itkQuadEdgeMesh: A Discrete Orientable 2-Manifold Data Structure for Image Processing

Release 1.00

A. Gouaillard[1], L. Flórez-Valencia[2,3] and E. Boix[4]

September 9, 2006

[1]Créatis Lab., INSA de Lyon. France.
[2]Theralys, Diagnostic & Therapeutic Image Analysis in Clinical Trials, Lyon, France.
[3]Universidad de los Andes, Facultad de Ingeniería, Santafé de Bogotá, Colombia.
[4]École Nationale Supérieure, Lyon. France.

Abstract

There is nowadays an increasing need in interaction between discrete surfaces (2-manifolds) and images. More specifically, segmentation and registration of n-dimensional images are taking advantage of a priori geometrical information, most often provided as discrete 2-manifolds.

Most of the publicly available libraries are oriented either toward mesh processing or image processing. Through a careful study we will show that none of those libraries are complete enough to fullfill image, surface and joint image-surface interaction.

We propose to implement in ITK library a powerful 2-manifold data structure. The choice of ITK was driven by the fact that it provides the best base framework along with a strong n-dimensional image kernel. Based on a Quad-Edge data structure, it has been specifically tailored not only to represent orientable 2-manifolds (surfaces of real objects) but also ease further processing.

We illustrate the integration of the design into ITK as a native object, enhancing existing algorithms. We also illustrate the power of the new design for further surface processing.

Keywords: Surface-Image Processing, 2-Manifold, Quad-Edge

Contents

1 Introduction

As said before, there should be an easy interaction in between discrete 3D surfaces and image processing. For instance segmentation and registration algorithm could use discrete representations of 3D surfaces as a-priori geometrical information to improve the quality of their result. Fields of application are numerous and include, to cite a few, medical image processing [6], computer vision [7] and multimedia processing [5].

In these cases, discrete surfaces are used to represent the geometry of the surface of an existing solid object. Surface of solid objects are always orientable and are always locally homeomorphic to a disk. We can thus reduce the notion of discrete surfaces to the notion of discrete orientable 2-manifolds. Orientation is of acute importance here as in most of the application involving deformation of a surface, the normal of the surface is part of the deformation kernel and should be uniquely defined. The 2-manifoldness is supposed in most of the mesh processing algorithms, unfortunately it is not always enforced by the mesh data structure, relying on the end user the handling of degenerated cases.

Discrete surface processing, taken as a tool for image processing, need specific interactions with images. Typically, a vertex of the discrete surface should be aware of the neighboring voxels of the image within which the surface is embedded. This suppose a specific design with a common layer for surfaces and meshes. In the next section we will review the principal publicly available libraries. We focus on the surface processing, image processing, and surface-image interaction features of those libraries.

1.1 Review of existing publicly available packages.

Major freely available packages have been listed in table 1.

We can observe that the packages are oriented toward either image processing or mesh processing with the exception of ITK which has specifically been designed with image and mesh interaction in mind.

The C-GAL library [1], a reference for mesh processing, does not unfortunately implement any kind of image data structure. Therefore, nothing is done to handle image - mesh interactions. Other mesh processing

[1] Computational Geometry Algorithm Library, [18]

	Image processing	Surface processing	Image-surface processing
VTK	**	**	*
ITK	***	*	***
C-GAL		****	

Table 1: This table list the major publicly available libraries that propose image processing, mesh processing, and/or joint image and mesh processing. We use a four-star rating system to indicate the quality of each package for image processing, surface processing, and joint image-surface processing. C-GAL, as being the reference for surface processing, deserves four stars. But since it does not provide image processing features, it does not get any star for image processing and joint image-surface processing. We clearly see that VTK is well-balanced between image and surface processing, but it lacks of a suitable image-surface processing. ITK provides a good image-surface framework, but does not include any surface processing algorithms. It only provides a common data-structure to handle those objects.

libraries like OpenMesh, GTS, ... could be listed in the table as well. As they all suffer from the same limitations of C-GAL, without being as good, they have been eluded here.

The VTK library [2], is dedicated to visualization. It integrates not only image and mesh data structures but also numerous filters for these structures. A specific data structure (PolyData) is provided for surfaces, another for 3D-meshes (UnstructuredGrid). Eventhough a PolyData could be seen as a subset of an UnstructuredGrid, the design of VTK in the actual version (5.0) define them as two different objects, and thus the filters must be implemented for both of them. Although it seems a well balanced library, drawbacks are twofold. First the PolyData structure does not enforce orientation neither require the mesh to be restricted to a 2-manifold. This is understandable from a visualization point of view, but this is not acceptable from a processing point of view. Second, the interaction between images and meshes are quite delicate, because the image and mesh data structures do not share the geometric layer. Thus the embedding of the mesh in the image and the localization of voxels surrounding a vertex of the mesh is left to the end user.

Finally ITK [3] is a library dedicated to medical image processing. The algorithms provided can be used for a broader range of applications. A discrete n-dimensional discrete mesh data structure (itk::Mesh) is provided. Also based on [20], itk::Mesh suffers from the same issues as the data structure in VTK, e.g. for representation of surfaces, it does not enforce the discrete mesh to be an orientable 2-manifold. In addition to which, very few mesh processing filters are provided, except some simplex mesh representation filters. ITK propose numerous image processing algorithms and some implicit surface processing algorithms. In contrast with VTK, the design of the data structures allow direct interaction between images and meshes.

We can clearly see that there is no package that would fulfill our needs in image processing, mesh processing and image-mesh joint processing. According to our study, the best way to achieve our requirement would be to implement an orientable discrete 2-manifold data structure in ITK.

1.2 Discrete orientable 2-manifolds data structures

General overview of structures

There are three main kinds of data structure that could handle orientable discrete 2-manifolds, namely Winged-Edge (WE), Half-Edge (HE), and Quad-Edge (QE) data structure.

[2] Visualization ToolKit [20]
[3] Insight ToolKit

	Winged-Edge (WE)	Half-Edge (HE)	Quad-Edge (QE)	Quad-Edge orientable
Modeling pace	orientable 2-manifold		2-manifold	orientable 2-manifold
Operations	Euler operators		Splice operator	
Duality	at compile time (explicit with adapter)		at runtime (rot operator)	
Holes in facets	yes		no	
Basic Transversal	Case distinction	Direct access	Direct access	
Min size per edge	4 ptr	4 ptr	2 ptr + 2 bits	2 ptr + 2 bits
Max size per edge	8 ptr	10 ptr	8 ptr + 12bits	8 ptr + 8 bits

Table 2: This table is an updated and enhanced version of the table proposed in [18].

In [18], the author compare the three data structures. The table 2 is a more complete version of the table provided in this paper. WE is not really interesting because of its basic transversal capability which depends on a case distinction. Both HE and QE have direct access to their neighborhood. Quad-Edge data structure is also very space efficient, especially in its orientable flavor. Finally QE, still in its orientable flavor, has only one base operator Splice (two if Rot() is included) upon which all other methods that modify the mesh are based. This greatly simplify the code maintenance.

In [18] the author reject QE claiming that the symmetry of the original design [16] has to be destroyed during the implementation. He says this has to be done because face type and vertex type will be different, even the more different as one want to add informations like color or normal. He also claims that the base operator Splice() would have to be written twice. The latter argument is not really an issue, but rather a matter of one template parameter to make the splice operator handle both primal and dual types. Only one piece of code remains to be maintained for Splice().

According to this study, HE and QE are both interesting, with a little advantage for QE (orientable) which is more space efficient, and which would require less code maintenance for integrity. The choice between HE and QE can only be made depending on the applications. In the next section, we define the classes of applications of interest. We then investigate which structure among HE and QE would be best for them.

The surface processing point of view

We have two kinds of surface processing applications in mind, geometry processing and topology processing. We will consider arbitrary modification of the connectivity as geometry processing.

Surface mesh processing includes for example smoothing ([22]), decimation ([11, 17]), computing discrete differential geometry invariants like curvature ([4, 8], subdivision ([23]), multi resolution ([19]), parameterization ([15]), remeshing ([1, 2]), and many more. A more elaborate survey of mesh processing can be found in [3]. This paper present only mesh processing algorithms, but we believe that image processing using mesh could take advantage of it. Several medical applications already include such processing [13, 14, 10].

Topology processing is also of interest. The result of isosurface extraction algorithms usually contains many extra numerous components that need to be removed before further processing. Many other algorithms can only process one component at a time and on top of that require to count, identify and/or remove components. Finally, about the interest of topology in extraction of isosurfaces from images, image noise can result

in topological inconsistencies that should be removed [24, 12]. We could define three kinds of topological algorithms: those which count things (counting components, borders, genus, Euler characteristic, ...), topological modifications (extracting and/or removing components, filling holes, cutting the surface open, ...) and computation of basis and other domains (homotopy basis, homology basis, fundamental domain, universal covering space). Stillwell, in its book [21] introduces basis of discrete computational topology that are of great potential in surface processing.

For all the above applications, the key functionality is local neighborhood access e.g. from a given vertex access the adjacent edges and/or faces. But one also wishes further reach access (local access with a superior edge based distance) e.g. neighbor vertices of a given vertex at edge distance of 2 or neighbor faces of a given face at edge distance of 3. With a HE data structure we need to distinguish the vertex neighbor local access algorithm from its face counterpart. This is to be opposed to the QE data structure for which we can design and maintain a single generic front algorithm for vertex local access (based on primal edges) or for face neighbor access (based on dual edges). If we move up to the next level, and implement Dijkstra's shortest path algorithm on top of a QE data structure we can indifferently use it either on primal or dual connectivity i.e. finding shortest path on vertices or on faces.

There are many others advantages for applications of having the dual representation for free. Some optimal algorithms require primal-dual approaches (delaunay-voronoi). Some very successful segmentation algorithms and surface modelization are based on simplex meshes [5, 6] Simplex meshes being the dual of triangulated meshes, the QE data structure provides native triangular-simplex implementation. This is in sharp contrast with ITK's implementation of simplex meshes, no transformation filter is required and thus there is no data redundancy and no extra transformation computational cost. The difference in design and implementation of a simplex mesh solution in ITK with and without the help of our data structure will be provided in section 3. Finally, it provides an algorithmic extension to algorithms only defined on triangulations, to simplex mesh (e.g. multi resolution, subdivision, ...).

Conclusion

Three data structures can handle discrete orientable 2-manifolds: WE, HE, and QE. We directly rejected WE because of its poor traversal capacities. We saw that although HE was the reference structure, QE has much to offer. QE is more space efficient and easier to maintain than HE. QE provides a robust base layer and constant complexity local accesses and modifications. Additionally, QE allows one to implement generic algorithms such as a neighborhood extraction algorithm that works for any kind of neighborhood (vertices, edges, faces). Finally, in many cases it is of much interest to have the dual representation of a discrete surface directly integrated in the structure. We will choose a QE data structure an integrate it in ITK. There are many ways to integrate a structure in a library. The next section will present the necessary technical background on both QE and ITK. It will also present our integration philosophy and illustrate the result.

2 Quad-Edge structure integration in ITK

2.1 Quad-Edge in details

The Quad-Edge data structure is presented in detail in [16]. We will only emphasis here the key points that will be needed for integration. One physical edge will be called Full Edge (FE) hereafter.

For each FE, there will be 4 QE in the structure, as illustrated in figure 1. the 4 QEs of a given FE are linked by the *Rot()* operator. This operator is cyclic, thus defining a *Rot Ring*.

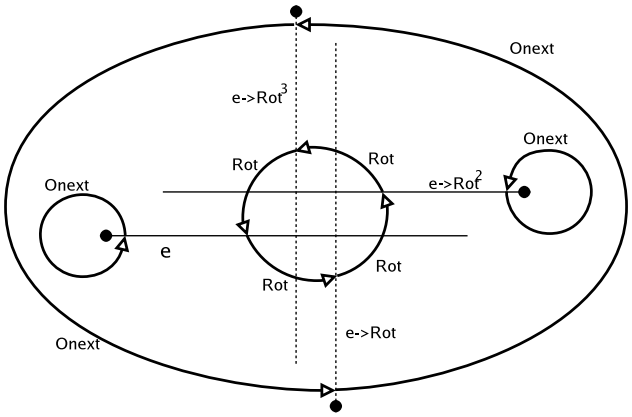

Figure 1: Illustration of itkQE::Mesh Rot Ring. Each physical edge (hereafter Full Edge: FE) is represented by four oriented edges. We thus can see each of those edges as one quarter of the FE, giving another ways of remembering the name QE. This notation would be consistent with the Full Edge (1 physical edge gives 1 edge in the structure) and Half Edge (1 physical edge gives 2 edges in the structure) namings. Unfortunately, the name has been chosen to be Quad-Edge. Each QE *e* accesses the next QE within the representation of a physical edge through the *Rot()* operator. Assuming that *e* is a primal edge, an odd number of calls to *Rot()* yields a dual edge, whereas an even number of calls to the same operator yields a primal edge. Calling *Rot()* 4 times brings you back to where you started. The *Rot()* structure is thus cyclic and is called *Rot Ring*. Another interesting property is that calling *Rot()* twice in a row gives you the opposite edge (same direction, opposite orientation).

All the QEs are also linked to the next QE on the surface, with respect to orientation, by the *ONext()* operator as illustrated on figure 2. This operator is also cyclic, defining an *ONext Ring*. It should be noted that although the *Rot Ring* always contains 4 QE, the number of QEs in a given *ONext Ring* is function of the connectivity of the discrete surface.

All traversal features can be composed from these two operators. For more details, the reader can refer to [16]. All topological changes are based on the *Splice()* operator, itself based on the two previous operators. See figure 3 for an illustration, and section 3.2 for an example.

To each *ONext Ring* can be attached data, to which each QE in the ring will refer to as *Org*. It can be, for example, a reference to the geometric layer by mean of vertex or face location in the corresponding container. It is illustrated in figure 4

This different operators and object are sufficient to implement the QE data structure. The next section will detail the itk::Mesh data structure. Then a following section will then show our implementation of a QE-Mesh in ITK.

2.2 itk::Mesh in details

The itk::Mesh data structure is described in [20]. It was initially designed for visualization and thus suffer from several drawbacks. First it does not enforce a surface to be a 2 orientable manifold as we would need for processing. But there is also a very high computational cost for any modification of the structure of the mesh, we are going to detail here.

Most of the local accesses in the itk::Mesh is made through two links tables that maintain neighborhood information about each vertex and face. Those tables are build (and unfortunately maintained) through a

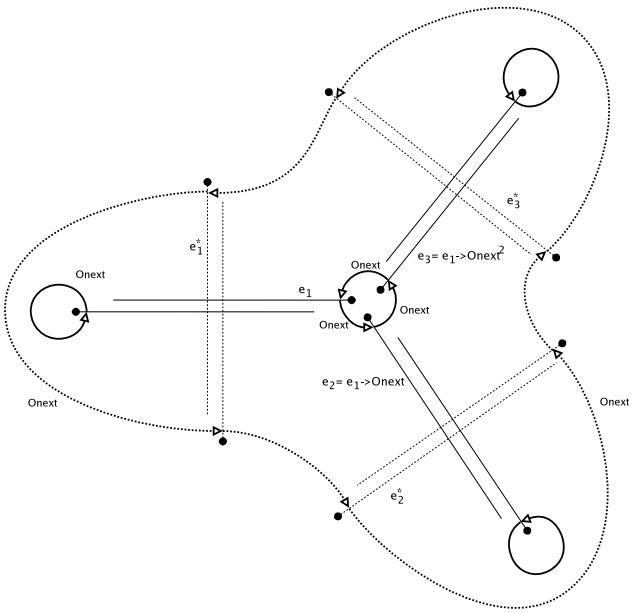

Figure 2: Illustration of itkQE::Mesh *Onext Ring*. Each QE can access its next edge around its origin through the *Onext()* operator. By construction, the QE are cyclically linked by *Onext()*, defining the *Onext Ring*. We can attach to each *Onext Ring* some additional data named *Org*. This is used to link the QE topological layer we illustrated in figure 1 and in this one, and the geometrical layer we will illustrate in following figures. At this level of the structure, it does not matter if the QE is primal or dual, i.e. if the data attached to the *Onext Ring* is related to a vertex or to a face. Also note that, in contrast with the *Rot Ring*, the number of edges in the *Onext Ring* is not constant and depends on the connectivity of the mesh. In this example, the central *Onext Ring* is made of three QEs.

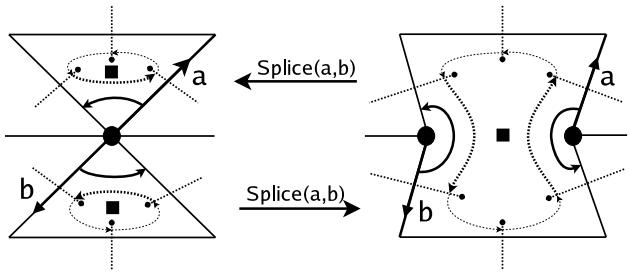

Figure 3: Illustration of itkQE::Mesh::Splice() operator. The *Splice()* operator is the only operator that modifies the connectivity of the mes. It is usually defined as **trading a vertex for a face**. In this illustration, on the left, the *a* and *b* QEs share the same *Org* but their dual don't. In other words, noticing that *a* and *b* are primal QE, they have a common origin vertex but do not share the left face. A call to *Splice(a,b)* will result in the inverse situation: *a* and *b* sharing the left face but having different origin vertices. We traded a 1 point, 2 faces for a 2 points one faces situation. Interestingly enough, *Splice()* is its own inverse. Two consecutive calls to *Splice(a,b)* will leave the mesh unchanged.

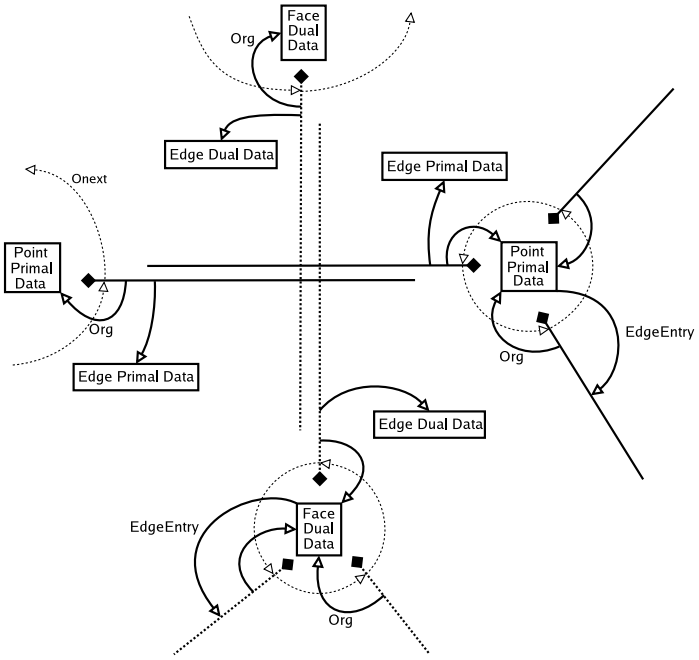

Figure 4: Illustration of itkQE::Mesh Geometric Layer. Figures 1 and 2 limited themselves to the topological operations, as defined in [16]. To be local, on top of this topological layer, we must plug the geometry. This is done through the definition of the origin (*Org*) of the *Onext Rings*. This drawing illustrates how the 4 QEs corresponding to the same FE access the geometry. On this drawing we can see that primal QEs have vertex information attached to their *Onext Rings*, whereas dual QEs (represented with dashed lines) have face informations. The link between topological layer and geometrical layer is made both ways, thought not symmetrically. Each QE has an entry to its *Org* (either vertex or face). Each *Org* has access to only one QE in the corresponding *Onext Ring*. Getting all the edges referencing this particular *Org* is then just an iteration around the *Onext Ring* away.

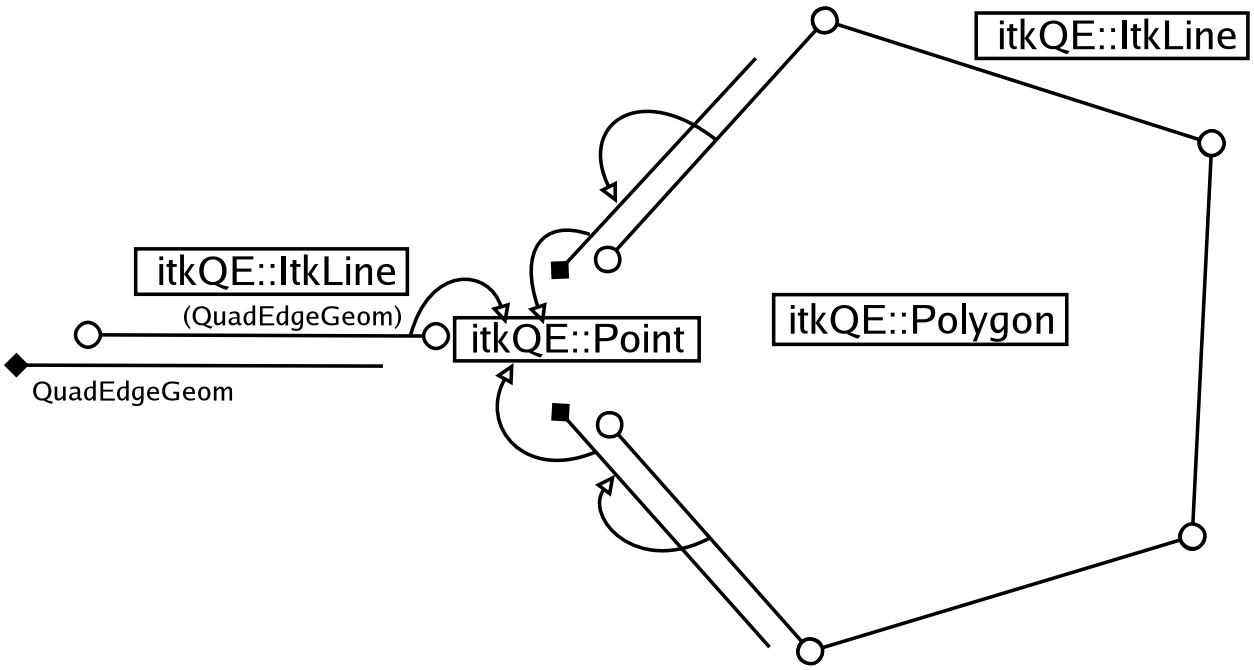

Figure 5: Illustration of the interface between QE layer and ITK layer.

itk::Mesh::BuildCellLink() method. It is necessary to be sure that these link tables are up to date before calling several methods of itk::Mesh to keep the integrity of the itk::Mesh structure. Hence after any modification a call to BuildLink() must be done. For each call, the links are deleted, then several transversal must be done on all the structure to (re)build the link tables again. The complexity of a local modification is thus at least linear with respect to the size of the mesh. This is in sharp opposition with the QE data structure for which any local change will have only local impact, in constant time. This is of acute importance for algorithms that iteratively build or modify a surface like delaunay triangulation of point clouds, decimation, ...

2.3 Integration

Our aim is to implement the QE structure as a native ITK object, inheriting from itk::Mesh. The end user must be able to replace any itk::Mesh (provided that it represents a surface) by a itkQE::Mesh in an existing code. Not only can the code work without any other change, but the user benefits of speed enhancements. A second advantage of this approach is the possibility to use ITK at its full potential. The pipeline design remains available, along with other parallelization features and multi resolution frameworks, to cite a few.

This has a great impact on the implementation. itk::Mesh benefit from a global cell interface design. In order to keep the genericity of the design, and the access to global cell iterators, we need to provide new cell and cell interface classes for the basic QE items. We also need to maintain the integrity of itk::Mesh based code at the itkQE::Mesh level all the methods that should not be called anymore like BuildLinks().

The figure 9 is an overview of the global design. We can see on the left the pure topological level of QE data structure. This layer being purely topological no distinction is made between primal and dual QEs. Next column on the right represents the geometrical layer. Here appears the *Org* and *Data* informations that were introduced in figure 4. These two fields are being differentiated depending on the type of the

QE (primal or dual). Finally,the last column on the right shows the integration of the QEs in ITK through new classes: itkQE::PolygonCell, itkQE::LineCell, itkQE::VertexCell. Normal arrows illustrate link, arrows with an empty triangle on target represent inheritance relations.

2.4 Our contribution: a neighborhood iterator

Our main contribution to the work of [16] is the definition of a special *neighborhood* iterator. This iterator extracts neighborhood, with the respect to a metric defined on the connectivity graph of the discrete surface. If a simple metric is used (unweighted graph) then it will extract neighborhoods depending only on the connectivity. If euclidean distance (defined on edges) is used, then it will extract neighborhood depending on the geodesic distance to the seed. We saw in section 1.2 that accessing and/or extracting neighborhoods was a basic in most of the discrete surface processing algorithms. This contribution will be of tremendous impact on further filters implementations.

Thanks to the symmetrical design of the QE structure, neighborhood can be defined on the primal connectivity or on the dual connectivity. If, for example, we take a triangular mesh, we can with the same algorithm get the neighboring vertices of a given vertex, or the neighboring faces of a given triangle.

One direct application of this iterator is extracting shortest paths. Indeed, while extracting neighborhood we keep track of followed paths, this would just be the implementation of a Dijkstra's shortest path algorithm. Seen from the primal point of view, the dual of this shortest path algorithm is a geodesic region growing algorithm. If, for example, we take a triangular mesh, the algorithm run on primal connectivity outputs a shortest path (and/or a list of visited vertices), and the algorithm run on dual connectivity, tagging the visited faces, can output a region/patch.

This functionality was implemented as an iterator to ease further processing. The iterator can be defined at any moment in the code, and the end user has full control over the processing. We also implement it as a class. It gives a finer control over the processing (with adaptative granularity), but the iterator gives a more elegant syntax.

2.5 Conclusion on integration

Our main concern during the integration was to fully comply with ITK design. It leads to redefining the cell interface layer to handle underlying QE data structure. The QE data structure (for orientable 2 manifold) along with the splice method were all implemented underneath. One of our contribution to the original [16] design is a neighborhood iterator as needed by most if not all discrete surface processing algorithms. In the next section we will first validate the integration in ITK, then we will test and illustrate the induced gains: speed, code syntax, code volume, duality, ...

3 Validation and examples

3.1 Validation

The validation process is twofold. First we tested the integration of the structure in ITK, we then tested the speed enhancement resulting from the new design allowing local accesses and modifications.

ITK provides a test suite. In order to test the integration, we ran the test suite using itkQE::Mesh wherever itk::Mesh was used. The impact on the code is minimal, being almost just a matter of replacing itk::Mesh

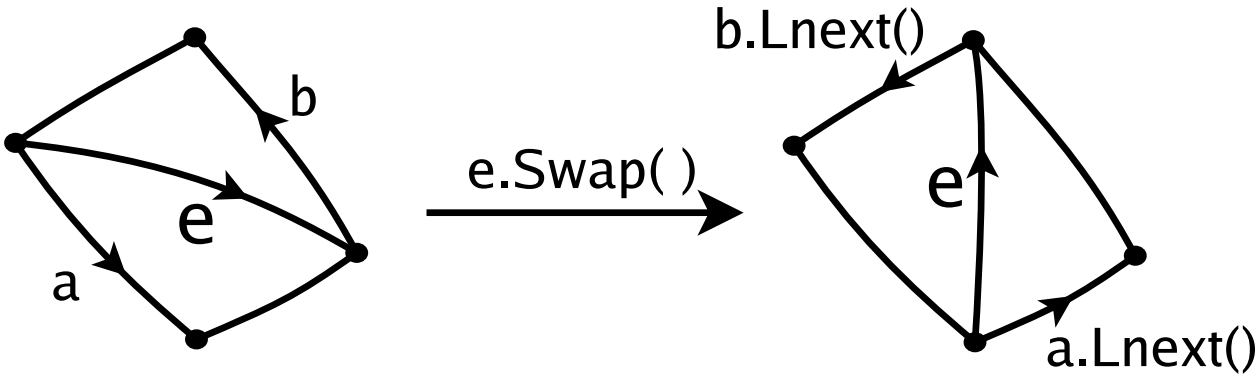

Figure 6: Swapping the edge e.

by itkQE::Mesh wherever the first appeared. All tests ran smoothly.

We tested the speed enhancement in the following way. Taking the same mesh, on the same computer (mobile P3 with 256Mo RAM, running WinXP) with all the codes compiled the same way (CMAKE 2.0, ITK 2.0.1, ITKApplication 2.0.1, MSVC 7.1 .NET 2003), we removed a face from the mesh and forced the integrity of the mesh. With the itk::Mesh design, this means calling the BuildCellLink() method, while with the itkQE::Mesh, this is just removing a face, the integrity of the surface being guaranteed at itkQE level. Thus, the complexity of removing a face is linear with itk::Mesh and constant with itkQE::Mesh. For example, with a 100000+ triangles model, it took 2.28 seconds to itk::Mesh and 0.0000249 seconds for itkQE::Mesh.

3.2 Example of Splice power: swapping an edge

In order to provide some evidence of the simplicity of usage of the Splice operator, let us consider the classical higher-order topological operator that swaps an edge (as illustrated by figure 6). The following code snippet is the direct translation within itkQE of the original design of the Swap operator presented in [16] (refer to chapter 6, page 104):

```
template< ... > bool QuadEdgeGeom< ... >::Swap( )
{
   Self* e = this;
   Self* a = e->GetOprev( );
   Self* b = e->GetSym( )->Oprev( );
   // Disconnect e from a and b:
   e->Splice( a );
   e->GetSym( )->Splice( b );
   // Reconnect e with a.Lnext() and b.Lnext():
   e->Splice( a->GetLnext( ) );
   e->GetSym( )->Splice( b->GetLnext( ) );
}
```

3.3 Example of simplicity of generic programming

We implemented a classical front based traversal algorithm that starts from a given reference edge and flows across the edges in a Dijkstra fashion. This traversal of the mesh is conveniently offered to the user in the form of an iterator as illustrated by the following code snippet.

```
typedef
   FrontIterator< MeshTypeArg,
                  MeshTypeArg::QEType > FrontIterator;
FrontIterator it;
for( it = mesh->BeginFront( edgeSeed );
     it != mesh->EndFront( );
     it++ )
{
    PointIdentifier org = it.Value( )->GetOrg( );
    ...  // Do something smart with the vertex
}
```

The very design of the QE data structure guaranties that each step of the iteration has a constant time complexity. Although by default this front traversal uses unweighted edges, the FrontIterator can be initialized with weights over edges (e.g. the ambient euclidean distance).

We can now take advantage of the native support for duality of the QE data structure and combine it with the generic algorithm implementation of the FrontIterator. The following code snippet provides an example of a dual version of the Dijkstra algorithm i.e. a Dijkstra algorithm walk on the dual edges, that can be seen as a front propagation over the faces of the primal mesh.

```
...
typedef
   FrontIterator< MeshTypeArg,
                  MeshTypeArg::QEDual > FrontDualIterator;
FrontDualIterator it;
for( it = mesh->BeginDualFront( );
     it != mesh->EndDualFront( );
     it++ )
{
    FaceIdentifier org = it.Value( )->GetOrg( );
    ...  // Do something smart with the face
}
```

Note that this is achieved by simply changing the types used for the FrontIterator instantiation (and of course the derived types). The figures 7 and 8 illustrate the direct application of this Dijkstra based iterator to the computation of the shortest path between two arbitrary vertices of triangulations.

3.4 Example of duality importance: simplex mesh implementation

Users most often deal with discrete surface as triangular meshes, that are the usual output of isosurface extraction algorithms, and also a standard for visualization. But simplex meshes have also proved to be very efficient discrete surface representation for segmentation. Interestingly, simplex meshes are dual to triangular meshes.

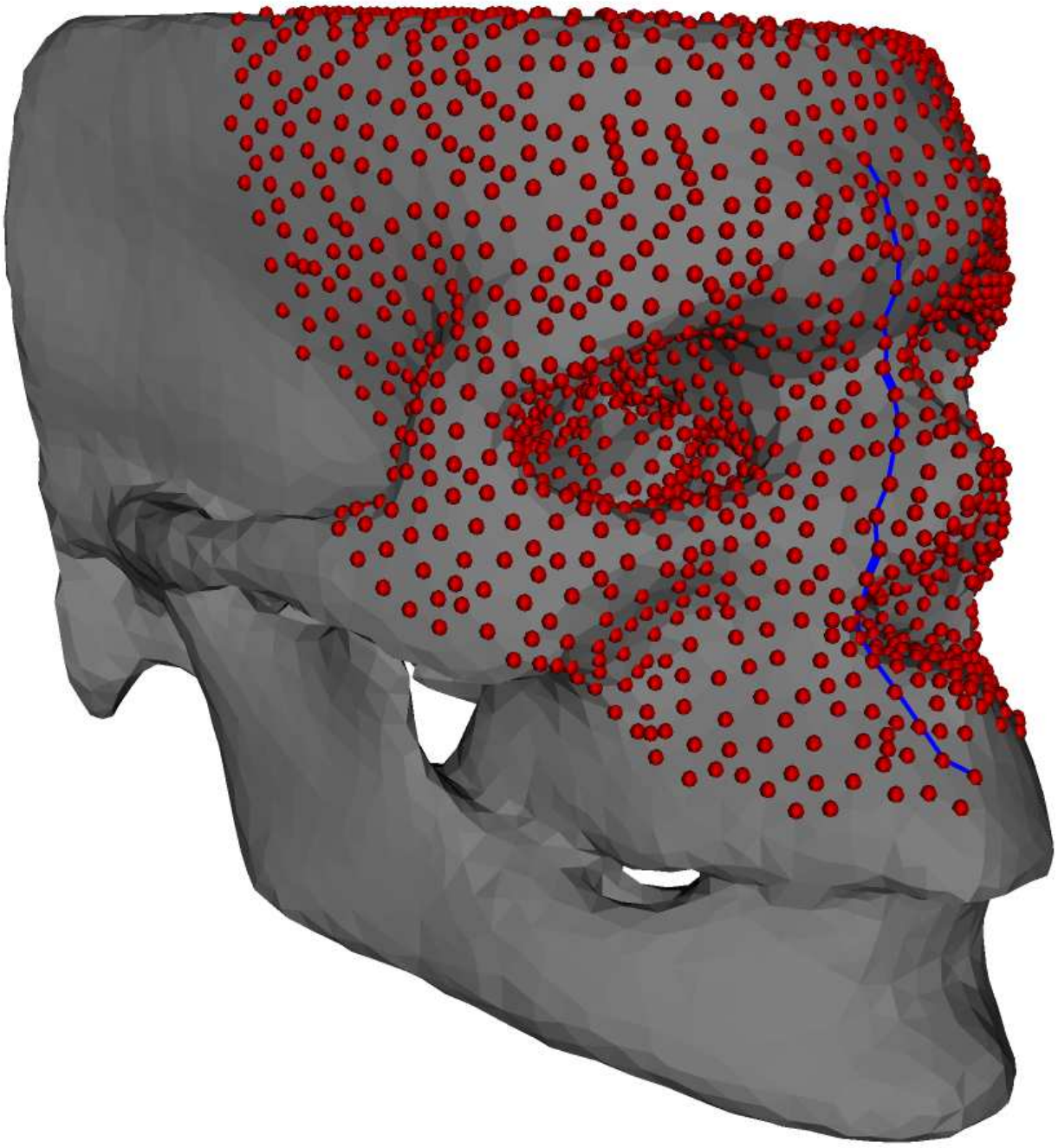

Figure 7: Shortest path between the seed forehead vertex and an arbitrary upper-lip vertex: the vertices explored by the front are displayed in red.

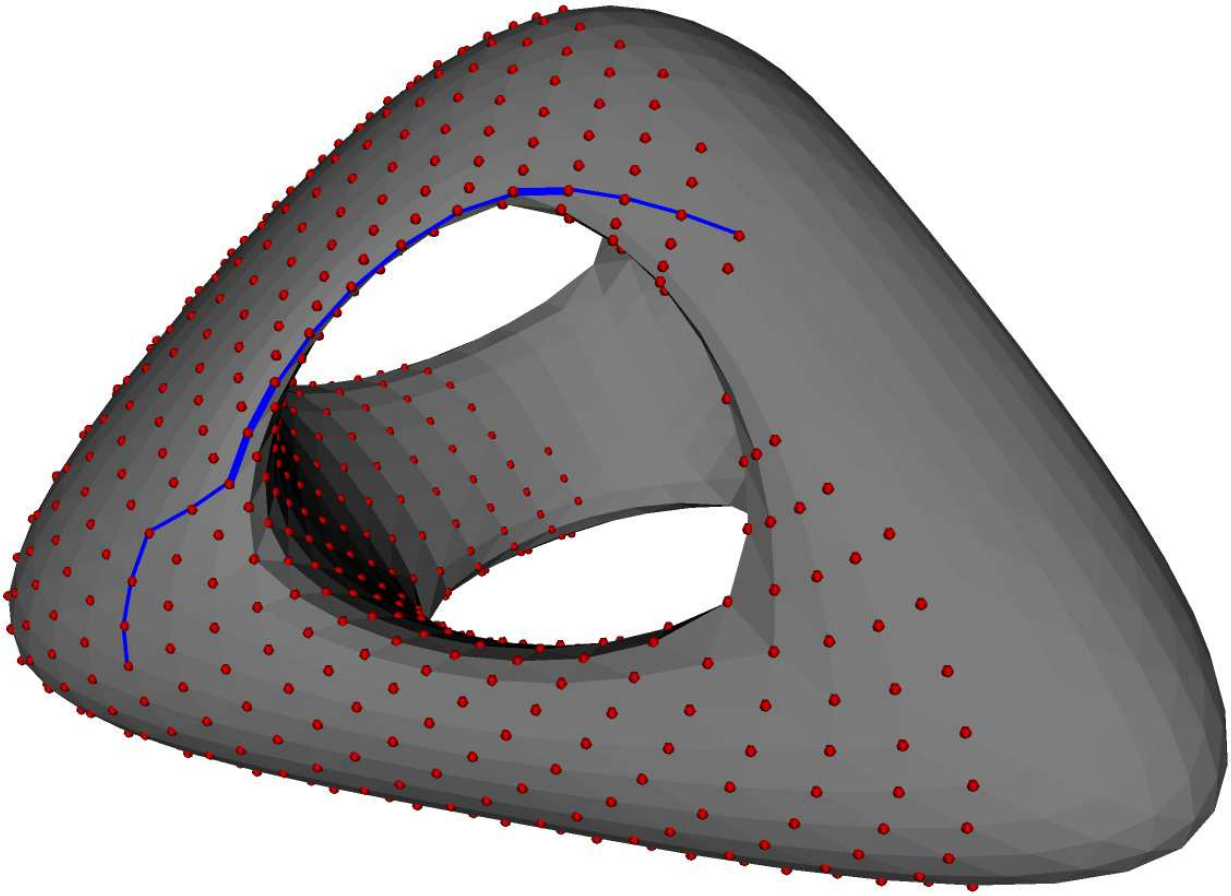

Figure 8: Shortest path between two arbitrary vertices of a genus three surface: the vertices explored by the front are displayed in red.

ITK actually integrates an active surface segmentation algorithm based on simplex meshes. Composed of 9 classes, this is the biggest application based on itk::Mesh in the toolkit. The implementation of this classes is particularly interesting when comparing itk::Mesh and itkQE::Mesh.

Let's first compare the data structures. itkQE::Mesh enforces orientation and benefits from local accesses to neighborhood. The simplex solution define a itk::SimplexMesh which inherits from itk::Mesh as the main representation of simplex mesh. As itk::Mesh has no traversal capacity, local information is gathered in the itk::SimplexMesh class as an array of neighbors, which has to be consistently updated when modifying the connectivity. As connectivity changes are becoming very difficult, a specific filter class, itk::SimplexMeshAdaptTopology, has been created. Also, the orientation not being enforced in itk::Mesh, the itk::TriangleMeshToSimplexMesh filter has to traverse the structure twice, once to create the dual cell, then to enforce orientation. Indeed, orientation of cells of itk::SimplexMesh is supposed consistent in all the filters

A second comparison could be made on duality. itkQE::Mesh handle both primal and dual representation of the mesh, while itk::Mesh does not. Hence, specific transformation filters must be made to transform a triangular mesh into a simplex mesh and vice-versa, namely itk::SimplexMeshToTriangleMeshFilter and itk:TriangularMeshToSimplexMeshFilter.

As a conclusion, at least 4 classes out of 9 are not needed to implement the same solution using itkQE::Mesh data structure as a base. The 5 classes left are the classes computing the internal and external forces whose computation is specific to simplex mesh. They would need to be implemented anyway, but would be smaller. The overall solution would also be faster as illustrated previously.

4 Conclusion and future work

We have implemented in ITK a Quad-Edge data structure to represent orientable 2-manifolds (borders of 3D solid objects). Any application handling orientable 2-manifolds will benefits of great enhancements in speed, robustness, genericity, maintenance cost, for lesser code. Moreover, some algorithms using objects dual to each others (voronoi - delaunay, triangular mesh - simplex mesh) can now be implement in a very elegant and efficient way.

This has a potential great impact on nowadays image processing. Indeed nD Image processing intensively uses explicit discrete surfaces to include a priori geometrical or topological information in the image processing algorithms. Unfortunately, the usage of explicit discrete surfaces was quite raw. Reasons were twofold: first most of the work on discrete surface processing was done in computer graphic and visualization field that did not had to deal with image, and second because the available explicit discrete surfaces data structure in image processing libraries were not adapted to processing. With our implementation of a QE data structure in ITK, we believe that the gap does not exist anymore, and that all surface processing algorithms developed in computer graphics and visualization fields can now directly be used for image processing.

We would like next to further enhance the data structure with Euler operator defined on top of Splice method. That would get us one step closer to C-GAL HE data structure features, without of course the same numerical quality, as no exact numerical kernel nor exact geometrical tests are included in ITK. At the application level, we would like in a close future to add more geometry and topology filters. It would provide ITK with filters it is lacking right now. We would begin by the most useful filters already existing in VTK, to avoid time/memory consuming and pipeline-breaking transitions between ITK and VTK nowadays. We would still need to switch to VTK for visualization, but not for processing anymore.

5 Acknowledgements

The authors would like to thanks Hugues BENOIT-CATTIN for his support throughout the project, and for his illuminating remarks on previous version of this manuscript. Many thanks again to all the developers and scientists behind ITK for bringing up such a useful library for the whole community of image processing.

References

[1] P. Alliez, M. Meyer, and M. Desbrun, Interactive geometry remeshing, *ACM Transaction on Graphics*, proc. of SIGGRAPH 2002, pp. 347-354. 1.2

[2] P.Alliez, D. Cohen-Steiner, O. Devillers, B. Levy, and M. Desbrun. Anisotropic polygonal remeshing, *ACM Transaction on Graphics*, proc. of SIGGRAPH 2003, pp. 485-493. 1.2

[3] S. Bischoff, L. Kobbelt, Teaching meshes, subdivision and multiresolution techniques, *Computer-Aided Design*, 2004, **36:14**, pp. 1483-1500. 1.2

[4] D. Cohen-Steiner and J.-M. Morvan. Restricted delaunay triangulations and normal cycle. *ACM Symposium on Computational Geometry*, 2003, pp. 237-246. 1.2

[5] H. Delingette and H. Watanabe. Simplex mesh modeling, 1993. Slides 11-13 selected at the ACM SIGGRAPH 93 Slides. 1, 1.2

[6] H. Delingette, Adaptive and deformable models based on simplex meshes. In IEEE Workshop of Non-Rigid and Articulated Objects, Austin, Texas, November 1994. 1, 1.2

[7] H. Delingette. General object reconstruction based on simplex meshes. International Journal of Computer Vision, 32(2):111-146, September 1999. 1

[8] M. P. DoCarmo, Differential Geometry of Curves and Surfaces, Prentice Hall College Div., 1976. 1.2

[9] M. Eck, T. DeRose, T. Duchamp, H. Hoppe, M. Lounsbery and W. Stuetzle, Multiresolution Analysis of Arbitrary Meshes, *ACM computer graphics*, proc. SIGGRAPH'95, pp. 173-182, 1995.

[10] L. Florez-Valencia, J. Montagnat, M. Orkisz: 3D graphical models for vascular-stent pose simulation. *Machine Graphics and Vision*, vol. 13, no. 3, 2004, pp. 235-248. 1.2

[11] M. Garland and P. Heckbert, Surface Simplification using quadric error metrics, *ACM Computer Graphics*, proc. of SIGGRAPH 1996, pp. 209-216. 1.2

[12] A. Gouaillard, H. Kawata, T. Kanai, C. Odet, X. Gu, Optimal Localization of Topological Artifacts on 3D Meshes, *11th International Conference on Geometry and Graphics*, 1-5 August, 2004, Guangzhou, China, pp. 14-20. 1.2

[13] A. Gouaillard, A. Gelas, S. Valette, E. Boix and R. Prost, Remeshing algorithm for multiresolution prior model in segmentation, *IEEE Int. Conf. on Image Processing ICIP 2004*, Singapore, October 24-27, 2004, Volume 4, pp. 2753-2756. 1.2

[14] A. Gouaillard, A. Gelas, S. Valette, E. Boix and R. Prost, Curvature-based Adaptive Remeshing for Wavelet-Based Multiresolution 3D Meshes, *IEEE Int. Conf. on Image Processing ICIP 2005*, Genova, Italy, September 11-14, 2005, in press. 1.2

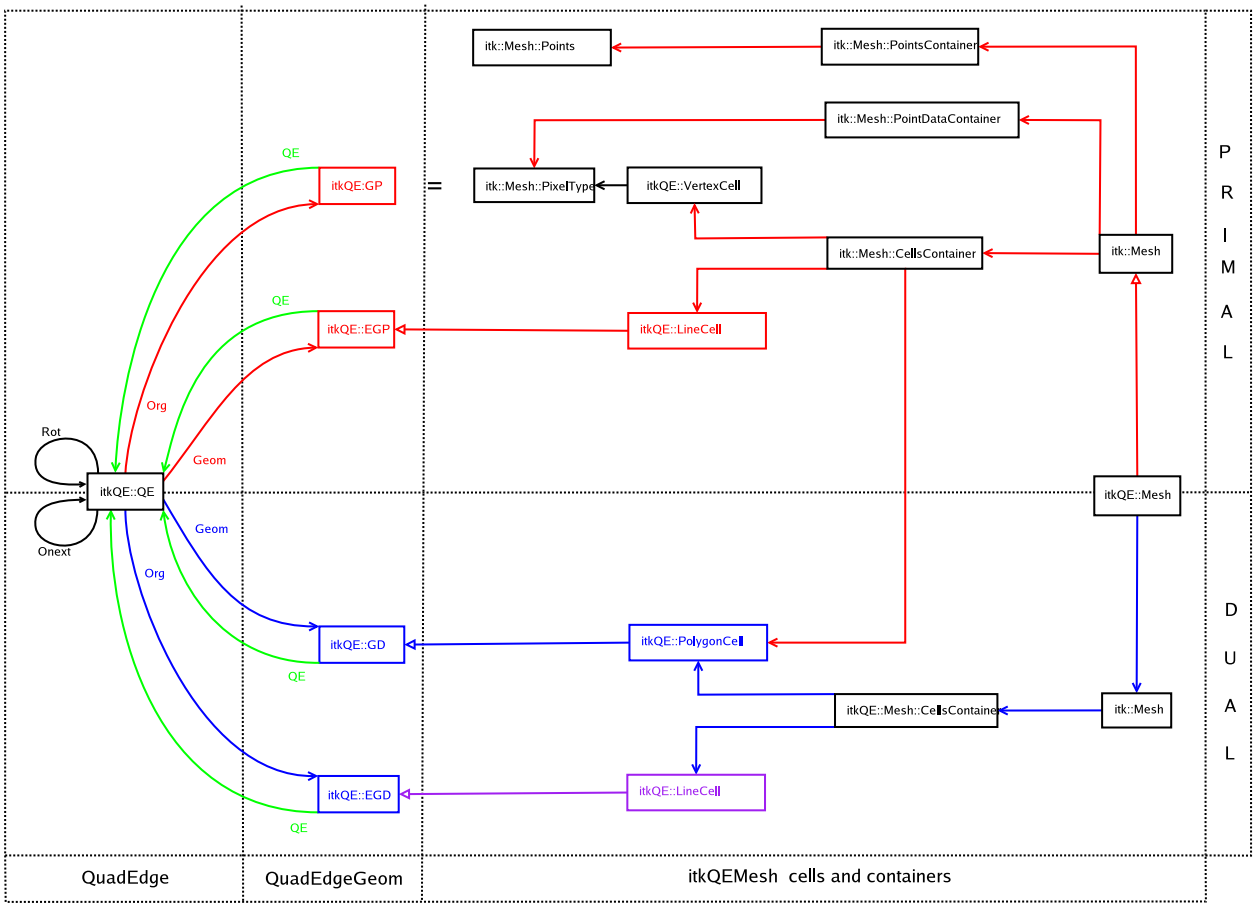

Figure 9: itkQE Global design. This drawing is an overview of the global design. On the left the topological layer defines QEs and the two associated links *Rot* and *Onext*. This layer being purely topological no distinction is made between primal and dual QEs. Next column on the right represents the geometrical layer. Here appears the *Org* and *Data* informations that were introduced in figure 4. These two fields are being differentiated depending on the type of the QE (primal or dual). Finally, the last column on the right shows the integration of the QEs in ITK through new classes: itkQE::PolygonCell, itkQE::LineCell, itkQE::VertexCell. Normal arrows illustrate link, arrows with an empty triangle on target represent inheritance relations.

[15] X. Gu, S.-T. Yau, Global Conformal Surface Parameterization. *Eurographics / ACM Symposium on Geometry Processing*, 2003, pp. 127-137. 1.2

[16] L. Guibas, J. Stolfi, Primitives for the manipulation of General Subdivisions and the Computation of Voronoi Diagrams, *ACM Transaction on Graphics* **4:2** April 1985, 74–123. 1.2, 2.1, 2.1, 4, 2.4, 2.5, 3.2

[17] H. Hoppe, Progressive meshes, *ACM Computer Graphics*, proc. of SIGGRAPH 1996, pp. 99-108. 1.2

[18] L. Kettner. Using generic programming for designing a data structure for polyhedral surfaces. *Comput. Geom. Theory Appl.*, 1999, **13**, pp. 65-90. 1, 2, 1.2

[19] M. Lounsberry, T. Derose, J. Warren, *Multiresolution Analysis for surface of Arbitrary Topological Type* ACM Transactions on Graphics, Vol. 16, No. 1, pages 34-73, 1997. 1.2

[20] W. Schroeder and B. Yamrom, A compact cell structure for scientific visualization *ACM Computer Graphics*, SIGGRAPH Course on Multiresolution Modeling, 1997. 1.1, 2, 2.2

[21] J. Stillwell, Classical Topology and Combinatorial Group Theory, 2nd ed. New York: Springer-Verlag, 1993. 1.2

[22] G. Taubin, A signal processing approach to fair surface design. *ACM Computer Graphics*, proc. of SIGGRAPH 1995, pp. 351-358. 1.2

[23] J.D. Warren and H. Weimer. *Subdivision Methods for Geometric Design*, Morgan Kaufman ed., 2001. 1.2

[24] Z. Wood, H. Hoppe, M. Desbrun, P. Schrder, Removing excess topology from isosurfaces. *ACM Transactions on Graphics*, **23:2**, April 2004, pp. 190-208. 1.2

A UML DIAGRAMS

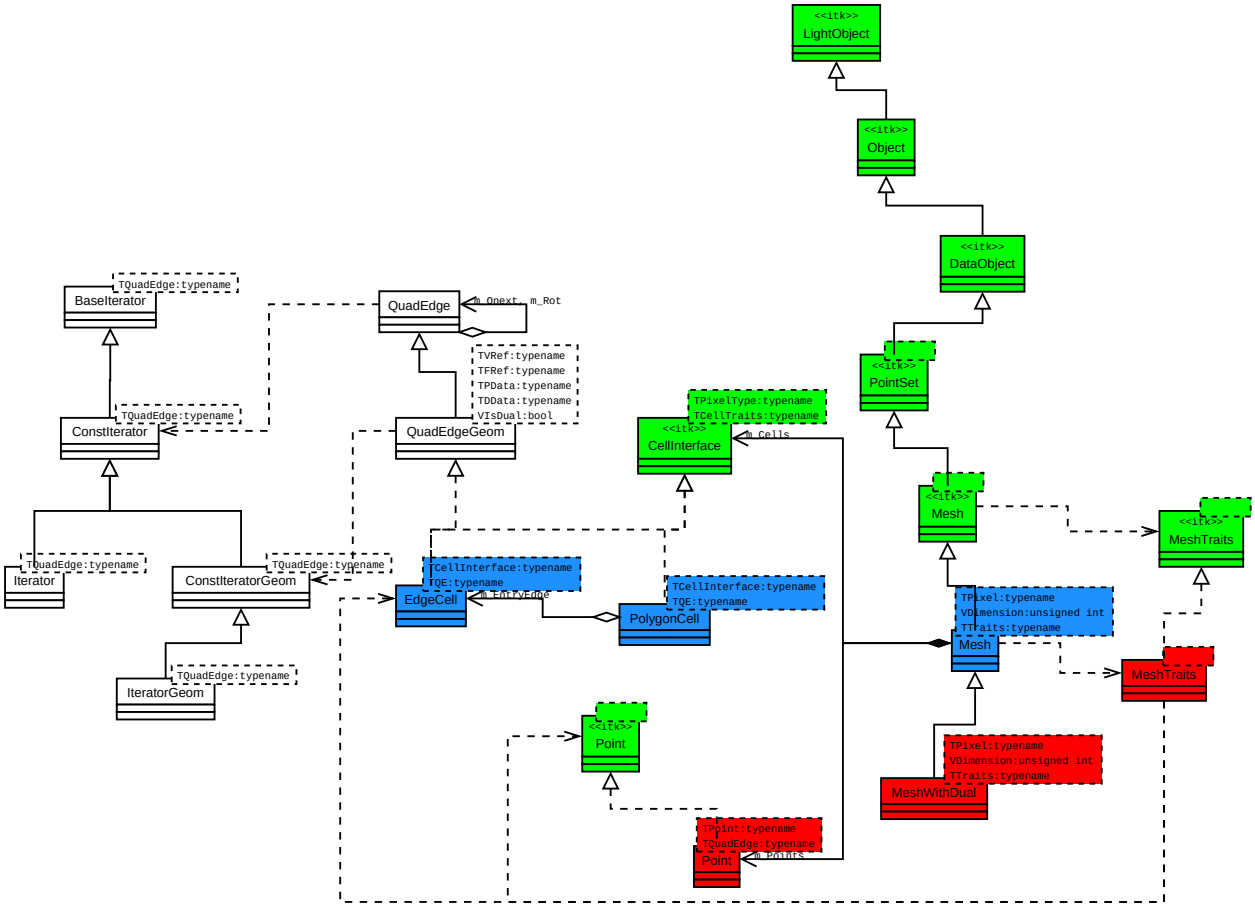

Figure 10: UML Diagram of itkQE::Mesh and its interaction with ITK.