Quad-Edge Data Structures in Two and Three Dimensions

T. H. Merrett*
McGill University, Montreal, Canada

April 13, 2005

Because Clifford algebra abstracts beyond absolute position, its elements cannot be directly drawn. They need to be supplemented by a coordinate system. Because numbers, both in such a coordinate system and in the coefficients for the Clifford elements are subject to round-off or truncation errors in the computer, both coordinates and Clifford elements also need complementary topological information. Coordinates and topology make up a data structure representing spatial data, supplemented by Clifford algebra to quantify angles. We start in two dimensions with the *quad-edge* data structure [GS85, MBC+02].

1 Two dimensions

The quad-edge structure exploits the duality of k-dimensional components in d-dimensional space ($0 \le k \le d$): each k-dimensional component has a dual $(k-1)$-dimensional component, as we see in the following table.

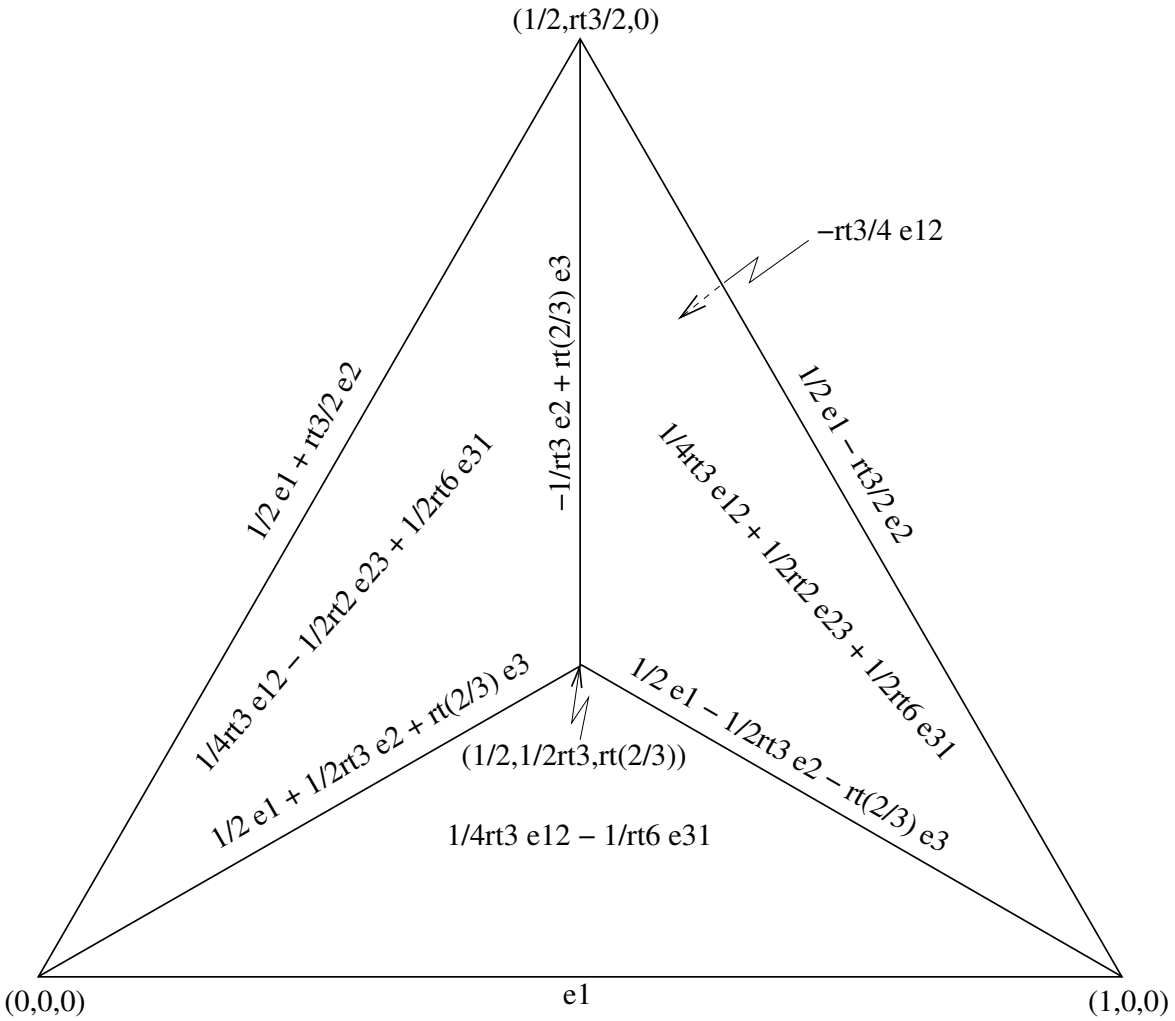
	v	e	f
# components	1	2	1
hasa_1	$\overset{2}{\longleftarrow}$	$\overset{c}{\longleftarrow}$	
hasa_2	$\overset{c}{\longrightarrow}$	$\overset{2}{\longrightarrow}$	
Euler	$v - e + f = 2$		

In this table, v stands for vertices, e for edges, f for faces, and c for cycles. The first row gives the number of independent Clifford components in two dimensions: one "point" or scalar; two edge projections, **e1** and **e2**; and one "face", **e12**.

The "hasa" rows say how many of each type of component can be adjacent to each other type in a polygon: hasa_1 says that any edge has 2 vertices and any face has a cycle of edges; hasa_2 says that any vertex has a cycle of edges and any edge has 2 adjacent faces.

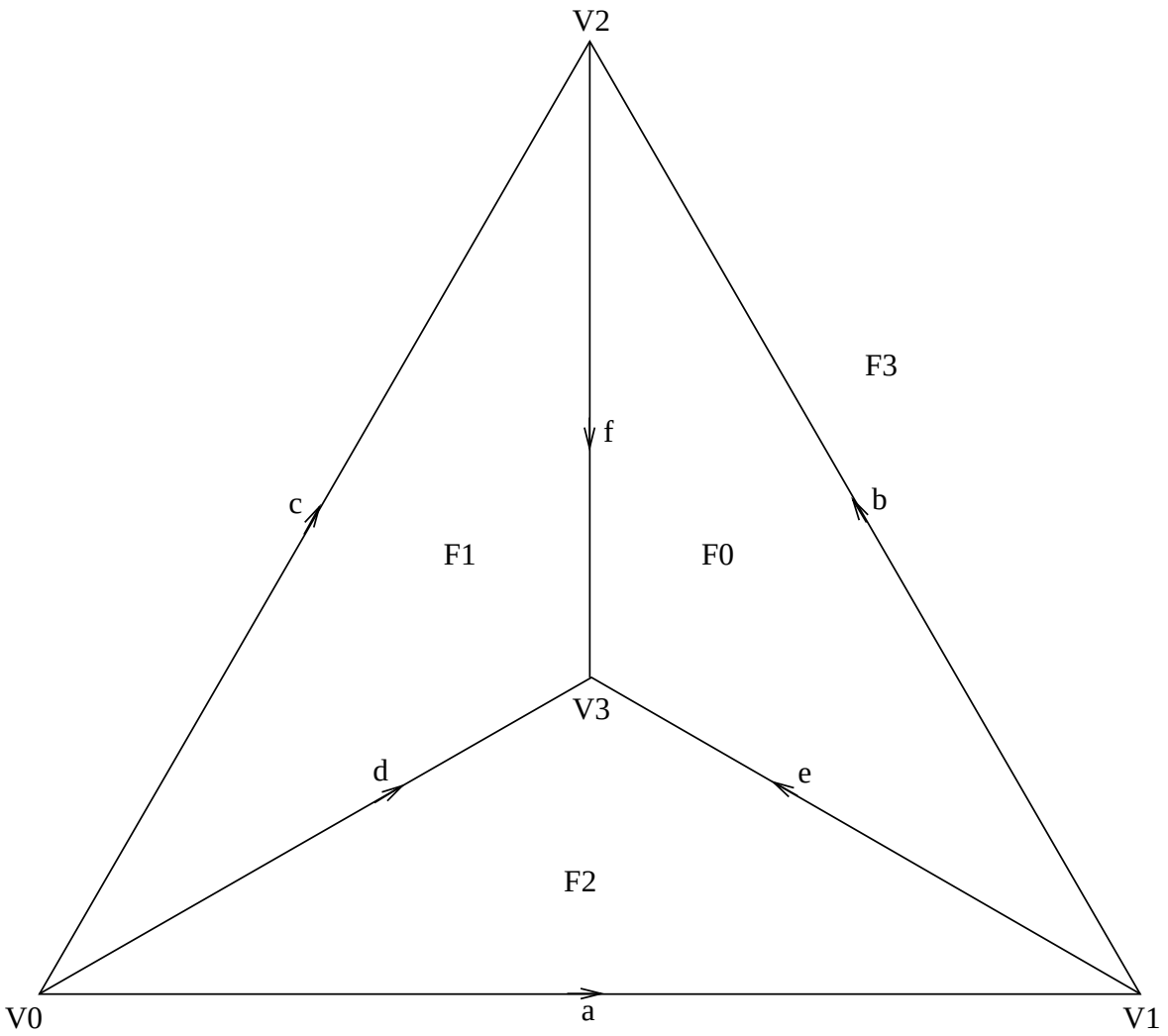
The final row is Euler's formula relating vertices, edges and faces in any network of polygons.

Check the hasa and Euler relationships in the tetrahedron:



You may wonder why we are using the 3-D tetrahedron to illustrate two dimensions. In fact, the topological rules we have just given apply to any network of polygons on a plane, supposing that we always count an external face (containing the point at infinity), or on a sphere. Thus the tetrahedron can also be seen as the following construct, with F3 labelling the external face instead of the face underneath the 3-D tetrahedron.

(Note that the Euler formula does not always work if the embedding space is a torus or a space with a higher number of "holes". The hasa relationships still hold in such a space.)



The quad-edge data structure results from combining hasa_1 with the dual hasa_2: if we ask what vertices or faces an edge has, the answer is 2 each; if we ask what edges a vertex or face has, we get cycles in both cases. The two pairs from the first question give the name of the data structure: each edge appears four times, for its start and end vertices and for its left and right faces. The cycles from the second question give the data structure its one operation for modifying polygon networks, as we shall see in section 1.1. (It is clear that geographical maps are essentially polygon networks, so that the quad-edge data structure will be ideal for geographical information systems.) What stops us from going completely to three dimensions with the ideas so far is that they cannot handle a network of *polyhedra*: for example, we cannot so far describe two tetrahedra stuck together at one face.

Here is the relation giving the quad-edge representation of the tetrahedron. It is grouped by the cycles, although it could equally well be grouped by the pairs. (For clarity, vertices and faces are not shown where they are repeated in their column.)

QE(*vf*	*cycle*	*edge*	*pairs*)
V0	1	d	0
	2	c	0
	3	a	0
V1	1	e	0
	2	a	2
	3	b	0
V2	1	f	0
	2	b	2
	3	c	2
V3	1	d	2
	2	e	2
	3	f	2

(*vf*	*cycle*	*edge*	*pairs*)
F0	1	b	3
	2	f	3
	3	e	1
F1	1	c	1
	2	f	1
	3	d	3
F2	1	a	3
	2	e	3
	3	d	1
F3	1	a	1
	2	c	3
	3	b	1

Note that the *pairs* attribute takes on four different values, even for vertices and odd for faces. We could have given it just two values, 0 and 1, except that we have combined vertices and faces into a single attribute, *vf*. These four values, apart from giving "quad-edge" its name, also have the significance of *directions* of the edges: 0 is a start vertex, 2 an end vertex, 1 a right face and 3 a left face. The cycles are also directed, counterclockwise, but note that the back (or external) face, F3, is counterclockwise only if seen from below.

Relation *QE* gives only topological information about all the connections among vertices, edges and faces. For metric information, we add the coordinates. Because our example is a tetrahedron, we'll use 3-D coordinates, but note that 2-D coordinates would usually be used for a network of polygons in the plane. Coordinates are only for vertices: can we add any metrical information for edges or faces? Yes, this is where Clifford algebra fits in. Each edge and face has a quantitative expression giving its orientation. In this 3-D case, but in 3-D only, we can take advantage of the fact that both edges and faces have three components each (see section 2) and show all metric information as triples.

Metric(*vef*	*x_yz*	*y_zx*	*z_xy*)
V0	0	0	0
V1	1	0	0
V2	1/2	$\sqrt{3}/2$	0
V3	1/2	$1/2\sqrt{3}$	$\sqrt{3/2}$
a	1	0	0
b	1/2	$-\sqrt{3}/2$	0
c	1/2	$\sqrt{3}/2$	0

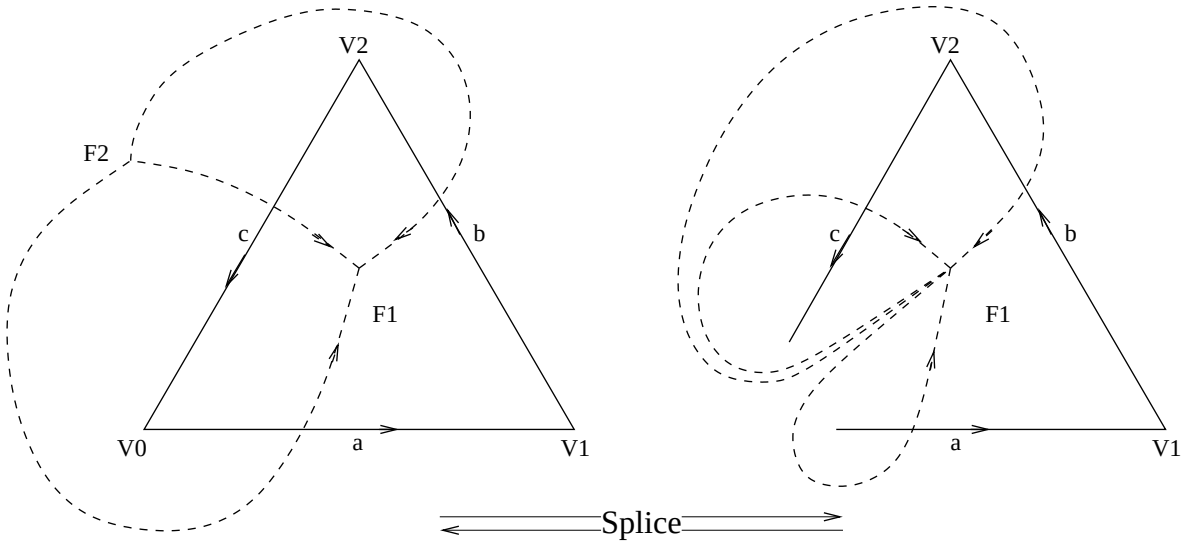
(*vef*	*x_yz*	*y_zx*	*z_xy*)
d	1/2	$1/2\sqrt{3}$	$\sqrt{3/2}$
e	1/2	$-1/2\sqrt{3}$	$-\sqrt{3/2}$
f	0	$-1/\sqrt{3}$	$\sqrt{3/2}$
F0	$1/2\sqrt{2}$	$1/2\sqrt{6}$	$1/4\sqrt{3}$
F1	$-1/2\sqrt{2}$	$1/2\sqrt{6}$	$1/4\sqrt{3}$
F2	0	$-1/\sqrt{6}$	$1/4\sqrt{3}$
F3	0	0	$-\sqrt{3}/4$

Here, *x_yz* is the attribute that gives the x-coordinate for vertices, the **e1** coefficient for edges, and the **e23** coefficient for faces. There are similar interpretations for *y_zx* and *z_xy*.

1.1 Splice

In the 2-D topology (whether it is accompanied by 2-D or 3-D (or 4-D, ..) metric information) a single remarkable operator provides the ability to modify the structures at will. This is a swapping operation called *splice* which both disconnects edges from vertices and connects them to vertices. (In fact, since it is a swap, it is its own inverse.)

We can practice splice on a simpler example, the triangle shown on the left.



To see what happens, it is easier to look at the "element pair" representation of the cycles than at the "enumerated sequence" representation we used for the tetrahedron. Here are both for the triangle (on the left).

QE(*vf*	*cycle*	*edge*	*pairs*)
V0	1	a	0
V0	2	c	2
V1	1	a	2
V1	2	b	0
V2	1	b	2
V2	2	c	0
F1	1	a	3
F1	2	b	3
F1	3	c	3
F2	1	c	1
F2	2	b	1
F2	3	a	1

Enumerated Sequences

Cycles(*edge1*	*pair1*	*edge2*	*pair2*)
a	0	c	2
c	2	a	0
a	2	b	0
b	0	a	2
b	2	c	0
c	0	b	2
a	3	b	3
b	3	c	3
c	3	a	3
c	1	b	1
b	1	a	1
a	1	c	1

Element Pairs

Splice does two things:

$$splice((c,p),(c',p')) \textbf{ is}$$
$$\{\ swap1)((c,p),(c',p'))\ ||\ \text{swap2})((c,p-1),(c',p'-1))\ \}$$

where the subtraction is done modulo 4. Swap1 exchanges the named (cycle,pair) values in the *cycle1, pair1* attributes, and swap2 exchanges the named (cycle,pair) values in the *cycle2, pair2* attributes.

The splice to disconnect edge `c` from its vertex with `a` is *splice*((`a`,0),(`c`,2)). The two swaps, which may be done in any order, are shown below as if *swap1* were done before *swap2*.

Cycles(*cycle1*	*pair1*	*cycle2*	*pair2*)
c	2	c	2
a	0	a	0
a	2	b	0
b	0	a	2
b	2	c	0
c	0	b	2
a	3	b	3
b	3	c	3
c	3	a	3
c	1	b	1
b	1	a	1
a	1	c	1

After *swap1*(a,0),(c,2)

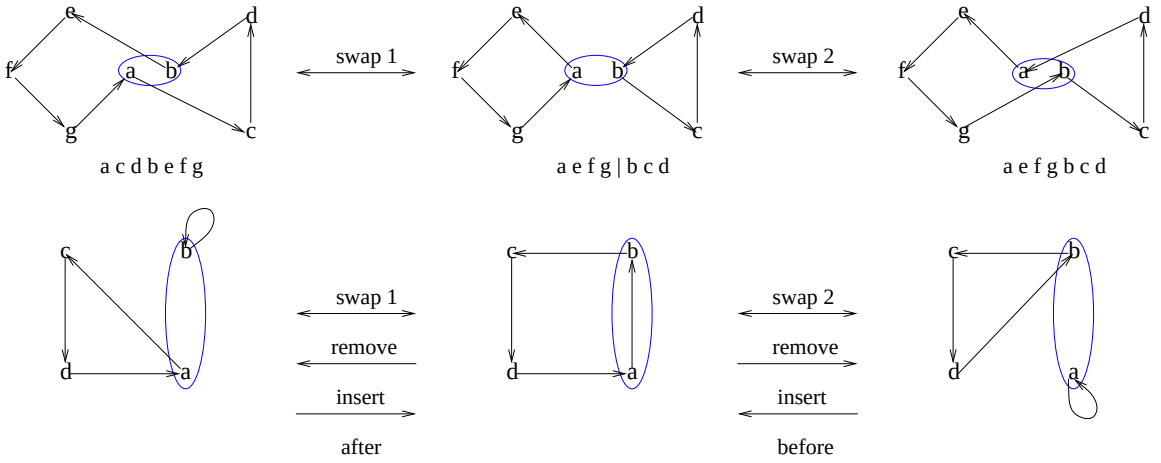
Cycles(*cycle1*	*pair1*	*cycle2*	*pair2*)
c	2	c	2
a	0	a	0
a	2	b	0
b	0	a	2
b	2	c	0
c	0	b	2
a	3	b	3
b	3	c	3
c	3	c	1
c	1	b	1
b	1	a	1
a	1	a	3

After *swap2*(a,3),(c,1)

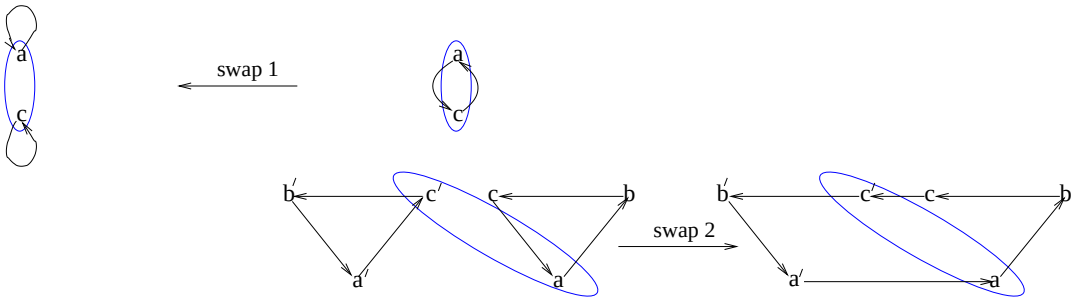
This gives us the opened triangle on the right of the above figure: notice (a) that (a,0) and (c,2) now each form their own cycle, so that vertex V0 has been replaced by two vertices (not named); and (b) that the two cycles for faces F1 and F2 have now merged into a single cycle, (a,3), (b,3), (c,3), (c,1), (b,1), a,1), so there is now only one face. The result is to open the triangle at the a–c vertex, replacing it by two vertices, and to fuse the two faces that were formerly separated by the closed triangle.

Obviously, performing *splice*((a,0),(c,2)) again will close the opened triangle and restore the original diagram.

To render this a little less magical, let's look at some cycles and see what happens to them when we swap *source* nodes (*swap1*) and *target* nodes (*swap2*).



Splicing the triangle

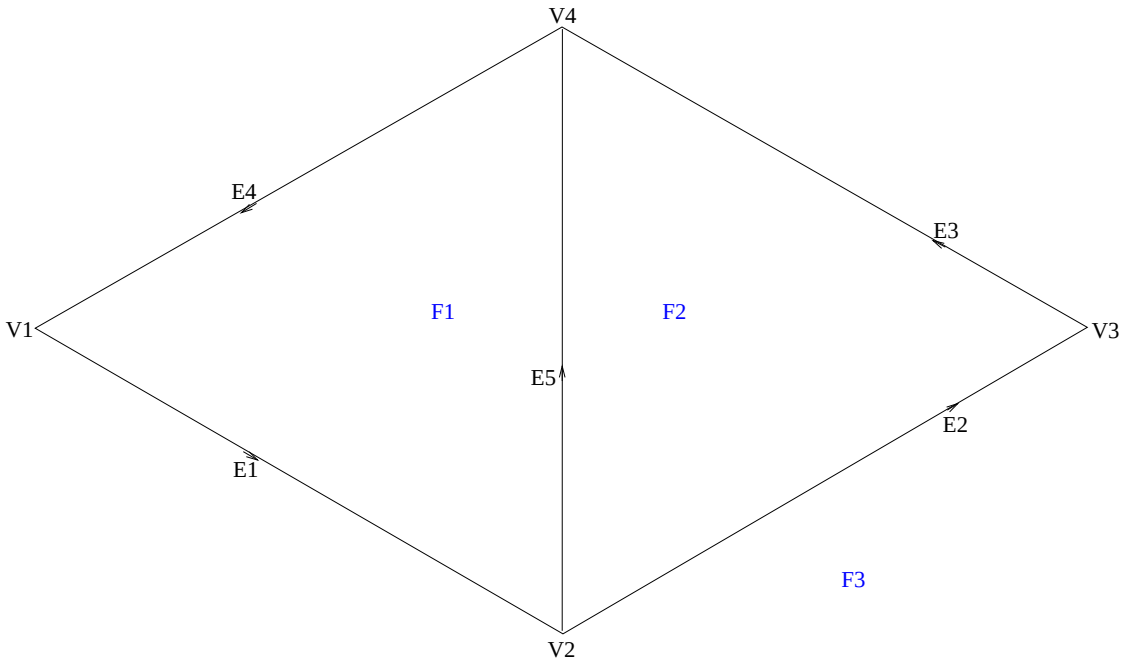


The figure has three parts. The top shows how *swap1* and *swap2* transform two cycles, aefg and bcd, into a single cycle (and vice-versa) in two ways. The middle part shows what happens when one of the cycles is a single item: *swap1* inserts the item (b) *after* the other item (a) in the

swap, in the sense of the cycle `a` is in; *swap2* inserts the item (`b`) *before* the other item (`b`) in the swap, in the sense of the cycle `b` is in. The inverse removes items: *swap1* removes the second of the two items, in the sense of the cycle they are in, while *swap2* removes the first.

The bottom part of the figure shows the two swaps that transformed the triangle, above, with triangle edges shown as vertices of the same names. (The "items" in the sequence are always edges in the 2-D quad-edge representation.) *Swap1* separates the two edges of the 2-cycle corresponding to vertex `V0`, while *swap2* combines the two cycles of edges corresponding to the faces `F1` and `F2` (the primed edges give the `F2` cycle: note that the order is reversed from the order in the `F1` cycle). Compare the results with the opened triangle shown earlier.

An example with two adjacent triangles shows us the repertoire of operations we can achieve with this single splice operation.

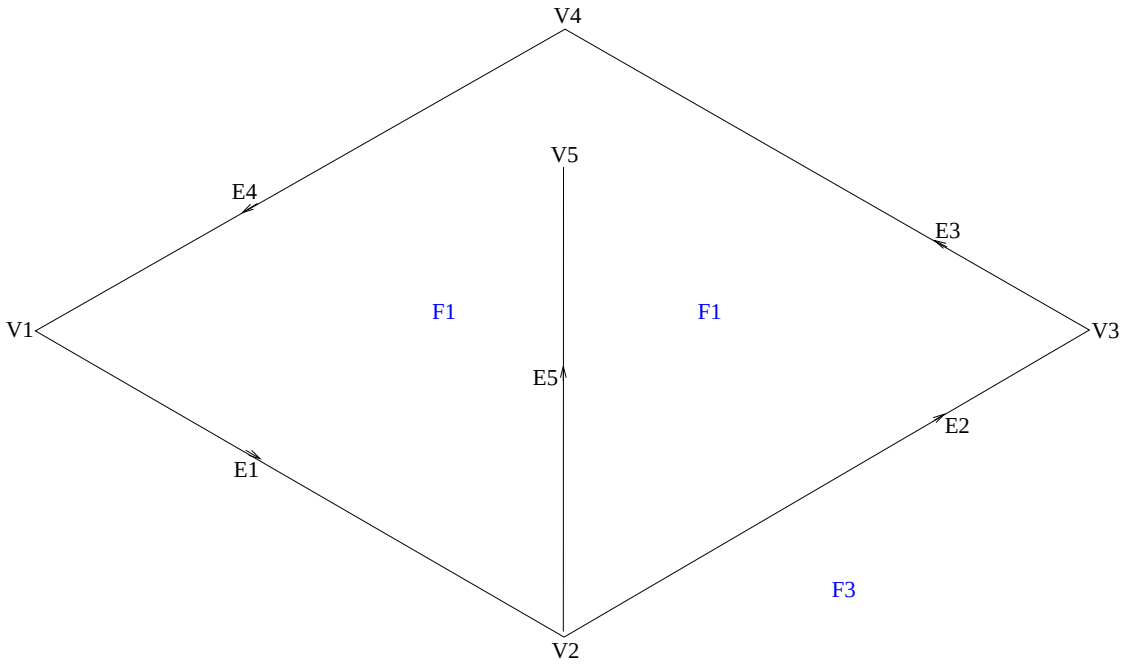


QE(*vf*	*cycle*	*edge*	*pairs*)
V1	1	E1	0
V1	2	E4	2
V2	1	E2	0
V2	2	E5	0
V2	3	E1	2
V3	1	E3	0
V3	2	E2	2
V4	1	E4	0
V4	2	E5	2
V4	3	E3	2
F1	1	E1	3
F1	2	E5	3
F1	3	E4	3
F2	1	E2	3
F2	2	E3	3
F2	3	E5	1
F3	1	E4	1
F3	2	E3	1
F3	3	E2	1
F3	4	E1	1

Enumerated Sequences

EP(*edge1*	*pairs1*	*edge2*	*pairs2*)
E1	0	E4	2
E2	2	E1	0
E2	0	E5	0
E5	0	E1	2
E1	2	E2	0
E3	0	E2	2
E2	2	E3	0
E4	0	E5	2
E5	2	E3	2
E3	2	E4	2
E1	3	E5	3
E5	3	E4	3
E4	3	E1	3
E2	3	E3	3
E3	3	E5	1
E5	1	E2	3
E4	1	E3	1
E3	1	E2	1
E2	1	E1	1
E1	1	E4	1

Element Pairs



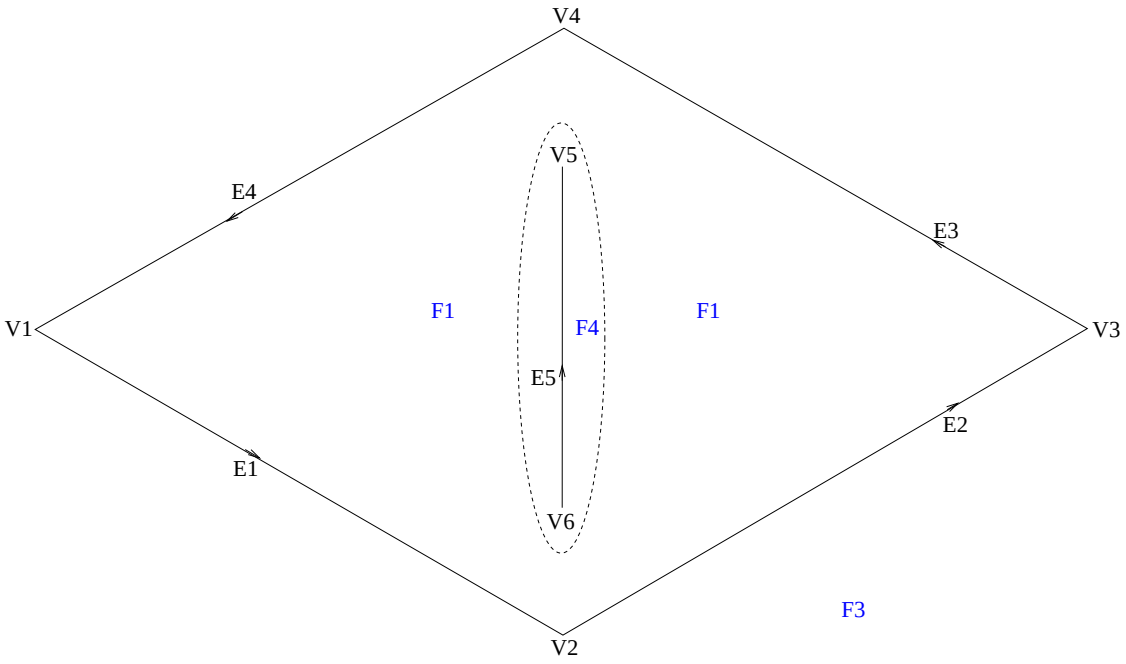
Remove edge E5 from vertex V4, i.e., after E4: splice((E4,0),(E5,2)) = swap1((E4,0),(E5,2)), swap2((E4,3),(E5,1)).

QE(*vf*	*cycle*	*edge*	*pairs*)
V1	1	E1	0
V1	2	E4	2
V2	1	E2	0
V2	2	E5	0
V2	3	E1	2
V3	1	E3	0
V3	2	E2	2
V4	1	E4	0
V5	1	E5	2
V4	3	E3	2
F1	1	E1	3
F1	2	E5	3
F1	6	E4	3
F1	4	E2	3
F1	5	E3	3
F1	3	E5	1
F3	1	E4	1
F3	2	E3	1
F3	3	E2	1
F3	4	E1	1

Enumerated Sequences

EP(*edge1*	*pairs1*	*edge2*	*pairs2*)
E1	0	E4	2
E2	2	E1	0
E2	0	E5	0
E5	0	E1	2
E1	2	E2	0
E3	0	E2	2
E2	2	E3	0
E5	2	E5	2
E4	0	E3	2
E3	2	E4	2
E1	3	E5	3
E5	3	E5	1
E4	3	E1	3
E2	3	E3	3
E3	3	E4	3
E5	1	E2	3
E4	1	E3	1
E3	1	E2	1
E2	1	E1	1
E1	1	E4	1

Element Pairs



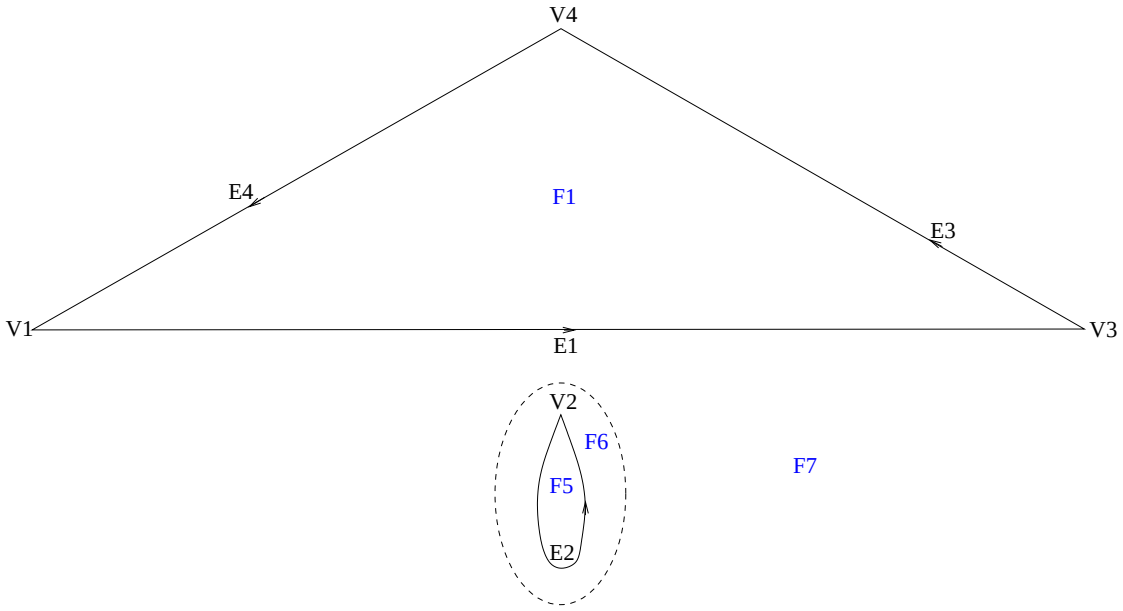
Remove edge E5 from vertex V2, i.e., after E2: splice((E2,0),(E5,0)) = swap1((E2,0),(E5,0)), swap2((E2,3),(E5,3)).

QE(*vf*	*cycle*	*edge*	*pairs*)
V1	1	E1	0
V1	2	E4	2
V2	1	E2	0
V6	1	E5	0
V2	2	E1	2
V3	1	E3	0
V3	2	E2	2
V4	1	E4	0
V5	1	E5	2
V4	3	E3	2
F1	1	E1	3
F4	1	E5	3
F1	4	E4	3
F1	2	E2	3
F1	3	E3	3
F4	2	E5	1
F3	1	E4	1
F3	2	E3	1
F3	3	E2	1
F3	4	E1	1

Enumerated Sequences

EP(*edge1*	*pairs1*	*edge2*	*pairs2*)
E1	0	E4	2
E2	2	E1	0
E5	0	E5	0
E2	0	E1	2
E1	2	E2	0
E3	0	E2	2
E2	2	E3	0
E5	2	E5	2
E4	0	E3	2
E3	2	E4	2
E1	3	E2	3
E5	3	E5	1
E4	3	E1	3
E2	3	E3	3
E3	3	E4	3
E5	1	E5	3
E4	1	E3	1
E3	1	E2	1
E2	1	E1	1
E1	1	E4	1

Element Pairs



Disconnect E2 from V3 (lose F3): splice((E3,0),(E2,2)) = swap1((E3,0),(E2,2)), swap2((E3,3),(E2,1)).
Connect E2 to V2 (create F5): splice((E3,0),(E2,2)) = swap1((E2,0),(E2,2)), swap2((E2,3),(E2,1)).
Disconnect E1 from V2 (create F6): splice((E2,0),(E1,2)) = swap1((E2,0),(E1,2)), swap2((E2,3),(E1,1)).
Connect E1 to V3 (create F7—was F3): splice((E3,0),(E1,2)) = swap1((E3,0),(E1,2)), swap2((E3,3),(E1,1)).

QE(*vf*	*cycle*	*edge*	*pairs*)
V1	1	E1	0
V1	2	E4	2
V2	1	E2	0
V6	1	E5	0
V3	2	E1	2
V3	1	E3	0
V2	2	E2	2
V4	1	E4	0
V5	1	E5	2
V4	3	E3	2
F1	1	E1	3
F4	1	E5	3
F1	3	E4	3
F5	1	E2	3
F1	2	E3	3
F4	2	E5	1
F7	2	E4	1
F7	3	E3	1
F6	1	E2	1
F7	1	E1	1

Enumerated Sequences

EP(*edge1*	*pairs1*	*edge2*	*pairs2*)
E1	0	E4	2
E3	0	E1	0
E5	0	E5	0
E2	2	E1	2
E2	0	E2	0
E2	2	E2	2
E1	2	E3	0
E5	2	E5	2
E4	0	E3	2
E3	2	E4	2
E1	3	E2	1
E5	3	E5	1
E4	3	E1	3
E2	3	E3	3
E3	3	E4	3
E5	1	E5	3
E4	1	E3	1
E3	1	E1	1
E2	1	E2	3
E1	1	E4	1

Element Pairs

Note that this last step can work only if the vertex removed, V2, has an edge cycle of exactly two. The trouble will arise fourth substep, trying to connect E1 to V3/

While the splice operation is elegantly implemented on element pairs—one pair of (*edge1, pairs1*) is swapped, and one pair of (*edge2, pairs2*)—it has two drawbacks. First, we must re-apply the names of vertices and faces, which were lost when we went from enumerated sequences to element pairs. Second, while it is easy to generate element pairs from enumerated sequences, the reverse process is much harder (but possible: try to figure it out!). (Third, when we go to three dimensions we will find that we apparently cannot get away with just one, self-invertible operation.) It would

be better, if less elegant, to find a way to work directly in the enumerated sequence representation. Here is the first step, above, which turns the double triangles into a diamond.

QE(*vf*	*cycle*	*edge*	*pairs*)
V1	1	E1	0
V1	2	E4	2
V2	1	E2	0
V2	2	E5	0
V2	3	E1	2
V3	1	E3	0
V3	2	E2	2
V4	1	E4	0
V4	2	E5	2
V4	3	E3	2
F1	1	E1	3
F1	2	E5	3
F1	3	E4	3
F2	1	E2	3
F2	2	E3	3
F2	3	E5	1
F3	1	E4	1
F3	2	E3	1
F3	3	E2	1
F3	4	E1	1

Enumerated Sequences

QE(*vf*	*cycle*	*edge*	*pairs*)
V1	1	E1	0
V1	2	E4	2
V2	1	E2	0
V2	2	E5	0
V2	3	E1	2
V3	1	E3	0
V3	2	E2	2
V4	1	E4	0
V5	1	E5	2
V4	3	E3	2
F1	1	E1	3
F1	2	E5	3
F1	6	E4	3
F1	4	E2	3
F1	5	E3	3
F1	3	E5	1
F3	1	E4	1
F3	2	E3	1
F3	3	E2	1
F3	4	E1	1

Enumerated Sequences

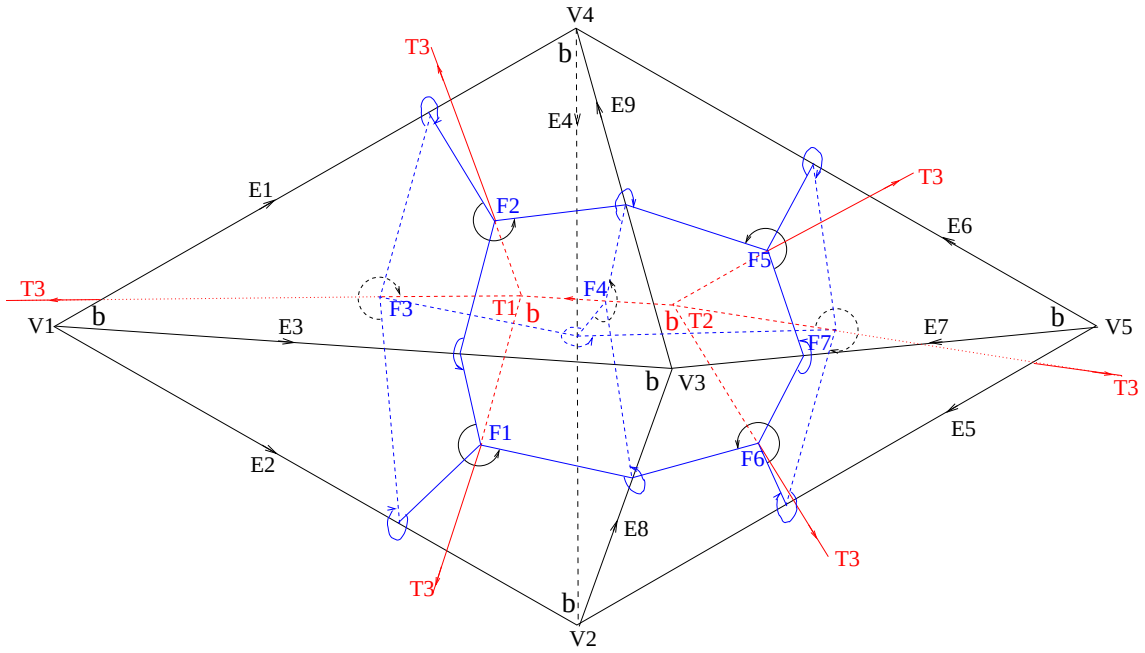
The complicated part of calculating this directly will be to open up the sequences for faces F1 and F2 at the right points and link them together by re-sequencing. This is an exercise.

2 Three dimensions

Three-dimensional structures add 3-D components to the 0-D, 1-D and 2-D components if two-dimensional space. We call these "3-topoi", from the Greek, topos = place, and use the abbreviation ${}^3t$ in the table below. (The general term, polytope, includes polygons (Greek: gania = angle) in two dimensions and polyhedra (Greek: edros = plane face) in three.) The relationship of its faces to a 3-topos is that they surround it in a ball, which we indicate by b. A vertex in three dimensions is correspondingly emballed by the edges adjacent to it: vertex and 3-topos are the duals of each other in 3-space, as are edge and face. The table otherwise resembles the two-dimensional one, but note that both edges and faces now have three orthogonal projections.

	v	e	f	${}^3t$
# components	1	3	3	1
hasa_1	$\overset{2}{\longleftarrow}$	$\overset{c}{\longleftarrow}$	$\overset{b}{\longleftarrow}$	
hasa_2	$\overset{b}{\longrightarrow}$	$\overset{c}{\longrightarrow}$	$\overset{2}{\longrightarrow}$	
Euler	$v - e + f - {}^3t = 0$			

Here is a double tetrahedron, showing in black edges and the cycles of edges making up faces, in blue faces and the cycles of faces making up edges, and in red topoi and their directed links to each other. Edges and faces are also directed, as before, since they are two-dimensional. Cycles are shown as directed arcs, counterclockwise about edges and about the normals to faces. The "balls" have complicated topologies, dictated by the pairs of edges where a face meets a vertex and pairs of faces where an edge bounds a topos, and are just indicated as "b", in black for vertices and in red for topoi.



Here is an enumerated-sequence representation. Cycles start arbitrarily. Note that the edge direction with respect to a face, *dirE*, is 1 if the edge is directed in opposition to the counterclockwise cycle of edges making up the face, and 3 otherwise. Similarly, the face direction with respect to an edge, *dirF*, is 1 if the face is directed in opposition to the counterclockwise cycle of faces making up the edge, and 3 otherwise. Edge and face directions 0 or 2 are used to indicate if they are outgoing or incoming, respectively from and to the start and end vertices or topoi.

EF(*face*	*dirF*	*seqF*	*seqE*	*edge*	*dirE*)
F1	3	1	1	E2	3
F1	3	1	2	E8	3
F1	1	2	3	E3	1
F2	3	1	1	E3	3
F2	3	2	2	E9	3
F2	1	2	3	E1	1
F3	3	1	1	E1	1
F3	1	2	2	E2	3
F3	3	2	3	E4	3
F4	3	1	1	E4	3
F4	3	3	2	E8	3
F4	3	1	3	E9	3
F5	1	2	1	E7	1
F5	3	1	2	E6	3
F5	1	3	3	E9	1
F6	1	2	1	E5	1
F6	3	1	2	E7	3
F6	1	2	3	E8	1
F7	3	3	1	E4	3
F7	3	1	2	E5	1
F7	1	2	3	E6	3

VE(*vertex*	*edge*	*dirE*)
V1	E1	0
V1	E2	0
V1	E3	0
V2	E2	2
V2	E4	2
V2	E5	2
V2	E8	0
V3	E3	2
V3	E7	2
V3	E8	2
V3	E9	0
V4	E1	2
V4	E4	0
V4	E6	2
V4	E9	2
V5	E5	0
V5	E6	0
V5	E7	0

FT(*topos*	*face*	*dirF*)
T1	F1	0
T1	F2	0
T1	F3	0
T1	F4	2
T2	F4	0
T2	F5	0
T2	F6	0
T2	F7	0
T3	F1	2
T3	F2	2
T3	F3	2
T3	F5	2
T3	F6	2
T3	F7	2

In this representation, topoi are not directly connected to edges or vertices, and verticies are not directly connected to faces or topoi. These connections can be inferred by joins:

FT **ijoin** *EF*
VE **ijoin** *EF*

FT **ijoin** EF **ijoin** VE

These joins also refine the topologies of the balls of edges around vertices and of faces around topoi. These are always 2-cycles. For instance, the edges around T1 link pairs of faces: (F1,F2), (F1,F4), (F1,F3), (F2,F3), (F2,F4) and (F3,F4). Similarly, the faces around V3 link five pairs of edges: (E3,E8), (E3,E9), (E7,E8), (E7,E9) and (E8,E9).

2.1 Operations

The kinds of connections and disconnections one can make in three dimensions are more intricate than in two, and there does not appear to be a single, invertible operator for all. We can use a form of splice for the cycles of edges around faces and the cycles of faces around edges, but we must interpret what it means. The connections to vertices and topoi involve 2-cycles, as we noted above, and sometimes 1-cycles, as we shall see. The 1-cycles cannot be handled by swaps, since these require us to specify the edge (say) before the edge to be disconnected or connected, and "before" means in some cycle. The 2-cycles are simple enough not to need the full swap machinery.

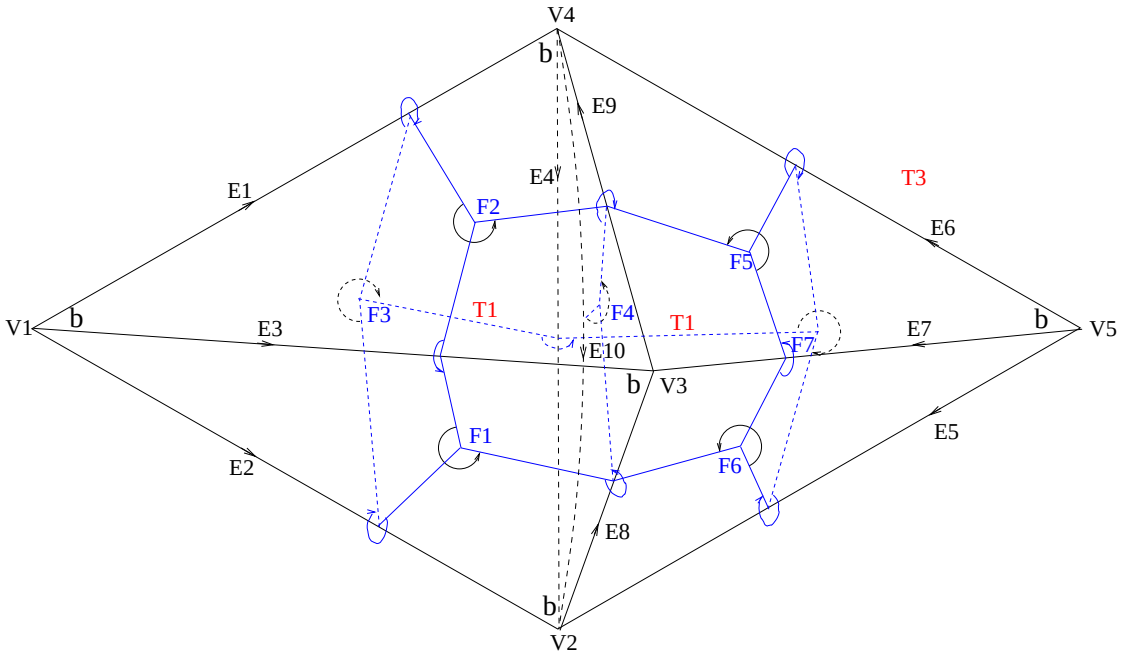
Within EF, there are two, dual, (dis)connections we can make. In each case, just as with disconnecting an edge from a vertex in 2-D, a new component is created (vertex in 2-D) and a component is lost through fusion (faces in 2-D). In 3-D we can disconnect a face from an edge, creating a new edge and fusing the two topoi formerly separated by the face. Or, trickier to imagine, but following directly from duality, we can disconnect an edge from a face, creating a new face and fusing the two vertices formerly connected by the edge.

There also is a special-case operator and its dual. This is the analog of removing a vertex between two edges in 2-D and so fusing the edges. In 2-D this is special because it won't work if more than the two edges meet at the vertex: the vertex must be a 2-cycle of edges. In 3-D, if an edge is a 2-cycle of faces, we can remove it from a vertex just as we did in 2-D, creating a new vertex and fusing the two faces. This is because an edge with only a pair of faces is a 2-D construct. An example would be to remove E3 from V3 in the double tetrahedron, fusing faces F1 and F2 into a single, four-sided face bounded by E2, E8, E9 and E1.

The dual of this removes a face from a topos if the face is a 2-cycle of edges, creating a new topos and fusing the two edges. (As was the case when we made a 2-D face out of one edge starting and ending at one vertex, this supposes that edges need not be straight: in this topological discussion we have gone beyond the restrictions to straightness imposed by the linear Clifford algebra.)

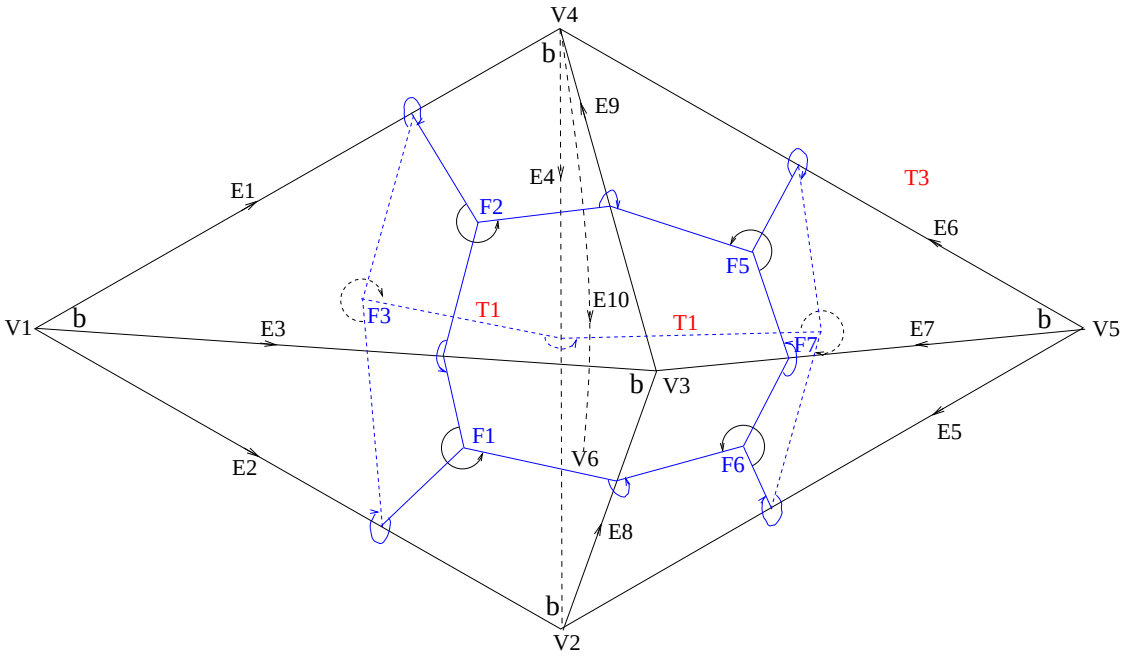
We must also consider edges which are 1-cycles of faces (we'll encounter one shortly). We have already seen a face which is a 1-cycle of edges.

Here is a sequence of three changes. First, we remove face F4 from edge E4, thus creating a new edge, E10, and merging topoi T1 and T2. Second, we remove this new edge from vertex V2 (this is legal because E10 has only one face, F4, and so its faces form a 1-cycle); this removes face F4, but leaves E10 hanging in space (a 0-cycle) ending at its new vertex, V6. Finally, we remove E3 from V3 (which is legal because E3 has a cycle of two faces), creating a new vertex, V7, and fusing faces F1 and F2. We could remove these dangling edges (E10 in the 3-space of T1, E3 in the 2-space of F1). We could pull face F1 away from an edge the way we did with F4 and E4, and then cut or remove that edge to merge topoi T1 and T3 and make a sort of envelope of four 2-D planes.



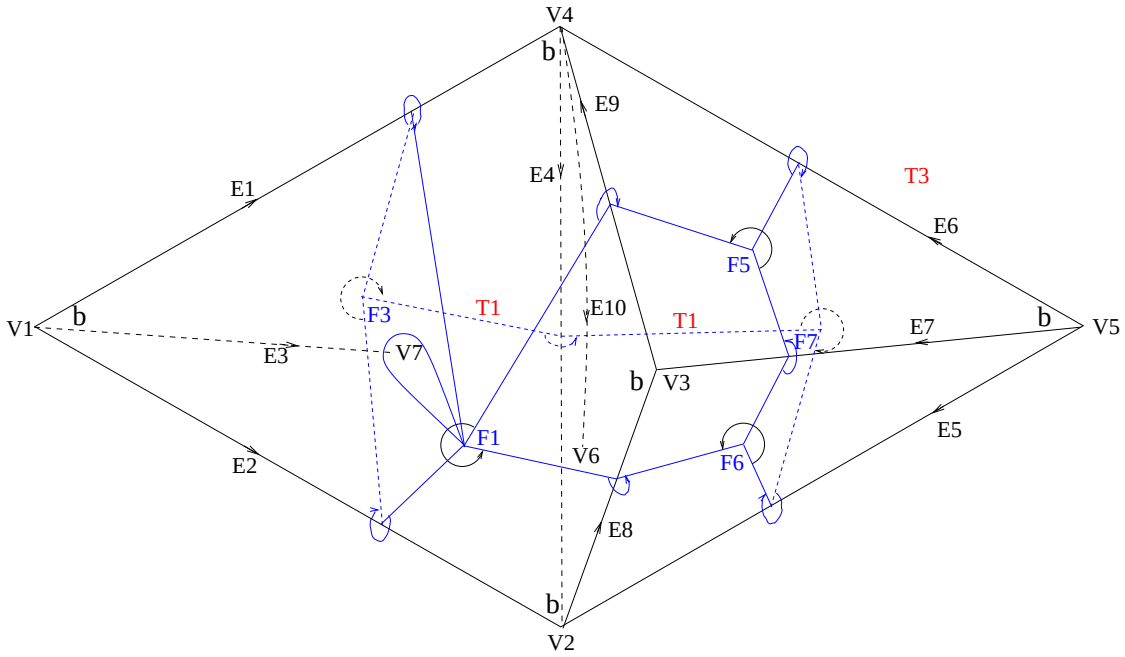
1. Remove F4 from E4.

EF(*face*	*dirF*	*seqF*	*seqE*	*edge*	*dirE*)	*VE*(*vertex*	*edge*	*dirE*)	*FT*(*topos*	*face*	*dirF*)
F1	3	1	1	E2	3	V1	E1	0	T1	F1	0
F1	3	1	2	E8	3	V1	E2	0	T1	F2	0
F1	1	2	3	E3	1	V1	E3	0	T1	F3	0
F2	3	1	1	E3	3	V2	E2	2	T1	F4	2
F2	3	2	2	E9	3	V2	E4	2	**T1**	**F4**	**0**
F2	1	2	3	E1	1	V2	E5	2	**T1**	**F5**	**0**
F3	3	1	1	E1	1	V2	E8	0	**T1**	**F6**	**0**
F3	1	2	2	E2	3	**V2**	**E10**	**2**	**T1**	**F7**	**0**
F3	3	2	3	E4	3	V3	E3	2	T3	F1	2
F4	**3**	**1**	**1**	**E10**	**3**	V3	E7	2	T3	F2	2
F4	3	3	2	E8	3	V3	E8	2	T3	F3	2
F4	3	1	3	E9	3	V3	E9	0	T3	F5	2
F5	1	2	1	E7	1	V4	E1	2	T3	F6	2
F5	3	1	2	E6	3	V4	E4	0	T3	F7	2
F5	1	3	3	E9	1	V4	E6	2			
F6	1	2	1	E5	1	V4	E9	2			
F6	3	1	2	E7	3	**V4**	**E10**	**0**			
F6	1	2	3	E8	1	V5	E5	0			
F7	**3**	**1**	**1**	**E4**	**3**	V5	E6	0			
F7	3	1	2	E5	1	V5	E7	0			
F7	1	2	3	E6	3						



2. Remove E10 from V2.

EF(*face*	*dirF*	*seqF*	*seqE*	*edge*	*dirE*)	*VE*(*vertex*	*edge*	*dirE*)	*FT*(*topos*	*face*	*dirF*)
F1	3	1	1	E2	3	V1	E1	0	T1	F1	0
F1	3	1	2	E8	3	V1	E2	0	T1	F2	0
F1	1	2	3	E3	1	V1	E3	0	T1	F3	0
F2	3	1	1	E3	3	V2	E2	2			
F2	3	2	2	E9	3	V2	E4	2			
F2	1	2	3	E1	1	V2	E5	2	T1	F5	0
F3	3	1	1	E1	1	V2	E8	0	T1	F6	0
F3	1	2	2	E2	3	**V6**	**E10**	**2**	T1	F7	0
F3	3	2	3	E4	3	V3	E3	2	T3	F1	2
						V3	E7	2	T3	F2	2
						V3	E8	2	T3	F3	2
						V3	E9	0	T3	F5	2
F5	1	2	1	E7	1	V4	E1	2	T3	F6	2
F5	3	1	2	E6	3	V4	E4	0	T3	F7	2
F5	1	3	3	E9	1	V4	E6	2			
F6	1	2	1	E5	1	V4	E9	2			
F6	3	1	2	E7	3						
F6	1	2	3	E8	1	V5	E5	0			
F7	3	1	1	E4	3	V5	E6	0			
F7	3	1	2	E5	1	V5	E7	0			
F7	1	2	3	E6	3						



3. Remove E3 from V3.

EF(*face*	*dirF*	*seqF*	*seqE*	*edge*	*dirE*)	*VE*(*vertex*	*edge*	*dirE*)	*FT*(*topos*	*face*	*dirF*)
F1	3	1	1	E2	3	V1	E1	0	T1	F1	0
F1	3	1	2	E8	3	V1	E2	0			
F1	**1**	**2**	**6**	**E3**	**1**	V1	E3	0	T1	F3	0
F1	**3**	**1**	**5**	**E3**	**3**	V2	E2	2			
F1	**3**	**2**	**3**	**E9**	**3**	V2	E4	2			
F1	**1**	**2**	**4**	**E1**	**1**	V2	E5	2	T1	F5	0
F3	3	1	1	E1	1	V2	E8	0	T1	F6	0
F3	1	2	2	E2	3	V6	E10	2	T1	F7	0
F3	3	2	3	E4	3	**V7**	**E3**	**2**	T3	F1	2
						V3	E7	2	T3	F2	2
						V3	E8	2	T3	F3	2
						V3	E9	0	T3	F5	2
F5	1	2	1	E7	1	V4	E1	2	T3	F6	2
F5	3	1	2	E6	3	V4	E4	0	T3	F7	2
F5	1	3	3	E9	1	V4	E6	2			
F6	1	2	1	E5	1	V4	E9	2			
F6	3	1	2	E7	3						
F6	1	2	3	E8	1	V5	E5	0			
F7	3	1	1	E4	3	V5	E6	0			
F7	3	1	2	E5	1	V5	E7	0			
F7	1	2	3	E6	3						

We see from the third step that the two-dimensional disconnection of E3 separating F1 and F2 does not follow the 2-D splice() operation on a *VFE* relation (although we could construct such a relation and apply splice() to it). Instead, the topological changes occur in *EF*, with a new V7 replacing one of the V3 tuples in *VE*, and the F2 tuples just deleted from *FT*. The deletions in particular are not invertible operations, and it would be worth finding a place for F2 somewhere so that an inverse operation could put it back. (In step 1, *VE* had some additions, for which there might be a similar consideration.)

References

[GS85] L. Guibas and J. Stolfi. Primitives for the manipulation of general subdivisions and the computation of voronoi diagrams. *ACM Transactions on Graphics*, 4:74–123, 1985.

[MBC+02] T. H. Merrett, Y. Bédard, D. J. Coleman, J. Han, B. Moulin, B. Nickerson, and C. V. Tao. A tutorial on database technology for geospatial applications. www.cs.mcgill.ca/∼tim/geodem/tutorial.ps, 2002.