Chapter 2

Triangulation of Link Complements

Overview

This chapter tells how link complements are split into ideal tetrahedra, and how the cusp and gluing equations are produced. The algorithm has six main parts:

(1) Decomposition of the link complement into two ideal polyhedra.

(2) Triangulation of the polyhedra's surfaces.

(3) Decomposition of the ideal polyhedra into ideal tetrahedra.

(4) Simplification of the ideal triangulation.

(5) Selection of the preferred longitude and meridian for each cusp.

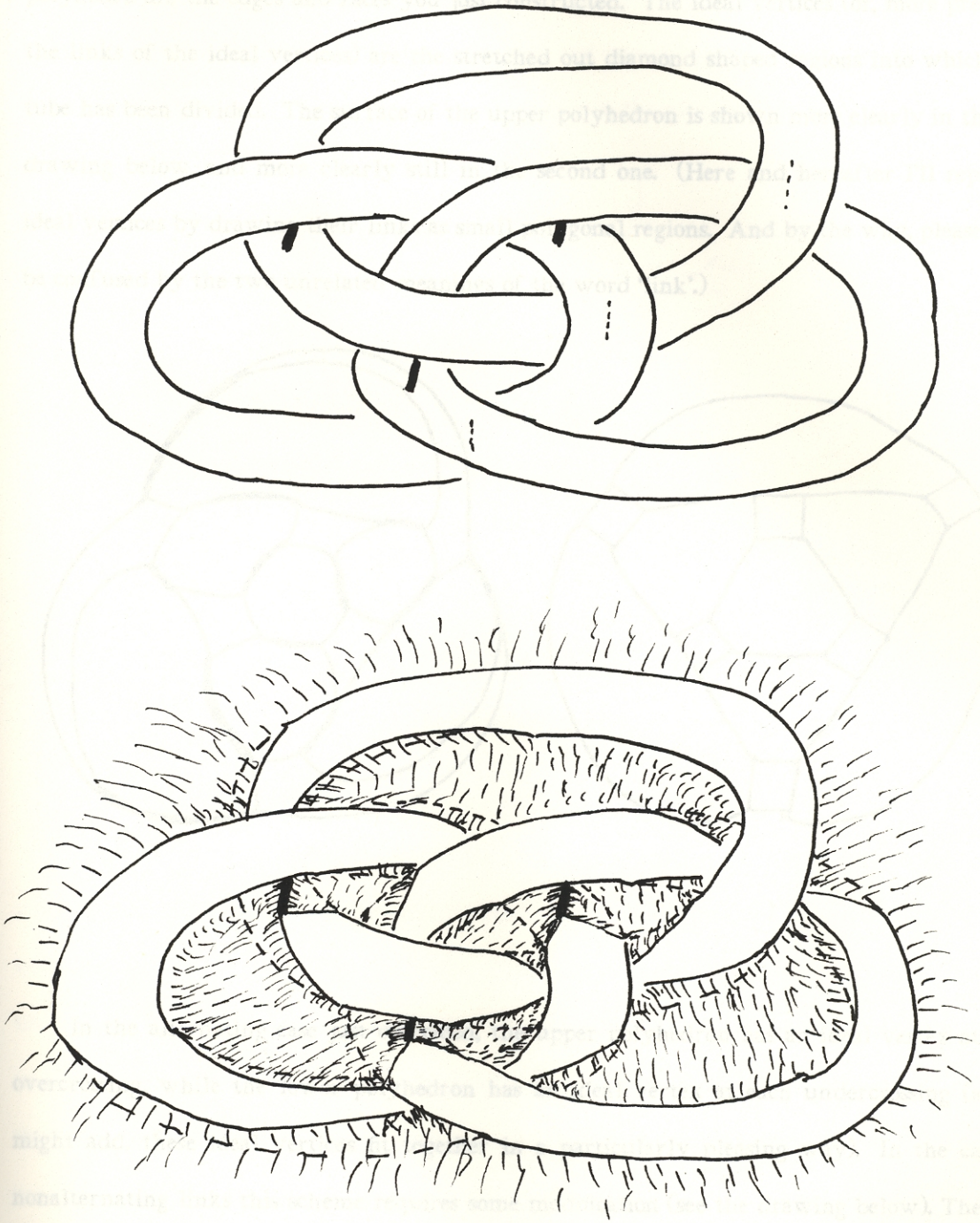
(6) Selection of a nonredundant set of gluing equations.

The program CARROT carries out the algorithm. The organization of this chapter follows the organization of CARROT quite closely. Source code for CARROT is located in WEEKS/DEHN/INNARDS1.

Decomposition of the link complement into two ideal tetrahedra

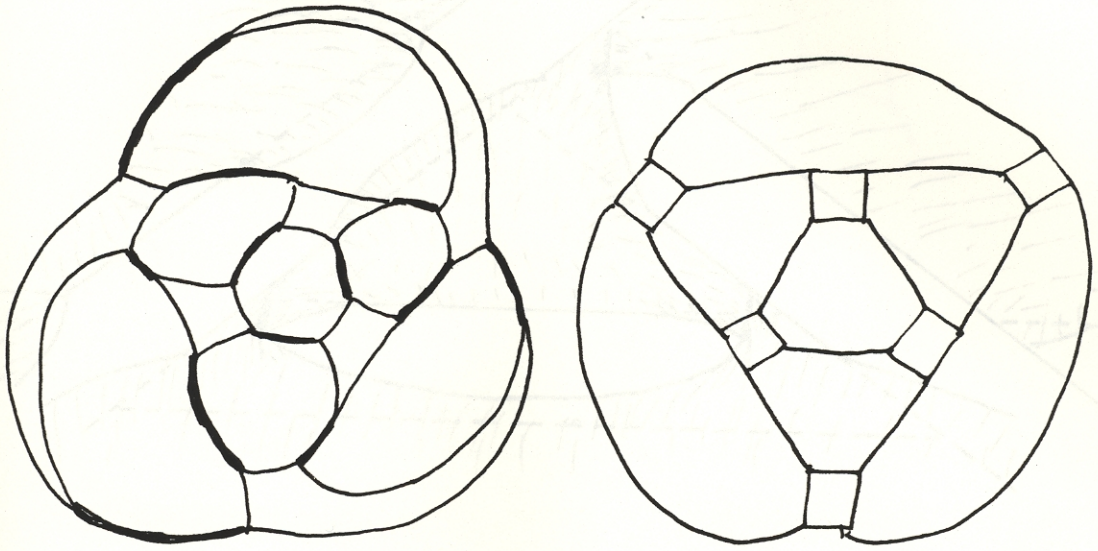
The decomposition is a little simpler for alternating links, so we'll look at those first and then move on to nonalternating ones. This procedure, at least in the alternating case, originated with Thurston [TH2] and was made explicit by Lawson [L] and Menasco [M]. I'm assuming the reader is familiar with Thurston's decomposition of the figure-eight knot complement. Readers familiar with Menasco's work should be forewarned that although the follow-

ing is topologically equivalent to Menasco's approach, it looks a little different subjectively.



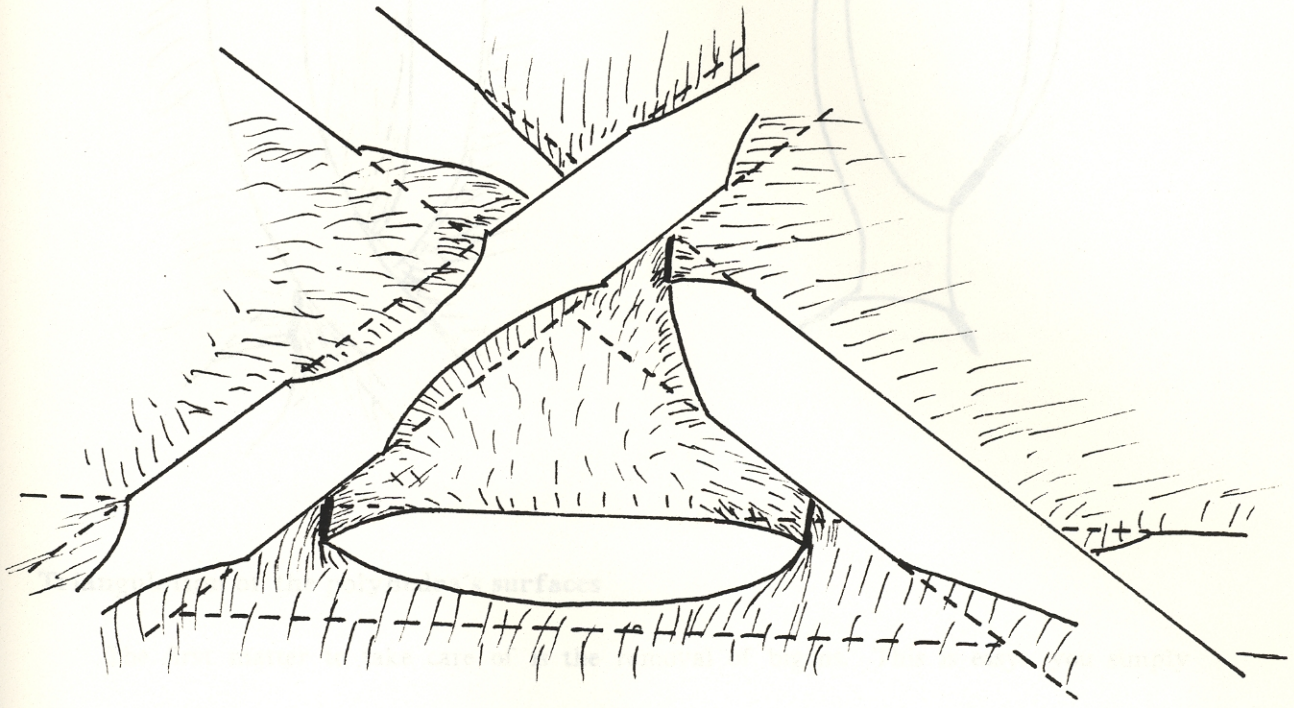
Thicken each link component into a tube, as shown in the above drawings of the borromean rings. At each crossing, run an edge from the lower strand of the link to the upper

strand, as shown in the first drawing above. Now fill in faces as shown in the second drawing. This divides the three-sphere into two ideal polyhedra. The edges and faces of the polyhedra are the edges and faces you just constructed. The ideal vertices (or, more precisely, the links of the ideal vertices) are the stretched out diamond shaped regions into which each tube has been divided. The surface of the upper polyhedron is shown more clearly in the first drawing below, and more clearly still in the second one. (Here and hereafter I'll represent ideal vertices by drawing their links as small polygonal regions. And by the way, please don't be confused by the two unrelated meanings of the word 'link'.)

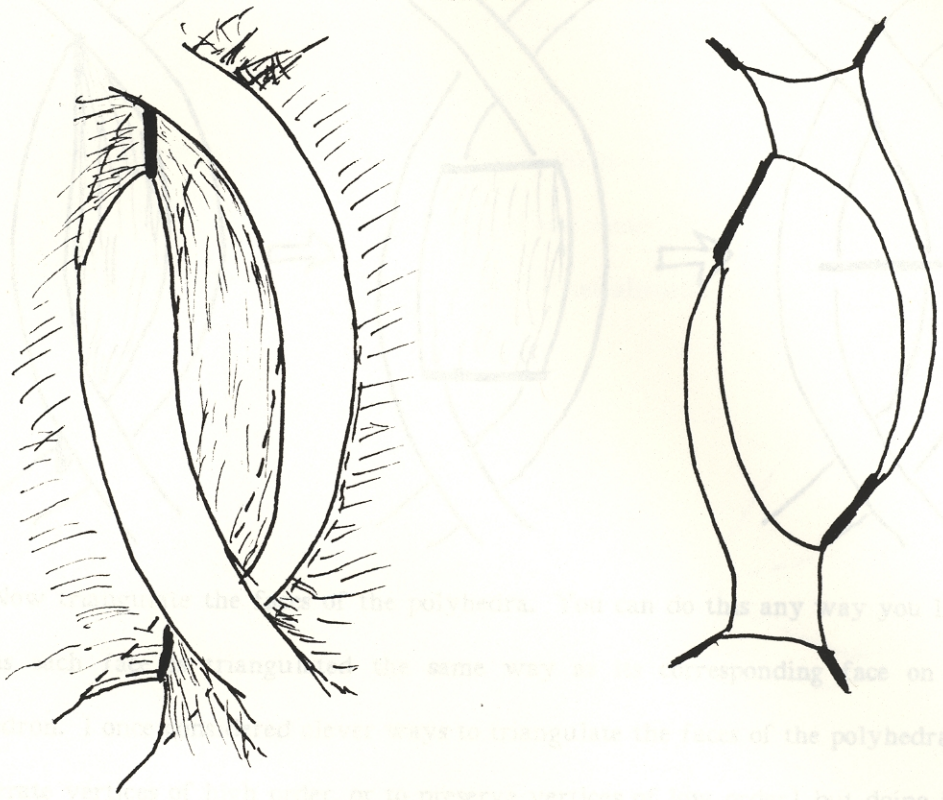


In the alternating case just described, the upper polyhedron has an ideal vertex at each overcrossing, while the lower polyhedron has an ideal vertex at each undercrossing (and, I might add, these ideal vertices fit together in a particularly pleasing way). In the case of nonalternating links this scheme requires some modification (see the drawing below). The philosophy will be the same as in the alternating case: the tubular neighborhoods of the link components will be divided into ideal vertices in such a way that the upper portions of the tubes become ideal vertices of the upper polyhedron and the lower portions of the tubes

become ideal vertices of the lower polyhedron. Thus when there are two or more consecutive overcrossings you get one long vertex for the upper polyhedron, and one or more little vertices for the lower polyhedron. Similarly, when there are two or more consecutive undercrossings you get a long vertex for the lower polyhedron, and one or more little vertices for the upper one. The edges and faces aren't much different than in the alternating case.

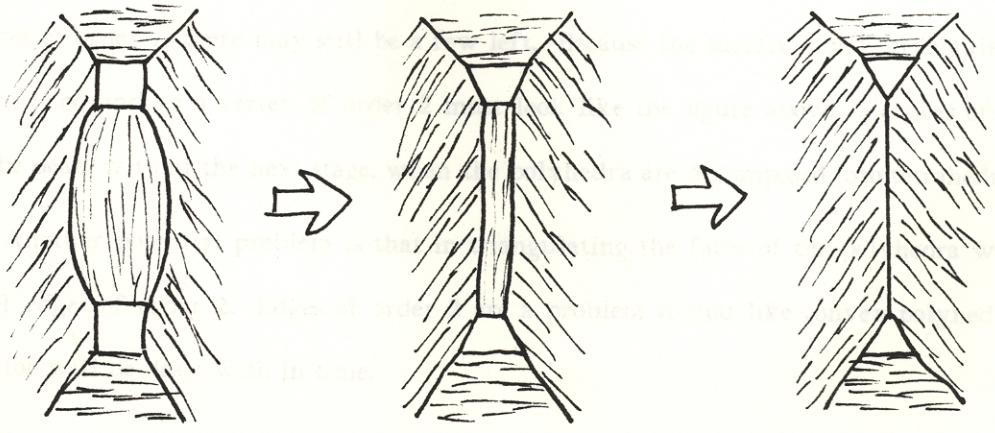


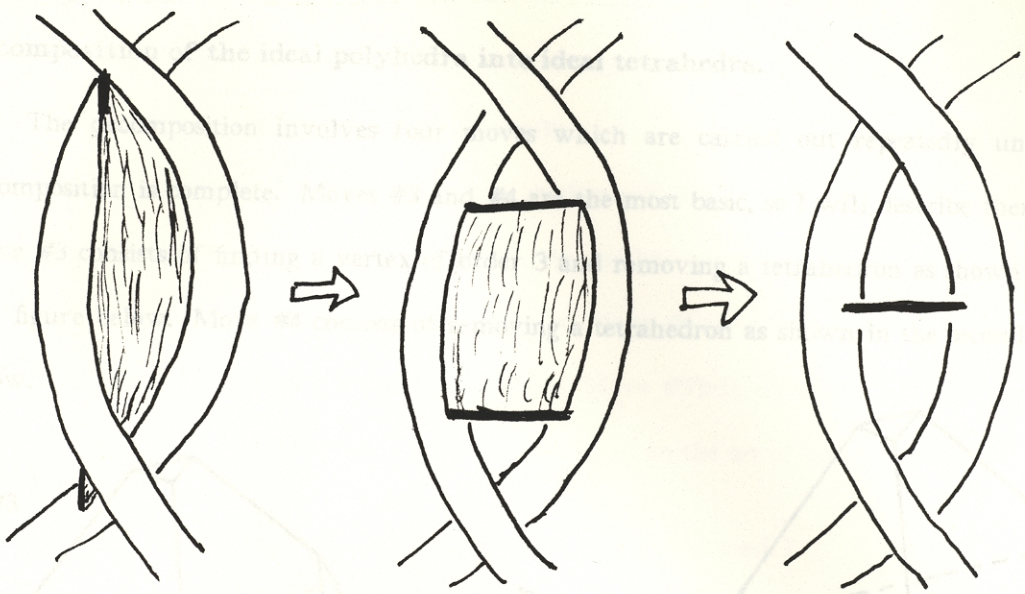
What we have is topologically just fine, but geometrically we would have a mess if we tried to realize the ideal polyhedra in hyperbolic space. There may be bigons (faces with only two sides) as shown in the drawing below. There may also be vertices of order 2, as shown in the nonalternating example above. Fortunately we don't need to realize these nasty polyhedra in hyperbolic space. Instead our strategy will be to continue working topologically, dividing the ideal polyhedra in ideal tetrahedra and eliminating the bigons and vertices of order 2 as we go. It will then be a simple matter to represent the ideal tetrahedra in hyperbolic space.



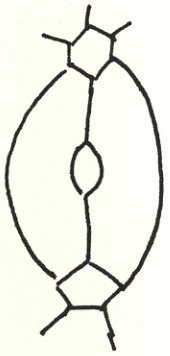
Triangulation of the polyhedra's surfaces

The first matter to take care of is the removal of bigons. This is easy: you simply squeeze the bigon out of existence. The first pair of drawings below shows what this looks like on the surface of an ideal polyhedron. The second pair of drawings shows the situation in the manifold itself. Just be sure the bigon's two edges are distinct—otherwise you've got an embedded cylinder or Mobius strip which cannot be removed.





Now triangulate the faces of the polyhedra. You can do this any way you like, just so long as each face is triangulated the same way as its corresponding face on the other polyhedron. I once considered clever ways to triangulate the faces of the polyhedra (e.g. so as to generate vertices of high order, or to preserve vertices of low order), but doing it stupidly seems to work just fine.



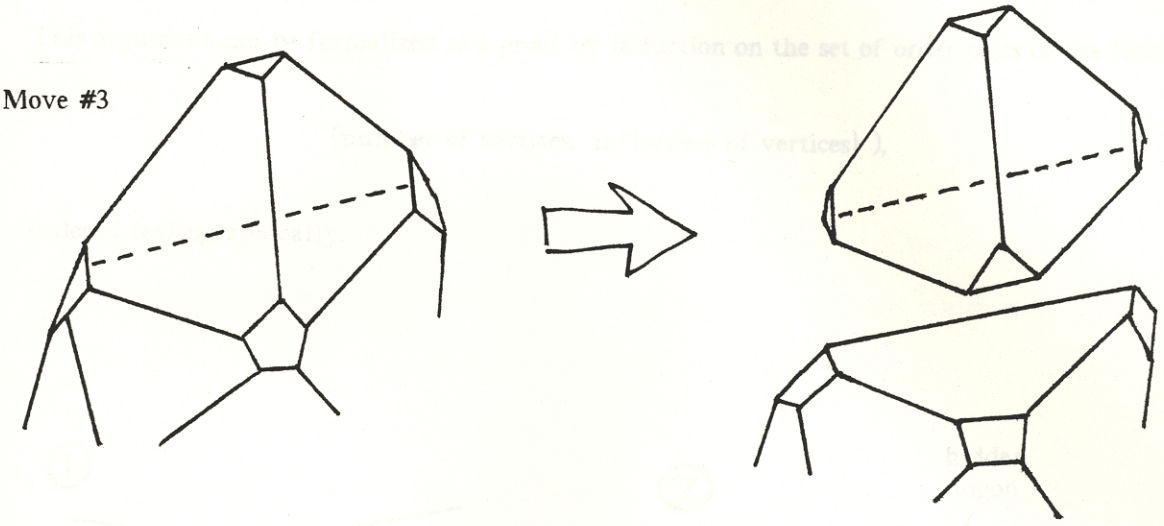
By the way, even though the triangulation process increases the order of some of the vertices of order 2, there may still be a few left. Because the surface is now triangulated, the regions surrounding a vertex of order 2 must look like the figure above. Vertices of order 2 will be dealt with at the next stage, when the polyhedra are decomposed into tetrahedra.

Another potential problem is that in triangulating the faces of the polyhedra we create lots of edges of order 2. Edges of order 2 are a problem if you like convex polyhedra. But these too will be dealt with in time.

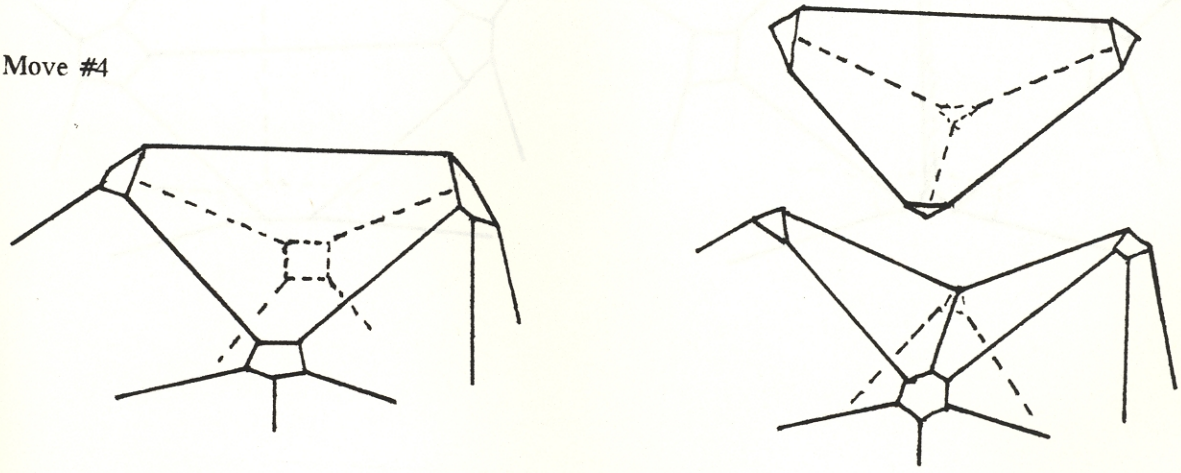
Decomposition of the ideal polyhedra into ideal tetrahedra.

The decomposition involves four moves which are carried out repeatedly until the decomposition is complete. Moves #3 and #4 are the most basic, so I will describe them first. Move #3 consists of finding a vertex of order 3 and removing a tetrahedron as shown in the first figure below. Move #4 consists of removing a tetrahedron as shown in the second figure below.

Move #3



Move #4

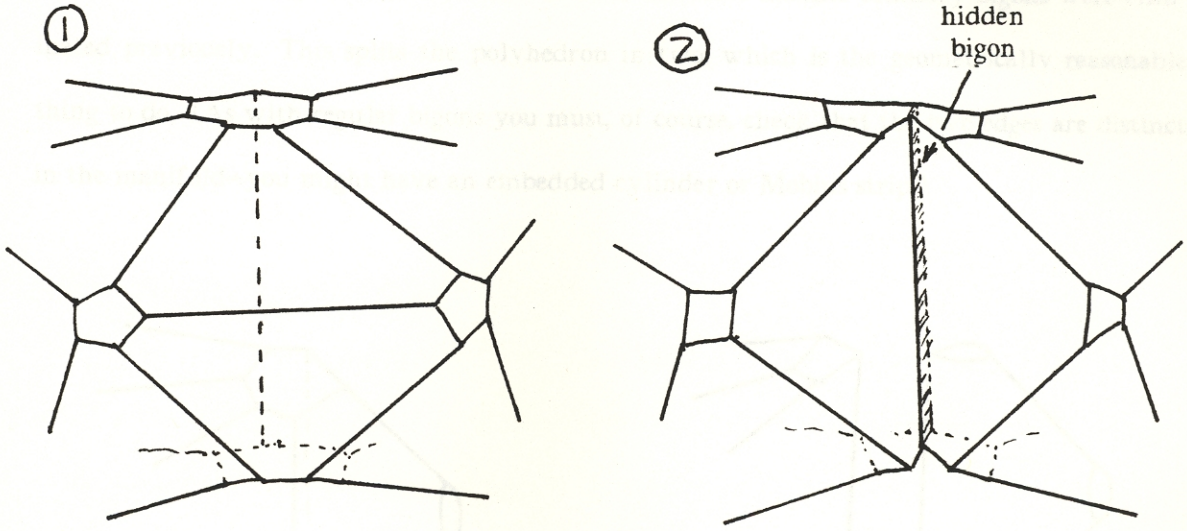


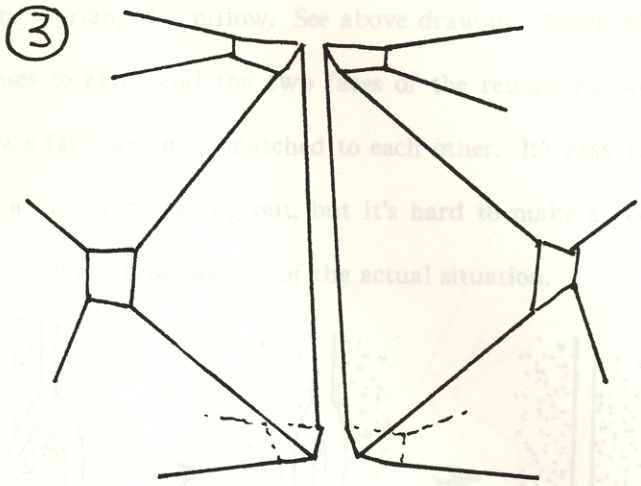
Move #4 decreases the order of two vertices, and increases the order of two others, so it's important to choose carefully where to apply it. It should be applied so that one of the vertices whose order is reduced is already a vertex of minimal order. (And the other one should

have order as small as circumstances will permit.) This guarantees that the decomposition will terminate in a finite number of steps: each application of Move #3 decreases the number of vertices remaining on the polyhedra, and each application of Move #4 decreases the minimum order of the vertices (getting you one step closer to another application of Move #3). In other words you are never more than a finite number of Move #4's away from another application of Move #3, and it takes only a finite number of Move #3's to complete the decomposition. This argument can be formalized as a proof by induction on the set of order pairs of the form

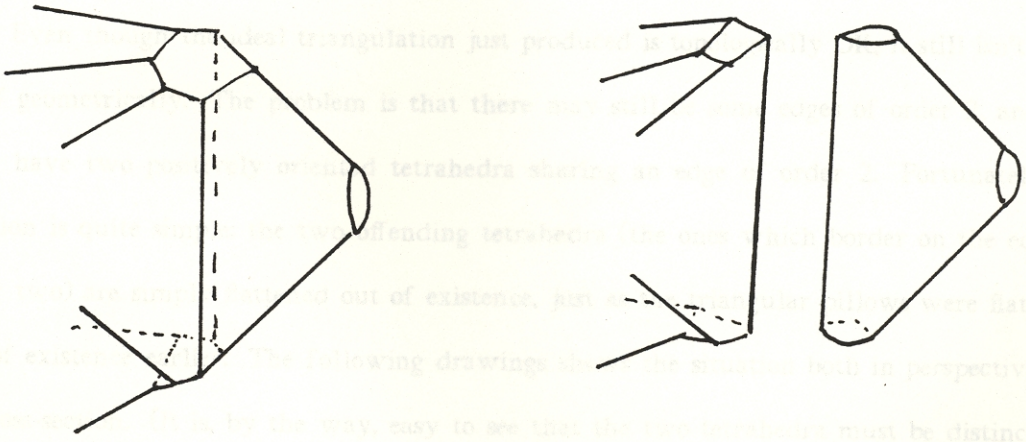
(number of vertices, inf{orders of vertices}),

ordered lexicographically.



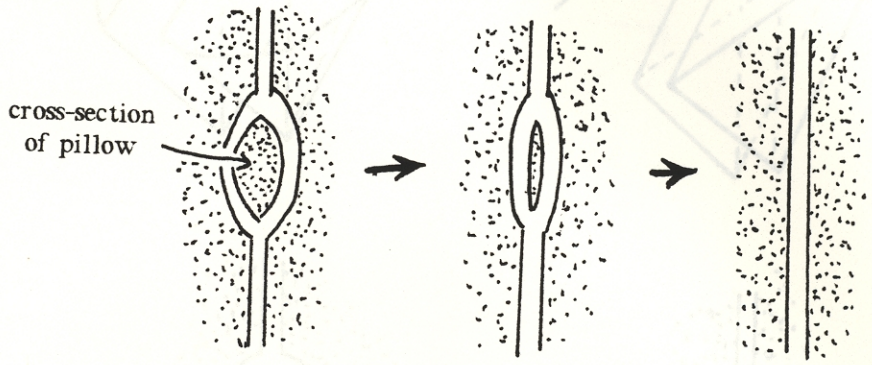


Occasionally Move #4 will try to split a polyhedron into two pieces as shown above. But it won't quite manage it. What you'll be left with is two edges which share the same ideal endpoints (see second drawing above). These edges bound a 'hidden' bigon. Move #1 finds these hidden bigons and eliminates them in the same manner ordinary bigons were eliminated previously. This splits the polyhedron in two, which is the geometrically reasonable thing to do. (As with regular bigons you must, of course, check that the two edges are distinct in the manifold--you might have an embedded cylinder or Mobius strip.)



Move #1 is also the first step in eliminating vertices of order 2. Each vertex of order 2 is surrounded by a hidden bigon, and eliminating this hidden bigon squeezes off a degenerate

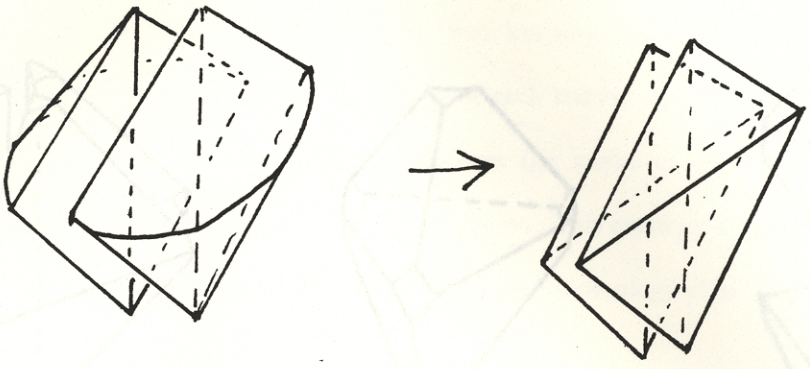
polyhedron resembling a triangular pillow. See above drawing. Move #2 flattens the pillows. The pillow itself ceases to exist, and the two faces of the remaining polyhedra which were matched to the pillow's faces are now matched to each other. It's easy to visualize a triangulated manifold with a pillow flattening out, but it's hard to make a good three-dimensional drawing; instead I've drawn a cross-section of the actual situation.

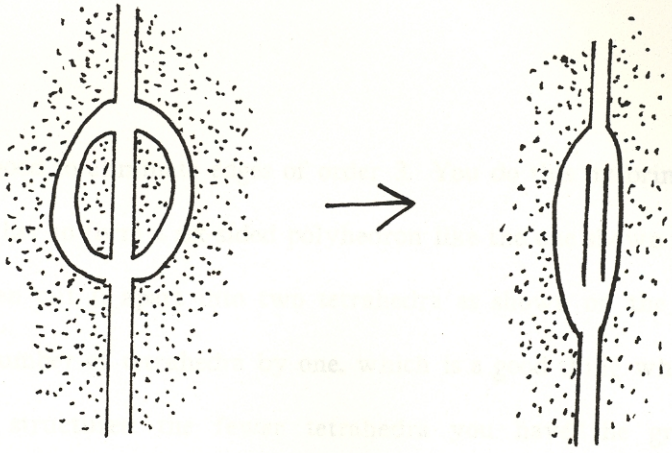


Thus the decomposition algorithm is to apply Moves #1-4 over and over, with the stipulation that no move is to be performed if some lower numbered move can be performed instead. (Moves #1 and #2 are performed before #3 and #4 to protect the later from degenerate configurations.) As mentioned above, this algorithm reduces the polyhedra to a collection of tetrahedra in a finite number of steps.

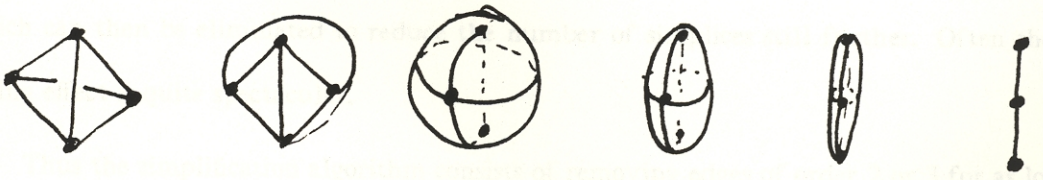
Simplification of the ideal triangulation.

Even though the ideal triangulation just produced is topologically OK, it still isn't up to snuff geometrically. The problem is that there may still be some edges of order 2, and you can't have two positively oriented tetrahedra sharing an edge of order 2. Fortunately the solution is quite simple: the two offending tetrahedra (the ones which border on the edge of order two) are simply flattened out of existence, just as the triangular pillows were flattened out of existence earlier. The following drawings shows the situation both in perspective and in cross-section. (It is, by the way, easy to see that the two tetrahedra must be distinct: the only time they can be the same is when you have an ideal tetrahedron whose faces are glued to each other in such a way as to form the lens space L(4,1) with a point removed, and in this case the link of the ideal vertex is a sphere not a torus.)



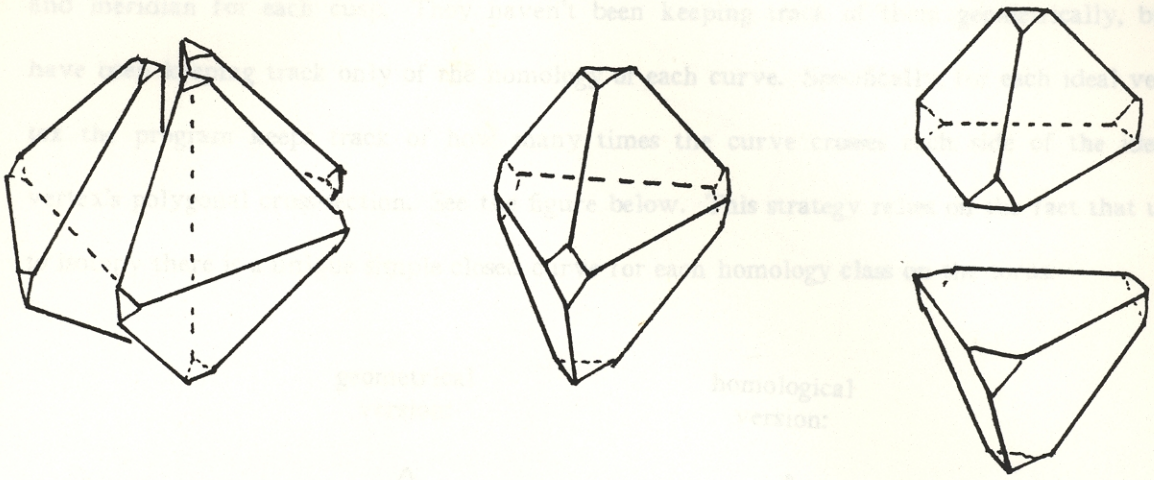


Note that flattening out the two tetrahedra reduces the order of a bunch of other edges. In principle this operation can produce an edge of order one. The problem can be dealt with by collapsing the offending simplex as shown in the following drawings:



Several adjacent simplices will flatten out as well. Fortunately edges of order one do not arise in practice (unless you go out of your way to make trouble, for example by feeding the

program complicated projections of the unknot), so I have not implemented a routine to deal with them.



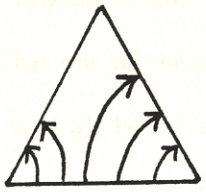
It is also good to eliminate edges of order 3. You do this by bringing the three relevant tetrahedra together to form a six-sided polyhedron like the one shown on the left in the above figure. You then cut it apart into two tetrahedra as shown on the right. Thus you have decreased the number of tetrahedra by one, which is a good thing when you later go looking for hyperbolic structures: the fewer tetrahedra you have the greater the volume per tetrahedron, and the more regular each tetrahedron will be (as you might expect, the computational methods work best when the tetrahedra aren't too far from regular). If the three original tetrahedra are not distinct, then this operation is not possible and you are stuck with the edge of order 3 (in practice this isn't a problem). Note that when you eliminate one edge of order 3 you reduce the orders of certain other edges. This may produce new edges of order 3, which can then be eliminated to reduce the number of simplices still further. Often the cascading effect is quite spectacular.

Thus the simplification algorithm consists of removing edges of order 2 or 3 for as long as possible. When both moves are possible, edges of order 2 are removed first.

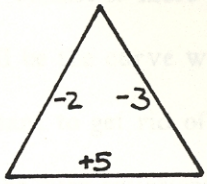
Selection of the preferred longitude and meridian for each cusp.

All the above algorithms have been secretly keeping track of the preferred longitude and meridian for each cusp. They haven't been keeping track of them geometrically, but have been keeping track only of the homology of each curve. Specifically, for each ideal vertex the program keeps track of how many times the curve crosses each side of the ideal vertex's polygonal cross-section. See the figure below. This strategy relies on the fact that up to isotopy there is a unique simple closed curve for each homology class on the torus.

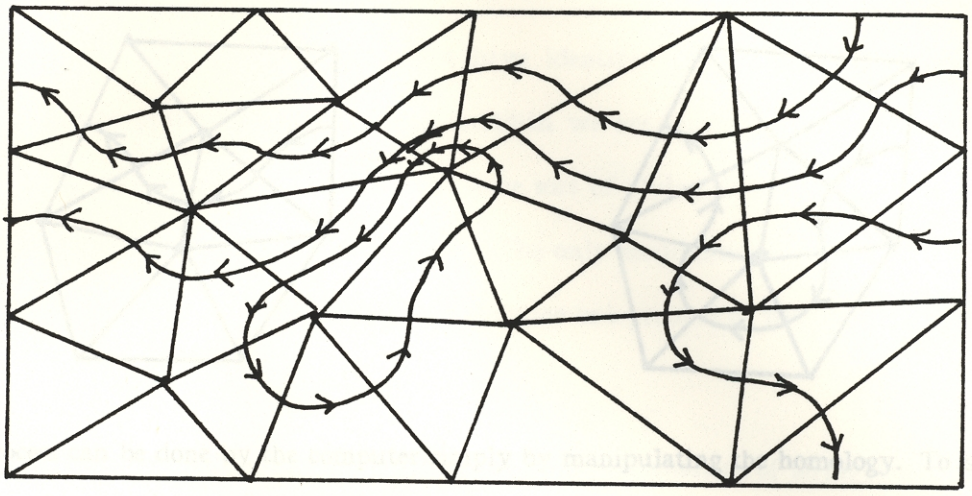
geometrical version:



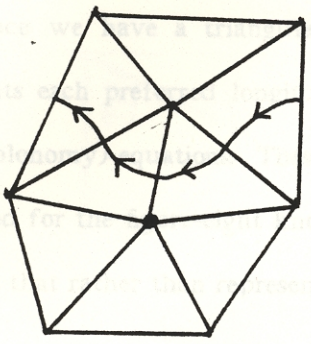
homological version:

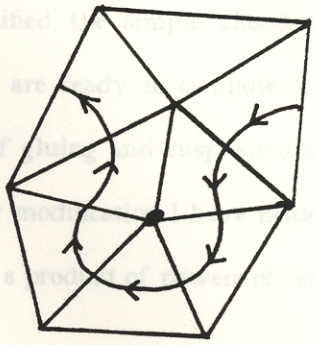


The idea for reconstructing the curve is the following. Say, for sake of discussion, that one edge is marked with a positive number, say +5, while the other two are marked with negative numbers, say -2 and -3. (The numbers must sum to zero since the curves leaves the triangle as often as it enters.) Draw five curve segments entering the triangle from the +5 edge. Two of them bend left and leave the triangle via the -2 edge, while the remaining three bending right and left via the -3 edge. Do this for the links of all the ideal vertices, let the segments join up as shown below, and you have a curve in the appropriate homology class.

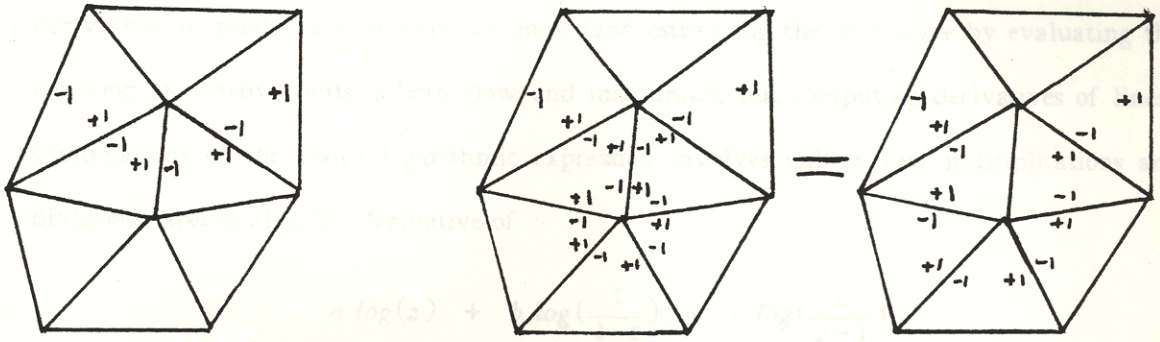


The catch is that this curve can—and often does—consist of more than one component. It is not hard to deduce that one of the components will be the curve we are looking for, while the other components will all be trivial. (It's important to get rid of the trivial components because they will introduce stray factors of 2π into the cusp equations.) Computationally it is surprisingly easy to pick out the nontrivial component, both in terms of the programmer's effort and the machine's effort. First copy out one component of the curve. If this component passes through any edge of any triangle more than once you know right away that this is the simple closed curve you are looking for. The reason is that the region on the left side of the curve is the same as the region on the right side, i.e. the curve is nonseparating and therefore nontrivial. However, if the component passes through each edge of each triangle at most once, then we are still unsure whether it is nontrivial or not. To find out which it is, start sliding the component leftward, passing it through one vertex at each step as shown below.





The process can be done by the computer simply by manipulating the homology. To 'slide the curve leftward' the computer adds in a circle of +1's and -1's as shown below. (If you reconstruct the curve at various intermediate stages you may find that it has split into several subcomponents, but this is OK. All we care about now is whether the component we started with is homologically trivial or not.)



If the component was trivial, then after a number of steps equal to the number of vertices in the triangulation of the torus there will be no curve left at all! (Proof: At each stage think of geometrically reconstructing the curve by the method mentioned above. If the curve was trivial--and therefore separating--then at each stage the region to its left will contain one less vertex.) If the component is nontrivial we will be left with a mess, but at least we'll know the component was nontrivial, and if we had the foresight to save a copy of the component we will be done.

Selection of a nonredundant set of gluing equations.

Once we have a triangulation and have identified the simple closed curves which represents each preferred longitude and meridian we are ready to compute the gluing and cusp (holonomy) equations. These are the same sort of gluing and cusp equations which are computed for the figure-eight knot in [TH2]. The only modification I have made to the usual set-up is that rather than representing each equation as a product of powers of terms like

$$z \qquad \frac{1}{1-z} \qquad \frac{z}{z-1}$$

I take the log of everything so I can work with linear combinations of terms like

$$\log(z) \qquad \log(\frac{1}{1-z}) \qquad \log(\frac{z}{z-1})$$

instead. This scheme offers two main advantages:

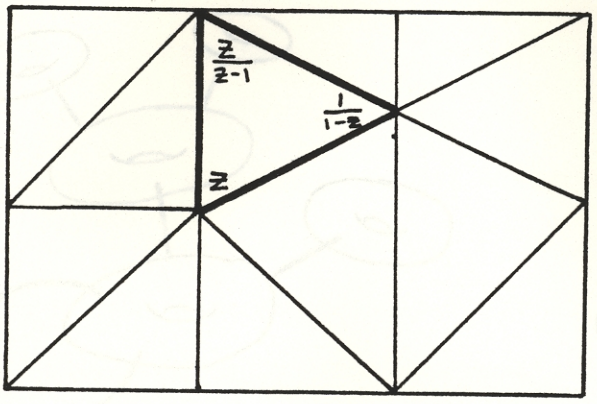
(1) The program which solves these equations uses Newton's method, and therefore needs to know the derivatives of the left-hand sides of the gluing and cusp equations. Computing derivatives of products of powers is a mess (and estimating the derivative by evaluating the function at nearby points is both slow and inaccurate), but computing derivatives of linear combinations of the above logarithmic expression involves only a few multiplications and divisions. Specifically, the derivative of

$$a\ \log(z) \ + \ b\ \log(\frac{1}{1-z}) \ + \ c\ \log(\frac{z}{z-1})$$

is

$$a\ \frac{1}{z} \ + \ b\ \frac{1}{1-z} \ + \ c\ \frac{1}{z\,(1-z)}$$

(2) The condition that the dihedral angles surrounding an edge in the manifold sum to 2π is checked automatically. Similarly, in working with the holonomies a rotation through an angle of, say, 2π is automatically distinguished from a rotation through an angle of 0.



$$\log(z) + \log(\frac{1}{1-z}) + \log(\frac{z}{z-1})$$

$$= 0 + \pi i$$

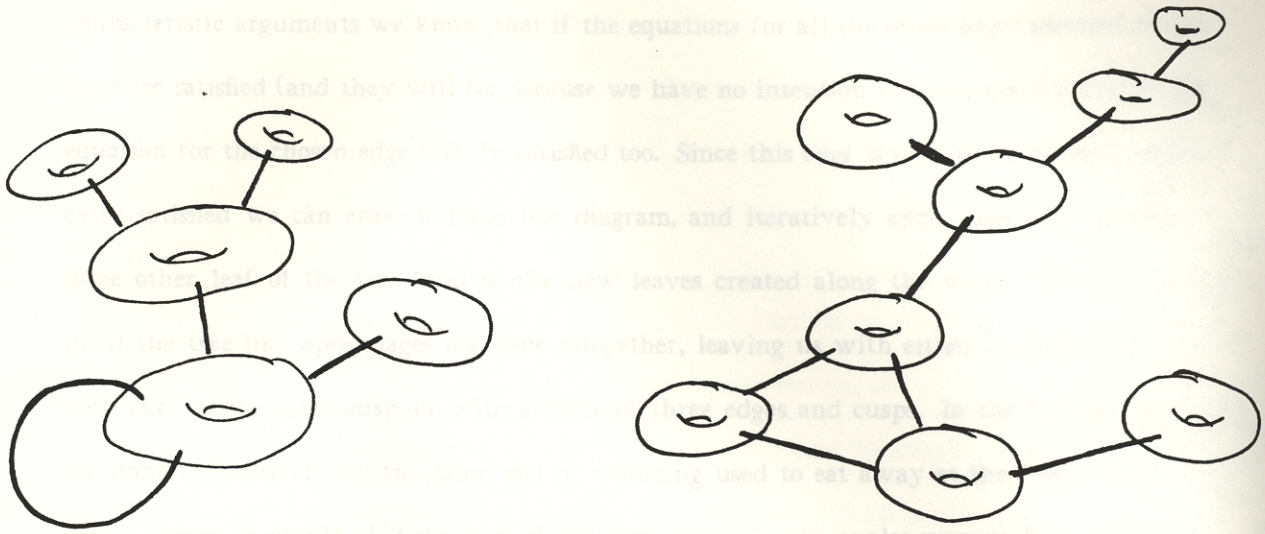
Anyhow . . . the main business of this section is to select a set of nonredundant gluing equations. To see why the gluing equations are redundant, think of each cusp as a torus which has been triangulated. The triangles in the triangulation are the triangular links of the ideal vertices which comprise the cusp. The sum of the logs of the cross-ratios of the three angles of each triangle add to $0 + \pi i$. So the sum of the logs of the cross-ratios of *all* the angles in the triangulation of the torus is $f\ \pi i$ (where f is the number of faces in the triangulation of the torus, e is the number of edges, and v is the number of vertices). But we also know that

$$v - e + f = 0$$

and

$$e = \frac{3}{2} f ,$$

so therefore $f = 2v$. Thus the sum of all the cross-ratios in the triangulation of the torus, namely $f\ \pi i$, can be written as $2v\ \pi i$. Each vertex in the triangulation of the torus corresponds to an edge in the ideal triangulation of the 3-manifold itself. So the fact that all the cross-ratios sum to v times $2\pi i$ means that if the gluing equations are satisfied for all but one of the edges incident to a given cusp, then the gluing equation for the remaining edge must be satisfied also.



If there are k cusps, then we will want to eliminate k redundant equations. We can't of course, just eliminate any k equations! We'll choose a collection of edges which are incident to the cusps in one of the two patterns shown above, and eliminate the equations corresponding to those edges. First I'll say why such a collection of edges can always be found, and then I'll explain how the corresponding equations can be safely eliminated.

Why the collection of edges can always be found: First we need to see that we can always find either an edge with both ends incident to the same cusp (as on the left above) or a cycle of three edges and cusps (as on the right). This is easy. Look at any face of any ideal tetrahedron. If its three ideal vertices are incident to three distinct cusps we have a cycle of order three; otherwise two (necessarily adjacent!) vertices will be incident to the same cusp and we'll have an edge looping back to the cusp it started at. Constructing the tree-like appendages is then straightforward: look for an edge with one end incident to what you've already constructed and the other end free, and include that edge and the cusp incident to its other end. Repeat until all cusps are included. (This procedure breaks down for split links, but since split links don't admit hyperbolic structures we don't care.)

How the equations can be eliminated: Start at a leaf of one of the tree-like appendages. There is only one chosen edge incident to it, and we want to check that the equation corresponding to this edge can be safely eliminated. This is no problem: from the above Euler

characteristic arguments we know that if the equations for all the other edges incident to this cusp are satisfied (and they will be, because we have no intention to eliminate them) then the equation for the chosen edge will be satisfied too. Since this edge is now know to be automatically satisfied we can erase it from our diagram, and iteratively apply this same process to some other leaf of the tree (and to the new leaves created along the way). We keep going until the tree-like appendages are gone altogether, leaving us with either a single edge with both ends at the same cusp, or with a cycle of three edges and cusps. In the former case we are done immediately by the same sort of reasoning used to eat away at the trees. The later case is almost as simple. Let the sum of the cross-ratios for the angles surrounding each of the three edges be represented by a, b and c. The Euler characteristic arguments from above give us a linear equation for each cusp which relates the sum of the cross-ratios for the two edges incident to that cusp. Since there are three cusps we get three equations:

$$\begin{aligned} a + b &= 4\pi i \\ a + c &= 4\pi i \\ b + c &= 4\pi i . \end{aligned}$$

These equations have the unique solution $a = b = c = 2\pi i$, i.e. the remaining three equations must be satisfied. This completes the proof that in a manifold with k cusps, there are k gluing equations which are automatically satisfied whenever the remaining gluing equations are, and can therefore be safely eliminated.

I'd like to conclude this section by checking that the number of equations will always be equal to the number of variables. Say our manifold has k cusps, and has been given a triangulation with n ideal tetrahedra. The number of variables is obviously n, one for each tetrahedron. Originally there is one gluing equation for each edge in the ideal triangulation, and by an Euler characteristic argument the number of edges equals the number of tetrahedra:

$$v - e + f - n = 0$$

$$0 - e + (2n) - n = 0$$

$$e = n.$$

As we saw above, though, k of the gluing equations will be redundant and can be eliminated, leaving us with n - k equations in n variables. But then we have k cusps equations, bringing us back to n equations in n variables.

Solving the equations

The computer program SNAPPEA uses Newton's method to solve the gluing and cusp equations. (As explained at the beginning of the previous section, the program works with the logarithms of the usual equations.) Normally the program uses Newton's method in the most straightforward way, but there is a parameter--called 'care'--which can be used to find solutions in more difficult cases. If, for example, care is set equal to two, then at each iteration the solution will be changed by only half the amount prescribed by Newton's methods. If care were equal to three, the solution would be changed by only one-third the amount, etc.

Normally care equals one, and Newton's method runs unaltered. When the program solves for a complete hyperbolic structure it first tries care = 1, and only if that fails does it try again with care = 2. It has never failed to find a complete solution with care = 2, but if it did one could modify the program to try care = 4. When doing Dehn surgeries the user controls the value of care via the 'care' command.

SNAPPEA has another feature for finding solution solutions at hard-to-get-to points in the Dehn surgery space. If it can't find a solution directly it divides the interval between the current point and the next point into ten subintervals, and looks for a solution at each intermediate point in turn. In addition, the last subinterval is itself divided into three subsubintervals.

The final tool for getting at difficult points is the analytic continuation feature, which allows the cross-ratios of the simplices to be analytically continued beyond the default range of $-\pi/2$ to $3\pi/2$. Such solutions, even though they are perfectly valid solutions to the equations, may or may not be geometrically meaningful in the sense that they may or may not correspond to hyperbolic structures. Similar caution must be shown with all solutions con-

taining negatively oriented simplices, even when the arguments fall within the default range. (Although as a practical matter simplices which have just barely gone negative are unlikely to cause trouble--they indicate that the triangulation has just started to fold back and overlap itself. You don't get into trouble until something crosses the singular set.)

There may be ways to deal with the problem of negatively oriented simplices. One promising approach is for the program to monitor the shapes of the simplices as you move from one point in the Dehn surgery space to another. If some simplex starts to flatten out--indicating that it may become negatively oriented--then the program does Whitehead moves which change the triangulation in such a way as to avoid having a flattened simplex. This scheme ought to work well at points where the volume is greater than zero, but it would of course go wild at the boundary of the hyperbolic region.