2014

A Consistent Algorithm for Implementing the Space Conservation Law

Venkata Pavan Pillalamarri Narasimha Rao
University of Massachusetts Amherst

A CONSISTENT ALGORITHM FOR IMPLEMENTING THE SPACE CONSERVATION LAW

A Thesis Presented

by

VENKATA PAVAN PILLALAMARRI NARASIMHA RAO

Submitted to the Graduate School of the
University of Massachusetts Amherst in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN MECHANICAL ENGINEERING

May 2014

Mechanical and Industrial Engineering

A CONSISTENT ALGORITHM FOR IMPLEMENTING THE SPACE CONSERVATION LAW

A Thesis Presented

by

VENKATA PAVAN PILLALAMARRI NARASIMHA RAO

Approved as to style and content by:

David P. Schmidt, Chair

James B. Perot, Member

Yahya Modarres-Sadeghi, Member

Donald L. Fisher, Department Chair
Mechanical and Industrial Engineering

To my mother

ACKNOWLEDGEMENTS

I had the privilege of working under the guidance of Dr. David Schmidt for this project. I am grateful for his consistent encouragement and support at every stage of my research. I deeply admire his accommodating nature and his brilliant approach towards solving the problems of Computational Fluid Dynamics (CFD). I would also like to thank Dr. Blair Perot and Dr. Yahya Modarres-Sadeghi for being on my thesis committee and providing valuable inputs during different stages of this work.

My time in the Multi-Phase Flow Simulation Laboratory has been very inspiring. As a CFD analyst accustomed to using GUI based CFD software, open-source software appeared difficult and tedious at first. But with help from my lab colleagues I was able to progress from struggling with it to working with it comfortably. Undoubtedly, these are some of the smartest people I ever met. In no particular order, I would like to thank Kyle Mooney, Michael Martell, Maija Benitz, Bradley Shields, Maryam Moulai, Eli Baldwin, Alex Chan, Christopher Zusi and Yang Song for their continued help and support.

Last but not least, I would like to thank the Physics department and the Auxiliary Services Department at University of Massachusetts-Amherst, the Army Research Office and Baker Hughes Incorporated for their financial support during different stages of my graduate studies.

ABSTRACT

A CONSISTENT ALGORITHM FOR IMPLEMENTING THE SPACE CONSERVATION LAW

MAY 2014

VENKATA PAVAN PILLALAMARRI NARASIMHA RAO

B.Tech., VELLORE INSTITUTE OF TECHNOLOGY, VELLORE

M.S.M.E., UNIVERSITY OF MASSACHUSETTS AMHERST

Directed by: Professor David P. Schmidt

Fluid flows occurring in moving and/or deforming environments are influenced by the transient nature of their containment. In Computational Fluid Dynamics (CFD), simulating such flow fields requires effort to maintain the geometric integrity of the transient flow domain. Convective fluxes in such domains are evaluated with respect to the motion of the boundaries of the control volume. These simulations demand conservation of space in addition to the conservation of mass, momentum and energy as the solution continues in time.

The Space Conservation Law in its continuous form can be inferred by using the rules of fundamental calculus. However, implementing it in a discrete form poses substantial challenges. During mesh motion, the surfaces enclosing the control volumes sweep through three-dimensional space. As per the Space Conservation Law, the change in the control volume has to match the sum of the swept volumes of all its faces exactly. The Space Conservation Law must be satisfied accurately and con-

sistently in order to avoid the occurrence of non-physical masses and to prevent the violation of the continuity equation.

In this work we have attempted to address the consistency issues surrounding the implementation of the Space Conservation Law in OpenFOAM®. The existing method for calculation of swept volumes falls short in terms of consistency. Moreover, its capabilities are limited when it comes to complex three-dimensional mesh motions. The existing method of calculation treats swept volumes as net fluxes emanating from cell faces. We have implemented an alternate algorithm in which the swept volumes are treated as intermittent virtual cells whose volumes can be calculated in a unique and consistent manner. We will conclude by validating our approach for mesh motions of varying degrees of complexity.

TABLE OF CONTENTS

LIST OF TABLES

Table **Page**

LIST OF FIGURES

Figure **Page**

CHAPTER 1

INTRODUCTION

Developing a mathematical model for most physical processes involves identifying a unique relationship between the participating variables. The relationship can then be articulated as equations which can eventually be solved either analytically or numerically. In a finite volume approach to Computational Fluid Dynamics (CFD), the Navier-Stokes system of equations are used as governing equations. Balancing these equations ensures the conservation of mass, momentum and energy within a control volume. Unfortunately, these equations have not yet been solved analytically. Therefore, they require the use of numerical methods.

The equations governing fluid flows in static and transient domains are nearly identical. The only difference is that the fluid velocities in transient control volumes are calculated relative to the velocities of the control volume boundaries, which are themselves determined by dedicated mesh motion solvers. Maintaining the geometric integrity of a transient domain is a priority in dynamic mesh simulations. Therefore, an additional equation for space conservation must also be satisfied simultaneously.

Fundamentally, the idea of space conservation originates from the Leibniz's integral rule [29]. According to this rule, for a continuous function $f(x,t)$ in a one-dimensional domain,

$$\frac{d}{dt}\int_{a(t)}^{b(t)} f(x,t)dx = \left[\int_{a(t)}^{b(t)} \left(\frac{\partial f(x,t)}{\partial t}dx\right)\right] + \left[f(x,t)\frac{dx}{dt}\right]_{a(t)}^{b(t)} \tag{1.1}$$

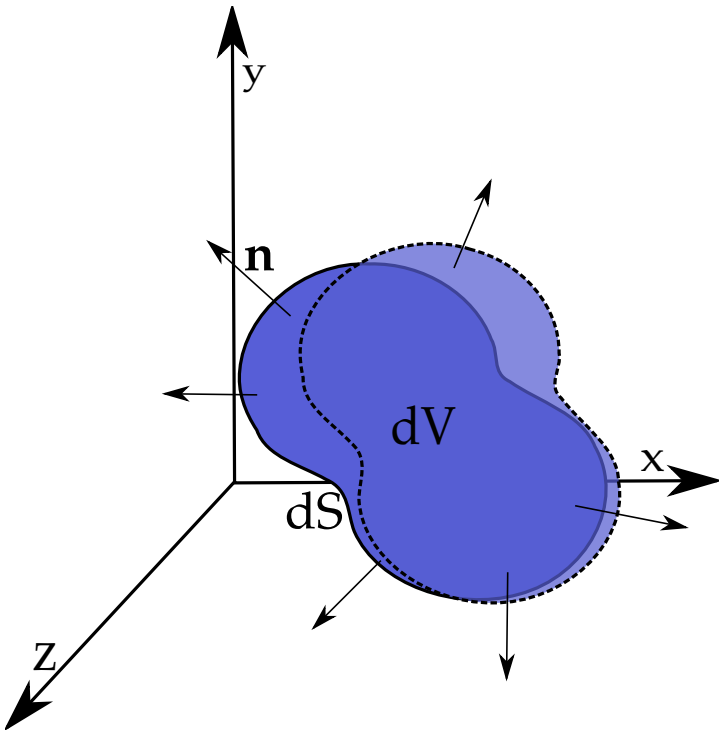


Figure 1.1: A deforming control volume.

$$\Rightarrow \frac{d}{dt}\int_{a(t)}^{b(t)} f(x,t)dx = \left[\int_{a(t)}^{b(t)} \left(\frac{\partial f(x,t)}{\partial t}dx\right)\right] + \left[f(b(t),t)\frac{d(b(t))}{dt}\right] - \left[f(a(t),t)\frac{d(a(t))}{dt}\right] \tag{1.2}$$

For a three-dimensional interpretation, let there be a function $f(x,y,z,t)$ which is continuous inside a transient control volume $V(t)$, and let $S(t)$, with normal $\mathbf{n}$ directed outwards, be the surface enclosing $V(t)$ [Figure 1.1]. Then, according to the Leibniz's rule [2],

$$\frac{d}{dt}\iiint_{V(t)} f(x,y,z,t)dV = \left[\iiint_{V(t)} \frac{\partial f(x,y,x,t)}{\partial t}dV\right] + \left[\iint_{S(t)} f(x,y,z,t)\mathbf{v}.\mathbf{n}dS\right] \tag{1.3}$$

where $\mathbf{v}$ is the velocity with which the function $f(x,y,z,t)$ and the control volume $V(t)$ are moving.

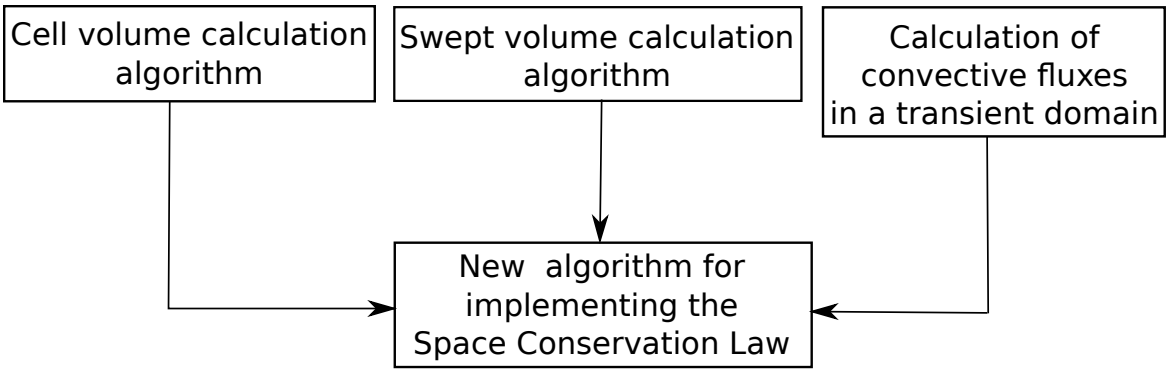


Figure 1.2: Building blocks for our implementation of the Space Conservation Law.

Equation (1.3) demonstrates the influence of a transient control volume on the values of $f(x, y, z, t)$. This three-dimensional form of the Leibniz's rule was derived by Osborne Reynolds. In his honor it is called the Reynolds' Transport Theorem.

Reynolds' Transport Theorem is the first step in deriving conservation equations in continuum mechanics. In a finite volume approach, it can be used to gain an understanding of variation in intensive properties due to a moving and/or deforming control volume. Applying the law of mass conservation to Equation (1.3) gives the following expression [12]:

$$\left[\frac{\partial}{\partial t}\iiint\limits_{V(t)} \rho dV\right] + \left[\iint\limits_{S(t)} \rho\left(\mathbf{v} - \mathbf{v}_b\right).\mathbf{n} dS\right] = 0 \tag{1.4}$$

where $\mathbf{v}_b$ is the velocity of the enclosing surface $S(t)$. Equation (1.4) is a generic form of the continuity equation.

Applying Equation (1.3) for momentum conservation of the i^{th} coordinate [12] gives the following result:

$$\left[\frac{\partial}{\partial t}\iiint\limits_{V(t)} \rho u_i dV\right] + \left[\iint\limits_{S(t)} \rho u_i\left(\mathbf{v} - \mathbf{v}_b\right).\mathbf{n} dS\right] = \left[\iint\limits_{S(t)} \left(\tau_{ij}\mathbf{i}_j - p\mathbf{i}_i\right).\mathbf{n} dS)\right] + \left[\iiint\limits_{V(t)} b_i dV\right] \tag{1.5}$$

When analysing Equation (1.4) for a uniform static material with no generation, we can say:

$$\left[\frac{\partial}{\partial t}\iiint_{V(t)} dV\right] - \left[\iint_{S(t)} \mathbf{v}_b.\mathbf{n}dS\right] = 0 \tag{1.6}$$

Equation (1.6) describes the idea of space conservation in a transient control volume. Working backwards, we can achieve continuity in a moving and/or deforming domain by balancing the space conservation equation and using the Reynolds' Transport Theorem [12] [3]. Balancing the space conservation equation prevents non-physical masses from entering the domain during the mesh-motion. CFD software that processes dynamic mesh simulations generally use the constraint of space conservation to maintain the geometric integrity of the simulation domain.

After analyzing the implementation of the Space Conservation Law in OpenFOAM®, we think that the consistency of the algorithm can be improved. In the chapters following the literature review, we will provide a detailed account of the consistency issues and describe an alternate algorithm for implementing the Space Conservation Law. In order to provide the groundwork for our algorithm, we will also review some parts of the existing source code. As illustrated in Figure 1.2, we will examine the source code in order to understand the existing procedure for calculating cell volumes, swept volumes and convective fluxes in OpenFOAM®.

CHAPTER 2

LITERATURE REVIEW

A general understanding of the Space Conservation Law in its continuous form stems from the rules of fundamental calculus. However, implementing it in a discrete domain has been one of the crucial goals of numerous publications in various journals. Guillard and Farhat [15] provided a comprehensive account of the significance of balancing the Space Conservation Law in problems involving transient domains. They proved that for a numerical scheme which produces n^{th} order accurate results on a static domain, satisfying the Space Conservation Law ensures that it produces first order accurate results while working on a transient domain.

Thomas and Lombard [30] identified the limitations of boundary conforming coordinate transformations in maintaining the global conservation in problems involving moving boundaries. In this pioneering work, they derived a differential form of the space conservation equation and carried out implicit solutions of the Navier-Stokes equations using finite difference methods. Demirdzic and Peric [9] published the first investigation addressing the issues of space conservation from a finite volume perspective. They provided a detailed explanation of the errors which appear in the solution as a consequence of overlooking the Space Conservation Law. It is worthwhile to discuss this paper in detail because contemporary CFD software often uses finite volume methods, including OpenFOAM®.

Examining Demirdzic and Peric's [9] analysis with regard to the discretized form of the Space Conservation Law, let us assume that at a particular time instant t there is a two-dimensional rectangular control volume which has sides e, w, n and s [Figure

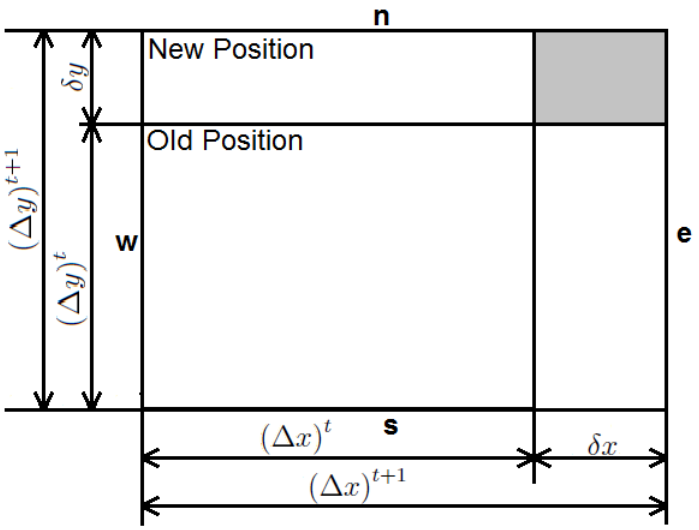


Figure 2.1: A deforming two-dimensional control volume. (Picture reproduced from Peric and Ferziger [13])

2.1]. Let us also assume that it contains a fluid which has constant density ρ. As time progresses, sides e and n move along the east and north directions respectively. The discrete form of the continuity equation [Equation (1.4)] over the control volume, using the explicit Euler time-stepping method, can be written as:

$$\frac{\rho\left[(V)^{t+1}-(V)^{t}\right]}{\Delta t}+\rho\left[(v_x-v_{bx})_e-(v_x-v_{bx})_w\right]^{t+1}(\Delta y)^{t+1}+$$
$$\rho\left[(v_y-v_{by})_n-(v_y-v_{by})_s\right]^{t+1}(\Delta x)^{t+1}=0 \quad (2.1)$$

where v is the fluid velocity and v_b is the velocity of control surface. The subscripts x and y denote the velocity components along the abscissa and the ordinate respectively. The subscripts e, w, n and s denote the movement of the respective boundaries. Assuming that the fluid is moving with a constant velocity in all directions, Equation (2.1) can be mathematically manipulated and rewritten as:

$$\frac{\rho\left[(V)^{t+1}-(V)^{t}\right]}{\Delta t}-\rho\left[(v_{bx})_e-(v_{bx})_w\right](\Delta y)^{t+1}-$$
$$\rho\left[(v_{by})_n-(v_{by})_s\right](\Delta x)^{t+1}=0 \quad (2.2)$$

As per the fundamental relations shown in Figure 2.1, we can infer the following relations:

$$[(v_{bx})_e - (v_{bx})_w] = \frac{\delta x}{\Delta t}$$
$$[(v_{by})_n - (v_{by})_s] = \frac{\delta y}{\Delta t}$$
$$(V)^{t+1} = (\Delta x \Delta y)^{t+1}$$
$$(V)^t = \left[(\Delta x)^{t+1} - \delta x\right]\left[(\Delta y)^{t+1} - \delta y\right]$$

Substituting the above relations in Equation (2.2) we get:

$$\frac{\rho\left[(\Delta x\Delta y)^{t+1} - \left((\Delta x)^{t+1} - \delta x\right)\left((\Delta y)^{t+1} - \delta y\right)\right]}{\Delta t} - \rho\left[\frac{\delta x}{\Delta t}\right](\Delta y)^{t+1} - \rho\left[\frac{\delta y}{\Delta t}\right](\Delta x)^{t+1} = 0 \quad (2.3)$$

After mathematical manipulation we get:

$$\rho\frac{\delta x \delta y}{\Delta t} = 0 \quad (2.4)$$

Equations (2.1) through (2.4) have been reproduced from Peric and Ferziger [13] with minor modifications in notation. Equation (2.4) clearly demonstrates that the continuity equation is not satisfied during the deformation of the control volume. The residual term on the left hand side of Equation (2.4) is basically the mass confined in the area which is represented by the shaded region and is shared by edges e and n during the movement. This quantity of mass will accumulate at every time-step for each cell in the mesh. However, the value of δx and δy depends on the size of the time-step [Figure 2.2]. We can reduce the value of the residual mass by reducing the time-step, but that would increase the computational cost.

Another way of circumventing the problem is by moving one boundary at a time. This effectively reduces the problem to a one-dimensional boundary movement case.

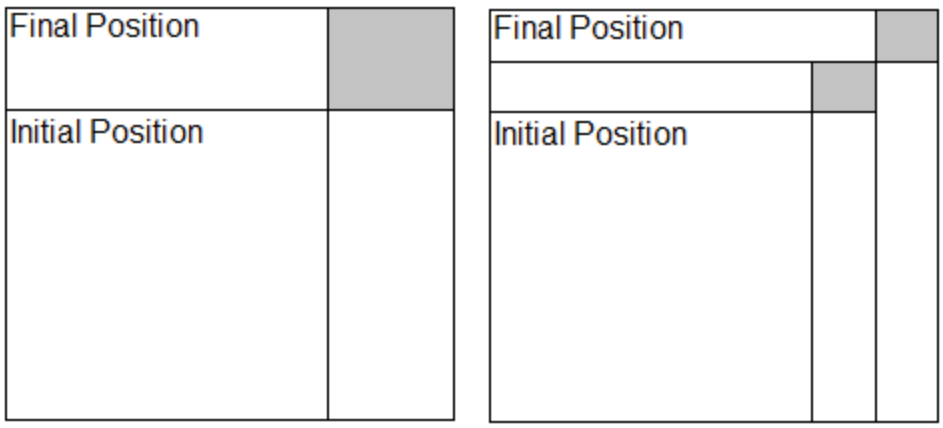


Figure 2.2: Effect of size of the time-step on residual mass. (Picture reproduced from Demirdzic and Peric [9])

However, that is not a pragmatic approach because it would lead to procedural problems during implementation as well as to an increase in computational cost. Furthermore, we cannot use this approach for problems like free surface flows. Therefore, it can be concluded that in a discretized system balancing the space conservation equation is critical.

Jeng and Chen [19] have also emphasized its importance. In their work using the SIMPLER algorithm on a two-dimensional structured mesh, they demonstrated that cancelling out non-physical convective fluxes is secondary in importance to satisfying the Space Conservation Law in moving mesh problems.

Before reviewing different approaches for implementing the Space Conservation Law, we must examine its discrete form and discuss the geometrical significance of the different terms included in it. The discrete form of Equation (1.6) can be written as:

$$\left[\frac{(V^{t+1} - V^t)}{\Delta t}\right] - \left[\sum_{faces} (\mathbf{v}_b.\mathbf{n}dS)\right] = 0 \tag{2.5}$$

Equation (2.5) is composed of two expressions on the left hand side, which must add up to zero when calculated exactly. The first term accounts for the change in

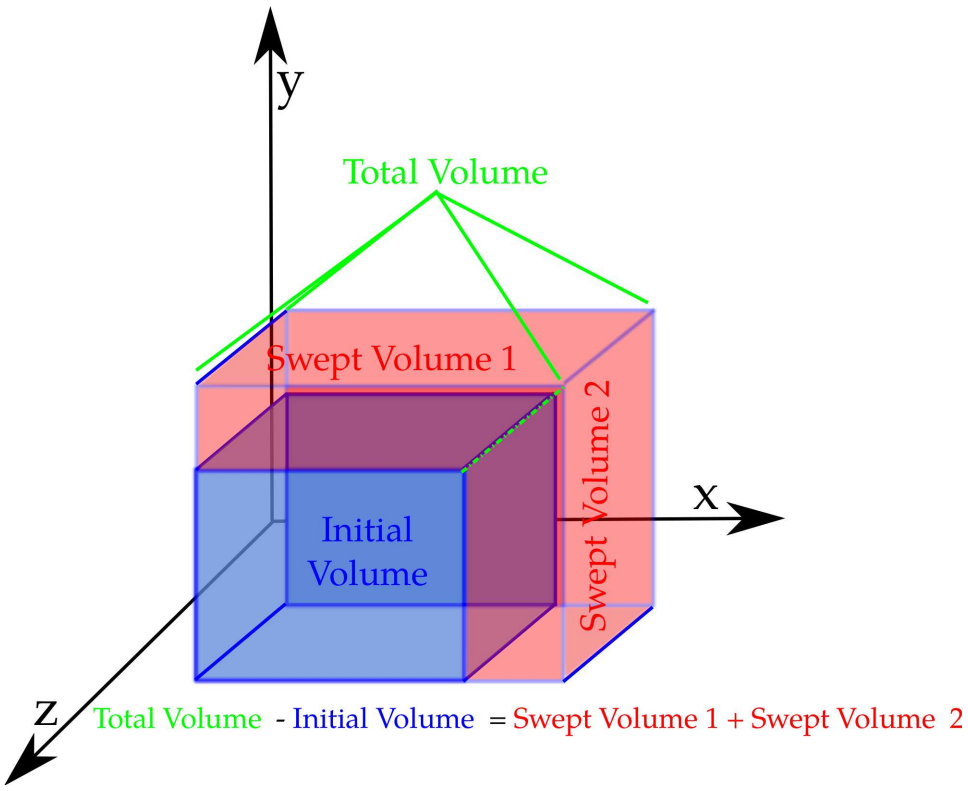


Figure 2.3: A deforming three-dimensional control volume describing the geometric significance of Equation (2.5).

volume of the cell during a particular time-step. The second term calculates the sum of volumes swept by individual faces of the cell during the same time-step [Figure 2.3]. The challenges in implementing the Space Conservation Law include dealing with polygonal faces and the change in the shapes of faces during an unconstrained three-dimensional mesh motion.

Satisfying the Space Conservation Law requires accomplishing two objectives. One objective is calculating the net flux of fluid across the control surface. The other objective is calculating the volumes created due to the movement of cell faces, hereafter referred to as the swept volumes. Because the calculation of net convective flux depends upon the calculation of swept volumes, a number of different attempts have been made to calculate the swept volumes and to thereby satisfy the Space Conservation Law exactly. A substantial number of publications have proposed and

implemented various methodologies to accomplish this since Demirdzic and Peric [9]. Some of them have limited their work to Cartesian meshes while others have used approaches to suit their particular problems of fluid flows and mesh motion.

The approaches used in all these publications can be broadly classified into two categories:

1. Algorithms for calculating the velocity of the cell boundary $\mathbf{v}_b$ in order to implicitly balance the Space Conservation Law, hereafter referred to as the Boundary Velocity Approach.

2. Algorithms for calculating the exact swept volumes, hereafter referred to as the Swept Volume Approach.

2.1 Boundary Velocity Approach

Demidzic and Peric [9] proposed an approach to define the cell boundary velocities $\mathbf{v}_b$ in such a way that the space conservation equation is satisfied automatically. Their work was confined to implementations in two-dimensional Cartesian mesh problems. According to their work, for a domain defined in Figure 2.1, defining the velocities of cell boundaries as:

$$v_{bx} = \frac{(\Delta y)^t + (\Delta y)^{t+1}}{2(\Delta y)^{t+1}} \frac{\delta x}{\Delta t} \tag{2.6}$$

$$v_{by} = \frac{(\Delta x)^t + (\Delta x)^{t+1}}{2(\Delta x)^{t+1}} \frac{\delta y}{\Delta t} \tag{2.7}$$

guaranteed that the Space Conservation Law would be satisfied. Demirdzic and Peric [10] extended the scope of this method to arbitrarily shaped domains with moving boundaries. However, this work was also limited to a two-dimensional paradigm.

Koutsavdis and Tsangaris [20] used the space conservation equation to specify the mesh velocity. Using primarily a two-dimensional structured mesh, they made an assumption of irrotationality about the velocity vector field generated by the nodal velocities. Given the cell volumes (which transform into face areas in 2D domains) Ω^t and $\Omega^{(t+1)}$, where t denotes the time-step, they introduced the grid velocity potential function:

$$(v_{bx}, v_{by}) = \nabla \Phi_b \tag{2.8}$$

Then they derived the following expression for balancing the differential form of the Space Conservation Law:

$$\nabla^2 \Phi_b = \frac{1}{I}\frac{\partial I}{\partial t} = \frac{1}{\Delta t}\frac{|\Omega|_{ij}^t}{|\Omega|^t}\left(\frac{|\Omega|^{t+1}}{|\Omega|^t} - 1\right) \tag{2.9}$$

where $|\Omega|_{ij}^t$ denotes the area of the i, j^{th} individual cell at the t^{th} time-step, $|\Omega|^t$ denotes the area of the entire domain at t^{th} time-step and Δt denotes the time interval. Their approach ensured that the cells were deformed according to their initial size. Cao et al. [6] also derived a differential form of the Space Conservation Law. While implementing a Space Conservation Law-based moving mesh method, they performed time-dependent coordinate transformations ($x = x(\xi, t)$) of the mesh locations. Following their derivation, the differential form of the Space Conservation Law can be written as:

$$\nabla . \mathbf{v}_b = \frac{1}{J}\frac{DJ}{Dt} \tag{2.10}$$

where $\mathbf{v}_b$ is the mesh velocity and the Jacobian for mesh adaptation is defined as:

$$J(\xi, t) = \frac{c(\xi)}{\rho(x(\xi, t), t)} \tag{2.11}$$

where ρ is a user-defined monitor function and $c = c(\xi)$ is a time-independent function determined by the initial coordinate transformation.

Shyy et al. [28] mapped a two-dimensional irregular physical domain to a fixed uniform computational space by defining the Cartesian coordinates in terms of generalized curvilinear coordinates(ξ, η) and time(τ) where

$$x = x(\xi, \eta, \tau),\ y = y(\xi, \eta, \tau),\ t = \tau$$

Their definition of the velocities of cell boundaries was similar to Demirdzic and Peric [9]. However, Shyy et al. implemented the requisite mathematics for expressing the grid velocities in terms of the new transformed coordinates.

Perot and Nallapati [27] also balanced the Space Conservation Law using the Boundary Velocity Approach. They presented the definitions for the velocities of cell boundaries in two and three-dimensional space. As their work shows, the velocities of the cell boundaries can be defined in the following way:

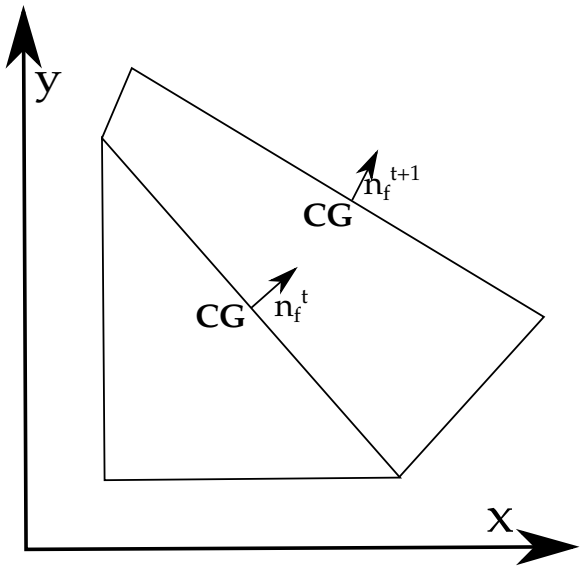


Figure 2.4: Two-dimensional transient domain. Here, **CG** denotes the center of gravity and $\mathbf{n}_f$ is the area normal vector

For a two-dimensional domain [Figure 2.4]:

$$U_f^{mesh} = \mathbf{u}_{mesh}^{CG} . \frac{1}{2} \left(\mathbf{n}_f^{t+1} A_f^{t+1} + \mathbf{n}_f^t A_f^t \right) \tag{2.12}$$

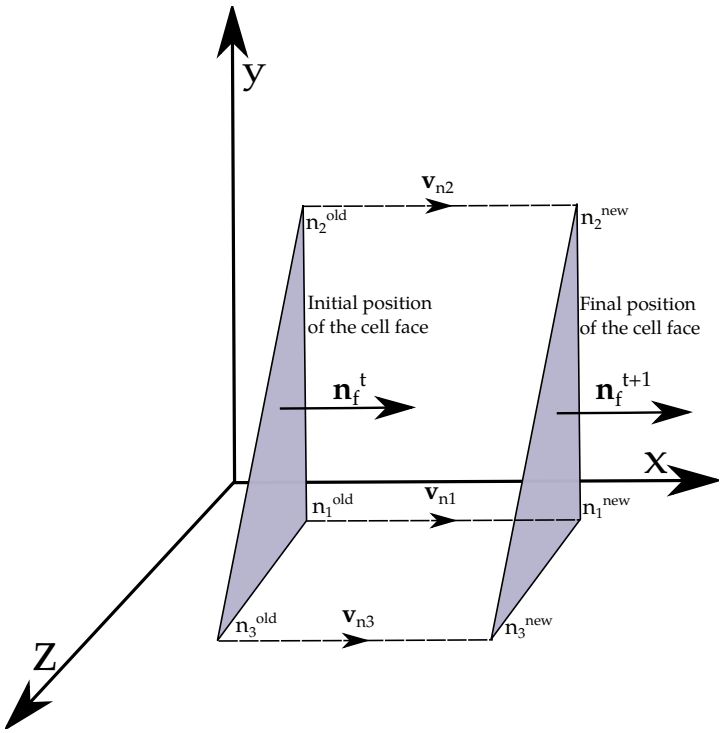


Figure 2.5: A moving and/or deforming face of a three-dimensional control volume.

and for a three-dimensional domain [Figure 2.5]:

$$U_f^{mesh} = \mathbf{u}_{mesh}^{CG}. \left[\frac{1}{2} \left(\mathbf{n}_f^{t+1} A_f^{t+1} + \mathbf{n}_f^t A_f^t \right) - \frac{\Delta t^2}{12} \sum_{e}^{face\ edges} \left(\mathbf{v}_{n1} \times \mathbf{v}_{n2} \right) \right] \tag{2.13}$$

Equations (2.12) and (2.13) have been reproduced from Perot and Nallapati [27]. Here, **CG** stands for center of gravity. In this case **CG** coincides with the centroid of the geometry. U_f^{mesh} is the velocity of the surface enclosing the control volume whose area is denoted by A and the area normal vector is denoted by $\mathbf{n}_f$. The variable $\mathbf{v}_n$ is the velocity of the face corners and the superscripts t and $(t+1)$ denote the positions of the corner nodes at the respective time steps. This method calculates the swept volumes exactly. It is one of the very few approaches which is exact as well as generic in nature. Dai [8] also used this approach in his research. A derivation of Equation (2.13) is available in Appendix A.

2.2 Swept Volume Approach

It can be noted from Equation(1.6) that the mesh velocity $\mathbf{v}_b$ appears only in the second term in the product of velocity and area. The expression $\left[\iint_{S(t)} \mathbf{v}_b.\mathbf{n}dS\right]$ is simply the rate at which the volume is swept by the boundaries of the control volume. Knowing the locations of the vertices which comprise each face at the commencement and completion of each time step is sufficient for calculating the volume swept by each face. Apsley and Hu [1] and Jasak and Tukovic [17] also provided similar reasons for using the Swept Volume Approach for simulating fluid flows involving two and three-dimensional mesh movements. The information regarding co-ordinates of face vertices is simple to extract and store. There are also no problems with the computational cost since the storage space can be allocated dynamically during simulation runtime. Several approaches for calculating swept volumes have been published since the work of Demirdzic and Peric [9].

Zhang et al. [32] published one of the initial implementations of the Space Conservation Law using the Swept Volume Approach. Primarily, they used face triangulation for a two-dimensional implementation and tetrahedral decomposition for a three-dimensional domain. As per their formulations in two dimensions, for the deforming pentagonal face shown in Figure 2.6 where the vertices from locations $1, 4$, and 5 are shifted to locations $1', 4'$, and $5'$, the areas $1-5-5'-1'$ and $5-4-4'-5'$ are the volumes swept by edges $1-5$ and $4-5$. They calculated the swept areas by decomposing the quadrilaterals into triangles by joining one set of diagonally opposite vertices. They also proposed the logic for selecting the vertices for the triangulation, resulting in the computed facial increment being independent of nodal motions. Therefore, if we consider the movement of edge $5-1$, the area swept by vector $\Delta r_{15} = r_5 - r_1$ can be expressed as the sum of the areas of triangles $1-5-1'$ and $5-5'-1$. As per Zhang et al. [32] the generic expression for the calculation of an area swept by the edge $\mathbf{r}_{12}$ can be written as:

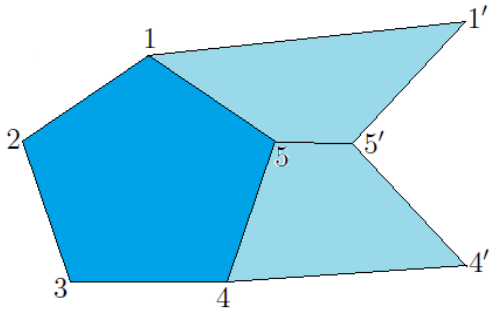


Figure 2.6: Edges $1-5$ and $4-5$ have transformed to $1'-5'$ and $4'-5'$.

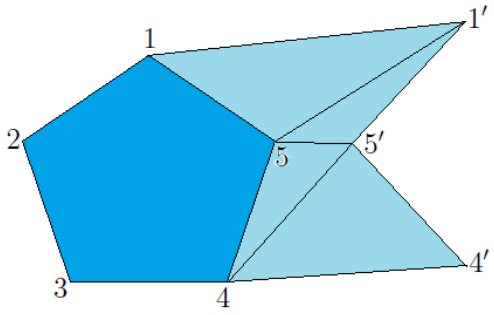


Figure 2.7: Decomposition of swept surfaces by joining $1'-5$ and $5'-4$.

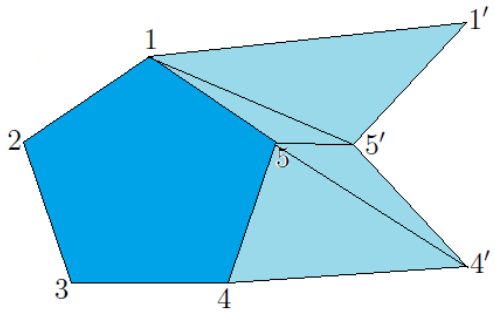


Figure 2.8: Decomposition of swept surfaces by joining $1-5'$ and $5-4'$.

$$\Delta V = \Delta t \mathbf{v}_b \times \Delta \mathbf{r}_{12}^{t+\frac{1}{2}} \tag{2.14}$$

where $\mathbf{v}_b = (\mathbf{v}_{b1} + \mathbf{v}_{b2})/2$. Extending their work to the three-dimensional domain, Zhang et al. [32] proposed to decompose the faces of a cell into triangles by connecting the face center to each vertex comprising the face [Figure 2.9]. Each such triangle had vertices defined in three-dimensional space by the vectors $\mathbf{r}_1$, $\mathbf{r}_2$ and $\mathbf{r}_3$. The volume swept by these vectors in a direction parallel to its area normal vectors was calculated by considering all six permutations for sequencing the vertices $\mathbf{r}_1$, $\mathbf{r}_2$ and $\mathbf{r}_3$. The swept volume of one of the permutations, $[1, 2, 3]$ for example, can be written as:

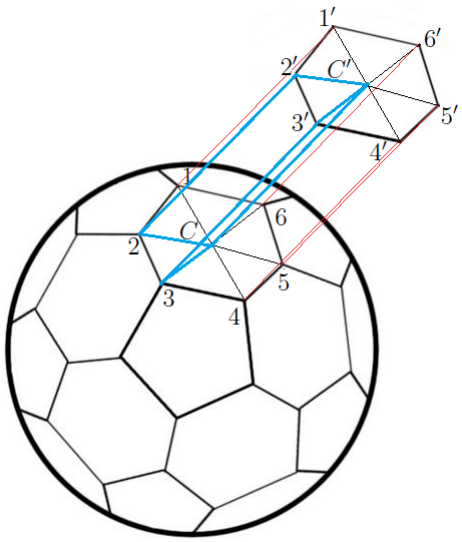


Figure 2.9: A representation of face triangulation for calculation of swept volumes proposed by Zhang et al. [32].

$$\Delta V_{[123]} = \left[\frac{\Delta t}{6}\mathbf{v}_{b1}.\mathbf{S}^t\right] + \left[\frac{\Delta t}{6}\mathbf{v}_{b2}.\left(\mathbf{r}_2^t - \mathbf{r}_1^{t+1}\right) \times \left(\mathbf{r}_3^t - \mathbf{r}_1^{t+1}\right)\right] + \left[\frac{\Delta t}{6}\mathbf{v}_{b3} \times \mathbf{r}_{12}^{t+1} \times \left(\mathbf{r}_3^t - \mathbf{r}_1^{t+1}\right)\right] \quad (2.15)$$

The swept volumes calculated with each of the permutations can be different and must therefore be calculated separately. An arithmetic mean of all such volumes was used for the volume swept by that particular triangular face. The sum of the swept volumes of all the triangles was added to get the volume swept by the polygonal face. A generic expression for calculating the swept volumes after the face triangulation was presented as:

$$\Delta V = \left[\Delta t \frac{\mathbf{v}_{b1} + \mathbf{v}_{b2} + \mathbf{v}_{b3}}{3}.\mathbf{S}^{t+\frac{1}{2}}\right] + \left[\frac{\Delta t^3}{24}\mathbf{v}_{b1}.\mathbf{v}_{b2} \times \mathbf{v}_{b3}\right] \quad (2.16)$$

Bos [5] implemented a different algorithm for calculating swept volumes. His method included tetrahedral decomposition of the prismatic volumes formed due to the sweeping action of triangulated cell faces. This method involved connecting one of the vertices of the triangular face at time $(t+1)$ with two of its opposite vertices at

time t, which created the first tetrahedron. The remaining volume of the prism was decomposed into two more tetrahedrons by connecting any of the diagonals of the quadrilateral face available in it. This led to two unique ways in which the remaining space could be decomposed [Figure 2.10].

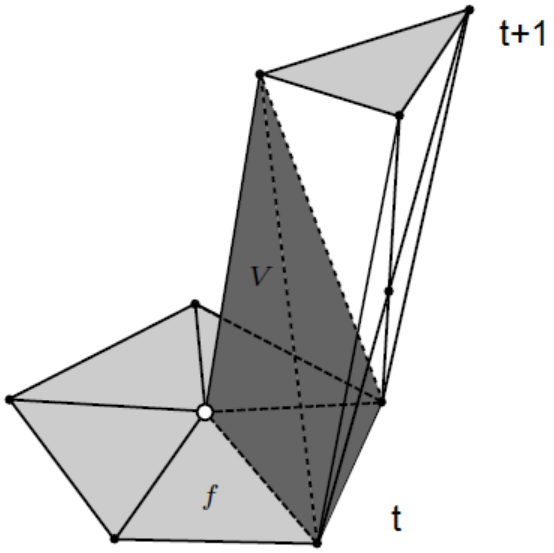


Figure 2.10: Calculation of swept volumes as implemented by Bos. (Picture reproduced from Bos [5])

As there is no single unique method of decomposition in this approach, the swept volumes are calculated using both approaches and an averaged value is used as the magnitude of the swept volume. Due to the proven accuracy of this approach, Bos's [5] method is currently used for calculation of swept volumes in OpenFOAM®. The expression for swept volumes using Bos's [5] method can be expressed as:

$$\Delta V_{total} = \frac{1}{12}\left[(V_1 + V_2 + V_3) + (V_1 + V_4 + V_5)\right] \tag{2.17}$$

where ΔV_{total} is the volume of the prism, and $V_n (n = 1, 2...)$ is the volume of the individual tetrahedrons. Bos's [5] implementation appears to be similar to Zhang et al. [32] in some aspects. Both of them used a similar method of tetrahedral de-

composition. However, Zhang et al. [32] used the values of velocity vectors in their implementation whereas Bos's [5] method completely relies on the spatial locations of the face vertices at successive time-steps.

Chau and Fringer's [7] work is one of the more recent attempts to implement the Space Conservation Law in its differential form. Their implementation was focussed on three-dimensional structured meshes and emphasized the importance of consistency in implementing the discrete form of the Space Conservation Law.

Tukovic and Jasak [31] [18] used a novel way of calculating the swept volumes. This particular implementation used a three level backward scheme for temporal discretization in order to satisfy the Space Conservation Law in the following form:

$$\frac{3V^t - 4V^{t-1} + V^{t-2}}{2\Delta t} - \sum_{faces} [V_f] = 0 \tag{2.18}$$

where V_f is the volume swept by each face of the polyhedral cell. The difference between cell volumes at consecutive time-steps was expressed as:

$$V^t - V^{t-1} = \sum_{faces} \delta V^t_{faces} \tag{2.19}$$

where δV^t_{faces} is the volume swept by cell face f while moving from its old position to its new position. After substituting Equation(2.19) in Equation(2.18), the swept volume for a face f can be expressed as:

$$\Delta V_f = \left[\frac{3}{2}\frac{\delta V^t_{faces}}{\Delta t}\right] - \left[\frac{1}{2}\frac{\delta V^{t-1}_{faces}}{\Delta t}\right] \tag{2.20}$$

All the implementations discussed so far were based on the Eulerian coordinate system. A number of publications have also implemented the Space Conservation Law using the Arbitrary Lagrangian Eulerian(ALE) framework. ALE methods are used extensively in problems involving fluid-structure interactions. Hu et al. [16] provided

a detailed explanation about the transformation of functions between the material and referential domains. For example, in an ALE formulation, a material derivative is generally written as:

$$\frac{D\mathbf{v}}{Dt} = \frac{\partial \mathbf{v}}{\partial t} + \mathbf{v}.(\nabla \mathbf{v}) \tag{2.21}$$

However, for the velocity of a fluid flow at a given point in time and space, the material derivative in a referential domain is represented as:

$$\frac{D}{Dt}\mathbf{v}(x,t) = \frac{\delta \mathbf{v}}{\delta t} + \nabla \mathbf{v}\left[(\mathbf{v} - \hat{\mathbf{v}})\right] \tag{2.22}$$

where

$$\frac{\delta}{\delta t}\mathbf{v}(x,t) = \frac{\partial}{\partial t}\mathbf{v}(x(\chi,t),t)|_{\chi}$$

$\hat{\mathbf{v}}$ is the velocity of the domain and is defined as:

$$\frac{d}{dt}x(\chi,t) = \hat{\mathbf{v}}$$

Hu et al. [16] discussed the significance of implementing the Space Conservation Law in the ALE formulation in appreciable detail.

Pan et al. [26] described a method of calculating the swept volumes while proposing a projection method for solving incompressible viscous flows in moving domains using an ALE formulation. Their work was primarily focused on Cartesian meshes, where they conducted the tetrahedral decomposition of the hexahedral swept volumes using the face centers and the cell centers as benchmark points. They connected all the vertices of the swept face at time instants t and $(t+1)$ to their respective face and cell centers [Figure 2.11]. Based on the results they obtained, they concluded that this method was consistent and highly accurate. Gopalakrishnan and Tafti [14] also used this formulation in implementing a parallel numerical solution procedure for simulating unsteady incompressible flows on a structured collocated multi-block mesh.

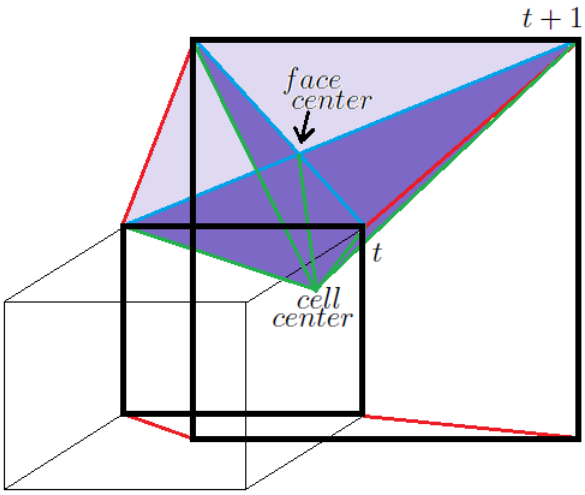


Figure 2.11: Calculation of swept volumes as implemented by Pan et al. [26]

Boffi and Gastaldi [4] reviewed the relationships between the constraint of the Space Conservation Law and the properties of accuracy, stability and convergence of different finite element schemes. They analyzed the application of time advancement schemes like Euler, Backward Differentiation and Crank-Nicolson. They demonstrated that in ALE formulations of some of these schemes, satisfying the Space Conservation Law is necessary. However, they also demonstrated that it is not a sufficient condition for maintaining the geometric integrity of the domain during mesh motion, unless the time-step is very small. Their work appears to discourage the use of ALE formulations for the simulation of fluid flows involving moving control volumes.

The views expressed by Boffi and Gastaldi [4] and Guillard and Farhat [15] contradict each other. Based on a similar observation Etienne et al. [11] termed the effect of the Space Conservation Law on the stability of ALE methods as "controversial". However, in partial agreement with the views of Boffi and Gastaldi [4], Etienne et al. [11] introduced an additional requirement that the accuracy of the time integrator should be linked to maintaining mesh accuracy.

After reviewing all the relevant publications, it can be concluded that implementation of the Space Conservation Law has received substantial attention in computa-

tional fluid dynamics since the latter half of the twentieth century. The implementations have evolved from one-dimensional applications to problems involving complex three-dimensional mesh movements. The importance of the Space Conservation Law and the errors arising due to its violation need no further emphasis.

A substantial number of authors have implemented the discrete form of the Space Conservation Law in order to address their particular selective set of issues. However, the options are limited when it comes to a generic algorithm for calculating the swept volumes in a polyhedral mesh structure. In addition to the generic nature of the algorithm, issues like consistency, accuracy and stability must also be addressed. We think that an ideal algorithm must be able to handle any form of three-dimensional mesh motion and restrict volume continuity errors to machine round-offs. In this work we analyzed the existing algorithm for implementation of the Space Conservation Law in OpenFOAM® and implemented an alternate algorithm which meets our expectations.

CHAPTER 3

ALGORITHMS FOR CALCULATION OF VOLUMES

Calculating swept volumes is the most critical part of balancing the space conservation equation in its discrete form. Our goal is to implement it in OpenFOAM®, so it is worthwhile to study the various algorithms implemented in OpenFOAM® for quantifying the magnitude of a three-dimensional space. In this chapter we will investigate the concepts and implementation that are currently used in OpenFOAM® for calculating cell volumes and swept volumes. There is very little official documentation explaining the mathematics implemented in OpenFOAM®. Therefore, this study was undertaken by examining the source code on a line-by-line basis.

Point locations are the building blocks of a topological hierarchy. They are the fundamental entities which can be defined in a Cartesian space as a set of three numbers. Each number specifies the distance of the point from the origin along the x, y and z coordinate axis. The unique connections between these points define edges whose unique mutual relationships define faces which connect to each other to form a volume [Figure 3.1]. In computational fluid dynamics, the data-structures pertaining to a mesh and its properties are fundamentally a list of volumes which exist at the top of a hierarchy of lists of faces, edges and points. The nomenclature *cell* will be used henceforth to refer to a single compartment of the three-dimensional dicretized domain. In a finite volume approach, each cell is treated as a control volume and its attributes are linked to the lists of the primitive entities which it comprises.

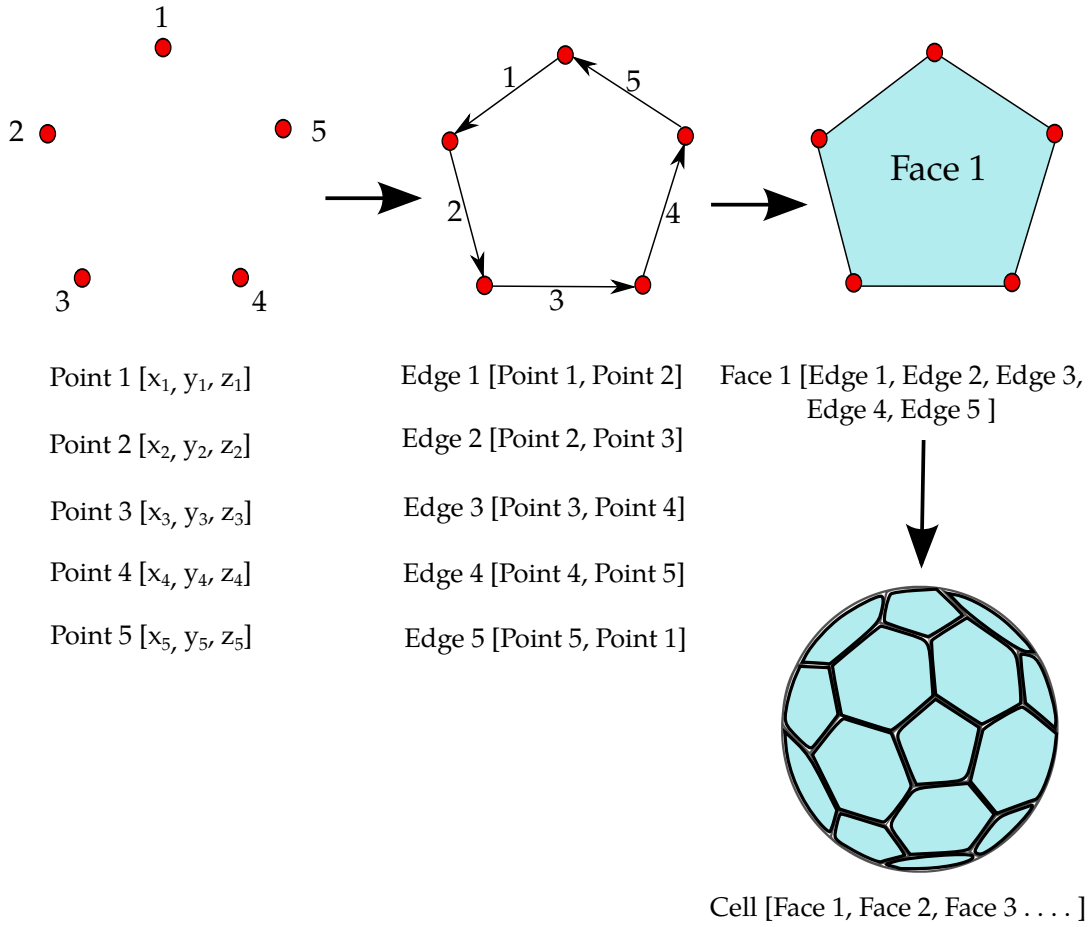


Figure 3.1: Data-structures required for defining a three-dimensional mesh.

3.1 Polyhedral mesh structure in OpenFOAM®

OpenFOAM® follows a polyhedral mesh format where every cell is primarily a polyhedron. There are no restrictions on the number of faces a cell can have and the concept extends to all mesh primitives as long as they form a closed volume. The sequence in which the points are indexed in the data-structure follows the right hand rule. For a set of points lying on the plane of this paper, the sequence orders them in a counter-clockwise direction so that from the reader's point of view the area normal vector of the face is directed out of the paper in an upward direction. Figure 3.2 illustrates this description.

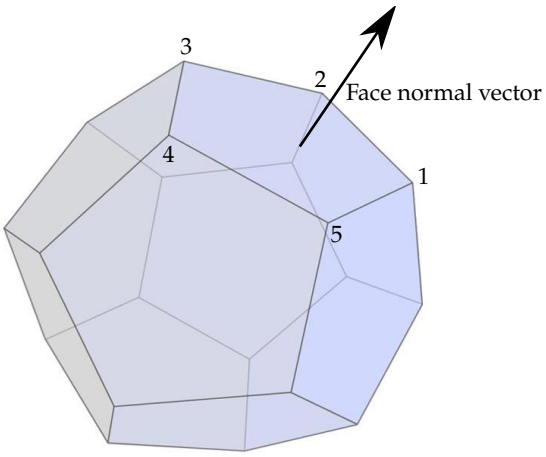


Figure 3.2: Indexing of point locations in OpenFOAM®.

The faces can be classified as internal faces and boundary faces. As the name suggests, the internal faces lie completely inside the domain whereas the boundary faces abut the domain boundaries. An internal face is shared by two neighboring cells. Therefore, in order to maintain consistency, the face is assigned to one of the participating cells. This cell is known as the *owner* and the other cell is known as the *neighbor*. The face normal vector projects in the outward direction with respect to the *owner* cell and vice-versa with respect to the *neighbor* cell. Therefore, as per this convention, the polyhedral cell illustrated in Figure 3.2 is an *owner*. In case the *owner* cell gets the face normal projecting inwards, it will be judged as a negative volume thereby causing an error due to segmentation fault.

The cells are the control volumes in which the values of flow variables are evaluated. Therefore it is necessary to uniquely identify each cell. In our case, this unique identification should also signify whether a cell is an *owner* or a *neighbor* with respect to a particular face. In order to ensure the integrity of the flow domain, each cell is identified with a label which is basically an index in the mesh data-structure. In OpenFOAM®, an *owner* cell is always designated as the lesser of the two neighboring cell indices.

Everything mentioned so far accounts for all the fundamental requirements which uniquely identify a three-dimensional control volume in Euclidean space. In addition to these entities, the mesh data-structure contains information about some additional features of the control volume. For each face a face center is calculated and for each cell a cell center is calculated. The calculation procedures are described later in this chapter. This information is tagged along with the mesh data-structures.

3.2 Calculation of cell volumes in OpenFOAM®

The process for calculating cell volumes in OpenFOAM® is built upon the aforementioned polyhedral mesh structure. The data-structures that define the mesh depend entirely on the list of point locations. The relations defined by the point locations consistently identify a particular cell and its unique properties. The functions used for calculating cell volumes have been designed to require only the lists of vertices that comprise the cell and the relationships among them. The hierarchical arrangement of mesh data-structures in OpenFOAM® ensures that the returned values are assigned to the right cell.

Each cell in the mesh undergoes a tetrahedral decomposition prior to its volume calculation. It must be noted that this process is similar to the work of Pan et. al. [26]. In a two-step process, the face center is first estimated as a simple average of the vertices comprising the face. The estimated face center is connected to the vertices which decomposes the face into triangles. The final face center is the weighted average of the centroids of the member triangles which are weighted using their respective face areas. This method is known as the weighted centroid method. The pseudo-code in Algorithm 3.1 paraphrases the source code available in the file *primitiveMeshFaceCentresAndAreas.C* of OpenFOAM-1.6-ext. It calculates face centers which are further used for calculating cell volumes.

Data: list of points, list of faces.
Result: face areas and face centers.
for *all faces* **do**
  if *number of points* $= 3$ **then**
    $face\ center\ = \left[\frac{\mathbf{p}_1+\mathbf{p}_2+\mathbf{p}_3}{3}\right]$;
    $face\ area\ = \left[\frac{(\mathbf{p}_2-\mathbf{p}_1)\times(\mathbf{p}_3-\mathbf{p}_1)}{2}\right]$;
  else
    $face\ center\ = \mathbf{p}_1$;
    for $i = 1 \rightarrow n$ ($n = number\ of\ vertices\ forming\ a\ face$) **do**
      $face\ center\ = face\ center + \mathbf{p}_i$;
    end
    $face\ center\ = \frac{face\ center}{n}$;
    for $i = 1 \rightarrow n$ **do**
      $next\ point = p\left[remainder\left(\frac{i+1}{n}\right)\right]$;
      $\mathbf{c} = \frac{\mathbf{p}_i+next\ point+face\ center}{3}$;
      $\mathbf{d} = (next\ point - \mathbf{p}_i) \times (face\ center - \mathbf{p}_i)$;
      $a = |\mathbf{d}|$;
      accumulate the values of $\mathbf{d}$, a and $(a.\mathbf{c})$
    end
    $New\ face\ center = \frac{1}{3}\frac{(a.\mathbf{c})_{accumulated}}{a_{accumulated}}$;
    $face\ area = \frac{\mathbf{d}_{accumulated}}{2}$;
  end
end

Algorithm 3.1: Calculation of face centers and face areas.

Consequently the cell centers are calculated as arithmetic means of the face centers followed by a recalculation using the weighted centroid method. The volume of the cell is calculated as a sum of the tetrahedrons formed by considering each triangulated cell face as the base and the cell center as the apex [Figure 3.3]. This method of volume calculation is widely used in the field of computational geometry for calculating volumes of irregular shapes. In the research done by Pan et. al [26] and Maric et. al. [23] it yielded accurate results.

In OpenFOAM®, there are numerous instances where the cell volumes are required and are therefore calculated in this way. The functions *mag(), V(), cellVolumes()* and the utility *checkMesh* are used for this purpose in order of increasing mesh hierarchy. The function *mag()*, for example, is the most basic form and takes mesh

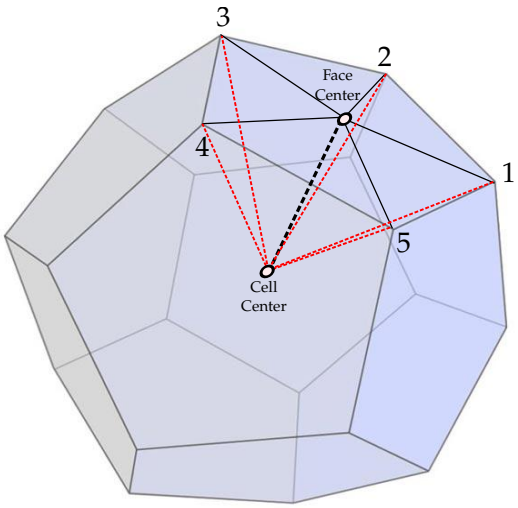


Figure 3.3: Calculation of cell volume using tetrahedral decomposition. (Picture adapted from Jasak [17])

primitives like face lists and point lists as arguments. Nevertheless, the basic concept for calculation remains the same for all of them. Table 3.1 lists the file locations of the definitions of these functions.

Table 3.1: Locations of the files containing the functions for calculating the cell volumes in OpenFOAM-1.6-ext.

Function	File location
mag()	src/OpenFOAM/meshes/meshShapes/cell/cell.C
V()	src/finiteVolume/fvMesh/fvMeshGeometry.C
CheckMesh	Applications/utilities/mesh/manipulation/checkMesh.C

3.3 Calculation of swept volumes in OpenFOAM®

A swept volume is defined as the three-dimensional space swept by the polygonal face of the control volume or the cell. Swept volumes appear only in moving mesh problems. In OpenFOAM®, the procedure for calculating swept volumes is somewhat different from the procedure for calculating cell volumes. The function $sweptVol()$ is used for this purpose. The sequence of operations during the mesh movement is

illustrated in Figure 3.4. For moving the mesh from its location at time t_0 to t_1, each face of the cell is triangulated and transformed to the new location without deleting the mesh location information at time t_0. Utilizing the location information of the triangulated mesh faces at two successive time instants, the function $sweptVol()$ calculates the volumes swept by the cell faces during the mesh movement. These values are used in balancing the Space Conservation Law.

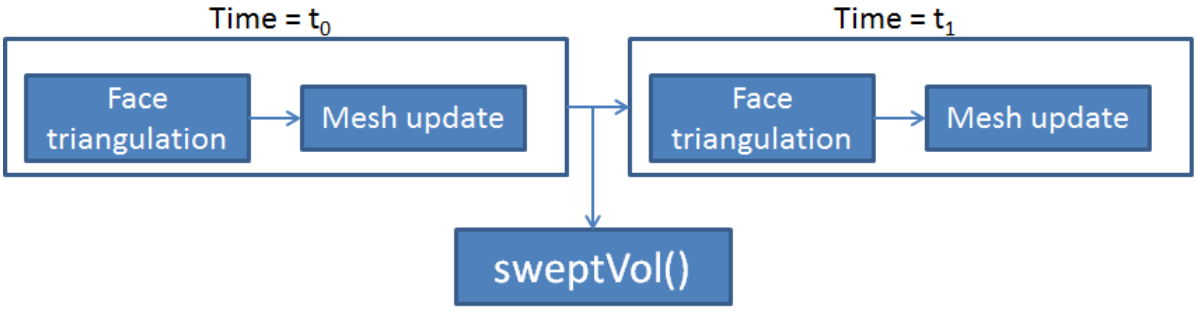


Figure 3.4: Work flow in dynamic mesh simulations.

The procedure used by the function $sweptVol()$ to calculate volume was implemented by Bos [5]. The volume swept by a triangular face forms a prism with a triangular base. The function $sweptVol()$ decomposes this prismatic volume into a set of three tetrahedrons [Figure 3.5]. The first tetrahedron is formed by connecting any one of the vertices at time t_1 to the two opposite vertices at time t_0. The remaining space in the prism will contain a full quadrilateral plane and the other two tetrahedrons can be realized by connecting one of its diagonals [Figure 3.6]. The volumes of all three tetrahedrons are summed up to obtain the volume of the prism. As observed by Bos [5], this decomposition can take place by connecting one diagonal at a time in two unique ways as illustrated in Figure 3.7. Hence the volume is calculated using both forms of decomposition and their arithmetic mean is returned as the swept volume by the function.

The pseudo-code in Algorithm 3.2 paraphrases the source code which defines the function $sweptVol()$. As noted earlier, the function only requires the locations of the

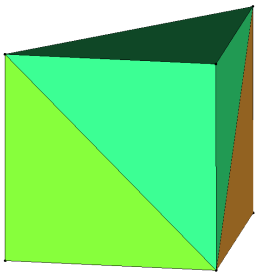


Figure 3.5: Tetrahedral decomposition of the prism.

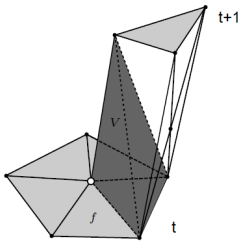


Figure 3.6: Creation of the first of the three tetrahedrons. (Picture reproduced from Bos [5])

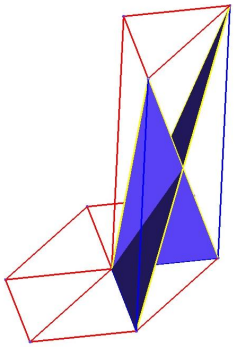


Figure 3.7: Two possible ways to uniquely decompose the remaining space (Picture adapted from Bos [5]).

Data: List of three vertices at time t and $(t+1)$.
Result: volume swept by each triangulated face.
for *each triangulated sweeping face* **do**

$$V_1 = (\mathbf{a}^{t+1} - \mathbf{a}^t).\left[(\mathbf{b}^t - \mathbf{a}^t) \times (\mathbf{c}^t - \mathbf{a}^t)\right]$$
$$+(\mathbf{b}^{t+1} - \mathbf{b}^t).\left[(\mathbf{c}^t - \mathbf{b}^t) \times (\mathbf{a}^{t+1} - \mathbf{b}^t)\right]$$
$$+(\mathbf{c}^t - \mathbf{c}^{t+1}).\left[(\mathbf{b}^{t+1} - \mathbf{c}^{t+1}) \times (\mathbf{a}^{t+1} - \mathbf{c}^{t+1})\right] ;$$

$$V_2 = (\mathbf{a}^{t+1} - \mathbf{a}^t).\left[(\mathbf{b}^t - \mathbf{a}^t) \times (\mathbf{c}^t - \mathbf{a}^t)\right]$$
$$+(\mathbf{b}^t - \mathbf{b}^{t+1}).\left[(\mathbf{a}^{t+1} - \mathbf{b}^{t+1}) \times (\mathbf{c}^{t+1} - \mathbf{b}^{t+1})\right]$$
$$+(\mathbf{c}^t - \mathbf{c}^{t+1}).\left[(\mathbf{b}^t - \mathbf{c}^{t+1}) \times (\mathbf{a}^{t+1} - \mathbf{c}^{t+1})\right] ;$$

$$\textit{swept volume} = \tfrac{1}{12}\left[V_1 + V_2\right];$$

end

Algorithm 3.2: Bos' [5] method for calculating swept volumes

points which constitute the swept volume. The variables $\mathbf{a}^t$, $\mathbf{b}^t$ and $\mathbf{c}^t$ in the code store the point locations at time t and the variables $\mathbf{a}^{t+1}$, $\mathbf{b}^{t+1}$ and $\mathbf{c}^{t+1}$ store the point locations at time $(t+1)$. A careful examination of the source code reveals the implementation of Equation (2.17).

In this chapter we provided a detailed description of the algorithms which are used to calculate of volumes in OpenFOAM®. The issues and solutions regarding the procedure of calculating swept volumes will be covered in the coming chapters.

CHAPTER 4

DEVELOPING AN ALTERNATE ALGORITHM

Any algorithm which implements the Space Conservation Law must calculate the swept volumes. It must perform accurately and consistently throughout the spacial and temporal regimes. In our work, we developed an alternate algorithm for achieving this objective in OpenFOAM®. The requirement for devising such an algorithm is driven by issues of consistency and accuracy surrounding the existing algorithm. In the following sections, we will discuss the characteristics and performance of the existing algorithm and describe the concept and implementation of our algorithm. The significance of our approach will be supported with data from the published literature, results from our own analysis and the fundamental concepts of computational geometry.

4.1 Current method for calculating swept volumes

As demonstrated in Equation (1.6), the conservation of space involves balancing the change in cell volume with the sum of the volumes swept by each of its faces during a particular time interval. In OpenFOAM®, cell volume is calculated using the function $V()$ of the $fvMesh$ class. The difference in the volume of a cell at subsequent points in time accounts for the change in cell volume. On the other hand, the swept volumes are calculated using the function $sweptVol()$. The functions $V()$ and $sweptVol()$ use different procedures for calculating the magnitude of a closed three-dimensional space.

The function $sweptVol()$ is customised for calculating the volumes of prisms which have a base in the shape of a triangle. Triangulation of all faces before the movement of the mesh is mandatory. This function decomposes the prism into three tetrahedrons. The programming logic used for decomposing and calculating the swept volume has already been discussed in Chapter 3. The tetrahedral decomposition process which is used in the function $sweptVol()$ carries the decomposition in two different ways and returns the average of volumes which were calculated using each approach [Equation (2.17)]. As illustrated in Figure 2.10, the function connects the face center at time $(t+1)$ to the vertices of the face at time t, while the remaining space is decomposed in the two possible ways as shown in Figure 3.7.

The procedure followed by the function $sweptVol()$ to calculate swept volumes can be done by connecting any single vertex of the triangular face at time $t+1$ with the opposite vertices of the same face at time t in order to realize the first tetrahedron. The remaining space can then be decomposed into two tetrahedrons in two different ways by connecting the diagonals of the quadrilateral face available in that space. The sequence in which the three vertices of each triangle are arranged determines the pattern in which the tetrahedral decomposition will take place. In particular, the sequence followed by the function $sweptVol()$ considers the face center at time $(t+1)$ as the first vertex. The other two vertices are chosen on the face at time t using the Right Hand Rule from the face center.

It has been proved by Zhang et al. [32] that the sequence in which we consider the vertices of each participating triangle plays a vital role in ascertaining the swept volume. They also found that the value of the swept volume that is calculated by using one sequence can be different from the value calculated using another sequence. This led them to consider all the permutations in which the vertices of a triangle can be sequenced for faces at time $t+1$ and t. A set of three points can be arranged in six different permutations. For example, a set of points $[1, 2, 3]$ can be sequenced as

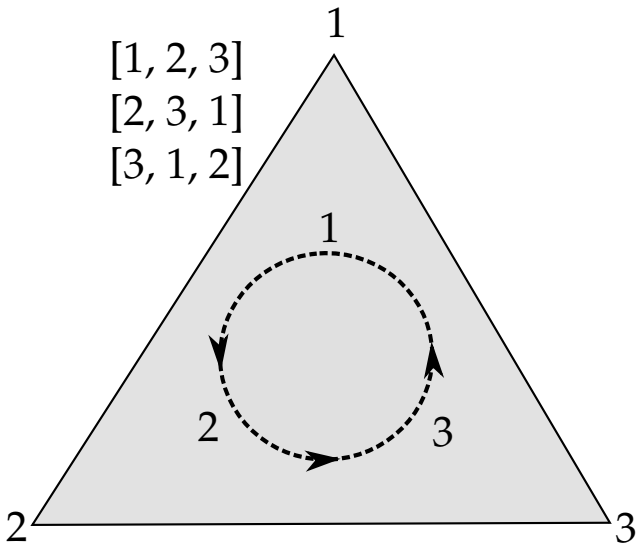


Figure 4.1: Possible ways of sequencing the vertices of a triangle using cyclic permutations.

$[1, 2, 3]$, $[2, 3, 1]$, $[3, 1, 2]$, $[1, 3, 2]$, $[2, 1, 3]$ and $[3, 2, 1]$. Each permutation is capable of generating two sets of tetrahedral decompositions. In their research, Zhang et al. [32] considered all possibilities to derive the final expression for calculating swept volumes.

Due to the form of data-structures and labelling conventions, we can only consider cyclic permutations, as illustrated in Figure (4.1). This leaves us with three different sequences in which the vertices of each triangular face of the prism can be considered. This description provides six different ways in which the tetrahedral decomposition can be processed. Figure 4.2 illustrates all of the approaches to decompose the triangular prism. However, the function $sweptVol()$ uses only one among these six ways.

In comparison to $sweptVol()$, the procedure followed for calculating the volume by the function $V()$ appears extremely robust because it uniquely identifies the attributes of each geometric entity. Furthermore, it provides a unique pattern of tetrahedral decomposition for each polyhedral cell. The algorithm implicitly provides a consistent method of decomposing internal faces which are shared by two cells. The face center is an exclusive property of the face which is determined from primitive

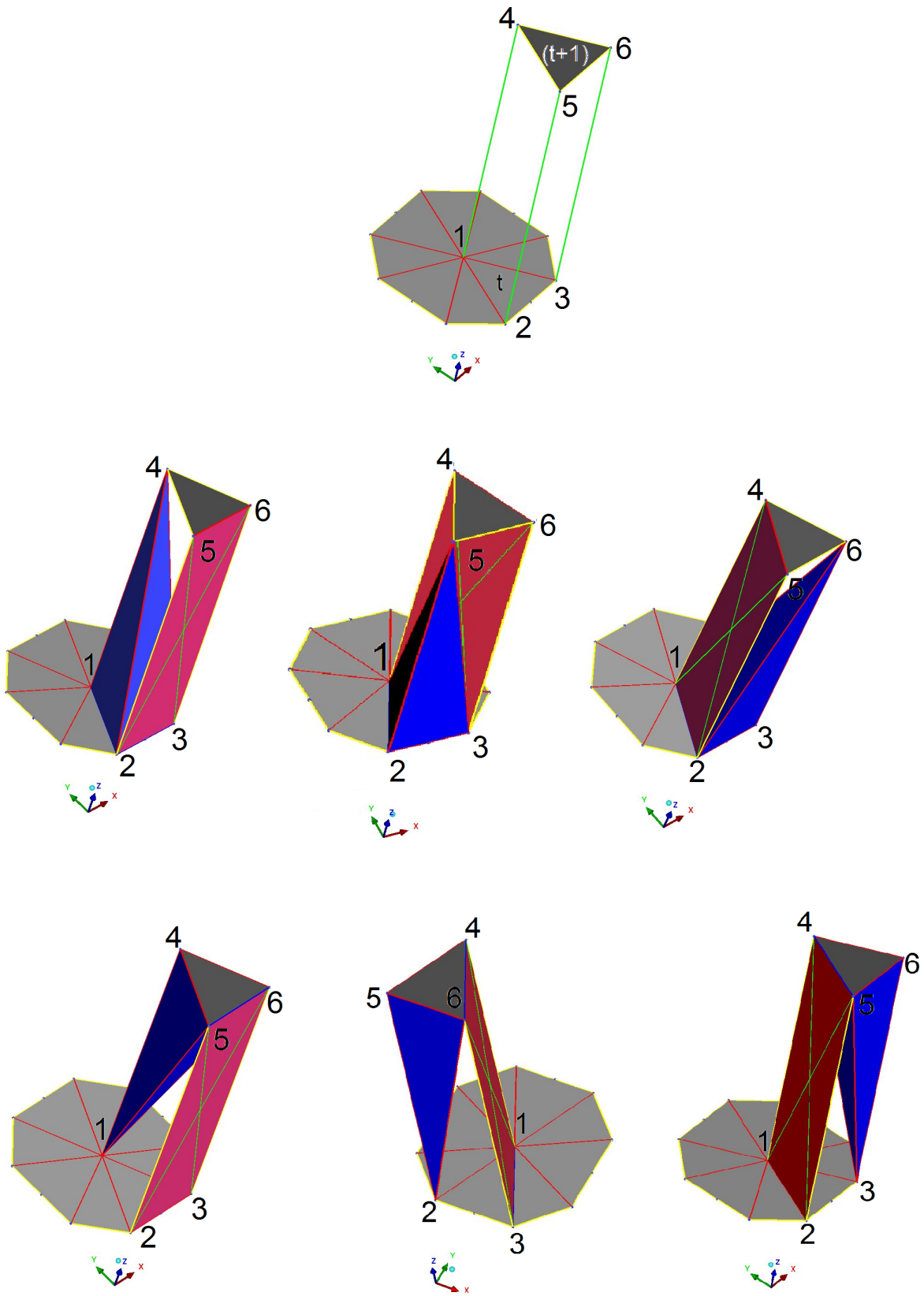


Figure 4.2: All possible ways of tetrahedral decomposition for a triangular prism.

entities, i.e. face vertices. The face centers are therefore consistent and unique for each face of the mesh. These properties also extend to cell centers. This eliminates the need to consider different permutations, as the vertices can be arranged in only one unique sequence in-line with the definitions of the *owner* and *neighbor* cells of a face. Therefore, the need for averaging the values obtained from different arrangements becomes redundant. From this perspective, the method used in the function $sweptVol()$ presents limitations in terms of accuracy, consistency and uniqueness.

4.1.1 Testing $sweptVol()$ function

We conducted a series of numerical experiments for testing the capabilities of the function $sweptVol()$. In these experiments, we created a prism which has a base in the shape of a triangle as shown in Figure 4.3. While keeping the orientation

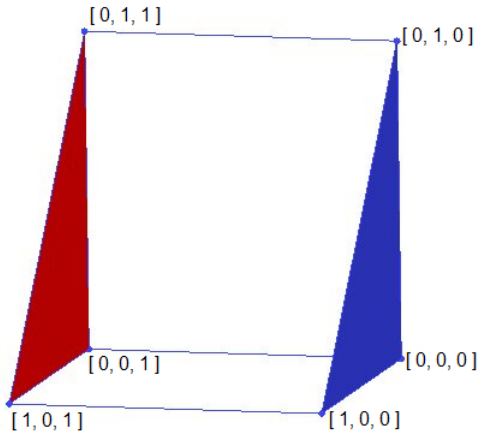


Figure 4.3: Initial geometry of the test case.

of the blue triangular face intact, we modified the orientation of the red triangular face in twenty-four different ways using all six degrees of freedom. Effectively, we are calculating the volumes swept by the blue face during complex three-dimensional mesh motions. One of the series of change in orientation of the red face is shown in Figure 4.4. Here, the red face is translating along the z-coordinate axis and rotating about an axis which has a direction vector along the x-coordinate axis. The rotation

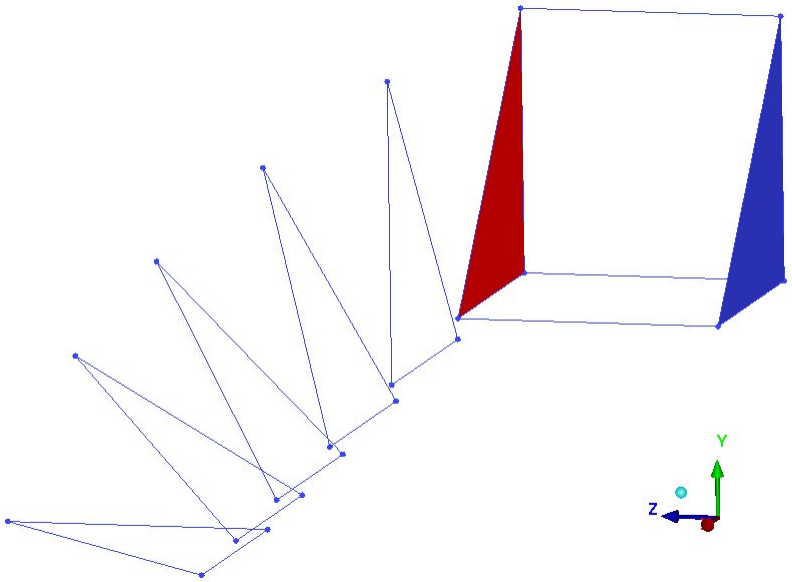


Figure 4.4: Changes in orientation of the red face due to rotation about the X-axis and translation along the Z-axis.

and translation are taking place simultaneously. The red face was rotated to form a series of shapes that ranged from simple to complex. The chart shown in Figure 4.5 provides a summarized description of the transformations underwent by the red face. For a unit translation along each direction, it was rotated from 0 to 90 degrees in six intervals in the counter-clockwise direction from the viewpoint of the origin. All the axes of rotations passed through the origin. Under each arrangement, a cell was built and its volume was calculated using the functions $sweptVol()$ and $V()$. The data were tested up to a precision of nineteen decimal places. The experiments were conducted only for the combinations which are marked with black dots in Figure 4.5. The combinations marked by red dots are extremely simple. Thus, the possibility of a discrepancy in the values returned by the two functions is negligible. The results obtained from three of these combinations are illustrated in Figures 4.6 through 4.8. A comprehensive list of results from all twenty-four cases is available in Appendix B.

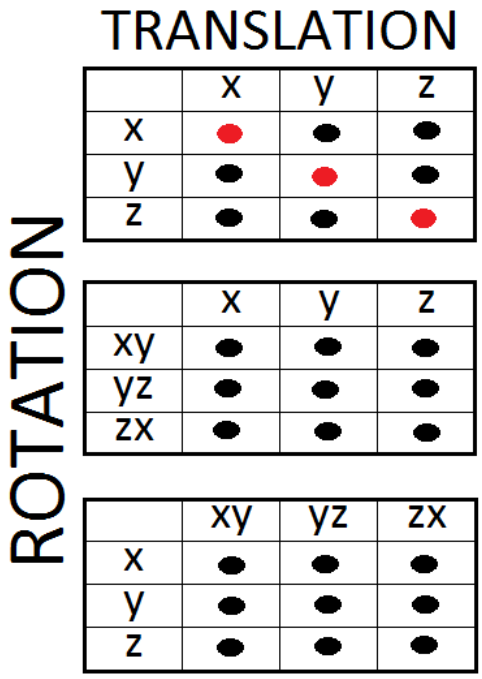
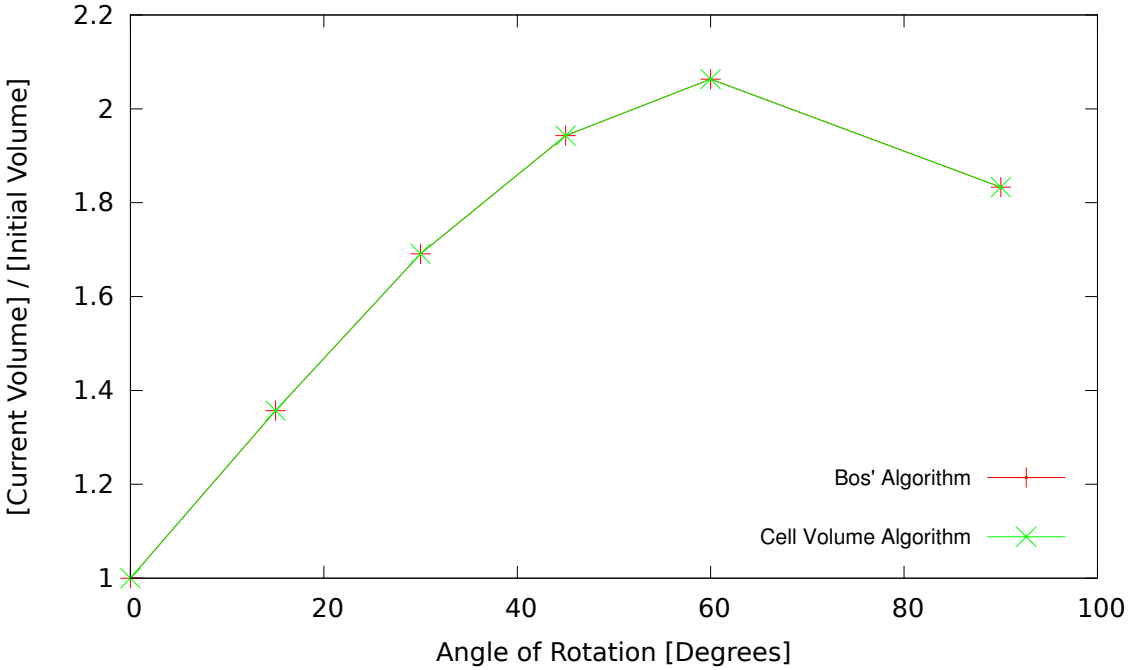


Figure 4.5: Combinations of translational and rotational motions used for generating complicated shapes.

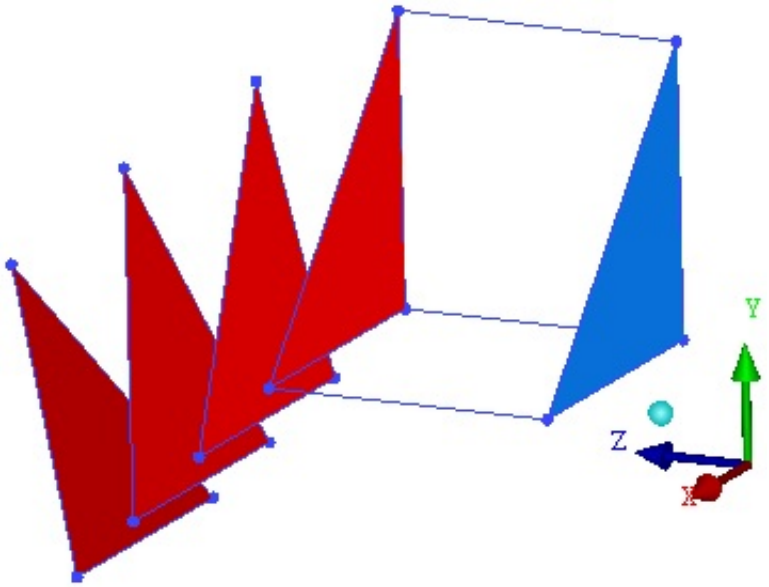
4.1.2 Observations

The volumes obtained in fifteen of the twenty-four experimental sets demonstrate a disagreement in the values which were calculated by the functions $sweptVol()$ and $V()$. In an ideal case, both of these functions are expected to return identical values for volume because they are defined by the same set of vertices. Considering our analysis so far, we assume that the values returned by the function $V()$ are correct at this point. This makes the results obtained from the function $sweptVol()$ erroneous in 62.5% of the cases in this sample set of experiments.

The magnitude of errors however changes from case to case. These experiments were conducted for calculating the volumes swept by a single triangulated face, so the errors will accumulate with each movement of each face of the mesh. This will eventually introduce a substantial amount of purely numerical mass into the control volume.

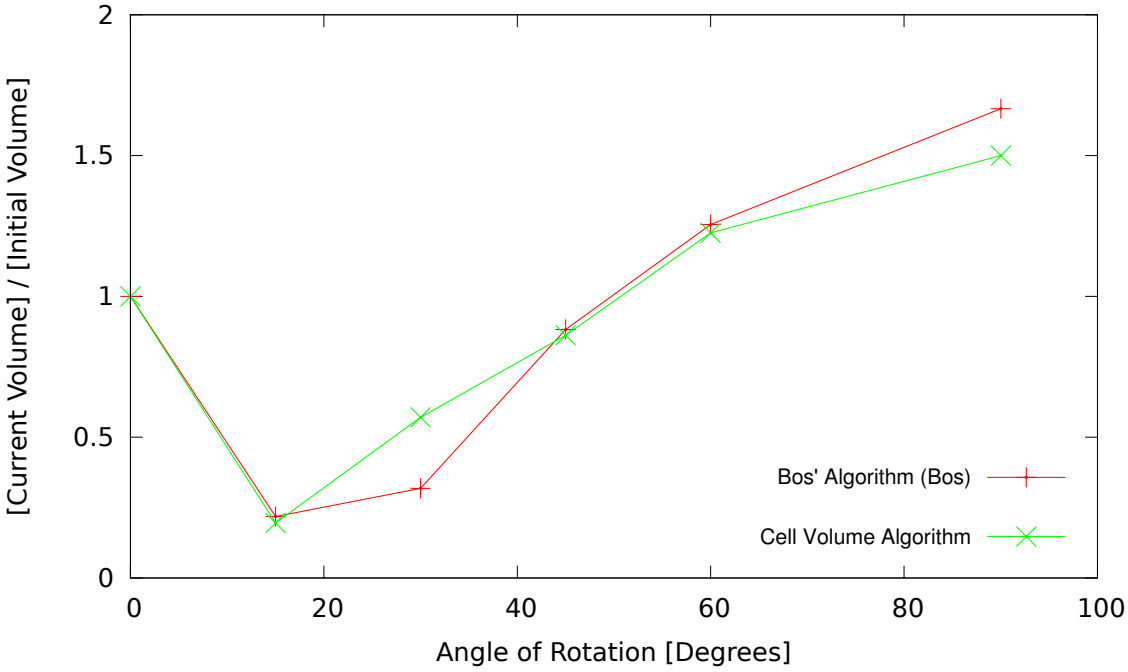


(a)

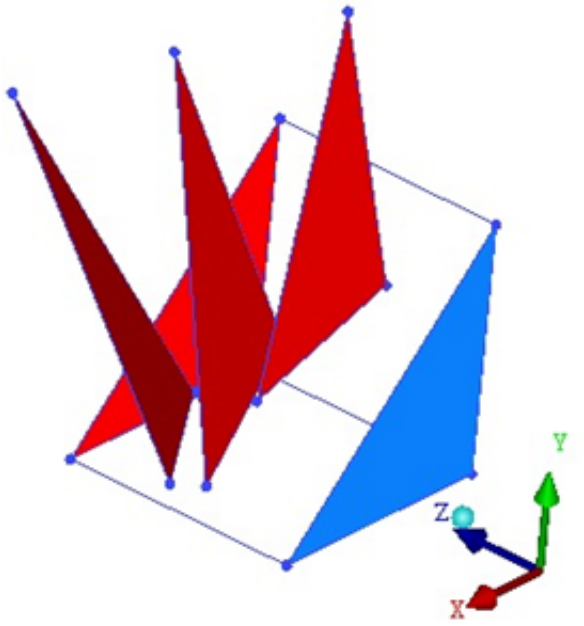


(b)

Figure 4.6: Rotation about the X-axis with translation along the Z-axis.

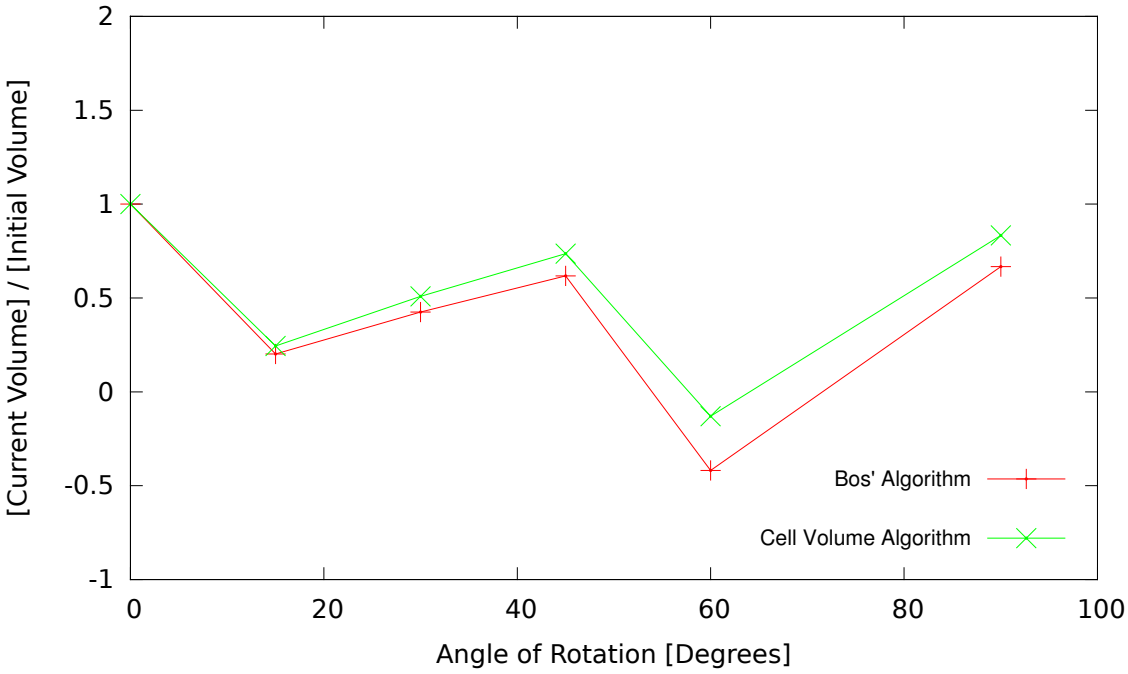


(a)

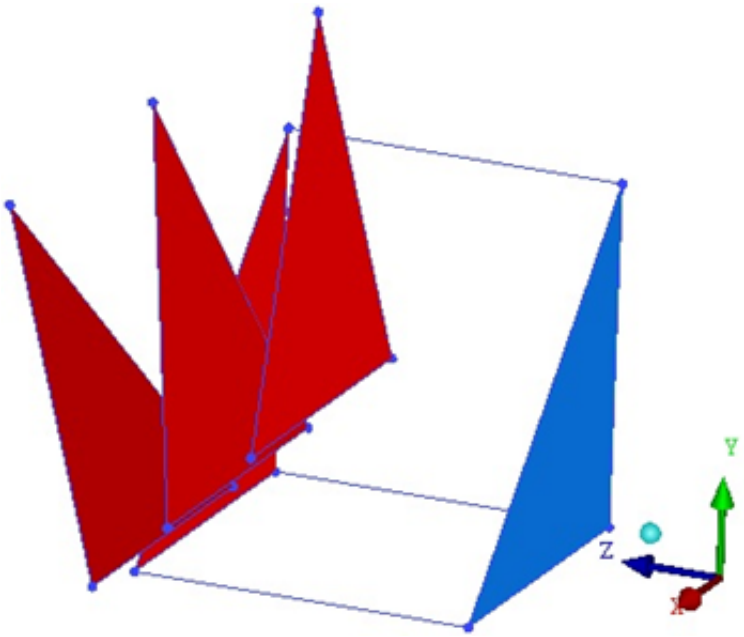


(b)

Figure 4.7: Rotation about the X and Y-axis simultaneously with translation along the Y-axis.



(a)



(b)

Figure 4.8: Rotation about the X-axis with translation along the X and Y-axis simultaneously.

4.2 A new perspective towards calculating swept volumes

The functions $sweptVol()$ and $V()$ perform the same job in two different ways. However, when we assigned the exact same task to both of these functions, they did not give the same result in most cases. From a broader perspective, this behavior brings to light our concerns regarding the consistency in OpenFOAM®. There are also some issues of accuracy but they are not as conspicuous when compared to the issues of consistency and coherence. Therefore, using the function $V()$ for calculating swept volumes can address the problem and build the required consistency into the OpenFOAM® architecture.

4.2.1 Software development

Calling the function $V()$ to calculate swept volumes is a challenging task. One of the fundamental reasons for this is that $V()$ resides at a higher level in the OpenFOAM® hierarchy, whereas $sweptVol()$ is defined at a rudimentary level. Furthermore, we have to make sure that such a call does not create any ripple effect on the general working of the software. We need to systematically understand the work flow stipulated by the existing source code prior to making any changes. The pseudo-code in Algorithm 4.1 paraphrases the definition of function $V()$ in OpenFOAM-1.6-ext. The function $V()$ calls another function $cellVolumes()$ for calculating the cell vol-

Data: List of points, list of faces.
Result: Cell volumes.
for *All cells* **do**
| $function\ call\ to\ cellVolumes()$
end

Algorithm 4.1: Cell volume calculation algorithm.

umes. $cellVolumes()$ further calls the function $calcCellCentresAndVols()$ for this task. The pseudo-code in Algorithm 4.2 paraphrases the definition for *calcCellCentresAndVols()*.

Data: List of points, list of faces, list of face centers.
Result: Cell volumes and cell centers.
$f \Rightarrow no.\ of\ cell\ faces,\ t \Rightarrow no.\ of\ tetrahedrons,\ \mathbf{c}_{tet} \Rightarrow tetrahedron\ centroid.$
for *All mesh domains* **do**

- *Initialize fields for estimated cell centers, actual cell centers and cell volumes.*

label-list own $= faceOwner()$; *label-list nei* $= faceNeighbor()$;
for *All owner cells* **do**
$cEst[i] = \sum_{j=1}^{f}(face\ Center)_j;$ $\quad \backslash\backslash\ cEst \Rightarrow Estimated\ cell\ center$
end

- *Repeat the above loop for all neighbor cells.*

for *All cEst* **do**
$cEst[i] = \frac{cEst_i}{Number\ of\ faces\ in\ the\ cell\ i};$
end
for *All owner cells* **do**

- *Create a label-list of all faces for each cell.*

if *the face is triangle* **then**
$V_{tet} = vol.\ of\ tetrahedron\ from(fv_1, fv_2, fv_3, cEst[i]);$
$\backslash\backslash\ fv_n \Rightarrow (face\ vertex)_n\ where\ (n = 1, 2, 3)$
$cellCtrs[i] = \sum_{u=1}^{t}(V_{tet}.\mathbf{c}_{tet})_u;$ $\quad cellVols[i] = \sum_{u=1}^{t}(V_{tet})_u;$
else
$V_{tet} = vol.\ of\ tetrahedron\ from(fv_1, fv_2, face\ Center, cEst[i]);$
$cellCtrs[i] = \sum_{u=1}^{t}(V_{tet}.\mathbf{c}_{tet})_u;$ $\quad cellVols[i] = \sum_{u=1}^{t}(V_{tet})_u;$
end
end

- *Repeat the above loop for all neighbor cells.*

for *All cells* **do**
$cellCtrs[i] = \frac{cellCtrs[i]}{cellVols[i]}$
end

- *Calculate cell volume by tetrahedral decomposition using the new cell center.*

end

Algorithm 4.2: Calculating cell centers and cell volumes.

It can be seen that the function $calcCellCentresAndVols()$ is using the functions $faceOwner()$ and $faceNeighbour()$ for accessing the constituent faces of each cell. These functions also provide information about other linked data sets like face centers. The file $primitiveMeshCellCentresAndVols.C$, where the function $calcCellCentresAndVols()$ is defined, does not posses the definitions for the functions $faceOwner()$ and $faceNeighbor()$. These functions belong to the $polyMesh$ class. Their definitions refer to the mesh directories for the requisite information. For calculating cell volumes the function $V()$ is called as a member of the $mesh$ class $[mesh.V()]$.

Moving on, the cell center is first estimated as the arithmetic mean of centers of all the faces which constitute the cell. The cell is then decomposed into tetrahedrons by using the center of a face and two of its vertices as points to form the base triangle and the estimated cell center as the apex [Figure 3.3]. Then the centroid of each tetrahedron is calculated. The cell center is then recalculated as the weighted average of the centroids of all the tetrahedrons which constitute the cell, which are weighted using their respective tetrahedral volumes. The tetrahedral decomposition is then readjusted to the new cell center. The sum of the volumes of all tetrahedrons in the cell at this stage is returned as the cell volume.

As we mentioned earlier, the function $sweptVol()$ resides at a rudimentary level in the OpenFOAM® hierarchy. The inheritance architecture stops us from accessing mesh directories at this level. Therefore, in our understanding, the simplest solution is to replicate the functioning of $V()$ in the function $sweptVol()$.

4.2.2 Replicating $V()$

The function $sweptVol()$ is given a set of six point vectors. The first three are the vertices of the triangulated control surface of a cell at time-step t. The next three are the locations of the same triangle at time-step $t + 1$. The function $V()$ is defined for

calculating the volume of a cell. Therefore, our first task is to create a cell using the six available point vectors. We define these point vectors as $\mathbf{a}^t$, $\mathbf{b}^t$, $\mathbf{c}^t$, $\mathbf{a}^{t+1}$, $\mathbf{b}^{t+1}$ and $\mathbf{c}^{t+1}$.

As per OpenFOAM® architecture, a cell is a list of face indices. Hereafter, a list of indices will be referred to as a label-list. Thus, a cell is a label-list of faces and a face is a label-list of points. The order of entries in the label-list of a face defines the connectivity between points, which defines the edges. Prior to realizing the cell, we need to define faces. The pseudo-code in Algorithm 4.3 paraphrases the procedure to define faces in OpenFOAM-1.6-ext. In this pseudo-code, we first

Data: List of points.
Result: Face definitions.

$pointField\ p[6];$

$p[0] = \mathbf{a}^t;\ \ p[1] = \mathbf{b}^t;\ \ p[2] = \mathbf{c}^t;$

$p[3] = \mathbf{a}^{t+1};\ \ p[4] = \mathbf{b}^{t+1};\ \ p[5] = \mathbf{c}^{t+1};$

$faceList\ pf[5];$ $\quad \backslash\backslash\ Label - list\ of\ faces$

for *Each face (i) of the prism* **do**

$\quad pf[i] = face(4); \backslash\backslash Assigning\ four\ vertices\ to\ each\ face$

end

Algorithm 4.3: Creating point lists and face list.

created an array of point vectors called p. Then we assigned $\mathbf{a}^t$, $\mathbf{b}^t$, $\mathbf{c}^t$, $\mathbf{a}^{t+1}$, $\mathbf{b}^{t+1}$ and $\mathbf{c}^{t+1}$ to successive memory locations in p. Then we created a list of five faces, since a prism with a triangular base consists of five faces. Each entry in the face list is then linked to a label-list of four entries. Effectively, this means that we are creating five faces and each of those five faces have four constituent vertices. The pseudo-code in Algorithm 4.4 illustrates the procedure which defines the point connectivity for each face. It must be noted that in our cell, we have three faces with four vertices and two faces with three vertices. However, in order to accommodate all the faces into a single list we defined four vertices in each face and then superimposed two of

Data: Empty face list, populated point list.
Result: Populated face list.

$pf[face0][vertex0] = 0; \quad pf[face0][vertex1] = 1;$
$pf[face0][vertex2] = 2; \quad pf[face0][vertex3] = 0;$

$pf[face1][vertex0] = 3; \quad pf[face1][vertex1] = 5;$
$pf[face1][vertex2] = 4; \quad pf[face1][vertex3] = 3;$

$pf[face2][vertex0] = 0; \quad pf[face2][vertex1] = 2;$
$pf[face2][vertex2] = 5; \quad pf[face2][vertex3] = 3;$

$pf[face3][vertex0] = 1; \quad pf[face3][vertex1] = 4;$
$pf[face3][vertex2] = 5; \quad pf[face3][vertex3] = 2;$

$pf[face4][vertex0] = 0; \quad pf[face4][vertex1] = 3;$
$pf[face4][vertex2] = 4; \quad pf[face4][vertex3] = 1;$

Algorithm 4.4: Defining point connectivity in order to realize faces.

the vertices to realize the triangular face [Figure 4.9]. This is a standard practice in OpenFOAM® [25] mesh structure. While defining the faces, we also needed to decide

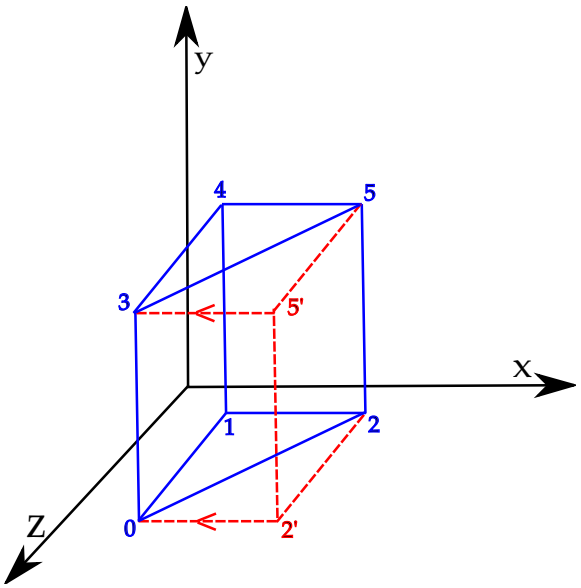


Figure 4.9: Superimposing vertices 2' onto 2 and 5' onto 5.

whether our final product would be an *owner* cell or a *neighbor* cell. This led us to investigate how OpenFOAM® would treat a single-cell mesh. As per our findings, and as per OpenFOAM® mesh data-structures, in such a mesh the only available cell

is designated as an *owner* cell. Therefore, we connected the points for each face as per the Right-Hand Rule so that all the face normal vectors project in the outward direction with respect to the cell [Figure 4.10].

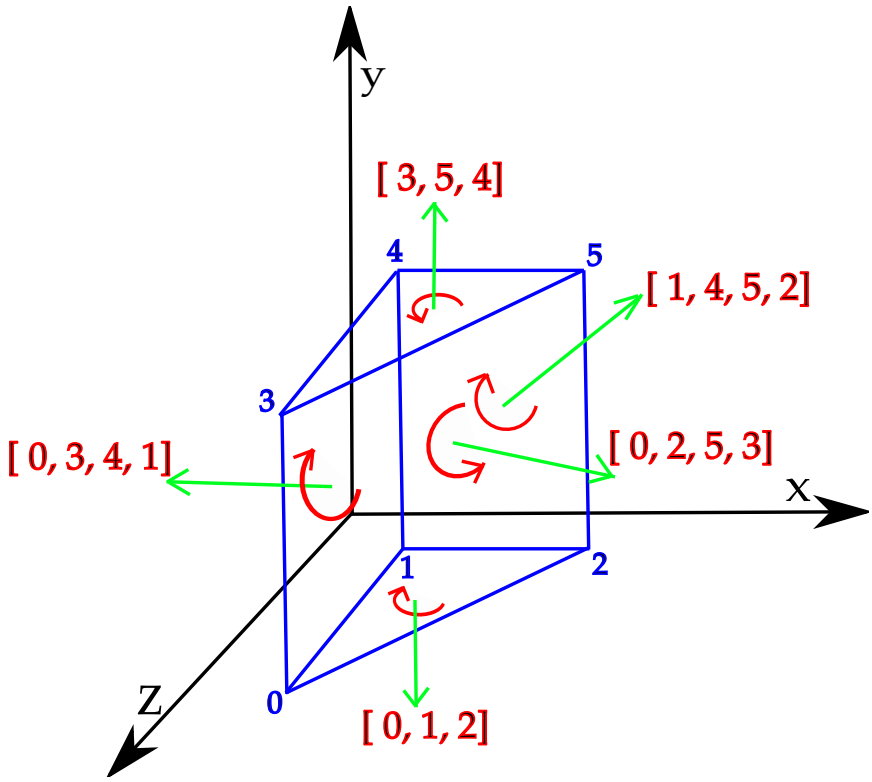


Figure 4.10: Arranging point vectors as per Right-Hand Rule in order to define an *owner* cell.

Once we are done defining faces, the next task is to define a cell. This is illustrated in the pseudo-code in Algorithm 4.5. It shows that a cell is nothing but a label-list of faces. The *cell* is a function template defined in OpenFOAM®. It takes the label-list of faces as an argument to realize a closed three-dimensional control volume.

The next task is to calculate the cell center. In order to keep the code as simple as possible, it is mandatory to use existing functions whenever they are accessible. The cell center was calculated using one such function available in the file *cell.C*. It takes the face list and corresponding point field as arguments and calculates the

Data: List of Faces.
Result: Definition of a cell.

label-list pC[number of faces in the cell];

for *i = 0 to No. of faces* **do**

    pC[i] = i;

end

cell cellName(label-list of faces);

Algorithm 4.5: Defining a cell.

cell center using a weighted centroid method. The face centers are calculated with a similar function which is available in the file $face.C$ [Algorithm 4.6].

Data: Face list, point fields.
Result: Cell center and face center.

$cellCenter = cellName.centre(pointField, faceList);$

$faceCenter0 = face0.centre(pointField\ of\ face0)$

$faceCenter1 = face1.centre(pointField\ of\ face1)$

$faceCenter2 = face2.centre(pointField\ of\ face2)$

$faceCenter3 = face3.centre(pointField\ of\ face3)$

$faceCenter4 = face4.centre(pointField\ of\ face4)$

Algorithm 4.6: Calculating cell centers and face centers

We can carry out the tetrehedral decomposition with this data. The decomposition and volume calculation for each tetrahedron was carried out using the function template $tetrahedron < Point, PointRef > (fv_{tet}).mag()$. The pseudo-code in Algorithm 4.7 illustrates the procedure for cell volume calculation.

The first three arguments in the function (fv_{tet}) are the vertices of the base triangle of each tetrahedron and the fourth argument is the apex. Defining the base triangle for each tetrahedron requires a meticulous sequencing of constituent point vectors. Here, if the face is a triangle we can use its vertices for the base triangle of the tetrahedron. For higher order shapes the first point of the base triangle is the face

Data: List of points, list of faces.
Result: Cell volumes and cell centers.

$$tet0 = tetrahedron < Point, PointRef > (\mathbf{a}^t, \mathbf{b}^t, \mathbf{c}^t, cC).mag()$$

$$tet1 = tetrahedron < Point, PointRef > (\mathbf{a}^{t+1}, \mathbf{c}^{t+1}, \mathbf{b}^{t+1}, cC).mag()$$

$$\begin{aligned} tet2 = {} & tetrahedron < Point, PointRef > (fC_2, \mathbf{a}^t, \mathbf{c}^t, cC).mag())+ \\ & tetrahedron < Point, PointRef > (fC_2, \mathbf{c}^t, \mathbf{c}^{t+1}, cC).mag())+ \\ & tetrahedron < Point, PointRef > (fC_2, \mathbf{c}^{t+1}, \mathbf{a}^{t+1}, cC).mag()+ \\ & tetrahedron < Point, PointRef > (fC_2, \mathbf{a}^{t+1}, \mathbf{a}^t, cC).mag(); \end{aligned}$$

$$\begin{aligned} tet3 = {} & tetrahedron < Point, PointRef > (fC_3, \mathbf{b}^t, \mathbf{b}^{t+1}, cC).mag()+ \\ & tetrahedron < Point, PointRef > (fC_3, \mathbf{b}^{t+1}, \mathbf{c}^{t+1}, cC).mag()+ \\ & tetrahedron < Point, PointRef > (fC_3, \mathbf{c}^{t+1}, \mathbf{c}^t, cC).mag()+ \\ & tetrahedron < Point, PointRef > (fC_3, \mathbf{c}^t, \mathbf{b}^t, cC).mag(); \end{aligned}$$

$$\begin{aligned} tet4 = {} & tetrahedron < Point, PointRef > (fC_4, \mathbf{a}^t, \mathbf{a}^{t+1}, cC).mag()+ \\ & tetrahedron < Point, PointRef > (fC_4, \mathbf{a}^{t+1}, \mathbf{b}^{t+1}, cC).mag()+ \\ & tetrahedron < Point, PointRef > (fC_4, \mathbf{b}^{t+1}, \mathbf{b}^t, cC).mag()+ \\ & tetrahedron < Point, PointRef > (fC_4, \mathbf{b}^t, \mathbf{a}^t, cC).mag(); \end{aligned}$$

$$cellVolume = tet0 + tet1 + tet2 + tet3 + tet4$$

$\backslash\backslash\ cC \ \Rightarrow\ cell\ center$

$\backslash\backslash\ fC_n \ \Rightarrow\ face\ center\ of\ face\ n\ where\ (n = 2, 3, 4)$

Algorithm 4.7: Calculating cell volume by tetrahedral decomposition.

center and the remaining two points are selected as per the Right-Hand Rule. In our implementation the area normal vector of the triangle is parallel to its parent face [Figure 4.11]. A prism with base in the shape of triangle, when discretized in this manner, realizes a total of fourteen tetrahedrons. The sum of the volumes of all these tetrahedrons is returned as the swept volume.

4.2.3 Testing the new algorithm

We repeated our sample set of experiments [Figure 4.5] with the above mentioned procedure for calculating swept volume. We further scrutinized our results by comparing them with the ones obtained using Perot and Nallapati's [27] method. Some of the results are illustrated in Figures 4.12 through 4.14. A comprehensive set of results from all the experiments is available in Appendix B.

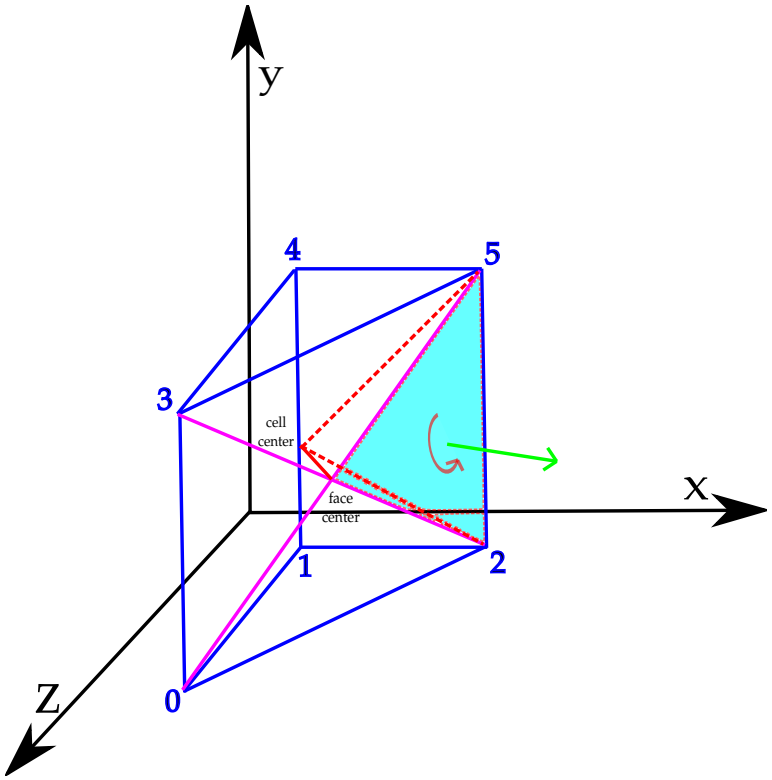


Figure 4.11: Defining point connectivity for tetrahedral decomposition

4.2.4 Observations

We observed that all the algorithms produced exactly the same results for simple mesh movements. But, as the movements increased in complexity the results from the function $sweptVol()$ deviated from the rest in its existing form. On the other hand, the results from our alternate approach matched with the results produced from the function $V()$ and Perot and Nallapati's [27] method exactly. This proves that our approach is accurate as well as consistent with respect to similar algorithms. With the new approach, the functions $sweptVol()$ and $V()$ will be calculating the volumes in exactly same manner. Therefore, this new implementation is a potential solution to our concerns regarding the consistency of dynamic mesh simulations in particular and the coherence of the OpenFOAM® architechture in general. We conducted further testing and validation of our algorithm to characterize its performance in real time for

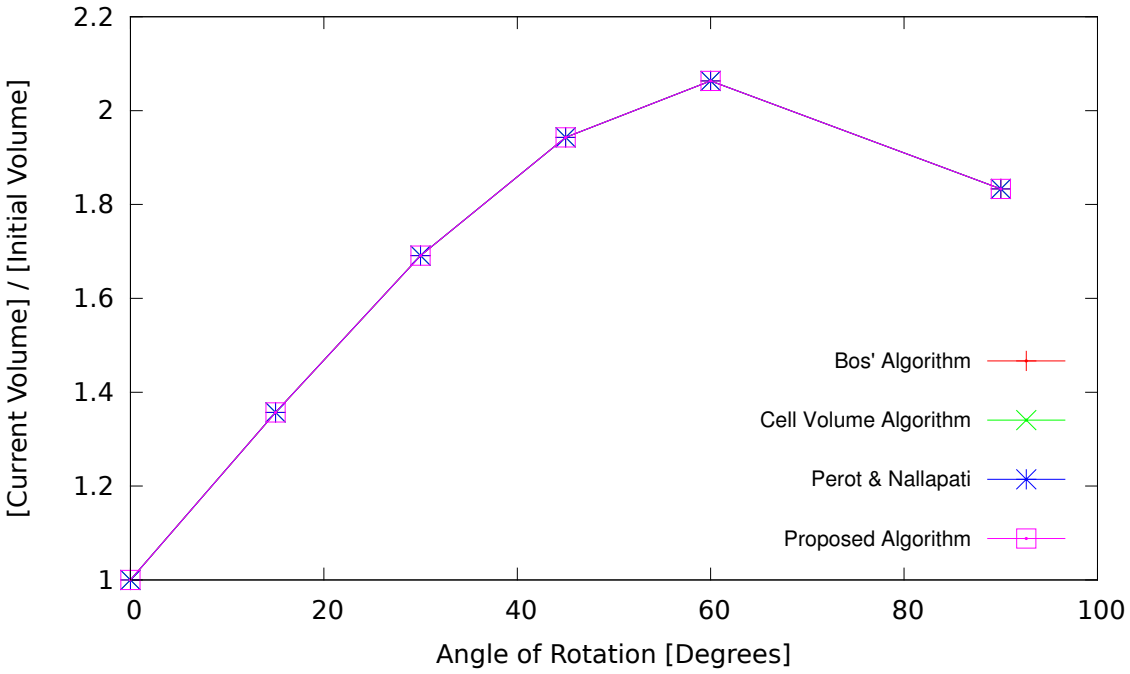


Figure 4.12: Rotation about the X -axis with translation along the Z-axis. (Results corresponding to Figure 4.6(b)).

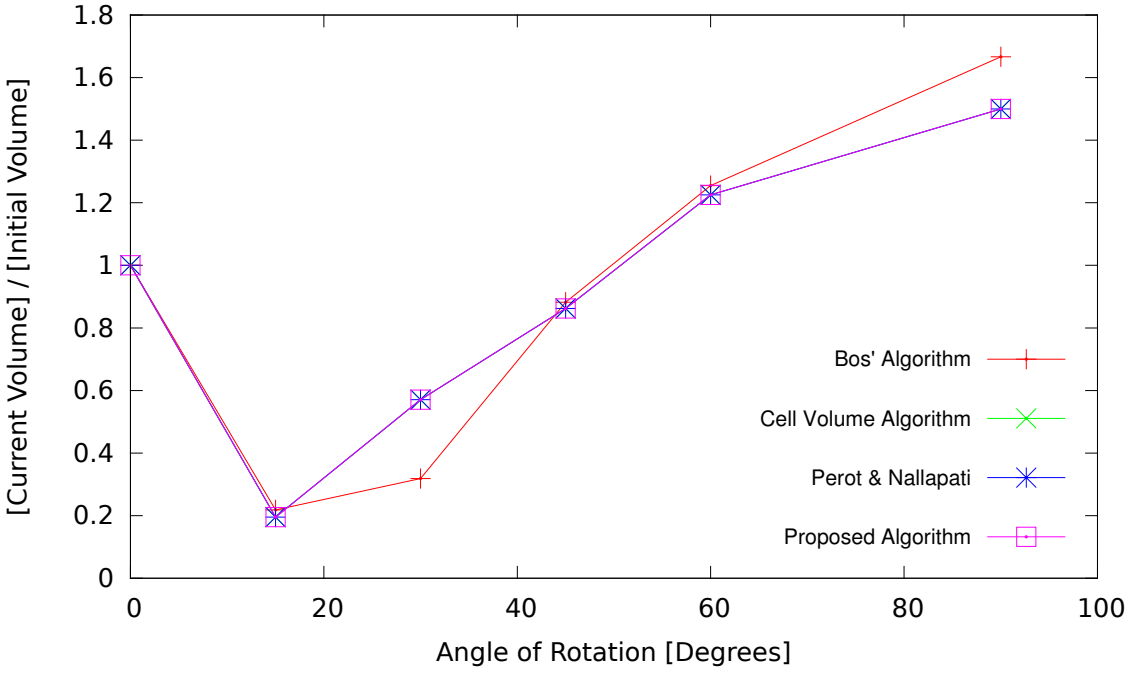


Figure 4.13: Rotation about the X and Y-axis simultaneously with translation along the Y-axis. (Results corresponding to Figure 4.7(b)).

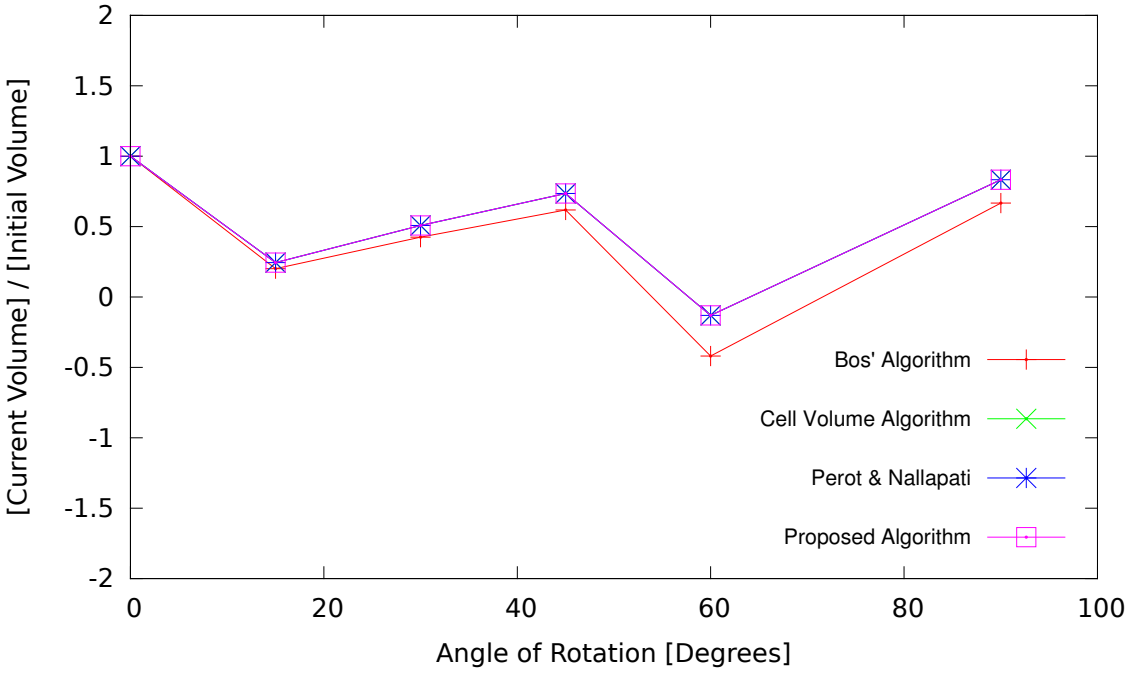


Figure 4.14: Rotation about the X-axis with translation along the X and Y-axis simultaneously (Results corresponding to Figure 4.8(b)).

elaborate dynamic mesh simulations in OpenFOAM®. The next chapter illustrates the test set-ups and corresponding results.

CHAPTER 5

VALIDATION STUDIES

This chapter describes the validation tests that were conducted for our new approach of calculating swept volumes. All test cases are basically dynamic mesh simulations involving mesh motions of varying complexity. In OpenFOAM®, mesh motion can be achieved in two ways:

1. Defining the movement scheme at a specified boundary and then displacing the internal points accordingly by using mesh motion solvers. These solvers implement a wide range of diffusion models for calculating the movement vectors of each internal point location. The user chooses the diffusion scheme from an available list.

2. Manually defining the movement equations for each point location in the mesh.

We used the first approach in all cases except the one discussed in Section 5.5. There, we manually defined the equations for the movement of each individual point location.

5.1 Testing parameter

The varying values of the local volume continuity errors was the testing parameter for all validation studies. A local volume continuity error is the absolute difference between the sum of the swept volumes of a cell and the change in cell volume during a time-step. The final error is the weighted average of errors from all the cells which are weighted using their respective current cell volumes. Calculating the weighted average is a method of error normalization. The pseudo-code in Algorithm 5.1 paraphrases the method.

Data: Cell volumes at time t and $t+1$ and swept volumes of the mesh faces.
Result: Local volume continuity error.
for *All cells* **do**

$$\textbf{meshFlux} = \sum_{f=1}^{No.\ of\ faces\ in\ the\ cell} (V_{swept})_f$$

$\backslash\backslash V_{swept} = Volume\ swept\ by\ a\ face$
$\backslash\backslash meshFlux\ will\ have\ a\ unit\ vector\ in\ direction\ of\ mesh\ movement$

$$a = \frac{\nabla.(\textbf{meshFlux})}{V^{t+1}}$$

$$d = (1 - \frac{V^t}{V^{t+1}})$$

$\backslash\backslash V^t = Cell\ volume\ at\ time\ t$
$\backslash\backslash V^{t+1} = Cell\ volume\ at\ time\ (t+1)$

$$e = |d - a|$$

end

$$x = \sum_{n=1}^{No.\ of\ cells} (e_n V_n^{t+1})$$

$$y = \sum_{n=1}^{No.\ of\ cells} (V_n^{t+1})$$

$$Local\ volume\ continuity\ error = \frac{x}{y}$$

Algorithm 5.1: Procedure for calculating the local volume continuity error.

5.2 Motion of a sphere inside a cube

The simulation domain is a cube with a side length of 60 units. A sphere with a diameter of 20 units is located inside the cube. The center of the sphere coincides with the center of the cube [Figure 5.1]. The domain consists of a purely hexahedral mesh which contains a total of 62,814 cells. The case was run for 18 seconds of simulation time with an increment of 0.1 seconds in each time-step. The sphere oscillates with an amplitude of 10 units along the x-coordinate axis. There was no change made to the mesh topology during the motion of the sphere. In the next two runs, sphere movement was graduated to two and then three-dimensional translation. Figures 5.2 through 5.7 illustrate the mesh movement and the corresponding local volume continuity errors.

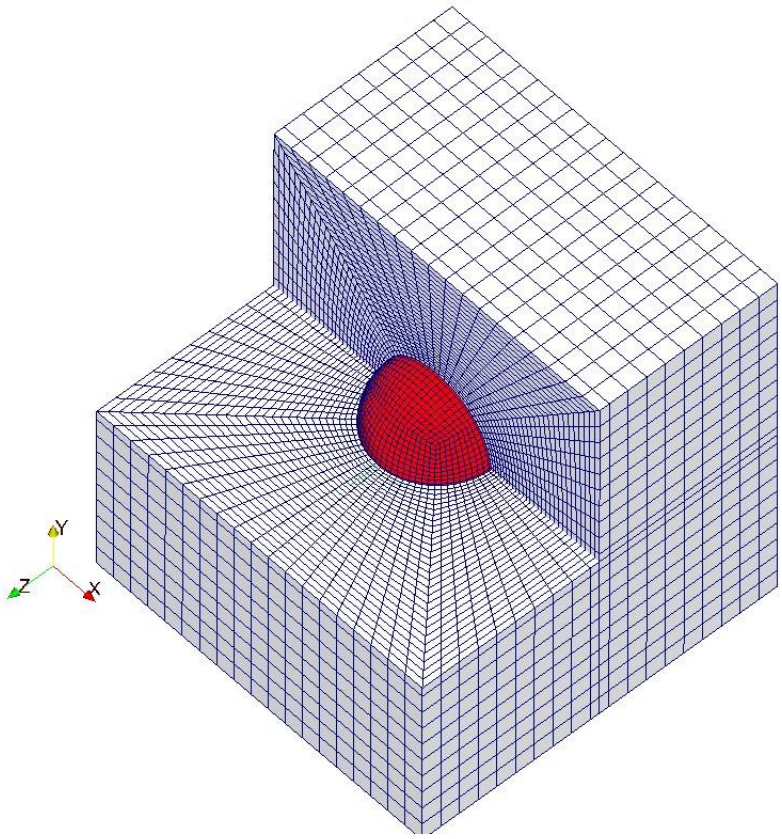


Figure 5.1: Section view of test domain.

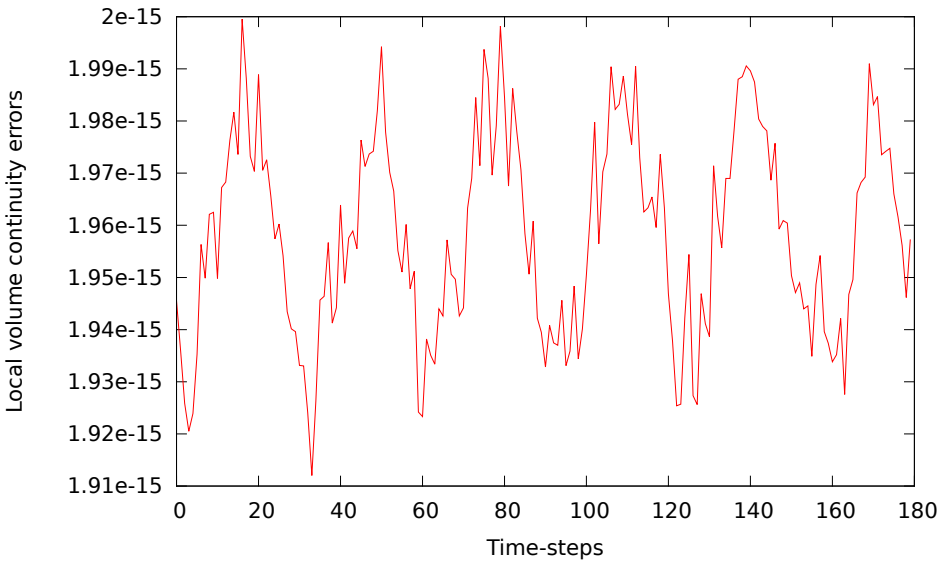
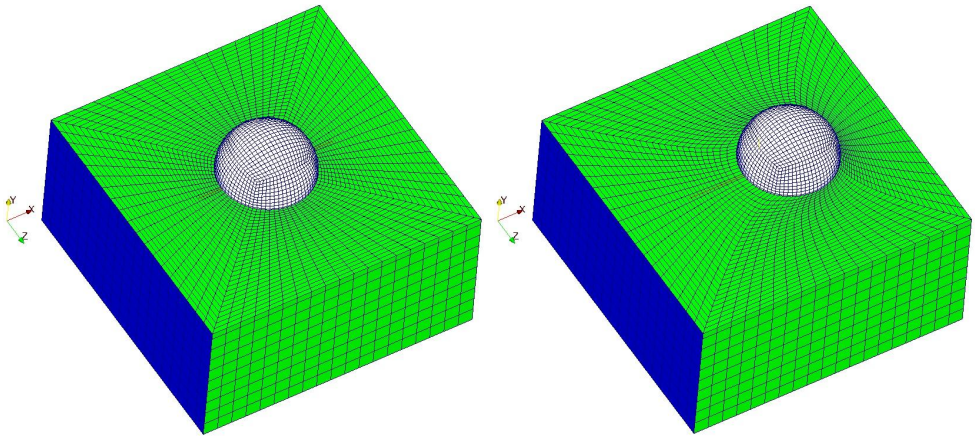
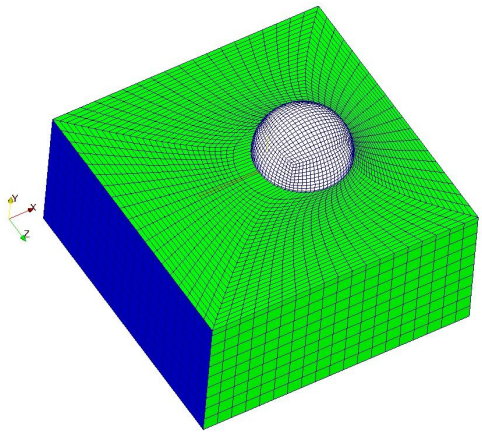


Figure 5.2: The recorded local volume continuity errors for one-dimensional oscillations of the sphere using the proposed method.

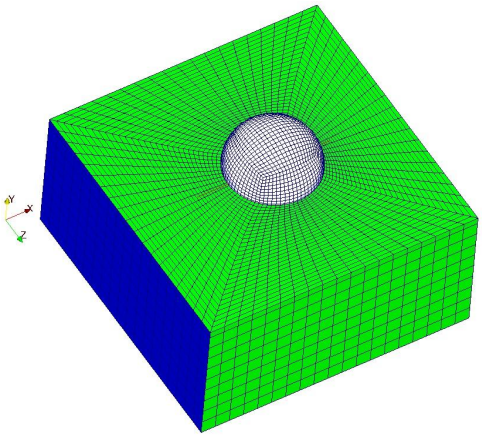


(a) 0 s

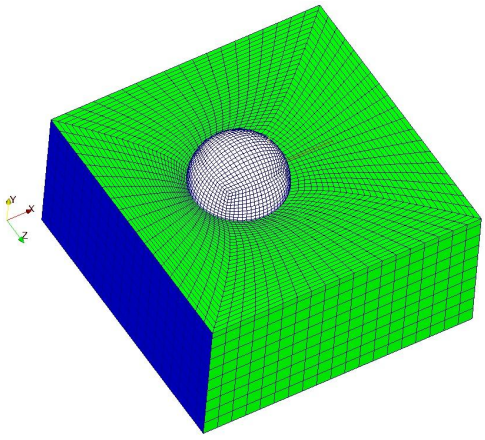
(b) 1 s



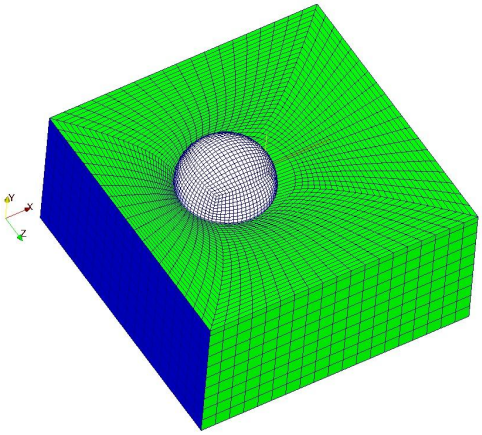
(c) 2 s



(d) 3 s

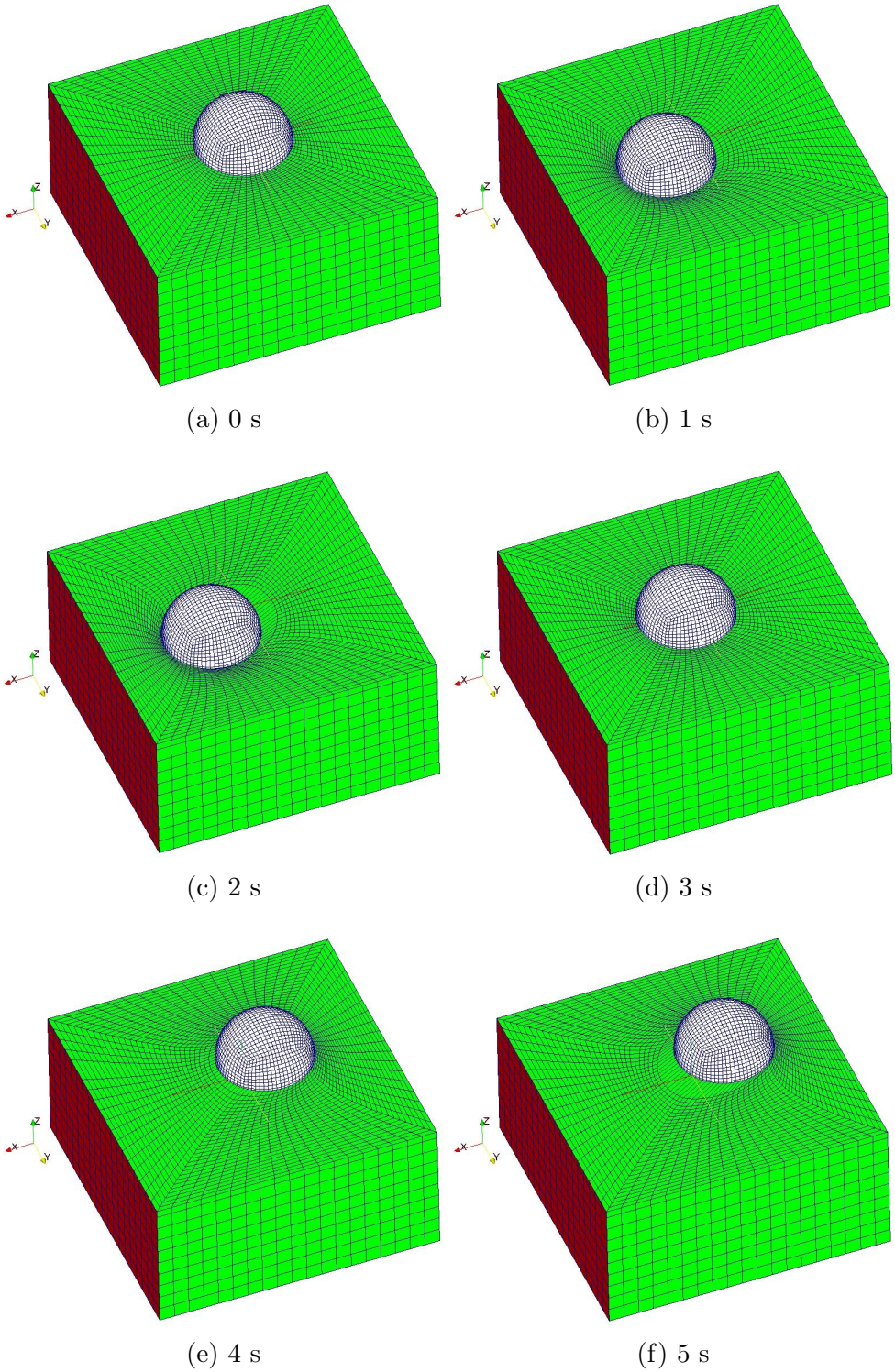


(e) 4 s



(f) 5 s

Figure 5.3: One-dimensional oscillations of the sphere inside a cube.

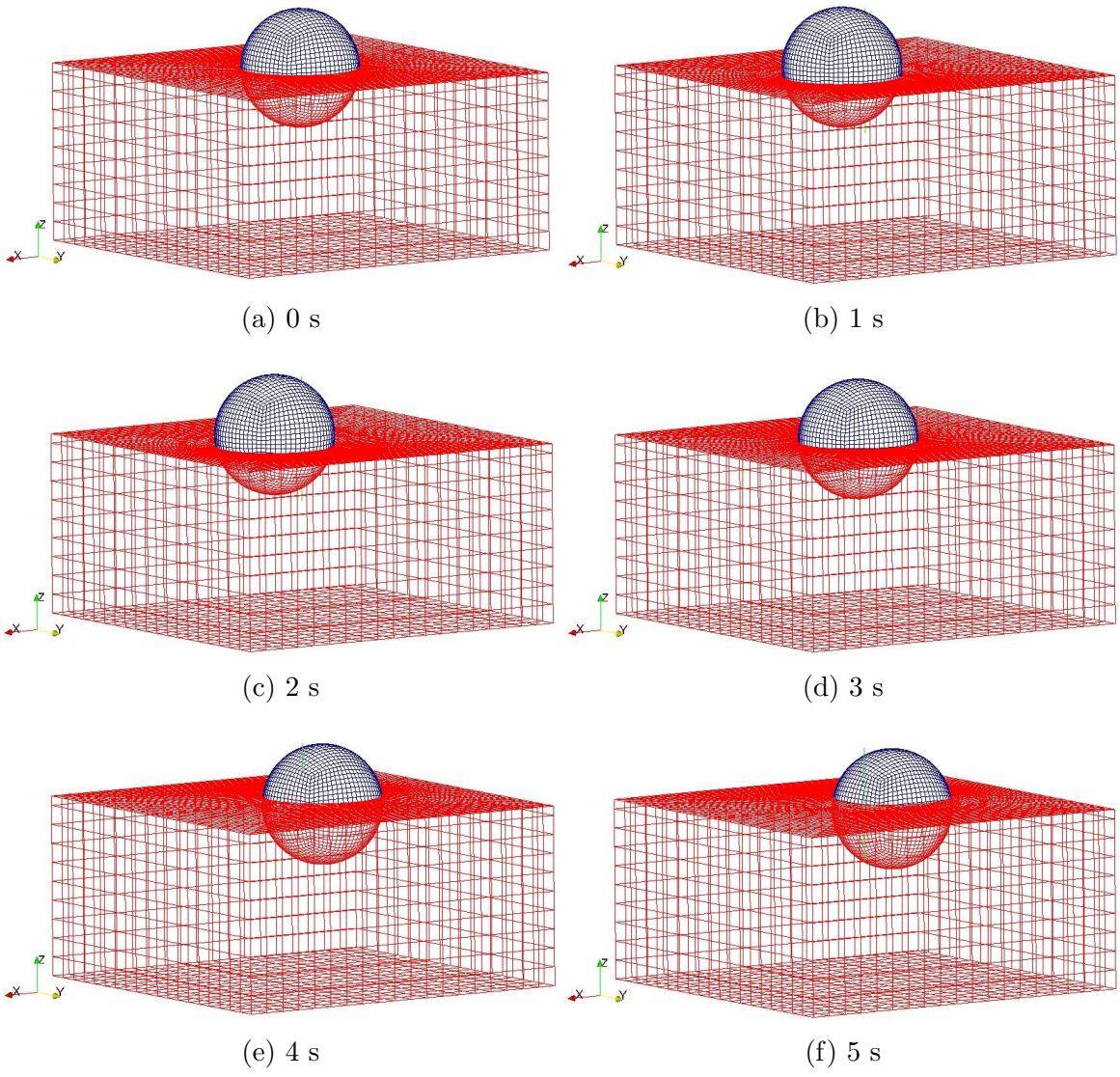


(a) 0 s (b) 1 s

(c) 2 s (d) 3 s

(e) 4 s (f) 5 s

Figure 5.4: Two-dimensional oscillations of the sphere inside a cube. *Movement vector* $= 9\hat{i} + 5\hat{j}$.



(a) 0 s (b) 1 s

(c) 2 s (d) 3 s

(e) 4 s (f) 5 s

Figure 5.5: Three-dimensional oscillations of the sphere inside a cube. *Movement vector* $= 9\hat{i} + 5\hat{j} + 2\hat{k}$.

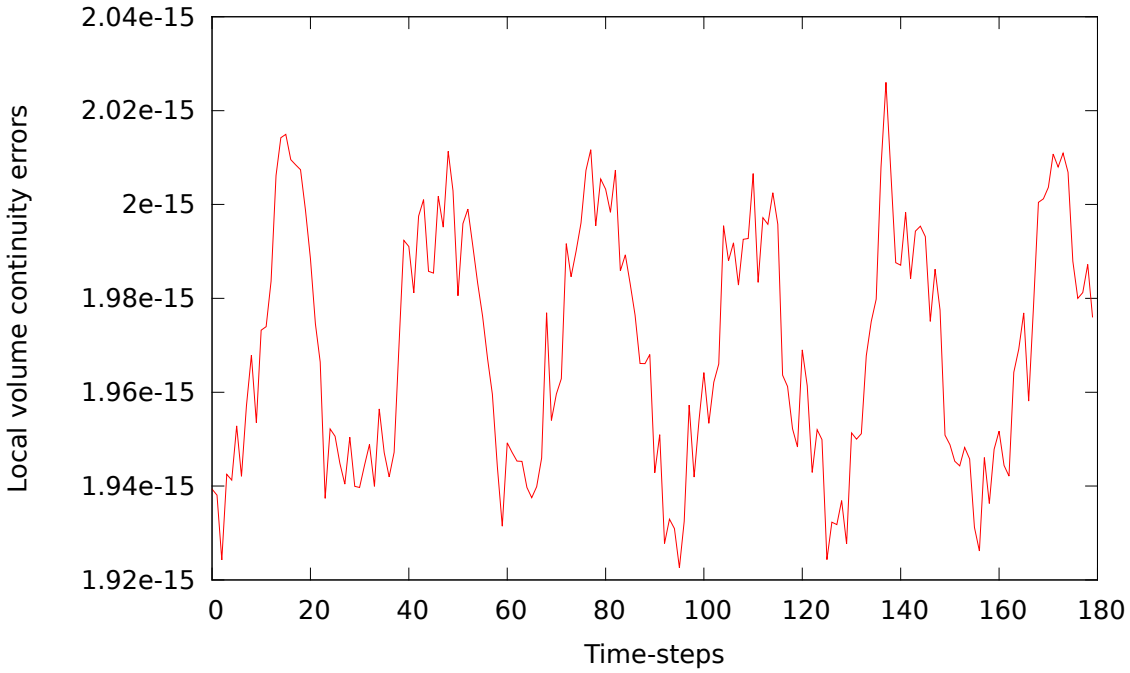


Figure 5.6: The recorded local volume continuity errors for two-dimensional oscillations of the sphere using the proposed method [Figure 5.4].

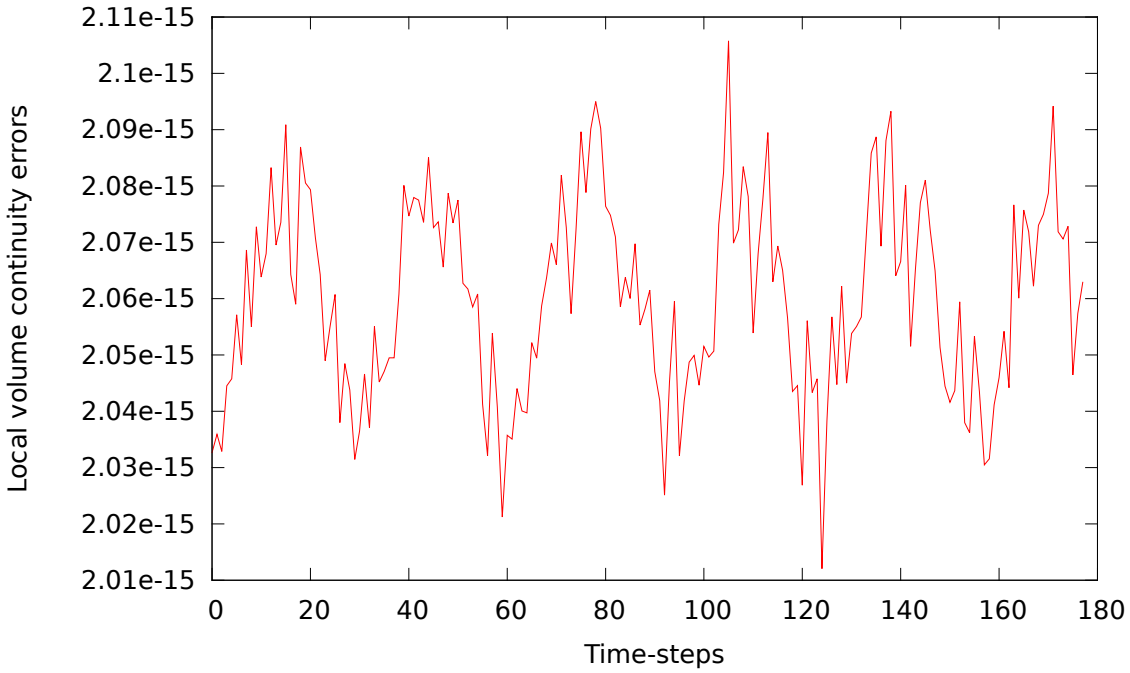


Figure 5.7: The recorded local volume continuity errors for three-dimensional oscillations of the sphere using the proposed method [Figure 5.5].

5.3 Pitching and rotating disc inside a cube

The sphere in Section 5.2 is replaced by a disc with a diameter of 20 units and a thickness of 4 units. The center of the disc coincides with the center of the cube. Figure 5.8 illustrates a schematic view of the domain. The domain consists of a tetrahedral mesh with a cell count of 26,058. The case was run for 20 seconds of simulation time with an increment of 0.1 seconds in each time-step. There was no change made to the mesh topology during the motion. The disc is pitching with an amplitude of 5 units along the y-coordinate axis. Simultaneously, it is also rotating about the z-coordinate axis through an angle of 15 degrees. Orientation of the disc at some selected time-steps is illustrated in Figures 5.9(a) through 5.9(f). The corresponding volume continuity errors are shown in Figure 5.10.

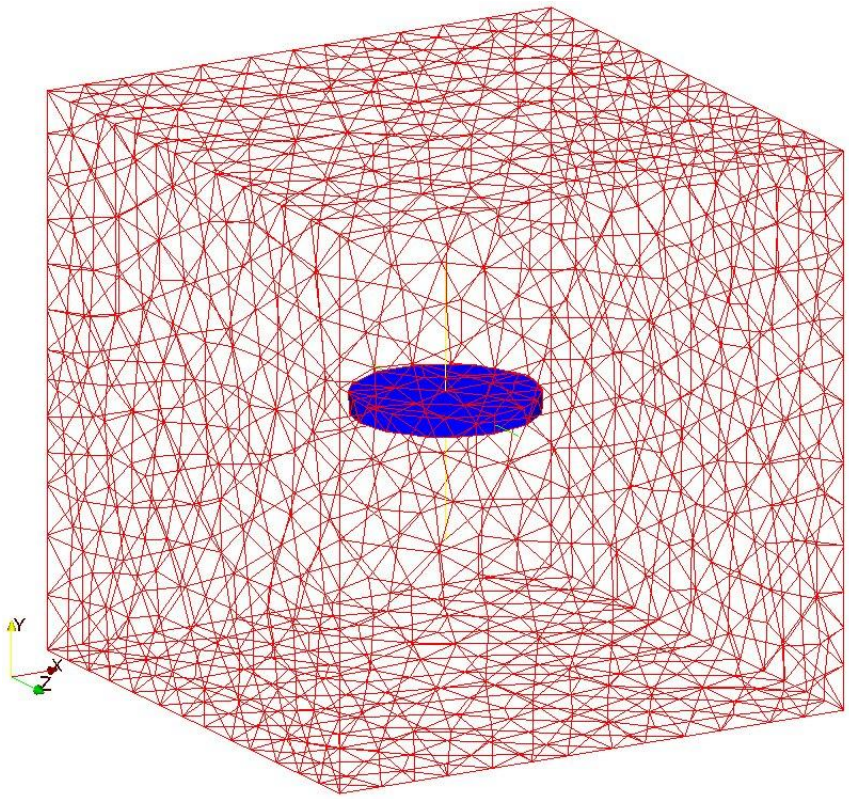
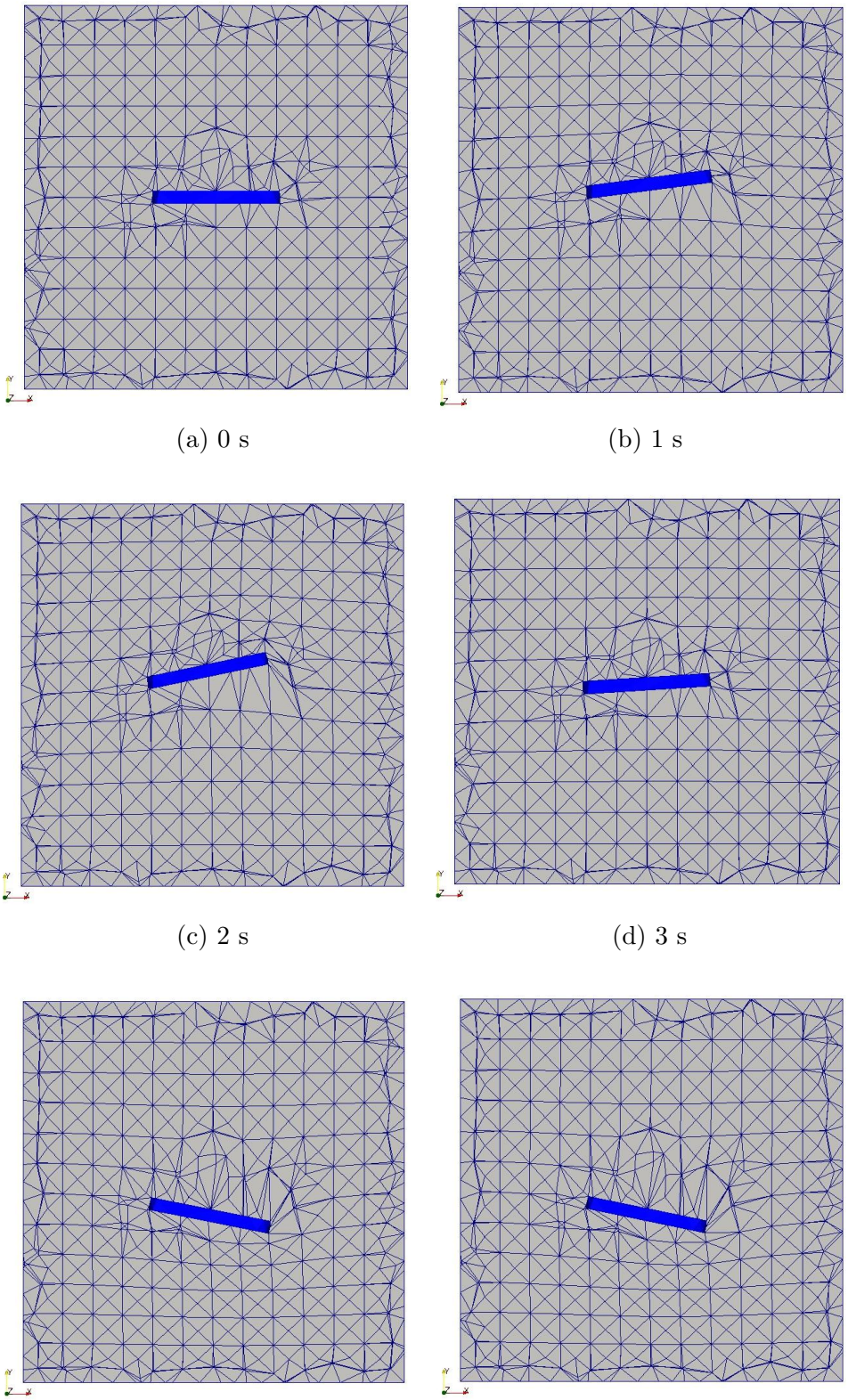


Figure 5.8: Test domain for the pitching and rotating disc.



(a) 0 s (b) 1 s

(c) 2 s (d) 3 s

(e) 4 s (f) 5 s

Figure 5.9: Simultaneous pitching and rotation of the disc.

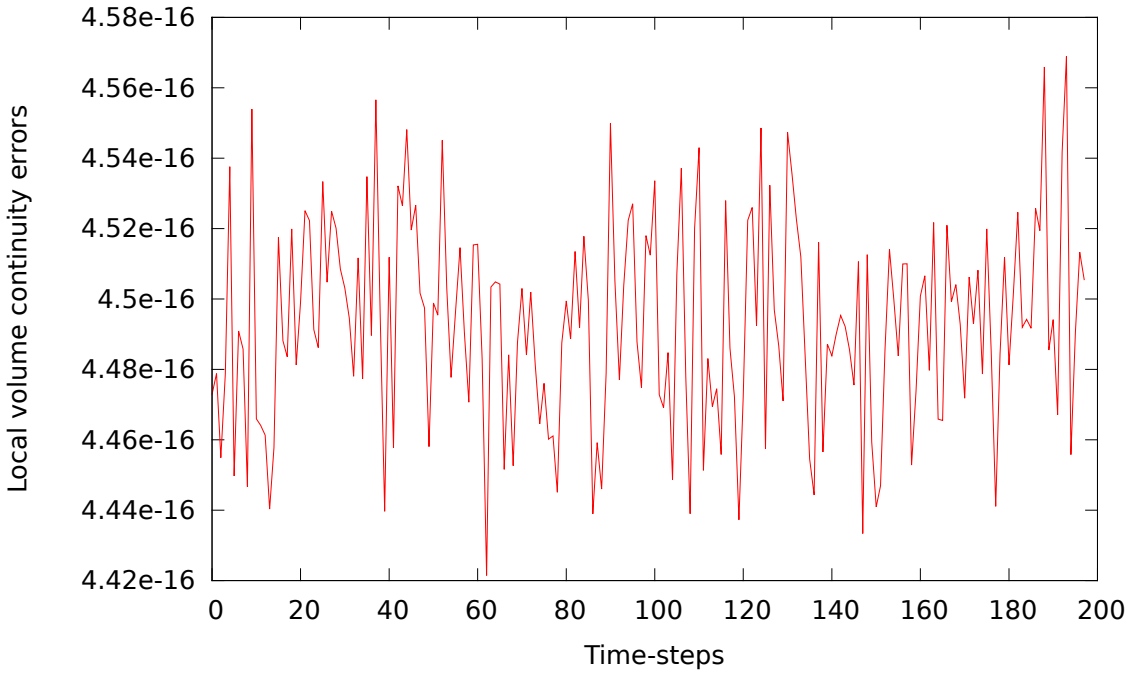
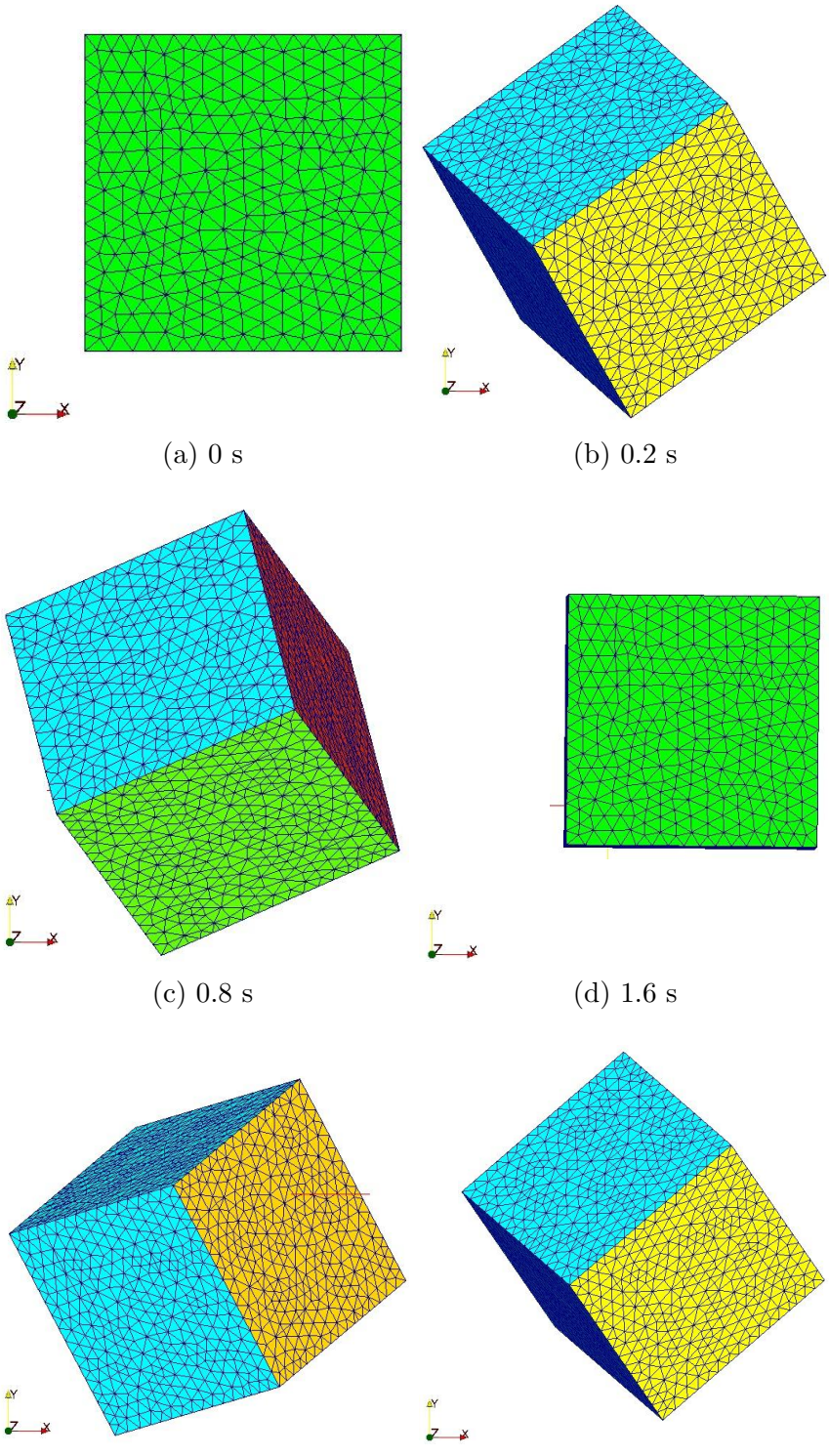


Figure 5.10: The recorded local volume continuity errors for the simultaneous pitching and rotating motion of the disc using the proposed method.

5.4 Arbitrary solid body motion of a cube

In this case, a cube with a side length of 60 units is modeled as a solid object. It is made to move in an arbitrary fashion. The cube is given the freedom to use any of the six degrees of freedom separately or simultaneously. The motion was specified using a 6DoF solver which generates a table that guides the mesh to rotate and translate accordingly. The mesh motion is defined so that the cube translates along the movement vector $(20\hat{i} + 20\hat{j} + 20\hat{k})$ and is free to rotate through 180 degrees about any principal coordinate axis. Therefore, in this case the three-dimensional translation and rotation takes place simultaneously. Furthermore, due to the solid body constraint the total volume of the domain remains intact throughout the simulation. Thus, the mesh flux is essentially divergence free. The domain consists of a tetrahedral mesh which has a total of 50,176 cells. The simulation was run for a total clock time of 20 seconds with an increment of 0.05 seconds in each time-step.



(a) 0 s (b) 0.2 s

(c) 0.8 s (d) 1.6 s

(e) 4.1 s (f) 6.5 s

Figure 5.11: Arbitrary motion of the cube.

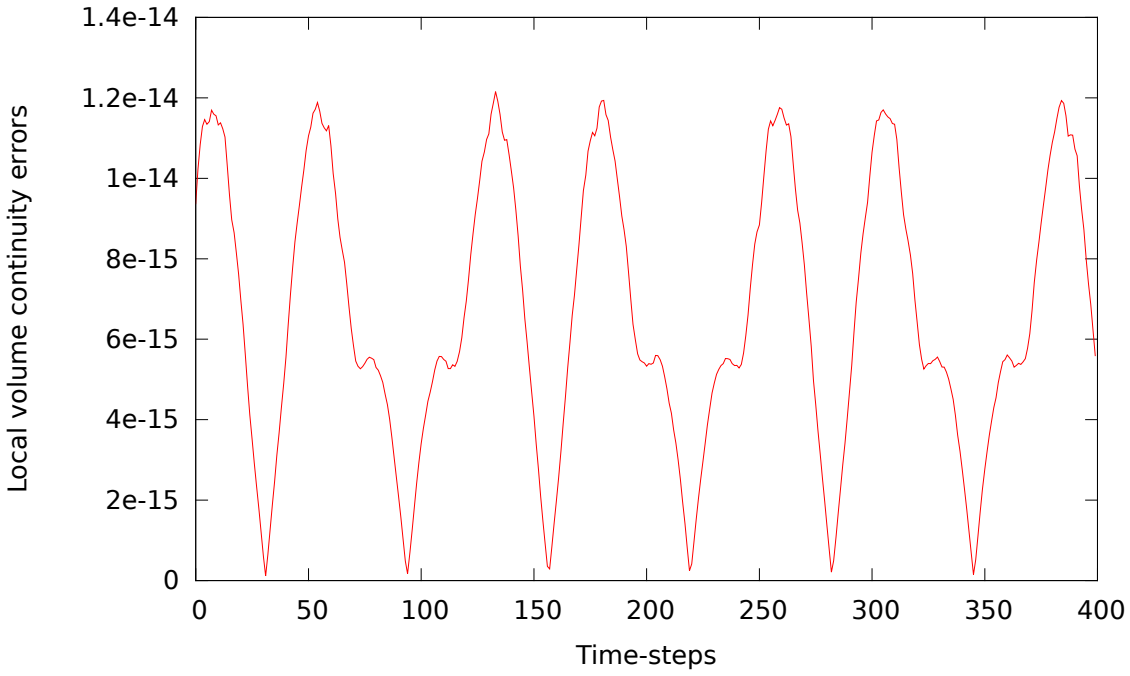


Figure 5.12: The recorded local volume continuity errors for the arbitrary solid body motion of the cube using the proposed method.

Figures 5.11(a) through 5.11(f) show the orientation and location of the cube at some selected time-steps. The corresponding volume continuity errors are shown in Figure 5.12.

5.5 Twisting motion of a cylindrical bar

In this simulation, a cylindrical bar is subjected to simultaneous twisting and translating motion. The cylinder has a radius of 10 units and a height of 40 units. The top face is fixed in space. The bottom face rotates through 405 degrees as it translates 20 units along its area normal vector. The first run was conducted using a hexahedral mesh followed by a run using tetrahedral mesh. Two additional runs were conducted by converting the tetrahedral mesh into a polyhedral mesh in two successive iterations. The hexahedral mesh has 481 cells while the tetrahedral mesh has 2,628 cells. The first conversion to polyhedral mesh brought the cell count to

10,512, which became 75,096 cells in the second iteration. The simulation was run for 5 seconds of clock time with an increment of 0.005 seconds in each time step. Figures 5.13(a) through 5.13(d) illustrate the different meshes which were used. The shape of the domain at some selected time-steps is shown in Figures 5.14(a) through 5.14(f). The corresponding volume continuity errors are shown in Figures 5.15 through 5.18.

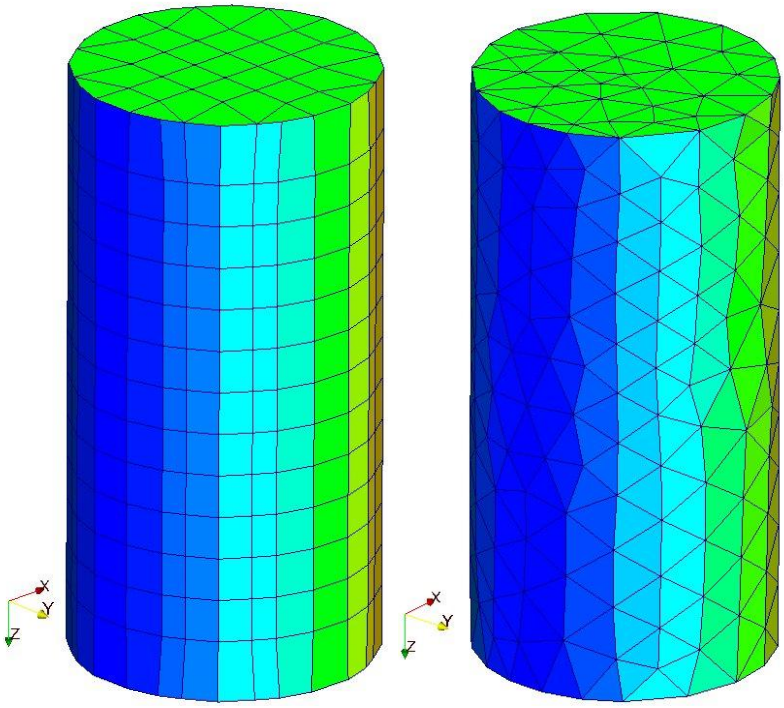
5.6 Observations

5.6.1 Periodicity

There is a periodicity in the volume continuity errors of each case. This is a result of the oscillatory nature of mesh motion. Domains in all validation cases contain an oscillating member. In one cycle, that member moves from its mean position upto a distance stipulated by the amplitude specification. The simulation-time comprises of several time-periods of such cycles. Consequently, we are recording repetitive error values.

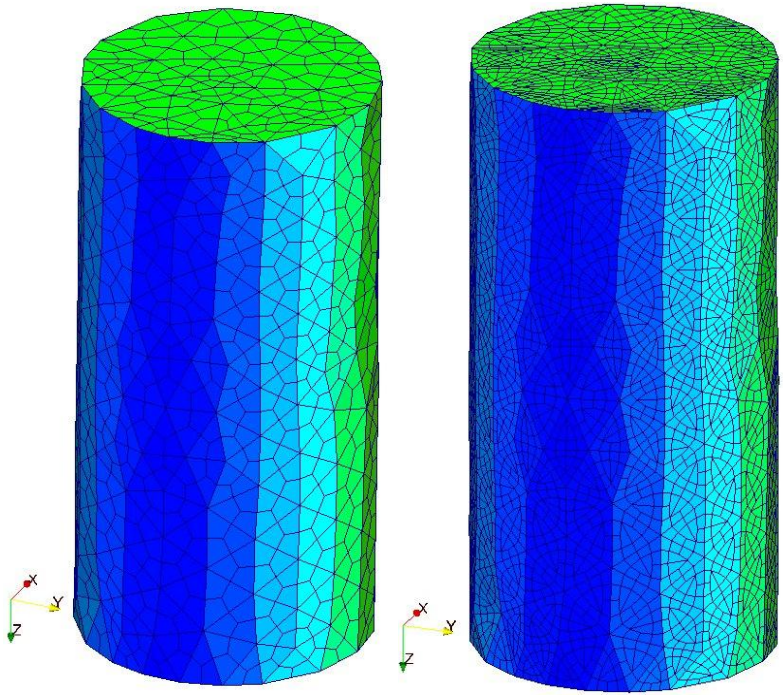
5.6.2 Concluding Remarks

The normalized volume continuity errors in all cases were recorded in the order of approximately 10^{-15}. The OpenFOAM® data-type *scalar* that is used for storing the values of volumes as well as errors has a precision of 16 decimal places. It can be asserted that the errors entered the range of order 10^{-15} due to accumulation of machine round-offs. Therefore, we have been successful in limiting the volume continuity errors to machine round-offs.



(a) Hexahedral Mesh

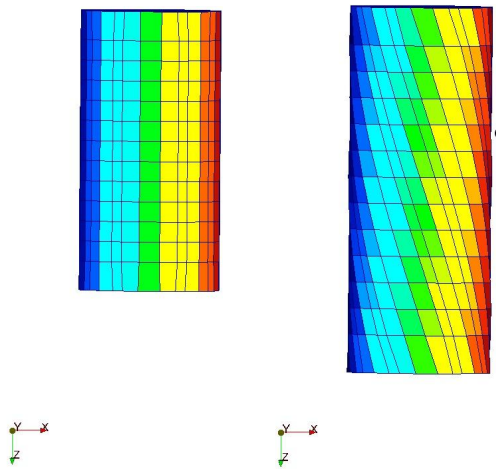
(b) Tetrahedral Mesh



(c) Polyhedral Mesh

(d) Polyhedral Mesh - Second conversion

Figure 5.13: Different meshes used for the simulation run.



(a) 0 s (b) 0.25 s

(c) 0.5 s (d) 0.75 s

(e) 1 s (f) 1.25 s

Figure 5.14: Twisting and translating motion of the cylindrical bar.

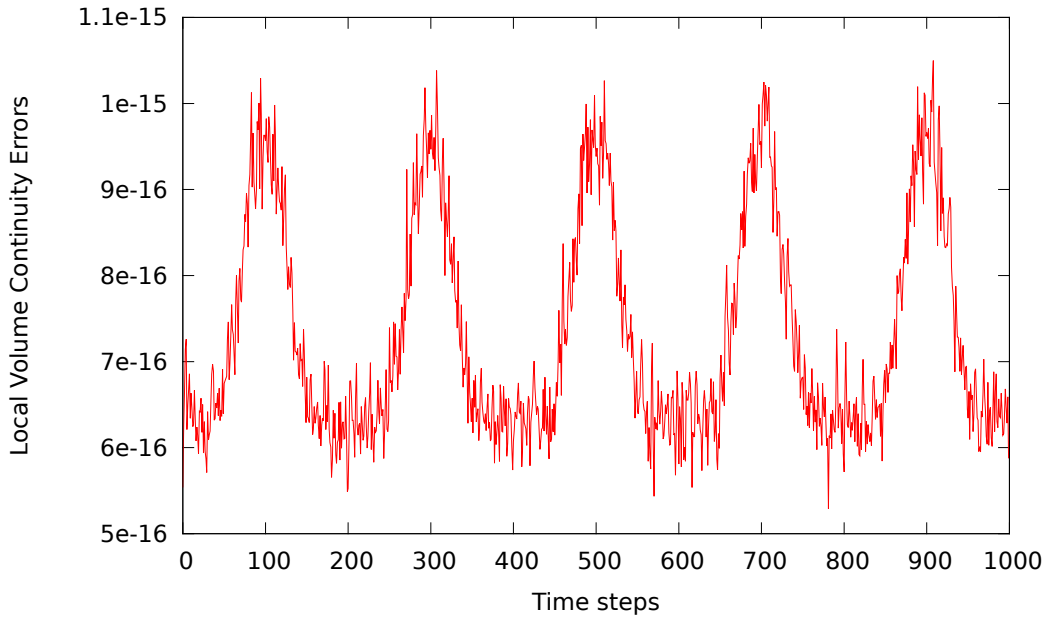


Figure 5.15: The recorded local volume continuity errors for simultaneous twisting and translating motion using the proposed method. Here,the cylindrical bar consists of a hexahedral mesh.

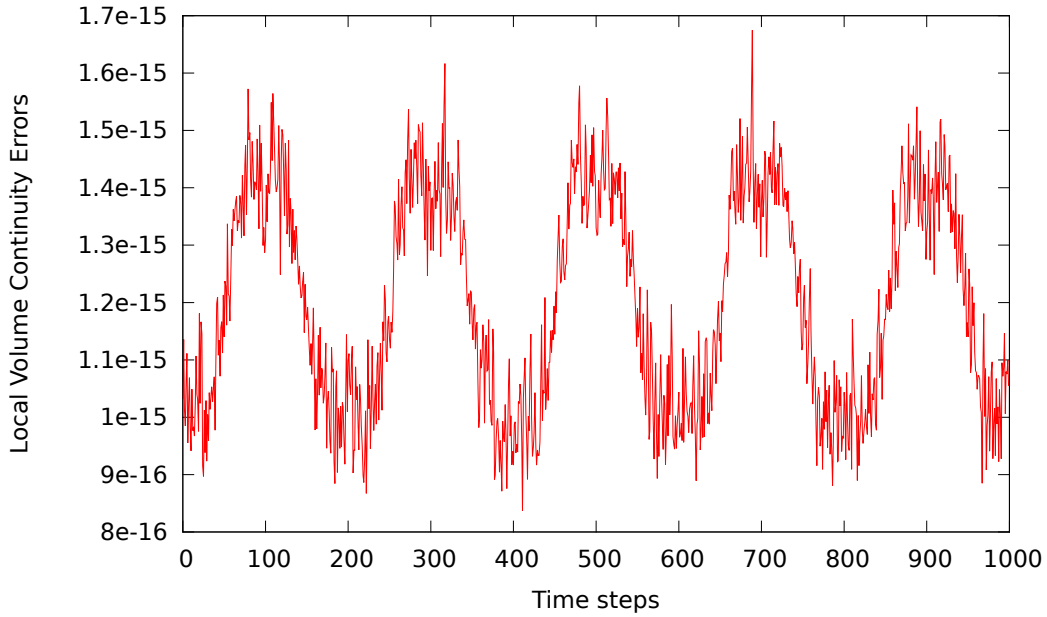


Figure 5.16: The recorded local volume continuity errors for simultaneous twisting and translating motion using the proposed method. Here,the cylindrical bar consists of a tetrahedral mesh.

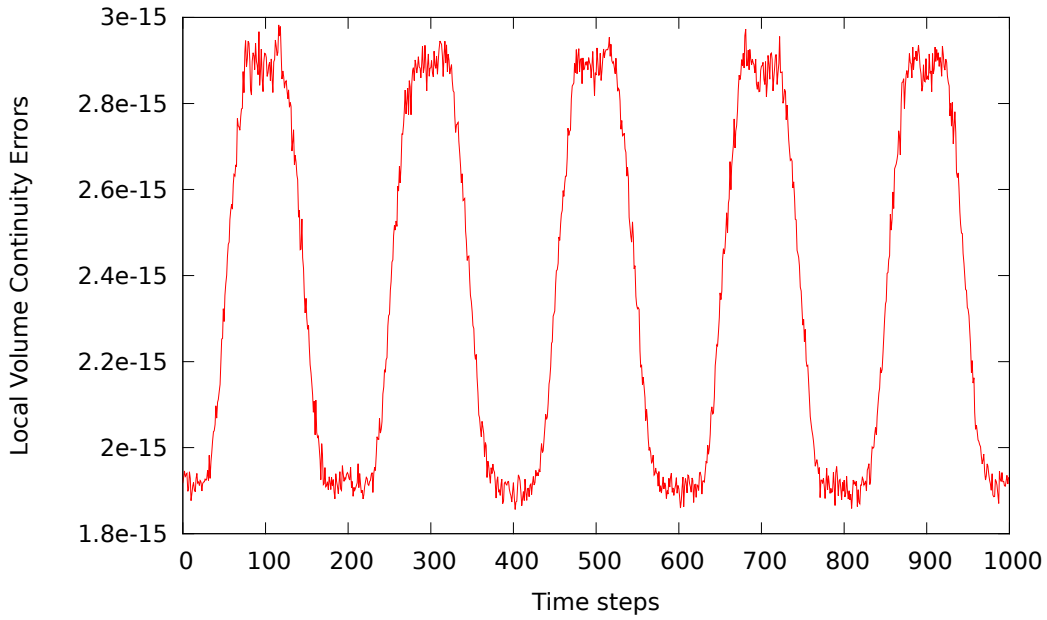


Figure 5.17: The recorded local volume continuity errors for simultaneous twisting and translating motion using the proposed method. Here, the cylindrical bar consists of a polyhedral mesh.

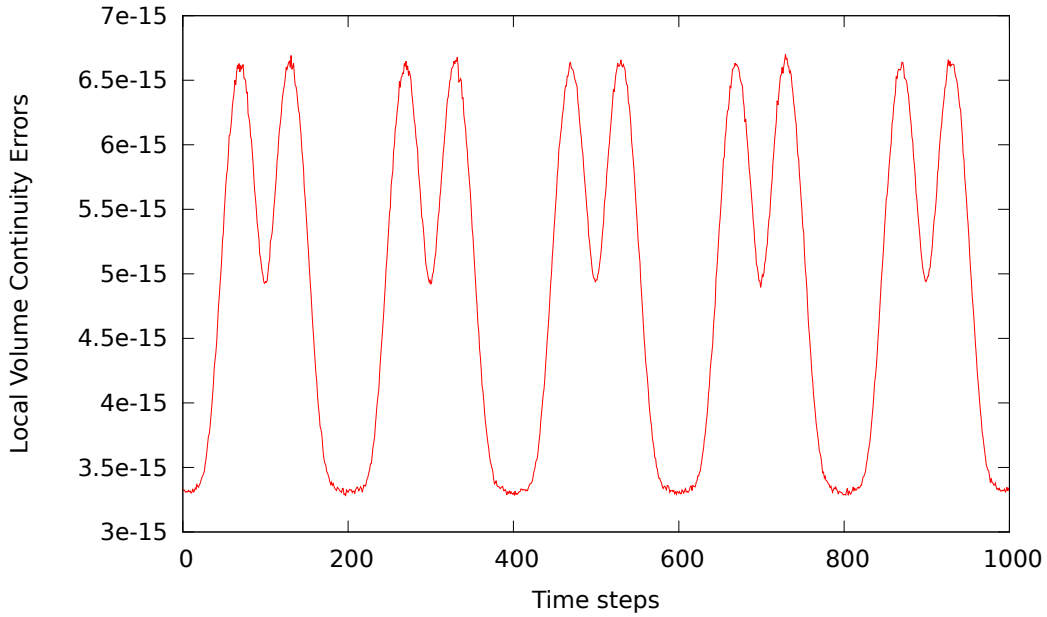


Figure 5.18: The recorded local volume continuity errors for simultaneous twisting and translating motion using the proposed method. Here the cylindrical bar consists of a polyhedral mesh which converted from an existing polyhedral [Figure 5.13(c)].

CHAPTER 6

CONCLUSIONS

First of all, the significance of implementing the Space Conservation Law was analyzed in detail. The different challenges in implementing it in discrete form were also explained. This was also an opportunity to understand the OpenFOAM® architecture in detail. The source code contains advanced levels of C++ programming, therefore it was mandatory follow a methodical approach to this development work. We attempted to display the shortcomings of the current algorithms and demonstrated some alternative ideas. In our understanding, these ideas can be of some benefit over the existing schematics.

Swept volumes play a crucial role in implementing the Space Conservation Law. The work presented in the preceding chapters has provided an alternate perception towards calculating swept volumes. Up to this point, swept volumes have been considered geometrically different from cell volumes. Swept volumes were viewed as mere fluxes emanating from the control surfaces. In this work, we have attempted to treat swept volumes and cell volumes identically. In effect, the swept volumes were recognized as virtual intermittent cells which existed in transition between successive time-steps. The volume continuity errors were limited to machine round-offs as a result of out work.

With this work we were able to address the accuracy and consistency issues surrounding the current method of calculating swept volumes. We benchmarked the accuracy of our approach with some standard results from literature. This work has brought consistency in two ways: first it provided a consistent and unique process of

calculating swept volumes. Second, we were able to bring a higher level of coherence to the OpenFOAM® architecture as a whole.

APPENDIX A

A DERIVATION FOR EQUATION (2.13)

Let there be a triangular face f at time $t = 0\ s$. The vertices of f are denoted by vectors $\mathbf{a}_0$, $\mathbf{b}_0$ and $\mathbf{c}_0$. During a time interval Δt the vertices $\mathbf{a}_0$, $\mathbf{b}_0$ and $\mathbf{c}_0$ move with constant velocities $\mathbf{v}_a$, $\mathbf{v}_b$ and $\mathbf{v}_c$ respectively. At time instant Δt, let the vertices of face f be denoted by $\mathbf{a}_1$, $\mathbf{b}_1$ and $\mathbf{c}_1$ respectively.

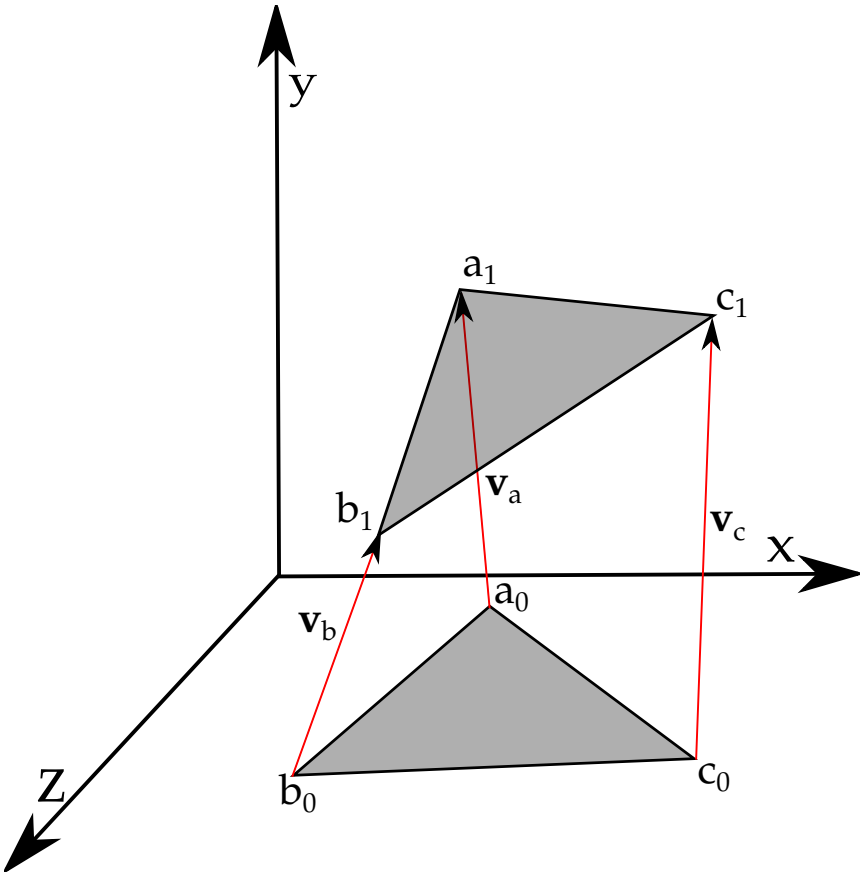


Figure A.1: Locations and vertices of the triangular face f.

The volume swept by f during a time interval Δt can be expressed as:

$$V_{swept} = \int_0^{\Delta t} \left[\int [\mathbf{v}_f . \mathbf{n} dS] \right] dt \tag{A.1}$$

where $\mathbf{v}_f$ is the velocity of f and $\mathbf{n}$ is the area normal vector. Perot and Nallpati [27] considered the velocity of the face centroid as the velocity of face f. Therefore we can express Equation (A.1) as:

$$V_{swept} = \int_0^{\Delta t} \left[\mathbf{u}^{CG} . \mathbf{S}(t) \right] dt$$

where $\mathbf{S}(t)$ is the area of f at some arbitrary time t. During the face movement, the initial and final locations of the vertices are known. Hence the vectors $\mathbf{a}_0$, $\mathbf{b}_0$, $\mathbf{c}_0$, $\mathbf{a}_1$, $\mathbf{b}_1$ and $\mathbf{c}_1$ are known to us. We also know the respective velocities of the face vertices. Therefore we can articulate $\mathbf{S}(t)$ as:

$$\mathbf{S}(t) = \frac{1}{2} \left[((\mathbf{b}_0 + \mathbf{v}_b t) - (\mathbf{a}_0 + \mathbf{v}_a t)) \times ((\mathbf{c}_0 + \mathbf{v}_c t) - (\mathbf{a}_0 + \mathbf{v}_a t)) \right]$$

$$\Rightarrow \mathbf{S}(t) = \frac{1}{2} \left[(\mathbf{b}_0 + \mathbf{v}_b t - \mathbf{a}_0 - \mathbf{v}_a t) \times (\mathbf{c}_0 + \mathbf{v}_c t - \mathbf{a}_0 - \mathbf{v}_a t) \right]$$

$$\Rightarrow \mathbf{S}(t) = \frac{1}{2} \left[((\mathbf{b}_0 - \mathbf{a}_0) - (\mathbf{v}_a - \mathbf{v}_b)\, t) \times ((\mathbf{c}_0 - \mathbf{a}_0) - (\mathbf{v}_a - \mathbf{v}_c)\, t) \right]$$

$$\Rightarrow 2\mathbf{S}(t) = [(\mathbf{b}_0 - \mathbf{a}_0) \times (\mathbf{c}_0 - \mathbf{a}_0)] - [(\mathbf{b}_0 - \mathbf{a}_0) \times (\mathbf{v}_a - \mathbf{v}_c)\, t] - [(\mathbf{v}_a - \mathbf{v}_b)\, t \times (\mathbf{c}_0 - \mathbf{a}_0)] + t^2 [(\mathbf{v}_a - \mathbf{v}_b) \times (\mathbf{v}_a - \mathbf{v}_c)] \tag{A.2}$$

We can express Equation (A.1) as:

$$V_{swept} = \int_0^{\Delta t} \frac{\left[\mathbf{u}^{CG} . (2\mathbf{S}(t)) \right] dt}{2} \tag{A.3}$$

$$\Rightarrow V_{swept} = \frac{\mathbf{u}^{CG}}{2} . \int_0^{\Delta t} [(2\mathbf{S}(t))]\, dt \tag{A.4}$$

Substituting Equation (A.2) in Equation (A.4) we can calculate the volume (V^1_{swept}) swept by f in time Δt. Here, the superscript 1 denotes our first approach for calculating swept volume.

$$V^1_{swept} = \underbrace{\left[\frac{\mathbf{u}^{CG}}{2} . \int_0^{\Delta t} [(\mathbf{b}_0 - \mathbf{a}_0) \times (\mathbf{c}_0 - \mathbf{a}_0)]\, dt\right]}_{1} - \underbrace{\left[\frac{\mathbf{u}^{CG}}{2} . \int_0^{\Delta t} [(\mathbf{b}_0 - \mathbf{a}_0) \times (\mathbf{v}_a - \mathbf{v}_c)\, t]\, dt\right]}_{2} -$$
$$\underbrace{\left[\frac{\mathbf{u}^{CG}}{2} . \int_0^{\Delta t} [(\mathbf{v}_a - \mathbf{v}_b)\, t \times (\mathbf{c}_0 - \mathbf{a}_0)]\, dt\right]}_{3} + \underbrace{\left[\frac{\mathbf{u}^{CG}}{2} . \int_0^{\Delta t} t^2\, [(\mathbf{v}_a - \mathbf{v}_b) \times (\mathbf{v}_a - \mathbf{v}_c)]\, dt\right]}_{4} \tag{A.5}$$

From a different perspective, we can also express $\mathbf{S}(t)$ in the following manner:

$$\mathbf{S}(t) = \frac{1}{2}\left[((\mathbf{b}_1 - \mathbf{v}_b t) - (\mathbf{a}_1 - \mathbf{v}_a t)) \times ((\mathbf{c}_1 - \mathbf{v}_c t) - (\mathbf{a}_1 - \mathbf{v}_a t))\right]$$

$$\Rightarrow \mathbf{S}(t) = \frac{1}{2}\left[(\mathbf{b}_1 - \mathbf{v}_b t - \mathbf{a}_1 + \mathbf{v}_a t) \times (\mathbf{c}_1 - \mathbf{v}_c t - \mathbf{a}_1 + \mathbf{v}_a t)\right]$$
$$\Rightarrow \mathbf{S}(t) = \frac{1}{2}\left[((\mathbf{b}_1 - \mathbf{a}_1) + (\mathbf{v}_a - \mathbf{v}_b)\, t) \times ((\mathbf{c}_1 - \mathbf{a}_1) + (\mathbf{v}_a - \mathbf{v}_c)\, t)\right]$$

$$\Rightarrow 2\mathbf{S}(t) = [(\mathbf{b}_1 - \mathbf{a}_1) \times (\mathbf{c}_1 - \mathbf{a}_1)] + [(\mathbf{b}_1 - \mathbf{a}_1) \times (\mathbf{v}_a - \mathbf{v}_c)\, t] +$$
$$[(\mathbf{v}_a - \mathbf{v}_b)\, t \times (\mathbf{c}_1 - \mathbf{a}_1)] + t^2\, [(\mathbf{v}_a - \mathbf{v}_b) \times (\mathbf{v}_a - \mathbf{v}_c)] \tag{A.6}$$

Substituting Equation (A.6) in Equation (A.4) we can again calculate the volume (V^2_{swept}) swept by f in time Δt.

$$V^2_{swept} = \underbrace{\left[\frac{\mathbf{u}^{CG}}{2}.\int_0^{\Delta t}\left[(\mathbf{b}_1-\mathbf{a}_1)\times(\mathbf{c}_1-\mathbf{a}_1)\right]dt\right]}_{1} + \underbrace{\left[\frac{\mathbf{u}^{CG}}{2}.\int_0^{\Delta t}\left[(\mathbf{b}_1-\mathbf{a}_1)\times(\mathbf{v}_a-\mathbf{v}_c)\,t\right]dt\right]}_{2} + \underbrace{\left[\frac{\mathbf{u}^{CG}}{2}.\int_0^{\Delta t}\left[(\mathbf{v}_a-\mathbf{v}_b)\,t\times(\mathbf{c}_1-\mathbf{a}_1)\right]dt\right]}_{3} + \underbrace{\left[\frac{\mathbf{u}^{CG}}{2}.\int_0^{\Delta t}t^2\left[(\mathbf{v}_a-\mathbf{v}_b)\times(\mathbf{v}_a-\mathbf{v}_c)\right]dt\right]}_{4} \tag{A.7}$$

The values V^1_{swept} and V^2_{swept} signify that the volume swept by f can be calculated in more than one way even while using identical approaches. Perot and Nallapati's [27] expression is an average of V^1_{swept} and V^2_{swept}. However, before calculating the average, we need to perform some algebraic manipulations in Equation (A.7). We know that $\mathbf{a}_0$, $\mathbf{b}_0$ and $\mathbf{c}_0$ took time Δt to reach $\mathbf{a}_1$, $\mathbf{b}_1$ and $\mathbf{c}_1$. Therefore, substituting $\mathbf{a}_1 = \mathbf{a}_0 + \mathbf{v}_a\Delta t$, $\mathbf{b}_1 = \mathbf{b}_0 + \mathbf{v}_b\Delta t$, $\mathbf{c}_1 = \mathbf{c}_0 + \mathbf{v}_c\Delta t$ in the second and fourth term of Equation (A.7):

$$\begin{aligned} V^2_{swept} = {} & \underbrace{\left[\frac{\mathbf{u}^{CG}}{2}.\int_0^{\Delta t}\left[(\mathbf{b}_1-\mathbf{a}_1)\times(\mathbf{c}_1-\mathbf{a}_1)\right]dt\right]}_{1} + \\ & \underbrace{\left[\frac{\mathbf{u}^{CG}}{2}.\int_0^{\Delta t}\left[(\mathbf{b}_0+\mathbf{v}_b\Delta t-\mathbf{a}_0-\mathbf{v}_a\Delta t)\times(\mathbf{v}_a-\mathbf{v}_c)\,t\right]dt\right]}_{2} + \\ & \underbrace{\left[\frac{\mathbf{u}^{CG}}{2}.\int_0^{\Delta t}\left[(\mathbf{v}_a-\mathbf{v}_b)\,t\times(\mathbf{c}_0+\mathbf{v}_c\Delta t-\mathbf{a}_0-\mathbf{v}_a\Delta t)\right]dt\right]}_{3} + \\ & \underbrace{\left[\frac{\mathbf{u}^{CG}}{2}.\int_0^{\Delta t}t^2\left[(\mathbf{v}_a-\mathbf{v}_b)\times(\mathbf{v}_a-\mathbf{v}_c)\right]dt\right]}_{4} \end{aligned} \tag{A.8}$$

$$\Rightarrow V_{swept}^2 = \underbrace{\left[\frac{\mathbf{u}^{CG}}{2}.\int_0^{\Delta t}\left[(\mathbf{b}_1-\mathbf{a}_1)\times(\mathbf{c}_1-\mathbf{a}_1)\right]dt\right]}_{1} + \underbrace{\left[\frac{\mathbf{u}^{CG}}{2}.\int_0^{\Delta t}\left[((\mathbf{b}_0-\mathbf{a}_0)+(\mathbf{v}_b-\mathbf{v}_a)\,\Delta t)\times(\mathbf{v}_a-\mathbf{v}_c)\,t\right]dt\right]}_{2} + \underbrace{\left[\frac{\mathbf{u}^{CG}}{2}.\int_0^{\Delta t}\left[(\mathbf{v}_a-\mathbf{v}_b)\,t\times((\mathbf{c}_0-\mathbf{a}_0)+(\mathbf{v}_c-\mathbf{v}_a)\,\Delta t)\right]dt\right]}_{3} + \underbrace{\left[\frac{\mathbf{u}^{CG}}{2}.\int_0^{\Delta t}t^2\left[(\mathbf{v}_a-\mathbf{v}_b)\times(\mathbf{v}_a-\mathbf{v}_c)\right]dt\right]}_{4} \quad \text{(A.9)}$$

$$\Rightarrow V_{swept}^2 = \underbrace{\left[\frac{\mathbf{u}^{CG}}{2}.\int_0^{\Delta t}\left[(\mathbf{b}_1-\mathbf{a}_1)\times(\mathbf{c}_1-\mathbf{a}_1)\right]dt\right]}_{1} + \underbrace{\left[\frac{\mathbf{u}^{CG}}{2}.\int_0^{\Delta t}\left[((\mathbf{b}_0-\mathbf{a}_0)\times(\mathbf{v}_a-\mathbf{v}_c)\,t)\right]\right]}_{2} + \underbrace{\left[\frac{\mathbf{u}^{CG}}{2}.\int_0^{\Delta t}\left[(\mathbf{v}_b-\mathbf{v}_a)\times(\mathbf{v}_a-\mathbf{v}_c)\right](\Delta t)tdt\right]}_{3} + \underbrace{\left[\frac{\mathbf{u}^{CG}}{2}.\int_0^{\Delta t}\left[(\mathbf{v}_a-\mathbf{v}_b)\,t\times(\mathbf{c}_0-\mathbf{a}_0)\right]dt\right]}_{4} + \underbrace{\left[\frac{\mathbf{u}^{CG}}{2}.\int_0^{\Delta t}\left[(\mathbf{v}_a-\mathbf{v}_b)\times(\mathbf{v}_c-\mathbf{v}_a)\right](\Delta t)tdt\right]}_{5} + \underbrace{\left[\frac{\mathbf{u}^{CG}}{2}.\int_0^{\Delta t}t^2\left[(\mathbf{v}_a-\mathbf{v}_b)\times(\mathbf{v}_a-\mathbf{v}_c)\right]dt\right]}_{6} \quad \text{(A.10)}$$

Adding Equation (A.5) and Equation (A.10):

$$
\begin{aligned}
&\left(V^1_{swept} + V^2_{swept}\right) = \\
&\underbrace{\left[\frac{\mathbf{u}^{CG}}{2}.\int_0^{\Delta t}\left[(\mathbf{b}_0-\mathbf{a}_0)\times(\mathbf{c}_0-\mathbf{a}_0)\right]dt\right]}_{1} - \underbrace{\left[\frac{\mathbf{u}^{CG}}{2}.\int_0^{\Delta t}\left[(\mathbf{b}_0-\mathbf{a}_0)\times(\mathbf{v}_a-\mathbf{v}_c)\,t\right]dt\right]}_{2} \\
&- \underbrace{\left[\frac{\mathbf{u}^{CG}}{2}.\int_0^{\Delta t}\left[(\mathbf{v}_a-\mathbf{v}_b)\,t\times(\mathbf{c}_0-\mathbf{a}_0)\right]dt\right]}_{3} + \underbrace{\left[\frac{\mathbf{u}^{CG}}{2}.\int_0^{\Delta t}t^2\left[(\mathbf{v}_a-\mathbf{v}_b)\times(\mathbf{v}_a-\mathbf{v}_c)\right]dt\right]}_{4} \\
&+ \underbrace{\left[\frac{\mathbf{u}^{CG}}{2}.\int_0^{\Delta t}\left[(\mathbf{b}_1-\mathbf{a}_1)\times(\mathbf{c}_1-\mathbf{a}_1)\right]dt\right]}_{5} + \underbrace{\left[\frac{\mathbf{u}^{CG}}{2}.\int_0^{\Delta t}\left[(\mathbf{b}_0-\mathbf{a}_0)\times(\mathbf{v}_a-\mathbf{v}_c)\,t\right]dt\right]}_{6} \\
&+ \underbrace{\left[\frac{\mathbf{u}^{CG}}{2}.\int_0^{\Delta t}\left[(\mathbf{v}_b-\mathbf{v}_a)\times(\mathbf{v}_a-\mathbf{v}_c)\right](\Delta t)tdt\right]}_{7} + \underbrace{\left[\frac{\mathbf{u}^{CG}}{2}.\int_0^{\Delta t}\left[(\mathbf{v}_a-\mathbf{v}_b)\,t\times(\mathbf{c}_0-\mathbf{a}_0)\right]dt\right]}_{8} \\
&+ \underbrace{\left[\frac{\mathbf{u}^{CG}}{2}.\int_0^{\Delta t}\left[(\mathbf{v}_a-\mathbf{v}_b)\times(\mathbf{v}_c-\mathbf{v}_a)\right](\Delta t)tdt\right]}_{9} + \underbrace{\left[\frac{\mathbf{u}^{CG}}{2}.\int_0^{\Delta t}t^2\left[(\mathbf{v}_a-\mathbf{v}_b)\times(\mathbf{v}_a-\mathbf{v}_c)\right]dt\right]}_{10}
\end{aligned}
\tag{A.11}
$$

In the Equation(A.11) the second, third, sixth and eighth terms add up to zero. Thus reducing Equation(A.11) to:

$$
\begin{aligned}
&\left(V^1_{swept} + V^2_{swept}\right) = \\
&\underbrace{\left[\frac{\mathbf{u}^{CG}}{2}.\int_0^{\Delta t}\left[(\mathbf{b}_0-\mathbf{a}_0)\times(\mathbf{c}_0-\mathbf{a}_0)\right]dt\right]}_{1} + \underbrace{\left[\frac{\mathbf{u}^{CG}}{2}.\int_0^{\Delta t}t^2\left[(\mathbf{v}_a-\mathbf{v}_b)\times(\mathbf{v}_a-\mathbf{v}_c)\right]dt\right]}_{2} \\
&+ \underbrace{\left[\frac{\mathbf{u}^{CG}}{2}.\int_0^{\Delta t}\left[(\mathbf{b}_1-\mathbf{a}_1)\times(\mathbf{c}_1-\mathbf{a}_1)\right]dt\right]}_{3} + \underbrace{\left[\frac{\mathbf{u}^{CG}}{2}.\int_0^{\Delta t}\left[(\mathbf{v}_b-\mathbf{v}_a)\times(\mathbf{v}_a-\mathbf{v}_c)\right](\Delta t)tdt\right]}_{4} \\
&+ \underbrace{\left[\frac{\mathbf{u}^{CG}}{2}.\int_0^{\Delta t}\left[(\mathbf{v}_a-\mathbf{v}_b)\times(\mathbf{v}_c-\mathbf{v}_a)\right](\Delta t)tdt\right]}_{5} + \underbrace{\left[\frac{\mathbf{u}^{CG}}{2}.\int_0^{\Delta t}t^2\left[(\mathbf{v}_a-\mathbf{v}_b)\times(\mathbf{v}_a-\mathbf{v}_c)\right]dt\right]}_{6}
\end{aligned}
\tag{A.12}
$$

In Equation (A.12), we can group the first term and third terms. Furthermore, we are adding the second and sixth terms and rearranging the variables in the fourth

term. Thus reducing it to:

$$\left(V^1_{swept} + V^2_{swept}\right) = \underbrace{\left[\frac{\mathbf{u}^{CG}}{2}.\int_0^{\Delta t}\Big[\big[\,(\mathbf{b}_0-\mathbf{a}_0)\times(\mathbf{c}_0-\mathbf{a}_0)\,\big]+\big[\,(\mathbf{b}_1-\mathbf{a}_1)\times(\mathbf{c}_1-\mathbf{a}_1)\,\big]\Big]dt\right]}_{1}$$
$$+\underbrace{\left[\frac{\mathbf{u}^{CG}}{2}.\int_0^{\Delta t}\left[(\mathbf{v}_a-\mathbf{v}_b)\times(\mathbf{v}_c-\mathbf{v}_a)\right](\Delta t)tdt\right]}_{2}+\underbrace{\left[\frac{\mathbf{u}^{CG}}{2}.\int_0^{\Delta t}\left[(\mathbf{v}_a-\mathbf{v}_b)\times(\mathbf{v}_c-\mathbf{v}_a)\right](\Delta t)tdt\right]}_{3}$$
$$+\underbrace{\left[\mathbf{u}^{CG}.\int_0^{\Delta t}t^2\left[(\mathbf{v}_a-\mathbf{v}_b)\times(\mathbf{v}_a-\mathbf{v}_c)\right]dt\right]}_{4} \quad \text{(A.13)}$$

Making the following substitutions in Equation A.13:

$$\big[\,(\mathbf{b}_0-\mathbf{a}_0)\times(\mathbf{c}_0-\mathbf{a}_0)\,\big] = 2\big[\mathbf{n}_0A_0\big]$$
$$\big[\,(\mathbf{b}_1-\mathbf{a}_1)\times(\mathbf{c}_1-\mathbf{a}_1)\,\big] = 2\big[\mathbf{n}_1A_1\big]$$

where

A_0 = Initial area of face f. A_1 = Final area of face f .

$\mathbf{n}_0$ = Initial area normal vector. $\mathbf{n}_1$ = Final area normal vector.

Followed by adding the second and third terms we get:

$$\Rightarrow \left(V^1_{swept} + V^2_{swept}\right) = \underbrace{\left[\mathbf{u}^{CG}.\int_0^{\Delta t}\Big[\mathbf{n}_0A_0+\mathbf{n}_1A_1\Big]dt\right]}_{1}+\underbrace{\left[\mathbf{u}^{CG}.\int_0^{\Delta t}\left[(\mathbf{v}_a-\mathbf{v}_b)\times(\mathbf{v}_c-\mathbf{v}_a)\right](\Delta t)tdt\right]}_{2}$$
$$+\underbrace{\left[\mathbf{u}^{CG}.\int_0^{\Delta t}t^2\left[(\mathbf{v}_a-\mathbf{v}_b)\times(\mathbf{v}_a-\mathbf{v}_c)\right]dt\right]}_{3} \quad \text{(A.14)}$$

Integrating the right hand side of Equation (A.14):

$$\left(V^1_{swept} + V^2_{swept}\right) = \underbrace{\left[\mathbf{u}^{CG}.\left[\mathbf{n}_0 A_0 + \mathbf{n}_1 A_1\right] t\right]_0^{\Delta t}}_{1} + \underbrace{\mathbf{u}^{CG}.\left[\left[(\mathbf{v}_a - \mathbf{v}_b) \times (\mathbf{v}_c - \mathbf{v}_a)\right](\Delta t)\frac{t^2}{2}\right]_0^{\Delta t}}_{2} + \underbrace{\mathbf{u}^{CG}.\left[\left[(\mathbf{v}_a - \mathbf{v}_b) \times (\mathbf{v}_a - \mathbf{v}_c)\right]\frac{t^3}{3}\right]_0^{\Delta t}}_{3} \quad \text{(A.15)}$$

$$\Rightarrow \left(V^1_{swept} + V^2_{swept}\right) = \underbrace{\mathbf{u}^{CG}.\left[\mathbf{n}_0 A_0 + \mathbf{n}_1 A_1\right](\Delta t)}_{1} - \underbrace{\mathbf{u}^{CG}.\left[(\mathbf{v}_a - \mathbf{v}_b) \times (\mathbf{v}_a - \mathbf{v}_c)\right]\frac{(\Delta t)^3}{2}}_{2} + \underbrace{\mathbf{u}^{CG}.\left[(\mathbf{v}_a - \mathbf{v}_b) \times (\mathbf{v}_a - \mathbf{v}_c)\right]\frac{(\Delta t)^3}{3}}_{3} \quad \text{(A.16)}$$

$$\Rightarrow \left(V^1_{swept} + V^2_{swept}\right) = \mathbf{u}^{CG}.\left[\mathbf{n}_0 A_0 + \mathbf{n}_1 A_1\right](\Delta t) - \mathbf{u}^{CG}.\left[(\mathbf{v}_a - \mathbf{v}_b) \times (\mathbf{v}_a - \mathbf{v}_c)\right]\frac{(\Delta t)^3}{6} \quad \text{(A.17)}$$

Now we can proceed for calculating the average of V^1_{swept} and V^2_{swept}

$$\frac{\left(V^1_{swept} + V^2_{swept}\right)}{2} = \frac{\mathbf{u}^{CG}}{2}.\left[\left[\mathbf{n}_0 A_0 + \mathbf{n}_1 A_1\right](\Delta t)\right] - \frac{\mathbf{u}^{CG}}{2}.\left[(\mathbf{v}_a \times \mathbf{v}_a) - (\mathbf{v}_a \times \mathbf{v}_c) - (\mathbf{v}_b \times \mathbf{v}_a) + (\mathbf{v}_b \times \mathbf{v}_c)\right]\frac{(\Delta t)^3}{6} \quad \text{(A.18)}$$

Let $\frac{\left(V^1_{swept} + V^2_{swept}\right)}{2} = V_{swept}$

Equation(A.18) can be rearranged to get:

$$\frac{V_{swept}}{\Delta t} = \mathbf{u}^{CG}.\left[\frac{1}{2}\left(\mathbf{n}_0 A_0 + \mathbf{n}_1 A_1\right) - \frac{(\Delta t)^2}{12}\left[(\mathbf{v}_a \times \mathbf{v}_b) + (\mathbf{v}_b \times \mathbf{v}_c) + (\mathbf{v}_c \times \mathbf{v}_a)\right]\right] \tag{A.19}$$

In [27], Perot and Nallapati denote the face vertices as **n1**, **n2** and **n3** which correspond to **a**, **b**, **c** respectively. The left hand side of the equation is denoted as U_f^{mesh} which corresponds to $\frac{V_{swept}}{\Delta t}$ in Equation (A.19). The notations for A_0 and A_1 are denoted as A_f^t and A_f^{t+1} respectively. The notations for $\mathbf{n}_0$ and $\mathbf{n}_1$ are n_f^t and n_f^{t+1} respectively.

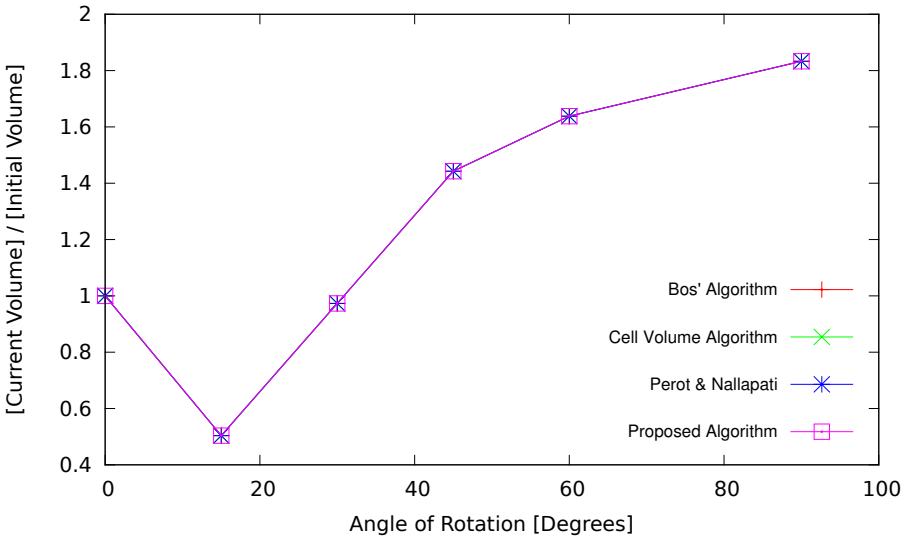
Thus as per the notation in Perot and Nallapati [27], Equation (A.19) can be represented in as:

$$U_f^{mesh} = \mathbf{u}^{CG}.\left[\frac{1}{2}\left(\mathbf{n}_f^t A_f^t + \mathbf{n}_f^{t+1} A_f^{t+1}\right) - \frac{(\Delta t)^2}{12}\left[(\mathbf{v}_{n1} \times \mathbf{v}_{n2}) + (\mathbf{v}_{n2} \times \mathbf{v}_{n3}) + (\mathbf{v}_{n3} \times \mathbf{v}_{n1})\right]\right] \tag{A.20}$$

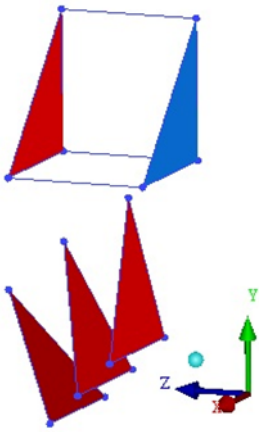
$$\Rightarrow \quad \boxed{U_f^{mesh} = \mathbf{u}^{CG}.\left[\frac{1}{2}\left(\mathbf{n}_f^t A_f^t + \mathbf{n}_f^{t+1} A_f^{t+1}\right) - \frac{(\Delta t)^2}{12}\sum_{e}^{face\ edges}(\mathbf{v}_{n1} \times \mathbf{v}_{n2})\right]} \tag{A.21}$$

APPENDIX B

TESTING RESULTS

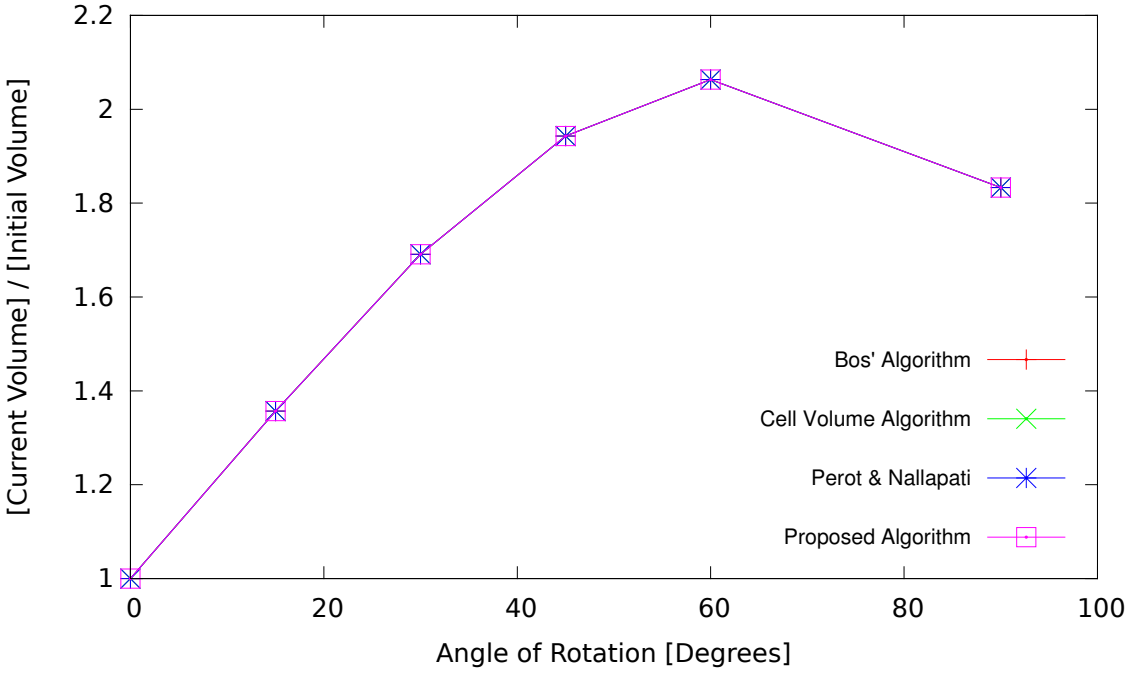


(a)

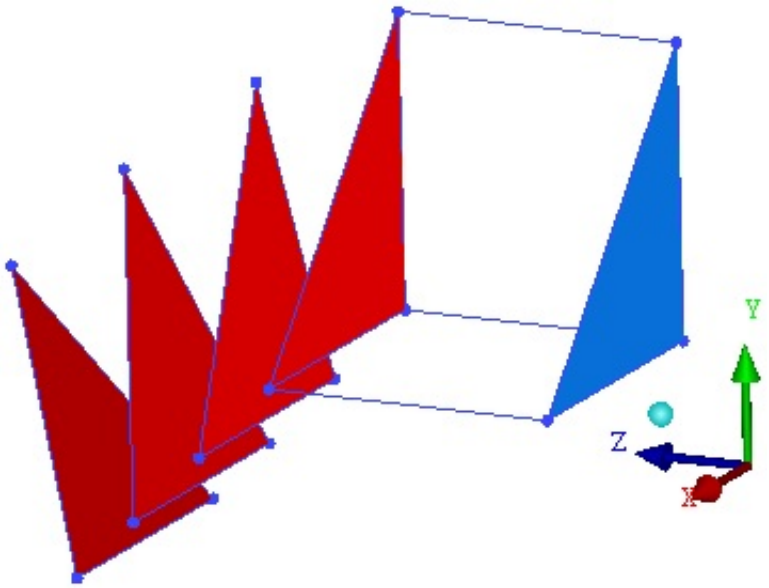


(b)

Figure B.1: Rotation about the X-axis with translation along the Y-axis.

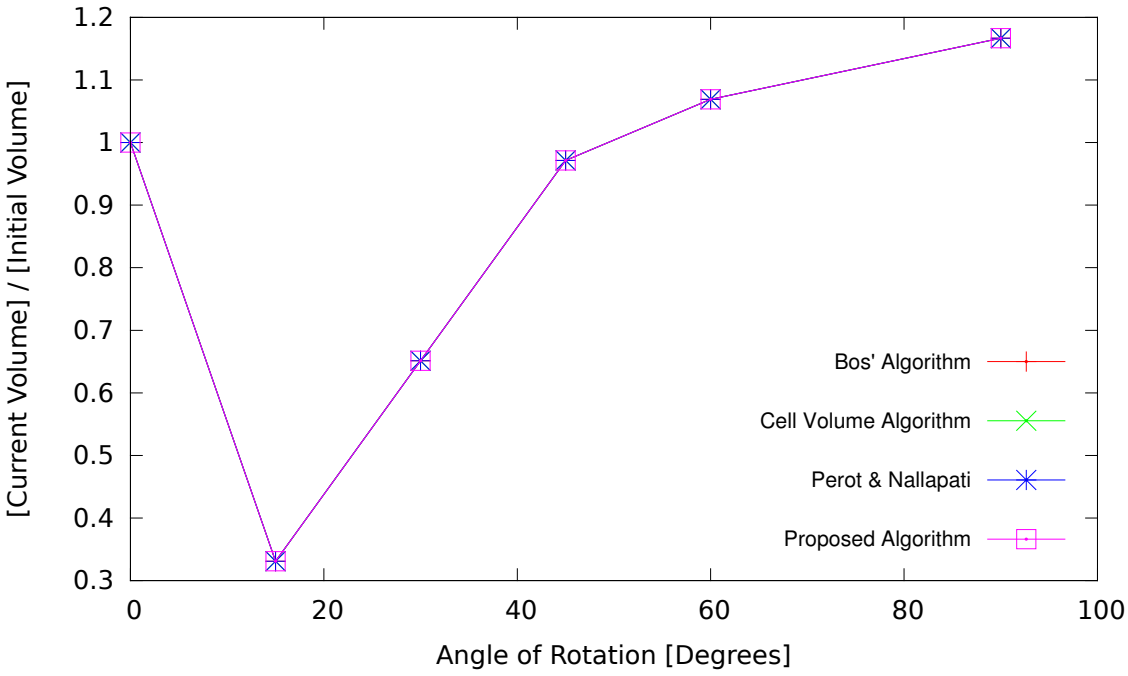


(a)

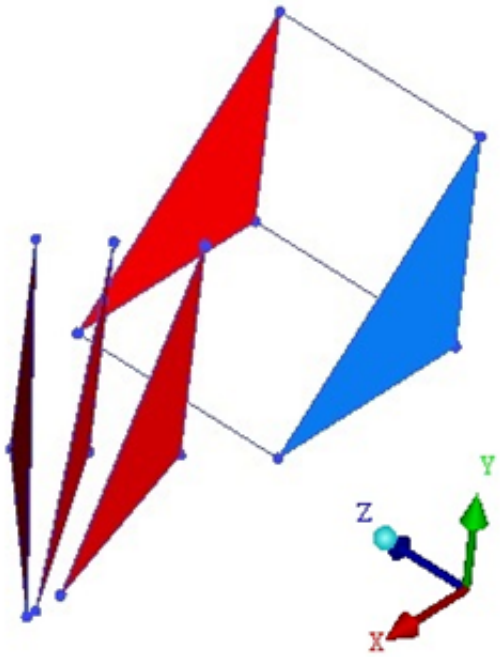


(b)

Figure B.2: Rotation about the X-axis with translation along the Z-axis.

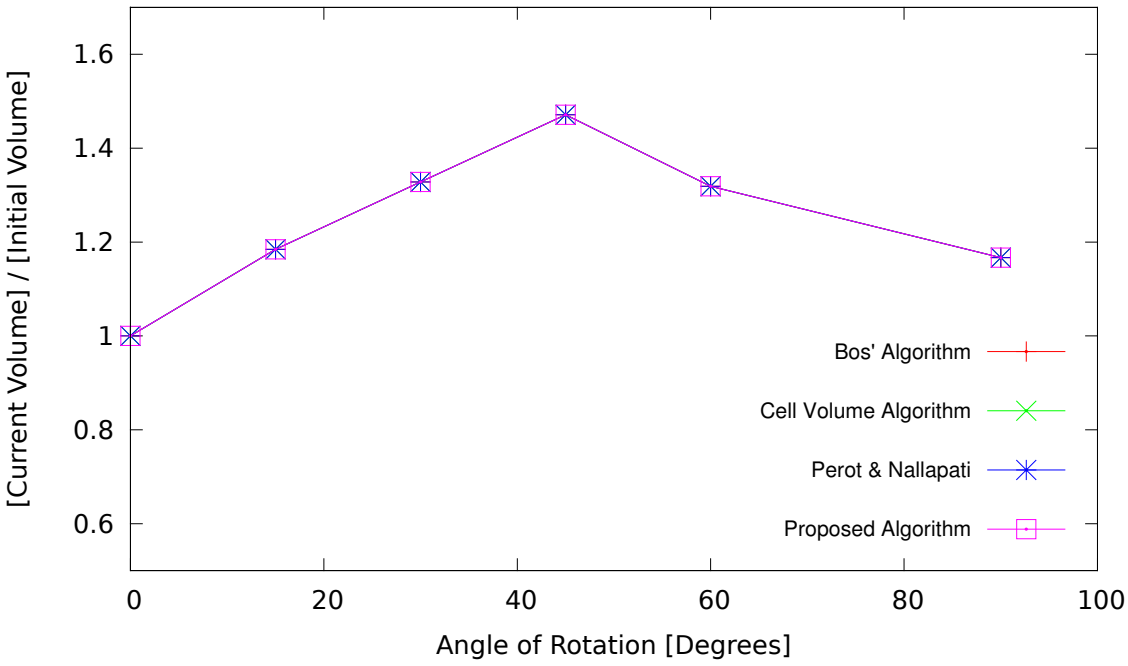


(a)

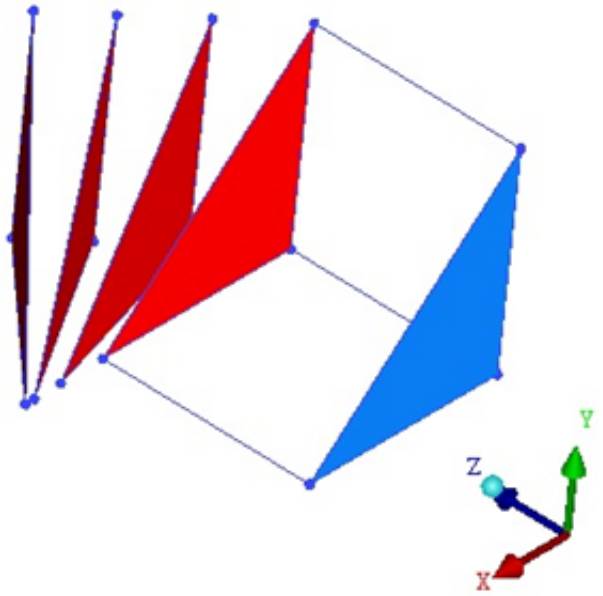


(b)

Figure B.3: Rotation about the Y-axis with translation along the X-axis.

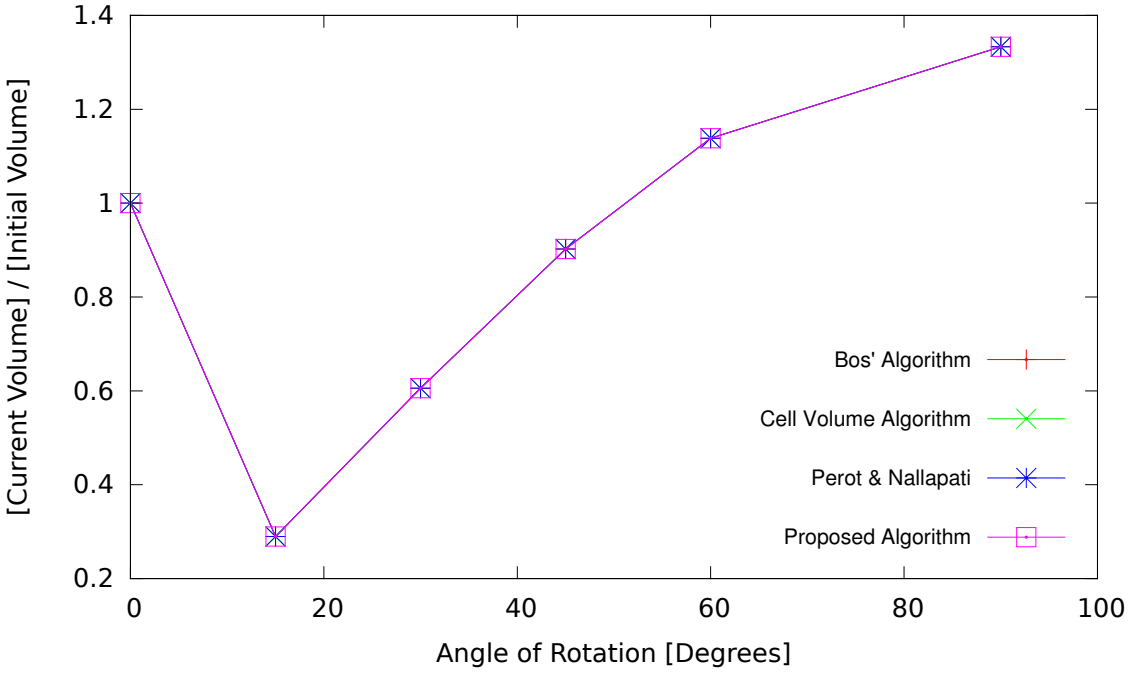


(a)

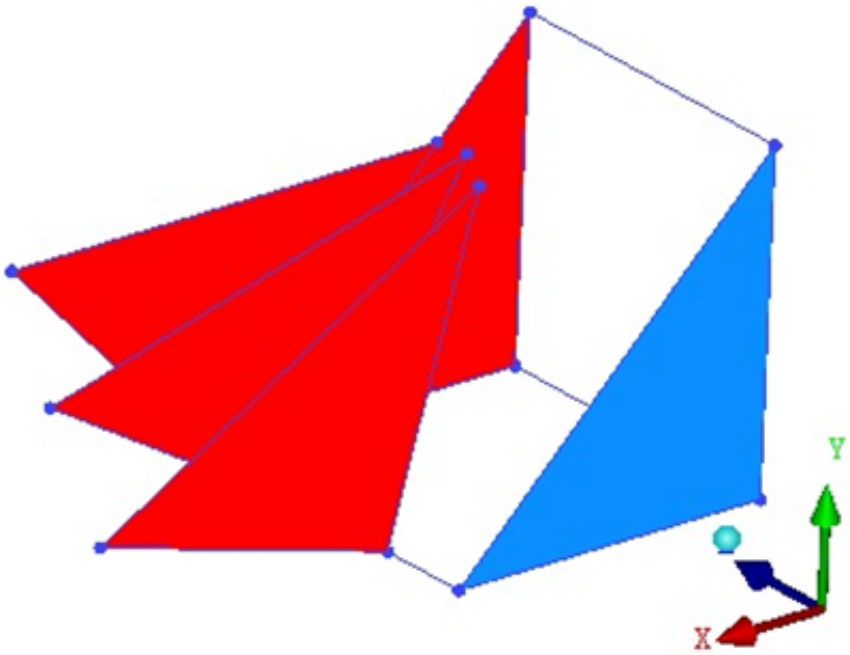


(b)

Figure B.4: Rotation about the Y-axis with translation along the Z-axis.

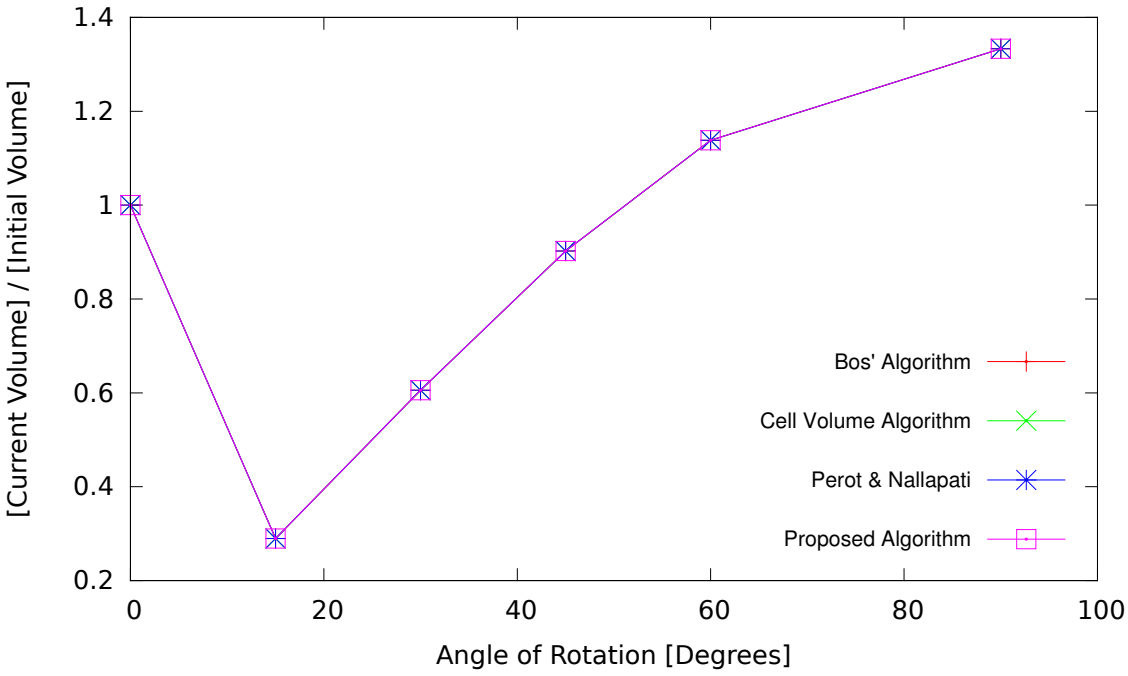


(a)

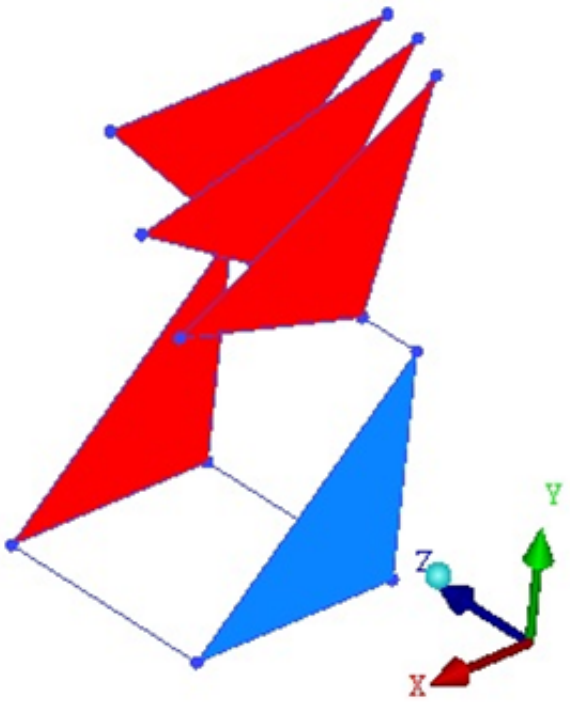


(b)

Figure B.5: Rotation about the Z-axis with translation along the X-axis.

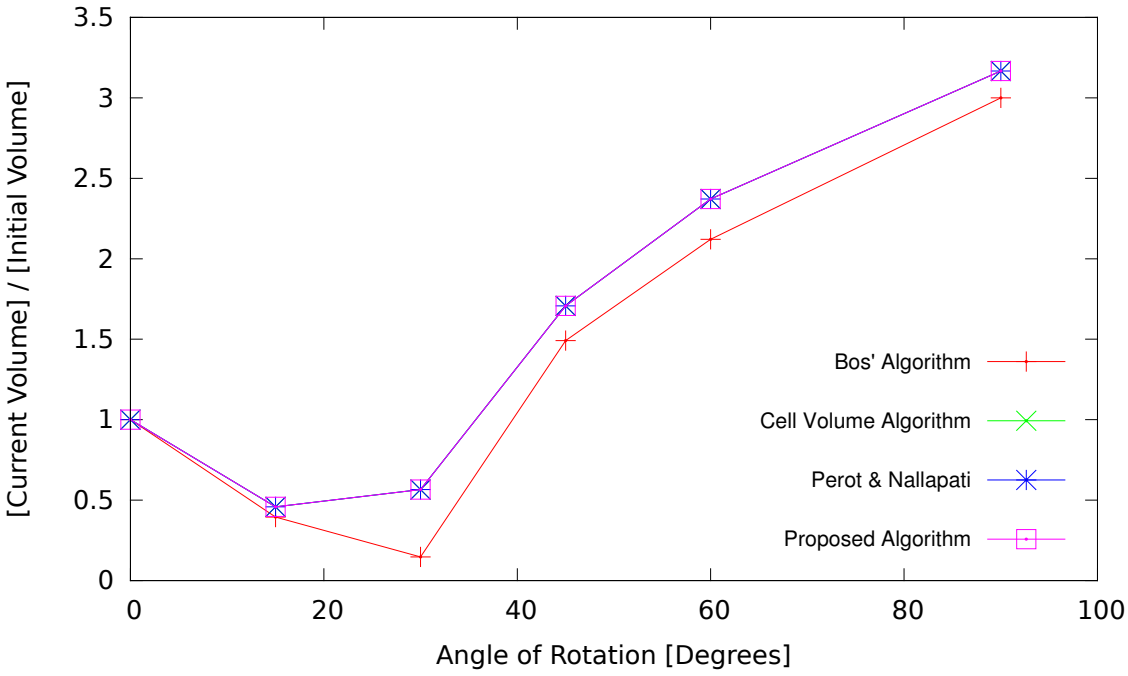


(a)

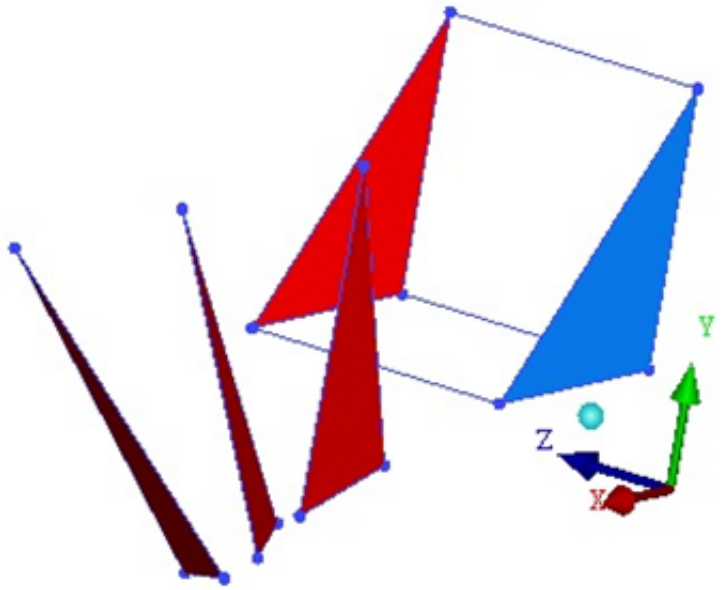


(b)

Figure B.6: Rotation about the Z-axis with translation along the Y-axis.

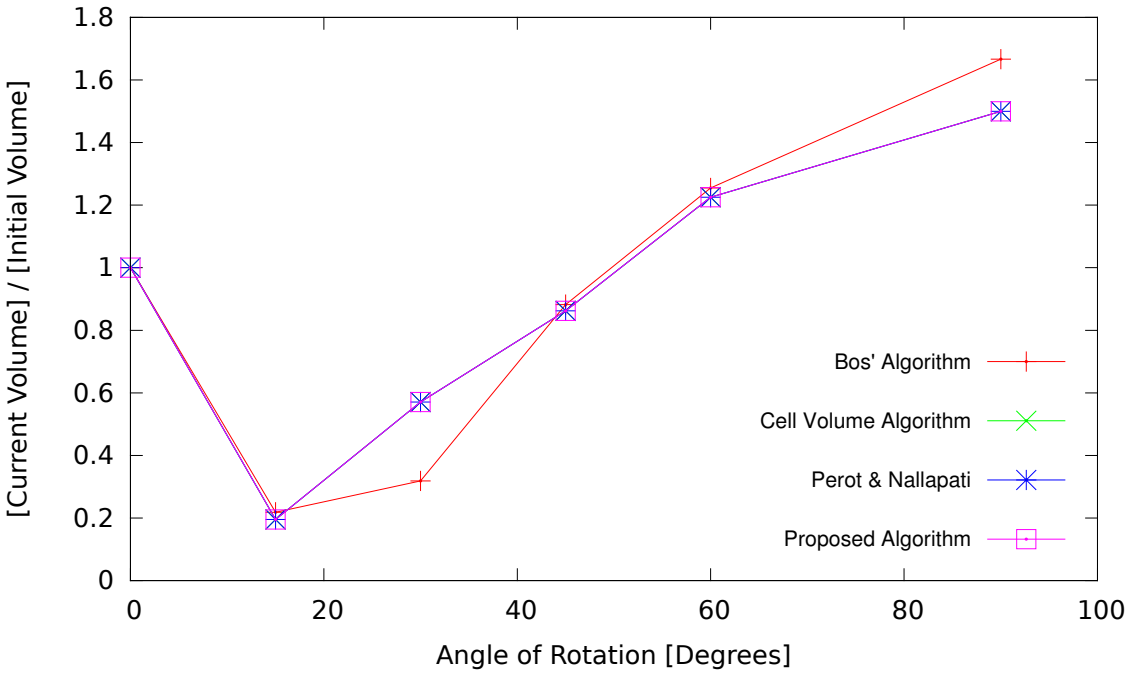


(a)

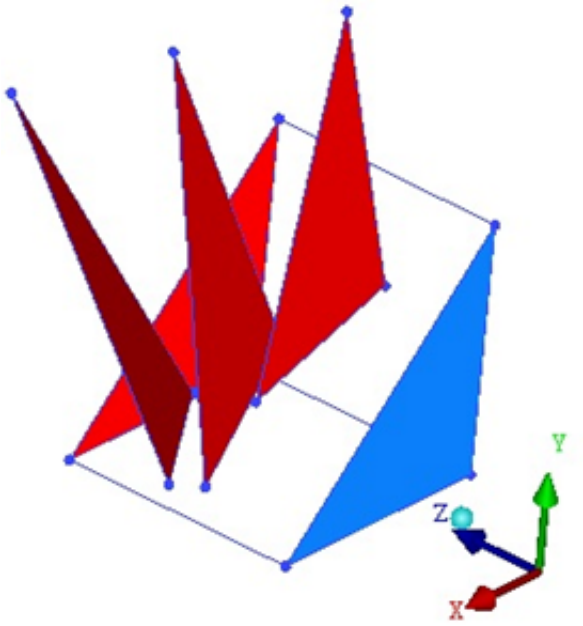


(b)

Figure B.7: Rotation about the X and Y-axis simultaneously with translation along the X-axis.

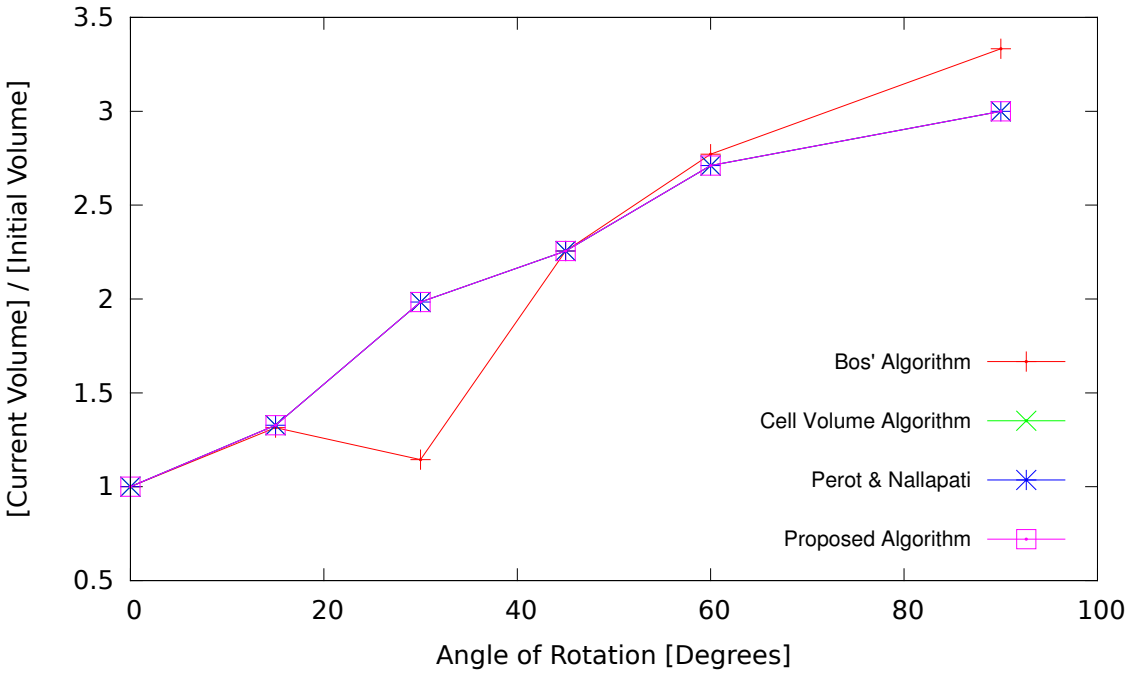


(a)

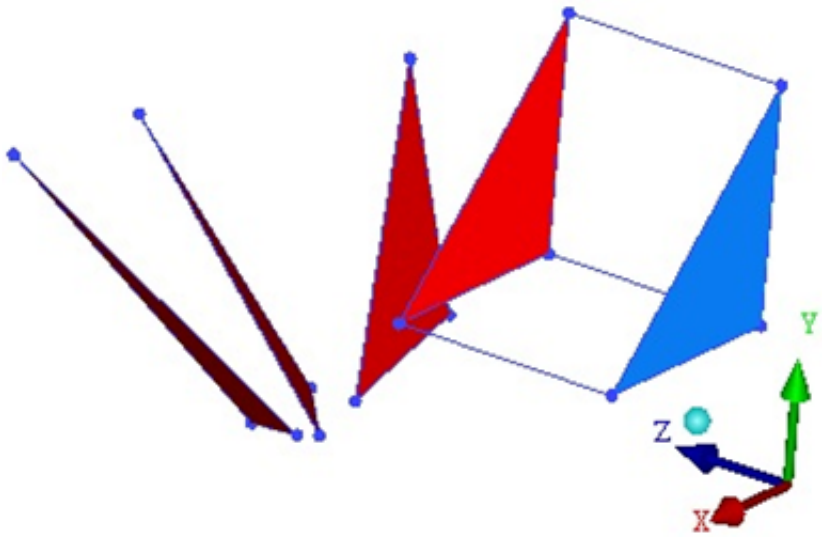


(b)

Figure B.8: Rotation about the X and Y-axis simultaneously with translation along the Y-axis.

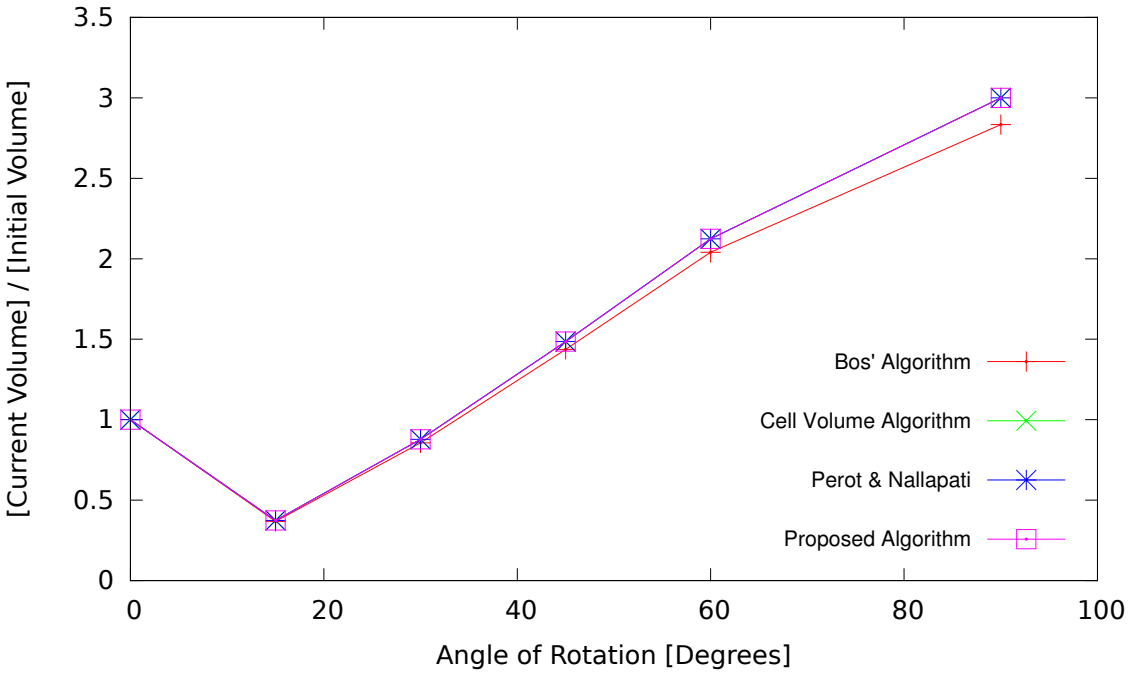


(a)

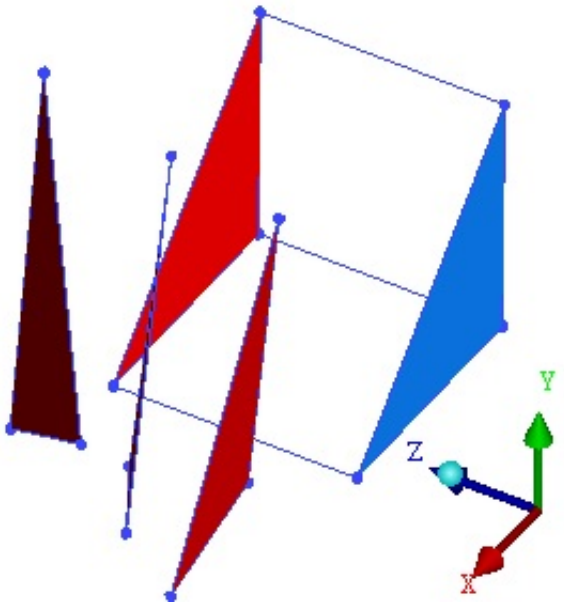


(b)

Figure B.9: Rotation about the X and Y-axis simultaneously with translation along the Z-axis.

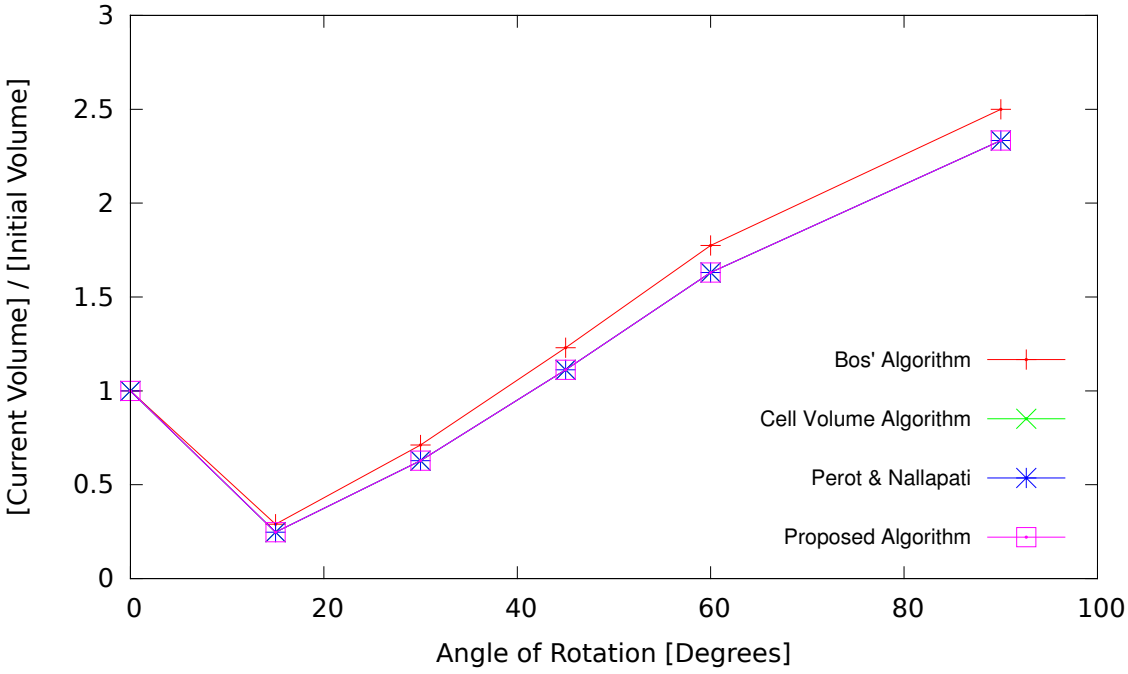


(a)

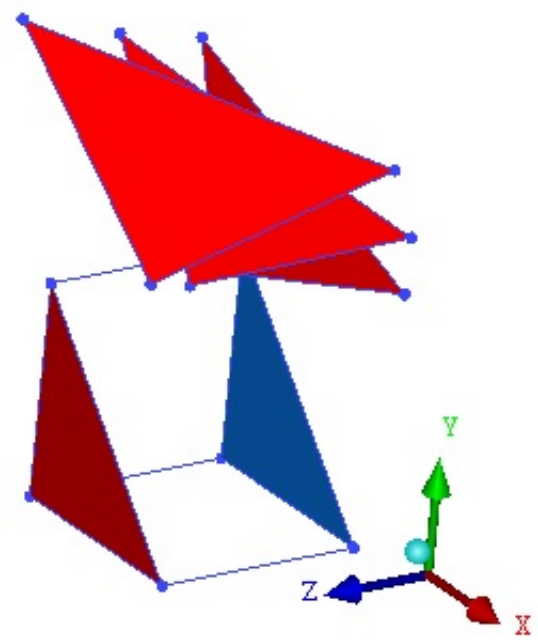


(b)

Figure B.10: Rotation about the Y and Z-axis simultaneously with translation along the X-axis.

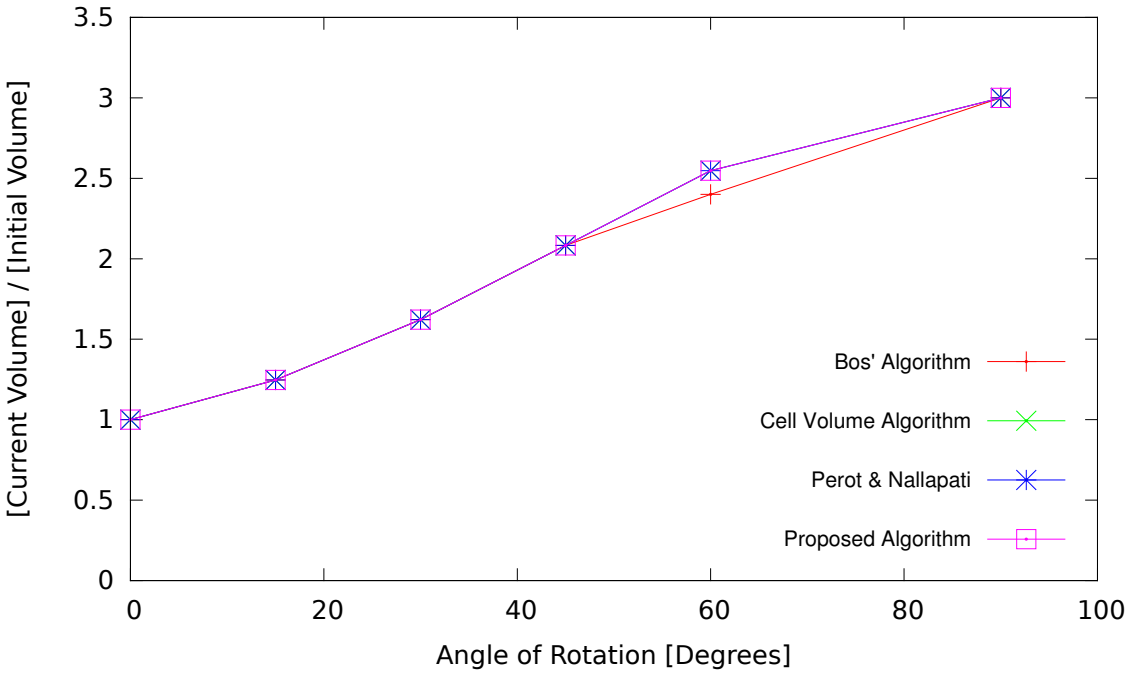


(a)

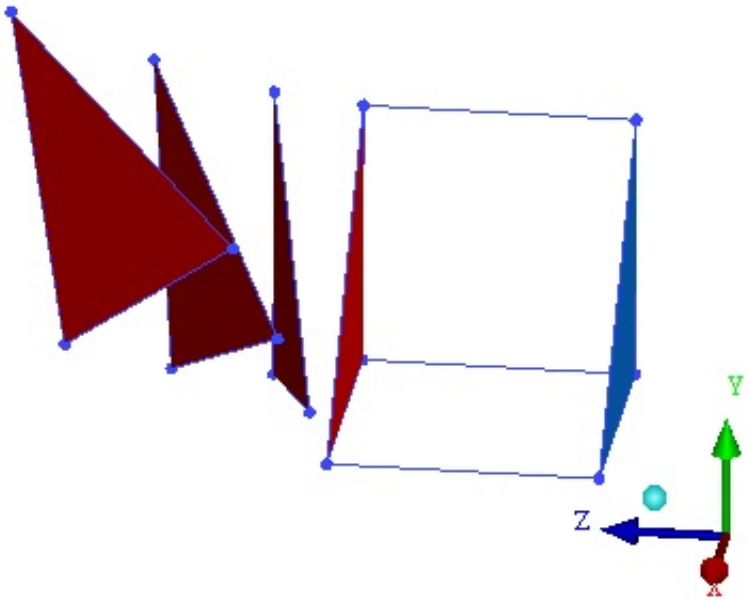


(b)

Figure B.11: Rotation about the Y and Z-axis simultaneously with translation along the Y-axis.

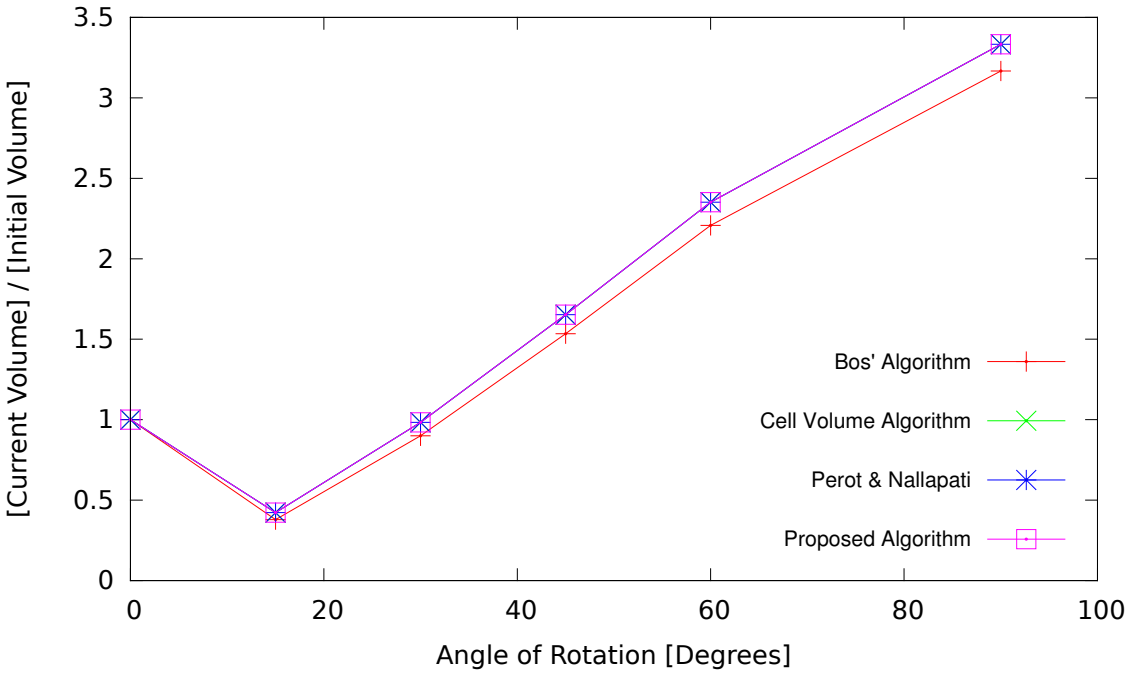


(a)

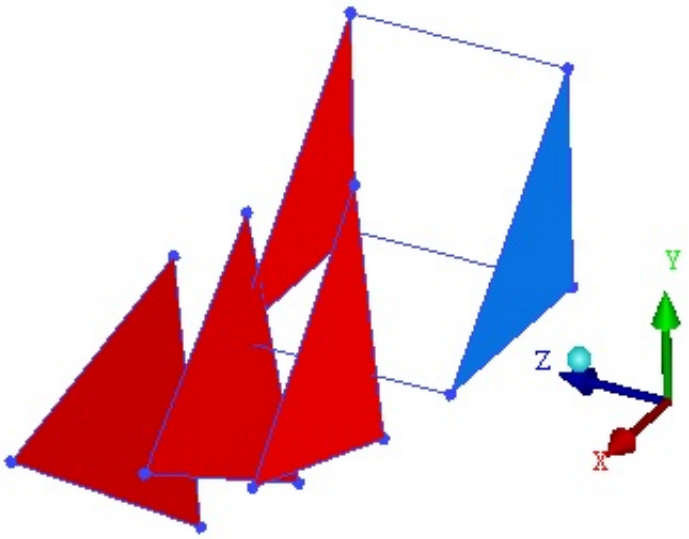


(b)

Figure B.12: Rotation about the Y and Z-axis simultaneously with translation along the Z-axis.

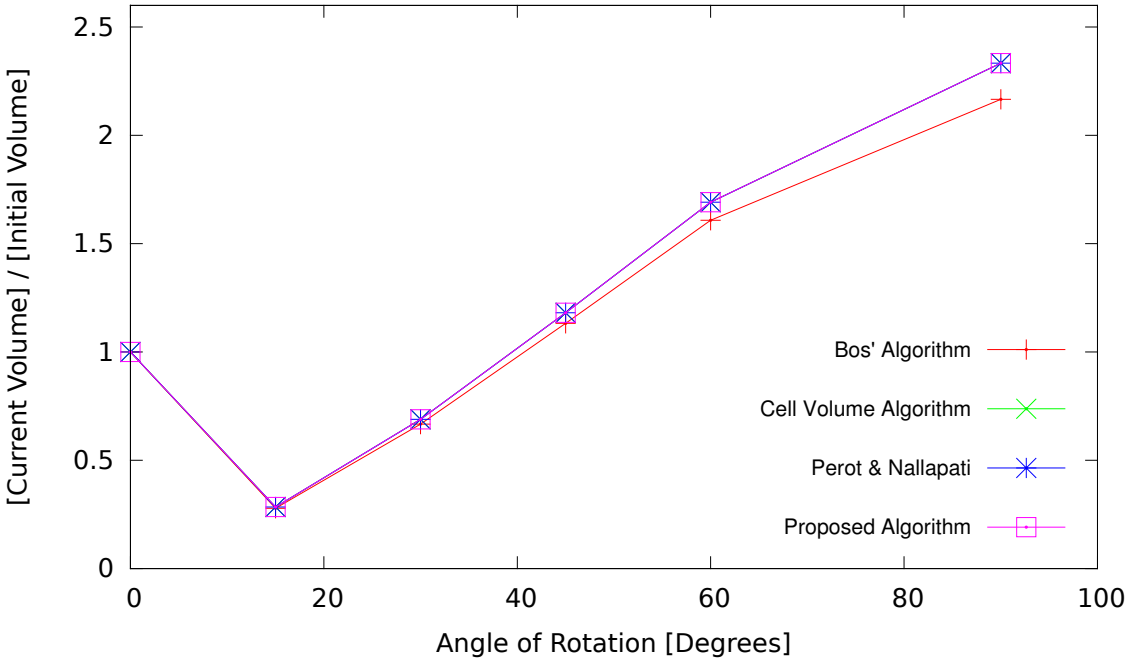


(a)

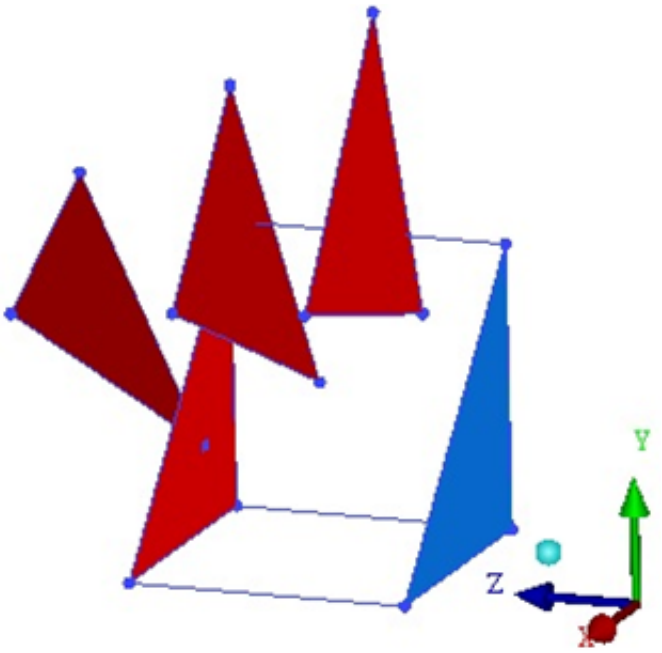


(b)

Figure B.13: Rotation about the Z and X-axis simultaneously with translation along the X-axis.

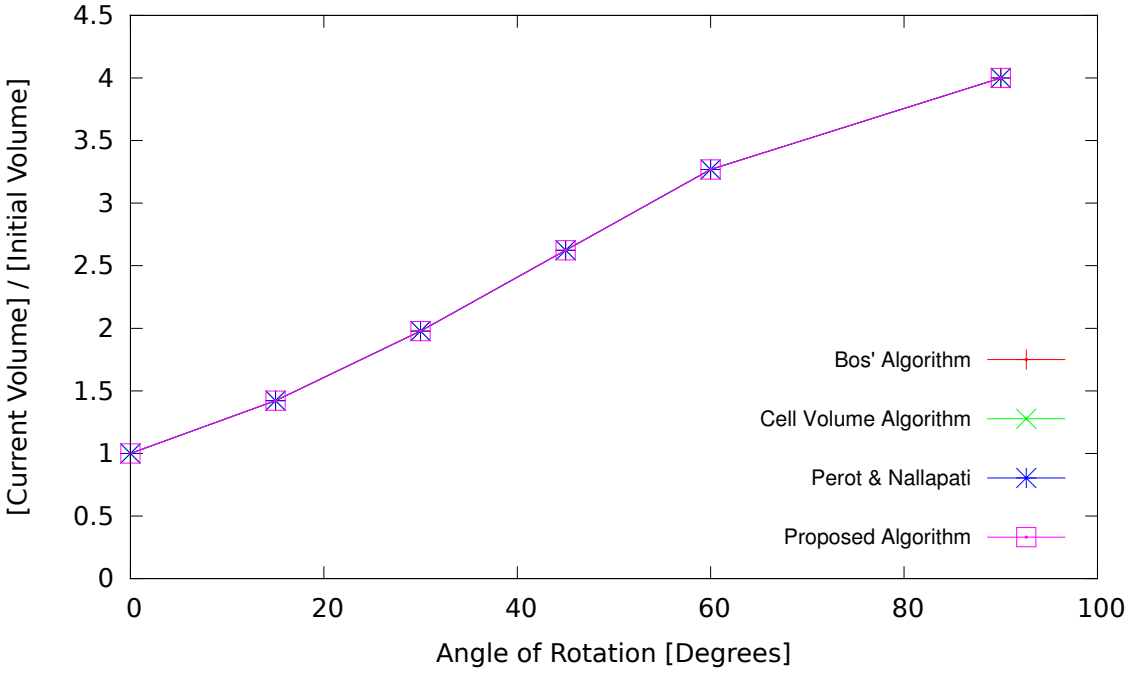


(a)

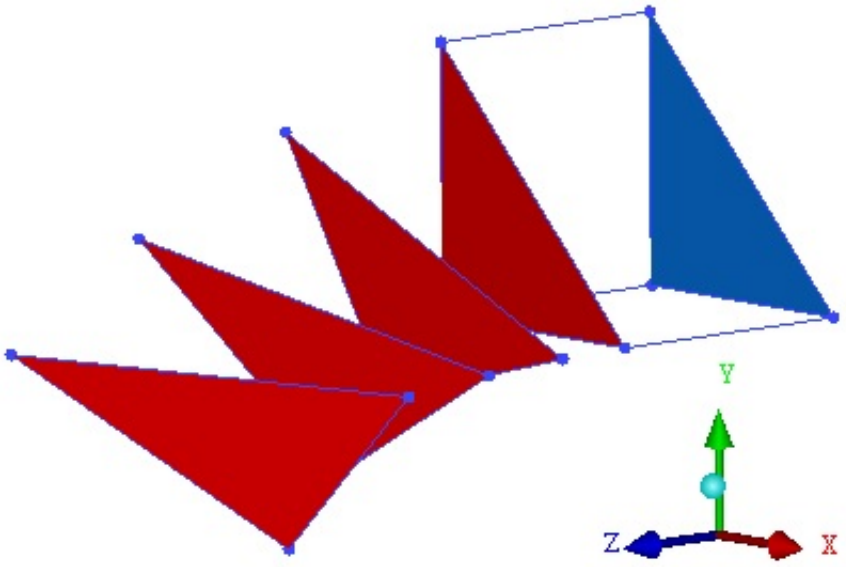


(b)

Figure B.14: Rotation about the Z and X-axis simultaneously with translation along the Y-axis.

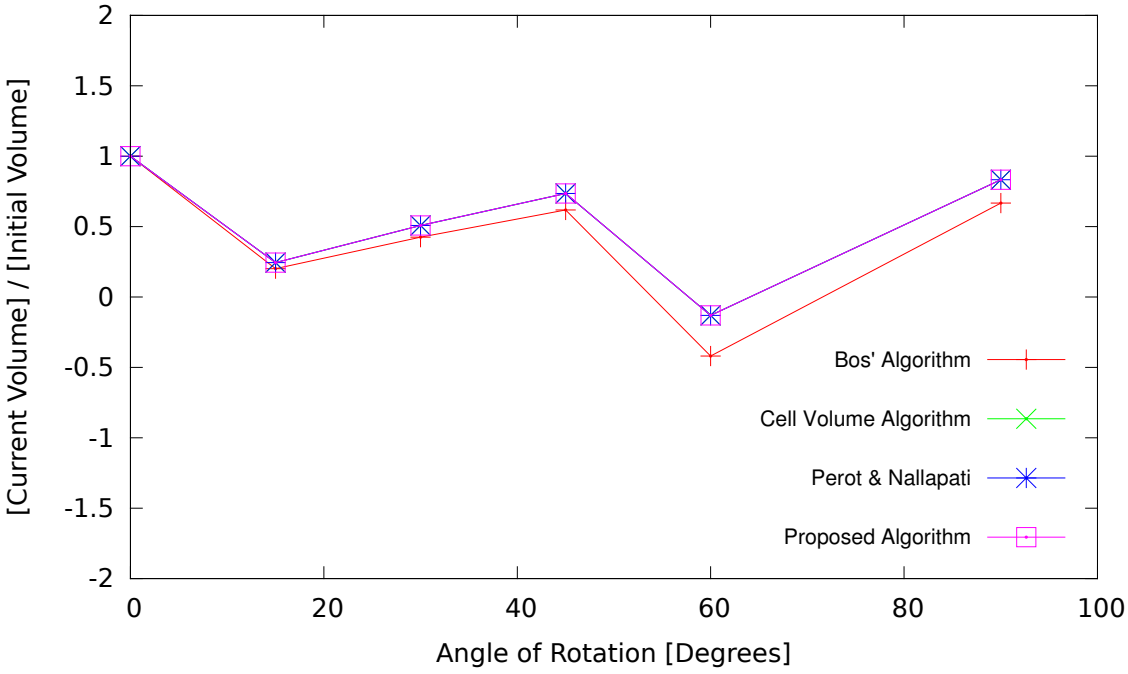


(a)

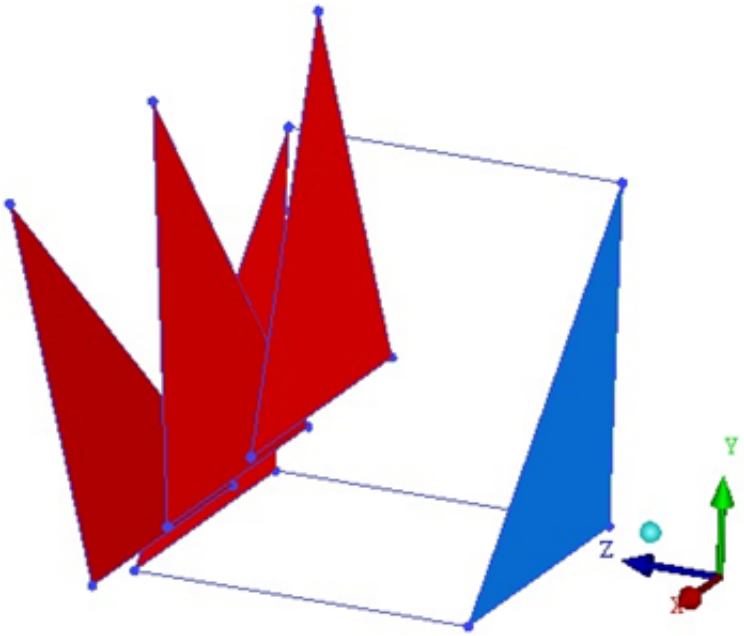


(b)

Figure B.15: Rotation about the Z and X-axis simultaneously with translation along the Z-axis.

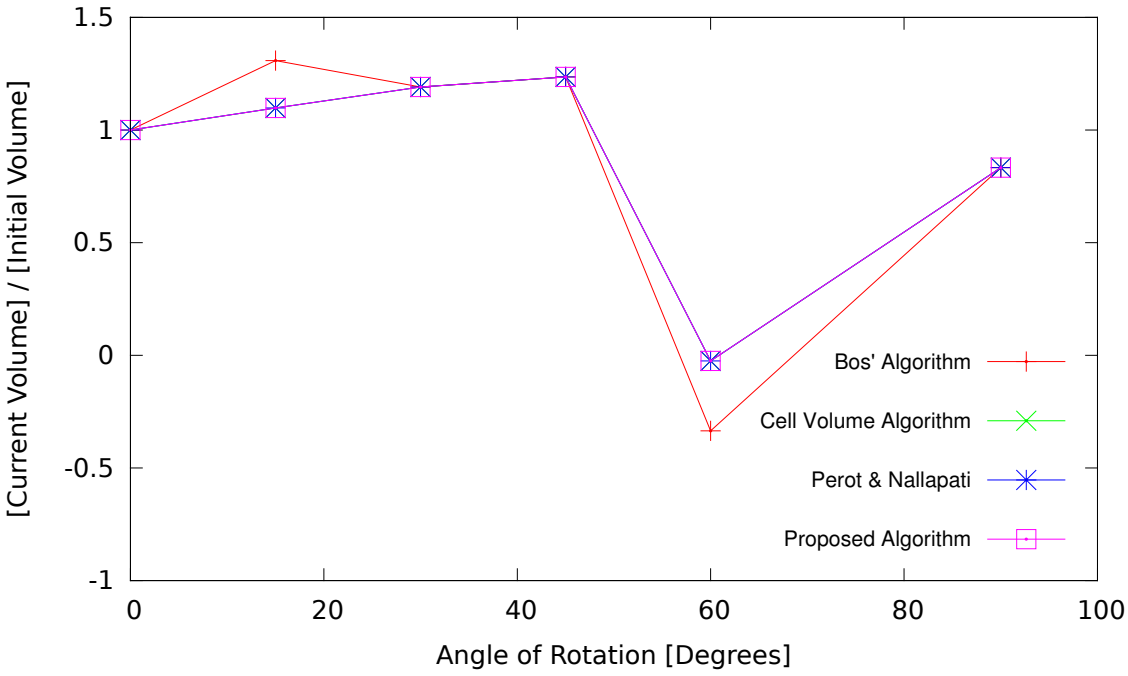


(a)

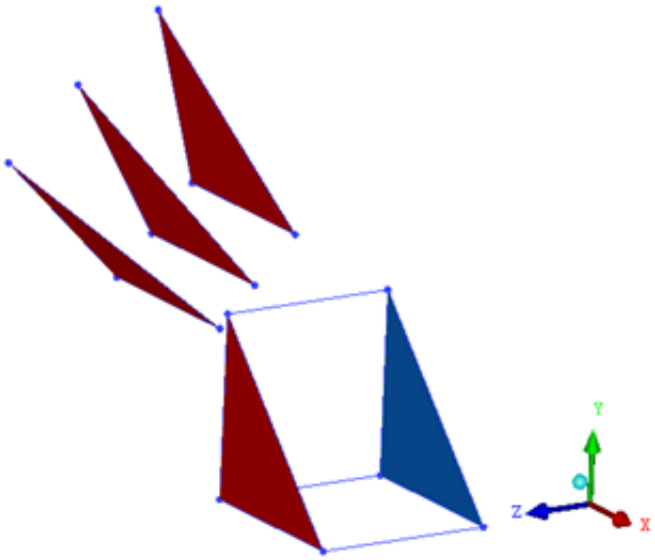


(b)

Figure B.16: Rotation about the X-axis with translation along the X and Y-axis simultaneously.

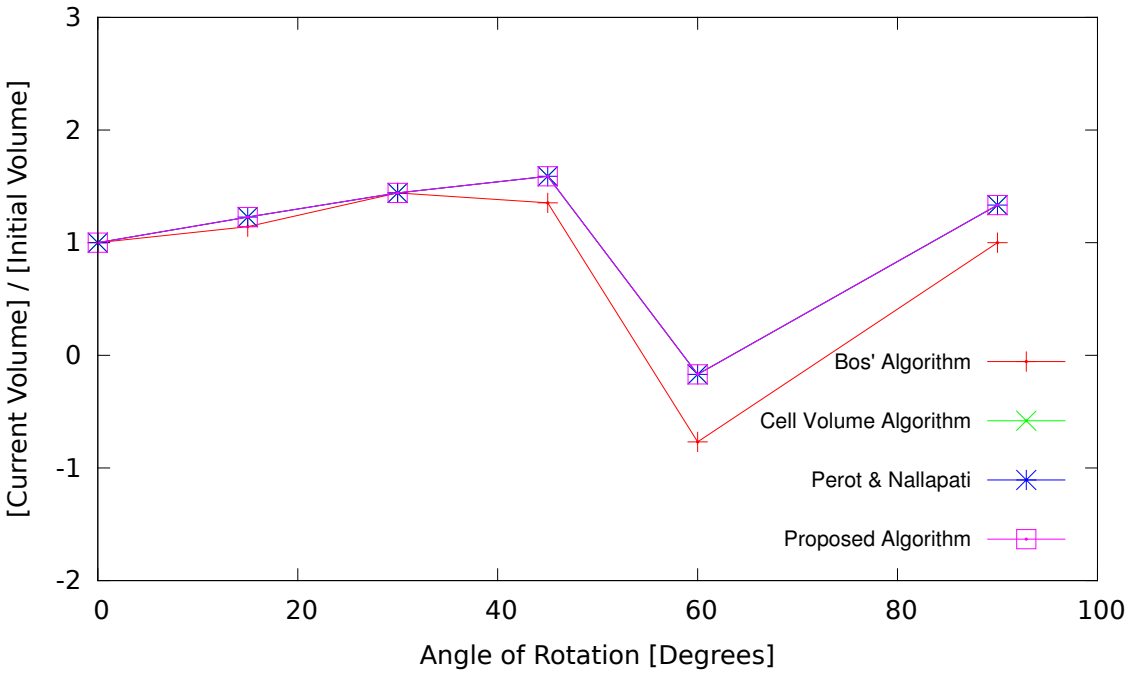


(a)

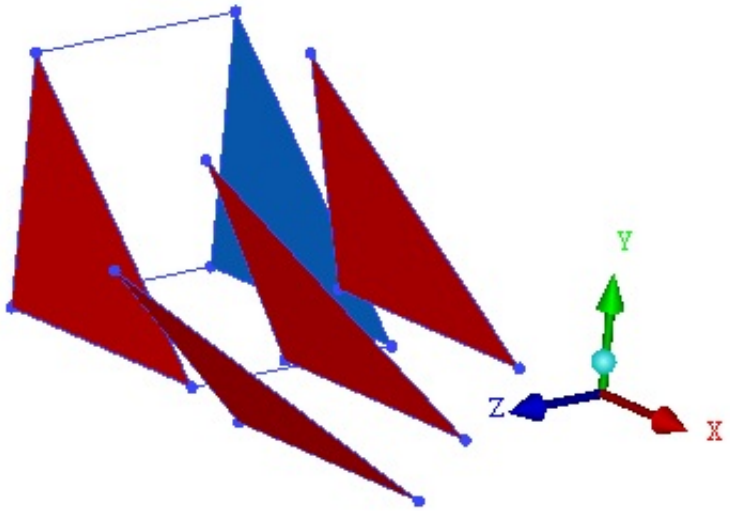


(b)

Figure B.17: Rotation about the X-axis with translation along the Y and Z-axis simultaneously.

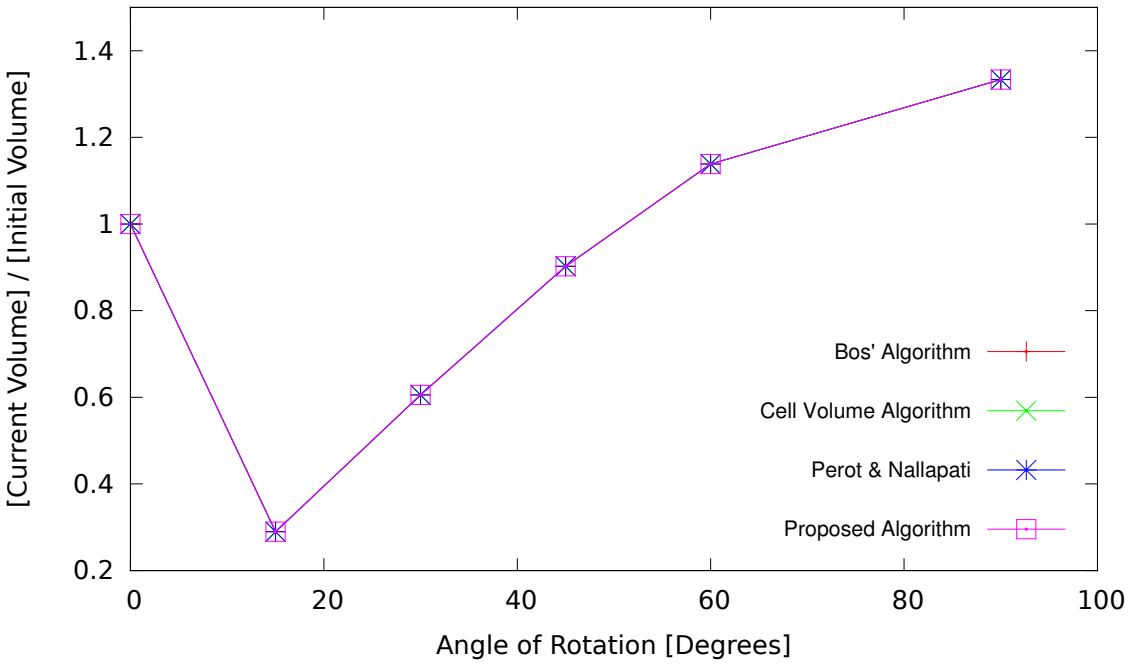


(a)

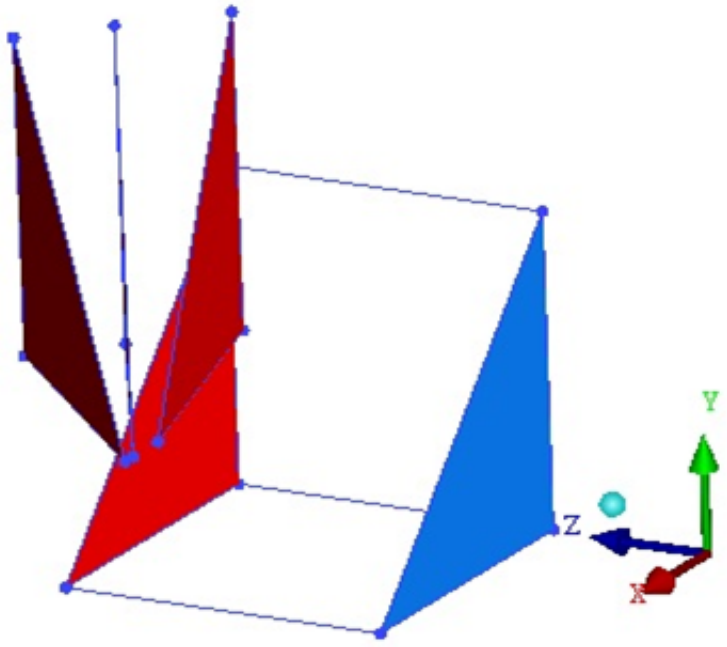


(b)

Figure B.18: Rotation about the X-axis with translation along the Z and X-axis simultaneously.

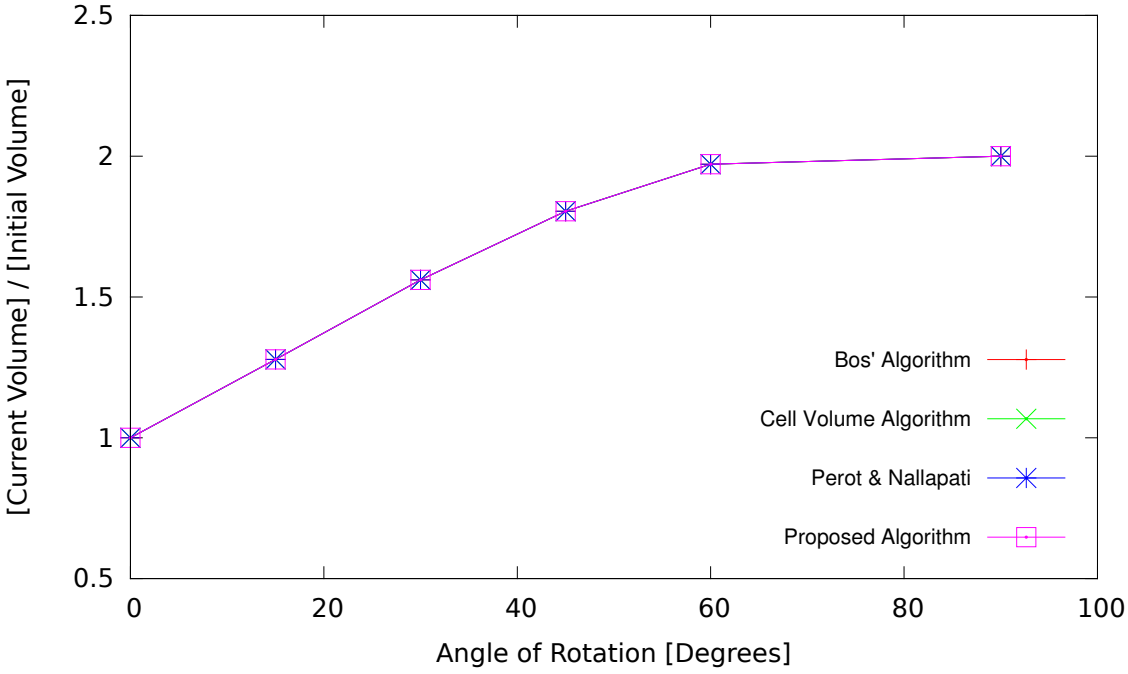


(a)

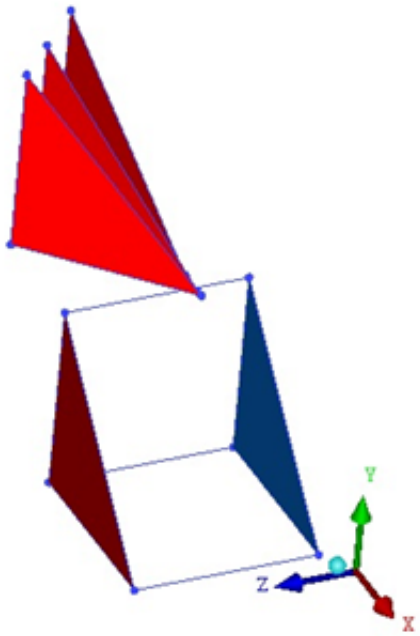


(b)

Figure B.19: Rotation about the Y-axis with translation along the X and Y-axis simultaneously.

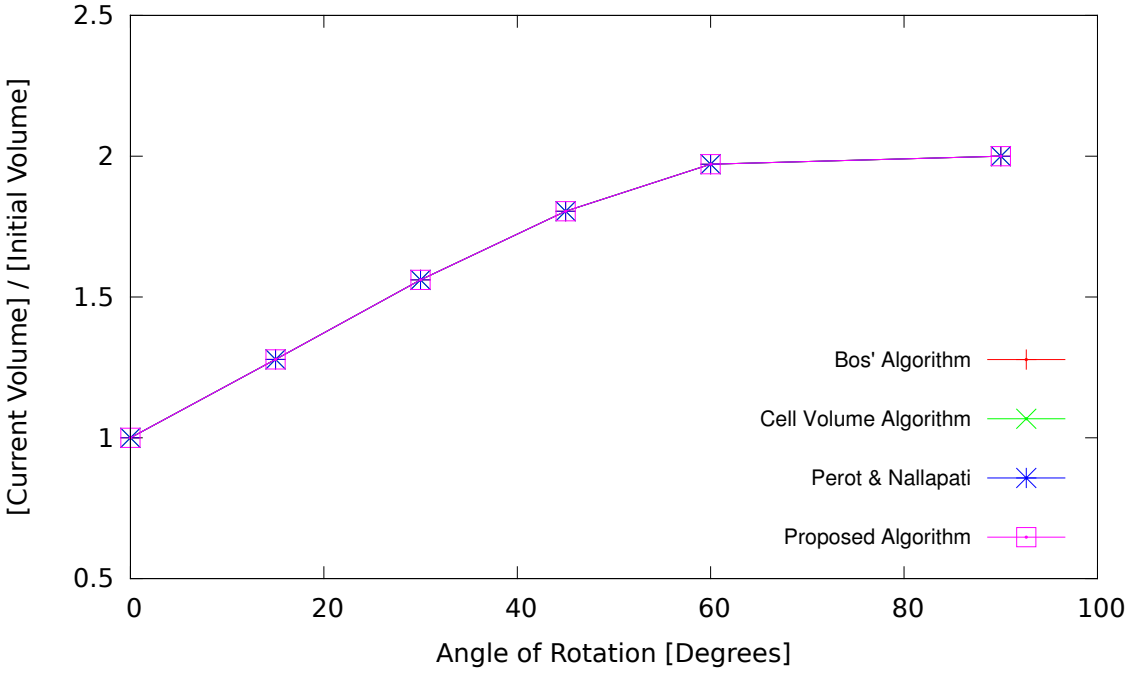


(a)

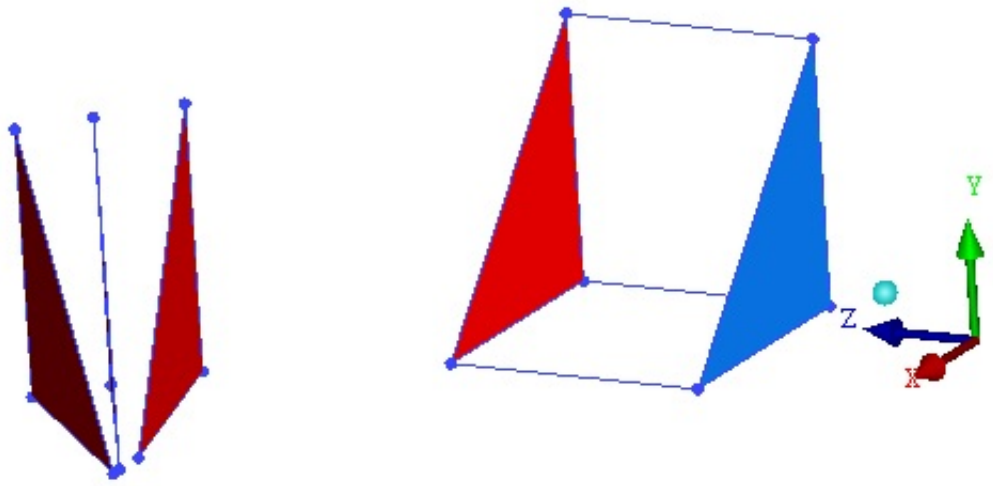


(b)

Figure B.20: Rotation about the Y-axis with translation along the Y and Z-axis simultaneously.

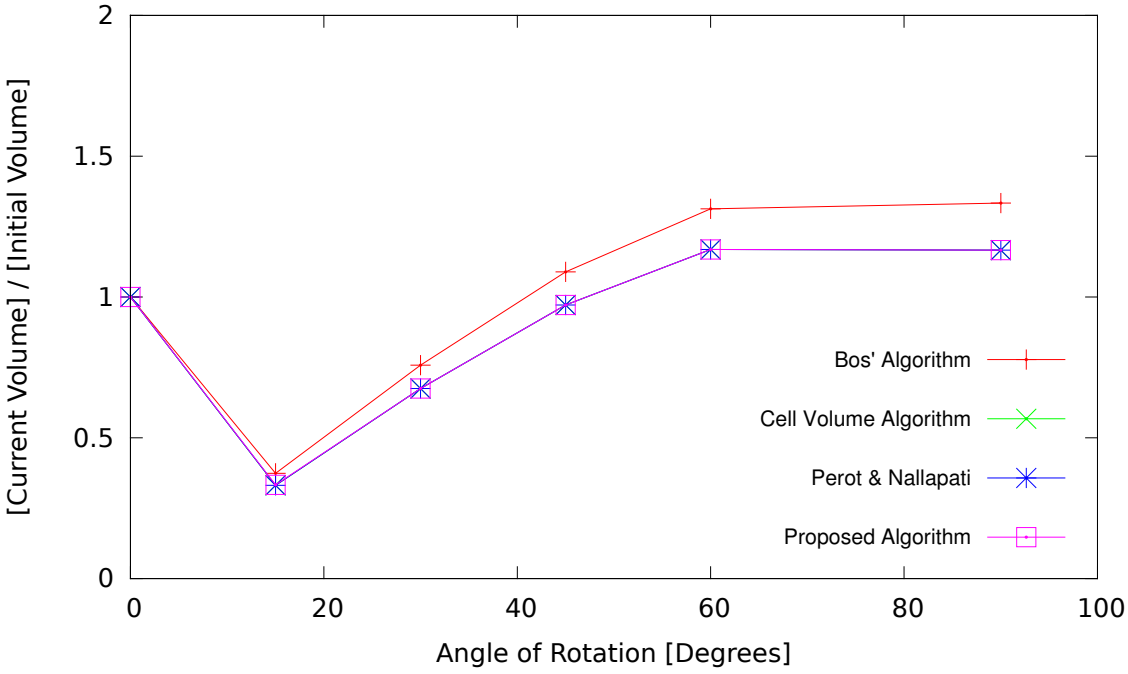


(a)

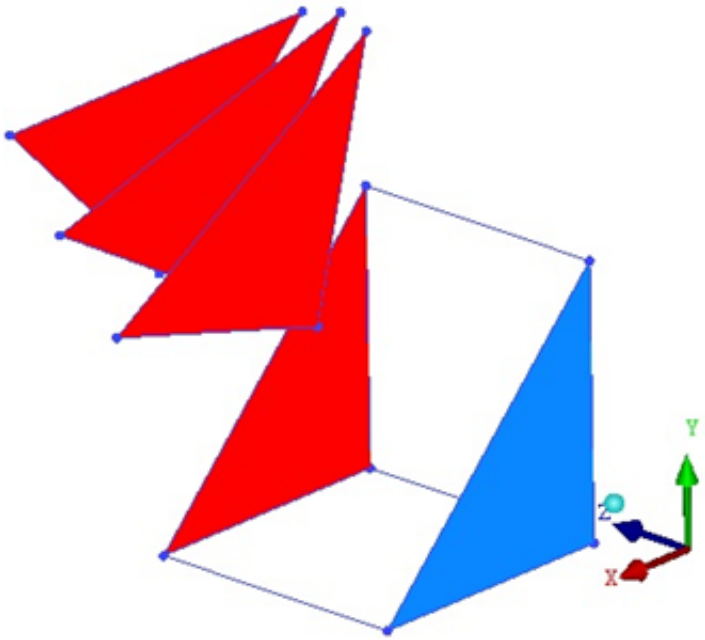


(b)

Figure B.21: Rotation about the Y-axis with translation along the Z and X-axis simultaneously.

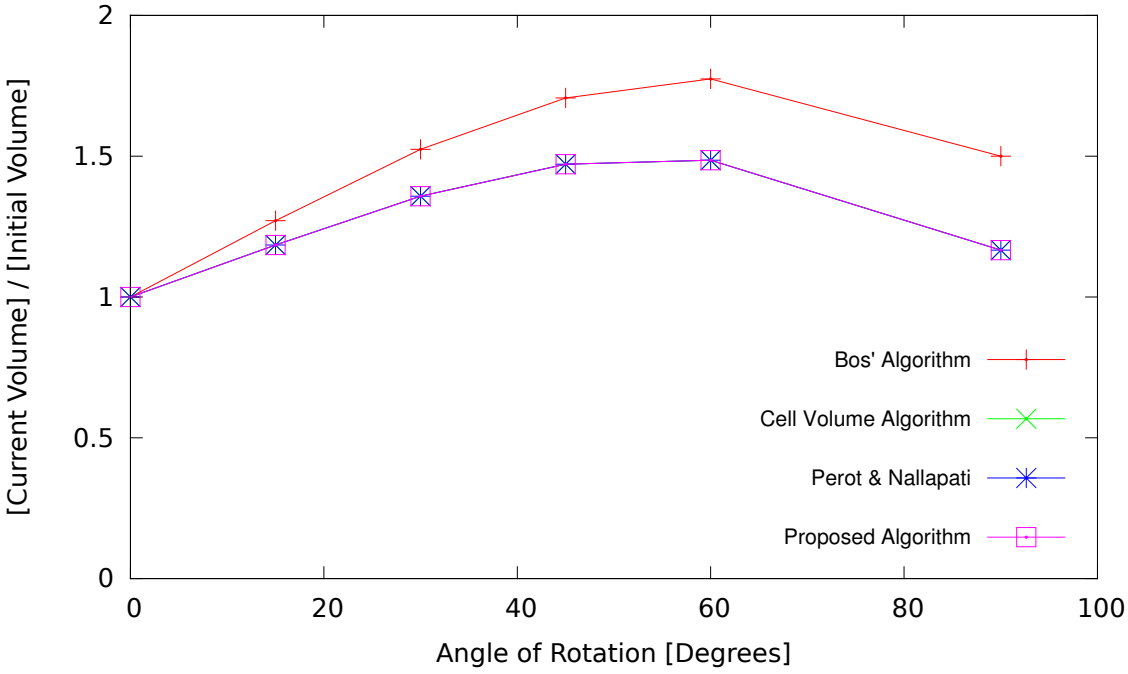


(a)

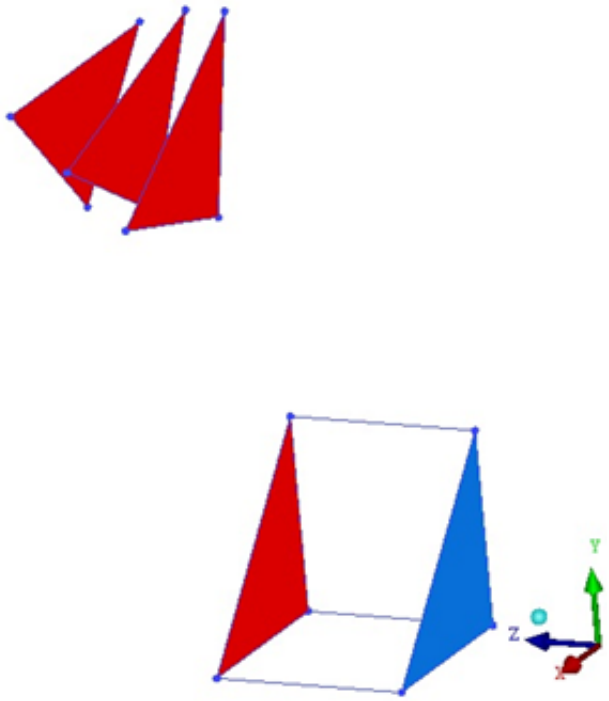


(b)

Figure B.22: Rotation about the Z-axis with translation along the X and Y-axis simultaneously.

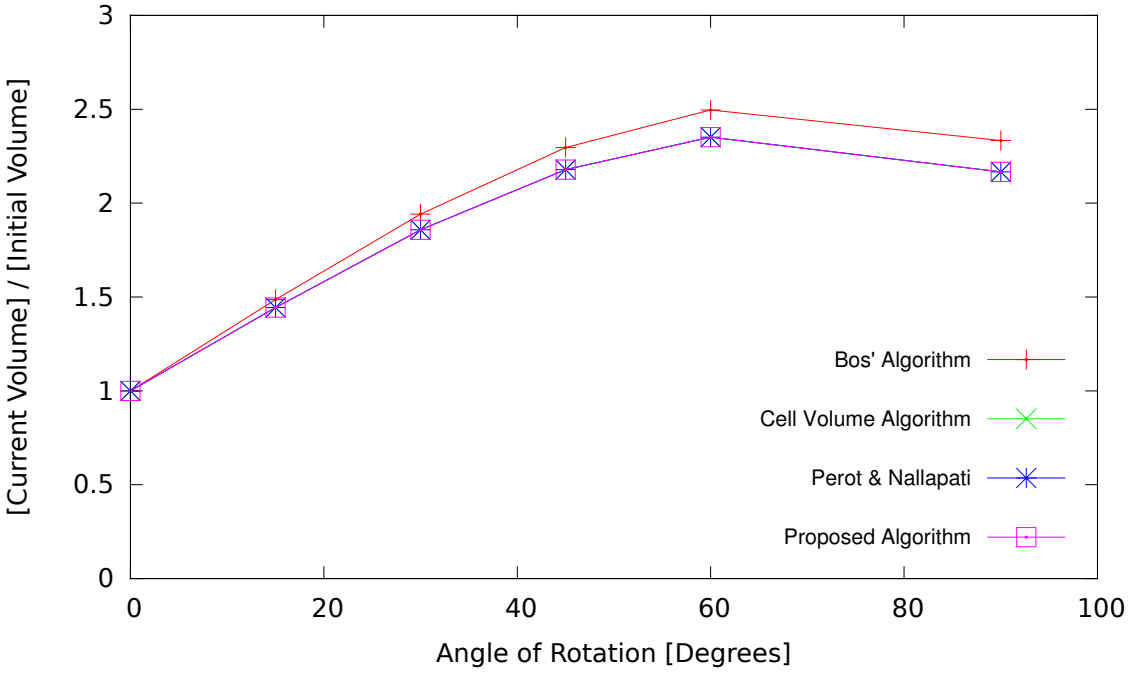


(a)

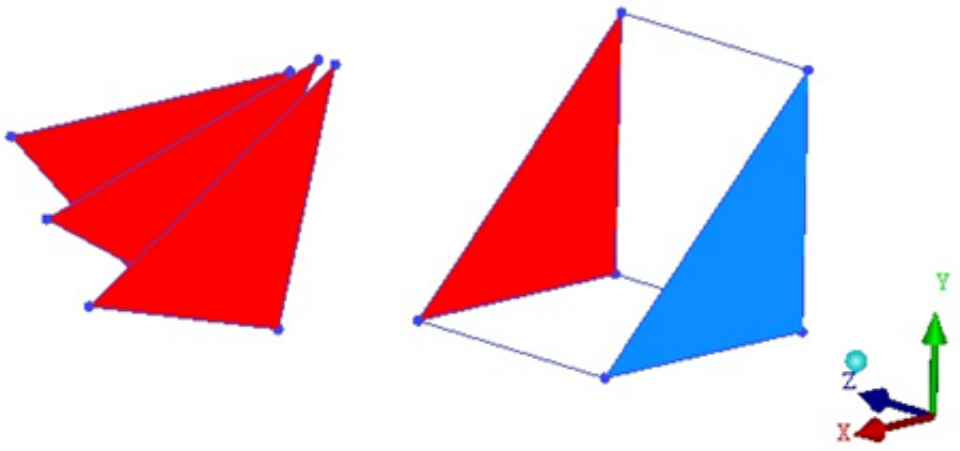


(b)

Figure B.23: Rotation about the Z-axis with translation along the Y and Z-axis simultaneously.



(a)



(b)

Figure B.24: Rotation about the Z-axis with translation along the Z and X-axis simultaneously.

BIBLIOGRAPHY

[1] Apsley, D, and Hu, W. CFD simulation of two- and three-dimensional free-surface flow. *International Journal for Numerical Methods in Fluids 42* (2003), 465–491.

[2] Aris, Rutherford. Reynolds' Transport Theorem. In *Vectors, Tensors and the Basic equations of fluid mechanics* (1989), New York - Dover Publications, pp. 84 –86.

[3] Baines, M J, Hubbard, M E, and Jamak, P K. Review article: Velocity-based moving mesh methods for non-linear partial differential equations. *Communications in Computational Physics 10*, 3 (2011).

[4] Boffi, D, and Gastaldi, L. Stability and Geometric Conservation Laws for ALE formulations. *Mathematics Subject Classification* (2003).

[5] Bos, F M. Numerical simulation of flapping wing foil and wing aerodynamics. *PhD Thesis, TU Delft* (2010).

[6] Cao, W, Huang, W, Udaykumar, H S, and Russel, D R. A moving mesh method base on the Geometric Conservation Law. *Journal of Scientific Computing 24*, 1 (2002), 118–142.

[7] Chou, Y J, and Fringer, O B. Consistent discretization for simulations of flows with moving generalized curvilinear coordinates. *International Journal for Numerical Methods in Fluids 62* (2010), 802–826.

[8] Dai, M. Numerical simulation of free surface flows using a moving unstructured mesh method. *PhD Thesis, University of Massachusetts, Amherst* (September 2005).

[9] Demirdzic, I, and Peric, M. Space conservation Law in finite volume calculations of fluid flow. *International Journal for Numerical Methods in Fluids 8*, 9 (1988), 1037–1050.

[10] Demirdzic, I, and Peric, M. Finite Volume Method for prediction of fluid flow in arbitrarily shaped domains with moving boundaries. *International Journal for Numerical Methods in Fluids 10*, 7 (1990), 771–790.

[11] Etienne, S, Garon, A, and Pelletier, D. Perspective on the geometric conservation law and finite element methods for ALE simulations of incompressible flow. *Journal of Computational Physics 228* (2009), 2313–2333.

[12] Ferziger, J H, and Peric, M. Conservation Principles. In *Computational Methods for Fluid Dynamics* (2008), Springer, pp. 3–4.

[13] Ferziger, J H, and Peric, M. Moving Grids. In *Computational Methods for Fluid Dynamics* (2008), Springer, pp. 373–381.

[14] Gopalakrishnan, P, and Tafti, D K. A parallel boundary fitted dynamic mesh solver for applications to flapping flight. *Computers and Fluids 38* (2009), 1592–1607.

[15] Guillard, H, and Farhat, C. On the significance of the geometric conservation law for flow computations on moving meshes. *Computer Methods in Applied Mechanics and Engineering 190* (200), 1467–1482.

[16] Hu, H H, Patankar, N A, and Zhu, M Y. Direct Numerical Simulations of Fluid-Solid systems using the Aribitrary Lagrangian-Eulerian technique. *Journal of Computational Physics 169* (2001), 427–462.

[17] Jasak, H, and Tukovic, Z. Automatic mesh motion for the unstructured finite volume method. *Elsevier Science* (February 2004).

[18] Jasak, H, and Tukovic, Z. Dynamic mesh handling in openfoam applied to fluid-structure interaction simulations. In *Proceedings of the fifth European conference in Computational Fluid Dynamics* (Lisbon, Portugal, 2010), J C F Pereira and A Sequeira, Eds., ECCOMAS CFD 2010.

[19] Jeng, Y N, and Chen, J L. Geometric Conservation Law of the finite volume method for the SIMPLER algorithm and a proposed upwind scheme. *Numerical Hear Transfer 22* (1992), 221–234.

[20] Koutsavdis, E K, and Tsangaris, S. Short communication: A potential model for Space Conservation Law calculations of moving curvilinear meshes. *International Journal of Computational Fluid Dynamics 6*, 3 (1996), 207–208.

[21] Kuhnlein, C, Smolarkiewicz, P K, and Dornbrack, A. Modelling atmospheric flows with adaptive moving meshes. *Journal of Computational Physics 231* (2012), 2741–2763.

[22] Li, S, and Petzold, L. Moving Mesh Methods with Upwinding schemes for Time Dependent PDEs. *Journal of Computational Physics 131* (1997), 368–377.

[23] Maric, T, Marschall, H, and Bothe, D. voFoam - A geometrical Volume of Fluid algorithm on arbitrary unstructured meshes with local dynamic adaptive mesh refinement using OpenFOAM. *ArXiv e-prints* (May 2013).

[24] Mavriplis, D J, and Yang, Z. Construction of the discrete geometric conservation law for high-order time-accurate simulations on dynamic meshes. *Journal of Computational Physics 213* (2005), 557–573.

[25] OpenFOAM. http://www.openfoam.org/docs/user/mesh.php. *Official website*.

[26] Pan, H, Pan, L S, Xu, D, Ng, T Y, and Liu, G R. A projection method for solving incompressible viscous flows on domains with moving boundaries. *International Journal for Numerical Methods in Fluids 45* (2004), 53–78.

[27] Perot, B, and Nallapati, R. A moving unstructured staggered mesh method for the simulation of incompressible free surface flows. *Journal of Computational Physics 184* (2003), 192–214.

[28] Shyy, W, Pal, S, Udaykumar, H S, and Choi, D. Structured moving grid and geometric conservation laws for fluid flow computation. *Numerical Heat Transfer 34* (1998), 369–397.

[29] Thomas, G B, and Finney, R L. Leibnitz integral rule. *Calculus* (1995).

[30] Thomas, P D, and Lombard, C K. Geometric Conservation Law and its application to flow computation on moving grids. *AIAA Journal 17*, 10 (1979), 1030–1037.

[31] Tukovic, Z, and Jasak, H. A moving mesh finite volume interface tracking method for surface tension dominated interfacial flow. *Computers and Fluids 55* (2012), 70–84.

[32] Zhang, H, Reggio, M, Trepanier, J Y, and Camarero, R. Discrete form of the GCL for moving meshes and its implementation in CFD schemes. *Computers and Fluids 22*, 1 (1993), 9–23.

[33] Zhang, X, Schmidt, D, and Perot, B. Accuracy and Conservation properties of a Three-Dimensional Unstructured Staggered Mesh scheme for Fluid Dynamics. *Journal of Computational Physics 175* (2002), 764–791.