Imperial College London

IRP Presentation

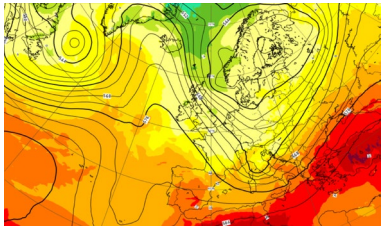
14/9/2022

Conservative Interpolation Between Unstructured Meshes that Preserves Heterogeneity and Structure for Modeling Three-Dimensional Bone

Nourhan Berjawi – nb621

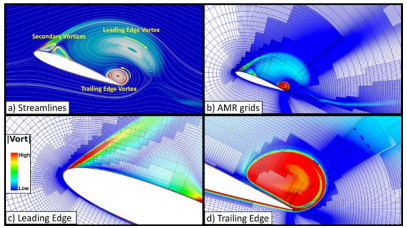
Meshing & Interpolation in Computational Science

When do we use meshing and mesh interpolation?



Weather Prediction

ECMWF's operational forecasts over Europe



Fluid Dynamics

NASA's Tools for Accurate Rotorcraft Analysis and Design

Limitations

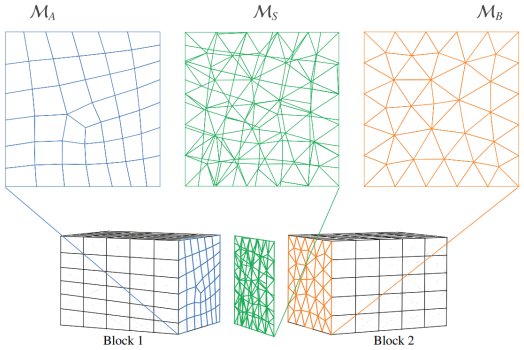
Accuracy

Memory

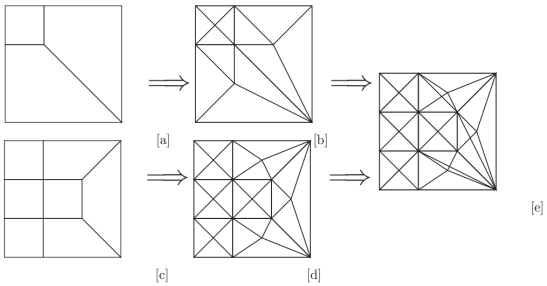
Time

Nourhan Berjawi – nb621

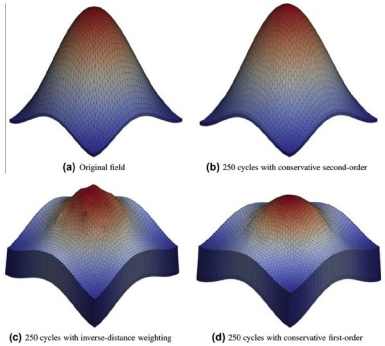
Previous Attempts at Creating/Using a 3D Supermesh



Flux-conserving treatment of non-conformal interfaces for finite-volume discretization of conservation laws (Rinaldi et al.)



A parallel adaptive viscoelastic flow solver with template based dynamic mesh refinement (Oner et al.)



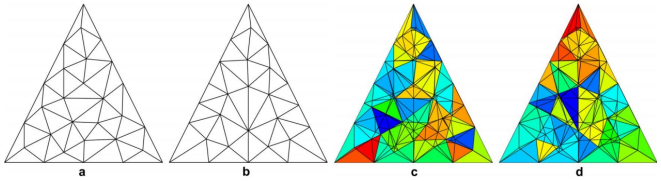
Conservative interpolation on unstructured polyhedral meshes: An extension of the supermesh approach to cell-centered finite-volume variables (Menon et al.)

Theoretical Background

A supermesh denoted $\mathcal{T}_C$ of parent meshes $\mathcal{T}_A$ and $\mathcal{T}_B$ is defined as any mesh that fulfills the following 2 conditions:

(1) Any node $\mathcal{N}$ present in the parent mesh must exist in the supermesh.

(2) The intersection of an element of $\mathcal{T}_C$ with any element of either of the 2 parent meshes must either be the empty set or the whole element $\mathcal{K}_C$.



A simple conservative interpolation operator

$$\sum_{K_A \in \mathcal{T}_A} \int_{K_A} q(\boldsymbol{x})\, dV = \sum_{K_B \in \mathcal{T}_B} \int_{K_B} \Pi_B[q](\boldsymbol{x})\, dV.$$

$$\sum_{K_A \in \mathcal{T}_A} \int_{K_A} q(\boldsymbol{x})\, dV = \sum_{K_B \in \mathcal{T}_B} \left(\sum_{K_C \in \chi_{BC}(K_B)} \int_{K_C} \Pi_C[q](\boldsymbol{x})\, dV \right),$$

$$\sum_{K_A \in \mathcal{T}_A} \int_{K_A} q(\boldsymbol{x})\, dV = \sum_{K_C \in \mathcal{T}_C} \int_{K_C} \Pi_C[q](\boldsymbol{x})\, dV.$$

$$\int_{K_C} \Pi_C[q](\boldsymbol{x})\, dV := \omega_{K_C} \int_{\chi_{CA}(K_C)} q(\boldsymbol{x})\, dV,$$

where

$$\omega_{K_C} := \int_{K_C} dV \Big/ \int_{\chi_{CA}(K_C)} dV.$$

$$q(\boldsymbol{x})|_{\boldsymbol{x} \in K_B} = \int_{K_B} q(\boldsymbol{x})\, dV / V(K_B),$$

conservation is retained

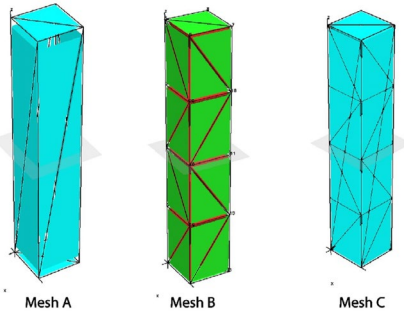
Steps to create 2D supermesh:

(1) Obtain the union $\mathcal{N}$ of all the nodes

(2) Obtain the union $\mathcal{D}$ of all the faces

(3) $\mathcal{N}$ and $\mathcal{D}$ form an imperfect piece-wise linear complex

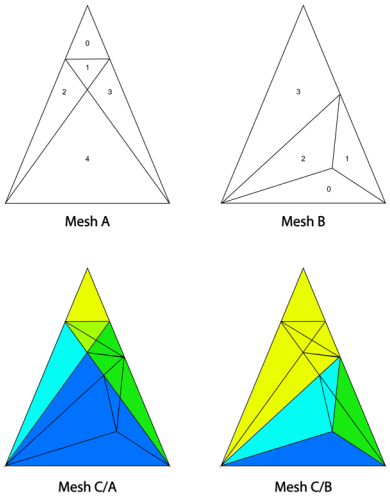
(4) $\mathcal{N}$ and $\mathcal{D}$ are given to the Delaunay Triangulation method and a triangulated supermesh is obtained



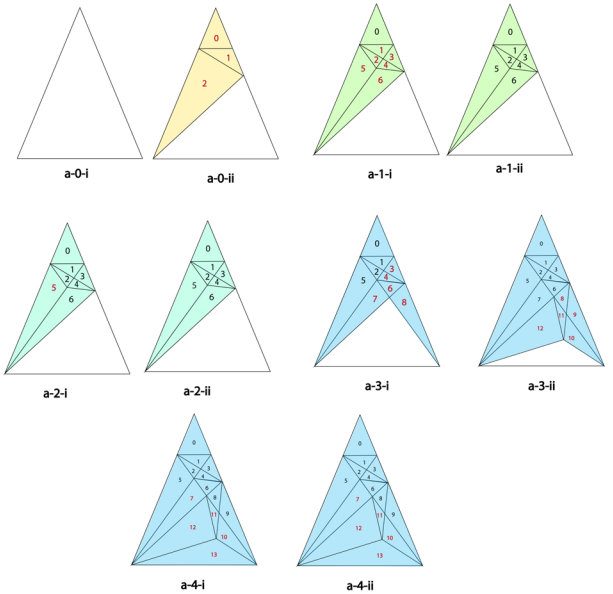
In this figure, Mesh A and Mesh B are two domains that were tetrahedralized by TetGen, with Mesh B being a more refined mesh. Mesh C shows the union of all the nodes of A and B as well as the union of their faces.

Method of Intersecting Tetrahedra

Demonstration



Parent meshes and mappings



Supermeshing process using the method of intersecting tetrahedra

Imperial College London

Method of Intersecting Tetrahedra

Pseudocode

Algorithm 1 Intersecting Tetrahedra Algorithm for Generating a Supermesh

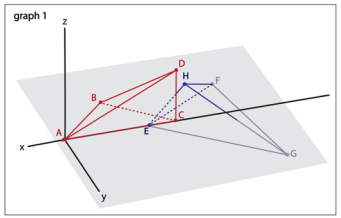
```
function supermesh(A, B):
    C = [ ] empty vector of tetrahedra
    marker = array of bools of size sizeof(B)
    for i in range(sizeof(A))
        AC , AB = [ ]  empty vectors of int
        temp = [ ] empty array of tetrahedra
        t , c = 0

        // start of BLOCK I
        if (AC ! empty)
            for j in range(sizeof(C))
                if A(i) intersects C(j)                *
                    temp.push(C(j))
        for j in range(sizeof(B))
            if (!marker[j])
                if A(i) intersects B(j)                *
                    AB.push(j)

        // start of BLOCK II
        while true
            if (temp[t] intersects C[AC[c]])           *
                X = get_intersection (temp(t),C[AC[c]] )
                if sizeof(X) == 1, go to else
                temp.remove(t)
                temp.push(X)
            else
                c++
                if ( c == sizeof ( AC ) )
                    c = 0;  t++
                    if ( t == sizeof ( temp ) )
                        break

        // start of BLOCK III
        temp1 = [ ] empty vector of tetrahedra


        for j in range ( sizeof ( temp ) )
            if (one of temp(j)'s parents is unintialized)
                    temp.remove(temp(j))
                    temp1.push(temp(j))
        t , c = 0
        if ( AB ! empty )
            while true
                if ( temp1(t) intersects B(AB[c]) )  *
                    X = intersection temp1(t) and B(AB[c])
                    if sizeof(X) == 1, go to else
                    temp1.remove(t)
                    temp.push(X)
                else
                    c ++
                    if ( c == sizeof(AB))
                        c = 0
                        t++
                        if ( t == sizeof (temp1))
                           break
        // start of BLOCK  IV
        for j in AB
            marker[j] = 1
        C.remove(all elements pointed at by AC and AB)
        C.push(temp, temp1)
```

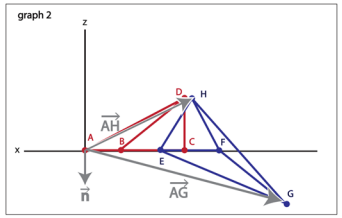
Pseudocode for the supermeshing algorithm which consists of 4 blocks

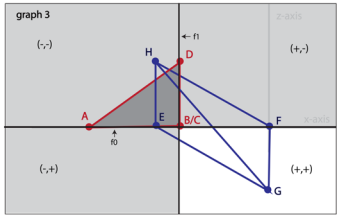
Method of Intersecting Tetrahedra

Key Methods

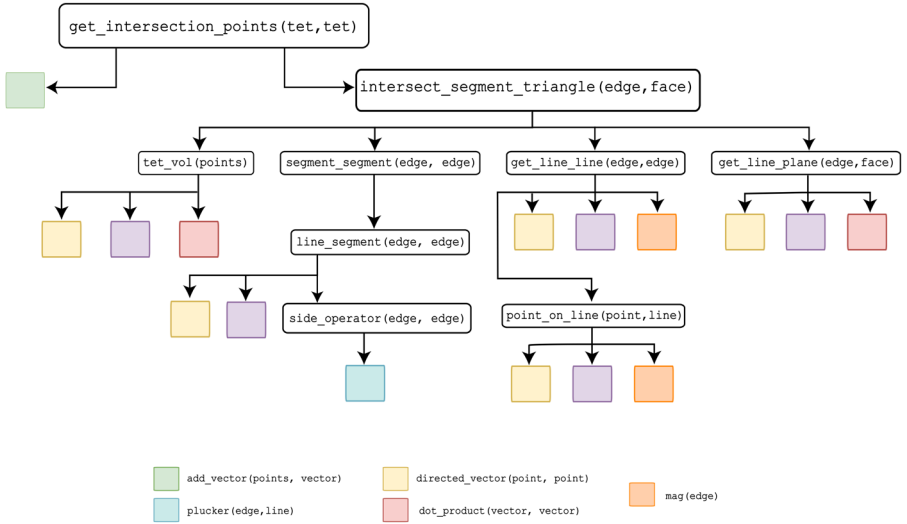
intersect(): check if two tetrahedra intersect



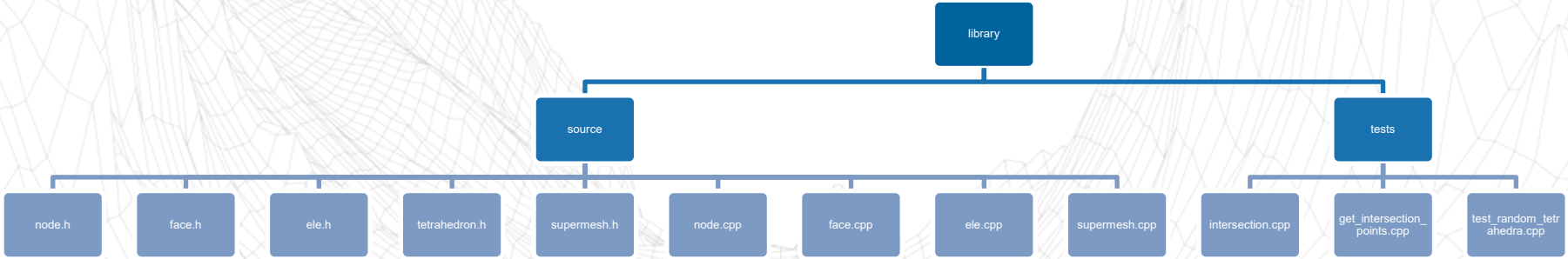




get_intersection_points(): get intersection points of 2 tetrahedra



Library Code Breakdown



Test Results

Steps:

1) Choose 3 intervals A, B and C such that A∩B != {} and C∩B != {} and A∩C = {}
2) Create 2000 random tetrahedra confined to each of regions A, B and C
3) Time intersect() method for tetrahedra in A vs. B, and B vs. C.
4) Time get_intersection_points() method for tetrahdera in A vs. B, and B vs. C.

```
Tests for intersect() function:
        Testing intersect() on a mix of tetrahedra:
        - - - - - - - - - - - - - - - - - - - -
         Checking all intersections takes 79144.5 ms, with the ratio of overlap/no overlap being: 0.152417

        now timing the intersect() function for another mix of tetrahedra:
        - - - - - - - - - - - - - - - - - - - -
         Checking all intersections takes 96018.1 ms, with the ratio of overlap/no overlap being: 0.288302
        On average, it took 0.0187265 ms to detect an overlap and 0.0164186 ms to detect no overlap.


Tests for get_intersection_points() function:
        Testing get_intersection_points() on mix of tetrahedra:
        - - - - - - - - - - - - - - - - - - - -
         Checking all intersections takes 95721.9 ms, with the ratio of overlap/no overlap being: 0.152417
        Testing get_intersection_points() on another mix of tetrahedra:
        - - - - - - - - - - - - - - - - - - - -
         Checking all intersections takes 116126 ms, with the ratio of overlap/no overlap being: 0.288302
        On average, it took 0.0226444 ms to detect an overlap and 0.0198584 ms to detect no overlap.
```

Future Work & Acknowledgements

Complete construct_intersection()

Optimize code

I would like to thank Dr. Thomas for his relentless efforts in helping me navigate the IRP during the past three months.
I also wish to thank Dr. Palszuny and all ACSE staff for all the guidance they have offered throughout the year.