Introduction to the Interface Reconstruction Library (IRL)

Overview of IRL

IRL was developed to simplify the calculation of volume moments for intersections of polytopes that can be represented by planes.

Distinct classes are used for each type of polytope, which embeds connectivity information for the vertices, implicitly defining edges and faces.

The planes that define the intersection regions are combined into a PlanarReconstruction, which essentially is a list of Plane objects that define the extent of the intersection region. Specializations of PlanarReconstruction exist to provide specialized behaviors and extend its capabilities.

Overview of IRL

The main application for IRL is enabling easier implementation of geometric Volume of Fluid (VOF) methods, although it should be applicable to a much wider class of problems involving geometric operations on polytopes.

Numerous interface reconstruction algorithms typically used for geometric VOF methods are included in IRL as well, with a straight forward interface that generally involves forming a neighborhood that constitutes a numerical stencil, and then providing this to the reconstruction method to obtain the reconstructed interface planes.

Necessitated by the interface reconstruction algorithms, general and easy to use optimization algorithms also exist in IRL for use by others.

getVolumeMoments<ReturnType, CuttingMethod> (a_polytope_type, a_reconstruction_type)

Volume moments for intersections of polytopes and regions represented by a collection of planes are handled by the getVolumeMoments function template given above.

The arguments are as follows:

- ReturnType : Type of volume moments to be returned from the intersection.
- CuttingMethod : Algorithm used to calculate the volume moments. Each will deliver the same answer, however have different performance characteristics.
- a_polytope_type : Volume that will be intersected with the PlanarReconstruction, a_reconstruction_type. Currently, polyhedra and polygons are supported.
- a_reconstruction_type : Type of PlanarReconstruction to use.

The getVolumeMoments function has a variety of different behaviors that are determined by the specific types of the arguments provided to it. In the following slides, each of the possible types will be explained, along with their role in getVolumeMoments.

ReturnType – Volume

Implemented in : src/moments/volume.h

Volume is the volume of the intersection between the polytope's volume and volume internal to the PlanarReconstruction. The Volume class is used for all zeroeth order volume moments, i.e. it is also used to represent the area in 2D or line length in 1D.

ReturnType – VolumeMoments

Implemented in : src/moments/volume_moments.h

VolumeMoments is the Volume and centroid of the intersection between the polytope's volume and volume internal to the PlanarReconstruction. As is true with most ReturnType's, it can be normalized and multiplied by the Volume using the methods normalizeByVolume() and multiplyByVolume(), respectively.

ReturnType – VolumeMomentsAndNormal

Implemented in : src/moments/volume_moments_and_normal.h

VolumeMomentsAndNormal is a paired storage of a VolumeMoments object and a (potentially un-normalized) normal vector.

NOTE: Only valid for Classes that can return its normal vector via a method Normal calculateNormal(void).

ReturnType – SeparatedMoments<MomentsType>

Implemented in : separated_volume_moments.h

SeparatedVolumeMoments<MomentsType> is a grouping of two MomentsType objects, where currently MomentsType can be Volume or VolumeMoments. This stores a MomentsType object for both the volume internal to the reconstruction as well as external to the reconstruction (but still lying inside the polytope).

SeparatedMoments<MomentsType> takes advantage of the ability to calculate one of the MomentsType objects from the other using the volume moments of the encompassing polytope.

ReturnType – ListedVolumeMoments<MomentsType>

Implemented in : src/moments/listed_volume_moments.h

ListedVolumeMoments<MomentsType> maintains a vector of MomentsType objects, which can be appended to using the += operator. This can be used with getVolumeMoments to return a list of individual simplex moments that are encountered during the cutting process.

Note: If HalfEdgeCutting cutting method is used, one moment will be appended to the list per half-edge polytope, not per simplex.

ReturnType – TaggedAccumulatedVolumeMoments<MomentsType>

Implemented in : src/moments/tagged_accumulated_volume_moments.h

TaggedAccumulatedVolumeMoments<MomentsType> is a vector of MomentsType objects that can be accessed via an ID for each moment. When using getVolumeMoments, this allows the storage of MomentsType objects that are tagged with a unique ID provided by the PlanarReconstruction. This can be used to traverse a network on PlanarReconstructions and store volume moments for individual PlanarReconstructions in the network.

CuttingMethod - RecursiveSimplexCutting

RecursiveSimplexCutting works through always representing volumes as a group of simplices. Therefore the act of separating a polytope by a plane only involves separating each simplex in its decomposition by the plane and then representing the two new sub-volumes by simplices.

In RecursiveSimplexCutting, this is implemented through recursion, where multi-plane reconstructions successively cut newly generated simplices until all planes have been intersected with the simplex group, at which point the ReturnType is calculated for the simplex.

This method is highly efficient when few planes exist in the reconstruction. As more planes are added, however, the cost increases with new simplices formed per plane intersection.

CuttingMethod - HalfEdgeCutting

HalfEdgeCutting represents the initial polytope using the half-edge data structure, where half-edges, vertices, and faces of the polytope are tracked. This avoids the initial simplicial decomposition of the polytope, preventing edges being formed inside the polytope's volume. Additionally, it is guaranteed that any half-edge structure can be intersected by a plane to generate a new, topologically valid, half-edge structure.

HalfEdgeCutting is implemented with all half-edges, vertices, and faces served from a static storage location used by all calls to getVolumeMoments. Each time a half-edge structure is intersected with a plane, the new half-edge structures for below and above the plane are generated. Once all planes have been intersected with the half-edge structure, simplices are formed through triangulation of the faces and a reference point. The volume moments are then calculated as the sum of the simplices volume moments.

CuttingMethod - SimplexCutting

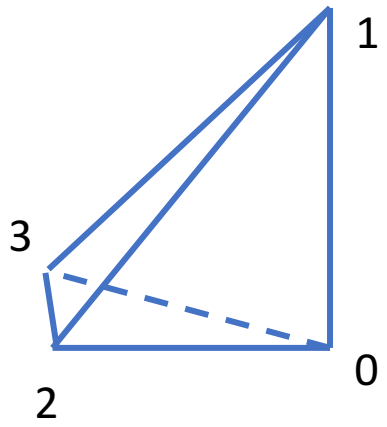
SimplexCutting is closely related to RecursiveSimplexCutting, however instead of recursion, the simplices are stored in SegmentedDecomposedPolytope objects. This provides some of the efficiency of the simplex operations, while also providing access to the whole object.

In SimplexCutting, new vertices are generated during an intersection and stored in a static vector. These new vertices are then used to form ProxyTet or ProxyTri objects, and stored in SegmentedDecomposedPolytope objects for the volumes underneath and above the plane.

a_poltyope_type – Tet

Assumptions :None.

Special Notes : The volume of this Tet is signed, just as the volumes of all IRL polytopes are according to the Right Hand Rule forming outward facing normal vectors.



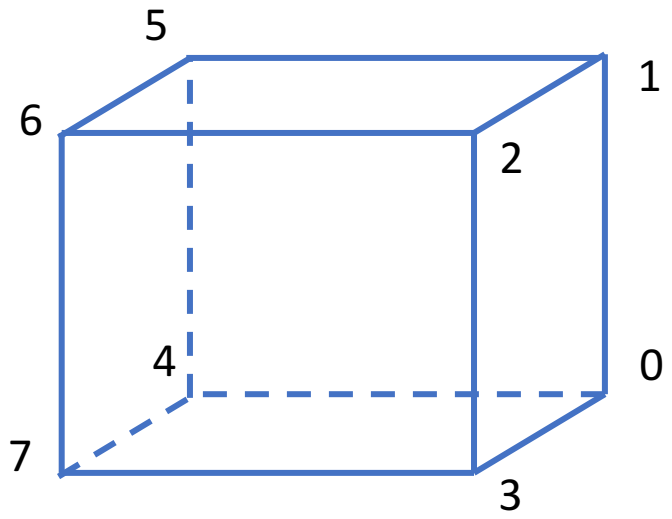
Number of Vertices : 4
Number of Faces : 4
Faces :

0 1 2
2 1 3
2 3 0
0 3 1

a_poltyope_type – RectangularCuboid

Assumptions : Forms a valid rectangular prism, with a corresponding positive volume equal to dx*dy*dz

Special Notes : Uses as 5-tetrahedron decomposition



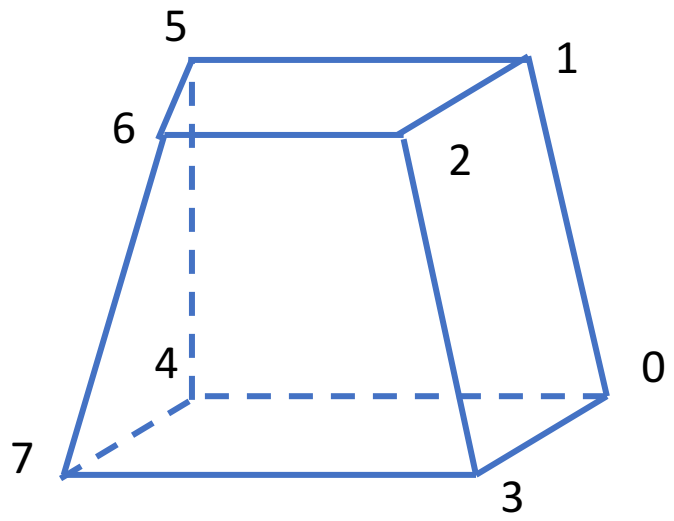
Number of Vertices : 8
Number of Faces : 6
Faces :

0 1 2 3
4 7 6 5
0 3 7 4
1 5 6 2
2 6 7 3
0 4 5 1

a_poltyope_type – Hexahedron

Assumptions : Forms a convex hexahedron with a positive volume

Special Notes : Uses as 5-tetrahedron decomposition

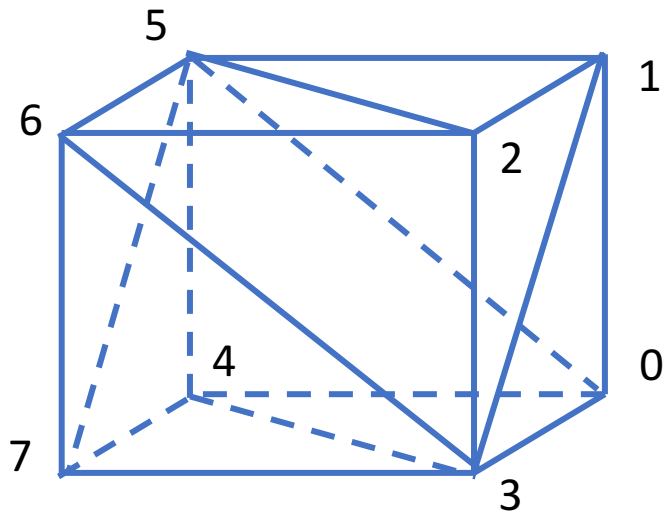


Number of Vertices : 8
Number of Faces : 6
Faces :

0 1 2 3
4 7 6 5
0 3 7 4
1 5 6 2
2 6 7 3
0 4 5 1

a_poltyope_type – Dodecahedron

Assumptions : None, beyond being a valid dodecahedron.



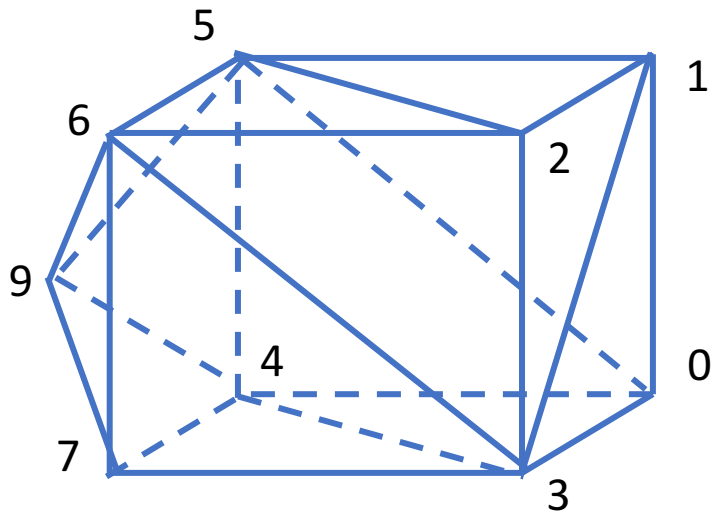
Number of Vertices : 9
Number of Faces : 14
Faces :

3 0 1	2 5 6
3 1 2	1 5 2
5 4 7	3 6 7
5 7 6	3 2 6
4 3 7	1 0 5
4 0 3	0 4 5

a_poltyope_type – CappedDodecahedron

Assumptions : None.

Special Notes : The left face in the diagram is triangulated with an extra point, forming a “cap” made of 4 triangles. This point can then be adjusted to modify the volume of the CappedDodecahedron with the adjustCapToMatchVolume method.



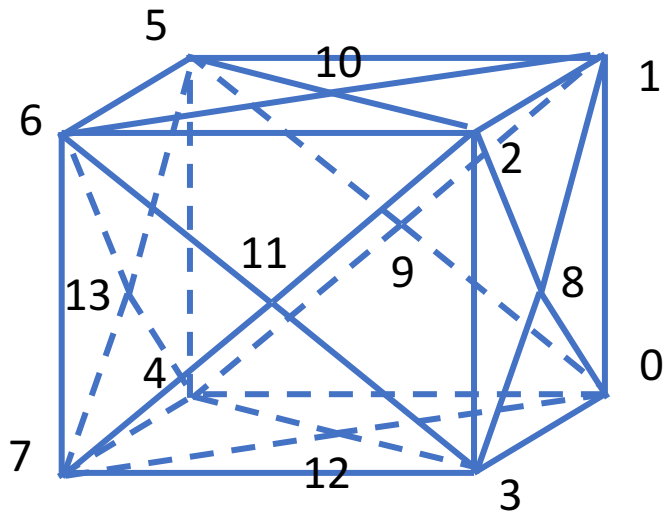
Number of Vertices : 9
Number of Faces : 14
Faces :

3 0 1
3 1 2
5 4 9
4 7 9
6 9 7
5 9 6
4 3 7
4 0 3
2 5 6
1 5 2
3 6 7
3 2 6
1 0 5
0 4 5

a_poltyope_type – Polyhedron24

Assumptions : None.

Special Notes : The left faces in the diagram, triangulated around point 13, form a “cap” made of 4 triangles. This point can be adjusted to modify the volume of the Polyhedorn24 with the adjustCapToMatchVolume method.



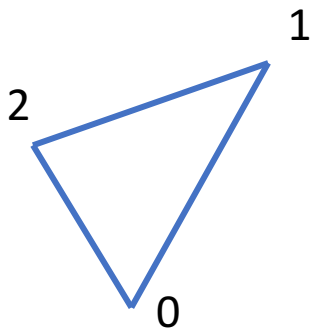
Number of Vertices : 14
Number of Faces : 24
Faces :

0 1 8	1 5 10	3 7 12
1 2 8	5 6 10	7 4 12
2 3 8	6 2 10	4 0 12
3 0 8	2 1 10	0 3 12
5 1 9	2 6 11	5 4 13
4 5 9	6 7 11	4 7 13
0 5 9	7 3 11	7 6 13
1 0 9	3 2 11	6 5 13

a_poltyope_type – Tri

Assumptions : Exists on plane set by method setPlaneOfExistence.

Special Note : Sign of the volume will be dependent on point ordering. If normal vector following Right Hand Rule is same as PlaneOfExistence normal, sign if positive, otherwise its negative.

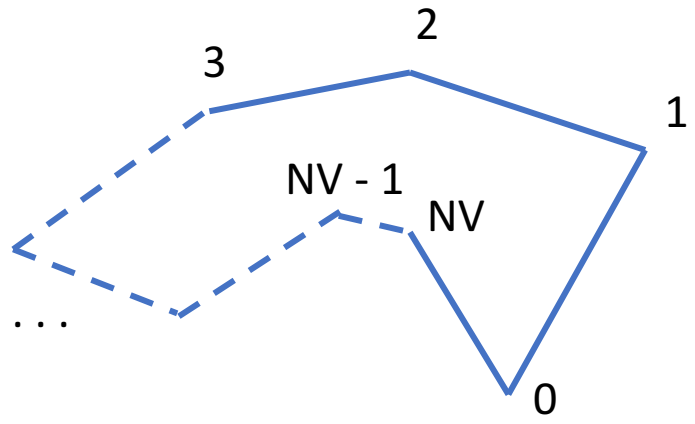


Number of Vertices : 3
Number of Triangles : 1
Triangles :
0 1 2

a_poltyope_type – Polygon

Assumptions : Exists on plane set by method setPlaneOfExistence.

Special Note : Sign of the volume will be dependent on point ordering. If normal vector following Right Hand Rule is same as PlaneOfExistence normal, sign if positive, otherwise its negative.



Number of Vertices (NV) : Variable
Number of Triangles : NV - 2
Triangles :

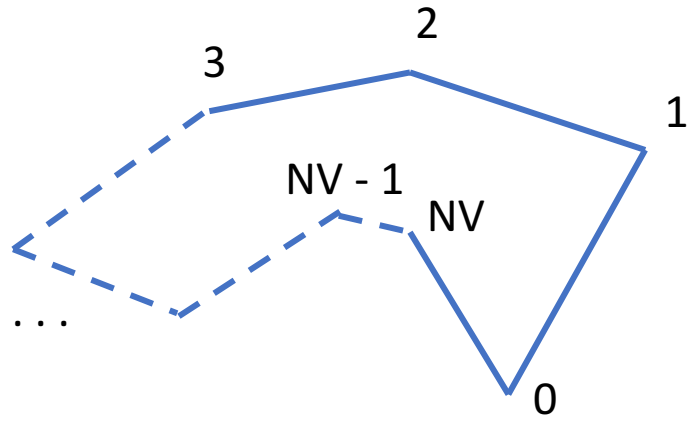
Tri = {0, tri_num+1, tri_num+2}

a_poltyope_type – DividedPolygon

Assumptions : Exists on plane set by method setPlaneOfExistence.

Special Note :

1. Sign of the volume will be dependent on point ordering. If normal vector following Right Hand Rule is same as PlaneOfExistence normal, sign if positive, otherwise its negative.
2. Same as Polygon class, but Tri decomposition is with respect to polygon centroid. (C_P) If polygon is modified, must call method resetCentroid



Number of Vertices (NV) : Variable
Number of Triangles : NV
Triangles :
For tri_num < NV
Tri = {C_P, tri_num, tri_num+1}
For tri_num == NV
Tri = {C_P, tri_num, 0}

a_reconstruction_type – PlanarLocalizer

Collection of Plane objects that localize a region of space to that internal to the reconstruction, where internal to the reconstruction means the intersection of the volumes under each plane. A PlanarLocalizer can be used to localize a volume into any convex polyhedron.

Assumptions : The collection of planes represent a convex volume.

Special Note : A PlanarLocalizer object can hold any number of planes, however number greater than global_constants::MAX_PLANAR_LOCALIZER_PLANES will cause a heap allocation.

a_reconstruction_type – PlanarSeparator

Collection of Plane objects that localize a region of space. PlanarSeparator contains methods to reset the distance to each plane to match a given internal volume for a give polytope.

Assumptions : Either the internal or external volume of the PlanarSeparators form a convex volume.

Special Note : A PlanarSeparator object can hold any number of planes, however a number greater than global_constants::MAX_PLANAR_SEPARATOR_PLANES will cause a heap allocation.

a_reconstruction_type – LocalizedSeparator

Pairing of a PlanarLocalizer and PlanarSeparator object (using pointers). When cutting by LocalizedSeparator, the volume is first localized by the PlanarLocalizer and then the separator is applied. This allows the PlanarSeparator to be used for only specific regions of a larger polytope volume.

Special Note : Since the pairing of the two objects is done through pointers, one must be careful not to delete the PlanarLocalizer or PlanarSeparator that is pointed to while the LocalizedSeparator is still in use.

a_reconstruction_type – LocalizedSeparatorLink

A LocalizedSeparator object treated as a node in an undirected graph. During cutting, the volume will be separated by each plane in the PlanarLocalizer of the LocalizedSeparator. The portion above the plane will be given to the neighboring LocalizedSeparatorLink in the undirected graph, and the portion beneath the plane will be cut by the next plane in the PlanarLocalizer. Once all planes in the PlanarLocalizer have been used, the PlanarSeparator will then be applied (just as for the LocalizedSeparator). Each LocalizedSeparatorLink in the graph can be given an ID number, which is then used to determine its storage for the ReturnType TaggedAccumulatedVolumeMoments<MomentsType>

Special Note : Since the pairing of the two objects, and the graph connectivity, are done through pointers, one must be careful not to delete the PlanarLocalizer, PlanarSeparator, or neighboring LocalizedSeparatorLink objects that is pointed to while the LocalizedSeparator is still in use.

a_reconstruction_type – LocalizerLink

The LocalizerLink object is the same as the LocalizedSeparatorLink object, however without the PlanarSeparator. This will return moments for the localized regions directly, with no additional separation after the fact.

Special Note : Since the pairing of the two objects, and the graph connectivity, are done through pointers, one must be careful not to delete the PlanarLocalizer or neighboring LocalizerLink objects that is pointed to while the LocalizerLink is still in use.