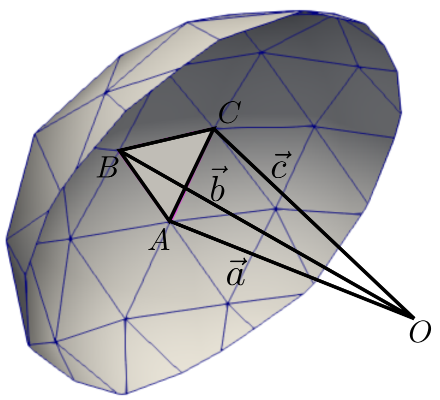

Figure 1: Portion of a closed shell surface constructed using a mesh of triangles is shown in this figure. The center of the closed shell is known to be at the origin of the coordinate system in which the coordinates of the vertices of the triangles are known. A tetrahedron has been constructed by joining the vertices of a triangle from the mesh with the origin. Care must be taken that the vertices of the triangle should be labeled in such a way (e.g. ABC, BCA, CAB) that they give a surface normal pointing outwards from the shell as per the right hand rule (ACB, BAC, CBA are examples of bad labeling).

1 Introduction

In our work with Oriented Particle Systems (OPS), we don't have any finite element mesh or shape functions. We generate a triangulation based on the particle positions. The triangular mesh has a spherical topology. We want to calculate the volume contained by this mesh and derivative of the volume as a function of particle positions. This information is required to impose a volume constraint in our model.

2 Volume Calculation

We make use of following assumptions.

1. The triangular mesh forms a closed surface.
2. Origin of the coordinate system in which vertex positions have been expressed lies inside the shell.
3. The connectivity defining the triangles in terms of the global vertex indices is such that surface normals of all the triangles are pointing outwards from the shell away from the origin.

Figure 1 shows the construction of a tetrahedron using position vectors of vertices of ΔABC. The vertices have been labeled such that they give an outward pointing surface normal as per the right hand rule. Total volume of the closed shell is equal to sum of volumes of all such tetrahedrons formed using triangles of the mesh. Let $\vec{a}, \vec{b}$ and $\vec{c}$ be the position vectors of the vertices A, B and C as shown in Figure 1. The volume v of tetrahedron $OABC$ is

$$V_{\Delta ABC} = \frac{1}{6}\left|\vec{a}\cdot\left(\vec{b}\times\vec{c}\right)\right|. \tag{1}$$

Note that if the vertices are labeled in a way that an outward normal is ensured, we don't need to take absolute value of the scalar triple product in equation 1. Getting rid of the absolute value is important in order to calculate the derivative. Thus, our equation for total volume V of the closed shell is

$$V = \sum_{\forall \Delta ABC} \frac{1}{6} \vec{a} \cdot \left(\vec{b} \times \vec{c} \right). \tag{2}$$

We sum over tetrahedrons formed by all such ΔABC possible.

3 Derivative of the Volume

We can write equation 1 in index notation as

$$V_{\Delta ABC} = \frac{1}{6} \varepsilon_{ijk} a_i b_j c_k, \tag{3}$$

where $i, j, k = 0, 1, 2$ and summation is implied over repeated indices. The derivative of $V_{\Delta ABC}$ is non-zero only with respect to the components of position vectors $\vec{a}, \vec{b}$ and $\vec{c}$.

$$\frac{\partial V_{\Delta ABC}}{\partial a_p} = \frac{1}{6} \varepsilon_{pjk} b_j c_k, \tag{4}$$

$$\frac{\partial V_{\Delta ABC}}{\partial b_p} = \frac{1}{6} \varepsilon_{ipk} a_i c_k, \tag{5}$$

$$\frac{\partial V_{\Delta ABC}}{\partial c_p} = \frac{1}{6} \varepsilon_{ijp} a_i b_j. \tag{6}$$

Substituting values of the indices, we get

$$\frac{\partial V_{\Delta ABC}}{\partial \vec{a}} = \begin{pmatrix} b_1 c_2 - b_2 c_1 \\ b_2 c_0 - b_0 c_2 \\ b_0 c_1 - b_1 c_0 \end{pmatrix}, \quad \frac{\partial V_{\Delta ABC}}{\partial \vec{b}} = \begin{pmatrix} a_2 c_1 - a_1 c_2 \\ a_0 c_2 - a_2 c_0 \\ a_1 c_0 - a_0 c_1 \end{pmatrix}, \quad \frac{\partial V_{\Delta ABC}}{\partial \vec{c}} = \begin{pmatrix} a_1 b_2 - a_2 b_1 \\ a_2 b_0 - a_0 b_2 \\ a_0 b_1 - a_1 b_0 \end{pmatrix}. \tag{7}$$

Each vertex belongs to more than one triangle. So, for each triangle we need to identify global vertex id numbers for the vertices. The vectors of equation 7 need to be assembled in a $3 \times N$ matrix, where N is the total number of vertices in the mesh based on the global id of the vertex.

4 Python Code Example

In the following code we implement two functions to calculate the volume of a closed shell and its derivatives with respect to the vertex positions. It heavily uses VTK library.

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Derivative of Volume of a Closed shell wrt position of the points
"""
```

```
import vtk as v
import numpy as np
import matplotlib.pyplot as plt

"""
The input of this class should be a polydata type with consistently labeled
triangles. It returns volume of the closed mesh formed by the triangles.
"""
def Vol(poly):
cells = poly.GetPolys()
cells.InitTraversal()
verts = v.vtkIdList()
V = 0.0
while( cells.GetNextCell( verts ) ):
coords = []
assert( verts.GetNumberOfIds() == 3 )
for i in range( 3 ):
coords.append( poly.GetPoint( verts.GetId(i) ) )
a,b,c = coords
V += (1/6)*np.dot( a, np.cross(b,c) )
return V

"""
The input to this class is a polydata object with triangles ordered
consistently. It returns a Matrix with N rows and 3 columms where N is the
number of points in the polydata
"""
def VolDiff(poly):
N = poly.GetNumberOfPoints()
poly.BuildLinks()
Out = np.zeros( (N,3) )
cells = poly.GetPolys()
cells.InitTraversal()
cellPts = v.vtkIdList()
while( cells.GetNextCell(cellPts) ):
ida = cellPts.GetId(0)
idb = cellPts.GetId(1)
idc = cellPts.GetId(2)
a = poly.GetPoint( ida )
b = poly.GetPoint( idb )
c = poly.GetPoint( idc )
dVa = np.array([b[1]*c[2]-b[2]*c[1],b[2]*c[0]-b[0]*c[2],b[0]*c[1]-b[1]*c[0]])
dVb = np.array([a[2]*c[1]-a[1]*c[2],a[0]*c[2]-a[2]*c[0],a[1]*c[0]-a[0]*c[1]])
dVc = np.array([a[1]*b[2]-a[2]*b[1],a[2]*b[0]-a[0]*b[2],a[0]*b[1]-a[1]*b[0]])
Out[ida,:] += (dVa*(1/6))
Out[idb,:] += (dVb*(1/6))
```

```
Out[idc,:] += (dVc*(1/6))
return Out

"""
Consistency test
"""
sp = v.vtkSphereSource()
sp.SetCenter(0.0,0.0,0.0)
sp.SetRadius(1.0)
sp.SetThetaResolution(5)
sp.SetPhiResolution(5)
sp.Update()
pd = sp.GetOutput()
writer = v.vtkPolyDataWriter()
writer.SetFileName('Sphere.vtk')
writer.SetInputData(pd)
writer.Write()

N = pd.GetNumberOfPoints()
hvec = np.logspace(-7,-2)
err = np.zeros(50)
dV = VolDiff(pd)
points = pd.GetPoints()
for z in range(50):
h = hvec[z]
dVh = np.zeros((N,3))
errh = np.zeros(N)
for i in range(N):
pt = np.array(points.GetPoint(i))
for j in range(3):
pt[j] += h
points.SetPoint(i, pt)
Vp = Vol(pd)
pt[j] += -2*h
points.SetPoint(i, pt)
Vm = Vol(pd)
dVh[i,j] = (Vp-Vm)/(2*h)
pt[j] += -2*h
points.SetPoint(i, pt)
errh[i] = np.linalg.norm(dV[i,:] - dVh[i,:])
err[z] = np.max( errh )

fig, ax = plt.subplots()
ax.loglog(hvec,err)
```