Trusscillator: a System for Fabricating Human-Scale Human-Powered Oscillating Devices

Robert Kovacs*
Hasso Plattner Institute, University of Potsdam, Germany
robert.kovacs@hpi.de

Lukas Rambold*
Hasso Plattner Institute, University of Potsdam, Germany
lukas.rambold@student.hpi.de

Lukas Fritzsche
Hasso Plattner Institute, University of Potsdam, Germany
lukas.fritzsche@student.hpi.de

Dominik Meier
Hasso Plattner Institute, University of Potsdam, Germany
dominik.meier@student.hpi.de

Jotaro Shigeyama
Hasso Plattner Institute, University of Potsdam, Germany
jotaro.shigeyama@hpi.de

Shohei Katakura
Hasso Plattner Institute, University of Potsdam, Germany
shohei.katakura@hpi.de

Ran Zhang
Hasso Plattner Institute, University of Potsdam, Germany
ran.zhang@hpi.de

Patrick Baudisch
Hasso Plattner Institute, University of Potsdam, Germany
patrick.baudisch@hpi.de

ABSTRACT

Trusscillator is an end-to-end system that allows non-engineers to create human-scale human-powered devices that perform oscillatory movements, such as playground equipment, workout devices, and interactive kinetic installations. While recent research has been focusing on generating mechanisms that produce specific movement-path, without considering the required energy for the motion (kinematic approach), Trusscillator supports users in designing mechanisms that recycle energy in the system in the form of *oscillating* mechanisms (dynamic approach), specifically with the help of coil-springs. The presented system features a novel set of tools tailored for designing the *dynamic experience* of the motion. These tools allow designers to focus on user experience-specific aspects, such as motion range, tempo, and effort while abstracting away the underlying technicalities of eigenfrequencies, spring constants, and energy. Since the forces involved in the resulting devices can be high, Trusscillator helps users fabricate from steel, by picking out appropriate steal springs, generating part lists, and by producing stencils and welding jigs that help weld with precision. To validate our system, we designed, built, and tested a series of unique playground equipment featuring 2-4 degrees of movement.

CCS CONCEPTS

• **Human-centered computing** → Human computer interaction (HCI); Interactive systems and tools.; • **Applied Computing** → Operations Research; Computer Aided Manufacturing..

*These authors contributed equally.

UIST '21, October 10–14, 2021, Virtual Event, USA

ACM ISBN 978-1-4503-8635-7/21/10.
https://doi.org/10.1145/3472749.3474807

KEYWORDS

personal fabrication, dynamics, mechanical oscillation, welding

ACM Reference Format:
Robert Kovacs, Lukas Rambold, Lukas Fritzsche, Dominik Meier, Jotaro Shigeyama, Shohei Katakura, Ran Zhang, and Patrick Baudisch. 2021. Trusscillator: a System for Fabricating Human-Scale Human-Powered Oscillating Devices. In *The 34th Annual ACM Symposium on User Interface Software and Technology (UIST '21), October 10–14, 2021, Virtual Event, USA.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3472749.3474807

1 INTRODUCTION

The related work in personal fabrication [3] offers numerous examples of so-called *kinematic systems* [29] that allow users to design and fabricate mechanisms that perform user-specified movement patterns. Examples include the 3D-printed pantograph from *Metamaterial Mechanisms* [17], the 5m-tall dinosaur from *TrussFormer* [23], and the animated cheetah created from *Computational Design of Mechanical Characters* [9] reproduced in Figure 2a.

In this paper, we want to extend this line of work towards machines that are *human-powered*, such as playground equipment, workout devices, and certain types of kinetic installations. "Human-powered" means that these devices need to be operated with the limited power that a human or, in some cases, a child can produce.

Unfortunately, when it comes to designing devices for which *limited power* plays a central role, the aforementioned systems for designing kinematic machines are of little help. Without support from a specialized software system, human-powered devices continue to be designed using time-consuming design cycles that iterate back-and-forth between guesswork and physical prototyping (see Section *4: "Expert interviews"*).

We present Trusscillator, a software system that enables users to create human-scale, *human-powered* machines, such as the playground equipment shown in Figure 1. Trusscillator achieves this by allowing users to add *springs* to their designs. As illustrated by Figure 2b, springs have the ability to transform movement (kinetic

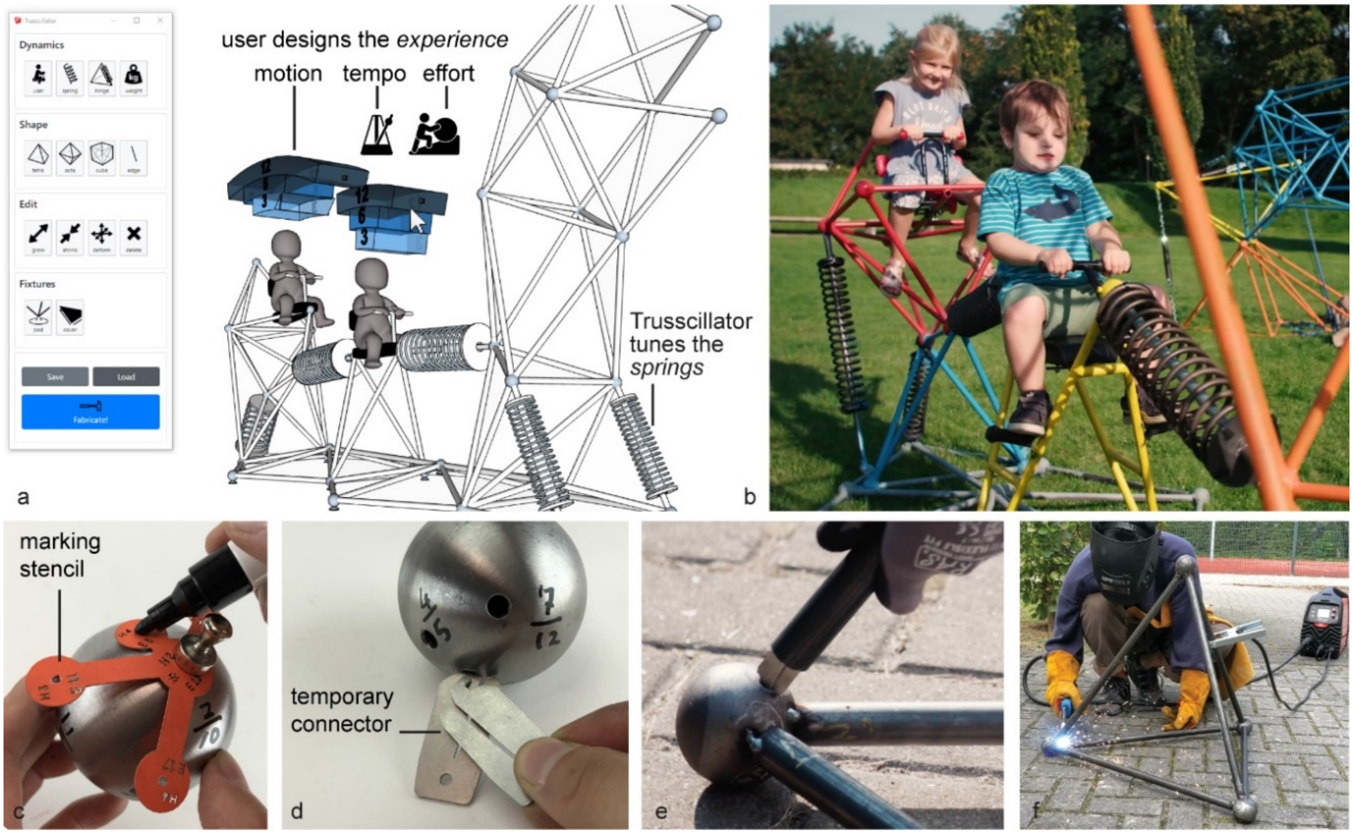

Figure 1: (a) Trusscillator is an integrated system that allows users to design human-scale, human-powered machines. Here designers are using it to design a dinosaur-inspired playground device. Trusscillator's user interface allows designers to interactively construct a steel truss structure, add coil springs, and specify the requirements in terms of motion range, speed, and physical effort. Trusscillator responds by adjusting the coil springs and adding mass so as to produce the desired behavior. (b) The resulting interactive dinosaur sculpture designed for two children challenges the riders to synchronize their movement to causes the sculpture's head to wiggle. (c) Given the scale of the involved forces, the structures created by Trusscillator are made from steel. Trusscillator supports steel truss fabrication by generating stencils that (d) show where to attach temporary connectors, (e) that hold steel rods in place, for (f) welding.

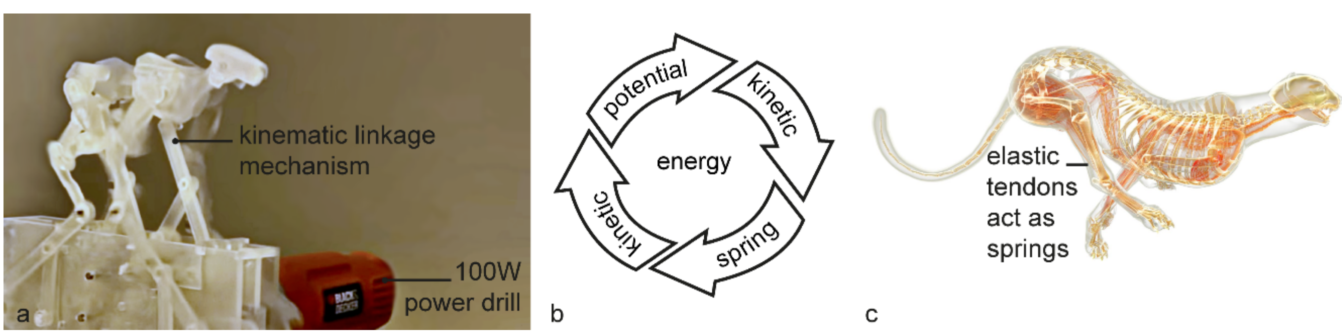

Figure 2: (a) The cheetah mechanism created using [9] is only resembling the movement pattern of a real one, without considering the forces involved during motion. (b) Energy conservation makes a real-life cheetah's[1] gallop efficient: (c) the elastic tendons store and release energy in every step.

energy) into compression (potential energy) and transform that back into movement. Consequently, springs help keeping these devices in motion with little effort and thus allow even larger machines to be *human-powered*. The resulting devices do not bear a lot of similarities with kinematic machines, such as the kinematic cheetah from Figure 2a, but instead bear more resemblance with an *actual* cheetah, which also uses springs (called *tendons*) to run efficiently

[1] https://www.dkfindout.com/us/animals-and-nature/cats/inside-cheetah

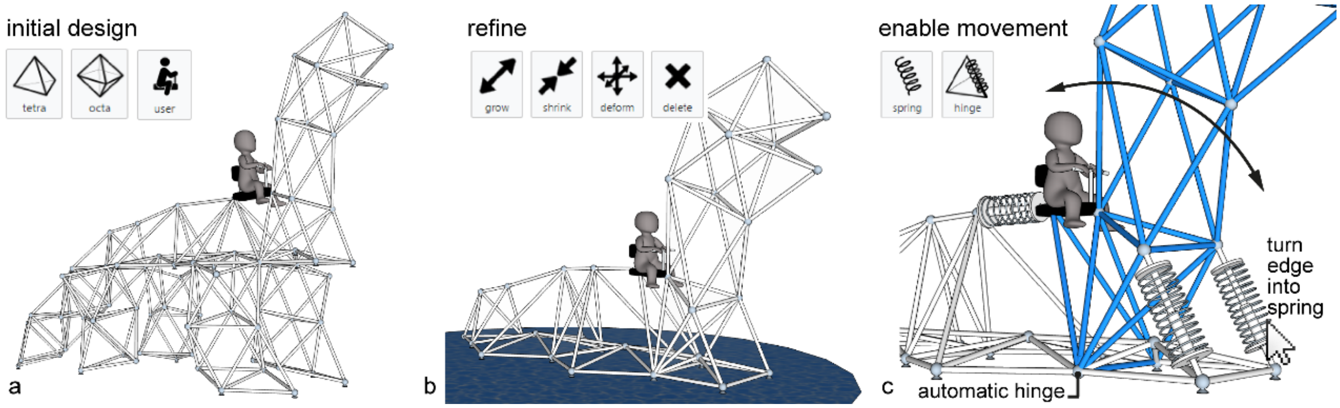

Figure 3: (a) The initial design of the brachiosaurus playground object is created using rigid truss primitives. (b) Designers adjust the shape and lower the height for safety reasons. (c) Using the spring tool, designers enable parts of the model to move. The newly created moving part of the model gets briefly highlighted in blue.

[30] (Figure 2c). These systems concerned with energy and motion are typically referred to as *dynamic systems* [33].

To allow designers to create human-powered movement, Trusscillator offers a novel set of tools, specifically designed for dynamic experiences (Figure 1a). These tools allow designers to focus on user experience-specific aspects, such as *motion range*, *tempo*, and *effort* while abstracting away the underlying technicalities of eigenfrequencies, spring constants, masses, and energy use. (c-f) Since the forces involved in the resulting devices can be high, devices designed using Trusscillator are made *from steel.* Trusscillator helps users to fabricate these devices not only by picking out appropriate springs but also by producing stencils and placing temporary connectors that help *welding* the resulting large-scale structures.

2 WALKTHROUGH

To demonstrate Trusscillator's workflow, we present a scenario in which two designers of playground equipment are designing the dinosaur-inspired device shown in Figure 1. The two designers, tasked to design a model for the playground associated with a natural history museum, are ideating around an interactive sculpture of a brachiosaurus.

2.1 A brachiosaurus swing for two

As shown in Figure 3a, the playground designers start by creating a rigid dinosaur sculpture by stacking truss-primitives, specifically tetrahedra, and octahedra (building on *TrussFab* [22]). They place a *ragdoll figure* onto the model, which inserts a matching seat for a child. (b) Given that Trusscillator will fabricate the model from steel, Trusscillator allows building models of any height. However, one of the designers is worried about safety issues resulting from the seat being located high up, so they place the dinosaur into "imaginary water", i.e., they remove its legs by delete truss elements.

As illustrated by Figure 3c, the two designers now turn the static structure into a very basic swing: they select the *spring tool* and use it to transform the three shown rods into *coil springs.* Trusscillator responds by placing hinges at the adequate points below the seat and acknowledges this by briefly highlighting the now movable part (in blue). The dinosaur's neck is not a hinging component and the sculpture has become a simple interactive device. A child can now bob back and forth, causing the dinosaur's neck to wiggle.

As illustrated by Figure 4a, Trusscillator displays the properties of this basic swing using what we call the *motion bar*: an average 6-year-old should be capable of making it rock roughly by the amplitude indicated by the middle curved blue bar labeled "6". Designers can play back a simulation of the child rocking by clicking on this bar.

Note that these properties are not coincidental: Trusscillator computed the swing the moment it was created and has picked a spring that is "just right", i.e., neither so soft as to that a 12-years-old could max out, nor so rigid as to that a 3-year-old would be unable to move it.

The designers decide to further fine-tune the experience. As discussed, the movement of a 12-year-old is ok per se (dark blue bar), but they are concerned that the dinosaur head would reach down far enough to hit someone. As shown in Figure 4b, the designers reduce the device's amplitude by grabbing the handle attached to the *motion bar* and drag it inwards. Trusscillator responds by re-running its optimization engine and replaces the springs in the model with springs that produce motion in the request range (Figure 4c).

The reduction in amplitude has now caused the ride to oscillate faster (0.6s period, indicated on hover). As shown in Figure 5a, Trusscillator considers this uncomfortable and displays a notification (in the shape of a metronome, together with the word "fast"). The designers click the notification to switch it to *comfortable.* Trusscillator responds by re-running its optimization to find a frequency in the range that is considered a pleasant rocking frequency (0.8-1.2s), which it achieves by making yet another adjustment to the springs, as well as by adding a weight to the head of the dinosaur as shown in Figure 5b.

At this point, designers notice a third concern: the *effort widget* suggests "*laborious"* (Figure 6a). This means the device requires more than 8 cycles to reach maximum amplitude, bearing the risk of children losing interest before getting it into full swing. One

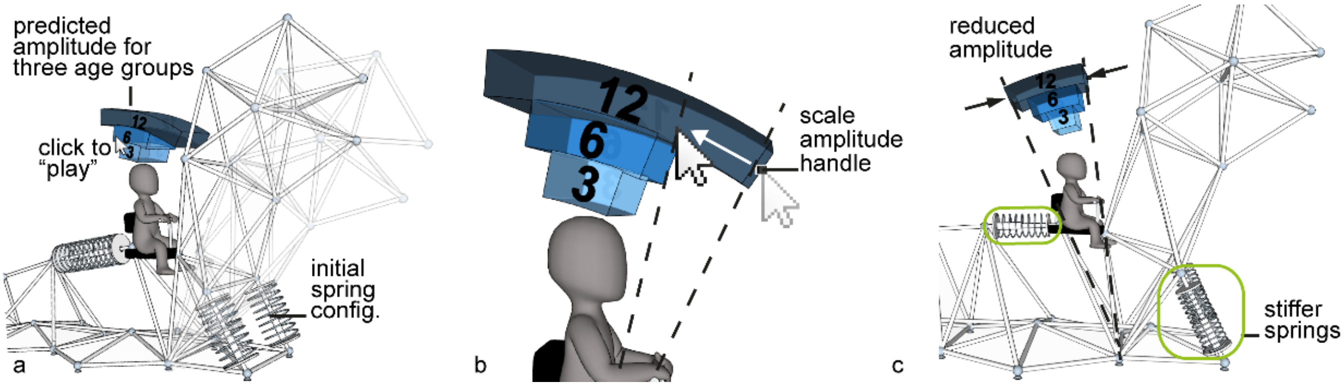

Figure 4: (a) Trusscillator initiates the model with a valid spring configuration. The resulting oscillating motion is summarized in form of a *motion-bar* above the user, calculated for multiple age groups. (b) When designers enlarge the motion space by dragging the *scale* handle, (c) Trusscillator finds a combination of softer springs that will produce the requested amplitude.

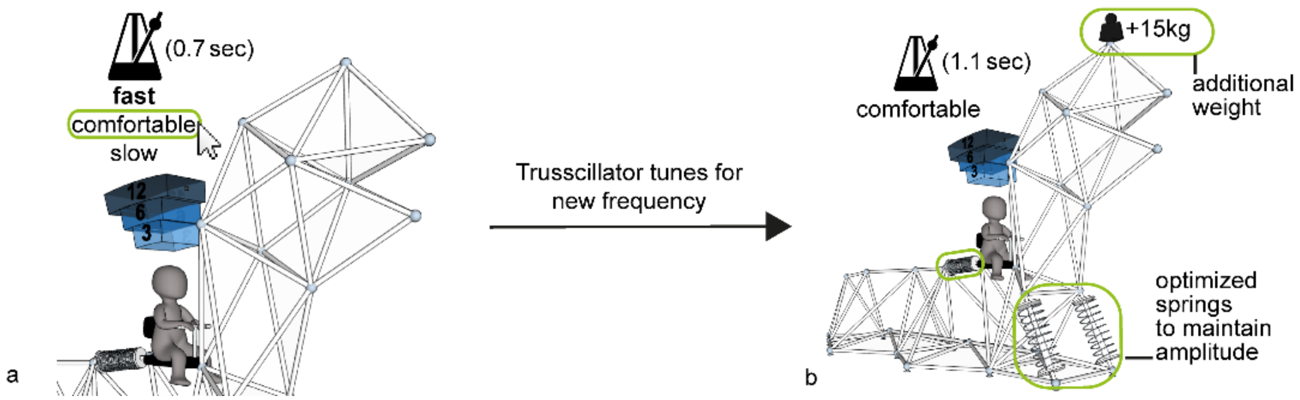

Figure 5: (a) When designers changing the tempo widget from *slow* to *comfortable* Trusscillator runs its optimization and (b) adds additional weight to the tip of the head to reduce the resonant frequency and tunes the springs again to maintain amplitude.

of the designers proposes clicking the effort widget to reduce the effort (see section 5.5), but the other designer sees the opportunity to add another level of excitement and challenge to the design by bringing in a second child. As illustrated by Figure 6b, they add a second seat and yet another spring .

This update changes the widget from l*aborious* to *just right* for both children, as they now both contribute power. More importantly, the resulting device has now created an additional challenge—a social challenge: First, it requires the first child to recruit another child as confederate to produce in order to successfully get the device to reach peak amplitude. Second, it requires the two children to synchronize their movement (or to decide to play against each other). Trusscillator allows for this by running its optimization procedure to tune the two seats to similar eigenfrequencies. To get a sense of what the resulting synchronization will feel like, the designers invoke simulations of the resulting movement (by clicking on the motion bars for each of the three age groups).

The designers are excited about this new perspective and move on to a physical prototype. They hit the *export and fabricate* button and proceed to fabricate their device.

2.2 Fabrication pipeline

Trusscillator now exports the designed structures for fabrication from steel rods, steel spheres, and steel springs, which users assemble using a power drill, an angle grinder, and an electric welding device.

The main challenge in assembling welded structures is to get all elements properly aligned prior to welding, as they cannot be adjusted anymore once a piece is welded. Trusscillator achieves this by supporting users in first creating a provisional assembly; only when everything is in place do users start to weld.

As a first step, Trusscillator producing a list with the lengths of required steel tubes, the number of steel balls to be purchased, and a list of the steel springs to be purchased (from a commercial spring catalog [13]).

Based on these elements, the fabrication process continues as illustrated by Figure 1c-f: (c) Trusscillator generates stencils for marking the connection points on the nodes-spheres. (d) Using the temporary connection system (e) users set up the provisional

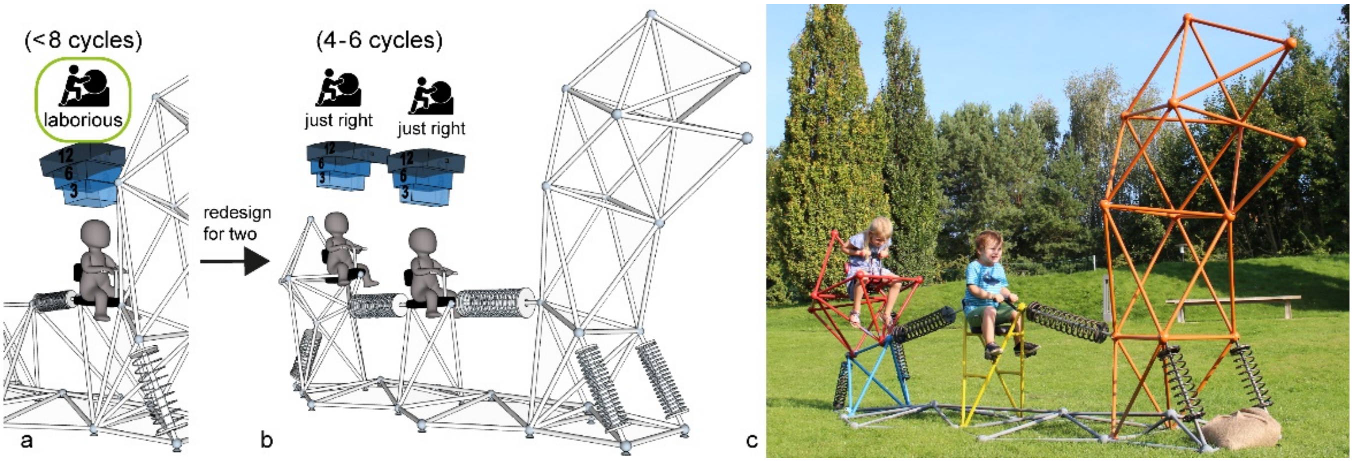

Figure 6: (a) Reducing the effort would require cutting on the weight of the structure, which designers can't do. (b) Instead, they add one more seating position. The final design comprises three spring-coupled inverted pendula, the head, middle seat, and tail. (c) Children induce resonance by synchronizing their motion.

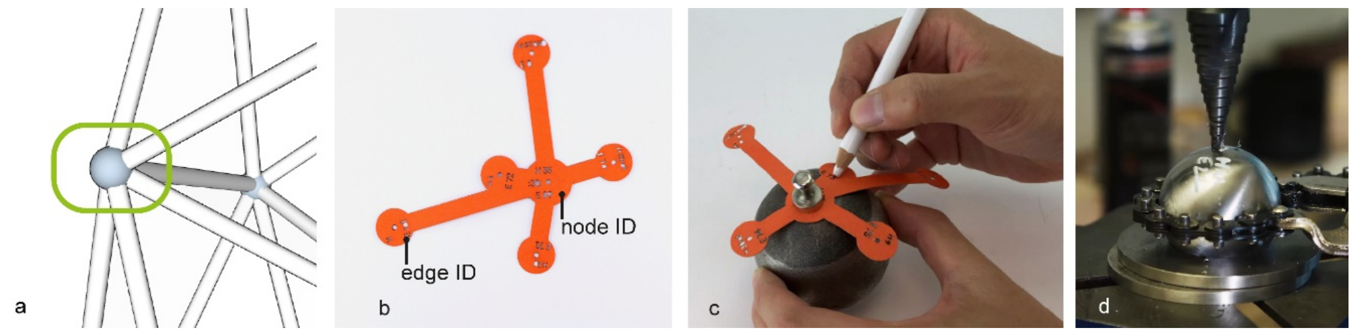

Figure 7: (a) Trusscillator exports this node in the 3D model (b) in the form of custom stencils. (c) Users mark one spot on the sphere, then attach the stencil at that point using a magnet, allowing them to mark the remaining incidence points. (d) Users then set up a stand-up drill with a round ring as a jig, and drill the spheres.

structure, and (f) finally weld the entire structure. Trusscillator supports this process as follows.

Trusscillator generates stencils as illustrated by Figure 7. (a) To minimizes the resulting gap between rod and sphere, thus maximizing the quality of the welded connections, Trusscillator helps users arrange rods and spheres so that rods hit spheres at a right angle. (b) To show users where on sphere connect with rods, Trusscillator generates custom stencils that mark the so-called *incidence point*s. Stencils form star-like shapes and Trusscillator exports them in SVG format. Users print and cut stencils manually using scissors or they send the SVG to a knife cutter or laser cutter. (c) Users attach a stencil to a sphere (using a magnet) and wrap the arms around the sphere so that each arm marks one incidence point. The stencils also help the assembly by displaying node IDs and edge IDs. Users transfer this information onto the spheres by marking the incidence points through small holes in the stencil. (d) Now users drill 6mm holes at the marked incidence points where the temporary connectors hook into.

Temporary connectors: Holding and welding the pieces in place is a challenging task, even for experienced welders. To overcome this difficulty, Trusscillator offers a system that helps pre-assemble the structure, allowing users to position all rods at the right places and at right angles with respect to the spheres before welding starts. For this purpose, we designed a thin metal connector piece that on one side hooks into the holes of the node-sphere, while its other side forms a cantilever spring that fits tightly into the metal tubes and resists slipping out, as shown in Figure 8a. For a secure connection, two of these metal pieces are inserted in every hole with opposite hook orientation, so none of them will be able to escape the hole when the tube holds them together (Figure 8b-c). This way they are holding the structure temporarily but firmly together for welding (Figure 8d). These connector pieces can be produced in a local metalworking shop using CNC machinery. They are considered as consumable material that stays locked inside the structure after welding.

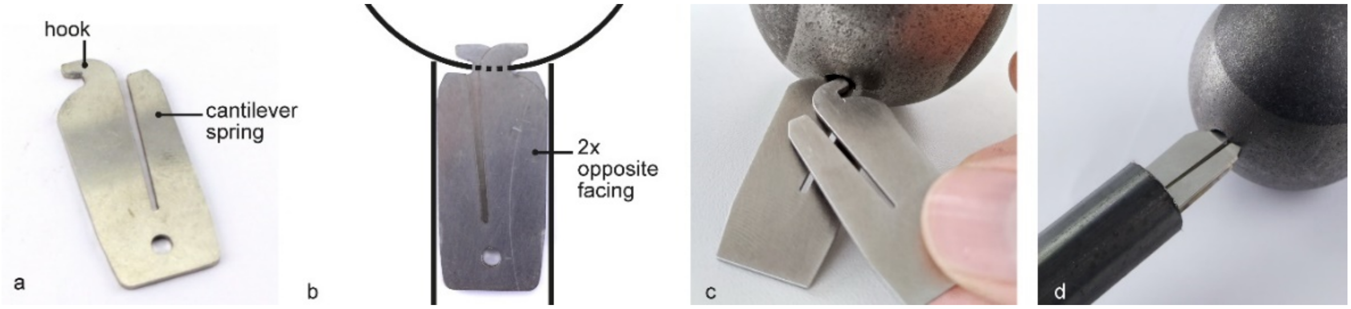

Figure 8: Trusscillator offers a temporary connector system to help position the edges for welding.

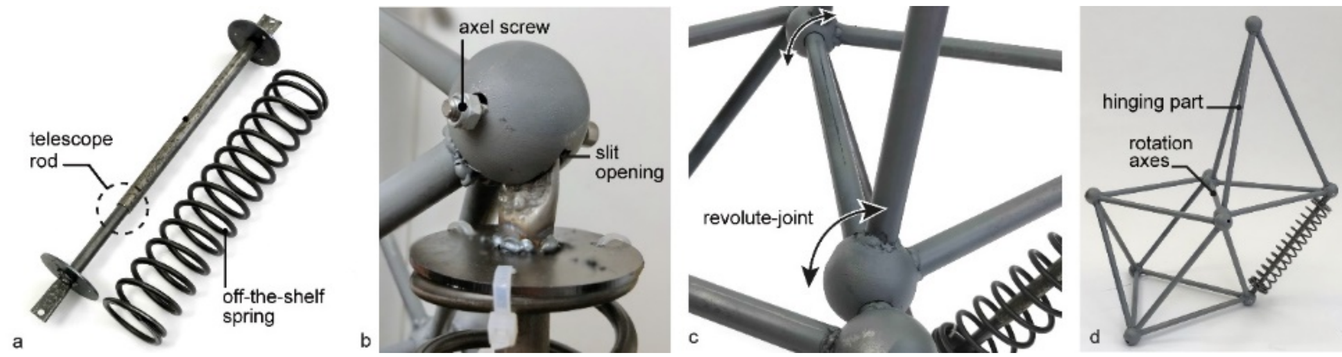

Figure 9: (a) Spring-telescope fabricated using two fitting tubes. (b) Slit opening on a sphere for inserting the telescope. (c) Revolute-joint connection. (d) Assembled chair model with a springy backrest.

This workflow of creating drilling stencils and using custom temporary connectors is our contribution to ease the otherwise hard to weld truss structures.

Spring telescopes and revolute hinges: To embed the off-the-shelf springs into the structure, users now create simple telescope elements by fitting two matching tubes into each other, as shown in Figure 9a. The metal discs at the two ends encompass the springs and prevent their buckling. These discs are then welded on the rods at a predefined position, to hold the spring in the right position.

As illustrated by Figure 9b, users mount spring telescopes into the structure by cutting a slit into a steel sphere. The corresponding holes for the axle-screw are also contained by the stencils.

As illustrated by Figure 9c, users now create revolute-joints by drilling large holes into the node-spheres where a tube can pierce through and form an axle. To fit two hinging parts together Trusscillator slightly insets the nodes of one part (here the backrest of the chair), so they can fit between the two outer nodes of the structure. Figure 9d shows the finished assembly of a chair model with a springy backrest.

We note that for safety reasons the motion range of the telescopes have to be constrained to prevent the structure from over-actuation, for example by adults. This can be achieved by adding mechanical stoppers, such as rubberized bumpers, or strings that prevent larger than expected motion (e.g., the blue straps in Figure 11), however this feature is currently not automated by the software.

2.3 Design space

We have used Trusscillator to design a wide range of devices. The samples are shown in Figure 10 including swings featuring 1D (b, e, j, m), and 2D motion (a, c, f, g), as well as kinetic installations (h, k) and balancing workout equipment (i).

While some of the devices feature collinear/coplanar spring arrangements (such as the brachiosaurus from our walkthrough), others create 2D motion paths, such as the "bird swing" shown in Figure 11

We created most of these models following the workflow we presented in the walkthrough section, i.e., we started by making a static shape and then added movement later ("shape-driven" design). However, other designs we created using a workflow that starts out with an already moving structure. As illustrated by Figure 12, Trusscillator supports this by offering predefined moving elements, such as a hinged tetrahedron.

3 CONTRIBUTION, BENEFITS, AND LIMITATIONS

Our main contribution is an end-to-end system that allows non-engineers to create human-scale human-powered devices that perform oscillatory movements, such as playground equipment, workout devices, and interactive kinetic installations.

Trusscillator consists of a custom software system that allows users to design trusses and add movement in the form of coil springs and hinges, as well as a series of novel hardware tools that support

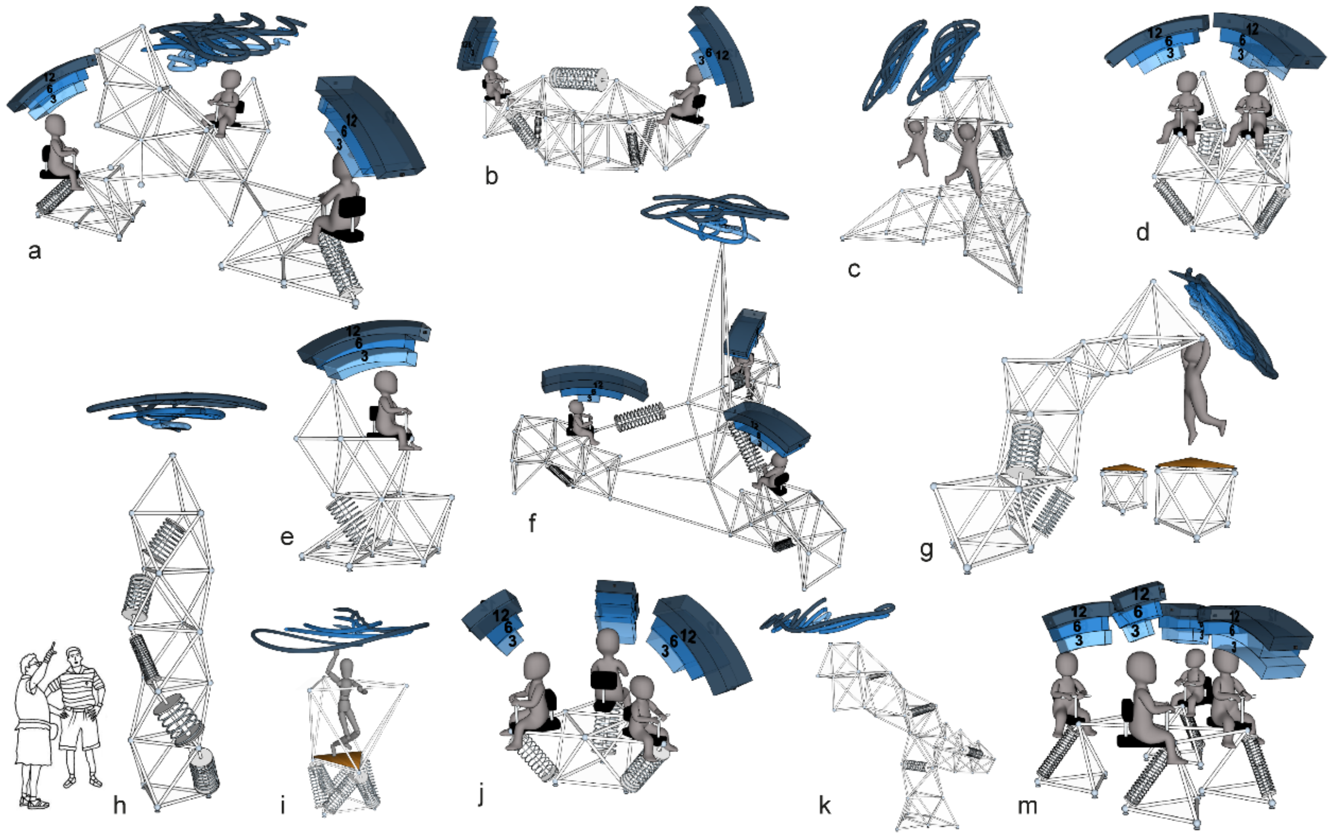

Figure 10: Some of the designs we created using Trusscillator.

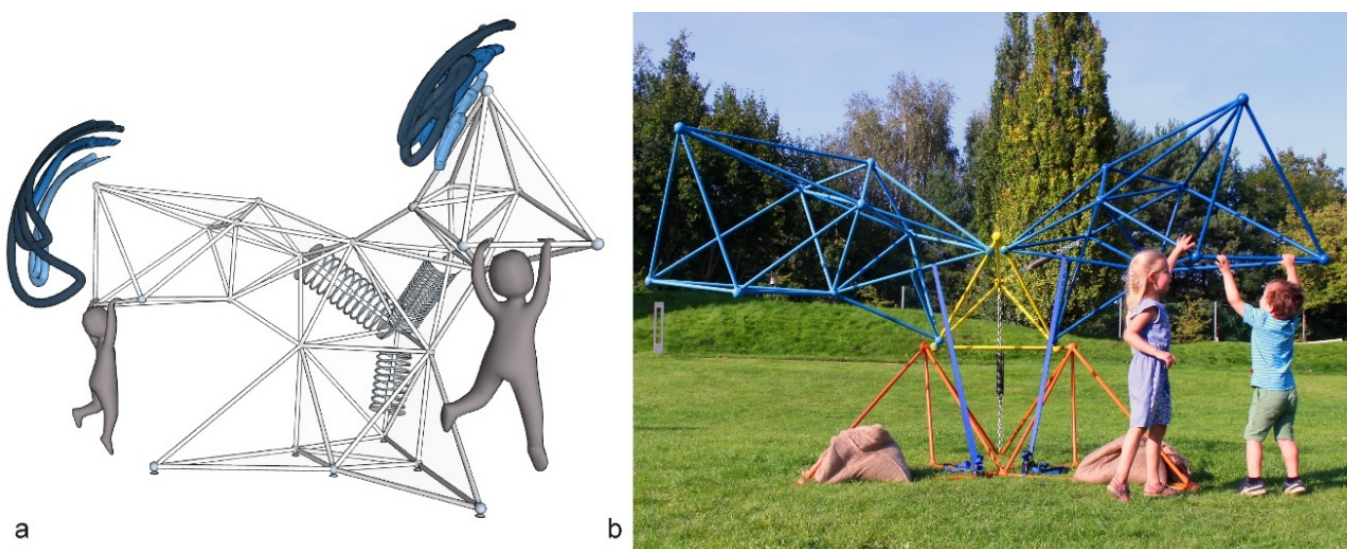

Figure 11: (a) This "bird-swing" structure was designed to allow children to swing in two-dimensional space and also to be able to influence each other's experience. (b) The physically built prototype in action.

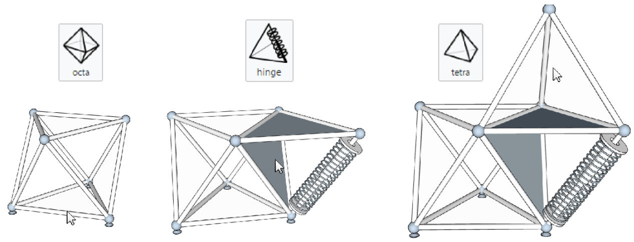

Figure 12: Building a model based on primitives containing springs speeds-up the design process. Here, the chair model is constructed using a tetrahedron with one spring and two hinges in only three steps.

the fabrication of the resulting steel structures, such as the drilling stencils and a temporary connector system that supports welding.

Trusscillator allows designers to consider not only the shape of a model, but also the *experience* it aims to produce, such as the right amplitude, an enjoyable oscillation frequency, and the effort it requires to be set in motion.

We identified the basic requirements for our software by interviewing professional playground designers, and we have validated our system by (1) designing 15 novel pieces of playground equipment, workout devices, and interactive kinetic installations, two of which we manufactured end-to-end, and by (2) conducting a technical evaluation of the technical aspects (simulation times and accuracy) of our approach.

Before devices designed using Trusscillator can be deployed, additional safety checks, such as height, size of triangles, safety stoppers, covering exposed springs, etc. need to be considered, according to the applicable regulatory requirements, such as DIN EN 1176 [10].

4 EXPERT INTERVIEWS

Before we started designing Trusscillator, we conducted semi-structured interviews with 3 professionals playground designers (P1-P3, all male between 40-55 years) recruited through purposive sampling. They had 20, 6, and 12 years of field experience respectively in a publicly listed company. Our objective was to learn about the opportunities and challenges that playground designers face, so we could address these using Trusscillator.

Before the interview session, we briefed the participants on the concept that we were interested in and the general workflows we wanted to support. Questions for the interview included the existing design workflows that the participants followed, in particular, their strategies of ensuring the users' safety, engagement, and tailoring their solutions to fit the needs of specific age groups. The interviews lasted between 90-120 minutes. All the interviews were audio-recorded with the participants' informed consent. We analyzed the interview transcripts using thematic analysis.

All three participants started by explaining their current workflow. They design using conventional CAD software (*Revit*, *SketchUp*, *Fusion360*), after which they validated and adjusted their designs against various safety standards and fabrication requirements. All three participants pointed out the *absence* of tools that would support the design of an experience.

P2 explained: "When creating equipment based on springs, we choose from a small ballpark of well-tested [very stiff] springs. We just assume that they'll work OK when we try it out. In case [they do] not, then we need to order a new set of springs. As a result, many of the spring-based toys at playgrounds are very hard to move, i.e., very restricted in their motion".

P1 gave us insights about the standards and norms that need to be taken into account. He also explained that different age groups fall into different safety categories. However, all equipment has to be designed safe for all age groups: "We like to create exciting toys. Having a certain level of danger is not inherently bad, as long as [the children] are made aware of that danger by design. This is how they learn to assess risk."

P3 saw potential in enabling a do-it-yourself approach: "Such tools could enable developing countries to build cheap playgrounds, that are not only fun, but the software could ensure that safety standards are also satisfied."

Our key insight was that current design tools tend to focus the on appearance, safety, and fabrication-related aspects. In contrast, participants expressed their desire to support not just the necessary technicalities in the design, but for designing the *experience* as well. This formed the basis for our main objective for the design of the Trusscillator system.

After the first development phase, we did follow up with the participants to show them the resulting software in the form of a video presentation. They were very excited about the result and expressed their appreciation for pushing forward this aspect of playground prototyping, that was non-existent before.

5 ALGORITHMS AND IMPLEMENTATION

The Trusscillator system is implemented in the form of three main modules: (1) interactive editor frontend, (2) simulation server, and the (3) exporter for fabrication. In order to allow our readers to replicate our results, we reproduce the underlying implementation and algorithms as follows.

5.1 Interactive editor frontend

Trusscillator builds on the editor components of *TrussFab* [22] and *TrussFormer* [23], which provide the core functionality to create, save, load, and export static and kinematic structures. Both the editors as well as, Trusscillator's frontend as well, are implemented as a plugin for *Sketchup Version 17* using the *Ruby* programming language.

In particular, Trusscillator's frontend extends Sketchup with UI elements that specifically refer to oscillating devices: (1) the motion-bar that users can drag to scale the motion range or click to play back the corresponding simulation sequence, (2) the tempo and effort widgets, and (3) the tools that add springs and hinges to the design.

To assist the users in placing the springs at the appropriate position, the Trusscillator frontend allows invoking a rigidity detection, which we implemented based on Zhang et. al. [46]. Using this approach, Trusscillator informs users whenever a new moving part has been enabled, or warns users when a placed spring is rigidly confined.

While the frontend takes care of modeling tasks and user adjustments, the oscillation characteristics and spring solutions are provided by the simulation server.

5.2 Simulation server

We implemented the simulation server in the *Julia* [6] programming language combined with the packages *DifferentialEquations.jl* [35] and *NetworkDynamics.jl* [25]. The *Julia* language is geared towards numerical computing and aims to combine the execution speed of low-level programming languages with the expressiveness of high-level languages.

In the context of Trusscillator, we get three central advantages from this stack: (1) The abstraction of *Julia* and *DifferentialEquations.jl* enables us to choose from a large library of solvers and choose the best performance/accuracy trade-off. (2) With the Just-in-time-compilation capabilities of Julia we generate efficient machine code for every given model without the need of introducing a separate compilation step, as it would have been necessary for similar systems like Modelica [11].

Trusscillator simulates dynamics by formulating a continuous-time system of differential equations that represents the given structure. The system uses highly optimized solvers to obtain a time-domain solution of the motion. We prefer this approach over a discrete-time model (as commonly found in real-time physically-based simulations) since it allows us to use variable step solvers that can adjust their step size dynamically to ensure that the result stays within specified tolerances. Furthermore, differential equation solvers are more robust against instabilities, such as the ones caused by fast oscillations, and better suited for modeling systems where maintaining energy conservation constraints plays a crucial role.

Using this approach, we have implemented a custom simulation package that can simulate the dynamics of arbitrary spring-damper-rod networks.

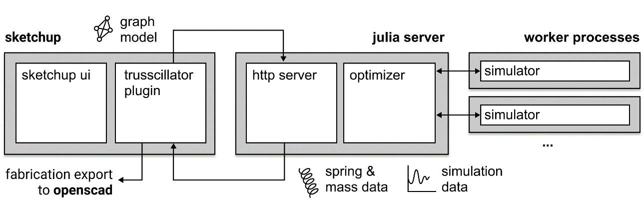

Figure 13: Trusscillator's high-level architecture.

As illustrated by Figure 13, Trusscillator's simulator and optimizer package runs as a stand-alone server and communicates via HTTP with the UI and the Sketchup Plugin. Sketchup transfers the model, encoded as a JSON string, to the simulation server. It contains the graph representation of the structure, including the lengths, spring and user positions, and the state of the requested behavior. For running a simulation, the server derives a system of equation from this structure by mapping the input graph structure onto simulation components, such that the entire model can be expressed in the following form: $\frac{du}{dt} = f(u, p, t)$, where u is the state vector of the system, p is the parameter vector, and t the time, as follows from [35]. This representation treats all the nodes essentially as ball-joint connections with point masses. For any arbitrary structure, the state of the system is uniquely defined by the positions and velocity of individual nodes.

With NetworkDynamics.jl, we provide a graph structure and specify the respective functions for every component separately, serving as a lightweight layer that separates concerns. Here, we specify four components: nodes, spring-dampers, rigid edges, and fixtures. These components are mapped 1:1 from the model created in the editor.

Node component is assigned to every node and together they define the state of the structure. They compute their movement from the forces of adjacent edges, their mass, and their actuation. Every node has a state vector that contributes to the global system state. It is defined by $u = [r_x, r_y, r_z, v_x, v_y, v_z]$, where $\vec{r}$ is the 3D-displacement vector and $\vec{v}$ is the velocity vector.

According to the formula above, we need to provide a function that returns the derivative of the state vector u, given any state vector (for reference, the derivative of displacement yields velocity, and the derivative of velocity yields acceleration). Computing the velocities is trivial, as they are already part of the function's input vector u. For obtaining the accelerations, we evaluate the term

$$\vec{a} = \frac{\sum_{edge \in E} \overrightarrow{F_{edge}}}{mass} + \frac{\overrightarrow{F_{act}}}{mass} + \overrightarrow{a_{gravity}}$$

where E is the set of the adjacent edges with their corresponding force vectors $\vec{F}_{edge}$ (see *rigid edge* components on how we obtain these values). To account for gravity, we also add a global gravitational acceleration force. Furthermore, we add an actuation force $\vec{F}_{act}$,in case the node has a ragdoll placed onto (see section *5.3*).

Thus, the result that we return back to the solver is $[v_x, v_y, v_z, a_x, a_y, a_z] = \frac{du}{dt}$.

Spring-damper components return the reaction force of a spring component, as given by Hooke's law and viscous-damping. They take the state vectors of the two nodes they connect and calculate a resulting force vector to both nodes as an output. We calculate the overall force by taking the sum of the spring force and damping: $F_{edge} = k \cdot (x - l) - d \cdot v$, where k is the spring constant, l is the uncompressed length of the spring, d is the damping coefficient, x is the distance between the two connecting nodes and v is the scalar velocity along the edge vector. The latter two are directly calculated from the connecting nodes' state vector. The resulting scalar is applied along the edge direction and presented as F_{edge} to the nodes.

Rigid edges are modeled as very stiff (essentially not movable) dampers, analogous to the damping term of the spring-damper component. They enforce a constant distance between the nodes.

Fixtures are anchor points of the structure, indicated by pods in the editor. From the perspective of the simulation, these simply expose a state vector with constant positions and without any velocity to the edges.

Finally, to run the simulation, we need to provide valid initial conditions i.e., a start assignment of the system's state vector to start the simulation (using the solver TRBDF2). For this, we obtain the positions of each node directly from the client and set all velocities to zero.

5.3 Simulating human actuation

By default, Trusscillator simulates the structure behavior for three age groups: 3, 6, and 12 years old (unless the user specifies otherwise). For approximating how children will interact with the structure, Trusscillator applies a periodic actuation force at the ragdoll's position. While an exact behavior would be hard to predict, Trusscillator assumes that the net power that a child exerts over time is roughly constant. Trusscillator assumes a 3-year-old to weigh 15 kg and output 30 Watts, a 6-year-old to weigh 25 kg and output 45 Watts, and a 12-year-old to weigh 40 kg and output 75 Watts, based on data from [34] and [19].

The actuation force is then applied in the direction of the actual velocity vector. To make sure that this force acts naturally on the system, respecting its natural frequency, we apply this force only during the acceleration phase of the movement. This behavior roughly mimics how humans push a swing back and forth. The value of this force is then calculated from the formula of power $F_{act} = \frac{P_{const}}{|\vec{v}|}$, to respect the constant net power input over time. To initialize the motion of the structure, Trusscillator simply applies a short push to set the structure in its natural oscillation.

5.4 Equilibrium instantiation

If the system would simply apply spring lengths from the catalog or use the edge length, the structure would immediately deform under its own weight and, therefore, deviate from the user's design intent. Trusscillator enables the creation of structures in their equilibrium positions without exposing its users to implementation details of uncompressed spring lengths or their static compression at rest. To achieve this abstraction, Trusscillator calculates, how much a spring needs to be pre-compressed, to ensure that they hold up the weight of the structure.

Trusscillator determines the level of pre-compression for static equilibrium by checking how the structure behaves without any adjustment. It runs a short-time simulation (e.g., 0.1s) and measures the resulting velocity along the spring vectors. Then it adjusts the springs' uncompressed lengths in proportion to this velocity to counter the initial movement. Trusscillator repeats this step until the process converges and the structure stops moving.

The resulting spring lengths are provided for the fabrication process, as well as, passed on to the simulation. Making the springs hold up the structure ensures that no unwanted initial potential energy gets introduced at the beginning of the simulation and actuates the structure beyond our model.

5.5 Trusscillator translates amplitude, frequency, and effort into mass, spring, damper configuration

The main objective behind Trusscillator is to allow users not only to design and build large-scale human-powered structures but also to help them to get the physical properties "right". The key idea here is to shield users from the underlying physics perspective (where devices are considered *mass-spring-damper* systems, see below) and to instead, let the users interact in user experience-related dimensions they are familiar with, i.e., range of motion (aka *amplitude*), frequency of the oscillation (aka *tempo*), and the time/energy required to swing up the device (aka *effort*), as illustrated in Figure 14a. For these input dimensions, Trusscillator determines spring constant and mass configuration to satisfy the user's design intent. The relationship between the mechanical properties and the experience attributes is illustrated in Figure 14b.

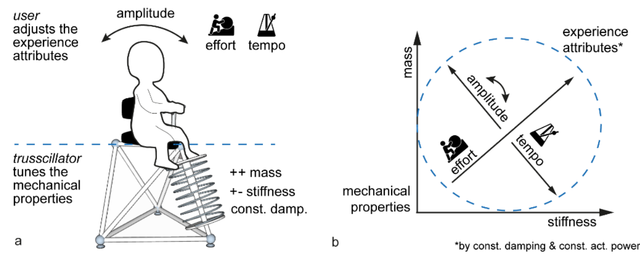

Figure 14: (a) The mechanical properties of the structure are defining the experience attributes. (b) The correlation between the mechanical properties and the motion experience (amplitude, tempo, and effort).

Trusscillator acquires the attributes of the oscillation by running a simulation sequence. During the simulation, the human-mimicking force starts to actuate the device and the amplitude is increasing as the energy is being accumulated in the system, as shown in Figure 15a. Consequently, the velocity of the movement also keeps increasing. However, proportionally to the velocity, viscous damping starts to increase ($F_{damp} = d \cdot v$), and this force is counteracting the movement. With the velocity increase, the damping action is dissipating more and more energy into heat; up until the point when the amplitude and velocity are so high that all the input energy of the user is being consumed by damping. The orange line in Figure 15b indicates this time point when the oscillating system has reached the energy equilibrium and the amplitude remains stable.

To exemplify this process, we take the simple bobbing saddle model from Figure 14a, fit with a catalog spring with the stiffness of $k = 3376$N/m, and damping $d = 50$ Ns/m, as shown in Figure 15a, and run the simulation for a 12-year-old user (40kg, 75W). Trusscillator then obtains the following information:

Amplitude: Trusscillator takes the largest amplitude from the simulated movement coordinates by finding out the maximum distance between any two points in the time-series for the node of interest. For the example above, it shows that the tip of the child's head will move about a 1m arc.

Effort: The time required to reach the energy equilibrium (ramp-up time) is what Trusscillator takes to estimate the effort required to swing up the device. Specifically, we take the amplitude measurements and compare at which point in time the occurrence of the largest amplitude drops below a 15% margin from the largest amplitude. The diagram in Figure 15c shows the velocity increase has stabilized after around 3.5s. Trusscillator interprets this effort as *easy* (up until 5s ramp-up time). From 5s to 10s it is considered *just-right* and above 10s is *laborious*, based on our observation of common swinging behavior. This information is then displayed in the *effort widget* to the user.

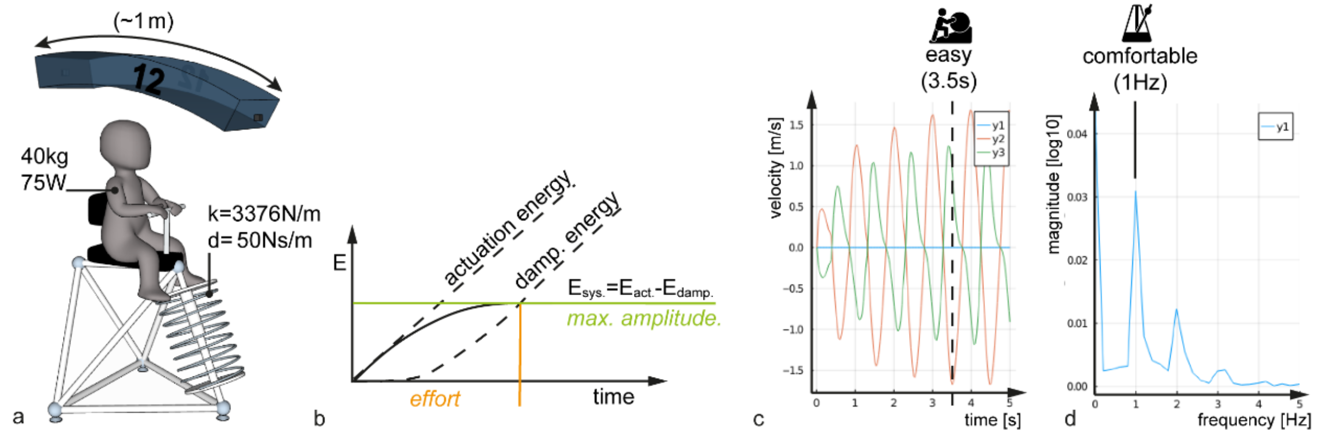

Figure 15: (a) Trusscillator simulates the model (b) until the time point when it reaches an energy equilibrium. (c) The time until the velocities don't increase anymore is considered for determining the effort. (d) The peak of the frequency spectrum determines the tempo metric.

Tempo/Frequency: Trusscillator analyzes this 3D velocity data from Figure 15c using Fast Fourier Transform (Figure 15d) and searches for the global maximum. In this example, the structure oscillates with the dominant frequency of 1Hz. This result is then classified as *comfortable* (0.5-1.5 Hz) based on input from [21]. Higher frequencies are classified as *shaky*, lower is *slow*. This information is displayed to the user in the *tempo widget*.

5.6 Optimization

To change the motion experience, Trusscillator has access to modify the two mechanical properties, namely mass (by adding weights to the structure) and stiffness (by choosing a spring from a catalog). We assume damping to be fixed as an inherent property of the material of the coil springs. This results in a challenging limitation for tuning the experience, where not all the criteria can be satisfied at all times. For this reason, Trusscillator utilizes a sampling-based optimization approach.

Figure 16 illustrates Trusscillator's optimization procedure, which is loosely inspired by the simulated annealing strategy. First, the algorithm searches for a viable baseline configuration. It assumes one global spring constant for all springs in the structure. It covers the range between 3kN/m and 20kN/m spring in intervals determined by the preset resolution (e.g., 10). After each simulation, we evaluate the simulation runs with the target metrics that we want to optimize and assign a distance to every sample using the distance function. We store the best (i.e., closest result) and proceed with optimizing the springs with a higher resolution one by one. We proceed analogously to the global sampling, only this time we don't consider the full spectrum of springs but only a window around the currently best assignment (e.g., ± 2kN/m), and every sample is being simulated with a range of additional masses. To avoid combinatorial explosion, we only place one mass in every local search step and place it at the highest point on one adjacent rigid group (heuristically assuming that this has the largest effect on the result). After every sampling round, we store the best parameter assignment and resume it for the next spring. After all the springs have been processed, we return the best matching parameter assignment of the last round.

This algorithm returns in O(n) sampling steps, where n is the number of springs, assuming that sufficient computing resources to run all simulations for a given sample in parallel are available. Parallelizing the simulations within one sampling round and reducing the dependencies of consecutive steps is key for reducing response times and enabling interactivity.

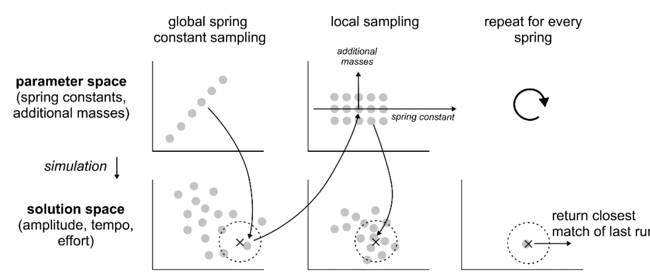

Figure 16: Spring optimization procedure.

For determining whether a design matches the expectation of what the user chooses, we define a distance metric that can be calculated from the simulation result: $\sum_{c \in C} 3 \cdot \Delta A_c + \Delta f_c + \Delta e_c + \sigma(F)$, where C is the set of children, and $\Delta A_c, \Delta f_c, \Delta e_c$ are the normalized differences of amplitude, frequency, and effort between target and measured data for the respective child. We emphasize the amplitude constraint with an additional weighting factor, as it is critical for the mechanical function of the structure. The last term ($\sigma(F)$), which denotes the statistical variance over the measured frequencies among the children. It incentivizes structureswhich are suitable for achieving resonance and therefore differences in frequencies are low.

The corresponding algorithm works as following:

For optimization, we only consider the oldest specified age group (here 12 years), as that age group exhibits the most extreme behavior, especially in terms of amplitude.

Algorithm 1 Spring optimization

```
best_parameter_vector = nothing
sampling_resolution = get_number_of_workers()
available_additional_masses = [0, 5, 15]
global_sampling = sample_all_springs(model,
range(1kN/m, 20kN/m, length=sampling_resolution))
best_parameter_vector =
select_best_guess(global_sampling)
for spring in springs
  spring_constant = get_spring_constant(spring,
best_parameter_vector)
  local_samples = sample_spring_and_masses(model,
spring, range(spring_constant - 2kN,
spring_constant + 2kN, length= sampling_resolution),
available_additional_masses)
  best_parameter_vector =
select_best_guess(local_samples)
end
return best_parameter_vector
```

Before returning the information back to the client, Trusscillator takes the closest matching springs from an online vendor catalog [13], configures the structure with that spring, and runs the simulation for all age groups.

5.7 Exporting stencils

Trusscillator renders the stencils using the parametric modeling tool *OpenSCAD* [28]. The key challenge behind this stencil design is that the longer an "arm" is, the larger the potential error caused by a user shearing the material while wrapping it around the sphere. We minimize this effect by choosing a star-like topology, where one incidence point acts as center based on which all other incidence points are being referred to. This prevents errors from propagating, as would be the case with designs that daisy-chain incidence points. Our algorithm picks the center point so as to minimize the distances to the other incidence points.

6 RELATED WORK

Trusscillator builds on previous work from the domains of mechanism design, springs and compliant mechanisms, dynamics-oriented systems in personal fabrication, and professional tools for physics simulation.

6.1 Software tools for mechanism design

Since the emergence of 3D printers, researchers in the HCI and computer graphics community have been looking into creating expert systems for helping everyday users in performing mechanical engineering tasks. One of these non-trivial engineering tasks is creating mechanisms, that have been researched in many flavors. *ChaCra* [31] is an interactive design system for rapid character crafting. Thomaszewski et. al. [41] looked into generating pleasing motion paths for animated kinematic characters. *Bend-it* [45] is a system for creating wire-bendable kinetic characters. *Roibot* [24] augments passive everyday objects by adding motorized actuation to them. Ion et. al. [17] proposed an interactive editor for creating mechanical metamaterial mechanisms. *TrussFab* [22] is an end-to-end system for creating large-scale static truss structures, while *TrussFormer* [23] also helps to animate these truss structures embedding linear actuators into them. All these tools are providing great help in automating specific engineering tasks of mechanism design; however, they concern very little about the energy consumption and dynamic properties of a mechanism.

Several software tools help the design of linkage-based mechanisms, such as *Mechanism Perfboard* [20], *LinkEdit* [5], or *LinkageDesigner* [26]. Some of these tools also allow users to explore certain dynamic aspects of the mechanisms, however, they are not (yet) suitable for simulating spring-based mechanisms.

6.2 Springs and compliant mechanisms

Springs, in their static and kinematic nature, have already been explored by the personal fabrication community. For example, *Ondulé* [15] helps novices to design parameterizable deformation behaviors in 3D-printable models using helical springs and embedded joints. Schumacher et. al. [38] have proposed a system for modifying the underlying microstructure of 3D printed objects in order to adjust their elasticity. Systems like [45] and [32] are focusing on compliant mechanisms that utilize the elasticity of the material to create motion. Roumen et. al. [37] have proposed *SpringFit*, a system for users of laser-cutters to make their models cross-device compatible by replacing the problematic press fit-based mounts and joints with cantilever-spring-based mounts and joints. Ion et. al. in [18] uses preloaded springs to mechanically transmit signals in digital metamaterials. Takahashi et. al. [40] have created a system for creating statically balanced planar spring mechanisms. The bistable nature of compliant mechanisms has been explored by Zhang et al. [46]. While all these works are focusing on springs and elastic behavior, they are mostly concerned about the shape, static balance, and static force the spring provides. Trusscillator expands these approaches to the dynamic domain and explores springs in motion.

6.3 Dynamics oriented systems in personal fabrication

Predicting the dynamic behavior of mechanisms has also been researched in the HCI and computer graphics community. Some interactive design tools also leverage physics simulation, such as *SketchChair* [38] and Umetani et. al. [43]. While the aforementioned examples are still mostly concerned about statics, other tools also help to explore the motion. For example, *Spin-it* [4] enables 3D printing arbitrary spinning tops by optimizing the internal rotational dynamic properties, while *Pteromys* [42] helps to optimize the aerodynamics of free-flight glider paper airplanes. Chang et. al. [7] have been developing haptic kirigami swatches that helps designing specialized springs that provide a well-defined resistance profile for haptic buttons and switches. Chen et. al. [8] proposed a system for accurate simulation of dynamic, elastic objects at interactive rates. Similarly, *Real2Sim* [14] is a system that estimates the material's visco-elastic parameters retrieved from dynamic motion data. Hoshyari et. al. [16] have created a workflow for reducing unwanted secondary oscillations in expressive robotic characters.

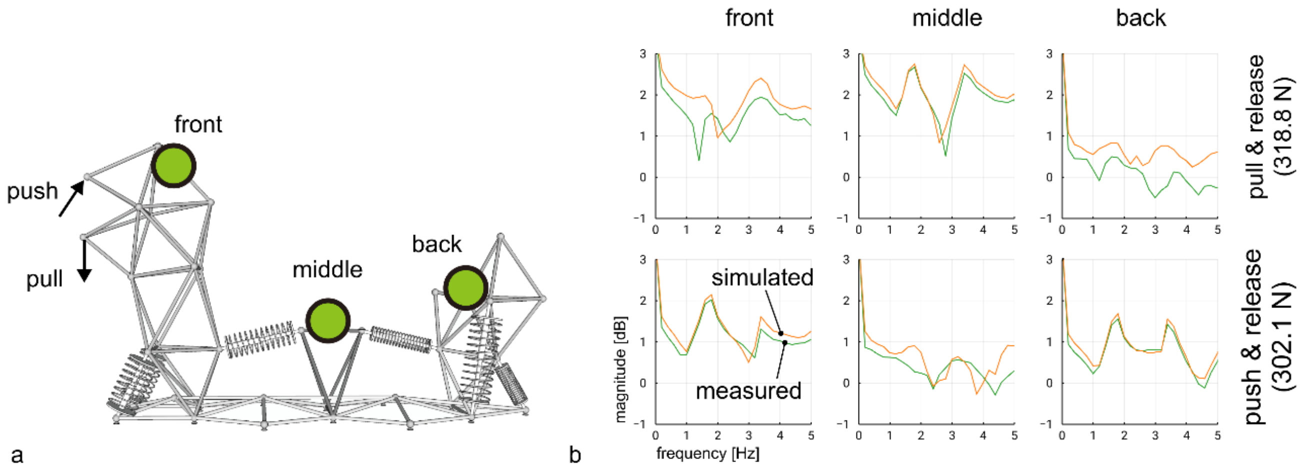

Figure 17: (a) The measurement points indicated on the real and virtual model. (b) Frequency response comparison of the push and pull experiments.

Tang et. al. [36] presented a harmonic balance approach for designing compliant mechanical systems with nonlinear periodic motions. All these projects are dealing with predicting dynamic motion and helping users in their design. Trusscillator extends this line of work to human-powered oscillating devices.

6.4 Professional tools for simulating dynamic physical systems

Physics simulation has become one of the most important enabling technologies for engineering physical artifacts. For example, commercial software like Fusion360 [2] readily offers finite element simulation capabilities for engineers. Some interactive editors utilize powerful frame-based simulation, such as *Algoryx Momentum* [1] or *Vortex Studio* [44]. These systems are great for real-time simulation of complex physical phenomena; however, repeatability and precision of the results is not always guarantied.

On the other hand, continuous-time cross-domain analytic solvers offer high accuracy and repeatability through a closed representation of the system. Examples of such systems are *Modelica* [11] and Mathworks' *Simscape* [27]. They are very powerful in simulating cross-domain physical processes; however, their use often requires a deep understanding of the simulated system and the actual language as well. Trusscillator bridges this gap by interfacing a custom analytic solver with a high-level UI tailored for designing spring-based oscillating mechanisms.

7 VALIDATION

To validate Trusscillator's functionality, we designed 15 models (Figure 10), including the two models that were fabricated physically, i.e., the "brachiosaurus" from Figure 1, the "bird swing" from Figure 11. Trusscillator allowed a team of two to design, cut, drill, assemble, weld, and paint each model in 2-3 days.

7.1 Simulation accuracy

We conducted a technical evaluation assessing the accuracy of Trusscillator's simulation, in which we compared the frequency response measured for our "brachiosaurus" device with the frequency response predicted by our simulation. We chose this evaluation to determine whether our simulation approach is suitable for representing the real world prototype across the entire frequency spectrum.

Figure 17 (left) shows the evaluation setup. Three IMU loggers (G-Sensor Logger [12]) were placed on the three moving parts of the dinosaur swing, recording 60 data points per second. We measure the "step response" of the mechanism in response to pushing the dinosaur head node upwards and then rapidly releasing it, as well as the response to pulling the "chin" downwards and releasing it. We also measured the peak force applied to the system using a SAUTER HP-5K digital force sensor and this same value was also applied in the simulation environment.

Results: Figure 17b shows frequency spectra measured and simulated. We applied FFT on the acceleration data obtained from the IMU on the real model (green line), and on the simulation data of the respective node (orange line). As shown in Figure 17b the simulation data resembles the real-world observations closely.

The slight differences between our demo model and the simulated data can be interpreted by the imprecision in fabrication, increased friction, and slack in the joints, that causes additional shocks and loss of energy. These parameters can be empirically adjusted and implemented in the software; however, they are highly dependent on the actual material used, fabrication quality, lubrication, etc. Another source of error are the simplifications that the simulation assumes, such as lumping of masses on the nodes or nonlinearities in the damping and spring forces.

Table 1: Simulation benchmark results

Model	# nodes	# edges	# springs	Simulation time	Optimization time
chair	8	18	1	74 ms	929 ms
bird-swing	26	76	3	797 ms	5544 ms
brachiosaurus	32	103	6	179 ms	7770 ms

7.2 Performance of the simulator

Simulating the oscillating behavior is the computationally most expensive component of Trusscillator's system. To validate that the system can provide interactive design iteration cycles even for complex models, we benchmarked the simulation steps on three models: a simple chair with one spring in its backrest (Figure 9c), the bird-swing (Figure 11), and the brachiosaurus (Figure 1).

We ran the simulation on a DELL XPS 15 9600 with Intel Core i7-10750H 2.6 GHz CPU (2020 edition) running on Ubuntu 20.04. The output of the simulation is a common query used in our editor: 30 fps for 5s, resulting in 150 frames. We computed response times by performing 10 consecutive runs and averaging response times.

As shown in Table 1, all the simulations run under 1 second—appropriate for a turn-taking interaction.

We note that execution speed is sensitive to multiple factors, such as, required accuracy, number of spring combinations, number of refinements, frequency of the movement, actuation power and more. This is the main reason why the optimization is currently slower than the simulation time multiplied by the spring count (slowest simulation governs the time for one sampling round). Note that the times reported here, are for a full optimization round, where consecutive user interaction could also be reduced to a subset of the springs and samples. We see further potential for speed ups by not simulating every node position individually, but combining rigid parts of the structure and simulating them as a single entity (detected by the rigid group detection algorithm mentioned in section 5.1).

8 CONCLUSION

We presented Trusscillator, an end-to-end system that enables novice users to design and build human-scale human-powered machines. As we learned in our expert interviews, such devices are usually subject to long design and prototyping cycles. Trusscillator speeds up this process by encapsulating large parts of the required domain knowledge from designing structurally stable mechanisms, through tuning and verifying their dynamic behavior, to building process and tools.

Zooming out, we think of Trusscillator as a tool that pushes research on large-scale personal fabrication in two ways. First, it goes the next logistical step from systems supporting static construction to kinematic construction to now dynamic construction. Second, it provides a computer-assisted system for the personal fabrication of welded steel structures, thereby laying the groundwork for scaling this line of research to bigger structures and larger forces.

As future work, we plan to introduce dampers into large-scale personal fabrication, allowing users to design large-scale mass-spring-damper systems.

ACKNOWLEDGMENTS

We thank Philippa, Oli, and Tisza for play testing our demo objects, Daniela Vogel for the design of the UI elements, and Hany Elhassany, Paul Methfessel, and Martin Taraz for their help with the welding works.

REFERENCES

[1] Algoryx Momentum. Retrieved April 3, 2021, from https://www.algoryx.se/momentum/
[2] Autodesk Fusion 360. Retrieved April 3, 2021, from https://www.autodesk.com/products/fusion-360
[3] Patrick Baudisch and Stefanie Mueller. 2017. Personal Fabrication. Foundations and Trends in Human–Computer Interaction 10, no. 3–4 (2017): 165-293.
[4] Moritz Bächer, Emily Whiting, Bernd Bickel, and Olga Sorkine-Hornung. 2014. Spin-it: optimizing moment of inertia for spinnable objects. ACM Trans. Graph. 33, 4, Article 96 (July 2014), 10 pages. DOI:https://doi.org/10.1145/2601097.2601157
[5] Moritz Bächer, Stelian Coros, and Bernhard Thomaszewski. 2015. LinkEdit: interactive linkage editing using symbolic kinematics. ACM Trans. Graph. 34, 4, Article 99 (August 2015), 8 pages. DOI:https://doi.org/10.1145/2766985
[6] Jeff Bezanson, Stefan Karpinski, Viral B. Shah, Alan Edelma. 2012. Julia: A fast dynamic language for technical computing." arXiv preprint arXiv:1209.5145.
[7] Zekun Chang, Tung D. Ta, Koya Narumi, Heeju Kim, Fuminori Okuya, Dongchi Li, Kunihiro Kato, Jie Qi, Yoshinobu Miyamoto, Kazuya Saito, and Yoshihiro Kawahara. 2020. Kirigami Haptic Swatches: Design Methods for Cut-and-Fold Haptic Feedback Mechanisms. In Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems (CHI '20). Association for Computing Machinery, New York, NY, USA, 1–12. DOI:https://doi.org/10.1145/3313831.3376655
[8] Desai Chen, David I. W. Levin, Wojciech Matusik, and Danny M. Kaufman. 2017. Dynamics-aware numerical coarsening for fabrication design. ACM Trans. Graph. 36, 4, Article 84 (July 2017), 15 pages. DOI:https://doi.org/10.1145/3072959.3073669
[9] Stelian Coros, Bernhard Thomaszewski, Gioacchino Noris, Shinjiro Sueda, Moira Forberg, Robert W. Sumner, Wojciech Matusik, and Bernd Bickel. 2013. Computational design of mechanical characters. ACM Trans. Graph. 32, 4, Article 83 (July 2013), 12 pages. DOI:https://doi.org/10.1145/2461912.2461953
[10] DIN EN 1176 - Safety requirements and test methods for playground equipment. Retrieved April 3, 2021, from https://www.din.de/de/meta/suche/62730!search?query=DIN+EN+1176&submit-btn=Submit
[11] Peter Fritzson and Vadim Engelson. 1998. Modelica—A unified object-oriented language for system modeling and simulation. In European Conference on Object-Oriented Programming, pp. 67-90. Springer, Berlin, Heidelberg.
[12] G-Sensor Logger. Retrieved April 3, 2021, from https://play.google.com/store/apps/details?id=com.peterhohsy.gsensor_debug&hl=en
[13] Gutekunst Federn. Retrieved April 3, 2021, from https://www.federnshop.com/de/produkte/druckfedern.html
[14] David Hahn, Pol Banzet, James M. Bern, and Stelian Coros. 2019. Real2Sim: visco-elastic parameter estimation from dynamic motion. ACM Trans. Graph. 38, 6, Article 236 (November 2019), 13 pages. DOI:https://doi.org/10.1145/3355089.3356548
[15] Liang He, Huaishu Peng, Michelle Lin, Ravikanth Konjeti, François Guimbretière, and Jon E. Froehlich. 2019. Ondulé: Designing and Controlling 3D Printable Springs. In Proceedings of the 32nd Annual ACM Symposium on User Interface Software and Technology (UIST '19). Association for Computing Machinery, New York, NY, USA, 739–750. DOI:https://doi.org/10.1145/3332165.3347951
[16] Shayan Hoshyari, Hongyi Xu, Espen Knoop, Stelian Coros, and Moritz Bächer. 2019. Vibration-minimizing motion retargeting for robotic characters. ACM Trans. Graph. 38, 4, Article 102 (July 2019), 14 pages. DOI:https://doi.org/10.1145/3306346.3323034
[17] Alexandra Ion, Johannes Frohnhofen, Ludwig Wall, Robert Kovacs, Mirela Alistar, Jack Lindsay, Pedro Lopes, Hsiang-Ting Chen, and Patrick Baudisch. 2016. Metamaterial Mechanisms. In Proceedings of the 29th Annual Symposium on User Interface Software and Technology (UIST '16). Association for Computing Machinery, New York, NY, USA, 529–539. DOI:https://doi.org/10.1145/2984511.2984540

[18] Alexandra Ion, Ludwig Wall, Robert Kovacs, and Patrick Baudisch. 2017. Digital Mechanical Metamaterials. In Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems (CHI '17). Association for Computing Machinery, New York, NY, USA, 977–988. DOI:https://doi.org/10.1145/3025453.3025624

[19] Martin, J. C., R. P. Farrar, B. M. Wagner, and W. W. Spirduso. 2000. Maximal power across the lifespan. The Journals of Gerontology Series A: Biological Sciences and Medical Sciences 55, no. 6 (2000): M311-M316.

[20] Yunwoo Jeong, Han-Jong Kim, and Tek-Jin Nam. 2018. Mechanism Perfboard: An Augmented Reality Environment for Linkage Mechanism Design and Fabrication. In Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems (CHI '18). ACM, New York, NY, USA, DOI: https://doi.org/10.1145/3173574.3173985

[21] Takeshi Kawashima. 2015. 101 Basic study on comfortable fluctuation: Discussions about the period fluctuations of rhythms produced by humans for their own enjoyment. The Proceedings of the Symposium on Environmental Engineering. 2015.25. 10-13. 10.1299/jsmeenv.2015.25.10.

[22] Robert Kovacs, Anna Seufert, Ludwig Wall, Hsiang-Ting Chen, Florian Meinel, Willi Müller, Sijing You, Maximilian Brehm, Jonathan Striebel, Yannis Kommana, Alexander Popiak, Thomas Bläsius, and Patrick Baudisch. 2017. TrussFab: Fabricating Sturdy Large-Scale Structures on Desktop 3D Printers. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems* (CHI '17). ACM, New York, NY, USA, 2606-2616. DOI: https://doi.org/10.1145/3025453.3026016

[23] Robert Kovacs, Alexandra Ion, Pedro Lopes, Tim Oesterreich, Johannes Filter, Philipp Otto, Tobias Arndt, Nico Ring, Melvin Witte, Anton Synytsia, and Patrick Baudisch. 2018. TrussFormer: 3D Printing Large Kinetic Structures. In *Proceedings of the 31st Annual ACM Symposium on User Interface Software and Technology* (UIST '18). Association for Computing Machinery, New York, NY, USA, 113–125.DOI: https://doi.org/10.1145/3242587.3242607

[24] Jiahao Li, Jeeeun Kim, and Xiang 'Anthony' Chen. 2019. Robiot: A Design Tool for Actuating Everyday Objects with Automatically Generated 3D Printable Mechanisms. In *Proceedings of the 32nd Annual ACM Symposium on User Interface Software and Technology* (UIST '19). Association for Computing Machinery, New York, NY, USA, 673–685. DOI:https://doi.org/10.1145/3332165.3347894

[25] Michael Lindner, Lucas Lincoln, Fenja Drauschke, Julia Monika Koulen, Hans Würfel, Anton Plietzsch, Frank Hellmann. 2021. NetworkDynamics.jl – Composing and simulating complex networks in Julia. arXiv preprint arXiv:2012.12696.

[26] Linkage Designer. Gábor Erdős. Retrieved April 3, 2021, from http://www.linkagedesigner.com/

[27] Mathworks Simscape. Retrieved April 3, 2021, from https://www.mathworks.com/products/simscape.html

[28] OpenSCAD parametric solid modeling software. Retrieved April 3, 2021, from https://www.openscad.org/

[29] Asok Kumar Mallik, Amitabha Gosh, Günter Dittrich. 1994. Kinematic analysis and synthesis of mechanisms. CRC Press.

[30] Alexander R. McN. 1989. Elastic mechanisms in the locomotion of vertebrates. Netherlands Journal of Zoology 40, no. 1-2(1989): 93-105.

[31] Vittorio Megaro, Bernhard Thomaszewski, Damien Gauge, Eitan Grinspun, Stelian Coros, and Markus Gross. 2015. ChaCra: an interactive design system for rapid character crafting. In Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA '14). Eurographics Association, Goslar, DEU, 123–130.

[32] Vittorio Megaro, Jonas Zehnder, Moritz Bächer, Stelian Coros, Markus Gross, and Bernhard Thomaszewski. 2017. A computational design tool for compliant mechanisms. ACM Trans. Graph. 36, 4, Article 82 (July 2017), 12 pages. DOI:https://doi.org/10.1145/3072959.3073636

[33] James L. Meriam and L. Glenn Kraige. 2012. Engineering mechanics: dynamics. Vol. 2. John Wiley & Sons.

[34] Mercedes de Onis, Adelheid W. Onyango, Elaine Borghi, Amani Siyam, Chizuru Nishida, and Jonathan Siekmann. 2007. Development of a WHO growth reference for school-aged children and adolescents. Bulletin of the World health Organization 85 (2007): 660-667.

[35] Christopher Rackauckas and Qing Nie. 2017. Differentialequations.jl–a performant and feature-rich ecosystem for solving differential equations in julia." Journal of Open Research Software 5, no. 1 (2017).

[36] Pengbin Tang, Jonas Zehnder, Stelian Coros, and Bernhard Thomaszewski. 2020. A harmonic balance approach for designing compliant mechanical systems with nonlinear periodic motions. *ACM Trans. Graph.* 39, 6, Article 191 (December 2020), 14 pages. DOI:https://doi.org/10.1145/3414685.3417765

[37] Thijs Roumen, Jotaro Shigeyama, Julius Cosmo Romeo Rudolph, Felix Grzelka, and Patrick Baudisch. 2019. SpringFit: Joints and Mounts that Fabricate on Any Laser Cutter. In Proceedings of the 32nd Annual ACM Symposium on User Interface Software and Technology (UIST '19). Association for Computing Machinery, New York, NY, USA, 727–738. DOI:https://doi.org/10.1145/3332165.3347930

[38] Greg Saul, Manfred Lau, Jun Mitani, and Takeo Igarashi. 2010. SketchChair: an all-in-one chair design system for end users. In Proceedings of the fifth international conference on Tangible, embedded, and embodied interaction (TEI '11). Association for Computing Machinery, New York, NY, USA, 73–80. DOI:https://doi.org/10.1145/1935701.1935717

[39] Christian Schumacher, Bernd Bickel, Jan Rys, Steve Marschner, Chiara Daraio, and Markus Gross. 2015. Microstructures to control elasticity in 3D printing. ACM Trans. Graph. 34, 4, Article 136 (August 2015), 13 pages. DOI:https://doi.org/10.1145/2766926

[40] Takuto Takahashi, Jonas Zehnder, Hiroshi G. Okuno, Shigeki Sugano, Stelian Coros, and Bernhard Thomaszewski. "Computational Design of Statically Balanced Planar Spring Mechanisms." IEEE Robotics and Automation Letters 4, no. 4 (2019): 4438-4444.

[41] Bernhard Thomaszewski, Stelian Coros, Damien Gauge, Vittorio Megaro, Eitan Grinspun, and Markus Gross. 2014. Computational design of linkage-based characters. ACM Trans. Graph. 33, 4, Article 64 (July 2014), 9 pages. DOI:https://doi.org/10.1145/2601097.2601143

[42] Nobuyuki Umetani, Yuki Koyama, Ryan Schmidt, and Takeo Igarashi. 2014. Pteromys: interactive design and optimization of free-formed free-flight model airplanes. ACM Trans. Graph. 33, 4, Article 65 (July 2014), 10 pages. DOI:https://doi.org/10.1145/2601097.2601129

[43] Nobuyuki Umetani, Takeo Igarashi, and Niloy J. Mitra. "Guided exploration of physically valid shapes for furniture design." ACM Trans. Graph. 31, no. 4 (2012): 86-1.

[44] Vortex Studio. CM Labs. Retrieved April 3, 2021, from https://www.cm-labs.com/vortex-studio

[45] Hongyi Xu, Espen Knoop, Stelian Coros, and Moritz Bächer. 2018. Bend-it: design and fabrication of kinetic wire characters. ACM Trans. Graph. 37, 6, Article 239 (November 2018), 15 pages. DOI:https://doi.org/10.1145/3272127.3275089

[46] Ran Zhang, Thomas Auzinger, and Bernd Bickel. 2021. Computational Design of Planar Multistable Compliant Structures. ACM Trans. Graph. 1, 1, Article 1 (January 2021), 16 pages.