HAL
open science

3D Fabrication of 2D Mechanisms

Jean Hergel, Sylvain Lefebvre

To cite this version:

Jean Hergel, Sylvain Lefebvre. 3D Fabrication of 2D Mechanisms. Computer Graphics Forum, 2015, 10.1111/cgf.12555 . hal-01240344

HAL Id: hal-01240344
https://inria.hal.science/hal-01240344

Submitted on 10 Dec 2015

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

3D Fabrication of 2D Mechanisms

Jean Hergel and Sylvain Lefebvre

Inria Nancy Grand-Est, France

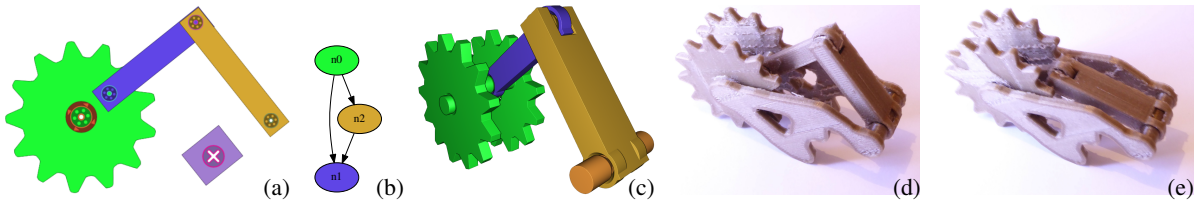

Figure 1: *Our algorithm takes as input a 2D design of a mechanism (a) and produces a 3D model (c) by computing a layout encouraging inclusion between parts, formulating the layout as a graph orientation problem (b). In the graph, edge orientation indicates inclusion relationships. The 3D model can be printed (c,d) and is functional. The chassis is automatically synthesized.*

Abstract

The success of physics sandbox applications and physics-based puzzle games is a strong indication that casual users and hobbyists enjoy designing mechanisms, for educational or entertainment purposes. In these applications, a variety of mechanisms are designed by assembling two-dimensional shapes, creating gears, cranks, cams, and racks. The experience is made enjoyable by the fact that the user does not need to worry about the intricate geometric details that would be necessary to produce a real mechanism.

In this paper, we propose to start from such casual designs of mechanisms and turn them into a 3D model that can be printed onto widely available, inexpensive filament based 3D printers. Our intent is to empower the users of such tools with the ability to physically realize their mechanisms and see them operate in the real world.

To achieve this goal we tackle several challenges. The input 2D mechanism allows for some parts to overlap during simulation. These overlapping parts have to be resolved into non–intersecting 3D parts in the real mechanism. We introduce a novel scheme based on the idea of including moving parts into one another whenever possible. This reduces bending stresses on axles compared to previous methods. Our approach supports sliding parts and arbitrarily shaped mechanical parts in the 2D input. The exact 3D shape of the parts is inferred from the 2D input and the simulation of the mechanism, using boolean operations between shapes. The input mechanism is often simply attached to the background. We automatically synthesize a chassis by formulating a topology optimization problem, taking into account the stresses exerted by the mechanism on the chassis through time.

1. Introduction

Designing and modeling mechanisms is a difficult, highly technical task. Therefore significant research effort is dedicated to automate their generation. For instance, the most advanced techniques are capable of synthesizing complex trains of gears and cams [ZXS*12], produce mechanical automatons from (virtual) animated characters [CTN*13, CLM*13, TCG*14], or generate functional designs with hinges and slides from a high level specification [Koo14]. This is a very important trend of research in a professional context, where efficiency and reliability of the created mechanisms is the top priority.

However, there are many indications that casual designers and hobbyist actually enjoy designing mechanisms, from the video games *The Incredible Machine* (*Dynamix, 1993*) and *Crayon Physics Delux* (*Hudson Soft*) to physics sandbox applications such as *Garry's mod* (*Facepunch Studios*), *Phyzicle* (by *A. Shirinian*) and *Algodoo* (*Algoryx*). Using the later, thousands of users create and share intriguing mechanisms modeled in two dimensions. These are used for educa-

tional and entertainment purposes, but are accurate enough for early prototyping of small mechanisms.

In this paper, we seek to fabricate real mechanisms that can be 3D printed on inexpensive filament printers, starting from an underspecified two-dimensional model of a mechanism. Our goal is to empower casual mechanism designers — kids, teachers, hobbyists — with a way to bring their designs to reality, without having to go through the tedious and difficult modeling process required for fabrication.

The input to our algorithm is the unmodified output of a popular physics sandbox, in this case *Algodoo* (*Algoryx*). It contains the 2D geometry of the mechanical parts, the specification of joints (hinge, fixed), and information regarding which parts can interact during simulation and which are allowed to move across each others. The output of our algorithm is a mesh ready for fabrication on a fused filament fabrication printer.

To achieve our goals, we have to tackle several challenges. First, the mechanism functions in two-dimensions by allowing some parts to overlap. We have to produce a 3D model where these collisions do not occur, ensuring that the corresponding 3D parts are never in contact. Conversely, some contacts are exploited by the mechanism, allowing parts to interact to produce gears, cams, and sliding motions. These interactions have to be preserved within the final mechanism. Second, the two-dimensional specification of the parts is incomplete: The final 3D shape of each part has to be deduced from the simulation of the mechanism. This often involves producing parts with a complex profile in depth allowing other parts to slide within or along them. Finally, the mechanism has to be enclosed and supported by a *chassis*, which acts as the ground body. This chassis is typically missing from the specification and has to be automatically generated.

3D printed mechanisms are very sensitive to bending stresses, in particular orthogonally to the material deposition direction. Consider the model of Figure 1, for which two possible fabricated solutions are shown in Figure 2. If the parts are layered next to each others the wheel tends to move out of its plane (Figure 2, middle): the moving parts are attached to one extremity of their axles and bending stresses produce out-of-plane motions, an effect that cumulates on all axles. This is worsen by the necessary tolerances between moving parts and axles. To reduce this issue our mechanisms exploit both layering and *inclusion* (Figure 2, right). When parts include one another, the bending stresses are absorbed on both extremities of the axles, reducing out-of-plane motions and avoiding to propagate bending stress to other axles.

Contributions

- We solve for the 3D layout of the mechanism by exploiting *inclusions* whenever possible.
- We define the 3D shapes of the final parts by constructive solid geometry (CSG), considering the positions of the parts throughout the mechanical simulation.

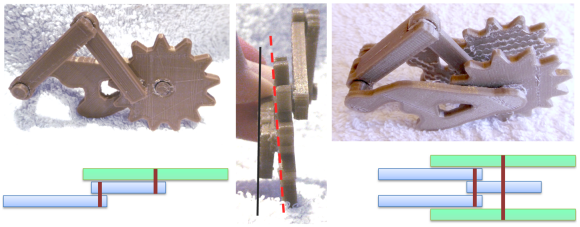

Figure 2: *Possible solutions for the example of Figure 1.* ***Left****: Print out of our result using layering only.* ***Middle****: Out-of-plane motion resulting from using layering.* ***Right****: Print out of our result using inclusion. It produces less out-of-plane motions as axles are connected on both extremities.*

- We automatically synthesize a chassis for the mechanism using topology optimization. Our approach takes into account the forces generated by the mechanism on the chassis during the entire simulation.

Our approach makes no assumption regarding the shape of the mechanical parts: gears, cranks and cams are not tagged in the input. They function properly through the preservation of the set of contacts and interactions from the 2D mechanism into the fabricated mechanism.

Limitations Our algorithm does not detect unrealistic mechanisms (i.e. mechanisms that may generate excessive stresses, that may exhibit singularities, or mechanisms where parts can fall or detach). Some over-constrained 2D designs cannot be resolved by layering or inclusion (Section 4.6).

2. Previous work

Modeling objects for 3D printing is a challenging and technical task, as many constraints from the physical world and the printing process have to be considered. Thus, a recent trend of research proposes novel algorithms to help designers identify and fix problematic geometries, for instance making objects stronger [SVB*12] or ensuring that designs are properly balanced [PWLSH13, BWBSH14]. Algorithms have also been proposed to fabricate articulated [BBJP12, CCA*12] or deformable characters [STC*13] from their virtual counterparts. This typically involves optimizing the geometry and automatically modeling joints or internal structures to achieve the desired degrees of freedom.

Several approaches focus specifically on mechanisms. The work of Zhu et al. [ZXS*12] proposes to automatically generate a mechanism producing a desired motion on multiple rigid objects, typically an animated toy. The mechanism is hidden in a box below the object. The work of Coros et al. [CTN*13] automatically generates a mechanism inside an already articulated model so as to reproduce an animation sequence. Duygu et al. [CLM*13] embed mechanisms called oscillation modules inside an articulated character. The oscillation modules are optimized to reproduce a target animation. Koo et al. [Koo14] automatically produce articulations in a design, from a high level specification of

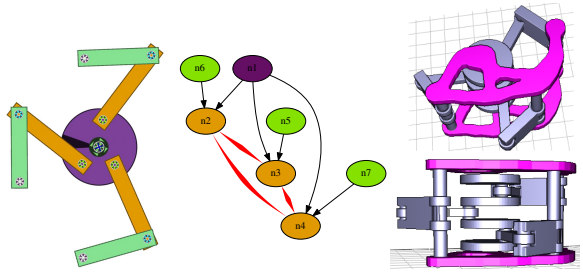

Figure 3: ***Left:*** *Input 2D mechanism.* ***Middle:*** *Graph used to construct the layout. Edge orientation indicates which part include which other, while the red edges show which parts have to be layered.* ***Right:*** *Two views of the generated mechanism. The synthesized chassis appears in pink. The crankshaft configuration automatically results from the inclusion and layering constraints.*

the designers intent: e.g. folding or opening panels covering certain areas. Thomaszewski et al. [TCG*14] and Megaro et al. [MTG*14] propose to assist the user in creating elegant linkage based animated figures. While the former focuses on professional designers, the work of Megaro et al. targets casual users through an accessible, simple interface.

These approaches focus on the automatic synthesis of the mechanism: the user does not directly model its inner workings. While we fully agree on the importance of automatic synthesis of mechanisms, our goal is to also promote the possibility for casual users to directly design mechanisms. Contrary to synthesis techniques, our approach never changes the input mechanism and therefore cannot improve or correct it. We however impose no constraint on the parts and their interactions: we support gears, cranks and cams of any shapes as their functions arise from their 2D geometries and interactions. In contrast synthesis techniques often rely on a set of pre-determined mechanical parts – typically bars and gears – to make the problem tractable.

Several of the aforementioned techniques [CTN*13, CLM*13, TCG*14, MTG*14] synthesize a 2D mechanism which is then turned into a fabricable 3D model. This is solved by a *layering* approach: the potential collisions are identified and the parts are assigned different layers to avoid collisions. This can be formulated as a constrained satisfaction problem (CSP). This is a natural solution, but it tends to place parts side by side on a same axle, sometimes with significant space in between, resulting in out-of-plane motions and bending stresses on the axles (Section 1, Figure 2). In contrast, our technique favors inclusion of parts within each others. Our system is able to locally switch to layering whenever inclusion cannot be used.

3. Overview

The input to our algorithm is a set of N *parts* $\mathcal{P} = \{P_i | i = 0..N-1\}$, each described by a 2D polygon. The polygon of a part may have holes, but has to form a single component. The parts may be connected through hinges and fixed joints. We denote the set of joints:

$$\mathcal{H} = \{(P_i, P_j) | P_i, P_j \text{ connected by an hinge or a fixed joint}\}$$

Our approach involves three steps: layout (Section 4), geometry synthesis (Section 5) and chassis synthesis (Section 6). These three steps are illustrated in Figure 3.

The layout step determines the relative positions of parts and assigns them with a depth interval. We denote $I(P_i) = [l_i, h_i]$ the interval assigned to part P_i, with $l_i \leq h_i$ two integers. The inclusions between parts are determined by orienting edges in a graph capturing contact and collision constraints. The geometry synthesis step determines the exact 3D geometry of each part, using the layout and the motions resulting from the simulated mechanism. The last step synthesizes a *chassis* for the mechanism: a geometry for the main body holding everything together.

4. Mechanical layout

Our approach produces mechanisms favoring inclusion between parts. This is used in particular to resolve cases where the parts overlap in the 2D specification without interacting. The main advantage of this approach is to reduce mechanical jitter: the hinge axles are much stronger when supported on both their extremities. Inclusion alone cannot work on all mechanisms due to additional geometric constraints and interactions between parts. For these cases our approach resorts to layering, but only locally.

The layout process starts by analyzing the simulated mechanism, tracking overlaps and interactions between parts (Section 4.1). This information provides us with a set of observations that are used to make hypotheses regarding which parts can include which others (Section 4.3). Our algorithm greedily add inclusions, until contradictions are detected (such as having both $A \supset B$ and $B \supset A$). These contradictions are transformed into layering rules. The final optimization assigns depth values to the intervals by solving a constrained satisfaction problem (Section 4.4).

4.1. Analysis

Our approach starts by simulating the mechanism to construct the set of observations. It keeps track of *overlaps* – parts that overlap without colliding – and *interactions* – parts that are allowed to be in contact during the simulation. This is done by simulating the input 2D mechanism using the *Box2D* library.

We assume mechanisms to have a periodic motion, and run the simulation until a prior configuration is encountered, or a user selected maximum time is reached. The result is a set of T time frames. We denote by M_i^t the position matrix of part P_i at time $t \in [0, T[$, and denote by $M_i^t P_i$ the polygon of part P_i at time t.

We record all overlaps between parts in a set:

$$\mathcal{O} = \{(P_i, P_j) | \exists\, t \text{ such that } P_i, P_j \text{ overlap at time } t\}$$

We keep track of all interactions between parts during simulation. We denote the set of interacting parts as:

$$\mathcal{C} = \{(P_i, P_j) | \exists\, t \text{ such that } P_i, P_j \text{ are in contact at time } t\}$$

Note that for any two parts P_i, P_j we have $(P_i, P_j) \in \mathcal{O} \Rightarrow (P_i, P_j) \notin \mathcal{C}$ and $(P_i, P_j) \in \mathcal{C} \Rightarrow (P_i, P_j) \notin \mathcal{O}$. We also have $\mathcal{H} \subset \mathcal{O}$ since all parts sharing a joint also overlap.

From these sets we define the mechanism graph as $\mathcal{G} = (\mathcal{P}, \mathcal{H} \cup \mathcal{O} \cup \mathcal{C})$. The set of edges is the union of the joint set $\mathcal{H}$, the overlap set and the contact set. Each edge in the graph is tagged to track which set it belongs to (edges in both $\mathcal{H}$ and $\mathcal{O}$ are tagged as belonging to $\mathcal{H}$).

Additional observations are made about the mechanism. The first are *erasing* cases. A part P_j is said to *erase* a part P_i if the temporal sweep of its polygon covers P_i entirely *and* they are not connected by a joint. More formally:

$$P_j \text{ erases } P_i \text{ if } P_i \setminus \cup_{t \in [0,T[} ((M_i^t)^{-1} M_j^t P_j) = \emptyset \text{ and } (P_i, P_j) \notin \mathcal{H}$$

In such cases, the part P_i cannot include the part P_j as it would have to be split in two independent parts to fit P_j within its depth. If P_i, P_j are connected by a joint, then P_i remains connected through the axle and is not erased by P_j.

The second are *detachment* cases. A part P_j is said to *detach* $P_i, P_k (k \neq j)$ if the temporal sweep of P_j overlaps with the joint between P_i, P_k. Note that the overlap test is considering the diameter of the joint axle (5 mm in practice). If P_j detaches P_i, P_k then we request that P_j includes both P_i and P_k to avoid having P_j cross the axle between P_i, P_k.

These cases are illustrated Figure 4.

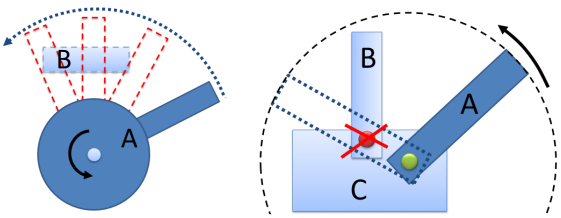

Figure 4: ***Left:*** *The bar A sweeps across the (fixed) bar B, covering it entirely during the simulation. A is said to erase B. In this case, B cannot possibly include A within its depth or it would be disconnected in two independent parts.* ***Right:*** *During simulation, the bar A sweeps across the hinge between B and C. A is said to detach B and C. In this case we request A to include B and C so that their axle is not cut by A. Note that these constraints will be combined and may conflict, in which case layering will be locally performed.*

4.2. Mechanical configurations

The mechanisms we generate can produce three types of mechanical configurations between two parts A and B. Each results in a rule regarding their intervals:

1. A includes B (denoted by $A \supset B$)
$$\Rightarrow I(A) \supset I(B)$$
2. A interacts with B
$$\Rightarrow (I(A) \setminus \cup_{\forall P \subset A} I(P)) \cap \left(I(B) \setminus \cup_{\forall Q \subset B} I(Q)\right) \neq \emptyset$$
3. A layered with B
$$\Rightarrow I(A) \cap I(B) = \emptyset$$

These configurations are depicted in Figure 5. They are mutually exclusive: if A is layered with B they cannot interact as they share no interval in depth. If A is layered with B, then B cannot be included within A (and vice-versa) as they also share no interval. If A includes B, then the volume of B is carved out from A (see Figure 5 top-left) and therefore they cannot interact.

Whenever possible our algorithm favors inclusion over the other two configurations.

4.3. Generating inclusions

The goal of this step of the algorithm is to generate as many inclusion configurations as possible. To this end, we exploit the observations made during analysis and greedily attempt to include parts into one another. Whenever inclusion cannot be used we fallback to layering. At this stage we only determine relationships between the parts intervals (e.g. $I(A) \supset I(B)$). The exact values of the depth intervals will be computed at the next stage (Section 4.4).

Inclusion relationships are determined by orienting the $\mathcal{H}$ and $\mathcal{O}$ edges of the mechanism graph $\mathcal{G}$ (Section 4.1). An edge $(P_i \rightarrow P_j)$ implies that $I(P_i) \supset I(P_j)$. The goal is to find an acyclic orientation: an inclusion cycle would produce an unsolvable case of circular inclusion, e.g. $I(P_i) \supset I(P_j) \supset I(P_i)$. When orienting the edges we take care not to contradict the interaction rules (recall that *A includes B* is incompatible with *A interacts with B*, see Section 4.2). This is done by verifying that no inclusion path is formed between two parts A and B if they interact in the input mechanism.

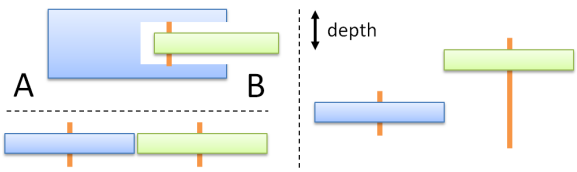

Figure 5: *Two parts in the three possible configurations, as seen from above. Inclusion (top-left), interaction (bottom left), layering (right). The configurations can be mixed, i.e. a part can include others that are in a layering configuration.*

The orientation of some of the edges is constrained by the *erasing* and *detachment* cases:

$$\begin{array}{lcl} P_j \text{ erases } P_i & \Rightarrow & I(P_j) \supset I(P_i) \\ P_j \text{ detaches } (P_i, P_k) & \Rightarrow & I(P_j) \supset I(P_i) \text{ and } I(P_j) \supset I(P_k) \end{array}$$

As a result of these constraints, contradictions may appear during the orientation process, where both $I(P_i) \supset I(P_j)$ *and* $I(P_i) \subset I(P_j)$ are required. Such cases are resolved by switching to a layering configuration for P_i and P_j.

We orient the edges of $\mathcal{G}$ considering edges in $\mathcal{H}$ first and then edges in $\mathcal{O}$. In each subset, we start by the edges with a constraint. This order is important: we prefer to avoid layering on joints (set $\mathcal{H}$) since this is the case where fragile axles are generated. We therefore orient the edges in $\mathcal{H}$ first, so that contradictions are more likely to appear on the edges of $\mathcal{O}$ which are later considered.

For each edge in sequence, we first determine which orientations are possible. This involves checking for an existing constraint, verifying whether an orientation violates any interaction rule, and then checking whether an orientation would produce an inclusion cycle. If two orientations are possible we use a heuristic to select the orientation of lowest cost (see below). If no orientation is possible, we have encountered a contradiction.

Contradictions. We deal with contradictions by removing the problematic edge from the graph and by adding a layering configuration rule between the parts: $I(P_i) \cap I(P_j) = \emptyset$. This forces P_i and P_j to be in different depth intervals instead of being included into one another. However, this introduces an additional requirement on the graph orientation: the parts P_i and P_j should not have any common descendant in the oriented graph. Let us assume such a descendant P_k exists. We would have $I(P_i) \supset I(P_k)$ and $I(P_j) \supset I(P_k)$ which gives $(I(P_i) \cap I(P_j)) \supset I(P_k)$. This directly contradicts $I(P_i) \cap I(P_j) = \emptyset$. To account for this, every time a contradiction is detected we add the additional constraint that P_i, P_j should not have a common descendant. The set of descendants is easily maintained during the graph orientation algorithm, and we reject any edge orientation that would violate such a requirement. When checking for common descendants we also follow interaction edges (edges in $\mathcal{C}$) to prevent the layered parts to include parts that have to interact, i.e. $P_u \subset P_i, P_v \subset P_j$ where P_v, P_u interact : this would result in a similar contradiction.

Resolving a contradiction by layering removes all detachment constraints between P_i, P_j: as both parts will be placed in non-intersecting depth intervals, they can no longer cross their respective axles.

Since the set of constraints is updated, and because some earlier choices may violate the new constraints, we restart the graph orientation every time a contradiction is resolved. When the process restarts edges are traversed the same order, skipping the edges removed due to contradictions.

Chassis. The chassis appears as a part in the graph. To guarantee that it includes all other parts, we orient the chassis edges prior to considering any other edge in the graph.

Figure 2 illustrates a case where the chassis is involved in a detachment constraint. In such cases, we request the detaching primitive to be the most included. That is, P_j detaches (P_i, C) implies $I(P_j) \subset I(P_i)$ and $I(P_j) \subset I(C)$, where C is the chassis. The axle between C and P_i will exist around P_j – even though P_j cuts it, it will exist in two parts attaching C and P_i on both sides. E.g. in Figure 2 the main wheel axle is cut by the inner arms. Nevertheless the wheel is properly connected on both sides to the chassis, and remains a single part through the axle with the arm. This is possible as long as P_j does not erase P_i, in which case layering would automatically be used between P_j and P_i. Indeed a contradiction would then appear when orienting the edge (P_i, P_j).

Orientation cost heuristic. Whenever we can freely choose the orientation of an edge, we apply the following cost heuristic. In absence of constraints our goal is to avoid thickening the parts too much. In other words, we want to keep the size of the depth intervals $|I(P_i)|$ small. The intervals are optimized at the next step (Section 4.4) and therefore we do not know their exact size during graph orientation. However, we can easily determine a lower bound. Consider a path in the oriented graph from part P_i to part P_k: $P_i \rightarrow ... \rightarrow P_k$ and denote L the length of this path. Since we have $P_i \supset ... \supset P_k$, it follows that $|I(P_i)| \geq L$. We therefore seek to minimize the length of the longest path from every node. When orienting an edge (P_i, P_j) we select the orientation minimizing $\mathcal{L}(\mathcal{G}_o, P_i) + \mathcal{L}(\mathcal{G}_o, P_j)$ where $\mathcal{L}(\mathcal{G}_o, P)$ computes the longest path from P to any other node in the oriented graph $\mathcal{G}_o$ (the graph with only the previously oriented edges, and the edge being tested).

4.4. Depth intervals

The previous step produces a number of rules relating the depth intervals of the parts. The goal of this section is to compute the depth values assigned to the lower and upper bounds of each interval $I(P_i) = [l_i, h_i]$. We assign integer depth values to the intervals, which are later mapped to physical thicknesses in the final object (see Section 5). Intervals where $l_i = h_i$ correspond to parts having the minimal thickness (2 mm in practice).

After the analysis we obtain three types of interval rules: inclusion (e.g. $I(P_i) \supset I(P_j)$), layering (e.g. $I(P_i) \cap I(P_j) = \emptyset$) and interactions.

Inclusion. Inclusions are captured by the oriented edges in $\mathcal{G}$. They result in the following inequalities:

$$I(P_i) \supset I(P_j) \Rightarrow l_i < l_j \text{ and } h_i > h_j$$

Note the use of strict inequalities, which guarantees that the including part (P_i) has one layer on each side of the included part (P_j). This ensures that axles are supported on both sides.

Layering. Layering rules are produced when resolving contradictions on edge orientations. We distinguish layering due to the removal of an overlapping edge ($\in \mathcal{O}$) from the layering due to the removal of an hinge edge ($\in \mathcal{H}$).

On overlapping edges:

$$P_i \text{ layered by overlap with } P_j \Rightarrow l_i > h_j + 1 \text{ or } l_j > h_i + 1$$

These inequalities guarantee that P_i, P_j will have a spacing of at least one between them, ensuring an including parent piece will support their axles on both sides.

On hinge edges:

$$P_i \text{ layered by hinge with } P_j \Rightarrow l_i = h_j + 1 \text{ or } l_j = h_i + 1$$

This constrains both parts to appear next to each others through depth, ensuring that the axle between them is as short as possible.

Interactions. These rules are necessary to enforce that contacts exploited by the mechanisms (gears, racks) are properly captured by the 3D model. They result in the following equalities:

$$(P_i, P_j) \in \mathcal{C} \Rightarrow l_i = l_j \text{ or } l_i = h_j \text{ or } h_i = l_j \text{ or } h_i = h_j$$

This ensures that the parts will properly interact through their top or bottom layers. This rule is more restrictive than it could be, since in principle the parts interact as long as they share an interval where no included part exists (see Section 4.2). However, we found it sufficient in practice while reducing the combinatorial complexity for the CSP solver.

4.5. Solver

We directly translate these rules into an integer constraint problem that we solve using *Minion* [GJM06]. We restrict the space of integer to $[0, 2 \cdot N]$, with N the number of parts.

Minion returns the solution minimizing the sum of part thicknesses, that is $\sum_{i=0}^{N} |I(P_i)|$. As Minion performs an exhaustive search within the solution space, we configure it to return the best solution found after at most 90 seconds.

Once the intervals are determined for each part, we are ready to generate the final geometry of the 3D parts.

4.6. Discussion, failure cases

At worse our algorithm eliminates all edges between the parts in the graph and includes everything in the chassis. The parts will then be layered within the chassis, which becomes a crankshaft hosting the layered mechanism.

However, this does not guarantee success as some mechanisms cannot be layered. Such a case is shown Figure 6. The only solution involves modifying the shape of the fixed joint between the two bars. This is not considered by our system nor, to the best of our knowledge, by any of the existing techniques. In such cases, the CSP solving for the intervals will admit no solution.

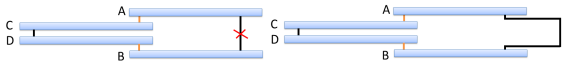

Figure 6: ***Left:*** *Four bars seen from the top. Black segments indicate a fixed joint, while orange indicates a hinge joint. C,D can rotate around A,B but will cut the fixed axle between A,B. This system cannot be resolved by assigning different layers to A,B,C,D.* ***Right:*** *The only solution is to change the shape of the axle, making room for C,D.*

5. Part geometry synthesis

This step of the process takes as input the set of parts $P_i, i = 0...N$ and their corresponding depth intervals $I(P_i) = [l_i, h_i]$. The output is the 3D geometry of each part.

In order to produce the 3D geometry we take into account the following:

- Parts have to be carved to allow for passage of other, included parts.
- Parts that come in contact have to be modified to take a spacing tolerance into account (0.4 mm in practice).
- Parts that slide along others without being attached to the chassis have to be maintained at their selected depth.
- The geometry of hinge joints has to be produced, enabling the mechanism to print pre-assembled.

The base shape of a part is formed by the linear extrusion along the z axis (depth) of its 2D shape. We denote by B_i the base volume of part P_i. The position of the part in depth is computed such that pieces allotted in consecutive layers are separated by a small space. For an interval $[l_i, h_i]$, the extrusion takes places between $z_i^l = (t+s)l_i$ and $z_i^h = (t+s)(h_i+1) - s$ where t is the minimal thickness (2 mm) and s the spacing tolerance between consecutive parts (0.4 mm).

The final shape S_i is obtained from B_i by subtracting from the initial volume the time sweep of the pieces included in P_i, as well as the time sweep of the pieces in contact with P_i. All subtracted pieces are dilated by the contact tolerance spacing. The volume S_i is precisely defined as:

$$S_i = B_i \setminus \left(\cup_{j \in \mathcal{N}(i)} \cup_{t \in [0,T[} \left((M_i^t)^{-1} M_j^t B_j \right) \oplus \mathcal{B}_{0.4}^z \right)$$

where $\mathcal{N}(i) = \{j | (P_i, P_j) \in \mathcal{O} \cup \mathcal{C}\}$, $\oplus$ is the dilation operator and $\mathcal{B}_{0.4}^z$ is a cylinder of axis z, of height and diameter 0.4 mm. We compute the shape by a combination of 2D operations using the *Clipper* library [Joh10], and a few 3D CSG operations.

Free assemblies. The mechanism might include parts, or sub–assemblies of parts that are not connected to the background by any hinge: they are only sliding along other parts. We call these *free assemblies*. They are detected as disconnected components in the graph with only $\mathcal{H}$ edges.

Free assemblies require special care: in 3D nothing prevents them from falling out of their assigned depth interval – recall however that we assume that they are properly locked

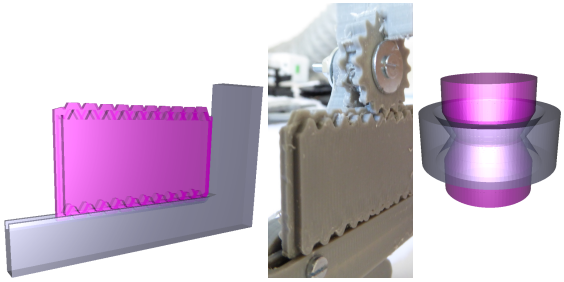

Figure 7: ***Left:*** *The part in pink slides along the L-shaped part. Our approach generates fins around the sliding part so that it is locked in depth inside surrounding parts.* ***Middle:*** *Printed result, note how the gear is also carved by the fin.* ***Right:*** *Interlocking shape of the axles allowing for pre-assembled printing on filament based printers.*

inside the 2D mechanism, see Section 1. We address this issue by creating fins (protrusions) along the sliding parts, see Figure 7. These fins are added to the base shapes B_i of the parts, and are thus subtracted from the other parts that are in contact (with a spacing tolerance). They physically constrain the parts to remain aligned in depth.

This approach in currently limited to the case of free assemblies sliding against non-free assemblies: we do not support several free assemblies sliding against each others.

Hinges. The geometry of the hinges is designed to allow for pre-assembled printing (see Figure 7, right). We add the hinges by CSG, carving the parts to let the axles through. If the part is too narrow the axle geometry will introduce a local bulge to ensure the axle fits properly – in some cases this can prevent the mechanism to function, e.g. if a surface along which two parts interact is modified. We do not currently address this issue.

After this stage the mechanism is almost ready to print, but still lacks a chassis.

6. Chassis synthesis

When designing mechanisms in 2D the user attaches primitives to the background 'wall'. When creating the 3D counterpart of the mechanism we have to synthesize a *chassis* acting as the background. We produce a 2D geometry that is used to sandwich the 3D mechanism in between two walls, as illustrated Figure 2. Alternatively the chassis could be extruded to the full mechanism depth and carved like any other part, see Section 5. This is unnecessary unless the chassis is a crankshaft, so to reduce material use and print time we prefer the sandwiching approach in practice.

The trade-offs in synthesizing the chassis are that we want it to be strong enough to support the efforts generated by the mechanism, but at the same time we would like to keep it small to reduce material use, print time and for aesthetics reasons (a thick chassis would hide the mechanism entirely). This problem is elegantly answered by topology optimization techniques [Ben89].

6.1. Background on topology optimization

We cast chassis synthesis as a case of 2D topology optimization for minimizing the compliance energy [Ben89, Sig01]. The optimization domain is a grid of square elastic elements where each element i, j takes a density $\rho_{i,j} \in [\rho_{min}, 1]$. We denote by ρ the vector of all element densities. Given a choice of densities and a set of fixed elements where the structure is anchored, the finite element method can be used to compute the planar deformation due to a set of forces $\mathbf{f} = f_1, ..., f_n$ located at the grid nodes (element corners). The displacement vector for all grid nodes $\mathbf{u}$ is obtained by solving $\mathbf{K}(\rho) \cdot \mathbf{u} = \mathbf{f}$ where $\mathbf{K}(\rho)$ is the global stiffness matrix assembled from the elements. The stiffness matrix of an element $\rho_{i,j}$ is given by $\rho_{i,j}^3 K_e$ where K_e is the 8×8 stiffness matrix of a square element in the target material.

The compliance energy is defined as $E(\rho) = \mathbf{f} \cdot \mathbf{u}$. Minimizing the compliance maximizes the rigidity of the system under the given forces. This energy is minimized by gradient descent under the constraint that $\sum_{i,j} \rho_{i,j} = A$, where A is the target area of the produced structure. Thanks to the cubic exponent in the per-element stiffness, the system tends to use only 0 or 1 for $\rho_{i,j}$.

6.2. Chassis optimization

The chassis has to resist to the forces exerted by the mechanism at all times. We therefore record the forces at the joints attached to the background for the entire simulation. This gives us a set of T force configurations, where T is the number of time frames. We optimize the chassis by maximizing the compliance over all time frames, that is $E(\rho) = \sum_{t \in T} \mathbf{f}_t \cdot \mathbf{u}_t$ where $\mathbf{K}(\rho)\mathbf{u}_t = \mathbf{f}_t$. This is made efficient by pre-factoring the stiffness matrix $\mathbf{K}$ and then solving for $\mathbf{u}_t$ for all time frames.

We select a resolution of 2 mm per pixel, a Poisson ratio of 0.35 and an elastic modulus of 2.3 GPa (ABS plastic material). The target area A is set to 10% and then reduced until the structure is disconnected. We select the last value generating a fully connected structure. The final outline of the shape is extracted by smoothing out the result with a box filter of size 3×3 and then contouring the isovalue 0.25. The process is summarized Figure 8.

7. Results

We modeled all our results using *Algodoo*, with the only constraint that the mechanisms have to function properly in 2D and be in a valid position (interacting parts such as gears should not overlap). By default the models are rescaled so that their bounding box fits a 150×150 mm square.

All the 3D mechanisms are generated automatically from the scenes created in *Algodoo*, without any user intervention.

Figure 8: *Chassis synthesis by topology optimization.* ***Left:*** *four forces resulting from hinges at a given time frame (pink) and attachment point (gray).* ***Middle:*** *Optimized densities for these forces.* ***Right:*** *Final shape after smoothing and contour extraction.*

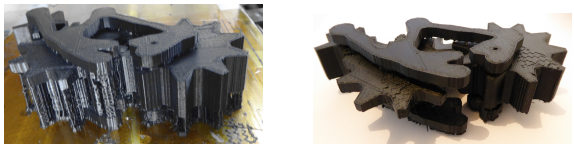

Figure 9: ***Left:*** *Model after printing, with weak infill support.* ***Right:*** *Cleaned model. The gears have rotated, note the small leftover from the infill pattern on top of the gears.*

Figure 10 shows a number of printed results. From top-left to bottom right: the *Gear Puppet* model is made of six gears with custom shapes. The small brown box indicates the base for the chassis. The *EG flag* model is a case of sliding part. The model has automatically generated fins along the sliding part that are carved from both the L-shaped part and the gear. This model broke during clean up and we used screws to repair it (see also Section 7.1). The *Scissor* model is a case where no chassis is needed. It is actioned by a hidden mechanism which is ignored by our system after simulation. This shows how our algorithm alternates inclusions as a result of the edge orientation cost heuristic. The *Wheel* model illustrates how part geometry is automatically created to allow for passage of included objects. The green box at the bottom of the design indicates the base of the chassis. The *Gear Train* model is case of cyclic interaction between gears, creating a layering between the two central gears. The *Three Leg* model is a case that results automatically in a crankshaft. It is also shown Figure 3.

The complexity of our printed results is limited by the capability of our printers. We show in Figure 11 and Figure 12 outputs of our algorithm on more complex designs. Our system is capable of keeping the overall design thickness small, while exploiting inclusions wherever possible.

Performance. Most results are computed in a few seconds. For instance, for the small model Figure 1 graph orientation takes 154 ms, Minion returns the CSP solution in 7 ms. For the larger model Figure 12 graph orientation takes 489 ms, Minion returns the CSP solution after 90 seconds as it explores for the best possible solution until the timeout. Running for longer does not return a better solution, but Minion has to finish exploring the space to guarantee the optimal is found. Chassis optimization takes 2.34 seconds.

7.1. 3D printing

We print all our objects on inexpensive filament printers in ABS and PLA plastic: a Replicator 1 from *Makerbot*, and Ultimakers 1 and 2 from *Ultimaker*.

All objects are printed in one piece, using support. We experimented both with dissoluble plastic and a weak filling pattern for support. We had better results using a weak infill pattern. See Figure 9 for an example of a print before and after cleanup.

Our mechanisms exploit the fact that 3D printers can produce pre-assembled articulated objects, so that we do not have to consider the assembly stage. However, this is not necessarily the best option of filament printers. In particular after printing some force has to be exerted to free the mechanism from inaccessible support. One issue we encountered is breaking of the plastic axles when applying force. The *EG flag* design, for instance, broke when we freed the sliding part and we had to use screws to reassemble it manually. An interesting direction of future work is to split the mechanism into pieces that are easy to assemble [LBRM12, VGB*14].

8. Conclusion

A major advantage of our algorithm is that it avoids searching for explicit kinematic configurations. The algorithm has no notion of what a gear or rack is, their function is solely given by their shape and the simulation. All the mechanical configurations, for instance the crankshafts, automatically emerge from the set of constraints. Inclusion produces parts holding axles on both sides, which is generally less sensitive to mechanical jitter and stress.

The are limitations in our current implementation. First, joints can only be defined between two parts. Second, we do not detect the case where a part is erased (Section 4.1) by the cumulative effect of two or more other parts. Both of these cases lead to a large number of possibilities for including parts into one another. One possibility is to fallback to layering in such situations.

There can be aesthetics considerations to using inclusion or layering, and to which part should include which other. User control can be achieved by directly editing the constraint graph, allowing to explore several possibilities.

Our approach makes designing mechanisms less difficult by automatically dealing with many of the intricate technical details required to generate a valid 3D geometry for a pre-assembled mechanism. We hope it will help hobbyists, teachers, educators and 3D printing enthusiasts to fully exploit the potential of their printers.

Acknowledgments This work was funded by ERC ShapeForge (StG-2012-307877). We thank the Région Lorraine for funding some of our equipment, as well as the anonymous reviewers for their help with improving the paper.

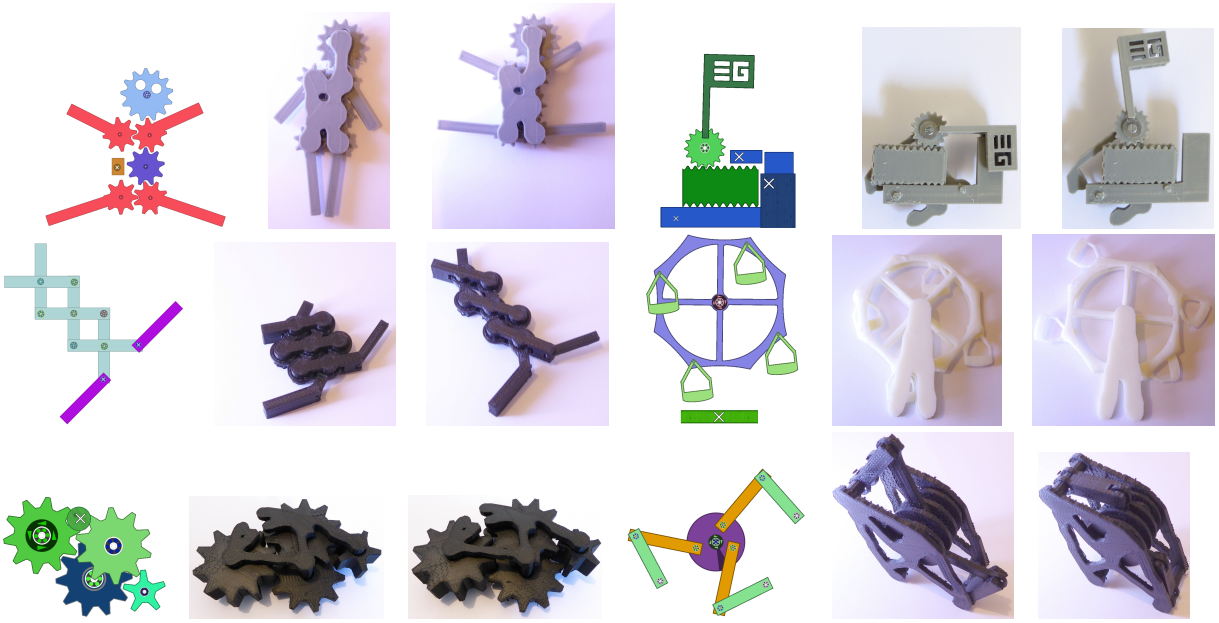

Figure 10: *A variety of results automatically generated from the input 2D design. Please also refer to the accompanying video clips to see them in motion. All these results are 3D printed on filament printers.*

References

[BBJP12] BÄCHER M., BICKEL B., JAMES D. L., PFISTER H.: Fabricating articulated characters from skinned meshes. *ACM Transactions on Graphics 31*, 4 (2012), 47:1–47:9. 2

[Ben89] BENDSØE M. P.: Optimal shape design as a material distribution problem. *Structural Optimization 1* (1989), 192–202. 7

[BWBSH14] BÄCHER M., WHITING E., BICKEL B., SORKINE-HORNUNG O.: Spin-it: Optimizing moment of inertia for spinnable objects. *ACM Transactions on Graphics 33*, 4 (2014), 96:1–96:10. 2

[CCA*12] CALÌ J., CALIAN D. A., AMATI C., KLEINBERGER R., STEED A., KAUTZ J., WEYRICH T.: 3d-printing of non-assembly, articulated models. *ACM Transactions on Graphics 31*, 6 (2012), 130:1–130:8. 2

[CLM*13] CEYLAN D., LI W., MITRA N. J., AGRAWALA M., PAULY M.: Designing and fabricating mechanical automata from mocap sequences. *ACM Transactions on Graphics 32*, 6 (2013), 186:1–186:11. 1, 2, 3

[CTN*13] COROS S., THOMASZEWSKI B., NORIS G., SUEDA S., FORBERG M., SUMNER R. W., MATUSIK W., BICKEL B.: Computational design of mechanical characters. *ACM Transactions on Graphics 32*, 4 (2013), 83:1–83:12. 1, 2, 3

[GJM06] GENT I. P., JEFFERSON C., MIGUEL I.: Minion: A fast, scalable, constraint solver. In *Proceedings of ECAI 2006* (2006), pp. 98–102. 6

[Joh10] JOHNSON A.: Clipper - an open source freeware library for clipping and offsetting lines and polygons, 2010. 6

[Koo14] Creating works-like prototypes of mechanical objects. *ACM Transactions on Graphics 33* (2014), 217:1–217:9. 1, 2

[LBRM12] LUO L., BARAN I., RUSINKIEWICZ S., MATUSIK W.: Chopper: Partitioning models into 3D-printable parts. *ACM Transactions on Graphics 31*, 6 (2012), 129:1–129:9. 8

[MTG*14] MEGARO V., THOMASZEWSKI B., GAUGE D., GRINSPUN E., COROS S., GROSS M.: Chacra: An interactive design system for rapid character crafting. In *Symposium on Computer Animation (SCA)* (2014), pp. 123–130. 3

[PWLSH13] PRÉVOST R., WHITING E., LEFEBVRE S., SORKINE-HORNUNG O.: Make It Stand: Balancing shapes for 3D fabrication. *ACM Transactions on Graphics 32*, 4 (2013), 81:1–81:10. 2

[Sig01] SIGMUND O.: A 99 line topology optimization code written in matlab. *Structural and Multidisciplinary Optimization 21*, 2 (2001), 120–127. 7

[STC*13] SKOURAS M., THOMASZEWSKI B., COROS S., BICKEL B., GROSS M.: Computational design of actuated deformable characters. *ACM Transactions on Graphics 32*, 4 (2013), 82:1–82:10. 2

[SVB*12] STAVA O., VANEK J., BENES B., CARR N. A., MECH R.: Stress relief: improving structural strength of 3d printable objects. *ACM Transactions on Graphics 31*, 4 (2012), 48:1–48:11. 2

[TCG*14] THOMASZEWSKI B., COROS S., GAUGE D., MEGARO V., GRINSPUN E., GROSS M.: Computational design of linkage-based characters. *ACM Transactions on Graphics 33*, 4 (2014), 64:1–64:9. 1, 3

[VGB*14] VANEK J., GALICIA J. A. G., BENES B., MÄŻCH R., CARR N., STAVA O., MILLER G. S.: Packmerger: A 3d print volume optimizer. *Computer Graphics Forum 33*, 6 (2014), 322–332. 8

[ZXS*12] ZHU L., XU W., SNYDER J., LIU Y., WANG G., GUO B.: Motion-guided mechanical toy modeling. *ACM Transactions on Graphics 31*, 6 (2012), 127:1–127:10. 1, 2

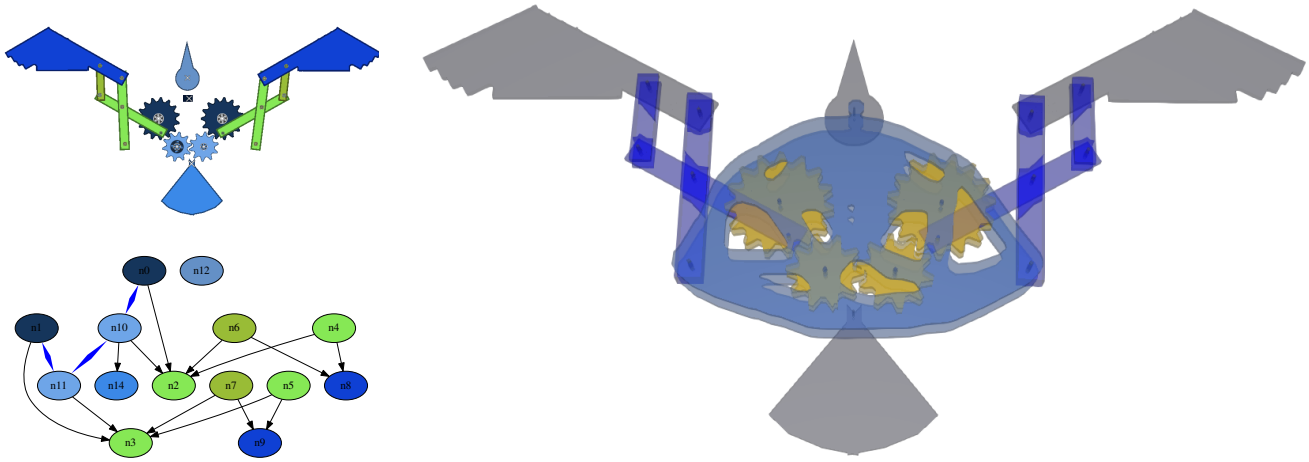

Figure 11: Eagle *model. The mechanism shown in transparency (bottom) is generated from the design (top). The mechanism graph (top left) shows inclusions and contact edges (bold blue). No contradictions were encountered: there is no layering. Note the double gears including the bars, as well as the space carved for the inner bars in the bars actioning the wings.*

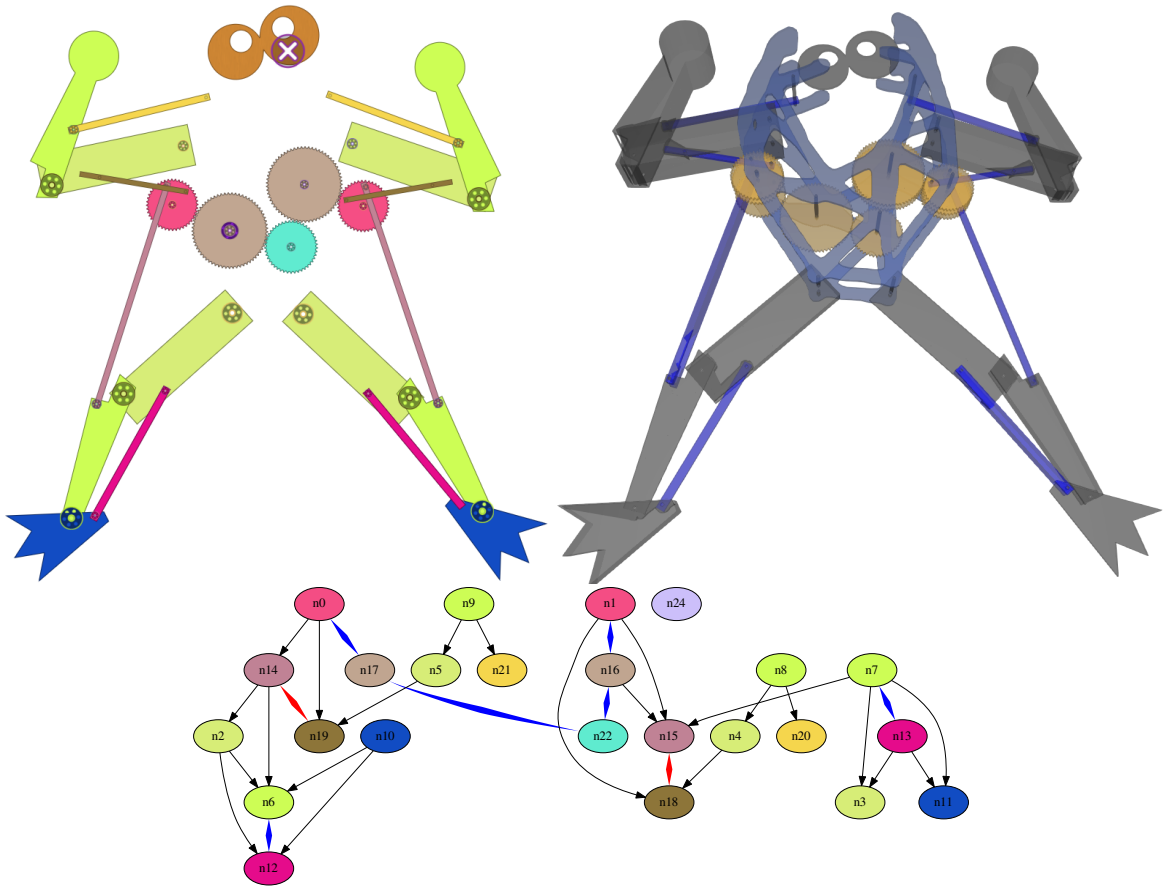

Figure 12: Frogger *model. The mechanism shown in transparency (top right) is generated from the design (top left). This model contains two crankshafts for steering the arms and legs. These were created by layering constraints (red bold edges in the graph). Inclusions are used in most places. Note the lightweight synthesized chassis. One of the large wheel is doubled to avoid the bottom bar of the neighboring crankshaft which overlaps during motion (edge between n15 and n16 in the graph).*