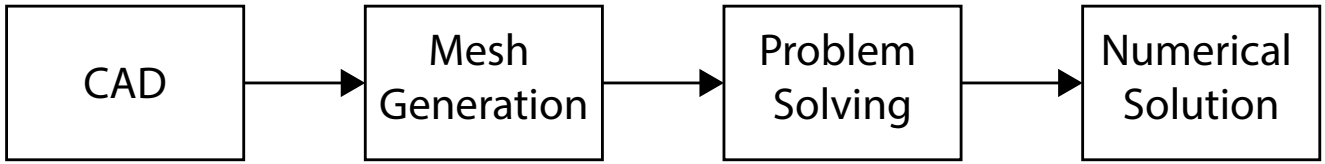
Visualisation rapide et interactive de maillages complexes (grandes tailles, courbes) et de solutions d'ordre élevé avec ViZiR 4

Matthieu Maunoury, Adrien Loseille

Inria Saclay
Équipe GAMMA (Génération Adaptative de Maillage et Méthodes numériques Avancées)

CANUM 2020
13 – 17 Juin 2022

Introduction



Visualization tools are necessary:

- **inspect** the CAD model
- check the **validity** and **quality** of the meshes
- **analyze** the potential problems on meshes and solutions
- **validate** algorithms

Applications: Computational Fluid Dynamic, acoustics, energy, electromagnetism or medical modeling...

Current approaches for visualization software

Many visualization software (e.g. ParaView, Gmsh, Medit, Tecplot, VisIt, Vizir Legacy) to analyze numerical results:

- Based on **linear** primitives as imposed by the commonly-used graphic pipeline.
- Many interesting **plugins** and **tools** to help the analyses.

Some limitations:

- **Interactivity** might be missing (time to open files and render meshes and solutions).
- Lack of tools to **efficiently manipulate** these meshes.
- High-order meshes are **generally not handled**.
- High-order solutions: **visualization error** due to *low order remeshing* (approximation of high order functions by affine functions).

Interactivity bottleneck: wall times comparisons

Laptop: MacBook Pro 2.6 GHz 6-core Intel Core i7 with 32 Gb of RAM, GPU is AMD Radeon Pro Vega 20 4 Gb.

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

Comparison of total rendering wall times (s) including files (mesh and solution) opening.

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

Comparison of wall times (s) to generate cut plane (clip).

Interactivity bottleneck: wall times comparisons

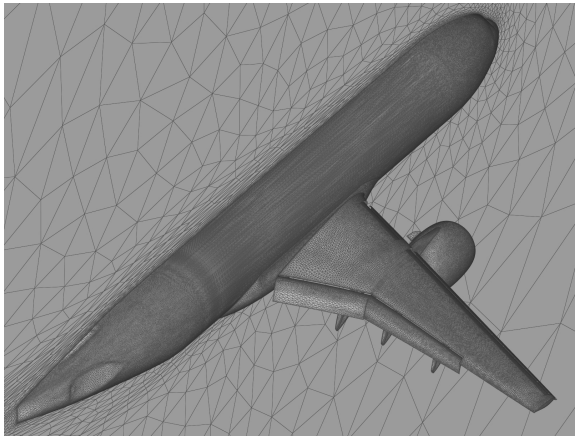
Case	# vertices	# triangles	ParaView (s)	ViZiR 4 (s)
640K	217 000	433 653	**7.2**	**0.01**
1280K	392 257	783 947	**13.9**	**0.01**
2560K	628 223	1 255 604	**24.6**	**0.01**
5120K	1 018 135	2 035 092	**39.2**	**0.01**
10240K	1 772 712	3 543 955	**63.1**	**0.01**
20480K	3 084 324	6 166 689	**109.3**	**0.01**

Comparison of wall times (s) to render isolines (contours) for different meshes composed only of triangles. **Done by the GPU in `ViZiR 4`.**

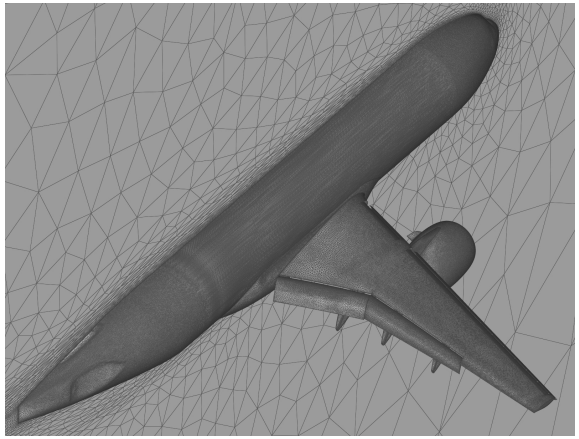
Some issues `ViZiR 4` tries to answer:

- **Interactivity** (i.e. fast) to develop meshes algorithms.
- Display with **high fidelity** the computed numerical solution.
- Handle **high-order** meshes and solutions.

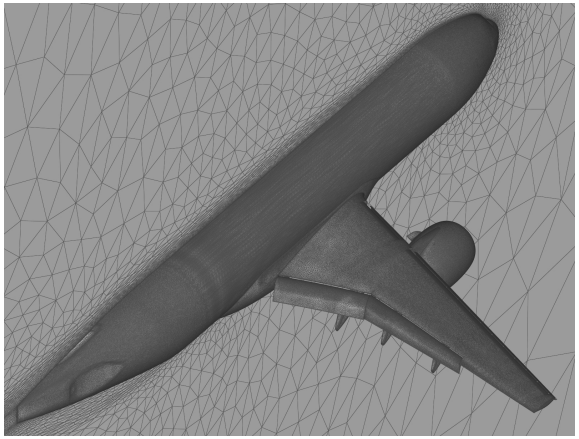
Interactivity bottleneck: wall times comparisons



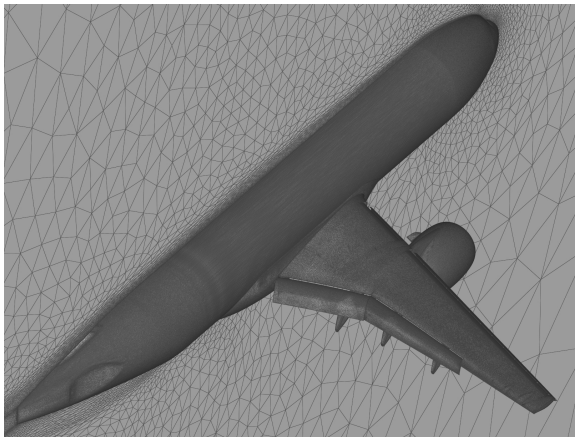
(a) 640K



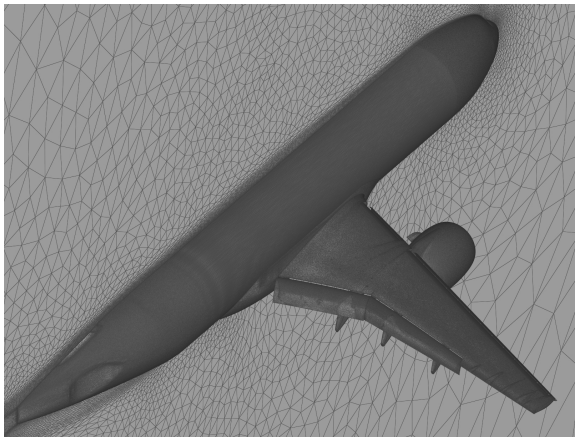
(b) 1280K



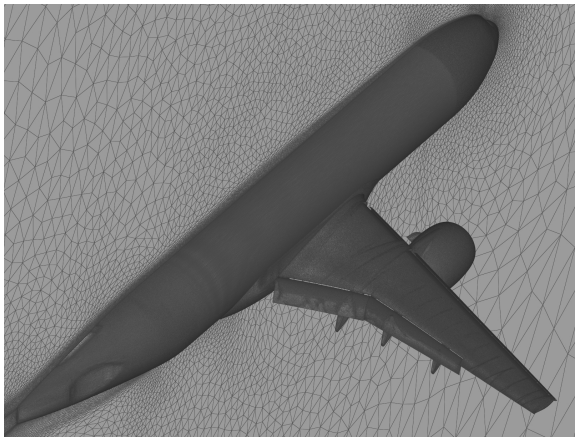
(c) 2560K



(d) 5120K



(e) 10240K

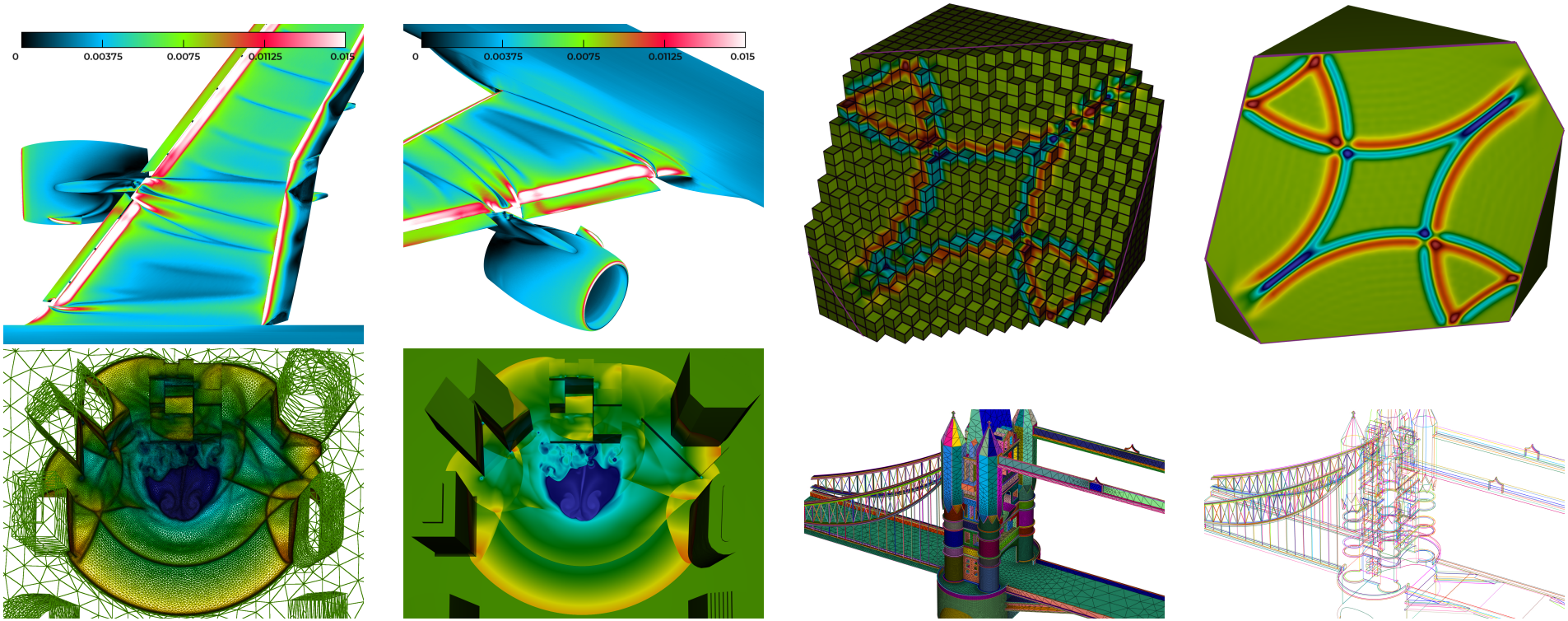


(f) 20480K

Adapted meshes used for comparisons

Main features of ViZiR 4:

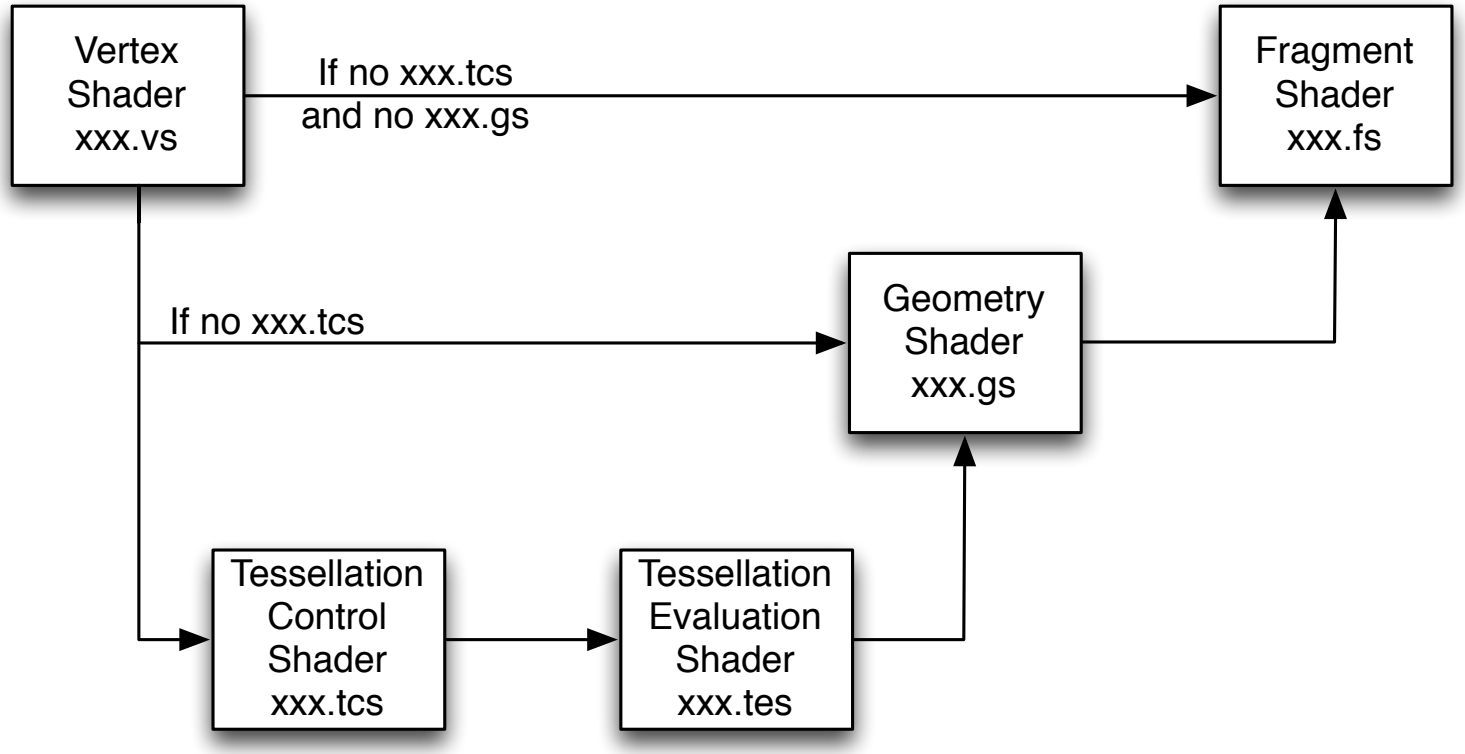
- **Light**, **simple** and **interactive** visualization software.
- **Surface** and **volume** (tetrahedra, pyramids, prisms, hexahedra) meshes.
- **Pixel exact** rendering of **high-order** solutions on straight elements.
- **Almost pixel exact** rendering on curved elements (high-order meshes).
- **Post-processing tools**, such as picking, isolines, clipping, capping.



OpenGL 4 graphic pipeline

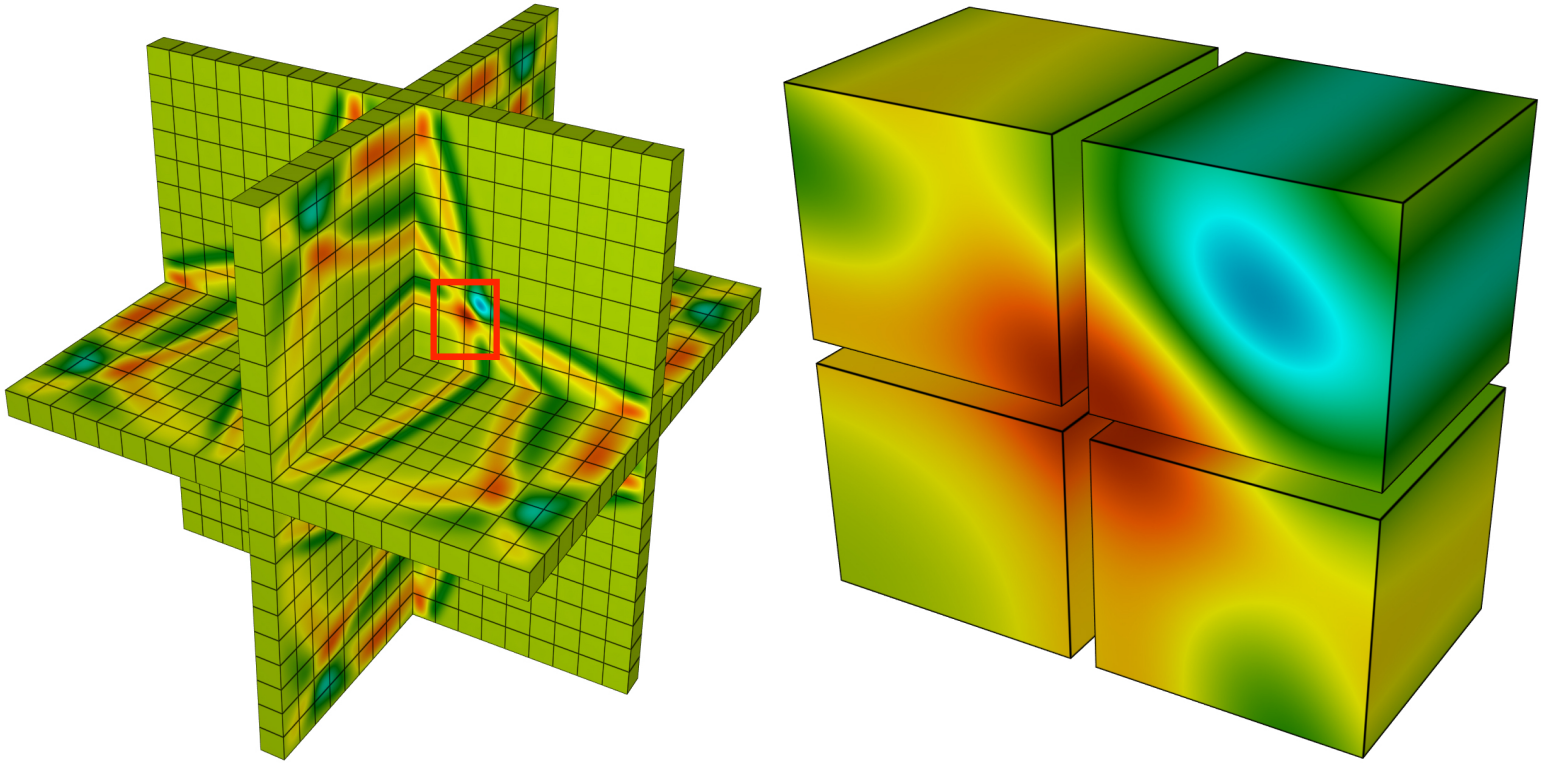
`ViZiR 4` is based on OpenGL Shading Language (GLSL) and shaders (GPU programs).

[Rémi Feuillet, Matthieu Maunoury, Adrien Loseille, On pixel-exact rendering for high-order mesh and solution, Journal of Computational Physics, 2021]



Pixel exact rendering on flat elements

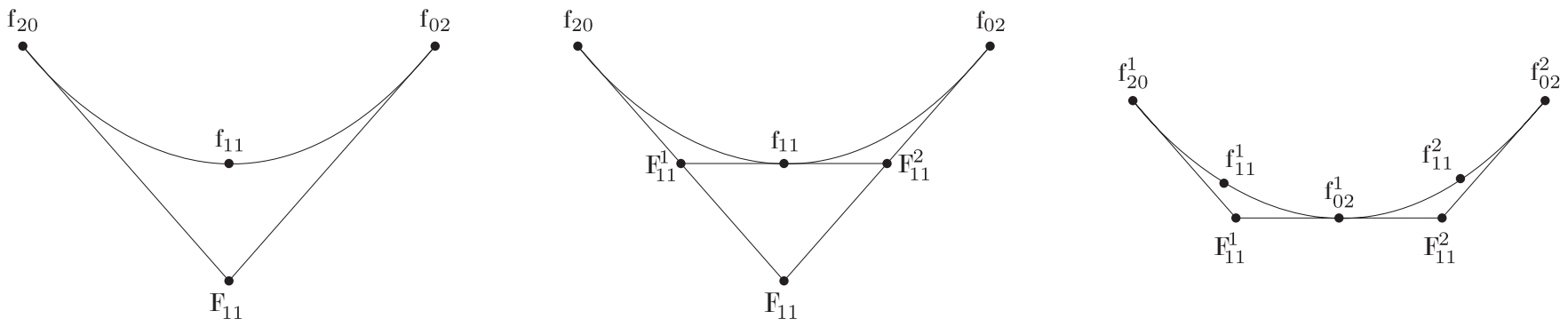
- For each pixel, Fragment shader determines the appropriate color.
- It certifies a faithful and interactive depiction (up to degree 10 polynomial function).
- High order solutions are natively handled by `ViZiR 4` on surface and volume (tetrahedra, pyramids, prisms, hexahedra) meshes.



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

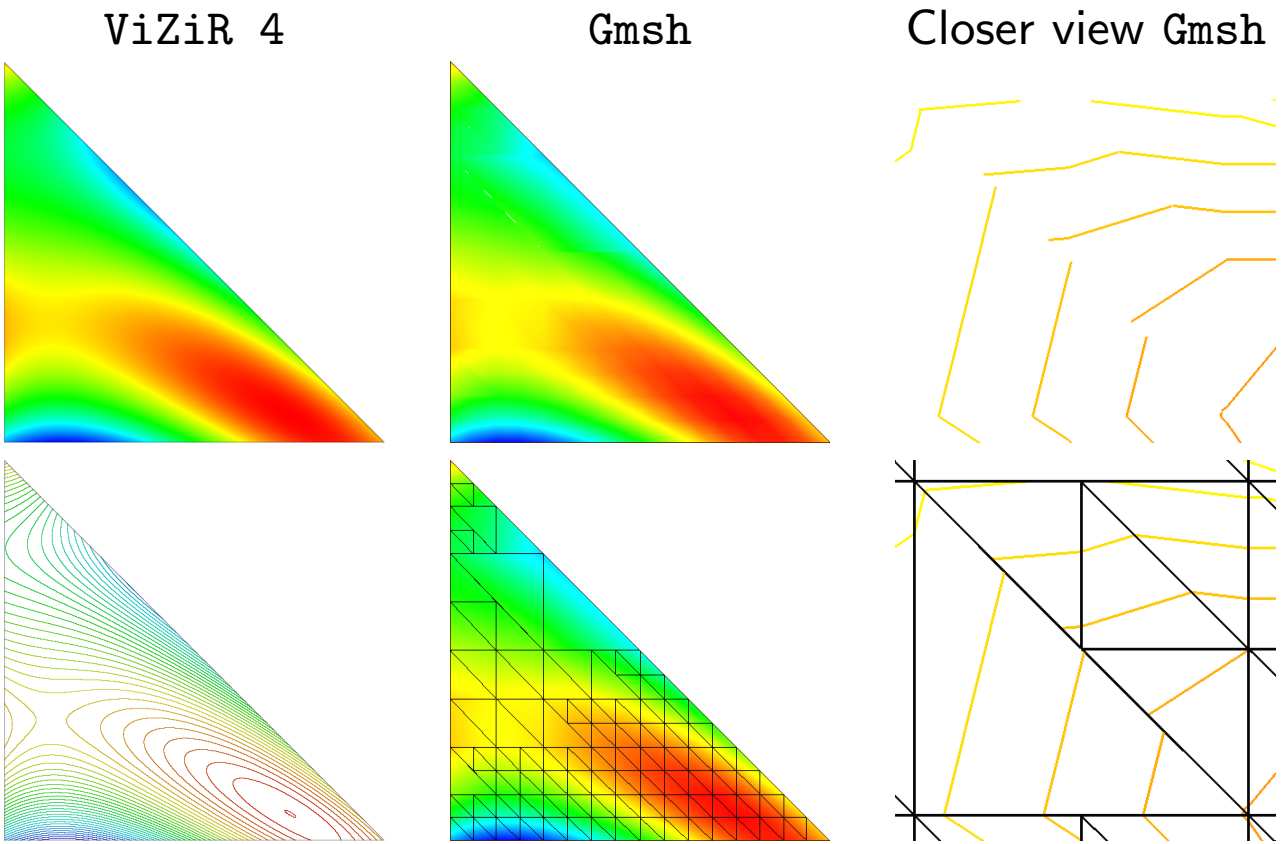
Computation of proper bounds for a high-order solution

- For affine solutions, extrema of solution lies on the element vertices.
- For HO solutions, extrema of solution can lie inside an element.
- Idea (pre-processing step): subdivide elements, evaluate solution in these sub-elements with a **de Casteljau** algorithm [Feuillet et al., JCP 2021] and update the bounds.



Recursive subdivision of a solution along an edge using a de Casteljau's algorithm

State of the art on HO solution visualization



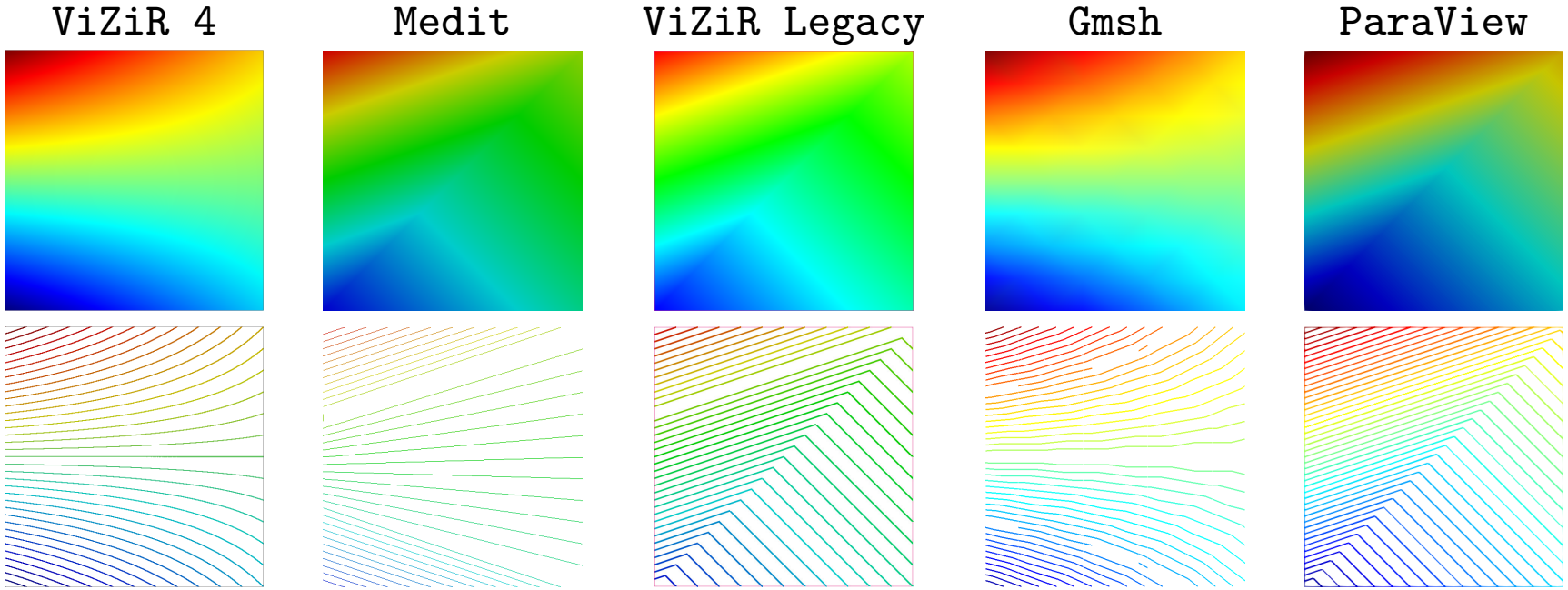
Rendering of a P^3-solution on a triangle.

Left: Our pixel-exact representation (top) and isolines (bottom).

Middle: Representation (top) by Gmsh, with 1% of visualization error, tessellation of **169 sub-triangles** (bottom).

Right: Zoom on **discontinuities** (top) due to non-conformal tessellation (bottom).

State of the art on HO solution visualization

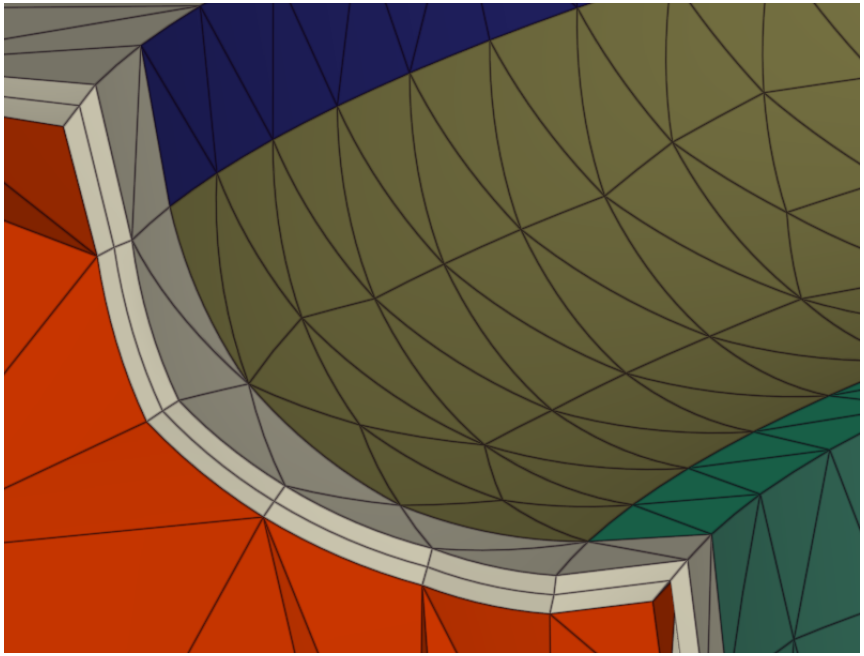


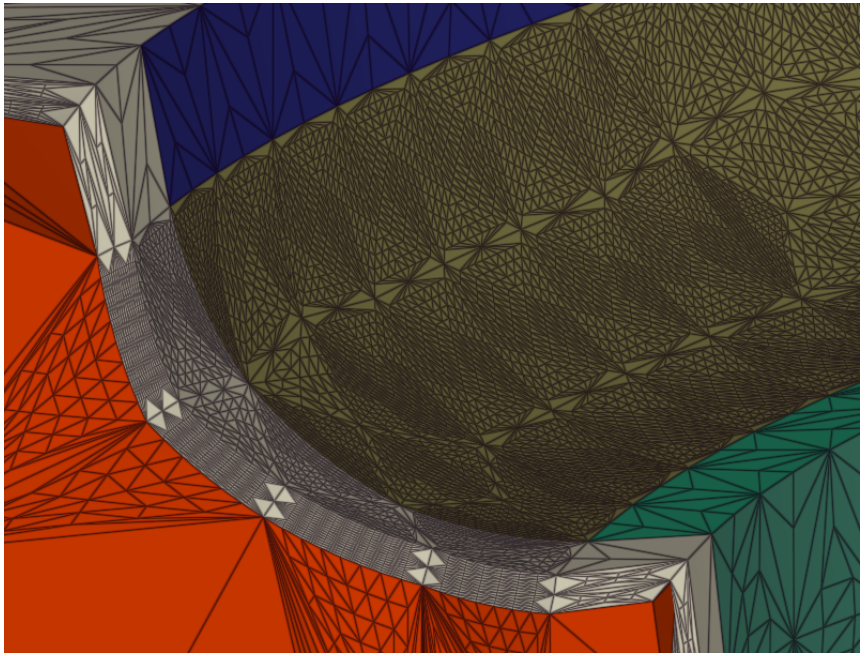
Rendering of Q^1-solution on Q^1-quadrilateral: representation (top) and isolines (bottom).

- **A subdivision** of 2 triangles is done in `Medit`, `ViZiR Legacy` and `ParaView`.
- **A tessellation** of 8×8 quadrilaterals is done in `Gmsh`.
- **No tessellation** in `ViZiR 4` (our method) as a pixel exact rendering is performed.

Tessellation on GPU for high-order elements

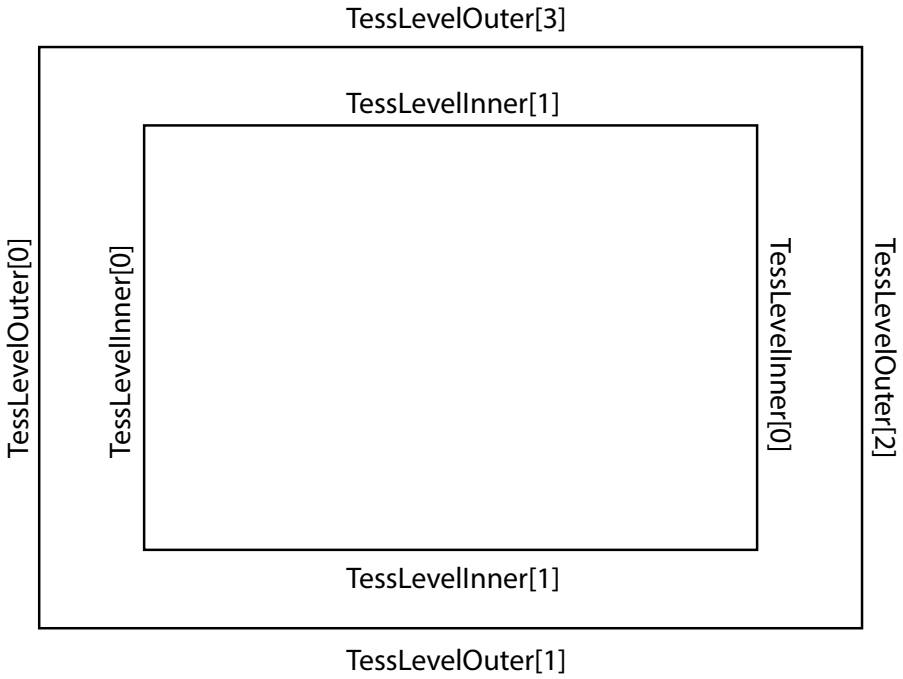
- Curved elements perform a better approximation of the geometry.
- Tessellation shaders: creation of sub-elements **on the fly** by the GPU.

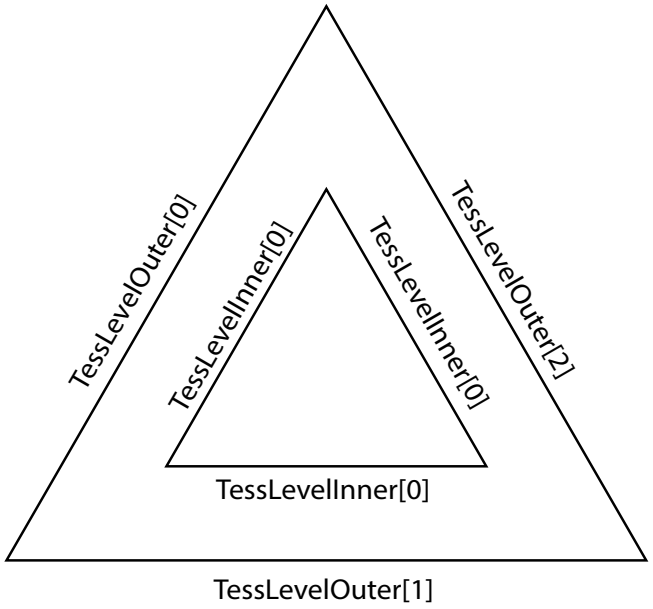




High-order mesh (left) and its tessellation constructed on the fly by the GPU (right).

Tessellation on GPU for high-order elements





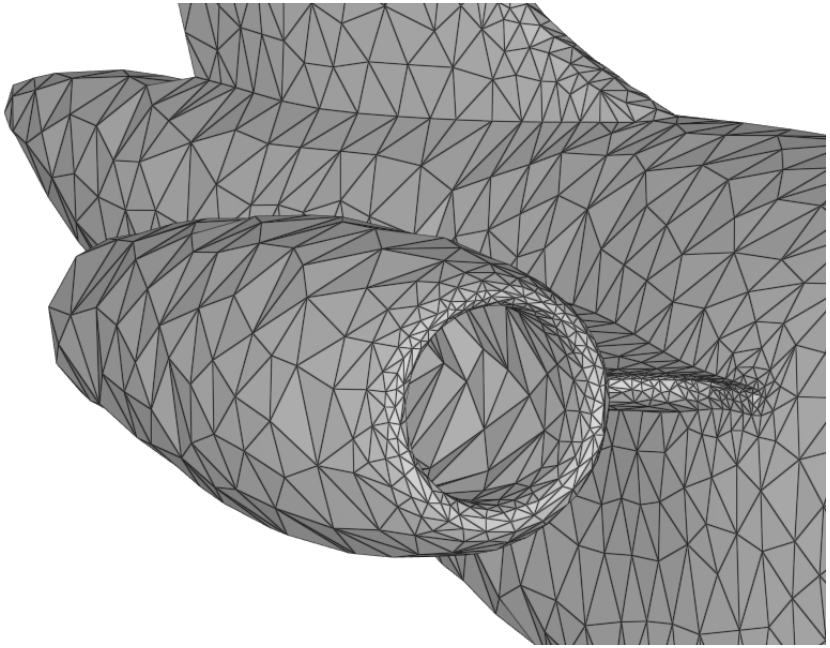
Variables controlling the subdivision in the Tessellation Control Shader.

- `TessLevelOuter[i]`: number of subdivisions along the $i-th$ line of the element
- `TessLevelInner`: number of subdivisions inside the element.

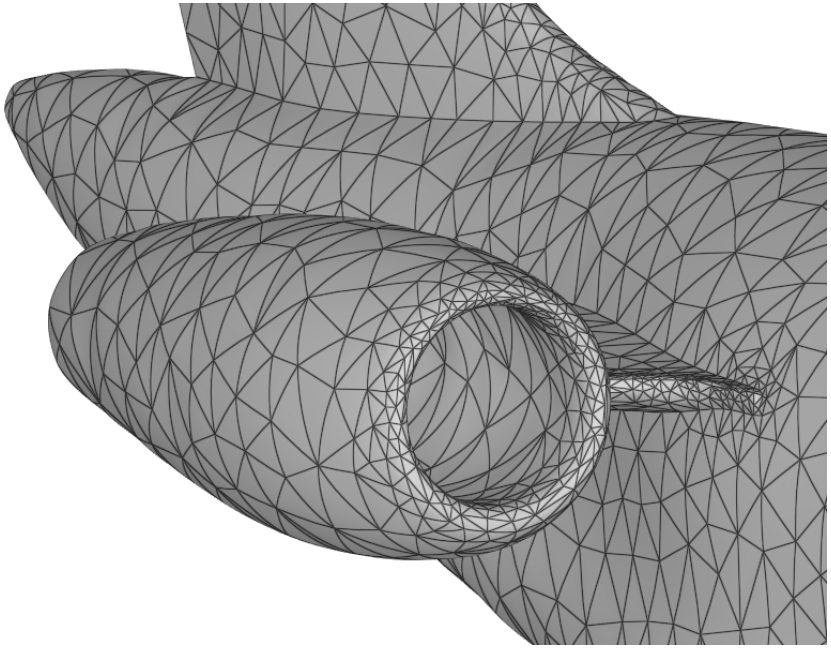
$\rightarrow$ Set up simple but **fast error estimates**.

Illustration of a high-order mesh

Degree 1 (P^1) mesh



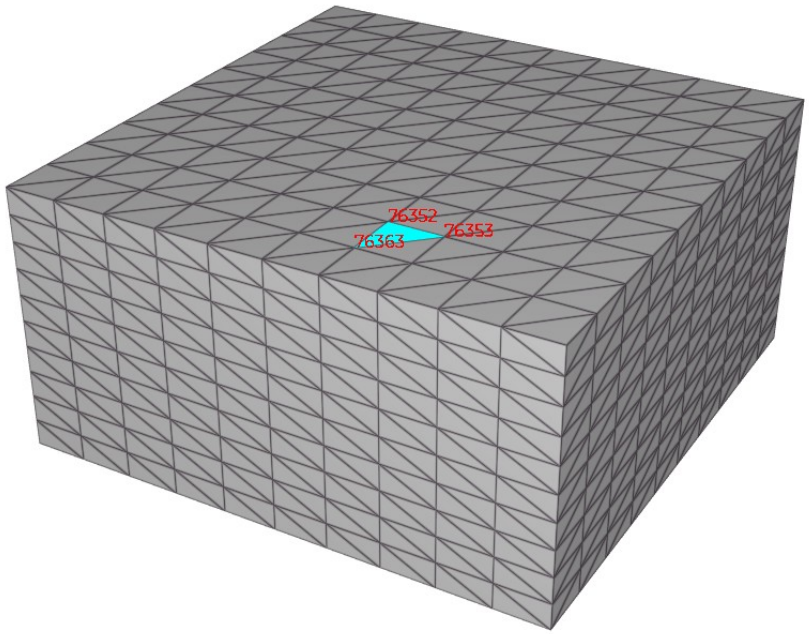
Degree 3 (P^3) mesh

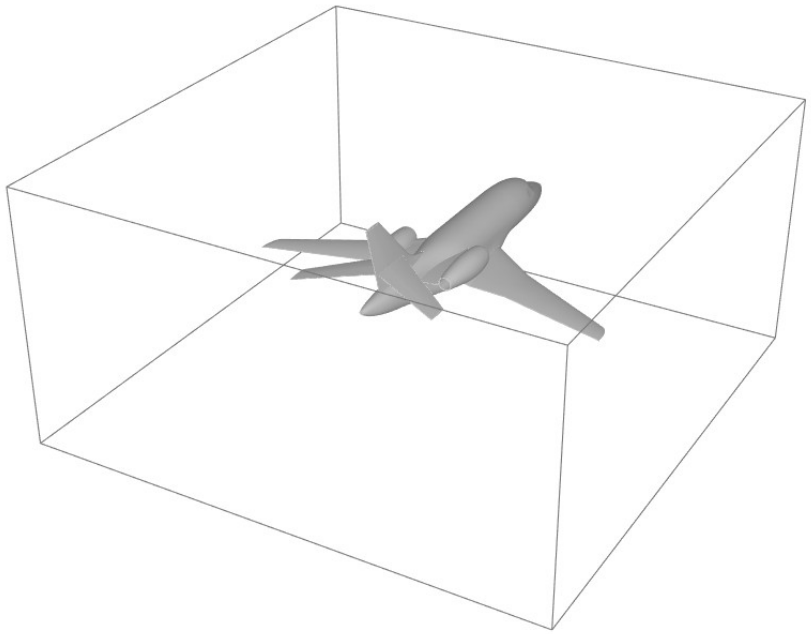


Meshes of degree 1 (left) and 3 (right) for the same number of elements.

Picking and hiding surfaces by reference

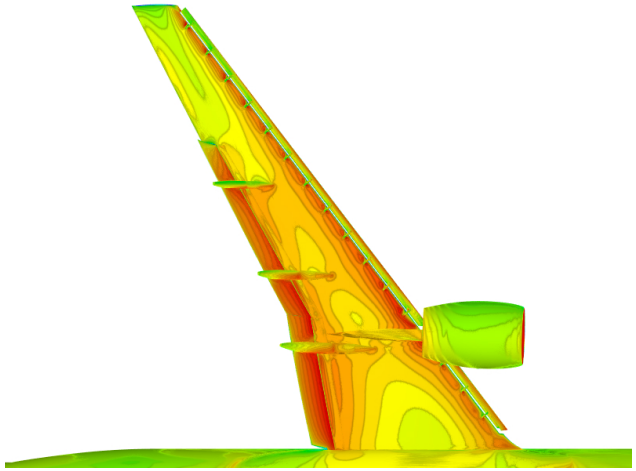
- Any element or vertex can be picked to get information: its index, indexes of vertices, coordinates, values of the solutions...
- After an element is picked, it is possible to hide all elements having the same reference id (corresponding typically to a patch).

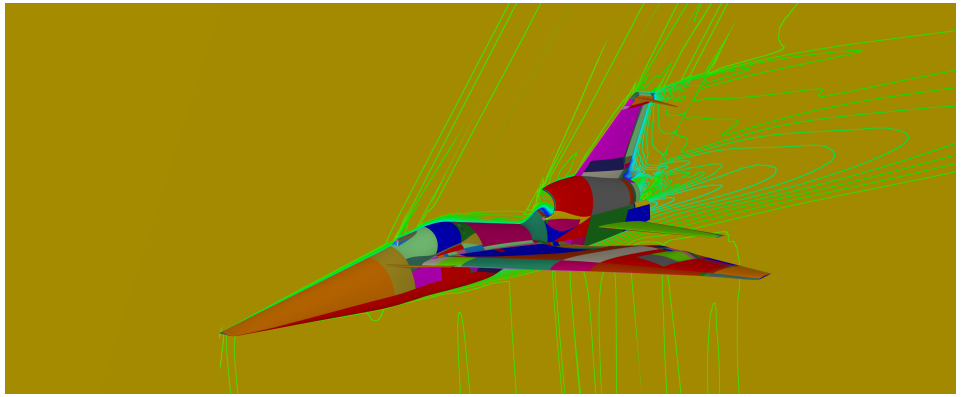


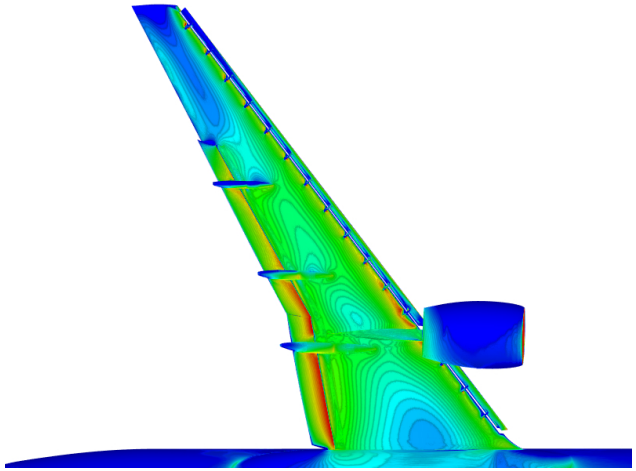


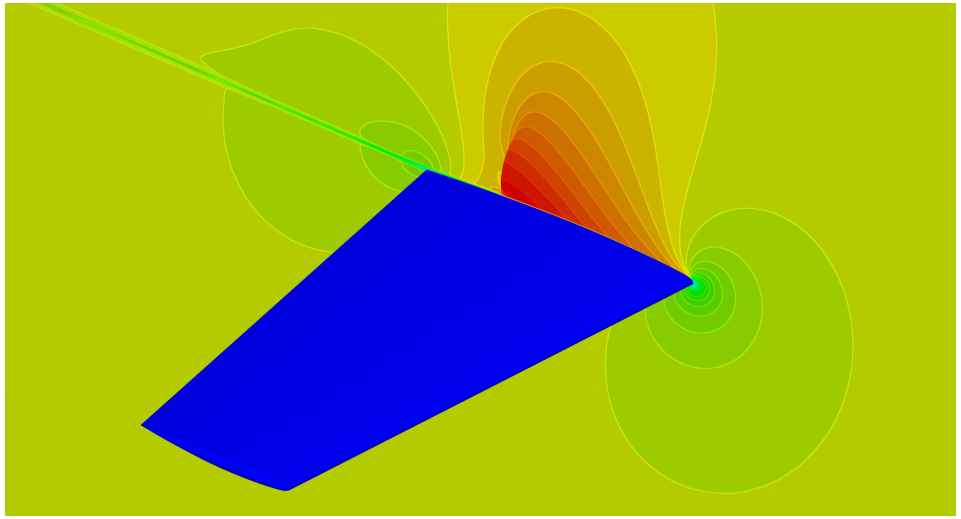
Example of picking (left) and hiding surfaces by reference (right).

Isolines (**instant** rendering) with possibly filled solution rendering

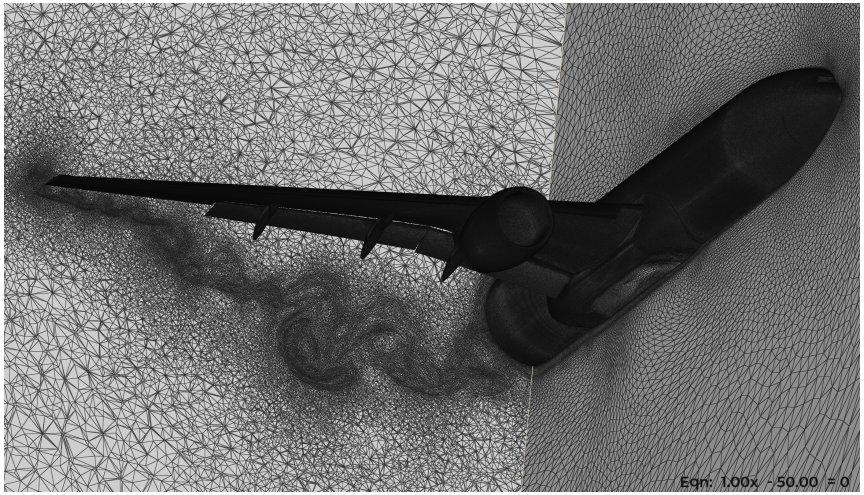


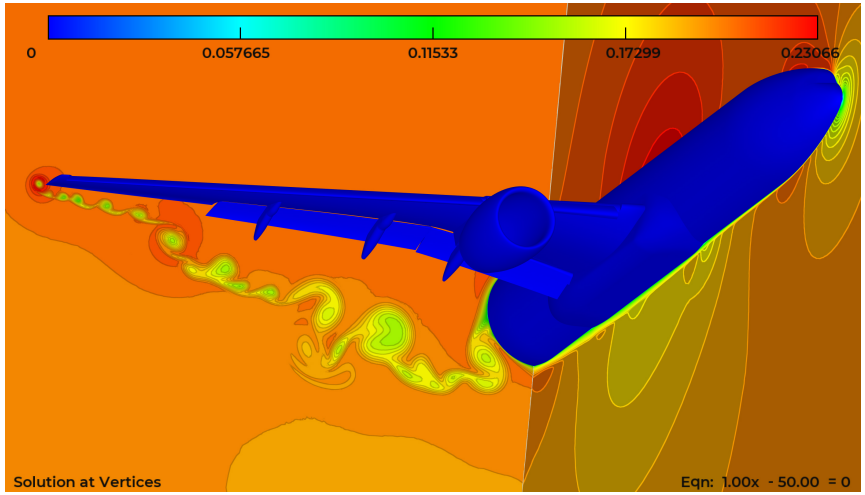


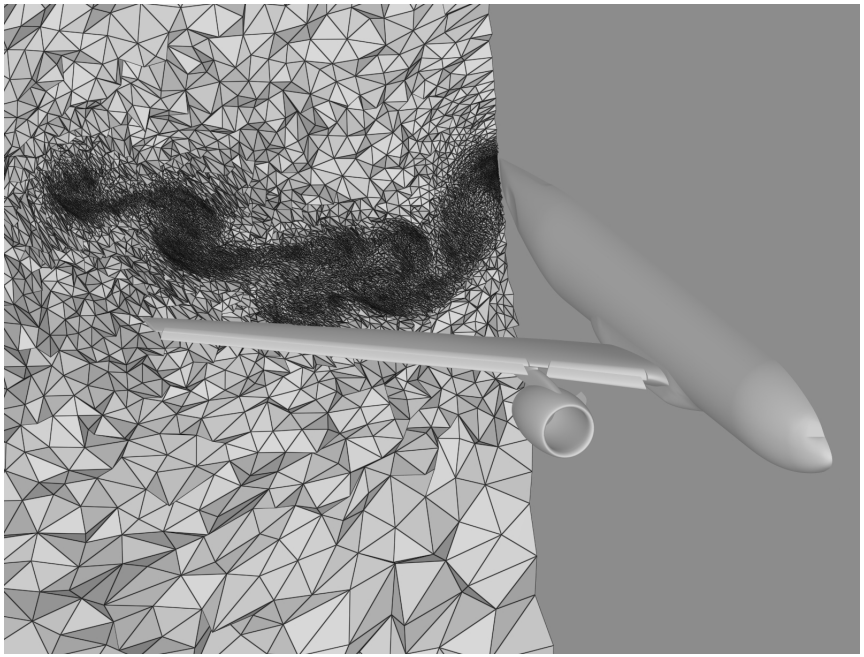


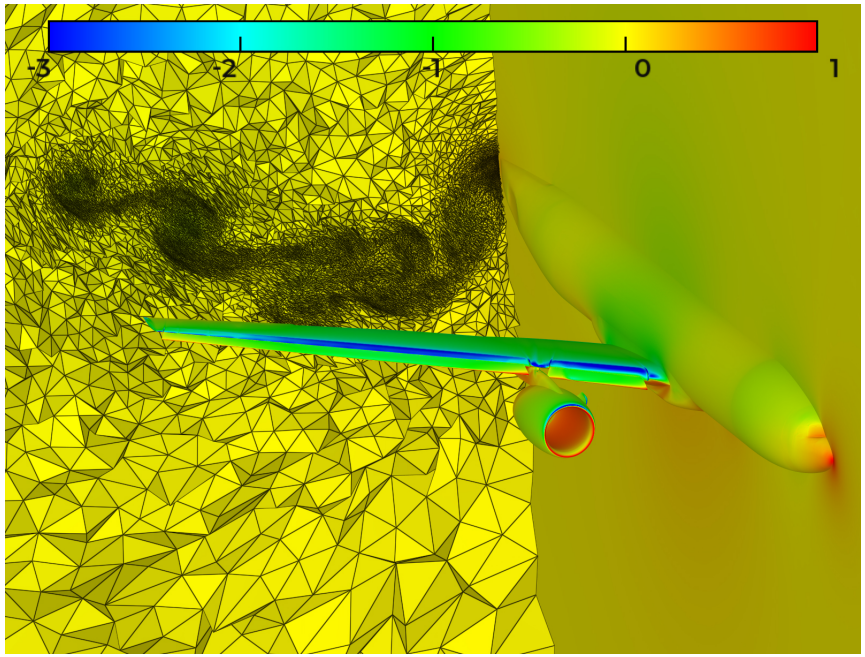


Cut Planes


Eqn: 1.00x - 50.00 = 0

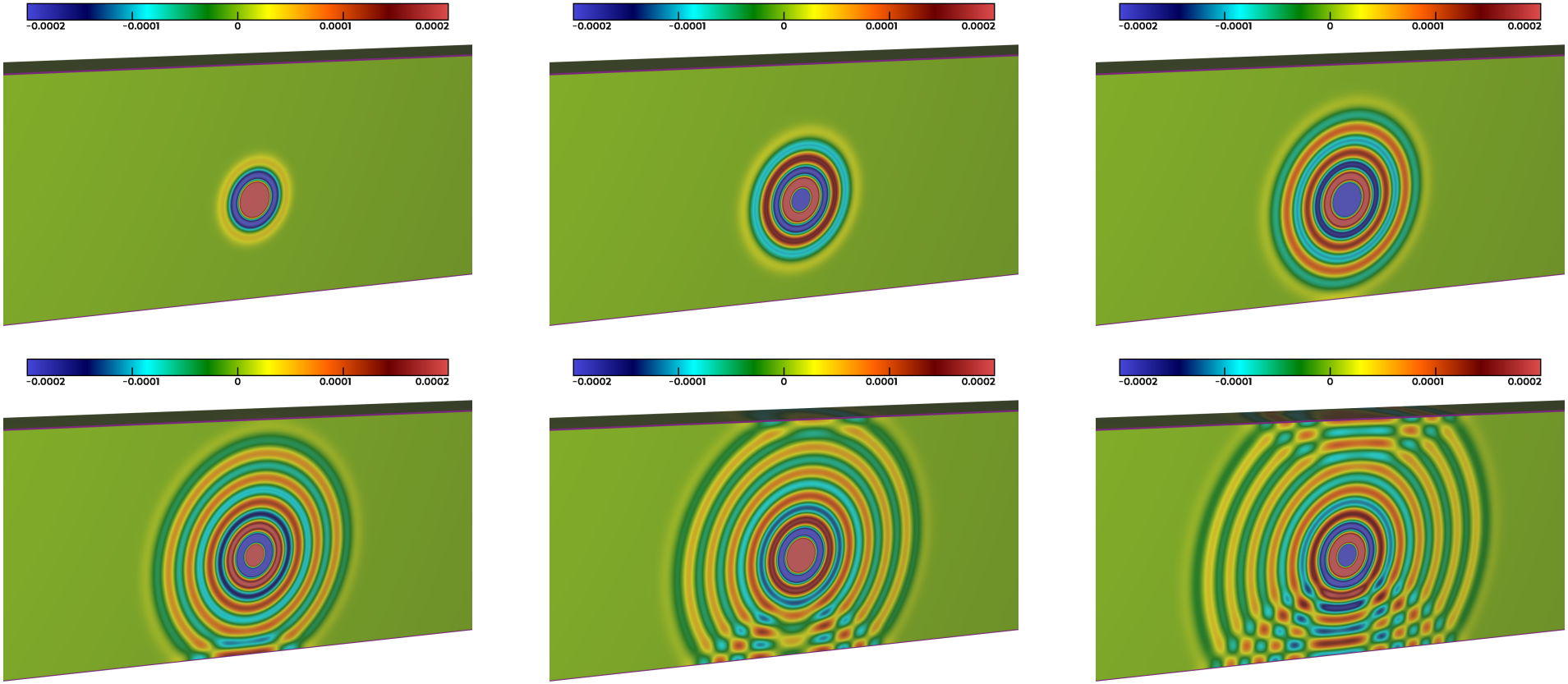

0
0.057665
0.11533
0.17299
0.23066
Solution at Vertices
Eqn: 1.00x - 50.00 = 0




-3
-2
-1
0
1

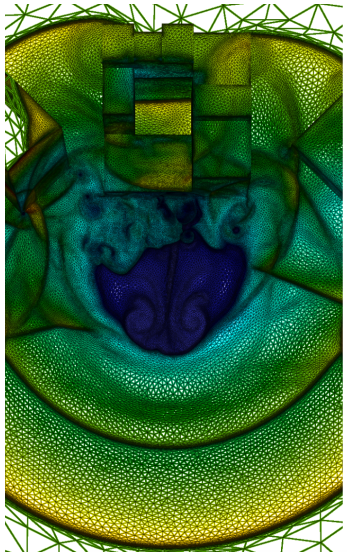
Scripting tools (for instance for non-stationary problems)

Scripting tools to easily generate images or go over large set of meshes / solutions.



Web site

`ViZiR 4` web site: `https://pyamg.saclay.inria.fr/vizir4.html` with executables (Mac, Linux, Windows), samples (meshes and solutions files) and user guide.



ViZiR 4

ViZiR 4 is a light, simple and interactive high-order meshes and solutions visualization software using OpenGL 4 graphic pipeline.

Its main features are:

- Light, simple and interactive visualization software.
- Surface and volume (tetrahedra, pyramids, prisms, hexahedra) meshes.
- Pixel exact rendering of high-order solutions on straight elements.
- Almost pixel exact rendering on curved elements (high-order meshes).
- Post-processing tools, such as picking, isolines, clipping, capping.

GO TO VIZIR 4 PAGE FOR MORE DETAILS

VIZIR LEGACY | BACK TO TOP

Download ViZiR

Please follow ViZiR user guide below to have the details on ViZiR installation.

Linux	MacOS	Windows
Download: Download ViZiR for Linux	Download: vizir4.2021.03.03.dmg	Download: vizir4.2021.01.21.windows.zip

All executables (previous or new versions) can be found in this directory

USER GUIDE | BACK TO TOP

Conclusions

- Fast I/O.
- Pixel-exact rendering of HO solutions (up to degree 10).
- Tessellation of HO elements on the fly by the GPU (up to degree 4).
- Many post-processing tools.
- Handle large, hybrid, HO, 3D meshes / solutions.

A. Loseille, R. Feuillet. Vizir: High-order mesh and solution visualization using OpenGL 4.0 graphic pipeline. American Institute of Aeronautics and Astronautics (AIAA) SciTech, 2018.

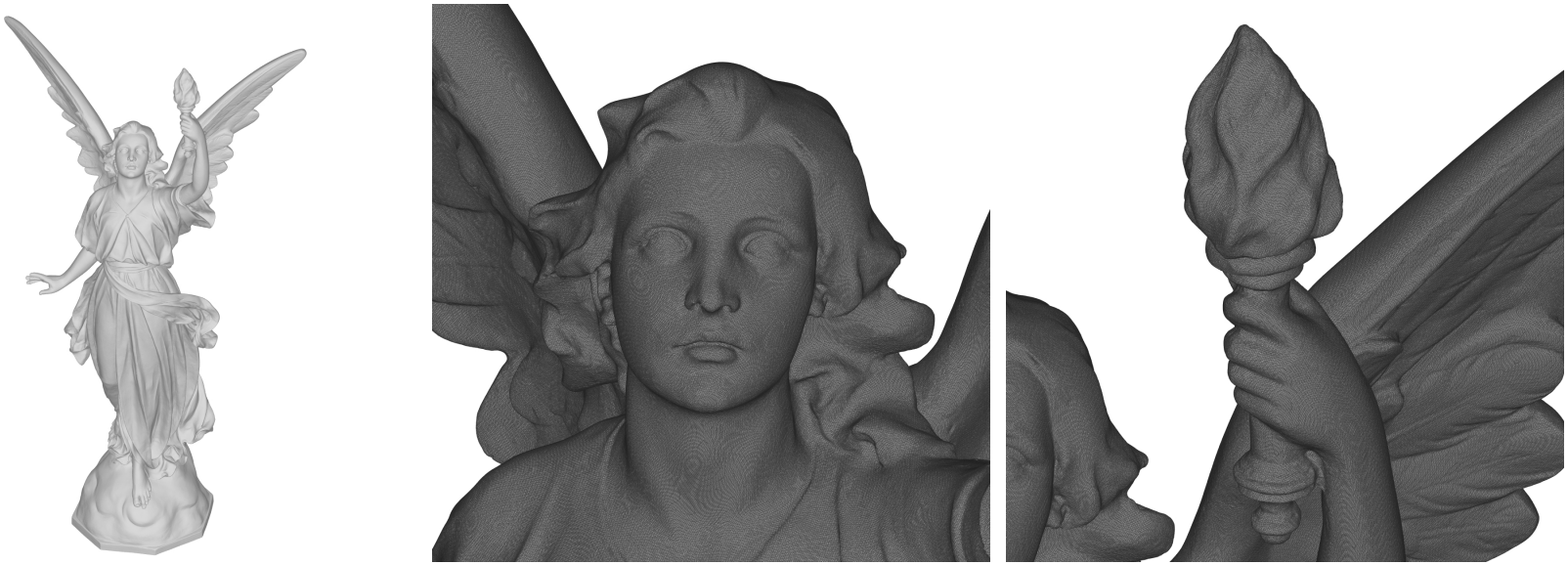
R. Feuillet, M. Maunoury, A. Loseille. On pixel-exact rendering for high-order mesh and solution. Journal of Computational Physics 424, 2021.

M. Maunoury, R. Feuillet, A. Loseille. Using ViZiR 4 to analyze the 4th AIAA CFD High Lift Prediction Workshop Simulations. American Institute of Aeronautics and Astronautics (AIAA) SciTech, 2022.

Thank you for your attention

Fast I/O

Input and output handled by the `libMeshb` library (Loïc Maréchal, Inria).

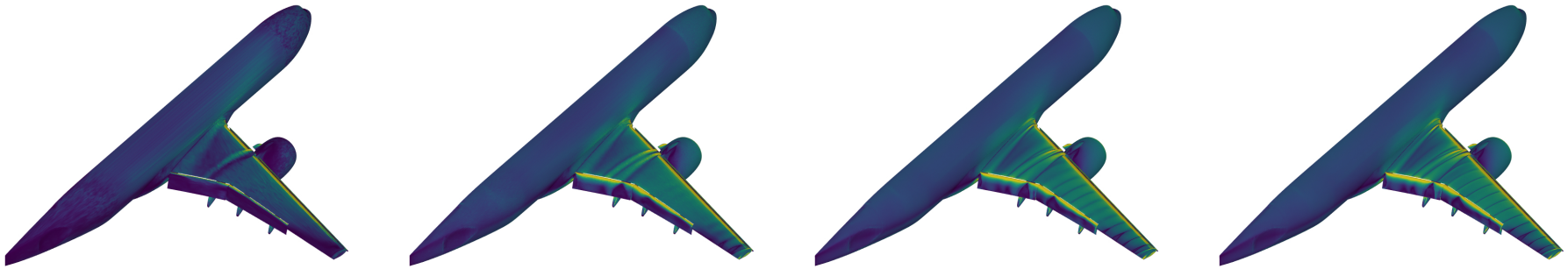


Mesh of Lucy:

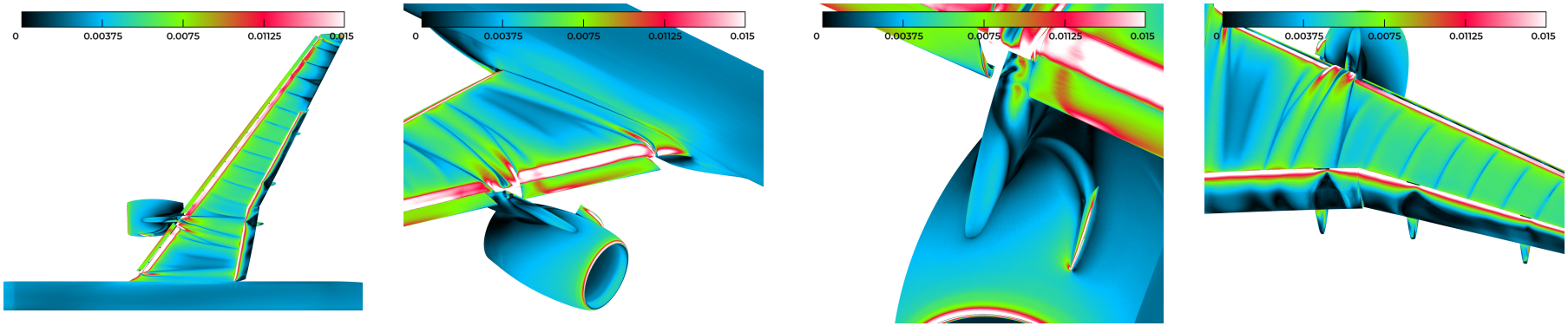
- **14 millions vertices and 28 millions triangles** (642 Mb).
- Mesh opened in less than **1.5 seconds**.
- Rendered in **7.5 seconds** (total time) on a laptop (MacBook Pro 2.6 GHz 6-core Intel Core i7 with 32 Gb of RAM, GPU is AMD Radeon Pro Vega 20 4 Gb).

Scripting tools to generate images among a large set of data

- Data files: save and load rendering options
- Movie mode: generate images from several meshes and possibly solutions



- Sequence mode: generate images from several data files



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

Error estimate for an edge or a line

For an edge, the point-wise error estimate ϵ considered is

$$\epsilon(P_i^{edge}) = \frac{||P_i^{edge} - P_i^{straight}||}{l_{edge}}, \tag{1}$$

where P_i^{edge} is the **control point** lying on the edge, $P_i^{straight}$ is its **equivalent control point on the straight edge** and l_{edge} the length of the edge defined by the extremities.

The error estimate ϵ_{edge} associated to an edge is:

$$\epsilon_{edge} = \max_i \epsilon(P_i^{edge}). \tag{2}$$

The associated `TessLevelOuter` is set by

$$\texttt{TessLevelOuter} = 1 + [5t\epsilon_{edge}],$$

where t is a user integer parameter defining the level of discretization of the line.

In particular, if the element is **straight**, the number of subdivisions is always equal to **1**.

Error estimates inside elements

For a face, the point-wise error estimate ϵ considered is

$$\epsilon(P_i^{face}) = \frac{||P_i^{face} - P_i^{straight}||}{l_{edge}^{max}}, \tag{3}$$

where P_i^{face} is the **control point** lying on the inner part of a face, $P_i^{straight}$ is its **equivalent control point on the straight** face and l_{edge}^{max} is the largest edge length of the straight element.

The error estimate ϵ_{face} associated to the inner part of face is then the largest of its control coefficients error estimates:

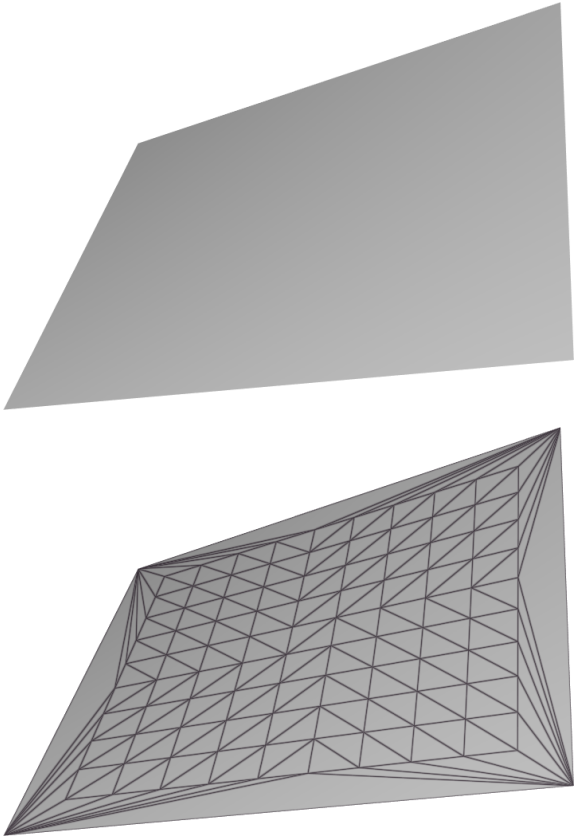
$$\epsilon_{face} = \max_i \epsilon(P_i^{face}). \tag{4}$$

When dealing with quadrilaterals of degree d, another error estimate ϵ_{quad} is considered:

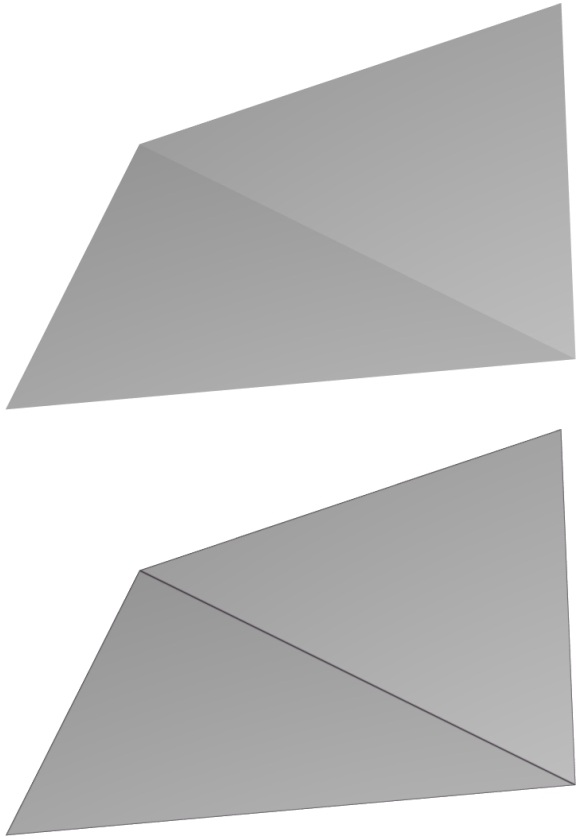
$$\epsilon_{quad} = \frac{|\det(P_{0d} - P_{00}, P_{dd} - P_{00}, P_{d0} - P_{00})|}{||P_{0d} - P_{00}||\ ||P_{dd} - P_{00}||\ ||P_{d0} - P_{00}||}. \tag{5}$$

This error estimate can detect if a quadrilateral is non-planar.

Adaptive tessellation



Naive subdivision



A **non-planar** quadrilateral (even Q^1) needs to be **tessellated** !

Tessellation inside elements

`TessLevelInner` is then set:

$$\texttt{TessLevelInner} = 1 + [5t \max(\epsilon_{edge}, \epsilon_{face}, \epsilon_{quad})], \tag{6}$$

using the same t as with `TessLevelOuter`.

For straight elements, all ϵ equal to 0 and $\texttt{TessLevelOuter} = \texttt{TessLevelInner} = 1$.

- Straight triangle and edge are not divided.
- Straight quadrilateral is divided into two triangles.

$\rightarrow$ Classic way to visualize these straight elements in OpenGL Legacy.

Tessellation controlled by:

- Fast error estimates.
- A user parameter `t` which can interactively be modified.