C-Space Tunnel Discovery for Puzzle Path Planning

XINYA ZHANG, The University of Texas at Austin
ROBERT BELFER, McGill University
PAUL G. KRY, McGill University
ETIENNE VOUGA, The University of Texas at Austin

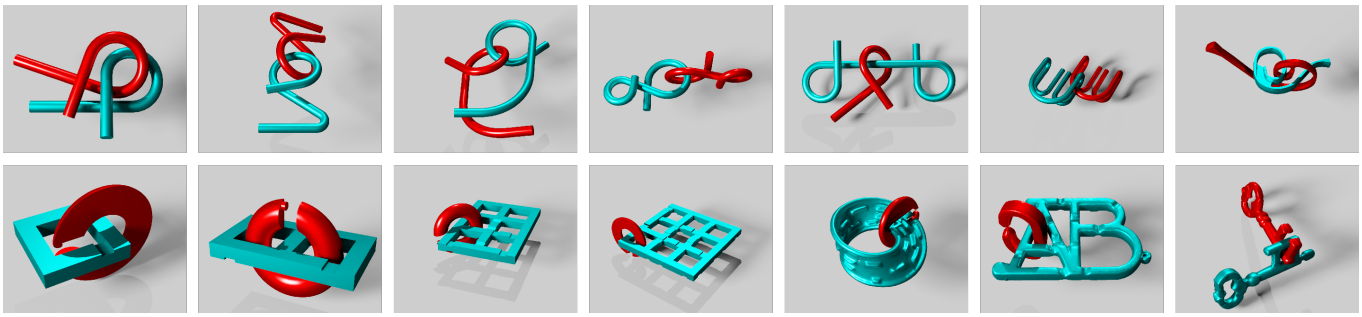

Fig. 1. Puzzles we solve using our approach, roughly ordered by difficulty from left to right. *Top row*: alpha, alpha-z, alpha-j, alpha-g, double-alpha, claw, and enigma. *Bottom row*: duet (with 4 different grid sizes), Mobius, ABC, and Key. All but the alpha variants are Hanayama puzzles.

Rigid body disentanglement puzzles are challenging for both humans and motion planning algorithms because their solutions involve tricky twisting and sliding moves that correspond to navigating through narrow tunnels in the puzzle's configuration space (C-space). We propose a tunnel-discovery and planning strategy for solving these puzzles. First, we locate important features on the pieces using geometric heuristics and machine learning, and then match pairs of these features to discover collision free states in the puzzle's C-space that lie within the narrow tunnels. Second, we propose a Rapidly-exploring Dense Tree (RDT) motion planner variant that builds tunnel escape roadmaps and then connects these roadmaps into a solution path connecting start and goal states. We evaluate our approach on a variety of challenging disentanglement puzzles and provide extensive baseline comparisons with other motion planning techniques.

CCS Concepts: • **Computing methodologies → Motion path planning**; *Neural networks*.

Additional Key Words and Phrases: sampling strategies

ACM Reference Format:
Xinya Zhang, Robert Belfer, Paul G. Kry, and Etienne Vouga. 2020. C-Space Tunnel Discovery for Puzzle Path Planning. *ACM Trans. Graph.* 39, 4, Article 104 (July 2020), 14 pages. https://doi.org/10.1145/3386569.3392468

Authors' addresses: Xinya Zhang, xinyazhang@utexas.edu, The University of Texas at Austin; Robert Belfer, belfer2470@gmail.com, McGill University; Paul G. Kry, kry@cs.mcgill.ca, McGill University; Etienne Vouga, evouga@cs.utexas.edu, The University of Texas at Austin.

0730-0301/2020/7-ART104 $15.00
https://doi.org/10.1145/3386569.3392468

1 INTRODUCTION

The *Piano Mover's Problem* asks whether one can move a piano between rooms through a sequence of rigid-body motions. This problem has inspired a great deal of work in computational geometry and robotics because it has a vast number of practical applications, from collision-free navigation in automated warehouses, to path planning in pharmaceutical drug design. Rigid disentanglement puzzles are an interesting variant of the problem because they are specifically designed to be difficult to take apart: they are notoriously difficult for both humans and computers to solve. One of the easiest puzzle in Figure 1, the *alpha puzzle* at top-left, requires a counter-intuitive twisting motion, and is frequently used as the most difficult benchmark when testing path-planning algorithms [Amato et al. 1998b; Kuffner 2004; Zhang et al. 2008; Zhang and Manocha 2008]. The reason for the difficulty can be understood by considering the geometry of the admissible configuration space.

Tunnels and Bubbles. Consider the 2D disentanglement puzzle shown in Figure 2, where a red brick can only escape the lower chamber through a thin gap. The configuration space (*C-space*) is $SE(2)$, that is, rigid translation and rotation of the red brick in the plane. The admissible region of C-space, $\mathcal{C}_{\text{free}}$, is all of the collision-free points in $SE(2)$, which we visualize as a volume in $\mathbb{R}^3$ at right in Figure 2. The red brick has a large range of motion within both lower and upper chambers. We call these large and open regions of $\mathcal{C}_{\text{free}}$ *bubbles*. Two very thin *tunnels* connect the bubbles, and correspond to translations of the brick as it slides through the gap at one of two possible vertical orientations. Any solution path for this puzzle must find and navigate through one of these tunnels, while not getting caught in dead-ends or complex geometric features of $\mathcal{C}_{\text{free}}$ (the creased boundary on the left side of the right tower in Figure 2 corresponds to configurations where the brick jams into the gap at non-vertical orientations). This tunnel-bubble geometry

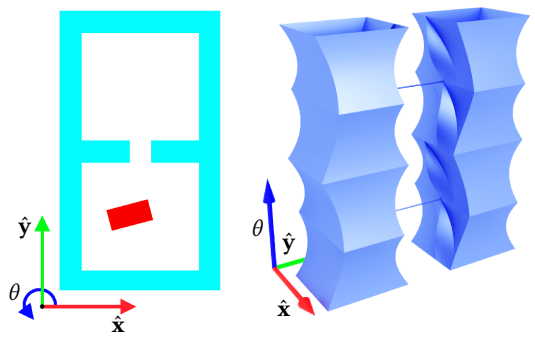

Fig. 2. A 2D disentanglement puzzle (*left*) consisting of a red brick that must pass through a narrow gap in the blue piece. The C-space is a subset of $SE(2)$, visualized (*right*) in 3D, with two translation axes, and a third periodic axis corresponding to rotation of the red brick. The collision-free C-space consists of two large *bubbles* (states where the red piece is trapped either in the upper or lower chamber), connected by two very thin *tunnels* (for each orientation that allows the brick to slide through the gap).

is characteristic of all disentanglement puzzles, such as those shown in Figure 1—except that C-space is now six-dimensional, the tunnels can be curved or corkscrewed, and often the bubbles and tunnels form a complex graph, so that disentangling the pieces requires passing through the correct sequence of tunnels and bubbles.

The Challenge. Any algorithm that plans a solution path through C-space must account for its bubble-tunnel geometry. Consequently algorithms must overcome several practical challenges:

- Curse of dimensionality precludes computing the exact geometry of C_{free} (or its five-dimensional boundary). Similarly, there is no obvious volumetric representation (based on voxels, or 2^6-trees, etc.) that can resolve the narrow tunnels accurately while scaling tractably to six dimensions.
- Sampling $SE(3)$ (either in top-down fashion, or via tree-based exploration) is extremely unlikely to find the narrow tunnels, since they have tiny volume and are local features (sampling the bubbles and sporadic points on their boundary does not give any hints as to where the tunnels might be). Strategies that treat C_{free} as a black box, and ignore information the 3D shapes of the pieces reveal about potential tunnel locations, are unlikely to succeed.
- Finding one tunnel is not enough: if the probability to find a path through a narrow tunnel is p, the probability to find a path through N tunnels may become as small as p^N if the algorithm design has not taken the bubble-and-tunnel structure of C_{free} into consideration.

Key Insight. As revealed by the challenges above, successfully solving a difficult path-planning problem, such as a disentanglement puzzle, centers on the ability to efficiently find and navigate through the problem's narrow tunnels [Hsu et al. 1999]. We believe that effective strategies for tunnel discovery must start from insights and priors about how the 3D geometry of the moving pieces induces narrow tunnels. Accordingly, in this paper we assume that most narrow tunnels correspond to alignment of geometric features on the two pieces: two narrow gaps sliding through each other, or a gap on one piece sliding over a thin part (*notch*) of the other piece.

We will show that this assumption is widely valid for disentanglement puzzles. Although we focused on puzzles as one particularly striking class of challenging path-planning problem, in other path-planning applications, from motion planning to industrial assembly to docking problems, narrow tunnels are often also the result of special alignment of geometric features on the parts.

In this paper, we design a planner around this connection between geometric features and the tunnel-bubble structure in C-space. Although our framework is more complex than black-box path-planning algorithms, it succeeds at finding narrow tunnels that the generic planners are very unlikely to stumble upon by chance, and so can solve puzzles with difficulty well beyond what was previously possible. Of course, in cases where our assumptions about the character of narrow tunnels are invalid (see Section 8.1 for one such example), our method loses its advantage, though it is easy to augment our overall approach to accommodate new problem-specific priors about narrow tunnels.

Contribution. We describe a path-planning pipeline that addresses the difficulties described above, and successfully solves disentanglement puzzles of unprecedented complexity, including all of the puzzles shown in Figure 1. Our main contributions are a strategy for detecting *key configurations* within each narrow tunnel (Section 5), based on detecting and pairing *important features* on the 3D geometry of the pieces (Section 4), and a parallel, distributed motion planner that accepts these key configurations and robustly and efficiently joins them together into a network of partial paths meeting within the bubbles (Section 6). We thoroughly evaluate our method on a wide variety of wire and Hanayama puzzles (Section 7).

2 RELATED WORK

The design and solution of geometric puzzles are problems of interest to researchers in the computer graphics and visual computing domains. Recent puzzle design algorithms have studied recursive interlocking puzzles [Song et al. 2012], burr puzzles [Xin et al. 2011], twisty puzzles like Rubik's cubes [Sun and Zheng 2015], and interlocking 3D jigsaw puzzles [Lo et al. 2009]. Likewise, a recent example of puzzle solving is the work by Huang et al. [2006], which reassembles fractured objects using geometric features, and is one example among many that use geometric analysis for restoration in cultural heritage applications [Pintus et al. 2016]. Computational geometry approaches can be used to solve jigsaws by analyzing their shape alone [Goldberg et al. 2004]. Recently, neural networks have shown promise in solving difficult puzzles, such as image jigsaw puzzles in 2D [Cho et al. 2010]. One particularly promising recent result [Ichter et al. 2018] trains a variational autoencoder conditioned on the obstacle geometry. This approach works well for path-planning around simple obstacles such as mazes with rectilinear walls and gaps, which can be easily represented using an occupancy grid, but it's not obvious how the method could be extended to disentanglement puzzles involving pairs of nontrivial 3D pieces. For general motion planning, deep reinforcement learning has shown success for tasks including robot grasping [Levine et al. 2018; Pinto and Gupta 2016], robot motion control [Finn and Levine

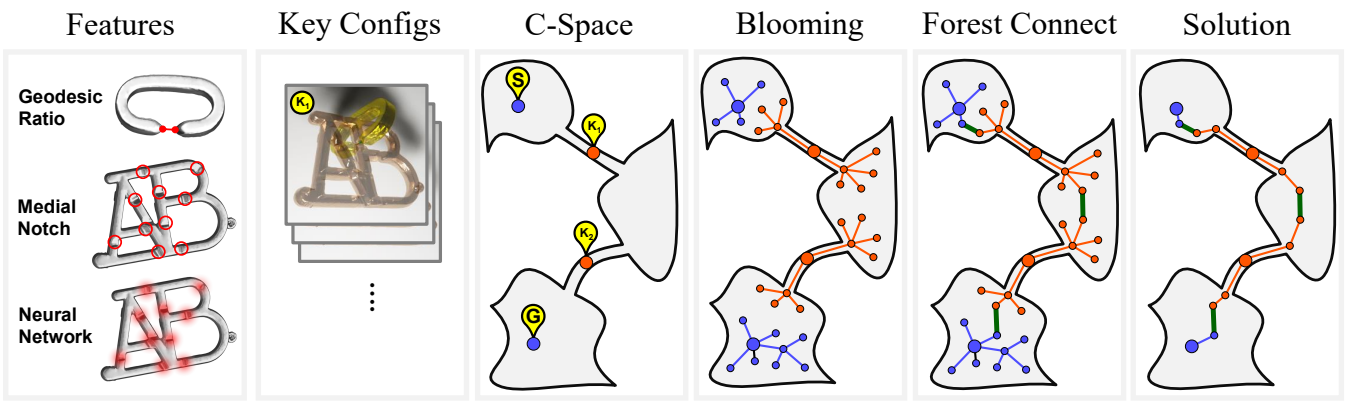

Fig. 3. An overview of our pipeline. Features are identified on the puzzle pieces with three techniques (shown here, for illustration purposes, on only one of the two pieces of the ABC puzzle each): Geodesic-Euclidean Ratio for gaps and notches (Section 4.1), Medial Axis Notches (Section 4.2), and neural-network-based features (Section 4.3). Features are paired, one from each piece, and valid collision-free *key configurations* are identified. The key configurations, along with start (S) and goal (G) states, seed the blooming step, shown here with an abstract 2D illustration. Blooming builds local road-maps within the *bubbles* around the start and end states, while also planning escape routes from key configurations, out of *tunnels*, and into adjacent bubbles. The forest connection step searches for collision-free trajectories between nodes of the different trees grown during the blooming step. Once the start and goal state trees belong to the same connected component, we construct the solution path.

2017; Zhang et al. 2015], and navigating video games [Jaderberg et al. 2016; Justesen et al. 2019].

Our work is based on the intuition that the pieces in a disentanglement puzzle have important geometric surface features that come into close proximity at key configurations, i.e., at crux points of the solution path. Certainly from visual inspection, human intuition can guess these features, and we rely on both geometric and neural network approaches in our solution. While our geometric features involve computing geodesic distance [Crane et al. 2013], and a smooth medial axis representation [Tagliasacchi et al. 2012], we note that there are many other shape features that might likewise be useful, as is the case for automatic geometry segmentation and labeling [Kalogerakis et al. 2010]. Neural networks now excel at solving classical geometry feature detection and processing tasks, like mesh classification [Lian et al. 2011] and segmentation [Wang et al. 2012]. Recent work likewise addresses a variety of geometry representations: multi-view images [Qi et al. 2016; Su et al. 2015], point clouds [Hanocka et al. 2019; Li et al. 2018; Qi et al. 2017], voxels with hierarchical structure [Wang et al. 2017, 2018], and meshes [Hanocka et al. 2019].

The core problem we solve in this work is the Piano-Mover's Problem, which has been known to be NP-complete since the 1970s [Reif 1979]. However, motivated by its importance in robot path-planning and manipulation tasks, there has been a tremendous amount of work on devising practical path-planning algorithms [LaValle 2006].

Sampling-based Path Planning Strategies. While exact approaches are possible in very simple cases, the majority of path planning research focuses on practical solutions that use sampling. Rather than exhaustively searching for solution paths, the focus is to connect sparse points sampled in C_{free}; these methods enjoy increased efficiency and practical effectiveness at the cost of only probabilistic completeness guarantees [Karaman et al. 2011; Kavraki et al. 1998; Ladd and Kavraki 2004].

The probabilistic roadmap method (*PRM*) [Kavraki et al. 1996] samples states in C-space and, if they are feasible, adds them to a roadmap graph. PRM attempts to connect together pairs of promising states, adding edges to the graph when a rigid motion through C_{free} exists between the states, and periodically checks for connectivity between the start and goal states. Various heuristics have been proposed both for sampling C-space and for choosing which pair of vertices to try to connect; at its simplest, PRM uniformly samples C-space and tries to connect each newly-sampled configuration to its nearest n vertices in the graph. More sophisticated heuristics are also possible, including those that combine several sampling strategies simultaneously [Hsu et al. 2005].

A second family of popular and effective path-planners are based on Rapidly-exploring Random/Dense Trees (*RRTs* and *RDTs*) [Lavalle 1998], with the main idea being to grow a connected tree incrementally from a starting configuration. At each iteration, a C-space state is sampled and added to the tree if it can be connected to its nearest vertex on the tree. The algorithm occasionally checks whether the goal state can be connected to the tree, and if so, terminates.

Many variants exist within the family of tree-based planners that vary in how they grow the tree and/or deal with contact constraints, including the popular RRT*, BIT* [Gammell et al. 2015], and EST [Hsu et al. 1999] methods. For example, *RRT-connect* [Kuffner and LaValle 2000] modifies RRT by growing two trees simultaneously, one rooted at the starting state and the other rooted at the goal. After adding a sampled configuration point to one tree, RRT-connect attempts to also connect the new point to the other tree. A solution path is found when an inter-tree edge is first detected. We call attention to one particular variant of RDT, described in LaValle [2006] chapter 14.4.3, which we will call *RDTo*. When a newly sampled state cannot be connected to its nearest neighbor

on the tree, RDTo shoots a ray from the nearest neighbor towards the sample, and adds the first intersection of the ray with C_{obs} (the inadmissible region; the complement of C_{free}) to the tree. RDTo trades a denser tree for increased ability to resolve complex obstacle geometry (such as that of disentanglement puzzles).

Methods for Handling Tunnels. Several planners have been developed that address the problem of navigating through narrow tunnels in C-space. While these methods can accelerate path-planning by improving the chance of successfully traversing a tunnel, we stress that often the hard part of the untangling problem is finding those tunnels in the first place. Biased exploration of C-space towards the medial axis of C_{free} is a strategy that has been applied to both RRTs [Denny et al. 2014; Rodriguez et al. 2006] and PRMs [Wilmarth et al. 1999]. Although it is intractable to compute the medial axis exactly, these methods push a given configuration in C_{free} to the medial axis by finding the closest configuration where the pair of closest points on the two pieces suddenly jumps. This search is more challenging for states in C_{obs}, and moreover scales poorly to six dimensions; these methods therefore choose to find the closest collision free state under translation only. For the example in Figure 2, a tunnel state would be found only if the orientation of the initial sample point happens to line up with the tunnel. In contrast, we find key configuration states within the tunnel directly.

Another family of path planners focus on generating samples on the boundary between C_{free} and C_{obs}. RDT+ [Vahrenkamp et al. 2011] proposes a parameter-free RRT-connect-like algorithm for dynamically adjusting the search distance to better sample within narrow tunnels. Obstacle-based PRM [Amato et al. 1998b] samples configurations by selecting pairs of points, one on each object, translating the objects so that the chosen points coincide in $\mathbb{R}^3$, and applying random translations and rotations together with binary search to try to find collision-free configurations near C_{obs}; our key configuration generation algorithm (Section 5) adopts a similar approach. Along similar lines, Rodriguez et al. [2006] propose multiple growth strategies for RRT, including choosing configurations corresponding to alignment and sliding of nearby triangles on the two moving objects. Amato et al. [1998b] also describe a three-stage connection strategy which uses powerful local planners to more reliably connect together components of the probabilistic roadmap. Pan and Manocha [2016] present a local planner with accelerated probabilistic collision detection based on hashing and reusing prior collision queries; to remain robust when navigating narrow tunnels, their method randomly augments probabilistic queries with occasional exact collision queries.

Finally, D-plan [Zhang et al. 2008] is based on RRT, but introduces retraction-based sampling [Zhang and Manocha 2008] and constrained interpolation [Zhang and Manocha 2010] for connecting configurations. Retraction finds samples in C_{free} that are close to C_{obs} using a combination of projections and local search. To check if two configurations connect, D-plan computes the closest points between the two objects at each configuration, and computes an interpolating trajectory that does not decrease the distance between the closest points. These trajectories are more likely to be collision-free, though there is still the challenge of successfully sampling states within the tunnel. Nevertheless, the key high-level idea of D-plan is to exploit available information about the 3D geometry of the objects during path-planning. We take this approach as well.

3 PIPELINE OVERVIEW

The greatest challenge for solving disentanglement puzzles is locating the narrow tunnels. If we knew a few *key configurations* in C_{free} located inside each narrow tunnel, we could grow roadmaps rooted at each of these key configurations. Eventually each roadmap would expand into the large bubbles adjacent to the tunnels, where they become easy to connect to each other.

The pipeline of our solution to disentanglement puzzles is shown in Figure 3. There are three main parts to our solution, corresponding to the strategy above: *feature detection* takes the geometry of the puzzle pieces and finds surface points that are likely involved in the solution; *feature alignment* takes pairs of features and produces a set of key configurations likely to be in narrow tunnels; and *planning* takes these key configurations as seeds, blooms them into roadmaps, and connects them together to produce a solution path as output. We briefly outline each part in more detail, before going into depth in the following sections.

Feature Detection. The goal of feature detection and alignment is to produce key configurations in C-space; since directly exploring C-space in the hopes of finding narrow tunnels is intractable, we instead exploit the observation in the introduction that most tunnels correspond to alignment of geometric features on the two pieces.

We locate feature points on the 3D geometry of the individual puzzle pieces, using several different strategies. Pairs of points on the surface that form a local minimum of the Euclidean-geodesic radio are a good indication of gaps, and we use a combination of gradient descent and random search to try find them all (Section 4.1). While this ratio can identify notches, it is not reliable for finding these largely local features. Instead, we can rely on finding notches by searching a smoothed medial axis representation for locations where the geometry is locally narrow (Section 4.2). Finally, we exploit neural networks to identify gaps, notches, and other features by training the network on easy puzzles with known solutions, and then using the network to identify the same types of features on novel puzzle pieces (Section 4.3).

Feature Alignment. With important features identified on the two pieces of a puzzle, we test pairs of features (one from each piece) by searching for collision free configurations where the pieces align at their features [Amato et al. 1998b]. This alignment involves testing a subset of rotations depending on the features involved. Because some puzzles may produce many configurations that are all in proximity, we use clustering to avoid placing too many key configurations into any given tunnel.

Planning. Path planning begins by appending the start and goal states to the key configurations and setting these C-space states as *roots* in separate RDTo planners. We call this process *blooming*, and expand the blooming trees until they reach a user-defined size. The benefit of growing separate trees include improved exploration by preventing the samples of one tree from interfering with another, and easy parallelization.

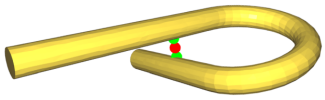

Fig. 4. Euclidean-geodesic ratio is minimized for the indicated pair of green points. The red dot marks the midpoint; it will be paired against another midpoint on the other piece to estimate a key configuration.

Given the forest produced by the blooming stage, we use k-nearest-neighbor search to connect each blooming tree with the others. We then use a breadth-first search to identify a path from the start to goal state. We called this process *forest connection*. Although our goal is to find any solution path, not necessarily the shortest or smoothest, we post-process our solution paths using OMPL's smoothing algorithms [Mainprice et al. 2011; Şucan et al. 2012] to improve them.

4 FEATURE DETECTION

Our approach to finding key configurations within C-space tunnels starts with identifying important feature points on the surface of each puzzle piece. The intuition is that narrow tunnels in puzzle solutions often align gaps from its two pieces, or align a gap from one piece against a notch from the other piece. This motivates our search for narrow tunnel configurations by locating gaps and notches on puzzle pieces. Likewise, given that human intuition can identify good features, we also train neural networks to identify important geometric features for solving puzzles.

4.1 Euclidean-Geodesic Ratio (EGR)

A pair of points with a low Euclidean-to-geodesic ratio are close together in Euclidean space, but distant with respect to the shortest path on the surface. A low ratio is typically indicative of a narrow gap or notch in the object, as shown in Figures 3 (top left) and 4.

The situation is more complex when the object exhibits symmetry. For intuition, consider some examples: for a sphere, every pair of antipodal points share the same optimal ratio $2/\pi$. Finding such antipodal points is useful, for alignment with a cylindrical gap on the other piece. For a general surface of revolution, there will similarly be circles of antipodal points with locally minimal EGR, but in this case we are most interested in pairs of points that also minimize the Euclidean distance: pairs of points that have a smaller Euclidean distance are part of the thinnest, tightest area of the gap (or notch), and are most likely to be part of a key configuration in a C-space tunnel. Thus, our search is for local minima of the EGR plus a small Euclidean bias,

$$r(\mathbf{u}, \mathbf{v}) = \frac{e(\mathbf{u}, \mathbf{v})}{g(\mathbf{u}, \mathbf{v})} + \alpha \cdot e(\mathbf{u}, \mathbf{v}), \tag{1}$$

where $\mathbf{u}$ and $\mathbf{v}$ represent points (not necessarily vertices) on the surface of the object, with $e(\cdot)$ and $g(\cdot)$ denoting Euclidean and geodesic distances respectively. This bias also helps us eliminate point-pairs on diametrically opposite sharp edges or corners, as they do not form useful features with respect to identifying gaps.

With many geodesic distances to compute, we use the heat kernel method [Crane et al. 2013], which provides smooth approximations of the distance, and a fast method for querying new vertices (i.e., back-solve with Cholesky factors). It is impractical to use brute force to find all local minima, even if we restrict the search to only pairs of surface vertices. Instead, we use a Monte Carlo algorithm to locate minima.

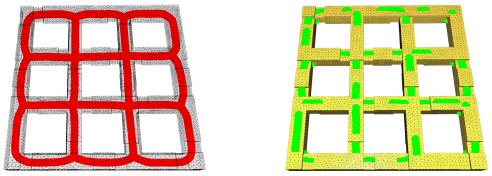

Fig. 5. Smoothed medial skeleton (*left*) for the Duet grid in red, and the location of identified notches in green (*right*).

We initially select a random vertex $\mathbf{u}$. We then find all vertices that locally minimize r when paired with $\mathbf{u}$, and select one vertex $\mathbf{v}$ from that list at random. Although selecting the vertex that globally minimizes $r(\mathbf{u}, \cdot)$ may seem like a good idea, doing so prevent us from finding other relevant important pairs: the global minimum is not always relevant for path planning. For example, the alpha-g puzzle has two gaps, and only the larger gap contributes to the solution as the smaller gap is too narrow. Therefore, we consider any pair that locally minimizes r.

We then iteratively improve our selection via a local search by moving either $\mathbf{u}$ or $\mathbf{v}$ to an adjacent vertex if it will lower the r value. Once none of the neighboring vertices can lower the value, we proceed to a continuous optimization constrained within the faces of the mesh using an active set conjugate gradient algorithm. Because the problems are very small, few iterations are needed to produce reasonable solutions.

We use $\alpha = 0.05$ to bias the ratio function during the initial search as well as during continuous optimization, and then polish the solution point pair further by repeating our continuous optimization with $\alpha = 0$ to remove the Euclidean bias, as its effect can be large at narrow gaps where the ratio is small. Finally, we reject the detected feature point pair if the solution is degenerate: if the pair involves points on adjacent faces, or if the line segment strictly between them intersects the object surface, as can occur for larger gaps.

We repeat the entire procedure above n_{gap} times with different random initializations $\mathbf{u}$ to find up to n_{gap} pairs of feature points.

4.2 Medial Axis Notch (MAN) Detection

The geometry of some objects can be thin in local regions. As mentioned above, we call this type of feature a *notch*. Although the Euclidean-geodesic ratio can sometimes find notches, it does not do so reliably, especially on pieces with many flat, parallel faces, such as the Duet puzzle piece in Figure 5 (a notch will not be found unless the initial randomly-sampled point $\mathbf{u}$ happens to be on a notch).

To better detect notches, we use a smoothed medial skeleton of the object, and search for medial points where the medial radius is small. A small radius indicates that a notch is present, or that the object is particularly thin in that area. We obtain the medial skeleton of each puzzle piece by first calculating its medial axis (a 2D surface), and then performing mean curvature flow on it until

only a curve remains [Gao et al. 2019; Tagliasacchi et al. 2012]. The smoothing and dimension-reduction simplifies our search, with the drawback that points on the curve only approximate the centerline of the object.

For each point on the skeleton, we approximate the medial radius by finding the closest point on the surface. This closest point becomes one endpoint of the notch; we then find the other endpoint by looking for the closest surface point in the opposite direction from the skeleton. As both points will not typically be equidistant from the skeleton, we shift the skeleton to the midpoint. We repeat this process one more time, i.e., identifying closest point, opposite point, and recomputing the midpoint, to gently refine skeleton positions and radii, but do not iterate further to avoid having skeleton points migrate far from their original position. The result is a set of refined skeleton points, radii, and surface point-pairs that may correspond to notches.

With the approximated medial radius from samples on the medial skeleton, we then locate the notches as the local minima of the medial radius. However, this simple approach is susceptible to noise. To make notch detection more robust, we first group nearby vertices into *vertex groups*, by repeatedly merging neighbors to groups seeded with medial skeleton vertices that have a locally minimal radius. Merging repeats as long as neighbors do not exceed $1 + \epsilon_0$ of the medial radius of the initial vertex. For each group, we denote the medial radius of the initial vertex as *group radius*.

After clustering vertices into groups, we check each group against all of its neighbouring groups. If the group radius is smaller than $(1 + \epsilon_1)^{-1}$ of all its neighbouring groups, this group is marked as a local minimal group, and the medial radius from all vertices in this local minimal group are marked as features of this geometry.

We use $\epsilon_0 = 0.1$ and $\epsilon_1 = 0.05$ throughout this paper. Empirically this choice effectively eliminates false positives while keeping the real notches.

4.3 Neural Network Feature Detection

In the previous sections, we described geometric strategies for identifying gaps and notches in the piece geometry. Given recent successes by machine learning algorithms in solving similar classification tasks in computer vision, a natural complementary strategy is to train a neural network to identify geometric features on rasterizations of the puzzle pieces from different camera angles. This approach can succeed when a narrow tunnel does not exactly correspond to gaps or notches detected in Sections 4.1 and 4.2, either due to variations in puzzle design or noise in the piece geometry.

Overview. Although it is surely possible to train a neural network to label gaps, notches, and other geometric features on a puzzle piece, given a sufficiently rich corpus of training data, we propose a different approach that does not require such data. Following classic path-planning work such as OBPRM [Amato et al. 1998b], for the neural network feature detector we will focus on finding contact configurations within narrow tunnels—key configurations where a point on each of the two pieces touch. We train three neural networks for three different types of features (gaps, notches, and protruding teeth), and each predicts a feature density function ψ on the surface of each puzzle piece, with $\psi(\mathbf{u})$ proportional to the probability that $\mathbf{u}$ participates as a contact point in a narrow tunnel contact configuration.

We generate training data by choosing easy puzzles that can be solved using a conventional planner, yet contain geometric features analogous to those we expect in the difficult puzzles: we used the alpha 1.15 puzzle, which is an easier variant constructed from the alpha puzzle [Amato et al. 1998a] with a wider gap, and a small piece of the Duet maze cut out from the main puzzle (see Figure 1, bottom-left), whose maze piece is representative of notch-type features, and whose ring piece is representative of protruding teeth.

4.3.1 Generating Feature Density Training Data. Given a training puzzle, we generate a ground-truth feature density function ψ based on the solution paths found by traditional path-planners. We assume that the training puzzle is simple enough that these planners always succeed in finding a solution path, and that the puzzle contains only one narrow tunnel, which any solution path must travel through. We represent the output ψ for each piece as a scalar function over a 4096×4096-pixel texture atlas parameterizing the piece surface.

To compute ψ, we solve the puzzle 100 times using the RDTo planner to collect 100 solution trajectories. For each trajectory, we uniformly sample 1024 configurations q_i along the path.

We then estimate whether each sampled configuration is inside a narrow tunnel, or inside a bubble. From q_i we shoot 1024 rays in random directions into C-space, and compute the average distance that the rays travel before intersecting C_{obs}. We declare the 3 sampled configurations with lowest average distance as lying inside a narrow tunnel. Note here we assume there is only one narrow tunnel in the training puzzle, and the goal of using multiple configurations is to capture more samples deep in the narrow tunnel (rather than just at the entrance).

If we believe that trajectory configuration q_i lies in a narrow tunnel, we shoot 1024 rays from q_i, and compute their intersection points q_{ij} with C_{obs}, as well as the pair of points $\mathbf{u}_{ij}$ and $\mathbf{v}_{ij}$ on each piece that are touching at the configuration q_{ij}. We increment $\psi(\mathbf{u}_{ij})$ by $\|q_{ij} - q_i\|^{-1}$, where the weighting accounts for the fact that configurations closer to q_i are more likely to be contact configurations in the narrow tunnel.

4.3.2 Network Design and Training. Each of the three networks is a single hourglass [Newell et al. 2016]. The input to the networks consists of a five-channel 256×256 image [Eigen and Fergus 2015; Qi et al. 2016; Socher et al. 2012; Su et al. 2015; Yang et al. 2019]: one for the luminance, one for depth, and three for the normal. For both training and testing we render the pieces using a Lambertian material with a single point light placed in a spot fixed with respect to the camera; we also place the piece with centroid centered in the camera's view at a fixed view distance. The output is a single-channel 256×256 image, the predicted ψ values for each pixel in the input image. Our network architecture consists of a single hourglass, without the downsampling from 256 to 64 pixels suggested by Newell et al. [2016]. To compensate for the increased input resolution, we use 6 residual modules instead of 4.

To train the network, we compute ψ for each of our two training puzzles, as described in section 4.3.1, and generate around $300,000$ image pairs for each puzzle piece, each with a randomly-chosen

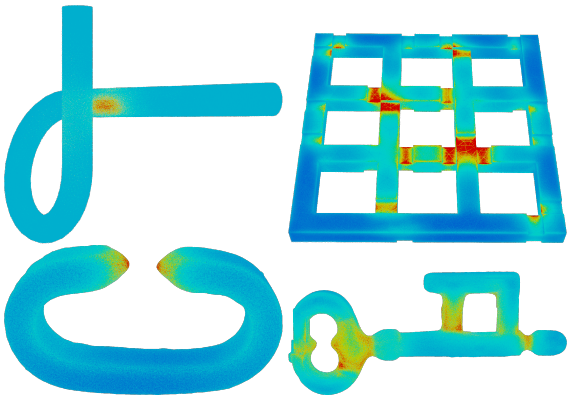

Fig. 6. The feature density function ψ predicted by our network, for four different puzzle pieces. Warmer colors indicate higher values of ψ, corresponding to points the network believes have higher probability of being feature points.

translation and rotation of the puzzle. We use Adam optimization [Kingma and Ba 2015] with learning rate $2.5 \cdot 10^{-4}$ to train our network.

4.3.3 Data Augmentation. Data augmentation is a classic approach for reducing overfitting on image data [Krizhevsky et al. 2012], and we find it is crucial in our case because of the low number of training geometries (two puzzles with two pieces each). The following augmentation filters are applied to the input images during training to reduce overfitting:

(1) we apply random noise with Poisson distribution $\lambda = 0.05$ to the luminance channel of whole image input image;
(2) we randomly pick a 64×64 tile which contains pixels with non-zero ψ value, and clear everything in the input image outside that tile; this filter is designed to make our hourglass focus more on local features.
(3) we randomly pick a 64×64 tile with ψ value identically zero, and delete that tile from the input image. Intuitively this filter removes random blocks from the input image, synthetically increasing puzzle variety.

For each rendered image, we randomly choose one of the filters to apply, with probabilities 0.1, 0.7 and 0.2 respectively.

4.3.4 Feature Prediction. We use the trained hourglass networks to predict ψ for new puzzle geometries. To get good coverage of the puzzle piece, we select 4096 random camera orientations, and rasterize the piece to a five-channel image for each chosen orientation in a manner identical to the setup during training. The predicted ψ for each image is essentially a sixth color channel on that image; we map the predicted ψ values to a texture atlas of the piece and aggregate ψ over all 4096 views for all three networks (see examples in Figure 6). Finally, we sample the texture atlas n_{NN} times, with probability distribution proportional to ψ, to select feature points for key configuration generation.

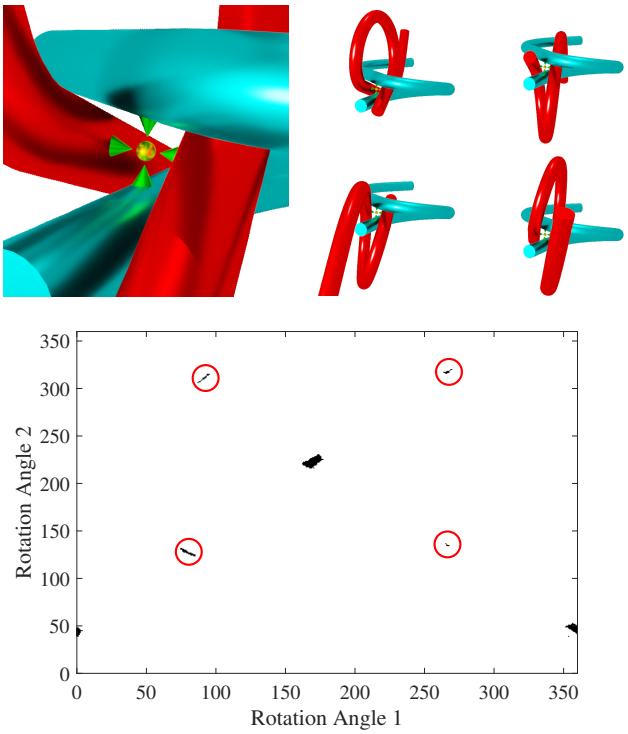

Fig. 7. The four key configuration (*top-right*) formed by aligning a pair of gaps in the alpha puzzle (*top-left*); here the cyan piece is fixed and the orientation of the red piece was found by sweeping over its possible orientations, with a search resolution of 1 degree. Visualization of the valid regions (dark) of this angle search space (*bottom*). Red circles identify the four key configurations. The other dark regions are alignments of the two pieces that are collision-free, but not in a narrow tunnel.

5 KEY CONFIGURATION GENERATION

The previous section describes different strategies for locating key points on the 3D geometry of the puzzle pieces. Next we must turn these key points into key *configurations* that lie in the tunnels of the six-dimensional puzzle C-space. A natural idea is to choose a feature point on each of the two pieces, and translate the pieces so that these points overlap. We then rotate the second piece while keeping the two points in contact, searching the space of orientations of the second piece for configurations that lie in C_{free}. This pairing and search process is slightly different for feature points that come from the EGR and MAN strategies than for those that come from the neural network, since in each case we have access to different orientation cues to help position the two pieces.

5.1 Aligning EGR and MAN Features

Recall that the features detected using the EGR strategy are pairs of points delineating a notch or gap, while MAN detects only notches. We classify each pair of EGR feature points as either a gap or notch—depending on whether the surface normal at the two feature points are oriented towards or away from each other—and form key configurations by pairing gaps with gaps, and gaps with notches.

Gap-gap Key Configurations. We form key configurations from pairs of gaps by choosing a matching pair of EGR feature points $(\mathbf{u}_1, \mathbf{v}_1)$ and $(\mathbf{u}_2, \mathbf{v}_2)$ on each piece, computing the midpoints (center of the notch or gap) $\mathbf{m}_1, \mathbf{m}_2$ of each pair, and positioning the two pieces so that $\mathbf{m}_1$ and $\mathbf{m}_2$ coincide, and the gap vectors $\mathbf{v}_2 - \mathbf{u}_2$ and $\mathbf{v}_1 - \mathbf{u}_1$ are orthogonal (since orthogonal alignment corresponds to orientations of maximum clearance between the two pieces, and so the most likely key configurations).

There are two rotational degrees of freedom for the second piece which keep the gap vectors orthogonal: rotation of the second piece about either of the two gap vectors. We sweep this two-dimensional space of rotations, in increments of one degree, looking for configurations where the two pieces do not intersect. Here and throughout the paper, we perform collision checking in C-space using discrete collision detection with the FCL library (the default implemented in OMPL). The alpha puzzle seen in Figure 7 shows a key configuration formed by pairing gaps in this way. The plot at the bottom of Figure 7 shows in dark areas the configurations in C_{free} found during the angle sweep.

While the alpha puzzle has one unique Euclidean-geodesic ratio minimum, after performing the angle sweep we discover six connected regions of potential key configurations in C_{free}. The two large regions contain states that do not contribute to a solution path (dead end tunnels) and the four small regions contain key configurations that are each on a distinct solution path. We note that the configuration space of the alpha puzzle has three bubbles: two where the pieces are tangled, and one where the pieces are completely separated. Due to the symmetry of the alpha puzzle pieces, there are two possible ways to untangle the pieces. Hence there are four distinct tunnels total, two from each tangled bubble to the solved state bubble. The small size of these regions demonstrates why the alpha puzzle is a tough challenge for planners to solve. They are unlikely to randomly sample a state in the tunnel region.

Since seeding near-identical configurations into our planner is unnecessary and can reduce performance, we select the key configuration with the highest clearance (separation distance of the two pieces) per connected valid region in the 2D angle grid.

Gap-notch Key Configurations. The process for pairing gap features above changes slightly when pairing a gap with a notch. First we trivially reject a gap-notch pairing if the gap is smaller than the notch thickness. We then align the gap and notch vectors so that they are parallel and search the one-dimensional space of rotations for collision free configurations. Since the gap and notch vectors can be parallel with the vectors pointing in the same or opposite directions, we perform the search twice.

5.2 Aligning Neural Network Features

Each feature from the neural network detector consists of a surface point $\mathbf{p}$ and its corresponding surface normal $\mathbf{n}$. We form key configurations from feature pairs $(\mathbf{p}_1, \mathbf{n}_1)$ and $(\mathbf{p}_2, \mathbf{n}_2)$ on the two pieces by rotating the geometry so that $\mathbf{n}_1$ and $\mathbf{n}_2$ are anti-parallel, and then translating the pieces so that they coincide with a slight normal offset: $\mathbf{p}_1 = \mathbf{p}_2 + \epsilon\mathbf{n}_2$. As in the case of gap-notch features, this setup leaves a one-dimensional family of rotations that we once again sweep for configurations in C_{free}. The offset ϵ (we use $\epsilon = 10^{-6}$) helps avoid collision false positives during the sweep.

Since neural network feature quality has high variance, we adopt an oversampling strategy when generating key configurations: to provide the planner with N key configurations, we sample $10N$ key configurations and only keep the N with lowest average distance to C_{obs} (computed as in Section 4.3.1). In this way key configurations are more likely to lie in narrow tunnels than in bubbles.

5.3 Key Configuration Clustering

The key configurations generated by the algorithms above tend to contain many duplicates that lie within the same narrow tunnel. To remove these duplicates, we connect the key configurations into a graph G, with edges between pairs of configurations that can be connected by a straight line in C_{free}. We then keep only one key configuration per connected component.

Finally, we reject any key configuration corresponding to a disentangled state (for which the two bounding boxes of the two pieces do not overlap).

6 PATH PLANNING WITH BLOOMING

The strategies described in Section 5 predict key configurations that lie in narrow tunnels. To turn these key configurations into a solution path, we must find trajectories that connect them together.

Perhaps the most straightforward solution would be to add all the key configurations as milestones of a PRM planner, and to keep sampling additional points in C-space until the planner is able to connect the points together into a complete solution path. Unfortunately, as we show in Section 7.3, and as also observed by Shi et al. [2014], it is difficult for PRM to find paths connecting narrow tunnel configurations to each other, without requiring an extremely large number of samples. LaValle's classic textbook [2006] suggests using multiple RDT trees rooted at different points in C-space, as an extension of the usual single- or dual-tree RDT algorithm; methods [Strandberg 2004] such as SparkPRM [Shi et al. 2014] further explore this idea by maintaining a global PRM data structure, and switching to the RRT algorithm locally if a narrow tunnel configuration is sampled. We propose a two-stage algorithm to solve disentanglement puzzles inspired by this approach.

First, we expand a forest of exploring trees rooted at each key configuration, using the RDTo planner (which we found to be far more effective than RRT; see also Section 7.3). We call this stage *blooming*, and the resulting trees *blooming trees*. Ideally, starting from the root inside a narrow tunnel, these trees grow into the large bubbles at both ends of the tunnel, where multiple blooming trees all occupy the same space. In the second *forest connection* stage, we attempt to connect the forest of blooming trees into a single graph.

Blooming. For each key configuration (as well as the start and goal state), we use RDTo to compute a local roadmap rooted at that configuration. We use three termination criteria: we stop growing the tree if it ever connects to another key configuration, exceeds a maximum user-defined size K, or if the running time reaches a user-defined limit (we chose 15 minutes). The latter is needed since the configuration space of a complex disentanglement puzzle often includes tiny disconnected components of C_{free} not connected to

the main admissible region of C-space. Inside these pockets, it may be impossible for a tree to grow to the desired size of K vertices.

Forest Connection. During the forest connection step, we have N blooming trees we want to connect together. For each tree T_i, we create a kd-tree nearest neighbor data structure for all vertices in the remaining $N-1$ trees. We then iterate through the vertices $v \in T_i$, compute its C nearest neighbors in the remaining $N-1$ trees, and add an inter-tree edge between v and its nearest neighbor if the edge lies entirely within C_{free}. We used $C = 8$.

After merging a tree T_j into T_i in this way, we delete all vertices of T_j from the nearest neighbor data structure. In this way, if T_j contains many vertices very close to those of T_i, it does not prevent our algorithm from connecting T_i to other blooming trees whose vertices are slightly more distant.

Finally, after completing the forest connection process, we check for a path from the start to the goal state using standard breadth-first search. Our method has succeeded if such a path is found.

6.1 Paralellization

When the number of key configurations is large,the wall-clock time required by the planning process can be reduced substantially by exploiting opportunities for parallelization. The blooming process is already embarrassingly parallel; the forest connection and extraction of a final solution path can also be parallelized with a few minor modifications.

During forest connection, we process each of the trees T_i independently in parallel, and accumulate the inter-tree edges that should be added that involve T_i (there are at most $N-1$ of these). After every tree has been processed, we construct an N-vertex tree-level graph from the inter-tree edges, and compute a path from the start to the goal state using BFS at the tree level. We now know which blooming trees the solution path traverses, and the start and end vertices s_i and g_i within tree T_i of each segment of the solution path. We find the path connecting s_i to g_i within T_i using BFS in parallel.

6.2 Sample Interference

A notable aspect of this planner's design is that the blooming trees are merged in the last step of planning, instead of earlier. A majority of the key configurations we find are false positives not located in tunnels, so it may seem that merging trees early would improve the efficiency of our planner, by allowing it to spend more time growing trees that are still exploring the tunnels. However, in our experiments we found that (in addition to posing parallelization challenges) eagerly merging the trees often *reduces* the performance of the planner, due to a phenomenon we call sample interference.

Sample interference occurs when a tree grows to a large size and covers a significant portion of configuration space. It is harder for a large tree to navigate a tunnel than a small one, as a consequence of the nearest neighbor heuristic used by RRTs: when a tree covers a large region in C-space, it is less likely that the nearest neighbor of a sampled state is a state in a tunnel.

In our early experiments with eager planners, we found that key configurations in tunnels connect quickly to one of the two adjacent bubbles, but then due to sample interference rarely connect to the other bubble. Sample interference has interesting implications for RRTs, and suggests that, to quickly find a solution, it is more efficient to run many small trials and hope to get lucky than to search for a tunnel in a single, extended trial.

7 EVALUATION

We evaluate our method on 17 disentanglement puzzles of varying type and difficulty:

- in addition to the traditional alpha and alpha-1.1 puzzles, we calipered and 3D-modeled six other wire-type puzzles: the Hanayama claw, and six alpha-puzzle variants from Teenitor's "10-piece Metal Iron Brain Teaser IQ Test Assembly & Disentanglement Puzzle Toy" box set;
- we also calipered and modeled Hanayama's Duet puzzle. The puzzle is actually two in one: two identical circular pieces must escape the same maze, in two different ways. We include both sub-puzzles (each involving only one circular piece and the maze) as duet-g9 and duet-g9a. We also cut the Duet maze into smaller pieces to yield easier puzzles (duet-g1 through duet-g4);
- Mobius, ABC, and key are Hanayama puzzles that we 3D-scanned. The Hanayama Enigma includes three pieces; our enigma is a sub-puzzle involving only two of those pieces, modeled by a 3D artist.

Our collection of puzzles was obtained by various methods and had different representations. We chose to represent them using meshes, since the collision detection algorithms we use require them as input. We illustrate many of these puzzles in Figure 1, and provide their meshes as supplementary material.

Since both the feature alignment and planning phases of our pipeline depend on random sampling, success or failure to find a solution path for a given puzzle can vary across executions. For each puzzle, we therefore run our pipeline with identical parameters ten times, and use the ratio of successes as an approximation of our method's probability of solving that puzzle.

We run our pipeline on an HTCondor cluster [Litzkow et al. 1987] of heterogeneous machines (ranging from Opteron 2218s to Xeon E5-2670v2), to which we distribute the work of running the blooming and forest-connection steps of our planner.

We show in Table 2 the overall chance of our method to solve each puzzle, using all key configuration prediction strategies (the *Combined* column). Videos of the solution paths for many of the puzzles are provided in the supplemental materials. We will publicly release the source code of our method on GitHub.

We selectively present a few disentanglement plans found during the experiments in Figure 8 and Figure 9.

7.1 Choice of Parameters

Most of our method's tunable parameters are held fixed throughout all experiments we report, and their values have been provided in the text. For the remaining parameters, we use three sets of values, each for a different puzzle difficulty. These parameters are listed in Table 1, and are intended to provide the planner with more guidance for harder puzzles, at the cost of increased computation. The last listed parameter specifies whether we do key configuration clustering to eliminate nearby configurations in narrow tunnels. By disabling

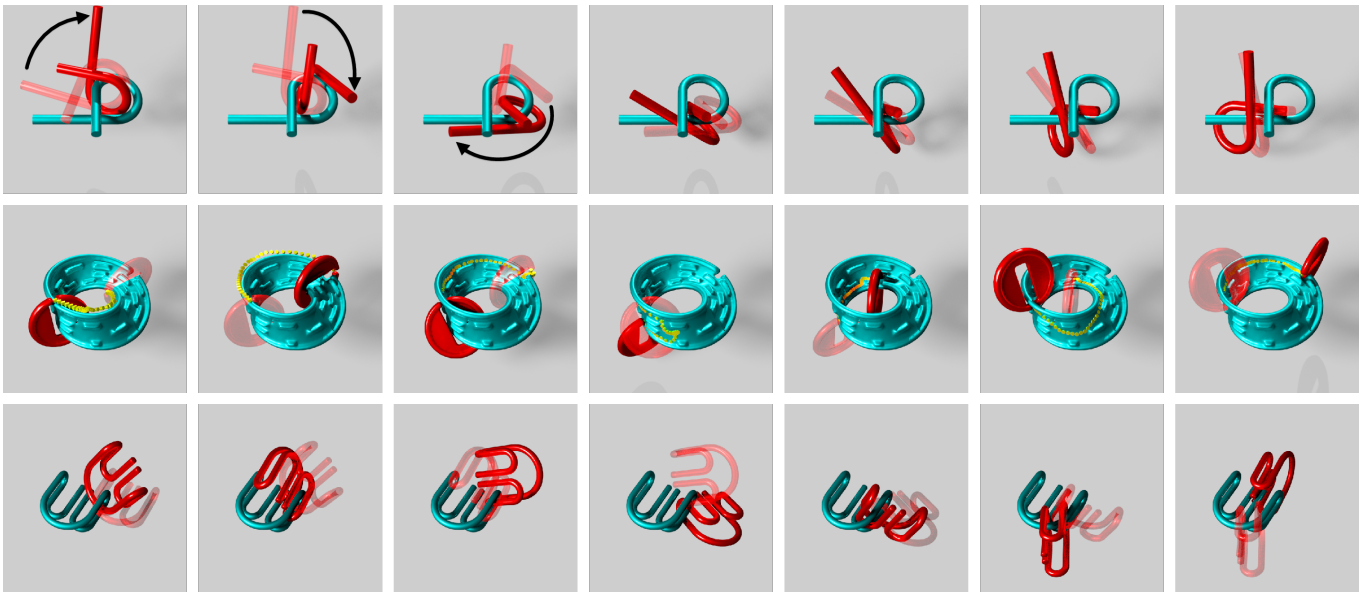

Fig. 8. Disentanglement plan of alpha-1.0, Mobius and claw found by our pipeline

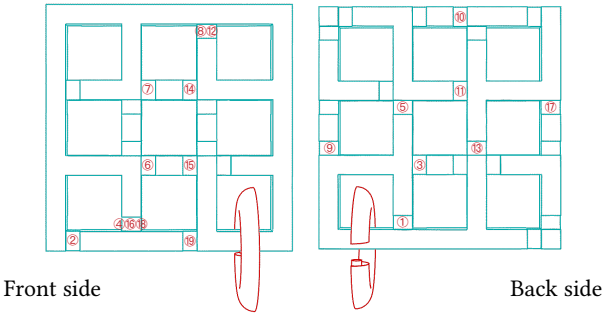

Fig. 9. Disentanglement plan of duet-g9. The number indicates the step at which the gap in the ring should pass through a notch in the grid.

Table 1. Parameters values for different difficulty levels.

Name	Section	Difficulty		
		Easy	Hard	Extreme
EGR detection attempts n_{gap}	4.1	12	20	32
NN feature samples n_{NN}	4.3.4	32	512	1024
Size of blooming tree K	6	3072	4096	6144
Key config. clustering	5.3	Yes	Yes	No

this step we allow multiple key configurations in the same narrow tunnel, simplifying the planning task.

7.2 Effectiveness of Individual Key Point Strategies

We test the effectiveness of each of our three key point discovery strategies (Euclidean-Geodesic Ratio, Medial Axis Notch Detection, and Neural Network Feature Detection) in an ablation experiment where we attempt to solve each puzzle ten times using only one of these strategies. The success rates for each strategy are listed in the *EGR*, *MAN*, and *NN* columns of Table 2. Note that since these experiments are equal-effort, in some cases the trial runs using only one strategy have a higher success rate than the trials using all three strategies combined. In Table 4, we also show how many narrow tunnels were detected by each strategy, as well as how many narrow tunnels along the solution path were detected by *only* one strategy.

A trend shown in the tables is that the geometric key point detection strategies are good at certain subsets of the puzzles, e.g., Euclidean-Geodesic Ratio for the *alpha*-type puzzles, and Medial Axis Notch for the *duet* puzzles. The neural-network-based strategy is applicable to a broader cross-section of puzzles, but generally not as effective on a given puzzle as the best geometric strategy.

7.3 Comparisons to Other Planners

To provide a fair equal-effort comparison, we measure the number of edge connections in C-space that our planner attempts to make, and limit other planners to the same number. This choice of metric is motivated by the observation that edge connection, which entails performing collision detection as the puzzle pieces sweep from one configuration to another, dominates the computational cost of planners (e.g. our planner spends over 60% of the total time on edge connections), and gives us a measure of planner performance that is invariant to hardware speeds or code optimization.

We tried the OMPL built-in planners (including RRT, EST, BIT*, RRT-Connect, etc) but none of them were able to complete even the basic alpha-1.0 puzzle after 100 equal-effort attempts, so we exclude them from our table. RDTo and RDTo-Connect (modified from their vanilla OMPL implementations as described in Section 2) fare better, and we run them end-to-end on each puzzle. We also run the PRM planner, with our key configurations seeded as initial milestones, to demonstrate the effectiveness of our blooming-based planner.

D-plan [Zhang et al. 2008] is far more effective at solving the alpha puzzle than RRT, and would be a natural point of comparison, but unfortunately we contacted D-plan's authors for source code, and the code no longer exists. We note, however, that our method succeeds in solving puzzles far more complex than the alpha puzzle benchmark studied by D-plan.

We measure the mean number of edge connections attempted by our planner over its ten attempts at each puzzle; these numbers are listed in the *Mean Edge Conn.* column in Table 2. We impose this edge connection budget on the three baseline planners, halting them if the number of edge connections exceeds the specified budget, or if the planner runs out of memory (around 10 GiB) or runs for more than 10 days without solving the puzzle. We run the baseline planners 100 times for the easy and hard puzzles, to achieve higher precision in the success rate. We only run them 10 times for the extreme puzzles, due to their great computational cost.

Overall, our approach is capable of solving difficult disentanglement puzzles on which the classical planners struggle. The most notable success stories are the duet-g9, duet-g9a, and ABC puzzles.

There is one exception: the RDTo planner is able to solve the key puzzle with 100% success rate, which exceeds that of our approach. We hypothesize that the C-space of this puzzle only contains one bubble connecting two tunnels, so that key's true difficulty is similar to that of the trivial duet-g2 puzzle, despite the visual impressiveness of the puzzle. Given the much higher edge connection budget for key (6.56 G vs 3.13 M), it is not surprising the classical planner can solve this puzzle with a high success rate.

7.4 Feature and Key Configuration Statistics

The number of detected features, and key configuration statistics, are reported in Tables 3 and 4, respectively. Note that different pipeline parameters, as set by Table 1, generate different numbers of features, and the notch statistics for the wire puzzles are not reliable, since all points are equidistant from the medial axis in these puzzles. The key configuration statistics show that our planner scales seamlessly up to the thousands of key configurations required by the toughest puzzles. Notice that different types of puzzles leaned more heavily on different types of key configurations.

7.5 System Resource Consumption

Our pipeline uses the HTCondor cluster to take advantage of the parallelism of our approach, but access to a computing cluster is not required to solve challenging puzzles using our method.

The three most time-intensive parts of our pipeline are:

(1) Computing neural network features, which consumes $3-7$ CPU hours for easy puzzles, $4-27$ hours for hard puzzles, and 52, 63 hours for the ABC and key puzzles respectively. Most of this time is spent computing average distances.
(2) Blooming all key configurations takes less than 3, $4-31$, and $255-320$ CPU hours total for the three difficulty levels.
(3) Forest connection takes less than 2 CPU hours for both the easy and hard puzzles, and only ABC and key require more time (95 and 135 hours respectively.)

Other parts of the pipeline consume relatively negligible time.

In total, using a 4-core 8-thread workstation alone, it takes less than one hour to finish all the stages of our pipeline for the easy puzzles, and up to three days for the most demanding ABC/key puzzles. With access to HTCondor, we are able to solve all puzzles in approximately five wall-clock hours (though performance depends on number of HTCondor machines available, their workloads and hardware specifications, etc.)

In terms of memory consumption, each blooming tree takes less than 100 MiB of memory, and the maximum memory usage during the forest connection stage is 4937 MiB.

8 CONCLUSION

We presented a new pipeline for solving the Piano Mover's Problem for two-piece disentanglement puzzles that stump both humans and existing motion-planning algorithms. The key to our method is a set of algorithms that extract geometric and visual features from objects, which we used to find puzzle configurations that are likely in narrow tunnels of C-space. To take advantage of this configuration prediction technique, we designed a parallel motion planning algorithm which can solve motion planning problems on a distributed system and can scale up to thousands of seed configurations. While there exists the rare case where a baseline planner can outperform our pipeline, they fail to produce any solution for the vast majority of puzzles that we can solve.

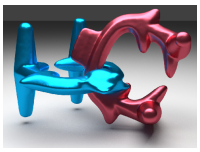

Fig. 10. Hanayama Elk

8.1 Future Work: Challenges, and Beyond SE(3)

Some puzzles we investigated elude our efforts to find key configurations. The Elk puzzle shown in Figure 10 is an example where our geometric features do not lead to useful key configurations, and an easier version of the puzzle on which to train the network is not available. The narrow tunnels in this puzzle involve joint features from both pieces of the geometry, while all our feature detectors focus on each piece independently.

A key assumption underlying the design of our three key point selection strategies in Section 4 is that narrow tunnels correspond to alignment of gap and notch features on the puzzle pieces. In cases like the Elk where this assumption is not valid, our framework could be adapted by augmenting or replacing the three strategies with new geometric strategies tailored to additional types of narrow tunnel geometries. Other potential extensions include improvements to the existing strategies (such as using geodesic loops [Xin et al. 2012] to detect gaps and notches); broadening our neural network approach by training on new exemplars of new narrow tunnel features; or exploring entirely new modalities for key point discovery such as manual guidance.

Throughout this paper we assume that puzzles consist of pairs of rigid moving pieces in 3D. However, many puzzles either contain more than two pieces (the full Enigma), and others involve non-rigid elements like ropes. In these cases, our planner may still work (albeit at additional cost due to the higher-dimensional configuration space), but the key configuration generation would require joint prediction from multiple features on multiple pieces, which is non-trivial compared with our current approach. An interesting start

Table 2. *Left*: our method's success rate at solving each puzzle. We show the success rate when using all three proposed strategies for key point prediction (*Combined* column) as well as compare to our results if only one strategy is used (*EGR*, *MAN*, and *NN* columns). *Right*: comparison against baseline planners, when run with equal effort, measured in terms of number of attempted edge connections. *Notes*: we do not use the medial-axis notch heuristic on the wire puzzles, for which it is meaningless. We write OOM for cases where the planner always runs out of memory before solving the puzzle, and OOT for cases where the planner fails to solve the puzzle before 10 days.

Difficulty	Puzzle Name	Our Success Rate %				Mean Edge Conn.	Baseline Success Rate %		
		EGR	MAN	NN	**Combined**		RDTo	RDTo-connect	PRM
Easy	alpha-g	0	—	0	**20**	18.34 M	4	1	0
	alpha-j	70	—	0	**60**	1.14 M	0	0	0
	alpha-j2	70	—	10	**90**	3.85 M	0	0	0
	alpha-1.0	50	—	20	**30**	4.61 M	0	0	0
	alpha-1.1	90	—	30	**80**	3.56 M	1	0	0
	alpha-z	60	—	0	**80**	1.37 M	0	0	0
	claw	40	— *	0	**10**	93.02 M	0	0	0
	double-alpha	60	—	0	**70**	8.73 M	0	0	0
	duet-g1	60	90	90	**90**	888.40 K	100	98	10
	duet-g2	10	60	100	**100**	3.13 M	0	63	0
Hard	duet-g4	10	30	10	**20**	103.68 M	27	28	0
	duet-g9	0	0	0	**20**	789.74 M	0	OOM	OOM
	duet-g9a	0	10	10	**30**	806.57 M	0	OOM	OOM
	Enigma	0	20	20	**40**	244.30 M	0	0	OOM
	Mobius	0	0	0	**10**	746.12 M	4	OOM	OOM
Extreme	ABC	0	10	60	**10**	5.52 G	OOT	OOM	OOM
	Key	40	0	60	**20**	6.56 G	100	OOM	OOM

Table 3. Mean and standard deviation σ of the number of feature points identified by different feature detection strategies.

Difficulty	Puzzle Part	EGR		MAN	
		Mean	Stdev	Mean	Stdev
Easy	alpha-1.1	3.60	0.92	42.70	123.44
	alpha-1.0	4.50	1.07	23.20	91.72
	alpha-g	6.65	1.19	3.90	1.09
	alpha-j	6.67	1.75	24.82	99.51
	alpha-z	6.85	1.28	2.90	0.89
	double-alpha	5.80	1.66	2.30	0.78
	claw	12.25	2.23	N/A	N/A
	duet ring	4.92	1.57	18.72	59.68
	duet-g1 grid	5.50	1.63	32.40	1.69
	duet-g2 grid	4.70	1.85	63.60	35.11
Hard	duet ring	6.33	1.35	15.33	55.15
	duet-g4 grid	8.70	1.55	183.20	26.37
	duet-g9(a) grid	11.30	1.71	420.00	42.90
	Enigma part 1	13.80	3.74	276.40	33.89
	Enigma part 2	14.80	1.89	173.90	15.92
Extreme	ABC part AB	17.00	2.00	366.20	35.43
	ABC part C	18.50	1.80	185.90	5.13
	Key part 1	18.80	2.44	176.00	10.46
	Key part 2	19.00	2.57	175.00	5.39

would be to place two pieces into a key configuration, and then try to place the third piece into the bubbles formed by the first two. The high-dimensional path planning problems that arise in puzzles with more than two pieces would clearly benefit from our general strategy of seeding path planning with key configurations.

ACKNOWLEDGMENTS

We would like to thank Alec Jacobson, Dave Levin, L. Mahadevan, and Richard Tsai for their discussions and feedback related to this project, and the NSF (CHS-1910274), NSERC, Side Effects Software Inc., and Adobe Inc. for their generous support. The Bellairs Workshop on Computer Animation was instrumental in incubating the research presented in this paper.

REFERENCES

Nancy M Amato, O Burchan Bayazit, Lucia K Dale, Christopher Jones, and Daniel Vallejo. 1998a. Choosing good distance metrics and local planners for probabilistic roadmap methods. In *Robotics and Automation, 1998. Proceedings. 1998 IEEE International Conference on*, Vol. 1. IEEE, 630–637.

Nancy M Amato, O Burchan Bayazit, Lucia K Dale, Christopher Jones, and Daniel Vallejo. 1998b. OBPRM: An obstacle-based PRM for 3D workspaces. In *Proc. Int. Workshop on Algorithmic Foundations of Robotics (WAFR)*. 155–168.

Taeg Sang Cho, Shai Avidan, and William T Freeman. 2010. A probabilistic image jigsaw puzzle solver. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. IEEE, 183–190.

Keenan Crane, Clarisse Weischedel, and Max Wardetzky. 2013. Geodesics in Heat: A New Approach to Computing Distance Based on Heat Flow. *ACM Trans. Graph.* 32, 5, Article 152 (Oct. 2013), 11 pages. https://doi.org/10.1145/2516971.2516977

J. Denny, E. Greco, S. Thomas, and N. M. Amato. 2014. MARRT: Medial Axis biased rapidly-exploring random trees. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*. 90–97. https://doi.org/10.1109/ICRA.2014.6906594

David Eigen and Rob Fergus. 2015. Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture. In *Proceedings of the IEEE international conference on computer vision*. 2650–2658.

Chelsea Finn and Sergey Levine. 2017. Deep visual foresight for planning robot motion. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2786–2793.

Table 4. Key point strategy statistics. For each puzzle, we count the number of narrow tunnels in the puzzle, and the number along a solution path. For each strategy, we list the mean and standard deviation of the number of its key configurations that were used by the planner (over all trials of the puzzle). We also count how many narrow tunnels were found by that strategy, and how many of those tunnels were *critical*: on the solution path, and where only one strategy found that narrow tunnel. Numbers in italics are lower bounds only, due to the difficulty in distinguishing narrow tunnels in some puzzles.

Puzzle Name	Tunnels		EGR-EGR				EGR-MAN				NN				Combined	
	Total	On-path	Mean	Stdev	Tun.	Cr. Tun.	Mean	Stdev	Tun.	Cr. Tun.	Mean	Stdev	Tun.	Cr. Tun.	Mean	Stdev
alpha-g	3	1	16.70	7.01	3	1	0.00	0.00	0	0	25.80	3.46	0	0	42.20	8.20
alpha-j	4	1	31.20	9.00	4	1	0.60	1.80	0	0	25.50	3.14	4	0	54.00	11.05
alpha-j-2	4	1	31.10	6.47	4	1	0.60	1.80	0	0	27.20	2.40	2	0	56.10	7.13
alpha-1.0	4	1	13.00	5.10	4	0	2.60	2.15	0	0	43.70	35.31	1	0	55.00	33.03
alpha-1.1	4	1	8.10	1.14	4	0	0.00	0.00	0	0	25.60	1.69	4	0	31.70	2.72
alpha-z	4	1	36.00	21.80	4	1	0.00	0.00	0	0	28.00	1.26	1	0	59.40	19.17
claw	4	1	161.50	32.69	4	1	-	-	-	-	29.70	0.46	0	0	188.60	32.36
double-alpha	8	2	24.50	11.22	8	2	1.70	1.68	0	0	27.20	1.72	3	0	51.80	11.15
duet-g1	1	1	1.20	1.99	0	0	15.10	10.77	1	0	24.40	2.76	1	0	37.60	10.37
duet-g2	4	2	1.10	1.70	0	0	39.40	30.79	4	0	26.80	1.78	4	0	63.60	30.94
duet-g4	*15*	5	1.10	1.70	0	0	17.30	12.51	*7*	4	81.50	8.03	*10*	1	86.90	7.11
duet-g9	*33*	19	0.50	0.50	0	0	40.10	38.12	*22*	4	112.10	6.55	*25*	10	132.20	20.53
duet-g9a	*33*	17	0.80	0.40	0	0	64.70	31.51	*22*	8	113.70	9.23	*25*	2	144.40	18.33
Enigma	4	1	72.70	18.42	4	0	4.20	2.75	4	0	189.10	14.62	3	0	237.20	20.38
Mobius	-	-	23.40	10.41	-	-	31.60	3.41	-	-	180.50	16.49	-	-	166.40	15.27
ABC	12	7	107.70	31.97	*4*	0	980.20	211.16	*10*	3	1022.00	0.00	*12*	0	2109.90	242.32
Key	3	3	99.20	24.98	*0*	0	1225.10	348.65	*2*	2	1022.00	0.00	*0*	0	2346.30	355.84

Jonathan Gammell, Siddhartha Srinivasa, and Timothy Barfoot. 2015. Batch Informed Trees (BITâĹŰ): Sampling-based optimal planning via the heuristically guided search of implicit random geometric graphs. *Proceedings - IEEE International Conference on Robotics and Automation* 2015 (06 2015), 3067–3074. https://doi.org/10.1109/ICRA.2015.7139620

Xiang Gao, Sébastien Loriot, and Andrea Tagliasacchi. 2019. Triangulated Surface Mesh Skeletonization. In *CGAL User and Reference Manual* (4.14 ed.). CGAL Editorial Board. https://doc.cgal.org/4.14/Manual/packages.html#PkgSurfaceMeshSkeletonization

David Goldberg, Christopher Malon, and Marshall Bern. 2004. A global approach to automatic solution of jigsaw puzzles. *Computational Geometry* 28, 2 (2004), 165–174. https://doi.org/10.1016/j.comgeo.2004.03.007

Rana Hanocka, Amir Hertz, Noa Fish, Raja Giryes, Shachar Fleishman, and Daniel Cohen-Or. 2019. MeshCNN: a network with an edge. *ACM Transactions on Graphics (TOG)* 38, 4 (2019), 1–12.

David Hsu, Jean-Claude Latombe, and Rajeev Motwani. 1999. Path Planning In Expansive Configuration Spaces. *International Journal of Computational Geometry & Applications* 09, 04n05 (1999), 495–512. https://doi.org/10.1142/S0218195999000285

David Hsu, Gildardo Sánchez-Ante, and Zheng Sun. 2005. Hybrid PRM sampling with a cost-sensitive adaptive strategy. In *Proceedings of the 2005 IEEE international conference on robotics and automation*. IEEE, 3874–3880.

Qi-Xing Huang, Simon Flöry, Natasha Gelfand, Michael Hofer, and Helmut Pottmann. 2006. Reassembling Fractured Objects by Geometric Matching. *ACM Trans. Graph.* 25, 3 (July 2006), 569âĂŞ578. https://doi.org/10.1145/1141911.1141925

Brian Ichter, James Harrison, and Marco Pavone. 2018. Learning Sampling Distributions for Robot Motion Planning. 7087–7094. https://doi.org/10.1109/ICRA.2018.8460730

Max Jaderberg, Volodymyr Mnih, Wojciech Marian Czarnecki, Tom Schaul, Joel Z Leibo, David Silver, and Koray Kavukcuoglu. 2016. Reinforcement learning with unsupervised auxiliary tasks. *arXiv preprint arXiv:1611.05397* (2016).

Niels Justesen, Philip Bontrager, Julian Togelius, and Sebastian Risi. 2019. Deep learning for video game playing. *IEEE Transactions on Games* (2019).

Evangelos Kalogerakis, Aaron Hertzmann, and Karan Singh. 2010. Learning 3D Mesh Segmentation and Labeling. In *ACM SIGGRAPH 2010 Papers* (Los Angeles, California). Association for Computing Machinery, New York, NY, USA, Article 102, 12 pages. https://doi.org/10.1145/1833349.1778839

S. Karaman, M. R. Walter, A. Perez, E. Frazzoli, and S. Teller. 2011. Anytime Motion Planning using the RRT*. In *2011 IEEE International Conference on Robotics and Automation*. 1478–1483. https://doi.org/10.1109/ICRA.2011.5980479

L. E. Kavraki, M. N. Kolountzakis, and J. . Latombe. 1998. Analysis of probabilistic roadmaps for path planning. *IEEE Transactions on Robotics and Automation* 14, 1 (Feb 1998), 166–171. https://doi.org/10.1109/70.660866

L. E. Kavraki, P. Svestka, J. . Latombe, and M. H. Overmars. 1996. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation* 12, 4 (Aug 1996), 566–580. https://doi.org/10.1109/70.508439

Diederik P Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*. https://arxiv.org/abs/1412.6980

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*. 1097–1105.

J. J. Kuffner. 2004. Effective sampling and distance metrics for 3D rigid body path planning. In *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA '04. 2004*, Vol. 4. 3993–3998 Vol.4. https://doi.org/10.1109/ROBOT.2004.1308895

J. J. Kuffner and S. M. LaValle. 2000. RRT-connect: An efficient approach to single-query path planning. In *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation*, Vol. 2. 995–1001 vol.2. https://doi.org/10.1109/ROBOT.2000.844730

A. M. Ladd and L. E. Kavraki. 2004. Measure theoretic analysis of probabilistic path planning. *IEEE Transactions on Robotics and Automation* 20, 2 (April 2004), 229–242. https://doi.org/10.1109/TRA.2004.824649

Steven M. Lavalle. 1998. *Rapidly-Exploring Random Trees: A New Tool for Path Planning*. Technical Report. Report No. TR 98-11, Computer Science Department, Iowa State University.

Steven M LaValle. 2006. *Planning algorithms*. Cambridge university press.

Sergey Levine, Peter Pastor, Alex Krizhevsky, Julian Ibarz, and Deirdre Quillen. 2018. Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *The International Journal of Robotics Research* 37, 4-5 (2018), 421–436.

Yangyan Li, Rui Bu, Mingchao Sun, Wei Wu, Xinhan Di, and Baoquan Chen. 2018. Pointcnn: Convolution on x-transformed points. In *Advances in neural information processing systems*. 820–830.

Zhouhui Lian, Afzal Godil, Benjamin Bustos, Mohamed Daoudi, Jeroen Hermans, Shun Kawamura, Yukinori Kurita, Guillaume Lavoué, Hien Van Nguyen, Ryutarou Ohbuchi, et al. 2011. SHREC'11 Track: Shape Retrieval on Non-rigid 3D Watertight Meshes. *3DOR* 11 (2011), 79–88.

Michel J Litzkow, Miron Livny, and Matt W Mutka. 1987. *Condor-a hunter of idle workstations*. Technical Report. University of Wisconsin-Madison Department of Computer Sciences.

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

Jim Mainprice, E Akin Sisbot, Léonard Jaillet, Juan Cortés, Rachid Alami, and Thierry Siméon. 2011. Planning human-aware motions using a sampling-based costmap planner. In *2011 IEEE International Conference on Robotics and Automation*. IEEE, 5012–5017.

Alejandro Newell, Kaiyu Yang, and Jia Deng. 2016. Stacked hourglass networks for human pose estimation. In *European Conference on Computer Vision*. Springer, 483–499.

Jia Pan and Dinesh Manocha. 2016. Fast probabilistic collision checking for sampling-based motion planning using locality-sensitive hashing. *The International Journal of Robotics Research* 35, 12 (2016), 1477–1496. https://doi.org/10.1177/0278364916640908

Lerrel Pinto and Abhinav Gupta. 2016. Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours. In *2016 IEEE international conference on robotics and automation (ICRA)*. IEEE, 3406–3413.

Ruggero Pintus, Kazim Pal, Ying Yang, Tim Weyrich, Enrico Gobbetti, and Holly Rushmeier. 2016. A Survey of Geometric Analysis in Cultural Heritage. *Comput. Graph. Forum* 35, 1 (Feb. 2016), 4–31. https://doi.org/10.1111/cgf.12668

Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. 2017. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 652–660.

Charles R Qi, Hao Su, Matthias Nießner, Angela Dai, Mengyuan Yan, and Leonidas J Guibas. 2016. Volumetric and multi-view cnns for object classification on 3d data. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 5648–5656.

J. H. Reif. 1979. Complexity of the mover's problem and generalizations. In *20th Annual Symposium on Foundations of Computer Science (sfcs 1979)*. 421–427. https://doi.org/10.1109/SFCS.1979.10

Rodriguez, Xinyu Tang, Jyh-Ming Lien, and N. M. Amato. 2006. An obstacle-based rapidly-exploring random tree. In *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006.* 895–900. https://doi.org/10.1109/ROBOT.2006.1641823

K. Shi, J. Denny, and N. M. Amato. 2014. Spark PRM: Using RRTs within PRMs to efficiently explore narrow passages. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*. 4659–4666. https://doi.org/10.1109/ICRA.2014.6907540

Richard Socher, Brody Huval, Bharath Bath, Christopher D Manning, and Andrew Y Ng. 2012. Convolutional-recursive deep learning for 3d object classification. In *Advances in neural information processing systems*. 656–664.

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

M. Strandberg. 2004. Augmenting RRT-planners with local trees. In *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA '04. 2004*, Vol. 4. 3258–3262 Vol.4. https://doi.org/10.1109/ROBOT.2004.1308756

Hang Su, Subhransu Maji, Evangelos Kalogerakis, and Erik Learned-Miller. 2015. Multi-view convolutional neural networks for 3d shape recognition. In *Proceedings of the IEEE international conference on computer vision*. 945–953.

Ioan A. Şucan, Mark Moll, and Lydia E. Kavraki. 2012. The Open Motion Planning Library. *IEEE Robotics & Automation Magazine* 19, 4 (December 2012), 72–82. https://doi.org/10.1109/MRA.2012.2205651 http://ompl.kavrakilab.org.

Timothy Sun and Changxi Zheng. 2015. Computational Design of Twisty Joints and Puzzles. *ACM Trans. Graph.* 34, 4, Article 101 (July 2015), 11 pages. https://doi.org/10.1145/2766961

Andrea Tagliasacchi, Ibraheem Alhashim, Matt Olson, and Hao Zhang. 2012. Mean Curvature Skeletons. *Comput. Graph. Forum* 31, 5 (Aug. 2012), 1735–1744. https://doi.org/10.1111/j.1467-8659.2012.03178.x

N. Vahrenkamp, P. Kaiser, T. Asfour, and R. Dillmann. 2011. RDT+: A parameter-free algorithm for exact motion planning. In *2011 IEEE International Conference on Robotics and Automation*. 715–722.

Peng-Shuai Wang, Yang Liu, Yu-Xiao Guo, Chun-Yu Sun, and Xin Tong. 2017. O-cnn: Octree-based convolutional neural networks for 3d shape analysis. *ACM Transactions on Graphics (TOG)* 36, 4 (2017), 1–11.

Peng-Shuai Wang, Chun-Yu Sun, Yang Liu, and Xin Tong. 2018. Adaptive O-CNN: A patch-based deep representation of 3D shapes. *ACM Transactions on Graphics (TOG)* 37, 6 (2018), 1–11.

Yunhai Wang, Shmulik Asafi, Oliver Van Kaick, Hao Zhang, Daniel Cohen-Or, and Baoquan Chen. 2012. Active co-analysis of a set of shapes. *ACM Transactions on Graphics (TOG)* 31, 6 (2012), 1–10.

S. A. Wilmarth, N. M. Amato, and P. F. Stiller. 1999. MAPRM: a probabilistic roadmap planner with sampling on the medial axis of the free space. In *Proceedings 1999 IEEE International Conference on Robotics and Automation*, Vol. 2. 1024–1031. https://doi.org/10.1109/ROBOT.1999.772448

S. Xin, C. Fu, and Y. He. 2012. Efficiently Computing Exact Geodesic Loops within Finite Steps. *IEEE Transactions on Visualization and Computer Graphics* 18, 06 (jun 2012), 879–889. https://doi.org/10.1109/TVCG.2011.119

Shiqing Xin, Chi-Fu Lai, Chi-Wing Fu, Tien-Tsin Wong, Ying He, and Daniel Cohen-Or. 2011. Making Burr Puzzles from 3D Models. *ACM Trans. Graph.* 30, 4, Article 97 (July 2011), 8 pages. https://doi.org/10.1145/2010324.1964992

Zhenpei Yang, Jeffrey Z Pan, Linjie Luo, Xiaowei Zhou, Kristen Grauman, and Qixing Huang. 2019. Extreme relative pose estimation for RGB-D scans via scene completion. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 4531–4540.

Fangyi Zhang, Jürgen Leitner, Michael Milford, Ben Upcroft, and Peter Corke. 2015. Towards vision-based deep reinforcement learning for robotic motion control. *arXiv preprint arXiv:1511.03791* (2015).

Liangjun Zhang, Xin Huang, Young J. Kim, and Dinesh Manocha. 2008. D-Plan: Efficient Collision-Free Path Computation for Part Removal and Disassembly. *Computer-Aided Design and Applications* 5, 6 (2008), 774–786. https://doi.org/10.3722/cadaps.2008.774-786

Liangjun Zhang and D. Manocha. 2008. An efficient retraction-based RRT planner. In *2008 IEEE International Conference on Robotics and Automation*. 3743–3750. https://doi.org/10.1109/ROBOT.2008.4543785

Liangjun Zhang and Dinesh Manocha. 2010. *Constrained Motion Interpolation with Distance Constraints*. Springer Berlin Heidelberg, Berlin, Heidelberg, 367–384. https://doi.org/10.1007/978-3-642-00312-7_23