bits) so that a neighbor query not only returns the neighbor triangle, but returns in which of three orders it is held.

There are many closely related data structures based on edges, including the doubly connected edge list,[28] winged-edge,[29] half-edge,[30] and quad-edge[31] structures. In addition to triangulated meshes, these data structures can all be used for polygonal meshes. In these data structures each edge maintains pointers to its two neighboring vertices and to neighboring edges cyclically around the neighboring faces and vertices. Each edge might also maintain pointers to the neighboring faces and to edge data. The most space efficient of these data structures can maintain for each edge a pointer to the two neighboring vertices and to just two neighboring edges, one around each face and vertex. Assuming no data needs to be stored on a face or edge, this requires 4 pointers per edge, which for a manifold triangulation is equivalent to the 6 pointers per triangle used by the triangle structure ($|E| = 3/2|T|$). The half-edge data structure,[30] used by CGAL,[26] LEDA[32] and HGAM,[33] maintains two structures per edge, one in each direction. These half-edges are cross referenced, requiring an extra two pointers per edge. The winged-edge and quad-edge structures maintain pointers to all four neighboring edges, requiring 6 pointers per edge (9 per triangle).

In three dimensions there are analogous data structures based either on tetrahedra or on faces and edges. Again the simplest data structure is to use a structure per tetrahedron. Each tetrahedron has 4 pointers to adjacent tetrahedra, and 4 to its corner vertices. Assuming no data this requires 8 pointers per tetrahedron. This data structure is used by Pyramid[25] and CGAL.[27] The face and edge data structures are often called boundary representations (b-reps). Such boundary representations are more general than the tetrahedron data structures, allowing the representation of polytope meshes, but tend to take significantly more space. Dobkin and Laszlo[34] suggest a data structure based on edge-face pairs, which in general requires 6 pointers per edge-face. For tetrahedral meshes this data structure can be optimized to 9 pointers per face (6 to the adjacent faces rotating around its 3 edges, and 3 to the corner vertices). This corresponds to 18 pointers per tetrahedron. Weiler's radial-edge representation,[35] Brisson's cell-tuple representation,[36] and Linehard's G-map representation[37] all take more space.

In summary, the most efficient standard data structures of simplicial meshes use 6 pointers per triangle in 2D and 8 pointers per tetrahedron in 3D. At least one extra pointer is required to store data on triangles in 2D or tetrahedra in 3D.

3. Preliminaries

In this section we review some basic notions of combinatorial topology used in this paper. For a more detailed discussion the reader can refer to Ref. [38] and Ref.[39] among others.

An *(abstract) simplicial complex* K is a collection of finite sets which is closed under taking subsets. The elements of K are called *simplices*. The underlying set