A Description of *PlotVol*

C. Godsalve
F2 Swainstone Road
Reading
RG2 ODX
United Kingdom
Tel: (0118) 9864389

email:`seagods@hotmail.com`

1 The Problem

In the SurfPlot routine, we had a function $F(x, y)$ defined at a set of nodes in the (x, y) plane. These points had a triangulation. The triangulation could have been defined on a regular grid, or it may have been a Delaunay triangulation on a set of arbitrary points.

We then mapped the triangulation in the two dimensional plane to a two dimensional surface embedded in a three dimensional space. That is we mapped $(x_i, y_i) \rightarrow (x_i, y_i, z_i)$ where $z_i = F(x_i, y_i)$. As an alternative, we could have used ContPlot, where we do not map the triangulation into a three dimensional space, but use contours instead. Of course we view the two dimensional contour map from a point outside the two dimensional plane.

The PlotVol routine faces a similar problem. We have a function defined at a set of nodes (x_i, y_i, z_i). Instead of a two dimensional region being decomposed into triangles, we have a three dimensional region decomposed into tetrahedra. It may be a regular mesh such as is given by *basicCube*, or it may be an irregular mesh generated by a three dimensional delaunay ”triangulation”.

We cannot do the same ”trick” as we did in SurfPlot, as we cannot view the three dimensional ”surface” embedded in four dimensions. So, we must use the same technique as in ContPlot. Instead of a set of one dimensional contours embedded in a two dimensional space, we shall have two dimensional isosurface ”contours” embedded in a three dimensional space.

There are differences between these two dimensional contours and the one dimensional contours of ContPlot. First, we cannot view the contours from ”outside” as we can with the contours of a function of two variables. This means that we must use transparancy and lighting to visualise the contours. An immediate consequence is that the surfaces will have to be oriented. (We simply ignore the possibility of a non-orientable surface such as a Klein Bottle.)

Now, orientation is a global property. If we have a function $W = W(x, y, z)$, and a set of contour levels W_c we can find the intersection of $W = W_c$ with a tetrahedron via linear interpolation of the function values at the nodes. This will either be a triangle where the surface ”cuts off” one of the corners, or a quadrilateral where the surface separates two edges. However, this gives us no information whatsoever about the global orientation. All we can do is ensure that neighbouring facets of the isosurface have the same orientation as given by surface normal of the facets. We may change the orientation of all the facets of a given isosurface later. What we cannot have is an isosurface where neigbours have opposite surface normals. This will mess up the lighting which we need to visualise the surface with.

A simple example would be a sphere made up of triangular facets. Here we might end up with all the surface normals pointing outward, or all pointing inward. For lighting, we would want them all pointing outwards. All we have to do to swap is calculate the surface normals as $\mathbf{b} \times \mathbf{a}$ instead of $\mathbf{a} \times \mathbf{b}$.

So, what strategy is needed to ensure that the isosurface has a global orientation? We shall use the following method. Given a W_c, we visit each tetrahedron in turn, until we find one which intersects the surface $W = W_c$. We assign an arbitrary orientation, and call a function called $SurfGrow$ passing it the identity of the cut tetrahedron. On a first call, the first facet of the surface is found. Then $SurfGrow$ will call itself with the identity neighbouring tetrahdra, and so on, ensuring the intersecting facets in the neighbours have the same orientation as facet in the tetrahedron that calls $SurfGrow$. In this way, we grow the surface while maintaining a global orientation. We shall now go into detail as to how this is done.

2 The Tetrahedron

In Fig.1, we have a tetrahedron. We have nodes one to four. Node 1 shall be the origin of a local coordinate system. The vector $\vec{12}$ from node 1 to node 2 shall be labelled $\mathbf{a}$. The vector $\vec{13}$ shall be $\mathbf{b}$, and the vector $\vec{14}$ shall be $\mathbf{c}$. The three edge vectors $\mathbf{e_1}$, $\mathbf{e_2}$, and $\mathbf{e_3}$ shall correspond to $\mathbf{a}$, $\mathbf{b}$, and $\mathbf{c}$. Edges $\mathbf{e_4}$, $\mathbf{e_5}$, and $\mathbf{e_6}$, are $\vec{23}$, $\vec{24}$, and $\vec{34}$. The faces 1 to 4 are numbered so that face 1 does not contain node 1, face 2 does not contain node 2, and so on.

We assume that our tetrahedral mesh is such that $\mathbf{a}\cdot(\mathbf{b}\times\mathbf{c})$ is always positive. Further, we encode the following information. Each tetrahedron shall store the tetrahedron numbers of the neighbours on faces 1 to 4, and it shall also store the face numbers that a point crossing its face would arrive at. For instance, if we have a point on face 1 of the current tetrahedron, this point might be on face 3 of the neighbour on face 1.

An isosurface can intersect a tetrahedron in different ways. For the present we ignore the possibility that the isosurface passes through the tetrahedron nodes. Now, we shall look at edges 1 to 6 in the order depicted in Fig.1. We shall have either three cut edges or four. Our isosurfaces shall consist of (mostly triangular) facets, the facets shall have 3 or for corners refered to as nodes t which are not nodes in the tereahedral mesh. We call the nodes that cut the tetrahedron t_1, t_2, t_3, and t_4 if there is a cut on a fourth edge.

For the moment, we assume that we arrive in the $SurfGrow$ function from the main function, that is to say, it is a first call before any recursion begins. We have a boolean matrix $Done[itet][idmax]$. This is set false for each tetrahedron. Just as with a contour

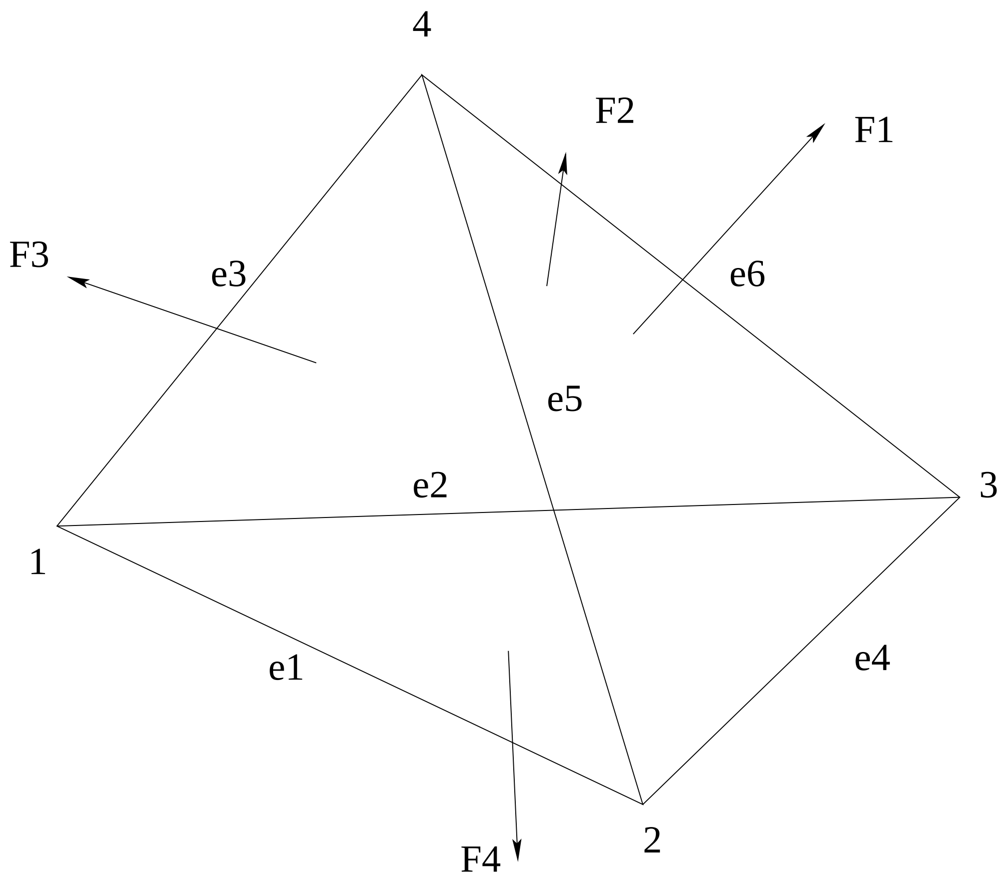

Figure 1: The node, edge, and face numbers for the tetrahedron

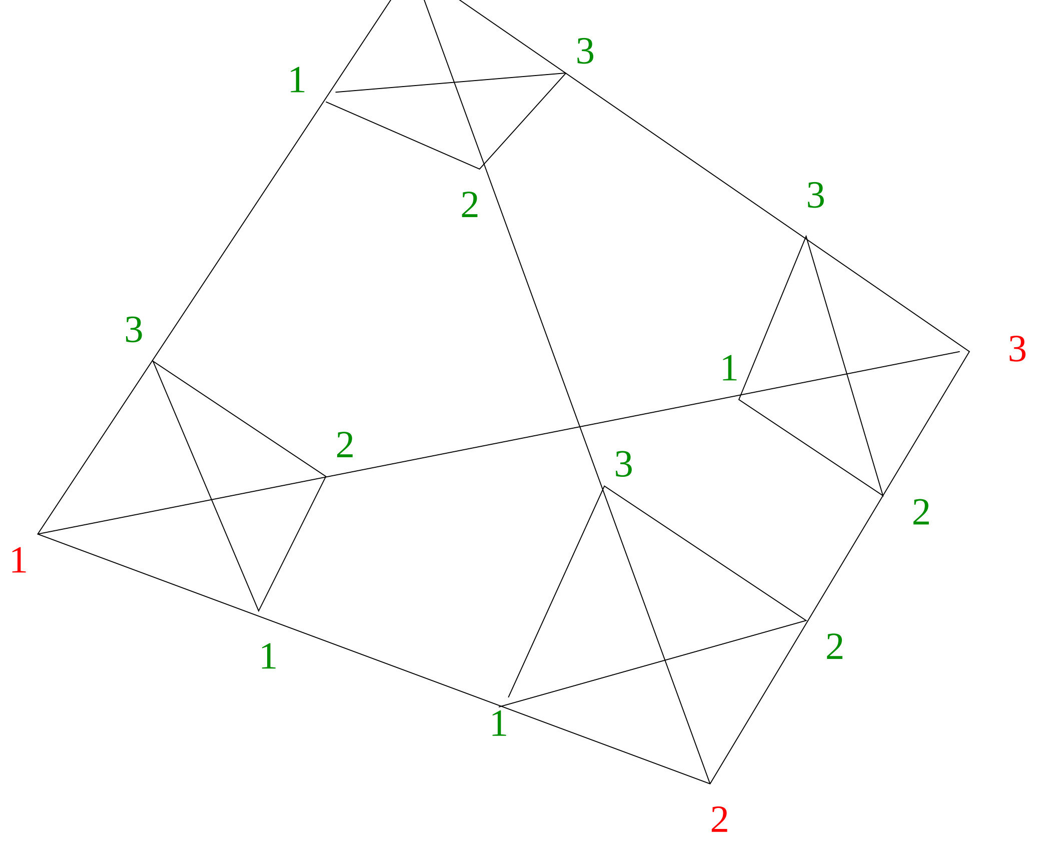

Figure 2: Triangle numbering for the possibilities of cut off nodes

map, we may have more than one contour for each contour level. At the moment we allow for a maximum number of isosurfaces per level to be $idmax$.

In Fig.2, we see the possibilities where there are only three cut edges. As we visit the edges 1 to 6 in order, then t_1, t_2, and t_3 are the last three new nodes (in order). On inspection, we soon arrive at table 1. Because we don't know the orientation of the surface, we do not insist that the three triangle nodes must be clockwise or anticlockwise when viewed from inside or outside of the tetrahedron. Reading the first three lines of the tetrahedron, we see that if tetraheron node 1 is cut off, there are cuts on faces 2, 3, and 4. Each of these cut faces are cut by a line segment between nodes t_a and t_b. So that face 2 is cut by new triangle nodes t_2 and t_3, Face 3 is cut by nodes t_3 and t_1, and face 4 is cut by nodes 1 and 2. It's not much of a table as the third and fourth column's repeat for every "cut off" node.

So, if we have three cuts we must call $SurfGrow(in, t_a, t_b, if, W_c)$ three times for the neighbours in on three cut faces, with triangle node numbers t_a and t_b. When it arrives in $SurfGrow$ on this recursive call, in tells $SurfGrow$ which tetrahedron it is in, and that the calling tetrahedron is on face if of the current tetrahedron. Lastly, W_c is the contour level.

Node	Face	t_a	t_b
1	2	2	3
1	3	3	1
1	4	1	2
2	1	2	3
2	3	3	1
2	4	1	2
3	1	2	3
3	2	3	1
3	4	1	2
4	1	2	3
4	2	3	1
4	3	1	2

Table 1: The faces cut by nodes t_1, t_2, and t_3 if the node in the first column is "cut off" by the isosurface.

Before looking at what happens in the recursive call, we consider the possibility of four cut nodes in a first call to $SurfGrow$. This happens when the isosurface separates two edges. The possiblilities are depicted in Figs.3, 4, and 5. We note that the linear interpolation simplifies things a great deal in that we can have only one line segment and two edge cuts on a facet. If we had four edge cuts in total, but the specified edge pairs were not separated, this would imply three edge cuts on the same facet.

This is not possible. If we have cuts on edge 1 (node 1 - node 2) and edge 2 (node 1 - node 3) of a tetrahedron facet, then this implies that either nodes 2 and 3 are *both* greater to W_c or *both* less than W_c. Since the interpolation is linear, there cannot be a third cut. More than four edge cuts also implies more than two cuts on a facet. So, the three possibilities in Figs.3 to 5 are all that remain. Again, we note that we visit edges 1 to 6 in order, and that t_1 to t_4 "arrive" in order.

So if the first call to $SurfGrow$ has four cut edges, $SurfGrow$ will have to be called from itself on all four faces with the node numbers given in table 2.

Now, on a first call to $SurfGrow$, the tetrahedron number, face numbers and triangle node numbers are all set negative, and *Done* is set false. On arriving in $SurfGrow$, we check if *Done* is true or not. If true, we just return 0. Then, if *Done* is false, we set it to be true. Next, if the isosurface triangle numbers are positive, we skip the first call stuff. The first call just works out the cuts, introduces three new triangle nodes, and calls $SurfGrow$ on all the cut faces.

So, what happens when we arrive in $SurfGrow$ on a recursive call? When we call $SurfGrow$, we call it passing the neighboring tetrahedron numbers in. Inside the recursive call, this is the current tetrahedron it. We are given the tetrahedron number it, the values of t_a and t_b (internally refered to as $icut1$ and $icut2$. We are also given the face $ifacelook$ of it on which these nodes lie.

There are two possibilities. The tetrahedron it may have one or two *extra* cuts depending on whether it contains a triangular or a quadrilateral facet. We sum things up in table 3. The first column is $ifacelook$, the face which is already cut by two nodes $icut1$ and

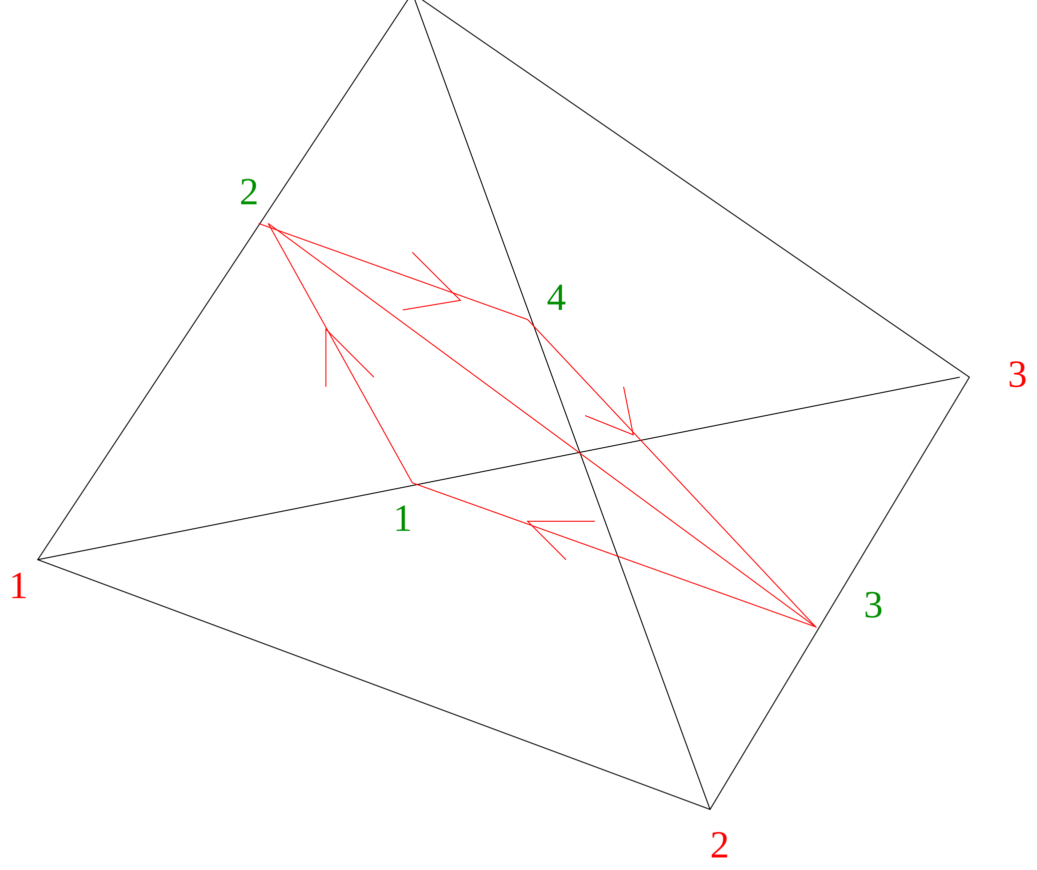

Figure 3: Edge 1 and edge 6 are separated

Edge Pair	Face	t_a	t_b
1 6	1	4	3
1 6	2	1	2
1 6	3	2	4
1 6	4	3	1
2 5	1	4	3
2 5	2	2	4
2 5	3	1	2
2 2	4	3	1
3 4	1	4	3
3 4	2	2	4
3 4	3	3	1
3 4	4	1	2

Table 2: The faces cut by nodes t_1, t_2, t_3, and t_4 if the given an edge pair separated by the isosurface.

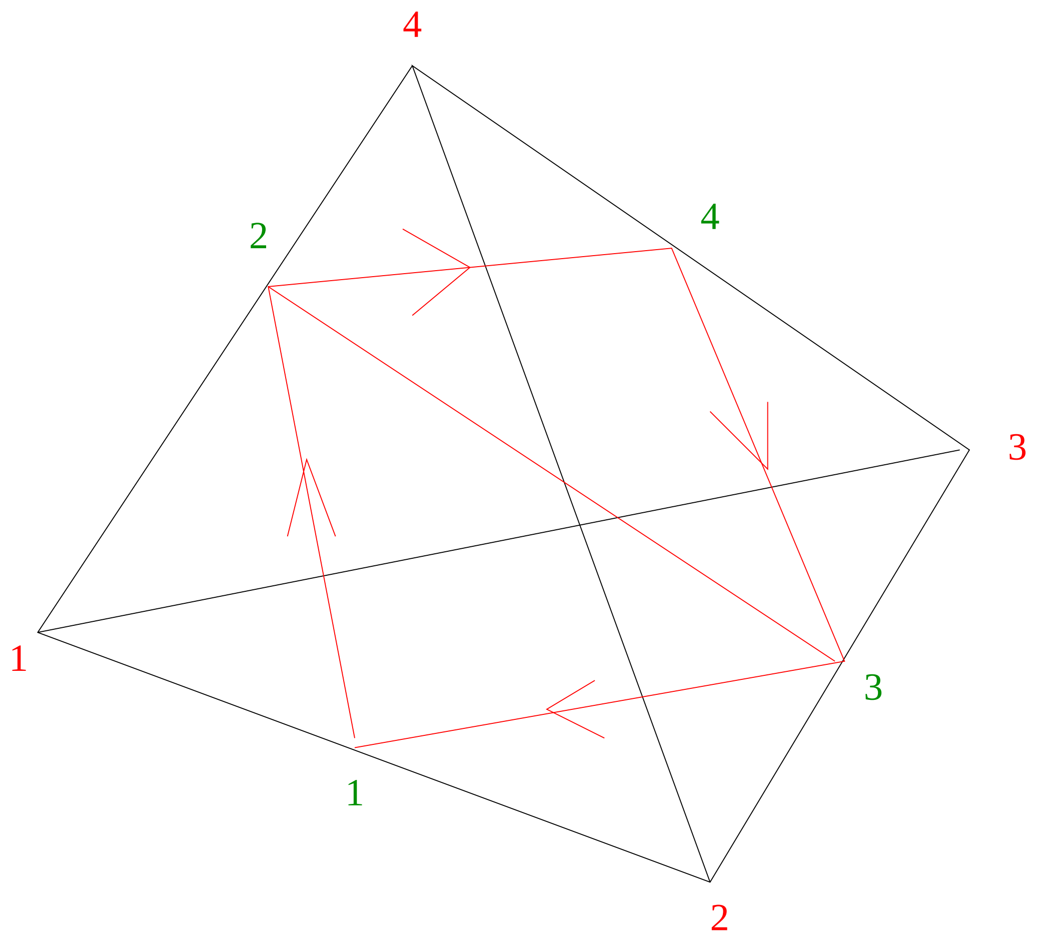

Figure 4: Edge 2 and edge 5 are separated

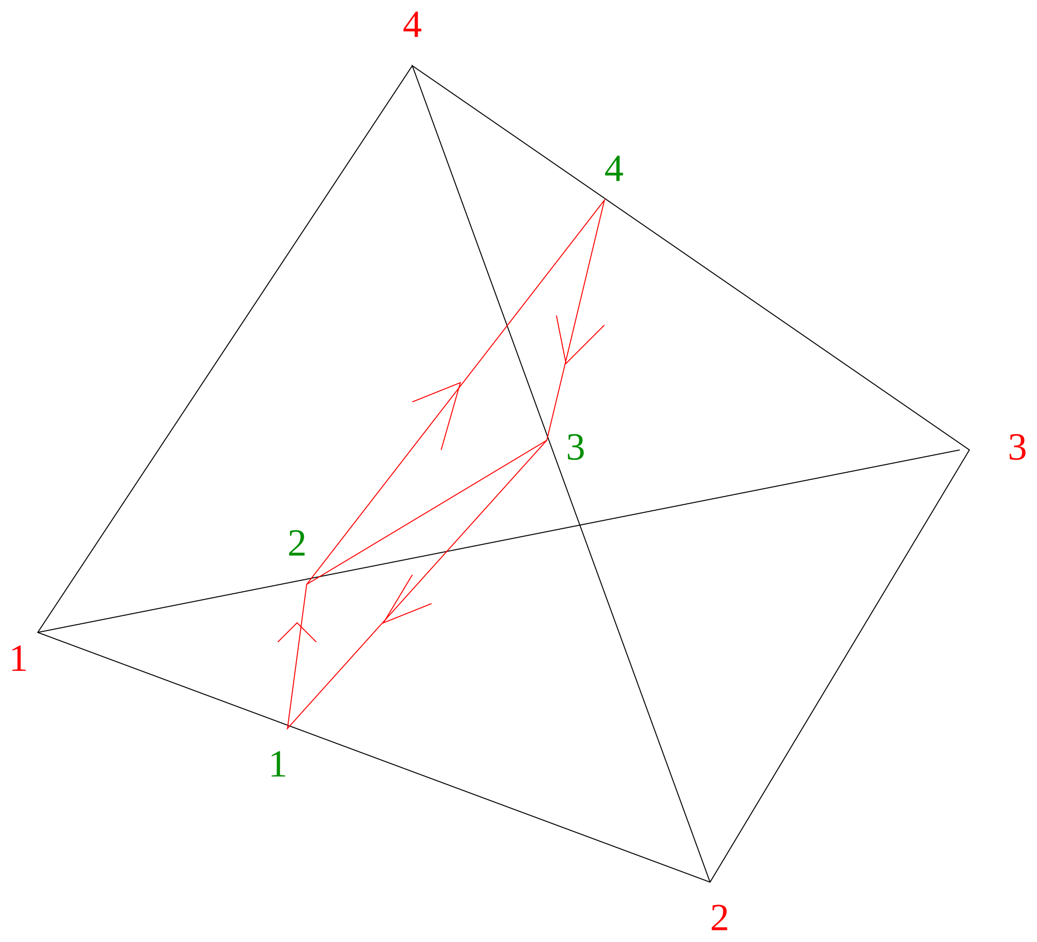

Figure 5: Edge 3 and edge 4 are separated

face	edge/faces			Fig.3	Fig.4	Fig.5
1	[1] (3,4)	[2] (2,4)	[3] (2,3)	[2](4,2)[3](2,3)	[1](4,3)[3](3,2)	[1](3,4)[2](4,2)
2	[1] (3,4)	[4] (2,4)	[5] (1,4)	[5](3,1)[4](1,4)	[4](1,4)[1](4,3)	[5](1,3)[1](3,4)
3	[2] (3,4)	[4] (1,4)	[6] (1,2)	[4](1,4)[2](4,2)	[6](2,1)[4](1,4)	[2](4,2)[6](2,1)
4	[3] (2,3)	[5] (1,3)	[6] (1,2)	[3](2,3)[5](3,1)	[3](3,2)[6](2,1)	[6](2,1)[5](1,3)

Table 3: Which edges might be cut given if $ifacelook$=1,2,3 and 4.

$icut2$. We have put edge numbers in brackets, and face pairs in parentheses. The next three columns represent cases where there is only one extra edge cut, and the next three columns after these represent cases where there are two extra edge cuts as in Figs.3-5.

We shall write out explictly the meaning of the row for face 1 having already been cut by two nodes. There are 6 possibilities. Edge 1 may have a cut, in which case $SurfGrow$ must call itself on neighbours on faces 3 and 4. Edge 2 may have a cut, in which case $SurfGrow$ must call itself on neighbours on faces 2 and 4. Edge 3 may have a cut, in which case $SurfGrow$ must call itself on neighbours on faces 2 and 3. Then there are the possibilities that edge 2 and edge 3 are cut: edge 2 is shared by faces 2 and 4, edge 3 is shared by faces 2 and 3. Then it may be that edge 1 and edge 3 are cut: edge 1 is shared by faces 4 and 3, edge 3 is shared by faces 3 and 2. Lastly it may be that edge 1 and edge 2 are cut: edge 1 is shared by faces 4 and 3, edge 2 is shared by faces 3 and 2.

The reader may prefer a diagram to the last 3 columns of Table.3. In Fig.6, we summarise Figs.3 to 5. For instance, if face 4 is cut by $icut1$ and $icut2$, there may be two further cuts on face 1 on the opposite side of the square. These will be t_4 and t_3 in our naming convention, and these will be on edges six and 5 of the tetrahedron. Edge 6 is common to face 2 and face 1, while edge 5 is common to face 1 and face 2 of the tetrahedron.

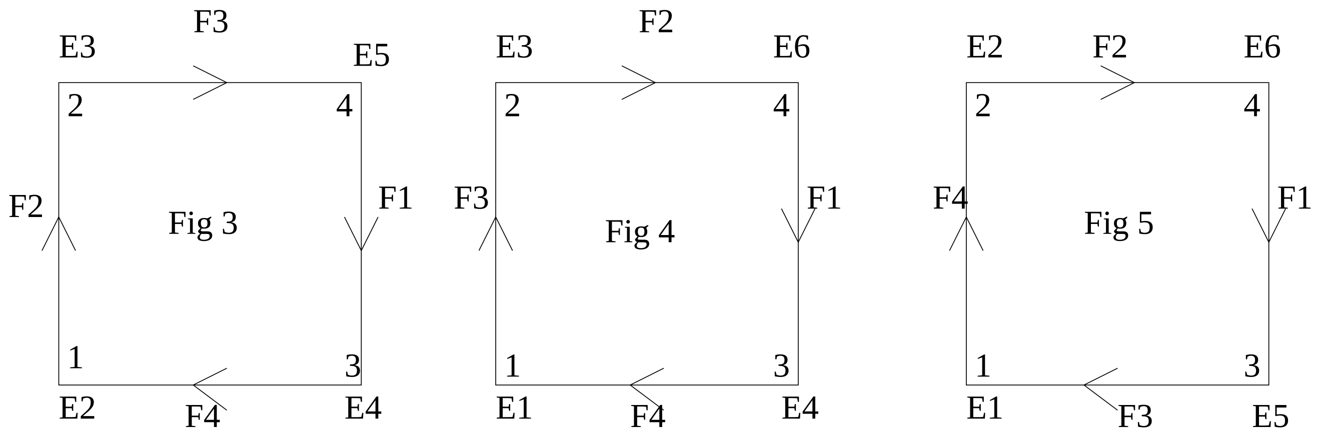

Figure 6: A summary of Figs.3 to 5. t_1 to t_4 are marked on the insides of the squares. The edge numbers are marked outside the corners, and the faces shared by an edge are marked on the edges of the squares.