Using ViZiR 4 to analyze the 4th AIAA CFD High Lift Prediction Workshop Simulations

Matthieu Maunoury, Rémi Feuillet, Adrien Loseille

Using ViZiR 4 to analyze the 4th AIAA CFD High Lift Prediction Workshop Simulations

Matthieu Maunoury*, Rémi Feuillet† and Adrien Loseille ‡
GAMMA Team, Inria Saclay Ile-de-France, Palaiseau, France

ViZiR 4 is an interactive visualization software that uses OpenGL 4 graphic pipeline. It can be used to analyze large meshes with possibly solutions and in this paper the focus is made with results from the 4th AIAA CFD High Lift Prediction Workshop. To perform such simulations, it is necessary to have fast, precise and interactive tools to analyze, check and validate the numerical results obtained. Fast I/O and rendering is important to inspect results and comparisons with ParaView show that ParaView is much slower (ratio between 5 and 50). Many post-processing tools, such as picking, hiding surfaces by reference, isolines rendering and clip planes generation, allow to quickly investigate meshes and solutions. Pixel exact rendering permits to have a precise preview of solutions and tessellations on GPU are used to render high-order and curved elements. Some scripting tools allow to generate quickly images and go over sequences of several meshes that is useful when mesh adaptation is involved. All along the paper, results from the workshop are shown to illustrate the capabilities of ViZiR 4.

I. Introduction

The High-Lift Prediction Workshop (HLPW) series aim at assessing the state-of-the-art in computational fluid dynamics (CFD) methods for simulation of high lift-configurations and to provide a forum for exchange of ideas and practices related to this class of problems. Four goals were defined for the 4th AIAA CFD High Lift Prediction Workshop. First, assess the numerical prediction capability (meshing, numerics, turbulence modeling, high-performance computing requirements, etc.) of current-generation CFD technology/codes for swept, medium-to-high-aspect ratio wings for landing/take-off (high-lift) configurations. Then, develop practical modeling guidelines for CFD prediction of high-lift flow fields. Thirdly, determine the elements of high-lift flow physics that are critical for modeling to enable the development of more accurate prediction methods and tools. Finally, enhance CFD prediction capability for practical high-lift aerodynamic design and optimization.

To accomplish such simulations, pre- and post-processing tools are necessary. The visualization of meshes and solutions is a keystone of the numerical simulations process as it allows to check the validity and quality of the meshes, display the computed numerical solutions and analyze the potential problems on meshes and solutions.

To analyze linear meshes (elements of degree 1) and linear solutions (constant or of degree 1), many visualization softwares (e.g. ParaView [1], Visit [2], Tecplot [3], Gmsh [4]) exist and many interesting plugins and tools have been implemented to help the analyses. However, a bottleneck is the time to open large files and render the results. In Section II, comparison are done between `ParaView` and `ViZiR 4` and the ratios are huge, between 5 and 50. For large meshes, interactivity might be missing as well as tools to manipulate efficiently these meshes. Again, in Section II, comparisons are done in term of time to render isolines or generate cut plane and the difference of time needed is very important. Furthermore, in the context of the workshop, anisotropic meshes might be generated and `ViZiR 4` is able to handle this kind of meshes.

The post-processing of high order meshes and solutions is still a current and complex challenge. Indeed, most of the standard visualization softwares are based on linear primitives as imposed by the commonly-used baseline graphic pipeline. To bypass these limitations, a low-order remeshing strategy exists. Its principle is to define a sub-mesh and affine representations which approximate the solution. To this end, a visualization error, corresponding to the gap between the numerical solution and its representation, is introduced and controlled [4–8] and the rendering obtained is, as a consequence, inaccurate. Some other approaches are based on raycasting [9–11]. For each pixel, rays are cast to determine the color for this pixel. However, this solution has limited interactive capabilities [9].

*Research Engineer, matthieu.maunoury@inria.fr
†Research Engineer, remi.feuillet@inria.fr
‡Research Scientist, adrien.loseille@inria.fr, AIAA Member.

To address some of the previous issues, we are developing `ViZiR 4` [12, 13], an interactive and reliable high order meshes and solutions visualization platform based on OpenGL Shading Language (GLSL). The goal of this paper is to show how we used `ViZiR 4` to analyze all the meshes and solutions we generated for the 4th AIAA CFD High Lift Prediction Workshop in the context of adaptive and adapted meshes (highly anisotropic). The adapted meshes were generated by the remesher AMG/feflo.a [14, 15] and the solver is Wolf [16], both developed at Inria.

The paper is outlined as follows. Section II gives an overview of `ViZiR 4`. Section III deals with the visualization of high-order meshes. Section IV is devoted to scripting tools.

II. Overview of `ViZiR 4`

In this section, the main features of `ViZiR 4` are presented. Some comparisons in term of CPU time are done with `ParaView`. All the results collected in this paper have been generated with the same laptop: a MacBook Pro with details given in Table 1.

Hardware	Details
CPU	Intel Core i7 2.6 GHz 6-core
GPU	AMD Radeon Pro Vega 20 4 Gb
Mem	32 Gb of RAM 2400 MHz DDR4
OS	Mac

Table 1 Hardware used for testing.

A. Presentation of the OpenGL 4 graphic pipeline

The OpenGL 4 rendering pipeline can be customized with up to five different shader stages (see Fig. 1). These shaders are GLSL source code files that replace parts of the OpenGL pipeline. In general, a shader receives its input via programmer-defined input variables, and the data for those variables come either from the main OpenGL application or previous pipeline stages (other stages). Data can also be provided to any shader using uniform variables or textures [17]. More details on OpenGL 4 and in particular OpenGL Shading Language (GLSL) can be found in [17, 18].

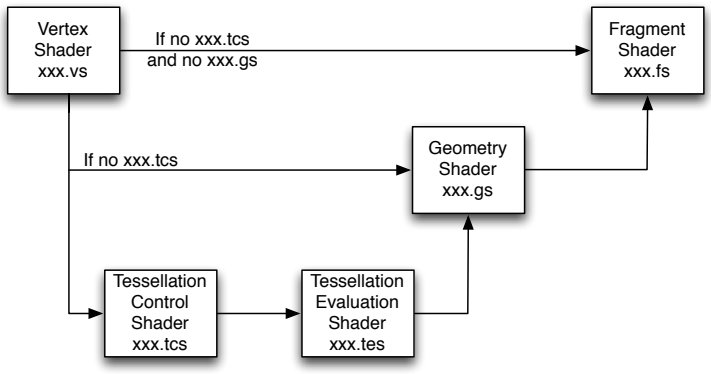


Fig. 1 Shaders used for the OpenGL graphic pipeline.

Two shaders are enough to define a graphic pipeline, the vertex shader and the fragment shader. The vertex shader handles the vertices. The data corresponding to the vertices positions are tranformed into clip coordinates. The fragment shader determines the color for each pixel. Many parameters can affect the color like a shading, a solution, a isoline, or a wireframe rendering. For the storage of raw data (like high-order solutions), textures are used.

Besides these two shaders, a geometry shader can be added to govern the processing of primitives. It allows to create new geometries on the fly. With this in mind, it can be preceded by the two tessellation shaders: the tessellation

control shader and the tessellation evaluation shader. They are used to control the tessellation of the primitives, in other words, in how many sub-elements the elements should be divided.

B. Fast I/O and rendering

A key to have an efficient visualization is to be able to quickly open mesh and solution files. Input and output are handled by the `libMeshb`* library. The files follow the GMF format provided by this library. For instance, the mesh of Lucy (see Fig. 2) with more than 14 millions vertices and 28 millions triangles (642 Mb) is opened in less than 1.5 seconds.

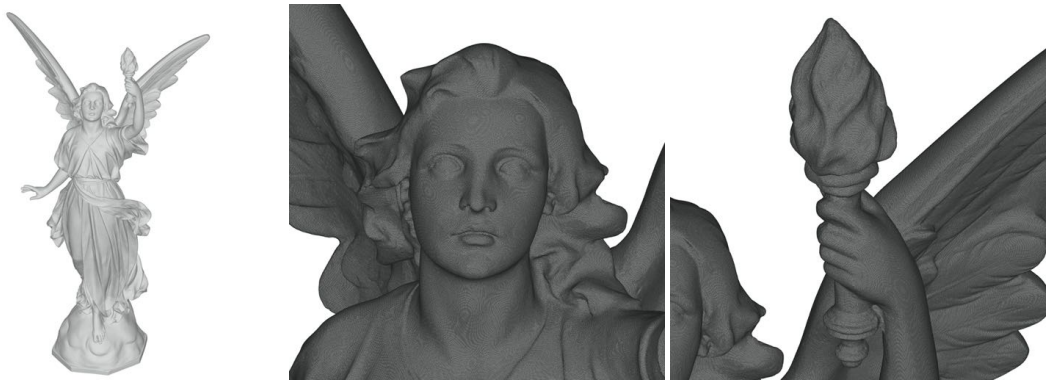


Fig. 2 Rendering of a large mesh of 14M vertices and 28M triangles in 7.5 seconds (total time) on a laptop.

Fig. 3 shows the rendering different adapted meshes. Examples of solution rendering will be shown in the following. A comparison between `ParaView` and `ViZiR 4` in terms of CPU time for a sequence of meshes of increasing complexity is done in Table 2. The time is the total time to display the mesh with its solution including mesh and solution files opening. The ratio is huge, between 5 and 60, and is explained by the fact that the time to open the mesh file in `ParaView` is long and because a surface reconstruction (when there are volume elements) is done in `ParaView` and this step is very expansive. The difference in the memory footprint is also noticeable. For the 10240K mesh, 8.0 Gb for `ParaView` versus 4.0 Gb for `ViZiR 4` and for the 20480K mesh, 42.16 Gb for `ParaView` versus 7.79 Gb for `ViZiR 4`.

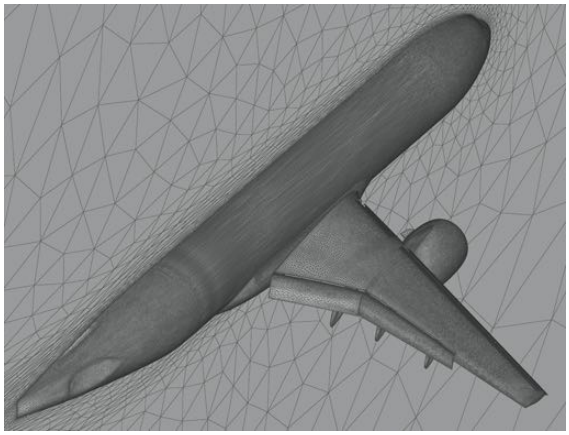
Case	# vertices	# triangles	# tetrahedra	`ParaView` (s)	`ViZiR 4` (s)	Ratio
20480K	3 084 324	6 166 689	0	11.2	2.25	5.0
640K	1 342 310	446 158	7 370 829	14.9	0.76	19.6
1280K	2 699 131	802 316	14 968 807	32.8	1.20	27.3
2560K	5 415 482	1 285 472	30 541 700	71.4	1.80	39.7
5120K	10 784 310	2 080 672	61 563 158	155.1	3.12	49.7
10240K	21 695 268	3 614 018	124 736 423	333.6	7.78	42.9
20480K	43 380 172	6 275 672	250 898 971	980.2	16.05	61.1

Table 2 Comparison of total rendering CPU time (s) including files (mesh and solution) opening.

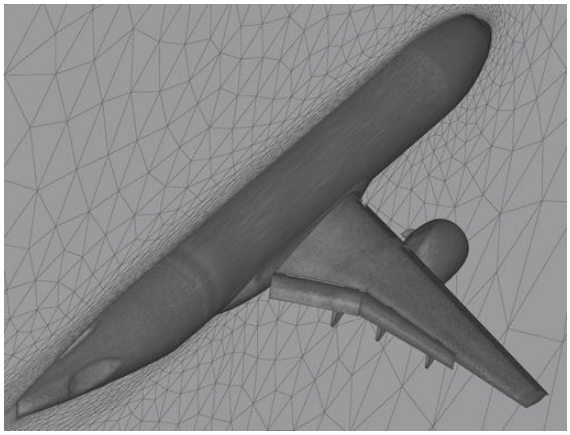
C. Pixel exact rendering on flat elements

OpenGL 4 graphic pipeline flexibility allows to compute on the fly the solution. It leads to a pixel exact rendering when flat elements (of degree one) are considered regardless of the degree of the solution. This recent language (GLSL) enables `ViZiR 4` to certify a faithful and interactive depiction. High order solutions are natively handled by `ViZiR 4`

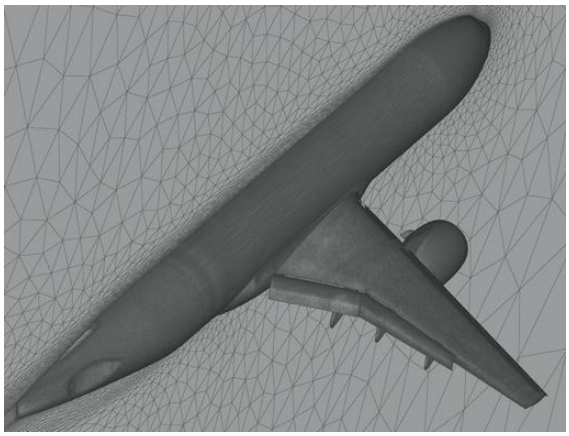
*https://github.com/LoicMarechal/libMeshb



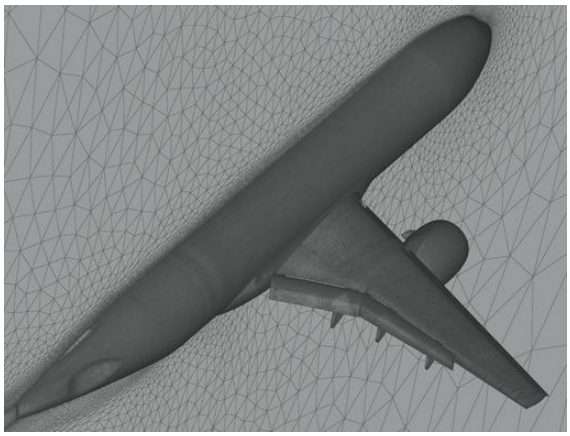
(a) 640K



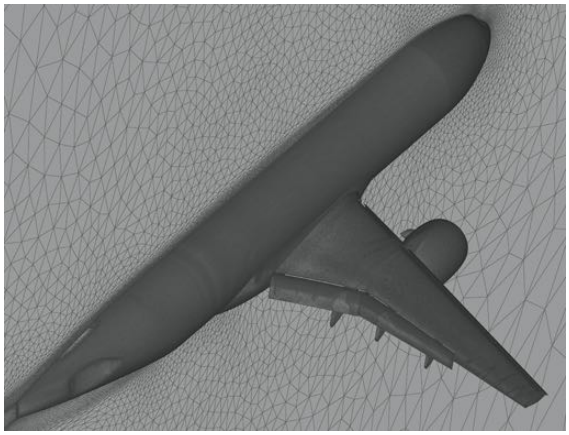
(b) 1280K



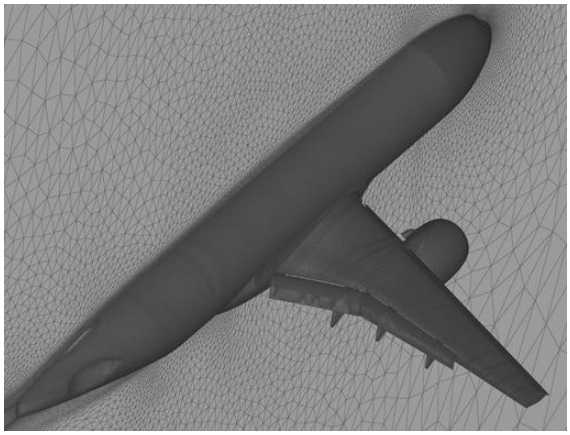
(c) 2560K



(d) 5120K



(e) 10240K



(f) 20480K

Fig. 3 Different adapted meshes used for comparisons in Table 2.

on surface and volume (tetrahedra, pyramids, prisms, hexahedra) meshes which can naturally be hybrid. Fig. 4 shows an example of pixel exact rendering of high-order solution.

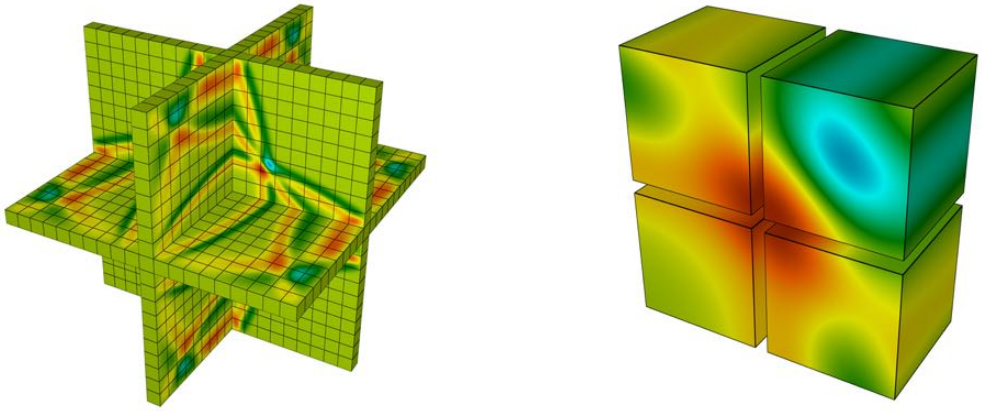


Fig. 4 High-order (degree Q^6) solution of a wave propagation problem. Right: zoom of the solution on 4 hexahedra. Courtesy of Sébastien Impériale (Inria) [19].

D. Tessellation on GPU for high-order elements

When more complex geometries are considered, curved elements perform a better approximation of the geometry. In this case, tessellation shaders occur in OpenGL pipeline (see [12, 13] for more details on the shaders pipeline) to tessellate all elements directly on the GPU. For solutions on such curved elements, almost pixel exact rendering is ensured. An example of curved mesh is shown on Fig. 5 and an example of tessellation is shown on Fig. 6.

E. Post-processing tools and interactivity

Many post-processing tools are available to make analysis results possible. Some of them are presented in this section. Such interactivity is fundamental to develop and validate new algorithms.

1. Picking and hiding surfaces by reference

Any element or vertex can be picked to get information. Some information is directly printed on the window and others are printed on the terminal. For instance, for a vertex we get its number, its coordinates and the values of the solutions if any. For an element, we get its type, its number, the list of its indexes and we can ask additional information such as the value of the angles, the solution, the jacobian. An example is shown in Fig. 7. To inspect meshes, it is interesting to hide some elements. After an element is picked, it is possible to hide all elements having the same reference id (corresponding typically to a patch). An example is shown in Fig. 7 where the green surface is hidden in right figure to show the elements behind.

2. Isolines rendering

To enhance solution rendering, isolines can be displayed as in Fig. 8. The rendering of isolines in `ViZiR 4` is instant while it can take a while in `ParaView` as shown in Table 3. In `ParaView`, the contour filter is used with 35 values and 35 isovalues are also displayed in `ViZiR 4`. The meshes used are only composed of triangles and the solution is affine on each element (C_f).

3. Clip planes

It is often difficult to navigate in 3 dimensional meshes. For this reason, it is convenient to use clip planes where all volume elements belonging to a plane are displayed (see Fig. 9). A comparison of CPU time to generate such clipping in `ParaView` and in `ViZiR 4` is done in Table 4 and the difference is important with ratios between 6 and 9.

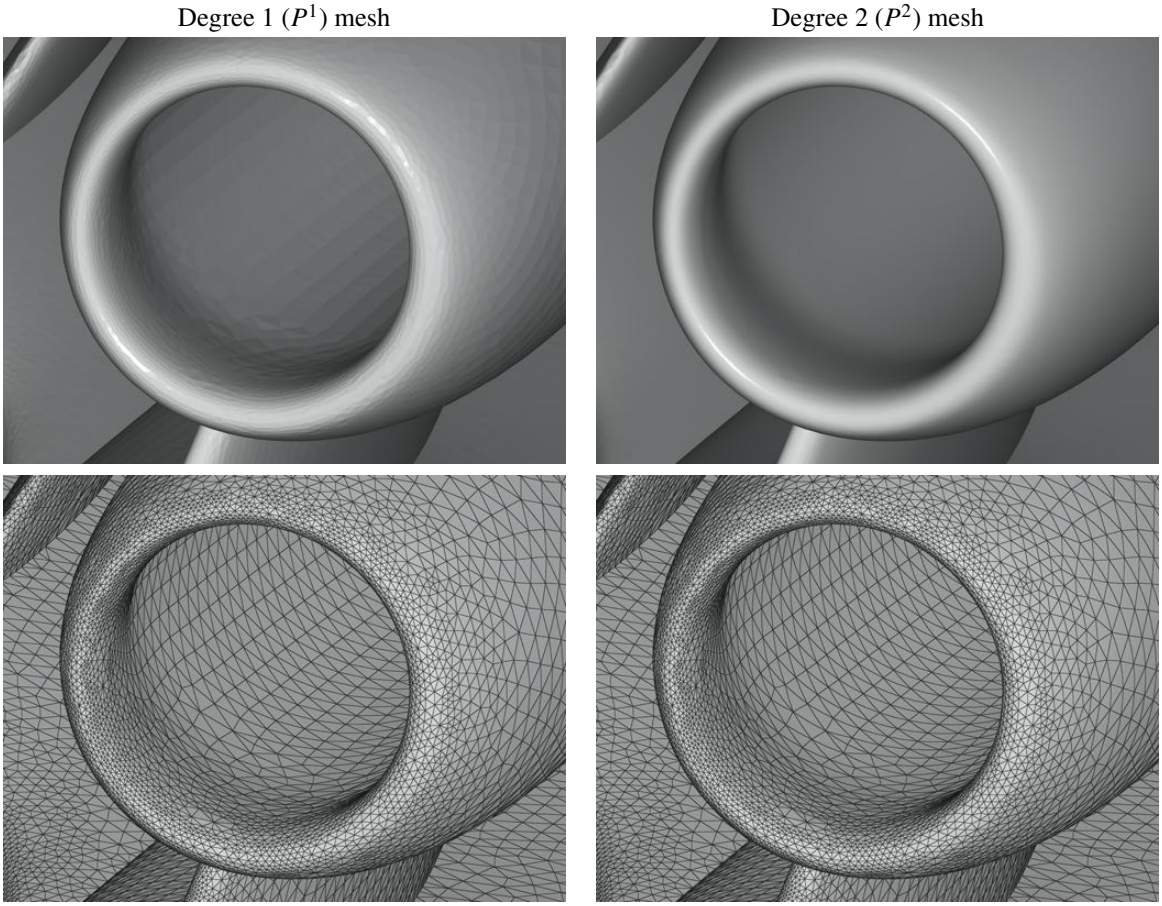


Fig. 5 Comparison of rendering of meshes of degree 1 (left) and 2 (right) for the same number of elements.

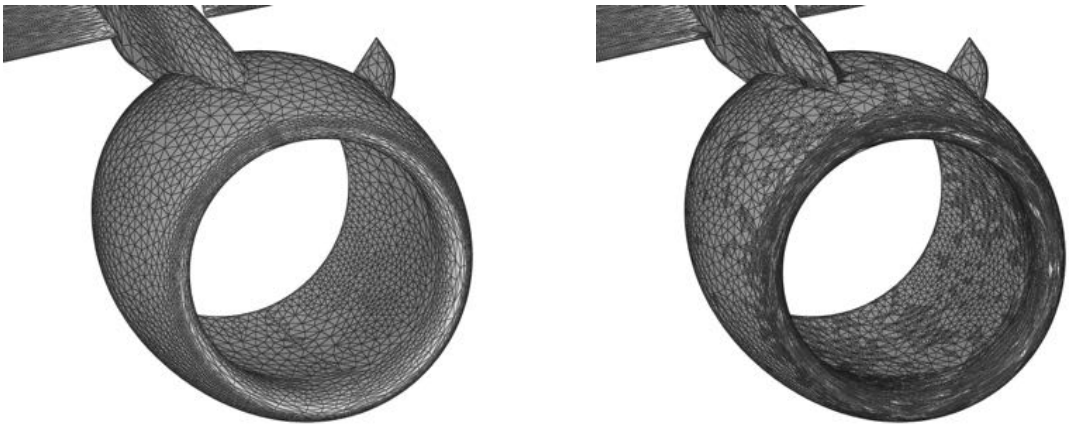


Fig. 6 Rendering of high-order mesh (left) and its tessellation constructed on the fly by the GPU (right).

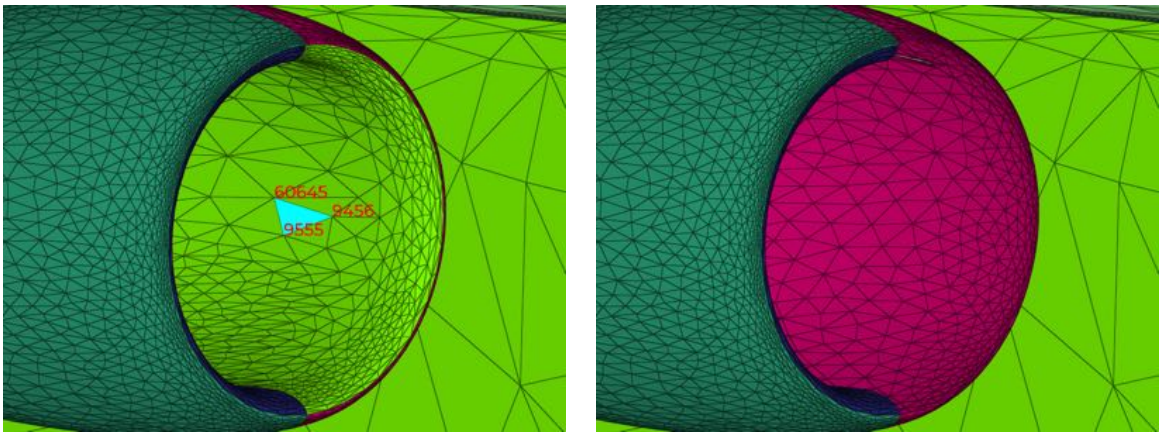


Fig. 7 Example of picking (left) and hiding surfaces by reference (right).

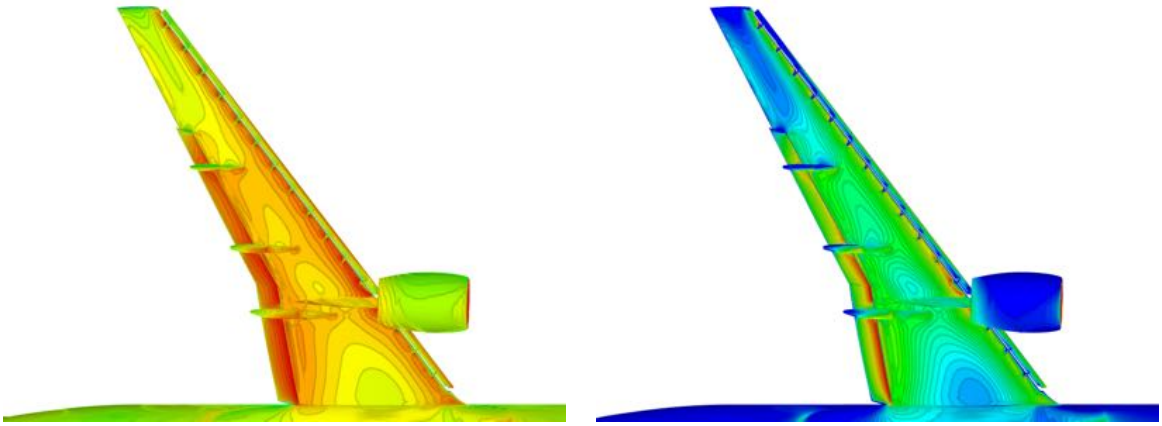


Fig. 8 Examples of isolines rendering.

Case	# vertices	# triangles	`ParaView` (s)	`ViZiR 4` (s)
640K	217 000	433 653	7.2	0.01
1280K	392 257	783 947	13.9	0.01
2560K	628 223	1 255 604	24.6	0.01
5120K	1 018 135	2 035 092	39.2	0.01
10240K	1 772 712	3 543 955	63.1	0.01
20480K	3 084 324	6 166 689	109.3	0.01

Table 3 Comparison of CPU time (s) to render isolines (contours) for different meshes with only triangles.

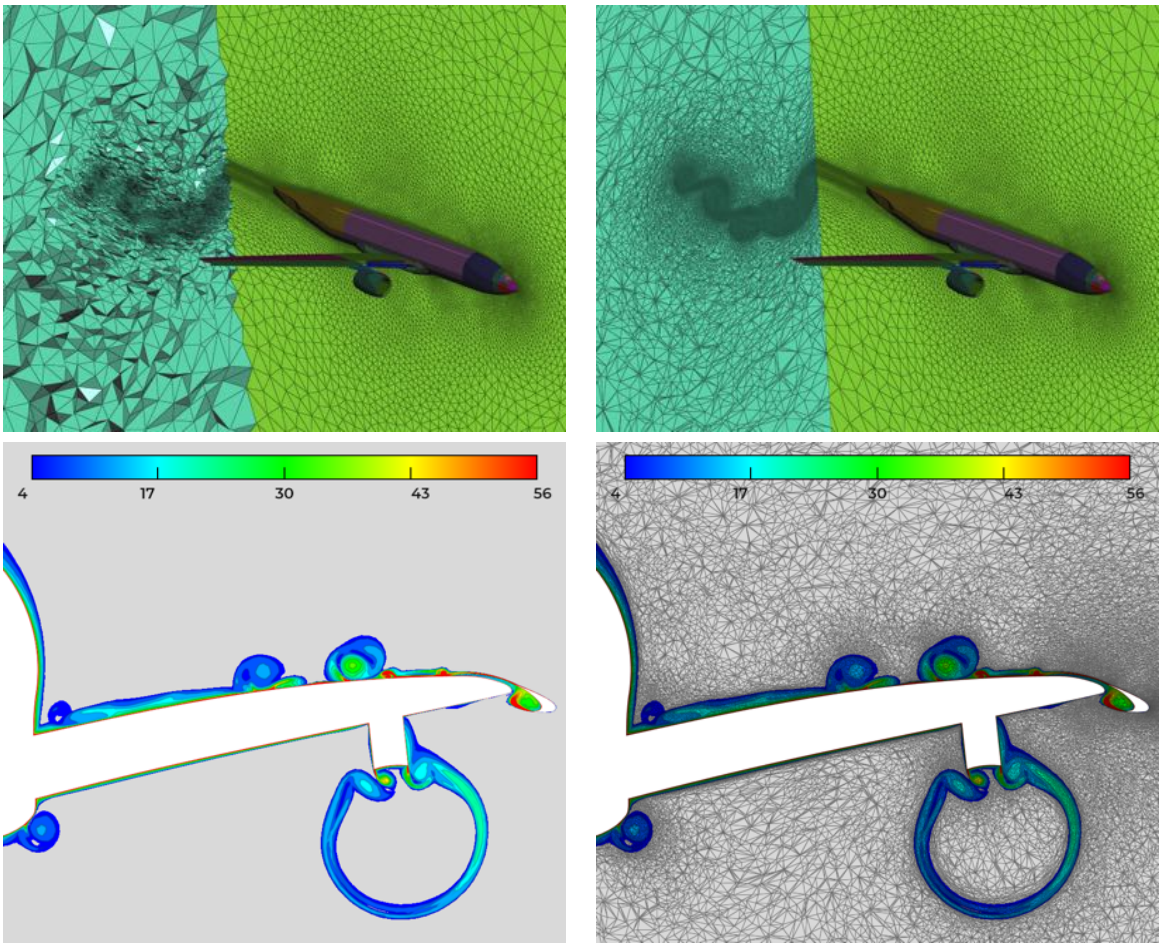


Fig. 9 Examples of cut planes. Top: clipping and capping. Bottom: vorticity in a slice.

Case	# vertices	# triangles	# tetrahedra	`ParaView` (s)	`ViZiR 4` (s)	Ratio
640K	1 342 310	446 158	7 370 829	1.6	0.18	8.9
1280K	2 699 131	802 316	14 968 807	3.1	0.48	6.5
2560K	5 415 482	1 285 472	30 541 700	6.4	0.93	6.9
5120K	10 784 310	2 080 672	61 563 158	13.9	1.99	7.0
10240K	21 695 268	3 614 018	124 736 423	28.6	4.80	6.0
20480K	43 380 172	6 275 672	250 898 971	91.3	10.04	9.1

Table 4 Comparison of CPU time (s) to generate cut plane (clip).

III. High-Order meshes

The definition and construction of high-order meshes is detailed in many articles such as [20–24]. In this section, we recall some basic concepts and notations and we focus on the jacobian, the main feature to analyze the quality of meshes. In particular, elements with negative jacobian are non-valid. To properly define the geometry and these functions, a reference space (parameter space) $\widehat{K}$ is defined. The element K, also called physical element, is thus the image of $\widehat{K}$ via a mapping, denoted F_K (see Fig. 10). More specifically, for each point M of K, there is a point $\widehat{M}$ of $\widehat{K}$ such that $M = F_K(\widehat{M})$. In particular, the position of a point M inside K is defined by

$$M = \sum_{i=1}^{n} \phi_i^n(\widehat{M}) A_i, \tag{1}$$

where n is the number of nodes, $A_i = F_K(\widehat{A}_i)$ with $\widehat{A}_i$ the nodes of the reference element which map to A_i the nodes of the physical element, and ϕ_i^n are the Lagrange polynomial functions defined such that:

$$\phi_i^n(\widehat{A}_j) = \delta_{ij} \quad \text{and} \quad \sum_{i=1}^{n} \phi_i^n = 1.$$

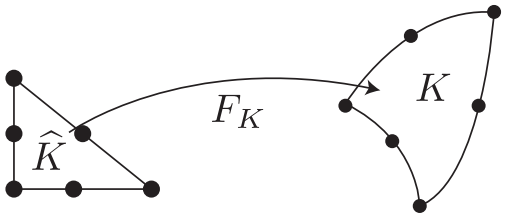


Fig. 10 Mapping F_K from $\widehat{K}$ to K.

Now, we will assume that the shape functions ϕ_i^n are polynomial functions of degree d of the parameters space and therefore define high-order finite elements. In this case, the number of nodes n is a consequence of the degree of the element.

A. The Jacobian

The Jacobian is a good tool to spot invalid elements and to evaluate the quality of a mesh. The Jacobian $\mathcal{J}$ is defined as $\mathcal{J} = \left|\frac{\partial F_K}{\partial \hat{x}} \frac{\partial F_K}{\partial \hat{y}}\right|$ in 2D or $\mathcal{J} = \left|\frac{\partial F_K}{\partial \hat{x}} \frac{\partial F_K}{\partial \hat{y}} \frac{\partial F_K}{\partial \hat{z}}\right|$ in 3D where $\left|..\right|$ in 2D and $\left|...\right|$ in 3D denotes the determinant. A convenient way to handle high-order meshes is to write F_K into Bézier form [20, 21] using Bernstein polynomials. For more details on the Jacobian, its definition and computation see [20, 24–28].

The notable thing is that the Jacobian is simply a polynomial function which can be easily visualized with `ViZiR 4` as a high-order solution. However, the degree of the jacobian is greater than the degree of the element and this rise is not linear (see Tables 5 and 6). The impact of the increase of this degree is the number of control coefficients and the number of terms to compute which increase drastically with the degree of the element (see Tables 5 and 6). Quadrilaterals and hexahedra are obtained by tensorization while the pyramids are constructed as degenerated hexahedra [20, 27]. Finally, prisms are defined as tensor product of a triangle and an edge [20].

Fig. 11 (left) shows an example of rendering of the jacobian polynomial. In this example, all the elements are valid as all their values are positives. Nevertheless, it is difficult to see where the worst elements are. For this reason, it might be more appropriate to display the minimal jacobian as in Fig. 11 (right), that is a constant value for each element corresponding to the minimal polynomial on this element. In this way, it is easier to identify the worst elements as we see more easily the contrasts. Another advantage of the minimal rendering is that the storage is smaller and the number of frames per seconds is better as a constant is rendered instead of a polynomial. Note that the maximum of the palette changes but not the minimum.

B. Filters

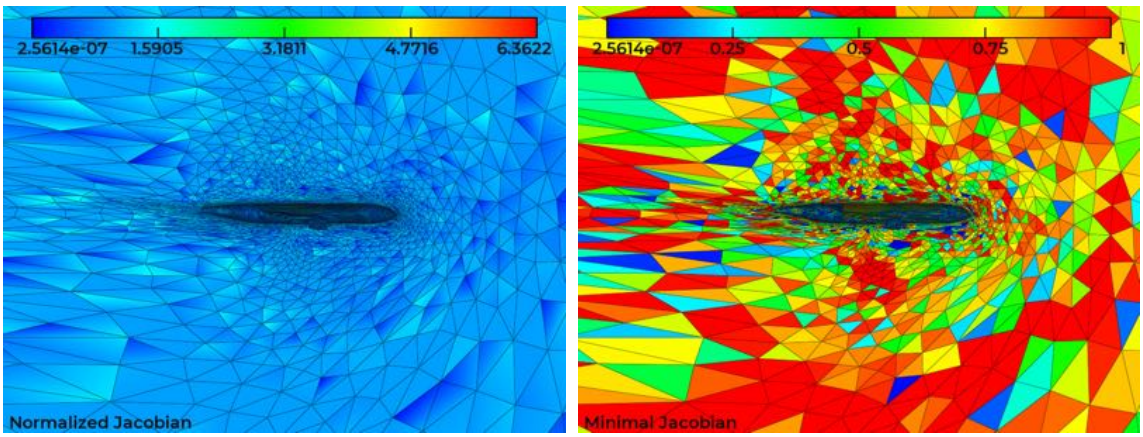
To facilitate the analysis of the Jacobian and to isolate some elements, a filtering tool exists. According to a critierion, for instance the minimal jacobian or an element quality, all elements in a given range of values are filtered and displayed

	Degree q of $\mathcal{J}$	Number of coefficients
Triangle	$2\,(d-1)$	$(q+1)\,(q+2)\,/2$
Quadrilateral	$2d-1$	$(q+1)^2$
Tetrahedron	$3\,(d-1)$	$(q+1)\,(q+2)\,(q+3)\,/6$
Prism	$3d-1$	$(q+1)\,(q+1)\,(q+2)\,/2$
Pyramid	$3d-1$	$(q+1)^3$
Hexahedron	$3d-1$	$(q+1)^3$

Table 5 Degree q effectively displayed and number of control coefficients of the polynomial Jacobian $\mathcal{J}$ for an element of degree d. Note that pyramid is defined as degenerated hexadron and prism as a tensor product of a triangle and an edge.

	Nodes	Degree $\mathcal{J}$	Control coeff.	Terms to compute
Tetrahedron P^1	4	1	1	1
Tetrahedron P^2	10	3	20	64
Tetrahedron P^3	20	6	84	1 000
Tetrahedron P^4	35	9	220	8 000
Hexahedron Q^1	8	2	27	64
Hexahedron Q^2	27	5	216	5832
Hexahedron Q^3	65	8	729	110 592
Hexahedron Q^4	125	11	1 728	1 000 000

Table 6 For each element, number of nodes, degree q of Jacobian $\mathcal{J}$ and numbers of coefficients and terms (i.e. determinants) to compute the polynomial $\mathcal{J}$ for an element of degree d.



(a) Exact jacobian (b) Minimal jacobian

Fig. 11 Example of jacobian rendering: exact (left) and minimal (right).

in a different color than others In the case of the minimal jacobian, filter all negative values permit to display all invalid elements (see Fig. 12).

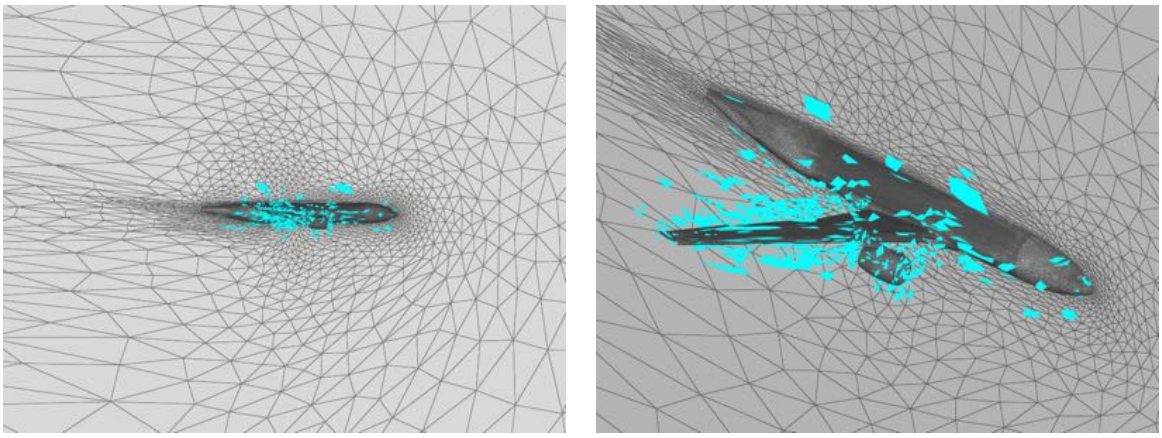


Fig. 12 Use of filters: all elements in light blue appear as they belong to the range of the filter (for a given criterion).

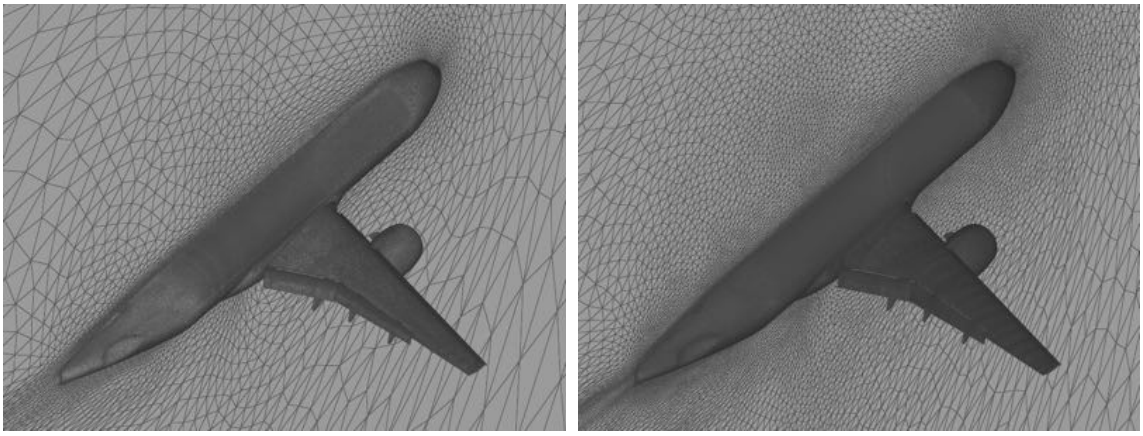
IV. Scripting tools

If `ViZiR 4` was initially intended for interactive rendering, new features allow to generate renderings among a large set of data.

A. Data files: save and load rendering options

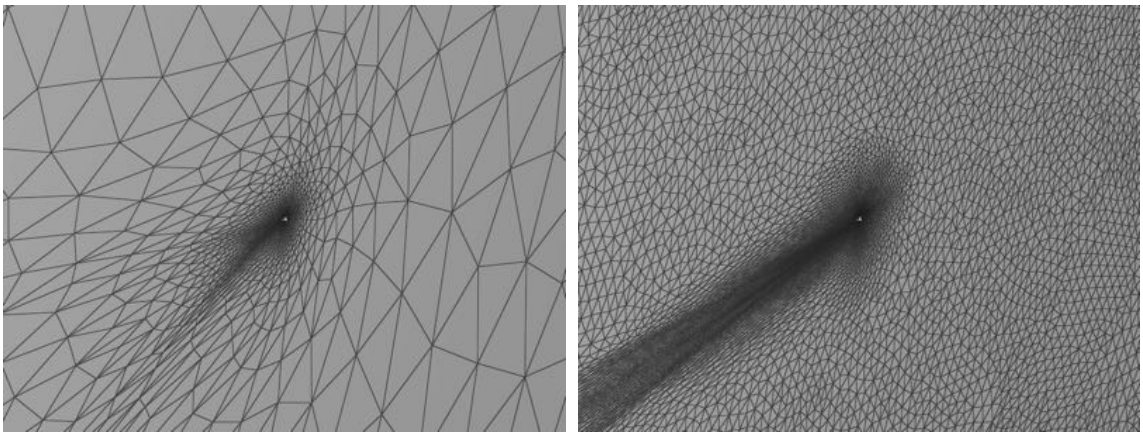
It is possible to save rendering options and to load them in `ViZiR 4`. For this purpose, data files are used. These files contain information on the view (center of the view, rotation, translation...), the plane equation, the mesh, the solution, the isolines, the level of tesselation for high-order elements, the lines' thickness, the window size and so on. Thanks to these data files, it is very easy to compare different meshes and solutions and to make images.

Fig. 13 and 14 show an example of comparison of two meshes with the same data file, the same view and rendering options are used in both images.



(a) 640K mesh (b) 10240K mesh

Fig. 13 Comparison of two meshes with the same data file.



(a) 640K mesh (b) 10240K mesh

Fig. 14 Comparison of two meshes with the same data file (far view).

B. Movie mode: generate images from several meshes and possibly solutions

The movie mode consists of generating several images for several meshes and possibly solutions. A unique data file can be given to have the same view and rendering options for all the pictures. A "movie" file contains for each picture (each line) the name of the mesh file, possibly the name of the solution file and possibly the name of the output image. An example of such file is shown in Listing 1 and the result in Fig. 15.

```
adap_640K/file.meshb     adap_640K/file.Cf.solb     movie_640K.jpg
adap_1280K/file.meshb    adap_1280K/file.Cf.solb    movie_1280K.jpg
adap_2560K/file.meshb    adap_2560K/file.Cf.solb    movie_2560K.jpg
adap_5120K/file.meshb    adap_5120K/file.Cf.solb    movie_5120K.jpg
adap_10240K/file.meshb   adap_10240K/file.Cf.solb   movie_10240K.jpg
adap_20480K/file.meshb   adap_20480K/file.Cf.solb   movie_20480K.jpg
```

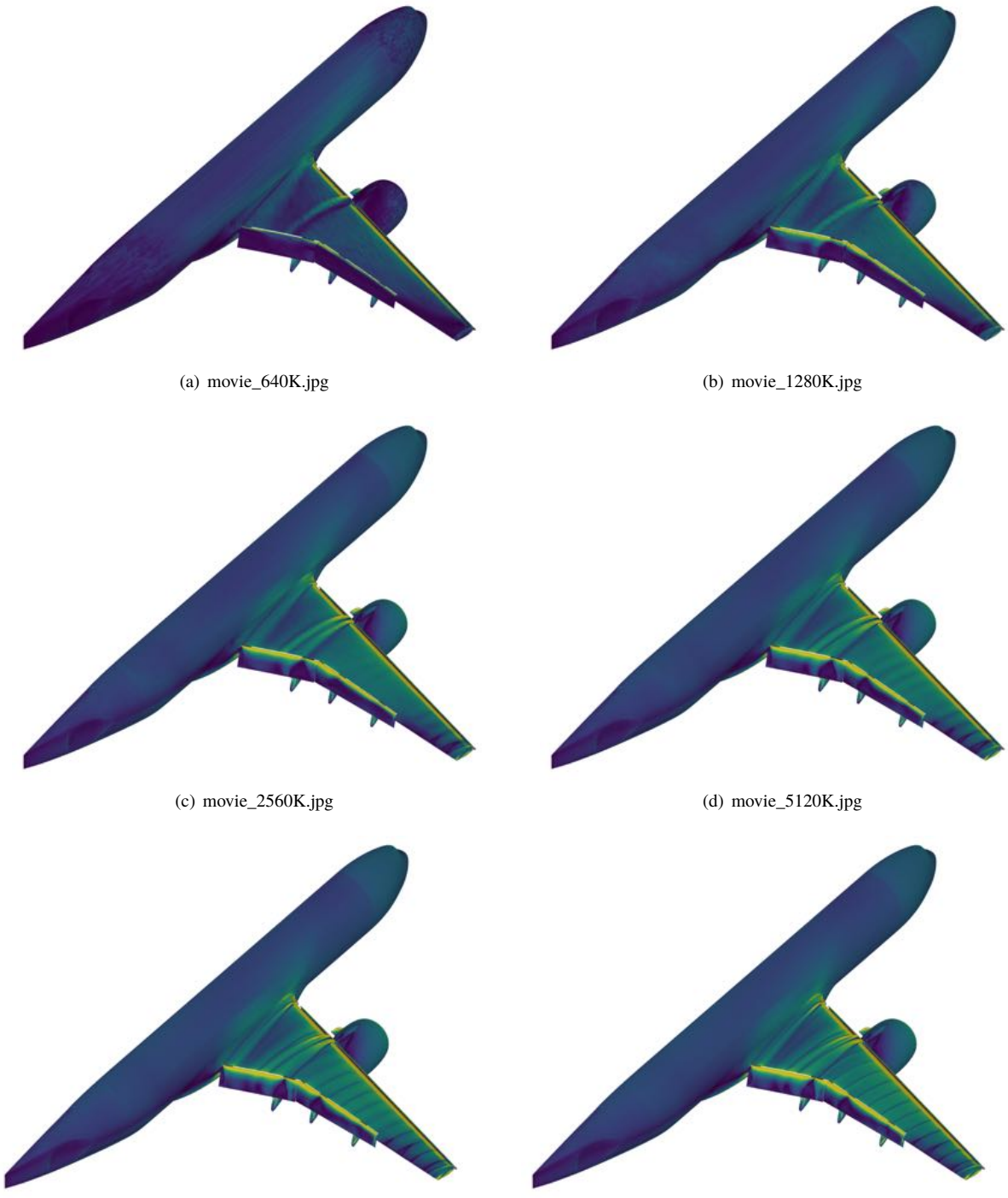
Listing 1 Example of file `vizir.movie`

C. Sequence mode: generate images from several data files

The sequence mode is used to generate several images for a single mesh and possibly solution file but with different data files. The sequence file contains the data names and possibly the name of the image outputs. Fig. 16 shows as example of such sequence.

D. `Pyviz 4`: easy generation of data and sequence files in python

`Pyviz 4` offers functions to easily create scripts in python to generate data and sequence files. The idea is to start from an existing data file, modify keywords and generate new data files. Furthermore, sequences files can be generated. Listing 2 shows an example of file. In this example, a data file is read, two new data files are created as well as a sequence file containing these three data files names. Then, the sequence mode can be launched to generate the three images shown in Fig 17. There are 3 types of keywords: boolean keywords, admissible keywords and admissible multiple lines keywords. The boolean keywords can be activated/deactivated with `enable/disable`. The admissible keywords can be modified with `set` and an appropriate number of parameters.



(a) movie_640K.jpg

(b) movie_1280K.jpg

(c) movie_2560K.jpg

(d) movie_5120K.jpg

(e) movie_10240K.jpg

(f) movie_20480K.jpg

Fig. 15 Friction Coefficient (C_f) solutions for different adapted meshes using the same data file and generated with the movie mode.

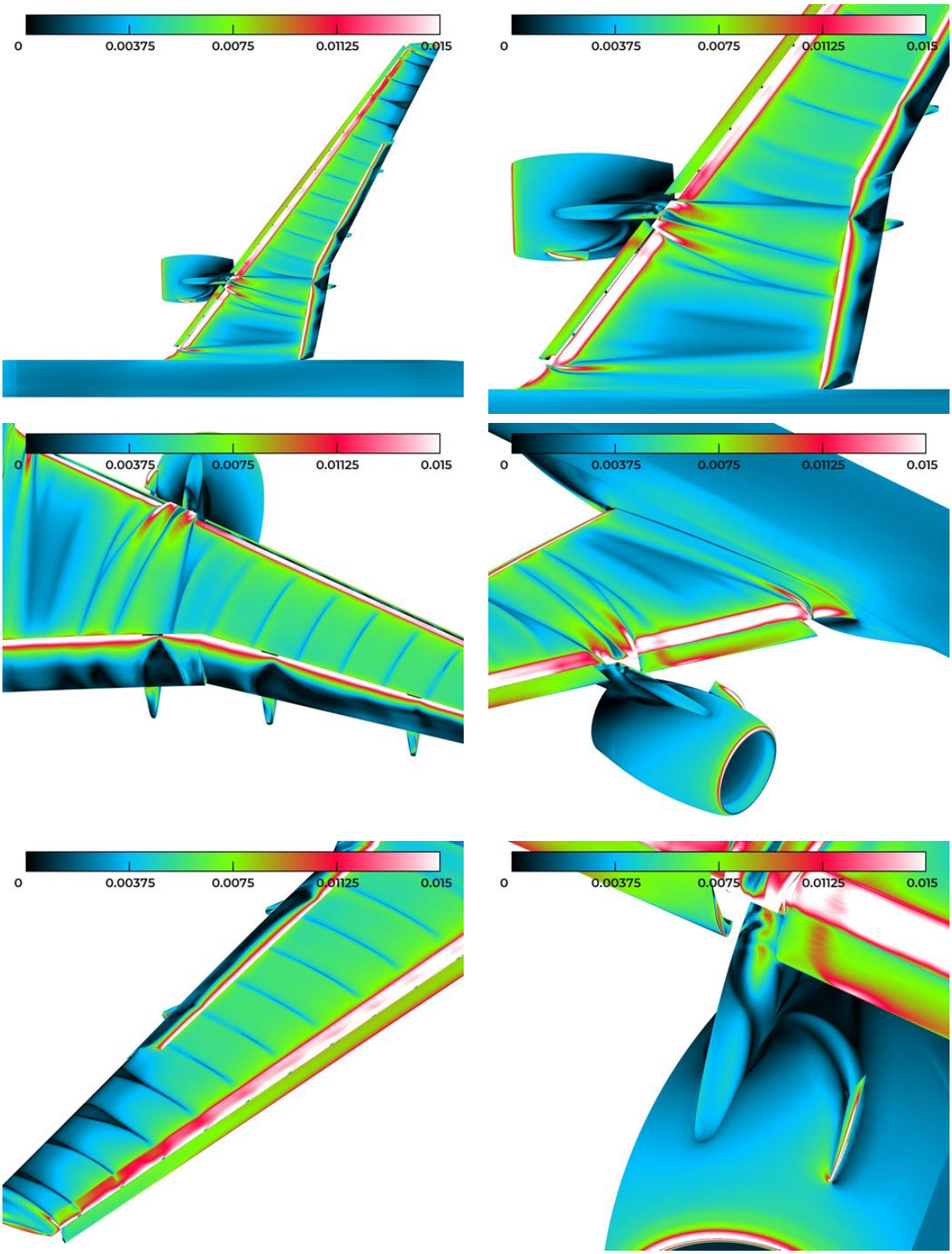


Fig. 16 Sequence mode: generate images from several data files

```
import pyviz

d = pyviz.Data()
# read an existing data file
d.read('cut_mesh.data')

# modify parameters and write a new data file
d.enable('UsePalette')
d.set('Palette', -3, -2, -1, 0., 1)
d.enable('PalOn')
d.set('SolOn', 1)
d.write('cut_sol.data')

# modify others parameters and write an other data file
d.set('IsoOn', 1)
d.set('Palette', 0, 0.25, 0.5, 0.75, 1)
d.disable('PalOn')
d.disable('LineOn')
d.set('SolOn', 2)
d.write('cut_iso.data')

# declare a sequence list of data + output required
s = pyviz.Sequence()
# add all data in the sequence file
s.add('cut_mesh.data', 'cut_mesh.jpg')
s.add('cut_sol.data', 'cut_sol.jpg')
s.add('cut_iso.data', 'cut_iso.jpg')

s.write('vizir.seq')
```

Listing 2 Example of file using `Pyviz 4`

E. Postprocessing: Mean Surface Pressures and Skin Friction Extraction

The organizers of the workshops ask the participants to extract surface data in several given planes. The configuration is shown in Fig. 18. These extractions have been added to `ViZiR 4`.

The idea is for each plane to find all surface triangles intersecting the plane. In the general case, two edges of these triangles are intersected by the plane. These intersection points can be easily computed and the solution evaluated too. These two points give form to a new extracted edge. Fig. 19 shows an example of all these new extracted edges.

Once all these edges and intersected points are created, the vertices are renumbered such that all vertices are sorted (i.e. follow the surface in the same order). For this purpose, for each vertex, we count to how new edges the vertex belongs: only 1 means the vertex is at an extremity, otherwise it should be 2. Then, we use the list of edges to cover all the vertices one by one. Finally, a file is created with all the vertices, their coordinates and the values of the solution. Fig. 20 shows an example of extractions obtained for the wing (from A to F following configuration given in Fig. 18).

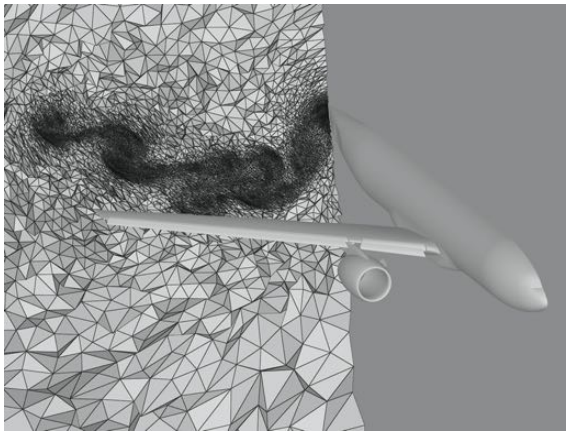
V. Conclusions

In this paper, we illustrated the capabilities of `ViZiR 4` with large meshes and solutions from the 4th AIAA CFD High Lift Prediction. In particular, we showed that `ViZiR 4` is much faster than `ParaView`. We showed that many fast, precise and interactive tools, such that picking and hiding, isolines, cut planes, allow to analyze and investigate quickly the results. Furthermore, scripting tools allow to generate quickly images and go over sequences of several meshes that is useful when mesh adaptation is involved.

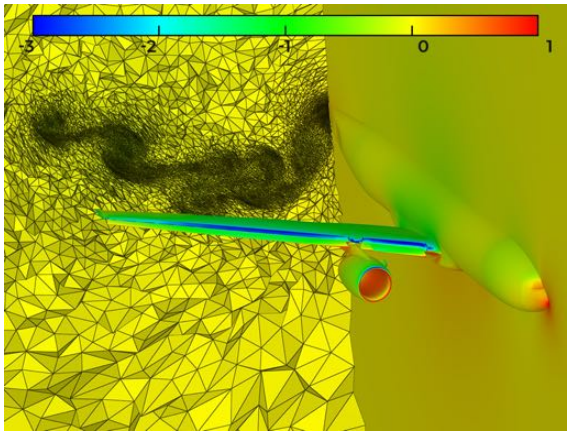
All these developments are part of a the new generation of `ViZiR` software, `ViZiR 4` [12, 13] which can be downloaded in its dedicated web site `https://vizir.inria.fr`, that is developed at Inria by GAMMA team. The software is at the time able to render any geometry from degree 1 to 4 and any polynomial solution from degree 0 to 10. These degrees are arbitrary as sufficient to answer today's needs.

VI. Acknowledgments

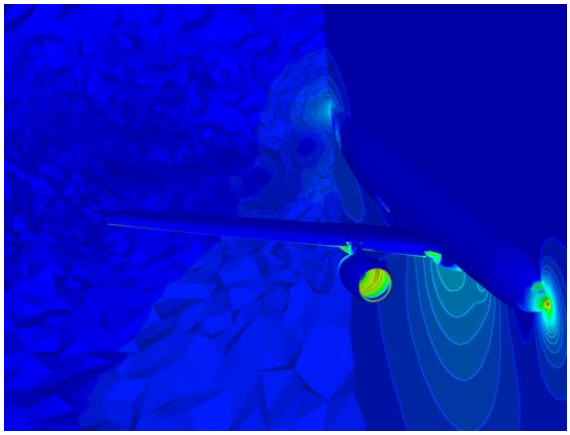
This work was supported by the public grant ANR Impacts, reference ANR-18-CE46-0003. The authors are also grateful to Lucien Rochery (Inria) for providing high-order meshes, Loïc Maréchal (Inria) for providing the libMeshb library, Frédéric Alauzet for providing numerical solutions with the Wolf solver and Julien Vanharen (ONERA) for his help in the development of the scripts in python.



(a) cut_mesh.jpg



(b) cut_sol.jpg



(c) cut_iso.jpg

Fig. 17 Results of Listing 2 on an adapted mesh of 20480K and its pressure coefficients solution.

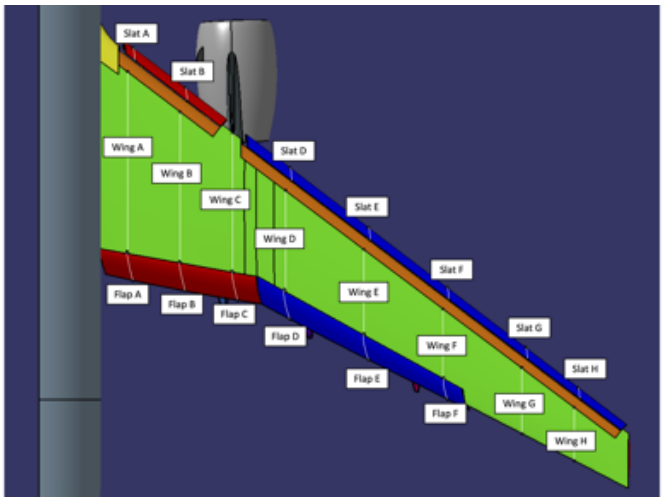


Fig. 18 Configuration of mean surface pressures and skin friction extraction. Image from the workshop website.

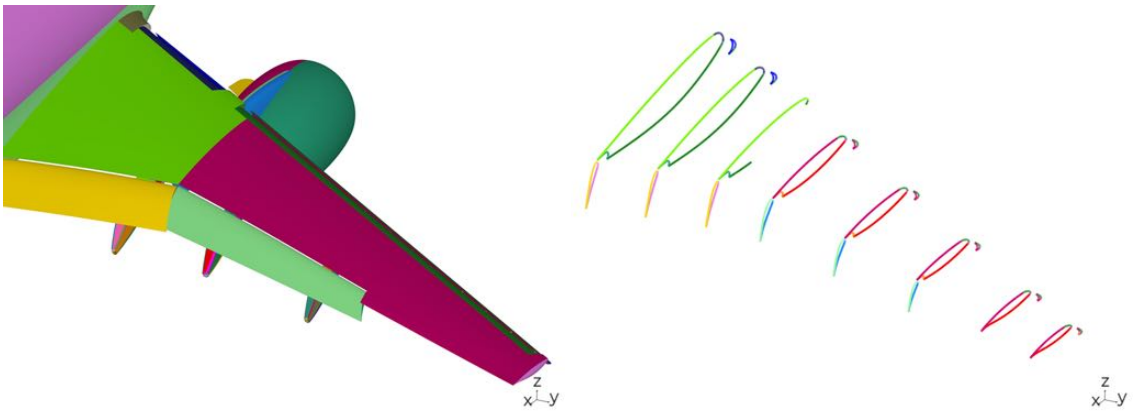


Fig. 19 An example of edges extractions according to the configuration shown in Fig 18. The surface (left) and all the new created edges (right).

References

[1] Ayachit, U., *The paraview guide: a parallel visualization application*, Kitware, Inc., 2015.

[2] Childs, H., "VisIt: An end-user tool for visualizing and analyzing very large data," 2012.

[3] TecPlot Inc., "TecPlot," https://www.tecplot.com/, 2021.

[4] Geuzaine, C., and Remacle, J.-F., "Gmsh: A 3-D finite element mesh generator with built-in pre-and post-processing facilities," *International journal for numerical methods in engineering*, Vol. 79, No. 11, 2009, pp. 1309–1331.

[5] Schroeder, W. J., Bertel, F., Malaterre, M., Thompson, D., Pebay, P. P., O'Bara, R., and Tendulkar, S., "Methods and framework for visualizing higher-order finite elements," *IEEE Transactions on Visualization and Computer Graphics*, Vol. 12, No. 4, 2006, pp. 446–460.

[6] Xu, L., Ren, X., Xu, X., Li, H., Tang, Y., and Feng, Y., "An adaptive visualization tool for high order discontinuous galerkin method with quadratic elements," *2017 IEEE International Conference on Computer and Information Technology (CIT)*, IEEE, 2017, pp. 176–183.

[7] Maunoury, M., Besse, C., Mouysset, V., Pernet, S., and Haas, P.-A., "Well-suited and adaptive post-processing for the visualization of hp simulation results," *Journal of Computational Physics*, Vol. 375, 2018, pp. 1179–1204.

[8] Maunoury, M., "Méthode de visualisation adaptée aux simulations d'ordre élevé. Application à la compression-reconstruction de champs rayonnés pour des ondes harmoniques." Ph.D. thesis, 2019.

[9] Nelson, B., Liu, E., Kirby, R. M., and Haimes, R., "Elvis: A system for the accurate and interactive visualization of high-order finite element solutions," *IEEE transactions on visualization and computer graphics*, Vol. 18, No. 12, 2012, pp. 2325–2334.

[10] Nelson, B. W., "Accurate and interactive visualization of high-order finite element fields," Ph.D. thesis, 2012.

[11] Peiro, J., Moxey, D., Jordi, B., Sherwin, S., Nelson, B., Kirby, R., and Haimes, R., "High-order visualization with ElVis," *IDIHOM: Industrialization of High-Order Methods-A Top-Down Approach*, Springer, 2015, pp. 521–534.

[12] Loseille, A., and Feuillet, R., "Vizir: High-order mesh and solution visualization using OpenGL 4.0 graphic pipeline," *56th AIAA Aerospace Sciences Meeting, AIAA Scitech*, 2018. https://doi.org/10.2514/6.2018-1174.

[13] Feuillet, R., Maunoury, M., and Loseille, A., "On pixel-exact rendering for high-order mesh and solution," *Journal of Computational Physics*, Vol. 424, 2021, p. 109860. https://doi.org/https://doi.org/10.1016/j.jcp.2020.109860, URL https://www.sciencedirect.com/science/article/pii/S0021999120306343.

[14] Alauzet, F., and Loseille, A., "A decade of progress on anisotropic mesh adaptation for computational fluid dynamics," *Computer-Aided Design*, Vol. 72, 2016, pp. 13–39. 23rd International Meshing Roundtable Special Issue: Advances in Mesh Generation.

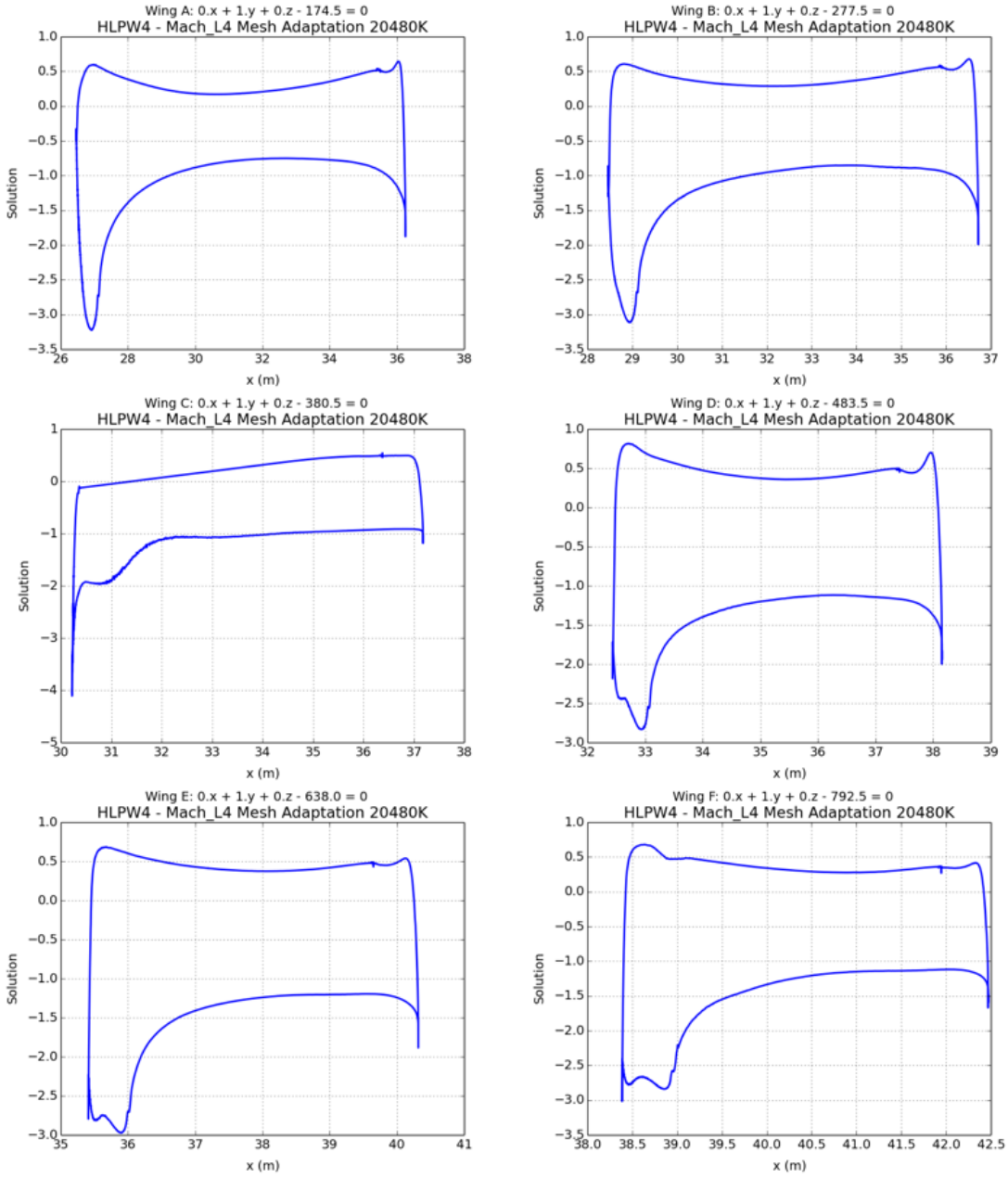


Fig. 20 Solution extracted on the wing.

[15] Loseille, A., "Mesh Generation and Adaptation for scientific computing," 2020.

[16] Loseille, A., Dervieux, A., and Alauzet, F., "Fully anisotropic goal-oriented mesh adaptation for 3D steady Euler equations," *Journal of Computational Physics*, Vol. 229, No. 8, 2010, pp. 2866–2897.

[17] Wolff, D., *OpenGL 4.0 Shading Language Cookbook*, Packt Publishing, 2011.

[18] Sellers, G., Wright, R., and Haemel, N., *OpenGL SuperBible, Sixth Edition*, Addison-Wiley, 2013.

[19] Baffet, D. H., Grote, M. J., Impériale, S., and Kachanovska, M., "Energy Decay and Stability of a Perfectly Matched Layer For the Wave Equation," *Journal of Scientific Computing*, Vol. 81, No. 3, 2019, pp. 2237–2270.

[20] Feuillet, R., "Embedded and high-order meshes: two alternatives to linear body-fitted meshes," Ph.D. thesis, Paris Saclay, 2019.

[21] Borouchaki, H., and George, P. L., *Meshing, Geometric Modeling and Numerical Simulation 1: Form Functions, Triangulations and Geometric Modeling*, John Wiley & Sons, 2017.

[22] George, P. L., Borouchaki, H., Alauzet, F., Laug, P., Loseille, A., and Marechal, L., *Meshing, Geometric Modeling and Numerical Simulation, Volume 2: Metrics, Meshes and Mesh Adaptation*, John Wiley & Sons, 2019.

[23] George, P., and Borouchaki, H., "Construction of tetrahedral meshes of degree two," *International Journal for Numerical Methods in Engineering*, Vol. 90, No. 9, 2012, pp. 1156–1182.

[24] George, P.-L., Borouchaki, H., and Barral, N., "Geometric validity (positive jacobian) of high-order Lagrange finite elements, theory and practical guidance," *Engineering with computers*, Vol. 32, No. 3, 2016, pp. 405–424.

[25] George, P.-L., and Borouchaki, H., "Validité des éléments finis de Lagrange de degré 1 et 2," 2013.

[26] Roca, X., Gargallo-Peiró, A., and Sarrate, J., "Defining quality measures for high-order planar triangles and curved mesh generation," *Proceedings of the 20th International Meshing Roundtable*, Springer, 2011, pp. 365–383.

[27] Johnen, A., Remacle, J.-F., and Geuzaine, C., "Geometrical validity of curvilinear finite elements," *Journal of Computational Physics*, Vol. 233, 2013, pp. 359–372.

[28] Toulorge, T., Geuzaine, C., Remacle, J.-F., and Lambrechts, J., "Robust untangling of curvilinear meshes," *Journal of Computational Physics*, Vol. 254, 2013, pp. 8–26.