Chapter 4
Incorporating a Discontinuous Galerkin Method into the Existing Vertex-Centered Edge-Based Finite Volume Solver Edge

Sven-Erik Ekström and Martin Berggren

Abstract. The discontinuous Galerkin (DG) method can be viewed as a generalization to higher orders of the finite volume method. At lowest order, the standard DG method reduces to the cell-centered finite volume method. We introduce for the Euler equations an alternative DG formulation that reduces to the *vertex-centered* version of the finite volume method at lowest order. The method has been successfully implemented for the Euler equations in two space dimensions, allowing a local polynomial order up to $p = 3$ and supporting curved elements at the airfoil boundary. The implementation has been done as an extension within the existing edge-based vertex-centered finite-volume code Edge.

1 Introduction

The finite volume (FV) method is presently the most widely used approach to discretize the Euler and Navier–Stokes equations of aerodynamics. A basic choice when implementing a finite volume method is whether to employ a *cell-centered* or *vertex-centered* approach. The control volumes coincide with the mesh cells in the cell-centered approach (left in Figure 1), whereas in the vertex-centered approach, the control volumes are constructed from a dual mesh, consisting in two dimensions of polygons surrounding each vertex in the original primal mesh (right in Figure 1).

The vertex-centered approach has a number of features that helps to explain its current popularity. A vertex-centered scheme has about half the memory footprint on the same mesh as a cell-centered scheme, and has more fluxes per unknown,

Sven-Erik Ekström
Department of Information Technology, Uppsala University,
Box 337, SE-751 05 Uppsala, Sweden
e-mail: sven-erik.ekstrom@it.uu.se

Martin Berggren
Department of Computing Science, Umeå University, SE-901 87 Umeå, Sweden
e-mail: martin.berggren@cs.umu.se

N. Kroll et al. (Eds.): ADIGMA, NNFM 113, pp. 39–52, 2010.
springerlink.com

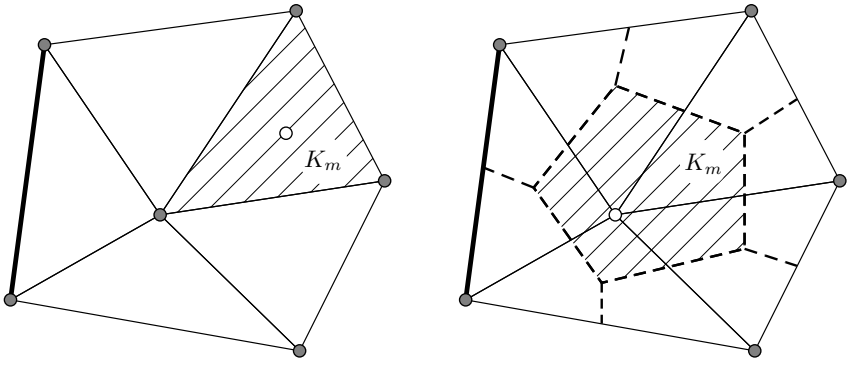

Fig. 1 Control volume choices for finite volume methods. A computational node (white circle) is either associated with a mesh cell center (left) or a mesh vertex (right), giving a cell-centered or a vertex-centered control volume respectively.

as discussed by Blazek [7]. Also, for instance Abgrall [1] argues that reconstruction schemes are more easily formulated for vertex-centered control volumes. Moreover, as opposed to cell-centered schemes, the treatment of boundary conditions are facilitated by the fact that control volume centers are located precisely on the boundary. The main computational effort in a typical finite volume code concerns the residual computations. A solver using a vertex-centered scheme may be implemented to support what Haselbacher *et al.* [16] call *grid transparency*: to assemble the residual, the solver loops over all edges in the mesh, regardless of the space dimension or the choice of mesh cell type (triangles, quadrilaterals, tetrahedrons, prisms, hexahedrons). Note, however, that an analogous construction is also possible for cell-centered methods, by looping over a list of cell surfaces. For a detailed discussion on cell-centered versus vertex-centered methods, we refer to Blazek's book [7] and the review by Morton and Sonar [20]. Also, there are other numerical schemes, besides finite volumes, that use vertex-centered control volumes or loop over edges. Some examples are residual distribution schemes [2, 10], edge-based finite elements [19], and edge-based SUPG [8].

Since the discontinuous Galerkin (DG) method constitutes a higher-order generalization of the finite volume method, it is tempting to reuse and extend existing finite volume codes to higher orders to avoid a costly rewrite of a complex and familiar software system. A hurdle for such an approach is that the standard discontinuous Galerkin method is a higher-order version of the cell-centered finite-volume method whereas many of today's codes, such as DLR-Tau [22], Edge [12, 14], Eugenie [13], Fun3D [15], and Premo [23], are vertex-centered.

The contribution of Uppsala University to the ADIGMA project consists of a case study in which a vertex-centered version of the discontinuous Galerkin method is implemented within the Edge system [12, 14] so the user, depending on the need for accuracy, can choose whether to use a classical finite volume scheme or a discontinuous Galerkin discretization. The vertex-centered discontinuous Galerkin

method we use was first introduced by Berggren [5] in the context of a linear first-order hyperbolic model problem. A further analysis of the method was presented by Berggren, Ekström, and Nordström [6] for higher-order elements and for a nonlinear problem with a shock. Here, we present for the first time the use of the scheme for the Euler equations. We first introduce the basic scheme of the method and then further discuss the method as developed within the Edge system.

2 The Discontinuous Galerkin Method

The target application for the proposed scheme is aeronautical CFD with the Euler and Navier–Stokes equations, where computations of steady states are particularly prominent. We here present the scheme for the steady Euler equations, written in the compact form

$$\nabla \cdot \mathscr{F}(U) = 0 \quad \text{in } \Omega, \tag{1}$$

where $U = [\rho, \rho\mathbf{u}, \rho E]^T$ is the solution vector of conservative variables (mass density ρ, momentum density $\rho\mathbf{u}$, and total energy density ρE), $\mathscr{F}$ is the Euler flux function, and Ω the computational domain. We also need to impose appropriate boundary conditions in the far field and at solid walls. The flux function for the Euler equations is

$$\mathscr{F}(U) = \begin{pmatrix} \rho\mathbf{u} \\ \rho\mathbf{u} \otimes \mathbf{u} + \mathbb{I}p \\ \rho\mathbf{u}H \end{pmatrix}, \tag{2}$$

where $\rho H = (\rho E + p)$ is the total enthalpy density, and the pressure p satisfies the equation of state

$$p = (\gamma - 1)\left[\rho E - \frac{|\rho\mathbf{u}|^2}{2\rho}\right], \tag{3}$$

in which $\gamma = 1.4$ for air.

We divide the domain, Ω, into a set of non-overlapping control volumes K_m such that $\overline{\Omega} = \bigcup_{m=1}^{M} \overline{K}_m$. Denote by $\mathscr{V}_h^{d+2}$, where d is the space dimension, the space of numerical solutions to equation (1). Each component of a vector function in $\mathscr{V}_h^{d+2}$ belongs to the space $\mathscr{V}_h$, which comprises functions that are continuous inside each K_m but in general discontinuous across boundaries between control volumes. The restriction on each K_m of functions in $\mathscr{V}_h$ are polynomials for the standard DG method. Here we consider a different choice of functions, as described in Secion 3. Each $V_h \in \mathscr{V}_h^{d+2}$ can be expanded as

$$V_h(\mathbf{x}) = \sum_{m=1}^{M} \sum_{i=1}^{N_m} V_i^m \phi_i^m(\mathbf{x}) = \sum_{i=1}^{N_{\text{dof}}} V_i \phi_i(\mathbf{x}), \tag{4}$$

where M is the number of control volumes, N_m is the number of local degrees of freedom in control volume K_m, $\{\phi_i^m\}_{i=1}^{N_m}$ is the set of basis functions associated with control volume K_m, N_{dof} is the total number of degrees of freedom, and ϕ_i are the basis functions in a global numbering.

The DG method is obtained by multiplying equation (1) by a test function $V_h \in \mathcal{V}_h^{d+2}$, integrating over each control volume, integrating by parts, and introducing the numerical flux $\mathscr{F}^*$ on the boundaries. This procedure yields that $U_h \in \mathcal{V}_h^{d+2}$ solves the variational problem

$$\int_{\partial K_m} V_h \cdot \mathscr{F}^*(U_L, U_R, \hat{\mathbf{n}})\,\mathrm{d}s - \int_{K_m} \nabla V_h \cdot \mathscr{F}(U_h)\,\mathrm{d}V = 0, \ \ \forall K_m \subset \Omega, \quad \forall V_h \in \mathcal{V}_h^{d+2}, \tag{5}$$

where subscripts L and R denote local ("left") and remote ("right") values on the boundary ∂K_m of control volume K_m, and $\hat{\mathbf{n}}$ is the outward unit normal. The remote values are either taken from the neighboring control volume's boundary or, when ∂K_m intersects the domain boundary, from the supplied boundary condition data. Thus, the boundary conditions are imposed weakly through the numerical flux.

In our implementation, we use the Roe numerical flux by default,

$$\mathscr{F}^*_{\mathrm{ROE}}(U_L, U_R, \hat{\mathbf{n}}) = \frac{1}{2}\hat{\mathbf{n}} \cdot \big(\mathscr{F}(U_L) + \mathscr{F}(U_R)\big) - \frac{1}{2}\bar{R}|\bar{\Lambda}^*|\bar{L}^{-1}(W_R - W_L), \tag{6}$$

where the last term in expression (6) is a dissipation term whose factors are described in the following. Let matrices Λ and R be the diagonal eigenvalue matrix and the right eigenvector matrix in the eigendecomposition of the Jacobian with respect to the conservative variables, that is,

$$\frac{\partial(\hat{\mathbf{n}} \cdot \mathscr{F})}{\partial U} = R\Lambda R^{-1}. \tag{7}$$

Moreover, let $W = [\rho, \mathbf{u}, p]^T$ be the primitive variables vector, and let matrix L satisfy

$$L^{-1}(W_R - W_L) = R^{-1}(U_R - U_L). \tag{8}$$

Explicit expressions of L, R, and Λ can for example be found in the books by Blazek [7] and Hirsch [17]. The elements of matrices $\bar{R}$, $\bar{\Lambda}^*$, and $\bar{L}^{-1}$ in expression (6) are evaluated using the Roe averages

$$\begin{aligned} \bar{\rho} &= \sqrt{\rho_L \rho_R}, \quad \bar{\mathbf{u}} = \frac{\mathbf{u}_L\sqrt{\rho_L} + \mathbf{u}_R\sqrt{\rho_R}}{\sqrt{\rho_L} + \sqrt{\rho_R}}, \\ \bar{H} &= \frac{H_L\sqrt{\rho_L} + H_R\sqrt{\rho_R}}{\sqrt{\rho_L} + \sqrt{\rho_R}}, \quad \bar{c} = \sqrt{(\gamma - 1)\,[\bar{H} - |\bar{\mathbf{u}}|^2]}. \end{aligned} \tag{9}$$

Also, to prevent the eigenvalues to vanish, we utilize a standard entropy fix

$$|\bar{\lambda}_i^*| = \begin{cases} \dfrac{\bar{\lambda}_i^2 - \delta^2}{2\delta}, & |\bar{\lambda}_i| \le \delta, \\ |\bar{\lambda}_i|^2, & |\bar{\lambda}_i| > \delta. \end{cases} \tag{10}$$

Also available in our implementation is the much simpler Local Lax–Friedrichs flux

$$\mathscr{F}^*_{\mathrm{LLF}}(U_L,U_R,\hat{\mathbf{n}}) = \frac{1}{2}\hat{\mathbf{n}}\cdot\big(\mathscr{F}(U_L)+\mathscr{F}(U_R)\big) - \frac{C}{2}(U_R-U_L), \tag{11}$$

where C is an upper bound for the absolute values of the eigenvalues of the Jacobian, $\partial(\hat{\mathbf{n}}\cdot\mathscr{F}(U))/\partial U$. We use

$$C = \max\left(|\hat{\mathbf{n}}\cdot\mathbf{u}_L|+c_L, |\hat{\mathbf{n}}\cdot\mathbf{u}_R|+c_R\right), \tag{12}$$

where $c=\sqrt{\gamma p/\rho}$ is the speed of sound. The choice of Roe or Local Lax–Friedrichs numerical flux appears not to cause any significant differences performance wise, neither regarding computational effort nor accuracy.

Boundary conditions are usually imposed simply by appropriate modifications of the remote condition U_R. At the outer free stream, we apply $U_R = U_\infty$ where U_∞ is the free stream values of the solution vector. To impose the solid wall boundary condition, we have implemented two options: the first imposes symmetry and the second uses an explicitly modified flux to enforce non penetration. Both give similar results, both with respect to computational effort and accuracy.

The first option uses the the same Roe or Local Lax–Friedrichs flux at the wall as everywhere else, but imposes symmetry by defining U_R from U_L using a reflected momentum vector:

$$\rho_R = \rho_L, \quad (\rho\mathbf{u})_R = (\rho\mathbf{u})_L - 2(\hat{\mathbf{n}}\cdot(\rho\mathbf{u})_L)\hat{\mathbf{n}}, \quad (\rho E)_R = (\rho E)_L. \tag{13}$$

The second option uses the numerical flux

$$\mathscr{F}^*_{\mathrm{W}}(U,\hat{\mathbf{n}}) = \begin{pmatrix} 0 \\ p\hat{\mathbf{n}} \\ 0 \end{pmatrix}, \tag{14}$$

that is, $\mathscr{F}^*_{\mathrm{W}}(U,\hat{\mathbf{n}}) = \hat{\mathbf{n}}\cdot\mathscr{F}(U)|_{\hat{\mathbf{n}}\cdot\mathbf{u}=0}$ at the wall, instead of the Roe or Local Lax–Friedrichs flux.

The discussion above covers both the usual cell-centered formulation of the DG method and our vertex-centered scheme. In the next sections, we introduce the particularities with our approach: the construction of a macro element on the dual mesh and the use of curved boundaries for these elements.

3 The Vertex-Centered Macro Element

The finite elements of the standard (cell-centered) DG method use polynomials defined separately on each mesh cell, which means that the control volumes K_m discussed in Section 2 coincide with the mesh cells, typically triangles or quadrilaterals in two space dimensions. In our vertex-centered DG method we use another choice of control volumes K_m, defined on a so-called dual mesh.

A preprocessor constructs the dual mesh and necessary data structures; Figure 2 illustrates the procedure. From the primal mesh, top left in Figure 2, the preprocessor constructs the dual mesh shown in top right in Figure 2. A dual mesh can be constructed in several ways, as discussed by Barth [3]; here we choose just to connect the centroids of adjacent triangles to each other with a new edge. Although Figure 2 shows a uniform mesh, dual meshes can be constructed for any nondegenerate mesh. Next we triangulate the dual mesh (bottom left in Figure 2) and define our finite element on each dual cell as the macro element consisting of standard triangular Lagrange elements of order p. That is, the functions are continuous on each K_m, and piecewise polynomials of degree p on each sub triangle of K_m. Note that we allow discontinuities in the solution between adjacent dual cells, but that the solution within each dual cell is continuous. Thus, no flux evaluations are necessary at the internal edges between the sub triangles in the dual cells. Indeed, any internal flux contribution would vanish since the left and right states are identical due to the continuity of the approximating functions across such edges. Note also that the element type and order may be different on different dual cells. At the bottom right in Figure 2, we see an example where all boundary macro elements and the leftmost of the three inner macro elements are of constant type ($p = 0$), which corresponds to the vertex-centered FV method, whereas the center and right interior macro elements are linear ($p = 1$) and quadratic ($p = 2$), respectively. Note the multiple nodes, associated with the possibility of jump discontinuities, occurring at boundaries of the dual cells.

A common method to solve a problem such as the discrete version of equation (1) is to march an unsteady version of the equation to steady state using an explicit

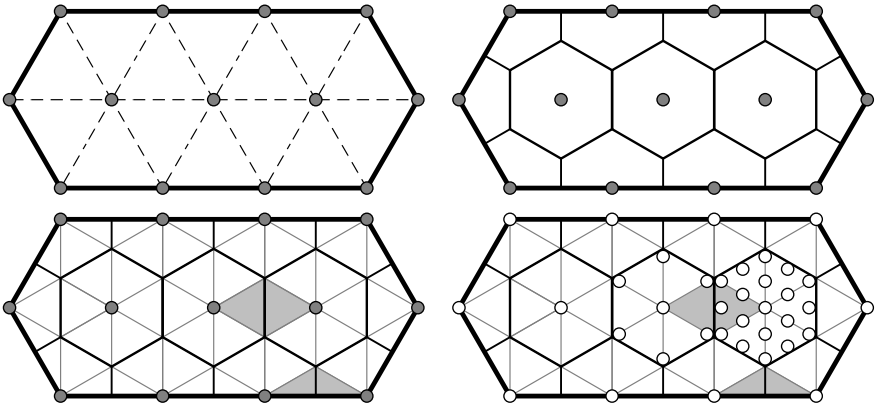

Fig. 2 Preprocessing stages for generating macro elements on the dual mesh. From the primal mesh (top left) the preprocessor constructs a dual mesh (top right) that contains as many polygonal dual cells K_m as the number of mesh vertices in the primal mesh. We triangulate each dual cell (bottom left) and define a macro element on each dual cell (bottom right), where the white circles are the computational nodes.

Runge–Kutta scheme. This strategy is often combined with convergence acceleration strategies such as local time stepping and multigrid; both have been implemented and the multigrid approach is discussed in our other contribution to this volume [11]. A crucial step in such an algorithm is the computation of the residual, which is a vector of dimension equal to the number of degrees of freedom for $\mathcal{V}_h$ for each equation that is solved.

As mentioned in the introduction, implementations of the residual calculation for vertex-centered FV methods are typically *edge-based*, and we now shortly indicate how to extend this approach to the current DG method.

Pointers to the degrees of freedom for $\mathcal{V}_h^{d+2}$ as well as geometric information are stored in a list associated with the edges in the primal mesh. An additional list associated with the domain boundary edges is also required to set boundary conditions.

For each edge in the primal mesh, we associate the two primal mesh vertices i and j connected by the edge, and the normal vector $\mathbf{n}_{ij}$ (with $|\mathbf{n}_{ij}| = |\partial T_{ij}|$) to the intersection of the boundaries of control volumes K_i and K_j. To compute the contribution to the residual associated with this edge, the values of the unknowns at nodes i and j, and the normal $\mathbf{n}_{ij}$ constitute all the information that is needed for $p = 0$ (the finite volume case). For higher orders, we also associate two sub triangles, T_i and T_j, of control volumes K_i and K_j, and a larger set of nodes indices $(i_1, i_2, \ldots; j_1, j_2, \ldots)$ associated with the added degrees of freedom (left in Figure 3). Nodes i_1 and j_1 coincide with primal mesh vertices i and j of the FV method.

To impose boundary conditions, we utilize a list of boundary edges. For each such edge that is not curved, as illustrated to the right in Figure 3, we associate boundary vertices i and j connected by the edge and boundary normals $\mathbf{n}_{ib}$ and $\mathbf{n}_{jb}$ (of equal length, so stored as one normal $\mathbf{n}_{ijb} = \mathbf{n}_{ib} + \mathbf{n}_{jb}$) associated with the intersection of the boundaries of control volumes K_i and K_j with the domain boundary. For curved boundaries, more nodes are needed along the boundary together with miscellaneous additional information, as discussed in Section 4.

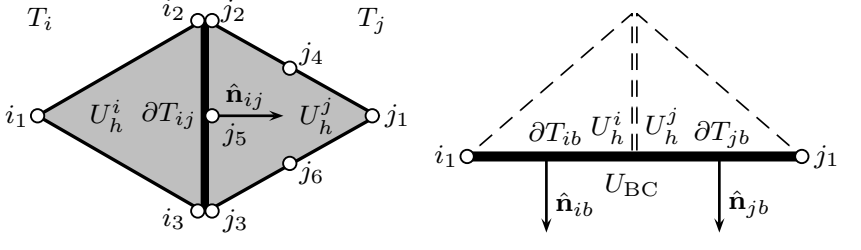

Fig. 3 Edge data structures with geometric data and spatial placement of the computational nodes. Left: The triangles T_i and T_j share the boundary ∂T_{ij}, and the outward unit normal of T_i on ∂T_{ij} is $\hat{\mathbf{n}}_{ij}$. Restrictions of the function $U_h \in \mathcal{V}_h^{d+2}$ onto T_i and T_j are denoted U_h^i and U_h^j. Right: The boundary edge is the union of ∂T_{ib} and ∂T_{jb}, and the domain boundary outward unit normals are $\hat{\mathbf{n}}_{ib}$ and $\hat{\mathbf{n}}_{jb}$. The domain boundary data, which is given by the boundary condition, is denoted u_{BC}.

A pseudo-code version of the algorithm can be found in the the article by Berggren *et al.* [6]; in essence, it is the same loops as for the FV code plus the volume integral of the DG scheme. Standard tabulated Gauss quadrature rules, as given, for example, in the book by Solin *et al.* [24], are used to evaluate the integrals, and the evaluation of the basis functions at these points are done once and for all in the beginning of the computation.

The numerically observed convergence rate s in $\|u-u_h\|_{L^2(\Omega)} \leq Ch^s$ for the model problem investigated in [6] is of optimal order for the vertex-centered scheme, that is, $s = p+1$. Also, as expected, the number of degrees of freedom necessary to attain a given error bound is substantially decreased by increasing the order of the method.

4 Curved Geometries

A higher order scheme, such as DG, also needs a higher-order representation of the geometry, and curved elements have to be introduced when curved objects, for example airfoils, are present in the computations. Bassi and Rebay [4] and Cockburn *et al.* [9], for example, has showed that a piecewise-linear approximation of the boundary does not yield satisfactory results. We also observed strong oscillatory solutions in our scheme when using piecewise-linear approximations of the boundary in combination with orders $p \geq 1$.

To account for the curved boundaries, we use a common finite-element strategy, namely polynomial approximations of the curved element edges combined with evaluations of the integrals on reference elements using coordinate transformations. The nodes marked a, b, and c to the left in Figure 4 are vertices in the primal mesh.

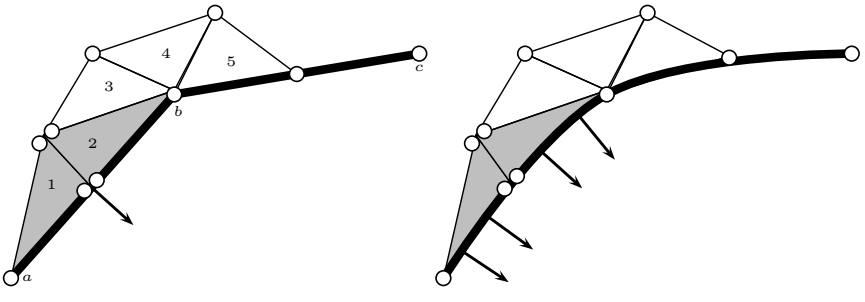

Fig. 4 Boundary elements. Left: The piecewise-linear boundary as obtained by constructing the dual mesh similarly as in the interior. Points a, b, and c are vertices on primal mesh. Triangle 1 is associated to macro element a and triangles $2, 3, 4$, and 5 are associated to macro element b. A boundary edge is marked in gray together with its associated normal. Right: The same part of the boundary as in the left image but taking into account the curvature of the boundary. Now a specific normal, each precomputed, is associated with each quadrature point on the boundary.

If dual meshes are constructed at the boundary in the same way as in the interior, the result will be as depicted to the left in Figure 4. This piecewise-linear boundary generated massive spurious oscillations at the boundary for $p \geq 1$. A simple strategy that yields a somewhat more accurate approximation of the boundary is to move the nodes on the dual mesh that are located between primal nodes (for instance the double nodes located in between nodes a and b in the left of Figure 4) so that they lie at the underlying curved boundary. This change did not significantly reduce the oscillations. Krivodonova's [18] simplified approach to represent a curved boundary was not successful either. However, a polynomial approximation of the boundary and associated normals at a sufficiently high order, as visualized in the right of Figure 4 and as shortly outlined below, worked well.

There will be at most one side that is curved in any sub triangle, since the other two triangle sides are located inside the domain. Now consider a boundary edge and the two associated curved triangles, T_i and T_j (for instance the gray triangles to the right in Figure 4). Denote by $\hat{T}$ the reference triangle with corners at $(0,0)$, $(1,0)$, and $(0,1)$. If we use the polynomial order p_b to approximate the boundary shape, and we use the polynomial order p for the basis functions of the elements, we will have *subparametric* elements if $p_b < p$, *isoparametric* elements if $p_b = p$, and *superparametric* elements if $p_b > p$. Our mesh generator is implemented such that any order p_b can be chosen, regardless of element order p. Numerical experiments indicate that a $p_b \geq 2$ is qualitatively sufficient for the calculations we have made for $p = 0,1,2,3$.

Denote by Φ the mapping from reference triangle $\hat{T}$ to any curved triangle T (T_i or T_j) and by Φ^{-1} the inverse mapping,

$$\Phi(\xi,\eta) = \begin{pmatrix} x \\ y \end{pmatrix}, \quad \Phi^{-1}(x,y) = \begin{pmatrix} \xi \\ \eta \end{pmatrix}. \tag{15}$$

Points $(\xi,\eta) \in \hat{T}$ are transformed into points $(x,y) \in T$ according to

$$x = \sum_{m=0}^{p_b} \sum_{n=0}^{p_b-m} \gamma_{mn} \xi^m \eta^n, \quad y = \sum_{m=0}^{p_b} \sum_{n=0}^{p_b-m} \delta_{mn} \xi^m \eta^n, \tag{16}$$

where p_b is the polynomial order of the boundary and γ_{mn}, δ_{mn} are constants that are computed in the preprocessor. The Jacobian J of Φ,

$$\begin{aligned} J(\xi,\eta) &= \nabla\Phi(\xi,\eta) \\ &= \sum_{m=0}^{p_b} \sum_{n=0}^{p_b-m} \begin{pmatrix} \frac{\partial}{\partial\xi}\gamma_{mn}\xi^m\eta^n & \frac{\partial}{\partial\eta}\gamma_{mn}\xi^m\eta^n \\ \frac{\partial}{\partial\xi}\delta_{mn}\xi^m\eta^n & \frac{\partial}{\partial\eta}\delta_{mn}\xi^m\eta^n \end{pmatrix} \\ &= \sum_{m=0}^{p_b} \sum_{n=0}^{p_b-m} \begin{pmatrix} \gamma_{mn} m \xi^{m-1}\eta^n & \gamma_{mn} n \xi^m \eta^{n-1} \\ \delta_{mn} m \xi^{m-1}\eta^n & \delta_{mn} n \xi^m \eta^{n-1} \end{pmatrix}, \end{aligned} \tag{17}$$

will not be constant for $p_b > 1$. The Jacobians are precomputed at each quadrature point, and are stored and used during computations. For a function F on a curved triangle T, we have

$$\int_T F \,\mathrm{d}V \;=\; \int_{\hat{T}} F \circ \boldsymbol{\Phi}\, |J| \,\mathrm{d}V, \tag{18}$$

where $|J|$ denotes the determinant of J. The gradient of a function g on T is transformed to corresponding function $\hat{g} = g \circ \boldsymbol{\Phi}$ on $\hat{T}$ according to

$$\nabla g = J^{-T} \nabla \hat{g}, \tag{19}$$

where the left- and right-hand sides are evaluated at corresponding points related by mapping $\boldsymbol{\Phi}$. Finite element basis functions ϕ_i on the curved triangle T are defined by the mapping $\phi_i = \hat{\phi}_i \circ \boldsymbol{\Phi}$, where $\hat{\phi}_i$ are the standard Lagrangian basis functions on the reference triangle $\hat{T}$. Let $\mathscr{F}_m$ be row m ($m = 1$, 2, or 3) of flux function (2)[1]. By expressions (18) and (19), it follows that row m of the second term (the volume integral term) in equation (5) can be written

$$\begin{aligned} \int_T \nabla \phi_i \cdot \mathscr{F}_m(U_h) \,\mathrm{d}V &= \int_{\hat{T}} J^{-T} \nabla \hat{\phi}_i \cdot \mathscr{F}_m(U_h \circ \boldsymbol{\Phi}) |J| \,\mathrm{d}V \\ &\approx \sum_{k=1}^{N_{ep}^T} w_k J(\xi_k, \eta_k)^{-T} \nabla \hat{\phi}_i(\xi_k, \eta_k) \cdot \mathscr{F}_m(U_h \circ \boldsymbol{\Phi}(\xi_k, \eta_k)) \, |J(\xi_k, \eta_k)| \end{aligned} \tag{20}$$

where w_k, (ξ_k, η_k), and N_{ep}^T are the weights, coordinates in $\hat{T}$, and number of evaluation points for some appropriate integration rule for triangles.

Let the curved side $\partial T \cap \partial \boldsymbol{\Omega}$ be the image of the restriction of the map $\boldsymbol{\Phi}$ to $\hat{I} = (0,1) \times \{0\}$ (the intersection of $\hat{T}$ with the x-axis). Also, let $\mathscr{F}_m^*(U_L, U_R, \hat{\mathbf{n}})$ be row m in the numerical flux function. The curved-boundary contribution from the first term in equation (5) can then be written

$$\begin{aligned} \int_{\partial T \cap \partial \Omega} \phi_i \mathscr{F}_m^*(U_L, U_R, \hat{\mathbf{n}}) \,\mathrm{d}s &= \int_{\hat{I}} \hat{\phi}_i \mathscr{F}_m^*(U_L \circ \boldsymbol{\Phi}, U_R \circ \boldsymbol{\Phi}, J^{-T} \hat{\mathbf{n}}) |J| \,\mathrm{d}s \\ &\approx \sum_{k=1}^{N_{ep}^I} z_k \hat{\phi}_i(\chi_k, 0) \, |J(\chi_k, 0)| \, \mathscr{F}_m^*(U_L \circ \boldsymbol{\Phi}(\chi_k, 0), U_R \circ \boldsymbol{\Phi}(\chi_k, 0), J(\chi_k, 0)^{-T} \hat{\mathbf{n}}_k) \end{aligned} \tag{21}$$

where N_{ep}^I is the number of evaluation points for the chosen integration rule on the unit interval, and z_k, χ_k are the corresponding weights and coordinates. Note that a specific normal $\mathbf{n}_k = |J(\chi_k, 0)| J(\chi_k, 0)^{-T} \hat{\mathbf{n}}_k$ is used for each evaluation point, and that all normals are precomputed in the preprocessor. Since the curved boundary part of the domain is typically much smaller than the rest of the domain, this extra storage and computational work is mostly negligible.

[1] The rows correspond to the mass, momentum, and energy equations, respectively. Note that the first and third rows are scalar equations, whereas the second row is a system of d equations.

5 Results

In Figure 5 we present results from one of the Mandatory Test Cases (MTC) of the ADIGMA project, MTC1. The figures depict the pressure coefficients using successively higher order elements, $p = 0, 1, 2, 3$, on the same mesh. The element sizes around the airfoil are visible in the $p = 0$ solution at the top left in Figure 5. Some overshoots are visible for the linear element solution, $p = 1$ (top right of Figure 5). At the bottom left of Figure 5, the $p = 2$ solution, small kinks are visible in the element on the leading edge, which vanish for the $p = 3$ solution at the bottom right of Figure 5.

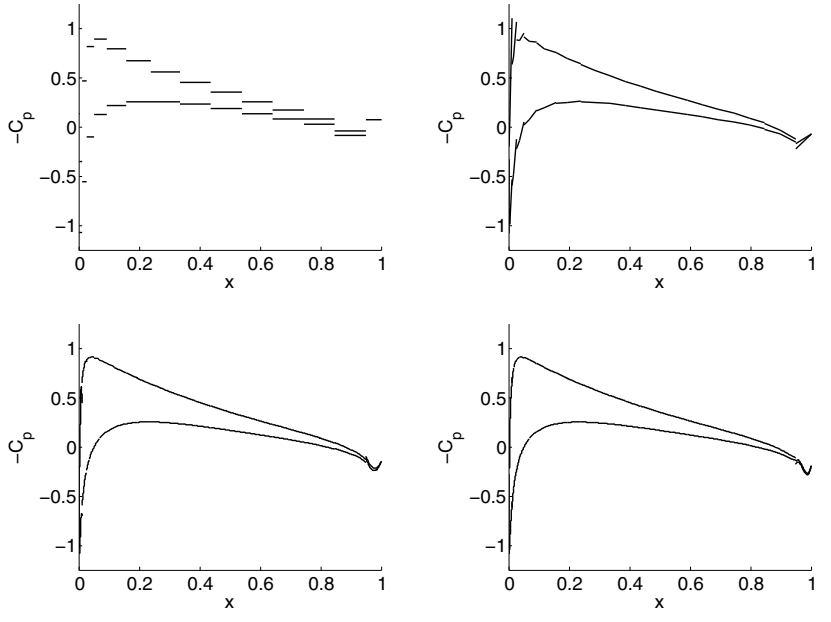

Fig. 5 MTC1 of the ADIGMA project: Pressure coefficients, 2𝔻 Euler, NACA0012, $M = 0.5$, $\alpha = 2.0^\circ$. The actual discontinuous functions, of order $p = 0, 1, 2, 3$, are plotted without any postprocessing or smoothing.

In Figure 6 we show two solutions for MTC2 on the same mesh, but with different order elements. The left picture of Figure 6 depicts the pressure coefficient for constant, $p = 0$ elements, solved with a second order central flux, i.e. a standard finite volume solution provided by Edge. Both shocks are rather poorly resolved and a finer mesh would be required. The right picture of Figure 6 depicts the pressure coefficient for linear, $p = 1$, elements. Note that the oscillations due to shocks are localized to elements adjacent to the shock, and that the oscillations do not spread out into the domain. No shock capturing technique is used. An *hp*-adaption along the lines previously developed for model problems [6] could be used to better resolve

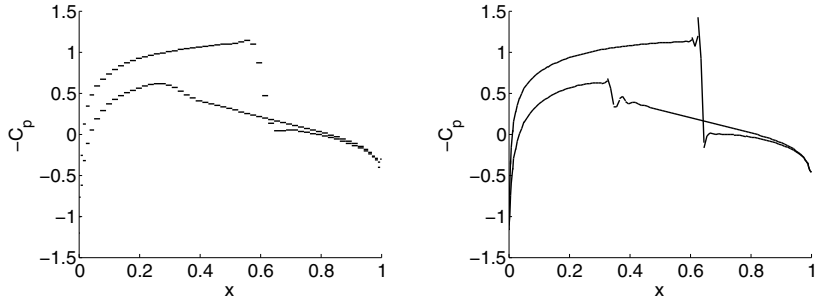

Fig. 6 MTC2 of the ADIGMA project: Pressure coefficients, 2𝔻 Euler, NACA0012, $M = 0.8$, $\alpha = 1.25°$. Left: $p = 0$, central flux (2^{nd} order). Right: $p = 1$, upwind flux (2^{nd} order). The actual discontinuous functions, of order $p = 0, 1$, are plotted without any postprocessing or smoothing.

the shock; another strategy would be to use the sub-cell shock capturing with artificial viscosity by Persson and Peraire [21]. The use of higher orders, $p = 2, 3$, for this transonic case and on this mesh appears to require a shock capturing technique; the solution blows up when approaching a steady-state solution.

6 Conclusions

The effort reported here constitutes a pioneering, first-ever implementation for the Euler equations of the current vertex-centered DG scheme. To the best of our knowledge, this is also the first time that a DG scheme has been implemented within the framework of an edge-based FV code of the type very common in the aeronautical industry.

Acceleration techniques such as local time stepping and agglomerated multigrid has been implemented successfully; the latter is further discussed in our other contribution to this volume [11].

The data structures needed for the current scheme are complex, but compact and memory efficient. The local polynomial approximations take place on the sub elements within the macro elements. Thus, the appropriate grid-size parameter h is the size of the sub elements, and *not* the size of the primal mesh elements. Since no fluxes need to be computed at the boundaries occurring strictly inside the macro elements, the current implementation will be significantly more memory efficient compared to a standard DG based on a mesh with comparable interpolation bound.

Artificial oscillations have been shown to stay localized around shocks. Nevertheless, shock handling will still need to be implemented in order to stabilize higher order approximations and improve the convergence properties of the solver. Another research issue is h-adaption in the current context, which may be more complicated to implement compared to the case for the standard DG scheme.

Although our results are encouraging, more research is needed to decisively determine the benefit of the vertex-centered DG approach. In particular, the method needs to be evaluated on viscous, $3\mathbb{D}$ and turbulent calculations. In addition, particular effort should be guided towards unsteady and aeroacoustic calculation, since the need for high accuracy everywhere in the computational domain may be even more crucial in such cases, compared to steady aerodynamic calculations.

References

1. Abgrall, R.: On essentially non-oscillatory schemes on unstructured meshes: analysis and implementation. J. Comput. Phys. 114(1), 45–58 (1994)
2. Abgrall, R.: Toward the ultimate conservative scheme: following the quest. J. Comput. Phys. 167(2), 277–315 (2001)
3. Barth, T.J., Jespersen, D.C.: The design and application of upwind schemes on unstructured meshes. In: 27th Aerospace Sciences Meeting, AIAA 89-0366, Reno, Nevada (1989)
4. Bassi, F., Rebay, S.: High-order accurate discontinuous finite element solution of the 2d Euler equations. J. Comp. Phys. 138, 251–285 (1997)
5. Berggren, M.: A vertex-centered, dual discontinuous Galerkin method. J. Comput. Appl. Math. 192(1), 175–181 (2006)
6. Berggren, M., Ekström, S.-E., Nordström, J.: A discontinuous Galerkin extension of the vertex-centered edge-based finite volume method. Commun. Comput. Phys. 5, 456–468 (2009)
7. Blazek, J.: Computational Fluid Dynamics, 2nd edn. Elsevier, Amsterdam (2005)
8. Catabriga, L., Coutinho, A.: Implicit SUPG solution of Euler equations using edge-based data structures. Comput. Meth. Appl. Mech. Eng. 191(32), 3477–3490 (2002)
9. Cockburn, B., Karniadakis, G.E., Shu, C.W.: Discontinuous Galerkin Methods: Theory, Computation and Applications. Springer, Heidelberg (1999)
10. Csík, A., Ricchiuto, M., Deconinck, H.: A Conservative Formulation of the Multidimensional Upwind Residual Distribution Schemes for General Nonlinear Conservation Laws. J. Comput. Phys. 179(1), 286–312 (2002)
11. Ekström, S.-E., Berggren, M.: Agglomeration multigrid for the vertex-centered dual discontinuous Galerkin method. In: Kroll, N., et al. (eds.) ADIGMA. NNFM, vol. 113, pp. 301–308. Springer, Heidelberg (2010)
12. Eliasson, P.: EDGE, a Navier–Stokes solver, for unstructured grids. Technical Report FOI-R-0298-SE, Swedish Defence Research Agency (2001)
13. Fezoui, L., Stoufflet, B.: A class of implicit upwind schemes for Euler simulations with unstructured meshes. J. Comput. Phys. 84(1), 174–206 (1989)
14. FOI. Edge - Theoretical Formulation. Technical Report FOI dnr 03-2870, Swedish Defence Research Agency (2007) ISSN-1650-1942
15. Fun3D. Fully Unstructured Navier–Stokes, `http://fun3d.larc.nasa.gov/`
16. Haselbacher, A., McGuirk, J.J., Page, G.J.: Finite volume discretization aspects for viscous flows on mixed unstructured grids. AIAA J. 37(2), 177–184 (1999)
17. Hirsch, C.: Numerical Computation of Internal and External Flows, vol. 2. Wiley, Chichester (1990)
18. Krivodonova, L., Berger, M.: High-order accurate implementation of solid wall boundary conditions in curved geometries. J. Comp. Phys. 211(2), 492–512 (2006)
19. Luo, H., Baum, J.D., Löhner, R.: Edge-based finite element scheme for the Euler equations. AIAA J. 32, 1182–1190 (1994)

20. Morton, K.W., Sonar, T.: Finite volume methods for hyperbolic conservation laws. Acta Numer. 16, 155–238 (2007)
21. Persson, P.-O., Peraire, J.: Sub-Cell shock Capturing for Discontinuous Galerkin Methods. AIAA Paper, 2006-112 (2006)
22. Schwamborn, D., Gerhold, T., Heinrich, R.: The DLR TAU-Code: Recent Applications in Research and Industry. In: Wesseling, P., Onate, E., Périaux, J. (eds.) ECCOMAS CFD 2006 (2006)
23. Smith, T.M., Ober, C.C., Lorber, A.A.: SIERRA/Premo–A New General Purpose Compressible Flow Simulation Code. In: AIAA 32nd Fluid Dynamics Conference, St. Louis (2002)
24. Solin, P., Segeth, K., Dolezel, I.: Higher Order Finite Element Methods. Taylor & Francis Ltd., Abington (2003)