A Suggestive Interface for Untangling Mathematical Knots

Huan Liu and Hui Zhang

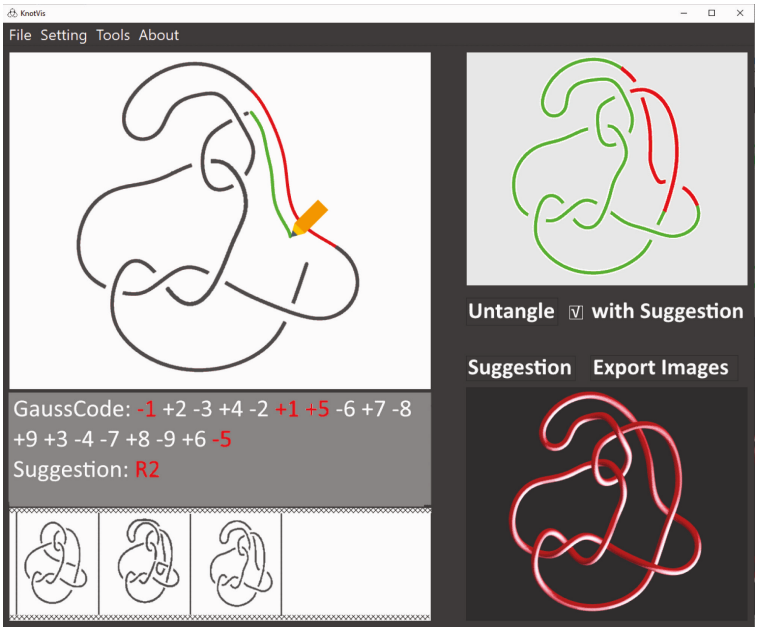


Fig. 1. Our suggestive knot diagram interface. Upper-left — the knot canvas for one to draw, edit, and deform knot diagrams. Upper-right — a visual panel to suggest the Reidemeister moves necessary for one to continue untangling the knot; the knot portion is highlighted in red with text information displayed at the bottom of the knot canvas to facilitate our understanding of the suggestion. Bottom-left — "key moments" captured to summarize all the critical changes that have occurred during the entire deformation. Bottom-right — knot display window that supports 3D and planar knot diagram modes.

Abstract— In this paper we present a user-friendly sketching-based suggestive interface for untangling mathematical knots with complicated structures. Rather than treating mathematical knots as if they were 3D ropes, our interface is designed to assist the user to interact with knots with the right sequence of mathematically legal moves. Our knot interface allows one to sketch and untangle knots by proposing the Reidemeister moves, and can guide the user to untangle mathematical knots to the fewest possible number of crossings by suggesting the moves needed. The system highlights parts of the knot where the Reidemeister moves are applicable, suggests the possible moves, and constrains the user's drawing to legal moves only. This ongoing suggestion is based on a Reidemeister move analyzer, that reads the evolving knot in its Gauss code and predicts the needed Reidemeister moves towards the fewest possible number of crossings. For our principal test case of mathematical knot diagrams, this for the first time permits us to visualize, analyze, and deform them in a mathematical visual interface. In addition, understanding of a fairly long mathematical deformation sequence in our interface can be aided by visual analysis and comparison over the identified "key moments" where only critical changes occur in the sequence. Our knot interface allows users to track and trace mathematical knot deformation with a significantly reduced number of visual frames containing only the Reidemeister moves being applied. All these combine to allow a much cleaner exploratory interface for us to analyze and study mathematical knots and their dynamics in topological space.

Index Terms—Knot theory, Reidemeister moves, Gauss code, Mathematical visualization, Suggestive interface

1 Introduction

Our work in this paper is mainly concerned with the illustration of the topology of mathematical knots, i.e., 1-dimensional strings embedded in 3-space. The main properties of these strings to be visualized and studied in our work, are in their topological space. In topology, a very small circle is the "same" as a huge one, because we can stretch the small one to make it exactly like the big one. More generally, two strings are going to be considered the "same" if we can deform one into the other without cutting. When communicating about mathematical knots, we often draw and view 3D figures to help our perception and understanding of the knots' geometric structures [10]. However, it can be a far more challenging task to communicate about the knot's topology [5, 22], which studies its geometric properties and spatial relations unaffected by the continuous change of shape or size of figures.

Relating two fundamentally identical entities that appear to very different has long been a fascinating challenge problem. For example, the idea of knot equivalence [8, 11, 19, 23] is to give a precise definition of when two knots should be considered the same even when positioned quite differently in space. Even more challenging is to mathematically

- *Huan Liu is with University of Louisville. E-mail: huan.liu@louisville.edu.*
- *Hui Zhang (corresponding author) is with University of Louisville. E-mail: hui.zhang@louisville.edu.*

Manuscript received xx xxx. 201x; accepted xx xxx. 201x. Date of Publication xx xxx. 201x; date of current version xx xxx. 201x. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org. Digital Object Identifier: xx.xxxx/TVCG.201x.xxxxxxx

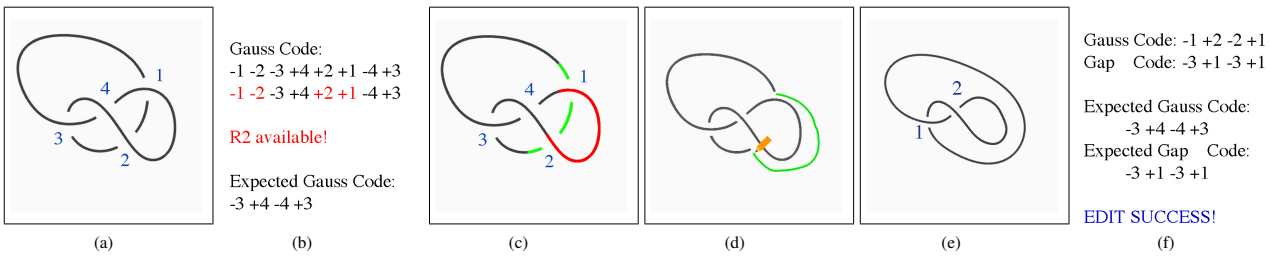
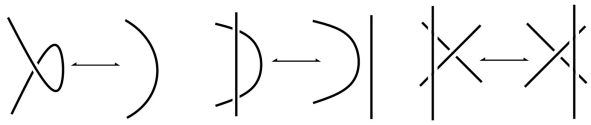


Fig. 2. Our core design idea is to equip the sketching based knot diagram interface ((a), (c), (d), and (e)) with a "mathematician's brain" — a numerical analysis kernel to suggest and validate every single move to apply to untangle the knot ((b) and (f)).

move and deform knots towards the fewest possible number of crossings through a sequence of Reidemeister moves (see e.g., Fig. 3), the three core moves necessary to fully untangle a knot [6, 18].

Our task in this paper is to show how the combination of computer graphics, numerical methods, and suggestive visual interfaces can assist us in untangling mathematical knots with the Reidemeister moves in Fig.3. This paper starts from simple tasks such as drawing a mathematical knot in our canvas, and reconstructing a knot from a pre-rendered knot diagram raster image. Having established the basic mechanisms and intuition of this artifice, we proceed to showing how a numerical method can be constructed to read the three-dimensional geometry of a knot in its Gauss code notation and suggest the Reidemeister moves required to untangle the knot to the fewest possible number of crossings. We then translate the numerical results into a suggestive visual interface that enables the intuitive and guided user experience with mathematical knots. We showcase examples and their visual proofs generated from our knot interface. By exploiting such a combination of visual interface and mathematical analysis, we feel that we can make a novel contribution to building intuition about classes of geometric and topological problems in knot theory.



(a) R1 (b) R2 (c) R3

Fig. 3. The three Reidemeister Moves. (a) Twist and untwist. (b) Move one loop over another. (c) Move a string over or under a crossing.

2 Related Work

The idea of visualizing mathematical knots has developed in many directions with the recent advances in computer graphics and computer interfaces. Carter generates nicely rendered figures for many complicated yet beautiful examples in modern topology [1]. Weeks' SnapPea software displays and manipulates the over/under crossings of mathematical knots [27]. Some combine haptic interfaces and 3D graphics to represent mathematical curves in 3D and knotted surfaces in 4-dimensional space (see, e.g., Phillips [17], Spillmann [24], and Zhang [28, 29]). Fish proposes a novel use of visual algebraic proofs as a means of representing certain algebraic proofs for unknot detection [3]. Kurlin introduces simple codes and fast visualization tools for knotted structures in molecules and neural networks [12].

Visualization efforts focused on untangling knots typically employ a physical or pseudo-physical relaxation approach. For example, Scharein's *Knotplot* has been widely used to construct and relax 3D mathematical knots using an energy model [20]. Ladd makes combined use of energy minimization and randomized tree-based planning [13]. These methods work completely in 3D scenario, which are different from how mathematical knots are deformed and untangled with the Reidemeister moves. A few other efforts have focused on the use of a sketching interface for proposing knot deformation. For example, Zhang's *KnotPad* [30, 31] proposes a knot diagram sketching interface that only allows the Reidemeister moves that can be drawn by (mostly expert) users. However, proposing the right sequence of Reidemeister moves necessary to untangle a complex knot in the sketching interface is very difficult for most non-expert users. We thus have been motivated to investigate the possibility of building a suggestive interface that can possibly understand how the knot is being changed, suggest and validate the next mathematical moves towards the fewest possible number of crossings. The paper aims at answering this question — "How can we build a knot interface that can understand and communicate the mathematical moves necessary to fully untangle a knot?"

3 Overview of Our Suggestive Knot Interface

The core idea driving the development of our system (Fig.2) is to allow the user to sketch and apply the Reidemeister moves to untangle mathematical knots in a suggestive interface, where user's sketching process can be guided if desired. The implemented interface consists of four major areas as shown in Fig.1: 1) a major sketching canvas on the upper-left corner of the interface, 2) a Reidemeister move instruction window on the upper-right corner, to suggest and validate users' Reidemeister moves while the sketching process evolves, 3) a "key moments" window on the bottom-left corner, to capture and list all the critical changes that have occurred during the entire knot deformation, and 4) the bottom-right window for users to view knots in 2D/3D.

Our interface is a data-driven approach which automatically generates 3D mathematical knot structures while the sketching evolves. In the sketching canvas, one can create and edit knot diagrams with a "virtual pen" and the system automatically queries the user for an explicit choice of over/under- crossing to ensure no part of the curve runs into another in 3D. The user can push and pull part of the curve for further geometry control, and the result can be rendered as a classic knot-crossing diagram or a smooth 3D knot figure.

At each step when untangling the knot, our interface can suggest the Reidemeister moves necessary to continue simplifying the knot by utilizing a Reidemeister move analyzer behind the visual interface, that reads the two-dimensional projection of the knot in its Gauss code notation and determines how further Reidemeister moves may be applied to fully untangle the knot. The user can apply the suggested moves from the interface or propose his or her own Reidemeister moves by explicitly editing the knot. Our interface is designed to mathematically validate each move performed by the user and alert the user when illegal attempts are recognized, and will only accept legal and validated moves to change the knot. In addition, our interface can recognize and "remember" each valid move to transform one knot diagram to another, therefor one can take advantage of such a memory system behind our interface to track and trace the sequence of the Reidemeister moves generated during the entire interactions.

4 Details of Implementation Models

In this section, we describe the families of models used to implement the interaction procedures, visual elements, and our Reidemeister move kernel engine behind the suggestive interface. Our fundamental techniques are based on a wide variety of prior art, including sketching

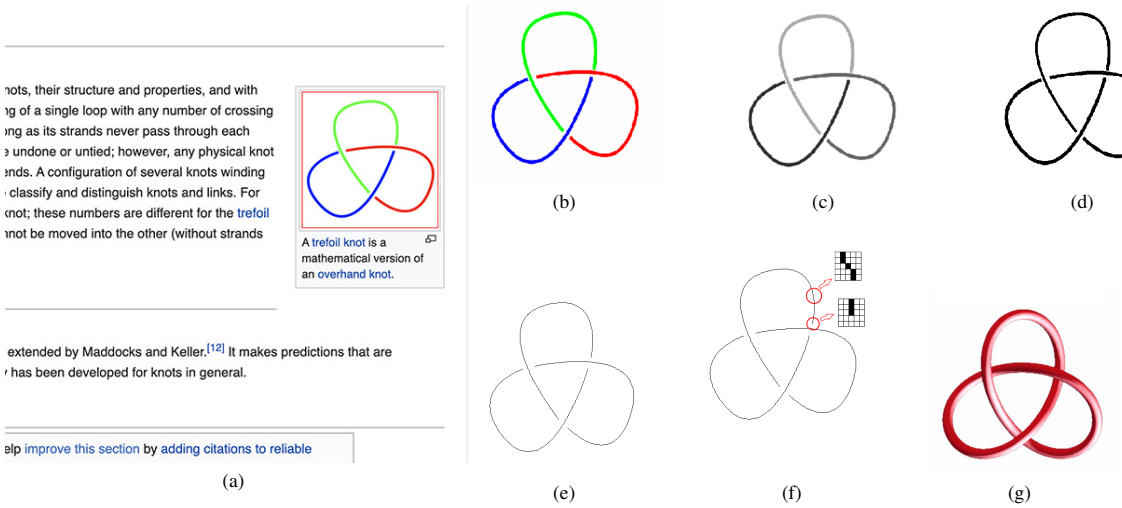


Fig. 4. Reconstructing a Trefoil knot from a knot diagram raster image available at Wikipedia. (a) Crop the knot diagram image. (b) Resize the image. (c) Turn into a gray-scale image. (d) Turn into a binary image. (e) Thinning — convert binary image into 1-pixel wide lines. (f) Crossing point identification and 3D reconstruction. (g) The resultant 3D smooth knot with color and shading.

interfaces focusing on the creation of 3D objects [9, 15], the suggestive interfaces used in 3D design and analysis [2, 14, 26], and other variants on computer graphics and visual interfaces for mathematical visualization including, e.g., the work of [7, 21, 25, 30].

4.1 Knot Creation

We first focus on the creation of 2D knot diagrams of 3D mathematical knots. The core methods and visual elements of our suggestive knot diagram interface will be treated in later sections.

Constructing Knots From Raster Images. The diagram we render to represent the knot in Fig.2 is called knot diagram, which is the planar representation of the three-dimensional knot, with additional information, i.e., over/under-crossing information, recorded by means of visual breaks in the arcs. Such two-dimensional representations are widely rendered in printed and digital media when mathematical knots are described (see e.g., Fig.4(a), the trefoil knot diagram at Wikipedia). It is possible to computationally recognize and reconstruct the three-dimensional knot structure from a knot diagram raster image with the following key steps:

- *Crop and Resize.* For example, the user can crop down a knot diagram from a larger raster image, and our system will then re-scale it to the needed dimensions.
- *Gray-scale Image Conversion.* While most knot diagrams consist of only gray tones of colors, some are color images on purpose (see e.g., Fig.4(a)). Knot diagrams are converted into gray-scale images to reduce the inherent complexity (see Fig.4(c)).
- *Binary Image Conversion.* This transformation is useful in detecting our objects of interest — the mathematical curves, and it further reduces the computational complexity (see e.g., Fig.4(d)).
- *Thinning*. Thinning [32] is a morphological operation that is used to remove selected foreground pixels from binary images, somewhat like erosion or opening. In our scenario, we apply the *thinning* operation particularly for skeletonization — to tidy up the output of knot diagrams by reducing all line segments to just single-pixel thickness. Fig.4(e) shows the resultant mathematical trefoil knot diagram with lines of just one-pixel thickness.
- *3D Reconstruction.* A 3D knot is represented by a sequence of 3D points sampled along the curve, and now the 3D points are reconstructed from the foreground pixels in the knot diagram figure. These foreground pixels are not fully connected, due to the rendered "breaks" in the raster image. To reconstruct the 3D geometry, we need to identify where these "breaks" are in the raster image. This can be done by scanning each pixel's 8-connected neighbors, i.e., the eight neighboring pixels. As Fig.4(f) shows, a foreground pixel inside a knot curve will find two foreground pixels in their 8-connected pixels; those at the end of a knot curve will only find one foreground pixel in its 8-connected pixels, and these foreground pixels can be paired and 3D points can be reconstructed between them to represent the under-crossing arcs that correspond to the "breaks" in the knot diagram. Fig.4 shows the sequence of intermediate outcomes and the resultant 3D trefoil knot, reconstructed from a raster image cropped from the knot gallery at Wikipedia.

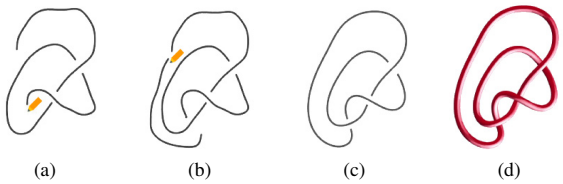


Fig. 5. (a)-(c) Drawing a knot in our sketching canvas via a sequences of under, over, under ... (d) Rendering with light and material adds apparent 3D geometry, depth, and shape to the 2D image.

Drawing a Knot. The user can also draw a knot diagram in our knot sketching canvas. The drawing is done mainly in 2D — one can construct an initial configuration for an object while neglecting most issues of geometric placement. This is possible, for example, when knot diagrams are drawn, it is just bare projections with relative depth ordering indicated at crossings since precise 3D depth information is unimportant. Fig.5 shows a sequence of screen images while the knot sketching process is evolving. When a collision occurs between a piece of an edited curve piece and an existing piece in canvas, users must make explicit over and under choices, e.g., by using modifier keys. The completed mathematical knot in R^3 can be represented as a linked node

list, $K = (V, E)$, where $V = \{v_1, v_2 ..., v_n\}$ is the finite set of *vertices* of the list and E is the set of *edges* $\{e_1, e_2 ..., e_n\}$.

4.2 Suggesting Mathematical Moves for Untangling Knots

We now turn to our main objective in this paper, which is to create unique experience for one to interact with the mathematical knot and untangle it to a simplified (but topologically equivalent) structure. This problem has been approached in different ways. The widely-used *KnotPlot* [21] relaxes and untangles knots in three-dimensional space with a pseudo-physical model. *KnotPad* [30] is a sketching interface for one to only propose the Reidemeister moves to deform mathematical knots, which is only practical when working with knot diagrams with a small number of crossings (for most non-expert users).

We first focus on the integration of numerical and visual approaches to implement a suggestive knot interface that can read the mathematical knot and suggest the moves to untangle complex knots step by step to the fewest possible crossings. We of course exploit and customize numerical approaches to knot deformation [4, 16] behind our suggestive sketching interface. Before we detail the logical series of steps, several terminologies are in order.

Gauss code. The numerical approach we are going to leverage is based on an extended knot notation called Gauss Code. It is a sequence of labels for the crossings with each label repeated twice to indicate a walk along the diagram from a given starting point and returning to that point. Take the trefoil knot in Fig.6 as an example. First, we label all crossings in the knot diagram. Then, we traverse the knot from a given point and along one direction (see the starting point and direction indicated by the red arrow in Fig.6). Once we encounter a crossing, write down the crossing label with a "+" or "-" for the head of each crossing, we will obtain a series of signed number, called Gauss Code. In this trefoil knot example, the Gauss Code will be generated as "+1, -2, +3, -1, +2, -3".

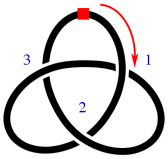
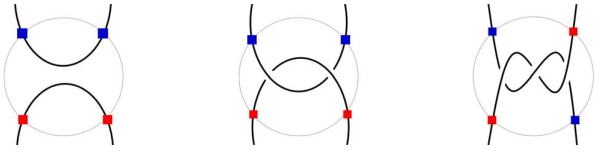


Fig. 6. A trefoil knot and its Gauss code generated as "+1, -2, +3, -1, +2, -3".

Visual Tangle. A visual tangle is a region of a knot where our suggestive interface will highlight and guide the user to perform the mathematical moves. The original definition of *Tangle* was proposed by Foley in [4] to numerically untangle knots. A tangle is a closed region of a knot, where the knot crosses the region exactly four times, with the following two basic properties:

- *Size* — the number of crossings in a tangle. For example, Fig.7 shows three different tangles with sizes 0, 2, and 3.
- *Parity* — the parity of the tangle size. For example, the tangle in Fig.7(b) is an even tangle and in Fig.7(c) is an odd tangle. As defined, in each tangle, two knot segments will cross the tangle boundary exactly four times. When the tangle parity is even, each segment will leave two consecutive crossing points when crossing the region (see e.g., Fig.7(a)(b)); when the tangle parity is odd, crossing points generated by different segments will be neighbors on the boundary (see e.g., Fig.7(c)).

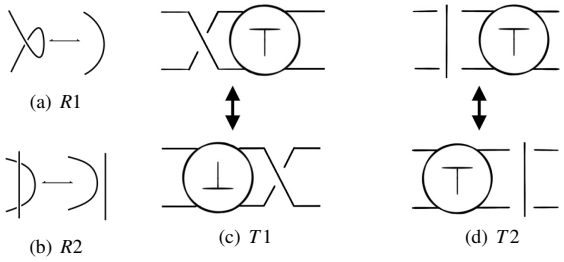


(a) Tangle with size 0 (b) Tangle with size 2 (c) Tangle with size 3

Fig. 7. Visual tangle examples in our suggestive interface, with sizes 0, 2, and 3.

4.2.1 Predicting the Moves and Tangles by Gauss Code

The Reidemeister moves have been proven to be the core moves necessary to fully untangle a knot. In this section, we will detail models and algorithms to suggest the Reidemeister moves in our knot interface. The core prediction capability of our knot interface is based on the numerical approach proposed by Foley in [4]. In Foley's approach the third Reidemeister move is replaced with two generalized translation moves, and the proposed numerical method can read a knot in its Gauss code notation and automatically fully untangle the knot in its Gauss code notation corresponding to the four basic moves listed in Fig.8. The key implementation steps in our implementation can be detailed as follow:



(a) $R1$ (b) $R2$ (c) $T1$ (d) $T2$

Fig. 8. The four generalized Reidemeister moves in our interface. (a) $R1$: the first Reidemeister move. (b) $R2$: the second Reidemeister move. (c) $T1$: translation move 1 to remove the original crossing and create one on the opposite side of the tangle. (d) $T2$: translation move 2 to relocate the strand intersecting both tangle segments to the opposite side of the tangle.

1. Read the knot's Gauss code notation and identify how the Reidemeister moves may be applied with rules detailed below.
2. Identify and perform $R2$ first — look for two adjacent crossings with the same sign; then locate the negatives of these integers, and determine if those numbers are also adjacent. $R2$ can be performed if these conditions are true, and the four numbers will be removed after $R2$ is performed (see e.g., Fig.9(b)).
3. Then identify and perform $R1$ — look for two adjacent integers which are negatives of each other. When this condition is found, the numbers can be removed after $R1$ is performed (see Fig.9(a)).
4. Look for all tangles with size greater than 0. A tangle can be identified or combined from existing tangles with the following three rules:
   - *The sum of the signed integers in a Tangle's Gauss code is* 0. E.g., Tangle 1 in Fig.10 contains two crossing points and the sum of all the Gauss codes is $(-5)+(-4)+(+4)+(+5) = 0$.
   - *A tangle's Gauss code string can be divided into two Gauss code strings belonging to two different knot segments.* E.g., within tangle 2 in Fig.10 the Gauss code $[-1,+2]$ belongs to one arc, and $[-3,-2,+1,+3]$ belongs to a different arc.
   - *Two tangles can be combined if their Gauss code strings are adjacent sub-strings in the knot's Gauss code*. For example, in Fig.10 the knot's Gauss code is $[-1,+2,+6,-5,-4,-3,-2,+1,+3,+4,+5,-6]$. The Gauss code of tangle 1 can be divided into two Gauss code strings: $[-5,-4]$ and $[+4,+5]$ belonging to the two different arcs in tangle 1. Similarly tangle 2 has two Gauss code strings: $[-1,+2]$ and $[-3,-2,+1,+3]$. Since $[-5,-4]$ from tangle 1 is adjacent to $[-3,-2,+1,+3]$ from Tangle 2 in the knot's Gauss code. Tangle 1 and tangle 2 can thus be combined into a tangle of larger size, i.e., the tangle 3. Our program starts with all tangles with size 1, and

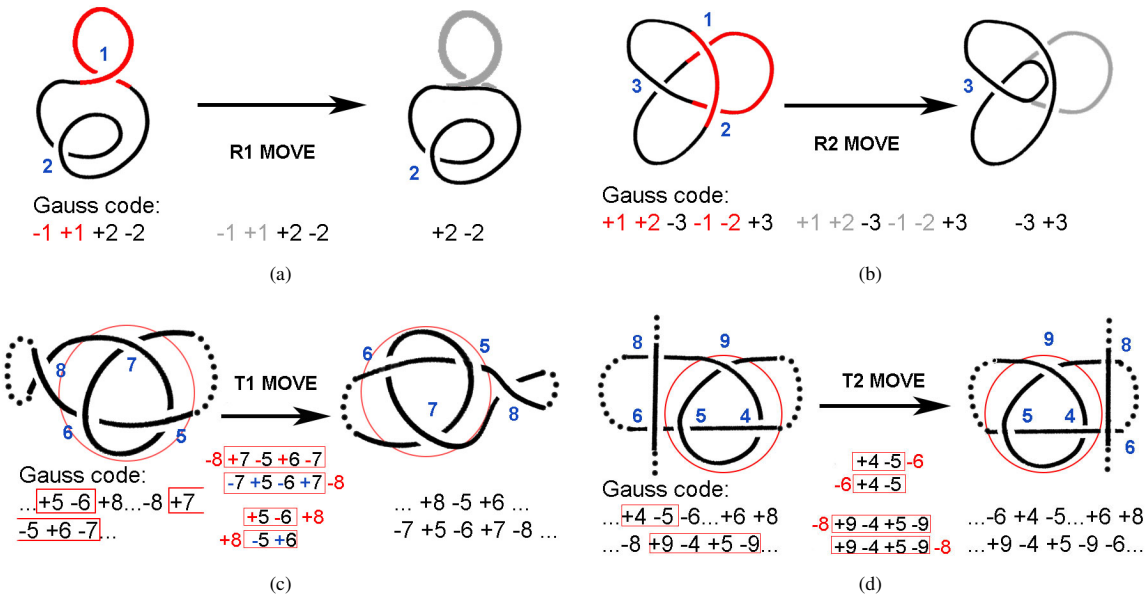


Fig. 9. The four generalized Reidemeister moves used in our interface and the corresponding Gauss code notations before and after the move.

recursively combine tangles to larger sizes until no more combinations can be performed.

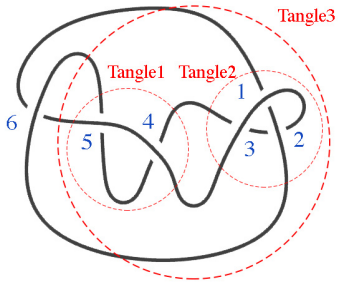


Fig. 10. Tangle 1 and Tangle 2 can be combined as Tangle 3.

5. Check if $T1$ can be applied to any identified tangles. $T1$ should be performed if the following two conditions are true:

   - any two of the crossing integers beyond the ends of the tangle are negations of each other. For example, in Fig.9(c) the crossing immediately beyond the tangle has two crossing integers negative of each other, i.e., "-8" and "$+8$", so we can perform a valid $T1$.
   - $R2$ and/or $R1$ are available upon the execution of $T1$.

   When a $T1$ is performed against a tangle, we first negate the value of every number composing the tangle, flipping it over. Next the two numbers composing the crossing adjacent to the tangle are removed from their original positions and inserted in the locations on the opposite side of the tangle (see Fig.9(c)).

6. Check if $T2$ can be applied to any identified tangles. $T2$ should be performed if the following two conditions are true:

   - the two crossings immediately beyond two adjacent ends of the tangle are both overpasses or underpasses. For example, the two crossings, "-6" and "-8", immediately beyond the tangle in Fig.9(d), are both negative and adjacent to each other in the knot's Gauss code, so we can perform a valid $T2$.
   - $R2$ and/or $R1$ are available upon the execution of $T2$.

   When a $T2$ is performed, all integers in the tangle's Gauss code stay unchanged, and the two adjacent integers immediately beyond the tangle will be relocated to the opposite side of the tangle (see e.g., Fig.9(d)).

4.2.2 From Numerical Results To Suggestive Visual Interface

In this section we will focus on the translation of numerical results to visual information in our interface to guide the user on fully untangling mathematical knots.

System Suggestions on Tangles and Moves. Our objective here is to provide a suggestive visual interface rather than a numerical expression, and thus we have adopted a customized hybrid approach. For example, we can utilize the numerical method behind our interface that reads the knot diagram in its Gauss code and determines how to continue untangling a knot in its Gauss code notation. Rather than actually untangle the knot by just changing its Gauss code notation, we will translate the numerical information into visual and geometric information to suggest visual tangles where the next moves should be made. As illustrated in Fig.11, our visual interface provides the following elements to suggest tangles and moves during each action performed by the user:

- Portions of the knot, where the $R1$ and/or $R2$ moves can be applied, are highlighted as editable pieces in green.
- Visual tangles, where the $T1$ and $T2$ moves can be applied to introduce more $R1$ and/or $R2$ moves, are highlighted in red; while the editable pieces, immediately beyond the tangle for $T1$ and/or $T2$ moves, are highlighted in green.
- For each suggested move, the interface provides a dialogue to illustrate how to make the four moves (see e.g., Fig.11(e)-(h)).

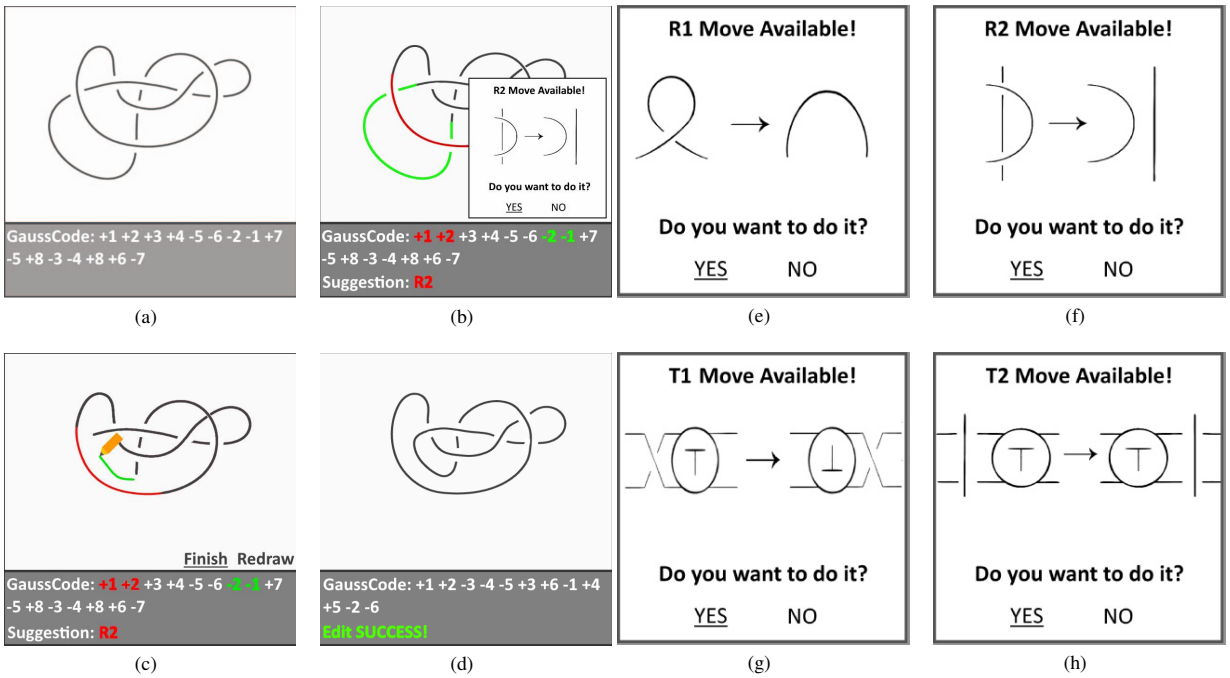


Fig. 11. (a)-(d): Moves and tangles are translated to highlighted knot portions in our interface. (e)-(h): Dialogues to illustrate the four moves to end users.

User-Defined Visual Tangles. In addition to following the system suggestion on the tangles and available moves, one can also explicitly define a tangle of interest, by "brushing" a portion of the mathematical knot in our interface. The knot portion where the user brushes will be highlighted in green (see e.g., Fig.12), and the interface will match the user-defined tangle with all tangles our system has numerically identified and will suggest moves applicable to the user-defined tangle. When the user decides to apply a suggested move, our system will erase the portion to be edited and wait for the user to complete the sketching. The system will validate the moves sketched by the user using methods that we are going to treat in detail next. In Fig.12, the user brushes a visual tangle and our system suggests a $T2$ move, and eventually accepts the move sketched by the user following validation.

Fig. 12. One can explicitly define a visual tangle to edit a knot in our interface. The interface will suggest all Reidemeister moves applicable to the user-defined visual tangle through a pop-up menu.

Move Validation. As designed, our knot interface will constrain the user's knot drawing to the four mathematically legal moves, i.e., $R1$, $R2$, $T1$, and $T2$ moves. To validate each move sketched by the user, our system will first "generate" the expected move numerically, i.e., in the format of Gauss code notation. When a move is sketched and committed by the user, our system will need to validate this potential move. The validation performed by our system is based on the Gauss code notation — ideally if the sketched move is "same" as what is recommended by our numerical analysis, we should be looking at two identical Gauss codes. However, depending on giving starting point and traverse direction one knot could have different Gauss code notations. To validate the proposed moves, we use a slightly different notation we called *Gap code*, which is derived from Gauss code but can be helpful to further determine if two different Gauss codes actually represent the same mathematical knot.

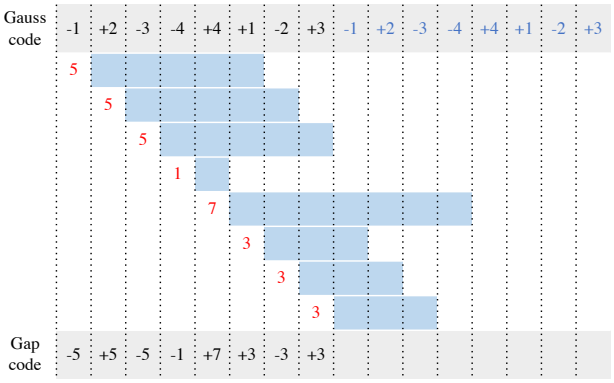
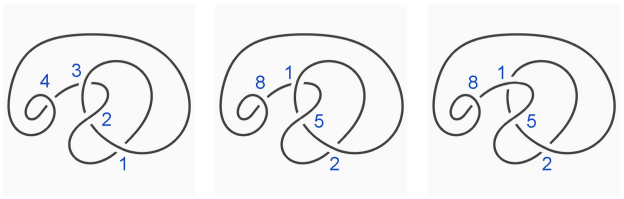


Fig. 13. Convert a knot's Gauss code to our *Gap code*.

The method to derive *Gap code* is explained in Fig.13. We start from the first signed integer in the Gauss code string, and for each such signed integer in the string, we measure its distance to its negated integer by walking towards the end of the Gauss code string (if needed Gauss code string is repeated). For each integer in the Gauss code string we generate a corresponding signed integer in its Gap code string, by taking the measured distance as its absolute value and keeping the sign from the negated integer. When two Gap codes are identical or negated of each other, the two knots are the "same". For example, Fig.14(a) and Fig.14(b) are showing two different Gauss code strings generated from one knot with different starting point and the traverse direction.

The knot in Fig.14(c) looks slightly different but represents a totally different topology. While the Gauss code in Fig.14(a) appears to be totally different from that in Fig.14(b), their Gap codes are just negated of each other. The Gauss code and Gap code in Fig.14(c) are totally different from those in Fig.14(a) or Fig.14(b), and the knot should be treated as a completely different topology.



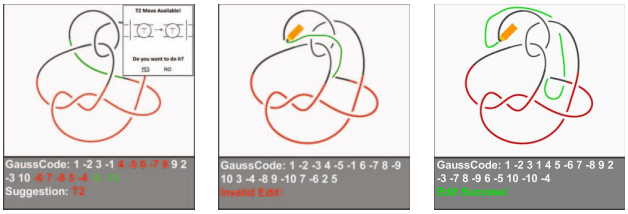
(a) Gauss code:
-1 +2 -3 -4 +4 +1 -2 +3.
Gap code:
-5 +5 -5 -1 +7 +3 -3 +3

(b) Gauss code:
+1 -5 +2 +8 -8 -1 +5 -2.
Gap code:
+5 -5 +5 +1 -7 -3 +3 -3

(c) Gauss code:
-1 -5 +2 +8 -8 +1 +5 -2.
Gap code:
-5 -5 +5 +1 -7 +3 +3 -3

Fig. 14. Comparing knot equivalence using Gauss code and Gap code.

Fig.15 shows the typical screen images of our suggestive interface when validating every single move proposed by the user. Our system will accept fully validated move before making actual change to the underlying knot structure.



(a) Suggested move (b) Invalid move (c) Valid move

Fig. 15. Our interface validates every single move proposed by the user.

4.2.3 Smoothing the Knot

Upon the commit of each move proposed by the user, the system will smooth the mathematical knot before the underlying numerical method identifies a new tangle/move to be performed. The reason is twofold:

- Hand-sketched curves can be jittery especially when one keeps editing portions of the diagrams. A smoothing operation can help the knot diagram to maintain smooth and elegant.
- To identify new translation moves, i.e., $T1$ and $T2$, the numerical method need to perform multiple-level recursive searching to look for tangles and $T1$ and $T2$ moves, which significantly increases our system's run-time complexity. Smoothing the knot can potentially help identify tangles and moves for the user to continue untangling the knot diagram.

Smoothing in our system is performed with a relaxation algorithm. The basic ideas is to use a force model that can improve the layout of a mathematical knot's linked nodes by re-positioning them at more balanced locations in the space. The force model uses two forces [21]:

- *Attractive force* applied between all pairs of immediately adjacent nodes in the knot diagram; the attractive force is a generalization of Hooke's law, allowing for an arbitrary power of the distance r between nodes, $F_m = H(\frac{r}{d})^{1+\beta}$,
- *repulsive force* applied between all pairs of non-adjacent nodes in the knot diagram, defined by $F_e = K(\frac{r}{d})^{-(2+\alpha)}$,

where r is the distance between two nodes, and d is the balanced distance of two points, and H, K, α, and β are constants. We set $\alpha = 5$, $\beta = 2$, and both H and K are 1. When performing smoothing, we calculate the overall force at each node, and move the node along the overall force's direction while avoiding collisions between all pairs of non-adjacent segments.

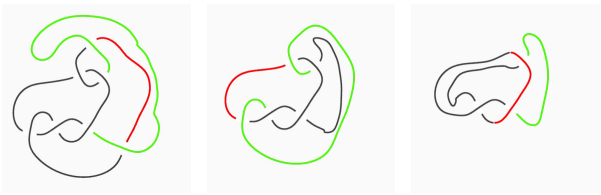
4.2.4 Key Moments of the Mathematical Deformation

To further improve our experience with mathematical knots, our system "remembers" every mathematically legal move that has been applied during the entire process. Each such move, i.e., $R1/R2/T1/T2$, is considered a critical change in the deformation. Our interface displays an array of snapshots for one to track and trace the key moments where these critical changes have occurred. Fig.16 shows the eight key moments to summarize how a string that appears to have ten crossings at the beginning was fully untangled step by step in our interface.

(a) Gauss Codes:(1, -2, 3, -1, 4, -5, 6, -7, 8, 9, 2, -3, 10, -6, 7, -8, 5, -4, -9, -10)
Suggestion: T2

(b) Gauss Codes:(1, -2, 3, -1, 4, 5, -6, 7, -8, 9, 2, -3, -7, 8, -9, 6, -5, 10, -10, -4)
Suggestion: R1

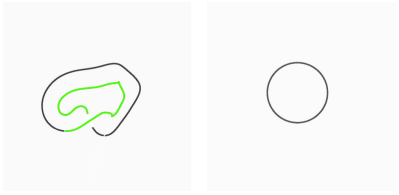
(c) Gauss Codes:(-1, 2, -3, 4, -2, 1, 5, -6, 7, -8, 9, 3, -4, -7, 8, -9, 6, -5)
Suggestion: R2



(d) Gauss Codes:(-1, 2, -3, 4, 5, -6, -2, 3, -4, 1, 7, -5, 6, -7)
Suggestion: R2

(e) Gauss Codes:(-1, 2, 3, -4, 5, 1, -2, -3, 4, -5) R2

(f) Gauss Codes:(-1, 2, 3, 1, -2, -3)
Suggestion: R2



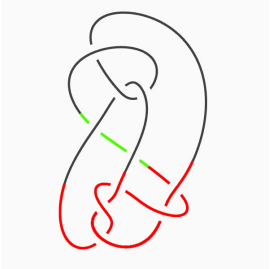
(g) Gauss Codes:(-1, 1)
Suggestion: R1

(h) Gauss Codes:()
Suggestion: NO

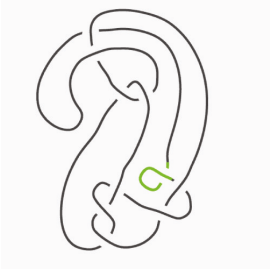
Fig. 16. Displaying the 8 critical changes to turn an unknotted string of 10 crossings into a smooth circle.

5 Implementation Environment And More User Interfaces

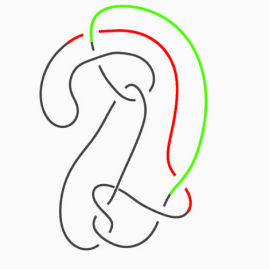
We have used $C++$ programming language for models and algorithms detailed in this paper, and have used the *Qt* framework to build the graphical user interface. Our core rending capability, including the planar knot diagram and the 3D rendering for mathematical knots, is based on *OpenGL*. The software currently runs on a MacBook Pro with 2.2GHz 6-Core Intel Core i7 Processor and Radeon Pro 555X graphic processor. The software was also migrated and tested on windows platform workstations with NVDIA graphics card. Fig.18 shows representative views of mathematical knots rendered in our interface.



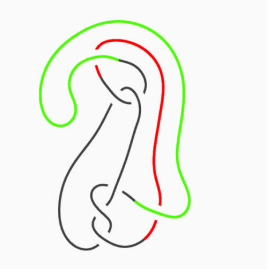
(a) Gauss Codes:(1, -2, 3, -4, 5, 6, 7, -8, 9, -3, 4, -5, 2, -1, -6, -9, 10, -7, 8, -10)
Suggestion: T2



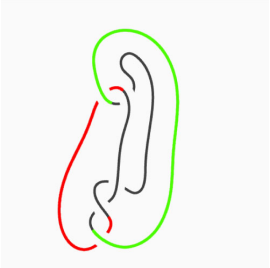
(b) Gauss Codes:(1, -2, 3, -1, 4, 5, -6, 7, -8, 9, 2, -3, -7, 8, -9, 6, -5, 10, -10, -4)
Suggestion: R1



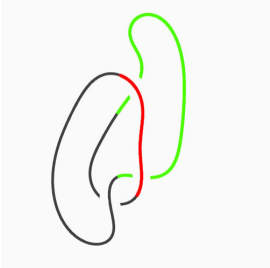
(c) Gauss Codes:(-1, 2, -3, 4, -2, 1, 5, -6, 7, -8, 9, 3, -4, -7, 8, -9, 6, -5)
Suggestion: R2



(d) Gauss Codes:(-1, 2, -3, 4, 5, -6, -2, 3, -4, 1, 7, -5, 6, -7)
Suggestion: R2

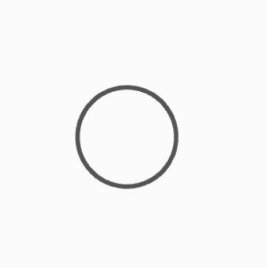


(e) Gauss Codes:(-1, 2, 3, -4, 5, 1, -2, -3, 4, -5)
Suggestion: R2



(f) Gauss Codes:(-1, 2, 3, 1, -2, -3)
Suggestion: R2

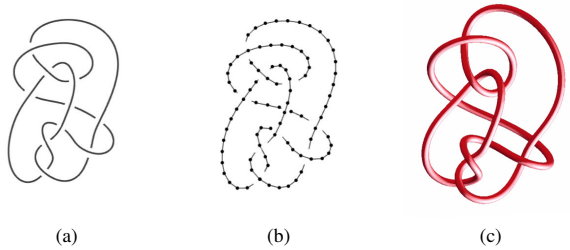
(g) Gauss Codes:(-1, 1)
Suggestion: R1



(h) Gauss Codes:()
Suggestion: NO

Fig. 17. Turning an unknotted string of 10 crossings into a trivial circle with just seven moves.

The user can load and draw a variety of mathematical knots and links for investigation, and is also presented a wide selection of options. Fig.19 shows one uses our system to fully reconstruct the Borromean rings, a link with three components each equivalent to the unknot, by "loading" a raster image available at the mathematical link Wikipedia page. Fig.17 and Fig.20 show further screen images of fully untangling mathematical curves that otherwise appear to have a large number of crossings.



(a) (b) (c)

Fig. 18. Various rendering modes for visualizing knots in our interface. (a) Knot Diagram. (b) Linked-Node Representation. (c) Smooth 3D Rendering.

6 Preliminary Usability Test

We have performed a preliminary usability test in the UofL VCL (Visual Computing Lab) to evaluate our suggestive knot interface by inviting a group of 10 non-expert participants to perform the three unknotting tasks with the three complex knots shown in Fig.16, Fig.17, and Fig.20 respectively. Participants were asked to perform the tasks in three different interfaces using as much time as they desired (until they either completed the tasks or they no longer wished to work on the tasks): 1) the *Knotplot* software [21] which approaches unknotting problem

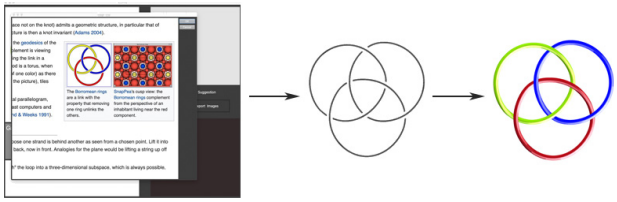
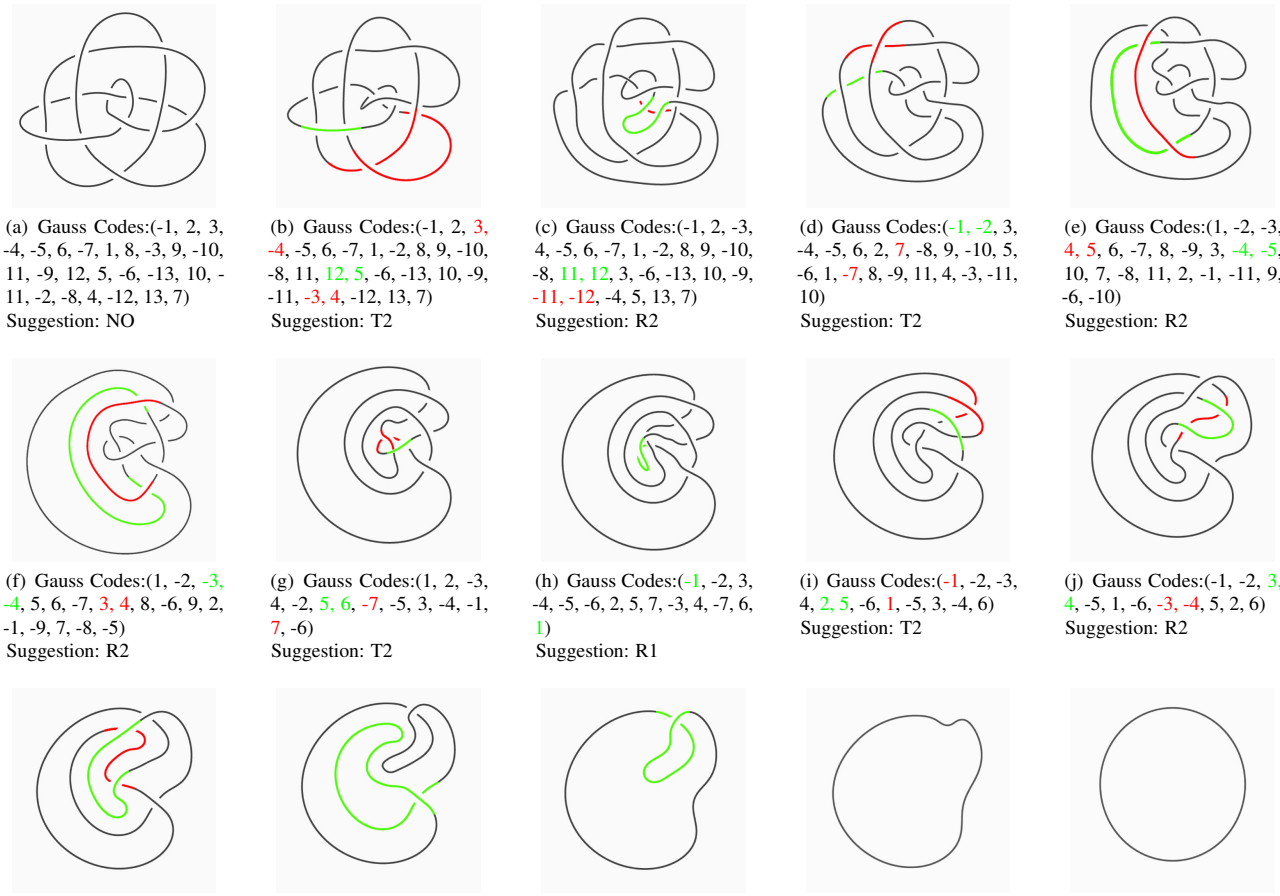


Fig. 19. Reconstructing a mathematical link from a raster image from Wikipedia.

using pseudo-physical simulations, 2) a sketching-only interface (by turning off our software's suggestive interface elements, essentially the *KnotPad* [30]), and 3) our sketching based knot interface with the system-generated suggestions for knot editing.

The usability test measured the **task completion rate**, **average completion time**, and **average user interactions** needed in the task. We have also collected participants' numeric rating (5-point scale, 5 for strongly agree and 1 for strongly disagree) on the interfaces' **user guidance**, **learnability**, and **overall rating**. Table 1 summarizes the results of this preliminary usability test. All the participants were able to complete the tasks of untangling the three complicated knots with *Knotplot* and our suggestive interface. However with *Knotpad*, only one participant was able to complete one task successfully and another participant nearly completed a task; other participants stopped working on the tasks after an average of 8 minutes. Interviews with them revealed that it was nearly impossible for them to untangle knots with complicated structures without system-generated suggestions.

The completion rate and completion time of the participants performing the three tasks on our new knot interface shows the system-generated suggestions can not only make it faster to untangle a knot, but also make it *possible* to untangle a complicated knot. The participants all agree that our suggestive interface can help them understand the



(a) Gauss Codes:(-1, 2, 3, -4, -5, 6, -7, 1, 8, -3, 9, -10, 11, -9, 12, 5, -6, -13, 10, -11, -2, -8, 4, -12, 13, 7)
Suggestion: NO

(b) Gauss Codes:(-1, 2, 3, -4, -5, 6, -7, 1, -2, 8, 9, -10, -8, 11, 12, 5, -6, -13, 10, -9, -11, -3, 4, -12, 13, 7)
Suggestion: T2

(c) Gauss Codes:(-1, 2, -3, 4, -5, 6, -7, 1, -2, 8, 9, -10, -8, 11, 12, 3, -6, -13, 10, -9, -11, -12, -4, 5, 13, 7)
Suggestion: R2

(d) Gauss Codes:(-1, -2, 3, -4, -5, 6, 2, 7, -8, 9, -10, 5, -6, 1, -7, 8, -9, 11, 4, -3, -11, 10)
Suggestion: T2

(e) Gauss Codes:(1, -2, -3, 4, 5, 6, -7, 8, -9, 3, -4, -5, 10, 7, -8, 11, 2, -1, -11, 9, -6, -10)
Suggestion: R2

(f) Gauss Codes:(1, -2, -3, -4, 5, 6, -7, 3, 4, 8, -6, 9, 2, -1, -9, 7, -8, -5)
Suggestion: R2

(g) Gauss Codes:(1, 2, -3, 4, -2, 5, 6, -7, -5, 3, -4, -1, 7, -6)
Suggestion: T2

(h) Gauss Codes:(-1, -2, 3, -4, -5, -6, 2, 5, 7, -3, 4, -7, 6, 1)
Suggestion: R1

(i) Gauss Codes:(-1, -2, -3, 4, 2, 5, -6, 1, -5, 3, -4, 6)
Suggestion: T2

(j) Gauss Codes:(-1, -2, 3, 4, -5, 1, -6, -3, -4, 5, 2, 6)
Suggestion: R2

(k) Gauss Codes:(-1, 2, -3, 1, 4, 3, -2, -4)
Suggestion: R2

(l) Gauss Codes:(1, -2, 2, -1)
Suggestion:R1

(m) Gauss Codes:(-1, 1)
Suggestion: R1

(n) Gauss Code:()
Suggestion: NO

(o) Gauss Codes:()
Suggestion: NO

Fig. 20. Unknotting a mathematical curve that appears to have 13 crossings.

important theoretical results by allowing them to interact with the knots step by step. Therefore they gave the highest numeric rating to our interface on *Learnability*. These results suggest the new knot interface is an improved interface that can enable and enrich one's mathematical experience with knots with complicated structures.

Table 1. Comparing three approaches to unknotting problems: physical simulation (e.g., Knotplot), pure sketching interface (e.g., KnotPad), and our suggestive interface.

Knot Interfaces Comparison			
	Physical	Sketching-only	Suggestive Sketching
Completion Rate	100%	5%	100%
Completion Time	$\approx 51s$	∞	$\approx 20s$
User Interactions	≈ 0	∞	$\approx 10times$
User Guidance	1	1	5
Learnability	2	1	5
Overall Rating	3	2	4.5

7 Conclusion

In this paper we have presented how interactive graphics, numerical methods, and visual interfaces can combine to enable one's unique and intuitive experience with mathematical knots. We have introduced methods to create and model mathematical knots, interaction procedures to approach sketching based unknotting tasks, and visual interface elements to allow suggestion and validation while the sketching process is evolving to untangle a mathematical knot towards the fewest possible number of crossings. Compared to other approaches and interfaces to the unknotting problems, our interface has been designed to approach knots in the closest and most intuitive way to the mathematical problem being studied. Our preliminary user study has suggested that non-expert users interested in understanding knot theory can benefit from the interface to gain intuition. The interface's knot drawing and editing suggestion capability also has the potential to assist knot theory experts to illustrate geometric structures and Reidemeister moves of knots to non-expert audience through graphics interaction with knots.

Starting from this basic framework, we plan to extend the range of objects for which we can support to include more complex mathematical entities where only a suggestive interface can help to make possible the mathematical experience. Other direction of future work also includes migrating our suggestive interface to multi-touch interface to allow even easier user interaction with mathematical entities. We also plan to perform a more thorough user studies for the result to be reported in a relevant venue.

Acknowledgments

This work was supported by NSF award # 1651581.

REFERENCES

[1] J. S. Carter. Reidemeister/roseman-type moves to embedded foams in 4-dimensional space. 2012.

[2] J. Dorsey, S. Xu, G. Smedresman, H. Rushmeier, and L. McMillan. The mental canvas: A tool for conceptual architectural design and analysis. In *15th Pacific Conference on Computer Graphics and Applications (PG'07)*, pp. 201–210. IEEE, 2007.

[3] A. Fish, A. Lisitsa, and A. Vernitski. Visual algebraic proofs for unknot detection. In *International Conference on Theory and Application of Diagrams*, pp. 89–104. Springer, 2018.

[4] D. Foley. Automated reidemeister moves: A numerical approach to the unknotting problem. *arXiv preprint arXiv:1501.05365*, 2015.

[5] G. K. Francis. *A Topological Picturebook*. Springer, 1987.

[6] C. M. Gordon. Some aspects of classical knot theory. In *Knot theory*, pp. 1–60. Springer, 1978.

[7] A. J. Hanson, T. Munzner, and G. Francis. Interactive methods for visualizable geometry. *Computer*, 27(7):73–83, 1994.

[8] J. Hass, J. C. Lagarias, and N. Pippenger. The computational complexity of knot and link problems. *Journal of the ACM (JACM)*, 46(2):185–211, 1999.

[9] T. Igarashi, S. Matsuoka, and H. Tanaka. Teddy: a sketching interface for 3d freeform design. In *ACM SIGGRAPH 2006 Courses*, pp. 11–es. 2006.

[10] K. Jordan, L. E. Miller, E. Moore, T. Peters, and A. Russell. Modeling time and topology for animation and visualization with examples on parametric geometry. *Theoretical Computer Science*, 405(1):41 – 49, 2008. Computational Structures for Modelling Space, Time and Causality.

[11] A. Kawauchi. *A survey of knot theory*. Birkhäuser, 2012.

[12] V. Kurlin. A linear time algorithm for visualizing knotted structures in 3 pages. In *IVAPP*, 2015.

[13] A. M. Ladd and L. E. Kavraki. Using motion planning for knot untangling. *The International Journal of Robotics Research*, 23(7-8):797–808, 2004.

[14] P. Lv, P. Wang, W. Xu, J. Chai, M. Zhang, Z. Pan, and M. Xu. A suggestive interface for sketch-based character posing. In *Computer Graphics Forum*, vol. 34, pp. 111–121. Wiley Online Library, 2015.

[15] M. Masry, D. Kang, and H. Lipson. A freehand sketching interface for progressive construction of 3d objects. In *ACM SIGGRAPH 2007 courses*, pp. 30–es. 2007.

[16] C. Petronio and A. Zanellati. Algorithmic simplification of knot diagrams: New moves and experiments. *Journal of Knot Theory and Its Ramifications*, 25(10):1650059, 2016.

[17] J. Phillips, A. M. Ladd, and L. E. Kavraki. Simulated knot tying. In *ICRA*, pp. 841–846. IEEE, 2002.

[18] K. Reidemeister. *Knot theory*. BCS Associates Moscow, Idaho, 1983.

[19] D. Roseman. Motions of flexible objects. In T. L. Kunii and Y. Shinagawa, eds., *Modern Geometric Computing for Visualization*, pp. 91–120. Springer Japan, Tokyo, 1992.

[20] R. G. Scharein. *Interactive Topological Drawing*. PhD thesis, Department of Computer Science, The University of British Columbia, 1998.

[21] R. G. Scharein. *Interactive topological drawing*. PhD thesis, University of British Columbia, 1998.

[22] G. F. Simmons and J. K. Hammitt. *Introduction to topology and modern analysis*. McGraw-Hill New York, 1963.

[23] J. K. SIMON. Energy functions for polygonal knots. *Journal of Knot Theory and Its Ramifications*, 03(03):299–320, 1994. doi: 10.1142/S021821659400023X

[24] J. Spillmann and M. Teschner. An adaptive contact model for the robust simulation of knots. *Comput. Graph. Forum*, 27(2):497–506, 2008.

[25] D. Terzopoulos and A. Witkin. Physically based models with rigid and deformable components. *IEEE Comput. Graph. Appl.*, 8(6):41–51, 1988.

[26] S. Tsang, R. Balakrishnan, K. Singh, and A. Ranjan. A suggestive interface for image guided 3d sketching. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '04, p. 591–598. Association for Computing Machinery, New York, NY, USA, 2004. doi: 10.1145/985692.985767

[27] J. Weeks. Snappea: a computer program for creating and studying hyperbolic 3-manifolds, 2001.

[28] H. Zhang and A. Hanson. Shadow-driven 4d haptic visualization. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1688–1695, 2007.

[29] H. Zhang and A. J. Hanson. Physically interacting with four dimensions. In G. Bebis, R. Boyle, B. Parvin, D. Koracin, P. Remagnino, A. V. Nefian, M. Gopi, V. Pascucci, J. Zara, J. Molineros, H. Theisel, and T. Malzbender, eds., *ISVC (1)*, vol. 4291 of *Lecture Notes in Computer Science*, pp. 232–242. Springer, 2006.

[30] H. Zhang, J. Weng, L. Jing, and Y. Zhong. Knotpad: Visualizing and exploring knot theory with fluid reidemeister moves. *IEEE transactions on visualization and computer graphics*, 18(12):2051–2060, 2012.

[31] H. Zhang, J. Weng, and G. Ruan. Visualizing 2-dimensional manifolds with curve handles in 4d. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):2575–2584, Dec 2014. doi: 10.1109/TVCG.2014.2346425

[32] T. Zhang and C. Y. Suen. A fast parallel algorithm for thinning digital patterns. *Communications of the ACM*, 27(3):236–239, 1984.