview-docs/latest/python/paraview.simple.html.