AIAA XX-XX January Scitech, 2019, San Diego, California

Parallel high-order anisotropic meshing using discrete metric tensors

Dirk Ekelschot[*], Marco Ceze[†], Scott M. Murman[‡]
NASA Ames Research Center, Moffett Field, CA, USA

This paper presents a metric-aligned meshing algorithm that relies on the L_p-Centroidal Voronoi Tesselation approach. A prototype of this algorithm was first presented at the Scitech conference of 2018[1] and this work is an extension to that paper. At the end of the previously presented work, a set of problems were mentioned which we are trying to address in this paper. First, we show a significant improvement in code performance since we were limited to present relatively benign (analytical) test cases. Second, we demonstrate here that we are able to rely on discrete metric data that is delivered by a Computational Fluid Dynamics (CFD) solver. Third, we demonstrate how to generate high-order curved elements that are aligned with the underlying discrete metric field.

I. Introduction

Mesh generation and adaptation are the most time-consuming steps when performing Computation Fluid Dynamics (CFD) simulations around complex geometries which has been specifically highlighted in NASA's CFD Vision 2030 Study.[2] Mesh refinement is usually required when anisotropic flow features like shear layers or shock waves are dominating the flow and need to be predicted within a certain bound of accuracy in order to capture the aerodynamic forces around a vehicle for example. Our research group focusses on solving compressible flow problems using a Discontinuous Galerkin spectral-element approach.[3–5] The mesh generation procedure can be fully unstructured and we can adapt both in mesh size (h) and approximation order (p). However, it also adds further complexity to the mesh generation procedure in that the boundary-conforming elements ultimately must be curved.

The current state-of-the-art meshing techniques are based on manual and empirical methods that require significant user interaction. The main goal here is to develop a meshing algorithm that provides a reliable adaptive capability which is build upon a foundation of provable mathematical methods. Consequently we would like to develop an algorithm that is capable of aligning the elements along a certain predefined metric that is prescribed by the flow solution. The metric field is used to locally describe how length is measured along a manifold, and in this way, controls the orientation and the size of each element. Alauzet[6] provides a good review of the current state-of-the-art of metric aligned meshing strategies.[7–10] Hecht and Mohammadi[?] were the first ones to generate anisotropic meshes by computing a unit mesh under a given Riemannian metric space. Next to the fact that we want to generate metric-aligned meshes, we also ultimately prefer to generate a mesh that is quad-dominant namely for two reasons: First, for quadrilateral, and ultimately hexahedral elements, the tensor-product formulations simplify significantly and are more efficient compared to the tensor-product formulations for triangles or tetrahedra. Second, for the quadrilateral/hexahedral elements, the number of edges are significantly reduced which for the DG method reduces the computational cost.

One of the approaches that recently has gained more attention is a variational method called Centroidal Voronoi Tesselation (CVT)[11] which relies on the Delaunay and Voronoi tesselation of a given set of points. By solving a minimization problem, a mesh is derived for which the generation points (mesh points) are the centroids (center of mass) of their corresponding Voronoi cell. This method has been used to derive smooth meshes with equilateral triangles. Levy and Lui[12] extended the CVT approach by first incorporating a

[*]USRA/NASA Postdoctoral Program Fellow.
[†]Former USRA/NASA Postdoctoral Program Fellow.
[‡]NASA ARC. AIAA Member.

metric tensor field to align the axes of the Voronoi cells with a predefined tensor field (metric) and second by increasing the norm of the energy functional in order to retrieve right-angled triangles which potentially can be recombined as quadrilateral elements. This method is currently referred to as L_p-CVT and Levy and Lui[12] demonstrated that this method generates metric-aligned quad-dominant meshes for periodic surfaces and periodic volume meshes. Baudouin et al.[13] presented an L_p-CVT algorithm for quadrilateral surface mesh generation and applied the method for bounded domains rather than periodic surfaces. They also compute the energy and its gradient using Gaussian integration techniques as opposed to analytical formulas that were used by Levy and Lui.[12]

Caplan et al.[14] also pursue a more mathematically rigorous mesh generation approach. Their mesh generation strategy produces anisotropic meshes that are Delaunay-based and it relies on an embeddy ing algorithm that transforms the anisotropic problem to a uniform problem using a Riemannian metric field. They demonstrate the isometrically embedding of arbitrary mesh-metric pairs in higher dimensional Euclidean spaces. Once the mesh-metric pair is embedded, isotropic mesh generation strategies are applied in the embedded space. However the emphasis in this work lies on the generation of meshes that consists of simplices.

More recently, a prototype of a metric-aligned quad-dominant meshing algorithm for bounded computational domains has been presented by the authors[1] which relies on L_p-CVT. The prototype implementation was capable on running metric tensor fields that were derived from analytic functions rather than from real discrete data. The aim of this paper is to extend the work presented by Ekelschot et al.[1] where the extension is threefold. First, we demonstrate our optimized implementation where the costly computations are done in C^{++} and are also implemented in parallel using Message Passing Interface (MPI). Second, we demonstrate how L_p-CVT can be used when dealing with discrete metric information within a bounded computational domain. Third, we demonstrate how to generate a metric-aligned high-order mesh using the current implementation. In order to apply L_p-CVT as a means of mesh adaptation, we need to use an error indicator. For now, the Hessian that is derived from a flow field quantity is used to drive this method but ideally we would like to use some form of adjoint-based error indicator.[5]

The outline of the paper is as follows: First, a general overview of L_p-CVT is given in section II. This is followed by demonstrating L_p-CVT for a simple bounded computational domain using an analytic metric tensor field. In section IV, we outline the workflow that is required in order to handle discrete metric data that is fed into the algorithm. This includes the discussion of the different metric interpolation methods that are at hand. At the end of this section we set-up a test case for which we intent to demonstrate L_p-CVT. In section V, we describe how we use L_p-CVT to generate high-order metric aligned meshes. Finally we draw conclusions and describe future work in section VI.

II. L_p-Centroidal Voronoi Tesselation

The method of L_p-Centroidal Voronoi Tesselation generates a mesh for which the vertices of the mesh are in the centroid of their corresponding Voronoi cell, taken into account a metric tensor that describes the anisotropy of the local element distribution. This is achieved by minimizing an energy functional that is evaluated by integrating over the Voronoi diagram.[12] This minimization problem is defined as

$$\min_{\mathbf{x}_i} f(\mathbf{x}_i) \tag{1}$$

where

$$f(\mathbf{x}_i) = \sum_{i=1}^{N_v} \int_{\Omega_i} \|\mathbf{M}(\mathbf{y})\,(\mathbf{x}_i - \mathbf{y})\,\|_p d\mathbf{y} \tag{2}$$

N_v is the number of Voronoi cells, Ω_i is the domain that corresponds to the i^{th} Voronoi cell and $\mathbf{x}_i$ is the centroid of the i^{th} Voronoi cell. $\mathbf{M}$ is the metric tensor prescribing the local directionality and size of the elements and $\mathbf{y}$ is the integration variable. The norm of the integrand is defined by p and is used to enforce right angles in the Voronoi diagram and consequently in the resulting simplex mesh.

We tesselate the Voronoi cell itself so that we can easily evaluate the integral over the Voronoi cells using well-known Guassian quadrature rules. Equation (2) is rewritten as

$$f(\mathbf{x}_i) = \sum_{i=1}^{N_v} \sum_{j=1}^{m_i} f_{i,j}(\mathbf{x}_i) \tag{3}$$

where m_i is the number of simplices that form the i^{th} Voronoi cell.

$$f_{i,j}(\mathbf{x}_i) = \int_{S_j} \|\mathbf{M}(\mathbf{y})\,(\mathbf{x_i} - \mathbf{y})\,\|_p d\mathbf{y} \tag{4}$$

where S_j refers to the simplex defined by the mesh point $\mathbf{x}_i$ and $\mathbf{c}_{i,j}$ and $\mathbf{c}_{i,j+1}$ as illustrated by the red simplex in Fig. 1.

The gradient of the energy can be evaluate using

$$\frac{df_{i,j}}{d\mathbf{x}_i} = \frac{\partial f_{i,j}}{\partial \mathbf{x}_i} + \sum_{s=1}^{N_v}\sum_{t=1}^{m_i} \frac{\partial f_{i,j}}{\partial \mathbf{c}_{s,t}} \frac{\partial \mathbf{c}_{s,t}}{\partial \mathbf{x}_i} \tag{5}$$

where the last term of the right hand side of equation (5) essentially represents the coupling of the gradient between the mesh vertices.

In Fig. 1 a schematic representation of the gradient evaluation for one isolated triangle ($f_{0,0}$) is given. The

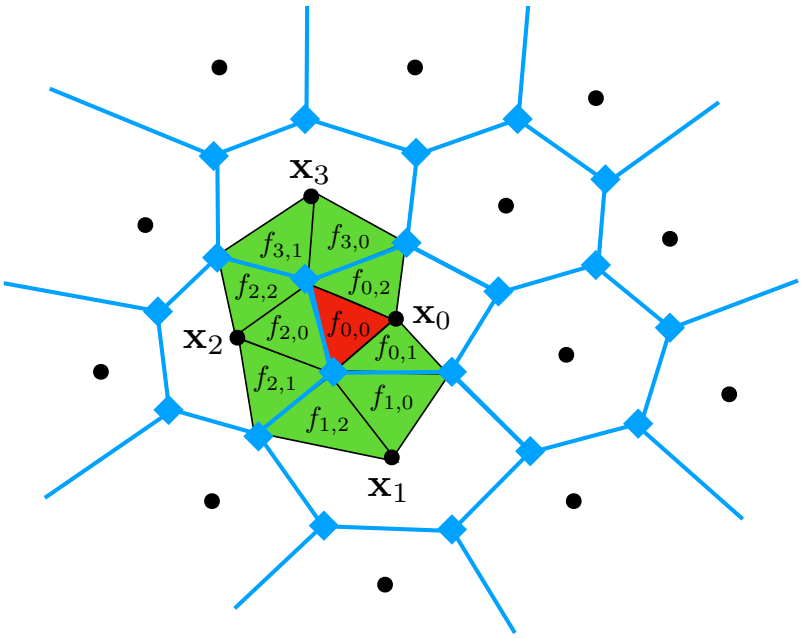


Figure 1. Graphical representation of the connectivity data that is required to evaluate the energy gradient for an isolation integration triangle.

three vertices that form $f_{0,0}$ are the mesh vertex $\mathbf{x}_0$ and the two Voronoi vertices $\mathbf{c}_{0,1}$ and $\mathbf{c}_{0,2}$. The coupling terms indicated by the second term on the right-hand side of equation (5) are schematically represented by the green triangles in Fig. 1. It shows the connectivity between the integration simplices.

We are using this method to optimize meshes for bounded domains. In order to do that we use a reconstruction method for the boundary Voronoi cells that was introduced by Ekelschot et al.[1] such that the energy and gradient evaluation is performed correctly. The tedious Voronoi clipping can be avoided in this way.

III. Parallel implementation

The results that were presented in Ekelschot et al.[1] were obtained using a prototype implementation in Python. One of the conclusions of that paper was that the evaluation of the energy and gradient become too costly once we want to optimize meshes that have more $\mathcal{O}(10^3)$ points. An effort has been made to optimize the algorithm. This is still an ongoing process but a current status and expected improvements are elaborated in this section.

In our previous work, we concluded that the energy and gradient evaluation are most costly. However we would still like to use some of the off-the-shelf capabilities that are already available in various modules

that can be loaded into Python. Particularly the optimizers available in the *SciPy* module and the post-processing capabilities available in Python. The minimization call is done in Python where we typically use L-BFGS or SLSQL as an optimization algorithm. The energy and gradient computations are currently written in $C++$ and are wrapped such that they can be called in Python. The optimization algorithms are inherently serial and we therefore measured the serial speed-up compared to the previous implementation which is approximately $\mathcal{O}(10^2)$ faster.

As described in the previous section, we evaluate the energy and its corresponding gradient using Gaussian quadrature over these integration simplices that follow from the tessellation of each Voronoi cell. The integration over these integration simplices results in repetitive calculations and array operations that are perfectly suitable for Single Instruction Multiple Data (SIMD) vectorization on modern hardware. Vectorization is the process of rewriting a loop so that it processes n elements of the array simultaneously instead of n times a single element depending on which hardware the application is run on. As a result, we stack the integration simplices described in the previous section in an array and a vectorized loop is written to evaluate the integration over each simplex in order to calculate the global energy and corresponding gradient. The vectorization of the energy and gradient loops improves the performance with a factor of 2.

Next to the fact that we have optimized the energy and gradient evaluation as a serial calculation, we have also ensured that both function calls run in parallel using a Message Passing Interface (MPI) implementation. A schematic representation of the current implementation is shown in Fig. 2. We control whether the energy

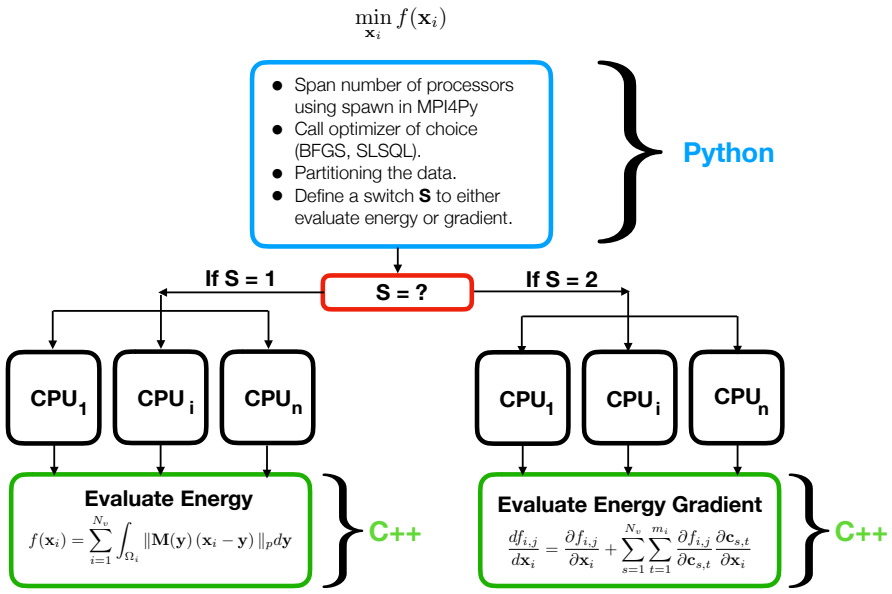
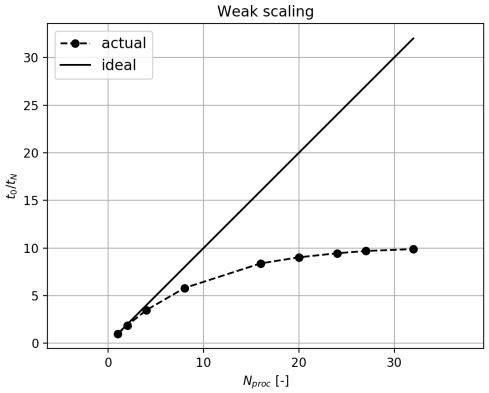


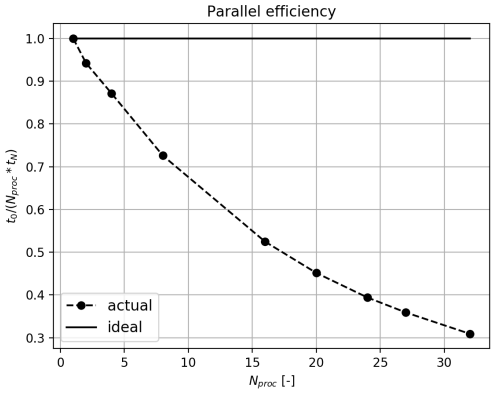
Figure 2. Schematic representation of the code

or the gradient is evaluated using a simple switch. For each energy and gradient computation, we need to repartition the mesh since the mesh points have moved. This is a computationally costly procedure particularly when calculating the energy gradient since the connectivity needs to be figured out. For the energy itself the partitioning is rather straightforward since we can treat each integration simplex independently.

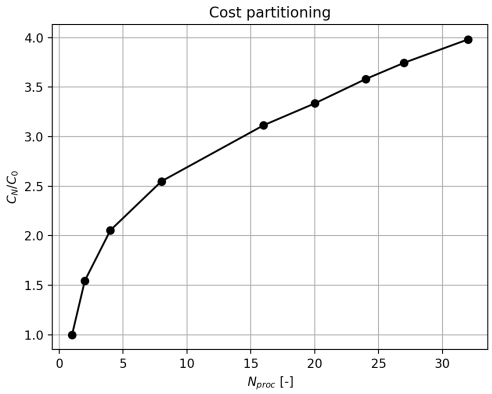
In Fig. 3 we present preliminary weak scaling and parallel efficiency plots. Fig. 3(a) and 3(b) demonstrate that the parallel implementation achieves a speed-up of 10 times compared to the serial implementation. Fig. 3(c) shows the cost of repartitioning of the data and it illustrates how the partitioning cost grows and becomes dominant when we want to run on more cores. At the moment we are working on including a parallel partitioner like ParMetis and eventually incorporating a parallel Delaunay algorithm to mitigate these serial bottlenecks.



(a)



(b)



(c)

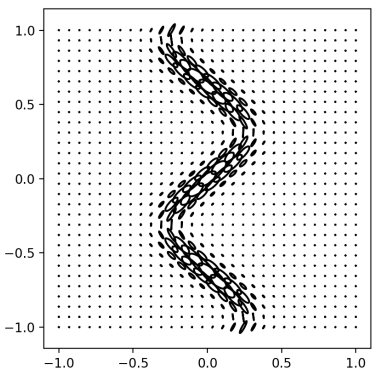
Figure 3. a)Weak scaling of the current implementation plotted together with ideal weak scaling. b) The parallel efficiency plotted against the number of processors used. c) The cost of partitioning plotted against the number of processors.

S-shaped metric tensor

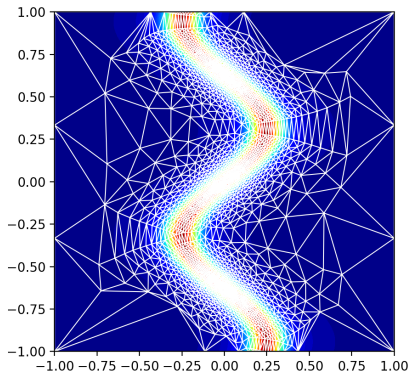
To test the new implementation, we use an analytical function that mimics an S-shaped discontinuity.

$$f(x, y) = a \arctan\left(b(\sin 5y - cx)\right) \tag{6}$$

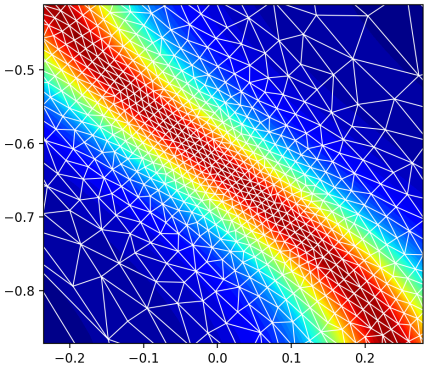
The Hessian is derived from this function and used as a metric tensor field to measure the length of the edges locally. The metric is represented by the ellipses in Fig. 4(a). Note that the size of the ellipse is inversely proportional to the resulting element size locally. We see this as well when we look at the final mesh that is plotted in Fig. 4(b). The function $f(x, y)$ is plotted underneath the mesh to indicate how well the mesh follows the underlying metric field. In Fig. 4(c) we have plotted a zoomed in view showing the nicely aligned



(a)



(b)



(c)

Figure 4. (a) Metric field for $f(x, y) = a \arctan\left(b(\sin 5y - cx)\right)$. (b)Converged mesh for a given bandwidth of allowable edge length.

simplices. Note that the size of the simplices are slightly bigger near the inflection point in the discontinuity. This can be ascribed to the local metric field since Fig. 4(a) also shows smaller ellipses (and this larger elements) in these regions. The meshes regarding this case that were presented by Ekelschot et al.[1] had a number of mesh points that was in the order of 200 points and the edge length near the discontinuity were longer than 2. The results presented in Fig. 4 were ran on a single core and the optimized mesh counts 5246 elements. With the new implementation we are able to start running more realistic cases where the number of mesh points is in the order $\mathcal{O}(10^5)$-$\mathcal{O}(10^6)$.

IV. Discrete metric fields

In this paper, the goal is to use L_p-CVT to optimize a mesh that relies on a discrete metric tensor. In this case, we rely on a flow solution that is typically represented discretely. Consequently, we need to be able

to locally interpolate the metric tensor and the corresponding gradient of the metric while preserving the Symmetric Positive Definite (SPD) property of the metric tensor field. We rely on a background mesh, that is defined on $\Omega_b = (x_b, y_b)$, on which a sampled metric tensor field is derived from a statistically steady flow solution. This flow solution is determined using an initial (coarse) mesh. During the optimization procedure, the mesh and Voronoi vertices that form the integration simplices move around in the computational domain. Hence, we should be able to query the metric tensor at every location in the computational domain. The metric tensor at an arbitrary point is determined by finding the appropriate simplex in the background mesh that surrounds the queried vertex. The metric tensor at the queried location is interpolated using the metric tensors corresponding to the vertices of the surrounding triangle. We can linearly interpolate the metric field based on the barycentric coordinates, β_i, of the surrounding triangle which preserves the SPD property of the metric field. The interpolated metric becomes:

$$\hat{\mathbf{M}} = \sum_{i=1}^{3} \beta_i \mathbf{M}_i \tag{7}$$

where $\hat{\mathbf{M}}$ refers to the interpolated metric tensor and $\mathbf{M}_i$ refers to the metric tensor of the i^{th} vertex of the surrounding triangle.

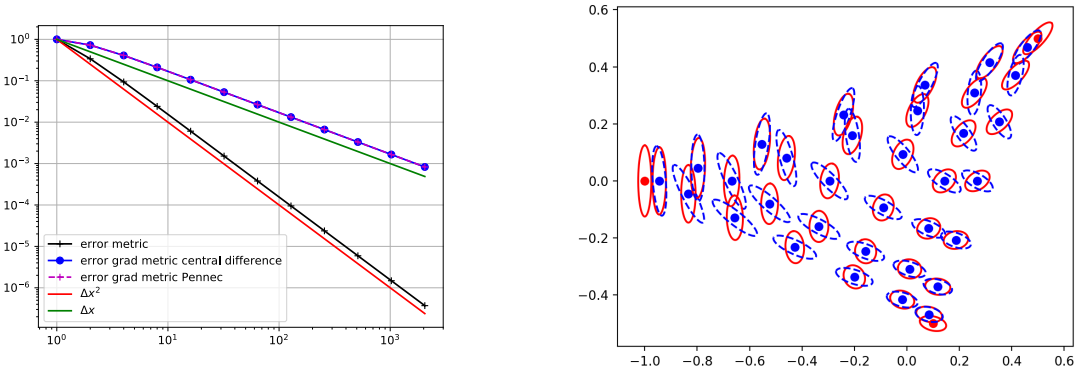
Pennec et al.[15] considered standard operations which are valid for vector spaces like addition subtraction and measuring distance and generalized these concepts for Riemannian manifolds using exponential and logarithmic maps. Using these maps, they derive an iterative metric interpolation method that essentially determines a weighted mean of the metric tensor in log-space:

$$\hat{\mathbf{M}}_{n+1}(\mathbf{x}) = \hat{\mathbf{M}}_n^{\frac{1}{2}} \exp\left(\sum_{i=1}^{N} w_i(\mathbf{x}) \log\left(\hat{\mathbf{M}}_n^{-\frac{1}{2}}(\mathbf{x}) \mathbf{M}_i \hat{\mathbf{M}}_n^{-\frac{1}{2}}(\mathbf{x})\right)\right) \hat{\mathbf{M}}_n^{\frac{1}{2}} \tag{8}$$

From equation (4) and (5) follows that we also need to determine the gradient of the metric at the interpolated location. We can do this either by applying a finite difference approach by using equation (7) to evaluate the metric at the perturbed locations. Furthermore, Pennec et al.[15] propose the following expression to approximate the directional derivative of the metric that is consistent with the expression given in equation (8).

$$\partial \mathbf{M}_\epsilon \approx \mathbf{M}^{\frac{1}{2}}(\mathbf{x}) \log\left(\mathbf{M}^{-\frac{1}{2}}(\mathbf{x}) \mathbf{M}^{\frac{1}{2}}(\mathbf{x}+\epsilon) \mathbf{M}^{-\frac{1}{2}}(\mathbf{x})\right) \mathbf{M}^{\frac{1}{2}}(\mathbf{x}) \tag{9}$$

Currently, both methods are implemented and the convergence of both interpolation methods together with the evaluation of the gradient is plotted in Fig. 5.



(a) (b)

Figure 5. (a) Error convergence of the metric interpolation methods. (b). Comparison between Pennec's metric interpolation method (blue) and linear interpolation (red).

Fig. 5 shows that the interpolation of the metric itself comes down at second order and the error in gradient at first order for both methods. However, when we plot the variation in metric shape for both methods, we

see a significant difference. We are currently looking at the effect of using different interpolation methods when using discrete metric tensor fields in terms of accuracy and code performance.

Next to the previously introduced interpolation methods, we are able to exploit the high-order spline interpolation method once the background mesh for the metric is structured. Once the spine coefficients are obtained, we are able to query the metric tensor components and their corresponding gradients at each location within the computational domain using the high-order interpolation method. This interpolation method is used in the examples shown in the remainder of this paper.

Supersonic ($M = 1.3$) jet flow

To test the L_p-CVT implementation that uses discrete metric data, we consider a supersonic jet flow at $M = 1.3$. The flow solution that is shown in Fig. 6 is computed using the high-order space-time spectral element code that uses a Discontinuous Galerkin discretization.[3–5] Since this is an axisymmetric case, we average the field in azimuthal and temporal direction in order to obtain a statistically steady Mach distribution. The Hessian is computed and used as a metric tensor. The metric data is computed on a structured grid so we use a spline interpolation to approximate the metric components throughout the computational domain. The total number of sample points in x and y direction are 70 and 200 respectively.

First, the location of the boundary nodes are determined by solving the minimization problem given in equation (1) locally along each boundary edge. This is followed by introducing 50 mesh points randomly within the domain and solve the minimization problem for the volume. By setting a minimum edge length, l_{min} and a maximum edge length, l_{max}, we either introduce or remove mesh points. For now, we introduce a point halfway along an edge that is longer that l_{max}. Once an edge is smaller than l_{min}, we remove both points and introduce a new point halfway along the edge. The bandwidth of allowed edge lengths that is defined by l_{min} and l_{max} is reduced iteratively. For this jet case we set $l_{min} = 0.5$ and $l_{max} = 2.5$.

Fig. 6(a) shows the adapted mesh for the supersonic jet case. There is a clear alignment of the mesh near the shear layer and once the gradient decreases we see a smooth transition towards the far-field particularly further downstream. A detailed view on the mesh near the shear layer is shown in Fig. 6(b). Elongated stretched triangles are obtained with right angles which potentially could be recombined into quadrilateral elements for example using the Blossom Quad algorithm.[16]

The convergence history is plotted in Fig. 7(a). Note that the jumps in the convergence history are due to restarting the optimization procedure. Usually the optimizer stalls after some number of iterations and based on the final mesh is used as a new initial condition. Based on l_{min} and l_{max} points are removed and introduced at the appropriate locations and a new optimization iteration is started. Typically, the energy increases when more points are removed than inserted but eventually the energy value decreases and converges as shown in Fig. 7(a). When we inspect the statistical distribution of the edge lengths, we see that the mean edge length is approximately 1 as expected and the bandwidth is more or less defined by $l_{min} = 0.5$ and $l_{max} = 2.5$.

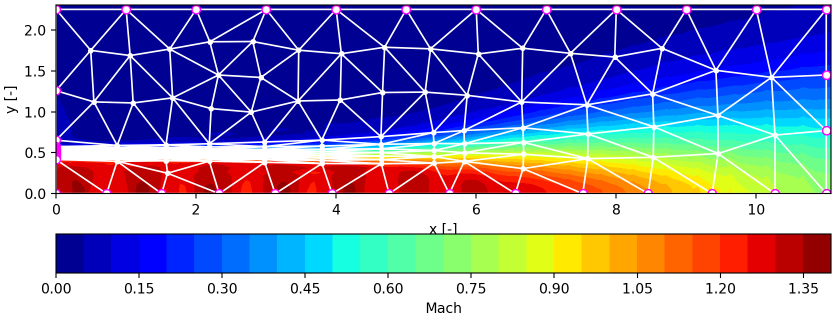
V. Generating the high-order mesh

In this section, a description is given of the high-order metric-aligned mesh generation strategy. We demonstrate the process using the test case of subsonic ($M = 0.3$, $Re = 2900$) flow past a sphere where the metric field is derived by computing the Hessian of the Mach number distribution (see Fig. 8).

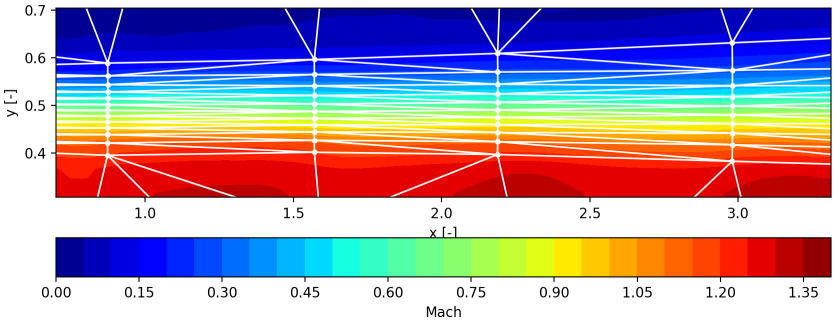
The L_p-CVT optimization process is carried out within a convex reference domain. We map the physical domain, $\mathbf{x}_p = (x_p, y_p)$, which is typically non-convex, to a convex reference domain where the reference domain is a square with coordinates $\boldsymbol{\xi} = (\xi, \eta)$. The physical domain is defined on a very fine structured grid which is used to compute spline coefficients. The spline coefficients provide us with a high-order isogeometric mapping, $\mathbf{S}_{r\to p}$, that maps the reference coordinates to physical coordinates. However, we need to compute the inverse of this mapping, $\mathbf{S}_{p\to r}$, in order to map the metric in physical space, $\mathbf{M}_p$, to a metric in reference space, $\mathbf{M}_r$ which we can use in our L_p-CVT formulation. $\mathbf{M}_r$ is calculated as follows:

$$\mathbf{M}_r = J^t \mathbf{M}_p J \tag{10}$$

where the Jacobian is derived based on $\mathbf{S}_{p\to r}$ and is defined as $J = \partial\mathbf{x}/\partial\boldsymbol{\xi}$. Since we do not have $\mathbf{S}_{p\to r}$ by $\mathbf{S}_{r\to p}$, we need to determine J for each physical mesh point iteratively using a classical Newton method. In Fig. 9, the physical computational domain for the previously introduced cylinder flow case is given in

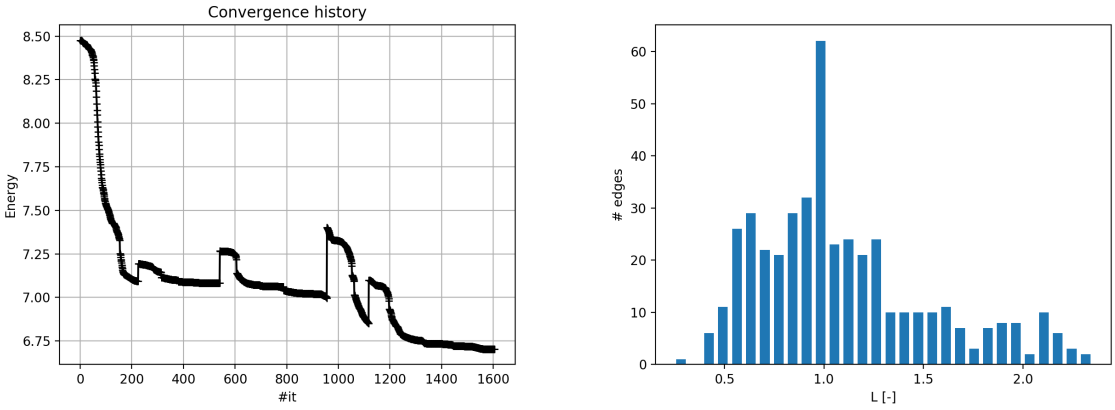


(a)



(b)

Figure 6. (a) Initial random point distribution. (b) Point distribution after x number of optimization iterations.



(a) (b)

Figure 7. (a) Convergence history by plotting the energy value at each iteration. (b)Histogram of the number of edges of a particular length for the converged mesh for the supersonic jet test case.

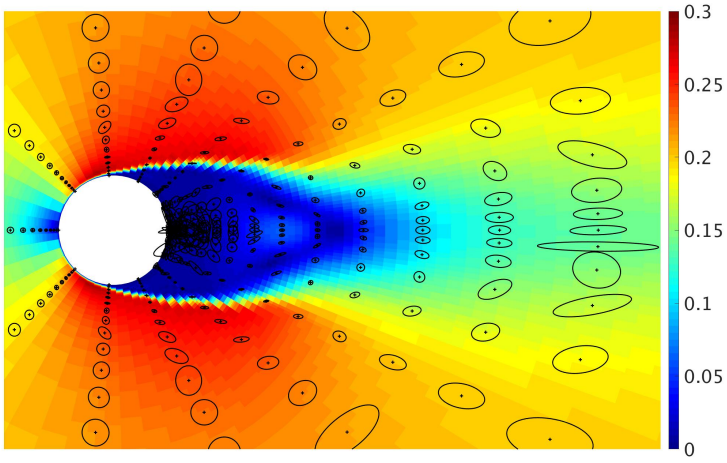
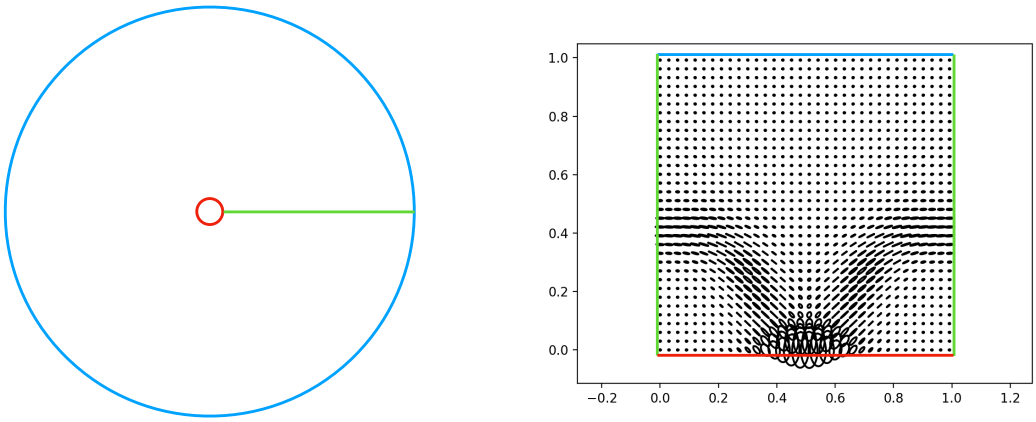


Figure 8. Mach number distribution for subsonic flow past a sphere together with the metric tensor field indicated by the ellipses.

Fig. 9(a). This domain is mapped using $\mathbf{S}_{p\to r}$ into a reference space that is defined as $\xi = [0,1]^2$, shown in Fig.9(b). In Fig. 9, the mapping from physical domain to reference domain is indicated by the color-coded



(a) (b)

Figure 9. (a) The computational domain in physical space $\mathbf{x} = (x, y)$. (b) The metric field in reference space $\boldsymbol{\xi} = (\xi, \eta)$ for the cylinder mesh.

edges. The green edge in Fig. 9 is periodic in this case. The red edge represents the geometry and gets mapped to the bottom edge in reference space. The outer edge in physical space gets mapped to the upper edge in reference space. The metric in reference space ($\mathbf{M}_r$) is shown using ellipses in Fig. 9 (b) which shows larger ellipses near the edge of the wall of the cylinder which ultimately shoudl result in smaller elements near geometry boundary.

After we have computed $\mathbf{M}_r$, we run the L_p-CVT optimization process described in section II and a converged mesh is obtained that satisfies the allowed edge length tolerances, we transform the mesh back into physical space using $\mathbf{S}_{r\to p}$. The steps that are taken to generate the high-order metric aligned mesh are listed here:

1. Compute primal solution and obtain the metric data.
2. Transform the metric data in physical space to reference space.
3. Compute the optimized mesh using the method described in section II

4. Transform the optimized mesh back into physical space.

5. Potentially recombine the triangles into quadrilaterals (not shown here).

For the cylinder case, we again rely on a discrete metric field that is defined on a structured grid of 21 points in radial direction and 31 points in angular direction. For this case we ran two experiments: first we used a relatively wide bandwidth of allowed edge lengths ($l_{min} = 0.5$ and $l_{max} = 2.5$). The optimized mesh in reference space is shown in Fig. 10(a). The colors in Fig. 10(a) indicate the edge length. It can be seen that the edges are slightly longer near the areas where the metric tensor field is stretched in a certain direction. The statistical distribution of edge lengths is indicated in Fig. 10 (b). It shows that the mean edge length lies just below one. The convergence history is shown in 10(b). Note that we started off with a small number of points ($N = 50$). The convergence history shows that at each restart, more points were added than removed since the energy goes monotonically down. Once a converged mesh in reference space is computed, we map the mesh back into physical space and end up with a high-order mesh that is given in Fig. 10(c) and 10(d). As expected, we see a clear refinement near the wall of the cylinder. Furthermore we see anisotropy of the elements at the location where the shear layer is present.. Based Fig. 10(c) we see that the range of edges are limited by the bandwidth of edge lengths that we prescribe beforehand.

As an experiments, we narrowed down the bandwidth of allowable edge lengths to see what the effect is on local accuracy of the resulting mesh. We repeat the previously described process but now for $l_{min} = 0.5$ and $l_{max} = 1.5$. The final mesh that is obtained is shown in Fig. 11(a). The colors of the edges indicate the edge length. In Fig. 11(b) the a statistical distribution of edge lengths is given and it can be seen that the bandwidth is indeed more narrow and corresponds with the prescribed bandwidth. The mean edge length is reduced to approximately 0.6. If we look at the mesh in physical space, we see a nearly symmetric point distribution. There is a clear refinement near the wall of the cylinder as expected and the elements near the wake are becoming more stretched in downstream direction. Improved results are expected once this procedure is adopted in an iterative adaptive procedure where the quality of the discrete metric data will improve with each iteration.

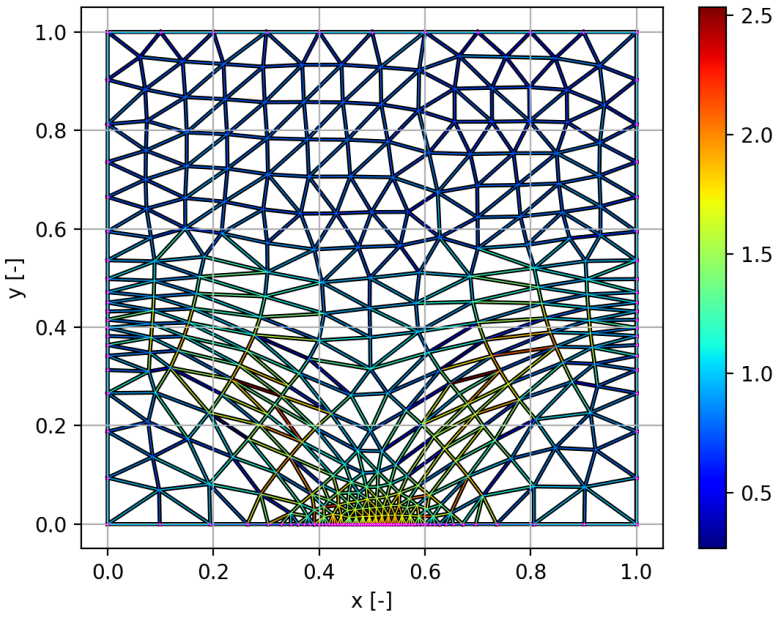
VI. Conclusions

We have demonstrated in this paper that we can apply the previously introduced meshing algorithm that relies on L_p-CVT on discrete metric data that is provided by flow solver in this case *eddy*.[3–5] A significant serial speed-up of the code has been achieved ($\mathcal{O}(10^2)$ compared to the previous implementation[1]) by identifying and optimizing the routines that are computationally expensive. These routines are written in $C++$ and wrapped in Python. This allows us to use useful off-the-shelf modules like optimizers and post-processing tools that are already available in Python. Furthermore, we are able to run the energy and gradient computation in parallel using MPI. Fig. 3 shows that the parallel implementation achieves a speed-up of 10 times compared to the serial implementation. The partitioning issues that we are currently seeing are expected to be resolved by using a parallel partitioner.

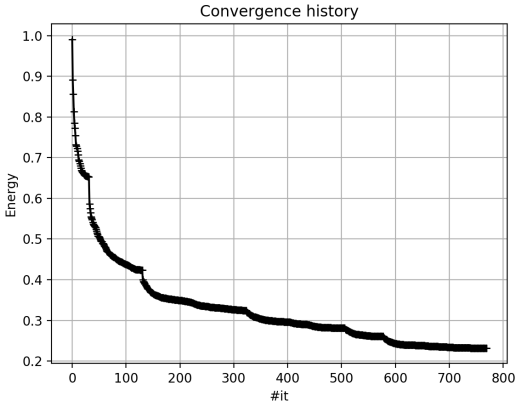
Furthermore, we demonstrated a parallel implementation of L_p-CVT that uses discrete metric tensors. We are able to apply various metric interpolation methods in order to compute the metric at any arbitrary point in the computational domain. Finally, we have shown the workflow for generating high-order metric-aligned meshes that rely on L_p-CVT for bounded domains. The results presented for the cylinder case are preliminary but encouraging for capturing anisotropic flow features like boundary layers and shear layers. The next steps are to incorporate the high-order metric-aligned meshing procedure in an iterative mesh adaptation procedure. Next to that, we are investigating appropriate stopping criteria for the L_p-CVT optimization that are based on the unit-volumes rather than unit-length of the edges. At the moment, we adopt a higher p-norm to enforce right-angled triangles which are easy to recombine into quadrilaterals. At the moment we would have to rely on a recombination procedure as a post-processing step but we are currently looking into incorporating new criteria next to the increased p-norm in the L_p-CVT minimization itself.

References

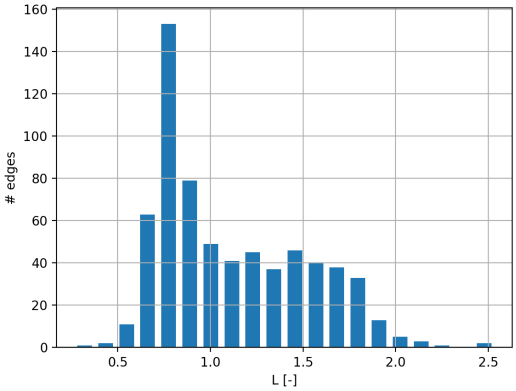
[1]Ekelschot, D., Ceze, M., Garai, A., and Murman, S., "Robust metric aligned quad-dominant meshing using Lp centroidal Voronoi tessellation," *AIAA SciTech Forum, AIAA Paper 2018-1501*, Kissimmee, Florida, US, 2018.



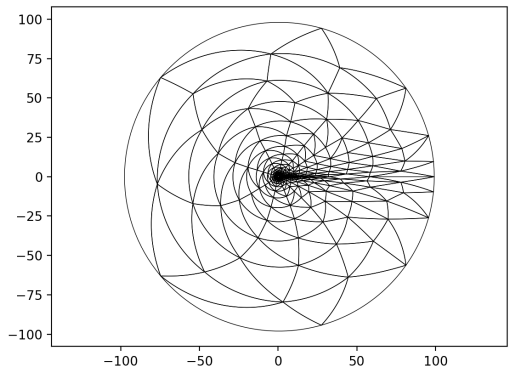
(a)



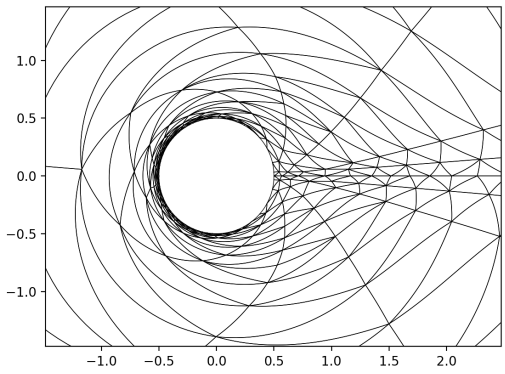
(b)



(c)

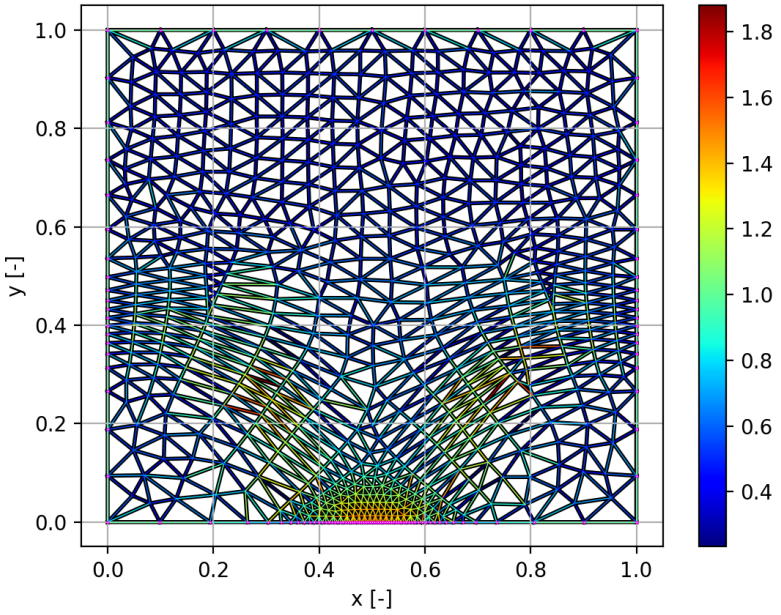


(d)

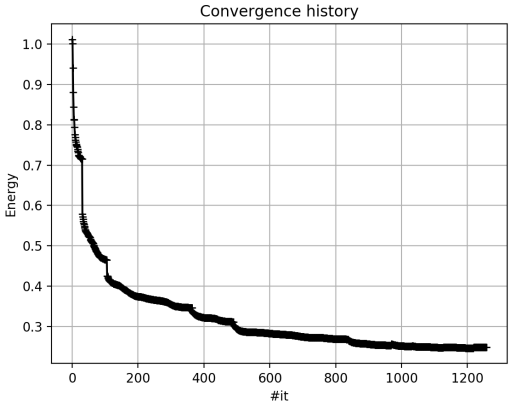


(e)

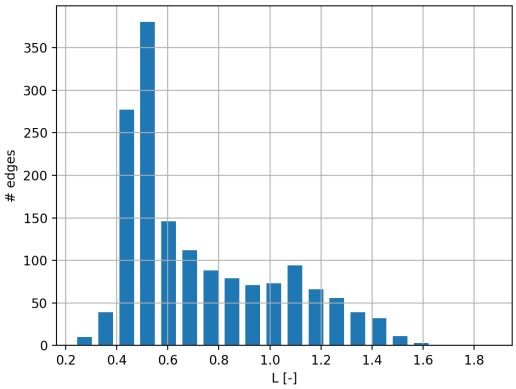
Figure 10. (a) The final mesh in reference space. (b) Convergence history by plotting the energy value at each iteration. (c) Histogram of the number of edges of a particular length for the final mesh. (d) Zoomed out view of the curved mesh around a cylinder based on the Hessian of the Mach number. (e) Detailed view of the curved mesh around a cylinder based on the Hessian of the Mach number.



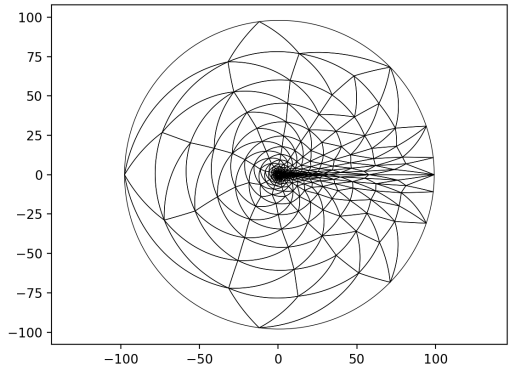
(a)



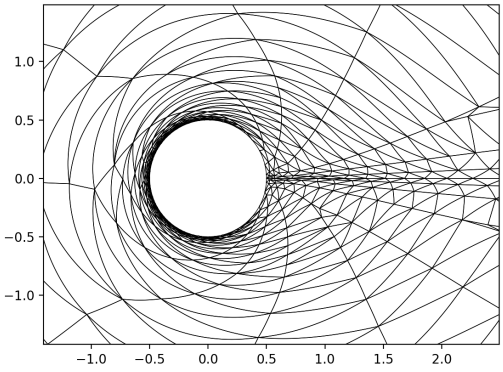
(b)



(c)



(d)



(e)

Figure 11. (a) The final mesh in reference space. (b) Convergence history by plotting the energy value at each iteration. (c) Histogram of the number of edges of a particular length for the final mesh. (d) Zoomed out view of the curved mesh around a cylinder based on the Hessian of the Mach number. (e) Detailed view of the curved mesh around a cylinder based on the Hessian of the Mach number.

[2]Slotnick, J., Khodadoust, A., Alonso, A., Darmofal, D., Gropp, W., Lurie, E., and Mavriplis, D., "CFD Vision 2030 Study: A Path to Revolutionary Computational Aerosciences," Tech. Rep. CR-2014-218178, NASA, 2014.

[3]Diosady, L. and Murman, S., "Design of a Variational Multiscale Method for Turbulent Compressible Flows," *21st AIAA Computational Fluid Dynamics Conference, AIAA Paper 2013-2870*, San Diego, California, US, 2013.

[4]Diosady, L. and Murman, S., "Higher-Order Methods for Compressible Turbulent Flows Using Entropy Variables,," *21st AIAA Computational Fluid Dynamics Conference, AIAA Paper 2015-0294*, San Diego, California, US, 2015.

[5]Ceze, M., Diosady, L., and Murman, S., "Development of a high-order space-time matrix-free adjoint solver," *54st AIAA Aerospace Sciences Meeting, AIAA Paper 2016-0833*, San Diego, California, US, 2016.

[6]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.

[7]Loseille, A., Dervieux, A., and Alauzet, F., "A 3D goal-oriented anisotropic mesh adaptation applied to inviscid flows in aeronautics," *48th AIAA aerospace sciences meeting including the new horizons forum and aerospace Exposition, AIAA Paper 2010-1067*, Orlando, Florida, US, 2010.

[8]Loseille, A., "Metric-orthogonal Anisotropic Mesh Generation," *Procedia Engineering*, Vol. 82, No. Supplement C, 2014, pp. 403 – 415, 23rd International Meshing Roundtable (IMR23).

[9]Marcum, D. and Alauzet, F., "Aligned Metric-based Anisotropic Solution Adaptive Mesh Generation," *Procedia Engineering*, Vol. 82, 2014, pp. 428–444.

[10]Barral, N., Alauzet, F., and Loseille, A., "Metric-Based Anisotropic Mesh Adaptation for Three-Dimensional Time-Dependent Problems Involving Moving Geometries," *53rd AIAA Aerospace Sciences Meeting, AIAA Paper 2015-2039*, Kissimi, Florida, US, 2015.

[11]Du, Q., Faber, V., and Gunzburger, M., "Centroidal voronoi tessellation: Applications and algorithms," *SIAM Review*, Vol. 41, 1999, pp. 637–676.

[12]Lévy, B. and Lui, Y., "L_p Centroidal Voronoi Tessellation and Its Applications," *ACM Transactions on Graphics*, Vol. 9, No. 4, 2010, pp. 1–11.

[13]Baudouin, T., Remacle, J.-F., Marchandise, E., Lambrechts, J., and Henrotte, F., "Lloyd's energy minimization in the Lp norm for quadrilateral surface mesh generation," *Engineering with Computers*, Vol. 30, No. 1, 2014, pp. 97–110.

[14]Caplan, P., Haimes, R., Darmofal, D., and Galbraith, M., "Anisotropic geometry-conforming d-simplicial meshing via isometric embeddings," *Procedia Engineering*, Vol. 203, No. Supplement C, 2017, pp. 141 – 153, 26th International Meshing Roundtable, IMR26, 18-21 September 2017, Barcelona, Spain.

[15]Pennec, X., Fillard, P., and Ayache, N., "A Riemannian Framework for Tensor Computing," *International Journal of Computer Vision*, Vol. 66, 2005, pp. 41–66.

[16]Remacle, J.-F., Lambrechts, J., Seny, B., Marchandise, E., Johnen, A., and Geuzainet, C., "A non-uniform quadrilateral mesh generator using a minimum-cost perfect matching algorithm," *International Journal for Numerical Methods in Engineering*, Vol. 89, 2012, pp. 1102–1119.