# Surface and Hypersurface Meshing Techniques for Space-Time Finite Element Methods

Jude T. Anderson\*

Department of Mechanical Engineering, The Pennsylvania State University, University Park, Pennsylvania 16802

David M. Williams

Department of Mechanical Engineering, The Pennsylvania State University, University Park, Pennsylvania 16802

Andrew Corrigan

Laboratories for Computational Physics and Fluid Dynamics, Naval Research Laboratory, Washington, DC 20375

#### Abstract

A general method is introduced for constructing two-dimensional (2D) surface meshes embedded in three-dimensional (3D) space time, and 3D hypersurface meshes embedded in four-dimensional (4D) space time. In particular, we begin by dividing the space-time domain into time slabs. Each time slab is equipped with an initial plane (hyperplane), in conjunction with an unstructured simplicial surface (hypersurface) mesh that covers the initial plane. We then obtain the vertices of the terminating plane (hyperplane) of the time slab from the vertices of the initial plane using a space-time trajectory-tracking approach. Next, these vertices are used to create an unstructured simplicial mesh on the terminating plane (hyperplane). Thereafter, the initial and terminating boundary vertices are stitched together to form simplicial meshes on the intermediate surfaces or *sides* of the time slab. After describing this new mesh-generation method in rigorous detail, we provide the results of multiple numerical experiments which demonstrate its validity and flexibility.

*Keywords:* Surface meshing, Hypersurface meshing, Space time, Four dimensional space, Finite element methods *2010 MSC:* 65M50, 52B11, 31B99, 76M10

Preprint submitted to Computer-Aided Design

<sup>\*</sup>Corresponding author

Email address: jta29@psu.edu (Jude T. Anderson )

<sup>&</sup>lt;sup>1</sup>Distribution Statement A: Approved for public release. Distribution is unlimited.

## 1. Introduction

Since its inception, the finite element method has often been limited to stationary three-dimensional (3D) geometries due to the available meshing capabilities. However, in the last two decades, research has been conducted with the goal of accurately simulating fluid-structure interactions (FSI) for 3D moving bodies. Towards this end, one may extrude or extend a 3D object along the temporal direction in order to capture its movement in a four-dimensional (4D) space-time setting. As one may imagine, this process is not intuitive, as the entirety of the domain is no longer directly visible and can only be observed through projections or hyperplane cross sections. Furthermore, extending the current technology to properly mesh these domains has proven to be a difficult task. The existing meshing technologies include methods for generating structured and semi-unstructured 4D meshes; however, the literature does not appear to contain a method for fully-unstructured mesh generation with boundary recovery in 4D. In what follows, we briefly review some of the relevant work on space-time volume meshing and classical surface meshing; then we provide an overview of our current efforts to extend this work to create fully-unstructured. 4D meshes.

Some of the earliest work related to space-time finite element methods in one spatial dimension plus time (1D+t) can be found in the papers of Hughes and Hulbert [1, 2]. Thereafter, Behr [3] developed a method for semi-unstructured temporal extrusion that applies to both two-dimensional (2D) and 3D meshes. Broadly speaking, Behr introduced a process for extruding the triangular elements of a 2D surface mesh along the temporal direction to create triangular prisms that can each be discretized into tetrahedra conforming to the Delaunay criterion. The process is similar for 3D hypersurface meshes, where a 3D hypersurface mesh of tetrahedra are extruded along the temporal direction to form 4D tetrahedral prism elements, which are subsequently discretized into pentatopes also conforming to the Delaunay criterion. This approach has been successfully applied to a wide range of applications, as evidenced by the work in [4, 5, 6, 7, 8].

The prism extrusion method of Behr was expanded upon by von Danwitz et al. [9]. In particular, they extend the method to accommodate time-variant topology through what they call a 4D-elastic mesh update method. Essentially, this does not change the connectivity of the 4D mesh but merely deforms the existing elements to conform to the varying surface topology. The most recent work on this particular topic appears to be that of Karyofylli and Behr [10].

In addition, a number of researchers have extended the extrusion-based method to accommodate rotational motions. For example, Wang and Persson [11] employ a similar method to Behr in 2D+t in order to generate an initial tetrahedral mesh; thereafter, they subdivide the mesh into a stationary region, a rotating region, and a buffer region that resides at the interface between the two. During rotation, the connectivity between the rotational region and the stationary region is maintained via reconnections (edge flips or face flips) in the buffer region. Wang and Persson's approach is essentially a

space-time, *sliding-mesh* approach. It has been extended to 3D+t for very simple cases [12]. Its viability appears to hold only for applications in which the boundary motion is purely rotational, and is known *a priori*. We note that a very similar sliding-mesh approach has been recently developed by Horváth and Rhebergen [13]. This work appears to extend, and in some ways improve upon the previous work of Wang and Persson.

In contrast to the extrusion-based methods (above), Foteinos and Chrisochoides [14] were able to generate unstructured 4D hypervolume meshes using a typical Delaunay-based mesh generator up-scaled to accommodate four dimensions. Although this work is significant, it does not present a clear mechanism for recovering the boundary, i.e. recovering the surface mesh that resides on the boundary. This issue of 'boundary recovery' is a common problem in Delaunaybased meshing techniques. Many researchers, such as Si et al. [15] and Liu et al. [16], detail various strategies for recovering a surface mesh in 3D, in conjunction with a Delaunay mesh generator. However, due to the lack of boundary recovery strategies in 4D, extrusion-based methods similar to Behr's remain dominant.

As another alternative to the methods proposed above, traditional advancing front techniques [17, 18, 19] were expanded upon to create a space-time mesh generation method that is known as *pitching tents* [20, 21, 22]. Here, each vertex in the spatial domain is projected along the temporal direction to generate a new vertex. New elements (one dimension higher than the original mesh) are created by tessellating the region formed by the neighboring faces of the original vertex and the new vertex. Recently, this method has been applied to hyperbolic systems [23, 24, 25] and the Maxwell equations [26]. These methods are usually best-suited for wave-propagation problems.

Most of the existing research (above) focuses on the generation of space-time volume meshes in 2D+t and hypervolume meshes in 3D+t. To our knowledge, researchers have not rigorously explored techniques for generating space-time surface meshes in 2D+t and hypersurface meshes in 3D+t. Part of the reason for this omission is that boundary conformity is automatically enforced for extrusion-based meshing approaches for problems where the boundary is stationary. Furthermore, boundary motion can (sometimes) be accommodated via the aforementioned elasticity-based approach. Of course, this comes at the cost of failing to accommodate arbitrary, large-scale boundary motions. More importantly, most volume or hypervolume meshes generated via extrusion are not fully unstructured in both space and time. Therefore, there is some incentive to investigate general surface meshing techniques which can accommodate fully-unstructured, constrained-Delaunay meshing strategies in 4D.

Naturally, there is already a wealth of literature which discusses surface meshing techniques for traditional, stationary, 3D applications. The majority of the work in this area relies on a method known as parametric mapping, which involves constructing a surface mesh for a 3D application on a reference 2D domain according to a specified metric. Once the 2D domain is meshed, it is mapped to the 3D domain. Interesting applications and discussions of this method can be found in [27, 28, 29, 30, 31, 32, 33]. Variations and improvements

to this method include separating a surface into patches [34], using high-order elements [35], and using Voronoi diagrams [36], among other notable works [37, 38, 39]. In addition, Lan and Lo [40] and Cass et al. [41] developed their own alternatives to parametric mapping that remain in 3D space and employ techniques such as curvature sizing functions to generate valid surface meshes.

The key issue with this existing surface meshing literature is that it is generally limited to 2D surface meshes which are embedded in 3D space. Furthermore, the meshing techniques often depend on a detailed knowledge of the underlying CAD, which is easily available for 3D problems, but may not be fully characterized for 4D space-time problems.

In this work, we discuss a new approach to 4D meshing, specifically the generation of a hypersurface mesh embedded in 3D+t space time. A key component of this method, is that vertices from the previous time slab change their positions in accordance with tracking space-time trajectories, (computed based on the local hypersurface velocity). In principle, the method can successfully track the movement of any 3D object in question. In addition, it allows us to create hyperplanes that are stitched together by tetrahedra in order to form a complete, conforming hypersurface mesh on each time slab. Once this hypersurface mesh is generated, we can create a fully-unstructured hypervolume mesh that conforms to the hypersurface mesh on the slab. Note that this latter step will be reserved for subsequent work. In what follows, we discuss preliminary concepts relating to surface meshing, then move on to 2D+t and 3D+t illustrations of our technique. We then present the results of some numerical experiments and conclude by suggesting future work.

## 2. Preliminaries

We begin by partitioning the space-time domain into slabs. This decomposition process is performed by intersecting the domain with spatial hyperplanes located at regular intervals, (see Figure 1). In 3D, each slab contains an "initial plane" (at  $t = t_n$ ), a "terminating plane" (at  $t = t_{n+1}$ ), and an "intermediate surface" which connects the initial plane to the terminating plane. We note that the intermediate surface does not need to be planar. When taken together, the initial plane, terminating plane, and intermediate surface form the boundaries of the space-time slab. These boundaries are 2D surfaces embedded in a 3D space time (2D+t). We note that the space-time slabs are generally not formed all at once. Instead, they are formed sequentially on an "as-needed basis", starting with those at the earliest times, and then continuing on with those at later times. This is a particularly important point, when we consider that the geometry of the space-time domain may not be known *a priori* for certain FSI applications, and a predictor-corrector approach may be necessary to form the topology of the individual space-time slabs, as the simulation evolves.

Before proceeding further, it is important for us to distinguish between "continuous space-time surfaces" and "discrete space-time surfaces". A continuous space-time surface is the continuous, CAD definition of a surface, which can only be generated if suitable knowledge of the boundary motion is available.



Figure 1: Entire space-time domain in 2D+t (left) and subdivision of this domain into space-time slabs (right).

A discrete space-time surface is the discrete, surface mesh that is (frequently) associated with an underlying continuous space-time surface. For the case of 2D+t, this surface mesh usually consists of triangles and their associated vertices embedded in 3D space time. For the case of 3D+t, the surface is actually a hypersurface which usually consists of tetrahedra and their associated vertices embedded in 4D space time. In this work, we are primarily interested in generating suitable surface meshes (i.e., discrete space-time surfaces). We can summarize our objectives in the following problem statement for 2D+t:

"Given a surface mesh on the previous space-time slab, find new surface meshes on the initial, intermediate, and terminating surfaces of the next space-time slab, while limiting the amount of space-time CAD information required."

We can obtain an equivalent statement for the case of 3D+t by replacing the word "surface" with the word "hypersurface" in the statement above.

## 3. Surface and Hypersurface Meshing

For the 2D case, we begin by extracting the terminating plane of the previous space-time slab, which is located at  $t = t_n$ . We assume that this terminating plane is covered with a discrete triangular surface mesh. Of course, if we consider the first space-time slab in our entire space-time domain, the previous spacetime slab does not exist. In this case, we simply assume that there is a ghost slab that spans the space from  $t = t_{-1}$  to  $t = t_0$  and provides us with a terminating surface mesh located at  $t = t_0$ . Once the terminating surface mesh is identified, our objective is to create new surface meshes on the initial, intermediate, and terminating surfaces of the next space-time slab from  $t = t_n$  to  $t = t_{n+1}$ . With this in mind, we start by setting the surface mesh on the initial plane of our new space-time slab to be identical to the terminating surface mesh of the previous space-time slab. This ensures that the subsequently generated volume mesh on the new space-time slab will maintain conformity with the volume mesh on the previous space-time slab. Next, it is possible to generate the intermediate surface mesh on the "sides" of the space-time slab, and thereafter, the surface mesh on the terminating plane. In what follows, we will describe the remainder of the surface meshing process in 2D. Thereafter, we discuss the extension to 3D.

#### 3.1 The Two-Dimensional Case (2D+t)

In order to begin building the intermediate 2D surface mesh, we extract and mark the edges which represent the discrete boundaries of the surface mesh on the initial plane. Next, we compute the space-time trajectories of these vertices using the local velocity of the space-time CAD surface (which should be known or predicted *a priori*), in conjunction with the well-known ordinary differential equation

$$\boldsymbol{v}\left(t\right) = \frac{d\boldsymbol{x}\left(t\right)}{dt}.$$

In order to solve this equation, the time interval  $dt = t_{n+1} - t_n$  is subdivided into M subintervals, where the time-step for each subinterval is simply dt/M. On each subinterval, we solve the differential equation above using the latest information about the surface velocity, in conjunction with a standard explicit time-stepping method, such as the forward Euler time-stepping method. In this way, the trajectories of the edge vertices are computed until their location at the final time  $(t_{n+1})$  is determined. The final location of the vertices may be impacted by time-integration errors, and therefore, we perform a simple projection procedure to ensure that the vertices lie exactly on the CAD surface at the final time. Note: throughout this process, we assume that the connectivity of the edge vertices does not change. After the vertex trajectories and final locations have been computed, we have two sets of vertices: one on the initial plane at  $t = t_n$ , and one on the terminating plane at  $t = t_{n+1}$ . The vertices on the terminating plane are then connected to one another in order to form edges. Thereafter, these edges are connected to the corresponding edges on the initial plane in order to form quadrilateral elements on the intermediate surface. These quadrilateral elements can be subdivided into triangular elements by inserting a Steiner point at the centroid of each quadrilateral element and connecting each Steiner point to the quadrilateral element's vertices. By following this procedure, we succeed in forming a linearly interpolated surface mesh of triangles on the intermediate surface. Lastly, we collect the edges and vertices on the terminating surface, and send them to a 2D constrained Delaunay mesh generation program (such as Shewchuk's Triangle program [42]) in order to generate a surface mesh for the terminating plane. We conclude by synchronizing the connectivity of the initial surface mesh, the intermediate surface mesh, and the terminating surface mesh. The resulting agglomeration of surface meshes provides a hull of triangular elements on the space-time slab from  $t = t_n$  to  $t = t_{n+1}$ .

The process for generating the surface mesh on a space-time slab is summarized below:

- 1. Extract the surface mesh from the terminating plane of the previous spacetime slab.
- 2. Construct the surface mesh for the initial plane of the new space-time slab using the surface mesh from step 1.
- 3. Extract the boundary edges and vertices of the surface mesh from step 2. Compute the vertex trajectories from  $t = t_n$  to  $t = t_{n+1}$ . Project final point locations to the CAD surface.
- 4. Connect vertices on the terminating plane to create edges.
- 5. Connect edges on the terminating plane to edges on the initial plane to create quadrilaterals.
- 6. Subdivide the quadrilaterals into triangles to generate a triangular surface mesh on the intermediate surface.
- 7. Use the edges on the terminating plane to generate a triangular surface mesh on the terminating plane.

Thereafter, the surface meshes from steps 2, 6, and 7 are combined to generate a surface mesh for the entire space-time slab. The overall process is shown in Figure 2.





Figure 2: An illustration of the process for generating a surface mesh on a space-time slab in 2D+t. The numbered steps are explained in the text.

#### 3.2 The Three-Dimensional Case (3D+t)

In order to build the 3D hypersurface mesh, we first extract the tetrahedral hypersurface mesh on the terminating hyperplane of the previous space-time slab. Following the approach used in the 2D case, we use this hypersurface mesh in order to tessellate the initial hyperplane of the next space-time slab. Thereafter, it remains for us to construct the intermediate hypersurface mesh, and the terminating hypersurface mesh for the space-time slab. Towards this end, we identify the outer boundaries of the initial hypersurface mesh. These boundaries correspond to the set of triangles which lie on the boundaries of the spatial domain at  $t = t_n$ . Once these triangles have been identified, we can extract their vertices, compute the corresponding space-time trajectories, and project the final point locations to the CAD surface (see the 2D procedure for details). Thereafter, we will have triangle vertices on the initial hyperplane (at  $t = t_n$ ) and on the terminating hyperplane (at  $t = t_{n+1}$ ). The vertices on the terminating hyperplane can be connected to form a triangulation, then the triangles on the initial and terminating hyperplanes can be connected in order to form triangular prisms. Note that these are 3D triangular prisms which are embedded in 4D space time. Once the prisms have been formed, they can be split into tetrahedral elements. We are aware of at least five different splitting strategies (see Figure 3). However, an arbitrary splitting is not possible, as it is important to preserve the conformity of adjacent triangular prism faces. Therefore, we require that all splittings of the quadrilateral faces of the triangular prisms are identical under reflections and rotations of the prism unto itself. With this in mind, we prefer two particular splitting techniques. The first technique involves splitting the quadrilateral faces of the triangular prisms into triangles by inserting Steiner points at the centroids of the quadrilateral faces and connecting these points to the quadrilateral's vertices. Thereafter, the triangular faces of the split prism can be connected to an additional Steiner point located at the prism's centroid. This results in a total of fourteen tetrahedral elements (see Figure 3, E). This splitting is natural because it is merely a generalization of the splitting employed for the 2D case. However, this splitting is actually slightly suboptimal. An improved splitting strategy involves the insertion of only three Steiner points (instead of four) and subdivides the triangular prism into ten tetrahedral elements (see Figure 3, C). This strategy is our foremost preference, as it produces a smaller number of elements relative to the first approach, while maintaining an identical pattern of splitting on the quadrilateral faces of the prism. Nevertheless, we make use of the more traditional splitting (the splitting into fourteen tetrahedra) in our subsequent numerical experiments due to its greater simplicity of implementation. The more optimal splitting (the splitting into ten tetrahedra) will be explored in future work.

For the sake of completeness, we introduce quantitative definitions for the aforementioned triangular prism splitting strategies on a reference triangular prism, denoted by  $R^*$ . We assume that  $R^*$  has the following vertices

$$R^* = [r_1(0,0,0), r_2(1,0,0), r_3(0,1,0), r_4(0,0,1), r_5(1,0,1), r_6(0,1,1)].$$

In addition, we introduce the following Steiner points at the centroid and on the quadrilateral faces of  $R^*$ ,

$$r_{7} = \frac{1}{4}(r_{2} + r_{3} + r_{5} + r_{6}), \quad r_{8} = \frac{1}{4}(r_{1} + r_{2} + r_{4} + r_{5}),$$
  
$$r_{9} = \frac{1}{4}(r_{1} + r_{3} + r_{4} + r_{6}), \quad r_{10} = \frac{1}{6}(r_{1} + r_{2} + r_{3} + r_{4} + r_{5} + r_{6})$$

Based on the description above, we can define the following ten tetrahedra as part of subdivision strategy  $\mathcal C$ 

$$\begin{aligned} \mathcal{C}R^* &= \left\{ S_1(r_1, r_2, r_3, r_7), \, S_2(r_4, r_5, r_6, r_7), \, S_3(r_1, r_2, r_7, r_8), \\ & S_4(r_2, r_5, r_7, r_8), \, S_5(r_4, r_5, r_7, r_8), \, S_6(r_1, r_4, r_7, r_8), \\ & S_7(r_1, r_3, r_7, r_9), \, S_8(r_3, r_6, r_7, r_9), \, S_9(r_4, r_6, r_7, r_9), \, S_{10}(r_1, r_4, r_7, r_9) \right\}, \end{aligned}$$

where  $\mathcal{C}$  yields the subdivision strategy illustrated in Figure 3, C. In addition,

we can define the following fourteen tetrahedra as part of subdivision strategy  ${\cal E}$ 

$$\begin{split} \mathcal{E}R^* &= \bigg\{ S_1(r_2,r_3,r_5,r_{10}), \, S_2(r_2,r_3,r_6,r_{10}), \, S_3(r_2,r_5,r_6,r_{10}), \\ &\quad S_4(r_3,r_5,r_6,r_{10}), \, S_5(r_1,r_2,r_4,r_{10}), \, S_6(r_1,r_2,r_5,r_{10}), \\ &\quad S_7(r_1,r_4,r_5,r_{10}), \, S_8(r_2,r_4,r_5,r_{10}), \, S_9(r_1,r_3,r_4,r_{10}), \\ &\quad S_{10}(r_1,r_3,r_6,r_{10}), \, S_{11}(r_1,r_4,r_6,r_{10}), \, S_{12}(r_3,r_4,r_6,r_{10}), \\ &\quad S_{13}(r_1,r_2,r_3,r_{10}), \, S_{14}(r_4,r_5,r_6,r_{10}) \bigg\}, \end{split}$$

where  $\mathcal{E}$  yields the subdivision strategy illustrated in Figure 3, E.

Once the triangular prisms have been successfully subdivided into tetrahedra, then we recover a valid tetrahedral hypersurface mesh for the intermediate hypersurface. Thereafter, it remains for us to construct the hypersurface mesh on the terminating hyperplane. Towards this end, we collect the triangular elements and vertices associated with the terminating hyperplane (at  $t = t_{n+1}$ ), then we send them off to a volume meshing program (such as Hang Si's TetGen program [43]). Once the terminating hypersurface mesh has been constructed, we synchronize its connectivity with the connectivity of the initial and intermediate hypersurface meshes. The resulting agglomeration of hypersurface meshes results in a hull of tetrahedral elements for the space-time slab from  $t = t_n$  to  $t = t_{n+1}$ .

The process for generating the hypersurface mesh on a space-time slab is summarized below:

- 1. Extract the hypersurface mesh from the terminating hyperplane of the previous space-time slab.
- 2. Construct the hypersurface mesh for the initial hyperplane of the new space-time slab using the hypersurface mesh from step 1.
- 3. Extract the boundary triangular faces, edges, and vertices of the hypersurface mesh from step 2. Compute the vertex trajectories from  $t = t_n$  to  $t = t_{n+1}$ . Project final point locations to the CAD surface.
- 4. Connect vertices on the terminating hyperplane to create triangular faces.
- 5. Connect triangles on the terminating hyperplane to triangles on the initial hyperplane to create triangular prisms.
- 6. Subdivide the triangular prisms into tetrahedra to generate a tetrahedral hypersurface mesh on the intermediate hypersurface.
- 7. Use the triangular faces on the terminating hyperplane to generate a tetrahedral hypersurface mesh on the terminating hyperplane.

The hypersurface meshes from steps 2, 6, and 7 are combined to generate a hypersurface mesh for the entire space-time slab. The overall process is shown in Figure 4.



Figure 3: Illustrations of the five different strategies for subdividing a triangular prism into tetrahedra elements. The strategies (A-E) subdivide the prism into 3, 6, 10, 12, and 14 tetrahedral elements, respectively.







2.





Figure 4: An illustration of the process for generating a hypersurface mesh on a space-time slab in 3D+t. The numbered steps are explained in the text.

## 4. Numerical Experiments

In this section, we present numerical experiments using the surface meshing algorithm from the previous section. This algorithm was implemented as an extension of the JENRE<sup>®</sup> Multiphysics Framework used in earlier work for space-time finite element methods [44].

### 4.1 Stationary Circle

The spatial geometry for this test case consisted of a circle with constant radius R = 1, located inside of a square with constant edge length L = 10. The circle was positioned at the centroid of the square and was kept stationary during the time interval  $t \in [0,1] = [t_0,t_f]$ . The combination of the spatial domain and the temporal interval formed a space-time geometry consisting of a space-time cylinder inside a 3-cube. The geometry for this configuration is illustrated in Figure 5. We note that this simple test case *can* be treated by



Figure 5: An illustration of a stationary 2-sphere inside of a 2-cube. Under these circumstances, the space-time geometry consists of a cylinder embedded inside of a 3-cube. Note: this drawing is *not* to scale.

conventional mesh-extrusion methods. For example, a triangular surface mesh located at time  $t_0 = 0$  can be extruded along the temporal direction to form a tetrahedral volume mesh that fills the space between the space-time cylinder and the surrounding cube. Furthermore, during this process, the boundary of the tetrahedral volume mesh automatically functions (or serves) as a triangular surface mesh which conforms to the space-time geometry. However, despite the simplicity of this test case and its ability to be successfully treated with other methods, it nonetheless serves as a useful 'sanity-check' in order to ensure that our surface meshing algorithm is working as expected. With this justification in mind, we proceeded by constructing a preliminary triangular surface mesh for the spatial geometry at  $t_0 = 0$ . Thereafter, we constructed a family of surface meshes for the entire space-time domain using the techniques from the previous section. For each of these surface meshes, the characteristic element size near the circle,  $h_{\text{circle}}$ , and the characteristic element size near the square boundary,  $h_{\text{square}}$ , were specified on the initial surface mesh at  $t_0 = 0$ . In addition, the mesh spacing of the domain along the temporal direction,  $h_{\text{time}}$ , was specified. Next, a total of nine surface meshes for the space-time slab were generated, each with a greater number of elements than the previous mesh in the sequence. Note that the surface meshes that appeared later in the sequence were larger than the earlier ones because they had progressively smaller values of  $h_{\text{circle}}$ ,  $h_{\text{square}}$ , and  $h_{\text{time}}$ . These spacing parameters were usually decreased by a factor of between 1.25 and 2.0 between successive meshes. The essential properties of the resulting surface meshes are summarized in Table 1.

Mesh	Elements	Vertices
1	2,714	$1,\!357$
2	$5,\!308$	$2,\!654$
3	10,022	5,011
4	20,142	10,071
5	38,792	19,396
6	$76,\!524$	38,262
7	$154{,}570$	$77,\!285$
8	$305,\!602$	$152,\!801$
9	613,418	306,709

Table 1: The number of triangular elements and vertices for a sequence of surface meshes for the stationary circle test case.

The validity of each surface mesh was assessed by comparing its approximate surface area to the exact, analytically-determined surface area of the spacetime geometry. The approximate surface area for each mesh was calculated by summing the areas of all triangles in each mesh. In particular, the area of each individual triangle was calculated using Heron's formula,

$$A_{\text{approx}} = \sum_{T_k} A_k,$$

where  $T_k$  is a generic triangle in a given surface mesh and

$$A_k = \sqrt{s_k(s_k - a_k)(s_k - b_k)(s_k - c_k)}, \qquad s_k = \frac{1}{2} (a_k + b_k + c_k),$$

where  $a_k$ ,  $b_k$ , and  $c_k$  are the edge lengths of the k-th triangle.

The exact surface area of the space-time geometry was calculated by the following formula

$$A_{\text{exact}} = 2\left(L^2 - \pi R^2\right) + \left(4L + 2\pi R\right)\left(t_f - t_0\right).$$

In a natural fashion, the error was calculated as follows

$$A_{\text{error}} = |A_{\text{exact}} - A_{\text{approx}}|.$$

Figure 6 shows a plot of the error in the surface area versus the number of elements to the -1/2 power. Here, we can see that the error decays at a rate of 2nd-order as the mesh resolution increases. This rate of convergence agrees well with our expectations, as straight-sided triangular elements are expected to generate 2nd-order convergence rates for most finite element applications.



Figure 6: Each point on the plot above represents the error between the area of a surface mesh and the exact surface area for the stationary circle test case. The errors are plotted against the characteristic mesh spacing for a sequence of increasingly refined surface meshes. In addition, a dashed line associated with 2nd-order convergence is plotted for reference.

## 4.2 Expanding Circle

In this test case, the circle from the previous case was allowed to expand. In particular, the radius of the circle was calculated based on the following function

$$R(t) = mt + R_0, (4.1)$$

where  $R_0$  is the initial radius of the circle, and m is the radial expansion speed of the circle. In this case, we elected to set  $R_0 = 1$  and m = 0.25, and we allowed the circle to expand during the time interval [0, 1]. The final radius of the circle was  $R_f = 1.25$ . The space-time geometry for this case is a conical frustum with initial radius  $R_0$  and final radius  $R_f$  inside of a 3-cube with edge length L = 10. In a natural fashion, the axis of revolution for the conical frustum is aligned with the temporal axis. Figure 7 shows an illustration of this geometric configuration. We created a family of nine surface meshes for the chosen space-time geometry, using the meshing parameters and techniques described in Section 4.1. The properties of the surface meshes for the expanding circle are summarized in Table 2. We assessed the validity of the surface meshes by calculating the area of each mesh, and comparing it with the following analytically-determined exact



Figure 7: An illustration of an expanding 2-sphere inside of a 2-cube. Under these circumstances, the space-time geometry consists of a conical frustum embedded inside of a 3-cube. Note: this drawing is *not* to scale.

Mesh	Elements	Vertices
1	2,696	1,348
2	5,288	$2,\!644$
3	10,016	5,008
4	20,086	10,043
5	38,734	19,367
6	76,406	38,203
7	$154,\!362$	$77,\!181$
8	$305,\!194$	$152,\!597$
9	$612,\!452$	306,226

Table 2: The number of triangular elements and vertices for a sequence of surface meshes for the expanding circle test case.

area for the space-time slab

$$A_{\text{exact}} = 2L^2 - \pi (R_f^2 + R_0^2) + 4L (t_f - t_0) + \pi (R_f + R_0) \sqrt{(R_f - R_0)^2 + (t_f - t_0)^2}.$$

Figure 8 shows a plot of the surface area error versus the approximate mesh spacing. As expected, the error decreases at a rate of 2nd-order with increasing mesh resolution.

## 4.3 Stationary Sphere

The geometry for this experiment consisted of a stationary 3-sphere with radius R = 1 inside of a 3-cube with edge length L = 10. The sphere was located at the centroid of the cube. In addition, the surface of the sphere was kept static



Figure 8: Each point on the plot above represents the error between the area of a surface mesh and the exact surface area for the expanding circle test case. The errors are plotted against the characteristic mesh spacing for a sequence of increasingly refined surface meshes. In addition, a dashed line associated with 2nd-order convergence is plotted for reference.

without any changes in size. The associated space-time geometry consists of a hypercylinder embedded inside of a tesseract (4-cube). This geometry is shown in Figure 9. The region between the surface of the sphere and the walls of the



Figure 9: An illustration of a stationary 3-sphere inside of a 3-cube. Under these circumstances, the space-time geometry consists of a hypercylinder embedded inside of a tesseract. Note: this drawing is *not* to scale.

cube was filled with an unstructured mesh of tetrahedral elements at time  $t_0 = 0$ .

This mesh served as a hypersurface mesh for the initial hyperplane. With this as a starting point, an entire family of hypersurface meshes was formed for the space-time slab using the construction techniques described in Section 3. In order to create a well-behaved family of meshes, we specified the mesh spacings on the surface of the sphere,  $h_{\rm sphere}$ , on the surface of the cube walls,  $h_{\rm cube}$ , and along the temporal direction,  $h_{\rm time}$ . The mesh properties are summarized in Table 3.

Mesh	Elements	Vertices
1	144,431	30,813
2	$368,\!587$	$78,\!637$
3	958,500	$204,\!440$
4	$2,\!630,\!598$	561,013
5	$7,\!127,\!597$	1,519,319
6	$19,\!165,\!615$	4,087,387
7	56,118,477	11,961,783
8	149,470,428	31,879,852

Table 3: The number of tetrahedral elements and vertices for a sequence of hypersurface meshes for the stationary sphere test case.

We compared the volume of each hypersurface mesh to the exact, analyticallydetermined volume of the hypersurface for the space-time slab. The volume of each hypersurface mesh was calculated by adding up the individual volumes of all tetrahedral elements in each mesh as follows

$$V_{\text{approx}} = \sum_{T_k} V_k,$$

where

$$V_k = \sqrt{\frac{\det(\Theta)}{288}}, \qquad \Theta = \begin{bmatrix} 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & d_{ab}^2 & d_{ac}^2 & d_{ae}^2 \\ 1 & d_{ab}^2 & 0 & d_{bc}^2 & d_{be}^2 \\ 1 & d_{ac}^2 & d_{bc}^2 & 0 & d_{ce}^2 \\ 1 & d_{ae}^2 & d_{be}^2 & d_{ce}^2 & 0 \end{bmatrix},$$

and where the "d" quantities above are the pairwise distances between the vertices a, b, c, and e of the k-th tetrahedron,  $T_k$ .

The analytically-determined, exact volume was computed as follows

$$V_{\text{exact}} = 2\left(L^3 - \frac{4}{3}\pi R^3\right) + \left(6L^2 + 4\pi R^2\right)(t_f - t_0).$$

The error in the hypersurface volume was then obtained as follows

$$V_{
m error} = |V_{
m exact} - V_{
m approx}|$$

Figure 10 shows the volumetric error for each hypersurface mesh plotted versus the approximate mesh spacing. Here, the mesh spacing was estimated by raising the total number of elements in each mesh to the -1/3 power. As expected, the error appears to consistently decrease with a rate of approximately 2nd order.



Figure 10: Each point on the plot above represents the error between the volume of a hypersurface mesh and the exact volume for the stationary sphere test case. The errors are plotted against the characteristic mesh spacing for a sequence of increasingly refined hypersurface meshes. In addition, a dashed line associated with 2nd-order convergence is plotted for reference.

#### 4.4 Expanding Sphere

For this experiment, we used the stationary sphere geometry from the previous section. However, in this case, the radius of the sphere was allowed to increase in time in accordance with Eq. (4.1), during the time interval [0, 1]. Here, we let  $R_0 = 1$  and m = 0.25. During the time interval in question, the sphere expanded to a final radius of  $R_f = 1.25$ . The associated space-time geometry consisted of a hyper-conical frustum embedded inside of a tesseract. This geometry is shown in Figure 11. For this geometry, we created a family of hypersurface meshes using the procedure described in the previous section. The mesh properties are summarized in Table 4. The total volume of each mesh was compared with the exact volume of the slab's hypersurface, which was computed as follows

$$V_{\text{exact}} = 2L^3 - \frac{4}{3}\pi \left(R_f^3 + R_0^3\right) + 6L^2 \left(t_f - t_0\right) + \frac{4}{3}\pi \left(\frac{R_f^3 - R_0^3}{R_f - R_0}\right) \sqrt{(R_f - R_0)^2 + (t_f - t_0)^2}.$$

Figure 12 shows a plot of the volumetric error versus the approximate mesh spacing. As expected, the error in the approximation deceases with increasing mesh resolution, and the rate of decrease is approximately 2nd order.



Figure 11: An illustration of an expanding 3-sphere inside of a 3-cube. Under these circumstances, the space-time geometry consists of a hyper-conical frustum embedded inside of a tesseract. Note: this drawing is *not* to scale.

Mesh	Elements	Vertices
1	$144,\!345$	30,801
2	368,477	$78,\!620$
3	958,364	$204,\!417$
4	$2,\!630,\!263$	560,955
5	$7,\!126,\!629$	$1,\!519,\!159$
6	$19,\!163,\!704$	4,087,112
7	$56,\!114,\!273$	$11,\!961,\!083$
8	149,461,495	$31,\!878,\!349$

Table 4: The number of tetrahedral elements and vertices for a sequence of hypersurface meshes for the expanding sphere test case.

## 4.5 Rotating Ellipsoid

The geometry for this test case consisted of a single ellipsoid with semi-axes of a = 1 in the x-direction, b = 3 in the y-direction, and c = 2 in the z-direction. The ellipsoid was located at the center of a 3-cube with edge length L = 16. In this 3-cube, the ellipsoid rotated around the z-axis with a constant speed of  $\omega = \frac{\pi}{2}$  rads/s, during the time interval [0, 1]. The resulting space-time geometry consisted of an 'ellipsoidal hyper-helix' contained inside of a tesseract. With this geometric configuration in mind, we generated a family of hypersurface meshes using the procedure described in the previous section. The hypersurface meshes were parameterized by the following quantities:  $h_{\text{ellipsoid}}$  was used to specify the mesh spacing near the ellipsoid surface,  $h_{\text{cube}}$  was used to specify the spacing



Figure 12: Each point on the plot above represents the error between the volume of a hypersurface mesh and the exact volume for the expanding sphere test case. The errors are plotted against the characteristic mesh spacing for a sequence of increasingly refined hypersurface meshes. In addition, a dashed line associated with 2nd-order convergence is plotted for reference.

near the cube walls, and  $h_{\text{time}}$  was used to specify the spacing along the temporal direction. The properties of the resulting hypersurface meshes are summarized in Table 5. In addition, Figure 13 shows some representative snapshots of the coarsest (lowest-resolution) hypersurface mesh.

Mesh	Elements	Vertices
1	355,798	$75,\!655$
2	$944,\!622$	200,867
3	$2,\!644,\!079$	562,277
4	$7,\!236,\!458$	$1,\!538,\!687$
5	$20,\!536,\!468$	$4,\!365,\!033$
6	$54,\!872,\!975$	$11,\!676,\!192$
7	$162,\!044,\!418$	$34,\!449,\!255$

Table 5: The number of tetrahedral elements and vertices for a sequence of hypersurface meshes for the rotating ellipsoid test case.

We compared the volume of the hypersurface mesh at the final time  $(t_f = 1)$  against the exact volume of the hypersurface. The exact volume at  $t_f = 1$  was calculated as follows

$$V_{\text{exact}}(t_f) = L^3 - \frac{4}{3}\pi abc.$$

Figure 14 shows a plot of the volumetric error versus the mesh resolution. Second-order convergence is obtained, as expected.

## 4.6 Rotating Tandem Ellipsoids

For this test case, the geometry consisted of a pair of ellipsoids with semi-axes of  $a_1 = 1, b_1 = 3, c_1 = 2$  and  $a_2 = 3, b_2 = 1, c_2 = 2$ , respectively. Both ellipsoids were placed inside of a 3-cube with edge length L = 20, and bounds given by  $[-7.5, 12.5] \times [-10, 10] \times [-10, 10]$ . The first ellipsoid was centered at (0, 0, 0)and the second at (5, 0, 0). In addition, the first ellipsoid rotated with angular velocity  $(0, 0, \pi/2)$  rads/s, and the second rotated with velocity  $(0, 0, -\pi/2)$ rad/s. On the time interval [0, 1], the motion of the ellipsoids created a pair of elliptical hyper-helixes that were contained inside of a tesseract. A family of hypersurface meshes was generated for this test case using the procedure described in the previous section. Table 6 summarizes the properties of these meshes. Furthermore, Figure 15 shows several characteristic snapshots of the coarsest hypersurface mesh.

Mesh	Elements	Vertices
1	603,432	$128,\!055$
2	$1,\!627,\!918$	$345,\!616$
3	$4,\!523,\!078$	959,759
4	12,082,322	2,566,188
5	$35,\!245,\!617$	$7,\!478,\!525$
6	$94,\!464,\!074$	20,066,876
7	278,561,736	$59,\!127,\!488$

Table 6: The number of tetrahedral elements and vertices for a sequence of hypersurface meshes for the rotating tandem ellipsoids test case.

The exact hypersurface volume at final time  $t_f = 1$  is given by

$$V_{\text{exact}}(t_f) = L^3 - \frac{4}{3}\pi \left(a_1 b_1 c_1 + a_2 b_2 c_2\right)$$

Figure 16 shows a plot of the volumetric error versus the mesh resolution. We obtain second-order convergence as expected.

#### 5. Conclusion

We have described in detail a general method for developing surface meshes in 2D+t space time and hypersurface meshes in 3D+t space time based on temporal planes (hyperplanes) derived from vertex trajectory-tracking through space time. These methods have been verified through numerical experiments by extruding/extending 2D and 3D objects along the temporal direction and comparing the approximate simplical surface areas or hypersurface volumes to the expected analytical results. All numerical errors demonstrate 2nd-order convergence as the element densities of the surface (hypersurface) meshes increase, which demonstrates that our methods are working as expected. In our future work, we will explore methods for Delaunay-based hypervolume meshing in 3D+t space time. This work will include the development of methods for recovering a hypersurface boundary mesh once an initial (unconstrained) hypervolume mesh has been generated.

# **Declaration of Competing Interests**

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.

# Funding

This research is sponsored by the Office of Naval Research, Code 351, through the Jet Noise Reduction Program under Program Officer, Dr. Steven Martens. Penn State University received funding under contract number N00173-22-2-C008.



Figure 13: Snapshots of a rotating ellipsoid at t = 0, t = 0.5, and t = 1, (top to bottom). On the left, a cross section of the coarsest hypersurface mesh is shown alongside the ellipsoid CAD. On the right, a zoomed-in view of the surface mesh on the ellipsoid is shown.



Figure 14: Each point on the plot above represents the error between the volume of a hypersurface mesh and the exact volume for the rotating ellipsoid test case. The errors are plotted against the characteristic mesh spacing for a sequence of increasingly refined hypersurface meshes. In addition, a dashed line associated with 2nd-order convergence is plotted for reference.



Figure 15: Snapshots of rotating tandem ellipsoids at t = 0, t = 0.5, and t = 1, (top to bottom). On the left, a cross section of the coarsest hypersurface mesh is shown, alongside the ellipsoids' CAD. On the right, a zoomed-in view of the surface meshes on the ellipsoids is shown.



Figure 16: Each point on the plot above represents the error between the volume of a hypersurface mesh and the exact volume for the rotating tandem ellipsoid test case. The errors are plotted against the characteristic mesh spacing for a sequence of increasingly refined hypersurface meshes. In addition, a dashed line associated with 2nd-order convergence is plotted for reference.

#### References

- T. J. Hughes, G. M. Hulbert, Space-time finite element methods for elastodynamics: formulations and error estimates, Computer Methods in Applied Mechanics and Engineering 66 (3) (1988) 339–363.
- [2] G. M. Hulbert, T. J. Hughes, Space-time finite element methods for second-order hyperbolic equations, Computer Methods in Applied Mechanics and Engineering 84 (3) (1990) 327–348.
- [3] M. Behr, Simplex space-time meshes in finite element simulations, International Journal for Numerical Methods in Fluids 57 (9) (2008) 1421–1434.
- [4] L. Pauli, M. Behr, On stabilized space-time FEM for anisotropic meshes: Incompressible Navier–Stokes equations and applications to blood flow in medical devices, International Journal for Numerical Methods in Fluids 85 (3) (2017) 189–209.
- [5] M. von Danwitz, V. Karyofylli, N. Hosters, M. Behr, Simplex space-time meshes in compressible flow simulations, International Journal for Numerical Methods in Fluids 91 (1) (2019) 29–48.
- [6] V. Karyofylli, L. Wendling, M. Make, N. Hosters, M. Behr, Simplex space-time meshes in thermally coupled two-phase flow simulations of mold filling, Computers & Fluids 192 (2019) 104261.
- [7] M. Make, T. Spenke, N. Hosters, M. Behr, Spline-based space-time finite element approach for fluid-structure interaction problems with a focus on fully enclosed domains, Computers & Mathematics with Applications 114 (2022) 210–224.
- [8] M. von Danwitz, I. Voulis, N. Hosters, M. Behr, Time-Continuous and Time-Discontinuous space-time finite elements for advection-diffusion problems, arXiv preprint arXiv:2206.01423.
- [9] M. von Danwitz, P. Antony, F. Key, N. Hosters, M. Behr, Four-dimensional elastically deformed simplex space-time meshes for domains with time-variant topology, International Journal for Numerical Methods in Fluids 93 (12) (2021) 3490–3506.
- [10] V. Karyofylli, M. Behr, Simplex space-time meshes in engineering applications with moving domains, arXiv preprint arXiv:2210.09831.
- [11] L. Wang, P.-O. Persson, A high-order discontinuous Galerkin method with unstructured space-time meshes for two-dimensional compressible flows on domains with large deformations, Computers & Fluids 118 (2015) 53–68.
- [12] L. Wang, Discontinuous Galerkin methods on moving domains with large deformations, Ph.D. thesis, University of California, Berkeley (2015).
- [13] T. L. Horváth, S. Rhebergen, A conforming sliding mesh technique for an embeddedhybridized discontinuous Galerkin discretization for fluid-rigid body interaction, International Journal for Numerical Methods in Fluids 94 (11) (2022) 1784–1809.
- [14] P. Foteinos, N. Chrisochoides, 4D space-time Delaunay meshing for medical images, Engineering with Computers 31 (3) (2015) 499–511.
- [15] H. Si, K. Gärtner, 3D boundary recovery by constrained Delaunay tetrahedralization, International Journal for Numerical Methods in Engineering 85 (11) (2011) 1341–1364.
- [16] Y. Liu, S. Lo, Z.-Q. Guan, H.-W. Zhang, Boundary recovery for 3D Delaunay triangulation, Finite Elements in Analysis and Design 84 (2014) 32–43.

- [17] R. Löhner, P. Parikh, Generation of three-dimensional unstructured grids by the advancing-front method, International Journal for Numerical Methods in Fluids 8 (10) (1988) 1135–1149.
- [18] P. L. George, É. Seveno, The advancing-front mesh generation method revisited, International Journal for Numerical Methods in Engineering 37 (21) (1994) 3605–3619.
- [19] D. S. Lo, Finite element mesh generation, CRC Press, 2014.
- [20] A. Üngör, A. Sheffer, Pitching tents in space-time: Mesh generation for discontinuous Galerkin method, International Journal of Foundations of Computer Science 13 (02) (2002) 201–221.
- [21] J. Erickson, D. Guoy, J. M. Sullivan, A. Üngör, Building spacetime meshes over arbitrary spatial domains, Engineering with Computers 20 (4) (2005) 342–353.
- [22] R. Abedi, S.-H. Chung, J. Erickson, Y. Fan, M. Garland, D. Guoy, R. Haber, J. M. Sullivan, S. Thite, Y. Zhou, Spacetime meshing with adaptive refinement and coarsening, in: Proceedings of the twentieth annual symposium on Computational geometry, 2004, pp. 300–309.
- [23] J. Gopalakrishnan, P. Monk, P. Sepúlveda, A tent pitching scheme motivated by Friedrichs theory, Computers & Mathematics with Applications 70 (5) (2015) 1114–1135.
- [24] J. Gopalakrishnan, J. Schöberl, C. Wintersteiger, Mapped tent pitching schemes for hyperbolic systems, SIAM Journal on Scientific Computing 39 (6) (2017) B1043–B1063.
- [25] D. Drake, J. Gopalakrishnan, J. Schöberl, C. Wintersteiger, Convergence analysis of some tent-based schemes for linear hyperbolic systems, Mathematics of Computation 91 (334) (2022) 699–733.
- [26] J. Gopalakrishnan, M. Hochsteger, J. Schöberl, C. Wintersteiger, An explicit mapped tent pitching scheme for Maxwell equations, in: Spectral and high order methods for partial differential equations – ICOSAHOM 2018, 2020, pp. 359–369.
- [27] H. Borouchaki, P. Laug, P.-L. George, Parametric surface meshing using a combined advancing-front generalized Delaunay approach, International Journal for Numerical Methods in Engineering 49 (1-2) (2000) 233–259.
- [28] H. Borouchaki, P. J. Frey, P. L. George, Unstructured triangular-quadrilateral mesh generation. application to surface meshing, in: Proc. of 5th Intl. Meshing Roundtable, Citeseer, 1996, pp. 229–242.
- [29] J. R. Tristano, S. J. Owen, S. A. Canann, Advancing front surface mesh generation in parametric space using a Riemannian surface definition., in: IMR, 1998, pp. 429–445.
- [30] Y. Zheng, R. W. Lewis, D. T. Gethin, Three-dimensional unstructured mesh generation: Part 2. Surface meshes, Computer Methods in Applied Mechanics and Engineering 134 (3-4) (1996) 269–284.
- [31] C. Lee, R. Hobbs, Automatic adaptive finite element mesh generation over rational Bspline surfaces, Computers & Structures 69 (5) (1998) 577–608.
- [32] S. A. Canann, Y.-C. Liu, A. V. Mobley, Automatic 3D surface meshing to address today's industrial needs, Finite Elements in Analysis and Design 25 (1-2) (1997) 185–198.
- [33] J.-C. Cuillière, An adaptive method for the automatic triangulation of 3D parametric surfaces, Computer-Aided Design 30 (2) (1998) 139–149.

- [34] C. Lee, Automatic metric 3D surface mesh generation using subdivision surface geometrical model. Part 2: mesh generation algorithm and examples, International Journal for Numerical Methods in Engineering 56 (11) (2003) 1615–1646.
- [35] S. Sherwin, J. Peiró, Mesh generation in curvilinear domains using high-order elements, International Journal for Numerical Methods in Engineering 53 (1) (2002) 207–223.
- [36] B. Lévy, N. Bonneel, Variational anisotropic surface meshing with Voronoi parallel linear enumeration, in: Proceedings of the 21st International Meshing Roundtable, Springer, 2013, pp. 349–366.
- [37] H. Borouchaki, J. Villard, P. Laug, P. George, Surface mesh enhancement with geometric singularities identification, Computer Methods in Applied Mechanics and Engineering 194 (48-49) (2005) 4885–4894.
- [38] P. Frey, H. Borouchaki, Surface meshing using a geometric error estimate, International Journal for Numerical Methods in Engineering 58 (2) (2003) 227–245.
- [39] Z. Zhong, L. Shuai, M. Jin, X. Guo, Anisotropic surface meshing with conformal embedding, Graphical Models 76 (5) (2014) 468–483.
- [40] T. Lan, S. Lo, Finite element mesh generation over analytical curved surfaces, Computers & Structures 59 (2) (1996) 301–309.
- [41] R. J. Cass, S. E. Benzley, R. J. Meyers, T. D. Blacker, Generalized 3-D paving: an automated quadrilateral surface mesh generation algorithm, International Journal for Numerical Methods in Engineering 39 (9) (1996) 1475–1489.
- [42] J. R. Shewchuk, Triangle: Engineering a 2D quality mesh generator and Delaunay triangulator, in: Workshop on Applied Computational Geometry, Springer, 1996, pp. 203–222.
- [43] S. Hang, TetGen, a Delaunay-based quality tetrahedral mesh generator, ACM Trans. Math. Softw 41 (2) (2015) 11.
- [44] A. Corrigan, A. Kercher, D. Kessler, The moving discontinuous Galerkin method with interface condition enforcement for unsteady three-dimensional flows, in: AIAA (Ed.), 2019 AIAA SciTech Forum, 2019, AIAA-2019-0642. doi:10.2514/6.2019-0642.