knotEd
A program for studying knot theory

John Mount

Hewlett Packard, mailstop 44uk

19447 Pruneridge Avenue

Cupertino, California 95014

February 1989

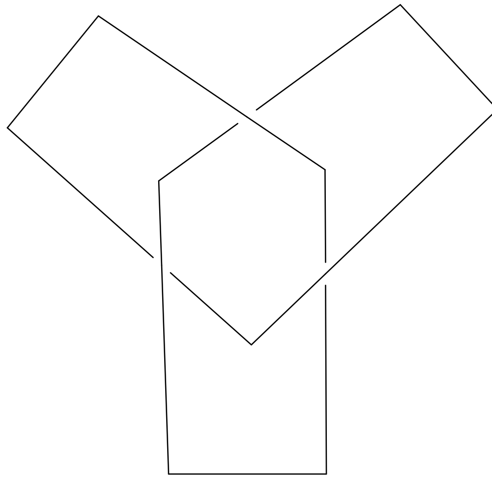
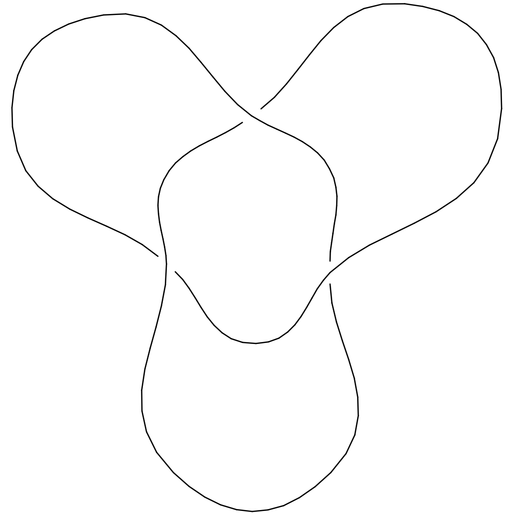
DRAFT

1. Elementary knot theory, a brief introduction

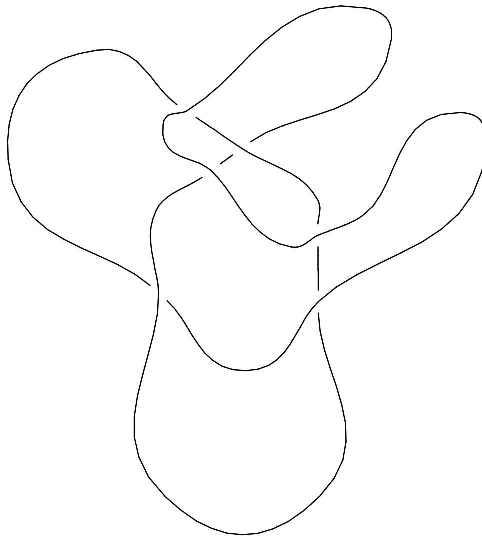
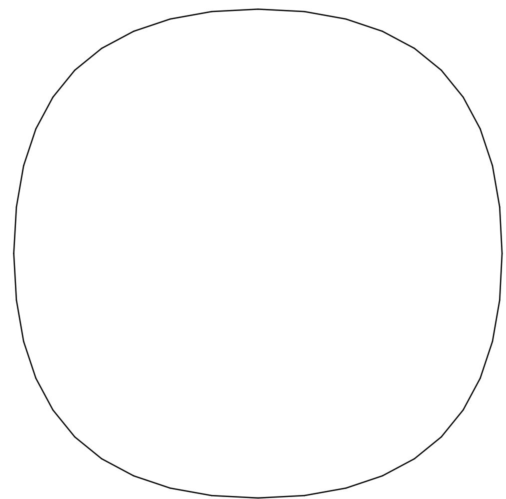
The theory of knots has had constantly waxing and waning popularity. The popularity knots have enjoyed is most likely due to the fact that knot theory really is the theory of knots: twisted and linked pieces of string. Also knots were a proving ground for a lot of the early work in topology. The central question of knot theory is "when do two diagrams represent the same knot?" To answer this question we first must define some terms.

A knot is always a piece of string with both ends attached (if the ends were not attached there would be no theory, as any piece of string can be stretched straight, but not all knots are equivalent to a simple loop). The first point to be made is that all knots discussed here will be "tame knots". A "tame knot" is a piece of string that has only a finite amount of twisting. Tameness is a property shared by all knots tied in actual string (since all real string has non-zero thickness and finite length). The mathematical way to approach this is to study only knots that are built by connecting a finite number of line segments (when altering a knot we treat these as not being able to pass through each other and having thickness) in 3-space (this is also called a simplicial approximation). Such a stiff definition of a knot has the additional advantage that it is easy to draw a diagram representing the knot. The knot in 3-space is simply projected onto a plane. The resulting shadow is then a collection of line segments (some possibly crossing). Now since the knot is made of a finite number of segments it is easy to see that there are only a finite number of points on the projection where lines cross, it is also true that with a slight change in the angle of the projection we can break a crossing that involves 3 or more line segments into several crossings involving only 2 line segments. Furthermore, since everything is finite, it is alway possible to find a projection such that all crossings involve only 2 line segments. These crossings can then be drawn such that we can see which segment passes under which. An example of the diagram of a simple knot, called the trefoil, can be seen on the left. It is customary to ignore the fact that knots are polygons and draw the figures in the more relaxed fashion of the one on the right.

DRAFT

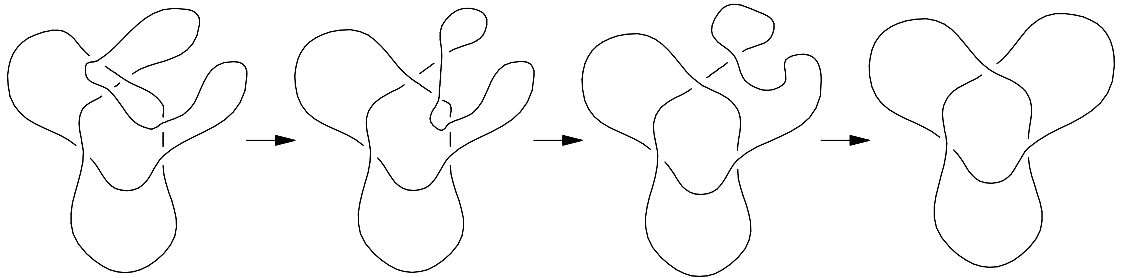



As we said the central question is determining when two diagrams represent the same knot. A concrete example would be to prove that one of the following diagrams is equivalent to the trefoil pictured above and that one is not.

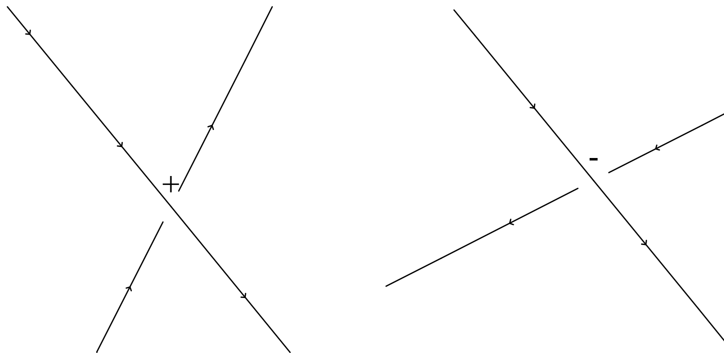



The knot on the left can be deformed (without allowing pieces to pass through each other) into the trefoil in three steps (illustrated below). Reidemeister proved that two diagrams represent the same knot if and only if they could be deformed into one another using his 3 different types of

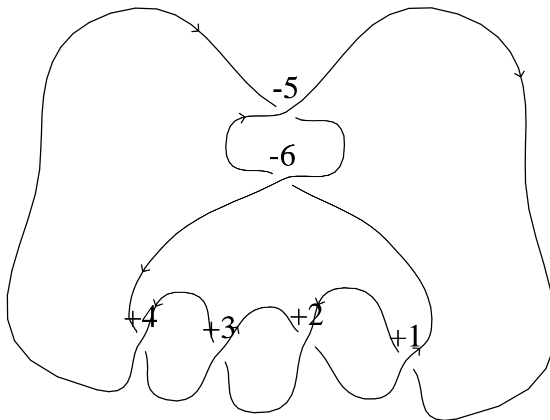
Reidemeister moves (and their inverses). The moves are demonstrated as we fix the trefoil. First the string is pulled over a crossing (Reidemeister move number 3) then the string is pulled off another string (Reidemeister move number 2) and finally the spurious loop is removed from the string (Reidemeister move number 1).



Two diagrams that can be deformed into each other obviously represent the same knot (since none of the Reidemeister moves require a piece of string to pass through another piece of string) but the usefulness of these moves is that Reidemeister proved that two diagrams represent the same knot only if they can be deformed into one another with the Reidemeister moves. This theorem allows us to study knots without using any topology. In fact knot theory can be reduced to a grammar problem in the following manner: First label the n crossings in a given knot diagram with the labels 1 through n. Then mark an arbitrary (but consistent) directional arrow on all of the string and give each crossing a sign of "+" if the top string would be to point to the right if you were standing on the crossing facing in the direction of the bottom string, else give the crossing a sign of "-". Signed crossings are demonstrated below:



Now walk along the knot one time and each time you encounter a crossing call out the sign, the label, and whether you are on the top or bottom level. Thus the following knot could be marked as shown and would yield the sentence: "+1down to +2up to +3down to +4up to -5down to -6up to +4down to +3up to +2down to +1up to -6down to -5up".



Then the Reidemeister moves could be rephrased as some kind of context sensitive grammar. Unfortunately, like many grammar problems, no algorithm is known for generating a sequence of Reidemeister moves to transform one knot into another. And because the number of crossings do not help determine an upper bound on the number of Reidemeister transformations required brute force searching is not an effective method (it merely shows that the problem of determining if two diagrams are knot-isotopy problem is no harder than the halting problem, which is not saying much).

Despite this the Reidemeister moves are very useful in knot theory. The most common use of them is to develop invariants. Many papers present algorithms that given a diagram calculate a polynomial from that diagram. If the method of calculation is unaffected by all 3 Reidemeister moves then it is easy to see that if two diagrams have different polynomials they do indeed represent different knots (though the converse is often not true, nobody has yet found a simple invariant that proves diagram equivalence). Many polynomials are able to differentiate the trivial loop from the trefoil.

DRAFT

2. Design objectives

The design objectives for knotEd were to supply the user with an easy way to draw and alter a knot. By drawing a knot we mean to enter a knot into the computer in such a way so that the user can control the appearance and at the same time the computer understands the semantics of the knot being drawn. This is the main point of departure for knotEd from graphic editors like gremlin, xfig and others. If the software knows what knot the user has drawn it can automatically generate the sentence describing the knot (as discussed in section 1) which can then be taken into programs that automatically calculate invariants. For a human to generate the sentence from a diagram is both tedious and error prone and many of the algorithms for calculating invariants require time exponential in the number of crossings in a diagram. Programs to calculate invariants (and other things) from a sentence have been developed by several graduate students and professors of the University of California at Berkeley. knotEd has been used in conjunction with several of them. It is the intent of the author that if knotEd is released to the mathematics community that it should be bundled and integrate with these programs (some of the algorithms are extremely sophisticated). Another feature that could leverage off a program that knew what knot a user's diagram represented is an idea call an isotopy lock. A user could activate the lock and from then on the program would only allow alterations to the knot that were obviously reducible to a sequence of Reidemeister moves. Thus the knot editor could be used as an intelligent chalk board for educational purposes, or if the user saved all of the intermediate diagrams the editor could automate some aspects of demonstrating the equivalence of two knots. Thus an automated illustrator would lead to a semi automated theorem prover (in the limited realm of knot theory). Additional applications envisioned for the knot editor include aiding in the preparation of papers (all diagrams in this paper were produced by knotEd) and also as a teaching aid.

The biggest question was how the program should interact with the user. Several different operating metaphors were considered, the one finally settled on we call "non-physical". This name is derived from the fact that many of the models brought forth involved realistic 3 dimensional perspective and physics. One of the most popular with the electrostatic model where a knot would be thought of as being a collection of rigid

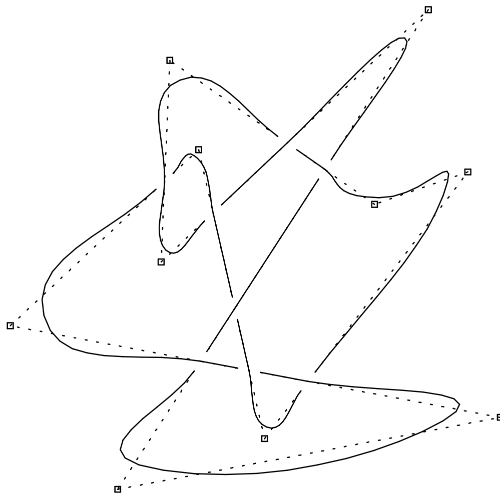
tubes sitting in 3-space with balls where they joined. The balls would carry electrostatic charges and the tubes ends could move around on the surface of the balls. The configuration would then move around or "relax" until it had reached the lowest energy configuration and would thus (when drawn in perspective) give a "pleasant" looking knot. The user could change his view point and alter the knot by removing a sequence of tubes and replacing them with another. The major drawbacks of this hypothetical model were the computation, the difficulty the user would encounter in specifying a path in 3-space and the dependence on so much of knot theory on actual diagrams. As discussed in section 1 a knot is reduced to a sentence by examining the crossings on its diagram. But a knot sitting in three space has no crossings, the lines appear to cross on our 2 dimensional projections but never cross in 3-space. It was felt that with the given resources it would be impossible to implement such a model and to do so would be contrary to how knots are thought of in mathematics (thus making the program a burden instead of a tool).

Another model considered was a two plan model. Again the knot would be thought as ball and pipe affair but this time all of the pipes would be trapped in two planes (one slightly above the other) with vertical rods connecting the pipes in the two planes. The user would then specify which pipes were to be attached where. This combined a physical realization of knot in 3-space with the diagrams because the program could associate a unique diagram with the tubeworks by viewing the knot from above. This model does not seem to have any major defects except that specifying pipes could become quite tedious.

The model selected was derived by reading numerous books and articles on knots and observing how diagrams were drawn freehand and what properties of diagrams were actually used in theorems. As alluded to before the 3-space realization of a knot is of little use when working on a knot. The important relationship is not between a diagram and its 3-space realization but between the diagram and its sentence. The diagram should serve as an aid in altering the knot sentence. In keeping with this the knot is realized in knotEd as a graph with vertices that all join either 2 or 4 edges. The 2 valent vertices are called control points (the user may add, delete or move them around) and the 4 valent vertices are the crossings (they can be thought of as vertices that have additional state information as to which edge passes over and which edge passes under). The user can

remove, move, or replace any sequence of control points and the program will then automatically place the crossings in the correct locations. Then the program tries to associate these new crossings with the one in the previous knot so that they can inherit the state (which edge is up etc) from them. The user can imaging that the edges are passing over and under (like in the pipe model) but is not troubled about details in fitting them together.

What the user actually sees while working is exactly like the diagrams in this paper (knotEd is a what you see is what you get program). Though the user can turn on additional display features (such as marking control points, etc). The next drawing shows a knot as the user of the program might see it while working on it. The boxes represent the control points and the dotted lines represent the actual lines of the diagram. The knot is based off these lines and not the smooth curves because finding the intersections of these curves would involve solving simultaneous 3rd degree polynomials (possible but extremely messy). As you can see the program often generates a handsome diagram from a small number of control points.



3. Implementation

The program was conceived by Professor David Goldschmidt of the Berkeley mathematics department during a demonstration by the author of a previous X application. The author then proceeded to design and implement the program under the direction of Professor Goldschmidt and with the approval of Professor Richard Fateman who had referred the author to Professor Goldschmidt. The bulk of the work was done during the Spring 1988 semester.

The tools originally available included Sun 3/50 workstations with monochrome monitors and X version 10. The diagrammatic approach taken fit very the mouse driven workstation very well. X was chosen over SunTools (another window manager used on the math department machines) because of the portability enjoyed by X applications. A deep concern in the implementation was that while may of the mathematics graduate students and faculty regularly used the Sun 3/50 computers they used them mostly for text processing and typesetting. With this in mind much care was take to "bullet proof" the program. All signals are trapped and any sort of catastrophic termination of the program results in the users work be saved in a "panic file". The prototype would even log the disaster and its circumstances in a record file in my account and mail the user a letter describing where they could find their saved work. The log file has since been removed since it was considered intrusive though it did allow the author on several occasions to approach users with "the program booted you out last night, what went wrong?"

The program has since been changed into an X11 application (the X10 version will be allowed to die) and has been expanded to include color support. This feature is especially useful when dealing with multiple knots that are tangled together, though the program is fully functional on monochrome workstations.

The original hardcopy was produced by emitting Tektronics drawing commands and filtering these through a Postscript translator. This cumbersome method was used instead of dumping a bitmap image of the screen to make the image independent of the resolution of the screen (since most monitors are nowhere near the 300 dots per inch pixel density that is common in laser printers) and actually turned out to be a

tremendous performance improvement over dumping bitmaps. The drawings in this paper were emitted directly from the program as PIC commands which were then typeset (along with this text) by troff. Additional back ends are planned. The hardcopy model was made purposefully "stupid"; the program requires only operations to draw hardcopy: the ability to draw a line segment and the ability to draw text at a given location. Erasure is not used to draw the undercrossings.

The isotopy lock is based on a natural generalization of the Reidemeister moves. The reader certainly noticed that the Reidemeister moves took three steps to straighten the mangled trefoil, where simply erasing the area in question and redrawing it would obviously have been legal. In fact it is easy to see that all three Reidemeister moves can be performed by erasing a segment of a knot that involves either crossings that are entirely over or crossings that are entirely under and redrawing the segment anywhere constrained only that it must be entirely over or entirely under the rest of the knot (depending on if it was originally over or under, a segment that didn't cross can be redrawn entirely over or entirely under) or such that it does not cross the rest of the knot at all and that it does not cross itself. Since this move is clearly legal and is able to generate the Reidemeister moves we see that it is necessary and sufficient to generate (by repeated application) all legal knot transformations. Every time the user alters a knot the isotopy lock (if activated) checks that the replaced segment meets the above criteria (actually it relaxes the criteria a little in allowing the replaced segment to ignore trivial self crossings of the type shown in the Reidemeister 1 move). This method was chosen above the method of pointing at a crossing and specifying what Reidemeister move to perform because the Reidemeister moves are in no way natural (they were contrived to prove theorems) and this method of altering the knot would require that the editor have extensive routing capabilities to draw the new segment so that it did not introduce spurious crossings. Also for a mathematician working on a chalk board the "always over or always under" rule appears to be the one they actually use. Thus knotEd would have allowed the user to fix the example trefoil in one step.

The smoothed curves are actually based on two different spline models. The first model treats each line segment as a parameterized arc in 3-space and fits two 3rd degree polynomials to generate the curve. The polynomials are determined such that they match value at the endpoints

and such that the derivative at each endpoint is a line parallel to the line segment formed by drawing a line form the control point preceding the endpoint to the control point succeeding the endpoint (this method of determining the derivatives was inspired by memories of the mean value theorem for derivatives from freshman calculus). The endpoints of the smoothed curve are allowed to miss the control points (they can go to a point determined by the weighted average of the control point and its two immediate neighbors) but they must hit the crossings (since we don't wish to determine where the splines would cross we force it) though the bottom string always stops drawing just before a crossing.

The second model is again a Hermetian spline and picks its derivatives in the same way the first one does. The difference is that this model insists on hitting all control points and that it fits only one polynomial. This is done by rotating the line segment to be splined so that it is horizontal. In this configuration it is not necessary to parameterize and y can be a function of x. The spline is then rotated back into the proper orientation. The second model has the advantage that it does not allow the splines to cross their selves (they may still spuriously cross each other if draw too close) or to form cusps. This model has the disadvantage that a division is used to compute the derivative (deltaY/deltaX) so if deltaX (in the rotated perspective) is small the derivative can become excessively large (causing the curve to run away). The run away can be checked by inserting an extra control point.

4. Future directions

Some plans for knotEd include:

1) Performance enhancement. The routine find_cross checks edges that are already known not to cross every time it is called. It could keep track of this and reduce the time complexity of the routine. Many other calculations could be improved.

2) Kirby Calculus and other difficult operations should be done by the editor. This way the user benefits in (hopefully) three ways where he uses knotEd: basic operations are easy, operations are all checked for legality (isotopy lock) and complex operations are entirely automatic.

3) The ability to merge two stored knots into one. The data structures rely heavily on the absolute location of records in an array (this made find_cross INFINITELY easier) so a little code would be needed here.

4) Machine independent storage. Current method of storing knots uses fwrite to write out records this is machine dependent and unreadable to both humans and other programs. Some simple grammar would be sufficient for this task.

5) More methods of getting hard copy. Currently hardcopy comes only through Tektronics format or saving the screen to a file. The ability to draw in PostScript, MetaFont, or some grammar would be nice.

6) Smarter redraw. Currently we update the whole screen. The ability to redraw portions would improve performance and cut down on annoying flicker.

7) More knot theory. Actually have the program try to reduce knots into a simpler form.

8) More calculations available. Calculating software (ala Goldschmidt, Walker, and Baxter) make knotEd more useful (and vise versa).

9) More knots. Somebody (I don't have the time at the moment) should sit down with Rolfsen's knot theory and draw all the knots in the appendix into the knot library.

10) knotEd ported to other machines. This should be easy as the program I based knotEd on ported easily.

12) Some decent documents.

5. References

A good reference for some of the theory involved in the knot editor is Kaufman's article "New invariants in knot theory", The American Mathematical Monthly, March 1988, p. 195. Rolfsen's book. my X11 manuals. Reidemeister's book. The pic book. The reference from Boyle (if I ever get it). Kaufman's book (on Knots). Armstrong's topology text. The Springer-Verlag topo text. Dara's article. Aaron's article.

Conte- DeBoor numerical analysis text

CONTENTS

1. Elementary knot theory, a brief introduction 1
2. Design objectives 6
3. Implementation 9
4. Future directions 12
5. References 14