Aeroacoustics Conferences
30 May - 1 June, 2016, Lyon, France
22nd AIAA/CEAS Aeroacoustics Conference

A Hybrid 3D Discontinuous Galerkin Code for CAA Applications

Markus Lummer*

A convenient approach for CAA calculations for complex geometries are discontinuous Galerkin (DG) methods on unstructured meshes. Such methods are quite expensive compared with finite difference methods on cartesian grids and it is desirable to combine both approaches: relatively small unstructured meshes in the vicinity of bodies with cartesian grid blocks in the far field. A hybrid CAA code of this kind is under development at German Aerospace Center (DLR) and is presented here. The coupling procedure between cartesian blocks and unstructured parts of the mesh is described and details of the implementation are given. Arbitrary unsteady boundary conditions can be applied. Array calculations on the cartesian blocks are optimized using the `Blitz++` library. A combination of MPI and OpenMP parallelization is used. Fully unstructured meshes can be distributed on different MPI processes using the `Metis` software. The accuracy of the code is checked by calculating monopole scattering at a sphere and comparison with a known analytical solution. For large problems using hybrid meshes instead of fully unstructured ones the wall clock time can be reduced by about 64% and the memory requirements by about 80%.

Contents

List of Figures

*Research Engineer, Department of Technical Acoustics, Institute of Aerodynamics and Flow Technology, German Aerospace Center (DLR), Lilienthalplatz 7, D-38108 Braunschweig, markus.lummer@dlr.de

List of Tables

Nomenclature

Greek Symbols

Δ	Distance between to grid lines on a cartesian block
λ	Wavelength
ω	$= \frac{2\pi c_0}{\lambda}$ angular frequency
$\Phi_{\mathfrak{n}}(x_i)$	Lagrange functions on computational cell
$\varphi(x_k)$	Galerkin test function
ϱ_0	Mean flow density

Indices

$\alpha, \beta, \ldots$	Variable indices $(0, 1, 2, 3)$
$\mathfrak{m}, \mathfrak{n}, \ldots$	Collocation point indices on tetrahedron $\mathfrak{m} \in (0, 1, 2, \ldots, 19)$
$i, j, k, \ldots$	Space indices $(0, 1, 2)$

Latin Symbols

$G^p_{\mathfrak{l}\mathfrak{n}}$	$= \sum_{\mathfrak{m}} \tilde{M}^{-1}_{\mathfrak{l}\mathfrak{m}} \tilde{G}^p_{\mathfrak{m}\mathfrak{n}}$ Matrix in DG equations
$K^j_{\mathfrak{l}\mathfrak{n}}$	$= \sum_{\mathfrak{m}} \tilde{M}^{-1}_{\mathfrak{l}\mathfrak{m}} \tilde{K}^j_{\mathfrak{m}\mathfrak{n}}$ Matrix in DG equations
$\boldsymbol{\Lambda}^{p\mathfrak{n}}$	Diagonal matrix of eigenvalues of $D^{p\mathfrak{n}}_{\alpha\beta}$
$\boldsymbol{R}^{p\mathfrak{n}}$	Matrix of right eigenvectors of $D^{p\mathfrak{n}}_{\alpha\beta}$
$\{\mathfrak{n}_p\}$	Index set of collocation points on face p
$\tilde{G}^p_{\mathfrak{m}\mathfrak{n}}$	Surface integral $\int_{A^p} dA \Phi_{\mathfrak{m}} \Phi_{\mathfrak{n}}$
$\tilde{K}^j_{\mathfrak{m}\mathfrak{n}}$	'Stiffness' matrix $\int_V dV \Phi_{\mathfrak{m},j} \Phi_{\mathfrak{n}}$
$\tilde{M}_{\mathfrak{m}\mathfrak{n}}$	'Mass' matrix $\int_V dV \Phi_{\mathfrak{m}} \Phi_{\mathfrak{n}}$
$\mathcal{R}_{RAM}$	Reduction in percent RAM of hybrid vs. fully unstructured calculation
$\mathcal{R}_{WCT}$	Reduction in percent WCT of hybrid vs. fully unstructured calculation
∂V	Surface of cell
A^p	Surface p of cell
$A^{j\mathfrak{n}}_{\alpha\beta}$	Value of $A^j_{\alpha\beta}$ at cell collocation point $\mathfrak{n}$
$A^j_{\alpha\beta}$	Flux matrices
c_0	Mean flow sound speed
$D^{p\mathfrak{n}}_{\alpha\beta}$	Value of $A^j_{\alpha\beta} n_j$ at collocation point $\mathfrak{n}$ of face p. Also written $\boldsymbol{D}^{p\mathfrak{n}}$
dA	Surface element
dV	Volume element
F^j_α	$= A^j_{\alpha\beta} u_\beta$ Flux vectors
$F^{p\mathfrak{n}}_\alpha$	Flux of variable α in point $\mathfrak{n}$ on face p of cell
$G^p_{\mathfrak{l}\mathfrak{n}}$	Matrix depending on cell geometry

$H_{\alpha\beta}^{p\mathfrak{n}\pm}$	Upwind flux matrices in point $\mathfrak{n}$ of face p
k	$= \frac{2\pi}{\lambda} = \frac{\omega}{c_0}$. Wave number
$K_{[\mathfrak{n}}^{p}$	Matrix depending on cell geometry
L	Edge length of computational domain
N	Number of data points on the edge of the computational domain. N^3 is approximately the number of data points (degrees of freedom – DOF) of a calculation
n_j	Face normal
p	Acoustic pressure
R_S	Radius of source sphere
s_α	Acoustic source vector
u_α	Acoustic variable vector $(p \;\; v_i)^\mathsf{T}$
V	Volume of cell
V_i	Mean flow velocity
v_i	Acoustic velocity
$x_i^{\mathfrak{m}}$	Lagrangian collocation points on cell

Abbreviations

CPW	Resolution in cells per wavelength
DG	Discontinuous Galerkin
DOF	Degrees of Freedom, number of data points that represent a field
DRP	Dispersion Relation Preserving
FRPM	Fast Random Particle-Mesh
MPE	MPI Parallel Environment
MPI	Message Passing Interface
RAM	Random Access Memory
RAM_{Hybrid}	RAM for hybrid calculation
RAM_{Tetra}	RAM for fully unstructured calculation
STL	Standard Template Library
WCT	Wall Clock Time
WCT_{Hybrid}	WCT for hybrid calculation
WCT_{Tetra}	WCT for fully unstructured calculation

I. Introduction

Computational aeroacoustic (CAA) calculations for complex geometries are most easily performed on unstructured meshes. In this case, convenient methods to solve the governing equations with adequate accuracy are discontinuous Galerkin (DG) methods [1, 2]. Unfortunately, DG-methods require significantly more computational work compared, e.g., with dispersion relation preserving (DRP) finite difference schemes [3] on cartesian grids. Therefore, it is tempting to combine both approaches: a DG-method on relatively small unstructured meshes in the vicinity of bodies and DRP-schemes on large cartesian grid blocks in the far field. The classical 7-point DRP-schemes [3] are 4th-order accurate and a simple coupling with 20-point 4th-order accurate tetrahedral DG-Cells [4] is possible.

Therefore, a new hybrid 3D-DG-DRP code is under development for some time at German Aerospace Center (DLR), complementing the 3D-DRP `Piano` [5–9] and 2D-DG `DISCO` [1, 10, 11] codes. Hybrid DG-DRP CAA approaches are not new. E. g., in [12] the high order DG-Solver NoisSol is coupled with the `Piano` code. Nevertheless, the approach presented below seems to be especially simple.

This paper is structured as follows. In the next section the governing equations are given. The new code solves acoustic perturbation equations (APE) [13] with a quadrature-free DG-method of 4-th order [1, 2]. A short overview over the implemented boundary conditions follow. In the subsequent section the coupling of the unstructured mesh with the cartesian blocks is described. The key idea is to cover the outer 3 cell layers of each cartesian block with tetrahedra. Each group of $3^3 = 27$ block surface cells is covered with 6 tetrahedra. The data points of the tetrahedra correspond exactly to cartesian grid points and no interpolation procedure between the structured and unstructured parts of the mesh is necessary. Furthermore, from outside, the cartesian block can be treated as unstructured mesh part too. This allows simple automatic combination of structured and unstructured mesh parts in the code. For unstructured mesh generation the `Gmsh` code is used [14]. `Gmsh` has a scripting capability and can process CAD files in IGES and STEP formats. The cartesian blocks are generated by the DG code on the fly.

In the following section some details of the implementation will be given. The code is written in C++ and relies

heavily on the usage of templates. Parallelization is prodvided by a hybrid MPI/OpenMP approach. The unstructured mesh cells can be distributed among the MPI processes using the `Metis` software [15]. Array calculations on the cartesian grid blocks are performed using the expression template library `Blitz++`. The OpenMP parallelization approach to `Blitz++` is briefly discussed.

Unsteady boundary conditions can be applied by weak coupling using ghost cells. The performance of the code is demonstrated using a simple application, the scattering of an acoustic monopole at a sphere. The solution is calculated on a fully unstructured and a hybrid mesh and compared with an analytical one, obtained from the Green's function of the problem. In case of large problems, replacing fully unstructured by hybrid meshes can reduce the wall clock time by about 34% and the memory requirements by about 80%.

II. Governing Equations

The code solves linear acoustic perturbation equations (APE, cf. [1, 13]) which read in index notation (greek indices are variable indices (0,1,2,3), latin indices are space indices(0,1,2), a summation convention applies)

$$\partial_t u_\alpha + F^j_{\alpha,j} - s_\alpha = 0 \tag{1}$$

where s_α is a source vector and the flux vectors F^j_α have the form $F^j_\alpha = A^j_{\alpha\beta} u_\beta$. The $A^j_{\alpha\beta}$ are matrices and $u_\alpha = (p \;\; v_i)^\mathsf{T}$ is the acoustic variable vector which is formed from the acoustic pressure p and the acoustic velocity v_i. Writing V_i, ϱ_0, and c_0 for the mean flow velocity, density and sound speed, the matrices $A^j_{\alpha\beta}$ can be written in the form (dots indicate zero entries)

$$A^0_{\alpha\beta} = \begin{pmatrix} V_0 & \varrho_0 c_0^2 & . & . \\ \frac{1}{\varrho_0} & V_0 & V_1 & V_2 \\ . & . & . & . \\ . & . & . & . \end{pmatrix}, \; A^1_{\alpha\beta} = \begin{pmatrix} V_1 & . & \varrho_0 c_0^2 & . \\ . & . & . & . \\ \frac{1}{\varrho_0} & V_0 & V_1 & V_2 \\ . & . & . & . \end{pmatrix}, \; A^2_{\alpha\beta} = \begin{pmatrix} V_2 & . & . & \varrho_0 c_0^2 \\ . & . & . & . \\ . & . & . & . \\ \frac{1}{\varrho_0} & V_0 & V_1 & V_2 \end{pmatrix}. \tag{2}$$

Eq. (1) is solved by a quadrature-free discontinuous Galerkin procedure. A brief derivation of the discontinuous Galerkin equations is given in appendix A. Generally, Galerkin procedures require expensive integrals over the computational cells. However, if it is possible to map the computational cells linearly to some reference cell, so-called quadrature-free methods are possible [2]. In this case, the integrations can be performed in advance for the reference cell and only matrix-vector multiplications are required during the integration. Such kind of linear mapping is possible, e. g., for tetrahedrals.

Therefore, the quadrature-free DG method is implemented on a tetrahedral mesh. For a 4th-order discontinuous Galerkin method 20 collocation points are selected on each tetrahedron, cf. [4], and the acoustic quantities are approximated by the vectors $u^\mathfrak{n}_\alpha, \mathfrak{n} = 0, 1, \ldots, 19$ which are defined in each collocation (or data) point of the cell (The fraktur letters $\mathfrak{n}, \mathfrak{m}, \ldots$ denote the collocation points on the cells). The quadrature-free DG approximation of Eq. (1), derived in appendix A, then becomes the following system of ordinary differential equations on each cell

$$\dot{u}^\mathfrak{l}_\alpha + \sum_{p=0}^{3} \sum_{\mathfrak{n}\in\{\mathfrak{n}_p\}} G^p_{\mathfrak{l}\mathfrak{n}} F^{p\mathfrak{n}}_\alpha - \sum_{\mathfrak{n}=0}^{19} \sum_{j=0}^{2} K^j_{\mathfrak{l}\mathfrak{n}} \sum_{\beta=0}^{3} A^{j\mathfrak{n}}_{\alpha\beta} u^\mathfrak{n}_\beta - s^\mathfrak{l}_\alpha = 0, \tag{3}$$

$$F^{p\mathfrak{n}}_\alpha \equiv \sum_{\beta=0}^{3} D^{p\mathfrak{n}}_{\alpha\beta} u^\mathfrak{n}_\beta, \qquad D^{p\mathfrak{n}}_{\alpha\beta} \equiv \left(A^j_{\alpha\beta} n_j\right)^{p\mathfrak{n}}. \tag{4}$$

$G^p_{\mathfrak{l}\mathfrak{n}}$ and $K^j_{\mathfrak{l}\mathfrak{n}}$ are matrices depending on the geometry of the cell and $A^{j\mathfrak{n}}_{\alpha\beta}$ and $D^{p\mathfrak{n}}_{\alpha\beta}$ are matrices depending on the mean flow quantities. The summation over p indicates a summation over the four faces of the cell and $F^{p\mathfrak{n}}_\alpha$ is a flux of variable α in point $\mathfrak{n}$ on face p. One has 10 collocation points on each face of the cell and $\left\{\mathfrak{n}_p\right\}$ is the index set of collocation points on face p. In the form given in Eq. (3) the solutions on the cells are independent of each other. In order to obtain a physically meaningful solution for the acoustic field, the cells must be coupled, i. e. the flux vector $F^{p\mathfrak{n}}_\alpha$ must be approximated by an appropriate upwind flux, which takes into account also the acoustic quantities on the face of the adjacent cell. Presently a Steger-Warming flux ([16], [1]) is used

$$F^{p\mathfrak{n}}_\alpha = H^{p\mathfrak{n}+}_{\alpha\beta} u^{\mathfrak{n}+}_\beta + H^{p\mathfrak{n}-}_{\alpha\beta} u^{\mathfrak{n}-}_\beta. \tag{5}$$

Assuming a face normal n_j pointing out of the cell the $u^{\mathfrak{n}+}_\beta$ denote the data values inside of the cell, and the $u^{\mathfrak{n}-}_\beta$ the data values of the adjacent cell. The upwind matrices $H^{p\mathfrak{n}\pm}_{\alpha\beta}$ are usually calculated by a similarity transformation of the

flux matrix $D^{p\mathfrak{n}}_{\alpha\beta} \equiv \boldsymbol{D}^{p\mathfrak{n}}$ (cf. Eq. (4))

$$\boldsymbol{D}^{p\mathfrak{n}} = \boldsymbol{R}^{p\mathfrak{n}} \boldsymbol{\Lambda}^{p\mathfrak{n}} \boldsymbol{R}^{p\mathfrak{n}-1}, \qquad \boldsymbol{H}^{p\mathfrak{n}\pm} = \frac{1}{2}\left[\boldsymbol{R}^{p\mathfrak{n}} \left(\boldsymbol{\Lambda}^{p\mathfrak{n}} \pm |\boldsymbol{\Lambda}^{p\mathfrak{n}}|\right) \boldsymbol{R}^{p\mathfrak{n}-1}\right] \tag{6}$$

where $\boldsymbol{R}^{p\mathfrak{n}}$ is the matrix of the right eigenvectors of $\boldsymbol{D}^{p\mathfrak{n}}$, $\boldsymbol{\Lambda}^{p\mathfrak{n}}$ is the diagonal matrix of the eigenvalues, and $|\boldsymbol{\Lambda}^{p\mathfrak{n}}|$ is the diagonal matrix of the modulus of of the eigenvalues. Using a simple rotation of the coordinate system the matrices $\boldsymbol{H}^{p\mathfrak{n}\pm}$ can be calculated analytically. This is shown in appendix B.

III. Boundary Conditions

Boundary conditions must be provided at outer boundaries of the mesh, i. e., at faces with only one adjacent cell. In the code they are implemented by setting special values for the flux matrices $\boldsymbol{H}^{p\mathfrak{n}\pm}$. Here, only the implemented form is given. Some more details can be found in [1].

At solid walls it is set $\boldsymbol{H}^{p\mathfrak{n}-} = 0$ and

$$\boldsymbol{H}^{p\mathfrak{n}+} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ \frac{n_0}{\varrho_0} & n_0 V_0 & n_0 V_1 & n_0 V_2 \\ \frac{n_1}{\varrho_0} & n_1 V_0 & n_1 V_1 & n_1 V_2 \\ \frac{n_2}{\varrho_0} & n_2 V_0 & n_2 V_1 & n_2 V_2 \end{pmatrix}. \tag{7}$$

At far field boundaries it is set $\boldsymbol{H}^{p\mathfrak{n}-} = 0$. Then, waves normal to the face can leave the computational domain without reflection. Waves impinging the farfield boundary under an angle scatter some noise back into the computational domain. So far these spurious wave components have been found to be small.

Unsteady values of pressure and velocity are fed into the computational domain by attaching a ghost cell at the boundary face. Then, the face is treated like an interior face of the domain and the values of pressure and velocity in the ghost cell are taken from the boundary condition.

IV. Hybrid Approach

An inspection of Eq. (3) reveals a significant amount of floating point operations necessary for each cell. Even more important seem to be the high memory requirements. For each cell the 960 floating point numbers of the matrices $A^{j\mathfrak{n}}_{\alpha\beta}$ must be stored and for each face 320 elements of the matrices $\boldsymbol{H}^{p\mathfrak{n}\pm}$. Since one has usually twice as many faces as cells, for each cell of the mesh about 1600 floating point values must be stored. Thus, using single precision data, at least 6.4 GB of memory is necessary for 10^6 cells. Moreover, the description of acoustic sources by synthetic turbulence, e. g. produced by the Fast Random Particle-Mesh (FRPM) method [6], requires the calculation of long time series for the accurate prediction of the spectral properties of the acoustic field. Consequently, efficiency improvements of the algorithm are desirable. In case of large scale problems a significant reduction in computational resources can be expected using highly efficient finite difference methods on equidistant cartesian blocks for most of the computational domain. This requires an appropriate coupling procedure between the unstructured and structured parts of the mesh.

Different coupling mechanisms are possible. In [12] the discontinuous Galerkin code NoisSol is coupled with the `Piano` code using a flexible space-time interpolation procedure. This allows different cell sizes and time steps on the structured and unstructured parts of the mesh. Often, however, the mesh cell size is determined by enabling proper wave propagation and therefore doesn't change much across the mesh. In this case, the following simple coupling procedure that avoids interpolation can be used. It requires a box like outer surface of each unstructured part of the mesh, which has to be covered by a special regular triangulation.

Then, coupling to cartesian grid blocks is possible by covering some outer grid layers of each cartesian block by tetrahedra whose collocation points correspond to grid points of the block, cf. Fig.1. These surface tetrahedra (depicted in blue) can be treated like all other unstructured cells and allow to pass appropriate boundary data to the cartesian block. The boundary of the tetrahedra to the interior of the block is treated as an outer boundary of the unstructured mesh, where unsteady surface data can be applied as usual unsteady boundary conditions by attached ghost cells (depicted in orange). At beginning of each Runge-Kutta sub step, the appropriately updated variables from interior block points are copied to these ghost cells.

The surface tetrahedra are constructed as follows. The number of cartesian grid cells is assumed to be a multiple of 3 in each space direction. Thus, for example in x-direction, the equidistant grid points are $x_i = i\Delta$, $i = 0, 1, \ldots, N_x$. Each group of 3^3 cartesian surface grid cells is then covered by 6 tetrahedra. This is depicted in Figs. 2 and 3. The 27 cartesian cells of the group are splitted in two triangular prisms first and each prism is then subdivided into 3 tetrahedra. The 20 data points of each tetrahedron are identical with points on the cartesian grid and no interpolation is necessary.

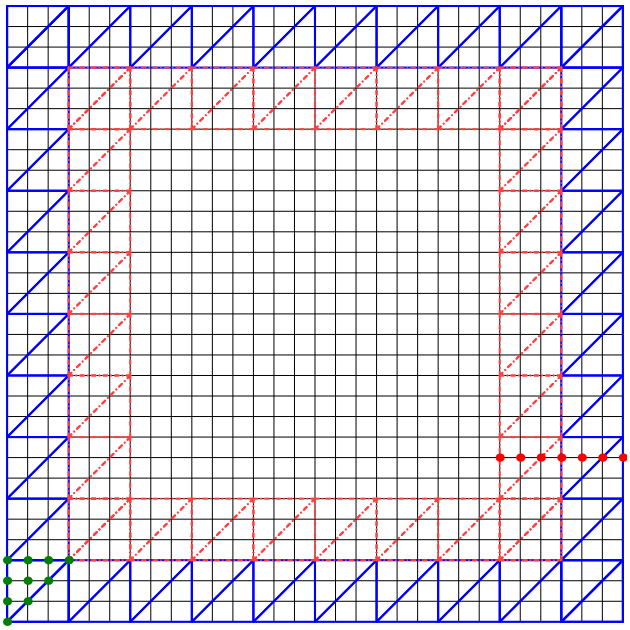
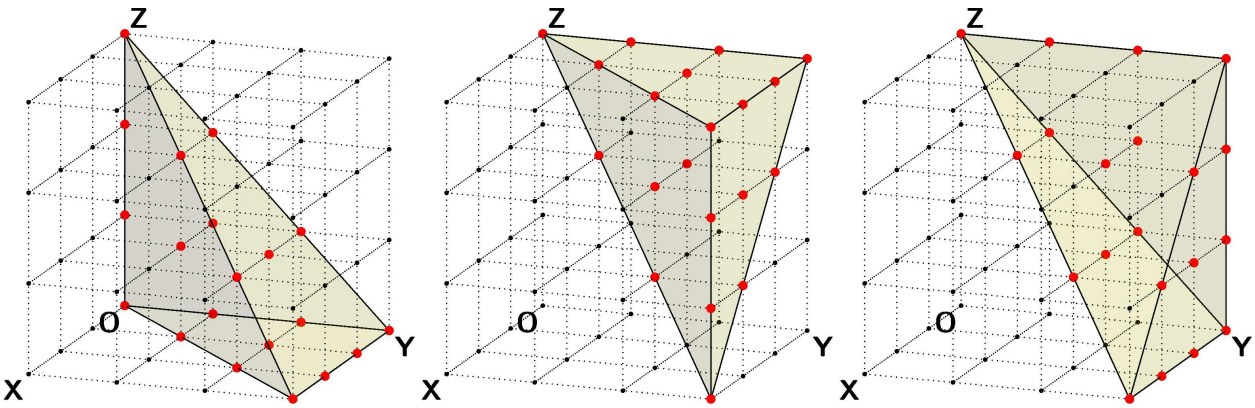


Figure 1. Cut through cartesian block showing the surface tetrahedral cell layers. Each outer 3^3 cartesian cell group is divided in 2 triangular prisms, depicted in blue, and each prism then in 3 tetrahedra. Ghost cells for data transfer from the block interior to the surface cells are depicted in dotted orange. Data points on a selected tetrahedral face are depicted in green. A (symmetrical) 7-point DRP stencil is indicated by the red dots.



a) Tetrahedron 0 **b) Tetrahedron 1** **c) Tetrahedron 2**

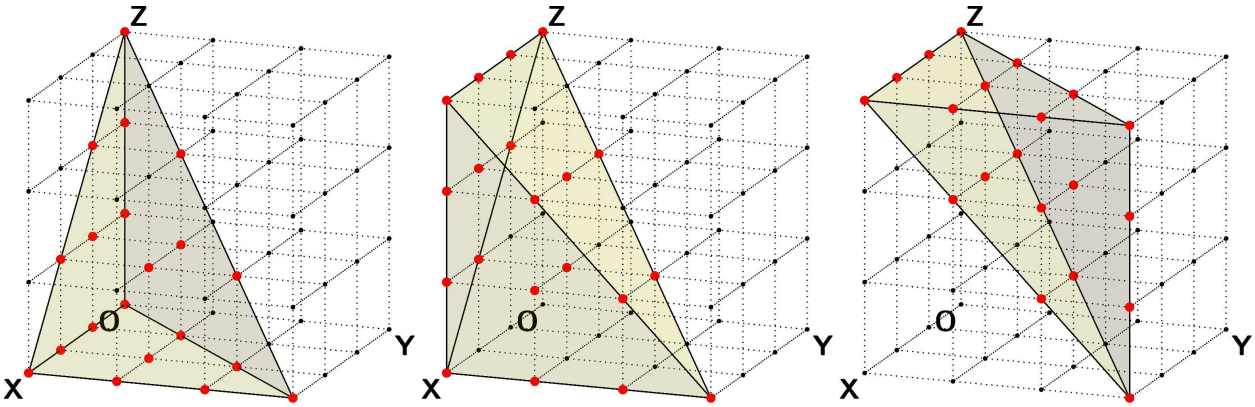
Figure 2. First triangular prism – Data points of tetrahedra are identical with grid points and depicted in red

The price which must be payed, is to use an appropriate regular triangulation on the surface of the unstructured parts of the mesh. This is possible without problems using the `Gmsh` meshing software [14]. Furthermore, since the governing equations must only be approximated inside of the block, symmetrical 7-point DRP stencils can be used for the calculation of the derivatives inside of the block.

V. Time Integration

The time integration is done by a classical 4th-oder Runge-Kutta procedure using four sub steps. Writing for the differential equations $\dot{\boldsymbol{u}} = \boldsymbol{f}(t, \boldsymbol{u})$, the variable update from time t^n to $t^{n+1} = t^n + \Delta t$ reads

$$\boldsymbol{u}^{n+1} = \boldsymbol{u}^n + \Delta t \sum_{j=1}^{4} b_j \boldsymbol{k}_j, \qquad \boldsymbol{k}_j = \boldsymbol{f}\left(t^n + c_j \Delta t, \boldsymbol{w}_j^n\right), \qquad \boldsymbol{w}_j^n = \boldsymbol{u}^n + \Delta t \sum_{l=1}^{j-1} a_{jl} \boldsymbol{k}_l \tag{8}$$



a) Tetrahedron 3 b) Tetrahedron 4 c) Tetrahedron 5

Figure 3. Second triangular prism – Data points of tetrahedra are identical with grid points and depicted in red

where the b_j, c_j, and a_{jl} are the standard constants. At the beginning of each sub step j the work variables $\boldsymbol{w}_j^n$ are updated according to Eq. (8) on the data points of the cells and the interior points of the cartesian blocks. Then the ghost cells of each block fetch the necessary interior values from the block grid points and the block surface cells copy their work variables into the boundary points of the cartesian block. Then, the differental equations can be evaluated inside the block and for the cells.

VI. Implementation

The hybrid DG-DRP code is written in C++. In order improve data locality, intensive use of templates has been made which avoids a lot of heap allocation. The matrices of Eq. (3) are stored as far as possible as parts of the cell and face data structures. Single and double precision data format can be selected at compile time. The code is parallelized using a combined MPI/OpenMP approach. A cartesian block together with its tetrahedral surface layers is restricted to one MPI process, but each MPI process can handle several blocks. The cells of the unstructured mesh around bodies can be distributed among several MPI processes using the `Metis` software[a].

The efficiency of the distribution of the cartesian blocks and the unstructured cells on the several MPI processes can easily be checked by the implemented Multi-Processing Environment (MPE)[b] package. Some more information about MPE is given below. An interpolation procedure for CFD mean flow fields on to the CAA mesh is integrated into the code [17]. For the description of acoustic sources by synthetic turbulence created from RANS mean flow data the DLR Fast Random Particle-Mesh (FRPM) method [6] is implemented. The FRPM module is taken from the `Piano` code and written in Fortran. Usually, it is run as a separate MPI process.

Since finally the code is bound to be used by industrial customers, providing a user friendly operation has been kept in mind. In order to use the hybrid features of the code, the user must only provide an unstructured mesh with an appropriate regular surface triangulation, and the cartesian grid blocks can be generated by the code on the fly. During the integration process, the calculated solution can be visualized using the Python interface of the VisIt software[c]. Of interest in industrial applications is also a large flexibility concerning in- and output data formats as well as, e. g., the implementation of different unsteady boundary conditions. This is achieved by providing the possibility to code appropriate separate plugins, which can be loaded dynamically at run time [18]. The interface of the plugins is well defined and they are completely independent of the core code. It is planned to provide this plugin feature also for changing the governing equations on the code.

A. Unstructured Mesh

Some brief information about the used data structures will be given. The fundamental geometric entities of the unstructured part of the code are cells and faces which have in common the set of cell and face corner points of the mesh. A corner point is represented by a C++ class which stores its coordinates as three double precision numbers. A series

[a] Metis software from http://glaros.dtc.umn.edu/gkhome/views/metis
[b] MPE documentation from http://www.mcs.anl.gov/perfvis
MPE software from http://www.mpich.org/static/mpe/downloads
[c] VisIt from https://wci.llnl.gov/simulation/computer-codes/visit

of methods like, e. g., scalar and vector products is defined for this class. All corner points of the cells are stored in a global C++ Standard Template Library (STL) vector container.

The cells and faces of the mesh are represented by a C++ class Cell and Face, respectively. At start of a calculation, each MPI process reads the corner points and tetrahedral cells of the mesh from a file and the faces of the mesh are then determined by analysis of the connectivity of the cells. In order to reduce the memory requirements during this phase of the calculation, the Cell and Face classes are derived from some base classes CellBase and FaceBase which store only the geometrical informations necessary for the analysis of the mesh. Again, the cells and faces are stored in two global C++ STL vector containers.

B. Cartesian Grid Blocks

Each cartesian grid block is represented by a C++ class called HybridBlock. The blocks together with their coupling cells are created at beginning of the calculation, i. e., before the unstructured parts of the mesh are read. Thus, the cells of each block are stored contiguous at beginning of the global cell container. The variables on each block are stored in large three dimensional arrays. Originally, a self designed array class was implemented, but finally, for increased performance, the array class from the `Blitz++` library [19–22] was used. `Blitz++` is a C++ library, which obtains high performance for multidimensional array calculations using expression templates. Customized evaluation kernels for multidimensional loops can be generated at compile time and many loop transformations which increase the performance can be performed [21]. `Blitz++` is a serial library and OpenMP parallelizations must be performed by exchanging some of the `Blitz++` Range objects [22] by explicit loops. `Blitz++` Range objects are used to access subarrays of `Blitz++` arrays. Listing 1 shows pseudo code of two forms of a simple 2-point stencil operation using Ranges. In line 2 two 3-dimensional `Blitz++` arrays are defined. Line 5 defines range objects for the several dimensions and line 8 demonstrates a simple 2-point stencil operation on array *a* storing the result in array *b*. In lines 12 and 13 the second index range *J* is replaced by a simple C++ loop which can be parallelized by OpenMP statements. Replacing the *J* range by a loop allows `Blitz++` to perform stencil optimizations for the *I* range.

Listing 1. Blitz++ stencil operations using Range objects

```
1  // 3-dimensional Blitz++ arrays
2  Array<float,3> a(n1,n2,n3), b(n1,n2,n3);
3
4  // Range objects
5  Range I(1,n1-2), J(0,n2-1), K(0,n3-1);
6
7  // Standard Blitz++ stencil operation
8  b(I,J,K) = a(I+1,J,K) - a(I-1,J,K);
9
10 // Looping over an index is also possible
11 #pragma omp parallel for
12 for (int J=0; J<n2; ++J)
13   b(I,J,K) = a(I+1,J,K) - a(I-1,J,K);
```

Calculating the spatial derivatives using 7-point DRP stencils is quite expensive and its speed-up using `Blitz++` and OpenMP parallelization has been examined. Listing 2 shows pseudo code for calculation of the x-, y-, z-derivatives using OpenMP parallel C++ loops. Listing 3 shows pseudo code for the same calculations using OpenMP parallel `Blitz++`. The OpenMP directives are the C++ preprocessor `#pragma` statements. The number of threads can be varied at run time using the C++ `nThread` variable. The OpenMP `schedule(dynamic,1)` directive means that at start of the loop each thread is assigned a chunk of length 1 of the loop. When a thread finishes its chunk, it is dynamically assigned another.

Listing 2. 7-Point DRP derivatives using simple loops

```
// nThread is the number of threads
// d[7] are the DRP coefficients

// ========== diffXloop ==========
#pragma omp parallel for num_threads(nThread) \
  schedule(dynamic,1)
for(i=3; i<n1-3; ++i)
for(j=0; j<n2;   ++j)
for(k=0; k<n3;   ++k)
   ax(i,j,k) = d[0]*a(i-3,j,k) + ... + d[6]*a(i+3,j,k);

// ========== diffYloop ==========
#pragma omp parallel for num_streads(nThread) \
  schedule(dynamic,1)
for(i=0; i<n1;   ++i)
for(j=3; j<n2-3; ++j)
for(k=0; k<n3;   ++k)
   ay(i,j,k) = d[0]*a(i,j-3,k) + ... + d[6]*a(i,j+3,k);

// ========== diffZloop ==========
#pragma omp parallel for num_streads(nThread) \
  schedule(dynamic,1)
for(i=0; i<n1;   ++i)
for(j=0; j<n2;   ++j)
for(k=3; k<n3-3; ++k)
   az(i,j,k) = d[0]*a(i,j,k-3) + ... + d[6]*a(i,j,k-3);
```

Listing 3. 7-Point DRP derivatives using Blitz++ Ranges

```
// nThread is the number of threads
// d[7] are the DRP coefficients

// ========== diffX ==========
Range I(3,n2-4), K(0,n3-1);
// J-Loop is parallelized
// I/K-Loops can be optimized by Blitz++
#pragma omp parallel for num_threads(nThread) \
  schedule(dynamic,1)
for (int J=0; J<n2; ++J))
   ax(I,J,K) = d[0]*a(I-3,J,K) + ... + d[6]*a(I+3,J,K);

// ========== diffY ==========
Range J(3,n2-4), K(0,n3-1);
#pragma omp parallel for num_threads(nThread) \
  schedule(dynamic,1)
for (int I=0; I<n1; ++I))
   ay(I,J,K) = d[0]*a(I,J-3,K) + ... + d[6]*a(I,J+3,K);

// ========== diffZ ==========
Range J(0,n2-1), K(3,n3-4);
#pragma omp parallel for num_threads(nThread) \
  schedule(dynamic,1)
for (int I=0; I<n1; ++I))
   az(I,J,K) = d[0]*a(I,J,K-3) + ... + d[6]*a(I,J,K+3);
```

Exemplary, Fig. 4 shows the floating point performance for arrays of size 400^3 in GFlops up to a number of 24 threads. The calculations were performed on the DLR CASE-cluster (for some hardware details see appendix C). The values for thread 0 are calculated using `Blitz++` ranges without OpenMP acceleration. The `Blitz++` calculations reach their maximum performance at about 10 threads where they are about twice as fast as simple C++ loops. Even using 24 threads the `Blitz++` loops are about 20% faster.

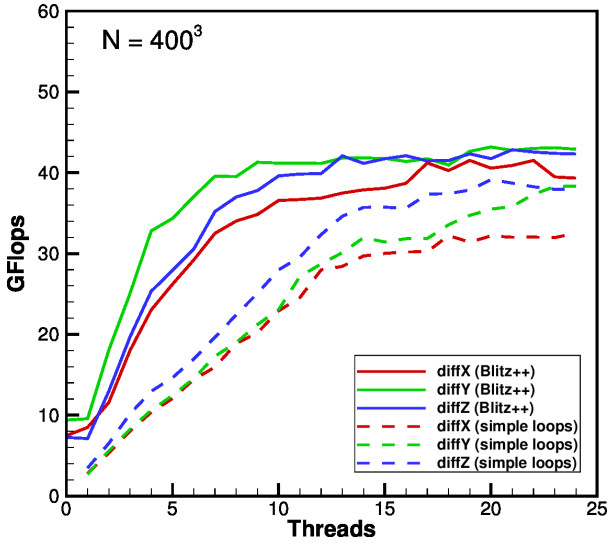


Figure 4. OpenMP parallelization of simple and `Blitz++` loops for a 3-d array of size 400^3. Thread 0 is the performance using `Blitz++` ranges alone.

VII. Monopole Scattering Test Case

In order to assess the accuracy and efficiency of the code, calculations of the scattering of the sound field of an acoustic monopole at a sound hard unit sphere were performed. The analytical solution of this problem is known [23] and can be used for assessing the accuracy of the calculation. The computational domain is depicted in Fig. 5. The scattering

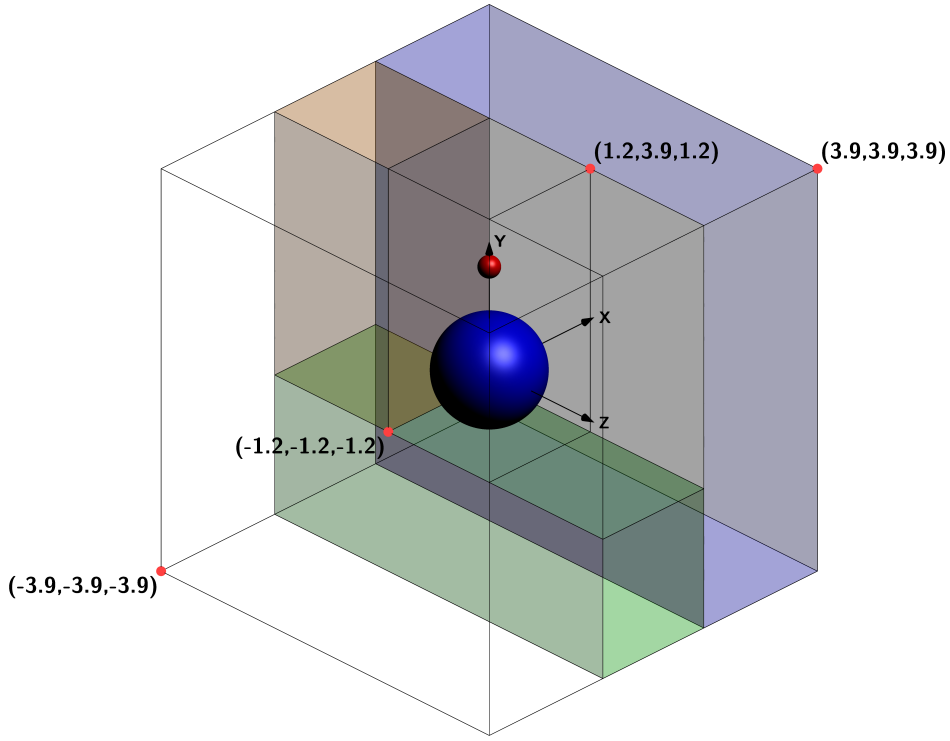


Figure 5. Block structure of computational domain for sphere scattering test. (Blue: scattering sphere, red: surface where monopole boundary conditions are applied). Three of the five cartesian blocks are colored.

sphere with radius 1, shown in blue, is centered in the origin of the coordinate system The acoustic monopole is located at $\boldsymbol{x}_0 = (0, 2, 0)$ above the sphere. Its pressure and velocity field is

$$p(\boldsymbol{x}, t) = \frac{-ikc_0}{4\pi r} e^{ikr - i\omega t}, \qquad \boldsymbol{v}(\boldsymbol{x}, t) = \frac{-1}{4\pi\varrho_0}\left(\frac{ik}{r} - \frac{1}{r^2}\right) e^{ikr - i\omega t} \nabla_x r, \qquad \nabla_x r = \frac{\boldsymbol{x} - \boldsymbol{x}_0}{r}, \qquad r = |\boldsymbol{x} - \boldsymbol{x}_0| \tag{9}$$

The mean flow density ϱ_0 and mean flow sound speed c_0 have been set to 1 in the calculations. The pressure has an unusual wave number k dependent amplitude factor, since in the code the point mass source (acoustic monopole) in a uniform subsonic flow from [24] is implemented. This sound field is fed into the computational domain, using unsteady boundary conditions on a small sphere with radius $R_S = 0.2$, depicted in red, around the monopole position. It must be emphasized that this source sphere is a hole in the computational domain and some discrepancies between calculation and analytical solution are expected. This will be discussed later. The whole computational domain is a cube with edge length $L = 7.8$, $(-3.9 < x, y, z < 3.9)$ which is divided into 6 boxes. The scattering sphere as well as the source sphere is located inside a box with $-1.2 < x, z < 1.2$ and $-1.2 < y < 3.9$ which encloses a fully unstructured tetrahedral mesh. The remaining 5 boxes can be treated either as cartesian blocks or also filled with tetrahedra. Three of them are colored in Fig.5. The cartesian blocks can be filled with tetrahedra by covering each group of 3^3 cartesian cells with 6 tetrahedra. This guarantees that the hybrid as well as the fully unstructured meshes have identical data points. The distance Δ of two grid lines on the cartesian blocks is constant for all blocks and serves as measure for the mesh resolution.

4 different grid resolutions have been considered, whose parameters are depicted in table 1. The first column contains the mesh identifier. The finest mesh is denoted $L100$ and the coarsest one $L300$. The second column depicts 3Δ, i. e. triple the distance of two grid lines on a cartesian block. The next column gives the number $N = L/\Delta + 1$ of data points on an outer edge of the computational domain and the following one its cube N^3 . N^3 is approximately the total number of data points in the computational domain, which will also be denoted as Degrees Of Freedom (DOF) of

Table 1. Mesh parameter

Mesh	Spacing 3Δ[a]	Edge Points N[b]	$N^3 \approx$ DOF[c]	Core Cells[d]	Cells[e]
L100	0.050	469	103.0×10^6	1468×10^3	22.1×10^6
L150	0.075	313	30.7×10^6	224×10^3	6.6×10^6
L200	0.100	235	13.0×10^6	102×10^3	2.8×10^6
L300	0.150	157	3.9×10^6	37×10^3	0.8×10^6

[a] Δ is distance of two grid lines
[b] Data points on egde of computational domain: $N = L/\Delta + 1$ where $L = 7.8$ is the edge length of the computational cubic domain
[c] Approximate number of DOFs assuming the whole computational domain is covered by a cartesian grid.
[d] On unstructured central box around geometry and source
[e] Fully unstructured calculation – cartesian blocks covered with tetrahedra

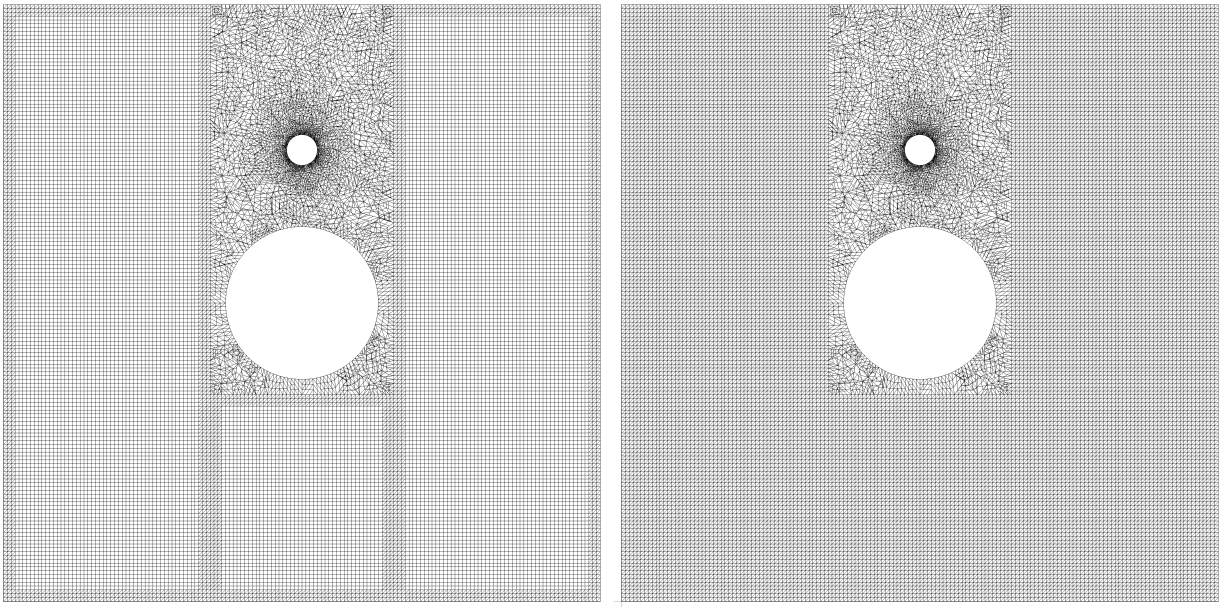
the calculation. The last two columns contain the numbers of cells of the unstructured core box around the geometry and the number of cells of the fully unstructured calculations. The unstructured mesh around the scattering sphere and source surface has been created by `Gmsh` using a characteristic length of 3Δ. This gives on the unstructured mesh a similar resolution like on the cartesian blocks.

Table 2 depicts the achievable wavelengths on each mesh for 6, 9, and 12 Cells Per Wavelength (CPW) resolution. A computational cell has the edge length Δ. For 6 CPW the minimum wavelength ranges from $\lambda = 0.10$ on the mesh $L100$ to $\lambda = 0.30$ on mesh $L300$. Fig. 6 depicts (x,y)-cuts through a hybrid and a fully unstructured $L300$ mesh.

Table 2. Mesh resolution CPW and wavelength λ

Mesh	6 CPW	9 CPW	12 CPW[a]
L100	0.10	0.15	0.20
L150	0.15	0.225	0.30
L200	0.20	0.30	0.40
L300	0.30	0.45	0.60

[a] The relation between CPW, λ, and N is $CPW = \lambda(N-1)/L$.

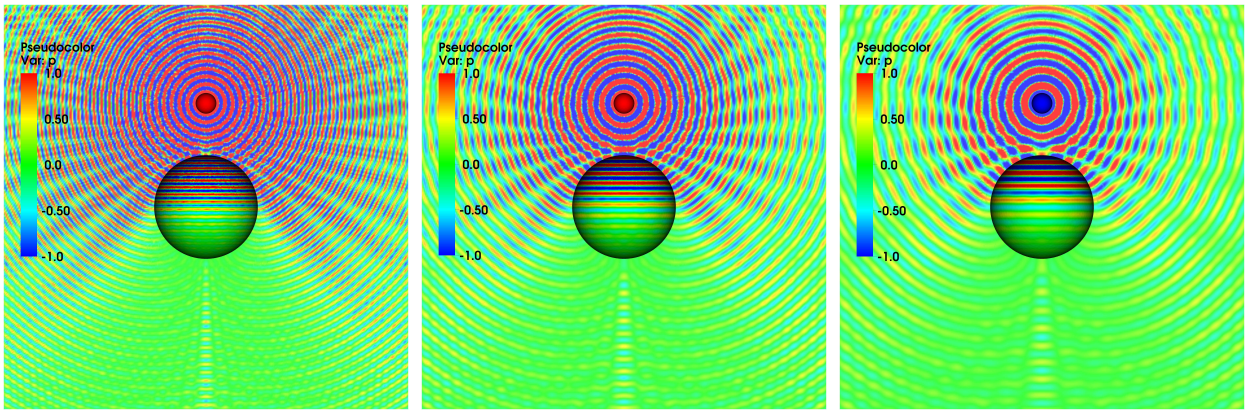


a) Hybrid tetrahedral/cartesian mesh **b) Tetrahedral mesh (cartesian blocks covered with tetrahedra)**

Figure 6. (x,y)-cut through $L300$ meshes

VIII. Numerical Accuracy

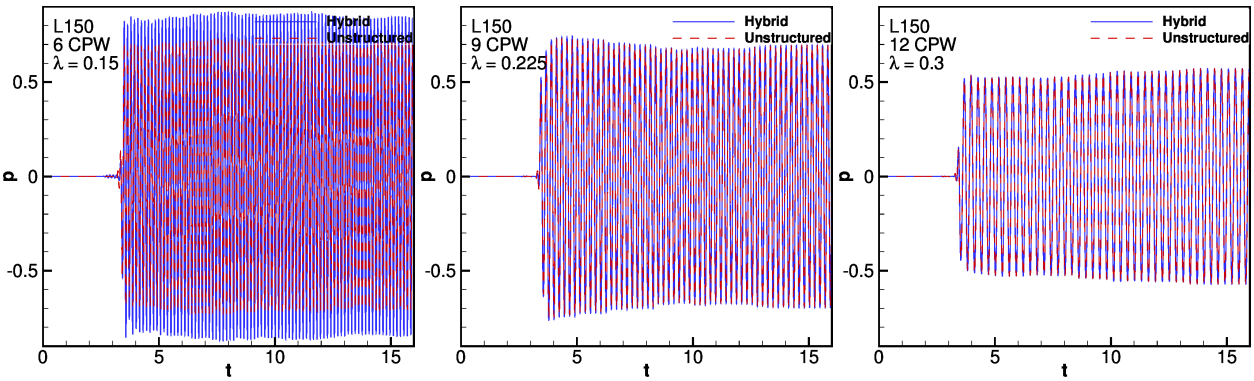
Before selected results of the calculations are discussed in more detail, Fig. 7 gives an impression of a typical scattered sound field. It depicts (x,y)-cuts through the pressure field after 20000 time steps for the $L150$ hybrid mesh and three different wavelengths, corresponding to 6, 9, and 12 CPW resolution. One identifies the interference pattern of the incident and reflected wave and the large shadow zone below the sphere. Directly below the sphere, along the negative y-axis, a narrow beam with higher pressures, the so-called Arago spot, is visible. In order to compare the different



a) 6 CPW, $\lambda = 0.15$ **b) 9 CPW,** $\lambda = 0.225$ **c) 12 CPW,** $\lambda = 0.30$

Figure 7. Pressure field after 20000 **time steps. (x,y)-cut through hybrid** $L150$ **mesh.**

solutions, the pressure signal on points of a circle of radius $R = 3$ around the scattering sphere in the (x,y)-plane is evaluated. Fig. 8 shows the pressure signal at $\boldsymbol{x} = (-3, 0, 0)$ for the hybrid and unstructured $L150$ mesh and 6, 9, and 12 CPW resolution. The calculations have been run for 20000 time steps. After an initial phase an almost periodically time signal evolves. For 9, and 12 CPW the signal on the hybrid and unstructured mesh is almost the same, while for 6 CPW a larger difference in amplitude is visible. This is not unexpected since 6 CPW is usually the minimum resolution necessary to obtain any meaningful results for 4th-order accurate CAA calculations. Fig. 8 shows the difference of the

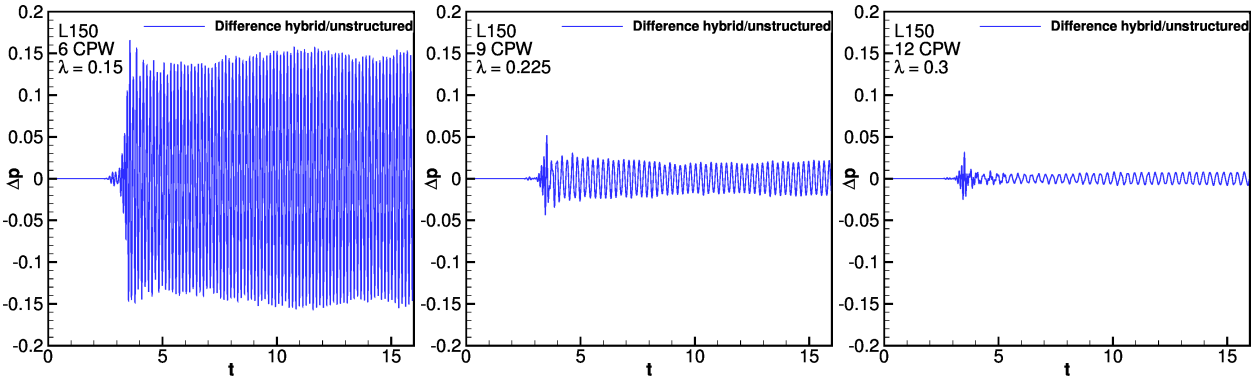


a) 6 CPW, $\lambda = 0.15$ **b) 9 CPW,** $\lambda = 0.225$ **c) 12 CPW,** $\lambda = 0.30$

Figure 8. Pressure signal at $\boldsymbol{x} = (-3, 0, 0)$ **on** $L150$ **hybrid and unstructured mesh**

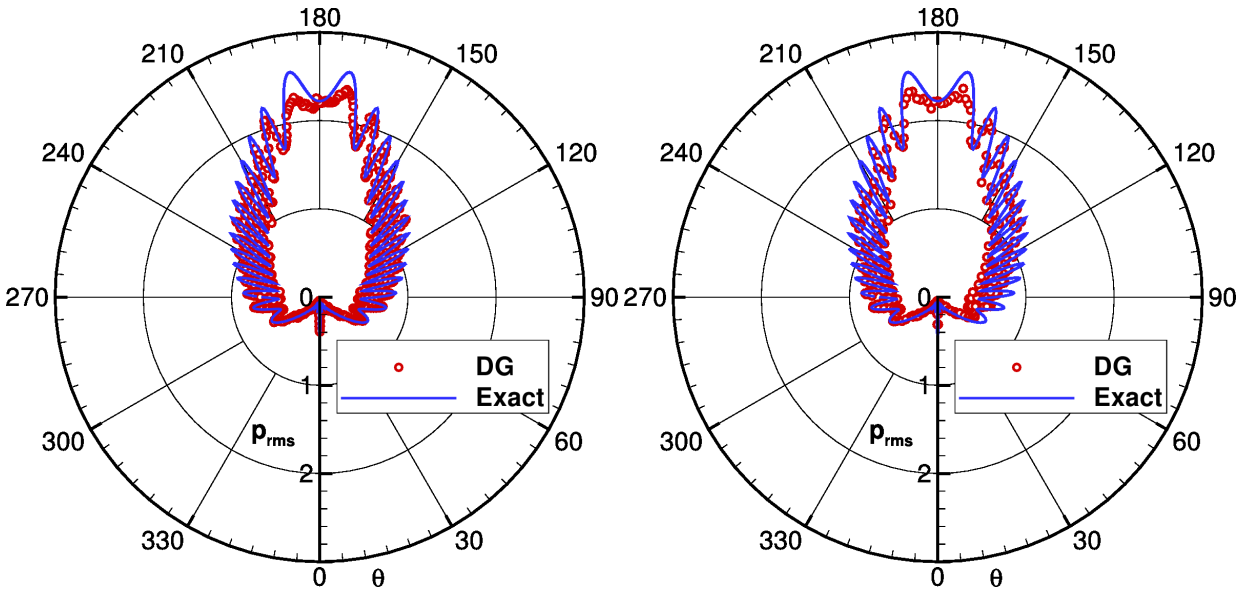
pressure signal for the hybrid and unstructured $L150$ meshes. The relative error is about 3% for 9 CPW and 2% for the 12 CPW calculation. It should be mentioned that the signal at the point considered is relatively small and thus the error is relatively large.

In order to assess the overall accuracy of the calculations, the so-called directivity, i. e. the root mean square (RMS) value of the pressure signal on the circle is compared with the analytical solution, taken from [23]. Fig. 10 depicts the directivity for $\lambda = 0.15$ and 6 CPW resolution on the hybrid and fully unstructured $L150$-mesh. In spite of the large error seen in Fig. 9a the overall correspondance between the calculated and analytical directivity is acceptable. Fig. 11 depicts the directivity in case of 9 CPW resolution on hybrid meshes for the wavelengths $\lambda = 0.15$, $\lambda = 0.225$, and



a) 6 CPW, $\lambda = 0.15$ **b) 9 CPW,** $\lambda = 0.225$ **c) 12 CPW,** $\lambda = 0.30$

Figure 9. Pressure difference at $x = (-3, 0, 0)$ **between** $L150$ **hybrid and unstructured mesh**



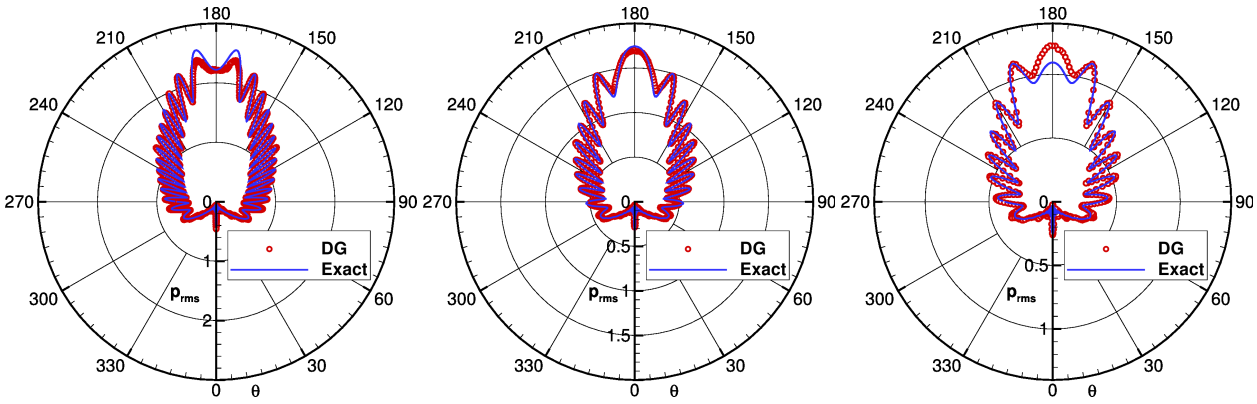
a) Hybrid mesh **b) Fully unstructured mesh**

Figure 10. Directivity for $\lambda = 0.15$ **and 6 CPW,** $L150$**-mesh**

$\lambda = 0.3$. The source position is at $\theta = 180°$ and by interference of the direct and reflected signal, the RMS value has its maximum there. Below the sphere there is a large shadow region. By and large, one sees good agreement between the analytical and calculated solution on most of the circle. Merely near the source direction larger differences, e. g. for $\lambda = 0.3$ can be seen. As mentioned above the source position is a mesh singularity (a hole) and these differences are not unexpected. In order to check the influence of the size of the source sphere on the solution the calculations for $\lambda = 0.3$ have been repeated for a source sphere of half the size, i. e. $R_S = 0.1$. Fig. 12 depicts the directivity for $\lambda = 0.3$ for different resolutions using hybrid meshes and the small source sphere. One sees a significantly better fit of calculated and analytical solution compared with Fig. 11c. Even in case of the coarsest mesh, i. e. in case of 6 CPW only small differences are visible.

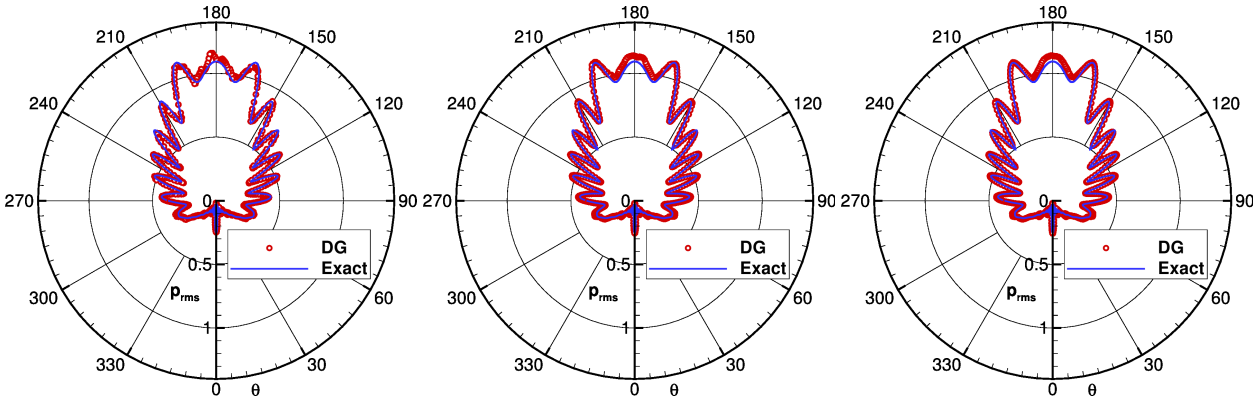
IX. Numerical Efficiency

The meshes were distributed on 3 MPI processes running on 3 nodes on the DLR CASE-cluster (cf. appendix C). On each node 24 OpenMP threads were used. Table 3 shows the used computational resources for the hybrid and unstructured computations. The first two columns show the approximate number of data points and the used time step for each mesh. Then, the wall clock time (WCT) for 1000 time steps of the hybrid and unstructured calculations



a) L100 $\lambda = 0.15$ **b) L150** $\lambda = 0.225$ **c) L200** $\lambda = 0.3$

Figure 11. Directivity as function of wavelength for 9 CPW resolution and hybrid meshes



a) L300, 6 CPW **b) L200, 9 CPW** **c) L150, 12 CPW**

Figure 12. Directivity for $\lambda = 0.3$**, hybrid meshes and small source sphere**

follow. Finally the total memory is given.

The WCT for the $L300$-mesh hybrid calculation is reduced by 32% compared to the fully unstructered one. This reduction increases to 64% in case of the large $L100$-mesh. The WCT reduction becomes better with increasing problem size since the numerical effort spent on the tetrahedral coupling cells becomes relatively smaller with increasing block size. The reduction in memory consumption using hybrid meshes is even larger. For the $L300$-mesh the memory is reduced by about 47% and for the $L100$-mesh by 80% using hybrid instead of fully unstructured meshes.

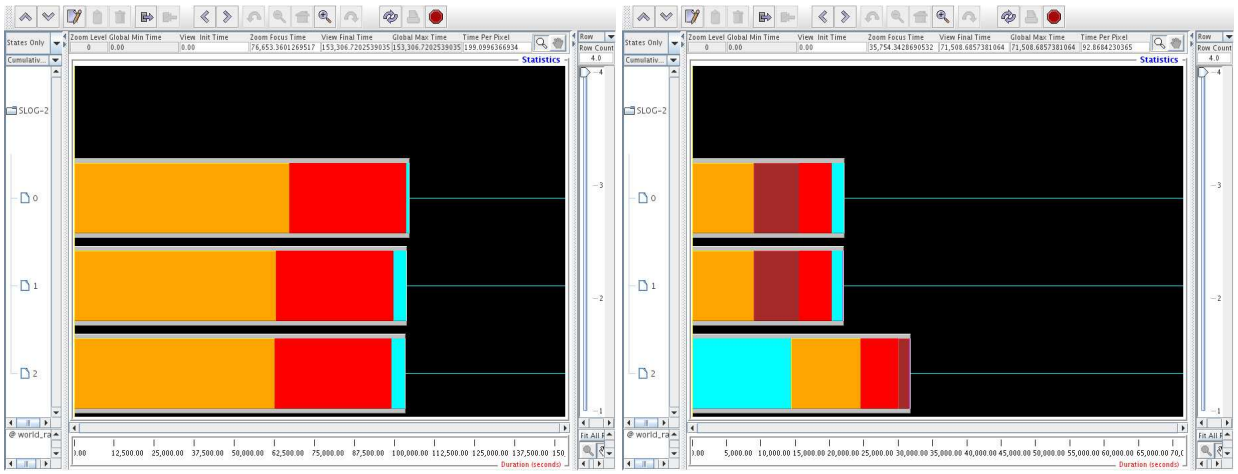
The efficiency of the parallelization can be assessed by examination of the MPE logging output. Fig. 13 shows the MPE logging output for 20000 time steps using 3 nodes on a $L150$ mesh for (a) the fully unstructured mesh and (b) the hybrid mesh. For each of the 3 MPI processes the accumulated WCT for several phases of the calculation is depicted in different colors. The WCT spend for cell calculations is shown in red, the one for face calculations in orange, and the WCT for cartesian blocks in brown. The WCT that is used for the MPI exchange of ghost cell data between the processes is shown in cyan. It should be mentioned that the x-axis for the unstructured calculation, Fig. 13a, covers about 125×10^3 [s] and the one for the hybrid calculation, Fig. 13b, about 62×10^3 [s]. One sees that the fully unstructured calculation is well balanced between the 3 processes and most of the computational work is done on the faces while only little time is spend for MPI data transfer between the cells. In the hybrid mesh calculation the computational work on cells and faces is significantly reduced in favor of work on cartesian blocks. However, the distribution of work between the processes is unbalanced and the third process spends a lot of time for MPI cell data transfer. In the present case, the 2 large cartesian blocks of Fig. 5 (i. e. the regions $x < -1.2$ and $1.2 < x$) are assigned to the processes 0 and 1 and the remaining cartesian blocks as well as the unstructured part of the mesh on process 2. Thus, a disadvantage of hybrid calculations is the difficulty to distribute the calculation well balanced on several MPI

Table 3. Performance using 3 computational nodes

Mesh	DOF[a]	Time Step	WCT_{Hybrid}[b]	WCT_{Tetra}[b]	$\mathcal{R}_{WCT}$[c]	RAM_{Hybrid}[d]	RAM_{Tetra}[d]	$\mathcal{R}_{RAM}$[e]
L100	103.0×10^6	0.4×10^{-3}	7500 [s]	20700 [s]	64%	52.6 [GB]	263.5 [GB]	80%
L150	30.6×10^6	0.8×10^{-3}	3030 [s]	6500 [s]	53%	20.5 [GB]	68.3 [GB]	70%
L200	13.0×10^6	0.8×10^{-3}	1770 [s]	2900 [s]	39%	11.0 [GB]	29.0 [GB]	62%
L300	3.9×10^6	0.8×10^{-3}	840 [s]	1230 [s]	32%	4.5 [GB]	8.5 [GB]	47%

[a] Approximate number of DOFs assuming the whole computational domain is covered by a cartesian grid.
[b] Wall clock time for 1000 steps on 3 nodes/24 threads
[c] The WCT reduction in percent is defined by $\mathcal{R}_{WCT} = \left(1 - \frac{WCT_{Hybrid}}{WCT_{Tetra}}\right) \times 100\%$
[d] Total process memory size on 3 nodes
[e] The RAM reduction in percent is defined by $\mathcal{R}_{RAM} = \left(1 - \frac{RAM_{Hybrid}}{RAM_{Tetra}}\right) \times 100\%$

processes.



a) **Fully unstructured mesh (x-axis covers about** 125×10^3**[s])** b) **Hybrid mesh (x-axis covers about** 62×10^3**[s])**

Figure 13. MPE Logging output for 3 MPI processes. $L150$**-mesh. Accumulated WCT: Brown: work on cartesian block(s), Orange: work on all faces, Red: work on all cells, Cyan: MPI cell exchange time.**

An advantage of the fully unstructured calculation is the possibility to distribute the mesh easily on an arbitrary number of MPI processes. In order to obtain informations about the speed-up of the calculation by using more and more computational nodes, calculations for the fully unstructured $L100$-mesh were performed using 3, 12, 18 , and 24 nodes. Fig. 14 shows the obtained speed-up of the calculations with increasing number of computational nodes. In the range considered the speed-up scales quite well with the number of cores.

X. Summary and Outlook

A new hybrid CAA code is under development at DLR which combines the advantages of unstructured meshes around bodies and cartesian grid blocks in the far field. The structured and unstructured parts of the mesh are weakly coupled by covering the surface of the cartesian blocks by appropriate tetrahedral layers. Arbitrary unsteady boundary conditions can be applied using ghost cells. Acoustic sources can be described using synthetic turbulence created by the FRPM method. The numerical work on the cartesian blocks is optimized using the `Blitz++` library. In order to allow easy extension of the code by the user, dynamically loadable modules, e. g. for different mesh formats, boundary conditions, and mean flow fields, are provided. It is planned to implement this feature also for the governing equations, which would allow an easy exchange of the acoustic perturbation equations, e.g., by the linearized Euler equations.

The accuracy of the code has been checked by calculation of monopole scattering at a sphere using meshes of different type and resolution. The differences between hybrid, fully unstructured and analytical solutions are sufficiently small even for a quite coarse resolution of 6 CPW. For large problems the wall clock time can be reduced by about

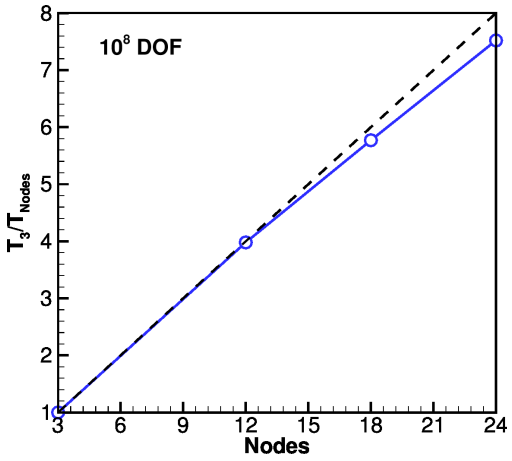


Figure 14. MPI scaling of fully unstructered calculations. Speed-up as function of the number of computational nodes. $L100$**-mesh,** $\approx 100 \times 10^6$ **data points**

64% using hybrid meshes instead of fully unstructured ones and the memory requirements by about 80%.

Fully unstructured calculations can easily be distributed on different MPI processes using the `Metis` library. Up to 24 computational nodes the speed-up of large fully unstructured calculations scales almost linearily with the number of nodes.

A disadvantage of the hybrid approach is the difficulty to achieve a well balanced distribution of the mesh on to a larger number of MPI processes. Presently, the topology and distribution of the cartesian blocks must be provided by the user. In future, the code should be able to analyse the computational domain and to propose an appropriate block structure of the mesh.

A. Galerkin Approximation of the Acoustic Perturbation Equations

Here, a brief derivation of the discontinuous Galerkin approximation of the acoustic perturbation equations (APE), Eq. (3), in 3-dimensions is given. The 2d case can be found, e. g. in [1]. The APE (cf. [1, 13]) read in index notation (greek indices are variable indices, latin indices are space indices)

$$\partial_t u_\alpha + F^j_{\alpha,j} - s_\alpha = 0, \qquad F^j_\alpha = A^j_{\alpha\beta} u_\beta. \tag{10}$$

The acoustic variable vector $u_\alpha = (p \;\; v_i)^\mathsf{T}$ is formed from the acoustic pressure p and velocity v_i, and the source vector s_α contains appropriate source terms for the pressure and momentum equation. Writing V_i, ϱ_0, and c_0 for the mean flow velocity, density and sound speed, the flux matrices $A^j_{\alpha\beta}$ can be written in the form (dots indicate zero entries)

$$A^0_{\alpha\beta} = \begin{pmatrix} V_0 & \varrho_0 c_0^2 & . & . \\ \frac{1}{\varrho_0} & V_0 & V_1 & V_2 \\ . & . & . & . \\ . & . & . & . \end{pmatrix}, \; A^1_{\alpha\beta} = \begin{pmatrix} V_1 & . & \varrho_0 c_0^2 & . \\ . & . & . & . \\ \frac{1}{\varrho_0} & V_0 & V_1 & V_2 \\ . & . & . & . \end{pmatrix}, \; A^2_{\alpha\beta} = \begin{pmatrix} V_2 & . & . & \varrho_0 c_0^2 \\ . & . & . & . \\ . & . & . & . \\ \frac{1}{\varrho_0} & V_0 & V_1 & V_2 \end{pmatrix}. \tag{11}$$

Multiplication of Eq. (10) with a test function $\varphi(x_k)$ yields

$$\partial_t \varphi u_\alpha + (\varphi F^j_\alpha)_{,j} - \varphi_{,j} F^j_\alpha - \varphi s_\alpha = 0. \tag{12}$$

Integration over a volume V gives then a weak form of the APE

$$\partial_t \int_V dV \; \varphi u_\alpha + \int_V dV \; (\varphi F^j_\alpha)_{,j} - \int_V dV \; \varphi_{,j} F^j_\alpha - \int_V dV \; \varphi s_\alpha = 0. \tag{13}$$

Using the Gaussian theorem the second volume integral in Eq. (13) can be transformed in an integral over the surface of V

$$\partial_t \int_V dV \; \varphi u_\alpha + \int_{\partial V} dA \; \varphi F^j_\alpha n_j - \int_V dV \; \varphi_{,j} F^j_\alpha - \int_V dV \; \varphi s_\alpha = 0 \tag{14}$$

where n_j is the outer unit normal of the volume and dA a surface element. One has

$$F_\alpha^j n_j = A_{\alpha\beta}^j n_j u_\beta \tag{15}$$

and it is convenient to introduce the matrix $D_{\alpha\beta}$ by

$$D_{\alpha\beta} \equiv A_{\alpha\beta}^j n_j = \begin{pmatrix} V_j n_j & \varrho_0 c_0^2 n_0 & \varrho_0 c_0^2 n_1 & \varrho_0 c_0^2 n_2 \\ \frac{n_0}{\varrho_0} & V_0 n_0 & V_1 n_0 & V_2 n_0 \\ \frac{n_1}{\varrho_0} & V_0 n_1 & V_1 n_1 & V_2 n_1 \\ \frac{n_2}{\varrho_0} & V_0 n_2 & V_1 n_2 & V_2 n_2 \end{pmatrix}. \tag{16}$$

The weak form, Eq. (14), now becomes

$$\partial_t \int_V dV\ \varphi u_\alpha + \int_{\partial V} dA\ \varphi D_{\alpha\beta} u_\beta - \int_V dV\ \varphi_{,j} A_{\alpha\beta}^j u_\beta - \int_V dV\ \varphi s_\alpha = 0. \tag{17}$$

Now one chooses V to be the volume of a *computational cell* and select a set of Lagrangian collocation points $x_i^{\mathfrak{m}}$ as well as Lagrange functions $\Phi_{\mathfrak{n}}(x_i)$ with (Fraktur letters denote collocation point indices on computational cells)

$$\Phi_{\mathfrak{n}}(x_i^{\mathfrak{m}}) = \delta_{\mathfrak{m}\mathfrak{n}}. \tag{18}$$

The Lagrange functions for a 'cubic' tetrahedron, i. e. a tetrahedron with 20 collocation points, can be found in [4]. The Galerkin ansatz now reads

$$u_\alpha(x_k, t) = \sum_{\mathfrak{n}} u_\alpha^{\mathfrak{n}}(t)\Phi_{\mathfrak{n}}(x_k), \qquad s_\alpha(x_k, t) = \sum_{\mathfrak{n}} s_\alpha^{\mathfrak{n}}(t)\Phi_{\mathfrak{n}}(x_k), \tag{19}$$

$$D_{\alpha\beta} u_\beta = \sum_{\mathfrak{n}} D_{\alpha\beta}^{\mathfrak{n}} u_\beta^{\mathfrak{n}}(t)\Phi_{\mathfrak{n}}(x_k), \qquad A_{\alpha\beta}^j u_\beta = \sum_{\mathfrak{n}} A_{\alpha\beta}^{j\mathfrak{n}} u_\beta^{\mathfrak{n}}(t)\Phi_{\mathfrak{n}}(x_k). \tag{20}$$

As set of test functions one can choose $\varphi = \Phi_{\mathfrak{m}}$. Before the surface integral is analysed the so-called 'mass' matrix $\tilde{M}_{\mathfrak{m}\mathfrak{n}}$ and 'stiffness' matrix $\tilde{K}_{\mathfrak{m}\mathfrak{n}}^j$ are defined by the volume integrals

$$\tilde{M}_{\mathfrak{m}\mathfrak{n}} \equiv \int_V dV\ \Phi_{\mathfrak{m}}\Phi_{\mathfrak{n}}, \qquad \tilde{K}_{\mathfrak{m}\mathfrak{n}}^j \equiv \int_V dV\ \Phi_{\mathfrak{m},j}\Phi_{\mathfrak{n}}, \tag{21}$$

and one obtains

$$\sum_{\mathfrak{n}} \tilde{M}_{\mathfrak{m}\mathfrak{n}} \dot{u}_\alpha^{\mathfrak{n}} + \int_{\partial V} dA\ \varphi D_{\alpha\beta} u_\beta - \sum_{\mathfrak{n}} \tilde{K}_{\mathfrak{m}\mathfrak{n}}^j A_{\alpha\beta}^{j\mathfrak{n}} u_\beta^{\mathfrak{n}} - \sum_{\mathfrak{n}} \tilde{M}_{\mathfrak{m}\mathfrak{n}} s_\alpha^{\mathfrak{n}} = 0. \tag{22}$$

Now the integral over the surface of the cell is analysed. It is assumed that the cell is bounded by plane faces A^p and one can split the surface integral in a sum over the faces

$$\int_{\partial V} dA\ \varphi D_{\alpha\beta} u_\beta = \sum_p \sum_{\mathfrak{n}\in\{\mathfrak{n}_p\}} D_{\alpha\beta}^{p\mathfrak{n}} u_\beta^{\mathfrak{n}} \int_{A^p} dA\ \Phi_{\mathfrak{m}}\Phi_{\mathfrak{n}}. \tag{23}$$

The face index p is added to $D_{\alpha\beta}^{p\mathfrak{n}}$ because $D_{\alpha\beta}$ depends explicitly on the normal n_j of the face. The surface integral is abbreviated by

$$\tilde{G}_{\mathfrak{m}\mathfrak{n}}^p \equiv \int_{A^p} dA\ \Phi_{\mathfrak{m}}\Phi_{\mathfrak{n}}. \tag{24}$$

The weak formulation Eq. (14) now becomes

$$\sum_{\mathfrak{n}} \tilde{M}_{\mathfrak{m}\mathfrak{n}} \dot{u}_\alpha^{\mathfrak{n}} + \sum_p \sum_{\mathfrak{n}\in\{\mathfrak{n}_p\}} \tilde{G}_{\mathfrak{m}\mathfrak{n}}^p D_{\alpha\beta}^{p\mathfrak{n}} u_\beta^{\mathfrak{n}} - \sum_{\mathfrak{n}} \tilde{K}_{\mathfrak{m}\mathfrak{n}}^j A_{\alpha\beta}^{j\mathfrak{n}} u_\beta^{\mathfrak{n}} - \sum_{\mathfrak{n}} \tilde{M}_{\mathfrak{m}\mathfrak{n}} s_\alpha^{\mathfrak{n}} = 0. \tag{25}$$

Multiplication with the inverse of the mass matrix $\tilde{M}_{\mathfrak{l}\mathfrak{m}}^{-1}$ yields a formulation suitable to be fed into a Runge-Kutta procedure

$$\dot{u}_\alpha^{\mathfrak{l}} + \sum_p \sum_{\mathfrak{n}\in\{\mathfrak{n}_p\}} \sum_{\mathfrak{m}} \tilde{M}_{\mathfrak{l}\mathfrak{m}}^{-1} \tilde{G}_{\mathfrak{m}\mathfrak{n}}^p D_{\alpha\beta}^{p\mathfrak{n}} u_\beta^{\mathfrak{n}} - \sum_{\mathfrak{n}} \sum_{\mathfrak{m}} \tilde{M}_{\mathfrak{l}\mathfrak{m}}^{-1} \tilde{K}_{\mathfrak{m}\mathfrak{n}}^j A_{\alpha\beta}^{j\mathfrak{n}} u_\beta^{\mathfrak{n}} - s_\alpha^{\mathfrak{l}} = 0. \tag{26}$$

As last step before specifiying the fluxes one can combine the matrices

$$G^p_{\mathfrak{l}\mathfrak{n}} \equiv \sum_{\mathfrak{m}} \tilde{M}^{-1}_{\mathfrak{l}\mathfrak{m}} \tilde{G}^p_{\mathfrak{m}\mathfrak{n}}, \qquad K^j_{\mathfrak{l}\mathfrak{n}} \equiv \sum_{\mathfrak{m}} \tilde{M}^{-1}_{\mathfrak{l}\mathfrak{m}} \tilde{K}^j_{\mathfrak{m}\mathfrak{n}}, \tag{27}$$

and obtains

$$\dot{u}^{\mathfrak{l}}_\alpha + \sum_p \sum_{\mathfrak{n}\in\{\mathfrak{n}_p\}} G^p_{\mathfrak{l}\mathfrak{n}} D^{p\mathfrak{n}}_{\alpha\beta} u^{\mathfrak{n}}_\beta - \sum_{\mathfrak{n}} K^j_{\mathfrak{l}\mathfrak{n}} A^{j\mathfrak{n}}_{\alpha\beta} u^{\mathfrak{n}}_\beta - s^{\mathfrak{l}}_\alpha = 0. \tag{28}$$

Assuming a 'cubic' tetrahedron with 20 collocation points and writing down explicitly the sums over β and j gives the form used in Eq. (3)

$$\dot{u}^{\mathfrak{l}}_\alpha + \sum_{p=0}^{3} \sum_{\mathfrak{n}\in\{\mathfrak{n}_p\}} G^p_{\mathfrak{l}\mathfrak{n}} \sum_{\beta=0}^{3} D^{p\mathfrak{n}}_{\alpha\beta} u^{\mathfrak{n}}_\beta - \sum_{\mathfrak{n}=0}^{19} \sum_{j=0}^{2} K^j_{\mathfrak{l}\mathfrak{n}} \sum_{\beta=0}^{3} A^{j\mathfrak{n}}_{\alpha\beta} u^{\mathfrak{n}}_\beta - s^{\mathfrak{l}}_\alpha = 0. \tag{29}$$

Now, one considers a point $\mathfrak{n}$ on a face p and replaces the flux vector $F^{p\mathfrak{n}}_\alpha \equiv D^{p\mathfrak{n}}_{\alpha\beta} u^{\mathfrak{n}}_\beta$ there by an appropriate upwind flux. This is the essential step for the DG coupling of neighbor cells. To start with, one can use so-called Steger-Warming fluxes (cf. [16], [1])

$$F^{p\mathfrak{n}}_\alpha = \left[H^+_{\alpha\beta} u^+_\beta + H^-_{\alpha\beta} u^-_\beta \right]^{p\mathfrak{n}} \tag{30}$$

where the

$$H^\pm_{\alpha\beta} \equiv \frac{1}{2} \left[D_{\alpha\beta} \pm |D_{\alpha\beta}| \right] \tag{31}$$

are a kind of upwind flux matrices. u^+_β denotes the variable values on the back side of A^p, i. e., the values of the cell out of which the face normal points, and u^-_β the values on the front side of A^p, i. e., the values on the face of the adjacent cell. The explicit form of $H^\pm_{\alpha\beta}$ is derived in the next section.

B. Determination of the Upwind Fluxes

In order to derive the explicit form of the upwind flux matrix $H^\pm_{\alpha\beta}$, a special coordinate system is chosen were the face normal points into x-direction. The flux matrix $D_{\alpha\beta}$ is written in symbolic notation[d]

$$\boldsymbol{D} = \begin{pmatrix} \boldsymbol{v}_0^\mathsf{T} \boldsymbol{n} & \varrho_0 c_0^2 \boldsymbol{n}^\mathsf{T} \\ \frac{\boldsymbol{n}}{\varrho_0} & \boldsymbol{n}\boldsymbol{v}_0^\mathsf{T} \end{pmatrix} \tag{32}$$

where $\boldsymbol{n}$ denotes the face normal. In order to rotate the face normal $\boldsymbol{n}$ into the x-axis $\boldsymbol{e}_x = (1,0,0)^\mathsf{T}$, a rotation matrix $\boldsymbol{A}$ with $\boldsymbol{n} = \boldsymbol{A}\boldsymbol{e}_x$ is constructed. In spherical coordinates the face normal $\boldsymbol{n}$ and a vector $\boldsymbol{a}_y$, orthonormal to $\boldsymbol{n}$, is given by

$$\boldsymbol{n} = \begin{pmatrix} \cos\varphi \sin\vartheta \\ \sin\varphi \sin\vartheta \\ \cos\vartheta \end{pmatrix}, \qquad \boldsymbol{a}_y = \begin{pmatrix} -\sin\varphi \\ \cos\varphi \\ 0 \end{pmatrix}, \text{ if } \sin\vartheta \neq 0, \text{ or } \boldsymbol{a}_y = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} \text{ otherwise.} \tag{33}$$

The third vector $\boldsymbol{a}_x$ of the tripod is then obtained by the vector product $\boldsymbol{a}_x = \boldsymbol{a}_y \times \boldsymbol{n}$. Then, the columns of matrix $\boldsymbol{A}$ are the column vectors $(\boldsymbol{n}, \boldsymbol{a}_x, \boldsymbol{a}_y)$. The inverse of $\boldsymbol{A}$ is its transposed $\boldsymbol{A}^{-1} = \boldsymbol{A}^\mathsf{T}$. Thus, one obtains for the rotated face normal $\tilde{\boldsymbol{n}}$

$$\tilde{\boldsymbol{n}} = \boldsymbol{A}^T \boldsymbol{n} = \begin{pmatrix} \boldsymbol{n}^\mathsf{T} \\ \boldsymbol{a}_x^\mathsf{T} \\ \boldsymbol{a}_y^\mathsf{T} \end{pmatrix} \boldsymbol{n} = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} = \boldsymbol{e}_x. \tag{34}$$

Introduction of rotated vectors $\tilde{\boldsymbol{n}}, \tilde{\boldsymbol{v}}_0$ by

$$\boldsymbol{n} = \boldsymbol{A}\tilde{\boldsymbol{n}}, \qquad \boldsymbol{v}_0 = \boldsymbol{A}\tilde{\boldsymbol{v}}_0, \tag{35}$$

[d] In symbolic notation the scalar product is the product of a row and a column vector, i. e., e. g. $n^2 = \boldsymbol{n}^\mathsf{T}\boldsymbol{n}$. The product of a column and a row vector is a matrix

$$\boldsymbol{n}\boldsymbol{v}^\mathsf{T} = \begin{pmatrix} n_0 v_0 & n_0 v_1 & n_0 v_2 \\ n_1 v_0 & n_1 v_1 & n_1 v_2 \\ n_2 v_0 & n_2 v_1 & n_2 v_2 \end{pmatrix}.$$

and substitution into the flux matrix Eq. (32) yields

$$\boldsymbol{D} = \begin{pmatrix} \tilde{\boldsymbol{v}}_0^\mathsf{T}\boldsymbol{A}^\mathsf{T}\boldsymbol{A}\tilde{\boldsymbol{n}} & \varrho_0 c_0^2 \tilde{\boldsymbol{n}}^\mathsf{T}\boldsymbol{A}^\mathsf{T} \\ \boldsymbol{A}\frac{\tilde{\boldsymbol{n}}}{\varrho_0} & \boldsymbol{A}\tilde{\boldsymbol{n}}\tilde{\boldsymbol{v}}_0^\mathsf{T}\boldsymbol{A}^\mathsf{T} \end{pmatrix} = \begin{pmatrix} \tilde{\boldsymbol{v}}_0^\mathsf{T}\tilde{\boldsymbol{n}} & \varrho_0 c_0^2 \tilde{\boldsymbol{n}}^\mathsf{T}\boldsymbol{A}^\mathsf{T} \\ \boldsymbol{A}\frac{\tilde{\boldsymbol{n}}}{\varrho_0} & \boldsymbol{A}\tilde{\boldsymbol{n}}\tilde{\boldsymbol{v}}_0^\mathsf{T}\boldsymbol{A}^\mathsf{T} \end{pmatrix} = \begin{pmatrix} 1 & . \\ . & \boldsymbol{A} \end{pmatrix} \underbrace{\begin{pmatrix} \tilde{\boldsymbol{v}}_0^\mathsf{T}\tilde{\boldsymbol{n}} & \varrho_0 c_0^2 \tilde{\boldsymbol{n}}^\mathsf{T} \\ \frac{\tilde{\boldsymbol{n}}}{\varrho_0} & \tilde{\boldsymbol{n}}\tilde{\boldsymbol{v}}_0^\mathsf{T} \end{pmatrix}}_{\equiv \tilde{\boldsymbol{D}}} \begin{pmatrix} 1 & . \\ . & \boldsymbol{A}^\mathsf{T} \end{pmatrix}. \tag{36}$$

Considering that $\tilde{\boldsymbol{n}} = \boldsymbol{e}_x$ and writing for the mean flow velocity $\tilde{\boldsymbol{v}}_0 = (U, V, W)^\mathsf{T}$ the rotated flux matrix $\tilde{\boldsymbol{D}}$ becomes

$$\tilde{\boldsymbol{D}} = \begin{pmatrix} \tilde{\boldsymbol{v}}_0^\mathsf{T}\tilde{\boldsymbol{n}} & \varrho_0 c_0^2 \tilde{\boldsymbol{n}}^\mathsf{T} \\ \frac{\tilde{\boldsymbol{n}}}{\varrho_0} & \tilde{\boldsymbol{n}}\tilde{\boldsymbol{v}}_0^\mathsf{T} \end{pmatrix} = \begin{pmatrix} U & c_0^2\varrho_0 & . & . \\ \frac{1}{\varrho_0} & U & V & W \\ . & . & . & . \\ . & . & . & . \end{pmatrix}. \tag{37}$$

The Steger-Warming upwind fluxes (cf. [16], [1]) are now introduced by the similarity transform $\tilde{\boldsymbol{D}} = \boldsymbol{R}\boldsymbol{\Lambda}\boldsymbol{R}^{-1}$ using the matrices of the eigenvalues $\boldsymbol{\Lambda}$ and right eigenvectors $\boldsymbol{R}$ of $\boldsymbol{D}$

$$\boldsymbol{R} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 & 1 & . \\ \frac{-1}{c_0\varrho_0} & \frac{1}{c_0\varrho_0} & \frac{-U}{c_0^2\varrho_0} & . \\ . & . & . & 1 \\ . & . & \frac{U^2-c_0^2}{c_0^2\varrho_0 W} & \frac{-V}{W} \end{pmatrix}, \quad \boldsymbol{\Lambda} = \begin{pmatrix} U-c_0 & . & . & . \\ . & U+c_0 & . & . \\ . & . & . & . \\ . & . & . & . \end{pmatrix}, \quad \boldsymbol{R}^{-1} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & -c_0\varrho_0 & \frac{-c_0\varrho_0 V}{U-c_0} & \frac{-c_0\varrho_0 W}{U-c_0} \\ 1 & c_0\varrho_0 & \frac{c_0\varrho_0 V}{U+c_0} & \frac{c_0\varrho_0 W}{U+c_0} \\ . & . & \frac{2c_0^2\varrho_0 V}{U^2-c^2} & \frac{2c_0^2\varrho_0 W}{U^2-c^2} \\ . & . & 2 & . \end{pmatrix}. \tag{38}$$

The similarity transform was performed using the `Maxima` computer algebra system[e]. The upwind flux matrices then become

$$\tilde{\boldsymbol{H}}^- \equiv \frac{\tilde{\boldsymbol{D}} - |\tilde{\boldsymbol{D}}|}{2} = \boldsymbol{R}\frac{\boldsymbol{\Lambda} - |\boldsymbol{\Lambda}|}{2}\boldsymbol{R}^{-1}, \quad \tilde{\boldsymbol{H}}^+ \equiv \frac{\tilde{\boldsymbol{D}} + |\tilde{\boldsymbol{D}}|}{2} = \boldsymbol{R}\frac{\boldsymbol{\Lambda} + |\boldsymbol{\Lambda}|}{2}\boldsymbol{R}^{-1}. \tag{39}$$

In the subsonic case, it is $U < c_0$ and one obtains

$$\frac{\boldsymbol{\Lambda} - |\boldsymbol{\Lambda}|}{2} = \begin{pmatrix} U-c_0 & . & . & . \\ . & . & . & . \\ . & . & . & . \\ . & . & . & . \end{pmatrix}, \quad \frac{\boldsymbol{\Lambda} + |\boldsymbol{\Lambda}|}{2} = \begin{pmatrix} . & . & . & . \\ . & U+c_0 & . & . \\ . & . & . & . \\ . & . & . & . \end{pmatrix}. \tag{40}$$

Substitution in Eq. (39) yields

$$\tilde{\boldsymbol{H}}^\pm = \frac{1}{2} \begin{pmatrix} U \pm c_0 & \pm c_0\varrho_0(U \pm c_0) & \pm c_0\varrho_0 V & \pm c_0\varrho_0 W \\ \pm\frac{U\pm c_0}{c_0\varrho_0} & U \pm c_0 & V & W \\ . & . & . & . \\ . & . & . & . \end{pmatrix}. \tag{41}$$

Finally, the upwind flux vector $\tilde{\boldsymbol{F}}_{up} = \tilde{\boldsymbol{H}}^+\tilde{\boldsymbol{u}}^+ + \tilde{\boldsymbol{H}}^-\tilde{\boldsymbol{u}}^-$ becomes

$$\begin{aligned} \tilde{\boldsymbol{F}}_{up} = \frac{1}{2} &\begin{pmatrix} U(p^- + p^+) + c_0^2\varrho_0(u^- + u^+) \\ \frac{p^- + p^+}{\varrho_0} + U(u^- + u^+) + V(v^- + v^+) + W(w^- + w^+) \\ . \\ . \end{pmatrix} - \\ - \frac{c_0}{2} &\begin{pmatrix} (p^- - p^+) + \varrho_0\left(U(u^- - u^+) + V(v^- - v^+) + W(w^- - w^+)\right) \\ (u^- - u^+) + \frac{U}{c_0^2\varrho_0}(p^- - p^+) \\ . \\ . \end{pmatrix} \end{aligned} \tag{42}$$

The face normal vector points outside of the cell volume. Thus, the $\tilde{\boldsymbol{u}}^+ = (p^+, u^+, v^+, w^+)^\mathsf{T}$ denotes the variable vector on the back side of the face, i. e. inside of the cell volume, and $\tilde{\boldsymbol{u}}^- = (p^-, u^-, v^-, w^-)^\mathsf{T}$ are the values outside of the cell, i. e. at the front side of face.

[e] Maxima computer algebra system: http://maxima.sourceforge.net

C. DLR CASE-Cluster

The calculations were performed on the CASE cluster of the DLR Institute of Aerodynamics and Flow Technology in Braunschweig. The cluster consists of 560 compute nodes with two Intel Xeon E5-2695V2 processors (IvyBridge architecture) and 128 GB RAM each. Each processor has a base frequency of 2.4 GHz, 12 cores, and 30 MB level 3 (L3) cache. Without oversubscription and Hyper-Threading 24 cores can be used on each each node.

References

[1] Bauer, M., "Airframe noise prediction using a discontinuous Galerkin method," Forschungsbericht 2011-11, Deutsches Zentrum für Luft- und Raumfahrt, Institut für Aerodynamik und Strömungstechnik, 2011.

[2] Atkins, H. L. and Shu, C.-W., "Quadrature-free implementation of discontinuous Galerkin method for hyperbolic equations," *AIAA journal*, Vol. 36, No. 5, 1998, pp. 775–782.

[3] Tam, C. K. W. and Webb, J. C., "Dispersion-Relation-Preserving Finite Difference Schemes for Computational Acoustics," *Journal of Computational Physics*, Vol. 107, 1993, pp. 262–281.

[4] Zienkiewicz, O. C., Taylor, R. L., and Zhu, J. Z., *The Finite Element Method: Its Basis and Fundamentals*, Butterworth-Heinemann, Oxford, 2005.

[5] Delfs, J., Bauer, M., Ewert, R., Grogger, H., Lummer, M., and Lauke, T., "Numerical Simulation of Aerodynamic Noise with DLRs aeroacoustic code PIANO," Tech. rep., Deutsches Zentrum für Luft-und Raumfahrt eV, Institut für Aerodynamik und Strömungstechnik, 2008.

[6] Ewert, R., Dierke, J., Siebert, J., Neifeld, A., Appel, C., Siefert, M., and Kornow, O., "CAA broadband noise prediction for aeroacoustic design," *Journal of Sound and Vibration*, Vol. 330, 2011, pp. 4139–4160.

[7] Dierke, J., Ewert, R., Chappuis, J., Lidoine, S., and J, R., "Influence of realistic 3D viscous mean flow on shielding of engine-fain noise by a 3-element high-lift wing," Tech. Rep. AIAA-2010-3917, 16th AIAA/CEAS Aeroacoustics Conference, Stockholm, Sweden, June 7–9, 2010.

[8] Dierke, J., Appel, C., Siebert, J., Bauer, M., Siefert, M., and Ewert, R., "3D Computation of Broadband Slat Noise from Swept and Unswept High-Lift Wing Sections," Tech. Rep. AIAA-2011-2905, 17th AIAA/CEAS Aeroacoustics Conference, Portland, Oregon, USA, June 05–08, 2011, http://elib.dlr.de/73668/.

[9] Neifeld, A. and Ewert, R., "On the Contribution of Higher Azimuthal Modes to the Near- and Far-Field of Jet Mixing Noise," Tech. Rep. AIAA-2012-2114, 18th AIAA/CEAS Aeroacoustics Conference, Colorado Springs, Colorado, USA, June 04–06, 2012.

[10] Bauer, M., Dierke, J., and Ewert, R., "Application of a discontinuous Galerkin method to discretize acoustic perturbation equations," *AIAA journal*, Vol. 49, No. 5, 2011, pp. 898–908.

[11] Bauer, M., Dierke, J., and Ewert, R., "On the performance of airframe noise prediction on unstructured grids," Tech. Rep. AIAA-2012-2148, 18th AIAA/CEAS Aeroacoustics Conference, Colorado Springs, Colorado, USA, June 04–06, 2012.

[12] Birkefeld, A., *Computational Aeroacoustics with a High Order Discontinuous Galerkin Scheme*, Ph.D. thesis, Technische Universität Stuttgart, October 2012, http://elib.uni-stuttgart.de/opus/volltexte/2013/7976/.

[13] Ewert, R. and Schröder, W., "Acoustic Perturbation Equations based on Flow Decomposition via Source Filtering," *Journal of Computational Physics*, Vol. 188, 2003, pp. 365–398.

[14] Geuzaine, C. and Remacle, J.-F., "Gmsh: A 3-D finite element mesh generator with built-in pre-and post-processing facilities," *International Journal for Numerical Methods in Engineering*, Vol. 79, No. 11, 2009, pp. 1309–1331.

[15] Karypis, G. and Kumar, V., "A fast and high quality multilevel scheme for partitioning irregular graphs," *SIAM Journal on Scientific Computing*, Vol. 20, No. 1, 1999, pp. 359–392.

[16] Steger, J. L. and Warming, R. F., "Flux Vector Splitting of the Inviscid Gasdynamic Equations with Application to Finite-Difference Methods," *Journal of Computational Physics*, Vol. 40, 1981, pp. 263–293.

[17] Sarfaraz, M. S., *Interpolation of mean flow fields between unstructered CFD and CAA grids*, Master's thesis, Technische Universität Braunschweig, April 2014.

[18] Isotton, A., "C++ dlopen mini HOWTO," Tech. rep., 2006, http://www.tldp.org/HOWTO/C++-dlopen/index.html.

[19] Veldhuizen, T., "Scientific computing: C++ versus Fortran," *DOCTOR DOBBS JOURNAL*, Vol. 22, 1997, pp. 34–41.

[20] Veldhuizen, T. L. and Jernigan, M. E., "Will C++ be faster than Fortran?" *Scientific Computing in Object-Oriented Parallel Environments*, Springer, 1997, pp. 49–56.

[21] Veldhuizen, T. L., "Arrays in blitz++," *Computing in object-oriented parallel environments*, Springer, 1998, pp. 223–230.

[22] Veldhuizen, T., "Blitz++," Tech. rep., 2012, https://sourceforge.net/projects/blitz/files.

[23] Meyer, P. S., *Axisymmetric Acoustic Scattering by Interpolation*, master thesis, Courant Institute of Mathematical Sciences, May 2001, http://phe.rockefeller.edu/perrin/psm_masters.pdf.

[24] Delfs, J., "Grundlagen der Aeroakustik (Basics of Aeroacoustics)," 2014, http://www.dlr.de/as/en/desktopdefault.aspx/tabid-191/401_read-22566/.