Available online at www.sciencedirect.com

ScienceDirect

Journal of Computational Physics 227 (2008) 5286–5303

JOURNAL OF COMPUTATIONAL PHYSICS

www.elsevier.com/locate/jcp

An adaptive multitime multigrid algorithm for time-periodic flow simulations

H. van der Ven

National Aerospace Laboratory NLR, P.O. Box 90502, 1006 BM Amsterdam, The Netherlands

Received 13 September 2007; received in revised form 14 January 2008; accepted 24 January 2008
Available online 5 February 2008

Abstract

The multiscale behaviour and multidisciplinary nature of rotorcraft aerodynamics has delayed the introduction of CFD techniques for rotorcraft aerodynamics. The numerical dissipation of standard CFD algorithms may destroy tip vortices before blade–vortex interaction takes place. More advanced CFD algorithms, using high order discretization and/or local grid refinement, are better suited to tackle the problem. These algorithms, however, generally increase the computational complexity of the simulation and their efficiency should be improved before they can be applied to rotorcraft aerodynamics. In this paper, a four-dimensional solution algorithm will be presented which significantly improves the efficiency by exploiting the periodic nature of rotor flows.

The efficiency of the algorithm is attained by changing a dynamic problem into a static problem, which simplifies the coupling with other models (blade dynamics and elastics, rotor trim), allows local grid refinement in space and time without dynamic load balancing issues, and solves a periodic problem by construction.

A three-dimensional multigrid algorithm for DG discretisations on curvilinear structured meshes is extended to four-dimensional, curvilinear meshes with local grid refinement. Simulations for an oscillating airfoil in subsonic and transonic conditions confirm the performance of the multigrid algorithm and its insensitiveness to highly irregular grid features. The temporal stability restriction of the pseudo-time step is removed by treating the diagonal term of the time derivative implicitly.

PACS: 02.60.Cb; 02.70.Dh; 03.40.Gc

Keywords: Space–time discontinuous Galerkin method; Time-periodic; Time domain; Multigrid

1. Introduction

The simulation of rotorcraft aerodynamics is considerably more complex than the simulation of fixed wing aircraft aerodynamics. Rotorcraft flow is inherently dynamic, the inertial and elastic forces of the rotor blades interact with the aerodynamic forces, and aerodynamic interference of the rotor wake with the fuselage and

E-mail address: venvd@nlr.nl

0021-9991/$ - see front matter © 2008 Elsevier Inc. All rights reserved.
doi:10.1016/j.jcp.2008.01.039

tail rotor is important in many flight conditions. The flow condition known as blade–vortex interaction (BVI) is an important example of such interactions. Especially in low-speed descent, the rotor blades fly in their own wake. The interaction of tip vortices and rotor blades may cause strong pressure fluctuations on the blade, responsible for the typical 'wopwop' sound of helicopters. Prediction of BVI is challenging: the blade motion under inertial, elastic and aerodynamic forces must be predicted correctly and the convection of the tip vortices must be accurate enough to retain the vortices for, typically, one and a half rotor revolution.

The requirement to correctly represent the blade motion has lead to the development of time-accurate flow solvers which are coupled with dynamics solvers or rotor comprehensive codes ([4,17,1]; the reader is also referred to the excellent review paper on rotorcraft CFD by Datta et al. [6]). Several efforts have been undertaken to improve the vortex capturing capability of standard flow solvers, such as local grid refinement [3], Chimera techniques with specific vortex grid systems [7,15], and high order methods [24]. None of these techniques has been particularly successful or efficient, and successful BVI predictions have only been obtained by brute-force methods, using meshes of millions of cells and time-marching several rotor revolutions before structural dynamics, trim and aerodynamics have balanced out. For example, Lim et al. [11] use a series of grids of which the fine grid contains 100 million elements and apply a time step of 0.05 azimuthal degrees. Although the simulations exhibit BVI, the authors are not satisfied with the vortex resolution of the simulation as compared with experiment. They estimate a mesh containing 7 billion elements is necessary for the simulations to agree with experiment. This number is remarkably close to the estimates given by Caradonna [5] in his review article. Clearly, such simulations cannot be run in a routine way and despite the qualitative success so far, there is a need for more efficient algorithms.

In essence, BVI is a multiscale problem, both spatially and temporal. A typical vortex core is 1% of the rotor radius. Accurate capturing of this core requires at least 10 cells in each direction for conventional second order schemes. So the required spatial resolution is 0.1% of the macro length scale. The time scale of the interaction between vortex and blade is in the order of one degree azimuth, while it takes one and a half revolution to get the vortex at the interaction location: a ratio of time scales in the order of 500. As the location of the tip vortices is not known beforehand, local grid refinement based on vorticity sensors is a likely candidate to solve the problem.

Application of local grid refinement for time-dependent simulations is, however, not straightforward. In order to retain efficiency in terms of number of elements, the grid needs to be refined and de-refined at each time step. Given the size of the grids for helicopter applications, the simulation will likely be run on a massively parallel machine using a domain decomposition algorithm to partition the mesh over the processors. Each mesh adaptation will lead to a computational load imbalance which can only be resolved using a relatively expensive repartitioning algorithm. This well-known problem of dynamic load balancing has not been solved yet, and renders grid refinement algorithms for time-dependent simulations on massively parallel machines inefficient in terms of turnaround time.

In this paper a solution algorithm is analysed which addresses the above complications by exploiting the periodic nature of rotor flow. The algorithm itself, a multitime multigrid convergence acceleration algorithm, has first been introduced in [21]. The dynamics are not solved time step after time step, but are solved for all time steps at once. Treating time like space, the equations are solved on a four-dimensional mesh using multigrid techniques to accelerate convergence. Essentially, this turns a dynamic problem into a steady-state problem. This has important consequences for local grid refinement (no dynamic load balancing problems anymore) and the coupling with blade dynamics (no need for intricate time-accurate coupling schemes as in Piperno et al. [16] and Wagner et al. [23]). Moreover, given a suitable numerical scheme, the four-dimensional mesh can also be refined locally in time.

The idea to exploit the periodic nature of rotorcraft problems is not new. Several authors [10,12] have developed frequency domain methods for the Euler and Navier–Stokes equations with applications in rotorcraft and turbomachinery aerodynamics. The current method is a time domain method, which allows local grid refinement in time and simplifies the coupling with time domain dynamics and comprehensive codes.

The underlying numerical scheme is a second order accurate, discontinuous Galerkin space–time method for the Euler equations [20,22], which, using the conventional time-serial solution algorithm, has been successful in predicting rotor flow [2]. The numerical method is briefly described in Section 2. The multitime multigrid algorithm is described in Section 3. The algorithm is based on the spatial multigrid algorithm of Klaij et al. [8]

which is extended to four-dimensions and locally refined meshes. Section 4 discusses some practical details with respect to 4D grid generation, motion representation and local grid refinement.

As described above the BVI problem can in principle be tackled with higher order methods or local grid refinement, and probably a combination of the two is required. The proposed multitime multigrid algorithm is suitable for both approaches. The spatial multigrid algorithm, which is the basis of the MTMG algorithm, has been shown to be efficient for higher order DG discretizations ([9], as yet unpublished), so it is expected that it will be efficient in a multitime multigrid context as well. In Section 5 the MTMG algorithm is applied to locally refined meshes, the same type of meshes which are required for the prediction of BVI. The effectiveness of the MTMG algorithm for these simulations demonstrates its suitability for BVI simulations.

2. Numerical method

In this section, the Euler equations in a moving and deforming flow domain and their space–time discontinuous Galerkin discretization are presented.

2.1. Euler equations in a moving and deforming space–time domain

The Euler equations of gas dynamics are considered in a time-dependent flow domain. Since the flow domain boundary is moving and deforming in time, no explicit separation between the space and time variables is made and the Euler equations are considered directly in $\mathbb{R}^4$. Let $\mathcal{E} \subset \mathbb{R}^4$ be an open domain. A point $x \in \mathbb{R}^4$ has coordinates $(x_1, \ldots, x_4)$, but the notation $t = x_4$ is also frequently used for the time coordinate. The flow domain $\Omega(t)$ at time t, $(t_0 < t < T)$, is defined as $\Omega(t) := \{\bar{x} \in \mathbb{R}^3 | \, (\bar{x}, t) \in \mathcal{E}\}$, with t_0 and T the initial and final time of the evolution of the flow domain. The flow domain $\Omega(t)$ at time t is also referred to as the spatial domain at time t to distinguish it from the space–time domain $\mathcal{E}$. The space–time domain boundary $\partial\mathcal{E}$ consists of the hypersurfaces $\Omega(t_0) := \{x \in \partial\mathcal{E} | x_4 = t_0\}$, $\Omega(T) := \{x \in \partial\mathcal{E} | x_4 = T\}$ and $\mathcal{Q} := \{x \in \partial\mathcal{E} | t_0 < x_4 < T\}$.

Let $\mathcal{F} : \mathbb{R}^5 \to \mathbb{R}^{5\times4}$ denote the flux tensor, which is defined as

$$\mathcal{F} = \begin{pmatrix} \rho u_1 & \rho u_2 & \rho u_3 & \rho \\ \rho u_1^2 + p & \rho u_1 u_2 & \rho u_1 u_3 & \rho u_1 \\ \rho u_1 u_2 & \rho u_2^2 + p & \rho u_2 u_3 & \rho u_2 \\ \rho u_1 u_3 & \rho u_2 u_3 & \rho u_3^2 + p & \rho u_3 \\ (\rho E + p) u_1 & (\rho E + p) u_2 & (\rho E + p) u_3 & \rho E \end{pmatrix},$$

with ρ, p and E the density, pressure, and specific total energy, respectively, and u_i the velocity components in the Cartesian coordinate directions x_i, $i \in \{1, 2, 3\}$ of the velocity vector $u : \mathcal{E} \to \mathbb{R}^3$. Let the vector $U : \mathcal{E} \to \mathbb{R}^5$ denote the conservative flow variables with components:

$$U_i = \mathcal{F}_{i4},$$

then the Euler equations of gas dynamics are defined as

$$\operatorname{div} \mathcal{F}(U(x)) = 0, \quad x \in \mathcal{E}, \tag{1}$$

together with either initial or periodic conditions,

$$\begin{cases} U(x) = U_0(x), & \bar{x} \in \Omega(t_0), \quad \text{or} \\ U(\bar{x}, T) = U(\bar{x}, t_0), & \bar{x} \in \Omega(t_0) = \Omega(T) \end{cases}$$

and boundary conditions:

$$U(x) = \mathcal{B}(U, U_w), \quad x \in \mathcal{Q}.$$

Here $U_0 : \Omega(t_0) \to \mathbb{R}^5$ denotes the initial flow field, $\mathcal{B} : \mathbb{R}^5 \times \mathbb{R}^5 \to \mathbb{R}^5$ the boundary operator and $U_w : \mathcal{Q} \to \mathbb{R}^5$ the prescribed boundary flow field data. The divergence of a second order tensor is defined as $\operatorname{div} \mathcal{F} = \frac{\partial \mathcal{F}_{ij}}{\partial x_j}$, and the summation index is used on repeated indices in this article. The Euler equations are completed with the equation of state for a calorically perfect gas: $p = (\gamma - 1)\rho(E - \frac{1}{2} u_i u_i)$, with γ the ratio of specific heats.

The importance of (1) for the design of solution strategies is that the system of equations is hyperbolic in space and time.

2.2. Geometry of space–time elements

The tesselation $\mathcal{T}_h$ of the space–time domain $\mathcal{E}$ is the union of four-dimensional hexahedral elements[1]:

$$\mathcal{T}_h := \{\mathcal{K}_j | \cup_{j=1}^N \mathcal{K}_j = \mathcal{E} \text{ and } \mathcal{K}_j \cap \mathcal{K}_{j'} = \emptyset \text{ if } j \neq j',\ 1 \leqslant j,\ j' \leqslant N\}.$$

Each element $\mathcal{K} \in \mathcal{T}_h$ is related to the master element $\widehat{\mathcal{K}} = (-1,1)^4$ through the mapping $F_{\mathcal{K}}$:

$$F_{\mathcal{K}} : \widehat{\mathcal{K}} \to \mathcal{K} : \xi \mapsto x = \sum_{i=1}^{16} x_i(\mathcal{K})\chi_i(\xi),$$

with $x_i(\mathcal{K}) \in \mathbb{R}^4$, $1 \leqslant i \leqslant 16$, the space–time coordinates of the vertices of the hexahedron $\mathcal{K}$ and $\chi_i(\xi) = \frac{1}{16}\prod_{j=1}^4 (1 \pm \xi_j)$ the quadlinear finite element shape functions for hexahedra (with the combination of signs depending on the vertex index).

Remark 1. This definition of the tesselation of the space–time domain allows elements which are arbitrarily oriented in space–time. In practice, grid generators are not capable of generating such meshes, and the elements are created by linearly interpolating two three-dimensional, possibly deforming, hexahedra in time (see Section 4.1 for details). It should be noted that the numerical algorithm described in this paper is suited for space–time meshes containing arbitrarily oriented hexahedra. Such meshes are not just a nice mathematical idea: truely four-dimensional meshes would allow the representation of a full helicopter, including fuselage and rotating main and tail rotor, without grid folding and/or the need for Chimera techniques.

2.3. Flow field expansion

The discontinuous Galerkin finite element discretization is obtained by approximating the flow field $U(\bar{x},t)$ and test functions $W(\bar{x},t)$ with polynomial expansions in each element $\mathcal{K}$, which are discontinuous across element faces, both in space and time. In the master element $\hat{\mathcal{K}}$ the basis functions $\hat{\phi}_m$ $(m = 0,\dots,4)$ are defined which are linear in space and time:

$$\hat{\phi}_m(\xi) = \begin{cases} 1, & m = 0, \\ \xi_m, & m > 0. \end{cases}$$

The basis functions $\phi_m^{\mathcal{K}}$ in an element $\mathcal{K}$ are related to the basis function in the master element $\hat{\mathcal{K}}$ through the parametrization $F_{\mathcal{K}}$: $\phi_m^{\mathcal{K}} = \hat{\phi}_m \circ F_{\mathcal{K}}^{-1}$ $(m = 0,\dots,4)$. The superscript $\mathcal{K}$ is dropped when the element in question is clear.

Let $V_h^1(\mathcal{T}_h)$ be the discrete broken function space defined as

$$V_h^1(\mathcal{T}_h) = \{f : \mathcal{T}_h \to \mathbb{R}^5 | f_{|\mathcal{K}} \subset \text{span}\{\phi_m | 0 \leqslant m \leqslant 4\}\}.$$

A solution vector U_h in $V_h^1(\mathcal{T}_h)$ will be written as

$$U_{h|\mathcal{K}} = \sum_{m=0}^{4} \widehat{U}_m^{\mathcal{K}} \phi_m^{\mathcal{K}},$$

with $\widehat{U}_m^{\mathcal{K}}$ the expansion coefficients. The first expansion coefficient is also written as $\overline{U} = \widehat{U}_0$, and is equal to the cell-average if the mesh is cartesian. The other expansion coefficients are referred to as gradients, closely related to the directional derivative in one of the computational directions. With this choice of basis functions the method is second order accurate in space and time.

[1] The correct name for a four-dimensional hexahedron is probably hyper-octahedron, motivated by the name for hypercube and the fact that the polyhedron has eight faces. For convenience sake, the elements are still called hexahedra in this paper.

2.4. Weak formulation of the Euler equations

The weak formulation of the Euler equations is obtained by multiplying the (space–time) Euler equations with a test function W_h, integrating over a space–time element $\mathcal{K}$ and using Gauss' theorem to obtain face flux integrals.

In order to ensure that the weak formulation of the Euler equations is well defined, the broken space $V(\mathcal{T}_h)$ is introduced:

$$V(\mathcal{T}_h) := \{U : \mathcal{T}_h \to \mathbb{R}^5 \mid (\operatorname{grad} U^1)^{\mathrm{T}} : \mathcal{F}(U^2)|_{\mathcal{K}} \in L^1(\mathcal{K});$$
$$\gamma^-(U^1) \cdot \left(n_{\mathcal{K}}^{\mathrm{T}}\mathcal{F}(\gamma^-(U^2)) + n_{\mathcal{K}}^{\mathrm{T}}\mathcal{F}(\gamma^+(U^3))\right) \in L^1(\partial\mathcal{K}); \quad \forall (U^1, U^2, U^3) \in V(\mathcal{T}_h), \quad \forall \mathcal{K} \in \mathcal{T}_h\},$$

with L^1 the space of Lebesgue integrable functions, $\gamma^{\pm}(U) = \lim_{\epsilon\downarrow 0} U(x \pm \epsilon n_{\mathcal{K}})$ the traces of U at $\partial\mathcal{K}$, $n_{\mathcal{K}} \in \mathbb{R}^4$ the unit outward normal vector at $\partial\mathcal{K}$, and superscript T denoting the transposition of a vector.

The gradient operator grad : $\mathbb{R}^5 \to \mathbb{R}^{4\times 5}$ is defined as $(\operatorname{grad} U)_{ij} = \frac{\partial U_j}{\partial x_i}$ and the symbol: represents the dyadic product of two second order tensors and is defined for $\mathcal{A}, \mathcal{B} \in \mathbb{R}^{n\times m}$ as $\mathcal{A} : \mathcal{B} = \mathcal{A}_{ij}\mathcal{B}_{ij}$.

The weak form of the Euler equations is

$$\sum_{j=1}^{N} \left\{ -\int_{\mathcal{K}_j} (\operatorname{grad} W_h)^{\mathrm{T}} : \mathcal{F}(U_h) \mathrm{d}\mathcal{K} + \int_{\partial\mathcal{K}_j} \gamma^-(W_h) \cdot H(\gamma^-(U_h), \gamma^+(U_h), n_{\mathcal{K}}) \mathrm{d}\partial\mathcal{K} \right\} = 0. \tag{2}$$

The numerical flux $H = H(\gamma^-(U_h), \gamma^+(U_h), n_{\mathcal{K}})$ is introduced to stabilize the central flux. In this article, the extension of the HLLC flux for moving meshes is used for the flux through space–time faces. For time faces, which are parallel to the spatial directions, a pure upwind flux is used.

The DG system of equations is obtained by replacing W_h in (2) by $\phi_l^{\mathcal{K}}$ for each $\mathcal{K} \in \mathcal{T}_h$ and $0 \leqslant l \leqslant 4$. The details are of no importance in the current paper and the system of equations is summarised as

$$\mathcal{L}_h(U_h) = 0.$$

Details of the system of equations and its derivation are given in [20].

3. Multitime multigrid algorithm

3.1. Solution algorithm

Conventionally, the tesselation of the space–time domain will be the union of tesselations of time slabs with a single time layer and the equations are solved time slab after time slab. As described in [20] the system of equations per time slab constitute an implicit system, which is solved by adding a pseudo-time derivative to the equations:

$$|\mathcal{K}| \frac{\partial U_h(\mathcal{K})}{\partial \tau} + \mathcal{L}_h(U_h)_{|\mathcal{K}} = 0,$$

where τ is the pseudo-time. The resulting equations are converged to steady-state using standard multigrid techniques for hyperbolic systems and Runge–Kutta smoothers.

This approach can be extended to arbitrary tesselations of the space–time domain. The multigrid algorithm is extended to four-dimensions and the equations are solved for all time levels simultaneously. For general applications, the multigrid algorithm will not be efficient since time waves travel in one direction only. In other words, the solution has to converge in pseudo-time first in the first time slab before a converged solution in the next time slab can be obtained. For time-periodic problems, however, this is no longer the case: all time slabs influence one another.

Details of this four-dimensional multitime multigrid (MTMG) algorithm for time-periodic simulations are described in the following sections.

Remark 2. The multitime multigrid algorithm can be applied to any numerical discretization of any hyperbolic system of equations. The DG method has the added bonus of the functionality for local grid refinement, in both space and time, without the need of interpolation methods between regions with different time steps. The nodes at the boundary of time-refined regions are hanging nodes in time and can be treated by the DG method just like hanging nodes in space. See Fig. 5 in Section 4 for an illustration of a space–time mesh with temporal hanging nodes.

3.2. Smoother

In compressible CFD, usually a full multigrid algorithm is applied with Runge–Kutta smoothers. The smoother for the multitime multigrid algorithm is the Runge–Kutta 5 scheme of [20], with optimised coefficients for the DG discretisation of the advection equation.

The stability and smoothing characteristics of the Runge–Kutta scheme will be analysed for the time-dependent, one-dimensional convection equation $\frac{\partial u}{\partial t} + c\frac{\partial u}{\partial x} = 0$, with $c > 0$. The equation is discretised on a uniform space–time mesh, periodic in space and time, with mesh width δx, and with time step δt. The aspect ratio of the space–time cells is related to the physical CFL number, $\mathrm{CFL}_f = c\delta t/\delta x$.

For clarity of presentation, the convection equation is discretised using only the constant basis function, and the linear basis function in time (so the spatial gradient is discarded). Hence in each cell there are two variables: the cell-averaged quantity $\bar{u}$ and the time gradient $\hat{u}$. Let $\boldsymbol{u}_i^n = (\bar{u}_i^n, \hat{u}_i^n)^{\mathrm{T}}$ be the solution in spatial cell i in time slab n. Deriving the equations from the weak form, we get the following two equations, where the first is obtained by substituting the cell-average basis function as test function in the weak form, and the second by substituting the time gradient basis function

$$\begin{aligned}&\left(\bar{u}_i^n + \hat{u}_i^n - \bar{u}_i^{n-1} - \hat{u}_i^{n-1}\right)\delta x + c\left(\bar{u}_i^n - \bar{u}_{i-1}^n\right)\delta t = 0,\\ &\left(-\bar{u}_i^n + \hat{u}_i^n + \bar{u}_i^{n-1} + \hat{u}_i^{n-1}\right)\delta x + \frac{c}{3}\left(\hat{u}_i^n - \hat{u}_{i-1}^n\right)\delta t = 0,\end{aligned}$$

which are abbreviated to $\mathcal{L}'_h\boldsymbol{u}_h = 0$, where $\boldsymbol{u}_h = \{(\boldsymbol{u}_i^n) | 1 \leqslant i \leqslant N_x, 1 \leqslant n \leqslant N_t\}$ is the solution vector (with N_x, resp. N_t, the number of spatial cells, resp. time steps).

In order to solve the equations a psuedo-time derivative is added, properly scaled with the space–time volume:

$$\delta t\delta x\frac{\partial \boldsymbol{u}_h}{\partial \tau} + \mathcal{L}'_h\boldsymbol{u}_h = 0.$$

In order to make clear the relationship with the physical and pseudo-time step, this equation is rewritten as

$$\frac{\partial \boldsymbol{u}_h}{\partial \tau} + \frac{1}{\delta t}\mathcal{L}_h\boldsymbol{u}_h = 0,$$

with $\mathcal{L}_h = \mathcal{L}'_h/\delta x$. The matrix $\mathcal{L}_h$ is a matrix consisting of block 2×2 matrices. All block-diagonals are the same and equal to

$$\begin{pmatrix} 1 & 1 \\ -1 & 1 \end{pmatrix} + \mathrm{CFL}_f\begin{pmatrix} 1 & 0 \\ 0 & \frac{1}{3} \end{pmatrix}, \tag{3}$$

the first matrix comes from the time derivative, the second from the convective term.

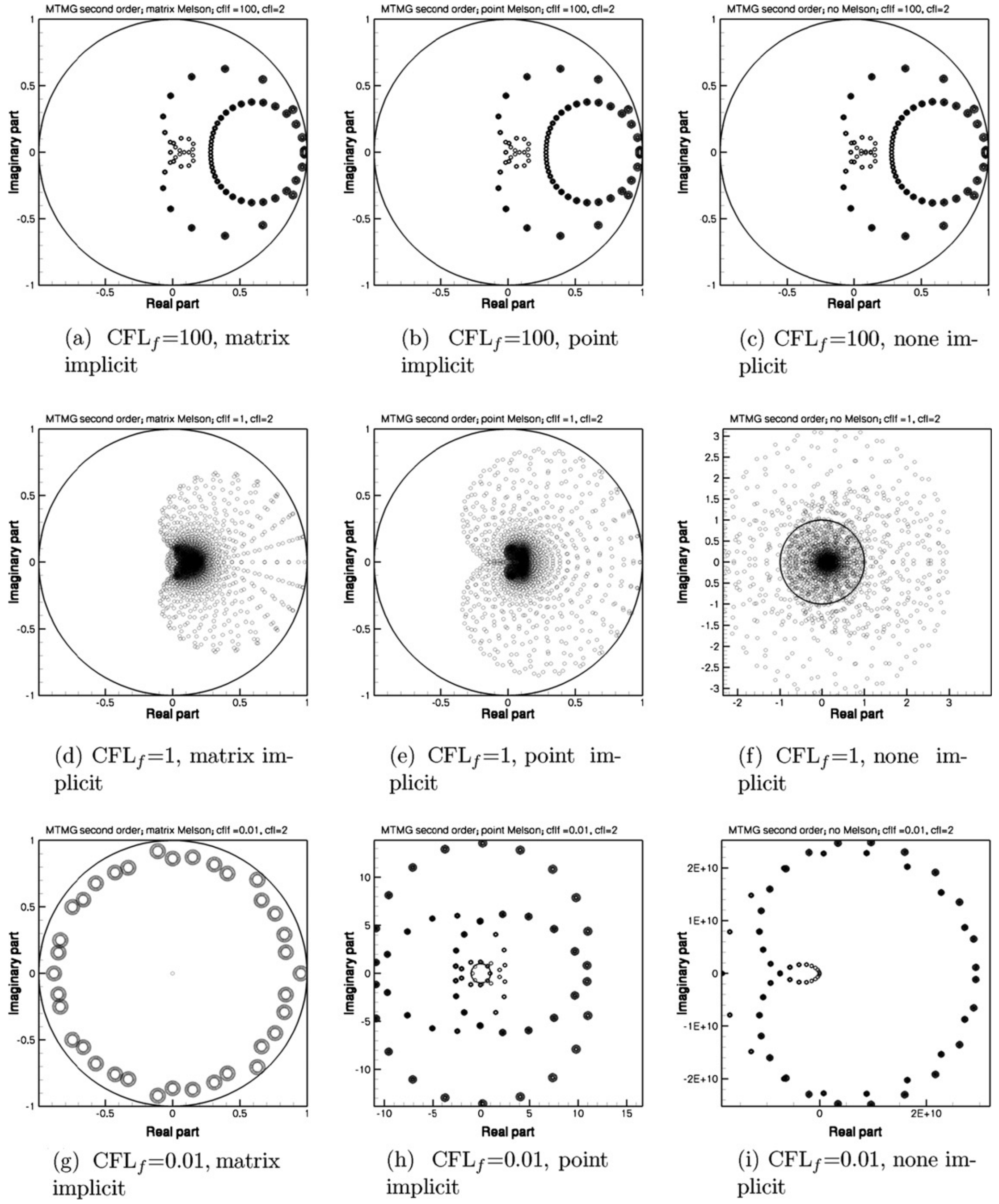
The system is converged to steady-state using a low-storage Runge–Kutta scheme. Each stage of such a scheme consists of

$$\boldsymbol{u}_i^{n,s} = \boldsymbol{u}_i^{n,s-1} - \alpha_s\frac{\delta\tau}{\delta t}\mathcal{L}_i^n(\boldsymbol{u}_h^{s-1}), \tag{4}$$

where $\delta\tau$ is the pseudo-time step, and $\boldsymbol{u}_h^{s-1} = \{(\boldsymbol{u}_i^n)^{s-1}\}$ is the solution vector at the previous stage.

Since the equations are solved for space and time simultaneously, there are two stability constraints, determined by the CFL numbers $\mathrm{CFL}_x = \frac{c\delta\tau}{\delta x}$ in the spatial direction and $\mathrm{CFL}_t = \frac{\delta\tau}{\delta t}$ in the temporal direction. The temporal CFL restriction is the most restrictive when the physical CFL number is less than one. For conventional schemes this time step restriction can be removed by treating the discretised time derivative implicitly [13]. The same approach will be taken here.

Remembering that the solution $\boldsymbol{u}_i^n$ is a vector consisting of the cell-average and the time gradient, implicit treatment of the diagonal term of the time discretisation will lead to



(a) CFL$_f$=100, matrix implicit

(b) CFL$_f$=100, point implicit

(c) CFL$_f$=100, none implicit

(d) CFL$_f$=1, matrix implicit

(e) CFL$_f$=1, point implicit

(f) CFL$_f$=1, none implicit

(g) CFL$_f$=0.01, matrix implicit

(h) CFL$_f$=0.01, point implicit

(i) CFL$_f$=0.01, none implicit

Fig. 1. Stability analysis of the space–time DG discretization of the convection equation for the RK5 scheme, with matrix-implicit, point-implicit, or no treatment of the time derivative; for different physical CFL numbers. Shown is the amplification factor in the complex plane, together with the unit circle. All figures with CFL = 2. Note the different scaling of the axes for the unstable methods.

$$\left(1+\alpha_s\frac{\delta\tau}{\delta t}A\right)\boldsymbol{u}_i^{n,s}=\boldsymbol{u}_i^{n,s-1}-\alpha_s\frac{\delta\tau}{\delta t}(\mathcal{L}_i^n(\boldsymbol{u}_h^{s-1})-A\boldsymbol{u}_i^{n,s-1}), \tag{5}$$

where A is the matrix given in (3):

$$A=\begin{pmatrix}1 & 1\\ -1 & 1\end{pmatrix}.$$

Eq. (5) is obtained from (4) by adding $\alpha_s\frac{\delta\tau}{\delta t}A\boldsymbol{u}_i^{n,s}$ to the LHS and $\alpha_s\frac{\delta\tau}{\delta t}A\boldsymbol{u}_i^{n,s-1}$ to the RHS. This method will be referred to as the matrix-implicit Melson correction.

For completeness' sake also the Melson correction of Van der Vegt et al. [20] is considered in the analysis. In this method the matrix A is replaced by the identity matrix. This method will be referred to as the point-implicit Melson correction.

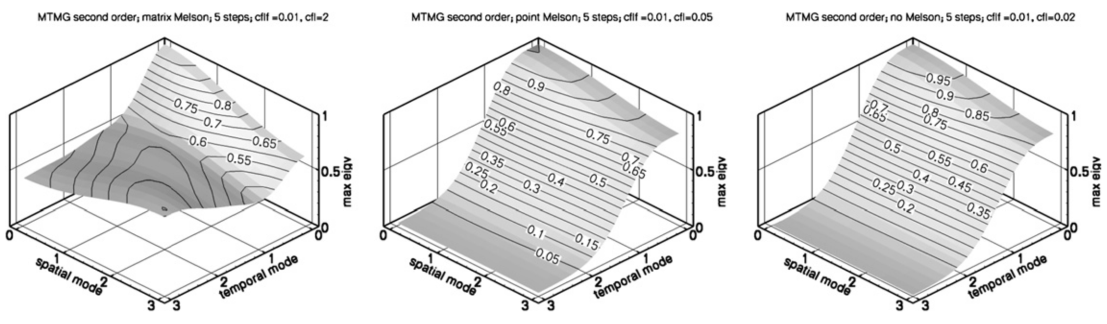
The Runge–Kutta scheme with and without point-implicit or matrix implicit Melson correction has been analysed for different physical CFL numbers. The amplification factor of the Runge–Kutta scheme is computed with standard Fourier analysis. The results are plotted in Figs. 1 and 2. The amplification factor is plotted for different Fourier modes, in space and (physical) time. For a stable scheme all modes should have an absolute value of the amplification factor smaller than one. For convenience the unit circle is plotted in the figures. All analyses have been done with a pseudo-CFL number of two (CFL $=\frac{c\delta\tau}{\delta x}$), which is the stable CFL number for this Runge–Kutta scheme for spatial simulations.

From Fig. 1 it is clear that all schemes are stable for CFL$_f>1$. For CFL$_f=1$ the point-implicit scheme is still stable, but the standard scheme without implicit terms is unstable. For CFL$_f<1$, for which the implicit methods are devised, only the matrix-implicit scheme remains stable (note the different scaling of the axes for the unstable schemes). The instability of the Runge–Kutta scheme without the corrections is to be expected since the dominating pseudo-time step constraint is ignored.

Fig. 2 compares the smoothing properties of the schemes for CFL$_f=0.01$. For each scheme the maximum stable pseudo-time step is taken. The maximum eigenvalue for each Fourier mode is plotted. Five Runge–Kutta steps are taken to make the differences in damping properties more clear. It is clear from the figure that the matrix-implicit scheme has much better smoothing properties for the high spatial modes than the other schemes. This is of importance to the efficiency of the multigrid algorithm.

Hence the matrix implicit method improves the stability of the pseudo-time integration, by effectively making the stability of the scheme independent of the size of the physical time step. Moreover, the smoothing property of the scheme is improved.

Concluding this analysis, it is good to realise that from the four-dimensional view point the temporal CFL restriction dominates for high-aspect ratio cells, where the physical time step δt is much smaller than the spatial mesh width δx. The analysis has shown that the smoothing properties of the Runge–Kutta smoother can



(a) CFL=2, matrix implicit (b) CFL=0.05, point implicit (c) CFL=0.02, none implicit

Fig. 2. Stability analysis of the space–time DG discretization of the convection equation for the RK5 scheme, with matrix implicit, point implicit, or no treatment of the time derivative; for physical CFL number equal to 0.01. The pseudo-CFL numbers are such that the scheme is stable.

be improved when $\delta t \ll \delta x$, but the non-optimal smoothing properties when $\delta x \ll \delta t$ remain. The reason for this is that the temporal flux is much easier to analyse than the spatial fluxes, and does not require a linearisation of the flux. A block-diagonal implicit system arising from a linearisation of the spatial fluxes and ignoring all off-diagonal terms may improve the smoothing properties of the Runge–Kutta scheme, analogous to the non-linear element Jacobi smoother of Nastase et al. [14], while retaining the locality of the DG scheme.

Remark 3. In earlier papers on the space–time DG method the importance of matrix implicit Melson was not noted because the stability analysis was performed in the space domain (since the space–time equations were solved time slab after time slab), and not in the space–time domain.

3.3. Multigrid algorithm

The multigrid algorithm is taken from Klaij et al. [8]. As explained in Section 2.1 the time derivative has the same behaviour as the convective operator with convection velocity equal to one. Hence there is no need to repeat the two-level analysis presented in [8]. The multigrid algorithm is simply applied to four instead of three-dimensional flow simulations. It remains to extend the restriction and prolongation operators to four-dimensions and to locally refined meshes.

The prolongation operator is defined as the L_2-projection of the coarse grid solution to the fine grid solution. In order of this definition to make mathematical sense, the fine grid function space should be embedded in the coarse grid function space. This embedding is satisfied for uniform meshes, but for curvilinear meshes this is in general not the case.

On a one-dimensional mesh, a coarse grid cell $\mathcal{K}_H$ is the union of two fine grid cells $\mathcal{K}_h^\pm$ where the sign is determined by the coarse grid cell basis function:

$$\pm\phi_1^{\mathcal{K}_H}{}_{|\mathcal{K}_h^\pm} \geqslant 0.$$

The L_2-projection in one-dimension on a uniform mesh is easily computed to be

$$\left(\begin{array}{cc|cc} 1 & -\frac{1}{2} & 1 & \frac{1}{2} \\ 0 & \frac{1}{2} & 0 & \frac{1}{2} \end{array}\right) \tag{6}$$

which maps $(\overline{U}^{\mathcal{K}_H}, \widehat{U}^{\mathcal{K}_H})^{\mathrm{T}}$ to $(\overline{U}^{\mathcal{K}_h^-}, \widehat{U}^{\mathcal{K}_h^-} | \overline{U}^{\mathcal{K}_h^+}, \widehat{U}^{\mathcal{K}_h^+})^{\mathrm{T}}$. Note that the prolongation operator contains no geometrical details.

On locally refined meshes the coarse grid levels are obtained by agglomeration of fine grid cells using the grid refinement tree. The grid refinement tree keeps track of the grid refinements of a given cell. If that cell is refined anisotropically in one direction, two kid cells are added to the refinement tree at the cell's location. The kid cells are the newly created cells. The refinement tree also keeps track of the refinement direction. The agglomeration algorithm traverses the refinement tree backwards until a sufficiently large number of cells is agglomerated into a coarse grid cell. For the four-dimensional MTMG algorithm the number of agglomerated cells should be about 16 (2^4). Fig. 3 illustrates the algorithm in two-dimensions. Note that on locally refined meshes it is no longer ensured that a coarse grid cell is obtained by coarsening once in every direction. The advantages of this agglomeration algorithm are firstly that each agglomerated cell is a hexahedron, for which the DG discretization is defined, and secondly that the fine grid cells are defined with respect to the coarse grid cell as a sequence of local grid refinements in certain directions. This facilitates the definition of the prolongation operator: it is defined as the product of prolongation operators per refinement direction, corresponding to the sequence of grid refinements needed to obtain the fine grid cell.

To be more precise, given a coarse grid cell $\mathcal{K}_H$, let $C_l^\pm : \mathcal{K}_H \mapsto \mathcal{K}_h^\pm$ be the refinement operator which refines the coarse grid cell in the lth computational direction ($1 \leqslant l \leqslant 4$):

$$\mathcal{K}_H = \mathcal{K}_h^- \cup \mathcal{K}_h^+ = C_l^-(\mathcal{K}_H) \cup C_l^+(\mathcal{K}_H), \quad \pm\phi_l^{\mathcal{K}_H}{}_{|\mathcal{K}_h^\pm} \geqslant 0.$$

The corresponding prolongation operator $P_l^\pm : V_H^1(\mathcal{K}_H) \to V_h^1(\mathcal{K}_h^\pm)$ is defined as $P_l^\pm(U^{\mathcal{K}_H}) = \sum_k p_{lk}\phi_k^{\mathcal{K}_h^\pm}$ where the fine grid cell expansion coefficients are given by

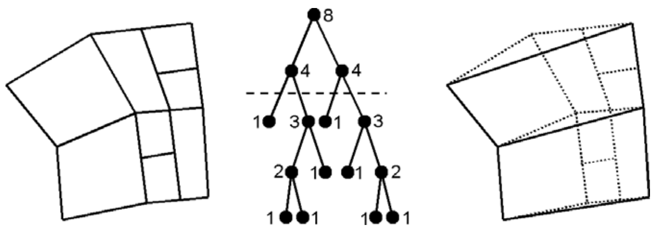


Fig. 3. Illustration of the agglomeration algorithm in two-dimensions. To the left a group of 8 cells is shown, created by adaptation of an original group of four cells. The corresponding refinement tree is shown in the middle. Each node represents a cell in the refinement process, the numbers correspond to the number of leaves in the subtree starting at the node. The leaves of the tree correspond to the cells at the finest grid level. The dashed line in the refinement tree signifies the location where the refinement tree is pruned to obtain the coarse grid level. The resulting group of cells is shown to the right. Note that the orientation of the two coarse grid cells (horizontal in the figure) depends on the order of the refinements in the refinement tree.

$$\begin{cases} p_{l0} = \widehat{U}_0^{\mathcal{K}_H} \pm \frac{1}{2}\widehat{U}_l^{\mathcal{K}_H}, & k = 0, \\ p_{ll} = \frac{1}{2}\widehat{U}_l^{\mathcal{K}_H}, & k = l, \\ p_{lk} = \widehat{U}_k^{\mathcal{K}_H}, & \text{otherwise.} \end{cases}$$

Note that this prolongation operator is obtained by extending the operator defined in (6) to the identity for the gradients which are in the other directions than the refinement direction. For an arbitrary fine grid cell $\mathcal{K}_h$ and a coarse grid cell $\mathcal{K}_H$ which contains $\mathcal{K}_h$, there is a sequence of n refinement operators $C_{l_i}^{s_i}$, $1 \leqslant i \leqslant n$, such that

$$\mathcal{K}_h = \left(\prod_i C_{l_i}^{s_i}\right)(\mathcal{K}_H), \quad 1 \leqslant l_i \leqslant 4, \;\; s_i \in \{\pm\}.$$

The prolongation operator $P : V_H^1(\mathcal{K}_H) \to V_h^1(\mathcal{K}_h)$ is then defined as

$$P = \prod_i P_{l_i}^{s_i}.$$

Since the prolongation operators in different computational directions commute, the definition of the prolongation operator is independent of the order of the refinement operators. Ignoring the geometrical details, such as skewness, in the definition of the prolongation operator is different from what is described in [8]. The main motivation to ignore the geometrical details is that multigrid algorithms concern the solution vector rather than geometry. Another motivation is that under the assumption that locally the coarse and fine meshes are geometrically similar, the prolongation operator does not contain geometrical details. Results in Section 5 will show that the algorithm with the simplified prolongation operators is effective on locally refined meshes.

The restriction operator for the residuals is defined as the transpose of the prolongation operator. The restriction operator for the solution vector is defined as the inverse of the prolongation operator. These

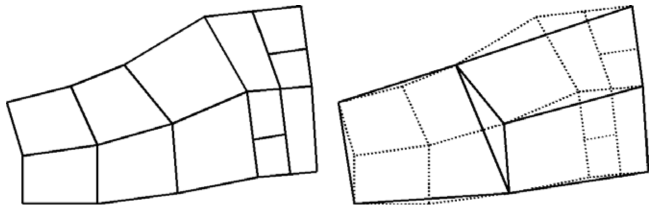


Fig. 4. Illustration of the grid folding caused by the agglomeration algorithm on curvilinear, locally refined meshes. To the left a set of fine grid cells are shown: the fine grid cells of Fig. 3 with a neighbouring group of four cells without refinement. To the right the resulting set of coarse grid cells is shown. The four cells are coarsened to a single cell, whereas the cells of Fig. 3 are coarsened to two cells. The resulting hanging node is located on its original location of the curvilinear mesh, and not on the straight edge of the coarse grid cell on its left. So the coarse grid cells do not cover the computational domain.

definitions have been shown to be effective for the multigrid solution of the DG discretization of the laminar Navier–Stokes equations.

It should be noted that on locally refined, curvilinear meshes the computational domain will in general not be a disjunct union of the coarse grid cells obtained from the above agglomeration algorithm. This is illustrated in Fig. 4, where the agglomeration algorithm leads to coarse grid cells which do not completely cover the computational domain. The reverse, overlapping coarse grid cells, may also happen. The effect of this type of grid folding on the coarse levels is difficult to analyse (no need to mention that it would be disastrous on the fine level), but the forcing function in the multigrid algorithm contains the residual of the grid folding as well, and hence it is expected that the algorithm is robust. This will be demonstrated experimentally in Section 5.

4. 4D grid generation

4.1. Simple approach

As remarked before, there are no 4D grid generators, hence the 4D space–time meshes must be generated 'by hand'. The most simple way to construct a 4D mesh is collating a series of spatial meshes to form a four-dimensional mesh. The spatial meshes, including grid deformation to accommodate the geometry motion, are generated in the conventional way. An in-house block-structured grid generator called ENGRID [19] is used to generate the original mesh. The meshes at a given time are deformed to accommodate the geometry motion. In this way each space–time element in the 4D mesh is the linear interpolation in time of a 3D element at a time $t = t_n$ and a 3D element at time $t = t_{n+1}$.

4.2. Motion representation under time refinement

The standard local refinement algorithm obtains the coordinates of new grid points by linearly interpolating between two existing grid points. Grid consistency for internal cells is ensured if the original cell is a hexahedron parameterised by a quadlinear mapping. Obviously, this refinement algorithm does not preserve the curvature of the geometry. A remedy is to project newly created grid points on the geometry onto the geometry definition. In principle, this may lead to grid folding if the original mesh does not capture the curvature sufficiently well. For most meshes the original curvature is captured well, so this is a feasible algorithm for spatial refinement.

For time refinement, however, this is not the case. In general, the physical CFL number for cells near the geometry will be in the order of 100, implying an aspect ratio of the same magnitude. Projecting a newly created geometry point (due to time refinement) onto the geometry (that is, ensuring the correct grid motion) is hence much more likely to incur grid folding. On the other hand, accuracy is seriously impaired if the location of geometry points created under time refinement does not correspond to the analytically defined geometry motion. The motion is then only correctly represented on the original time resolution of the 4D mesh.

To overcome this problem, several strategies have been tried. The most obvious one is to generate a 4D mesh as described in Section 4.1 with sufficient time resolution as an initial mesh. For rotor simulations, where a spatial mesh typically contains half a million grid cells, and 100 time steps per period are required to capture phenomena as BVI, this would lead to meshes of 50 million elements, even before spatial refinement to capture the vortices, forfeiting the efficiency claims of the MTMG algorithm.

Another approach is to leave the geometry points created under time refinement at their linearly interpolated position, but modify the slip boundary condition so as to accommodate the actual grid motion. This boundary condition is known as the transpiration boundary [18]. The geometry motion and grid motion need not coincide, and the difference is compensated by prescribing a transversal flow along the boundary (effectively a linearization of the correct boundary motion on the incorrect mesh). This is an efficient approach for potential methods and obliterates the need for a grid deformation algorithm. However, it turned out that the linearization underlying the transpiration boundary condition was not capable of removing the non-smoothness of the linearly interpolated grid motion.

The final approach is described in the next section.

4.3. Partial time levels

During anisotropic refinement new time levels may be created which do not cover the original computational space-domain. The ability of the numerical method to accommodate such meshes can also be exploited during grid generation: not all time levels need to cover the computational space-domain. So the idea is to limit the space-domain for selected time levels to a region near the geometry. This approach reduces the number of grid cells on the initial four-dimensional mesh, while correctly representing the geometry motion for a sufficient number of time levels.

An illustration is given in Fig. 5 for a simple spatially one-dimensional flow domain, representative of a 1D piston problem. The left side of the spatial domain oscillates in time. The space–time mesh contains three full ($\Omega(t_0), \Omega(t_4)$ and $\Omega(t_8)$) and six partial time levels. The spatial end points of the partial time levels are hanging nodes in the space–time mesh. In order to ensure that the space–time mesh is folding-free, the end points of the partial time levels should be located on the straight lines between the corresponding spatial points of the full time levels. This is most easily accomplished by fixing the location in space of the end points of the partial time levels.

Summarising, connected domains around the moving geometries are chosen which will comprise the spatial computational domain of the partial time levels. The grid deformation algorithm used to accommodate the geometry motion is only used in this domain. Dirichlet boundary conditions are applied for the deformation algorithm: equal to the geometry motion at the geometry and zero on 'farfield' boundary of the domain. For the full time levels grid points outside the spatial computational domain of the partial time levels are not allowed to move.

Geometrically, the extent of this region is restricted by the ability of the grid deformation algorithm to accommodate the geometry motion. Aerodynamically, the initial extent of this region is irrelevant, since the region can be extended during the simulation using local grid refinement in time.

The proposed algorithm on the one hand allows to start with a sufficiently fine time resolution about the geometry, while on the other hand restricting the total number of cells in the space–time mesh. A time-uniform mesh of the space–time domain shown in Fig. 5 would contain 64 cells, the mesh shown contains only 40 cells. For the 2D simulations in the next section, the mesh with 64 time levels, of which only 4 are full, only contains 8500 elements, whereas a time-uniform mesh would contain 32,000 elements. For rotor applications the benefit is even greater.

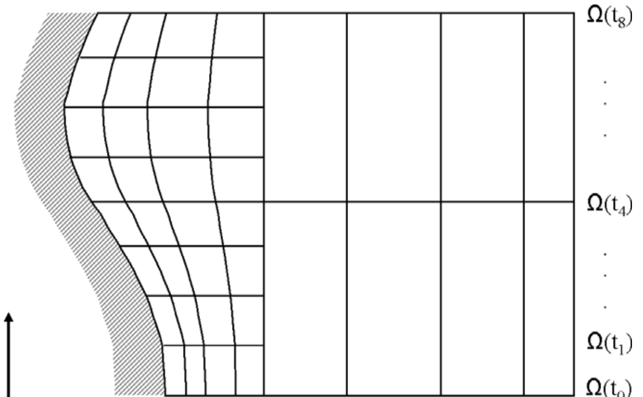


Fig. 5. Illustration of the partial time levels for a spatially one-dimensional domain with a moving boundary on the left. The space–time mesh contains three full ($\Omega(t_0), \Omega(t_4)$ and $\Omega(t_8)$) and six partial time levels.

5. Results

In this section, the MTMG algorithm will be applied to spatially two-dimensional simulations. Application to rotorcraft simulations will be the subject of future papers. The aim of the simulations is to demonstrate the feasibililty of the MTMG algorithm for computations on locally refined meshes, such as are needed for BVI simulations. All simulations will concern an oscillating NACA0012 airfoil, either in subsonic or transonic conditions. The airfoil oscillates at a reduced frequency $\omega^* = \frac{\omega c}{a_\infty} = 0.4917$ with an amplitude of three degrees around an angle of attack of zero degrees, where c is the chord length, and a_∞ is the freestream speed of sound.

The first simulation is a demonstration of the grid independence of the convergence rate. A three-dimensional mesh (two spatial and one temporal direction) with 32 time levels and 8192 spatial cells at all time levels is created. The mesh contains five multigrid levels with, respectively, 64, 512, 4096, 32,768 and $262{,}144 = 32 \cdot 8192$ space–time cells. A full-multigrid simulation over the three finer levels has been conducted, using a V cycle with two pre-relaxations and two post-relaxations at each grid level (denoted as '2-2 cycles' in the legend of the figures). At a Mach number of 0.5 the flow is subsonic.

The convergence of the four equations (cell-averaged, two spatial gradients and one temporal gradient) are shown in Fig. 6. The dashed lines in the figure are parallel and it is clear from the figures that the asymptotic convergence rate obtained on the coarse mesh (over three grid levels) is also obtained on the medium mesh (over four levels) and the fine mesh (over five levels). The fact that the convergence rate is similar on all grids demonstrates that the multigrid algorithm is functioning properly. The rate of convergence itself is determined by the smoother, and improvements in the smoother, such as a diagonally implicit treatment of the linearised spatial fluxes, will be the subject of future work.

The next simulations concern the performance of the multigrid algorithm on locally refined meshes. Three three-dimensional meshes with partial time levels are generated starting from a coarse spatial mesh of 512 elements. The three grids contain 16, 32, resp. 64 time steps for a period, and three, seven, resp. 15 partial time levels between full ones. Hence all grids contain four full time levels. The deformation region is extended half a chord length from the airfoil. Impressions of the grid are shown in Fig. 7. Transonic simulations at a Mach number 0.8 are performed. As part of the simulation, local grid refinement with a standard shock sensor is applied, which measures the jumps over faces in a certain computational direction and anisotropically refines the neighbouring cells in that direction which connect to faces with the largest jumps. The time resolution is restricted to the one on the original mesh: cells may be refined in time, but not below the original time step size of 16, 32, resp. 64 to a period.

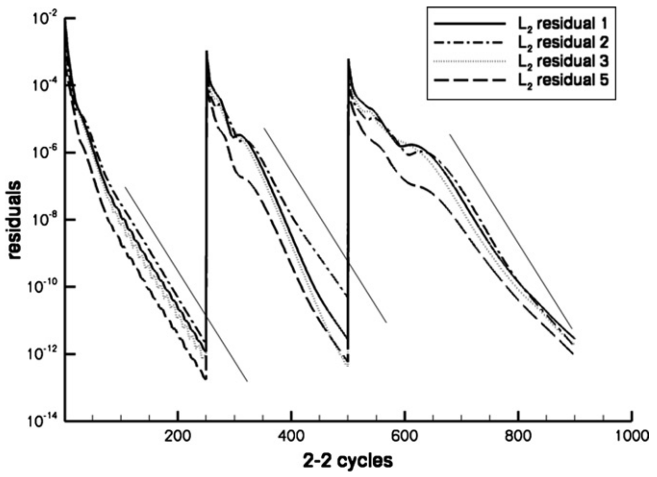


Fig. 6. Demonstration of grid independence of the convergence rate of the multigrid algorithm. Full multigrid simulation of a subsonic oscillating NACA0012 on a series of time-uniform meshes with 4096, 32,768, resp. 26,2144 elements and three, four, resp. five multigrid levels. The dashed lines are parallel.

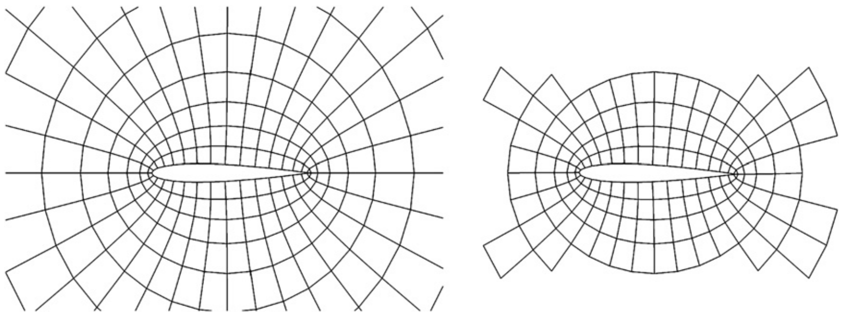


Fig. 7. Initial grid near the airfoil: full time level (left) and partial time level (right).

Each simulation on one of the three grids consists of a convergence to engineering accuracy on the original grid, six adaptations based on the flow solution to improve the shock capturing, and finally a convergence to machine accuracy. The convergence histories are shown in Figs. 8–10. The histories look very similar despite the different number of time steps and partial time levels. Note that the convergence rate on the final, locally refined mesh, is the same as on the original mesh for all three grids. As remarked in Section 3.3 the coarse grid levels of the locally refined meshes will contain grid folding, but the convergence results show that the performance of the multigrid algorithm is unaffected by it.

Impressions of the mesh and flow solution at various time levels for the case with 64 time steps are presented in Fig. 11. It is clear that the shock sensor is capable of anisotropically refining the mesh in order to improve the shock capturing. Note that for the partial time levels (time step index not a multiple of 16) the greater part of the computational domain is now covered through local time refinement. Some gaps in the spatial domains are still visible. Note that local grid refinement only takes place at the shock positions. For a conventional time-serial adaptation algorithm this can only be accomplished using local refinement and de-refinement at each time step since the shock changes position. In order to simulate one period for this case, 64 grid adaptations would be required, whereas for MTMG only six adaptations are required. For future rotorcraft simulations this is a relevant test case: for rotorcraft applications we want to trace vortices in stead of shocks, but they too change position. Locally refining the mesh at each time step becomes infeasible for three-dimensional applications because of dynamic load balancing issues on the needed parallel computers.

The last simulation compares the performance of the multitime multigrid algorithm with the performance of a single grid algorithm and with the performance of the multigrid algorithm of Van der Vegt et al. [20],

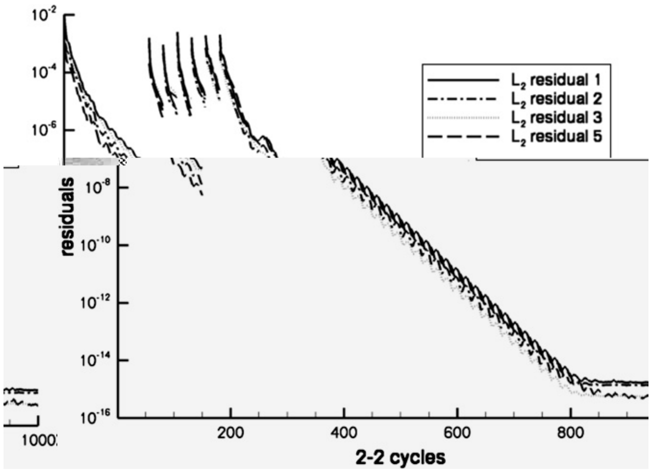


Fig. 8. Convergence history of transonic oscillating NACA0012 with grid refinement. The oscillation period is divided into 16 time steps and the mesh contains three partial time levels between full ones.

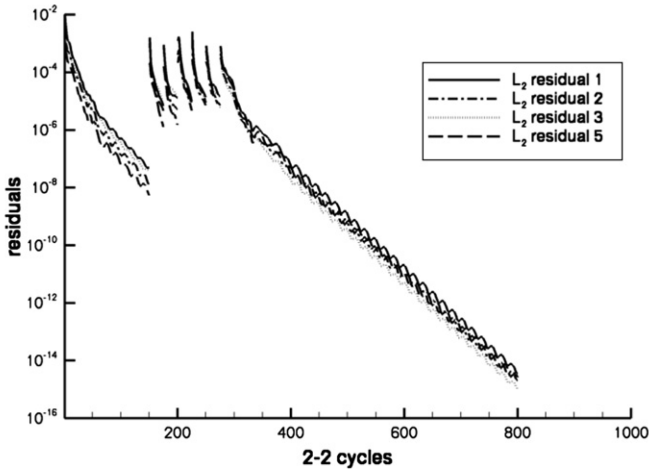


Fig. 9. Convergence history of transonic oscillating NACA0012 with grid refinement. The oscillation period is divided into 32 time steps and the mesh contains seven partial time levels between full ones.

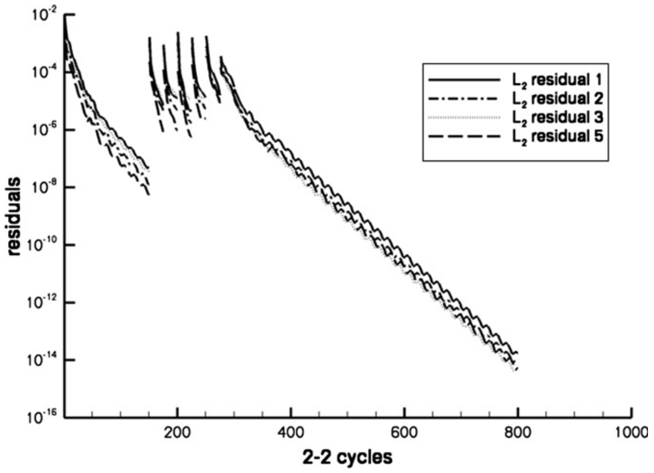


Fig. 10. Convergence history of transonic oscillating NACA0012 with grid refinement. The oscillation period is divided into 64 time steps and the mesh contains fifteen partial time levels between full ones.

which does not solve all equations on the coarse grid levels, but only retains the equations for the cell-averaged solution. A transonic simulation is performed on the final locally refined mesh with 16 time levels obtained from the previously described simulations. This mesh is an extreme test case for any convergence acceleration algorithm.

The convergence history for the three methods is shown in Fig. 12. The methods are compared with respect to the number of fine grid relaxations, which is a good metric for relative computational complexity. The work done on a coarse grid level by the multigrid algorithms is at most 1/16 of the work done on the fine levels, and as such is negligible. Clearly the multitime multigrid algorithm outperforms the other two. The MTMG algorithm is about four times faster than the single grid algorithm in obtaining a solution which is converged to machine accuracy. For practical applications, only engineering accuracy (three to four orders decrease in residual) is required. The MTMG algorithm obtains a residual level of 10^{-5} in 200 fine grid relaxations, whereas the single grid computation requires 1500 fine grid relaxations. So for practical application the relative performance of the MTMG algorithm is a factor of eight. Compared to the multigrid algorithm using a

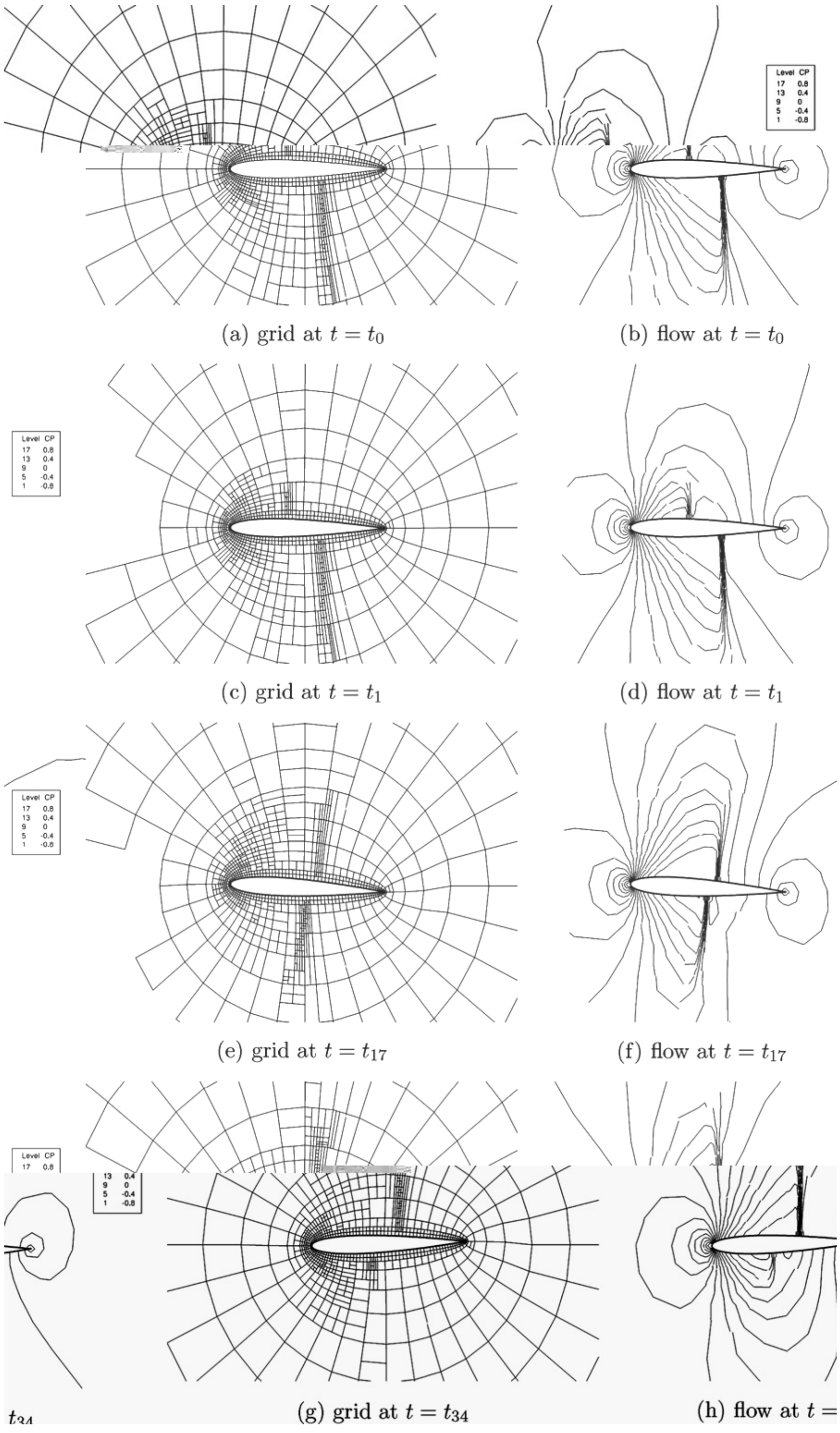


Fig. 11. Impression of grids and flow fields for the transonic oscillating airfoil with 64 time steps.

first order discretization on the coarse levels the relative performance of the MTMG algorithm is more than a factor two for engineering accuracy.

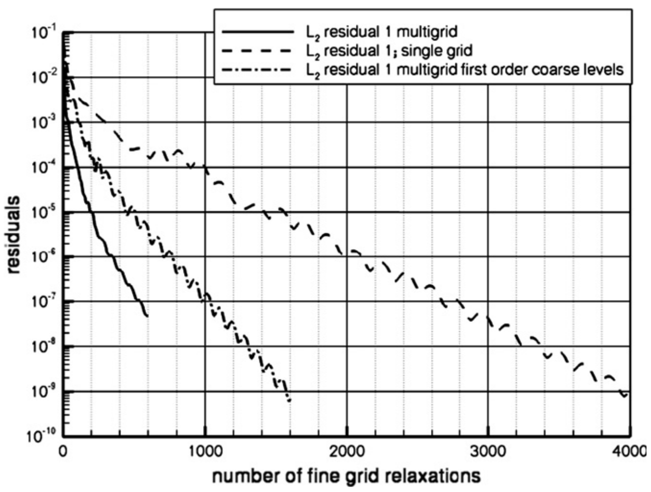


Fig. 12. Comparison of convergence history of transonic oscillating NACA0012 with grid refinement for the multigrid algorithm, the multigrid algorithm with first order discretization on the coarse grid levels and the single grid algorithm. The mesh is the final locally refined mesh containing 16 time levels.

6. Conclusions

The four-dimensional convergence acceleration algorithm MTMG for the simulation of time-periodic problems significantly decreases the computational complexity of rotorcraft CFD simulations. The multitime multigrid algorithm allows efficient simulations of time-periodic aero-elastic problems with local grid refinement to capture relevant flow features. The efficiency is attained by changing a dynamic problem into a static problem, which simplifies the coupling with other models (blade dynamics and elastics, rotor trim), allows local grid refinement in space *and* time without dynamic load balancing issues, and solves a periodic problem by construction.

A three-dimensional multigrid algorithm for DG discretisations on curvilinear structured meshes can be extended to four-dimensional, curvilinear meshes with local grid refinement. Simulations for an oscillating airfoil in subsonic and transonic conditions confirm the performance of the multigrid algorithm and its insensitiveness to highly irregular grid features. The temporal stability restriction of the pseudo-time step can be removed by treating the diagonal term of the time derivative implicitly.

In future publications the application of the algorithm to the simulation of rotorcraft will be presented.

Acknowledgements

The work described in this paper is partially carried out within the European project ADIGMA on the development of innovative solution algorithms for aerodynamic simulations, and partially by NLR's programmatic research.

References

[1] A.R.M. Altmikus, S. Wagner, G. Servera, P. Beaumier, A comparison: weak versus strong coupling for trimmed aeroelastic rotor simulations, in: Proceedings of the 29th European Rotorcraft Forum, September, 2003.
[2] O.J. Boelens, H. van der Ven, B. Oskam, A.A. Hassan, Boundary conforming discontinuous Galerkin finite element approach for rotorcraft simulations, J. Aircraft 39 (5) (2002) 776–785.
[3] C.L. Bottasso, M.S. Shephard, A parallel adaptive finite element Euler flow solver for rotary wing aerodynamics, AIAA J. 35 (6) (1997) 937–944.
[4] B. Buchtula, S. Wagner, Rotory wing aeroelasticity in forward flight with refined wake modeling, in: Proceedings of the 24th European Rotorcraft Forum, Marseille, September, 1998.
[5] F.X. Caradonna, Development and challenges in rotorcraft aerodynamics, AIAA paper 2000-0109.

[6] A. Datta, M. Nixon, I. Chopra, Review of rotor loads prediction with the emergence of rotorcraft CFD, J. Am. Helicopter Soc. 52 (4) (2007) 287–317.

[7] K. Duraisamy, J.D. Baeder, High resolution wake capturing methodology for hovering rotors, J. Am. Helicopter Soc. 52 (2) (2006) 110–122.

[8] C.M. Klaij, M.H. van Raalte, H. van der Ven, J.J.W. van der Vegt, *h*-Multigrid for space–time discontinuous Galerkin discretizations of the compressible Navier–Stokes equations, J. Comput. Phys. 227 (2007) 1024–1045.

[9] D.T.P. Köster, J.J.W. van der Vegt, H. van der Ven, C.M. Klaij, *h*-Multigrid for higher order space–time discontinuous Galerkin discretizations of the compressible Navier–Stokes equations, in: Proceedings of the Fifth ECCOMAS 2008 Congress, July 2008, Venice, 2008.

[10] M. Kumar, V.R. Murthy, Rotor blade response based on CFD in the frequency domain, AIAA paper 2006-438.

[11] J.W. Lim, R.C. Strawn, Prediction of HART II rotor BVI loading and wake system using CFD/CSD loose coupling, AIAA paper 2007-1281.

[12] M. McMullen, A. Jameson, J. Alonso, Application of a non-linear frequency domain solver to the Euler and Navier–Stokes equations, AIAA paper 2002-0120.

[13] N.D. Melson, M.D. Sanetrik, H.L. Atkins, Time-accurate Navier–Stokes calculations with multigrid acceleration, in: Proceedings of the Sixth Copper Mountain Conference on Multigrid Methods, NASA Conference Publication 3224, 1993.

[14] C.R. Nastase, D.J. Mavripilis, High-order discontinuous Galerkin methods using an *hp*-multigrid approach, J. Comput. Phys. 213 (2006) 330–357.

[15] A. Ochi, T. Aoyama, S. Saito, E. Shima, E. Yamakawa, BVI noise predictions by moving overlapped grid method, in: Proceedings of the AHS 55th Forum, Montreal, Canada, May 25–27, 1999.

[16] S. Piperno, C. Farhat, B. Larrouturou, Partitioned procedures for the transient solution of coupled aeroelastic problems, Part I: model problem, theory and two-dimensional application, Comput. Methods Appl. Mech. Eng. 24 (1995) 79–112.

[17] H. Pomin, S. Wagner, Aeroelastic analysis of helicopter rotor blades on deformable Chimera grids, J. Aircraft 41 (3) (2004) 577–584.

[18] L.N. Sankar, S.Y. Ruo, J.B. Malone, Application of surface transpiration in computational aerodynamics, AIAA paper 86-0511.

[19] S.P. Spekreijse, J.W. Boerstoel, Multiblock grid generation, in: 27th Computational Fluid Dynamics Course, Von Karman Institute for fluid dynamics, March 25–29, 1996.

[20] J.J.W. van der Vegt, H. van der Ven, Space–time discontinuous Galerkin finite element method with dynamic grid motion for inviscid compressible flows. Part I. General formulation, J. Comput. Phys. 182 (2002) 546–585.

[21] H. van der Ven, O.J. Boelens, B. Oskam, Multitime multigrid convergence acceleration for periodic problems with future applications to rotor simulations, in: P. Wilders, A. Ecer, J. Periaux, N. Satofuka, P. Fox (Eds.), International Parallel CFD 2001 Conference (May 21–23, 2002), North-Holland, Elsevier, 2002.

[22] H. van der Ven, J.J.W. van der Vegt, Space–time discontinuous Galerkin finite element method with dynamic grid motion for inviscid compressible flows. Part II. Efficient flux quadrature, Comput. Methods Appl. Mech. Eng. 191 (2002) 4747–4780.

[23] S. Wagner, A.R.M. Altmikus, H. Pomin, Coupled numerical simulation of aerodynamics and rotor dynamics of a helicopter in forward flight, in: Proceedings of the Fifth World Congress on Computational Mechanics, Vienna, Austria, July 7–12, 2002, <http://wccm.tuwien.ac.at/>.

[24] B.E. Wake, D. Choi, Investigation of high-order upwind differencing for vortex convection, AIAA J. 34 (2) (1996) 332–337.