

---

# Primal/Dual Mesh with Application to Triangular/Simplex Mesh and Delaunay/Voronoi

*Release 1.0*

Humayun Irshad<sup>1</sup>, Stéphane Rigaud<sup>1</sup> and Alexandre Gouaillard<sup>2</sup>

November 16, 2012

<sup>1</sup>Image & Pervasive Access Lab, National Centre for Scientific Research (CNRS), Fusionopolis, Singapore

<sup>2</sup>Singapore Immunology Network, Agency for Science, Technology and Research (A\*STAR), Biopolis, Singapore

## Abstract

This document describes an extension of ITK to handle both primal and dual meshes simultaneously. This paper describe in particular the data structure, an extension of *itk::QuadEdgeMesh*, a filter to compute and add to the the structure the dual of an existing mesh, and an adaptor which let a downward pipeline process the dual mesh as if it was a native *itk::QuadEdgeMesh*. The new data structure, *itk::QuadEdgeMeshWithDual*, is an extension of the already existing *itk::QuadEdgeMesh* [2], which already included by default the due topology, to handle dual geometry as well. Two types of primal meshes have been specifically illustrated: triangular / simplex meshes and Voronoi / Delaunay. A functor mechanism has been implemented to allow for different kind of computation of the dual geometry. This paper is accompanied with the source code and examples.

Latest version available at the [Insight Journal](http://hdl.handle.net/10380/3392) [ <http://hdl.handle.net/10380/3392> ]  
Distributed under [Creative Commons Attribution License](#)

## Contents

<b>1 Surfaces</b>	<b>2</b>
1.1 Orientable 2-Manifold Mesh: A discrete real-world object	2
1.2 Special Case A: Triangular Meshes	3
1.3 Special Case B: Simplex Meshes	3
<b>2 Duality</b>	<b>4</b>
2.1 Notion of Duality	4
2.2 Triangulation - Simplex Duality	5
2.3 Delaunay - Voronoi Duality	6

---

<b>3</b>	<b>Implement Duality in ITK</b>	<b>6</b>
3.1	Existing Data Structure for Meshes in ITK . . . . .	6
3.2	Extension in data structure, <i>QuadEdgeMeshWithDual</i> data structure . . . . .	7
3.3	Primal to primal/dual filter . . . . .	8
	Dual point functor . . . . .	9
	Dual borders calculation . . . . .	10
<b>4</b>	<b>Validation</b>	<b>10</b>
4.1	Test on planer triangular to simplex mesh with and without holes . . . . .	10
4.2	Test with Delaunay to Voronoi . . . . .	12
4.3	Test on non planar mesh . . . . .	12
<b>5</b>	<b>Usage</b>	<b>13</b>

---

## 1 Surfaces

### 1.1 Orientable 2-Manifold Mesh: A discrete real-world object

The surfaces of real world objects are oriented 2-manifolds. Those are usually represented in computer using meshes which are the sampled, discrete version of the underlying, supposedly continuous surface. The definition of surface mesh is of combinatorial nature [3], that improves reasoning about data structure like the same facet cannot appear on both sides of an edge. The surface mesh is a union of  $C = V \cup E \cup F$  of three disjoint sets together with an incidence relation where V the vertices, E the edges and F the facets of the mesh. The incident relation on C must be symmetric. No two elements from the same set V, E, F are incident. There are four additional conditions: (1) every edge is incident to two vertices, (2) every edge is incident to two facets, (3) for every incident pair of vertices or facets, there are exactly two edges incident to both and (4) every vertex and every facets is incident to at least one other element. The neighbourhood of a vertex are edges and facets which are incident to that vertex. Thus, the neighbourhood decomposes into disjoint cycles, where each cycle is an alternating sequence of edges and facets.

A surface mesh is 2-manifold if (1) each edge is incident to only one or two facets and (2) the facets incident to a vertex form a close or an open fan i.e. for each point on a 2-manifold there exists a neighbourhood that is homeomorphic to the open disc. If every vertex has a closed fan, the given 2-manifold has no boundary. If a vertex has a open fan, then edges that are incident to one facet; they are called border edges and they form the boundary of the 2-manifold mesh. A non-manifold example would be two tetrahedra glued together at a single vertex or common edge as shown in figure 1. A mesh is a 2-manifold if and only if the neighbourhood of each vertex decomposes into a single cycle. The next distinction is between orientable and non-orientable mesh. A surface mesh is oriented if each cycle around a facet is oriented and if, for each edge, the two cycles of its two incident facets are oriented in opposite direction. A 2-manifold mesh is orientable if there exists such an orientation. This new data structure only consider orientable 2-manifolds mesh representation with and without boundary.

Genus is a topologically invariant property of a surface defined as the largest number of non-intersecting closed curves that can be drawn on the surface without separating it. Also, it is a complete invariant in the sense that, if two orientable closed surfaces have the same genus, then they must be topologically equivalent.

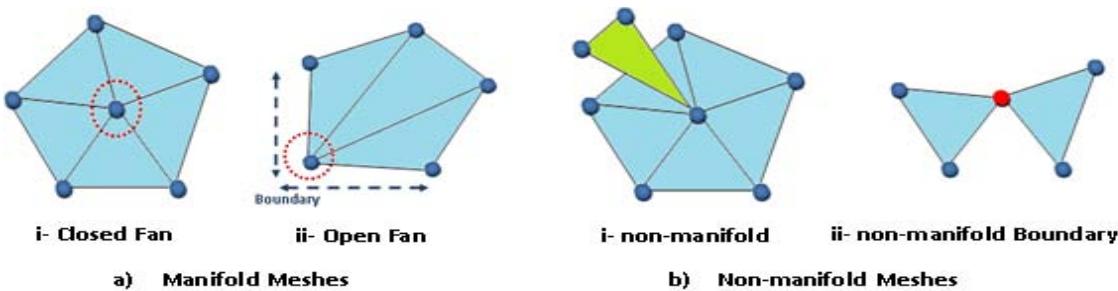


Figure 1: Examples of Meshes

The genus of a surface is related to the Euler characteristic  $\chi$ . For an orientable surface such as a sphere (genus 0) or torus (genus 1), the relationship is

$$\chi = 2 - 2g - b \quad (1)$$

With  $g$  being the genus, and  $b$  being the number of borders (for non-closed surfaces).

Given an arbitrary polygonal mesh  $\tau$  of a regular region  $R \subset S$  of a surface  $S$ , we shall denote by  $F$  the number of polygonal faces, by  $E$  the number of sides(edges), and by  $V$  the number of vertices of the triangulation. Another way to compute the Euler characteristic is then

$$F - E + V = \chi \quad (2)$$

Special cases of discrete 2-manifolds of interest to us are triangular meshes and simplex meshes illustrated in figure 2.

## 1.2 Special Case A: Triangular Meshes

A common representation of discrete surfaces are triangulation  $\tau$  for which the surface  $\mathbb{R} \subset S$  is composed of a set of adjacent triangles  $T_i, i = 1, \dots, n$ , such that

- $\bigcup_{i=1}^n T_i = \mathbb{R}$ .
- If  $T_i \cap T_j \neq \emptyset$ , then  $T_i \cap T_j$  is either a common edge of  $T_i$  and  $T_j$  or a common vertex of  $T_i$  and  $T_j$ .

Each triangles of a triangulation shares at least one of its edge with a neighbouring triangle. Triangles being the simplest polygon that can represent a surface, it has been used intensively in Computer graphics and is still ubiquitous today in surface representations and corresponding data formats.

## 1.3 Special Case B: Simplex Meshes

Simplex meshes are used for discrete surface representation. Simplex meshes have two main properties, (1) each vertex is adjacent to a fixed number of neighbouring vertices: 2 for a contour (1-simplex mesh), 3 for a surface (2-simplex mesh) and 4 for tetrahedron (3-simplex mesh, not treated here); and (2) the topology of a 2-simplex mesh is dual of a triangulation. A  $k$ -simplex can be referred a  $(k+1)$ -connected mesh. For instance, a segment of non-zero length is a 1-simplex, a triangle (polygon) of non-zero area is a 2-simplex

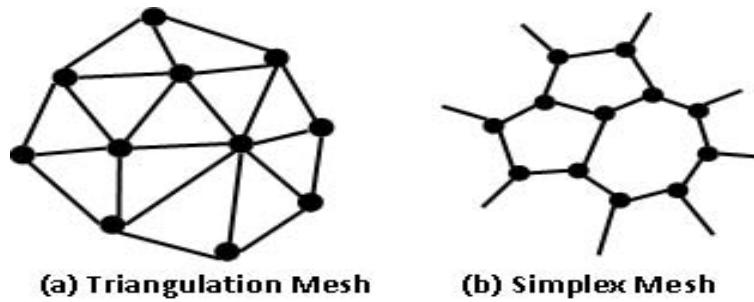


Figure 2: Triangulation and Simplex Mesh

and a tetrahedron of non-zero volume is a 3-simplex mesh. Formally, a  $k$ -Simplex Mesh ( $kSM$ ) of  $\mathbb{R}^3$  is defined as a pair  $(V(M), N(M))$  [1] where:

$$V(M) = \{P_i\}, \{i = 1, \dots, n\}, P_i \in \mathbb{R}^3 \quad (3)$$

$$\begin{aligned} N(M) : \{1, \dots, n\} &\longrightarrow \{1, \dots, n\}^{k+1} \\ i &\longmapsto (N_1(i), N_2(i), \dots, N_{k+1}(i)) \end{aligned} \quad (4)$$

$$\begin{aligned} \forall i \in \{1, \dots, n\}, \forall j \in \{1, \dots, k+1\}, \forall l \in \{1, \dots, k+1\}, l \neq j \\ N_j(i) \neq i \end{aligned} \quad (5)$$

$$N_l(i) \neq N_j(i) \quad (6)$$

$V(M)$  is the set of vertices of  $M$  and  $N(M)$  is the associated connectivity function. Equations (5) and (6) present a mesh from exhibiting loops. It is important to make a distinction between the topological nature of a mesh represented by its connectivity function  $N(M)$  and its geometric nature corresponding to the position of its vertices  $V(M)$ .

The structure of a simplex mesh is the one of a simply connected graph and does not in itself constitute a new surface representation. The simplex mesh representation has several desirable properties that makes them well suited for the recovery of geometric models from range data. The characteristics of simplex mesh for discrete surfaces includes generality (represents all types of orientable surfaces regardless of their genus and end numbers), simplicity (minimum number of vertices to represents a surface or shape) and adaptability.

## 2 Duality

### 2.1 Notion of Duality

We define  $A$  and  $B$  to be dual surface meshes i.e.,  $B$  is dual of  $A$  and vice versa, if the following conditions are satisfied.

- The number of vertices of  $A$  is the same as the number of face of  $B$ , so that they can be put into one-to-one correspondence.
- The number of vertices of  $B$  is the same as the number of face of  $A$ , so that they can also be put into one-to-one correspondence.

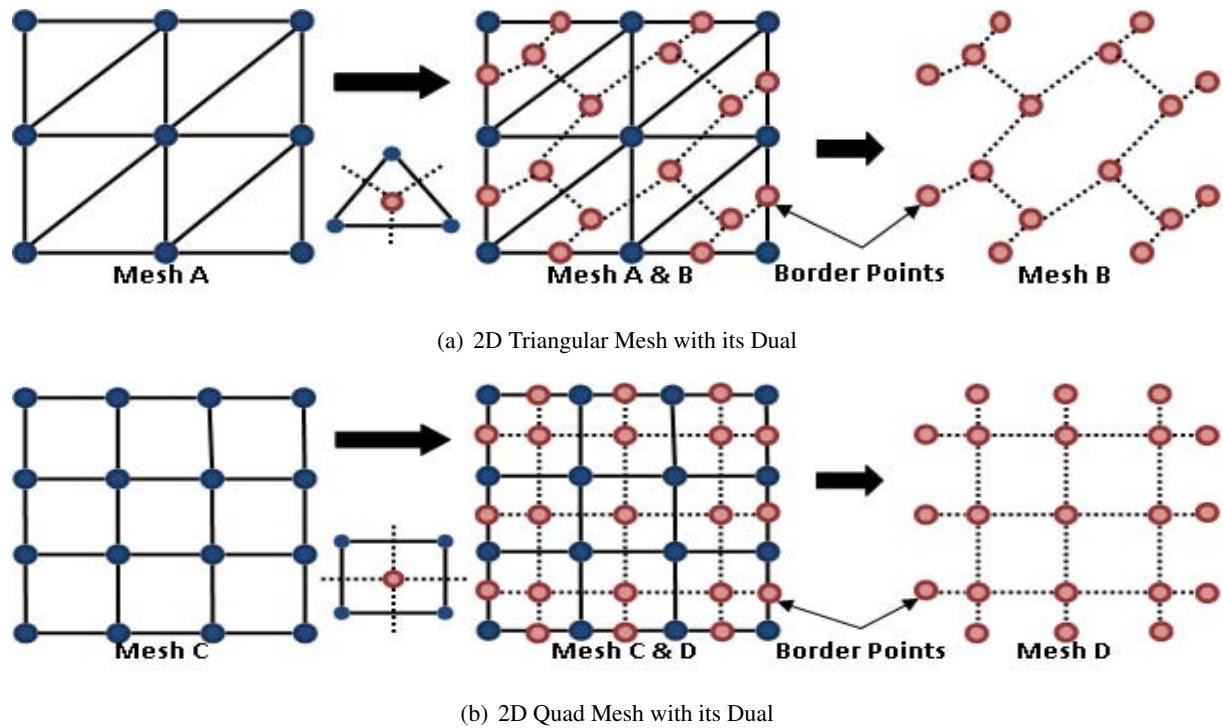


Figure 3: Examples of Dual Meshes. (We also sampled the border points.)

- Each pair of vertices of  $A$  that map to adjacent faces in  $B$  is joined by an edge which can be put into correspondence with the common edge of the associated pair of faces of  $B$ . The edges that join adjacent vertices of  $B$  can be put into the same correspondence with the common edges of the associated pairs of elements of  $A$ .

Figure 3 illustrates the duality of meshes. Each boundary edge of a face in mesh  $A$  is put into correspondence with a half-open edge in mesh  $B$  which starts at the corner corresponding to that face in  $A$  as shown in figure 3.

## 2.2 Triangulation - Simplex Duality

One of the most interesting way of considering simplex meshes is through duality of triangulations. The structure of a  $k$ -simplex mesh is indeed closely related to the structure of a  $k$ -triangulation. A  $k$ -triangulation of  $\mathbb{R}^d$  is composed of  $p$ -simplices ( $1 \leq p \leq k$ ) which are the  $p$ -faces of the triangulation. We define a  $p$ -face of a  $k$ -simplex mesh as being the dual of a  $(k-p)$  simplices of a  $k$ -triangulation. For instance, a 1-face of a 2-simplex mesh is an edge and a 2-face of a 2-simplex mesh is polygon. In general, a  $p$ -face of a  $k$ -simplex mesh is a  $(p-1)$ -simplex mesh and is, therefore, made of  $q$ -faces ( $q < p$ ). A Simplex mesh is said to be regular if all  $p$ -faces have the same number of vertices.

Simplex meshes are dual of triangulations. Thus, their connectivity functions  $N(M)$  are mapped by an homeomorphism. Simplex meshes are topologically equivalent to triangulations but not geometrically equivalent. We can define a topological transformation that associates a  $k$ -simplex mesh to a  $k$ -triangulation. This transformation is pictured in figure 4 and considers differently the vertices and edges located at the boundary of the triangulation from those located inside.

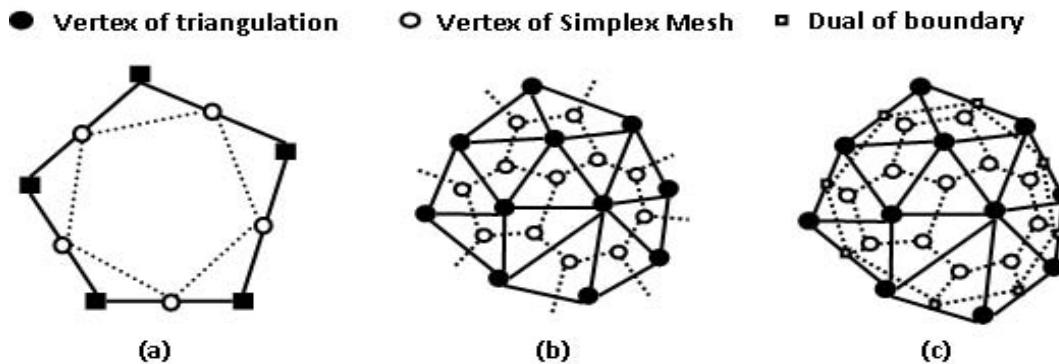


Figure 4: a) A 1-Simplex mesh and its dual; b) A 2-Simplex mesh and its dual triangulation; c) same as (b). The dual of the triangulation boundary is considered to extract the simplex mesh.

## 2.3 Delaunay - Voronoi Duality

Taking a set of point  $\mathcal{P}$  in  $\mathbb{R}^3$ , the Delaunay triangulation of  $\mathcal{P}$  is a specific triangulation of  $\mathcal{P}$  that respects the Delaunay criterion stating that no point of  $\mathcal{P}$  should be inside of the circumference circle of any triangle of the triangulation of  $\mathcal{P}$ . Taking a set of point  $\mathcal{P}$  in  $\mathbb{R}^3$ , the Voronoi diagram (or tesselation) is the partition of  $\mathbb{R}^3$  into  $n$  polyhedral regions such as each region  $T$  has a set of points in  $\mathbb{R}^3$  which are closer to  $T$  than to any other region.

The Voronoi diagram is the dual of the Delaunay triangulation, and the Delaunay triangulation is the dual structure of the Voronoi diagram. By dual, we mean to draw a line segment between two Voronoi vertices if their Voronoi polygons have a common edge, or in more mathematical terminology: there is a natural bijection between the two which reverses the face inclusions. The duality between Delaunay triangulations and Voronoi diagram is geometric because it depends on the position of its vertices.

## 3 Implement Duality in ITK

### 3.1 Existing Data Structure for Meshes in ITK

The *QuadEdgeMesh* data structure in itk, as depicted in figure 5, can handle discrete 2-manifold surfaces. It actually store the geometry and both primal and dual topology. It has a constant complexity local accesses an modifications. The *QuadEdgeMesh* data structure is a 3 layers structure in which the bottom layer is called *QuadEdge* (*QE*) layer that represents the topology, the intermediate layer is called *QE Geometric* (*QEGeom*) layer that linking topology and geometry and finally the upper layer is native to ITK called ITK layer. The *QE* data structure is presented in detail in [2]. For each edge, there are 4 *QEs* in the structure as illustrated in figure 5(b). It contains two primal *QEs* and two dual *QEs*. For the sake of simplicity, we only draw connection for one point and one face from *QE* to *QEGeom* and *QEGeom* to ITK layer as shown in figure 5(b), conversely both points and faces are equally linked in the data structure. This data structure only need three operators as *Rot*, *Onext* and *Splice* to implement all other modifications (Euler operator) and accessibility of the mesh. Currently, *QuadEdgeMesh* data structure have topological duality but lack geometrical duality as represent in table 1. There are only few filters available in ITK that transform triangular mesh to simplex mesh but it is specific not generic to duality. Additionally, in many cases it is of much interest to have the both representation of a discrete surface directly integrated in the structure. Our contribution includes an extension of data structure that contain both primal and dual mesh simultaneously,

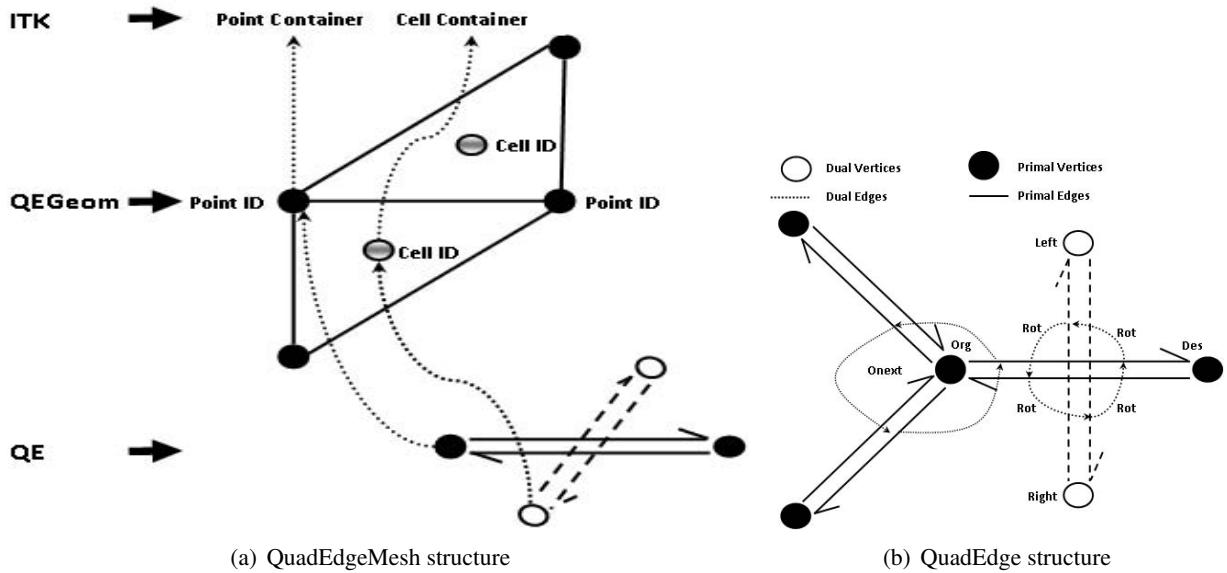


Figure 5: Existing data structures

Table 1: QuadEdgeMesh Data Structure

	Primal	Dual
Geometry	Yes	<u>No</u>
Topology	Yes	Yes

a filter that transform primal mesh to primal/dual mesh just using single data structure and an adaptor for displaying dual mesh.

### 3.2 Extension in data structure, *QuadEdgeMeshWithDual* data structure

We create a new class *itk::QuadEdgeMeshWithDual* derived from *itk::QuadEdgeMesh*. This class now stores both primal and dual mesh simultaneously. The new design of *QuadEdgeMeshWithDual* data structure is contained double reference i.e., one for primal point to dual cell and one for primal cell to dual point as depicted in figure 6(a). For the sake of simplicity, we only draw connection from QE layer to QEGeom layer and QEGeom layer to ITK layer for one point and one face instead of both points and both faces as shown in figure 6(a). The primal and dual overlapping structures of connections at QEGeom layer is shown in figure 6(b). Furthermore, this class contains three new containers; *DualPointsContainer* for dual points, *DualCellsContainer* for dual cells and *DualEdgeCellsContainer* for boundary edges and three new functions; *AddDualPoint* for adding dual point, *AddDualFace* for dual cells (polygon) and *AddDualEdge* for boundary edges.

In order to keep the primal-dual references in a single data structure, we have two design options. In first design, we use to maintain two look up tables; one table for storing references of primal cell to dual point and and second table for primal point to dual cell. The advantage of this approach is backward compatibility of code and test cases. The bad side of this design is to maintain these tables that having the complexity  $n \log(n)$  causing severe degradation of performance in case of large mesh. In second design, we modify the

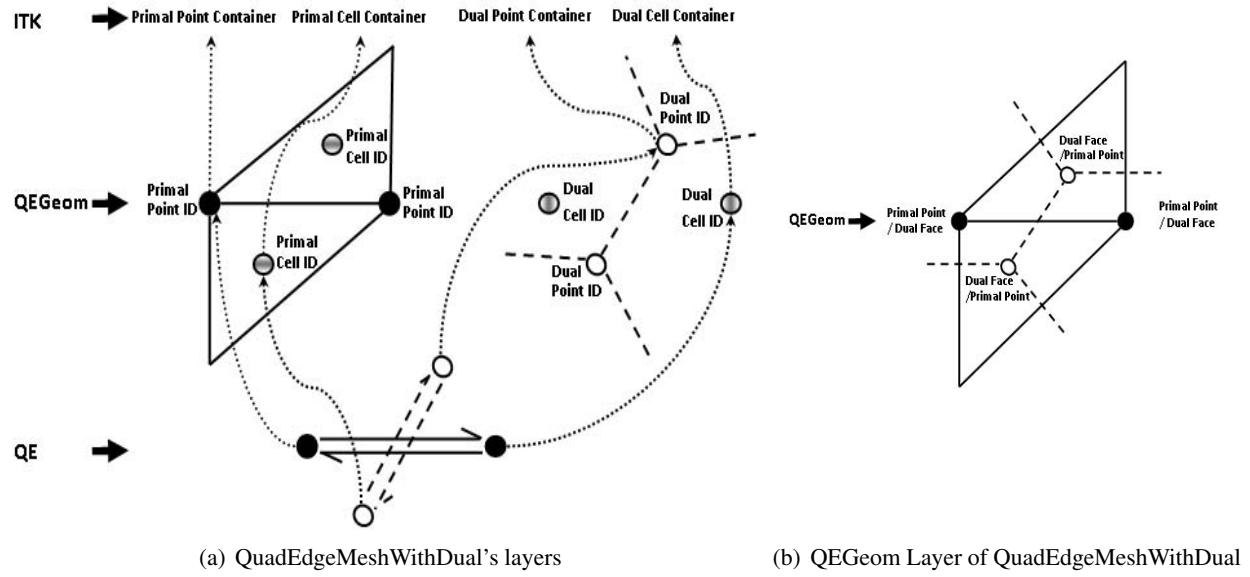


Figure 6: QuadEdgeMeshWithDual data structure

Table 2: Summaries of changes in new data structure

		Old Data Structure	New Data Structure
Changes	OriginRef Type	Point ID Cell ID	<i>Pair &lt; Point, Cell &gt;</i> <i>Pair &lt; Cell, Point &gt;</i>
Additions	Dual Containers	-	<i>DualPointsContainer</i> <i>DualCellsContainer</i> <i>DualEdgeCellsContainer</i>
Not yet implemented	Dual Data Containers	-	-

existing data structure by adding two reference pair; primal point to dual cell and primal cell to dual point as shown below. With this design, no look up table is required to maintain the primal and dual references. So it is very efficient approach but not compatible with respect to previous code and test cases.

```
typedef GeometricalQuadEdge<
    std::pair<PointIdentifier, CellIdentifier>,
    std::pair<CellIdentifier, PointIdentifier>,
    PrimalDataType,
    DualDataType
> QEPrimal;
```

A summaries of changes in new data structure can be depicted in Table 2

### 3.3 Primal to primal/dual filter

In order to transform primal mesh into dual mesh, we also create a new filter called *itk::QuadEdgeMeshToQuadEdgeMeshWithDualFilter*. This filter is templated with *QuadEdgeMeshWithDual* data structure. This filter generate dual mesh from primal mesh in three phases; first phase is computing

dual point from primal cells, second phase is computing dual cells from primal points, and in last phase, primal borders edges are used to generate dual border cell.

We also implement a new adaptor for connecting the dual mesh natively to a downward pipeline.

### Dual point functor

As explained before, there is no geometrical duality between the primal and the dual. Therefore any formula that compute points that satisfy the criteria of duality detailed in section 2.1 can be use. Not to restrict ourselves to a single option that may limit the application of the filter, a functor is used to compute the dual point. Depending on the case faced, the user is able to choose from the already two existing dual point functor, or use his own functor. Except from the classic ITK macro, *typedef* definition and constructor/destructor, the functor has only one method where the process is done.

```
template< class TInputMesh, class TOutputMesh=TInputMesh>
class DualPointFunctor
{
    typedef typename TInputMesh::CellsContainer CellsContainer;
    typedef typename CellsContainer::ConstIterator CellIterator;
    ...
    inline OutputPointType
    operator() ( const TInputMesh* primalMesh, CellIterator cellIterator )
    { ... }
};
```

**Barycentre** By default, the barycentre of each face is used to compute the location of the dual point. It has the advantage to be relatively straightforward to compute, to compute a dual point which is always located within the face and to work with any kind of face. The following equation is used to compute the centre

$$M = \frac{P_1 + P_2 + \dots + P_n}{n} \quad (7)$$

where  $P_1, P_2, \dots, P_n$  are the points retrieve from the current cell.

**Circumcentre** The circumcentre is a particular dual point of triangle mesh. It is the centre of the circumference circle of a triangle and is determine by the crossing point of the perpendicular bisectors. As such, it is not always within the face, and more costly to compute. The interest of this is in the case of the Delaunay triangulation in order to obtain its dual, the corresponding Voronoi tesselation. The following equation is used to compute the centre

$$M = P_1 + \frac{|P_3 - P_1|^2 [(P_2 - P_1) \times (P_3 - P_1)] \times (P_2 - P_1) + |P_2 - P_1|^2 [(P_3 - P_1) \times (P_2 - P_1)] \times (P_3 - P_1)}{2 |(P_2 - P_1) \times (P_3 - P_1)|^2} \quad (8)$$

where  $P_1, P_2, P_3$  are the points retrieved from the current triangle, and  $\times$  the cross product. In order to simplify the calculus and avoid the use of square roots the edge length are squared and the coordinates of all the point relative to the first point  $P_1$  are used. Due to the floating-point errors such solution may be unstable in the case of the denominator is close to 0. To prevent such case, the exact geometric predicate implemented for ITK [4] is use for the cross product calculation.

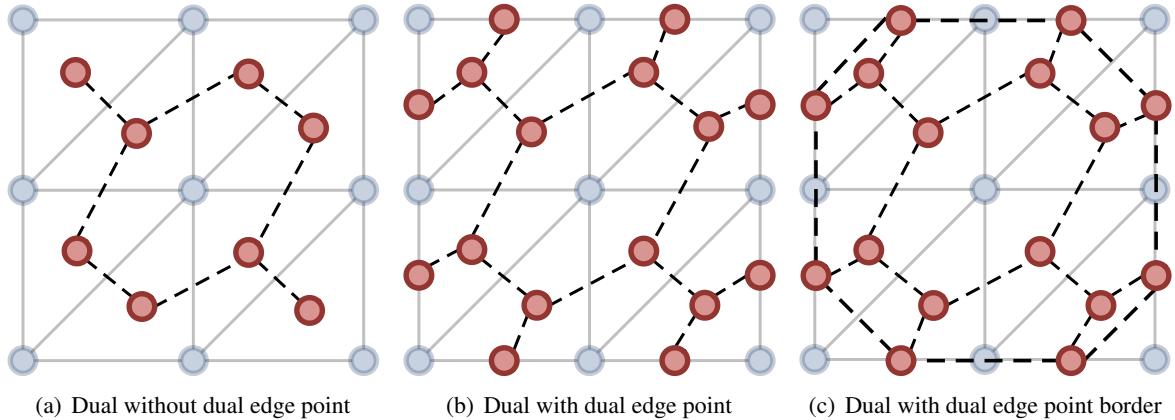


Figure 7: Dual borders management options

### Dual borders calculation

As shown in figure 3, the dual of a primal mesh that contain a border is not a closed mesh. The dual edge point obtain from the primal border are not included into any faces of the dual. This representation may be problematic to some other process which may not take into account the *EdgeContainer* that store the edges in the *QuadEdgeMesh* structure, and therefore discard edges that are not part of any face (*e.i.* Mesh writting filter). If the dual edge point are not compute, the effect still occur but is less important than in the previous case. Another option is to create a border by connecting the dual edge point, however this solution may lead to some flipped triangles in specific configuration. The *SetBorders()* methods allow the user to decide how the filter should manage the borders (Fig. 7). By default the filter will compute the dual edge point and create a border to the dual mesh.

## 4 Validation

### 4.1 Test on planer triangular to simplex mesh with and without holes

We create a square triangulated (primal) mesh as shown in figure 8(a). Green color represents primal points and cells. From this primal mesh, we would try to generate dual mesh. First, we generate dual points using the *BarycentreDualPointFunctor* on the primal cells as shown in figure 8(b) with red points. We add these dual points in *m\_DualPointsContainer* of *itk::QuadEdgeMeshWithDual* by using *AddDualPoint()*. Second, we iterate around each primal point to form dual cells and add dual cell in *m\_DualCellsContainer* of *itk::QuadEdgeMeshWithDual* by using *AddDualFace()*. By doing this we generate all dual cells except boundary cells. Dual cell are represented by red color in figure 8(b).

In order to tackle borders, first we get boundary edges of primal mesh. Select one boundary edge from list; create a new point (dual) in the middle of edge and add in *m\_DualPointsContainer* of *itk::QuadEdgeMeshWithDual* by using *AddDualPoint(...)*. In figure 8(c), red points on border lines represent boundary points of dual mesh. Then, find the dual point associated with the face on the left and make an edge between these two dual points. Now iterate along left triangle to form dual cell and add this dual cell into *m\_DualCellsContainer* of *itk::QuadEdgeMeshWithDual* by using *AddDualFace()*. In figure 8(c), red cells represent dual cells. The final dual mesh generated from primal mesh is shown in following figure 8(d). For testing this data structure and filter, we deleted one primal edge and re-run the whole code for getting

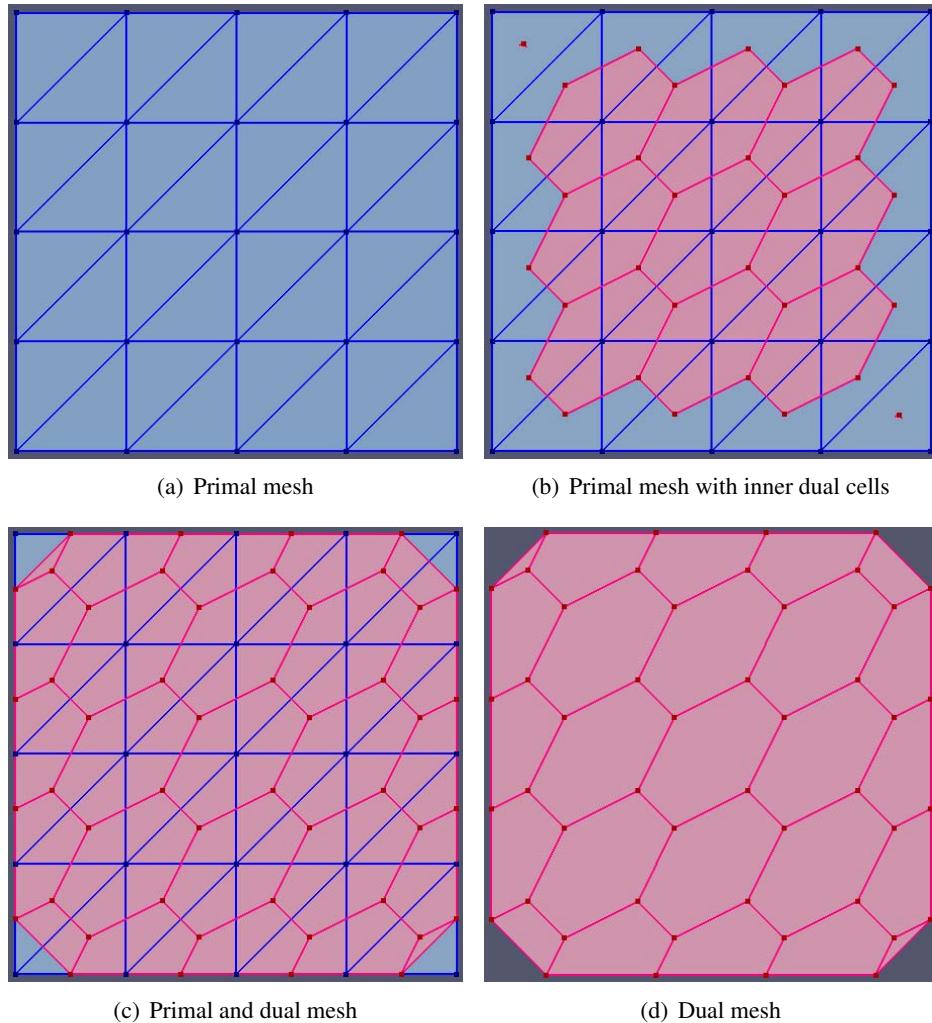


Figure 8: Primal to dual mesh

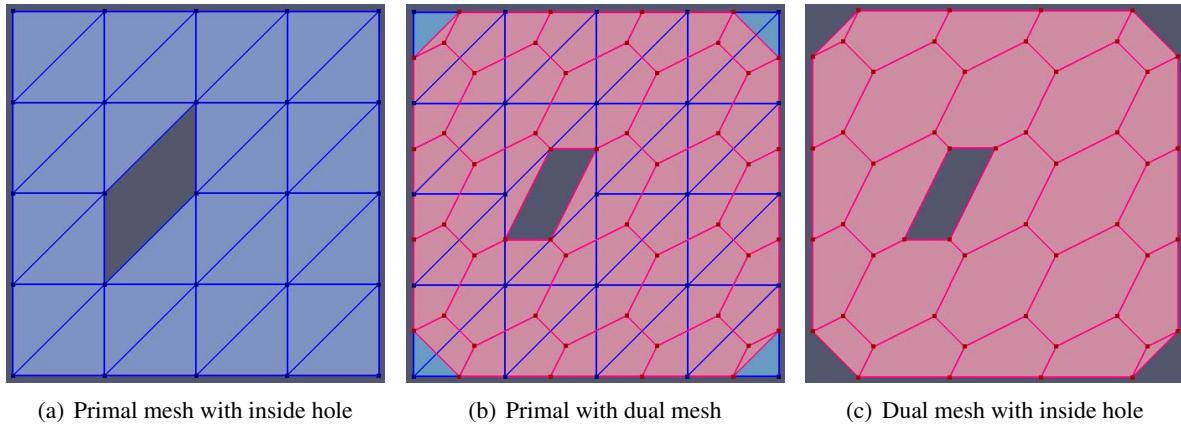


Figure 9: Primal to dual mesh with inside hole

dual mesh. The snapshot of re-run is shown in figure 9.

#### 4.2 Test with Delaunay to Voronoi

Using the *PointSetToDelaunayTriangulationFilter* [5], we tested this data structure on Delaunay mesh to Voronoi diagram. We input a planer Delaunay mesh into new data structure as shown in figure 10(a) and generate the corresponding Voronoi diagram by using *QuadEdgeMeshToQuadEdgeMeshWithDualFilter* and the *CircumcentreDualPointFunctor* as shown in figure 10(b). Later, the Voronoi diagram is shown in figure 10(c) using new adaptor *itk::QuadEdgeMeshWithDualAdaptor*.

#### 4.3 Test on non planar mesh

We perform last test on non-planer mesh. A spherical triangulation mesh can be seen in figure 11(a), generated simplex (dual) mesh along with triangulation (primal) mesh can be seen in figure 11(b) and finally, simplex (dual) mesh generated with new adaptor can be seen in figure 11(c).

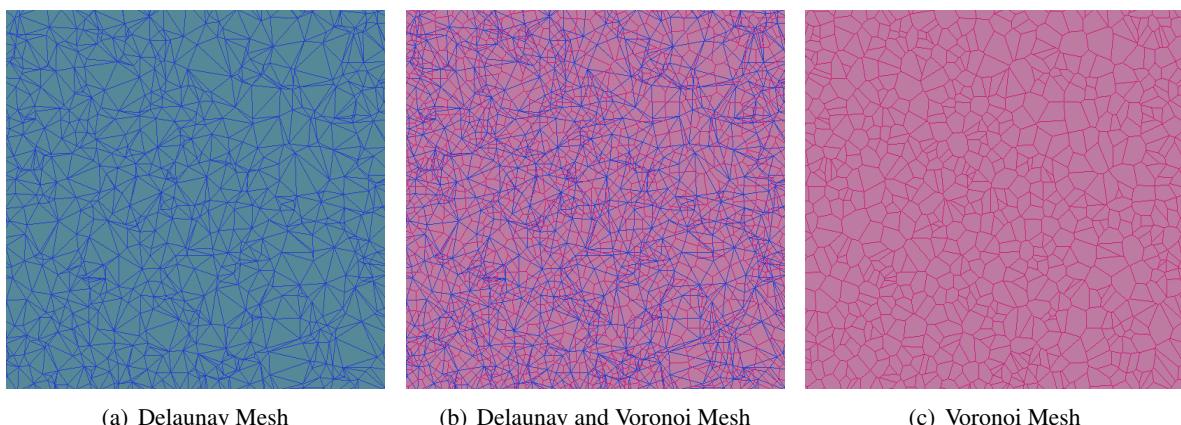


Figure 10: Delaunay to Voronoi Mesh

## 5 Usage

An example *SimplexMesh.cxx* is provided with the sources and is used for the tests. The filter *QuadEdgeMeshToQuadEdgeMeshWithDualFilter* is templated on float and 3 dimensions *itk::QuadEdgeMeshWithDual*.

```
// Typedef definition
typedef itk::QuadEdgeMeshWithDual< float, 3 > MeshType;
typedef itk::QuadEdgeMeshToQuadEdgeMeshWithDualFilter< MeshType >
FillDualFilterType;
typedef itk::QuadEdgeMeshWithDualAdaptor< MeshType > AdaptorType;

typedef itk::VTKPolyDataWriter< MeshType > MeshWriterType;
typedef itk::VTKPolyDataWriter< AdaptorType > DualMeshWriterType;

// Create primal mesh
MeshType::Pointer myPrimalMesh = MeshType::New();
CreateSquareTriangularMesh< MeshType >( myPrimalMesh );

// Create dual mesh
FillDualFilterType::Pointer fillDual = FillDualFilterType::New();
fillDual->SetInput( myPrimalMesh );
fillDual->Update();

AdaptorType* adaptor = new AdaptorType();
adaptor->SetInput( fillDual->GetOutput() );

// Write dual mesh
```

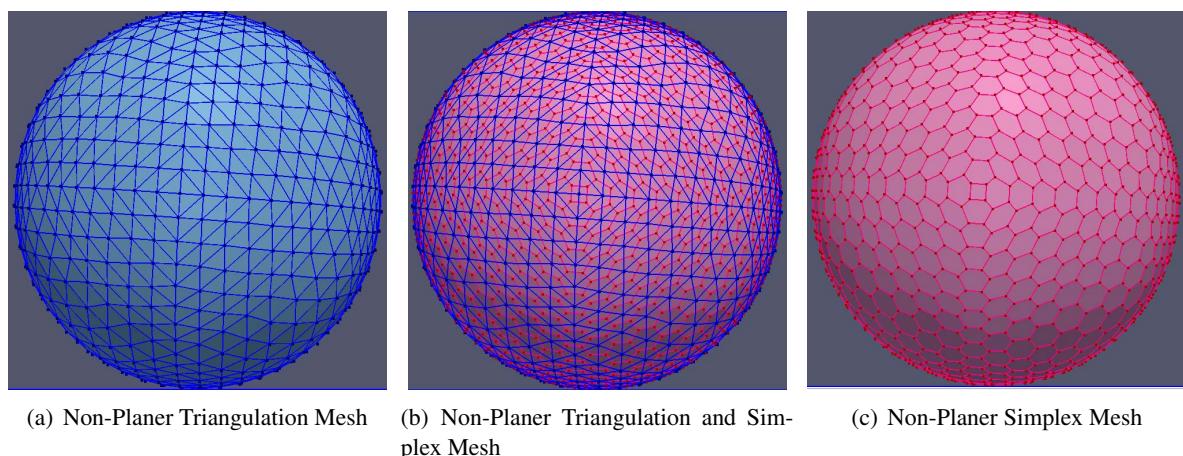


Figure 11: Non-Planer Mesh containing (Triangulation and Simplex Mesh)

```
DualMeshWriterType::Pointer writer = DualMeshWriterType::New();
writer->SetInput( adaptor );
writer->SetFileName( "TestSquareTriangularSimplexMesh.vtk" );
writer->Write();
```

## References

- [1] H. Delingette. Simplex meshes: a general representation for 3d shape reconstruction. In *Computer Vision and Pattern Recognition, 1994. Proceedings CVPR '94., 1994 IEEE Computer Society Conference on*, pages 856 –859, jun 1994. [1.3](#)
- [2] A. Gouaillard, L. Florez-Valencia, and E. Boix. Itkquadedgemesh: A discrete orientable 2-manifold data structure for image processing. *Insight Journal*, Sep 2006. [\(document\)](#), [3.1](#)
- [3] Lutz Kettner. Using generic programming for designing a data structure for polyhedral surfaces. *Comput. Geom. Theory Appl.*, 13:65–90, 1999. [1.1](#)
- [4] B. Moreau and A. Gouaillard. Exact geometrical predicate: Point in circle. *Insight Journal*, Nov 2011. [3.3](#)
- [5] S. Rigaud and A. Gouaillard. Incremental delaunay triangulation. *Insight Journal*, Jul 2012. [4.2](#)