CHALMERS
UNIVERSITY OF TECHNOLOGY

A semi-implicit slip algorithm for mesh deformation in complex geometries, implemented in OpenFOAM

Downloaded from: https://research.chalmers.se, 2025-12-10 00:25 UTC

Citation for the original published paper (version of record):
Salehi, S., Nilsson, H. (2023). A semi-implicit slip algorithm for mesh deformation in complex geometries, implemented in
OpenFOAM. Computer Physics Communications, 287. http://dx.doi.org/10.1016/j.cpc.2023.108703

N.B. When citing this work, cite the original published paper.

research.chalmers.se offers the possibility of retrieving research publications produced at Chalmers University of Technology. It covers all kind of research output: articles, dissertations, conference papers, reports etc. since 2004. research.chalmers.se is administrated and maintained by Chalmers Library

(article starts on next page)

Computer Physics Communications 287 (2023) 108703

Contents lists available at ScienceDirect

Computer Physics Communications

journal homepage: www.elsevier.com/locate/cpc

A semi-implicit slip algorithm for mesh deformation in complex geometries, implemented in OpenFOAM ☆,☆☆

Saeed Salehi *, Håkan Nilsson

Division of Fluid Dynamics, Department of Mechanics and Maritime Sciences, Chalmers University of Technology, Gothenburg SE-412 96, Sweden

ARTICLE INFO

Article history:
Received 17 May 2022
Received in revised form 31 January 2023
Accepted 15 February 2023
Available online 24 February 2023

Keywords:
Mesh motion
Semi-implicit slip
OpenFOAM
Kaplan turbine
Transient operation

ABSTRACT

Many engineering applications of computational fluid dynamics (CFD) comprise extensive movement of objects that necessitate complex dynamic mesh treatments. In particular, the mesh motion process frequently requires a proper slipping of mesh points on highly curved surfaces. The currently available implementation of explicit slip boundary conditions in OpenFOAM fails to allow large deformations of the mesh without severely degrading the mesh quality and inverting some of the cells. Thus, a robust semi-implicit slip algorithm, based on the Laplacian smoothing methodology, is developed in the present work to tackle this issue. The algorithm is in fact performed in two steps, one explicit and one implicit. The OpenFOAM implementation of the algorithm includes different mesh motion solvers and boundary conditions, based on the displacement or velocity of points. The method is first verified using simple, yet relevant, test cases, and it is shown that the developed algorithm significantly outperforms some of the well-known proprietary CFD codes. Then, it is applied to a complex practical CFD case study. An engineering application that requires the features of the developed mesh motion algorithm is the transient operation of Kaplan turbines. These double-regulated machines simultaneously adjust the guide vane and runner blade angles while changing the operating condition. CFD simulations of such transient operations are highly complex, as they involve mesh deformation of the guide vane passage and simultaneous mesh deformation and rigid-body rotation of the runner blade passage. The mesh deformation requires points to slip on the curved hub and shroud surfaces while preserving the cell quality in tiny blade clearances. Therefore, the feasibility of the developed algorithm is evaluated for a load rejection sequence of a Kaplan turbine model.

Program summary
Program Title: Semi-implicit slip mesh motion
CPC Library link to program files: https://doi.org/10.17632/wztc26vh7b.1
Developer's repository link: https://github.com/salehisaeed/semiImplicitSlip
Licensing provisions: GPLv3
Programming language: C++
Nature of problem: CFD simulations of numerous engineering fluid flows, such as transient operation of hydraulic turbines, involve an immensely complicated mesh motion process consisting of simultaneous mesh deformation and mesh slipping on highly curved surfaces. The available standard mesh motion methodology in OpenFOAM lacks some features to simulate this elaborate mesh motion. The introduced program addresses this problem by developing a new dynamic mesh algorithm.
Solution method: The program implements a robust semi-implicit algorithm for slipping the mesh points on curved surfaces. The algorithm includes two steps, namely, an explicit step based on the general slip condition and an implicit step based on the Dirichlet condition. It employs the Laplacian smoothing equations to spread the mesh deformation into the domain. Additionally, a solid-body rotation may be added on top of the deformed mesh, which could be useful for modeling the runner region in transient

☆ The review of this paper was arranged by Prof. Hazel Andrew.
☆☆ This paper and its associated computer program are available via the Computer Physics Communications homepage on ScienceDirect (http://www.sciencedirect.com/science/journal/00104655).

* Corresponding author.
E-mail addresses: saeed.salehi@chalmers.se (S. Salehi), hakan.nilsson@chalmers.se (H. Nilsson).

https://doi.org/10.1016/j.cpc.2023.108703
0010-4655/© 2023 The Author(s). Published by Elsevier B.V. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

operation of Kaplan turbines which contains simultaneous mesh deformation and solid-body rotation of the mesh.

© 2023 The Author(s). Published by Elsevier B.V. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

1. Introduction

Numerous real-life applications of computational fluid dynamics (CFD) involve moving objects, such as aeroelastic modeling of wings, biological flows, and vibration of turbomachinery blades. Transient operation of hydraulic turbines is another example of engineering flows that involves moving surfaces. During such operations, the flow rate varies through adjustable guide vanes that change their angle. In the case of Kaplan turbines, the runner blade angles are also adapted simultaneously to maintain a high efficiency over a wide range of flow rates.

Three main approaches are available for CFD of flows with moving objects, namely, mesh morphing [1] (also known as Arbitrary Lagrangian Eulerian, ALE), overset mesh [2,3], and the Immersed Boundary Method (IBM) [4]. Each approach has its own advantages and disadvantages that will be briefly discussed here.

Mesh morphing is historically the most widely used approach for flow simulations with moving or deforming boundaries. It is based on an initial body-fitted mesh that is deformed according to the motion of the boundaries. The main advantage of the approach is that it provides precise mass conservation by maintaining the original mesh topology and cell connectivity, while accurately capturing the boundary layer effects through the body-fitted mesh. A disadvantage is that large geometrical changes may significantly degrade the mesh quality (e.g., non-orthogonality and skewness). Further, although modern meshing tools are highly developed, it may be burdensome to generate the body-fitted meshes required for the mesh morphing approach, especially in the case of practical engineering applications with flows through complex geometries (such as hydraulic turbines).

The overset mesh approach is based on a background mesh and one overlapping body-fitted mesh for each moving object. All meshes that overlap exchange information during the simulation. The main convenience of this method is that the individual meshes do not deform during the simulation, even if they are moving with respect to each other. This allows very large mesh motion without changing the mesh quality. The mesh generation may also be easier than for the single body-fitted mesh needed for the mesh morphing approach. However, the approach has the disadvantage of performing the challenging and costly interpolation-based coupling between the meshes. It also involves complicated dynamic load balancing for parallel computations and elaborate priority management for multiple overlapping meshes [5]. The search and interpolation algorithms can become computationally expensive as the number of overset regions grows. A simulation of a Kaplan turbine transient sequence would for example require one overset mesh for each guide vane and runner blade. This could drastically affect the efficiency of the computations. More importantly, the coupling of overlapping meshes is based on an interpolation procedure which is restricted to the local information and does not guarantee global mass conservation. The accuracy of the interpolation may also be affected by the ratio of cell sizes in the region of interpolation. This may be a problem when an overset mesh comes close to the body-fitted boundary of another overset mesh or the boundary of the background mesh. Völkner et al. [5] analyzed the non-conservative coupling of the overset mesh approach and explained that it causes nonphysical mass and pressure fluctuations. Nevertheless, due to the convenience of the methodology, it has recently been evaluated for the simulation of guide vane movement during a load rejection process of a high-head Francis turbine [6] and pump-turbines during startup [7] and shutdown [8].

The Immersed Boundary Method (IBM) has primarily been developed to mitigate the difficulties to generate body-fitted meshes for complex geometries. The method can also be employed as a convenient alternative for modeling flows with moving objects. IBM was first introduced by Peskin [9] to model cardiac blood flow. Since then, many variants of this method have been introduced in the literature (e.g., [10–21]). In IBM, the equations are discretized on a background mesh. The stationary or moving object is specified by a surface that identifies where the equations on the background mesh should be influenced such that the presence of the object appropriately affects the resulting flow. This has the potential to make mesh generation extremely easy. However, the way to manipulate the equations in the background mesh is not a straightforward task, and the effects of the immersed boundary treatment on the conservation properties of the numerical models are not clear [4]. The main disadvantage of IBM is that the immersed boundary surfaces and their boundary layers are not accurately resolved, as in the other dynamic mesh approaches. To resolve the boundary layer at the immersed boundary, the background mesh in that region needs to be sufficiently fine. For a moving immersed boundary, this applies to the entire region where the object may appear. A remedy to this is to apply time-varying local refinement in the background mesh, which however introduces additional complexity and computational cost. Although the background mesh is sufficiently fine, the discretization at the immersed boundary will in most cases still not be ideal. This may lead to spurious numerical oscillations in the vicinity of the moving object, which can significantly deteriorate the accuracy of the solution [22]. The methodology has not yet been adopted for common use in complex engineering applications such as simulation of transient sequences of hydraulic machines [23].

As discussed above, the mesh morphing approach is still the most widely used alternative for CFD with moving or deforming boundaries. It is also the most mature in terms of accuracy, mass conservation and stability in general-purpose CFD codes. It has been repeatedly employed in the literature for simulations of load change operations of Francis turbines, i.e., load rejection [24–26], load acceptance [27], shutdown [28–31], and startup [32,33]. However, a thorough review of the literature reveals that very few studies have been dedicated to the transient operation of Kaplan turbines. This may be explained by the complexity of such procedures in both numerical and experimental investigations. In Kaplan turbines, both the guide vanes and runner blades change their angles while the runner is rotating, as opposed to Francis turbines where only the guide vanes are adjustable. Kaplan turbines also have special blade clearances that are not present in Francis turbines which augments the mesh deformation complexity. Hence, the current paper focuses on the mesh morphing methodology and addresses the challenges in the dynamic mesh process for highly curved surfaces, such as those in Kaplan turbines.

A mesh morphing framework, consisting of different boundary conditions and mesh motion solvers, is developed in the OpenFOAM open-source CFD code. The instabilities of an explicit point slip boundary condition on curved surfaces are explained in detail, and a novel robust semi-implicit algorithm is proposed to tackle this issue. The feasibility of the developed program is evaluated on a Kaplan turbine model subjected to load rejection. The current

paper extends and further improves the mesh deformation framework introduced by Salehi et al. [34].

It should particularly be noted that although the current dynamic mesh framework is primarily developed and tested for Kaplan turbines, which is the application area of the authors, the semi-implicit slip algorithm can be utilized for any mesh motion problem that involves slipping points on highly curved surfaces.

The paper is organized as follows. Section 2 briefly describes the theory of finite volume discretization on dynamic meshes. The classical mesh deformation algorithms and the challenges with the explicit slip boundary condition are explained in Section 3. Section 4 introduces the developed robust algorithm for mesh deformation of Kaplan turbines during transient operation. The performance of the developed numerical framework is assessed using a model Kaplan turbine case study in Section 5. Finally, conclusions are drawn in Section 6.

2. Finite volume discretization for dynamic meshes

This section discusses the theory of finite volume discretization for dynamic meshes, as well as its implementation in OpenFOAM. In the case of dynamic meshes, the surfaces of the control volumes (mesh faces) may move with a specific velocity and may deform over time. Using Gauss' theorem, it can be shown [1] that the integral form of the general incompressible transport equation for a scalar ϕ over a moving/deforming arbitrary control volume V with closed surface S, is stated as

$$\frac{\mathrm{d}}{\mathrm{d}t}\int_V \phi\,\mathrm{d}V + \oint_S \mathbf{n}\cdot(\mathbf{u}-\mathbf{u}_s)\phi\,\mathrm{d}S - \oint_S \gamma_\phi \mathbf{n}\cdot\nabla\phi\,\mathrm{d}S = \int_V s_\phi\,\mathrm{d}V. \quad (1)$$

Here, $\mathbf{n}$ is the unit normal vector of the control volume surface, pointing out of the control volume. $\mathbf{u}$ and $\mathbf{u}_s$ represent the fluid and control volume surface velocities, respectively. γ_ϕ is the diffusion coefficient and s_ϕ is the source term of scalar ϕ. One can see that the general conservation equation for a dynamic mesh can be derived by simply replacing the velocity in the convective term with the relative velocity in the same conservation equation for a fixed mesh. Finite volume discretization of Eq. (1), using the implicit second order time discretization for a fixed time step (referred to as `backward` in OpenFOAM), yields

$$\frac{3\phi_P^{\mathrm{n}}V_P^{\mathrm{n}} - 4\phi_P^{\mathrm{o}}V_P^{\mathrm{o}} + \phi_P^{\mathrm{oo}}V_P^{\mathrm{oo}}}{2\Delta t} + \sum_f (F_f^{\mathrm{n}} - F_s^{\mathrm{n}})\phi_f^{\mathrm{n}} - \sum_f (\gamma_\phi)_f^{\mathrm{n}} S_f^{\mathrm{n}} \mathbf{n}_f^{\mathrm{n}}\cdot(\nabla\phi)_f^{\mathrm{n}} = s_\phi^{\mathrm{n}} V_P^{\mathrm{n}}, \quad (2)$$

where subscripts P and f denote cell-centered and face-centered values. Superscripts n, o and oo represent current time (at time t), previous time (at time $t-\Delta t$), and the time before that (at time $t-2\Delta t$), respectively. $F_f = \mathbf{n}_f\cdot\mathbf{u}_f S_f$ is the volumetric flux through a stationary control volume face due to the fluid velocity, while F_s is the volumetric flux corresponding to the cell face moving through a quiescent fluid and needs to be calculated. Additionally, $\mathbf{n}_f$ and S_f represent the unit normal vector and area of the face, respectively.

The Space Conservation Law (SCL) relates the rate of change in volume of a control volume (V) with the velocity of its boundary surfaces ($\mathbf{u}_s$) as

$$\frac{\mathrm{d}}{\mathrm{d}t}\int_V \mathrm{d}V - \oint_S \mathbf{n}\cdot\mathbf{u}_s\,\mathrm{d}S = 0. \quad (3)$$

It has been shown [35] that the SCL must be satisfied to prevent the generation of artificial mass sources in the continuity equation. Discretizing Eq. (3) using the implicit `backward` second order time scheme yields

$$\frac{3V_P^{\mathrm{n}} - 4V_P^{\mathrm{o}} + V_P^{\mathrm{oo}}}{2\Delta t} - \sum_f F_s^{\mathrm{n}} = 0. \quad (4)$$

It is important that this discretization is done with the same time scheme as the general transport equation to avoid artificial mass sources. That is why OpenFOAM reimplements the mesh motion flux for each time discretization scheme. Therefore, in OpenFOAM, the SCL is satisfied by the fact that the flux due to the mesh motion is calculated through the swept volumes of the cell faces and not the mesh face velocity $\mathbf{u}_s$ [36,37].

3. Classical mesh morphing algorithms in OpenFOAM

In this section, the classical mesh morphing algorithms, commonly used in OpenFOAM, are first presented. Then, the explicit slip boundary condition and its deficiencies are assessed, since it is of great importance for the applicability of mesh morphing techniques in complex geometries.

In most mesh morphing problems, a certain type of motion is applied on one or multiple boundary surfaces. The boundary movement could be introduced through either a specified motion, prescribed as a boundary condition or a flow-driven motion calculated as a part of the solution where the solid boundary interacts with the fluid (fluid-structure interaction or six-degree-of-freedom solvers). Either way, the positions of the moving boundary mesh points evolve in time, which results in severe deformation of the first cell layer attached to the boundary unless the internal cells are consistently deformed to adapt to the moving boundary.

A valid computational mesh should have cells that fill the entire computational domain without any overlaps. The criteria for mesh validity suggest that the whole computational domain can be considered as an elastic solid medium encountering deformation governed by the Piola–Kirchhoff stress-strain equation [36]. However, the non-linearity of this governing equation makes the dynamic mesh calculations expensive. Therefore, cheaper alternatives such as the Laplace equation and the Solid-Body Rotation (SBR) stress equation are used in OpenFOAM for performing mesh deformation. The Laplacian mesh morphing method is employed in this study and thus explained in Section 3.1 while readers are referred to the literature for the SBR stress approach [38].

The mesh motion equation can solve for the displacement ($\boldsymbol{\delta}_{\mathrm{point}}$) or the velocity ($\mathbf{u}_{\mathrm{point}}$) of the mesh points (referred to as `pointDisplacement` and `pointMotionU` in OpenFOAM, respectively). The displacement approach calculates the displacement of the points with respect to the initial mesh at $t=0$, while the velocity approach calculates the velocity of the points such that they reach their new position during the time step (similar to computing the displacement with respect to their position at the previous time step). The velocity approach is typically more stable for large deformations, while displacement solvers are preferred for oscillating motions.

Relating to elastic solid deformation, the first discretization strategy that comes to mind is finite-element discretization. However, since OpenFOAM is a cell-centered finite volume CFD solver, the same functionalities are reused also for the mesh motion. Therefore, OpenFOAM solves the dynamic mesh equations for the displacement ($\boldsymbol{\delta}_{\mathrm{cell}}$) or velocity ($\mathbf{u}_{\mathrm{cell}}$) of the *cell centers* (referred to as `cellDisplacement` and `cellMotionU` in OpenFOAM, respectively). The corresponding results for the points are then calculated by interpolating the cell results to the points. The interpolation is usually performed using inverse distance weighting.

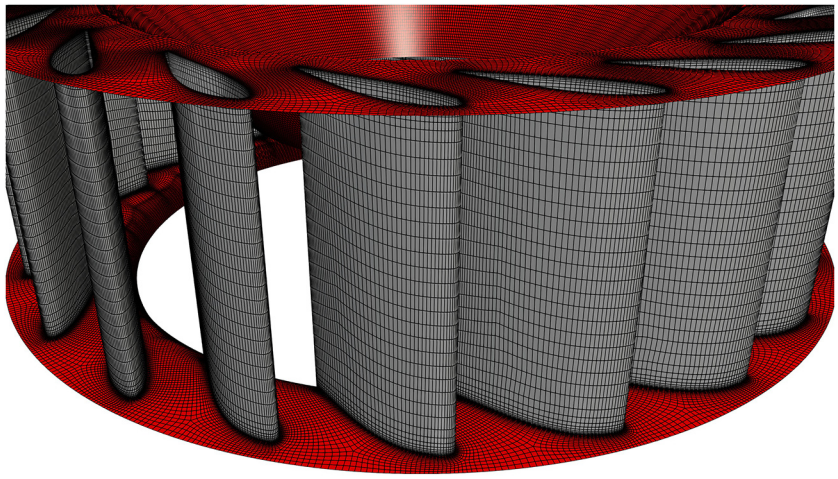


Fig. 1. Illustration of guide vane domain for a Kaplan turbine. (For interpretation of the colors in the figures, the reader is referred to the web version of this article.)

Once the displacements or velocities of the points have been determined for a time step, their actual new locations must be calculated. For the displacement approach, the new point locations can be acquired by

$$\mathbf{x}^{\mathrm{n}}_{\mathrm{point}} = \mathbf{x}^{0}_{\mathrm{point}} + \boldsymbol{\delta}^{\mathrm{n}}_{\mathrm{point}}, \tag{5}$$

where superscripts n and 0 denote the current and initial times, respectively. For the velocity approach, the new point locations are obtained through

$$\mathbf{x}^{\mathrm{n}}_{\mathrm{point}} = \mathbf{x}^{\mathrm{o}}_{\mathrm{point}} + \mathbf{u}^{\mathrm{n}}_{\mathrm{point}} \Delta t, \tag{6}$$

where superscript o indicates the point locations at the previous time step and Δt is the time step size, assuming a fixed time step. As seen in Eqs. (5) and (6), the displacement solver always uses the original locations of the mesh points (at $t = 0$) to calculate the new locations, while the velocity solver deals with the location of the points at the previous time step. Thus, the displacement solver is usually more suitable for oscillating types of motions whereas the velocity solver is more appropriate for large non-oscillating deformations.

Although it is the cell-centered displacement or velocity that is solved, the mesh motion boundary conditions are in OpenFOAM imposed on the point displacement or point velocity fields. These are automatically transferred to the cell-centered fields, which are the ones that are actually needed for the solution process. Both implicit Dirichlet and Neumann-type boundary conditions are applicable to the displacement and velocity fields. Explicit type boundary conditions that are corrected after obtaining the solution of the linear system can also be employed in the mesh motion calculations.

3.1. Laplacian mesh morphing

One of the most widely used approaches for mesh deformation in CFD codes is the diffusion-based smoothing algorithm, i.e. the Laplacian equation. The Laplace equation propagates and smoothens the boundary conditions for the mesh motion throughout the internal cells. The Laplace equation is linear and numerically cheap to solve. However, since it is employed for a vector field when applied to mesh deformation, a disadvantage is that the spatial directions are decoupled, which can lead to failure.

The vectorial Laplace equation for an arbitrary vector field $\mathbf{v}$ (may be mesh point displacement or velocity) is given by

$$\nabla \cdot (\Gamma \nabla \mathbf{v}) = \mathbf{0}. \tag{7}$$

Here, Γ is the mesh motion diffusivity coefficient, which specifies the rate of spreading of the boundary mesh motion to the interior mesh. The performance of the mesh deformation is strongly dependent on the diffusivity, which may be spatially varying. A common way of specifying a spatially varying diffusion coefficient is the distance-based method. For example, the diffusivity can be computed based on the inverse distance

$$\Gamma = \frac{1}{l^m}, \tag{8}$$

or the exponential of the distance

$$\Gamma = e^{-l}, \tag{9}$$

where l is the distance from the closest moving boundary. These approaches produce larger diffusivity close to moving boundaries, which makes the cells close to those boundaries behave more like solid bodies that follow the boundary motion and deform less than cells further away.

3.2. Slip condition

It is the boundary conditions that drive the mesh morphing equations toward their final solution. It is straightforward to set Dirichlet conditions for boundaries with specified mesh motion. However, some boundaries must allow the mesh to slip on the geometrical surfaces, and this is not straightforward for complex geometries and situations when there is both a prescribed motion and a slip condition at the same time.

In the present work, as a test case, we consider Kaplan turbines under transient operation, with both guide vanes and runner blades that continuously change their angles. For the sake of simplicity, we first discuss the guide vane domain. A typical guide vane domain of a Kaplan turbine is demonstrated in Fig. 1, where the gray surfaces are the guide vanes. In both Francis and Kaplan turbine transients, the guide vanes change their angle (either close down or open up) around their own individual axes. Thereby, for the CFD simulation of such procedures, one can prescribe the guide vane rotation using a Dirichlet boundary condition for the motion displacement or velocity field of the points, i.e., `pointDisplacement` or `pointMotionU` in OpenFOAM. The Dirichlet boundary condition is supposed to appropriately rotate each guide vane around its own axis. In most cases, the guide vane domain mesh is produced separately, requiring axis-symmetric upstream and downstream interfaces to the rest of the computational domain (not shown in Fig. 1). The flow is in OpenFOAM transferred

through those interfaces using an Arbitrary Mesh Interface (AMI), allowing a non-conformal connection between the meshes on both sides, and allowing a rotating mesh on one or both sides of the interface. The points on the interfaces are kept stationary or rotating as a solid body to avoid problems with the coupling. Hence, the mesh should neither deform nor slip on those boundaries, which puts constraints on the nearby mesh motion.

The only remaining boundary surfaces of the guide vane domain are the upper and lower ones, i.e., the red surfaces in Fig. 1. Since the guide vanes are rotating with respect to those surfaces, the mesh points need to slip on those surfaces according to the motion of the guide vanes, the internal morphing of the mesh, and the shape of the surfaces. As can be seen in Fig. 1, those surfaces are not entirely flat in the region where the points need to slip.

Listing 1: `evaluate()` member function of `basicSymmetryPointPatchField`.

```
template<class Type>
void Foam::basicSymmetryPointPatchField<Type>::evaluate
(
    const Pstream::commsTypes
)
{
    const vectorField& nHat = this->patch().pointNormals();

    tmp<Field<Type>> tvalues =
    (
        (
            this->patchInternalField()
          + transform(I - 2.0*sqr(nHat), this->patchInternalField())
        )/2.0
    );

    // Get internal field to insert values into
    Field<Type>& iF = const_cast<Field<Type>&>(this->primitiveField
      ());

    this->setInInternalField(iF, tvalues());
}
```

A seemingly obvious choice for slipping the points is to use the general `slip` boundary condition in OpenFOAM for the `pointDisplacement` or `pointMotionU` field, i.e. `slipPointPatchField`. This `slip` condition is however an explicit correction that does not influence the linear system while it is being solved. It is intended to make sure that the points eventually stay at the surface, although they may deviate from it during the solution of the linear system. After obtaining the solution of the motion field at a particular time step, and before moving the points, the `slip` condition removes the surface-normal component of the motion and only keeps the tangential components. For instance, for the motion velocity field, the slip condition reads

$$\mathbf{u}_{\text{point},\parallel} = \mathbf{u}_{\text{point}} - \mathbf{u}_{\text{point},\perp} = \mathbf{u}_{\text{point}} - (\mathbf{u}_{\text{point}} \cdot \hat{\mathbf{n}}) \cdot \hat{\mathbf{n}}. \tag{10}$$

Here, $\mathbf{u}_{\text{point},\parallel}$ is the surface-tangential component of the point velocity vector, and $\hat{n}$ is the unit normal vector of the boundary surface at each point. Accordingly, the normal component of the point velocity field ($\mathbf{u}_{\text{point},\perp} = (\mathbf{u}_{\text{point}} \cdot \hat{\mathbf{n}}) \cdot \hat{\mathbf{n}}$) is subtracted from the full point velocity vector to maintain only the tangential component ($\mathbf{u}_{\text{point},\parallel}$). The OpenFOAM implementation of the `slipPointPatchField` boundary condition is a simple wrapper around the `basicSymmetryPointPatchField`, for which the `evaluate` function is presented in Listing 1. The normal vectors of the patch surface at the points (`nHat`) are extracted through the `pointNormals()` member function and used to find the tangential component of the quantity of interest. It should particularly be noted that the `pointNormals()` member function uses the neighboring points at the patch to determine the local unit normal vector, which is prone to instabilities if the points are not exactly at the geometric surface, if the geometric surface has wiggles, or if some truncation errors accumulate. The accuracy of the function also reduces at the edges and in the corners.

Salehi and Nilsson [26] analyzed the effectiveness of the general `slip` condition for slipping points on a flat surface. It was shown that this type of explicit correction boundary condition, combined with the displacement type mesh motion solver, can be extremely sensitive and unstable and that it accumulates small errors and distortions because the normal vectors of the distorted points are called and used again. This results in the divergence of the mesh motion equations and the destruction of the mesh. Our experiences show that a combination of the slip condition with the velocity-type mesh motion solver is more stable, and allows a larger mesh deformation. Nevertheless, it is still not able to preserve the geometrical shape of the slip surfaces, especially in the presence of low-quality mesh faces on the surface. The lower and upper surfaces are distorted after a short while and the mesh is eventually destroyed.

The main reason for the instability of the general slip condition is that it is not constrained, and small distortions in the geometry grow fast and deform the surface. Different techniques were proposed and investigated to tackle this problem and to keep the points on the geometrical surface while slipping [26]. For instance, one could employ the `fixedNormalSlip` boundary condition, which calculates the tangential component using a prescribed normal direction and thus avoids introducing the small mesh distortions. However, this boundary condition is obviously only useful for flat surfaces. Another alternative that may also work on curved surfaces is the `surfaceSlipDisplacement` boundary condition. This boundary condition, which is only available for displacement-type solvers, projects the points onto a specified STL surface and thereby constrains the motion of the points to that STL surface. An alternative to the use of STL surfaces is to introduce the exact mathematical profiles of the geometry inside the dynamic mesh library and make the points follow that geometry. This was discussed in detail by Salehi and Nilsson [26].

Although the introduced alternatives to the `slip` boundary condition are more stable, they are still explicit corrections. These types of corrections may work fine for small mesh deformations or on surfaces with slight curvatures. However, they are not able to handle large mesh deformations on highly curved surfaces because the internal points do not adapt to the surface curvature while morphing. The internal points can even hit the patch surface and produce negative volume cells that lead to the termination of the CFD simulation.

4. Developed mesh motion framework

The developed numerical framework to tackle the problems described in the previous section is explained and tested for simple cases in the current section.

4.1. Semi-implicit slip algorithm

In Section 3.2, it was argued that the `slip` condition is not stable and cannot perfectly preserve the geometrical shape of the surface. On the other hand, the other more robust alternatives are mostly applicable to flat surfaces or surfaces with small curvature having a coarse mesh in the normal direction. As a practical example, in Kaplan turbines, the guide vane upper and lower surfaces, as well as the runner hub and shroud, are highly curved surfaces. The mesh is usually quite fine near these regions to accurately resolve the wall viscous effects. Therefore, one of the main purposes of the current study is to develop a robust algorithm for slipping the points on curved surfaces. The intention is to somehow *inform*

the internal points of the mesh about the curvature of the surface on which the mesh points slip and make them accommodate the curvature while perfectly preserving the geometrical shape of the surface.

The implicit Dirichlet boundary condition (`fixedValue` in OpenFOAM) can potentially be utilized to make internal points follow a specific geometry. Instead of explicit slip, one could implement an *ad-hoc* Dirichlet boundary condition and make the points follow a specific profile in time. However, this approach cannot have a general implementation as the geometrical configuration should be introduced inside the implementation. Thus, for each case, a new boundary condition should be developed. Additionally, the implementation could be very challenging and even impossible for complex geometries. An alternative is to first predict the new surface point positions, using some general method, and then to use those surface point positions as Dirichlet conditions when solving the Laplace equation. This is the fundamental concept of the proposed method that is described and tested in the following sections.

4.1.1. Algorithm details

The main idea in the current novel approach is to combine the explicit slip and implicit Dirichlet boundary conditions to introduce a general *semi-implicit* slip algorithm that works on any surface. In contrast to the classical mesh morphing approaches in OpenFOAM that solves one motion field, we will employ two different fields and solve them in two steps separately. The steps are explained as follows.

1. **Predictor step:** The first step, which we call the predictor, includes solving the mesh morphing equations (e.g., Laplace equation) on an *intermediate* motion field (could be displacement or velocity) with a specified diffusivity Γ_0. As an example, the Laplace equation for the intermediate motion velocity field can be written as

   $$\nabla \cdot (\Gamma_0 \nabla \mathbf{u}_0) = \mathbf{0}.$$

   The intermediate field ($\boldsymbol{\delta}_0$ or $\mathbf{u}_0$) exploits the explicit slip boundary condition to morph the points on the curved surfaces. The normal component of the intermediate field on the slip surfaces is removed and only the tangential component is kept. The intermediate motion field is only responsible for calculating the morphed points on the slip surfaces and will not be used to deform the mesh.
2. **Corrector step:** In the second step, called the corrector, the motion field of the slipped points on the curved surfaces is extracted and set as a Dirichlet (`fixedValue`) boundary condition for the main motion field ($\boldsymbol{\delta}$ or $\mathbf{u}$). Then, the set of mesh motion equations is solved for this field with diffusivity Γ (e.g., similar to Eq. (7)).
   The main motion field is in charge of morphing the entire mesh. Since a Dirichlet boundary condition is employed on the curved surface, the geometrical configuration of the surface affects the linear system. The points inside the domain *feel* the presence of the slip boundary surface and move accordingly to accommodate the geometry curvature.

The key point here is to solve the mesh motion equations twice in which the slip and Dirichlet conditions are employed, respectively. Imposing a Dirichlet type boundary condition with appropriate diffusivity for the main motion field allows proper mesh deformation that can follow the curvature of the slip surface and thus enables a larger and more robust mesh deformation procedure. The intermediate and main motion fields (e.g., $\mathbf{u_0}$ and $\mathbf{u}$) could employ two different mesh motion diffusivity fields as well (Γ_0 and Γ). The reason is that commonly, the diffusivity fields are calculated using a function which is proportional to the inverse distance of the points to the moving boundaries (Dirichlet condition). In the developed algorithm, the intermediate and main motion fields employ the Dirichlet conditions differently and thereby different diffusivities should preferably be used. Algorithm 1 lists the steps of the developed framework.

Algorithm 1 Semi-implicit slip algorithm.

Predictor:
1: Calculate diffusivity Γ_0
2: Update the implicit boundary conditions of the intermediate motion field ($\boldsymbol{\delta}_0$ or $\mathbf{u}_0$)
3: Solve the mesh motion equations for the intermediate field
4: Correct the explicit slip boundary conditions of the intermediate field

Corrector:
5: Calculate diffusivity Γ
6: Extract values of $\boldsymbol{\delta}_0$ or $\mathbf{u}_0$ on the slip boundary and set it as Dirichlet condition for $\boldsymbol{\delta}$ or $\mathbf{u}$
7: Update the implicit boundary conditions of the main motion field ($\boldsymbol{\delta}$ or $\mathbf{u}$)
8: Solve the mesh motion equation for the main field

9: Replace the internal values of the intermediate motion field by the main field (e.g., $\mathbf{u_0} \leftarrow \mathbf{u}$)

Since the main motion field is responsible for appropriately moving the points, it could serve as a better initial guess for the intermediate displacement field at the next time step. Therefore, in the last step of the algorithm, the intermediate motion field is assigned values from the current main field.

4.1.2. OpenFOAM implementation

The framework described in the previous section is implemented in a dynamic plug-in library for OpenFOAM, as a new subclass to the `fvMotionSolver` base class. The new mesh motion solvers are called `semiImplicitSlipDisplacementLaplacian` and `semiImplicitSlipVelocityLaplacian`. The described algorithm is mostly carried out in its `solve()` member function. The implicit boundary conditions (e.g., Dirichlet condition) are updated (steps 2 and 9 in Algorithm 1) with the construction of each discretized Laplacian equation (constructor of `fvMatrix` class) through calling the `updateCoeffs()` member function of the boundary conditions. The explicit boundary condition is updated after obtaining the solution of the linear system by calling the `evaluate()` method of the boundary conditions.

In Step 7 of Algorithm 1, the updated explicit slip boundary condition of $\mathbf{u_0}$ is extracted and set as a fixed condition for $\mathbf{u}$. This step is conducted through a newly developed boundary condition for the point field of the patches (`pointPatchField`). It is of Dirichlet type and inherits from the `fixedValuePointPatchField`. Listing 2 presents the implementation of the Dirichlet boundary condition. The `lookupObject` function is employed to find the intermediate displacement field and then its corresponding values at the current patch are set as a `fixedValue` condition for the final displacement field. Obviously, the boundary condition is only applied on the final displacement field and the intermediate field utilizes the `slip` condition (or one of its more robust alternatives, such as `surfaceSlipDisplacement`).

4.1.3. Verification case studies

Two verification case studies are visited to assess the performance and capabilities of the developed semi-implicit slip algorithm. The first test case is a simple 2D case, in which the details of the mesh motion are better visible and understandable, whereas the second case is a practical study on one single passage of a Kaplan turbine guide vane. In the second test case, the results are also compared to some commercial CFD software.

Listing 2: `updateCoeffs()` member function of `DirichletBC`.

```
void DirichletBCPointPatchVectorField::updateCoeffs()
{
    if (this->updated())
    {
        return;
    }

    // Read the pointDisplacement0_ field
    const pointVectorField& pointDisplacement0_ =
        this->db().objectRegistry::
        lookupObject<pointVectorField> ("pointDisplacement0");

    vectorField displacement(this->patchInternalField());
    labelList patchPoints = patch().meshPoints();

    forAll(displacement, idx)//loop over all patch points
    {
        displacement[idx] = pointDisplacement0_[patchPoints[idx]];
    }

    vectorField::operator=
    (
        displacement
    );

    fixedValuePointPatchField<vector>::updateCoeffs();
}
```

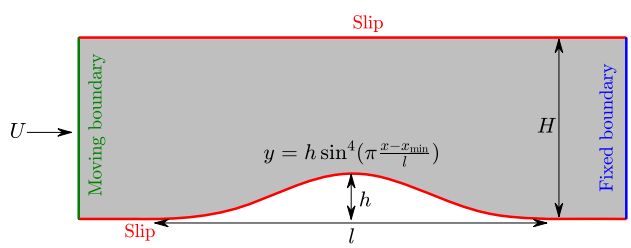


Fig. 2. Illustration of the designed simple test case for verification of the semi-implicit slip algorithm.

4.1.3.1. Two-dimensional bump The performance of the proposed semi-implicit slip algorithm is assessed using a two-dimensional mesh deformation test case. Fig. 2 illustrates the configuration of this verification case study, which is a simple structured hexahedral mesh on top of a bump with a smooth curved surface. The left (green) boundary is moving with a constant speed from left to right while the right (blue) boundary is fixed. The points are supposed to slip on the upper and lower boundaries (red). Although the designed configuration seems rather simple, it is quite challenging for a motion displacement solver due to the slipping points on the curved surface. This simple verification case study can be seen as a cheap test case that mimics the complex mesh deformation and slip procedure in Kaplan turbine transients. The curved lower surface may be considered as the hub or shroud of a Kaplan turbine, on which points should be able to slip smoothly in a transient sequence.

The total length of the channel is 1.2 m, whereas the bump length is $l = 1$ m. The channel and bump heights are $H = 0.4$ m and $h = 0.1$ m, respectively. The left boundary moves with a constant speed of $U = 0.1$ m/s. The `codedFixedValue` type boundary condition is used for the point displacement field of the left boundary to move the points with a constant horizontal velocity of $U = 0.1$ m/s while accommodating their vertical position to the changes in the boundary length. The right boundary points are fixed using a `fixedValue` condition. The top boundary is a flat surface. Consequently, the `fixedNormalSlip` boundary condition is a proper choice for this patch. However, the lower patch, where the points should be able to slip on the bump is the main challenge. Three different options are examined here, namely, slip (`slip`), slip on a prespecified surface (`surfaceSlipDisplacement`), and the developed semi-implicit slip. The first two options are only boundary conditions for the `pointDisplacement` field, whereas the semi-implicit slip approach is a combination of a new mesh motion solver and a new boundary condition, described in the previous section.

For the first two options (`slip` and `surfaceSlipDisplacement`), in which the classical Laplacian smoothing mesh deformation is employed, the diffusivity field (Γ) is calculated using the inverse distance to the moving boundary (left). However, the semi-implicit slip algorithm utilized two different diffusivity fields. Γ_0 is obtained using the inverse distance method with respect to the left boundary, whilst Γ is calculated using the inverse distance to the bottom boundary.

Fig. 3 displays the mesh motion solution for the employed options at different times (every 0.65 s). As explained in Section 3.2, the slip condition (`slip`) combined with the displacement-based solver is very sensitive to small errors, and tiny distortions in the mesh grow drastically which leads to early destruction of the mesh. The lower boundary wrinkles soon after the start of the simulation and the mesh is destroyed with negative volumes.

The most appropriate originally available option in OpenFOAM seems to be the slip-on-surface boundary condition (`surfaceSlipDisplacement`). The points are always projected to the specified surface. Thus, they will always remain on the surface during the morphing process and the surface geometry is perfectly preserved. However, as previously described, this is an explicit correction and does not affect the linear system of the mesh motion equations. The solution shown in Fig. 3 exhibits the main problem with the explicit correction. Although the points remain on the lower surface, the internal points do not follow its curvature and hit the surface which destroys the mesh with negative volumes as a consequence.

In contrast, the developed semi-implicit slip algorithm provides a very stable and smooth mesh deformation. Not only do the mesh points stay on the lower curved surface, but the internal points feel the presence of the bump and follow its curvature. The points on the left-hand side of the bump move upwards, while those on the right-hand side move downwards.

The performance of the developed mesh motion algorithm is further assessed using some mesh quality measures, namely, the maximum aspect ratio of the cells and the maximum non-orthogonal angle. The non-orthogonal angle is defined as the angle between a vector connecting two neighboring cell centers and the face normal vector. For a valid mesh, the angle should be less than 90°, and higher values indicate the existence of inverted cells (with negative volume). Fig. 4 illustrates the variation of maximum aspect ratio and maximum non-orthogonal angle during the mesh motion sequence of the bump case for all investigated approaches. The vertical dotted lines denote the creation of the first negative volume cell of the approach shown with the same color. None of the first two approaches can reach $t = 3$ s with a valid CFD mesh. On the other hand, the developed methodology does not produce any inverted cells up to $t = 6.7$ s.

Fig. 4 shows that the slip and slip-on-surface boundary conditions give poor quality measures shortly after the start of the simulation. The semi-implicit slip algorithm accomplishes a remarkably smooth mesh morphing in which the studied mesh quality measures only slightly increase during the simulation. The maximum aspect ratio and the non-orthogonal angle at $t = 6.7$ s are 3.3 and 36.3°, respectively. These quality measures are noticeably larger for the first two studied cases, even before the generation of inverted cells.

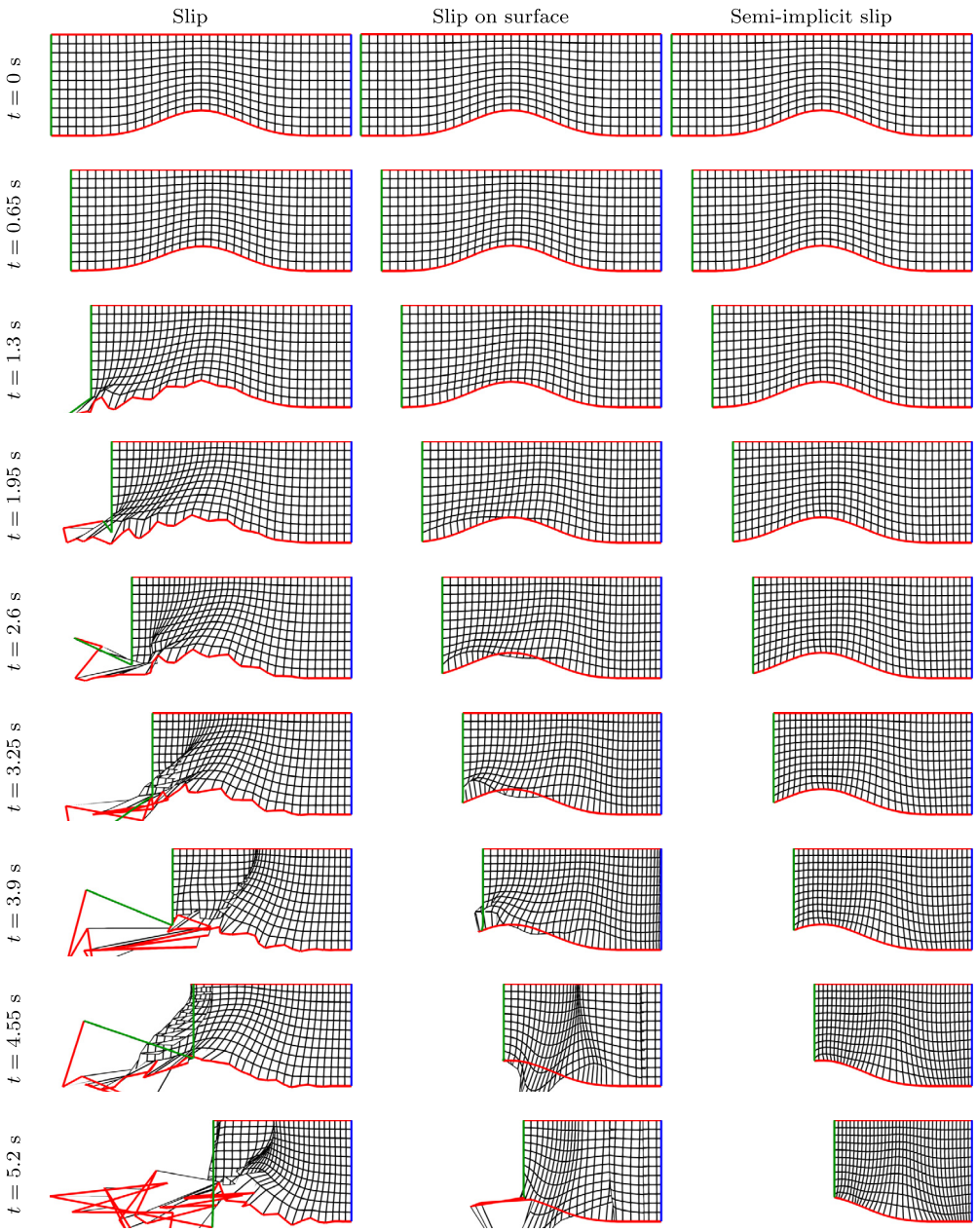
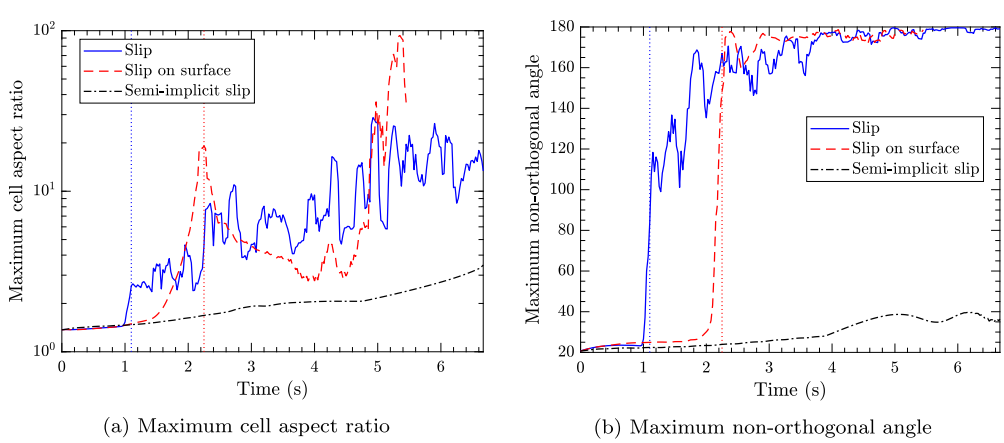


Fig. 3. Performance of the slip, slip on a prespecified surface, and the developed semi-implicit slip on the bump verification case study.



(a) Maximum cell aspect ratio (b) Maximum non-orthogonal angle

Fig. 4. Evolution of the mesh quality measures during mesh deformation for different approaches. The vertical dotted lines indicate appearance of the first negative-volume cell. Colors of the dotted lines are similar to the legends.

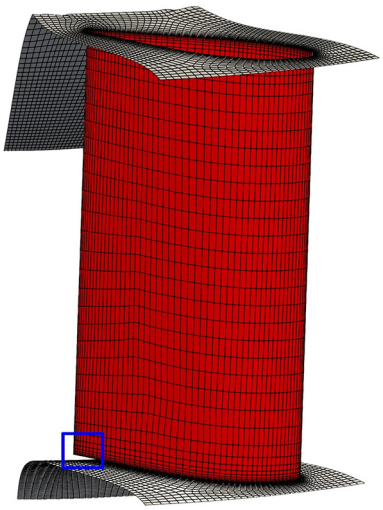


Fig. 5. Unmorphed mesh of the guide vane passage at Best Efficiency Point (BEP) condition with guide vane angle of $\alpha_{GV} = 26°$. The blue square highlights the trailing edge clearance as the guide vane hangs over the edge of the curved lower surface.

4.1.3.2. Kaplan turbine guide vane The performance of the developed methodology is further assessed using one single guide vane passage of a Kaplan turbine, in comparison with two of the widely used proprietary CFD software, i.e., Ansys Fluent and Ansys CFX. The unmorphed original mesh of the current case study is displayed in Fig. 5. As seen, a small clearance exists near the lower surface of the guide vane. Both the lower and upper surfaces on which the points should be able to slip are highly curved. These geometrical details make the mesh deformation procedure immensely challenging.

Four different approaches are employed to solve this mesh deformation problem. The developed methodology in OpenFOAM utilizing the motion velocity solver (i.e., `semiImplicitSlipVelocityLaplacian`) is adopted as the first approach and is compared to three other strategies, namely, OpenFOAM standard (built-in) motion velocity (i.e., `velocityLaplacian`), Ansys Fluent, and Ansys CFX. The Laplacian smoothing equations are solved in all the studied approaches.

The diffusion method is employed in Fluent, which solves the Laplacian equations for the velocity vector of the points (similar to Eq. (7)). On the other hand, CFX solves the same set of equations for the displacement field. However, it is possible to choose whether this displacement should be calculated with respect to the initial mesh or the previous time step mesh. In the current study, the mesh displacement is solved with respect to the previous mesh which resembles the motion velocity solver. The guide vanes are rotated with a constant angular speed of 2°/s around their axis. The same time step size is used for all investigated approaches.

The non-orthogonality of the cells is utilized to study the performance of the four employed strategies in the mesh deformation of the guide vane passage. Fig. 6 plots the number of severely non-orthogonal cells (cells with the non-orthogonal angle of more than 70°) with guide vanes rotation. Although, as expected, this number increases with guide vane rotation for all approaches, the growth rate is remarkably lower for the developed methodology. Obviously, the introduced method can largely deform the guide vane mesh without significantly degrading the mesh quality.

The vertical dotted lines indicate the destruction of the mesh with the creation of the first negative volume (inverted) cell of the method with the same color as in the legend. The OpenFOAM built-in solver can only deform the mesh for 0.72° of guide vane rotation. Ansys Fluent and CFX can morph the mesh up to 6.15°

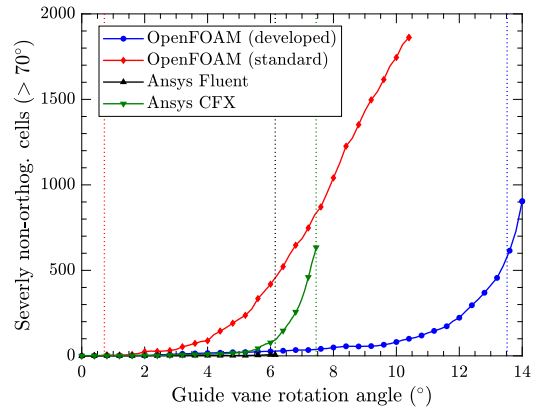


Fig. 6. Number of severely non-orthogonal cells (non-orthogonal angle > 70°) during guide vane rotation for the different approaches. The vertical dotted lines indicate the appearance of the first negative-volume cell. Colors of the dotted lines are similar to those in the legend.

and 7.45°, respectively. The CFX mesh quality rapidly degrades after 6° and the number of non-orthogonal cells increases. The presently developed methodology is able to smoothly deform the mesh up to 13.5° without producing any inverted cells.

The unmorphed (initial) and deformed mesh at the lower surface in the guide vane clearance after 6° of rotation is exhibited in Fig. 7. The view corresponds to the blue box shown in Fig. 5. The standard OpenFOAM, Fluent, and CFX approaches fail to smoothly deform the mesh and slip the points on the curved lower surface whilst keeping the surface geometry intact. Although the Fluent mesh quality measures are acceptable after 6° of rotation, the lower surface geometry is not preserved and the surface is noticeably distorted. In contrast, CFX maintained the geometrical shape of the lower surface (even though it may look distorted in the figure) at the expense of degrading mesh quality. Hence, the present developed methodology is the only alternative that can robustly deform the mesh, slip the points on the lower surface, and preserve the geometrical shape.

4.2. Simultaneous solid body rotation and mesh deformation

Many CFD applications require more complex dynamic mesh processes such as multiple types of movements. For instance, the CFD model can involve concurrent solid-body motion and mesh deformation. A practical example is Kaplan turbines working in a transient sequence which involves a simultaneous variation of both guide vane and runner blade angles at the same time as the runner is rotating as a solid body around the turbine axis. Accordingly, the mesh motion of the runner domain includes simultaneous axial solid-body rotation and mesh deformation.

In the developed methodology, the point motion field is always calculated with respect to the initial position of the mesh points (i.e., the points inside the OpenFOAM `constant` directory). Therefore, in order to combine a general solid-body motion with the developed mesh deformation algorithm, the point motion field also needs to be mapped with the same solid-body transformation. For instance, if the solid body rotation is to be combined with the mesh deformation, not only the mesh points should be rotated in each time step, but the point motion field (i.e., the deformation field) should also be rotated.

A new mesh motion solver is developed in OpenFOAM to address this issue. In each time step, the solver first morphs the initial mesh by obtaining the point motion field (either through displacement or velocity solvers) and adding it to the locations of the initial points. Subsequently, a solid-body transformation (in the Kaplan turbine case, a solid-body rotation around the turbine axis)

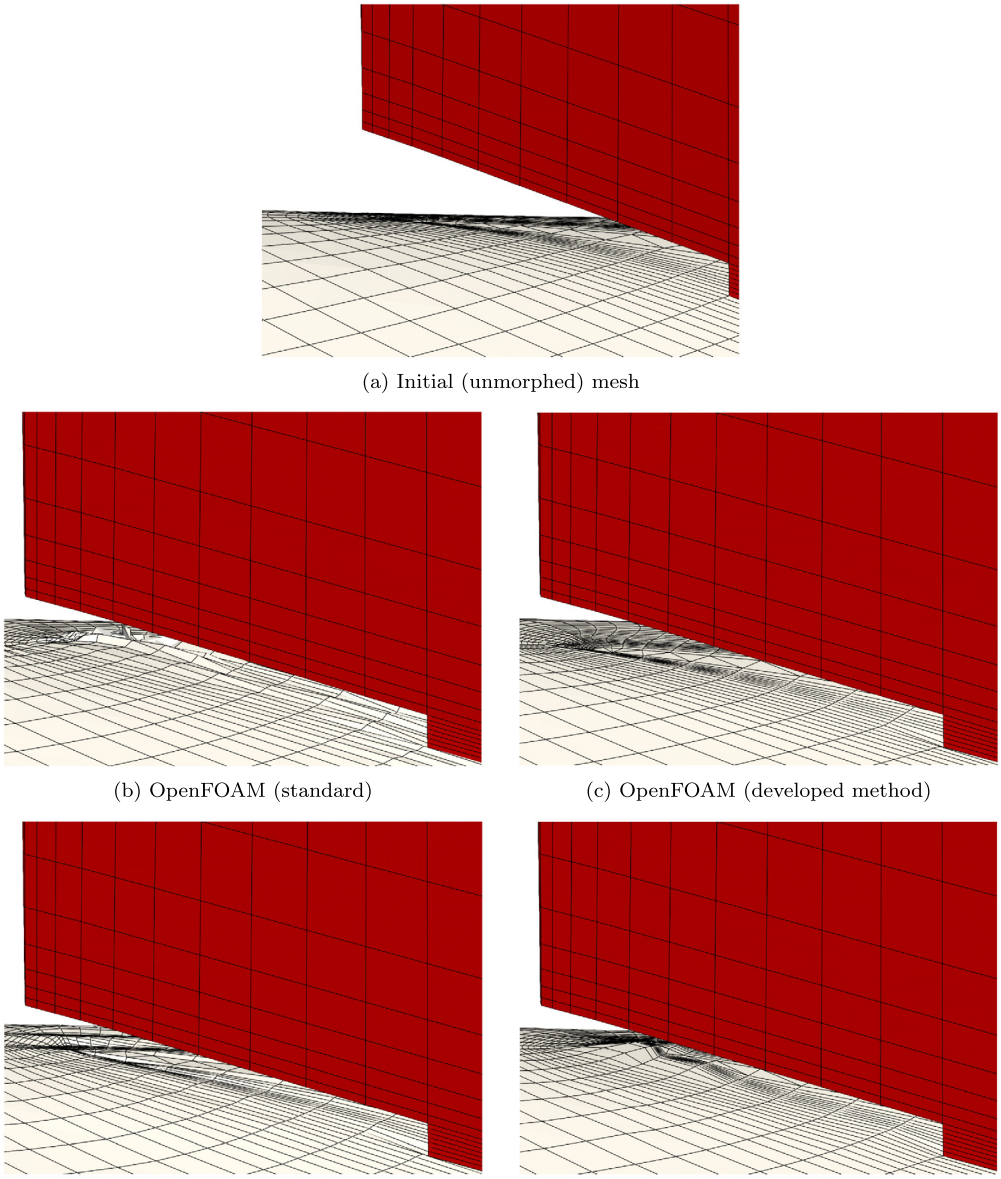


Fig. 7. Zoomed view of the lower surface in the guide vane clearance (the blue box shown in Fig. 5) of the unmorphed and morphed meshes after 6° of rotation ($\alpha_{GV} = 20°$) for different approaches. The red and gray surfaces represent the guide vane blade and lower surface, respectively.

is performed on the morphed mesh to obtain the final location of the points.

4.3. Final correction of the point locations

Although the explained framework works well for the mesh motion of complex geometries (such as Kaplan turbines), our experiences reveal that adding an extra correction step to the final calculated location of the points in each time step increases the stability of the dynamic mesh procedure. A more stable mesh motion provides the capability to deform the mesh to a larger extent without any mesh quality problems.

The final correction step makes sure that all the slipped points on the curved surfaces (which in the case of the Kaplan turbines would be the runner hub and shroud, and the guide vane lower and upper surfaces) follow the exact geometrical profile of those 3D surfaces. This correction is performed as the final step before updating the mesh at each time step. To implement such a correction, one can use the mathematical equations of the geometry inside the dynamic mesh class. Alternatively, predefined STL surfaces could also be utilized to project the points onto the desired surface to avoid hard-coding ad-hoc geometrical details. In this study, the former approach is employed as the final correction step.

4.4. Flowchart

Fig. 8 summarizes the complete implementation of the developed mesh motion algorithm inside OpenFOAM. The framework is employed inside the PIMPLE pressure correction algorithm. The mesh motion calculations are performed at the beginning of the first outer correction of the PIMPLE loop.

5. Kaplan turbine case study

In this section, the developed mesh motion framework is adopted to simulate a load change operation in a real Kaplan

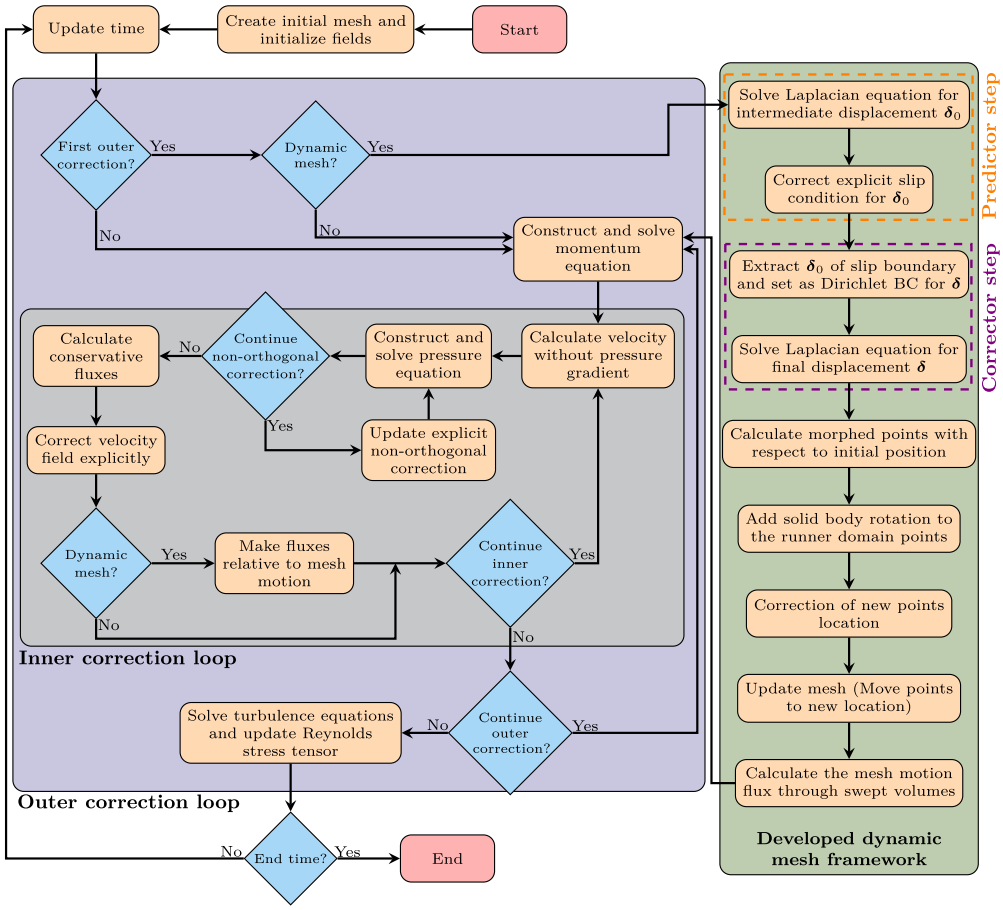


Fig. 8. Flowchart of the developed mesh motion framework inside the OpenFOAM PIMPLE algorithm. The mesh motion computations are performed in the first outer correction loop of each time step.

turbine model. One should note that the main intention of the current section is to investigate the capability and accuracy of the developed numerical framework for CFD simulation of transient operations of complex models such as Kaplan turbines. Therefore, a detailed analysis of the transient flow field will not be presented here and the chapter is mainly focused on the performance of the numerical framework.

5.1. The U9 model turbine

A Kaplan turbine model known as U9 was chosen as the investigated test case in the present study. The U9 model turbine is a new 1:3.875 scaled-down model of a prototype Kaplan turbine located in Porjus, Sweden. The previous U9 model, studied experimentally and numerically by different researchers (e.g. [39,40]), was a 1:3.1 scale model of the Porjus Kaplan turbine. The new model scale is under construction and no experimental data are yet available.

Fig. 9 displays a zoomed section view of the U9 model turbine assembly. The full computational domain consists of five regions, namely, spiral casing, stay vanes, guide vanes, runner, and draft tube. The model is assembled with 18 fixed stay vanes and 20 adjustable angle guide vanes. The diameter of the runner is $D = 0.4$ m and has 6 adjustable blades. At the Best Efficiency Point (BEP), the guide vanes and runner blades are at the opening of 26° and 32.8°. The angles are measured with respect to the fully closed position (blades touching each other). Each region is meshed separately and then merged together to create the full computational mesh with 12.7×10^6 cells.

5.2. Steady and transient operation

The new U9 model with a diameter of $D = 0.4$ m has a head of $H = 6.97$ m, a flow rate of $Q = 0.426$ m^3/s, and a runner rotational speed of $\omega = 839$ rpm at the BEP operating condition. Several of our previous studies on the same Kaplan turbine case study show that OpenFOAM produces comparable results to the experimental measurements at the steady operating condition [34,40–42], so we do not present the experimental validation at steady condition here.

The experimental research on the transient operation of the new U9 model is still ongoing. The exact transient sequences in the experiments are yet to be defined. Therefore, in the current paper, a transient sequence corresponding to the turbine load rejection is chosen for study purposes. It is assumed that the guide vane and runner blade angles decrease by six and five degrees, respectively, initiating from the BEP operating condition. The guide vanes and runner blades start to rotate around their own individual axes, and thus the flow rate reduces. It is assumed that the guide vane and runner blade angles change linearly in time from $\alpha_{GV} = 26.0°$ and $\alpha_{RB} = 32.8°$ at BEP to $\alpha_{GV} = 20.0°$ and $\alpha_{RB} = 27.8°$ at part load condition. The runner rotational speed is constant at $\omega = 839$ rpm during the whole sequence. The flow rate decreases throughout the transient sequence due to the closure of the guide vane and runner blades.

5.3. Computational details

This section briefly describes the computational framework and numerical aspects of the investigated case study. More information

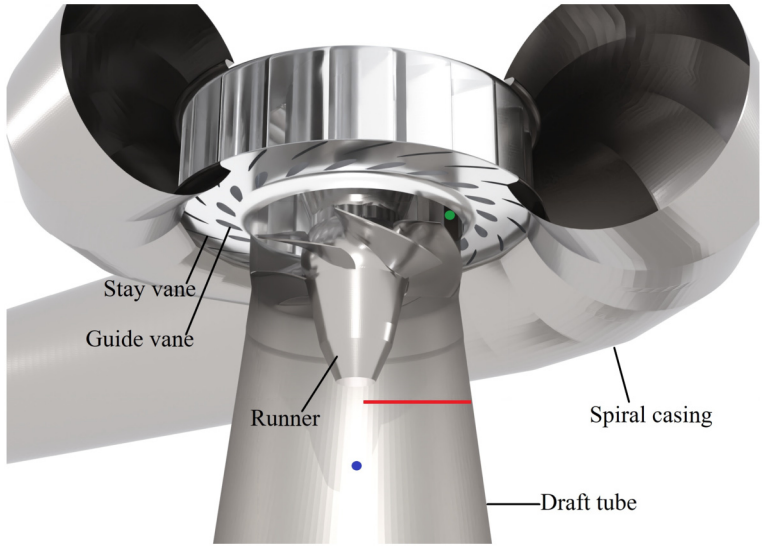
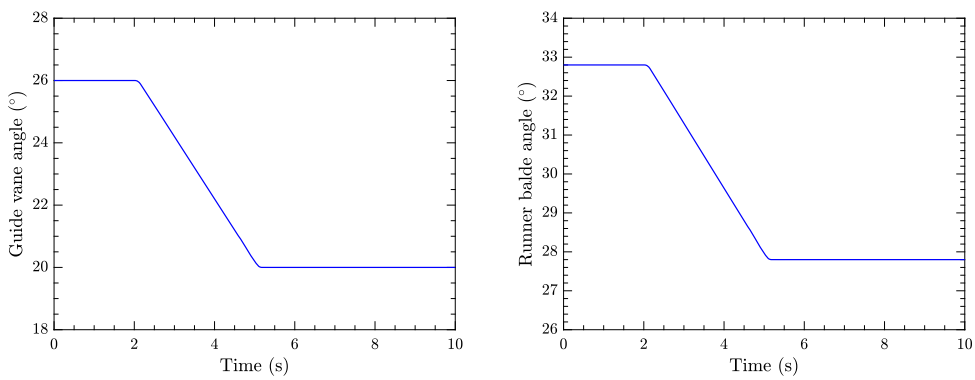


Fig. 9. A zoomed view of the U9 Kaplan turbine computational domain at best efficiency point. The red line, as well as the green and blue points are used for monitoring results.



(a) Guide vane angle (b) Runner blade angle

Fig. 10. Variation of guide vane and runner blade angles during the load rejection operation.

is provided by Salehi et al. [29] as similar numerical schemes were employed.

The Shear Stress Transport based Scale-Adaptive Simulation model (SST-SAS) [43] is used for turbulence modeling.

All the temporal derivatives were discretized using the implicit second-order backward scheme [44]. The transient flow is simulated using a fixed time step of $\Delta t = 5 \times 10^{-5}$ s, which corresponds to 0.25° runner rotation. The convective terms are discretized using the Linear-Upwind Stabilized Transport (LUST) scheme [45], blending 75% of the second-order linear scheme with 25% of the second-order upwind scheme, to enhance the stability of the simulations.

The pressure is coupled to the velocity field through the PIMPLE pressure correction algorithm with a maximum of 10 outer and two inner correction loops. To fully converge the explicit term in the non-orthogonal correction, two additional non-orthogonal correctors are carried out in each inner correction loop (also see Fig. 8).

A fixed total pressure boundary condition is used for the inlet of the spiral casing. The corresponding value is calculated through some preliminary stationary simulations at BEP to ensure the correct flow rate at the design condition. During the transient load rejection procedure, the inlet total pressure remains constant and the flow rate reduces adjusting the increasing pressure drop of the closing guide vanes and runner blades. The outlet boundary condition is set to constant static pressure.

The motions of the guide vanes and runner blades are imposed through a rotating boundary condition. It is assumed that the guide vanes and runner blades close down 6° and 5° degrees with a constant rotational speed. Fig. 10 presents the variation of the angle of both guide vanes and runner blades during the load rejection operation.

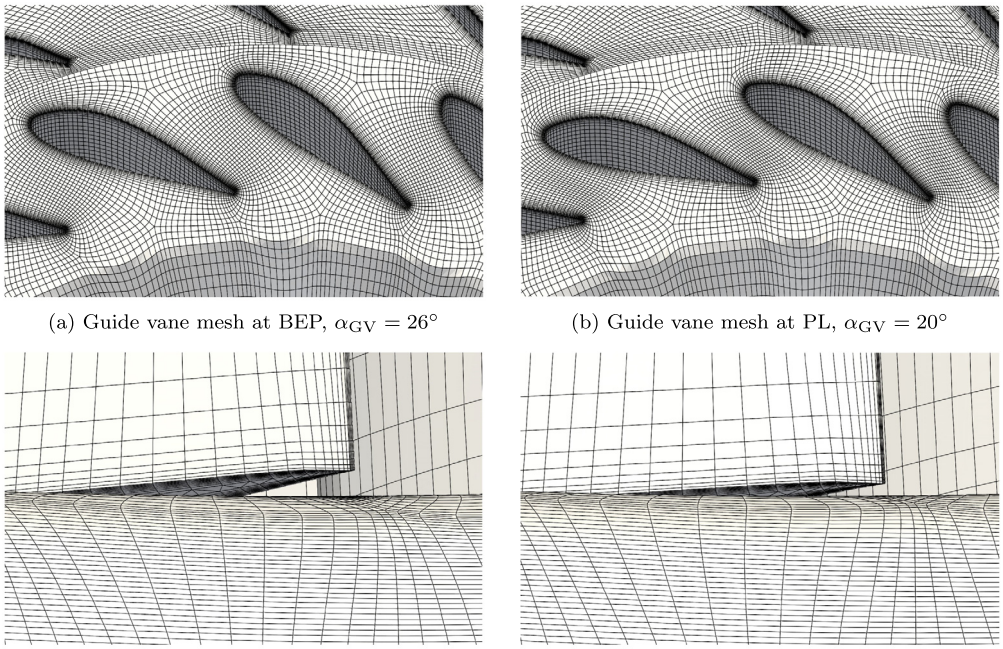
The `cyclicAMI` boundary condition is employed to transfer information on the non-conformal mesh interfaces between the different domains.

The typical average y^+ values in the domain are less than 84.7, and thus wall functions are employed to calculate the turbulence quantities at walls.

5.4. Load rejection sequence

As shown in Fig. 10, the simulated transient sequence starts with 2 s stationary operation at the BEP condition ($0 \leq t \leq 2$ s), followed by a 3 s load rejection ($2 \leq t \leq 5$ s), reaching the PL condition where it stays at stationary operation for 5 s ($5 \leq t \leq 10$ s). To reach a statistically stationary condition, before the start of this transient sequence, the flow field is simulated for 4 s at the BEP condition, corresponding to 56 runner revolutions.

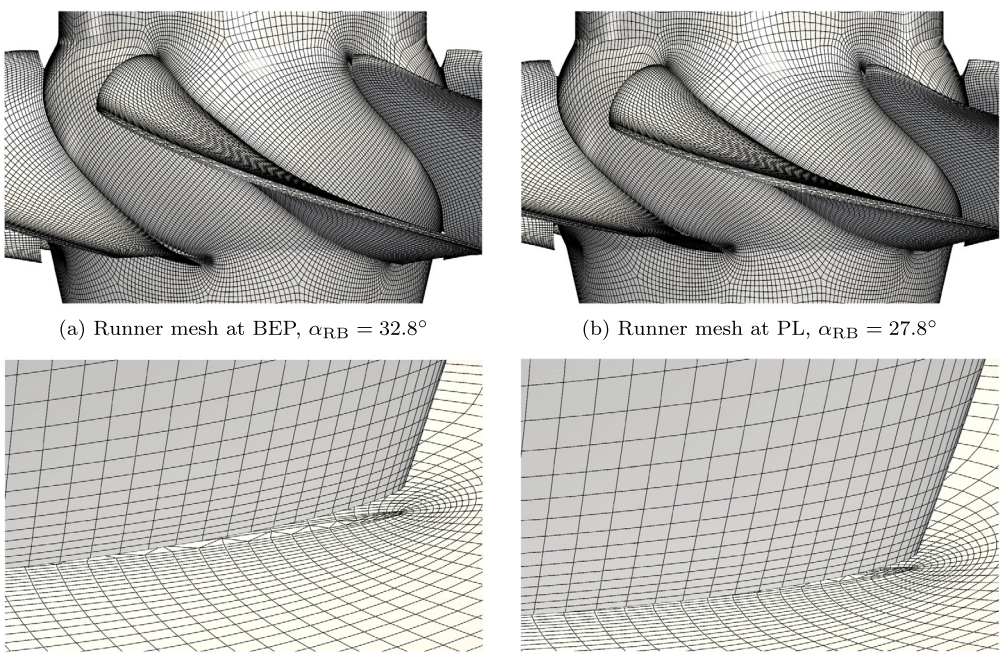
The mesh deformation of the guide vane and runner regions are displayed in Figs. 11 and 12, respectively. It is seen that the mesh morphing in both regions happens quite smoothly and the mesh quality is not severely affected. The guide vane and run-



(a) Guide vane mesh at BEP, $\alpha_{\mathrm{GV}} = 26°$ (b) Guide vane mesh at PL, $\alpha_{\mathrm{GV}} = 20°$

(c) Guide vane clearance at BEP, $\alpha_{\mathrm{GV}} = 26°$ (d) Guide vane clearance at PL, $\alpha_{\mathrm{GV}} = 20°$

Fig. 11. Initial (BEP) and morphed (PL) meshes of the U9 model guide vanes.



(a) Runner mesh at BEP, $\alpha_{\mathrm{RB}} = 32.8°$ (b) Runner mesh at PL, $\alpha_{\mathrm{RB}} = 27.8°$

(c) Runner tip clearance at BEP, $\alpha_{\mathrm{RB}} = 32.8°$ (d) Runner tip clearance at PL, $\alpha_{\mathrm{RB}} = 27.8°$

Fig. 12. Initial (BEP) and morphed (PL) meshes of the U9 model runner.

ner tip clearances are also shown in these two figures. Clearly, the clearances get smaller during the transient operation which makes the load rejection mesh motion more challenging. Fig. 13 assesses the non-orthogonality of both the runner and guide vane meshes during the transient sequence. The average non-orthogonal angles of the runner and guide vane meshes increase by 1.35° and 2.90°, which indicates a smooth mesh deformation.

The closing of the guide vanes and runner blades increases the overall pressure loss of the turbine. Since a total pressure boundary condition is employed at the inlet, the flow rate is expected to decrease during the load rejection sequence. The variation of the flow rate throughout the whole sequence is illustrated in Fig. 14. The guide vane angle is also shown in this figure for better understanding. The flow rate starts to decrease as the closing of the guide vanes and runner blades initiate. The flow rate reduction is linear, which can be explained through the fact that both the guide vanes and runner blades close down linearly (see Fig. 10). The flow rate decreases from 0.426 m^3/s at BEP to 0.321 m^3/s at PL, reaching a PL condition of 75%.

The unsteady static pressure is monitored at two different probe locations, namely, between the guide vanes and the runner near the shroud (Probe 1) and at the center of the draft tube

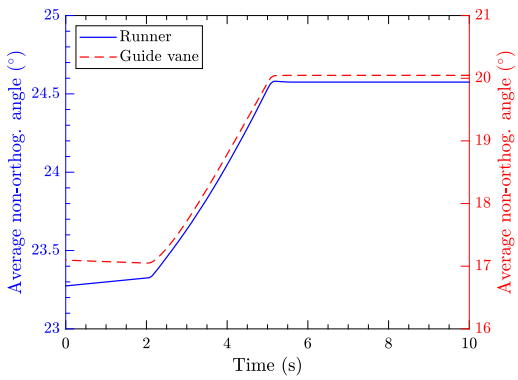


Fig. 13. Variation of average non-orthogonal angles during load rejection for runner and guide vane regions.

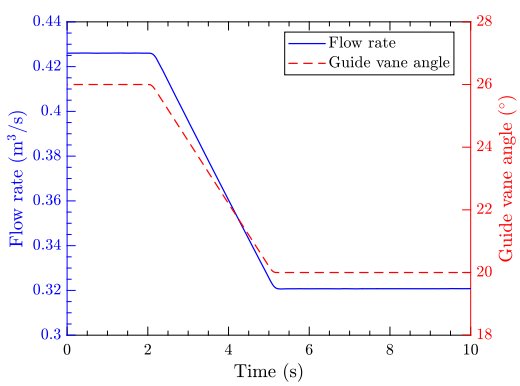


Fig. 14. Variation of turbine flow rate during load rejection sequence.

downstream of the runner (Probe 2). The probes are shown in Fig. 9 with green and blue colors, respectively.

In transient sequences, like turbine shutdown or startup, the pressure signals consist of both a variation due to the change of operation and fluctuations due to rotor-stator interaction and flow instabilities. Therefore, one must employ a signal analysis method to obtain the instantaneous mean and the fluctuations with respect to that instantaneous mean. In the present study, the Savitzky-Golay finite impulse response filter [46] is utilized to smooth out the captured pressure signals and acquire the pressure *instantaneous mean*.

The pressure data for both probes are depicted in Fig. 15. The instantaneous mean indicates the pressure variation due to the change in the operating condition. Distinct pulsations due to Rotor-Stator Interaction (RSI) are visible in Probe 1, which is placed right upstream of the runner. Therefore, as expected, the pressure mainly oscillates with the runner blade passing frequency at this probe location. However, the draft tube pressure shows low-frequency pulsations, due to complex flow structures in the draft tube, during the part-load condition. In-depth time-dependent frequency analysis can reveal important information such as variation of the amplitude of dominant frequencies, which is not the scope of the present study and can be proposed as future work.

6. Conclusion

Many practical applications of computational fluid dynamics require elaborate mesh motions that include points slipping on highly curved surfaces. The explicit slip boundary condition was elaborated on in detail, and it was shown that the boundary condition is unstable and unable to preserve the geometrical shape of the target surface. As a remedy, a novel semi-implicit algorithm was developed and implemented in OpenFOAM to robustly spread the boundary deformation into the mesh while slipping the mesh points on highly curved surfaces. The algorithm solves the mesh motion equations in two steps. First, the Laplacian mesh motion equations are employed to morph the points explicitly on the slip surfaces. Then, the same set of equations is solved again using the motion field of the morphed points as a Dirichlet condition. A further explicit correction step is also used to make sure that points remain on the intended surface. The full framework also includes a solid-body motion that is applied on top of the mesh morphing process to mimic the dynamic mesh processes with the concurrent mesh deformation and solid-body movement.

The developed framework was first verified on two computationally cheap test cases, namely, a 2D bump case and one guide passage of a Kaplan turbine. The mesh motion of the guide vane passage was also investigated using two proprietary CFD codes and it was demonstrated that the developed framework significantly outperforms the tested CFD codes and can deform the mesh to a larger extent without any mesh problems.

Finally, a load rejection sequence of a Kaplan turbine model was successfully studied using the developed program. It was shown that the developed methodology can preserve the mesh quality throughout the complex mesh motion of Kaplan turbines during transient operation. Numerical results of the transient flow field were presented and briefly discussed.

CRediT authorship contribution statement

Saeed Salehi: Conceptualization, Investigation, Methodology, Software, Writing – review & editing. **Håkan Nilsson:** Conceptualization, Funding acquisition, Supervision, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

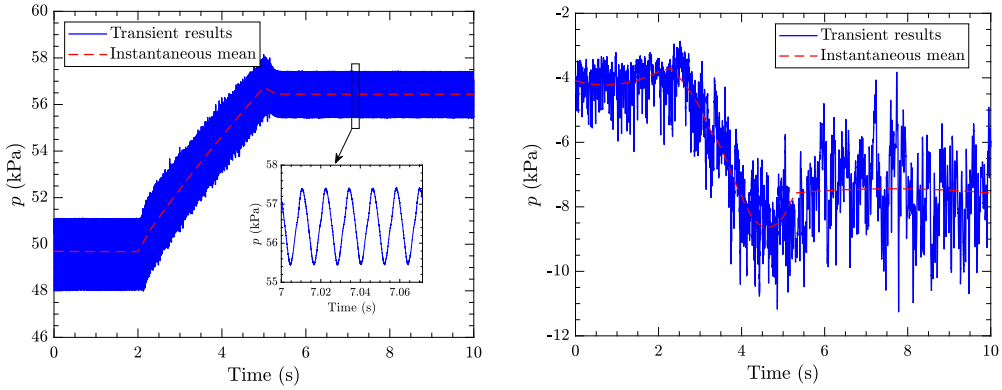
Acknowledgements

The current research was carried out as a part of the “Swedish Hydropower Centre - SVC”. SVC is established by the Swedish Energy Agency, EnergiForsk and Svenska Kraftnät together with Luleå University of Technology, The Royal Institute of Technology, Chalmers University of Technology and Uppsala University, www.svc.nu.

The computations were enabled by resources provided by the Swedish National Infrastructure for Computing (SNIC) at NSC, partially funded by the Swedish Research Council through grant agreement no. 2018-05973.

References

[1] J.H. Ferziger, M. Perić, R.L. Street, Computational Methods for Fluid Dynamics, Springer, 2020.
[2] J.L. Steger, F.C. Dougherty, J.A. Benek, in: K. Ghia, U. Ghia (Eds.), Advances in Grid Generation, ASME FED, vol. 5, 1983, pp. 59–69.
[3] W.M. Chan, Comput. Fluids 38 (3) (2009) 496–503, https://doi.org/10.1016/j.compfluid.2008.06.009.
[4] R. Mittal, G. Iaccarino, Annu. Rev. Fluid Mech. 37 (1) (2005) 239–261, https://doi.org/10.1146/annurev.fluid.37.061903.175743.
[5] S. Völkner, J. Brunswig, T. Rung, Comput. Fluids 148 (2017) 39–55, https://doi.org/10.1016/j.compfluid.2017.02.010.



(a) Between guide vanes and runner (b) Draft tube

Fig. 15. Variation of pressure at two probes during load rejection.

[6] L. Sun, P. Guo, J. Yan, Renew. Energy 171 (2021) 658–671, https://doi.org/10.1016/j.renene.2021.02.151.
[7] Z. Li, H. Bi, Z. Wang, Z. Yao, Proc. Inst. Mech. Eng. A, J. Power Energy 230 (6) (2016) 570–585, https://doi.org/10.1177/0957650916646911.
[8] Z. Li, H. Bi, B. Karney, Z. Wang, Z. Yao, J. Hydraul. Res. 55 (4) (2017) 520–537, https://doi.org/10.1080/00221686.2016.1276105.
[9] C.S. Peskin, J. Comput. Phys. 10 (2) (1972) 252–271, https://doi.org/10.1016/0021-9991(72)90065-4.
[10] D. Goldstein, R. Handler, L. Sirovich, J. Comput. Phys. 105 (2) (1993) 354–366, https://doi.org/10.1006/jcph.1993.1081.
[11] E. Fadlun, R. Verzicco, P. Orlandi, J. Mohd-Yusof, J. Comput. Phys. 161 (1) (2000) 35–60, https://doi.org/10.1006/jcph.2000.6484.
[12] J. Kim, D. Kim, H. Choi, J. Comput. Phys. 171 (1) (2001) 132–150, https://doi.org/10.1006/jcph.2001.6778.
[13] Y.-H. Tseng, J.H. Ferziger, J. Comput. Phys. 192 (2) (2003) 593–623, https://doi.org/10.1016/j.jcp.2003.07.024.
[14] M. Uhlmann, J. Comput. Phys. 209 (2) (2005) 448–476, https://doi.org/10.1016/j.jcp.2005.03.017.
[15] R. Mittal, H. Dong, M. Bozkurttas, F. Najjar, A. Vargas, A. von Loebbecke, J. Comput. Phys. 227 (10) (2008) 4825–4852, https://doi.org/10.1016/j.jcp.2008.01.028.
[16] A. Mark, B.G. van Wachem, J. Comput. Phys. 227 (13) (2008) 6660–6680, https://doi.org/10.1016/j.jcp.2008.03.031.
[17] S. Tenneti, R. Garg, S. Subramaniam, Int. J. Multiph. Flow 37 (9) (2011) 1072–1092, https://doi.org/10.1016/j.ijmultiphaseflow.2011.05.010.
[18] T. Kempe, J. Fröhlich, J. Comput. Phys. 231 (9) (2012) 3663–3684, https://doi.org/10.1016/j.jcp.2012.01.021.
[19] W.-P. Breugem, J. Comput. Phys. 231 (13) (2012) 4469–4498, https://doi.org/10.1016/j.jcp.2012.02.026.
[20] H. Yu, C. Pantano, J. Comput. Phys. 459 (2022) 111125, https://doi.org/10.1016/j.jcp.2022.111125.
[21] K. Kingora, H. Sadat-Hosseini, J. Comput. Phys. 453 (2022) 110933, https://doi.org/10.1016/j.jcp.2021.110933.
[22] A.E. Giannenas, S. Laizet, Appl. Math. Model. 99 (2021) 606–627, https://doi.org/10.1016/j.apm.2021.06.026.
[23] D. Li, X. Fu, Z. Zuo, H. Wang, Z. Li, S. Liu, X. Wei, Renew. Sustain. Energy Rev. 101 (2019) 26–46, https://doi.org/10.1016/j.rser.2018.10.023.
[24] C. Trivedi, J. Hydraul. Res. 58 (5) (2020) 790–806, https://doi.org/10.1080/00221686.2019.1671514.
[25] N. Sotoudeh, R. Maddahian, M.J. Cervantes, Renew. Energy 151 (2020) 238–254, https://doi.org/10.1016/j.renene.2019.11.014.
[26] S. Salehi, H. Nilsson, OpenFOAM J. 1 (2021) 47–61, https://doi.org/10.7910/DVN/31JGOM, https://journal.openfoam.com/index.php/ofj/article/view/26.
[27] Y. Dewan, C. Custer, A. Ivashchenko, J. Phys. Conf. Ser. 782 (2017) 012003, https://doi.org/10.1088/1742-6596/782/1/012003.
[28] P. Mössinger, R. Jester-Zürker, A. Jung, J. Phys. Conf. Ser. 782 (2017) 012001, https://doi.org/10.1088/1742-6596/782/1/012001.
[29] S. Salehi, H. Nilsson, E. Lillberg, N. Edh, Renew. Energy 179 (2021) 2322–2347, https://doi.org/10.1016/j.renene.2021.07.107.
[30] S. Salehi, H. Nilsson, E. Lillberg, N. Edh, IOP Conf. Ser. Earth Environ. Sci. 774 (1) (2021) 012060, https://doi.org/10.1088/1755-1315/774/1/012060.
[31] S. Salehi, H. Nilsson, Renew. Energy (2022), https://doi.org/10.1016/j.renene.2022.04.018.
[32] J. Unterluggauer, V. Sulzgruber, E. Doujak, C. Bauer, Renew. Energy 157 (2020) 1212–1221, https://doi.org/10.1016/j.renene.2020.04.156.
[33] S. Salehi, H. Nilsson, Renew. Energy 188 (2022) 1166–1183, https://doi.org/10.1016/j.renene.2022.01.111.
[34] S. Salehi, H. Nilsson, E. Lillberg, N. Edh, IOP Conf. Ser. Earth Environ. Sci. 774 (1) (2021) 012058, https://doi.org/10.1088/1755-1315/774/1/012058.
[35] I. Demirdžić, M. Perić, Int. J. Numer. Methods Fluids 8 (9) (1988) 1037–1050, https://doi.org/10.1002/fld.1650080906.
[36] H. Jasak, Z. Tukovic, Trans. FAMENA 30 (2) (2006) 1–20.
[37] H. Jasak, in: 47th AIAA Aerospace Sciences Meeting Including the New Horizons Forum and Aerospace Exposition, 2009, p. 341.
[38] F.M. Bos, B.W. van Oudheusden, H. Bijl, Comput. Fluids 79 (2013) 167–177, https://doi.org/10.1016/j.compfluid.2013.02.004.
[39] B. Mulu, An experimental and numerical investigation of a Kaplan turbine model, Ph.D. thesis, Luleå tekniska universitet, 2012.
[40] A. Javadi, H. Nilsson, Int. J. Heat Fluid Flow 63 (2017) 1–13, https://doi.org/10.1016/j.ijheatfluidflow.2016.11.010.
[41] O. Petit, B. Mulu, H. Nilsson, M. Cervantes, IOP Conf. Ser. Earth Environ. Sci. 12 (2010) 012024, https://doi.org/10.1088/1755-1315/12/1/012024.
[42] A. Javadi, H. Nilsson, IOP Conf. Ser. Earth Environ. Sci. 22 (2) (2014) 022001, https://doi.org/10.1088/1755-1315/22/2/022001.
[43] Y. Egorov, F. Menter, in: S.-H. Peng, W. Haase (Eds.), Advances in Hybrid RANS-LES Modelling, Springer Berlin Heidelberg, Berlin, Heidelberg, 2008, pp. 261–270.
[44] H. Jasak, Error analysis and estimation for the finite volume method with applications to fluid flows, Ph.D. thesis, Imperial College London, 1996.
[45] H. Weller, Mon. Weather Rev. 140 (10) (2012) 3220–3234, https://doi.org/10.1175/MWR-D-11-00221.1.
[46] A. Savitzky, M.J.E. Golay, Anal. Chem. 36 (8) (1964) 1627–1639, https://doi.org/10.1021/ac60214a047.