Polygonal Mesh Adaptation with Cell Size Control for Hypersonic Flow Simulations

Brieuc Praud*† Jérôme Breil*‡ Laurent Muscat§

Abstract

Unstructured meshes are essential in the field of physics computation, both for discretizing complex geometries and for adapting to the underlying physical phenomena. Polygonal meshes constructed from Voronoi diagrams, despite being tougher to construct, sometimes provide several assets over triangular meshes based on Delaunay triangulations. These specific assets depend on the numerical method that is used. High quality meshes are obtained from specific Voronoi diagrams called Centroidal Voronoi Tessellations (CVTs). The process for building such diagrams can also be harnessed to perform mesh adaptation. The purpose of this work is to introduce a formulation of cell size control on Voronoi meshes as a root finding problem relying on a new initialization method for weighted Lloyd's algorithm to speed up the construction of a proper mesh. The method is finally applied to the simulation of a hypersonic flow.

1 Introduction

Mesh adaptation is a highly effective tool that significantly enhances the accuracy of numerical simulations, particularly in the field of computational physics. Adapted unstructured simplicial meshes can be constructed by considering a metric space derived from an error estimator [1]. However, such methods have not much been developed for polygonal meshes in a more general sense in spite of the fact that this type of mesh provides some benefits. Polygonal meshes, particularly those constructed from Voronoi diagrams, enable better representation of complex geometries due to their flexibility and ability to conform to irregular shapes. For the Finite Volume Method (FVM) [7], among other interesting properties, polygonal cells provide a better rotational invariance [15]. This helps reducing numerical artifacts that can arise on anisotropic meshes, especially when simulating highly compressible flows, namely the carbuncle [13]. The carbuncle effect is a well-known challenge in fluid dynamics, leading to non-physical oscillations in solutions, which can significantly impact the reliability of simulations. The main concept of mesh adaptation is to optimize the computational time by equidistributing the discretization error throughout the computational domain. For Computational Fluid Dynamics (CFD) this is achieved using smaller cells in areas where the flow features are complex and in contrast, using larger cells in areas where the flow is more regular. The flow complexity can be detected in several ways such as looking at the magnitude of the gradient and curl of physical fields [17]. These measures provide insights into regions where the flow exhibits significant changes, guiding the adaptation process effectively. This work briefly introduces the concept of Voronoi diagram and its duality with the Delaunay triangulation. Then, the construction process of high quality polygonal meshes from CVTs is explained. Building a CVT relies on Lloyd's algorithm which can also be weighted to perform mesh adaptation given a mesh sizing field. A new initialization procedure to increase the convergence speed of this algorithm is introduced. Moreover, the cell size control is formulated as a root finding problem which is then demonstrated by Brent's root finding method [4]. Such cell size control is of utmost importance in CFD. Finally, the introduced method is showcased on a hypersonic computation by building a mesh sizing field adapted to flow features. The effectiveness of the method is evaluated by analyzing pressure coefficients, which provide insight into the flow dynamics and the impact of the mesh adaptation on the accuracy of the results. This study aims to demonstrate how the proposed approach can not only cut computational costs by decreasing the number of mesh cells but also maintain the resolution of key quantities of interest, contributing to more efficient and accurate simulations.

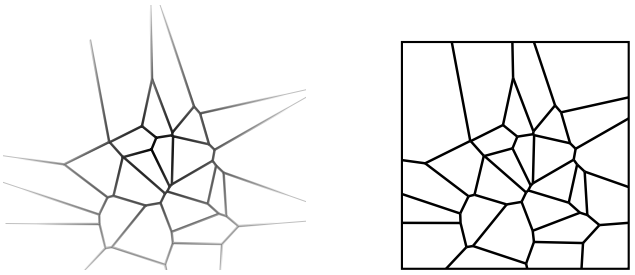
2 Voronoi Diagram and Polygonal Mesh

2.1 Definition and construction

Consider n unique points called "generators" $(\boldsymbol{z}_i)_{i=1}^n$ of the euclidean plane $\mathbb{R}^2$. The Voronoi cell $\mathcal{V}_i$ associated with the generator $\boldsymbol{z}_i$ is defined as:

$$\mathcal{V}_i = \left\{ \boldsymbol{x} \in \mathbb{R}^2 \mid \forall j \in [\![1, n]\!] \setminus \{i\} \ \|\boldsymbol{x} - \boldsymbol{z}_i\| \leq \|\boldsymbol{x} - \boldsymbol{z}_j\| \right\}$$

*I2M, UMR 5295, F-33400 Talence, France
†CEA-CESTA, Le Barp, France. brieuc.praud@cea.fr
‡CEA-CESTA, Le Barp, France. jerome.breil@cea.fr
§CEA-CESTA, Le Barp, France. laurent.muscat@cea.fr



(a) Voronoi diagram (b) Polygonal mesh

Figure 1: Intersection of the Voronoi diagram to form a polygonal mesh

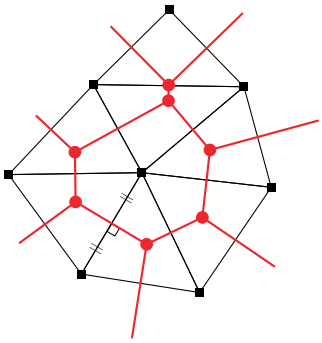


Figure 2: Delaunay-Voronoi duality (Delaunay triangulation in black, Voronoi diagram in red)

This means the Voronoi cell $\mathcal{V}_i$ is defined as the set of all the points of $\mathbb{R}^2$ that are closer to $\boldsymbol{z}_i$ than to all the other generators. The set $(\mathcal{V}_i)_{i=1}^n$ forms a plane tiling called "Voronoi tessellation". This mathematical concept can be generalized to $\mathbb{R}^3$ but all the work presented here takes place within a two-dimensional framework.

Some cells from the Voronoi diagram are unbounded, thus need to be intersected with a boundary to form a proper mesh (see fig. 1).

The Delaunay triangulation of a set of points is commonly used to generate an unstructured mesh. It has the specificity of maximizing the smallest angles of all the constructed triangles. This results, in a sense, in the least skewed triangular mesh based on the input vertices, which is convenient for the numerical schemes.

Given a set of points, the meshing tool proceeds as follows: initially, the Delaunay triangulation of those points is generated using the Bowyer-Watson algorithm. Subsequently, the Voronoi diagram is obtained as the dual of the Delaunay mesh. The generators of the Voronoi diagram are the circumcenters of the triangles and the edges are segments of the perpendicular bisectors of the edges of the triangles (see fig. 2). Finally, the Voronoi diagram is intersected with the boundary of the domain to make a proper polygonal mesh.

The Bowyer-Watson algorithm leverages the fact that it is possible to construct the Delaunay triangulation of $n+1$ points from the triangulation of the first n points with logarithmic time complexity as the insertion of one single point only affects the triangulation locally. Thereby inserting the n points in an enclosing triangle builds the triangulation with $\mathcal{O}(n \log n)$ time complexity. By sorting vertices appropriately to improve the proximity of consecutive points, it is even possible to perform insertion in constant time, thus achieving the triangulation in $\mathcal{O}(n)$ time complexity [12].

2.2 Centroidal Voronoi Tessellation

In the general case, a generator does not coincide with the centroid of the cell it generates. However, this particular case provides good properties for numerical methods, especially for the FVM, because the centroids are aligned with the perpendicular bisectors of the edges which improves the accuracy of fluxes computations. When all the cells in a Voronoi tessellation satisfy this property, the tessellation is said to be centroidal. This is known as CVT.

Given a Voronoi tessellation, one way to converge towards a CVT is to apply Lloyd's algorithm. This algorithm iteratively moves the generators to the centroids of the Voronoi cells they create: at iteration $k+1$, the new position $\boldsymbol{z}_i^{k+1}$ of the generator located at $\boldsymbol{z}_i^k$ is computed as:

$$\boldsymbol{z}_i^{k+1} = \frac{\iint_{\mathcal{V}_i} \begin{pmatrix} x \\ y \end{pmatrix} \mathrm{d}x\mathrm{d}y}{\iint_{\mathcal{V}_i} \mathrm{d}x\mathrm{d}y} \tag{2.1}$$

This algorithm makes the initial Voronoi diagram to converge toward a regular hexagonal tiling [8]. Such kind of mesh offers several advantages over regular cartesian meshes [15]. Among them, the hexagon provide two more edges than the rectangle which allows for a better flux approximation. Furthermore, the mass of the polygon is more evenly distributed around its centroid which makes it intuitively more adapted for FVM where the computed values are supposed to be the average values over the cells.

3 Polygonal Mesh Adaptation

3.1 Weighted Lloyd's algorithm

Lloyd's algorithm can be weighted by a density field ϕ

to prescribe mesh inhomogeneity, thus refinement areas:

$$\boldsymbol{z}_i^{k+1} = \frac{\iint_{\mathcal{V}_i} \phi(x,y) \begin{pmatrix} x \\ y \end{pmatrix} \mathrm{d}x\,\mathrm{d}y}{\iint_{\mathcal{V}_i} \phi(x,y)\,\mathrm{d}x\,\mathrm{d}y} \tag{3.2}$$

One could want to prescribe mesh sizes, therefore it is necessary to find an equation linking a mesh size field μ to the density field ϕ.

For each Voronoi cell a local error, akin to an energy, can be defined as:

$$E_i = \iint_{\mathcal{V}_i} \phi(\boldsymbol{x}) \|\boldsymbol{x} - \boldsymbol{z}_i\|^2 \,\mathrm{d}\boldsymbol{x} \tag{3.3}$$

Let $|\mathcal{V}_i|$ and μ_i denote respectively the volume and a characteristic size of the cell $\mathcal{V}_i$. This way, $|\mathcal{V}_i| \approx \mu_i^2$ so:

$$\begin{aligned} E_i &= \iint_{\mathcal{V}_i} \phi(\boldsymbol{x}) \|\boldsymbol{x} - \boldsymbol{z}_i\|^2 \,\mathrm{d}\boldsymbol{x} \\ &\approx \phi(\boldsymbol{z}_i) {\mu_i}^2 |\mathcal{V}_i| \\ &\approx \phi(\boldsymbol{z}_i) {\mu_i}^4 \end{aligned}$$

Let's consider an optimal CVT. Following [5], by assuming the error equidistribution over the mesh

$$\exists c \in \mathbb{R} \quad \forall i \quad E_i = c$$

therefore:

$$\phi(\boldsymbol{z}_i) \approx \frac{c}{{\mu_i}^4}$$

Consequently, the density field ϕ can be prescribed as the inverse of the fourth power of the mesh sizing field μ that one would like to obtain from the convergence of Lloyd's algorithm:

$$\phi(\boldsymbol{x}) = \frac{1}{\mu(\boldsymbol{x})^4} \tag{3.4}$$

It is worth mentioning that weighted Lloyd's algorithm is insensitive to any scaling of the density field ϕ. The sizing field μ therefore only prescribes inhomogeneity in mesh sizes. The number of cells (i.e. the number of generators) must also be specified to fully determine the actual size of a Voronoi cell.

3.1.1 Area and centroid computation

Area computation

The area $|\mathcal{V}_i|$ of a given polygon $\mathcal{V}_i$ is defined by

$$|\mathcal{V}_i| = \iint_{\mathcal{V}_i} \mathrm{d}x\,\mathrm{d}y \tag{3.5}$$

For a simple polygon, using Green's theorem

$$|\mathcal{V}_i| = \oint_{\partial\mathcal{V}_i} x\,\mathrm{d}y = \sum_{e \in \partial\mathcal{V}_i} \int_e x\,\mathrm{d}y$$

where $\partial\mathcal{V}_i$ denotes the set of the edges of the polygon $\mathcal{V}_i$.

Each edge e is defined by its ends (x_1^e, y_1^e) and (x_2^e, y_2^e). As each edge e is a line segment, the following equality holds:

$$(x_2^e - x_1^e)\,\mathrm{d}y = (y_2^e - y_1^e)\,\mathrm{d}x \tag{3.6}$$

Therefore, the area is given by:

$$|\mathcal{V}_i| = \sum_{e \in \partial\mathcal{V}_i} \int_{x_1^e}^{x_2^e} \frac{y_2^e - y_1^e}{x_2^e - x_1^e} x\,\mathrm{d}x = \frac{1}{2} \sum_{e \in \partial\mathcal{V}_i} \frac{y_2^e - y_1^e}{x_2^e - x_1^e} \left[x^2\right]_{x_1^e}^{x_2^e}$$

Finally, we obtain a manner of computing the area of the polygon known as the trapezoid formula:

$$|\mathcal{V}_i| = \frac{1}{2} \sum_{e \in \partial\mathcal{V}_i} (x_2^e + x_1^e)(y_2^e - y_1^e) \tag{3.7}$$

Unweighted centroid computation

Considering a constant field density, the centroid $\boldsymbol{G}_i$ of polygon $\mathcal{V}_i$ can be computed as:

$$\boldsymbol{G}_i = \frac{\iint_{\mathcal{V}_i} \begin{pmatrix} x \\ y \end{pmatrix} \mathrm{d}x\,\mathrm{d}y}{|\mathcal{V}_i|} \tag{3.8}$$

In a similar way, surface integrals can be converted into line integrals using Green's theorem which can then be expressed as a function of the vertices coordinates:

$$\begin{aligned} \boldsymbol{G}_i |\mathcal{V}_i| &= \frac{1}{2} \iint_{\mathcal{V}_i} \left[\frac{\partial}{\partial x} \begin{pmatrix} x^2 \\ 0 \end{pmatrix} + \frac{\partial}{\partial y} \begin{pmatrix} 0 \\ y^2 \end{pmatrix} \right] \mathrm{d}x\,\mathrm{d}y \\ &= \frac{1}{2} \oint_{\partial\mathcal{V}_i} \begin{pmatrix} x^2\,\mathrm{d}y \\ -y^2\,\mathrm{d}x \end{pmatrix} \\ &= \frac{1}{2} \sum_{e \in \partial\mathcal{V}_i} \int_e \begin{pmatrix} x^2 \frac{y_2^e - y_1^e}{x_2^e - x_1^e}\,\mathrm{d}x \\ -y^2 \frac{x_2^e - x_1^e}{y_2^e - y_1^e}\,\mathrm{d}y \end{pmatrix} \\ &= \frac{1}{6} \sum_{e \in \partial\mathcal{V}_i} \begin{pmatrix} \frac{y_2^e - y_1^e}{x_2^e - x_1^e} \left[x^3\right]_{x_1^e}^{x_2^e} \\ -\frac{x_2^e - x_1^e}{y_2^e - y_1^e} \left[y^3\right]_{y_1^e}^{y_2^e} \end{pmatrix} \\ &= \frac{1}{6} \sum_{e \in \partial\mathcal{V}_i} \begin{pmatrix} ({x_2^e}^2 + x_1^e x_2^e + {x_1^e}^2)(y_2^e - y_1^e) \\ ({y_2^e}^2 + y_1^e y_2^e + {y_1^e}^2)(x_1^e - x_2^e) \end{pmatrix} \end{aligned}$$

Finally,

$$\boldsymbol{G}_i = \frac{1}{6|\mathcal{V}_i|} \sum_{e \in \partial\mathcal{V}_i} \begin{pmatrix} ({x_2^e}^2 + x_1^e x_2^e + {x_1^e}^2)(y_2^e - y_1^e) \\ ({y_2^e}^2 + y_1^e y_2^e + {y_1^e}^2)(x_1^e - x_2^e) \end{pmatrix} \tag{3.9}$$

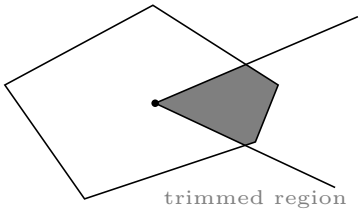
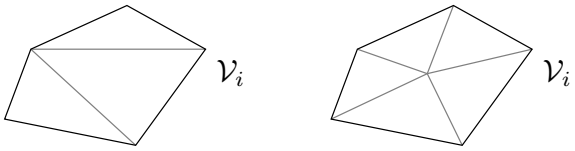


Figure 3: Clipped cell



(a) Ear clipping triangulation of a convex polygon (b) Gravity center-based triangulation of a polygon

Figure 4: Two manners of triangulating a polygon

Weighted centroid computation

More generally, the centroid of the polygon $\mathcal{V}_i$ within a density field ϕ is defined as:

$$\boldsymbol{G}_i = \frac{\iint_{\mathcal{V}_i} \phi(x,y) \begin{pmatrix} x \\ y \end{pmatrix} \mathrm{d}x\,\mathrm{d}y}{\iint_{\mathcal{V}_i} \phi(x,y)\,\mathrm{d}x\,\mathrm{d}y} \tag{3.10}$$

Some forms of density field could allow for an explicit computation of the centroid. However, if one wants to be able to compute centroids in a general case and with any order of precision, numerical integration with quadrature rules must be performed. To do so, a polygon can be divided into triangles based on its gravity center as long as this polygon is a star-convex domain with this center as a vantage point. As Voronoi cells are convex, this property in ensured. One exception is for the Voronoi cells that are clipped by the boundary. However, the clipping is done by considering the discretized boundary for which each segment endpoint is considered as a generator. By doing so, each cell can only be clipped in a V-shaped manner so that the generator can still be used as a vantage point (see fig. 3).

An ear clipping triangulation such as illustrated by fig. 4a could lead to skewed triangles that could reduce the numerical performances, whereas triangulation based on the centroid (fig. 4b) gives elements of better quality overall. Numerical integration is thus performed using this second option, assuming integration points will be more evenly distributed over the polygon.

Once the triangulation is done, usual quadrature rules over triangles can be used, then the quadrature rule over the polygon is given as the sum of the quadrature formulas of each individual triangle.

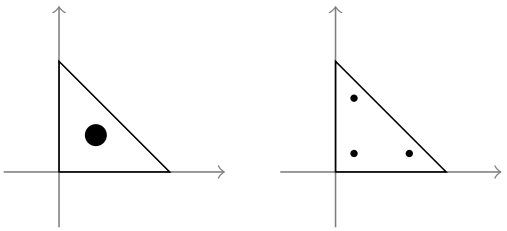


Figure 5: Quadrature rules on a reference triangle

In the following, two symmetric quadrature formulas are used: the centroid rule (1-point rule) and a 3-point rule [16]. They are both illustrated on a reference element in fig. 5, with a size proportional to their associated weight. If not stated otherwise, the 3-point quadrature is used.

The integral of a given function f (for example $f(x,y) = \phi(x,y)\,x$, in the weighted centroid computation) is approximated by:

$$\begin{aligned} \iint_{\mathcal{V}_i} f(x,y)\,\mathrm{d}x\,\mathrm{d}y &= \sum_{\Delta \in \mathcal{V}_i} \iint_{\Delta} f(x,y)\,\mathrm{d}x\,\mathrm{d}y \\ &\approx \sum_{\Delta \in \mathcal{V}_i} \sum_{j=1}^{N_q} \omega_j f(x_j, y_j) \end{aligned}$$

where Δ stands for the triangles that forms the polygon, N_q is the number of quadrature points over each triangle and $(\omega_j)_{j \in [\![1,N_q]\!]}$ are the weights associated to the quadrature points $(x_j, y_j)_{j \in [\![1,N_q]\!]}$.

3.1.2 Leveraging Lloyd's algorithm for mesh adaptation

The theory introduced above is now showcased by a numerical experiment on the square domain $[-1,1] \times [-1,1]$ in which one places 200 generators.

The mesh sizing field μ is chosen as $\mu = kd+1$ with d the euclidean distance field with respect to the center of the geometry. and k is a parameter which corresponds to a growth rate of the cell sizes with respect to the distance d. The one and only purpose of adding 1 is to avoid the division by 0 when inverting to get the density field. Note that one could choose any constant in place of 1. However, this is equivalent to scaling the parameter k (see section 3.1).

Meshes shown in this section and in the following are meshes that are obtained from convergence of Lloyd's algorithm, considering that the convergence is reached when the smallest cell size l_{min} and the largest cell size l_{max} are settled with tolerance of 1%. The con-

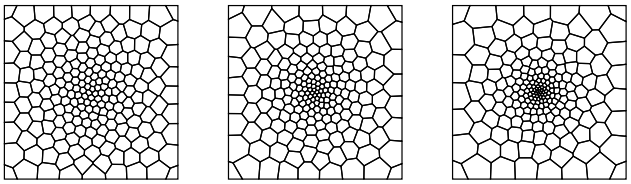


Figure 6: Effect of parameter k on mesh inhomogeneity

vergence analysis will follow.

The effect of parameter k is illustrated in fig. 6. The bigger the parameter, the farthest from homogeneity the mesh is ($k = 0$ resulting in a uniform mesh). In other words, increasing k lowers l_{min} and increases l_{max}. Here the mesh size is defined as the square root of its area.

It should be mentioned that no clear relation can be established between the smallest cell size l_{min} and the parameter k. Now, a manner of enforcing a specific l_{min} value will be detailed.

3.2 Enforcing a cell size using Brent's method

3.2.1 Method overview

Considering a density field ϕ, monotonic with respect to a parameter k. One would like to enforce the smallest cell size $l_{min,target}$ given a number of generators N_g. The goal is thus to find the parameter k leading to $l_{min} = l_{min,target}$. This is equivalent to finding the root of the following function:

$$\begin{array}{rccl} f_{N_g}: & [k_{min}, k_{max}] & \to & \mathbb{R} \\ & k & \mapsto & l_{min}(k) - l_{min,target} \end{array}$$

with $l_{min}(k) = \min_{\mathcal{V}_i \in \mathcal{V}(N_g,k)} \sqrt{|\mathcal{V}_i|}$ and $[k_{min}, k_{max}]$ two bounds for k to achieve l_{min}.

This root can be located thanks to Brent's method [4]. The algorithm combines the secant method, the inverse quadratic interpolation and the bisection method to efficiently locate the root of the function f_{N_g} [14]. The key idea is, at each iteration, to estimate which one of the three methods should bring the algorithm closer to the root. The secant method and the inverse quadratic interpolation are methods that converge quickly. However, a bad initialization or a non continuous derivative can lower their orders of convergence. These two methods can even diverge. In those cases, Brent's method uses bisection to avoid divergence, ensuring the stability of the process of searching k. Note that this method is very general and could be applied the other way around: for a given mesh sizing field μ one could consider the function $f_\mu : N_g \mapsto l_{min}(N_g) - l_{min,target}$ to find the number

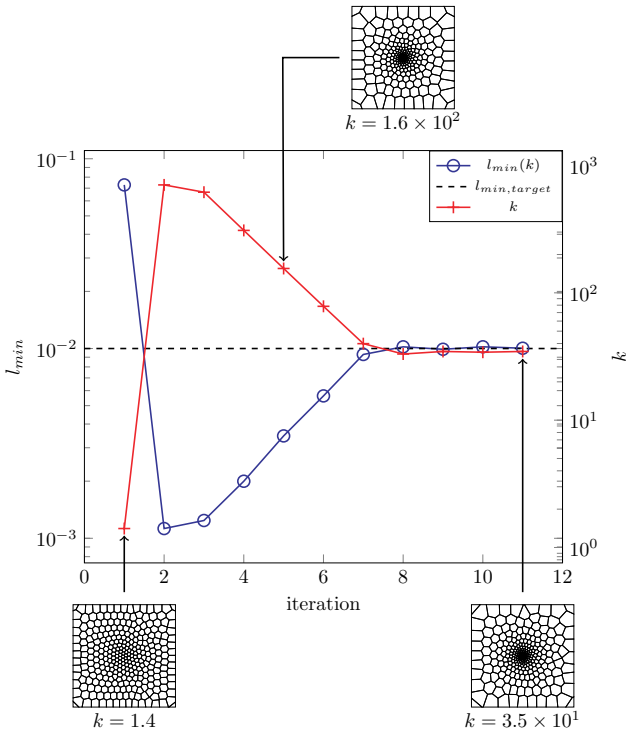


Figure 7: Convergence of Brent's algorithm toward a mesh featuring $l_{min} = l_{min,target}$

of generators required to achieve a given cell size. To be comprehensive, Brent's algorithm deals with real values so that one must round the value of N_g given by the algorithm to the nearest integer to get the actual number of generators.

3.2.2 Brent's method convergence analysis

Figure 7 shows the behavior of Brent's algorithm on the square test case. In red (cross-marked) the parameter k determined by the algorithm at each iteration. In blue (circle-marked), the smallest cell size l_{min} obtained with the parameter k. At iterations 1 and 2, the algorithm evaluates the function f_{N_g} at the bounds. Then the algorithm converges quickly toward a value close from the target, with around 5 iterations. However, 4 iterations are still necessary in order to reach the objective with a 1 % precision. This is due to both the fact that the function f_{N_g} is not sensitive to small changes of k and the randomness introduced when placing the generators at the start of the process. As a matter of a fact, this random placing makes Lloyd's algorithm to converge differently so that for two very close values of k, the smallest cell size could be obtained by the smallest k parameter whereas for larger differences, that is the opposite.

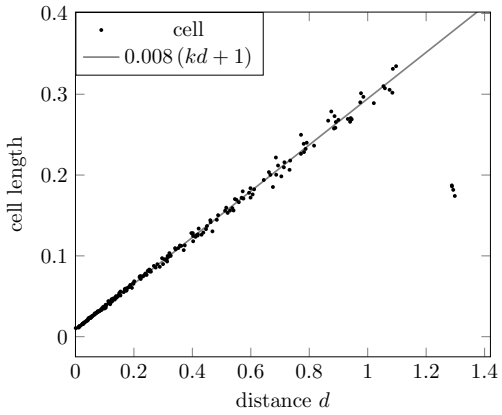


Figure 8: Cell size with respect to their distance to the center of the square

Figure 8 shows the cell sizes that are obtained with respect to the distance (measured from the center of the square to the centroid of the cell) as a point cloud plot after Brent's optimization. The theoretical cell size is also given as a line for comparison. It can be seen that all cells have sizes close to their theoretical value except for the four corner cells which are clipped by the meshing tool. Another fact that is worth mentioning is that the value $d = 0$ plugged in the line equation does not give the minimum cell size of 1×10^{-2} because the smallest cell is slightly off the origin.

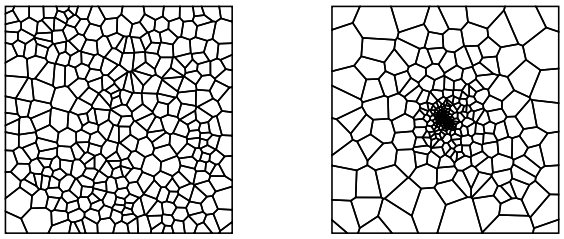
3.3 Speeding up Lloyd's algorithm with rejection sampling

3.3.1 Overview of the rejection sampling

At each iteration of Brent's algorithm, the convergence of Lloyd's algorithm must be obtained so that the process of finding the right k parameter can become computationally heavy. In order to speed up the convergence of Lloyd's algorithm, it is possible to place the N_g generators in the domain based on a probability distribution derived from the density field ϕ.

Using rejection sampling, the generators of the initial mesh are already closer to their converged position (see fig. 9b) than with a uniform initialization (see fig. 9a).

From the density field ϕ, which corresponds to a scaled probability density function, one can sample the Voronoi generators by rejection sampling. This method relies on the fact that for a pair of independent random variables (U, V) both sampled uniformly, the probability distribution f_X of a given random variable X is the one of U under the condition $\{V \leq f_X(U)\}$. This method is described by algorithm 3.1 on the rectangular domain $\Omega = [x_{min}, x_{max}] \times [y_{min}, y_{max}]$ for the probability density function



(a) Uniform initialization (b) Initialization by the rejection sampling method

Figure 9: Comparison between the two initialization methods

$$f : \Omega \longrightarrow [0,1], \quad \boldsymbol{x} \longmapsto \frac{\sqrt{\phi(\boldsymbol{x})}}{\iint_\Omega \sqrt{\phi(\boldsymbol{x})}}$$

The uniform disribution over the interval $[\alpha, \beta]$ where $\alpha, \beta \in \mathbb{R}$ is denoted as $\mathcal{U}[\alpha, \beta]$.

Algorithm 3.1. rejection sampling

Require: Bounding box $[x_{min}, x_{max}] \times [y_{min}, y_{max}]$ for the computational domain
Require: Density field ϕ reaching a maximum ϕ_{max}
Require: Number N_g of generators to sample
Ensure: Sampling of the computational domain according to a given density field

```
function REJECTION_SAMPLING
    n_g = 0
    while n_g < N_g do
        Sample x according to U[x_min, x_max]
        Sample y according to U[y_min, y_max]
        Sample z according to U[0, sqrt(φ_max)]
        if sqrt(φ(x, y)) > z then
            Take (x, y) as a generator
            Increment n_g
        end if
    end while
end function
```

Note that the normalization of the field ϕ is actually unnecessary to apply the rejection sampling algorithm. The sampling is made from the field $\sqrt{\phi}$, not directly from the field ϕ because the sampling is made according to the density of generators per unit area. One cell of length μ contains 1 generator (by definition of a Voronoi cell) so that the sampling density is defined as $\frac{1}{\mu^2}$ which is equal to $\sqrt{\phi}$.

Rejection sampling method thus allows to sample from a given probability density function based solely

on uniform samplings, which is convenient from a numerical perspective. The benefit of this initialization by rejection sampling is presented in the following.

3.3.2 Rejection sampling evaluation

In order to measure the convergence of Lloyd's algorithm, two errors are introduced:

$$\text{size error:} \quad \sqrt{\frac{1}{N_g} \sum_{\mathcal{V}_i \in \mathcal{V}} \left(1 - \frac{\mu_\text{r}}{\mu_\text{target}}\right)^2}$$

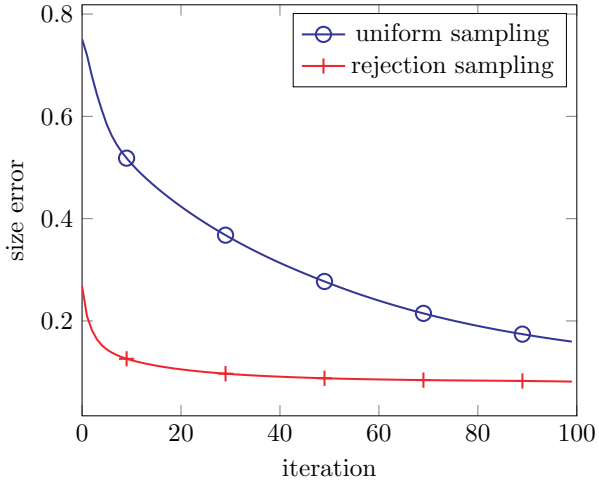
$$\text{shape error:} \quad \sqrt{\frac{1}{N_g} \sum_{\mathcal{V}_i \in \mathcal{V}} \left(1 - \frac{s_\text{min}}{s_\text{max}}\right)^2}$$

where $\mathcal{V}$ is the set of all Voronoi cells $\mathcal{V}_i$, μ_target is the expected cell size (i.e. square root of the area) of $\mathcal{V}_i$ while μ_r is the actual cell size of this cell. The lengths s_min and s_max are the length of the smallest and the biggest sides of the cell, respectively.

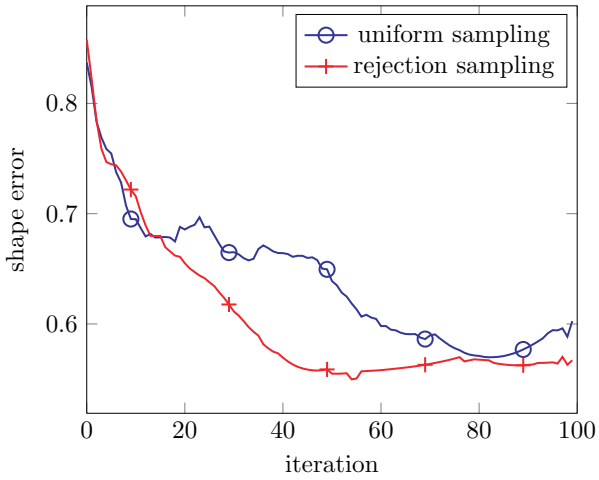
The size error measures how well the constructed mesh respects the prescribed cell size whereas the shape error is a measure of the quality of the elements. This error is lowered when the cells are regular polygons. However, this error cannot go to zero because each side of a cell is subjected to a slightly different value of density.

The non-uniform initialization by rejection sampling increases the convergence speed of Lloyd's algorithm, which is shown by fig. 10. This preliminary step has a negligible computational cost as long as the density probability function does not feature a highly localized maximum. In this case, almost all random samples would be rejected. On one hand fig. 10a highlights the ability of the rejection sampling method to place generators near their final location so that the convergence of the cell sizes is really quick (from 5 to 20 iterations according to the plot) whereas the uniform sampling leads to a slower and slower convergence throughout the step so that at 100 Lloyd's algorithm iterations, the size error is still fairly high. On the other hand, fig. 10b provides mixed results as for the first 20 iterations, both methods seems to give the same result in term of cell shape. However, from 20 to 40 Lloyd's iterations, the mesh initialized by the rejection sampling method maintains its convergence rate before reaching a steady state while the mesh initialized uniformly has a harder time converging in shape. This phenomenon could be explained by the fact that in that last configuration, cells are constantly drifting toward the center of the square so that their shape cannot settle correctly until the size convergence is achieved.

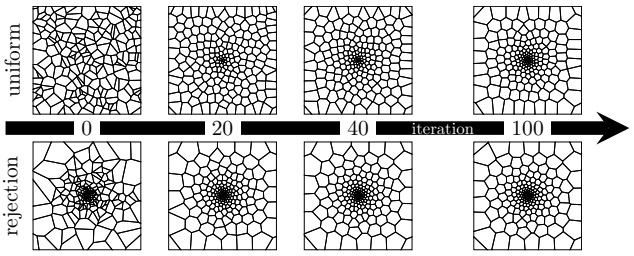
The quadrature rule should also be chosen with care. As illustrated by fig. 11b, the quadrature method does not seem to have any impact on the convergence



(a) size error convergence

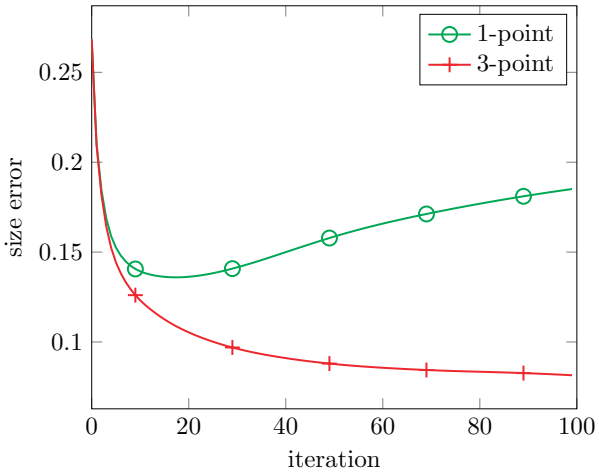


(b) shape error convergence

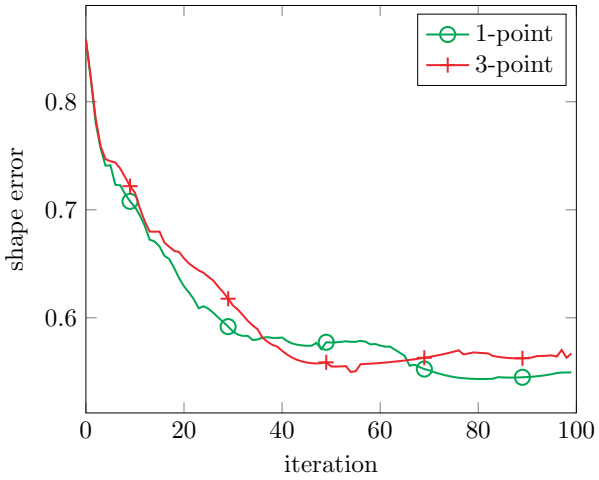


(c) meshes at convergence milestones

Figure 10: Impact of initialization on Lloyd's algorithm convergence



(a) size error convergence



(b) shape error convergence

Figure 11: Impact of quadrature method on Lloyd's algorithm convergence

in shape. However, considering fig. 11a, the 1-point quadrature rule seems to exhibit both slower convergence and a larger size error than the 3-point quadrature rule. This is explained by the fact that the method does not seem to achieve a linear cell size scaling, thus the large error. However, the quadrature method complexity also increases the computational cost of each individual iteration. A tradeoff is thereby to be found. As the order does not seem to have any significant impact on the convergence for the first 5 to 10 iterations, one idea would be to increase the precision of the quadrature throughout the iterations, in a similar way than [18].

To conclude this convergence analysis, two criteria have been introduced to measure how well the constructed mesh corresponds to what one could expect

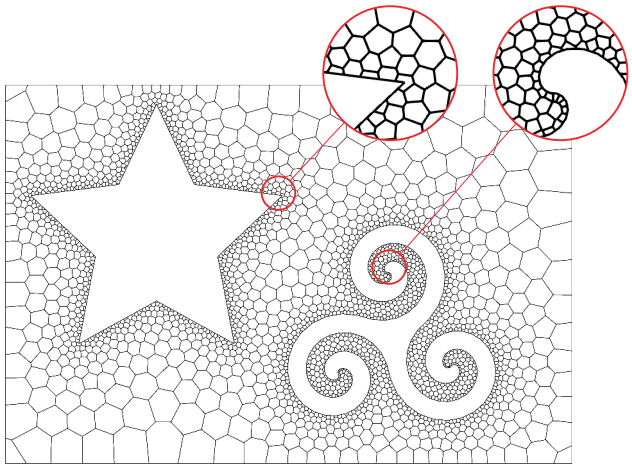


Figure 12: Triskelion and star mesh

from a computational mesh. This is to our knowledge a new approach which gives a more meaningful analysis of the convergence than the CVT energy traditionally used to perform this kind of analysis [11]. This criteria shows the clear advantage of using rejection sampling algorithm as an initial guess for Lloyd's algorithm.

This initialization could be coupled with a more efficient algorithm than Lloyd's (L-BFGS for example [11]) but this coupling still needs to be investigated.

In order to illustrate the versatility of the method, a more involved geometry is shown as fig. 12.

4 Application to hypersonic flows

This adaptation method based on weighted Lloyd's algorithm can be used to adapt a mesh for the simulation of hypersonic flows. In this case, it is relevant to use flow features to compute the sizing field μ or, equivalently, the density field ϕ.

4.1 Sizing field computation

The polygonal meshing tool has been coupled with a hypersonic flow solver to build an adaptation loop process. It works as follows: based on a uniform Voronoi mesh of the computational domain, a distance-to-geometry field is computed for each cell by solving a form of the eikonal equation. This method features several benefits against other simpler methods [2]. From this computation, a near-wall adaptation can be made as a first adaptation step. Subsequently, a first hypersonic computation can be done on this mesh. From this computation, flow features can be extracted to build a new density field. This adaptation aims to improve the global numerical solution therefore the steps of solving the flow then computing a new mesh can be done several times in order to compute the flow precisely.

There is a boundless number of manners to adapt a mesh to the flow. As an example [17] construct a linear combination of the temperature, pressure, velocity and density fields then adapt the mesh according to the gradient of this new field. Another method that is investigated by [17] is to adapt according to both the gradient and curl of the velocity field.

Here the goal is to precisely compute aerodynamic effects on the wall therefore the choice made here is to adapt the mesh according to two quantities: the wall distance and the gradient of the velocity field.

From the wall distance approximation d, one can define a mesh sizing field as $\mu_d = k_d d + 1$ with k_d a user defined-parameter. As small cells must be used to resolve strong gradients, one can choose another sizing field $\mu_g = k_g \varphi(g) + 1$ where k_g is another user-defined parameter,

$$g = \frac{1}{1 + \|\nabla \boldsymbol{V}\|}$$

with $\boldsymbol{V}$ the velocity field and φ is the affine transformation :

$$\begin{aligned} \varphi : [g_{min}, g_{max}] &\to [d_{min}, d_{max}] \\ g &\mapsto \frac{d_{max} - d_{min}}{g_{max} - g_{min}}(g - g_{min}) + d_{min} \end{aligned}$$

where d_{min} and d_{max} are respectively the minimum and maximum values of field d and g_{min} and g_{max} are respectively the minimum and maximum values of field g defined above.

This affine transformation is used so that parameters k_d and k_g of the same order of magnitude gives μ_d and μ_g of the same order of magnitude.

Lloyd's density field ϕ is then computed from a new mesh sizing field $\mu = \min(\mu_d, \mu_g)$ from the formula derived previously: $\phi = \frac{1}{\mu^4}$. In practice, it has been witnessed that strong variations of field ϕ (within a shock wave typically) leads to too steep mesh gradation. A corrective, similar to a diffusion process, has thus been introduced. It works by updating the field ϕ on each cell by the average of the field value on the mesh neighborhood, including the cell itself. This process can be repeated as many times as necessary, say m times. However, it is not required and even not beneficial to diffuse in areas where the variations of ϕ are small. As a matter of a fact, precision and so the benefit from mesh adaptation are lost in the process. To solve this issue, a conditional process has been introduced: if the diffusion process induces a relative variation δ over the cell then the diffusion process is performed. Otherwise, the field ϕ is kept unchanged. The parameters used in the following have been chosen empirically and are $m = 5$ and $\delta = 500\,\%$.

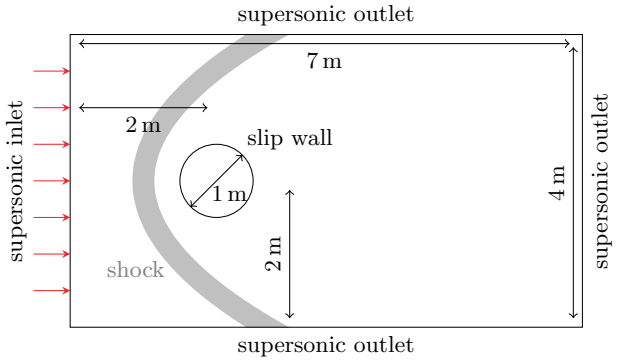


Figure 13: Hypersonic cylinder test case

4.2 Evaluation of the method on a hypersonic computation

In order to measure the benefit from this mesh adaptation, it is interesting to evaluate aerodynamic quantities by introducing a physical test case. The case of a hypersonic cylinder (see fig. 13) is interesting as the flow creates a detached shock wave along which the gradient adaptation feature of the introduced method is showcased.

To evaluate the method quantitatively, the pressure coefficient C_p is also introduced:

$$C_p = \frac{p - p_\infty}{\frac{1}{2}\rho_\infty V_\infty^2}$$

where ρ, p and V are respectively the density, the pressure and the velocity norm of the fluid, the "∞" index corresponding to the upstream quantities.

This pressure coefficient is important to compute accurately as it is indicative of near-wall aerodynamic effects. Euler equations are solved for this test case:

$$\left\{ \begin{array}{rcl} \frac{\partial \rho}{\partial t} + \boldsymbol{\nabla} \cdot (\rho \boldsymbol{V}) &=& 0 \\ \frac{\partial (\rho \boldsymbol{V})}{\partial t} + \boldsymbol{\nabla} \cdot (\rho \boldsymbol{V} \otimes \boldsymbol{V}) &=& -\boldsymbol{\nabla} p \\ \frac{\partial (\rho E)}{\partial t} + \boldsymbol{\nabla} \cdot (\rho E \boldsymbol{V}) &=& \boldsymbol{\nabla} (p \boldsymbol{V}) \end{array} \right.$$

where E is the total specific energy. This set of equations describes the flow of a non-viscous fluid. A comprehensive specification of the test case as well as the numerical scheme that is used are presented in table 1.

To be able to evaluate the pressure coefficient on the adapted meshes, one first needs to perform a mesh convergence analysis. Pressure coefficients for four uniform meshes are thus presented by fig. 15. The pressure coefficients are plotted against the curvilinear abscissa around the cylinder with the origin taken at the stagnation point. It can be observed that convergence is obtained with 15 000 cells as the pressure

Mach number M_∞	17.605
density ρ_∞	$1 \times 10^{-3}\,\mathrm{kg\,m^{-3}}$
velocity V_∞	$5 \times 10^{3}\,\mathrm{m\,s^{-1}}$
perfect gas : heat capacity ratio γ	1.4
Godunov-type FVM, HLL flux	

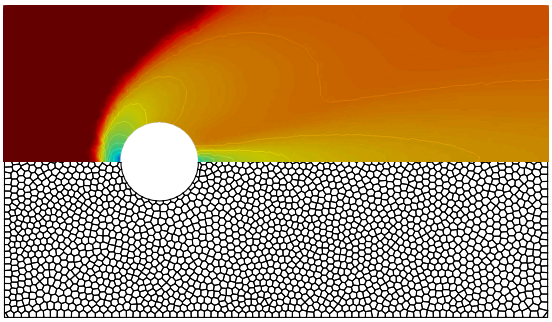
Table 1: Test case specification

coefficient curve matches the one obtained with 30 000 cells really closely. The model from Lester Lees [10] is also given for reference. The large difference to this model observed for high curvilinear abscissas is due to the fact that this model is designed to be more accurate near the stagnation point. The uniform meshes with $n = 3750$ and $n = 15\,000$ cells are represented with their corresponding Mach fields in fig. 14a and fig. 14b respectively.

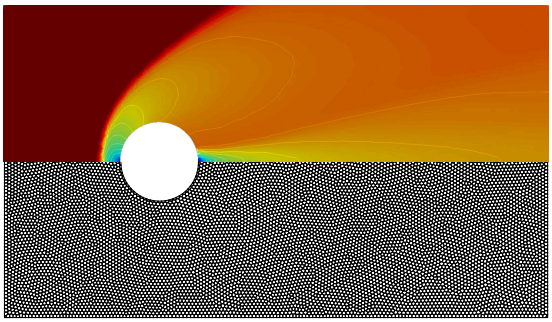
It has been observed in fig. 15 that 3750 cells are far from enough to obtain a decent pressure coefficient if the mesh is uniform ($k_d = 0.000$). As the uniform 30 000 cells mesh features a near-wall cell size of 0.04 m, one can use Brent's algorithm introduced earlier to find the parameter k_d which prescribes this same smaller cell size but with only $n = 3750$ cells. By doing so, Brent's algorithm finds $k_d = 0.035$ (see mesh in fig. 14c) which gives results much closer to the reference than the low resolution uniform mesh (fig. 16). Gradient-based adaptation has also been applied (see mesh in fig. 14d). However, no noticeable improvement can be seen on the pressure coefficient in fig. 16, probably because, as the whole adaptation process is done with a fixed number of cells, adapting according to gradients leads to less cells near the wall. Using more cells, the velocity gradient-based adaptation can be useful as shown by fig. 17b. As a matter of a fact, it allows for a better resolution of both the shock wave upstream and the compressional wave downstream than the adaptation based solely on the wall distance (fig. 17a) for a given number of cells ($n = 30\,000$ in this case) because the well placed small cells reduce the numerical diffusion in these specific areas. Figure 18 is also given to illustrate better the capabilities of our meshing tool.

5 Conclusion

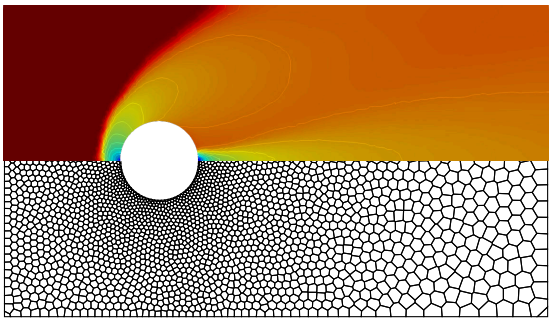
The process of constructing a Voronoi diagram from random "generators" has been discussed. It relies on first the construction of a Delaunay triangulation then proceeding by duality. Voronoi diagrams are unbounded thus need to be intersected with the geometry boundaries to form a proper mesh. A weighted version of Lloyd's algorithm is used to obtain a high quality mesh and to perform mesh adaptation. It relies on a mesh



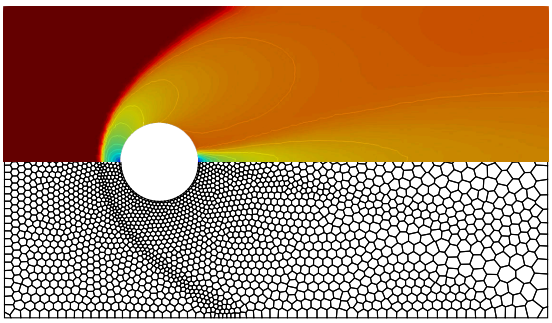
(a) 3750 cells, uniform



(b) 15 000 cells, uniform

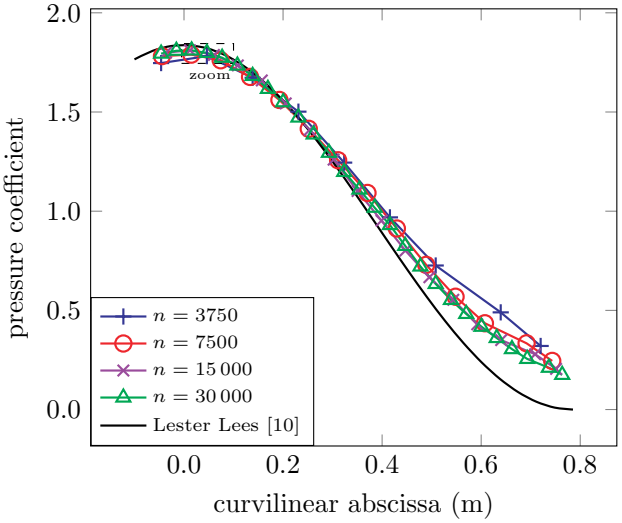


(c) 3750 cells, wall distance adaptation ($k_d = 0.035$)

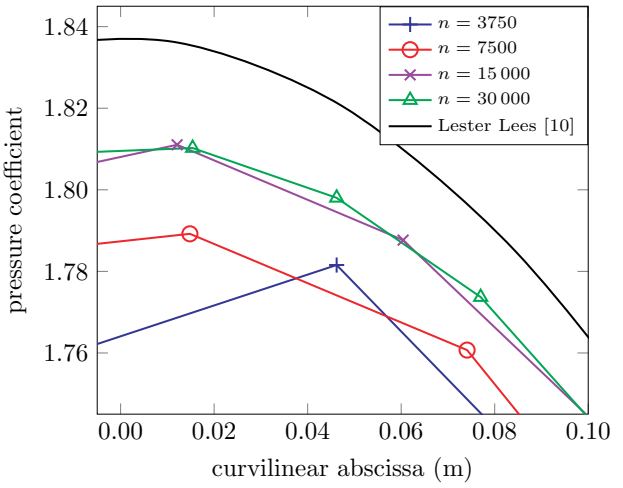


(d) 3750 cells, wall distance and gradient adaptation

Figure 14: Mach field for several meshes (logarithmic colorscale)

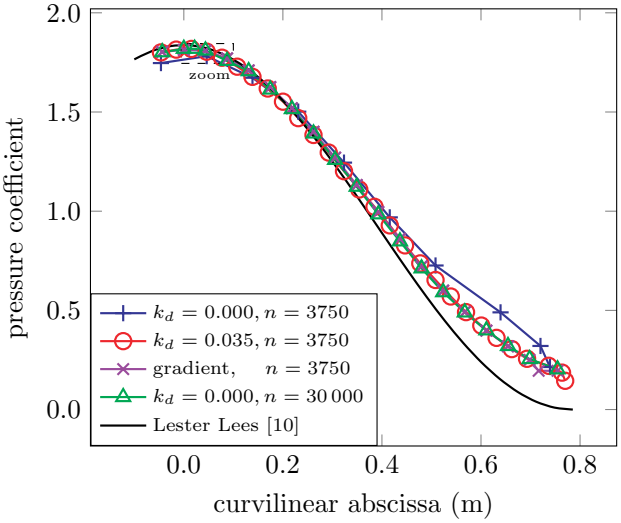


(a) Pressure coefficient along the curvilinear abscissa

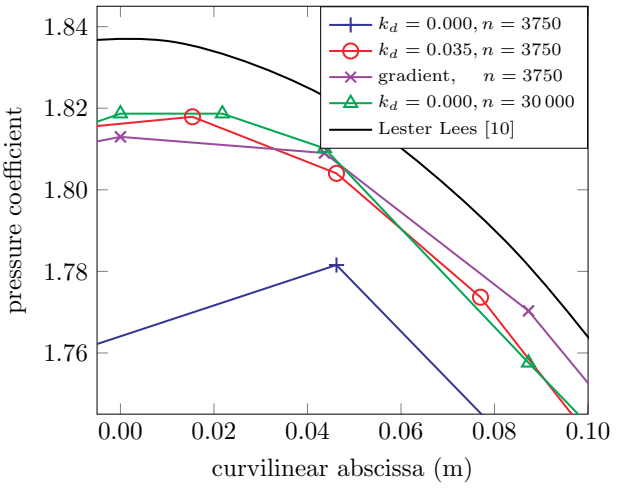


(b) Pressure coefficient along the curvilinear abscissa: zoom

Figure 15: Mesh convergence analysis on the pressure coefficient

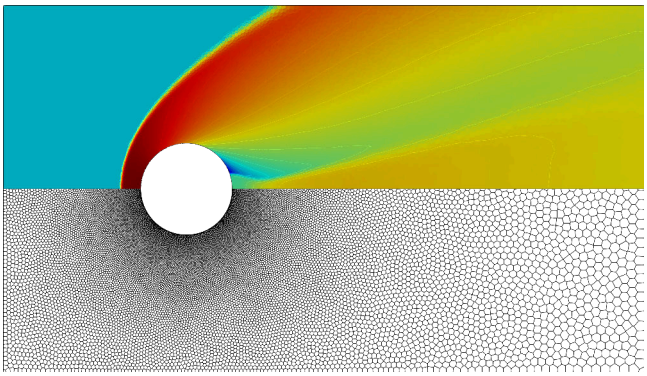


(a) Pressure coefficient along the curvilinear abscissa

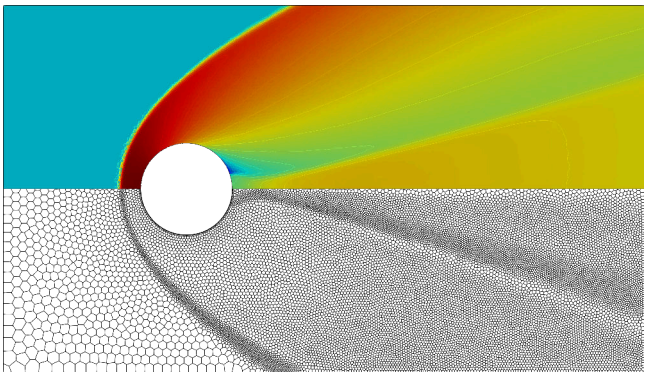


(b) Pressure coefficient along the curvilinear abscissa: zoom

Figure 16: Pressure coefficient for several growing factors k_d



(a) Adaptation according to wall distance



(b) Adaptation according to wall distance and velocity gradient

Figure 17: Qualitative comparison of the pressure field obtained on two distinct meshes (logarithmic colorscale)

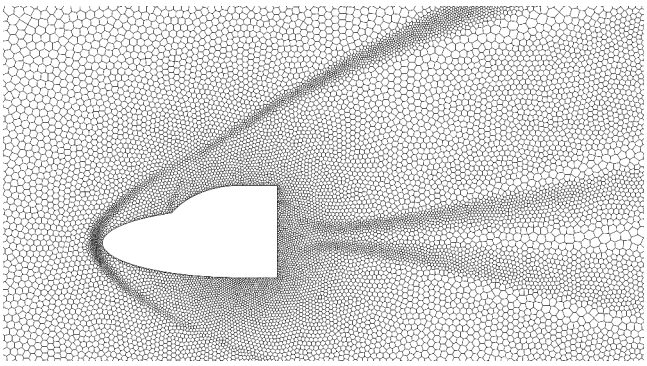


Figure 18: Mesh adaptation on a double ellipsoid geometry

density field which can be deduced from a more handy mesh sizing field using the equidistribution principle. Although in some cases an explicit formula can be derived for the weighted centroid, numerical integration must be used to be able to perform mesh adaptation from any given field. In this work, mesh adaptation has been done according to both wall distance and velocity gradient. It has been shown that this adaptation globally allows for a better resolution of flow features. Most of all, this work aimed at better control over the adaptation by introducing a root finding problem in order to reliably prescribe a cell size. As the process can become computationally heavy, some options were proposed. The initialization of Lloyd's algorithm by rejection sampling is a quick, simple yet efficient manner of speeding the process up by highly lowering the number of steps required for Lloyd's algorithm convergence. The quadrature precision could also be started low then increased throughout the convergence process of Lloyd's algorithm to reduce the computational cost of each individual iteration without increasing the total number of steps of the process. Enforcing a given cell size have several applications in CFD, especially in a Large Eddy Simulation (LES) context. The usefulness of mesh adaptation has been demonstrated on pressure coefficient computations for which similar results were obtained on both a uniform, highly refined mesh and a much coarser, but adapted one. To increase the computational efficiency of Lloyd's algorithm, gradient descent methods could be employed as an alternative to the iterative process applied to each individual generator [3]. Furthermore, fancier error estimators could be constructed in order to target more efficiently the areas that needs mesh refinement [9]. Finally, despite the fact that all the work presented here has been done in a two-dimensional framework, this can be extended to the three-dimensional case (see [6] for 3D centroidal Voronoi and [15] for evaluation of CFD problems on 3D Voronoi meshes).

Acknowledgements

triskelion geometry : *AnonMoos, Public domain, via Wikimedia Commons*

References

[1] F. Alauzet and L. Frazza, *Feature-based and goal-oriented anisotropic mesh adaptation for rans applications in aeronautics and aerospace*, Journal of Computational Physics, 439 (2021), p. 110340.

[2] A. Bouchard, *Wall distance evaluation via eikonal solver for RANS applications*, École Polytechnique, Montreal (Canada), 2017.

[3] D. P. Bourne and S. M. Roper, *Centroidal power diagrams, lloyd's algorithm, and applications to optimal location problems*, SIAM Journal on Numerical Analysis, 53 (2015), pp. 2545–2569.
[4] R. P. Brent, *Algorithms for minimization without derivatives*, IEEE Transactions on Automatic Control, 19 (1974), pp. 632–633.
[5] Q. Du and D. Wang, *Tetrahedral mesh generation and optimization based on centroidal Voronoi tessellations*, International journal for numerical methods in engineering, 56 (2003), pp. 1355–1373.
[6] ———, *The optimal centroidal Voronoi tessellations and the Gersho's conjecture in the three-dimensional space*, Computers & Mathematics with Applications, 49 (2005), pp. 1355–1373.
[7] B. Einfeldt, *On godunov-type methods for gas dynamics*, SIAM Journal on numerical analysis, 25 (1988), pp. 294–318.
[8] A. Gersho, *Asymptotically optimal block quantization*, IEEE Transactions on information theory, 25 (1979), pp. 373–380.
[9] H. Jasak and D. A. Gosman, *Residual error estimate for the finite-volume method*, Numerical Heat Transfer: Part B: Fundamentals, 39 (2001), pp. 1–19.
[10] L. Lees, *Hypersonic flow*, Journal of Spacecraft and Rockets, 40 (2003), pp. 700–735.
[11] Y. Liu, W. Wang, B. Lévy, F. Sun, D.-M. Yan, L. Lu, and C. Yang, *On centroidal voronoi tessellation—energy smoothness and fast computation*, ACM Transactions on Graphics (ToG), 28 (2009), pp. 1–17.
[12] C. Marot, J. Pellerin, and J. Remacle, *One machine, one minute, three billion tetrahedra*, International Journal for Numerical Methods in Engineering, 117 (2019), pp. 967–990.
[13] L. Muscat, T. Marques, and J. Breil, *Mitigating carbuncle effects using Voronoi hybrid meshes*, in AIAA Aviation forum and ascend 2024, 2024, p. 4506.
[14] W. H. Press, S. A. Teukolsky, V. W. T, and F. B. P, *Numerical Recipes in Fortran 77*, Cambridge University Press, 1992.
[15] E. Sozer, A. S. Ghate, G. K. Kenway, M. F. Barad, V. C. Sousa, and C. C. Kiris, *Evaluation of Voronoi meshes for large eddy simulations of high lift aerodynamics*, in AIAA SciTech 2023 Forum, 2023, p. 0255.
[16] G. Strang and G. Fix, *An Analysis of the Finite Element Methods, and Engineering*, SIAM, 2008.
[17] S. Tian and Z. Peng, *Mesh adaptation for simulating lateral jet interaction flow*, Aerospace, 9 (2022), p. 781.
[18] J. Tournois, P. Alliez, and O. Devillers, *2d centroidal voronoi tessellations with constraints*, Numerical Mathematics: Theory, Methods and Applications, 3 (2010), pp. 212–222.