Erschienen in: IEEE Transactions on Visualization and Computer Graphics. Institute of Electrical and Electronics Engineers (IEEE). 2025, 31(10), S. 7363-7376
https://doi.org/10.1109/tvcg.2025.3545481

Interweaving Mathematics and Art: Drawing Graphs as Celtic Knots and Links With `CelticGraph`

Niklas Gröne, Peter Eades, Karsten Klein, Patrick Eades, Leo Schreiber, Ulf Hailer, Hugo A. D. do Nascimento, and Falk Schreiber

***Abstract*—Celtic knots, an ancient art form often linked to Celtic heritage, have been used historically in the decoration of monuments and manuscripts, often symbolizing the notions of eternity and interconnectedness. This paper introduces the framework `CelticGraph` designed for illustrating graphs in the style of Celtic knots and links. The process of creating these drawings raises interesting combinatorial concepts in the theory of circuits in planar graphs. Further, `CelticGraph` uses a novel algorithm to represent edges as Bézier curves, aiming to show each link as a smooth curve with limited curvature. We also show that with our production mechanisms we can compute any 4-regular plane graph and thereby any celtic knot or link. The `CelticGraph` framework for drawing graphs as celtic knots and links is implemented as an add-on of `Vanted`, a network visualization and analysis tool.**

***Index Terms*—Celtic art, knot theory, interactive interfaces.**

I. Introduction

CELTIC knots are an ancient art form often attributed to Celtic cultures. These elaborate designs (also called "endless knots") were used to decorate monuments and manuscripts, and they were often used to symbolize eternity and interconnectedness. Celtic knots are a well-known visual representation made up of a variety of interlaced knots, lines, and stylized graphical representations. The patterns often form continuous loops with no beginning or end (knots) or a set of such loops (links). In this paper we will explore the use of the Celtic knot visualization metaphor to represent specific graphs in the form of "knot diagrams".

Received 27 December 2023; revised 22 January 2025; accepted 15 February 2025. Date of publication 25 February 2025; date of current version 5 September 2025. The journal paper was selected by the Graph Drawing program chairs and invited for publication in TVCG. This research was supported in part by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Grant 251654672 – TRR 161. Recommended for acceptance by S. G. Kobourov. *(Corresponding author: Niklas Gröne.)*

Niklas Gröne, Karsten Klein, and Ulf Hailer are with the University of Konstanz, 78464 Konstanz, Germany (e-mail: niklas.groene@uni-konstanz.de; karsten.klein@uni-konstanz.de; ulf.hailer@uni-konstanz.de).

Peter Eades and Patrick Eades are with the University of Sydney, Camperdown, NSW 2050, Australia (e-mail: peter.eades@uni-konstanz.de; patrick.eades@sydney.edu.au).

Leo Schreiber is with Independent Scholar, 88690 Uhldingen, Germany (e-mail: leo.schreiber@posteo.de).

Hugo A. D. do Nascimento is with Universidade Federal de Goiás, Goiânia 74690-900, Brazil (e-mail: hadn@ufg.br).

Falk Schreiber is with the University of Konstanz, 78464 Konstanz, Germany, and also with Monash University, Clayton, VIC 3800, Australia (e-mail: falk.schreiber@uni-konstanz.de).

We show how to draw a 4-regular planar graph[1] as a knot/link diagram. In this paper, we will first review the relevant background of related fields in Section II. Then, in Section III, we present an overview of our novel approach for creating aesthetically pleasing pictures of 4-regular planar graphs as knots and links. This process involves constructing specific circuits in the 4-regular planar graph, described in more detail in Section IV. In Section V, we show how to route graph edges so that the underlying links are aesthetically pleasing. This involves some optimization problems for cubic Bézier curves. We also provide an implementation of the presented methods as an add-on for `Vanted` [38]. This system supports the transformation of a graph into a knot (link) representation and interactively changes the layout of both the graph and the knot. In addition, knots can be exported to the 3D renderer Blender [7] to allow for artistic 3D renderings of the knot. Fig. 1 shows a 4-regular planar graph, its knot representation, and a rendering of the knot. In Section VI, we present production mechanisms that allow the construction of knot/link diagrams by merging two existing knots (graphs) or by applying simple operations to the graph, and we show that any 4-regular planar graph (and therefore any knot/link diagram) can be constructed using this method. Section VII provides formal proof for an interesting mathematical claim made in Section IV. The claim states that the number of threaded circuits in any threaded circuit partition of a 4-regular plane graph G is fixed by its combinatorial structure and remains invariant regardless of the particular topological embedding of G. Following this, Section VIII briefly describes the implementation of `CelticGraph` as an add-on for `Vanted`, as well as its rendering in Blender. Finally, Section IX concludes and presents several possible directions for further research regarding framework extensions.

II. Background

This paper has its roots in three disciplines: Mathematical knot theory, Celtic cultural history, and Graph Drawing. We briefly review the relevant parts of these diverse fields in Sections II-A, II-B, and II-C. Further, in Section II-D we review relevant properties of Bézier curves, which are a key ingredient to `CelticGraph`. Additionally, in Section II-E, we distinguish our novel approach to visualizing Celtic knots from the current state of research in related fields.

[1]Graphs used in this paper can contain multiple edges and loops (also called pseudographs or multigraphs).

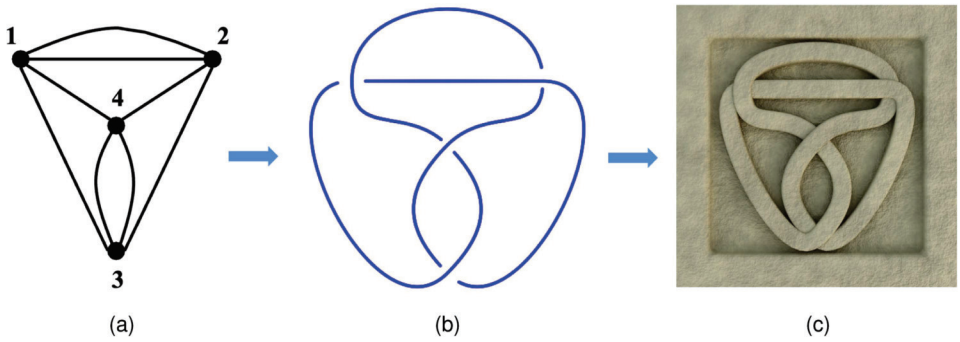


Fig. 1. Using the `CelticGraph` framework, we (a) take the graph K_4' (K_4 with some duplicate edges), and (b) create a knot drawing (knot diagram) of K_4'; from this we (c) render the graph as a Celtic knot with a sandstone texture.

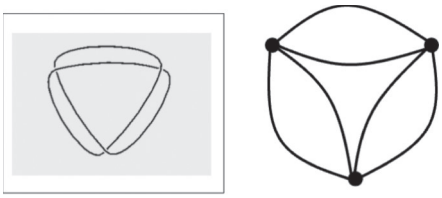


Fig. 2. (a) The *trefoil knot* (knot diagram) and (b) its resulting graph representation.

Fig. 3. Celtic art: (a) Helmet of Agris and (b) Belt Buckle Sutton Hoo. *Figures are in the public domain.*

A. Knot Theory

The mathematical theory of knots and links investigates interlacing curves in three dimensions; this theory has a long and distinguished history in Mathematics [2], [36], [43]. The motivating problem of Knot Theory is *equivalence:* whether two knots can be deformed into each other. A common technique involves projecting the given curves from three dimensions into the plane to form a "knot diagram"; the resulting graph representation is a 4-regular planar graph, with vertices at the points where the curve crosses itself (in the projection). For example, a picture of the *trefoil knot* and its graph representation are in Fig. 2.

A famous theorem states that two knots are equivalent if the knot diagram of one can be transformed to the knot diagram of the other using "Reidemeister moves" [62]. Algorithms to test equivalence have been a core topic of computational topology since the 1960 s. Currently no polynomial time algorithm is known for this test. Properties of a knot or link may be deduced from the knot diagram, and the equivalence problem can sometimes be solved using knot diagrams.

B. Celtic Art

Knot patterns ("Celtic knots") are often described as a characteristic ornament of so-called "Celtic art". In fact, since the epoch of the Waldalgesheim style ($4^{th}/3^{rd}$ century BC), Celtic art (and therefore also ornamentation) is characterized by complex, often geometric patterns of interlinked, opposing or interwoven discs, loops and spirals (see Fig. 3(a)). The floral models, e. g. tendrils and palmettes, originate from Mediterranean art; in the Celtic context, they were deconstructed, abstracted, arranged paratactically or intertwined [52], [56].

It is still unclear whether the import of Mediterranean ornamental models was accompanied by the adoption of their meaning. However, the selective reception of only certain motifs suggests rather an adaptation based on specific Celtic ideas, which we cannot reconstruct exactly due to a lack of written sources. These motifs and patterns possibly express a specific "Celtic" world view that can only be surmised speculatively, according to which all things are in a mystical, ever-changing connection with one another, in which there are transitions, changes of form, different states and alternating forms of being.

In today's popular understanding, a special role in the transmission of actual or supposed "Celtic" art is attributed to the early medieval art of Ireland [45], [60]. However, such a restriction of Irish or insular art to exclusively Celtic origins would ignore the historical development of the insular-Celtic context in Ireland and the British Isles. The early medieval art of Ireland is partially rooted in indigenous Celtic traditions, but was also shaped by Late Antique Roman, Germanic and Anglo-Saxon, Viking and Mediterranean-Oriental models [27].

The knot and tendril patterns of the $7^{th}/8^{th}$ century can also be traced back (indirectly) to Mediterranean-Oriental manuscripts. Such patterns were subsequently used in Anglo-Saxon art, transmitted by braided ribbon ornaments and other patterns in the Germanic "Tierstil" (e. g. on Late Antique soldiers' belts, Fig. 3(b)). For example, the famous Tara Brooch created in

Fig. 4. Celtic art: The late 7th or early 8th century Tara Brooch *Figure is in the public domain.*

(a) (b) (c)

Fig. 5. Celtic art: (a) Book of Kells, (b) Book of Lindisfarne, and (c) Book of Durrow *Figures are in the public domain.*

Ireland in the late 7th or early 8th century showcases a blend of native and Germanic motifs and ornaments [60] (Fig. 4). Also the knot patterns and braided/spiral ornaments described as typically "Celtic", such as in the Book of Kells [48] and other manuscripts (Book of Lindisfarne, 7th/8th century; Book of Durrow, 7th century) can be linked to Germanic/Anglo-Saxon and late Roman traditions (Fig. 5). The ornamentation today often perceived as "Celtic" is therefore less exclusive or typical "Celtic", but rather a result of diverse Late Antique-Roman, Germanic, Anglo-Saxon and Celtic-insular influences, that reflect an equally complex historical-political development [45]. So it is not surprising that the so-called Celtic motifs often presented in tattoo studios of the 21st century, such as braided bands, are not of Celtic but Germanic origin [52].

Note that while "Celtic knots" are related to the mathematical theory of knots, the prime motivation of the two topics is different. For example, the *Bowen knot* [12], a commonly used decorative knot that appears in Celtic cultures, is uninteresting in the mathematical sense (it is clearly an "unknot").

C. Graph Drawing as Art

In Computer Science, the discipline of *Graph Drawing* [17] investigates algorithms for constructing pictures of graphs. Visualizations of graphs are used to understand complex relationships between different data elements, to identify patterns, trends, and correlations between data sets, and to illustrate the structure of a network. Graph drawing methods are at the root of many Visual Analytics systems, applied to diverse contexts such as Biotechnology, network management, and Software Engineering. In particular, there is substantial interest in *planar graph drawing*, that is, algorithms to draw graphs without edge crossings [11], [15], [29], [37].

Note that the purpose of `CelticGraph` is different from most Graph Drawing systems such as yEd, OGDF, Cytoscape, and graphviz [14], [30], [54], [68]. Our aim is to produce decorative and artistically pleasing pictures of graphs, not to make pictures of graphs that effectively convey information and insight into data sets. Other examples of graph drawing systems that serve the purpose of producing decorative and artistically pleasing pictures of graphs include the system by Devroye and Kruszewski to render images of botanical trees based on random binary trees [16], the system `GDot-i` for drawing graphs as dot paintings inspired by the dot painting style of Central Australia [20], [33], and research on bobbin lacework [35]. Also related to our work are Lombardi graph drawings, artistic representations of graphs that contain edges represented as circular arcs and vertices represented with perfect angular resolution [19]. Experiments have shown that Lombardi graph drawings do not convey information any better than traditional graph drawings, nevertheless people prefer Lombardi graph drawings [55].

D. Bézier Curves

The Gestalt law of continuity [42] implies that humans are more likely to follow continuous and smooth lines rather than broken or jagged lines. To draw graphs as Celtic knots, certain circuits in the graph need to be drawn as smooth curves.

Computer Graphics has developed many models for smooth curves; one of the simplest is a *Bézier curve*. A *cubic Bézier curve* with control points p_0, p_1, p_2, p_3 is defined parametrically by:

$$p(t) = (1-t)^3 p_0 + 3(1-t)^2 t p_1 + 3(1-t) t^2 p_2 + t^3 p_3, \tag{1}$$

for $0 \leq t \leq 1$. The following properties of cubic Bézier curves are well-known [26]:

- The endpoints of the curve are the first and last control points, that is, $p(0) = p_0$ and $p(1) = p_3$.
- Every point on the curve lies within the convex hull of its control points.
- The line segments (p_0, p_1) and (p_3, p_2) are tangent to the curve at p_0 and p_3 respectively. We say (p_0, p_1) and (p_3, p_2) are *control tangents* of the curve.
- The curve is C^k *smooth* for all $k \geq 0$, that is, all the derivatives are continuous.

Drawing each edge of a graph as a cubic Bézier curve ensures smoothness in the edges, and can improve readability [67]. However, for `CelticGraph` we need certain *circuits* in the graph to be smooth curves, so we need the curves representing certain incident edges to be *joined smoothly*. Suppose that $p(t)$ and $q(t)$ are two cubic Bézier curves that meet at a common endpoint. Then the curve formed by joining $p(t)$ and $q(t)$ is C^1 smooth as long as the control tangents to each curve at the common endpoint form a straight line; see Fig. 6.

Mathematically, C^1 smoothness is adequate. However, the infinitesimality of Mathematics sometimes does not model human

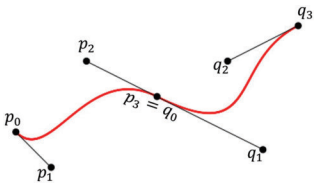


Fig. 6. Two cubic Bézier curves with control points p_0, p_1, p_2, p_3 and q_0, q_1, q_2, q_3, meeting at the point $p_3 = q_0$. The control tangents are shown in black; note that the points p_2, p_3, q_1 lie on a straight line and the join is C^1 and visually smooth.

Fig. 7. A cubic Bézier curve with a "kink", i. e. a point of large curvature, near the middle. The curve is C^1 smooth; but the kink, together with the limits of human perception and screen resolution, mean that the curve does not look smooth.

perception well. For example, the curve in Fig. 7 is mathematically smooth, but given a fixed-resolution screen and the limits of human perception, it appears to have a non-differentiable "kink".

For this reason, it is desirable that the *curvature* [22] of each edge is not too large. Informally, the curvature $\kappa(t)$ is the "sharpness" of the curve. More formally, $\kappa(t)$ is the inverse of the radius of the largest circle that can be placed tangent to the curve at $p(t)$ without intersecting it. For a cubic Bézier curve $p(t) = (x(t), y(t))$, the curvature at $p(t)$ is given by [26]:

$$\kappa(t) = \frac{|\dot{x}\ddot{y} - \ddot{x}\dot{y}|}{(\dot{x}^2 + \dot{y}^2)^{1.5}}, \tag{2}$$

where $\dot{f}$ denotes the derivative of f with respect to t. Note that $\kappa(t)$ is continuous except for values of t where both $\dot{x}(t)$ and $\dot{y}(t)$ are zero. For `CelticGraph`, we need C^1 smooth curves with reasonably small curvature. In Section V-C, we discuss the extreme of curvature and therefore what can be considered *reasonable*.

E. Related Work

1) Knot Diagrams as Lombardi Graph Drawings: Closely related to our types of drawings are Lombardi graph drawings, which are graph drawings with circular-arc edges and perfect angular resolution [19]. Lombardi style drawings have been shown to positively impact user preference, as their aesthetic appeal often makes them more visually engaging and pleasant to work with [55]. Previous studies have demonstrated that a significant group of 4-regular planar graphs can be represented as plane Lombardi graph drawings [39]. However, there are certain restrictions. Notably, if a non-trivial planar graph contains a loop, it cannot be depicted as a Lombardi drawing. In our approach, *every* 4-regular planar graph can be transformed into a knot or link.

2) Celtic Knots by Tiling and Algorithmic Design Methods: Celtic knots can be created using tiling and algorithmic design methods. George Bain introduced a formal method for creating Celtic knot patterns [4], which subsequently has been simplified to a three-grid system by Iain Bain [5], [28]. Klempien-Hinrichs and von Totth study the generation of Celtic knots using so-called "collage grammars" [40]. Even-Zohar et al. investigate sets of planar curves which yield diagrams for all knots [21]. None of those methods use graphs or are graph drawing approaches.

3) Gauss Codes: The graph theory of Celtic knots is strongly correlated to the study of a sequential labeling representation called *Gauss codes*. In fact, every closed curve in the plane that crosses itself in a finite number of times by passing twice in each intersection point can be described by such a code [59]. The Gauss code of a closed curve in a Celtic knot artwork (or of a circuit in a graph representing the Celtic knot, as investigated in this paper) can be obtained by uniquely labeling every crossing point (or vertex of the graph) and then writing the sequence of labels that follows the curve until it returns to the starting point. If the Celtic knot is composed of just one closed curve, then all labels appear twice in its Gauss code. For Celtic knots composed of two or more interlaced closed curves (links), each curve gives rise to an independent Gauss code. In that case, a crossing point shared by two curves is represented by a label that appears exactly once in the two corresponding Gauss codes. As an example, the Gauss code for the graph in Fig. 1(a), considering that each vertex represents a crossing point and starting at vertex 1, is the sequence "12432134".

Some graphical marks can be added to the Gauss code to convey extra information about the intersection points and the associated knot drawings. For instance, by using alternating plus and minus signs before the labels it is possible to indicate whether a curve passes over or under an intersection point, respectively. Furthermore, a common extension of the Gauss code notation explicitly defines whether a crossing is left- or right-handed to distinguish between its two mirrored versions. This last aspect can be denoted by drawing a mark on the top of the labels associated with left-handed crossings in the Gauss code. A more frequent and compact representation of crossing handedness, however, adopts the plus and minus signs differently in the second occurrence of each label in the Gauss code, with the left-handed crossings given by negative numbers. An extended Gauss code for the Celtic knot in Fig. 1 using plus and minus signs to denote over and under crossings and the mark $\hat{}$ to represent left-handed crossings is "$+1 - \hat{2} + \hat{4} - 3 + 2 - \hat{1} + \hat{3} - 4$".

Overall, the Gauss code concept has been employed to facilitate the representation and manipulation of knot drawings in some computational systems [13], [41] as well as for supporting their theoretical study.

4) Drawing Graphs With Bézier Curve Edges: A number of network visualization systems use Bézier curves as edges. These include `yWorks` [68], `GraphViz` [30], `Vanted` [38], `Vizaj` [58], and the framework proposed in [29]. In many cases, such systems allow the user to route the curves by adjusting control points, but few provide automatic computation of the curves. However, there are some exceptions. For example, in the `GraphViz` system, Bézier curve edges are routed within polygons to avoid edge crossings [1]. Force-directed methods are also popular for computing control points of Bézier curve edges [10], [23], [25]. Brandes et al. present a similar method to the "cross" method in Section V, applied to transport networks [9]. However, only [23] considers smoothness in more than one edge.

TABLE I
LIST OF IMPORTANT KNOT DRAWING TOOLS AND THEIR INPUT PARADIGM AS WELL AS MAIN FUNCTIONALITY

Tool	Input Paradigm	Main Focus
KnotPad [69]	Sketching	Exploration
KnotPlot [57]	Sketching, String	Visualization
KnotR [32]	SVG	Visualization
KnotVis [44]	Sketching	Exploration
Knotscape [34]	Sketching	Computational Knot Theory
SeifertView [65]	String	Visualization
Book Knot Simplifier [3]	String	Exploration
Gridlink [31]	String	Exploration
Andrew Bartholomew Webpage [6]	String	Visualization
KnotKit [61]	String	Computational Knot Theory
KnotFolio [47]	Sketching	Computational Knot Theory
Knoty [63]	SVG	Computational Knot Theory
Knotingham [24]	Sketching, Extension	Computational Knot Theory
KnotLikeObjects [64]	Sketching	Computational Knot Theory

5) Knot Drawing Tools: Drawing Celtic weaving patterns is closely related to creating knots and links, for which various specialized tools are available. These tools can be categorized according to their main interaction modality and core functionalities, as shown for a selection of tools in Table I. It is to be noted that our aim is not to present a comprehensive survey of all software available; but rather, to provide an initial overview of available options, accompanied by a classification of said tools based on our findings.

In Table I the second column displays the different input modalities. The 'Sketching' approach serves as an input method that allows users to create two-dimensional curves using a mouse or an equivalent input device. These curves then are automatically enhanced, for instance, through a special smoothing technique as provided in the `SeifertView` tool or preserved in their initial drawing with the capability to 'beautify' and refine at a later stage, like in tools such as `KnotFolio`.

The 'String' approach employs text-based inputs for the generation of knots and links. A prevalent method to achieve this involves the use of the Gauss notation (see Section II-E3) [3], [6]. Furthermore, some tools such as `SeifertView` have developed alternative notation, such as the 'Braid Notation', to enable the conversion of a string input into knots and links.

The tools `Knoty` and the `KnotR` represent a mentionable deviation from most other input paradigms, since both tools are capable of accepting SVG files (Scalable Vector Graphics) as input. In `Knoty`, a knot equation is derived from this SVG image, whereas, in KnotR, the resultant curves undergo a 'beautification' process. Nevertheless, this approach effectively transfers the task of the input generation to an external application, where knots have to be created in some form and then exported as an SVG file, and, in the case of KnotR, this process is again done by hand.

Beyond their input modalities, these tools also differ in their core functions, shown in Table I in the column 'Main Focus'. Here, we can distinguish between tools that focus primarily on visualization and those that compute theoretical properties of knots, such as important invariants. In the middle of these two categories lie tools dedicated to the visual exploration of knot structures, reflecting the main purpose in knot theory. Most of these tools offer Reidemeister steps (see II-A) for an interactive simplification, (see `KnotPad`). Some tools, such as `KnotVis`, even identify regions within a knot drawing where Reidemeister moves can be applied, suggest potential moves, and constrain the user to only make permissible operations.

`CelticGraph` now sets itself apart from existing programs by taking the novel approach of translating a given graph structure into knots and links, emphasizing aesthetic visualizations reminiscent to celtic art as well as exportability.

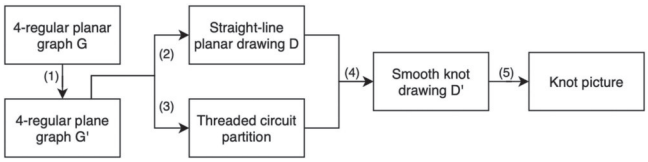


Fig. 8. The `CelticGraph` process.

III. OVERVIEW OF THE `CelticGraph` PROCESS

In this section we outline `CelticGraph`, our framework for creating aesthetically pleasing pictures of 4-regular planar graphs as knots.

Observing typical Celtic knots (e. g. the Book of Kells [48]), several key characteristics emerge, which our CelticGraph framework aims to replicate. These patterns typically involve one or more smooth, interwoven closed curves in three-dimensional space projected onto a two-dimensional plane. In this projection, a specific curve crosses itself or another curve; it either crosses over or under the other curve. In most celtic knots, we observe that the curve alternates between over and under the other curves. Another aspect of these patterns is the angle at which the curves intersect: typically, the curves intersect at a large angle.

The `CelticGraph` procedure is shown in Fig. 8; it has 5 steps:

1) Create a plane graph (that is a topological embedding [53]) G' of the input 4-regular planar graph G.
2) Create a planar straight-line drawing D of the plane graph G'. Note that we extend the standard definition of a planar

straight-line drawing to accommodate the overlay of multiple edges. In this extended definition, multiple edges are allowed to overlap while still being considered part of the planar straight-line drawing.

3) Create a special circuit partition of G', called a "threaded circuit partition". The vertices along each circuit in the circuit partition of G' have an alternating "under-over" assignment.
4) Using the straight-line drawing D and the threaded circuit partition C, create a drawing D' of G with cubic Bézier curves as edges. To ensure that the representations of the threaded circuits are smooth curves that cross at large (right) angles, we choose control tangents for Bézier curves in a special way.
5) Render the drawing D' as a knot, on the screen or with a 3D printer.

The first two steps can be done using standard Graph Drawing methods [17] such as the Hopcroft-Tarjan algorithm [46], [50] or the Boyer-Myrvold algorithm [8] for step (1) and the algorithm by De Fraysseix, Pach & Pollack [15] for step (2) or manual layouts. Steps (3) and (4) are described in the following sections, while step (5) can be done using standard rendering methods.

IV. Step (3): Finding the Threaded Circuit Partition

Here we define *threaded circuit partition*, a special kind of circuit partition of a plane graph, and show how to find it in linear time.

A *circuit* in a graph G is a list of distinct edges $(e_0, e_1, \ldots, e_{k-1})$ such that e_i and e_{i+1} share a vertex for $i = 0, 1, \ldots, k-1$ (here, and in the remainder of this paper, indices in a circuit of length k are taken modulo k). We can write the circuit as a list of vertices $(u_0, u_1, \ldots, u_{k-1})$ where $e_i = (u_i, u_{i+1})$. Note that a vertex can appear more than once in a circuit, but an edge cannot. A set $C = \{c_0, c_1, \ldots, c_{h-1}\}$ of circuits in a graph G such that every edge of G is in exactly one c_j is a *circuit partition* for G. Given a circuit partition, we can regard G as a directed graph by directing each edge so that each c_i is a directed circuit.

A path (α, β, γ) of length two (that is, two edges (α, β) and (β, γ)) in a 4-regular plane graph G is a *thread* if edges (α, β) and (β, γ) are not contiguous in the cyclic order of edges around β. This means that there is an edge between (α, β) and (β, γ) in both counterclockwise and clockwise directions in the circular order of edges around β. We say that β is the *midpoint* of the thread (α, β, γ). Note that each vertex in G is the midpoint of two threads; see Fig. 9(a). For every edge (α, β) in G, there is a unique thread (α, β, γ); we say that the edge (β, γ) is the *next edge after* (α, β). For each vertex u_j on a circuit $c = (u_0, u_1, \ldots, u_{k-1})$ with $k > 1$ there is a path $p_j = (u_{j-1}, u_j, u_{j+1})$ of length two such that u_j is the midpoint of p_j. In fact we can consider that the circuit c consists of k paths of length two. We say that the circuit c is *threaded* if for each j, the path $p_j = (u_{j-1}, u_j, u_{j+1})$ is a thread. Note that in such a circuit, the edge (u_j, u_{j+1}) is the (unique) next edge after (u_{j-1}, u_j) for each j. A circuit partition $C = \{c_0, c_1, \ldots, c_{h-1}\}$ is *threaded* if each circuit c_j is threaded. In the case that $h = 1$, a threaded circuit partition defines a *threaded Euler circuit*; see Fig. 9(b).

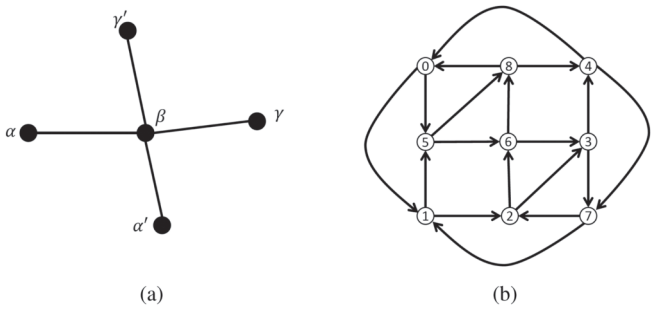


Fig. 9. (a) Two threads (α, β, γ) and $(\alpha', \beta, \gamma')$, each with midpoint β. (b) Plane 4-regular graph with a threaded Euler circuit (0,1,2,3,4,0,5,6,3,7,1,5,8,4,7,2,6,8).

An assignment $\upsilon(p) \in \{-1, +1\}$ of an integer -1 or $+1$ to each thread p of a 4-regular plane graph G is an *under-over assignment*. Note that for each vertex β of G, there are two threads p_β and p'_β with midpoint β. We say that an under-over assignment υ is *consistent* if $\upsilon(p_\beta) = -\upsilon(p'_\beta)$ for each vertex β.

An under-over assignment υ is *alternating* on the circuit $(p_0, p_1, \ldots, p_{k-1})$ if $\upsilon(p_i) = -\upsilon(p_{i+1})$ for each i. An under-over assignment for a graph with a threaded circuit partition C is *alternating* if it is alternating on each circuit in C.

Intuitively, a consistent under-over assignment designates which thread passes under or over which thread, and an alternating under-over assignment corresponds to an alternating knot or link [49].

The following theorem gives the properties of threaded circuit partitions that are essential for `CelticGraph`.

Theorem 1: Every 4-regular plane graph has a unique threaded circuit partition, and this threaded circuit partition has a consistent alternating under-over assignment. Further, this threaded circuit partition can be found in linear time.

Proof: The existence and uniqueness of the threaded circuit partition follows from the fact that every edge has a unique next edge. A simple linear-time algorithm to find the threaded circuit partition is to repeatedly choose an edge e that is not currently in a circuit, then repeatedly choose the next edge after e until we return to e.

We can direct every edge of a 4-regular plane graph G so that each circuit in a given threaded circuit partition C is a directed circuit. This means that we can sensibly define the "left" and "right" faces of an edge. Since a 4-regular plane graph is bridgeless [53], no face is both "left" and "right".

Since the planar dual graph of a 4-regular plane graph is bipartite [53], the faces can be colored *green* and *blue*, such that no two faces of the same color share an edge, see Fig. 10 . An immediate consequence is that the sequence of left faces to (directed) edges in a threaded circuit *alternate* in color. Now consider a thread (α, β, γ) in a (directed) threaded circuit in a threaded circuit partition. If the face to the left of (α, β) is green, then assign $+1$ to the path (α, β, γ); otherwise assign -1 to

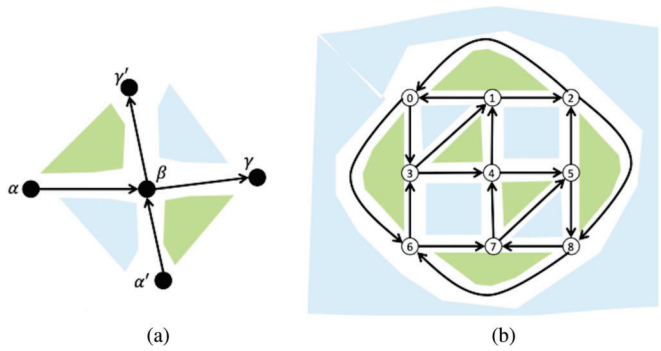


Fig. 10. (a) Two threads: (α, β, γ) has under-over assignment $+1$ (since the face to the left of (α, β) is green), and $(\alpha', \beta, \gamma')$ has under-over assignment -1 (since the face to the left of (α', β) is blue). (b) The faces of the graph are coloured according to its bipartition; note that each vertex has two incoming edges: one has a blue face to the left, the other has a green face to the left, and that the faces on the left of the threaded Euler circuit alternate in color.

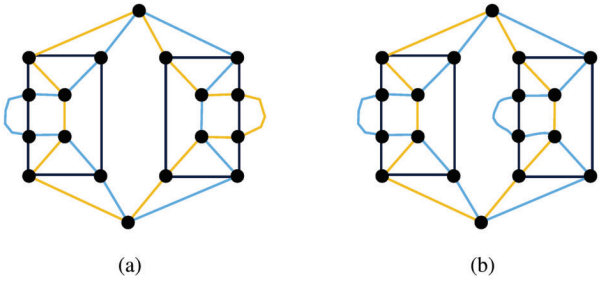


Fig. 11. Two topological embeddings of a planar graph. In (a), the plane graph has a threaded circuit partition of 4 circuits, with two circuits of size 6 (in black) and two circuits of size 12 (in blue and orange). In (b), the plane graph still has 4 threaded circuits: the two of size 6 are unchanged, but the lengths of the blue and orange circuits are 14 and 10 respectively.

(α, β, γ). Note that the face to the left of (β, γ) is the opposite color of the face to the left of (α, β), and so the under-over assignment is alternating. Further it is consistent, since at each vertex there is precisely one incoming arc with a green face on the left, and precisely one incoming arc with a blue face on the left. □

A. Threaded Euler Circuits

Celtic knots are sometimes called "endless knots", and can be used to symbolize eternity. For this reason, a threaded *Euler* circuit is desirable; such a circuit gives a drawing of the graph as a knot rather than a link. Using the algorithms in the proof of Theorem 1, one can test whether a given plane graph has a threaded Euler circuit in linear time. Note that different topological embeddings of a given planar graph may have different threaded circuit partitions; see Fig. 11. It is clear that, in some cases, we can increase the length of a threaded circuit by changing the embedding. It is tempting to try to find a method to adjust the embedding to get a threaded Euler circuit. However, in Section VII, we show that changing the embedding cannot change the *number* of threaded circuits in a threaded circuit partition.

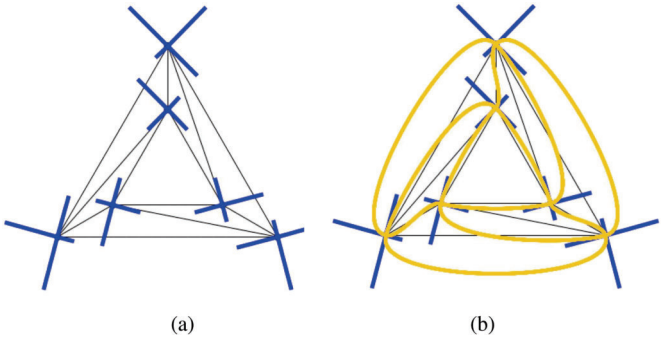


Fig. 12. (a) The 3-prism with a cross at each vertex. (b) Edges drawn as Bézier curves, using the arms of the crosses as control tangents.

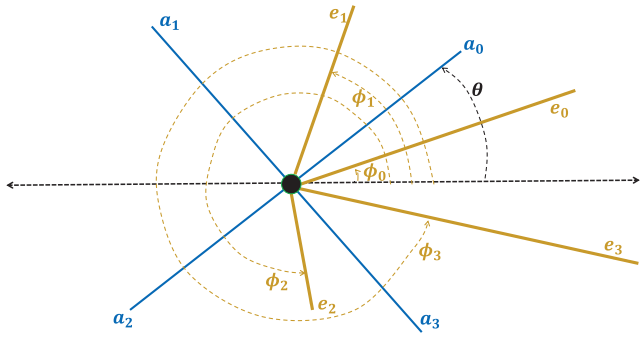


Fig. 13. A cross rotated by an angle of θ. Here the cross is in blue, the edges of the graph are in orange.

V. Step (4): Smooth Knot Drawing With Bézier Curves

Step (4) takes a straight-line drawing D of the input graph G, and replaces the straight-line edges by cubic Bézier curves in a way that ensures that each circuit in the threaded circuit partition found in step (3) is smooth.

A central concept for the smooth drawing method is a "cross" χ_u at each vertex u. For each u, χ_u consists of 4 line segments called "arms". The four arms are all at right angles to each other, leading to a perfect angular resolution. Each arm of χ_u has an endpoint at u.

This is illustrated in Fig. 12(a). Each edge (u, v) then is drawn as a cubic Bézier curve with endpoints u and v, and the control tangents of the curve are arms of the crosses χ_u and χ_v (illustrated in Fig. 12(b)).

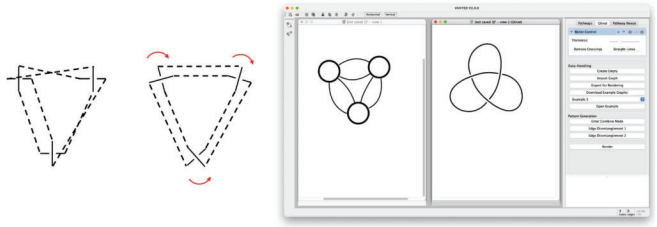
For this approach, we need to choose three parameters for each cross χ_u:

1) The mapping between the four arms of χ_u and the four edges incident to u.
2) The angle of orientation of the cross.
3) The length of each arm of the cross.

These parameters are discussed in the next subsections. The methods described in Section V-A and V-B are analogous to the methods in [9]; Section V-C is not.

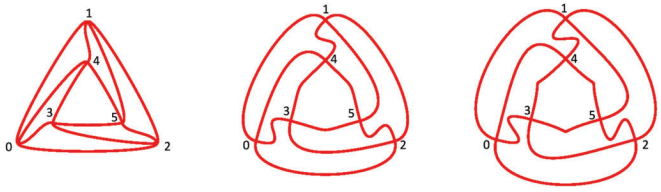
A. The Edge-Arm Mapping

Suppose that u is a vertex in the straight-line drawing D of the input plane graph. We want to choose the mapping



(a) (b) (c)

Fig. 14. A graph with crosses oriented to align with edges as much as possible before (a) and after (b) applying the algorithm, and shown in Vanted (c).



(a) (b) (c)

Fig. 15. Three drawings of the 3-prism, differing in edge curvature.

between the arms of the cross χ_u and the edges incident with u so that the arms are approximately in line with the edges.

Now suppose that the edges incident with u are e_0, e_1, e_2, e_3 in counterclockwise order around u. For each $i = 0, 1, 2, 3$ we choose an arm α_i of the cross χ_u corresponding to e_i so that the counterclockwise order of arms around u is the same as the order of edges around u; that is, the counterclockwise order of arms is $\alpha_0, \alpha_1, \alpha_2, \alpha_3$. Note that this method separates multi-edges.

B. The Orientation of the Cross

To improve the alignment of the arms of the crosses with the edges, we rotate each cross. Suppose that the counterclockwise angle that edge e_i makes with the horizontal direction is ϕ_i. We want to rotate the cross by an angle θ to align with the edges, as best as possible. This is illustrated in Fig. 13.

Consider the sum of squares error in rotating by θ; this is:

$$f(\theta) = \sum_{i=0}^{i=3} \left(\theta + \frac{i\pi}{2} - \phi_i \right)^2. \tag{3}$$

To minimize $f(\theta)$, we solve $f'(\theta) = 0$ and choose the optimum value:

$$\theta^* = \frac{1}{4} \left(\sum_{i=0}^{i=3} \phi_i \right) - \frac{3\pi}{4}. \tag{4}$$

In Fig. 14, we show a graph with crosses oriented by this method.

C. Arm Length

Recall that the "apparent smoothness" of an edge depends on its curvature. We illustrate this with Fig. 15, which shows three Bézier curve drawings of the 3-prism. This graph has 3 threaded circuits, and we want to draw it so that each one of these threaded circuits appears as a smooth curve with limited curvature. In Fig. 15(a), the arms of the crosses are all very short, resulting in a Bézier curve drawing which is very close to a straight-line drawing. Each edge has low curvature in the middle and high curvature around the endpoints. The high curvature near their endpoints results in a lack of apparent smoothness where two Bézier curves join (at the vertices); it is difficult to discern the three threaded circuits. The arms of the crosses are longer in Fig. 15(b), resulting in better curvature at the endpoints. However, here each of the edges $(0,3), (1,4)$, and (2,5) have two points of large curvature; this is undesirable. In Fig. 15(c), the arms of the crosses are longer still. Each of the edges $(0,3), (1,4)$, and (2,5) again have two points of large curvature, but the edges (3,4), (4,5), and (5,3) are worse: each has a "kink" (a point of very high curvature, despite being C^1-smooth).

(a) (b) (c)

Fig. 16. The uniformly proportional approach: (a) $\alpha = 0.2$; (b) $\alpha = 0.4$; (c) $\alpha = 0.6$.

Next we describe three approaches to choosing the lengths of the arms of the crosses, aiming to give sufficiently small curvature. The curvature of the edge varies with lengths of the arms, and we want to ensure that the maximum curvature in each edge is not too large.

1) Uniform Arm Lengths: The simplest approach is to use *uniform arm lengths*, that is, judiciously choose a global value λ and set the length of every arm length to λ. The drawings of the 3-prism in Fig. 15 have uniform arm lengths: λ in Fig. 15(a) is quite small, in Fig. 15(c) it is relatively large, and (b) is in between. In fact, the problem with the uniform arm length approach is typified in Fig. 15: if λ is small, the curvature is high near the endpoints for all edges, and increasing λ increases the curvature away from the endpoints, especially in the shorter edges. There is no uniform value of λ that gives good curvature in both short and long edges.

2) Uniformly Proportional Arm Lengths: An approach that aims to overcome the problems of uniform arm length is to use *uniformly proportional arm lengths:* we judiciously choose a global value α, and then set the lengths of the two arms for edge (u, v) to $\alpha d(u, v)$, where $d(u, v)$ is the euclidean distance between u and v. Fig. 16 shows typical results for the uniformly proportional approach. For $\alpha = 0.2$ the drawing displayed in Fig. 16(a) is similar to Fig. 15(a), and has similar problems. But for values of α near 0.5 (Fig. 16(b) and (c)), we have acceptable results; in particular, the shorter edges have acceptable curvature.

3) Optimal Arm Lengths: A third approach is to choose the arm lengths at each end of an edge (u, v) to minimize maximum curvature, as follows. Suppose $\kappa(t, \lambda_u, \lambda_v)$ is the curvature of the edge (u, v) at point t on the curve, when the arm lengths are

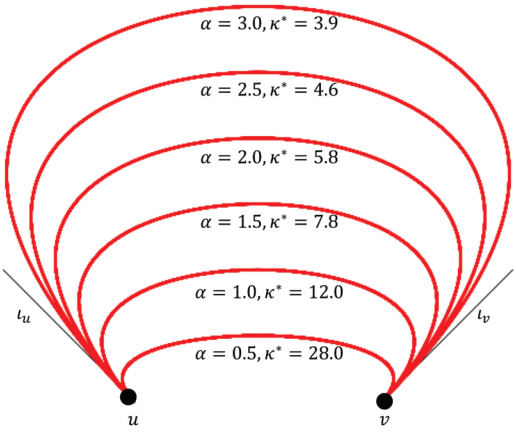


Fig. 17. "Ballooning" curves: as $\alpha = \lambda_u = \lambda_v$ increases, the curvature falls but the curve becomes very long.

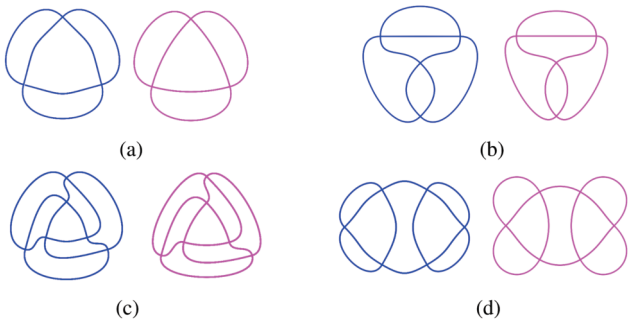


Fig. 18. Comparison of proportional arm length (blue, $\alpha = 0.5$) and optimal arm length (magenta): (a) Trefoil; (b) K_4 knot; (c) 3-prism; (d) Love knot.

λ_u and λ_v at u and v respectively. From Equation (2), we note that

$$\frac{\partial}{\partial t}\kappa\left(t, \lambda_u, \lambda_v\right) = \left| \frac{\dddot{x}\dot{y} - \dddot{y}\dot{x}}{\left(\dot{x}^2 + \dot{y}^2\right)^{1.5}} - \frac{\ddot{x}\dot{y} - \ddot{y}\dot{x}}{3\left(\dot{x} + \ddot{y}\right)^{2.5}} \right| \tag{5}$$

as long as $(\dot{x}^2 + \dot{y}^2) \neq 0$ and $\ddot{x}\dot{y} \neq \ddot{y}\dot{x}$. Since both x and y are cubic functions of t, equation (5) is not as complex as it seems, and it is straightforward (but tedious, because of the edge cases) to maximize $\kappa(t, \lambda_u, \lambda_v)$ over t; that is, to find the maximum curvature $\kappa^*(\lambda_u, \lambda_v)$:

$$\kappa^*\left(\lambda_u, \lambda_v\right) = \max_{0 \leq t \leq 1} \kappa\left(t, \lambda_u, \lambda_v\right).$$

Now we want to choose the arm lengths λ_u and λ_v to minimize $\kappa^*(\lambda_u, \lambda_v)$. Let ι_u and ι_v be the unit vectors in the directions of the appropriate arms of χ_u and χ_v. We can express the internal control points p_1 and p_2 of the Bézier curve in terms of λ_u and λ_v:

$$p_1 = u + \lambda_u \iota_u, \quad p_2 = v + \lambda_v \iota_v.$$

In this way, $\kappa^*(\lambda_u, \lambda_v)$ is linear in both λ_u and λ_v and finding a minimum point for $\kappa^*(\lambda_u, \lambda_v)$ is straightforward.

However, in some cases, an edge with globally minimum maximum curvature may not be desirable. In Fig. 17, for example, the curvature decreases as λ_u and λ_v increase; for large values of λ_u and λ_v the curvature is quite low. The problem is that these large values make the curve very long (it "balloons" out), which might also cause unintended edge crossings.

For this reason, we choose an upper bounds ϵ_u and ϵ_v and take a minimum constrained by $\lambda_u \leq \epsilon_u$ and $\lambda_v \leq \epsilon_v$:

$$\kappa^*_{\min} = \min_{0 \leq \lambda_u \leq \epsilon_u, 0 \leq \lambda_v \leq \epsilon_v} \kappa^*\left(\lambda_u, \lambda_v\right).$$

We have found that $\epsilon_u = \epsilon_v = 0.75d(u, v)$ gives good results, where $d(u, v)$ is the distance between u and v. Values of λ_u and λ_v that achieve the (constrained) minimum $\kappa^*_{\min}$ are then used by the Bézier curves. These bounds help mitigate the "ballooning" effect and prevent most unintended edge crossings. However, they do not provide a complete guarantee. To eliminate unintended edge crossings entirely, we calculate the intersection points of each pair of curves. If an intersection is found, the arm lengths of the intersecting curves must be adjusted to ensure they no longer cross. In practice, using such optimal arm lengths gives better results than using uniformly proportional arm lengths. In some cases the difference is not significant, but in others the optimal edges appear to be much smoother. See Fig. 18 for examples.

VI. Generating 4-Regular Plane Graphs

In the previous sections we presented a novel procedure for drawing a 4-regular planar graph as a knot/link diagram. Celtic interlace patterns can be quite complex; we wish to define operations to build very large and complex Celtic knots. With this in mind, we define two operations *edge-disentanglement* and *combine* for the creation and modification of 4-regular plane graphs. These operations aim to be easily implementable in an interactive tool. The edge-disentanglement operations, shown in Fig. 19, extends such a graph outwards through its outer face with minor alterations to its inner structure. The combine operation, shown in Fig. 20, merges two 4-regular plane graphs. In fact, we show that these operations can generate *every* 4-regular plane graph in Section VI-C.

A. Edge-Disentanglement

Here, we define the two edge-disentanglement operations O_1 and O_2 along with their inverse operations O_1^{-1} and O_2^{-1}.

Let $G = (V, E)$ be a 4-regular plane graph.

- O_1: Choose a vertex u and two edges $e_1 = (u, u_1)$ and $e_2 = (u, u_2)$ incident to vertex u, such that e_1 and e_2 lie on the same face of G, and u is not incident with a self-loop. Then delete edges e_1 and e_2, and add an an edge (u_1, u_2) and a self loop (u, u). This is illustrated in Fig. 19(a).
- O_2: Choose a vertex u and two edges $e_1 = (u, u_1)$ and $e_2 = (u, u_2)$ incident with vertex u, such that u is incident with a self-loop e, $e \neq e_1$, $e \neq e_2$, and e, e_1, and e_2 lie on the same face of G. Then delete the vertex u as well as edges e_1 and e_2, and add an edge (u_1, u_2). This is illustrated in Fig. 19(b).

Note that the resulting graph is a 4-regular plane graph for both operations.

For each of O_1 and O_2, we have an inverse operation:

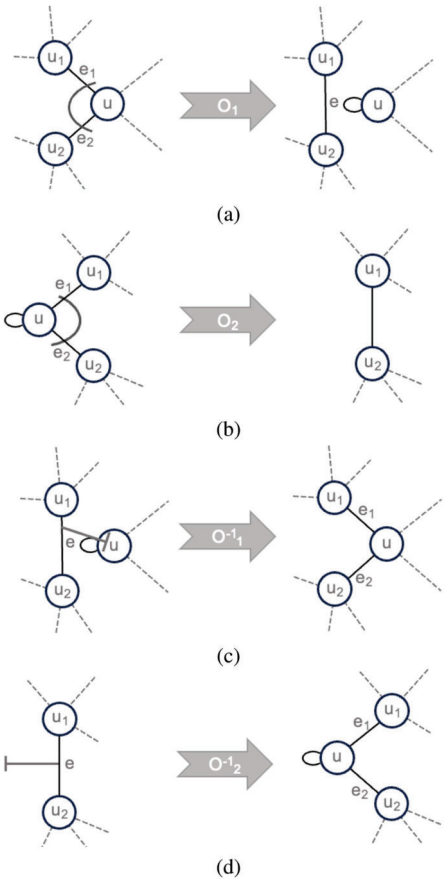


Fig. 19. Operations O_1, O_1^{-1}, O_2, and O_2^{-1}. The curve indicates which two edges are used for O_1 or O_2. The T-Stick indicates onto which vertex O_1^{-1} is performed or where the newly created vertex of O_2^{-1} lies.

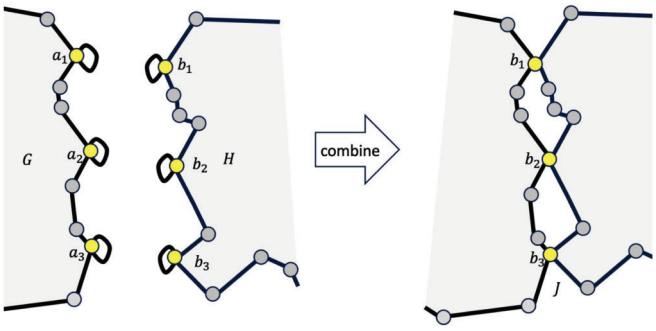


Fig. 20. Combine operation.

- O_1^{-1}: Suppose that $e = (u_1, u_2)$ is an edge of G and u is a vertex in G such that u is incident with a self-loop e', and e' shares a face with e. Then we remove e and e', and add edges $e_1 = (u, u_1)$ and $e_2 = (u, u_2)$.
- O_2^{-1}: Choose an edge $e = (u_1, u_2)$, add a new vertex u, replace e with two edges $e_1 = (u, u_1)$ and $e_2 = (u, u_2)$, then add a self-loop (u, u) to u.

Again, the graphs formed by operations O_1^{-1} and O_2^{-1} are clearly 4-regular plane graphs.

B. Combine

A vertex $a \in V$ of a 4-regular plane graph $G = (V, E)$ is *active* if a is on the outside face and is incident with a self-loop on the outside face. Suppose that $G = (V, E)$ and $H = (U, F)$ are two 4-regular plane graphs, and $A = (a_1, a_2, \ldots, a_k)$ and $B = (b_1, b_2, \ldots, b_k)$ are lists of active vertices in G and H respectively, with $|A| = |B|$. We also require that vertices in A appear clockwise and vertices in B are in counter-clockwise order around the outside face.

We *combine* G and H with respect to A and B by identifying vertex a_i in A with vertex b_i in B for $1 \leq i \leq k$. We create a new graph J whose vertices set is $(V \cup U) \setminus A$, and edge set is $(E \setminus E_A) \cup (F \setminus F_B)$, where E_A is the set of self-loops incident with vertices in A, F_B is the set of self-loops incident with vertices in B. The combine operation is illustrated in Fig. 20. Note that combining two 4-regular plane graphs yields a single 4-regular plane graph

C. Completeness

Having introduced both operations, we will now demonstrate that the edge-disentanglement operation alone is sufficient to construct all possible four-regular plane multi-graphs. Additionally, the combine operation provides an efficient and straightforward method to merge large sub-patterns without affecting the completeness of the edge-disentanglement operation, as combining two four-regular plane multi-graphs results in a four-regular plane multi-graph. Formally, we present the following theorem:

Theorem 2: Any four-regular plane multi-graph can be constructed, starting with the one-vertex graph with two self-loops, using a finite series of (inverse-) edge-disentanglement operations (and the combine operation).

Proof: We prove this by induction on the number of vertices in G first and the number of non-self-looping edges, denoted E', second. That is, assume that our claim holds for any four-regular plane graph with less than n vertices or with n vertices and less than k non-self-looping edges (*normal edges*), including multi-edges.

Let $G = (V, E)$ be a four-regular plane graph with $n > 1$ vertices and k normal edges. Let us now distinguish the three possible cases that might occur.

Case 1. G is disconnected: Let G be a disconnected graph. By our induction hypothesis, we know that each connected component can be created individually using a series of edge-disentanglement operations. Thus, our claim holds.

Case 2.1. G is connected and contains self-loops: Let u be a vertex incident to an arbitrary self-loop e and the two edges $e_1 = (u, u_1)$ and $e_2 = (u, u_2)$. Performing O_2 on u and the two edges e_1, e_2 results by definition in a graph $G' = (V', E')$, with $n - 1'$ vertices. Hence G' can, by induction, be created using the edge-disentanglement operations, and our claim holds.

Case 2.2. G is connected and contains no self-loops: Let u be any vertex incident to two arbitrary edges $e_1 = (u, u_1)$ and $e_2 = (u, u_2)$. Performing O_1 on u and e_1, e_2 per definition results in a graph $G' = (V', E')$, with $k - 1'$ normal edges, which again can by induction be created

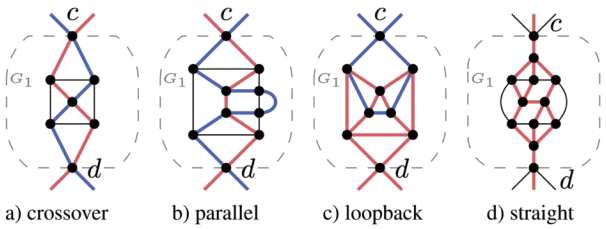


Fig. 21. Examples of the four types of subgraphs separated by a separation pair. The type of a subgraph is determined by a perfect matching on the edges in its cut; there are three ways to match four edges, and one way to match two edges. Threaded circuits crossing the cut are traced in blue and red.

using the edge-disentanglement operations and our claim holds.

Since these are the only possible cases and each edge disentanglement operation results in a 4-regular plane graph, our claim holds for any 4-regular plane graph with n vertices and k normal edges. Therefore, by induction, the claim is valid for all 4-regular plane graphs. □

VII. The Cardinality of a Threaded Circuit Partition

Here we provide a proof for the claim made in Section IV; namely that the *number* of threaded circuits in any threaded circuit partition of a 4-regular plane graph G is fixed by the combinatorial structure of G, and cannot be changed by taking a different topological embedding of G.

Since there is no choice of topological embedding for 3-connected graphs [66] we will be concerned only with 1- and 2-connected graphs. We will not consider disconnected graphs; in that case each connected component should be treated separately. Clearly a threaded circuit cannot span multiple connected components, so the cardinality of a threaded circuit partition of a disconnected graph is exactly the sum of the cardinalities of the threaded circuit partitions of its components.

Suppose that G is a 4-regular planar graph, and c is a cutpoint separating components G_1 and G_2 in G. Note that c has degree 2 in both G_1 and G_2, because a 4-regular graph is bridgeless. Now suppose that the two edges incident with c in G_1 are e_1 and e_2. For the threaded circuit containing e_1 to form a valid circuit, it must eventually include a second edge incident on vertex c. This edge could either be e_2 or one of the other two edges incident on c. However, both of these edges originate in G_2, and since c is a cut vertex, a circuit can only pass from G_1 to G_2 through an edge connecting G_1 to c. Therefore the threaded circuit containing e_1 must also contain e_2.

Pivoting G_1 around the cut vertex c connects e_1 to the threaded circuit formerly connected to e_2 and vice versa. However, since e_1 and e_2 are part of the *same* threaded circuit this pivot does not change the number of threaded circuits; in fact we can deduce the following theorem.

Theorem 3: Suppose that G is a 4-regular planar graph, and c is a cutpoint separating components G_1 and G_2 in G; denote by G'_i the 4-regular planar graph formed by adding a self-loop to c in G_i, for $i = 1, 2$. Suppose that $\hat{G}$ is a topological embedding of G, and $\hat{G}'_1$ and $\hat{G}'_2$ are plane subgraphs of $\hat{G}$ corresponding to G'_1 and G'_2 respectively. Suppose that $\hat{G}$, $\hat{G}'_1$ and $\hat{G}'_2$ have threaded circuit partitions C, C_1, and C_2 respectively. Then $|C| = |C_1| + |C_2| - 1$.

In contrast, "mirroring" about a *separation pair* [51] can change the threaded circuits; as demonstrated in Fig. 11. However, we will now show that the cardinality of a threaded circuit partition is a *graph property*, not an embedding property; that is, changing the embedding does not change the *number* of threaded circuits in a threaded circuit partition.

Theorem 4: Suppose that G is a 2-connected 4-regular planar graph, and C and C' are threaded circuit partitions of two topological embeddings of G. Then $|C| = |C'|$.

Proof: For a 2-connected planar graph, all topological embeddings can be found by "pivoting" (re-ordering and "mirroring") 3-connected components around separation pairs.[2] We show that such a pivoting does not change the cardinality of a threaded circuit partition.

Suppose that vertices c and d form a separation pair that separates G into two components, G_1 and G_2. We will show that pivoting G_2 neither creates new threaded circuits nor merges existing threaded circuits and hence does not change the number of threaded circuits.

First note that pivoting G_1 at c and d does not change any threaded circuit not incident on c or d; that is, threaded circuits entirely within G_1 or G_2 are unaffected. As G_1 is 4-regular (except for c and d), by the handshake lemma either *both* c and d have degree 2 in G_1 or they both have odd degree. If c has degree 3 in G_1 it has a unique neighbor c' that is not in G_1; in this case we consider the separation pair c' and d instead. Thus we may assume without loss of generality that c has degree 1 in G_1; the same argument holds for d. Therefore, either both c and d have degree 2 in G_1, or both c and d have degree 1 in G_1.

Every threaded circuit that enters G_1 must somehow exit in order to form a valid circuit. If c and d have degree 2, selecting one "entering" cut edge (that is, an edge in G_2 incident to c or d) leaves three possible edges it may use to "exit" G_1; fixing an exit edge forces the remaining two cut edges to be part of the same threaded circuit. On the other hand, if the c and d have degree 1, there is only a single possible connection: the circuit entering at c must exit at d. Consequently, considering both cases, one can deduce that, since the graph is 4-regular and planar, there are only four possible "interfaces" that G_1 can present to G_2; examples are enumerated in Fig. 21. The names "crossover", "parallel", "loopback", and " straight" reflect how G_1 appears to G_2.

To complete the proof, note that pivoting each type of subgraph does not change the cardinality of the threaded circuit partition, although it can change which of two paths connects to which threaded circuit, and hence the number of edges in the circuits. □

VIII. `CelticGraph` Implementation as a `Vanted` Add-On and Rendering

`CelticGraph` has been implemented as an add-on of `Vanted` [38], a tool for interactive visualization and analysis

[2] This is a well-known "folklore" result; we have not been able to determine the first proof of it. This result is used, for example, in [18].

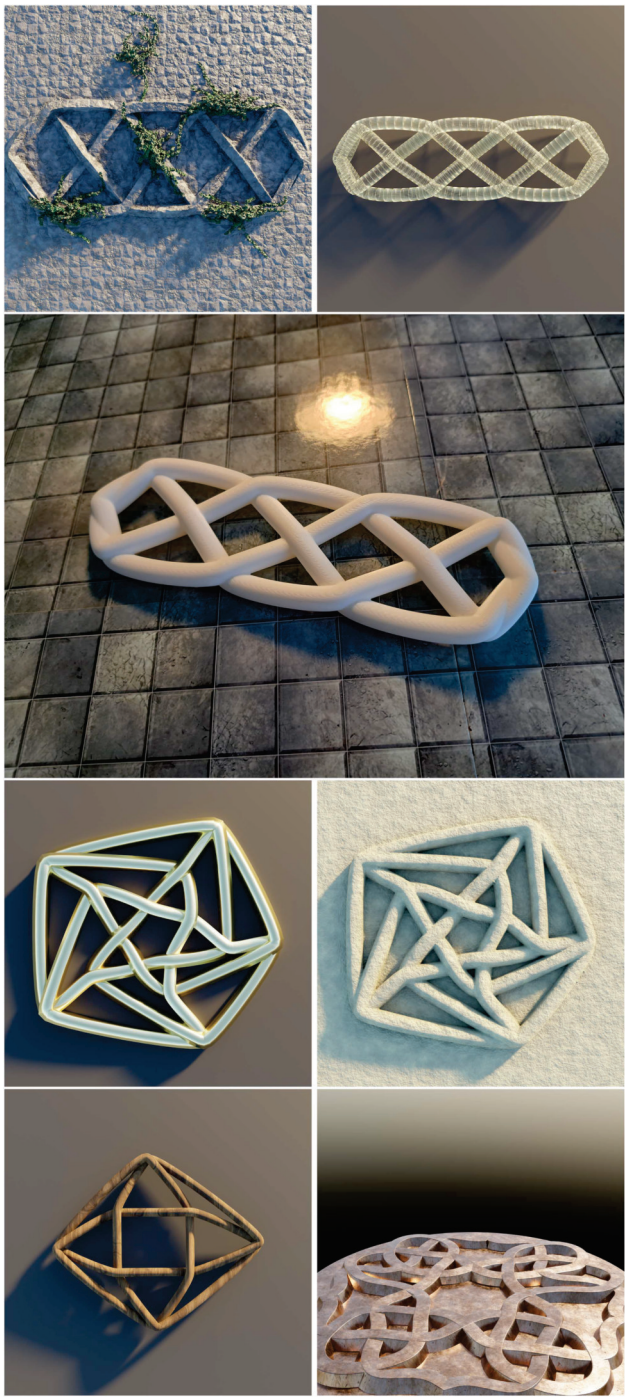


Fig. 22. Examples of Celtic knot renderings in different media including a 3D printed version (second row from top). The knot bottom right has been inspired by the central round part of the Belt Buckle Sutton Hoo in Fig. 3(b).

of networks. Fig. 1 shows an example workflow; the first step is implemented as a `Vanted` add-on, the second is done by Blender [7].

`Vanted` allows a user to load or create 4-regular graphs, either by importing from files (e. g. a. gml file), by selecting from examples, or by creating a new graph by hand. The individual vertices of the graph are then mapped into the data structure of a cross, containing position and the rotation and control points of the to-be-generated Bézier curves. The graph is translated into a knot (links) using the methods for optimal cross rotation and arm length computation described in Section V. Vertex positions can be interactively changed, either by interacting with the underlying graph, or by interacting directly with the visualization of the knot. Once a visualization satisfies the expectation of the user, the Bézier curves can be exported for further use in Blender.

We implemented a Python script and a geometry node tree in Blender which allows the import of information into Blender and rendering the knot (links), either using a set of predefined media or by user interaction; the script can also run as a batch process with selected parameters and media. Fig. 22 shows examples of Celtic knot renderings in different media such as in metal, in stone, with additional decoration and so on; knots can be also printed in 3D. More examples can be found in the gallery of our web page http://celticknots.online which also provides the `Vanted` add-on, Blender file and a short manual.

IX. Conclusion

This paper introduces `CelticGraph`, a framework for creating aesthetically pleasing pictures of 4-regular planar graphs in the style of Celtic knots and links. We show how to create these drawings and provide a construction mechanism to build any 4-regular plane graph. Further, `CelticGraph` uses a novel algorithm to represent edges as Bézier curves, aiming to show each link as a smooth curve with limited curvature. As shown in the gallery, `CelticGraph` and subsequent 3D rendering (with Blender) or 3D printing provide aesthetically pleasing pictures or objects of Celtic knots and links.

There are several directions the framework could be potentially extended:

- `CelticGraph` could be extended to handle vertices of degree 2 by adding dummy self-loops at each such vertex. This would extend the classes of planar graphs covered by the framework and enable us to include visualizations of specific angle-shaped endpoints commonly occurring in Celtic ornaments.
- The framework could be extended to support Celtic knot input and manipulation via the Gauss code mentioned in Section II-E3. This could provide another construction mechanism for Celtic knots.
- `CelticGraph` is a stand-alone tool. It could be included in an interactive web platform where users can create, share, and discuss their Celtic knot designs. This could foster a community of enthusiasts interested in Celtic art and graph theory.
- The framework could be extended to enable the creation of animated Celtic knots, where the knot design evolves dynamically based on user input or data.
- We finally note that the observation in Theorem 4 might provide an extension to the simple linear-time algorithm in Section IV that tests whether a graph has a threaded Euler circuit (in any embedding). Using a SPQR tree [51], one could decompose the graph into pivotable components and solve the following optimization problem: Given a

2-connected 4-regular planar graph, find a topological embedding in which the length of the longest threaded circuit is maximized.

Each of these extensions may pave the way for additional research in fields like the interplay between graphs and Celtic knots. It is hoped that this paper has sparked sufficient interest to attract more researchers to explore the link between graphs and (Celtic) art.

Acknowledgment

The authors would like to thank the anonymous referees for their helpful comments, especially for the suggestions to simplify one proof.

References

[1] J. Abello and E. Gansner, "Short and smooth polygonal paths," in *Proc. Theor. Inform., 3rd Latin Amer. Symp.*, Springer, 1998, pp. 151–162, doi: 10.1007/BFb0054318.

[2] C. C. Adams, *The Knot Book: An Elementary Introduction to the Mathematical Theory of Knots*. Providence, RI, USA: American Mathematical Society, 2004.

[3] M. Andreeva, I. Dynnikov, S. Koval, K. Polthier, and I. Taimanov, "Book knot simplifier," Accessed: Jan. 16, 2025. [Online]. Available: http://www.javaview.de/services/knots/doc/description.html

[4] G. Bain, *Celtic Art: The Methods of Construction*. New York, NY, USA: Dover, 1973.

[5] I. Bain, *Celtic Knotwork*. New York, NY, USA: Sterling, 1986.

[6] A. Bartholomew, "Andrew bartholomew's mathematics page," Accessed: Jan. 16, 2025. [Online]. Available: https://www.layer8.co.uk/maths/

[7] Online BlenderCommunity, *Blender–a 3D Modelling and Rendering Package*. Amsterdam, Netherlands: Blender Foundation, 2018.

[8] J. Boyer and W. Myrvold, "On the cutting edge: Simplified O(n) planarity by edge addition," *J. Graph Algorithms Appl.*, vol. 8, no. 3, pp. 241–273, 2004, doi: 10.7155/jgaa.00091.

[9] U. Brandes, G. Shubina, and R. Tamassia, "Improving angular resolution in visualizations of geographic networks," in *Proc. Data Visualization Eurographics*, Springer, 2000, pp. 23–32, doi: 10.1007/978-3-7091-6783-0_3.

[10] U. Brandes and D. Wagner, "Using graph layout to visualize train interconnection data," *J. Graph Algorithms Appl.*, vol. 4, pp. 135–155, 2000, doi: 10.1007/3-540-37623-2_4.

[11] C. C. Cheng, C. A. Duncan, M. T. Goodrich, and S. G. Kobourov, "Drawing planar graphs with circular arcs," *Discrete Comput. Geometry*, vol. 25, no. 3, pp. 405–418, 2001, doi: 10.1007/s004540010080.

[12] H. Clark, *A Short and Easy Introduction to Heraldry*. London, U.K.: H. Washbourn, 1827.

[13] G. Costagliola, M. De Rosa, A. Fish, V. Fuccella, R. Saleh, and S. Swartwood, "Knotsketch: A tool for knot diagram sketching, encoding and re-generation," *J. Vis. Lang. Sentient Syst.*, vol. 2, pp. 16–25, 2016, doi: 10.18293/DMS2016-035.

[14] Cytoscape, Accessed: Jan. 16, 2025. [Online]. Available: https://cytoscape.org

[15] H. de Fraysseix, J. Pach, and R. Pollack, "How to draw a planar graph on a grid," *Combinatorica*, vol. 10, no. 1, pp. 41–51, 1990, doi: 10.1007/BF02122694.

[16] L. Devroye and P. Kruszewski, "The botanical beauty of random binary trees," in *Proc. Symp. Graph Drawing*, Springer, 1995, pp. 166–177, doi: 10.1007/BFb0021801.

[17] G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis, *Graph Drawing: Algorithms for the Visualization of Graphs*. Englewood Cliffs, NJ, USA: Prentice Hall, 1999.

[18] G. Di Battista and R. Tamassia, "On-line planarity testing," *SIAM J. Comput.*, vol. 25, no. 5, pp. 956–997, 1996, doi: 10.1137/S0097539794280736.

[19] C. A. Duncan, D. Eppstein, M. T. Goodrich, S. G. Kobourov, and M. Nöllenburg, "Lombardi drawings of graphs," in *Proc. 18th Int. Symp. Graph Drawing*, Springer, 2010, pp. 195–207, doi: 10.1007/978-3-642-18469-718.

[20] P. Eades, S.-H. Hong, M. McGrane, and A. Meidiana, "GDot-i: Interactive system for dot paintings of graphs," in *Proc. Eurographics Conf. Visualization*, The Eurographics Association, 2022, pp. 5–6, doi: 10.2312/evp.20221106.

[21] C. Even-Zohar, J. Hass, N. Linial, and T. Nowik, "Universal knot diagrams," *J. Knot Theory Ramifications*, vol. 28, no. 07, 2019, Art. no. 1950031, doi: 10.1142/S0218216519500317.

[22] R. Ferguson, "An easier derivation of the curvature formula from first principles," *Australian Senior Math. J.*, vol. 32, pp. 16–22, 2018.

[23] M. Fink, H. Haverkort, M. Nöllenburg, M. Roberts, J. Schuhmann, and A. Wolff, "Drawing metro maps using Bezier curves," in *Proc. 20th Int. Symp. Graph Drawing*, Springer, 2013, pp. 463–474, doi: 10.1007/978-3-642-36763-2_41.

[24] L. Finke and E. Weitz, "A phenomenological approach to interactive knot diagrams," *IEEE Trans. Vis. Comput. Graph.*, vol. 30, no. 8, pp. 5901–5907, Aug. 2024, doi: 10.1109/TVCG.2024.3405369.

[25] B. Finkel and R. Tamassia, "Curvilinear graph drawing using the force-directed method," in *Proc. 12th Int. Conf. Graph Drawing*, Springer, 2004, pp. 448–453, doi: 10.1007/978-3-540-31843-9_46.

[26] J. D. Foley, A. van Dam, S. Feiner, and J. F. Hughes, *Computer Graphics - Principles and Practice*, 2nd Ed. Reading, MA, USA: Addison-Wesley, 1990, doi: 10.1007/978-3-7091-2684-4_31.

[27] J. Fries-Knoblach, *Die Kelten*. Stuttgart, Germany: Kohlhammer-Urban, 2012, pp. 138–142, doi: 10.36198/9783838543543.

[28] A. Glassner, "Celtic knotwork, part 1," *IEEE Comput. Graph. Appl.*, vol. 19, no. 5, pp. 78–84, Sep./Oct. 1999, doi: 10.1109/38.788804.

[29] M. T. Goodrich and C. G. Wagner, "A framework for drawing planar graphs with curves and polylines," in *Proc. 6th Int. Symp. Graph Drawing*, Springer, 1998, pp. 153–166, doi: 10.1006/jagm.2000.1115.

[30] Graphviz, Accessed: Jan. 16, 2025. [Online]. Available: https://graphviz.org/

[31] Gridlink, Accessed: Jan. 16, 2025. [Online]. Available: https://homepages.math.uic.edu/culler/gridlink/

[32] R. K. S. Hankin, "Visually pleasing knot projections," *J. Math. Arts*, vol. 17, no. 3/4, pp. 314–332, 2023, doi: 10.1080/17513472.2023.2185058.

[33] S.-H. Hong, P. Eades, and M. Torkel, "GDot: Drawing graphs with dots and circles," in *Proc. IEEE 14th Pacific Visual. Symp.*, 2021, pp. 156–165, doi: 10.1109/PacificVis52677.2021.00029.

[34] J. Hoste and M. Thistlethwaite, Knotscape,Knotscape, Accessed: Jan. 16, 2025. [Online]. Available: https://pzacad.pitzer.edu/jhoste/HosteWebPages/kntscp.html

[35] V. Irvine, T. Biedl, and C. S. Kaplan, "Quasiperiodic bobbin lace patterns," *J. Math. Arts*, vol. 14, no. 3, pp. 177–198, 2020, doi: 10.1080/17513472.2020.1752999.

[36] I. M. James, ed. *History of Topology*. Amsterdam, The Netherlands: North Holland, 1999, doi: 10.1016/B978-0-444-82375-5.X5000-7.

[37] M. Jünger and P. Mutzel, "Maximum planar subgraphs and nice embeddings: Practical layout tools," *Algorithmica*, vol. 16, no. 1, pp. 33–59, 1996, doi: 10.1007/BF02086607.

[38] B. H. Junker, C. Klukas, and F. Schreiber, "VANTED: A system for advanced data analysis and visualization in the context of biological networks," *BMC Bioinf.*, vol. 7, no. 109, pp. 1–13, 2006, doi: 10.1186/1471-2105-7-109.

[39] P. Kindermann, S. Kobourov, M. Löffler, M. Nöllenburg, A. Schulz, and B. Vogtenhuber, "Lombardi drawings of knots and links," in *Proc. 26th Int. Symp. Graph Drawing Netw. Visualization*, Springer, 2018, pp. 113–126, doi: 10.1007/978-3-319-73915-1_10.

[40] R. Klempien-Hinrichs and C. von Totth, "Generation of celtic key patterns with tree-based collage grammars," *Manipulation Graphs, Algebras Pictures, Electron. Commun. Eur. Assoc. Softw. Sci. Technol.*, vol. 26, pp. 205–222, 2010, doi: 10.14279/tuj.eceasst.26.356.

[41] "Further knot theory software - knot atlas," Accessed: Jan. 16, 2025. [Online]. Available: http://katlas.org/wiki/Further_Knot_Theory_Software#Knotscape

[42] K. Koffka, *Principles of Gestalt Psychology*. Orlando, FL, USA: Harcourt Brace and Co., 1935.

[43] W. B. R. Lickorish, *An Introduction to Knot Theory*. Berlin, Germany: Springer, 1997, doi: 10.1007/978-1-4612-0691-0.

[44] H. Liu and H. Zhang, "A suggestive interface for untangling mathematical knots," *IEEE Trans. Visualization Comput. Graph.*, vol. 27, no. 2, pp. 593–602, 2021, doi: 10.1109/TVCG.2020.3028893.

[45] B. Maier, *Die Kelten–Geschichte, Kultur und Sprache*, Freudenberg, Germany: UTB, 2015, pp. 158–159, doi: 10.36198/9783838543543.

[46] K. Mehlhorn and P. Mutzel, "On the embedding phase of the Hopcroft and Tarjan planarity testing algorithm," *Algorithmica*, vol. 16, no. 2, pp. 233–242, Aug. 1996, doi: 10.1007/BF01940648.
[47] K. Miller, KnotFolio, Accessed: Jan. 16, 2025. [Online]. Available: https://knotfol.io
[48] "Monks of St Columba's order of Iona," in *The Book of Kells*. Dublin Ireland: Old Library in Trinity College, 9th century.
[49] K. Murasugi, *Knot Theory and its Applications*, Boston, MA, USA: Birkhäuser, 1996, doi: 10.1007/978-0-8176-4719-3.
[50] P. Mutzel, "A fast O(n) embedding algorithm, based on the Hopcroft-Tarjan planary test," Informatik, Universität zu Köln, Tech. Rep. 92-107, 1992.
[51] P. Mutzel, "The SPQR-tree data structure in graph drawing," in *Proc. 30th Int. Colloq. Automata, Lang. Program.*, Springer, 2003, pp. 34–46, doi: 10.1007/3-540-45061-0_4.
[52] F. Müller, *Die Kunst Der Kelten*. München, Germany: C. H. Beck Wissen, 2012.
[53] T. Nishizeki and M. S. Rahman, *Planar Graph Drawing*, Singapore: World Scientific, 2004, doi: 10.1142/5648.
[54] "The open graph drawing framework ODGF," Accessed: Jan. 16, 2025. [Online]. Available: https://ogdf.uos.de
[55] H. Purchase, J. Hamer, M. Nöllenburg, and S. Kobourov, "On the usability of lombardi graph drawings," in *Proc. 20th Int. Symp. Graph Drawing*, 2012, pp. 451–462, doi: 10.1007/978-3-642-36763-2_40.
[56] S. Rieckhoff and J. Biel, *Die Kelten in Deutschland*. Stuttgart, Germany: Konrad Theiss Verlag, 2001, pp. 197–206, doi: 10.1515/ZCPH.2005.245.
[57] R. Scharein, "KnotPlot," 2005. Accessed: Jan. 16, 2025. [Online]. Available: https://knotplot.com
[58] T. Rolland and F. De VicoFallani, "Vizaj—a free online interactive software for visualizing spatial networks," *PLoS One*, vol. 18, no. 3, 2023, Art. no. e0282181, doi: 10.1371/journal.pone.0282181.
[59] P. Rosenstiehl, "A new proof of the Gauss interlace conjecture," *Adv. Appl. Math.*, vol. 23, no. 1, pp. 3–13, 1999, doi: 10.1006/aama.1999.0643.
[60] R. Röber, *Exhibition Catalogue Archäologisches Landesmuseum Stuttgart: Die Welt der Kelten. Zentren der Macht–Kostbarkeiten der Kunst*. Ostfildern, Germany: Jan Thorbecke Verlag, 2012, pp. 512–515.
[61] C. Seed, "Knotkit," Accessed: Jan. 16, 2025. [Online]. Available: https://github.com/cseed/knotkit
[62] A. Sossinsky, *Knots, Mathematics With a Twist*. Cambridge, MA, USA: Harvard Univ. Press, 2002.
[63] S. Stein, "Knoty," Accessed: Jan. 18, 2025. [Online]. Available: https://stani-stein.com/knoty/
[64] F. Swenton, "KLO (knot-likeobjects)," 2019. Accessed: Jan. 16, 2025. [Online]. Available: https://cat.middlebury.edu/mathanimations/klo/
[65] J. van Wijk and A. Cohen, "Visualization of the genus of knots," in *Proc. IEEE Visual.*, 2005, pp. 567–574, doi: 10.1109/VISUAL.2005.1532843.
[66] H. Whitney, "2-isomorphic graphs," *Amer. J. Math.*, vol. 55, pp. 245–54, 1933, doi: 10.1007/978-1-4612-2972-8_8.
[67] K. Xu, C. Rooney, P. Passmore, D.-H. Ham, and P. H. Nguyen, "A user study on curved edges in graph visualization," *IEEE Trans. Vis. Comput. Graphics*, vol. 18, no. 12, pp. 2449–2456, Dec. 2012, doi: 10.1109/TVCG.2012.189.
[68] yEd graph editor. Accessed: 16 Jan., 2025. [Online]. Available: https://www.yworks.com/products/yed
[69] H. Zhang, J. Weng, L. Jing, and Y. Zhong, "KnotPad: Visualizing and exploring knot theory with fluid reidemeister moves," *IEEE Trans. Vis. Comput. Graphics*, vol. 18, no. 12, pp. 2051–2060, Dec. 2012, doi: 10.1109/TVCG.2012.242.