COMMUNICATIONS IN COMPUTATIONAL PHYSICS
Vol. **5**, No. 2-4, pp. 456-468

Commun. Comput. Phys.
February 2009

A Discontinuous Galerkin Extension of the Vertex-Centered Edge-Based Finite Volume Method

Martin Berggren[1], Sven-Erik Ekström[2,*] and Jan Nordström[2]

[1] *Department of Computing Science, Umeå University, SE-901 87 Umeå Sweden.*
[2] *Division of Scientific Computing, Department of Information Technology, Uppsala University, Box 337, SE-751 05 Uppsala, Sweden.*

Received 1 October 2007; Accepted (in revised version) 15 February 2008

Available online 1 August 2008

Abstract. The finite volume (FV) method is the dominating discretization technique for computational fluid dynamics (CFD), particularly in the case of compressible fluids. The discontinuous Galerkin (DG) method has emerged as a promising high-accuracy alternative. The standard DG method reduces to a cell-centered FV method at lowest order. However, many of today's CFD codes use a vertex-centered FV method in which the data structures are edge based. We develop a new DG method that reduces to the vertex-centered FV method at lowest order, and examine here the new scheme for scalar hyperbolic problems. Numerically, the method shows optimal-order accuracy for a smooth linear problem. By applying a basic *hp*-adaption strategy, the method successfully handles shocks. We also discuss how to extend the FV edge-based data structure to support the new scheme. In this way, it will in principle be possible to extend an existing code employing the vertex-centered and edge-based FV discretization to encompass higher accuracy through the new DG method.

AMS subject classifications: 65N22, 65N30 , 65N50

Key words: Discontinuous Galerkin methods, finite volume methods, dual mesh, vertex-centered, edge-based, CFD.

1 Introduction

The finite volume (FV) method is currently the most widely used approach to discretize the equations of aerodynamics. The method balances exactly—with respect to the chosen numerical flux—the discrete values of mass, momentum, and energy between each control volume. The type of control volumes together with the choice of numerical flux

*Corresponding author. *Email addresses:* martin.berggren@cs.umu.se (M. Berggren), sven-erik.ekstrom@it.uu.se (S.-E. Ekström), jan.nordstrom@it.uu.se (J. Nordström)

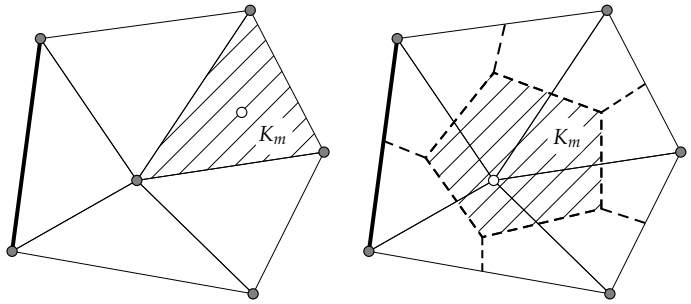
determines which particular flavor of the FV method that is employed. Unstructured meshes are supported naturally by these methods, which allows for the treatment of flows around geometrically complex bodies.

The accuracy of FV methods is typically limited to first or second order. Efforts to increase the accuracy of the basic method include the so-called high-resolution schemes (MUSCL, ENO, WENO), which attain better flux approximations through extrapolation from directions where the solution is smooth. These schemes modestly increase the memory requirements, but the computational complexity grows with the order, since the improved accuracy relies on enlarging the width of the computational stencil. The regularity requirements on the mesh are thus likely to be high in order to obtain improved results.

A different approach to increase the accuracy is the discontinuous Galerkin (DG) method. It is a finite element method that does not explicitly enforce continuity between the elements as in a classic finite element method, but instead imposes a coupling between the solution at different elements with the use of numerical fluxes, as in the FV method. The DG method reduces to a FV method at lowest order, and can thus be viewed as a generalization of the FV method to higher orders. The increased order is not the result of an extrapolation procedure, as in the high-resolution schemes, but stems from a local approximation of the differential operator. The computational stencil of DG methods thus remains local regardless of order, and the quality of approximation can be expected not to depend as much on the mesh regularity as for the high-resolution FV schemes. On the other hand, the computational complexity and memory requirements increase sharply with order for the DG methods. Nevertheless, since a coarser mesh can be used, a higher-order DG method requires considerably less degrees of freedom to attain a solution with a given error bound, compared with a FV method.

There has been a strong development of the original DG method (Reed & Hill [24]) since the early nineties; Cockburn et al. [9] review the state of the art at the turn of the century. Hesthaven and Warburton give a thorough introduction to DG methods in their recent book [19]. Currently there is a coordinated effort in Europe, in which we participate, through the EU research project ADIGMA [3], which involves the development and assessment of different higher order methods such as DG and residual distribution schemes [2, 10] for the next generation of CFD software aimed at the aeronautical industry.

Since DG is a generalization of the FV method, it is tempting to extend existing FV codes to encompass a DG method, in order to avoid a complete rewrite of large and sophisticated software systems. A serious hurdle for such a strategy is that the standard DG method is a higher-order version of the *cell-centered* FV method in which the control volumes coincide with the mesh cells (Fig. 1a), whereas many of today's codes are *vertex-centered* where the control volumes are constructed from a dual mesh, consisting in two dimensions of polygons surrounding each vertex in the original primal mesh (Fig. 1b). Some examples of vertex-centered FV codes are DLR-Tau [25], Edge [12], Eugenie [15], Fun3D [17], and Premo [26].



(a) Cell-centered. (b) Vertex-centered.

Figure 1: Control volume schemes for finite volume methods. The computational node (white circle) is either associated with mesh cell center (a) or the mesh vertex (b). The solution value in the node is the average over the control volume K_m.

The vertex-centered approach has a number of particular features that may help to explain its current popularity. Comparing a cell-centered and a vertex-centered scheme on the same mesh, the latter has fewer degrees of freedom—about half the total memory foot-print—and more fluxes per unknown, as mentioned by Blazek [6]. Abgrall [1] argues that reconstruction schemes are more easily formulated for vertex-centered control volumes. Moreover, as opposed to cell-centered schemes, 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 FV code concerns the residual computations. A solver using the vertex-centered schemes may be implemented to support what Haselbacher et al. [18] call *grid transparency*: the solver loops over all edges in the mesh to assemble the residual, 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 FV methods, we refer to Blazek's book [6] and the recent review by Morton & Sonar [23]. Also other schemes than FV use a vertex-centered control volume or loop over edges. Some examples are residual distribution schemes [2, 10], edge-based finite elements [22], and edge-based SUPG [8].

In the context of a linear first-order hyperbolic model problem, Berggren [5] introduced a vertex-centered and edge-based DG method. Below, we further explore the properties of this DG method for higher-order elements, and for a nonlinear problem with a shock.

2 The discontinuous Galerkin method

The target application for the proposed scheme is aeronautical CFD, where computations of steady states are particularly prominent. Therefore we here consider the steady

hyperbolic model problem

$$\begin{aligned} \nabla\cdot\mathcal{F}(u)+\gamma u &= f \qquad \text{in } \Omega, \\ u &= g \qquad \text{on } \Gamma^-, \end{aligned} \tag{2.1}$$

where u is the scalar unknown quantity, $\mathcal{F}$ is a flux function, $\gamma \geq 0$ is a constant, f is a source term in the computational domain $\Omega \subset \mathbb{R}^2$, and g defines u on the inflow boundary Γ^-. We divide the domain, Ω, into a set of non-overlapping control volumes K_m such that $\overline{\Omega} = \bigcup_{m=1}^{M} \overline{K}_m$. The finite-dimensional space $\mathcal{V}_h$ of numerical solutions to Eq. (2.1) comprises functions that are continuous on each K_m but in general contain jump discontinuities at the boundaries between control volumes. The restriction on each K_m of functions in $\mathcal{V}_h$ are polynomials for the standard DG method. Here we consider a different choice of functions, as described in Section 3. Each $v_h \in \mathcal{V}_h$ can be expanded as

$$v_h(\mathbf{x}) = \sum_{m=1}^{M}\sum_{i=1}^{N_m} v_i^m \phi_i^m(\mathbf{x}) = \sum_{i=1}^{N_{\text{dof}}} v_i \phi_i(\mathbf{x}),$$

where 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 globally numbered.

The DG method is obtained by multiplying Eq. (2.1) by a test function $v_h \in \mathcal{V}_h$, integrating over each control volume, integrating by parts, and introducing the numerical flux $\mathcal{F}^*$ on the boundaries. This yields that $u_h \in \mathcal{V}_h$ solves the variational problem

$$\int_{\partial K_m} v_L \mathcal{F}^*(u_L,u_R,\hat{\mathbf{n}})\,\mathrm{d}s - \int_{K_m} \nabla v_h\cdot\mathcal{F}(u_h)\,\mathrm{d}V + \gamma\int_{K_m} v_h u_h\,\mathrm{d}V = \int_{K_m} v_h f\,\mathrm{d}V, \quad \forall v_h\in\mathcal{V}_h,\ \forall K_m\subset\Omega,$$

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 condition is imposed weakly through the numerical flux. In our implementation we use the Roe numerical flux,

$$\mathcal{F}^*(u_L,u_R,\hat{\mathbf{n}}) = \frac{1}{2}\Big(\hat{\mathbf{n}}\cdot\mathcal{F}(u_L)+\hat{\mathbf{n}}\cdot\mathcal{F}(u_R) - \big|\hat{\mathbf{n}}\cdot\mathcal{F}'(\bar{u})\big|(u_R-u_L)\Big), \tag{2.2}$$

where $\bar{u}$ is the so called Roe average, a quantity that satisfies the mean-value property

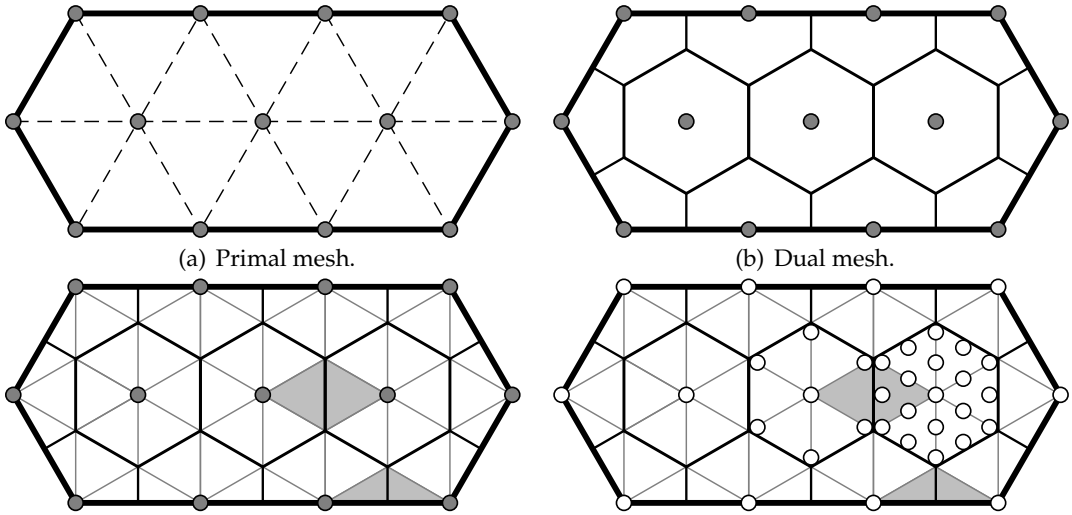
$$\hat{\mathbf{n}}\cdot\mathcal{F}(u_R) = \hat{\mathbf{n}}\cdot\mathcal{F}(u_L)+\hat{\mathbf{n}}\cdot\mathcal{F}'(\bar{u})(u_R-u_L),$$

which makes the Roe flux a nonlinear generalization of upwinding.

The stability of the above scheme for linear advection problems relies on the upwind flux, which generates a positive definite matrix representation of the operator, including boundary conditions [5], [14, pp. 359-360].

3 The vertex-centered macro element

The finite elements of the standard (cell-centered) DG method consist of 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 method we use another choice of control volumes K_m, defined on a so-called dual mesh.



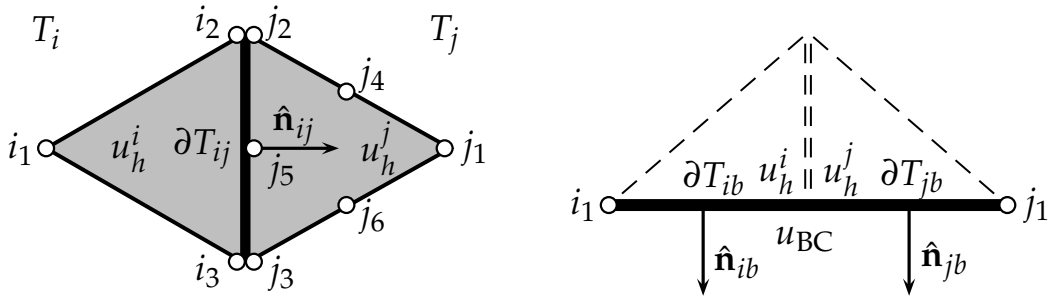
(a) Primal mesh. (b) Dual mesh.

(c) Triangulated dual mesh. (d) Macro elements.

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

A preprocessor constructs the dual mesh and necessary data structures; Fig. 2 illustrates the procedure. From the primal mesh, Fig. 2a, the preprocessor constructs the dual mesh shown in Fig. 2b. A dual mesh can be constructed in several ways, as discussed by Barth [4]; here we choose just to connect the centroids of adjacent triangles to each other with a new edge. Although Fig. 2 shows a uniform mesh, dual meshes can be constructed for any nondegenerate mesh. Next we triangulate the dual mesh, Fig. 2c, 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



(a) An generic edge . (b) A boundary edge.

Figure 3: Edge data structures with geometric data and spatial placement of the computational nodes. (a) The triangles T_i and T_j share the boundary ∂T_{ij}, and the outward unit normal ofT_i on ∂T_{ij} is $\hat{\mathbf{n}}_{ij}$. Restrictions of the function $u_h \in \mathcal{V}_h$ onto T_i and T_j are denoted u_h^i and u_h^j. (b) 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}.

type and order may be different on different dual cells.

Fig. 2d shows 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 Eq. (2.1) is to march an unsteady version of the equation to steady state using an explicit Runge-Kutta scheme. This strategy is often combined with convergence acceleration strategies such as local time stepping and multigrid. The 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 $\mathcal{V}_h$. The ith component of the residual is

$$\mathbf{r}_i(u_h) = \int_{\partial K_m} \phi_i \mathcal{F}^*(u_L, u_R, \hat{\mathbf{n}})\,\mathrm{d}s - \int_{K_m} \nabla\phi_i \cdot \mathcal{F}(u_h)\,\mathrm{d}V,$$

where K_m is the macro element containing the support of ϕ_i. Here we assume $\gamma=0$ and $f=0$ for simplicity.

Pointers to the degrees of freedom for $\mathcal{V}_h$ 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. This information is all 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,$\cdots$; j_1, j_2,$\cdots$)

associated with the added degrees of freedom (Fig. 3a). Nodes i_1 and j_1 coincide with primal mesh vertices i and j of the FV method.

To set boundary conditions, we utilize a list of boundary edges. For each such edge, as illustrated in Fig. 3b, we associate boundary vertices i and j connected by the edge and boundary normals $\mathbf{n}_{ib}$ and $\mathbf{n}_{jb}$ associated with the intersection of the boundaries of control volumes K_i and K_j with the domain boundary. For higher orders, more nodes are needed along the boundary. Additional information needs to be supplied for a curved boundary, a case that is beyond the scope of the current discussion.

The pseudo-codes for a standard edge-based FV residual computation, and the new extended edge-based DG version, are presented in Algorithms 3.1 and 3.2 respectively, with common line numbering. The necessary computational quantities are defined in Fig. 3, and in Algorithm 3.2 we use $\text{dof}(\mathcal{D}) = \{k \,|\, \mathcal{D} \subset \text{supp}(\phi_k)\}$ to denote the degrees of freedom for a basis function with support in the region $\mathcal{D}$. The residual contribution functions are

$$\begin{aligned}
&\texttt{CompFluxFV}(\partial T, u_L, u_R, \hat{\mathbf{n}}) = \int_{\partial T} \mathcal{F}^*(u_L, u_R, \hat{\mathbf{n}})\,\mathrm{d}s,\\
&\texttt{CompFluxDG}(\partial T, u_L, u_R, \hat{\mathbf{n}}, v) = \int_{\partial T} v\mathcal{F}^*(u_L, u_R, \hat{\mathbf{n}})\,\mathrm{d}s,\\
&\texttt{CompVolumeDG}(T, u, v) = -\int_T \nabla v \cdot \mathcal{F}(u)\,\mathrm{d}V.
\end{aligned}$$

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.

The computation of the residual consists of three stages,

1. a loop over all edges in the mesh to compute residual contributions from the equation (lines 2-12),
2. a loop over all boundary edges to set boundary condition data (lines 13-19),
3. a loop over all boundary edges to compute residual contributions from the boundary conditions (lines 20-28).

Again note that the DG method reduces to the FV method for constant basis functions, that is, Algorithm 3.2 is then equivalent to Algorithm 3.1 since the DG flux integral computation $\texttt{CompFluxDG}(\partial T, u_L, u_R, \hat{\mathbf{n}}, 1)$ is equal to its FV counterpart $\texttt{CompFluxFV}(\partial T, u_L, u_R, \hat{\mathbf{n}})$, and the DG volume integral computation $\texttt{CompVolumeDG}(T, u, 1)$ is equal to zero. We use Gauss quadrature to evaluate all integrals. The evaluation of the basis functions in the integration points is done once, in the preprocessor, on a reference element. When coding the algorithm, several performance-enhancing modifications can be done, for example at lines 5-6 of Algorithm 3.1. Since $\texttt{CompFluxFV}(\partial T_{ij}, u_h^i, u_h^j, \hat{\mathbf{n}}_{ij})$ is equal to $-\texttt{CompFluxFV}(\partial T_{ij}, u_h^j, u_h^i, -\hat{\mathbf{n}}_{ij})$, the numerical flux only has to be computed once. At the corresponding lines of Algorithm 3.2, a similar simplification can be done since the $\mathcal{F}^*(u_L, u_R, \hat{\mathbf{n}})$ part of the `CompFluxDG` integral has the same property for all nodes of the two triangles at a given quadrature point. Another

Algorithm 3.1: Edge-Based FV Residual.

```
1:  r = 0
2:  for all edges e do
3:      i,j,n̂_ij,∂T_ij ← GetEdgeInfo(e)
4:
5:      r_i ← r_i + CompFluxFV(∂T_ij,u_h^i,u_h^j,n̂_ij)
6:      r_j ← r_j + CompFluxFV(∂T_ij,u_h^j,u_h^i,−n̂_ij)
7:
8:
9:
10:
11:
12: end for
13: for all boundaries B do
14:     for all boundary edges e ∈ B do
15:         i,j,n̂_ib,n̂_jb,∂T_ib,∂T_jb ← GetEdgeInfo(e)
16:         u_BC ← SetBCData(∂T_ib,u_h^i,n̂_ib)
17:         u_BC ← SetBCData(∂T_jb,u_h^j,n̂_jb)
18:     end for
19: end for
20: for all boundaries B do
21:     for all boundary edges e ∈ B do
22:         i,j,n̂_ib,n̂_jb,∂T_ib,∂T_jb ← GetEdgeInfo(e)
23:
24:         r_i ← r_i + CompFluxFV(∂T_ib,u_h^i,u_BC,n̂_ib)
25:         r_j ← r_j + CompFluxFV(∂T_jb,u_h^j,u_BC,n̂_jb)
26:
27:     end for
28: end for
```

Algorithm 3.2: Edge-Based DG Residual.

```
1:  r = 0
2:  for all edges e do
3:      i,j,n̂_ij,∂T_ij,T_i,T_j ← GetEdgeInfo(e)
4:      for all m ∈ dof(T_i) ∩ dof(∂T_ij) and
              n ∈ dof(T_j) ∩ dof(∂T_ij) do
5:          r_m ← r_m + CompFluxDG(∂T_ij,u_h^i,u_h^j,n̂_ij,φ_m)
6:          r_n ← r_n + CompFluxDG(∂T_ij,u_h^j,u_h^i,−n̂_ij,φ_n)
7:      end for
8:      for all m ∈ dof(T_i) and n ∈ dof(T_j) do
9:          r_m ← r_m + CompVolumeDG(T_i,u_h^i,φ_m)
10:         r_n ← r_n + CompVolumeDG(T_j,u_h^j,φ_n)
11:     end for
12: end for
13: for all boundaries B do
14:     for all boundary edges e ∈ B do
15:         i,j,n̂_ib,n̂_jb,∂T_ib,∂T_jb ← GetEdgeInfo(e)
16:         u_BC ← SetBCData(∂T_ib,u_h^i,n̂_ib)
17:         u_BC ← SetBCData(∂T_jb,u_h^j,n̂_jb)
18:     end for
19: end for
20: for all boundaries B do
21:     for all boundary edges e ∈ B do
22:         i,j,n̂_ib,n̂_jb,∂T_ib,∂T_jb ← GetEdgeInfo(e)
23:         for all m ∈ dof(∂T_ib) and n ∈ dof(∂T_jb) do
24:             r_m ← r_m + CompFluxDG(∂T_ib,u_h^i,u_BC,n̂_ib,φ_m)
25:             r_n ← r_n + CompFluxDG(∂T_jb,u_h^j,u_BC,n̂_jb,φ_n)
26:         end for
27:     end for
28: end for
```

note is that some codes, for example Edge, loop over boundary nodes and not boundary edges, but to keep a consistency with the interior treatment of edges, we choose to loop over boundary edges.

4 Numerical results

We study a linear and a nonlinear model problem of the form (2.1). In both cases, the exact solution is known explicitly. The linear model problem is used to numerically determine the convergence rate and compare the number of degrees of freedom required to reach a certain error level, for the different orders of approximation. To demonstrate the method's ability to handle shocks, we consider Burgers' equation and apply a simple *hp*-adaption strategy to handle the shock region.

Table 1: The $L^2(\Omega)$-error, $\|u-u_h\|_{L^2(\Omega)}$, numerical order of convergence, s, and the number of degrees of freedom N_{dof}, for the linear model problem, for each mesh in a sequence of six successively refined meshes, using macro elements based on Lagrange triangles of different orders p.

	constant, $p=0$			linear, $p=1$		
$h_{\max}$	$\|u-u_h\|_{L^2(\Omega)}$	s	N_{dof}	$\|u-u_h\|_{L^2(\Omega)}$	s	N_{dof}
h_0	1.8558×10^{-1}	–	185	3.5491×10^{-3}	–	1249
$2^{-1}h_0$	1.1685×10^{-1}	0.67	697	8.5294×10^{-4}	2.06	4793
$2^{-2}h_0$	6.8506×10^{-2}	0.77	2705	2.0608×10^{-4}	2.05	18769
$2^{-3}h_0$	3.8030×10^{-2}	0.85	10657	5.0122×10^{-5}	2.04	74273
$2^{-4}h_0$	2.0356×10^{-2}	0.90	42305	1.2322×10^{-5}	2.02	295489
$2^{-5}h_0$	1.0660×10^{-2}	0.93	168577	3.0512×10^{-6}	2.01	1178753

	quadratic, $p=2$			cubic, $p=3$		
$h_{\max}$	$\|u-u_h\|_{L^2(\Omega)}$	s	N_{dof}	$\|u-u_h\|_{L^2(\Omega)}$	s	N_{dof}
h_0	2.9501×10^{-4}	–	3337	5.8787×10^{-6}	–	6449
$2^{-1}h_0$	4.3033×10^{-5}	2.78	12905	3.3308×10^{-7}	4.14	25033
$2^{-2}h_0$	5.8880×10^{-6}	2.87	50737	1.9522×10^{-8}	4.09	98609
$2^{-3}h_0$	7.7203×10^{-7}	2.93	201185	1.1674×10^{-9}	4.06	391393
$2^{-4}h_0$	1.0202×10^{-7}	2.92	801217	7.0708×10^{-11}	4.05	1559489
$2^{-5}h_0$	1.3564×10^{-8}	2.91	3197825	4.3273×10^{-12}	4.03	6225793

The linear model problem

The linear model problem of advection-reaction type is given by setting, in Eq. (2.1), $\mathcal{F}(u)=\boldsymbol{\beta}u$, with $\boldsymbol{\beta}=\left[\cos\frac{2\pi}{9},\sin\frac{2\pi}{9}\right]^{\mathrm{T}}$, $\gamma=1$, and $f=0$. The boundary conditions are given by

$$g(x,y)=\begin{cases}\sin 2\pi x, & x\in[0,1], \quad y=0,\\ -\exp\left(-\gamma y\sqrt{\beta_f^2+1}\right)\sin 2\pi\beta_f y, & x=0, \quad y\in(0,1],\end{cases}$$

where $\beta_f=\beta_1/\beta_2$. The expression above for $x=0$, $y\in(0,1]$ is derived so that the solution in $(0,1)\times(0,1)$ coincides with the one obtained by solving the same equation in the upper half plane subject to the inflow boundary condition $\sin 2\pi x$ on the x-axis. Note that the Roe average $\bar{u}$ in (2.2) does not have to be evaluated since $\mathcal{F}'(\bar{u})=\boldsymbol{\beta}$.

Table 1 presents the results of the convergence study. To solve the resulting linear system, the sparse direct solver SuperLU [11] was used. The problem is solved on each mesh of a sequence of six successively refined meshes, and for each implemented type of macro element. The numerically observed convergence rate s in $\|u-u_h\|_{L^2(\Omega)}\leq Ch^s$ is of optimal order, that is, $s=p+1$. Fig. 4 depicts the $L^2(\Omega)$-error as a function of the number

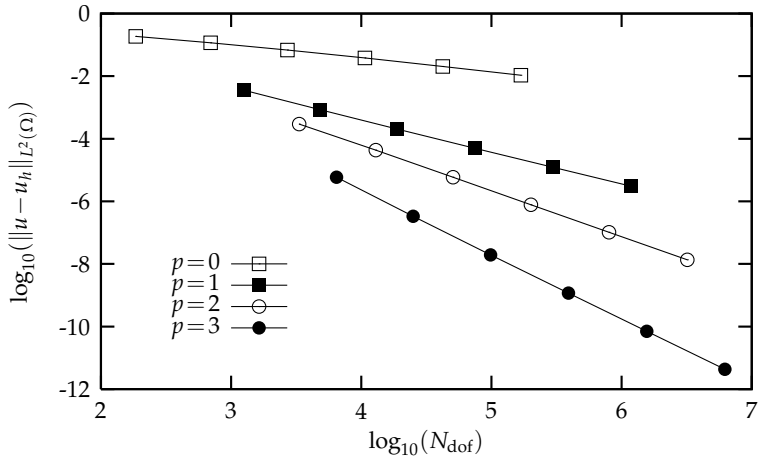


Figure 4: The $L^2(\Omega)$-error, $\|u-u_h\|_{L^2(\Omega)}$, depending on the number of degrees of freedom, N_{dof}, for macro elements based on Lagrange triangles of different orders p.

of degrees of freedom N_{dof}. The figure shows that an increase of the order of the method substantially reduces the number of degrees of freedom needed to compute a solution with a given error.

The nonlinear model problem

This model problem is chosen to assess the method's performance for a nonlinear case that develops a shock, and to demonstrate how *hp*-adaption can be utilized in the method. We consider the classic inviscid Burgers' equation in one dimension, in which the time t is viewed as a second space variable y:

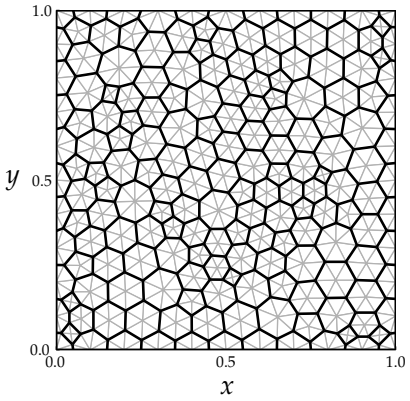
$$\frac{1}{2}(u^2)_x+u_y=0 \quad \text{in } \Omega.$$

Thus in Eq. (2.1), $\mathcal{F}(u)=\frac{1}{2}\left[u^2,2u\right]^{\mathrm{T}}$, $\gamma=0$, and $f=0$. The boundary conditions are given by

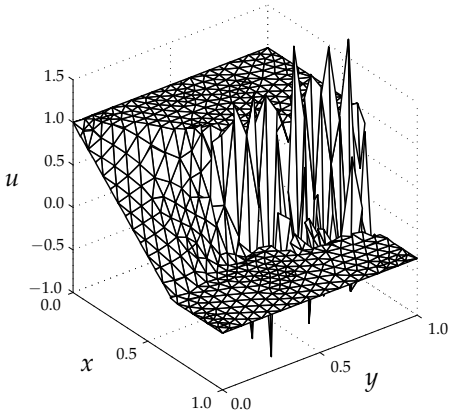
$$g(x,y)=\begin{cases} 1-2x, & x\in(0,2/3], \quad y=0, \\ -1/3, & x\in(2/3,1), \quad y=0, \\ 1, & x=0, \quad y\in(0,1], \\ -1/3, & x=1, \quad y\in(0,1]. \end{cases}$$

The Roe average $\bar{u}$ in (2.2) is $\bar{u}=(u_L+u_R)/2$ and $\mathcal{F}'(\bar{u})=[\bar{u},1]^{\mathrm{T}}$.

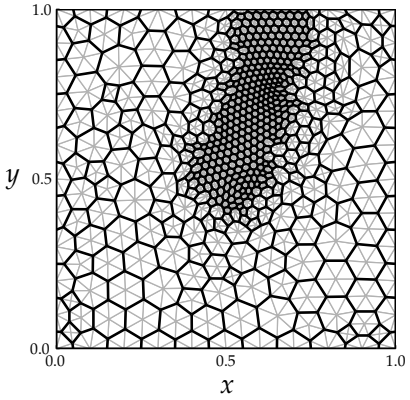
Fig. 5 presents the different stages of the solution process, using *hp*-adaption. The numerical solution using linear macro elements and the mesh of Fig. 5a is shown in Fig. 5b. Artificial oscillations are clearly visible locally on the elements covering the shock. The



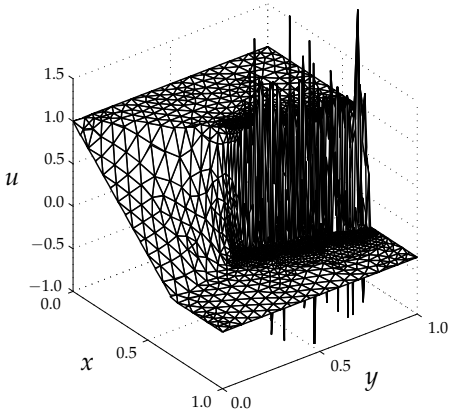
(a) Initial dual mesh.



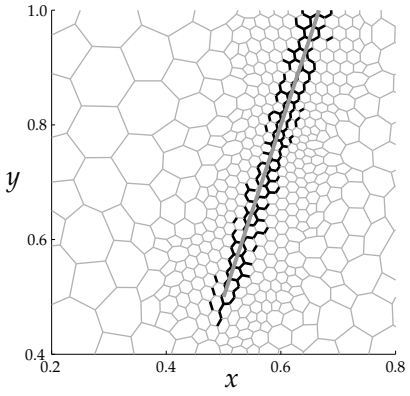
(b) Solution on initial dual mesh.



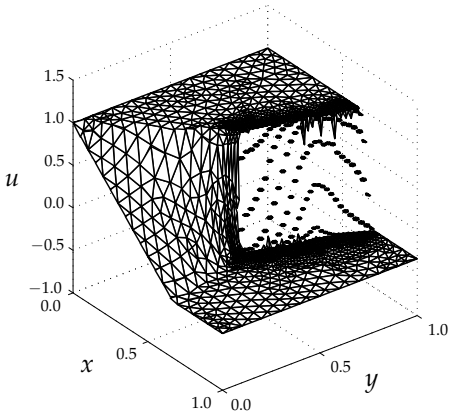
(c) Dual mesh after two h-refinements.



(d) Solution on dual mesh after two h-refinements.



(e) Close-up of the shock. Edges marked by the shock detector (black) and exact shock location (gray line).



(f) Solution on dual mesh after hp-adaption.

Figure 5: Process of solving the nonlinear model problem with initially linear macro elements, using hp-adaption.

shock detector by Krivodonova et al. [21] was then used to mark the elements in the vicinity of the shock. The marked elements are then h-refined, by refining the corresponding cells of the primal mesh, and then updating the dual mesh accordingly.

Next, a new solution is computed on the new mesh and the h-adaption is performed once more, resulting in the mesh shown in Fig. 5c. The solution on this twice refined mesh is shown in Fig. 5d. Fig. 5e shows a close-up of the shock region with the marked edges that were found by the shock detector shown in black. The gray straight line marks the exact location of the shock.

Finally, the orders of the macro elements marked in Fig. 5e are reduced to constants, and the solver is iterated to steady state. This effectively removes the artificial oscillations, as illustrated by Fig. 5f. The presented shock handling, by hp-adaption, is easy to implement and reduces the adverse effects of the reduced order on elements covering the shock. As with other numerical methods, lowering the order in the vicinity of the shocks, to avoid oscillations, provides a pollution effect in general (a rare exception is found in [20]). Thus for the Euler equations, for instance, we cannot expect better than first-order accuracy downstream of a shock, see [7, 13]. Alternative approaches for shock capturing, such as the use of limiters or artificial viscosity, are out of the scope of this article but will be studied in the future.

5 Conclusions

The new vertex-centered edge-based DG method shows great promise; besides sharing the advantages of the standard DG, the method admits previously incompatible FV software to be extended using higher-order DG. The numerical convergence rate is optimal for the chosen smooth linear model problem. Artificial oscillations have been shown to stay localized around discontinuities. Moreover, the oscillations can successfully be treated by using a shock detector and hp-adaption.

Acknowledgments

The authors were supported in part by the ADIGMA project [3] and the Graduate School in Mathematics and Computing, FMB [16].

References

[1] R. Abgrall. On essentially non-oscillatory schemes on unstructured meshes: analysis and implementation. *J. Comput. Phys.*, 114(1):45–58, 1994.

[2] R. Abgrall. Toward the ultimate conservative scheme: following the quest. *J. Comput. Phys.*, 167(2):277–315, 2001.

[3] ADIGMA. A European project on the development of adaptive higher order variational methods for aerospace applications. `http://www.dlr.de/`.

[4] T. J. Barth and D. C. Jespersen. The design and application of upwind schemes on unstructured meshes. In *27th Aerospace Sciences Meeting*, AIAA 89-0366, Reno, Nevada, 1989.
[5] M. Berggren. A vertex-centered, dual discontinuous Galerkin method. *J. Comput. Appl. Math.*, 192(1):175–181, 2006.
[6] J. Blazek. *Computational Fluid Dynamics*. Elsevier, Amsterdam, second edition, 2005.
[7] J. Casper and M. H. Carpenter. Computational Considerations for the Simulation of Shock-Induced Sound. *SIAM J. Sci. Comput.*, 19(3):813–828, 1998.
[8] L. Catabriga and A. Coutinho. Implicit SUPG solution of Euler equations using edge-based data structures. *Comput. Meth. Appl. Mech. Eng.*, 191(32):3477–3490, 2002.
[9] B. Cockburn, G. Karniadakis, and C.-W. Shu, editors. *Discontinuous Galerkin Methods: Theory, Computation and Applications*, volume 11 of *Lect. Notes in Comput. Sc. and Eng.*, Berlin, 2000. Springer.
[10] A. Csk, M. Ricchiuto, and H. Deconinck. A Conservative Formulation of the Multidimensional Upwind Residual Distribution Schemes for General Nonlinear Conservation Laws. *J. Comput. Phys.*, 179(1):286–312, 2002.
[11] J. W. Demmel, J. R. Gilbert, and S. X. Xiaoye. SuperLU users' guide. Technical Report LBNL-44289, Ernest Orlando Lawrence Berkeley National Laboratory, 1999.
[12] P. Eliasson. EDGE, a Navier–Stokes solver, for unstructured grids. Technical Report FOI-R-0298-SE, Swedish Defence Research Agency, 2001.
[13] B. Engquist and B. Sjgreen. The Convergence Rate of Finite Difference Schemes in the Presence of Shocks. *SIAM J. Numer. Anal.*, 35(6):2464–2485, 1998.
[14] M. Feistauer, J. Felcman, and I. Straškraba. *Mathematical and Computational Methods for Compressible Flow*. Oxford University Press, 2003.
[15] L. Fezoui and B. Stoufflet. A class of implicit upwind schemes for euler simulations with unstructured meshes. *J. Comput. Phys.*, 84(1):174–206, 1989.
[16] FMB. The Graduate School in Mathematics and Computing. `http://www.math.uu.se/fmb/`.
[17] Fun3D. Fully Unstructured Navier–Stokes. `http://fun3d.larc.nasa.gov/`.
[18] A. Haselbacher, J. J. McGuirk, and G. J. Page. Finite volume discretization aspects for viscous flows on mixed unstructured grids. *AIAA J.*, 37(2):177–184, 1999.
[19] J. S. Hesthaven and T. Warburton. *Nodal Discontinuous Galerkin Methods: Algorithms, Analysis and Applications*. Springer-Verlag New York Inc., 2007.
[20] G. Kreiss, G. Efraimsson, and J. Nordström. Elimination of First Order Errors in Shock Calculations. *SIAM J. Numer. Anal.*, 38:1986–1998, 2001.
[21] L. Krivodonova, J. Xin, J. F. Remacle, N. Chevaugeon, and J. E. Flaherty. Shock detection and limiting with discontinuous Galerkin methods for hyperbolic conservation laws. *Appl. Numer. Math.*, 48:323–338, 2004.
[22] H. Luo, J. D. Baum, and R. Lhner. Edge-based finite element scheme for the Euler equations. *AIAA J.*, 32:1182–1190, June 1994.
[23] K. W. Morton and T. Sonar. Finite volume methods for hyperbolic conservation laws. *Acta Numer.*, 16:155–238, 2007.
[24] W.H. Reed and T.R. Hill. Triangular mesh methods for the neutron transport equation. Technical Report LA-UR-73-479, Los Alamos Scientific Laboratory, Los Alamos, 1973.
[25] D. Schwamborn, T. Gerhold, and R. Heinrich. The DLR TAU-Code: Recent Applications in Research and Industry. In P. Wesseling, E. O nate, and J. Périaux, editors, *ECCOMAS CFD 2006*, 2006.
[26] T. M. Smith, C. C. Ober, and A. A. Lorber. SIERRA/Premo–A New General Purpose Compressible Flow Simulation Code. In *AIAA 32nd Fluid Dynamics Conference, St. Louis*, 2002.