MARTIN HAFSKJOLD THORESEN, ISTA, Austria

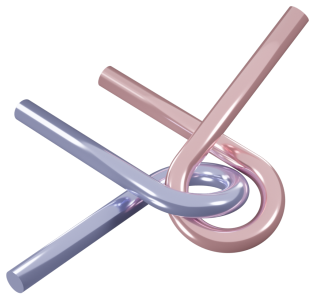
ACM Reference Format:

Martin Hafskjold Thoresen. 2022. . 1, 1 (August 2022), 3 pages.

1 DESIGN OF WIRE PUZZLES

In the past decade, graphics researchers have developed a wide range of computational tools for puzzle design, including types of puzzles like interlocking puzzles [Chen et al. 2022; Song et al. 2012], centrifugal puzzles [Kita and Saito 2020], polymino puzzles [Lo et al. 2009], and jigsaw puzzles [Elber and Kim 2022]. Puzzles are mainly recreational, but techniques and ideas developed in the context of puzzle design are often transferable to other domains.

In this project we want to design wire puzzles, which is a kind of entanglement puzzle consisting of rigid space curves that are tangled up (Figure 1), and the goal of the puzzle is to untangle them. A method for solving wire puzzles has been proposed [Zhang et al. 2020], but techniques for design are still missing. One challenge with wire puzzle design is that they consist of smooth curves which twist and slide in 3 dimensions. This contrasts the literature on puzzle design which mainly deal with discrete and often voxelized pieces that only move in orthogonal directions, so current techniques for computational puzzle design are not applicable to wire puzzles.


(a) The *alpha* puzzle.

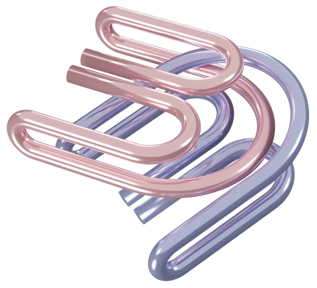

(b) The *devil's claw* puzzle.

Fig. 1. Two examples of wire puzzles. Both puzzles contain two identical pieces. The goal of the puzzle is to untangle the two pieces.

1.1 The Key Idea

The key idea we propose is to decouple the design of the puzzle with the geometry of the wire, and it is motivated by the following observation also made by Zhang et al. [2020]: the pieces in the *alpha* puzzle (Figure 1a) contain a gap where the wire cross, and solving the puzzle involves realizing that while the gap is too small for a wire to cross under, one can align the two gaps of the two pieces and twist the blue loop out of the red loop so that it is trivially removable (Figure 2). In a similar way, to solve *devil's claw* (Figure 1b) one needs to align the two long tunnels in an orthogonal fashion and slide one piece out of the other. We claim that once this observation is made, the puzzles are relatively easy as the only thing that remains is aligning the pieces.

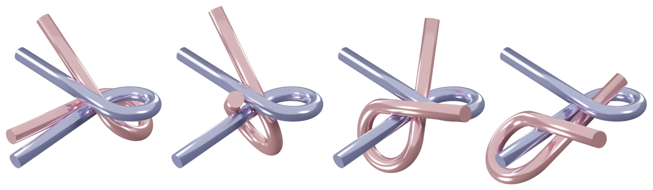


Fig. 2. The key part of the solution to the *alpha* puzzle is when the gap of the two pieces are aligned, which allows a twisting motion.

1.2 An Intrinsic Representation

What does it mean to design a wire puzzle without geometry? We can model the key idea directly by representing the gap as a Gap node in a "knot-like" graph representation (Figure 3). We call this the *intrinsic* puzzle, and the representation its *intrinsic representation* (IR). The graph models only models the topology of the puzzle, like connectivity and the different types of mechanisms that the we can have in wire puzzles, like the gap in *alpha* as a Gap node, or the tunnel in *devil's claw* as a Tunnel node.

There are multiple reasons for why we propose to design wire puzzles this way: (1) it allows the designer to focus on the intrinsic part of the puzzle, which we claim is the more interesting part; (2) the time required for design is reduced since the designer does not have to manually ensure that the geometry conforms to the intrinsic specification; (3) it has a very natural discrete representation which significantly reduces the complexity of computing solutions to a design.

1.3 Geometry

Still, we need wire geometry to realize the puzzle designs. Our observation is that many of the geometrical aspects of the puzzles are either of low importance or have significant leeway. In *alpha*, the gap has to be small enough for a wire not to cross and large enough for the twist to be possible, but within the range it could vary freely. Further, the size or shape of the loop, or the length of the straight

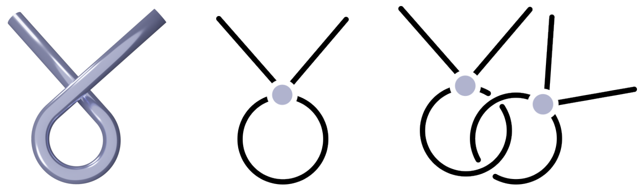


Fig. 3. A single piece from the *alpha* puzzle (left), our proposed intrinsic representation of the piece (middle), and the representation of the *alpha* puzzle (right). The blue node represents the gap where the wire cross.

Author's address: Martin Hafskjold Thoresen, mthorese@ista.ac.at, ISTA, Am Campus 1, Klosterneuburg, Austria.

2022. XXXX-XXXX/2022/8-ART $15.00
https://doi.org/

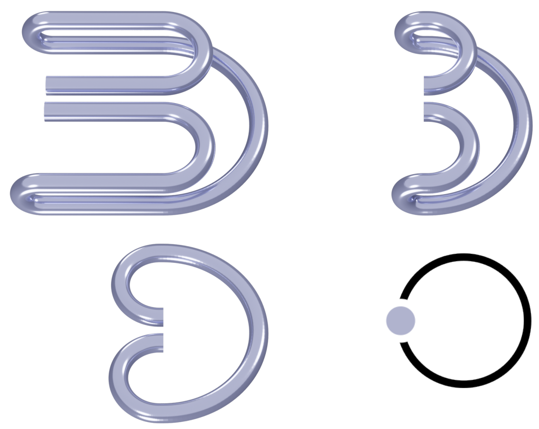


Fig. 4. The *devil's claw* can be deformed into a much simpler puzzle without breaking the intrinsic solution of the puzzle. We first collapse the straight rods of the puzzle to have ϵ length. Then we flatten the bend so that the whole structure is planar. The simplification changes the puzzles perceived difficulty dramatically, to being almost trivial. We conclude that changing the geometry of the puzzle is necessary to create interesting puzzles.

wires, are also not integral to the puzzle, although they do have constraints. The proposed approach is to automatically generate geometry that conforms to these constraints.

Some puzzles have a simple intrinsic solution, but a complex realization. The *devil's claw* is one example, which intrinsic solution is aligning the tunnels of the two pieces orthogonally so that one piece can slide through the other. The geometry of the puzzle is more complex than what the intrinsic solution would suggest, in order for the puzzle not to be trivial. Figure 4 shows a transformation from the puzzle piece to its IR. To allow the design of similar puzzles we output the generated geometry in a "CAD-like" parameterized form so that the designer can edit the geometry, under constraints to ensure that the puzzle is not made impossible.

1.4 Attacks and Open Problems

There are many interesting directions and sub-problems for the components of this project. Here we list the ones we find most important.

1.4.1 Intrinsic Representation. We have seen two mechanisms that can be used for puzzles, namely Gap and Tunnel. It is not clear which other mechanisms can be modeled in the same fashion, and a natural approach is to look at already designed wire puzzles and find patterns in how the pieces move. One can for instance imagine a mechanism that combines translation and rotation.

The entanglement of the pieces of the puzzle also has to be modeled. For instance, in *alpha* we can specify that the loop Loop_b of the blue piece goes through the hole Hole_r of the red piece as the relationship $\texttt{through}(\text{Loop}_b, \text{Hole}_r)$. Figure 5 shows a suggested in-memory representation of the components of the *alpha* IR. Using this graph we can compute that to align the Gaps we can move Gap_r to be in Hole_r, and since Gap_r is on Loop_r, they can be aligned. Further, we will likely need to explicitly impose an ordering of the adjacency's for each components so that we can ensure that the Rods does not obstruct the Hole.

1.4.2 Solving IR. One of the main strengths of the proposed model is that we have defined the intrinsic puzzle as a discrete system, so we can use graph search to find solutions. The initial configuration (Figure 5) can be a node in a *configuration graph*, and movements in the IR corresponds to following edges in the graph. A movement can induce constraints on the components of the puzzle, for instance that a piece fits through a hole. By ensuring that the constraint set is consistent, we can guarantee that the intrinsic puzzle is solvable.

It is not clear how to encode the puzzle moves in the IR, nor how to compute the constraints for a given move. One option is to restrict the possible motions to a predefined set which we know how to compute the constraints of. If the puzzle pieces are tangled in multiple positions, or if a piece is not completely rigid but consist of multiple sub-pieces like a linkage, computing moves becomes more difficult.

1.4.3 Generating Geometry. We can build a library of primitive shapes, like rods, helices, loops, and bends, all parameterized by lengths, radii, and so on, and use these primitives to generate the geometry for a IR. This offers flexibility of choice for a design, without being too restrictive. Having a discrete set of building blocks also makes the conversion convenient, and there are many natural mappings from the nodes to the geometric primitives: for instance, for a loop containing a Gap node we know that the loop cannot stay in one plane, and so a simple loop cannot be used, but a helix can.

The primitives have associated parameters, and constraints from the IR solution, so for each primitive we need to choose feasible parameters. The designer could easily add in their own constraints to this system, for instance to ensure that the curvature is always lower than a certain level. This would be very useful from a fabricational standpoint. For values with few or no constraints we can sample values from a predefined distribution that makes physical sense, tailored to each primitive.

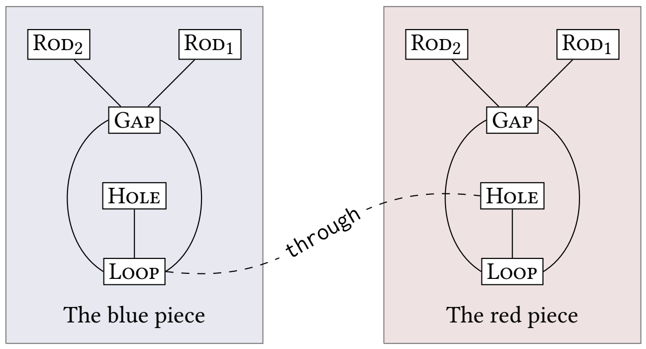


Fig. 5. A potential in-memory representation of the *alpha* puzzle. The Gap has four ordered connections, two to the two Rods and two to the endpoints of the Loop. The Loop has a reference to the Hole which it encloses. The blue loop goes through the red hole, represented by the dashed line. By symmetry, the red loop is also contained in the blue hole, but this is not shown.

1.4.4 Collision-aware Editing. When the user changes the geometry parameters we must make sure that the puzzle is still solvable. We can use that we already know the trajectory of the pieces for the solution, and constrain the parameter changes to be conforming to this solution. For two-piece puzzles we can fix one piece and sweep the other piece along the solution path to obtain a sweep volume that has to be free of geometry for the solution to be possible. We can then restrict the allowed changes of the parameterized shape to not intersect the swept volume. If the puzzle pieces are symmetric, we will need to update the swept volume as we change the parameters. For puzzle pieces consisting of multiple pieces we need another approach, because the order of the movements for the different pieces could be important.

REFERENCES

Rulin Chen, Ziqi Wang, Peng Song, and Bernd Bickel. 2022. Computational Design of High-level Interlocking Puzzles. *ACM Transactions on Graphics (SIGGRAPH 2022)* 41, 4 (2022), 150:1 – 150:15.

Gershon Elber and Myung-Soo Kim. 2022. Synthesis of 3D jigsaw puzzles over freeform 2-manifolds. *Computers & Graphics* 102 (2022), 339–348. https://doi.org/10.1016/j.cag.2021.10.014

Naoki Kita and Takafumi Saito. 2020. Computational design of generalized centrifugal puzzles. *Computers & Graphics* 90 (2020), 21–28. https://doi.org/10.1016/j.cag.2020.05.005

Kui-Yip Lo, Chi-Wing Fu, and Hongwei Li. 2009. 3D Polyomino Puzzle. *ACM Trans. Graph.* 28, 5 (dec 2009), 1–8. https://doi.org/10.1145/1618452.1618503

Peng Song, Chi-Wing Fu, and Daniel Cohen-Or. 2012. Recursive Interlocking Puzzles. *ACM Trans. Graph.* 31, 6, Article 128 (nov 2012), 10 pages. https://doi.org/10.1145/2366145.2366147

Xinya Zhang, Robert Belfer, Paul G Kry, and Etienne Vouga. 2020. C-Space Tunnel Discovery for Puzzle Path Planning. *ACM Transactions on Graphics (TOG)* 39, 4, Article 104 (2020), 14 pages.