

HUMAN-SCALE PERSONAL FABRICATION

Róbert Kovács

a dissertation submitted in partial fulfillment of the requirements for
the degree of

Doctor of Engineering (Dr. -Ing.)



Hasso Plattner Institute – Digital Engineering Faculty
University of Potsdam

2022

Unless otherwise indicated, this work is licensed under a Creative Commons License Attribution – NonCommercial 4.0 International. This does not apply to quoted content and works based on other permissions. To view a copy of this license visit:
<https://creativecommons.org/licenses/by-nc/4.0>

ADVISOR

Prof. Dr. Patrick Baudisch (Hasso Plattner Institute)

REVIEWERS

Prof. Dr. Emily Whiting (Boston University)

Prof. Dr. Jürgen Steimle (Saarland University)

MEMBERS OF THE COMMITTEE

Prof. Dr. Andreas Polze (Hasso Plattner Institute, secondary advisor)

Prof. Dr. Felix Naumann (Hasso Plattner Institute)

Prof. Dr. Robert Hirschfeld (Hasso Plattner Institute)

PUBLISHED ONLINE

Publication Server of the University of Potsdam:

<https://doi.org/10.25932/publishup-55539>

<https://nbn-resolving.org/urn:nbn:de:kobv:517-opus4-555398>

“Give me knowledge, so I may have kindness for all.”

— From a Plains Indian

ABSTRACT

The availability of commercial 3D printers and matching 3D design software has allowed a wide range of users to create physical prototypes – as long as these objects are not larger than hand-size. However, when attempting to create larger, "human-scale" objects, such as furniture, not only are these machines too small, but also the commonly used 3D design software is not equipped to design with *forces* in mind – since forces increase disproportionately with scale.

In this thesis, we present a series of end-to-end fabrication software systems that support users in creating human-scale objects. They achieve this by providing three main functions that regular "small-scale" 3D printing software does not offer: (1) subdivision of the object into small printable components combined with ready-made objects, (2) editing based on predefined elements sturdy enough for larger scale, i.e., *trusses*, and (3) functionality for analyzing, detecting, and fixing structural weaknesses. The presented software systems also assist the fabrication process based on either 3D printing or steel welding technology.

The presented systems focus on three levels of engineering challenges: (1) fabricating static load-bearing objects, (2) creating mechanisms that involve motion, such as kinematic installations, and

finally (3) designing mechanisms with dynamic repetitive movement where power and energy play an important role.

We demonstrate and verify the versatility of our systems by building and testing human-scale prototypes, ranging from furniture pieces, pavilions, to animatronic installations and playground equipment. We have also shared our system with schools, fablabs, and fabrication enthusiasts, who have successfully created human-scale objects that can deal with human-scale forces.

ZUSAMMENFASSUNG

Die Verfügbarkeit kommerzieller 3D-Drucker und die dazugehörige Software ermöglicht einer großen Bandbreite von Nutzern, physikalische Prototypen selbst herzustellen. Allerdings gilt dies oft nur für handgroße Objekte. Diese Limitation ist auf der einen Seite den kleinen Maschinengrößen von 3D-Druckern geschuldet, andererseits müssen aber auch signifikante, einwirkende Kräfte bereits im Entwurf berücksichtigt werden, was in aktuellen Anwendungen lediglich Benutzern mit entsprechendem Know-How vorbehalten ist.

In dieser Arbeit stelle ich eine Reihe von Software-Komplettlösungen vor, die es einer breiten Benutzergruppe erlaubt, große "human-scale" Strukturen, wie Möbel, zu entwerfen und herzustellen. Diese Systeme gehen in drei Kernaspekten über herkömmliche 3D-Druck-Entwurfsanwendungen hinaus: (1) Die Unterteilung von großen Strukturen in eine Kombination aus druckbaren Objekten und Standardteilen. (2) Entwurf von statisch tragenden Strukturen. (3) Funktionalität zum Erkennen, Analysieren und Beheben von strukturellen Schwachstellen.

Dabei beschränkt sich diese Arbeit nicht auf Softwarelösungen, sondern unterstützt die Benutzer im gesamten Herstellungsprozess,

sowohl bei Prozessen basierend auf dem FDM 3D-Druck, als auch beim Schweißen von Metallen.

Die verschiedenen Systeme, die hier vorgestellt werden, ermöglichen die Erstellungen von tragfähigen, statischen Strukturen über kinematische Installation bis hin zu dynamischen Konstruktionen. Solche gefertigten Konstrukte wie Möbel, Pavillons, Spielplatzgeräte, als auch animierte Installationen demonstrieren die Funktionalität und das weite Anwendungsspektrum des Ansatzes. Ergebnisse dieser Arbeit kamen bereits an Schulen, FabLabs und bei Privatpersonen zum Einsatz, die mit der Software erfolgreich eigene und funktionale "human-scale"-Großstrukturen entwarfen und herstellen konnten.

DECLARATION OF AUTHENTICITY

I declare that all material presented is my own work, or collaborations in which I was always the scientific lead, or fully and specifically acknowledged wherever adapted from other sources. This dissertation has not been previously submitted, in part or whole, to any university or institution for any degree, diploma, or other qualification.

Some ideas and figures have been previously published. Specific chapters and sections that are directly derived from these publications are listed in the following:

Ich erkläre, dass es sich bei allen vorgestellten Materialien um meine eigene Arbeit bzw. um Kollaborationen handelt, bei denen ich stets die wissenschaftliche Leitung innehatte, oder die vollständig und spezifisch anerkannt werden, wenn sie aus anderen Quellen adaptiert wurden. Diese Dissertation wurde bisher weder ganz noch teilweise an irgendeiner Universität oder Institution für irgendeinen Abschluss, ein Diplom oder eine andere Qualifikation eingereicht.

Einige Ideen und Bilder sind bereits veröffentlicht. Hier gelistet sind die Kapitel und Sektionen, die direkt aus diesen Veröffentlichungen abgeleitet sind:

- Parts of the *Introduction* were published as: Harshit Agrawal, Udayan Umaphathi, Robert Kovacs, Johannes Frohnhofen, Hsiang-Ting Chen, Stefanie Mueller, and Patrick Baudisch. 2015. Protopiper: Physically Sketching Room-Sized Objects at Actual Scale. In *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology (UIST'15)*, ACM, New York, NY, USA, 427–436. DOI: <http://doi.org/10.1145/2807442.2807505>
- *Chapter 3* was published and presented as: 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>.
- *Chapter 4* was published and presented as: 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>.
- *Chapter 5* was published and presented as: Robert Kovacs, Lukas Rambold, Lukas Fritzsche, Dominik Meier, Jotaro Shigeyama, Shohei Katakura, Ran Zhang, Patrick Baudisch. 2021. Trusscillator: a System for Fabricating Human-Scale Human-Powered Oscillating Devices. To appear in *Proceedings of the 34th Annual ACM Symposium on User Interface Software and Technology (UIST '21)*. Association for Computing Machinery, New York, NY, USA, 1074–1088. DOI: <https://doi.org/10.1145/3472749.3474807>

Robert Kovacs

Potsdam, December 1st 2021

ACKNOWLEDGMENTS

Foremost, I would like to thank my advisor, Patrick Baudisch for pushing me to think BIG. Apart from his high-level guidance and ingenious ideas, Patrick was my most passionate co-author with 24–7 enthusiasm & dedication. Besides being my advisor and mentor on this work, he also taught me a lesson for life: that every idea needs a *story* to tell.

Dissertations always end up looking like a solo effort. This could not be further from the truth. During the four core projects of this work, I've been lucky to work with brilliant students from HPI: Lukas Rambold, Lukas Fritzsche, Dominik Meier, Anna Seufert, Ludwig Wall, Florian Meinel, Willi Müller, Si-Jing You, Maximilian Brehm, Jonathan Striebel, Yannis Kommana, Alexander Popiak, Philipp Otto, Tim Oesterreich, Johannes Filter, Philip Otto, Tobias Arndt, Nico Ring, Melvin Witte, Martin Taraz, Harshit Agrawal, Udayan Umapathi, and Johannes Frohnhofen; but the list of helping hands remains incomplete.

Furthermore, the wonderful team at the HPI Human-Computer Interaction Lab was inspiring and supporting me every single day. I shall be thankful for spending my days with these wonderful colleagues, co-authors, and friends: Stefanie Müller, Pedro Lopes, Alexandra Ion, Lung-Pan Cheng, Thijs Roumen, Jotaro Shigeyama, Shohei Katakura, Abdullah Muhammed, Sebastian Marwecki, Dominik Schmidt, Hsiang Tim Chen, Oliver Schneider, Jack Lindsay, and Mirela Alistar.

Furthermore, I owe a huge thank for the generous support of the Hasso Plattner Institute and its Research School for creating superb working conditions for conducting my research, that was often requiring special equipment, space, and materials.

I'd like to thank the team at Microsoft Research in Redmond for hosting me during my internship, in particular my mentor Mike Sinclair and the rest of the EPIC team, Eyal Ofek, Mar Gonzalez Franco, Christian Holz, Ken Hinckley, and my intern fellow Alexandra Fay Siu.

I shall thank Prof. Volker Roth for sparking the serendipity to find my way to the HCI research community by inviting me to his research group at the Free University of Berlin. It was a pleasure to work with his wonderful team on their infamous miniature candy packaging plant.

The person who planted the first seed in me for this work was my first mentor, Mrdjanov Stojan "Cole", who taught me airplane modeling, hacking motorcycle combustion engines, and most importantly, nurtured the curious spirit for exploring the world, regardless of danger or cost. His words still echo in my ears: "Do it, do it, and you have made it!" ("*Csinálsz, csinálsz, és megcsinálsz!*" — as he used to say it in broken Hungarian). This is the recipe I use until today. Rest in peace Cole.

I endlessly appreciate the unconditional support of my parents, Rózsa and Zoltán, who created the fruitful foundation for pursuing my PhD studies. My brother Zoltán jr. was my solid anchor, who I could always call up for inspiring, critical conversations and listening ears when it was most needed.

Finally, this work was inevitably inspired and motivated by an architect, a wonderful person who joined me during the journey: Lisa Xuan. We started by designing pavilions and continued by raising two wonderful human beings, Tisza & András. Who could ask for more compassion?

Thank you.

TABLE OF CONTENTS

| | |
|---|----|
| Abstract..... | 4 |
| Zusammenfassung | 5 |
| Declaration of authenticity | 7 |
| 1 Introduction..... | 21 |
| 1.1 Creating large shapes..... | 22 |
| 1.2 Engineering for large forces | 23 |
| 1.3 Contributions..... | 25 |
| 1.4 Structure of the dissertation | 26 |
| 2 Related work..... | 28 |
| 2.1 Scaling up personal fabrication..... | 28 |
| 2.2 Designing mechanisms | 31 |
| 2.3 Designing mechanisms with dynamic behavior | 34 |
| 3 TrussFab: human-scale load-bearing structures | 36 |
| 3.1 Walkthrough of the TrussFab system..... | 38 |
| 3.2 Load-bearing truss structures | 42 |
| 3.3 TrussFab's editor | 43 |
| 3.4 Fabricating hubs and members..... | 50 |
| 3.5 Strength test and safety..... | 54 |

| | | |
|-----|---|-----|
| 3.6 | Implementation | 55 |
| 3.7 | Contribution, benefits, and limitations..... | 60 |
| 4 | TrussFormer: human-scale kinetic structures | 62 |
| 4.1 | Walkthrough of the TrussFormer system | 63 |
| 4.2 | Working principle of TrussFormer’s kinetic structures | 69 |
| 4.3 | Adding motion to the structure | 71 |
| 4.4 | Verifying and adapting forces..... | 74 |
| 4.5 | Matching simulated and real forces..... | 78 |
| 4.6 | TrussFormer’s hinge system..... | 79 |
| 4.7 | Implementation | 84 |
| 4.8 | Contribution, benefits, and limitations..... | 86 |
| 5 | Trusscillator: human-powered human-scale devices | 88 |
| 5.1 | Walkthrough..... | 90 |
| 5.2 | Design space | 97 |
| 5.3 | Expert interviews | 99 |
| 5.4 | Algorithms and implementation..... | 100 |
| 5.5 | Validation..... | 111 |
| 5.6 | Contribution, benefits, and limitations..... | 114 |
| 6 | Conclusion..... | 116 |
| 6.1 | Summary of contributions..... | 116 |
| 6.2 | Impact..... | 118 |
| 6.3 | Limitations and future challenges | 119 |
| 6.4 | Final remarks | 119 |
| 7 | References..... | 122 |

LIST OF FIGURES

- Figure 1: 3D printing at human-scale is limited by the size of the machinery and printing time. 22
- Figure 2: Protopiper is a computer-aided, hand-held fabrication device that allows users to sketch room-sized objects at actual scale – however, only for visual purposes. 22
- Figure 3: Four aspects of human-scale personal fabrication: (1) Protopiper: shape, (2) TrussFab: statics, (3) TrussFormer: kinematics, (4) Trusscillator: dynamics. 24
- Figure 4: TrussFab pavilion created from ~1200 PET bottles and 119 3D printed hubs, presented at CHI'17, Denver, USA. 36
- Figure 5: TrussFab is an end-to-end system that allows users to fabricate large structures sturdy enough to carry human weight. TrussFab considers bottles as beams that form structurally sound node-link structures also known as *trusses*, allowing it to handle the forces resulting from scale and load. 37
- Figure 6: (a) TrussFab's converter automatically turns this 3D model of a coffee table, into (b) a sturdy tetrahedral honeycomb structure, (c) which fabricated serves as a functional table. 39
- Figure 7: TrussFab's editor is implemented as an extension for *SketchUp*. Here the user is performing a stability check on the bridge from Figure 5. 40

- Figure 8: TrussFab generates the unique hub geometry for every node of the model. The hubs are labeled by embossed IDs to help the assembly process. 41
- Figure 9: A functional boat is assembled following the embossed hub IDs. (b) The bottle frame is simply covered by a tarp. (c) The fabricated boat seats two. 41
- Figure 10: Large objects have to withstand substantially larger forces because (1) the loads can be much higher, and (2) forces grow cubed with the size of the lever. 42
- Figure 11: (a) Large objects involve large levers, causing them (b) to break under load. (c) TrussFab instead affords structures based on closed triangles, here forming a tetrahedron. Such structures are particularly sturdy. (d) TrussFab extends this concept to tetrahedron-octahedron trusses of arbitrary size. 43
- Figure 12: The sequence of creating a chair with a backrest in TrussFab's editor. 44
- Figure 13: TrussFab implements the extra-long members using extra 3D print. 45
- Figure 14: (a) Adding *pods* to the bottom hubs and (b) a cover to the top adds stability to our design, as any load is now propagated through the hubs, saving members from buckling. 45
- Figure 15: To verify the chair's structural stability users indicate the load forces. The system now calculates the occurring compression and tension forces in every element. Here, no forces are exceeding the limit; thus the fabricated chair holds the human weight. 47
- Figure 16: (a) Pavilion created using the *beam* and *block* tools. (b) The roof is freely deformed by pulling upwards using the *deform* tool. 48
- Figure 17: (a) The load-bearing structure of this tipi is best made from trusses; (b) its sides can be filled using *facades*. (c) TrussFab makes 2D facade hubs quickly on a laser cutter. 49
- Figure 18: (a) Dome created using the *dome* tool. (b) We make the opening using the *delete* tool, and (c) add decoration

| | | |
|------------|---|----|
| | using the <i>triangle</i> and <i>line</i> tools. (d) The resulting dome is built using 512 bottles. | 50 |
| Figure 19: | 3D printed hub with snap-fit and threaded bottle connectors. | 51 |
| Figure 20: | TrussFab's cuff-connectors are easy to assemble and strong against axial loads. | 52 |
| Figure 21: | Laser-cut planar hubs are secured using a self-locking wedge. | 52 |
| Figure 22: | (a) Connecting bottles with a wood screw using an extra-long screwdriver and (b) with a double-ended screw. (c) Single-bottle edges require a bottom connector | 53 |
| Figure 23: | (a) During strength testing of this truss member the machine held the piece by square test-hubs. (b) Here a pulling test is performed, where the strain-stress diagram shows that the member broke at approximately 135 kg of tensile force, after a 5mm stretch. | 54 |
| Figure 24: | TrussFab's editor toolbar. | 56 |
| Figure 25: | The Stanford-bunny converted using the TrussFab converter in <i>facade</i> conversion mode. | 57 |
| Figure 26: | (a) TrussFormer is an end-to-end system that allows users to design and 3D print large-scale kinetic truss structures that deform and animate, such as this 4m tall animatronic T-Rex. | 62 |
| Figure 27: | (a) The static tetrahedron (b-c) is converted into a deformable structure by swapping one edge with a linear actuator. The only required change is to introduce connectors that enable rotation. | 63 |
| Figure 28: | Our T-Rex encompasses 5 degrees of freedom. | 64 |
| Figure 29: | Modeling the static shape of the T-Rex. Here, the user creates the jaws of the T-Rex by attaching tetrahedron primitives through the steps (a, b, c). | 65 |
| Figure 30: | (a) The user selects the <i>demonstrate movement tool</i> and pulls the T-Rex head downwards. (b) TrussFormer responds by adding an actuator to the T-Rex body so that it is capable of performing this type of motion. At this | |

- point, the system also places 9 hinging hubs to enable this motion (marked with blue dots). 65
- Figure 31: Animating the structure. Users set the desired pose using the sliders in the animation pane and orchestrate the movement by placing key-frames on the timeline. 66
- Figure 32: Verifying the inertial forces: (a-b) The forces are increasing with the acceleration of the structure. (c) The structure breaks when the direction of the movement changes rapidly. (d) TrussFormer resolves this by making the movement slower. 67
- Figure 33: (a) To fabricate our T-Rex model, TrussFormer exports: (b) the appropriate 3D printable hinging-hubs, (c) the specifications for the actuators, and finally the animation sequence as a JSON file for the controller. 68
- Figure 34: Attaching rigid primitives to (a) faces that do not contain an actuator (b) results in simple structures with only localized deformation. This structure acts as a hinge between the two octahedra. 69
- Figure 35: Attaching a rigid primitive (here an octahedron) to a face that contains actuator results in a larger deforming structure. 70
- Figure 36: The motion caused by one actuator propagates throughout the entire truss beam, making it bend. 70
- Figure 37: A selection of assets: (a) tetrahedron with 1DoF, (b) “robotic leg” asset, (c) hinging tetrahedra, (d) octahedron with 1DoF, (e) Stewart platform (6DoF), and (f) double-octahedron performing “bending” motion. 72
- Figure 38: (a) Users start the design by making the body of the walking robot. The predesigned 2DoF “leg” asset is added to the side triangles 6 times. (b) The fabricated robot. 73
- Figure 39: This bike’s steering column is based on the hinge asset, which is used without the actuator in this example. 73
- Figure 40: The “*turn edge into actuator*” tool allows users to turn any edge into an actuator. Here user replaces one edge in the T-Rex’s head to make its jaws move. 74

- Figure 41: In the background, TrussFormer tests each actuator to see if its extension leads to an invalid position, such as the structure tipping over, hitting the ground, or breaking any structural elements. 76
- Figure 42: If the user-defined animation breaks the model, TrussFormer offers to automatically reduce the speed or the motion range. 77
- Figure 43: (a) We measured the forces on the bottom front edge of the T-Rex (b) using a digital force gauge. (c) The measured forces agree with the simulated forces. 78
- Figure 44: (a) Spherical joint mechanism from [13] connecting 5 edges. (b) A segment of TrussFormer's 3D printable hinge design. 79
- Figure 45: (a) Octahedron with an actuator, still with rigid hubs. (b) After the first step, intermediate links are assigned inside all triangles, creating spherical joints. 80
- Figure 46: TrussFormer identified the rigid substructures (a) in the octahedron from Figure 45, and (b) in the T-Rex. 81
- Figure 47: (a) The intermediate hinge connections are reduced to rigid connections where rigid substructures are identified (black lines). (b) The fabricated hinging hub of the marked node of the octahedron. 81
- Figure 48: (a) Double-octahedral structure, where (b) violating three-way hinge connections appear. (c-d) TrussFormer finds the valid configurations by heuristic elimination and (d) chooses the structurally more stable closed chain. 82
- Figure 49: Combining rigid and hinging connections in one hub. 83
- Figure 50: TrussFormer's system diagram. 84
- Figure 51: Hardware setup for controlling the T-Rex with an Arduino, electronic pressure control valves, and a compressor. 85
- Figure 52: (a) Using Trusscillator designers specify the desired motion experience in terms of amplitude, speed, and physical effort; while the system responds by adjusting the coil springs so as to produce the desired behavior. (b) Here, the resulting interactive dinosaur swing

requires two children to synchronize their movement so as to make the sculpture's head wiggle. 88

Figure 53: (a) The cheetah mechanism created using [19] is only resembling the movement pattern of a real one, without considering the forces involved during motion. While (b) energy conservation makes a real-life cheetah's gallop efficient: (c) the elastic tendons store and release energy in every step. 89

Figure 54: Given the scale of the involved forces, the structures created by Trusscillator are made from steel. Trusscillator supports steel truss fabrication by (a) generating stencils that show where to attach the (b) temporary connectors, (c) that hold steel rods in place for (d) welding. 90

Figure 55: (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. 91

Figure 56: (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. 92

Figure 57: (a) When designers change 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. 93

Figure 58: (a) Reducing the effort would require cutting 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. 93

- Figure 59: (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. 95
- Figure 60: Trusscillator offers a temporary connector system to help position the edges for welding. 96
- Figure 61: (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. 97
- Figure 62: Some of the designs we created using Trusscillator. 98
- Figure 63: This “bird-swing” structure was designed to allow children to swing in two dimensions and influence each other’s experience. 98
- Figure 64: 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. 99
- Figure 65: Trusscillator’s high-level architecture. 103
- Figure 66: (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). 107
- Figure 67: (a) Trusscillator simulates the (b) excitation of the model until the point when it reaches an energy equilibrium – maximum *amplitude*. (c) The time after the velocities won’t increase anymore is considered as the *effort* metric. (d) The peak of the frequency spectrum determines the *tempo* metric. 107
- Figure 68: Spring optimization procedure. 110
- Figure 69: (a) The measurement points indicated on the real and virtual model. (b) Frequency response comparison of the push and pull experiments. 112

Figure 70: Our systems allow users to focus on the high-level design objectives, while the software takes care of the specific underlying engineering aspects. 117

Figure 71: Furniture created by tech-enthusiasts using the TrussFab system. 118

1

INTRODUCTION

Digital fabrication tools, such as 3D printers have become popular in today's society, as users in fablabs, makerspaces, or at home, create customized objects on demand. This trend was empowered by advancements in Human Computer Interaction (HCI) and Computer Graphics research, which enabled the usage of fabrication tools for fast prototyping [60], as well as to fabricate soft [32] and interactive objects [41][63], embed optical elements [107], or design kinematic characters [19][56]. It was the availability of 3D printers in a *desktop form factor*, that allowed fabrication to spread to the maker community [95] and the consumer market, empowering *personal fabrication* [83].

However, these desktop-sized personal fabrication machines are limited in that they fabricate at most desktop-sized objects; preventing users from creating larger objects, such as furniture pieces or pavilions, as illustrated in Figure 1. This is because of two major issues: (1) printing large objects requires large machinery, and (2) printing time increases proportional to the volume of the object, therefore cubed to the size. Even though, researchers have demonstrated how to break down designs into smaller pieces that can fit into printers [47], the printing time still remains an issue.

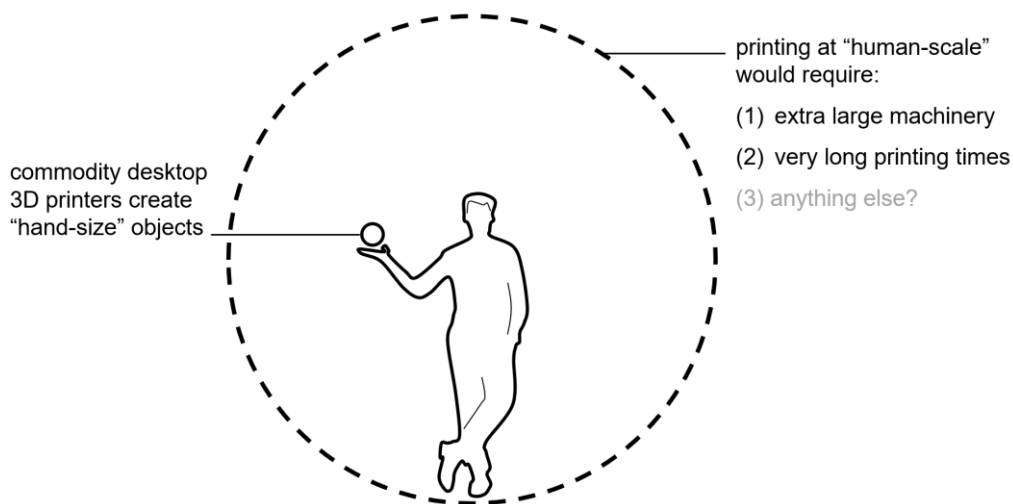


Figure 1: 3D printing at human-scale is limited by the size of the machinery and printing time.

1.1 CREATING LARGE SHAPES

As an alternative solution to achieve large-scale without the necessity of large fabrication machinery we have developed a handheld fabrication device, called Protopiper [2], shown in Figure 2. The key idea behind Protopiper is that it extrudes tubes from adhesive tape rolls. This allows designers to quickly sketch the outer frame of room-sized objects at actual scale and verify their design decisions, for example when furnishing a new apartment.



Figure 2: Protopiper is a computer-aided, hand-held fabrication device that allows users to sketch room-sized objects at actual scale – however, only for visual purposes.

While Protopiper gives a solution to fabricate human-scale objects with a (1) *small* fabrication device (2) in a *short* amount of time, it has a major limitation: the created objects serve purely visual purpose (*shape*).

However, fabricating human-scale objects is not only about achieving the size and reducing printing time, but more importantly, these large objects have to afford substantial external *loads*. Furniture, bridges, and vehicles, for example, all must be engineered to hold the weight of a human. Therefore, with large objects, the main design objective becomes to withstand *large forces*. Designing for large forces, however, requires substantial *engineering skills*, ranging from envisioning the appropriate structures in the first place to verifying their structural integrity [46].

1.2 ENGINEERING FOR LARGE FORCES

To address the mechanical engineering challenges emerging with the larger loads, we took a software engineering approach. We encompassed the required engineering know-how for creating human-scale objects into an end-to-end fabrication software system, called TrussFab [42].

TrussFab achieves the large scale by complementing 3D print with plastic bottles. It does not use these bottles as bricks though, but as beams that form structurally sound node-link structures, also known as *trusses* [46]. These structures are lightweight, yet able to handle the large forces resulting from large scale and corresponding higher loads. TrussFab automatically validates the design via integrated structural analysis and also takes care of the building process by generating the appropriate underlying 3D printable node geometries together with embedded assembly instructions.

TrussFab extends Protopiper's shape-only objects towards load bearing structures, as shown in Figure 3 on the horizontal axes – *static forces*.

As Figure 3 already suggests, our next challenge was to create human-scale objects that involve *motion*, aka. *kinematic* mechanisms. This class of devices raise a set of new challenges, including designing

the *movement path* of a mechanism, and also adding *hinges* to the structure. However, more importantly, when things start to move, the structures are not only exposed to static forces anymore, but we also need to consider the upcoming *inertial forces*, which can be momentarily orders of magnitude larger than the static loads.

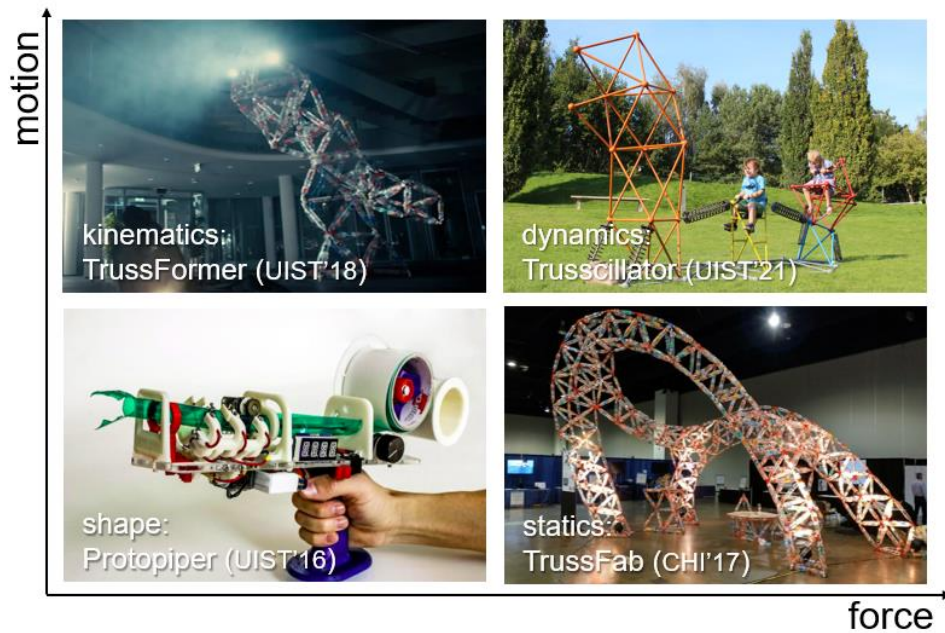


Figure 3: Four aspects of human-scale personal fabrication: (1) Prototpiiper: shape, (2) TrussFab: statics, (3) TrussFormer: kinematics, (4) Trusscillator: dynamics.

With TrussFormer [43] we address these motion-specific challenges within the context of animatronic devices. TrussFormer’s editor enables users to design the desired motion, while the system generates the underlying hinge mechanisms, and more importantly, it affords structurally sound design simulating the inertial forces during the motion.

While TrussFormer solves the motion-specific aspects of the design, there is still one component not to taken into account: the *energy* required to actuate such devices. This aspect is of increased importance when designing human-powered devices, such as playground equipment. Engineering human-powered mechanisms is not only about designing a specific movement path (aka. kinematics), but more importantly we

have to design the motion experience as a consequence of the input power (aka. *dynamics*).

With Trusscillator [44] we extended our approach towards human-scale, *human-powered* devices, within the context of playground equipment. The system features a novel set of interactive tools that 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 use. Since the resulting devices have to deal with high forces, Trusscillator helps users to fabricate them from welded steel. Trusscillator selects the appropriate springs, generates the parts-list, and produces stencils and jigs that help users to weld with precision and ease.

With these three end-to-end software systems (TrussFab, TrussFormer, and Trusscillator) we aim at exploring three aspects of human-scale personal fabrication that go beyond shape: statics, kinematics, and dynamics, as illustrated in Figure 3.

We validate our systems by physically building prototypes, ranging from furniture pieces, pavilions, animatronic installations, and playground equipment. We test their mechanical properties and check against our software predictions.

1.3 CONTRIBUTIONS

The main contribution of this thesis is the blueprint of three end-to-end fabrication systems, that embody the required engineering knowledge for human-scale fabrication. This aspect is identified as the *domain knowledge* category among the six challenges of personal fabrication [8].

As illustrated in Figure 3, our end-to-end software systems provide solution to achieve functionality beyond shape in three key aspects: (1) statics: load-bearing structures (TrussFab), (2) kinematics: moving structures (TrussFormer), and (3) dynamics: moving structures with regard to actuation forces (Trusscillator).

On the *user interaction* side, we contribute with a novel set of tools that afford creating structurally sound objects, such as designing using truss-based primitives that enable voxel-like editing. Automated checks and resolving design flaws are also key parts of the proposed systems. Furthermore, we introduced tools for creating animated movement and authoring the motion experience of devices powered by human input forces.

To *implement* the above-mentioned UI tools, we have integrated engineering solutions, such as differential equation solvers and parametric model generation, into a high-level user interface. One of the key challenges with embedding these often computationally intensive solutions was achieving interactive rates. This required implementing model simplifications, design restrictions, and taking several assumptions about the model into account. We count these technical solutions among our contributions.

On the *hardware* level, we provide a number of custom mechanical solutions for fabricating the structures that are supported by the software workflow. This ensures that all the designs are physically valid, buildable structures. These solutions include building trusses using plastic bottles and 3D printed hubs. We characterized the materials and created 3D printable designs for the connector nodes. We automated the generation of the 3D geometries using parametric design software. Furthermore, we developed a computer-assisted system for the personal fabrication of welded steel structures, thereby laying the groundwork for scaling this line of research to even bigger structures and larger forces.

1.4 STRUCTURE OF THE DISSERTATION

This dissertation is organized into six chapters. After this introduction (chapter 1), we discuss the prior work that we are building on, with a particular emphasis on personal fabrication in HCI (chapter 2). The three

subsequent chapters are dedicated to the three particular aspects of large-scale fabrication: statics, kinematics, and dynamics, covered by three software systems: TrussFab (chapter 3), TrussFormer (chapter 4), and Trusscillator (chapter 5). In chapter 6 we conclude the dissertation by summarizing and discussing the benefits and limitations of our approach, followed by outlining the open challenges of this emerging field.

2

RELATED WORK

This work builds on previous efforts in the following branches: large-scale personal fabrication, mechanism design, and designing dynamic behavior of the mechanism. These topics are discussed and further subdivided below.

2.1 SCALING UP PERSONAL FABRICATION

Architects and engineers have made efforts to scale up the additive manufacturing process (3D printing) to construct large-scale structures, such as houses or sculptures. These efforts involve scaling up the fabrication machinery [116], such as creating concrete printers [40], or breaking down the objects into smaller parts to print on desktop machines [47] [51].

Another significant approach for fabricating architectural-scale objects is by using mobile printing robots, which move on the ground [38] or fly [108] around the printed object. Yoshida et al. [111] proposed a computer-assisted ungrounded fabrication method for large-scale architecture that combines a hand-held chopstick dispenser with a projector-based guiding system. Lafreniere et al. [45] coordinate multiple workers while collaboratively fabricating a pavilion. Going even larger, Whiting et al. [106] have been exploring fabrication at environment-scale by replicating climbing experiences.

Lightweight, large structures are often built using inflatable contraptions. Swaminathan et al. [92] have explored how to fabricate room-scale inflatable structures with embedded input and output capabilities. Sareen et al. [79] and Murayama et al. [62] have built automated machines that fabricate human-scale structures from inflated tubes that can also be actuated. Sato et al. [80] have explored how to build inflatable mobility devices that are soft, yet humans can ride on them.

Construction kits are popular for fast prototyping and fabrication of larger objects. They offer a repertoire of prefabricated elements, which can be combined in various ways. Henrik and Kobbelt [113] developed a system to accurately approximate complex shapes using the *Zometool* mathematical modeling kit. Skouras et al. [84] created an interactive editor to computationally combine interlocking elements into a desired shape. Mueller et al. [61] proposed incorporating (Lego-) bricks into 3D printed models.

At the intersection of fabrication and virtual reality lies *TurkDeck* [17], a system that physically builds virtual environments around a virtual reality user on-the-fly. Finally, room-scale objects have been also instantiated by a large shape display by Suzuki et al. [91].

2.1.1 DESIGNING WITH READY-MADE OBJECTS

MixFab [103], and *Encore* [16] allow users to integrate existing objects into their design. For creating objects enclosing electronic components Ashbrook et al. [5] developed an augmented fabrication system. Devendorf and Ryokai [20] proposed a human-assisted fabrication system that helps users incorporate everyday objects into 3D print. Teibrich et al. [97] have developed a system that is capable of upgrading ready-made by printing on them. *Beady* [33] approximates models from beads and offers an editor for refining them. Gellért assembled wooden boards combined with 3D printed connectors in node-link structure [26].

Skilled individuals have stacked or tied plastic bottles in order to make art pieces, furniture, rafts, or houses [130]. Yamada et al. [110] proposed a system for arranging ready-made objects into 3D shapes using 3D-printed connectors.

2.1.2 TOOLS FOR CREATING STRUCTURALLY SOUND OBJECTS

Beyond achieving scale, engineering the structural stability of objects has been explored in previous research. For example, Smith et al. [85], achieve stability by automatically generating truss structures using non-linear optimization. Whiting et al. [105] proposed a system for procedural modeling of structurally-sound masonry buildings. Makris et al. [52] have developed a design tool that generates parametrically defined, semi-automatically analyzed, and visualized structures. Arora et al. [3] have explored the generative design of optimal Michell trusses depending on the load. Wang et al. [102] developed an automated method that minimizes material cost by converting solid 3D models into a skin-frame structure. Rosen [74] has proposed a workflow for CAD design for additive manufacturing of cellular structures. *SketchChair* [81] is an interactive chair design system that allows users to validate the structural integrity of their design by subjecting it to the weight of a human rag doll. Similarly, Umentani et al. [100] created a system for exploring physically valid shapes in furniture design. Furthermore, *Kyub* [9] and *Fasforce* [1] both aim at creating structurally sound laser-cut closed box structures that can hold human weight.

Commercial tools for engineering truss structures include *SkyCiv* [141], which allows users to analyze the force distribution in trusses. MiTek's *PAMIR* [135] is a specialized tool for creating timber rooftops. *TrussTool* [128] allows assembling constructions from ready-made trusses. While Autodesk's *Revit* [122] provides an overarching solution from architectural design to managing the building process on-site. On the open-source side, *RhinoVAULT* [139] is a research and

development platform for fabricating funicular self-supporting structures.

While these tools are of great help for experts, they might be overwhelming for walk-up users, the common participants in personal fabrication.

2.2 DESIGNING MECHANISMS

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. *TrussFormer* and *Trusscillator* draw from work on systems that assist users with creating mechanisms that involve motion and forces. For example, *ChaCra* [56] is an interactive design system for rapid character crafting. Thomaszewski et. al. [99] looked into generating motion paths for animated kinematic characters. *Bend-it* [109] is a system for creating kinetic characters from bending wire. *Roibot* [48] augments passive everyday objects by adding motorized actuation to them. Ion et. al. [34] proposed an interactive editor for creating mechanical metamaterial mechanisms.

Algorithmic tools can help users create moving mechanisms. For example, kinematic synthesis of mechanisms [90], or generation of personalized walking toys from a library of predefined template mechanisms [11]. These can be embedded in design support systems, for example, generating moving toys from motion input [115], or synthesized planar kinematic mechanisms from sketch-based motion input [19].

Several software tools directly help prototyping linkage-based mechanisms, such as *LinkEdit* [7], *LinkageDesigner* [22], and *Mechanism Perfboard* [36]. These tools sometimes include physical simulation of simple mechanisms (e.g., using hinges); examples include *Crayon*

Physics [124] and *freeCAD* [126]. These software tools provide great help in engineering mechanisms; however, they usually don't streamline the design process to make it fail-safe for novices.

2.2.1 VARIABLE GEOMETRY TRUSS MECHANISMS

TrussFormer's mechanisms are based on variable geometry trusses (VGT) [4][71][87]. An example of a VGT is the *Stewart Platform* [89], a common mechanism found in haptics/HCI. A Stewart platform uses actuators in every member to enable 6 DoF motion while maintaining the stability of a truss, crucial for scenarios that involve large inertial forces.

VGTs have been used extensively in robotics. *Tetrobot* [27] is built by chaining the tetrahedron edges with linear actuators, which unite at a vertex in a spherical joint. The design and mechanics of spherical joints have been extensively analyzed [86][88]. *Tetrobot* was designed to enable robots to reconfigure into different usages by reusing the same basic primitives. Researchers and engineers have explored variations of this VGT design in different contexts, for example, in space applications [4], reconfigurable robotic manipulators [4][27][98], and shape morphing trusses [86].

Other researchers introduced design variations in this basic cell, allowing the resulting structure to afford new qualities. For instance, the *Spiral Zipper* [18] is an extendable edge, based on extending a cylinder that allows for extreme expansion ratios (e.g., 14:1). Similarly, *Pneumatic Reel Actuator* [28] is based on a mechanism that extrudes and retracts a plastic (tape-like) tubing, to act as an actuator. The mechanism is designed to be lightweight and low-cost, while being limited in its robustness.

TrussFormer takes inspiration from VGTs and builds on the conceptual design of *Tetrobot*. To this work, TrussFormer contributes a spherical joint design that is automatically generated based on the designed truss geometry.

2.2.2 SOFTWARE TOOLS FOR ANIMATING MECHANISMS

Many HCI researchers have built software tools to empower users to animate robots [54] [72]. This is especially challenging when the users are novices and the intended results are expressive movements, such as imitating animal (organic) movements, i.e., animatronics [21].

Animatronics interfaces follow several designs, from manual control [54] to puppeteering using skeletal tracking [78]. Marti et al. designed an early example of an animatronics software tool for a small (puppet-sized) phone call handling agent, demonstrating two methods: manual control (user directly controls every single actuator using one GUI fader) and programming motion patterns using a sequencer [54]. Later work integrated robotics with keyframe editors, as for 3D animation [123] or video editing [118].

Previous tools suffice for animating small robots because actuating these robots (typically via small servo motors) does not involve moving large loads. With smaller robots, software tools do not have to simulate the adversarial effects of dynamics, e.g., inertia and resonance. However, when animating large animatronics, these forces affect not only the stability of the structure but also the desired animation

2.2.3 ANIMATING ROBOTIC MANIPULATORS

Programming robotic manipulators is a similar task to creating animation patterns for TrussFormer's mechanisms. The manufacturers of industrial robots usually provide their proprietary software packages for expert users, like ABB RobotStudio [117] or KUKA.Sim [131], while there are powerful open-source platforms as well for programming robot behavior, such as ROS [140]. Recently, also visual block-based interfaces become popular for non-expert users, like KUKA|prc [132] and *CoBlox* [104]. These software tools provide advanced programming capabilities; however, they still lack real-time physical simulation to visualize the occurring inertial forces during the motion.

2.3 DESIGNING MECHANISMS WITH DYNAMIC BEHAVIOR

In contrast to designing for static forces, such as balancing 3D objects [68], predicting the dynamic behavior of mechanisms has been also researched in the HCI and computer graphics community in the past. Interactive design tools leverage physics simulation, such as *Spin-it* [6] enables 3D printing of spinning tops by optimizing the internal rotational dynamic properties, while *Pteromys* [101] helps to optimize the aerodynamics of free-flight glider paper airplanes. Chang et al. [14] have been developing haptic kirigami swatches that are essentially highly specialized springs that provide a well-defined force resistance profile for buttons and switches. Chen et al. [15] proposed a system for accurate simulation of dynamic, elastic objects at interactive rates. Similarly, *Real2Sim* [29] is a system that estimates the material's visco-elastic parameters retrieved from dynamic motion data. Hoshyari et al. [31] have created a workflow for reducing unwanted secondary oscillations in expressive robotic characters. Tang et. al. [96] presented a harmonic balance approach for designing compliant mechanical systems with non-linear periodic motions.

All these projects are dealing with predicting dynamic motion and helping users in their design. TrussFormer and Trusscillator extend this line of work to concern inertial forces, energy, and oscillations.

2.3.1 PROFESSIONAL TOOLS FOR SIMULATING DYNAMIC SYSTEMS

Physics simulation has become one of the most important enabling technologies for engineering physical artifacts. For example, commercial software like *Fusion360* [120] readily offers finite element simulation capabilities for engineers. Some interactive editors utilize powerful frame-based simulation, such as *Algoryx Momentum* [119] or *Vortex Studio* [143]. These systems are great for real-time simulation of complex physical phenomena; however, repeatability and energy conservation in the model are not always guaranteed.

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* [25] and Mathworks' *Simscape* [133]. 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 the analytic solver with a high-level UI tailored for designing spring-based oscillating mechanisms.

2.3.2 SPRINGS AND COMPLIANT MECHANISMS

Springs, in their static and kinematic nature, have already been explored by the personal fabrication community. For example, *Ondulé* [30] helps novices to design parameterizable deformation behaviors in 3D-printable models using helical springs and embedded joints. Schumacher et. al. [82] have proposed a system for modifying the underlying microstructure of 3D printed objects in order to adjust their elasticity. Systems like *Bend-it* [109] and Megaro et al. [57] are focusing on compliant mechanisms that utilize the elasticity of the material to create motion. Roumen et. al. [76] 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 [35] uses preloaded springs to mechanically transmit signals in digital metamaterials. Takahashi et. al. [94] have created a system for creating statically balanced planar spring mechanisms. The bistable nature of compliant mechanisms has been explored by Zhang et al. [114].

While all these works are focusing on springs and elastic behavior, they are mostly concerned about the shape, static balance, and static force that the spring provides. Trusscillator extends these to authoring dynamic motion experiences, beyond the level of designing only motion paths.

3

TRUSSFAB: HUMAN-SCALE LOAD-BEARING STRUCTURES

Fabricating *large* objects requires access to specialized equipment, such as concrete printers that allow making houses [40] or robotic arms capable of 3D printing [37]. In contrast, the owners of the widespread desktop devices cannot participate in this evolution, because the underlying technology does not scale.



Figure 4: TrussFab pavilion created from ~1200 PET bottles and 119 3D printed hubs, presented at CHI'17, Denver, USA.

Even techniques that break down large models into printer-sized parts [47] ultimately do not scale, as large models consume material and time proportional to their size, which quickly renders 3D printing and related techniques intractable for larger-than-desktop-scale models.

As an alternative approach, fabrication enthusiasts have created large objects by combining 3D print with ready-made objects, such as plastic bottles [130]. In their simplest form, such objects wrapped in 3D print can serve as 3D voxel collages that approximate the volume of an object [110].



Figure 5: TrussFab is an end-to-end system that allows users to fabricate large structures sturdy enough to carry human weight. TrussFab considers bottles as beams that form structurally sound node-link structures also known as *trusses*, allowing it to handle the forces resulting from scale and load.

Going larger, however, is not only about scale and print volume. For large objects, the main design objective is typically to withstand *large forces*, as forces grow cubed with the size of the object. Also, large objects afford substantial external loads; furniture, bridges, and vehicles, for example, all must be engineered to hold the weight of a human. Designing for large forces, however, requires substantial engineering

skills [46] from envisioning appropriate structures in the first place to verifying their structural integrity.

In this chapter, we present *TrussFab*, an integrated end-to-end system that allows users to design large structures that are sturdy enough to carry human weight (Figure 5). TrussFab achieves this by taking a different perspective on bottles. Unlike previous systems that stacked bottles as if they were “bricks”, TrussFab considers them as beams and uses them to form structurally sound node-link structures based on closed triangles, also known as *trusses*. TrussFab embodies the required engineering knowledge, allowing non-engineers to design such structures and validate their integrity using its structural analysis functionality. We validate our approach via building physical prototypes, ranging from furniture pieces to a 6m tall pavilion, shown in Figure 4

3.1 WALKTHROUGH OF THE TRUSSFAB SYSTEM

TrussFab allows users to create structures either by modeling from scratch or by converting existing 3D models into trusses. This interactive workflow we summarize in the following.

STEP 1: AUTOMATIC CONVERSION

One way to create TrussFab structures is to convert an existing 3D model using TrussFab’s *converter*. As shown in the example in Figure 6, this tool converts a truncated cone-shaped coffee table into a tetrahedral honeycomb structure, allowing it to bear substantial load.



Figure 6: (a) TrussFab’s converter automatically turns this 3D model of a coffee table, into (b) a sturdy tetrahedral honeycomb structure, (c) which fabricated serves as a functional table.

STEP 2: EDITING

TrussFab’s editor allows users to refine an object created by automatic conversion (Figure 6) or to start a new object from scratch. We implemented TrussFab’s editor as an extension to the 3D modeling software *SketchUp* [142]. TrussFab’s editor offers all the functionalities of the original *SketchUp* system, plus custom functions that help users create sturdy structures. In particular, TrussFab’s editor offers primitives that are elementary trusses (tetrahedra and octahedra), tools that create large beams in the form of trusses, and tools for tweaking the shape of structures, while maintaining their truss structure. In Figure 7, the user placed a human weight on top of the bridge design. TrussFab’s integrated structural analysis shows no warnings, suggesting that the bridge is structurally sound.

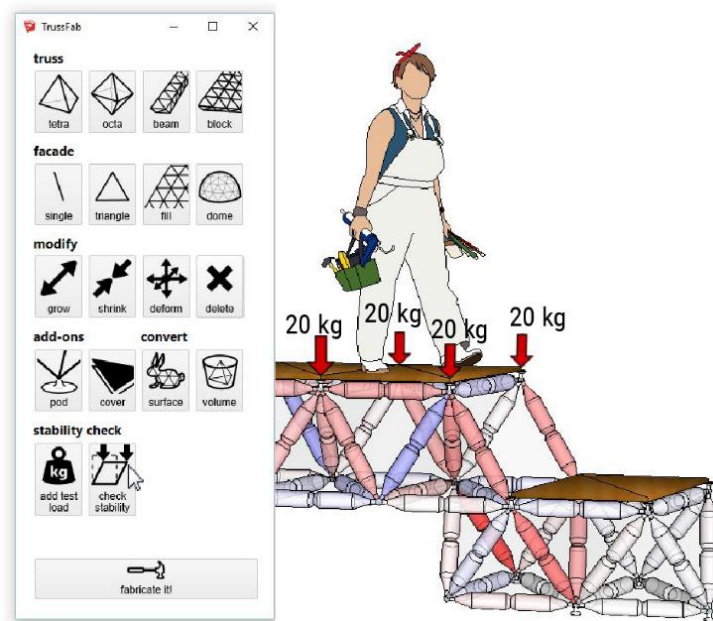


Figure 7: TrussFab’s editor is implemented as an extension for *SketchUp*. Here the user is performing a stability check on the bridge from Figure 5.

STEP 3: HUB GENERATION

After designing and verifying the structure, TrussFab’s *hub generator* generates the 3D models of all hubs. Figure 8 shows one of our 3D printed hub designs; here the connector on the top snaps into the bottleneck, while the bottom ones are holding the bottles by their threaded neck. When designing structures to carry a human weight, these hubs experience large forces. We discuss the details of TrussFab’s hub designs in the section “3.4 Fabricating hubs and members”.

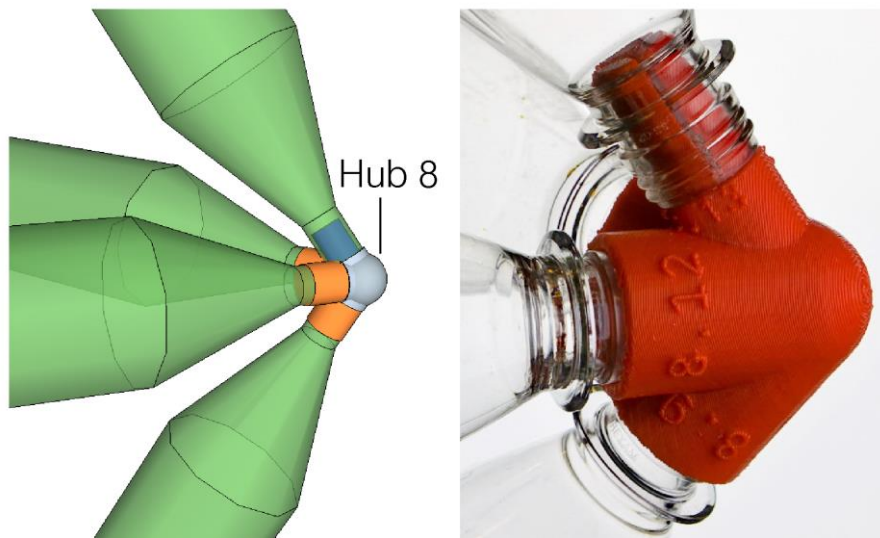


Figure 8: TrussFab generates the unique hub geometry for every node of the model. The hubs are labeled by embossed IDs to help the assembly process.

STEP 4: FABRICATION

Users then send the generated 3D model files (STL) produced by the hub generator to one or more 3D printers in order to manufacture them. Typically, a hub prints in 2-3 hours.

STEP 5: ASSEMBLY

Finally, users manually assemble their structures by following the unique IDs embossed into each hub (as shown in Figure 8b). In Figure 9, a boat for two is assembled using 124 PET bottles and 31 3D printed hubs. The assembly time required for this model is approximately ~2h.

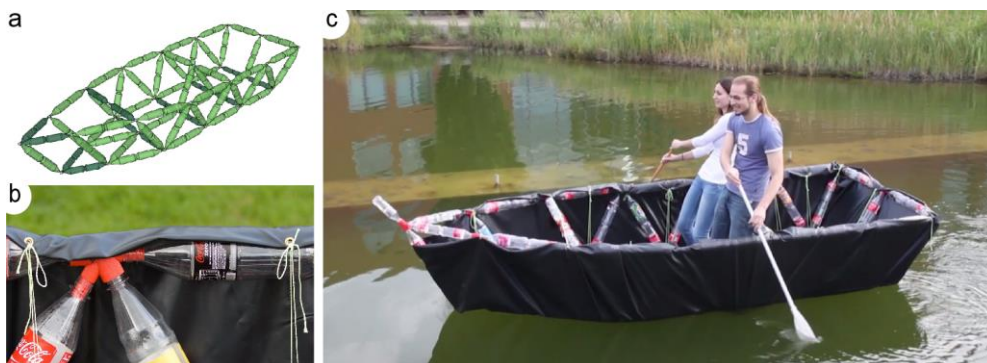


Figure 9: A functional boat is assembled following the embossed hub IDs. (b) The bottle frame is simply covered by a tarp. (c) The fabricated boat seats two.

3.2 LOAD-BEARING TRUSS STRUCTURES

TrussFab’s structures tend to be exposed to substantially larger forces than hand-size objects. As illustrated in Figure 10, this is due to two main reasons: (1) large objects tend to support much larger loads than their hand-sized counterparts, and, less obviously, (2) forces grow cubed with the size of the levers (when levers are involved). These two factors result in orders of magnitude higher forces in the construction, that need to be addressed by a proper underlying structure.

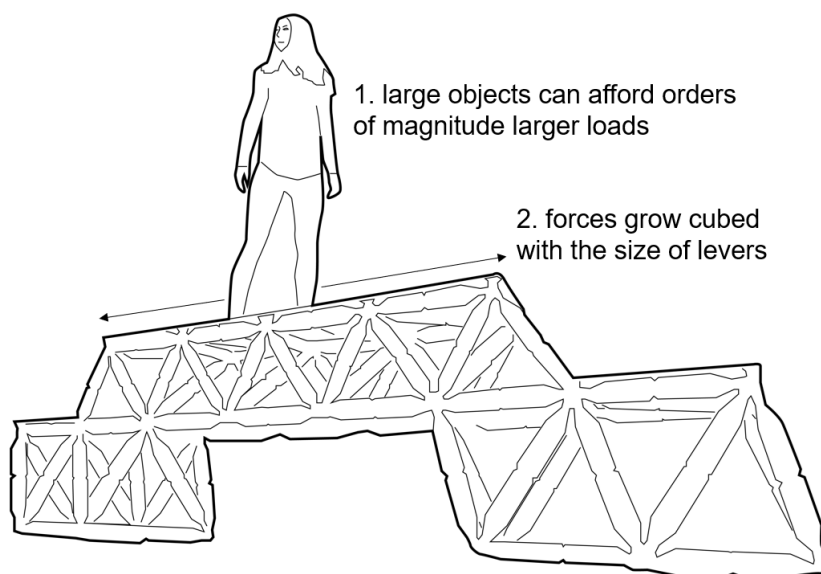


Figure 10: Large objects have to withstand substantially larger forces because (1) the loads can be much higher, and (2) forces grow cubed with the size of the lever.

The key idea behind TrussFab’s structures is to (1) employ bottles in their structurally most sturdy way, i.e., as beams from bottom to bottleneck, arranged in (2) sturdy “closed frame structures”, also known as *trusses* [46].

While freestanding bottles tend to break easily (Figure 11a/b), truss structures essentially consist of triangles. In such an arrangement, it is the structure that prevents deformation, not the individual bottle. The main strength of trusses is that they turn lateral forces (aka bending moments) into tension and compression forces along the length of the

edges (aka. *members*). Bottles make great members: while they buckle easily when pushed from the side, they are *very* strong when pushed or pulled along their main axis (see section “3.5 Strength test and safety”). (c) The resulting structures, such as this tetrahedron, are strong enough to bear the weight of one or more humans. (d) TrussFab affords building trusses by combining tetrahedra and octahedra into so-called tetrahedral honeycomb structures. The coffee table in Figure 6, for example, is cut from such a “tetra-octa” mesh. This structural arrangement is commonly used in truss design and provides great structural stability [46].

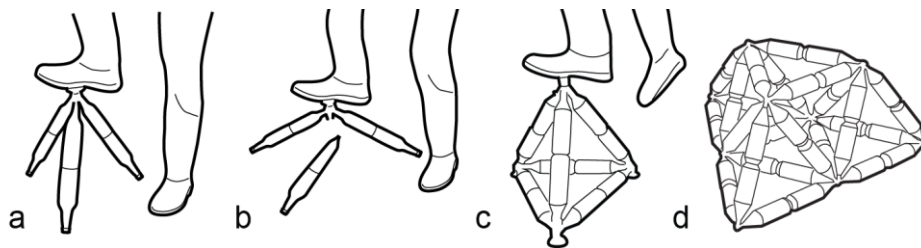


Figure 11: (a) Large objects involve large levers, causing them (b) to break under load. (c) TrussFab instead affords structures based on closed triangles, here forming a tetrahedron. Such structures are particularly sturdy. (d) TrussFab extends this concept to tetrahedron-octahedron trusses of arbitrary size.


Alternatively, users might also consider using larger sturdy primitives, such as the icosahedron, however, we opted for the tetrahedron and octahedron, as they are space-filling, thus providing a simpler construction grid than other geometries.

3.3 TRUSSFAB’S EDITOR

We now zoom in on Step 2 of our walkthrough: the TrussFab editor. The main design rationale behind the editor is to afford a stable structure. It achieves this by using bottles as the members of trusses. The key idea behind TrussFab’s editor is to start any design with primitives that already are trusses, i.e., tetrahedra and octahedra; additional functions then allow users to extend and tweak the structure while maintaining

the truss property at all times. Once the main truss structure has been created, users may add facades and decorative details.

We demonstrate this principle on our *chair* design in Figure 12.

(a)  As illustrated by Figure 12a, we start our design by creating the base of the chair. We select the *octahedron* tool from TrussFab’s drawing toolbar and click on the ground plane of our workplace, which creates an octahedron. By default, this octahedron is made from small (half-liter) bottles. We check its height using the standard SketchUp measurement tool—it is 45cm, which is a good height for an average person to sit on.

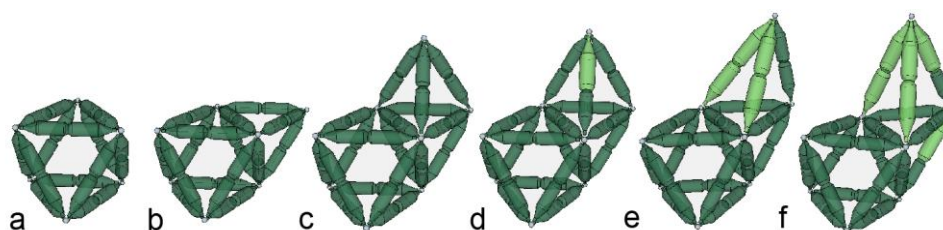




Figure 12: The sequence of creating a chair with a backrest in TrussFab’s editor.

(b)  To make the backrest, we now select the *tetrahedron* tool. We click on one of the sides of the base octahedron, which attaches a tetrahedron to it. (c) We click the top surface of the tetrahedron we just made, attaching one more tetrahedron to it. This gives us the rough shape of our chair and its backrest.

(d-e)  The backrest is a little short. We select the *grow* tool and click one of the three upper members of the backrest. This elongates one of the two bottles to the next supported size, i.e., a 1.5-liter bottle, here shown as light green. We click again, which causes the second bottle of this member to grow as well. Repeating this on the other side scales the backrest to the desired size.

(f) The chair is now too “laid back”. Still holding the *grow* tool, we click the rear edge of the backrest until the backrest is more vertical. Alternatively, instead of using the discrete *grow/shrink* tools, users can also use the *deform* tool, which allows freely dragging individual hubs

in space, while preserving the truss nature of the model (see section “3.3.1 Editing larger structures efficiently”). In addition to the standard member lengths governed by bottle sizes, TrussFab can implement extra-long members and in-between sizes by extending hubs with 3D print, as shown in Figure 13. This allows users for more freedom in their design.

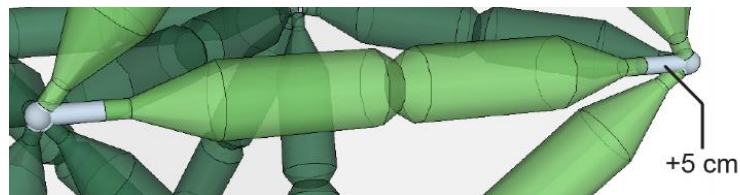


Figure 13: TrussFab implements the extra-long members using extra 3D print.

✂ We now select the *pod* tool and use it to add *Pods* to the bottom of our chair, as illustrated by Figure 14a. As discussed earlier, bottles are sturdy only when forces apply *along* their main axis. This is not the case for an octahedron directly touching the ground. If we tried to sit on it, our weight would cause the members touching the ground to buckle and break. The *Pods* avoid this by propagating the user’s weight into the truss, making the design robust.

▼ Finally, we select the *cover* tool and click the top of the octahedron (Figure 14b). This adds a plywood plate for users to sit on; it is supported by three upward-facing pods. Plates will later be exported as *SVG* files. We tend to fabricate them on a laser cutter.

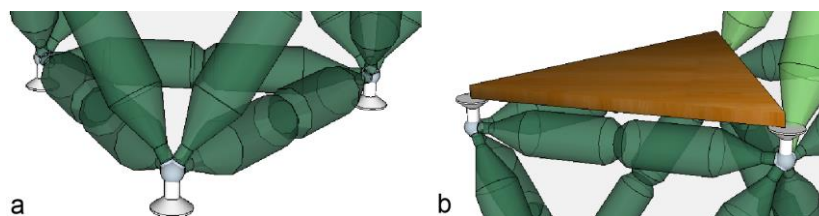




Figure 14: (a) Adding *Pods* to the bottom hubs and (b) a cover to the top adds stability to our design, as any load is now propagated through the hubs, saving members from buckling.

 To verify the chair's ability to carry a human user, we select the *add weight* tool and click on the seat plate (Figure 15). This places 10 kg weight on each of the three corners of the plate. We click three more times, which increases the weight in 5 kg steps, to sum up in total of 75 kg. A click at the backrest adds another 10kg load pushing into the backrest.

 Clicking the *check stability* icon causes TrussFab to compute the effect of these weights onto the structure. This happens in two steps.

First, the software looks for flaws in the truss structure, i.e., it searches for parts that are not completely rigid and are subject to shearing forces (see section "3.6 Implementation"). If found, the software would suggest placing additional stabilizing members. Our chair, however, is rigid, so there are no warnings.

Second, the software checks whether the structure will hold up to the weight we placed on it. Using finite element analysis, the software calculates the forces that apply to every member of the structure. As shown in Figure 15, TrussFab shades all members accordingly. The six vertical members of the octahedron now appear in shades of red, suggesting that these are experiencing compression. So does the chair's "backbone". All other members are tinted blue, suggesting that these are subject to tension.

TrussFab compares these forces with the maximal load members and hubs can hold (see section "3.4 Fabricating hubs and members") and it warns the user if the limits are exceeded. This is not the case here, so we now know that our chair model is structurally sound.

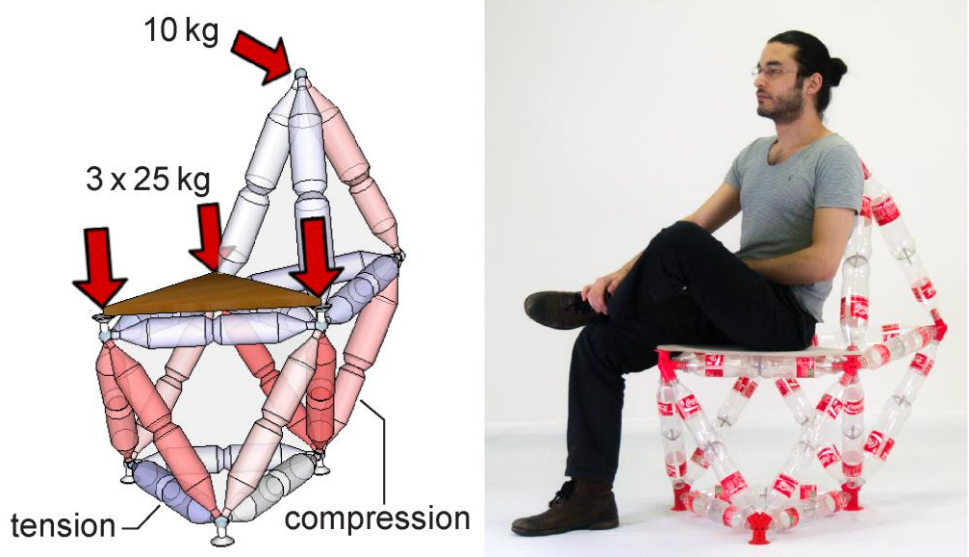


Figure 15: To verify the chair's structural stability users indicate the load forces. The system now calculates the occurring compression and tension forces in every element. Here, no forces are exceeding the limit; thus the fabricated chair holds the human weight.

→ Finally, the *fabricate it!* button causes TrussFab to generate 3D models of all hubs and export them in *STL* format to the 3D printer. For the wooden seat cover, TrussFab creates a 2D line drawing in *SVG* format and sends it to a laser cutter. Users now assemble hubs and bottles based on the embossed hub IDs, resulting in the chair shown in Figure 15b. This particular design prints in 90 min per hub on a *MakerBot 2X*, and takes about 20 minutes to assemble.

Note how the interaction we described afforded creating a stable structure. In particular, the octahedron we started with was a truss and thus stable. We then added tetrahedra, which turned our initial truss into a larger truss. Tweaking the length of individual members, finally, did not affect the structure of our design, so it remained a truss at all times.

3.3.1 EDITING LARGER STRUCTURES EFFICIENTLY

To create larger objects, TrussFab offers a number of tools that create larger trusses in a single interaction, thus resulting in a more efficient design process.

✂ The *beam* tool creates entire beams in one go. The bridge in Figure 5 and the pavilion in Figure 16 were created this way.

🔲 The *block* tool creates a tetra-octa plane in one go. We used it to create the roof of the pavilion in Figure 16. It can also serve, for example, as a stage.

✚ The *deform* tool allows users to deform trusses. In Figure 16b we applied this tool in order to obtain a curved roof. Using the tool, we grabbed a hub located in the middle of the roof and dragged it upwards. The tool accommodates this by growing and shrinking members throughout the truss (see section “3.6 Implementation”

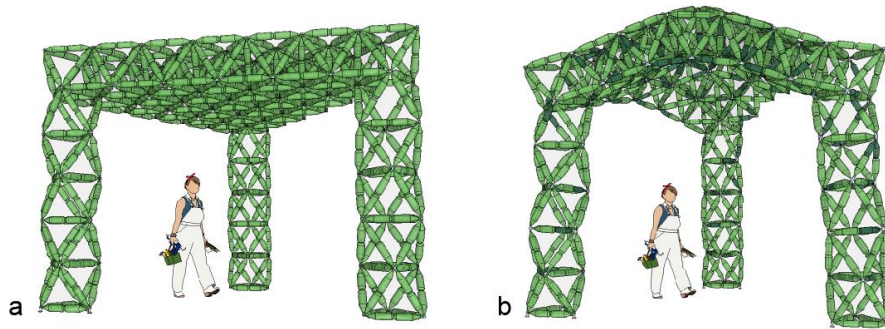


Figure 16: (a) Pavilion created using the *beam* and *block* tools. (b) The roof is freely deformed by pulling upwards using the *deform* tool.

🔲 The *facade* tool allows filling in a facade between two trusses. This tool flood fills the plane in between two trusses with a triangle mesh. In Figure 17, we used this tool to create the walls of a tipi. (a) Users start by creating a load-bearing skeleton structure in the form of truss beams. (b) Users then fill in the non-load-bearing sides as facades. The benefit of this two-stage process is that facades are particularly efficient. First, they are single-layer, thus require fewer bottles. Second, the hubs that form a facade are flat; this allows TrussFab to fabricate such hubs using a laser cutter, which is very fast (40x faster than 3D print, see section “3.4.3 Laser-cut hubs for facades”).

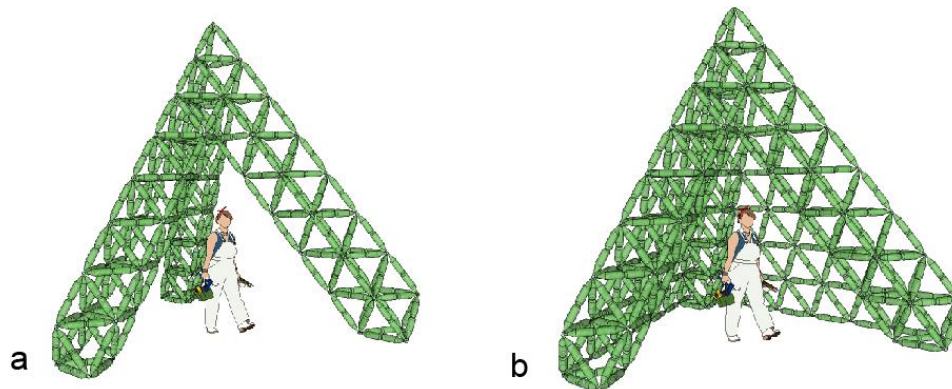



Figure 17: (a) The load-bearing structure of this tipi is best made from trusses; (b) its sides can be filled using *facades*. (c) TrussFab makes 2D facade hubs quickly on a laser cutter.

To support this classic layered architecture, TrussFab complements all of its truss tools with specialized facade tools. The editor offers facade tools for filling the large opening with facades (Figure 17b). The hub generator offers the aforementioned laser-cut facade hubs. And, for objects not expected to bear a load at all, TrussFab’s surface converter turns the outer hull into a hollow facade structure (Figure 25).

 The *dome tool* supports creating geodesic domes in one go (Figure 18). Domes are particular in that they support themselves by means of their own curvature, i.e., *without* any underlying truss structure. We created the shown tent by (a) creating a dome by selecting TrussFab’s *dome* tool and clicking on the ground. (b) We create an opening in the front using TrussFab’s *delete* brush. (c) Using the *triangle* and the *line* tools we add the decorative ears on top of the dome. (d) The resulting dome is assembled from 512 bottles, 68 3D-printed, and 63 laser-cut hubs.

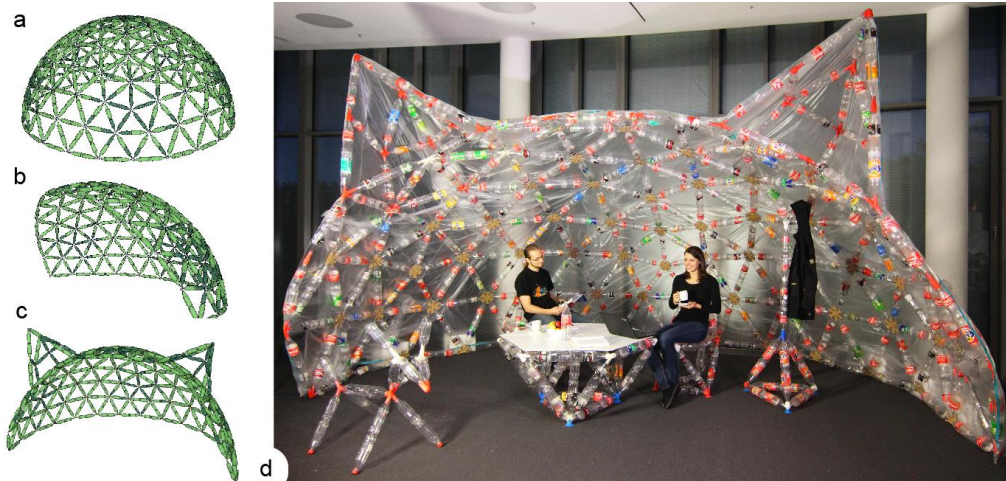


Figure 18: (a) Dome created using the *dome* tool. (b) We make the opening using the *delete* tool, and (c) add decoration using the *triangle* and *line* tools. (d) The resulting dome is built using 512 bottles.

3.4 FABRICATING HUBS AND MEMBERS

As mentioned earlier, TrussFab’s hubs are exposed to loads in the range of human weight, making their design crucial for achieving sturdy structures. Each hub connects two or more bottles by their necks. These regions we call the *connectors*. In the following, we describe our two 3D printable connector designs that use different mechanical principles, and a hub design suitable for laser cutting.

3.4.1 THREAD AND SNAP CONNECTORS

The thread and snap connector pair is shown in Figure 19. The threaded design is very straightforward, users simply screw the respective bottle into the connector. Unfortunately, trusses cannot be assembled from threaded connectors alone, because the last member of a closed contour would require turning one end left and the other right to screw both ends simultaneously. TrussFab, therefore, complements threaded connectors with a second type of connector that allows free rotation.

The snap connector slides into the bottleneck and holds the bottle from the inside (Figure 19). A three-way split in the connector forms a set of cantilever springs that are compressed when the connector is

inserted into a bottle, allowing the tip to slip in. When it reaches the point where the bottle widens, it expands and now resists being pulled out. To secure the connector against tension, users insert a pin into the hole from the opposite side, as shown in Figure 19, which prevents the prongs from squeezing. Releasing the snap-fit connector is done by pushing the pin all the way into the bottle.

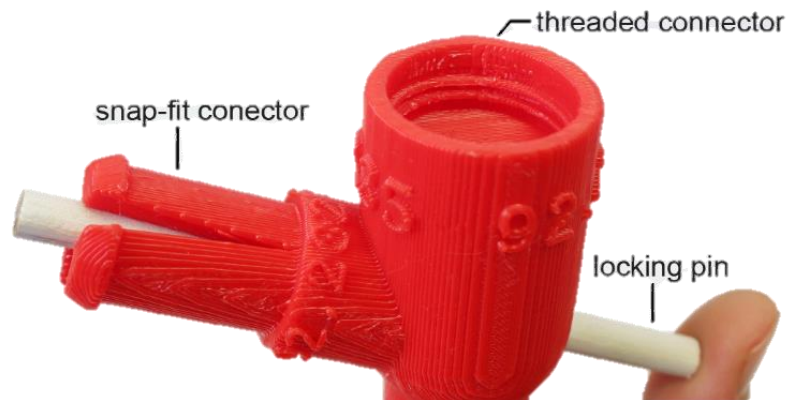


Figure 19: 3D printed hub with snap-fit and threaded bottle connectors.

Similar to the threaded connectors, it is generally not possible to implement a TrussFab structure using snap connectors alone. The reason is that if two snap connectors are facing close to the opposite on the same hub, the locking pin technically cannot be inserted.

TrussFab resolves this challenge automatically by using at least one snap connector per closed contour and by using threaded connectors opposite to any snap connector.

3.4.2 CUFF CONNECTOR

The cuff connector design is based on a generic flange and a separately printed cuff, that snaps on the bottle neck, as shown in Figure 20a. We used this type of connector for building our pavilion from Figure 4. This design has a number of advantages: (1) Quick assembly and disassembly. (2) The cuff can be printed separately, resulting in optimal slicing against axial loads (Figure 20b). (3) In case of breaking, usually only the cuff needs to be exchanged. (4) When using different

bottle types, only the cuffs need to be redesigned to make the new bottle fit.

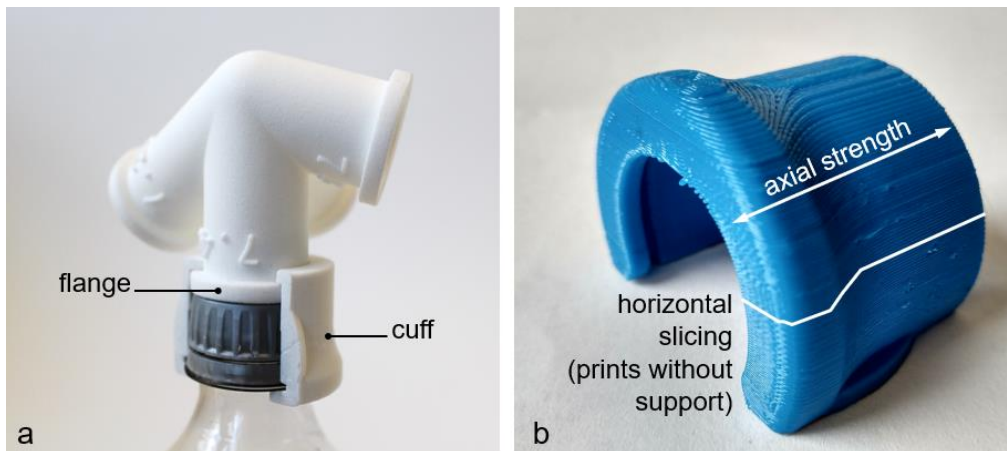


Figure 20: TrussFab's cuff-connectors are easy to assemble and strong against axial loads.

By default, TrussFab creates connectors that fit the bottlenecks of the most common bottles (*PCO 28mm* thread); However, connectors can be designed for any other ready-made objects in TrussFab's OpenSCAD script (see section "3.6.6 Hub generator").

3.4.3 LASER-CUT HUBS FOR FACADES

Figure 21 shows the laser-cut connectors we use for facades and domes. We tend to fabricate them from particleboard or optionally plywood for extra stability.

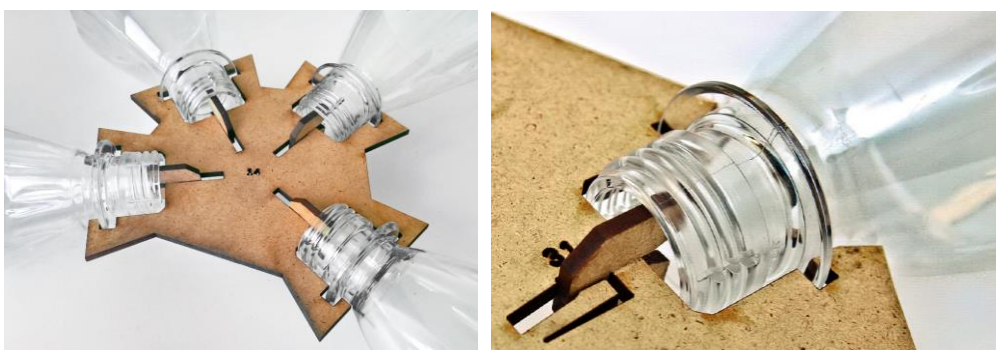


Figure 21: Laser-cut planar hubs are secured using a self-locking wedge.

Each laser-cut connector consists of two parts. The plate forms the hub itself. It is cut to accommodate the flange of the bottle, which prevents the bottle from moving in-and-out along its main axes. Inserting a wedge prevents the bottle from slipping out of the plane of the connector.

Even though hubs are flat when fabricated, assembling them into a curved structure, such as a dome (Figure 18), requires hubs to assume this curvature. TrussFab fabricates laser-cut connectors with play to allow for this.

3.4.4 FABRICATING MEMBERS FROM BOTTLES

As shown in Figure 22, TrussFab generally uses (a) long wood screws to connect the bottoms of two bottles or (b) double-ended screws, tightened by rotating the bottles in opposite directions. (c) In the rare case of short, single-bottle members, TrussFab uses bottom-to-hub wood screw connectors.

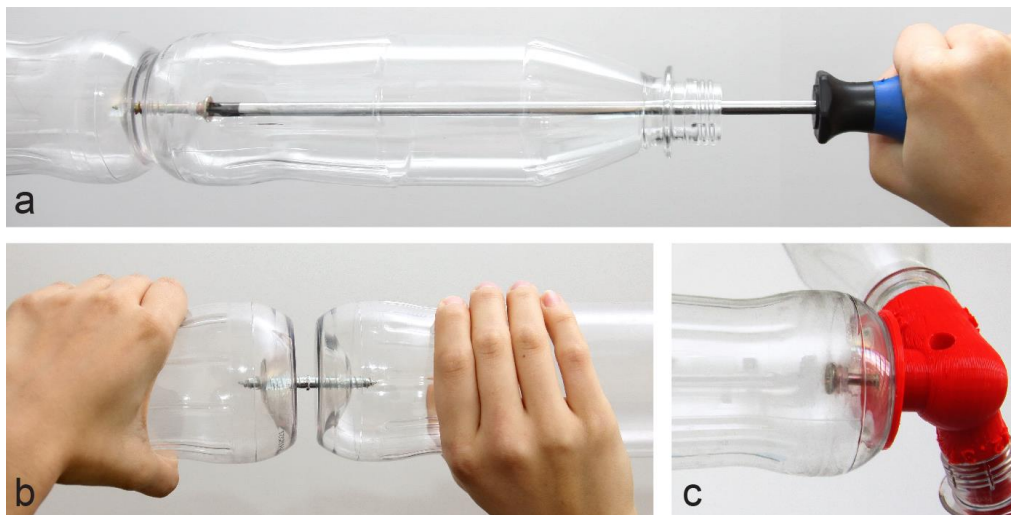


Figure 22: (a) Connecting bottles with a wood screw using an extra-long screwdriver and (b) with a double-ended screw. (c) Single-bottle edges require a bottom connector

We made all our TrussFab objects from *refillable* plastic bottles. Since these bottles are designed to be used multiple times, they feature thicker walls, resulting in sturdier structures.

For fabricating facades or non-load-bearing structures we connect bottles using 6" wide adhesive tape. This leaves the bottles intact, allowing us to return the bottles. It also works well with thinner, disposable bottles.

3.5 STRENGTH TEST AND SAFETY

The structures intended to carry a human weight need to undergo careful design and testing. Before building, users should verify the stability of *their particular* bottle members and hubs using an appropriate testing procedure. We describe this procedure in the following.

In order to determine the maximum load that our bottle members can undergo, we used the mechanical break testing machine shown in Figure 23a. We used the 3D-printed test connectors to attach the bottles to the machine. The machine then applied increasing tension or compression, until the tested element breaks, resulting in the strain-stress diagram shown in Figure 23b.

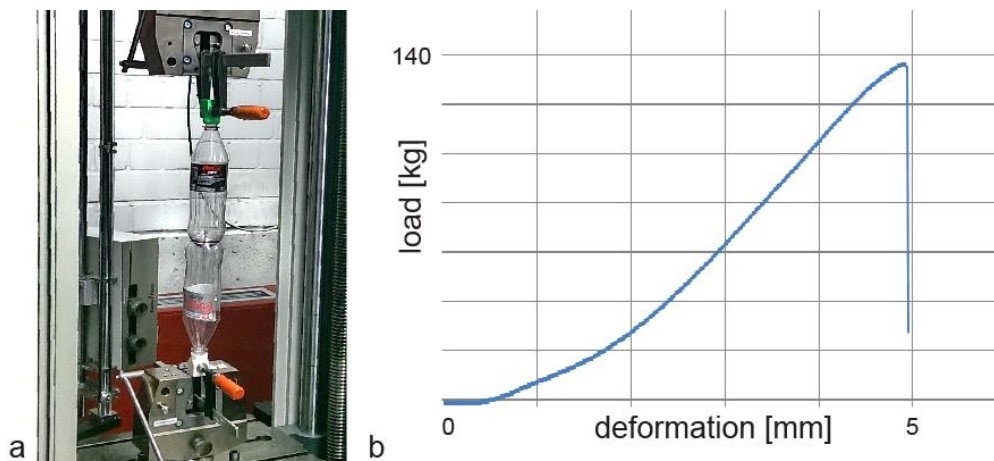


Figure 23: (a) During strength testing of this truss member the machine held the piece by square test-hubs. (b) Here a pulling test is performed, where the strain-stress diagram shows that the member broke at approximately 135 kg of tensile force, after a 5mm stretch.

Table 1 shows the maximum loads our hubs and members did withstand under idealized conditions, i.e., room temperature and

without any dynamic loads. For our members, we were able to apply up to 85kg of pressure (at which point the bottle buckles and collapsed) or 135 kg of tension (at which point the FDM-printed ABS hubs tore).

| | threaded connector | snap connector | bottle member |
|--------------------|---------------------------|-----------------------|----------------------|
| compression | (any) | (any) | 85 kg |
| tension | 135 kg | 145 kg | 180 kg |

Table 1: Breaking points of our threaded and snap connectors and bottle members.

Note that these results were obtained with thick refillable bottles—disposable bottles tend to break under smaller loads. Also, slicer settings, print-orientations, and hub materials may lead to different results. Thus, the testing procedure needs to be performed with the respective bottles and 3D printing technology at hand.

These measured values need to be complemented with additional safety factors that represent the expected dynamic loads and in the case of outdoor deployment also factors resulting from environmental conditions, such as wind forces, wear, and weather decay.

3.6 IMPLEMENTATION

To help readers replicate our results, we now describe the implementation of the main components of the TrussFab system: TrussFab’s editor, converter, force analyzer, and hub generator.

3.6.1 EDITOR FRONTEND

We implemented TrussFab as an extension to the 3D editor SketchUp [142]. It is written in Ruby and JavaScript. It allows users to create 3D models, use efficient editing tools, verify stability, and trigger TrussFab’s hub generator, as illustrated in Figure 24.

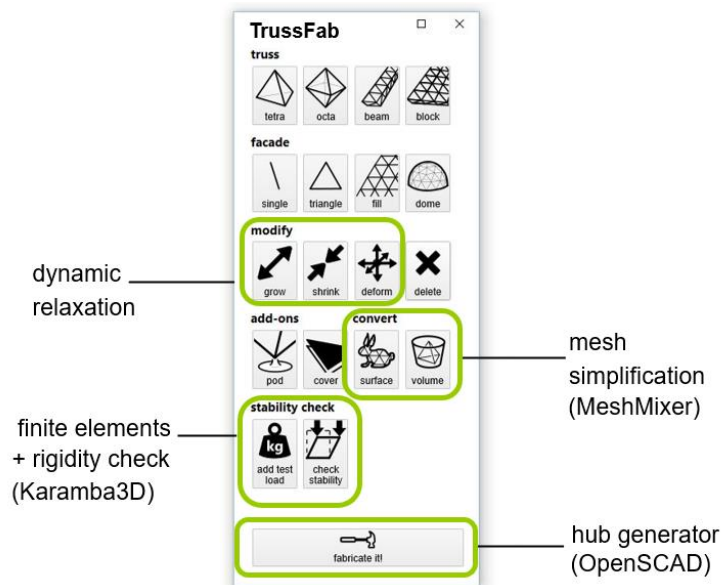


Figure 24: TrussFab's editor toolbar.

3.6.2 MODIFYING TRUSSES

In the *Modify* tab are the *grow*, *shrink* and *deform* tools, that are the most computationally intensive. They affect the lengths of members and consequently all the angles between members. After altering the position of a node, TrussFab restores the consistency of the 3D model by running a *dynamic relaxation* algorithm [65], i.e., neighboring members start to push-pull each other until they find the position that accommodates the change. TrussFab iterates up to 10,000 cycles or until 0.1 mm accuracy has been reached.

3.6.3 AUTOMATIC CONVERSION

TrussFab's converter offers two modes of operation: volumetric and surface conversion.

The volumetric conversion procedure is similar to traditional voxelization methods. However, instead of intersecting the given 3D model with a regular cubical grid, TrussFab intersects the model with a tetrahedral honeycomb, as demonstrated earlier on the example of a table in Figure 6. The algorithm also iterates to find those angles and positions that maximize the number of fully enclosed edges. We note

that our solution is focused on space-filling with fixed edge length, while more elaborate flexible space-filling algorithms have been proposed by Arora et al. [3], Mitra et al. [59], and Wang et al. [102].

The surface conversion procedure reproduces the object’s facade as members, as illustrated in Figure 25. The main challenge here is to ensure that every edge of the 3D model either fits the length of one of the bottle primitives or is slightly longer, in which case the converter will lengthen the edge by extending the respective connector.

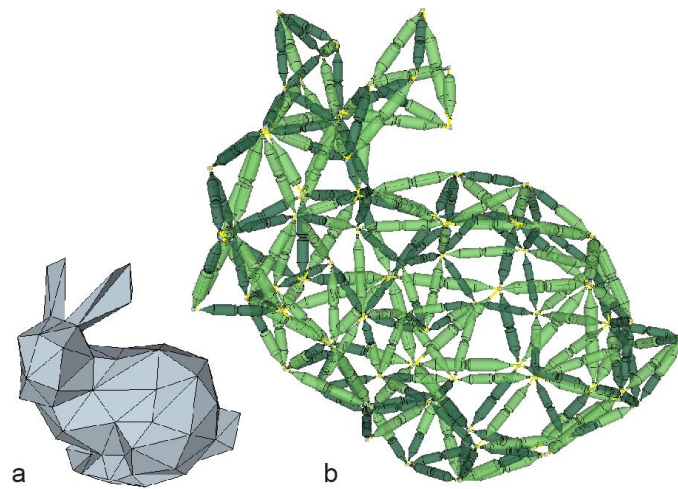


Figure 25: The Stanford-bunny converted using the TrussFab converter in *facade* conversion mode.

Our surface conversion tool, inspired by Richter and Alexa’s *Beam Meshes* [73], consists of two stages: mesh simplification and surface re-meshing. In the mesh simplification stage, we use the quadric-based edge collapse function in *MeshLab* [134] until it reaches the desired number of edges. We preserve certain features, such as the ears of the bunny in Figure 25, by manually simplifying the 3D model using the simplification brush in Autodesk *MeshMixer* [121].

In the surface re-meshing stage, we optimize the vertex position of the model so that all edges are of the valid length of bottle primitives and the distortion of the final mesh is minimized. The energy function has two terms, where the first term is the minimum distance between an

edge and the bottle primitives and the second term is the distance between the vertex position and the original simplified mesh.

More specifically, the energy function is of the form

$$E(\mathbf{V}) = \sum_{i=0}^m \min(\sum E_i - \mathbf{B}) + \alpha \sum_{i=0}^n \text{Dist}(v_i, \mathbf{S})$$

where \mathbf{V} are the vertices of the simplified 3D model, n and m are the numbers of vertices and edges respectively, E_i is the length of the edge i , \mathbf{B} is the set of valid lengths for all bottle primitives, $\text{Dist}(v_i, \mathbf{S})$ is the distance between vertex i and the surface \mathbf{S} of the given 3D model. We calculate the optimized vertex positions using Powell's COBYLA optimization routine [66].

The algorithm does not account for structural stability; therefore, optional reinforcement needs to be added manually. Also, physical self-intersections need to be corrected by the user. The converted models are exported to the TrussFab editor in the form of a JSON file.

3.6.4 FORCE CALCULATION

TrussFab uses *Karamba3D* [67] finite element analysis (FEA) for calculating the loads. This method models each edge as a spring of particular stiffness and calculates the displacement of the nodes under the given force [24]. TrussFab treats all hubs as ball joints, allowing for deformations without breaking. The bottle members are modeled as filled cylinders, which are rigid in shear. The pods touching the ground are considered anchor points.

TrussFab sends the geometry of the model together with the specified load forces to *Karamba3D* in JSON format, which returns the resulting compression and tension forces for each member.

3.6.5 RIGIDITY CHECK

Any TrussFab model is a node-link diagram that can be represented as graph G . Conveniently, by analyzing this graph, a check of structural rigidity can be performed [77]. Because of its relevance to this thesis, we describe this procedure below.

Let's consider a movement of the vertices given by specifying a velocity $\mu_i(t)$ for each vertex v_i at every point in time t . Let p_i be the initial position of v_i . Then the movement preserves the length of an edge $v_i v_j$, if and only if

$$(\mu_i(t) - \mu_j(t)) \cdot (p_i - p_j) = 0$$

holds for every point in time. Thus, to check G for rigidity, we can instead test whether velocities satisfy this equation for every edge. As each equation is linear, we obtain a system of linear equations. This system can be written as $A\mu = 0$ where μ is the vector of all velocities and each row of the matrix A corresponds to one equation. The matrix A is the above-mentioned rigidity matrix. Note that μ has dimension $3n$ as we have one velocity for each vertex and each velocity is 3-dimensional. Thus, if $\text{rank}(A) = 3n - 6$ then the solution space of $A\mu = 0$ is 6-dimensional, which covers exactly the trivial movements of rotating (in three dimensions) and translating (in three dimensions) the whole graph. Hence, if $\text{rank}(A) = 3n - 6$, no other edge-length preserving movement can exist, i.e., G is rigid.

3.6.6 HUB GENERATOR

The TrussFab Hub Generator generates the 3D models of the hubs using the mathematical solid modeling tool OpenSCAD [137].

The TrussFab Hub Generator receives its input from the TrussFab editor in OpenSCAD script file format (*.scad*). (1) For 3D printed hubs, this data file describes each connector using a direction vector for each connection, annotated with connector type, elongation, and ID. (2) For laser-cut hubs, the plug-in projects the connections onto a plane before exporting the hub as a 2D geometry.

TrussFab Hub Generator generates hubs by arranging the individual connector primitives around a sphere. The connector geometry is loaded from separate modular files, allowing users to include their own, custom connector types for using different ready-made objects in the design.

3.6.7 FABRICATION AND ASSEMBLY

We fabricated the 3D hubs on MakerBot 2X and Ultimaker 2 desktop FDM printers. Each hub consumes about 50-150 g of filament, resulting in \$2-5 cost. One average hub prints in about 1.5-2.5 h, using a 0.5 mm nozzle. We mostly used PLA and ABS materials, however also experimented with recycled PET material successfully. The laser-cut hubs are made from 5 mm particleboard, which took about 3 minutes/hub to cut on a *Universal UL 150D* laser cutter.

Table 2 summarizes the bottle-hub count, printing, and assembly time for all presented objects. The refillable bottles were acquired for their deposit value (\$0.15/piece).

| | number of bottles | number of hubs | printing time | assembly time |
|-----------------|--------------------------|---------------------------------|----------------------|----------------------|
| chair | 36 | 8 | ~16 h | ~10 min |
| table | 48 | 10 | ~20 h | ~20 min |
| boat | 124 | 31 | ~62 h | ~2 h |
| dome | 512 | 68 (3D print) 63 (laser-cut) | ~136 h ~3 h | ~8 h (for 2 ppl) |
| bridge | 174 | 30 | ~60 h | ~4 h |
| pavilion | ~1200 | 119 | ~300 h | 6h (for 8 ppl) |

Table 2: Summary of bottle/hub count, printing, and assembly time per example object.

3.7 CONTRIBUTION, BENEFITS, AND LIMITATIONS

TrussFab’s main contribution is the integrated end-to-end workflow that allows users to fabricate large structures that are sturdy enough to carry human weight—on desktop 3D printers. Unlike previous systems that build on up-cycled plastic bottles combined with 3D print, TrussFab considers bottles not as “bricks”, but as beams that form structurally sound node-link structures also known as *trusses*, allowing users to handle the forces resulting from scale and load.

TrussFab embodies the required engineering knowledge, allowing non-engineers to design such structures, and allows users to validate their designs using integrated structural analysis. In particular, TrussFab's editor offers primitives that are trusses (tetrahedra and octahedra), tools that create large beams that are trusses, and tools for tweaking the shape of structures, while maintaining its truss structure. On the mechanical side, we contribute the key structural elements that allow creating trusses, i.e., the 3D-printed and laser-cut hub design.

We have validated our system by designing and fabricating tables and chairs, a 2.5 m bridge strong enough to carry a human (Figure 5), a functional boat that seats two (Figure 9), and a 6 m tall pavilion built of 1200 bottles, shown in Figure 4.

Our approach is subject to the general limitations faced by ready-made objects. In particular, TrussFab can reproduce neither details smaller than a bottle nor closed surfaces. On the structural side, further development could aim at integrating automatic structural adjustments, in case the forces are exceeding the limits of the materials, by implementing algorithms, such as by Arora et. al [3].

4

TRUSSFORMER: HUMAN-SCALE KINETIC STRUCTURES

Large-scale fabrication systems, such as TrussFab, have shown to support a wide range of applications, from furniture to tradeshow pavilions, however, such systems are limited to creating static structures.

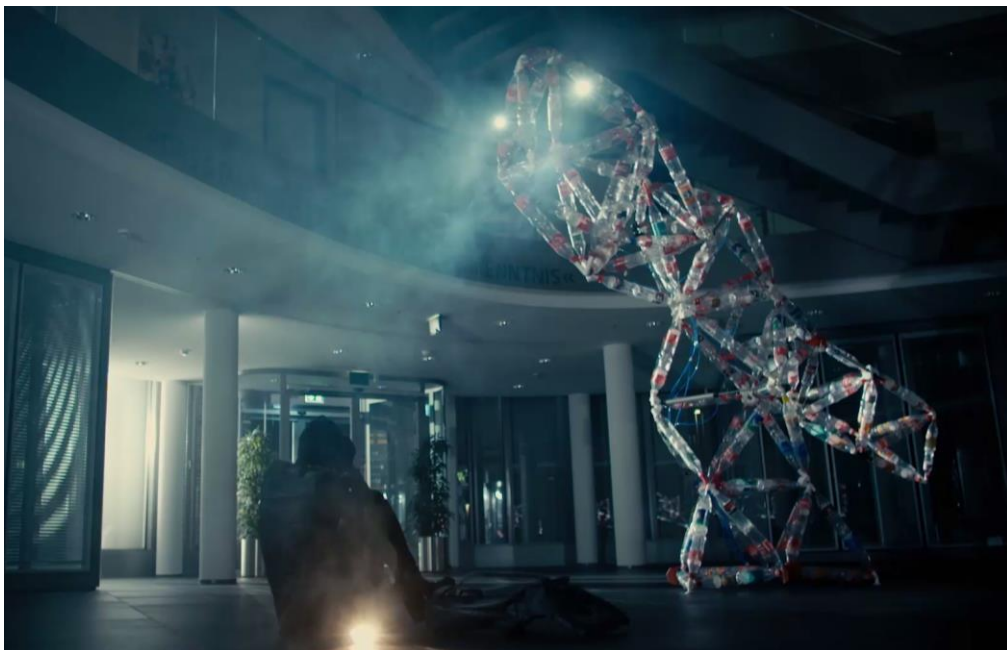


Figure 26: (a) TrussFormer is an end-to-end system that allows users to design and 3D print large-scale kinetic truss structures that deform and animate, such as this 4m tall animatronic T-Rex.

In this section we present a system that allows users to create large *kinetic* structures, i.e., structures that involve motion and deal with *inertial* forces, as they occur in animatronics devices, such as the animated Tyrannosaurus Rex, illustrated by Figure 26, and other large-scale machinery. TrussFormer embodies the required engineering knowledge from (1) creating the appropriate mechanism, (2) verifying its structural soundness, and (3) generating the underlying hinge system printable on desktop 3D printers.

4.1 WALKTHROUGH OF THE TRUSSFORMER SYSTEM

TrussFormer helps users to create the shape and design the motion of large-scale kinetic structures. It does this by incorporating linear actuators into rigid truss structures in a way that they move “organically”, i.e., hinge around multiple points at the same time. These structures are also known as variable geometry trusses [4]. Figure 27 illustrates this on the smallest elementary truss, (a) the rigid tetrahedron. (b) We swap one of the edges with a linear actuator, (c) resulting in a variable geometry truss. The only required change for this is to introduce hubs that enable rotation at the nodes. We call these *hinging hubs*.



Figure 27: (a) The static tetrahedron (b-c) is converted into a deformable structure by swapping one edge with a linear actuator. The only required change is to introduce connectors that enable rotation.

This simple approach to creating variable geometry truss mechanisms scales well to arbitrary larger structures. Our T-Rex model from Figure 26 contains five linear actuators and thus offers five degrees of freedom (DoF). It can (a) lift or lower its neck (1 DoF), (b) turn its head left and right (1 DoF), (c) sweep its tail (2 DoF), and (d) open its mouth (1 DoF), as shown in Figure 28.

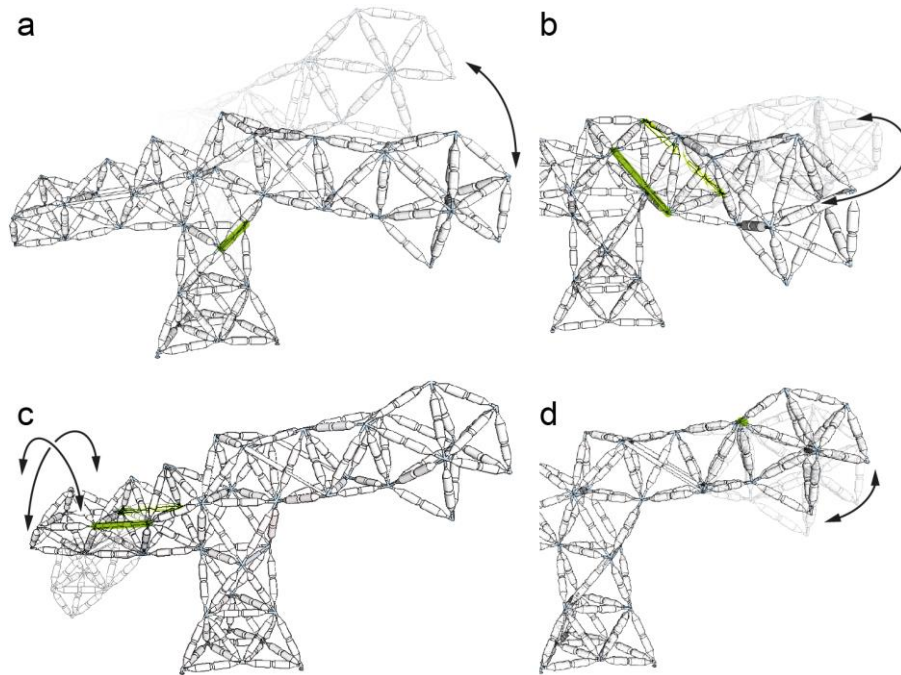


Figure 28: Our T-Rex encompasses 5 degrees of freedom.

In the following, we demonstrate how TrussFormer allows non-expert users to create such structures in six steps.

STEP 1: CREATING A STATIC STRUCTURE

As shown in Figure 29, this particular model was created by first modeling the T-Rex as a static structure in TrussFormer. Our editor's ability to create static structures is based on TrussFab [42]: users design the shape of their T-Rex using structurally stable primitives (tetrahedra and octahedra).

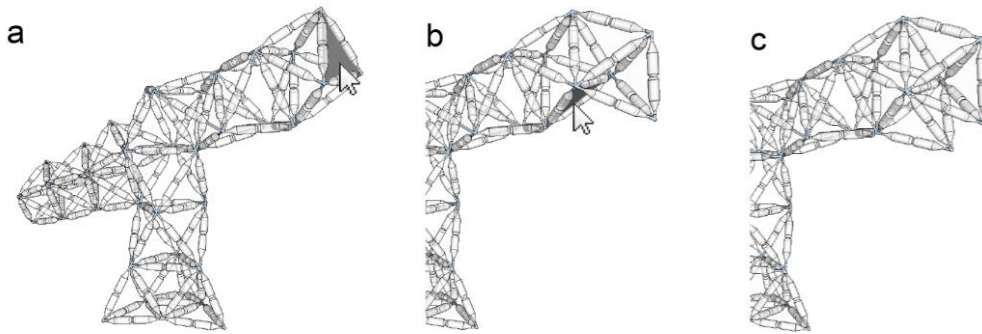


Figure 29: Modeling the static shape of the T-Rex. Here, the user creates the jaws of the T-Rex by attaching tetrahedron primitives through the steps (a, b, c).

STEP 2: ADDING MOVEMENT

To add movement to the static structure, users select the *demonstrate movement* tool and pull the T-Rex head downwards, as shown in Figure 30. TrussFormer responds by placing an actuator that turns the T-Rex body into a structure that organically moves and bends down. Together with the *Demonstrate movement* tool, TrussFormer provides three different approaches to animating structures, ranging from this (1) automated placement (for novice users), through (2) placing elements with predefined motion, called *assets*, to (3) manual placement (as users acquire engineering knowledge). We discuss these in section “4.3 Adding motion to the structure”.

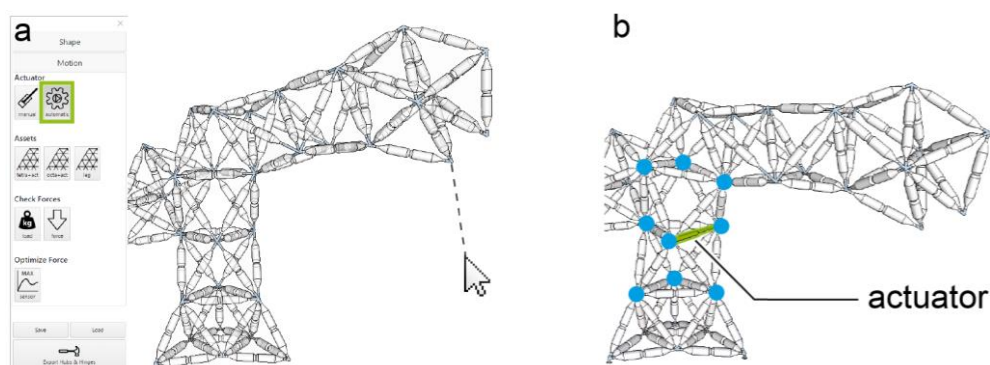


Figure 30: (a) The user selects the *demonstrate movement tool* and pulls the T-Rex head downwards. (b) TrussFormer responds by adding an actuator to the T-Rex body so that it is capable of performing this type of motion. At this point, the system also places 9 hinging hubs to enable this motion (marked with blue dots).

STEP 3: STABILITY CHECK ACROSS POSES

During this step, TrussFormer also verifies that the mechanism is structurally sound. In the background, TrussFormer finds the safe range of expansion and contraction of the placed actuator by simulating the occurring forces in a range of positions. If there is a pose where the forces exceed the pre-determined breaking limits or the structure would tip over, TrussFormer sets the limits for the actuator so it will not extend beyond them. This check prevents users from producing invalid configurations.

STEP 4: ANIMATION

To animate the structure users open the *animation pane* in the toolbar, as shown in Figure 31. First, they control the movement of the structure manually using sliders, to try out the movement. When they find the desired pose, they simply add it as a keyframe to the animation timeline. With this TrussFormer allows users to orchestrate the movement of all actuators using a simple timeline/keyframe editor. In Figure 31 we program a “feeding” behavior, where the T-Rex opens its mouth while reaching down and waving its tail.

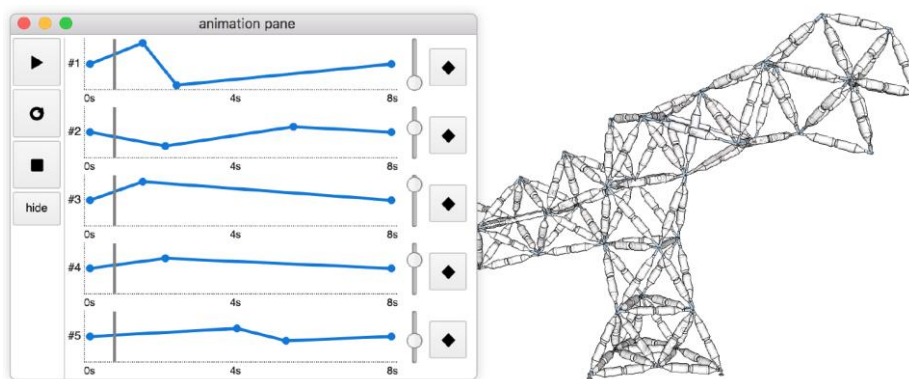


Figure 31: Animating the structure. Users set the desired pose using the sliders in the animation pane and orchestrate the movement by placing key-frames on the timeline.

STEP 5: CHECKING FORCES DURING THE MOTION

Once a movement has been defined, TrussFormer computes the *dynamic* forces. As shown in Figure 32a, the user creates an animation that moves the T-Rex body up and down. (b) TrussFormer computes the forces while T-Rex's body comes back up quickly after dipping down; the large acceleration of the long neck leads to very high inertial forces, exceeding the breaking limit of the construction, (c) causing the structure to fail at the indicated time point. These situations are hard to foresee because the *inertial forces* can be multiple times higher than the static load in the structure. (d) TrussFormer addresses this by automatically correcting the animation sequence by either limiting the acceleration or the range of the movement, assuring that the structure will now withstand the movement.

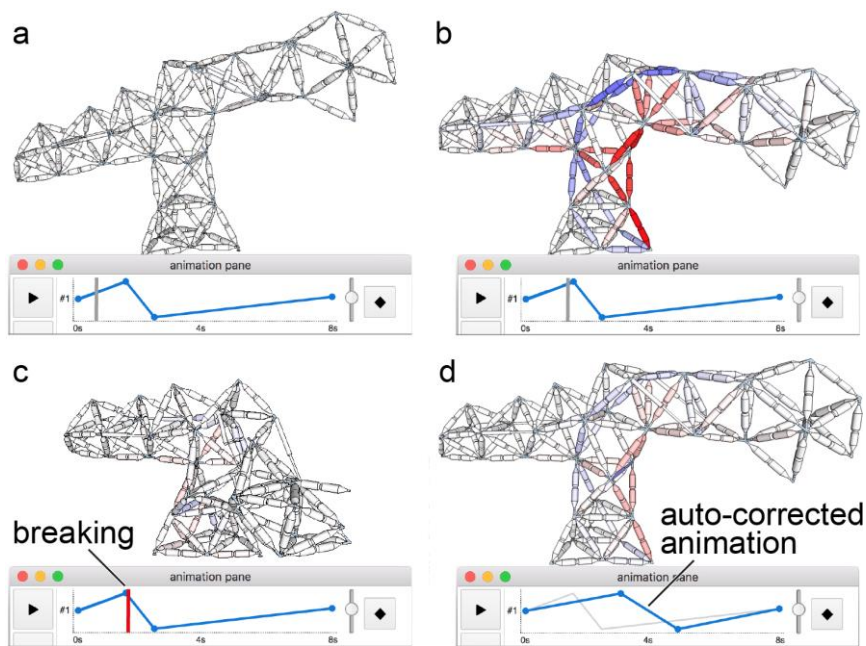


Figure 32: Verifying the inertial forces: (a-b) The forces are increasing with the acceleration of the structure. (c) The structure breaks when the direction of the movement changes rapidly. (d) TrussFormer resolves this by making the movement slower.

STEP 6: FABRICATION

When users are satisfied with their design (structure, movement, and animation), they click the *fabricate* button, shown in Figure 33a. This invokes (1) TrussFormer’s hinge generation algorithm, which analyzes the structure’s motion and generates the appropriate 3D printable hinge and hub geometries, annotated with imprinted IDs for assembly. In the case of the T-Rex, the system exports 42 3D printed hubs, consisting of 135 unique hinging pieces. (2) Next, TrussFormer exports the created animation patterns as a JSON file that can be uploaded for example to an Arduino that controls the mechanism. (3) Lastly, it outputs a specification, containing the force, speed, and motion range of the actuators, in order to achieve the desired animation pattern. Users find these actuators as standardized components.

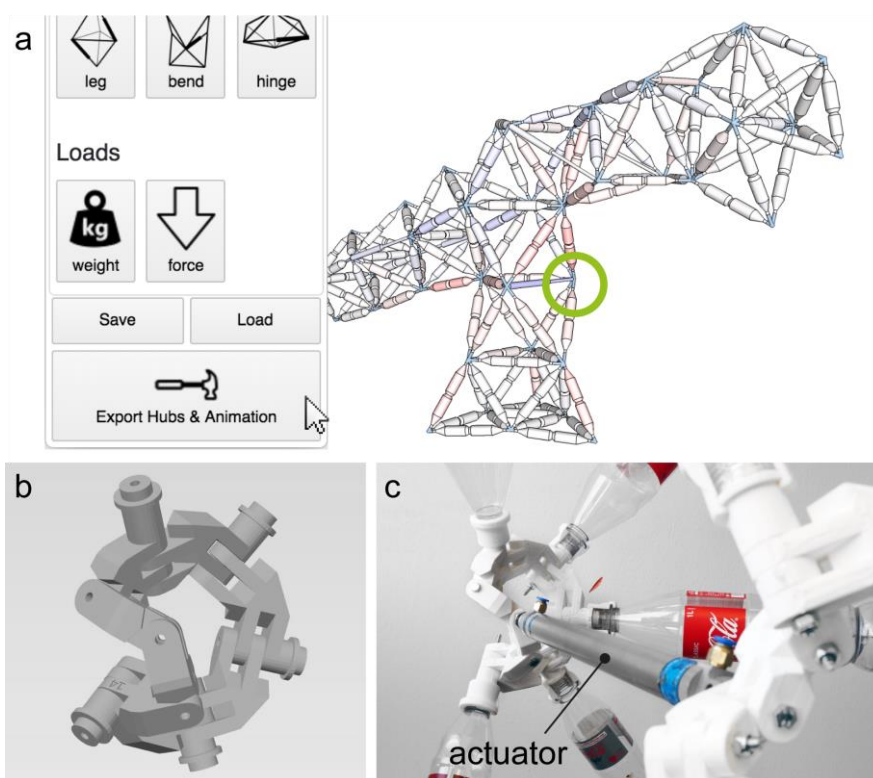


Figure 33: (a) To fabricate our T-Rex model, TrussFormer exports: (b) the appropriate 3D printable hinging-hubs, (c) the specifications for the actuators, and finally the animation sequence as a JSON file for the controller.

4.2 WORKING PRINCIPLE OF TRUSSFORMER'S KINETIC STRUCTURES

Before we discuss how TrussFormer allows users to define the motion path of the structure, we explain the underlying principle how these variable geometry trusses [4] work, where actuators can be placed and how their motion propagates.

A structure created in TrussFormer consists of unit cells, which can be tetrahedra or octahedra. Each cell can contain one or more linear actuators. When actuated, the actuators change the geometry of the cell and thus move the entire structure.

First, as an example, Figure 34 illustrates how inserting an actuator affects only its surrounding. (a) One way of thinking of the actuated tetrahedron is as a rotary hinge, with a triangle face at each side (shaded in gray). (b) Such structures can be extended by attaching a rigid structure to each of the two faces (here two octahedra). As a result, the two structures are hinging around each other. Since the motion is localized, this type of actuator placement is intuitively graspable.

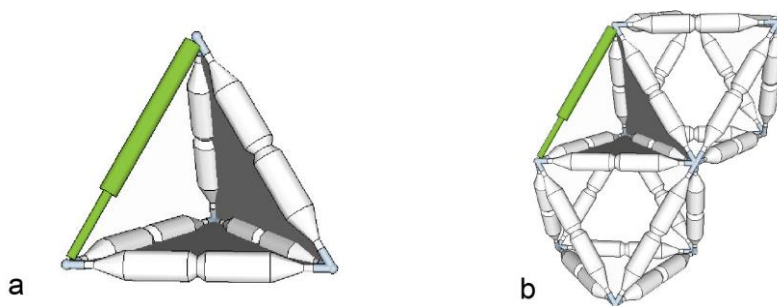


Figure 34: Attaching rigid primitives to (a) faces that do not contain an actuator (b) results in simple structures with only localized deformation. This structure acts as a hinge between the two octahedra.

Second, as illustrated by Figure 35, (a) we can produce more complex kinetic structures by attaching rigid primitives to the faces that contain an actuator (e.g., the one shaded in gray). (b) Now, the newly

placed primitive will also contain this actuator and therefore the result is a structure that moves in whole, resulting in more complex behavior.

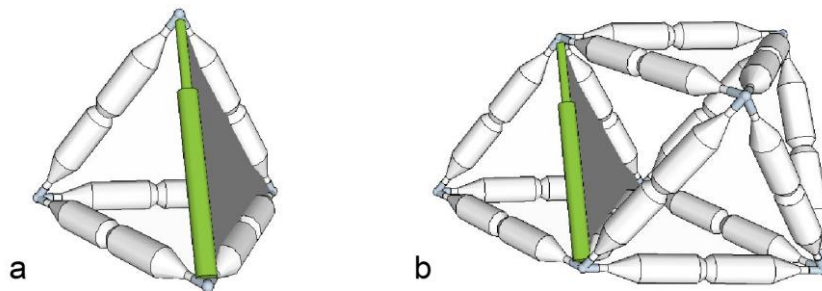


Figure 35: Attaching a rigid primitive (here an octahedron) to a face that contains actuator results in a larger deforming structure.

The third way to propagate motion is to build structures where the cells are interconnected through two or more moving faces. Figure 36a shows an octahedron with one actuator in it. (b) We attach two tetrahedra to the marked faces and place a second octahedron in between them. Since the two original connecting faces are moving with respect to each other, the two tetrahedra are moving as well, causing the second octahedron to deform. The second octahedron require removing one arbitrary edge (here on the top) to allow for deformation. (c) Applying this principle users can propagate the movement of one actuator through the truss.

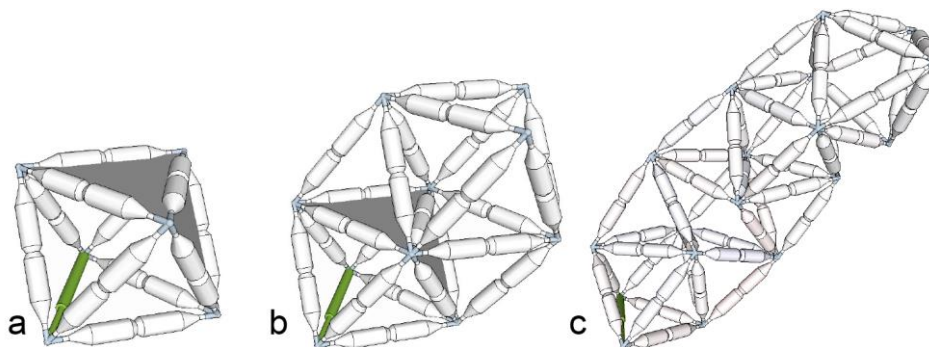


Figure 36: The motion caused by one actuator propagates throughout the entire truss beam, making it bend.

4.3 ADDING MOTION TO THE STRUCTURE

TrussFormer offers three ways for users to animate their structures: (1) by *demonstrating* the desired movement, as we discussed in our walkthrough, (2) using elements with predefined motion, which we call *assets*, and (3) by placing actuators *manually*.

The first two strategies are better suited for novice users, since they do not require knowledge about the mechanism, but rather focus on the shape of the structure and the movement they want to achieve. The third option is best suited for users with more experience, who have already gained a deeper understanding of variable geometry trusses.

4.3.1 AUTOMATIC ACTUATOR PLACEMENT BY DEMONSTRATION

As we briefly discussed in the walkthrough section and in Figure 30, TrussFormer enables users to create moving structures by offering automatic actuator placement. Users can focus on only designing the shape of their structure first. Then, they invoke the *demonstrate movement* tool and drag the static structure in the direction they want it to move. TrussFormer then replaces the edge with an actuator at the position which best satisfies the movement.

To identify which edge should be replaced with an actuator TrussFormer runs an exhaustive search by virtually replacing every member with an actuator one by one. At every replacement, it moves the actuator while measuring if the structure moved closer or further to the desired target position. Finally, it compares all the results and selects the actuator that produced the closest motion. A limitation of this simple method is that it works by naïve approximation, i.e., that it does not guarantee that the desired position will be exactly reached. To improve these results, further optimization algorithms can be utilized, similarly as demonstrated by Coros et al. [19] for planar mechanisms.

4.3.2 CREATING KINETIC STRUCTURES BASED ON ASSETS

Because the resulting motion of variable geometry trusses tends to be hard to predict, TrussFormer encapsulates them into predefined sub-assemblies, which we call *assets*. Assets connect to the existing geometry through a dedicated triangle surface. This results in structures that contain the asset's movement which is localized and thus easy to understand.

Figure 37 shows a selection of assets. The triangles marked in gray are their connectors, i.e., the side that connects to existing geometry when the asset is added to a structure. TrussFormer offers a basic selection of assets, however, users can easily create their own asset library by saving a custom asset into an asset folder.

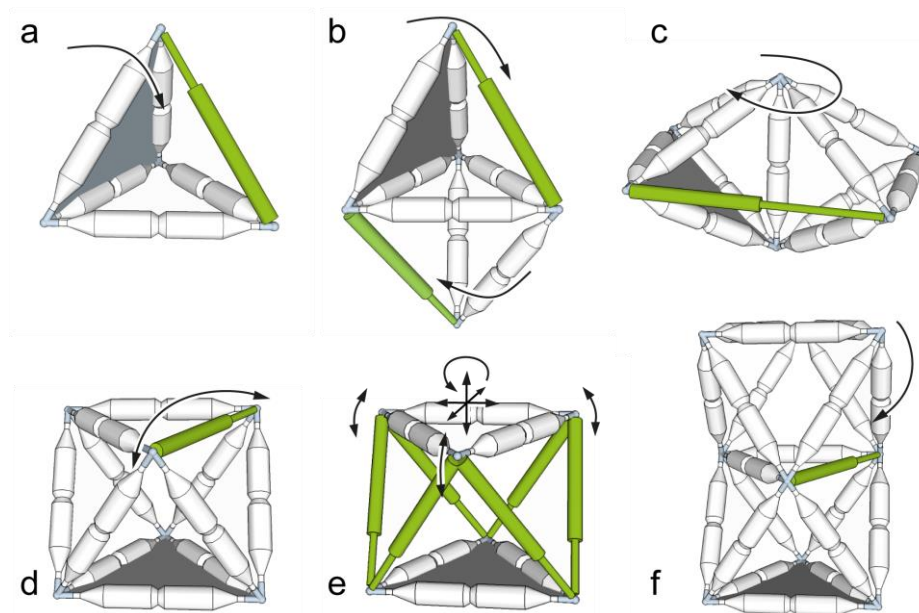


Figure 37: A selection of assets: (a) tetrahedron with 1DoF, (b) "robotic leg" asset, (c) hinging tetrahedra, (d) octahedron with 1DoF, (e) Stewart platform (6DoF), and (f) double-octahedron performing "bending" motion.

Figure 38 illustrates the workflow enabled by assets: a simple walking robot with six robotic legs. (a) Users start by creating the rigid body of the robot from tetrahedra and octahedra blocks. They design it to offer six connector faces, i.e., three on each side, (b) where they attach copies

of the *robotic leg* asset, shown in Figure 37b. (c) This results in an autonomous walking structure.

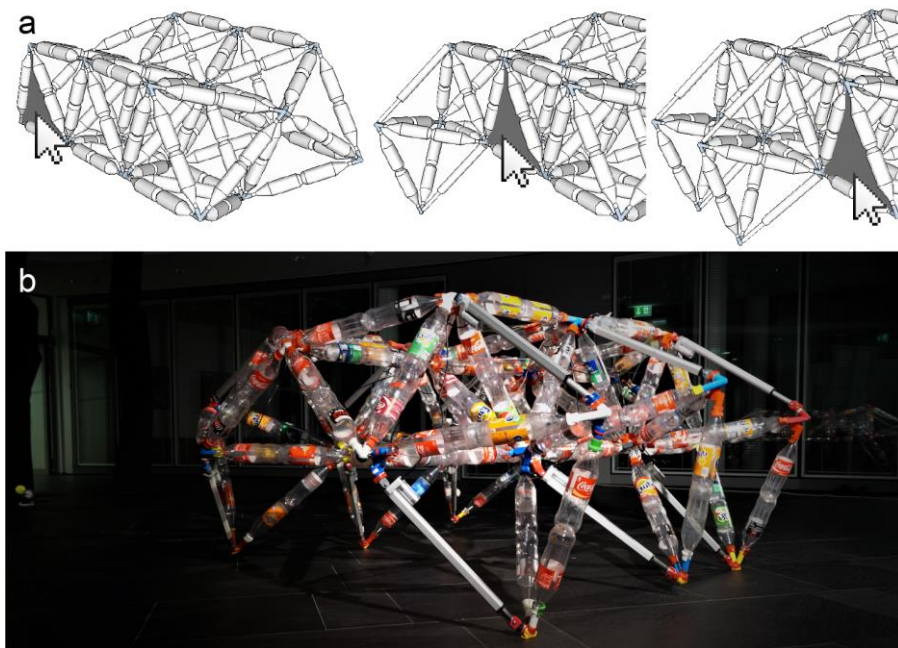


Figure 38: (a) Users start the design by making the body of the walking robot. The predesigned 2DoF “leg” asset is added to the side triangles 6 times. (b) The fabricated robot.

The concept of assets is useful beyond the use of actuators. Figure 39 for example, shows a bike we designed around the *hinge* asset that forms the steering column.

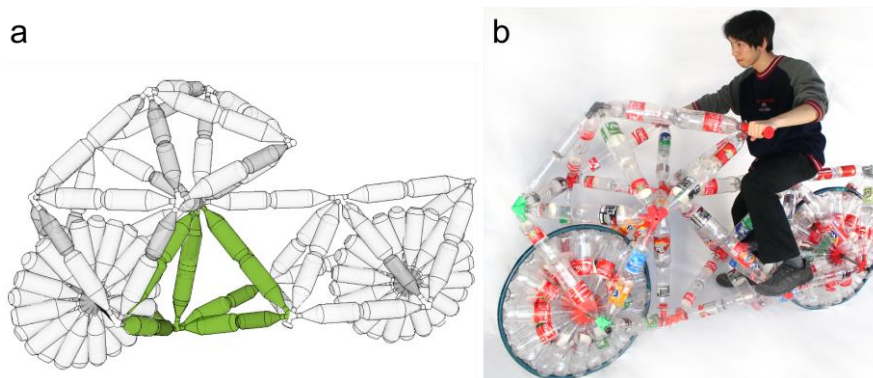


Figure 39: This bike’s steering column is based on the hinge asset, which is used without the actuator in this example.

4.3.3 MANUAL ACTUATOR PLACEMENT

As users gain expertise in creating variable geometry trusses, they may prefer to place actuators directly into their structures. TrussFormer’s *turn edge to actuator* tool allows users to transform rigid edges into actuators by simply clicking on them, as illustrated by Figure 40.

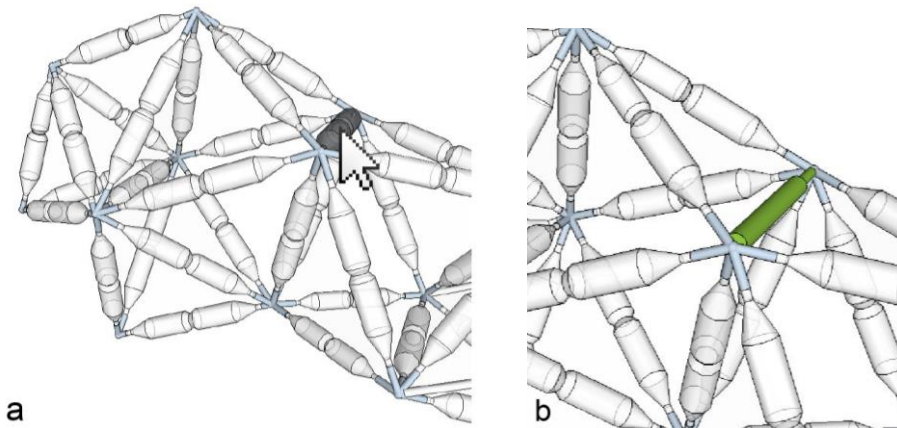


Figure 40: The “*turn edge into actuator*” tool allows users to turn any edge into an actuator. Here user replaces one edge in the T-Rex’s head to make its jaws move.

We designed this tool deliberately as a “turn existing edge into actuator” tool and not as a “place new actuator” tool. Normally, placing a new actuator edge into an already rigid structure would not allow for movement, however, by turning an existing edge into an actuator, users are essentially adding a degree of freedom to the structure.

4.4 VERIFYING AND ADAPTING FORCES

Our system helps users to create the shape and the motion of large-scale kinetic structures. To accomplish this, it helps users handle the dynamic forces that occur when large structures move, such as the T-Rex in Figure 26.

TrussFormer enables users to (1) constantly monitor the forces that occur within the structure at interactive rates. Furthermore, it (2) validates the poses of the structure and adapts the motion range of

the actuators to not damage the model, and (3) automatically adapts the user-defined animation sequence in case it breaks the structure.

To perform these tasks, TrussFormer takes into account the breaking limits of the building materials. The model is considered broken when the simulated peak stress value exceeds the entered breaking limit of the building material. We acquired these values from fracture testing the materials, in our case the plastic bottles, as described in TrussFab [42], resulting in max. 85 kg compression and max. 135 kg tension. If users decide to use different building materials, we recommend testing the forces these elements can withstand again. However, we expect users to share this information on platforms such as *thingiverse.com*.

4.4.1 CONTINUOUSLY VISUALIZING FORCES DURING ANIMATIONS

While the user animates the structure, TrussFormer is continuously simulating the forces using its built-in physical simulation. The forces are visualized as colored edges: red indicates compression, blue indicates tension, while saturation signalizes the intensity of the force.

This allows users not only to preview artifacts that arise from their current animation, e.g., the structure wobbling too much due to rapid changes from pose to pose; but, more importantly, it allows them to preview how the stress is distributed in the structure and even foresee breaking points when rapidly actuated.

4.4.2 VALIDATING POSSIBLE POSES

After users have placed an actuator in their structure, TrussFormer automatically determines their motion range, i.e., how far can it expand without damaging the structure.

Figure 41 shows that the structure can break due to various causes, such as the structure falling over, hitting the ground from too high of a movement allowance, or simply exerting too much force on another structural element (e.g., an edge).

To determine the limits of an actuator, TrussFormer iteratively increases the expansion until the simulated model breaks. TrussFormer then stores the previous valid expansion as the maximum length for that actuator. This value is then set as the upper bound for the motion in the keyframe editor. This way the user is never able to over-actuate them.

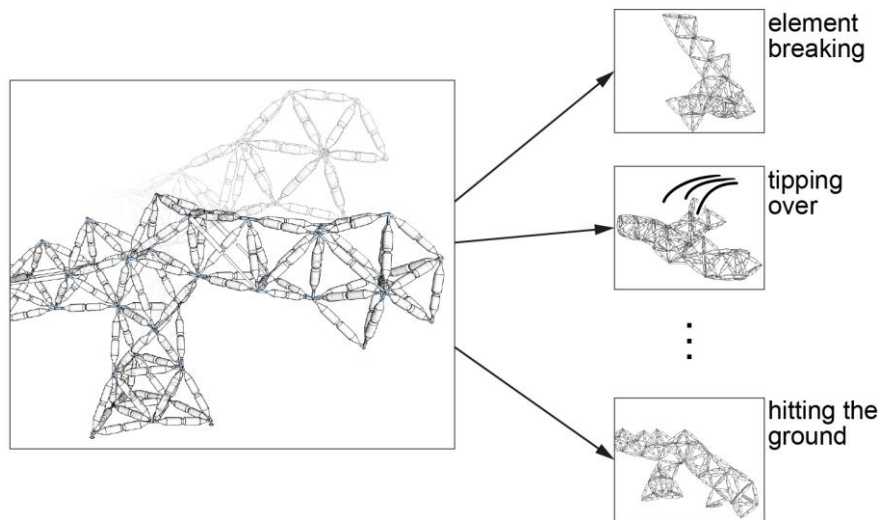


Figure 41: In the background, TrussFormer tests each actuator to see if its extension leads to an invalid position, such as the structure tipping over, hitting the ground, or breaking any structural elements.

This check is performed for each actuator individually. While a full factorial cross-check would be necessary to detect damaging interaction effects, unfortunately, such an exhaustive search does not scale well with the increasing number of actuators and would deteriorate the software's interactivity. Therefore, TrussFormer still checks if the structure breaks in the later animation step and offers automatic correction.

4.4.3 AUTOMATICALLY ADAPTING FORCES

After users create an animation sequence using the keyframe editor, shown in Figure 42a, TrussFormer continues to validate if the structure can withstand user-defined accelerations.

As we previously demonstrated in Figure 32, TrussFormer predicts that the T-Rex breaks if its neck is actuated too rapidly between a neck-

down and a neck-up pose. This happened due to the large inertial forces. Since the structure is large, its mass is large as well. Forces that act on the elements of the structure increase proportionally with the acceleration of the movement ($F = ma$). While the mass is a constant in the structure, the acceleration is what TrussFormer can alter to prevent it from breaking. When the model breaks in the simulation, TrussFormer offers two options to reduce the occurring inertial forces, as shown in Figure 42.

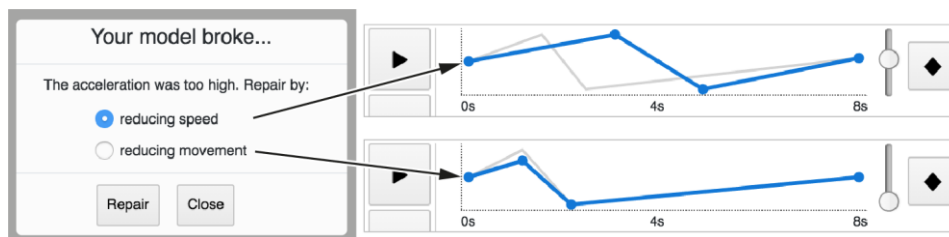


Figure 42: If the user-defined animation breaks the model, TrussFormer offers to automatically reduce the speed or the motion range.

TrussFormer offers to fix the animation slopes in two ways: (1) by reducing the speed of the motion, i.e., by stretching the time of the animation, or (2) by reducing the range of the movement. TrussFormer finds the valid actuation profiles by simulating the structure in the background and gradually reducing the speed or the extension of the actuation, depending on the users' choice.

The predicted force values during the simulation are also used to inform users about the properties of the actuators they need to buy to fabricate their structures, i.e., the minimum force that actuators must exert and the speed set in the animation. This force is defined as the maximum force that we measure during the simulation while the structure is performing the programmed animation.

4.5 MATCHING SIMULATED AND REAL FORCES

To match the predicted forces calculated by the physics simulation engine, we measured the real forces occurring in our T-Rex model and tuned our simulation based on these empirical measurements.

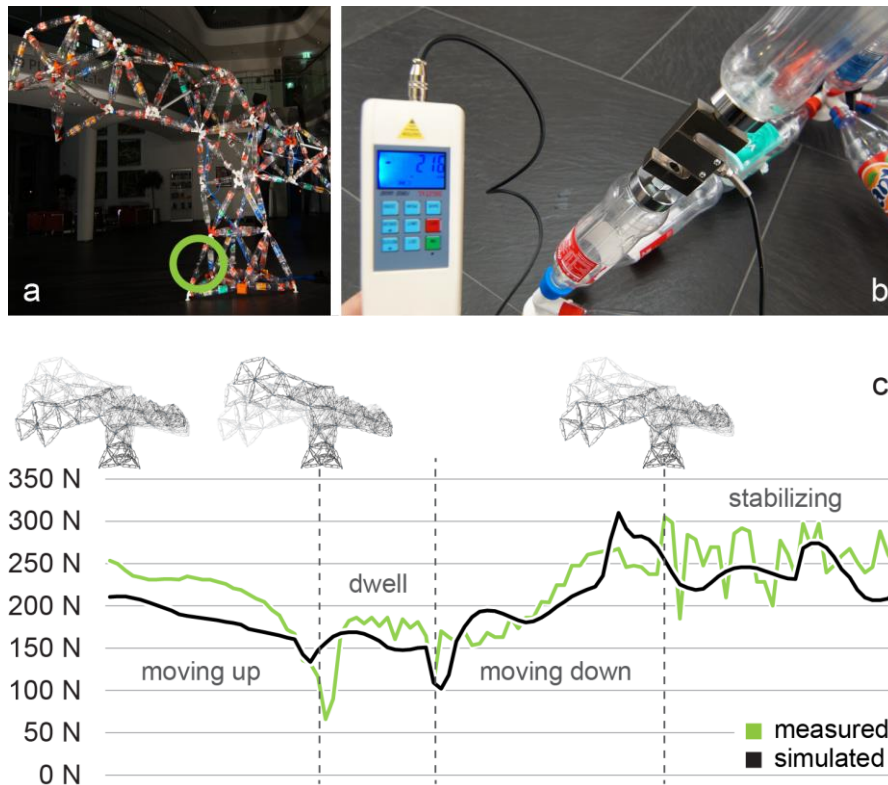


Figure 43: (a) We measured the forces on the bottom front edge of the T-Rex (b) using a digital force gauge. (c) The measured forces agree with the simulated forces.

As Figure 43a-b shows, we inserted the external force sensor (capacity: 5000 N, error: 0.5%) between two bottles at the bottom of the T-Rex structure. We chose this element as it bears the largest forces. We then actuated the T-Rex to move its entire head up to its highest position and down again to its lowest position. Figure 43c shows the measured and the simulated forces, in agreement with the forces we measured. We acknowledge the differences due to the fabrication imprecision, such as slack in the joints and friction.

4.6 TRUSSFORMER'S HINGE SYSTEM

A key element behind TrussFormer's kinetic structures is the 3D printable hinge system, that enables multiple edges to spherically pivot around a node point. While traditional ball joints allow for spherical motion, they are limited to connecting only two edges. To address this shortcoming, TrussFormer uses the generic design of a *spherical joint mechanism* [13], shown in Figure 44a, that allows for multiple edges to pivot around the same center point, as they were connected via a ball-joint. Figure 44b shows TrussFormer's rendering of the spherical joint mechanism, adopted for 3D printing.

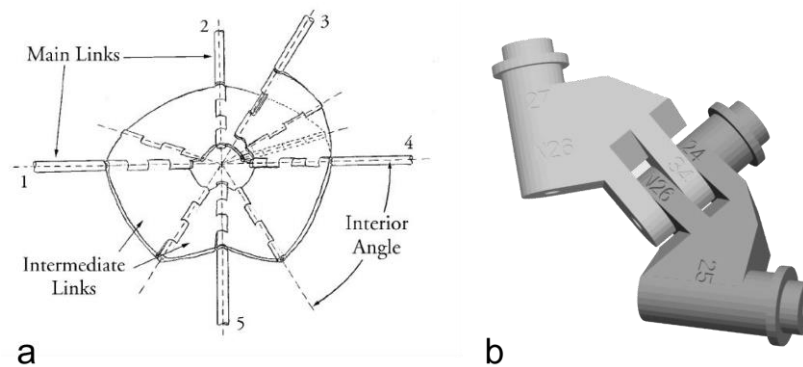


Figure 44: (a) Spherical joint mechanism from [13] connecting 5 edges. (b) A segment of TrussFormer's 3D printable hinge design.

To achieve the motion that users designed, TrussFormer arranges the necessary spherical joints automatically in the structure. Traditionally, determining the required mobility of the joints is done by evaluating the Grübler–Kutzbach mobility criterion. However, this analytical approach is hard to fit for spatial (i.e., 3D) parallel mechanisms, and it's still subject to active research [50]. Therefore, instead of attempting an analytical solution to this problem, TrussFormer tests the motion of the user-defined structure by using its built-in physical simulation and arranges the hinges heuristically. In the following, we describe TrussFormer's four-step hinge placement routine on the example of an octahedron with one actuator, shown in Figure 45a.

STEP 1: VIRTUALLY ASSIGNING ALL POSSIBLE HINGES

As a first step, TrussFormer assigns the intermediate link connections of the spherical joint mechanism, from Figure 44a, between all the edges forming a triangle in the structure, as illustrated with blue lines in Figure 45b. This provides 2DoF to all the edges, as they were connected via ball joints. This already gives a mechanically satisfying solution, however, it can be further optimized. In variable geometry truss structures, most of the edges are confined in triangles and larger rigid substructures, therefore not all movements are possible. Placing unnecessary hinges only adds complexity to assembly and reduces mechanical stability.

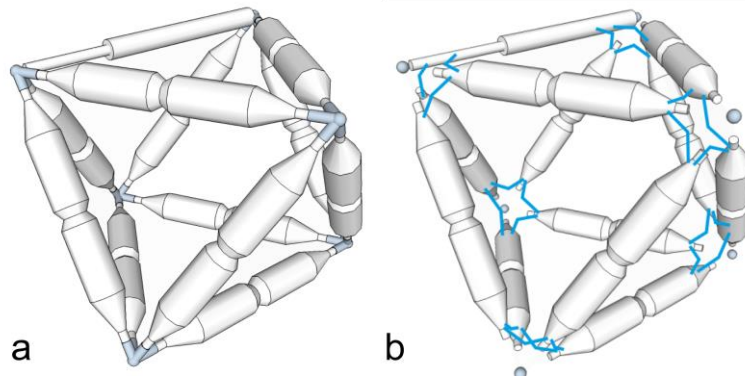


Figure 45: (a) Octahedron with an actuator, still with rigid hubs. (b) After the first step, intermediate links are assigned inside all triangles, creating spherical joints.

STEP 2: IDENTIFYING RIGID SUBSTRUCTURES

To identify rigid substructures in the structure, TrussFormer now runs the physical simulation and moves all the actuators simultaneously. It observes the angular movement between the edges and if the angle between two or more connected edges never changed, TrussFormer considers them as a *rigid substructure*. Figure 46a shows the rigid substructures identified in the octahedron, visualized in distinct colors. The triangles containing the actuator are not considered rigid, since their internal angles are changing. Figure 46b shows the result of this step on the example of the T-Rex. Here, the rigid substructures consisting of single triangles are left uncolored, for clarity.

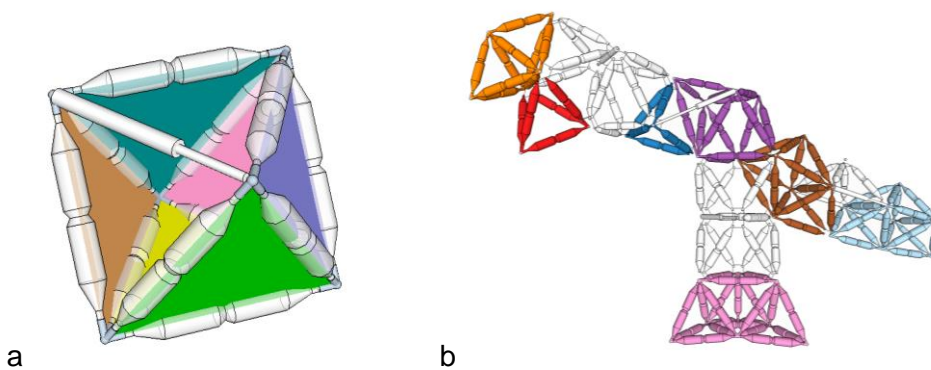


Figure 46: TrussFormer identified the rigid substructures (a) in the octahedron from Figure 45, and (b) in the T-Rex.

STEP 3: REDUCING THE EXCESS OF HINGES

Now that TrussFormer knows which parts of the structure are rigid, it can remove the unnecessary hinges between the edges which belong to the same rigid substructure and don't move in regards to each other. In Figure 47 we show this step on our octahedron example. Between the edges forming rigid substructures, the intermediate link connections (before blue lines) are reduced to rigid connections (black lines). Rigid substructures will still rotate with respect to each other. At this stage, the final hinge chain is already found for the octahedron example and the resulting 3D print is shown in Figure 47b.

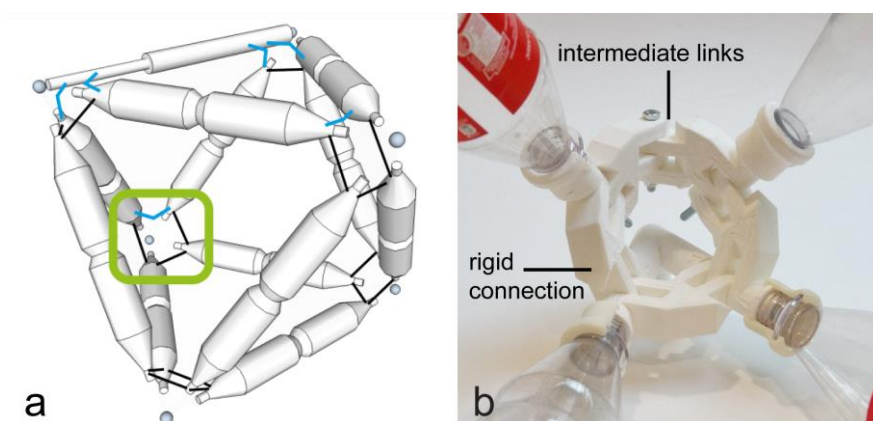


Figure 47: (a) The intermediate hinge connections are reduced to rigid connections where rigid substructures are identified (black lines). (b) The fabricated hinging hub of the marked node of the octahedron.

STEP 4: RESOLVING IMPOSSIBLE CONNECTIONS

At this point, TrussFormer has already assigned an optimized valid hinge configuration, however, not all the connections might be physically possible to physically assemble. TrussFormer's hinge design has the limitation that it only supports one-on-one hinge connections, as shown in Figure 44b. Three-way connections are not possible, i.e., three parts cannot physically hinge around the same axis.

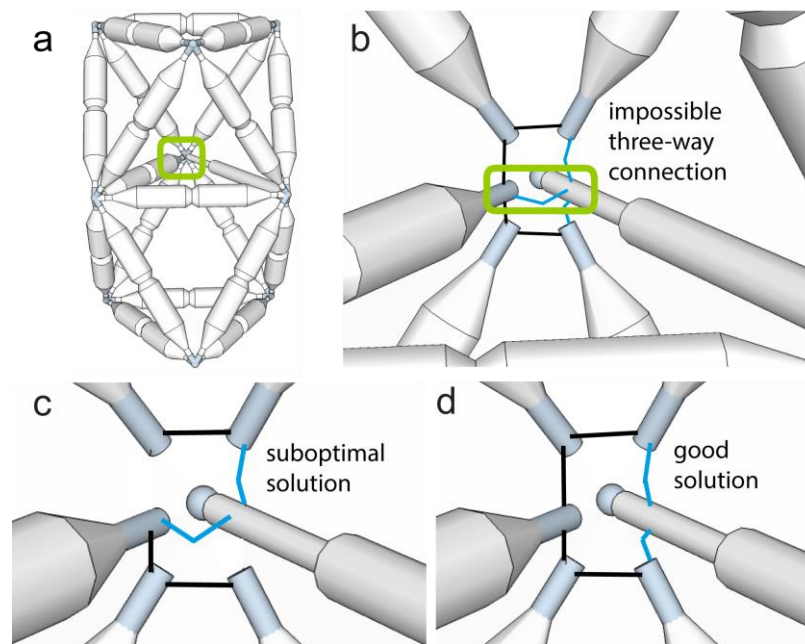


Figure 48: (a) Double-octahedral structure, where (b) violating three-way hinge connections appear. (c-d) TrussFormer finds the valid configurations by heuristic elimination and (d) chooses the structurally more stable closed chain.

However, after Step 3, there might be hubs violating this condition, e.g., where three hinges are meeting at the same axis. We demonstrate this case in Figure 48 on the example of the double-octahedra structure with one actuator, similar to the one found in the body of the T-rex. In Figure 48b we highlight the hub where three-way hinge connections are present after performing Step 3. Fortunately, these connections are redundant in TrussFormer's kinetic structures, and they can be resolved by eliminating some of the hinges, while still maintaining the hub's

structural integrity, i.e., all the edges remain interconnected via a continuous hinge chain.

TrussFormer resolves violating connections using a backtracking algorithm that removes connections heuristically. After each removed connection, it checks the validity of the resulting hinge configuration for two constraints: (1) all edges around the node are still interconnected directly or indirectly with each other, and (2) no more than two hinges are connected at each axis. If these constraints are satisfied, a valid hinge configuration was found. The algorithm continues until it finds all valid configurations. If available, TrussFormer will select the configuration with a closed hinge chain (Figure 48d) over an open chain (Figure 48c), for stability reasons. The fabricated hinge for this example was shown earlier in Figure 33b-c.

TrussFormer's hinging-hubs can also be combined with TrussFab's rigid hubs. Such an example is shown in Figure 49, where a node contains four rigidly connected edges combined with two hinging connections.

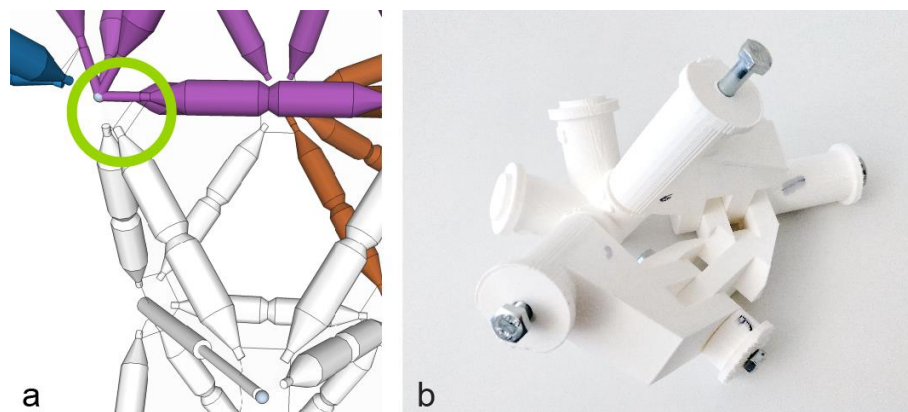


Figure 49: Combining rigid and hinging connections in one hub.

STEP 5: GENERATING THE HINGE GEOMETRIES FOR 3D PRINTING

After determining where the hinges should be placed in the structure, TrussFormer has all the necessary information to export the 3D printable geometries in the form of OpenSCAD [137] files. These files contain

information about the angle and connector lengths of the hinging pieces, as well as their imprinted IDs (as visible in Figure 48a). Users assemble the hinging hubs by matching the corresponding IDs. These IDs also contain information about the placement of the actual hub within the structure, the IDs of the neighboring hubs, and the bottle type to be inserted.

4.7 IMPLEMENTATION

To help readers replicate our results, we now describe the implementation of the main components of the TrussFormer software system and the hardware we used to actuate our prototypes.

4.7.1 SOFTWARE SYSTEM

TrussFormer builds on TrussFab’s for SketchUp [142], extending it with movement-specific tools. TrussFormer’s system consists of three core components, the 3D geometry exporter, the hinge placement routine, and the simulation with force analysis, as illustrated in Figure 50.

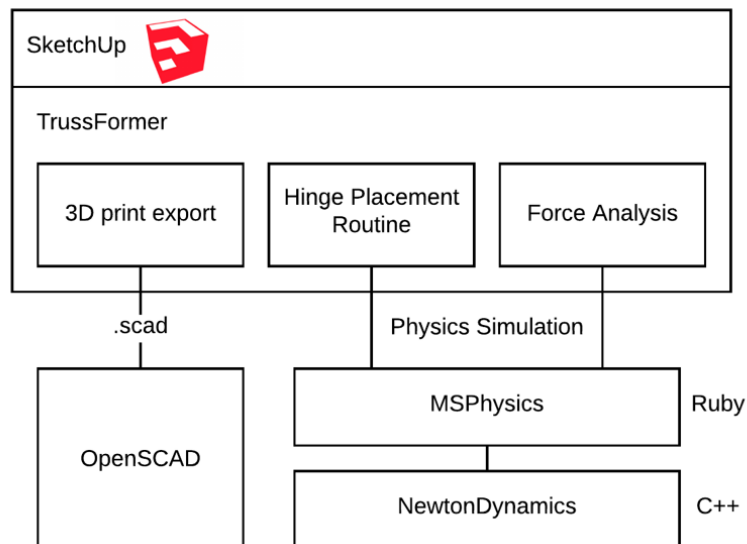


Figure 50: TrussFormer’s system diagram.

To simulate the movement and the force distribution in the 3D model, we use the physical simulation engine MSPhysics [93], a Ruby wrapper for the C++ physics library Newton Dynamics [136]. To achieve interactive performance, the only simulated components are the hubs, the edges are just animated on the scene. The hubs contain all the necessary information, such as weight, breaking force, and stiffness determining how much hubs can move in relation to their neighbors. User interface elements (e.g., the control or the animation pane) are displayed in a SketchUp-integrated Web Browser View. We implemented the UI in HTML and JavaScript to take advantage of UI frameworks such as React [138].

4.7.2 CONTROL SYSTEM AND ACTUATORS

Figure 51 shows the hardware we use to actuate our T-Rex example. We use pneumatic actuators interfaced with proportional valves (Festo VPPE and MPYE series) that are controlled by an Arduino Nano. The pneumatic cylinders have diameters from 25 to 35 mm and produce forces between 390 N and 770 N. We use an *Airpress HL 360* compressor that can provide up to 8 bar of pressure.

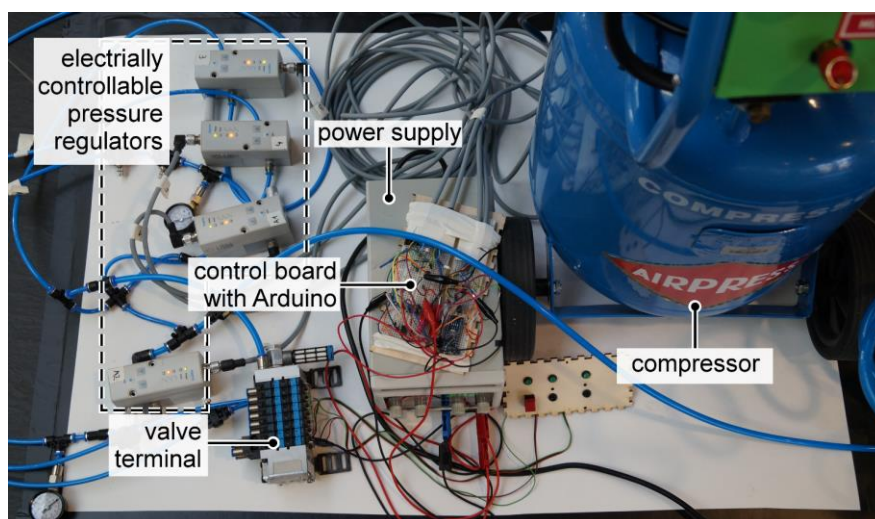


Figure 51: Hardware setup for controlling the T-Rex with an Arduino, electronic pressure control valves, and a compressor.

Our spider example in Figure 38 uses electric linear actuators similar to those found in garage doors. These actuators have a motion range of 45 cm and move rather slowly: 0.03 m/s compared to the speed of a pneumatic actuator that can reach 20 m/s.

4.7.3 BUILDING MATERIALS

For creating our models, we used refillable soda bottles and 3D printed the hubs on an Ultimaker 3 3D printer using PLA material. To increase stability, we set a 3mm wall thickness for our hubs. While the 3D printing process is rather time-consuming (5-8 hours/hub) the assembly of the hubs is quite fast (10-15min/hub). The overall structure is assembled in a reasonably short time; our T-Rex took approximately 1-2 hours for 3 persons.

We use plastic bottles as a building material as they are ecologically friendly and commonly available all around the world. However, TrussFormer also supports any other type of building materials. Users only need to create and copy the 3D models of their material primitives into TrussFormer's material library folder.

To create more realistic-looking animatronic creatures, users can also cover the structure with stretchable textile or other materials and attach smaller features (e.g., ears, fingers, etc.) using 3D printing or other fabrication techniques.

4.8 CONTRIBUTION, BENEFITS, AND LIMITATIONS

TrussFormer is an end-to-end system that enables novice users to design and build large animated structures, that would otherwise be privilege of industry, such as movie sets or theme parks.

TrussFormer helps users in the three main steps along the design process. (1) It enables users to animate large truss structures by adding linear actuators to them. It offers three tools for this purpose: manual actuator placement, placement of assets performing a predefined

motion, and creating motion by demonstration. (2) TrussFormer validates the design in real-time against static forces, static forces across all poses, and dynamic forces. (3) TrussFormer automatically generates the necessary 3D printable hinges for fabricating the structure. Its algorithm determines the placement and configuration of the hinges and their exact geometry.

To validate our system, we created a series of example objects, including a 6-legged walking robot and a 4m-tall animatronics dinosaur with five actuators, comprising 17 static and 25 hinging hubs.

TrussFormer is subject to the following limitations: (1) TrussFormer's simulation relies on the Newton Dynamics physics engine [136], which offers only limited accuracy for engineering purposes. Higher precision could be achieved by replacing Newton Dynamics with a scientific-grade physics engine (e.g., *Vortex Studio* [143] or *Algoryx Momentum* [119]). (2) When deployed, TrussFormer should be provided with additional safety features, such as the option to use stronger materials and additional safety margins in the computation.

5

TRUSSCILLATOR: HUMAN-POWERED HUMAN-SCALE DEVICES

The related work in personal fabrication offers numerous examples for creating so-called *kinematic systems* [53], that allow users to design and fabricate mechanisms that perform user-specified movement patterns. Examples include the 3D-printed pantograph from *Metamaterial Mechanisms* [34], the animatronic T-Rex from *TrussFormer* [43], and the animated cheetah by Coros et al. [19], reproduced in Figure 53a.

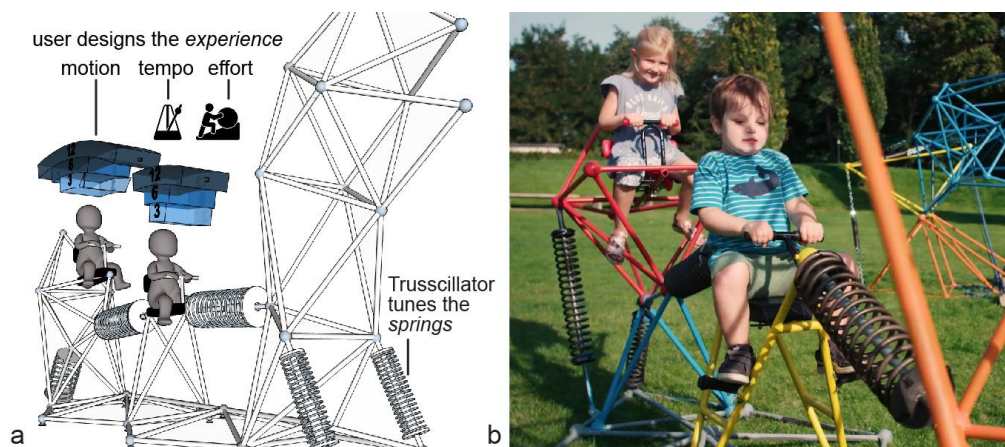


Figure 52: (a) Using Trusscillator designers specify the desired motion experience in terms of amplitude, speed, and physical effort; while the system responds by adjusting the coil springs so as to produce the desired behavior. (b) Here, the resulting interactive dinosaur swing requires two children to synchronize their movement so as to make the sculpture's head wiggle.

We build on this line of work and extend it 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 where *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 “5.3 Expert interviews”).

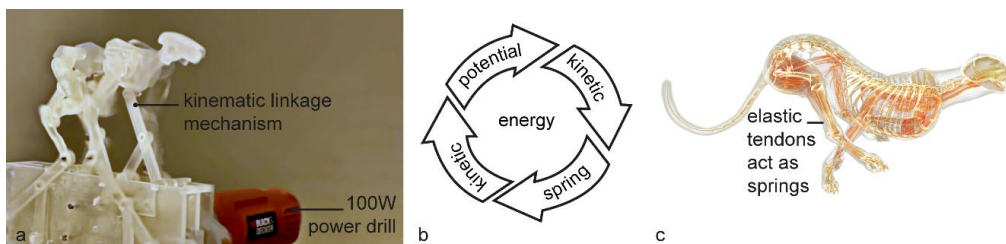


Figure 53: (a) The cheetah mechanism created using [19] is only resembling the movement pattern of a real one, without considering the forces involved during motion. While (b) energy conservation makes a real-life cheetah’s* gallop efficient: (c) the elastic tendons store and release energy in every step.

In this chapter, we describe Trusscillator, a software system that enables users to create human-scale, *human-powered* machines, such as the playground equipment shown in Figure 52. Trusscillator achieves this by allowing users to add *springs* to their designs. Springs have the ability to transform movement (kinetic energy) into compression (potential energy) and transform that back into movement, as illustrated in Figure 53b. Consequently, springs keep devices, typically referred to as *dynamic*

* <https://www.dkfindout.com/us/animals-and-nature/cats/inside-cheetah>

systems [58], 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 53a, but instead bear more resemblance with an *actual* cheetah, which also uses springs (called *tendons*) to run efficiently [55] (Figure 53c).

To allow designers to create human-powered movement, Trusscillator offers a novel set of tools, specifically designed for designing dynamic experiences (Figure 52a). 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. Since the forces involved in the resulting devices can be high, devices designed using Trusscillator are made *from steel*, as shown in Figure 54. Trusscillator helps users fabricate from steel not only by picking out appropriate masses and springs but also by (a) producing stencils, (b) placing temporary connectors that help (c) pre-assembling the structure for (d) welding.

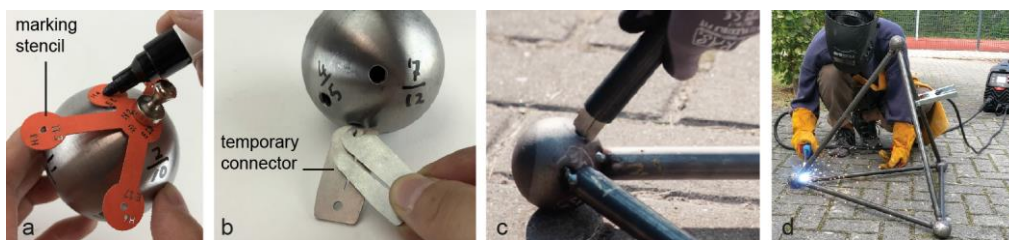


Figure 54: Given the scale of the involved forces, the structures created by Trusscillator are made from steel. Trusscillator supports steel truss fabrication by (a) generating stencils that show where to attach the (b) temporary connectors, (c) that hold steel rods in place for (d) welding.

5.1 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 52. 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.

5.1.1 DESIGNING A BRACHIOSAURUS SWING FOR TWO

As shown in Figure 55a, the playground designers start by creating a rigid dinosaur sculpture by stacking truss-primitives, specifically tetrahedra, and octahedra (building on *TrussFab* [42]). 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 deleting truss elements.

As illustrated by Figure 55c, 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.

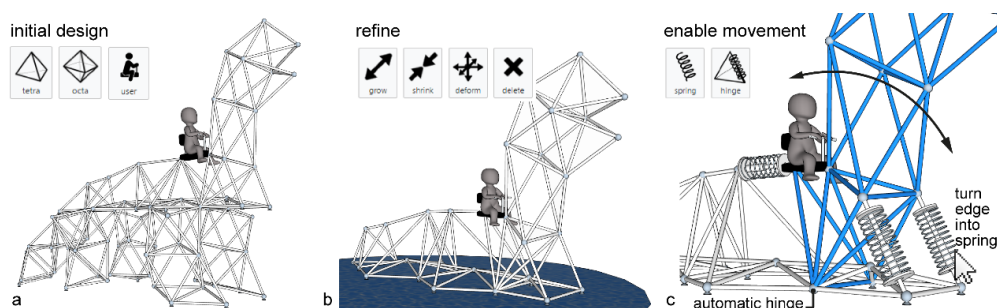


Figure 55: (a) The initial design of the brachiosaurus playground object is created using rigid truss primitives. (b) Designers adjust the shape and lower the height for safety reasons. (c) Using the spring tool, designers enable parts of the model to move. The newly created moving part of the model gets briefly highlighted in blue.

As illustrated by Figure 56a, 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.

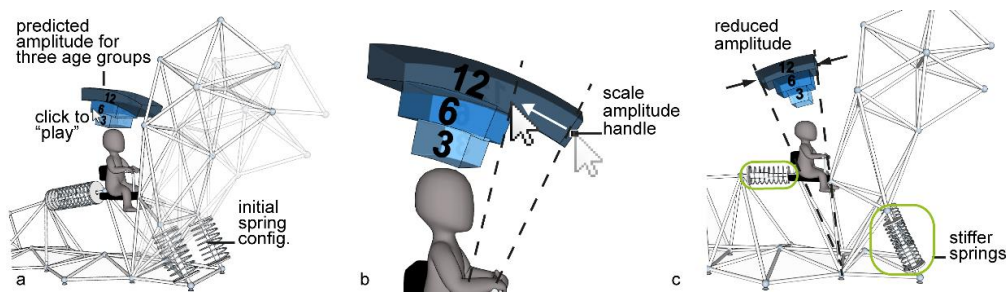


Figure 56: (a) Trusscillator initiates the model with a valid spring configuration. The resulting oscillating motion is summarized in form of a *motion-bar* above the user, calculated for multiple age groups. (b) When designers enlarge the motion space by dragging the *scale* handle, (c) Trusscillator finds a combination of softer springs that will produce the requested amplitude.

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 to reach down far enough to hit someone. As shown in Figure 56b, the designers reduce the device’s amplitude by grabbing the handle attached to the *motion bar* and dragging it inwards. Trusscillator responds by re-running its optimization engine and replacing the springs in the model with springs that produce motion in the request range (Figure 56c).

The reduction in amplitude has now caused the ride to oscillate faster (0.6s period, indicated on hover). As shown in Figure 57a, 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 57b.

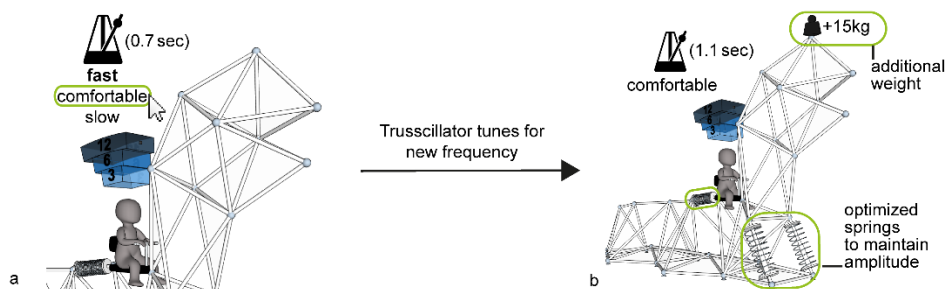


Figure 57: (a) When designers change the tempo widget from *slow* to *comfortable* Trusscillator runs its optimization and (b) adds additional weight to the tip of the head to reduce the resonant frequency and tunes the springs again to maintain amplitude.

At this point, designers notice a third concern: the *effort widget* suggests “*laborious*” (Figure 58a). 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 of the designers proposes clicking the effort widget to reduce the effort (see section 5.4.5), 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 in Figure 58b, they add a second seat and yet another spring.

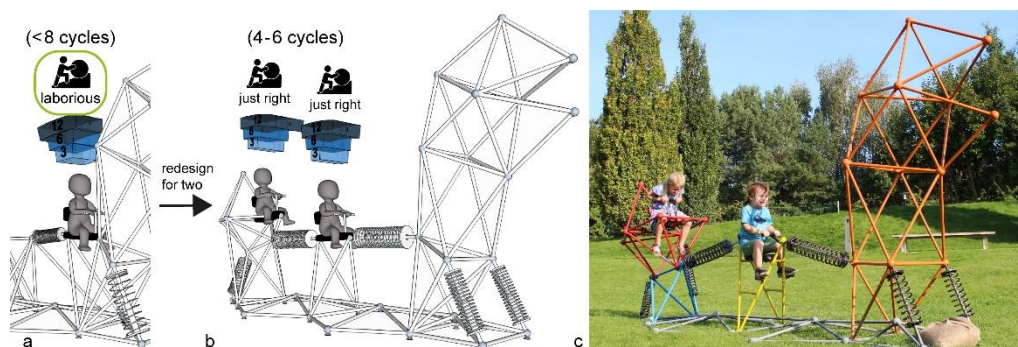


Figure 58: (a) Reducing the effort would require cutting the weight of the structure, which designers can't do. (b) Instead, they add one more seating position. The final design

comprises three spring-coupled inverted pendula, the head, middle seat, and tail. (c) Children induce resonance by synchronizing their motion.

This update changes the widget from *laborious* 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.

5.1.2 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 produces 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 [129]).

Based on these elements, the fabrication process continues as illustrated by Figure 52c-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 structure, and (f) finally weld the entire structure. Trusscillator supports this process as follows.

Trusscillator generates stencils as illustrated by Figure 59. (a) To minimize the resulting gap between rod and sphere, thus maximizing the quality of the welded connections, Trusscillator helps users arrange rods and spheres so that the rods hit the 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 points*. 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 stencil also displays node IDs and rod IDs, clarifying the placement of the edges. 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.

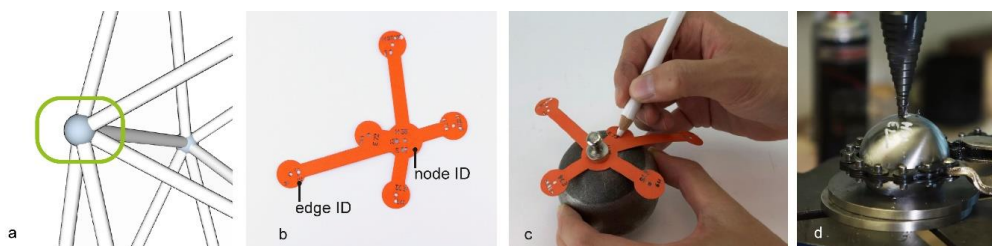


Figure 59: (a) Trusscillator exports this node in the 3D model (b) in the form of custom stencils. (c) Users mark one spot on the sphere, then attach the stencil at that point using a magnet, allowing them to mark the remaining incidence points. (d) Users then set up a stand-up drill with a round ring as a jig, and drill the spheres.

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 60a. 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 60b-c). This way they are holding the structure temporarily but firmly together for welding (Figure 60d). These connector pieces can be produced in a local metalworking shop using CNC machinery. They are considered as consumable material that stays inside the structure after welding.

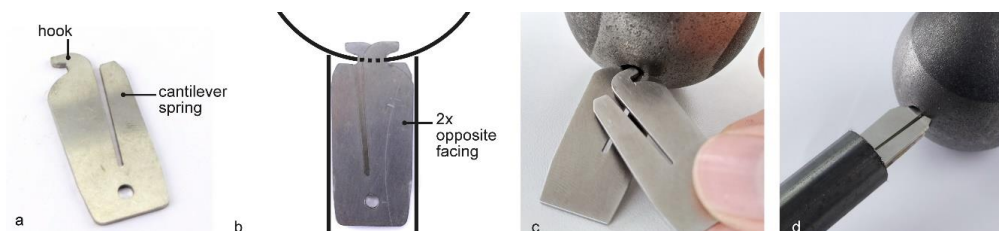


Figure 60: Trusscillator offers a temporary connector system to help position the edges for welding.

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 61a. 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 in Figure 61b, 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 61c, 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 61d shows the finished assembly of a chair model with a springy backrest.

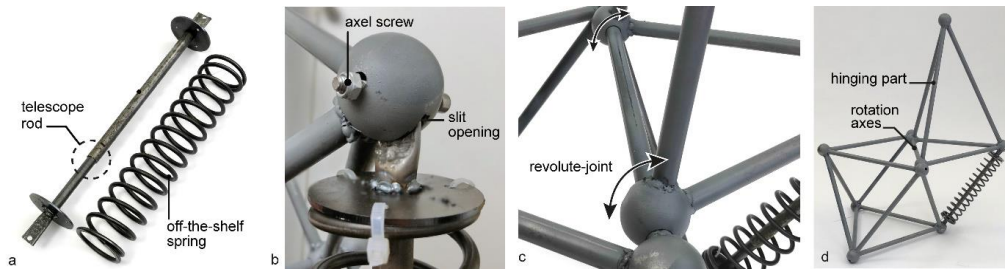


Figure 61: (a) Spring-telescope fabricated using two fitting tubes. (b) Slit opening on a sphere for inserting the telescope. (c) Revolute-joint connection. (d) Assembled chair model with a springy backrest.

5.2 DESIGN SPACE

We have used Trusscillator to design a wide range of devices. The samples are shown in Figure 62 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).

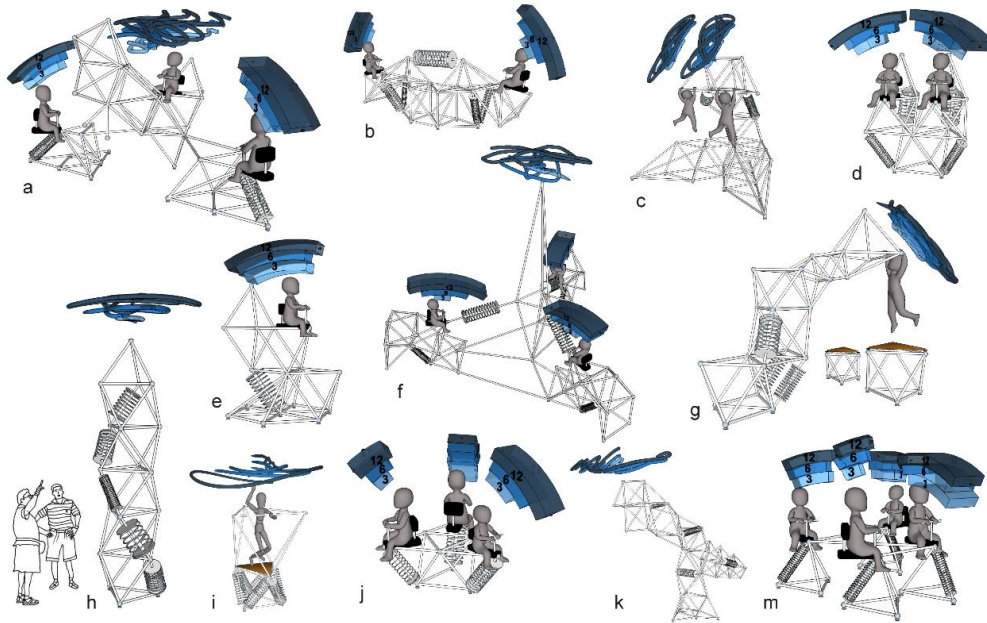


Figure 62: Some of the designs we created using Trusscillator.

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 63.

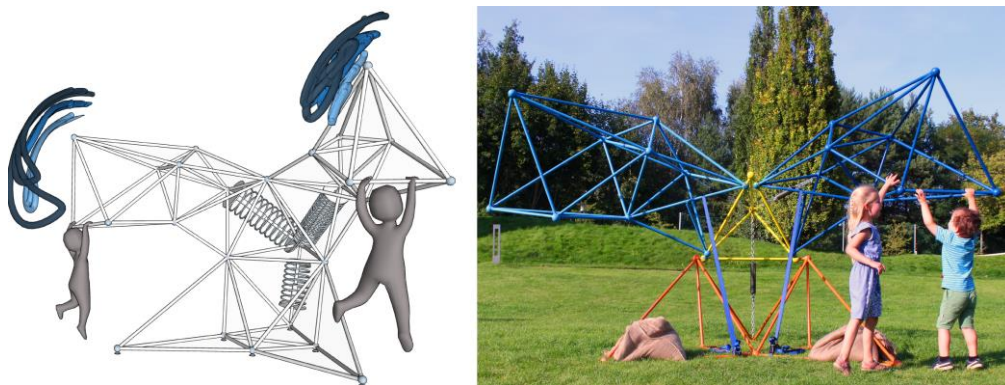


Figure 63: This “bird-swing” structure was designed to allow children to swing in two dimensions and influence each other’s experience.

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 in Figure 64,

Trusscillator supports this by offering predefined moving elements, such as a hinged tetrahedron.

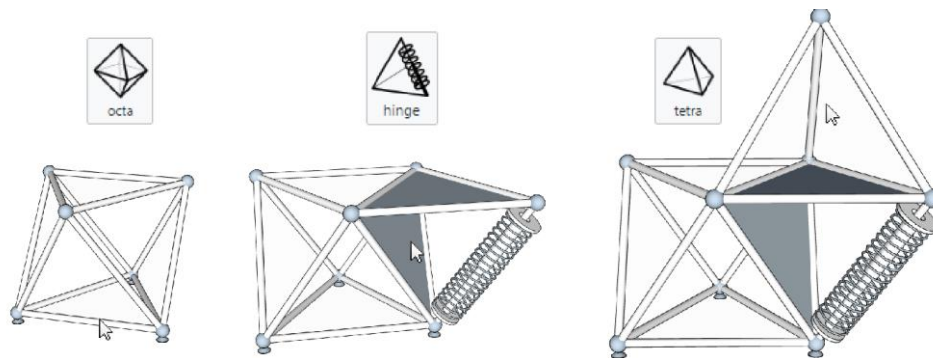


Figure 64: Building a model based on primitives containing springs speeds up the design process. Here, the chair model is constructed using a tetrahedron with one spring and two hinges in only three steps.

5.3 EXPERT INTERVIEWS

Before we started designing Trusscillator, we conducted semi-structured interviews with 3 professional 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* [122], *SketchUp* [142], *Fusion360* [120]), 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 can support the design of experiences.

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 into 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 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 basic objective for the design of our Trusscillator system.

5.4 ALGORITHMS AND IMPLEMENTATION

The Trusscillator system is implemented in the form of three main modules: (1) interactive editor frontend, (2) simulation server, and (3)

exporter for fabrication. In order to allow our readers to replicate our results, we reproduce the underlying implementation and algorithms as follows.

5.4.1 INTERACTIVE EDITOR FRONTEND

Trusscillator builds on the editor components of *TrussFab* [42] and *TrussFormer* [43], 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* [142] 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. [114]. 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 front end takes care of modeling tasks and user adjustments, the oscillation characteristics and spring solutions are provided by the simulation server.

5.4.2 SIMULATION SERVER

We implemented the simulation server in the *Julia* programming language [10] combined with the packages *DifferentialEquations.jl* [70] and *NetworkDynamics.jl* [49]. 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.

The two central advantages of using this stack for Trusscillator are: (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* [25].

Trusscillator simulates the dynamic behavior by formulating a continuous-time system of differential equations. Such differential equation systems provide a robust solution for modeling systems where maintaining *energy* constraints plays a crucial role, even in the case of fast oscillations (unlike discrete-time models, as commonly found in real-time physically-based simulations). The system uses highly optimized variable step solvers to obtain a time-domain solution of the motion that ensures that the result stays within specified tolerances. Using this approach, we have implemented a custom simulation package that can simulate the dynamics of arbitrary spring-damper-rod networks.

As illustrated by Figure 65, 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 equations 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 is the time, as follows from [70]. 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.

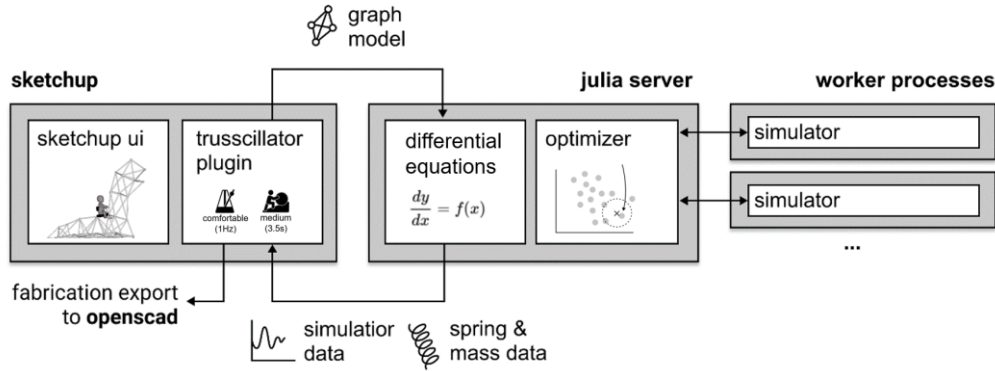


Figure 65: Trusscillator's high-level architecture.

With *NetworkDynamics.jl*, we provide a graph structure and specify the respective functions for every component separately. 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

$$\vec{a} = \frac{\sum_{edge \in E} \vec{F}_{edge}}{mass} + \frac{\vec{F}_{act}}{mass} + \vec{a}_{gravity}$$

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 it (see section “5.4.3 Simulating human actuation”). Thus, the result that we return 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. For this, we obtain the positions of each node directly from the client and set all velocities to zero.

5.4.3 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 [64] and [23].

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.

5.4.4 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.

5.4.5 TRANSLATING AMPLITUDE, FREQUENCY, AND EFFORT INTO MASS, SPRING, AND 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 66a. For these input dimensions, Trusscillator determines spring constant and mass configuration to satisfy the user’s design intent. The rough relationship between the mechanical properties and the experience attributes is illustrated in Figure 66b.

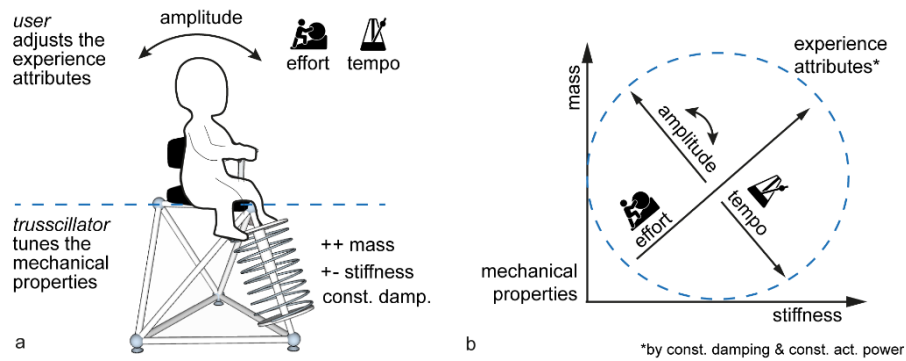


Figure 66: (a) The mechanical properties of the structure are defining the experience attributes. (b) The correlation between the mechanical properties and the motion experience (amplitude, tempo, and effort).

Trusscillator acquires the amplitude, tempo, and effort attributes by running a simulation sequence. To exemplify this process, we take the simple bobbing saddle model from Figure 66a, fit with a catalog spring with the stiffness of $k = 3376\text{N/m}$, and damping $d = 50\text{Ns/m}$, as shown in Figure 67a, and run the simulation for a 12-year-old user (40kg, 75W).

During the simulation, the human-mimicking force, described in chapter 5.4.3, starts to actuate the device and the amplitude is increasing as the energy is being accumulated in the system, as shown in Figure 67a. 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.

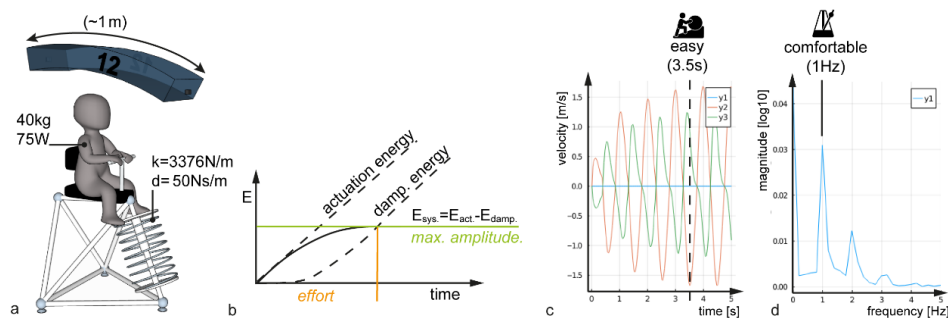


Figure 67: (a) Trusscillator simulates the (b) excitation of the model until the point when it reaches an energy equilibrium – maximum *amplitude*. (c) The time after the velocities won't increase anymore is considered as the *effort* metric. (d) The peak of the frequency spectrum determines the *tempo* metric.

With the increasing velocity, 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 67b indicates this time point when the oscillating system has reached the energy equilibrium and the amplitude remains constant. From this state the following attributes are extracted:

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 67c 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.

Tempo/Frequency: Trusscillator analyzes this 3D velocity data from Figure 67c using Fast Fourier Transform (Figure 67d) 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 [39]. Higher frequencies are classified as *shaky*, lower are *slow*. This information is displayed to the user in the *tempo widget*.

5.4.6 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 68 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., $\pm 2\text{kN/m}$), and every sample is being simulated with a range of additional masses. 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.

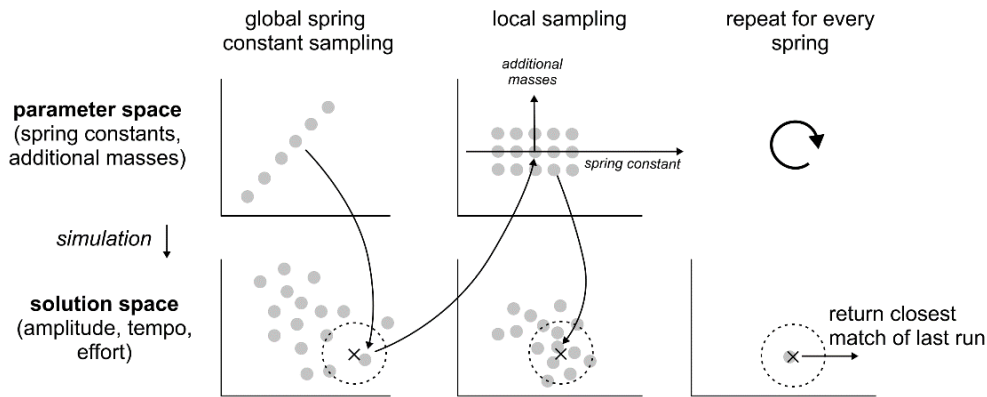


Figure 68: Spring optimization procedure.

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: $\sum_{c \in \mathcal{C}} 3 \cdot \Delta A_c + \Delta f_c + \Delta e_c + \sigma(f)$, where \mathcal{C} is the set of children and ΔA_c , Δf_c , Δe_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 ($\sigma(f)$) incentivizes structure where multiple children that can achieve resonance by increasing the distance, where the standard deviation of the measured frequencies at the child's nodes is high. The corresponding algorithm works as follows:

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

```

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.

Before returning the information back to the client, Trusscillator takes the closest matching springs from an online vendor catalog [129], configures the structure with that spring, and runs the simulation for all age groups.

We note that when optimizing for multiple parameters the algorithm might overwrite previously set values, in case can't satisfy all criteria; therefore, navigating this multi-dimensional parameter space is not always a straightforward process, but rather an open-ended exploration. Alternative UI solutions for such high-dimensional design exploration problems have been proposed by Yue et al. [112].

5.4.7 EXPORTING STENCILS

Trusscillator renders the stencils using the parametric modeling tool *OpenSCAD* [137]. 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 the center based on which all other incidence points are being referred. 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.

5.5 VALIDATION

To validate Trusscillator's functionality, we designed 15 models (Figure 62), including two models that were fabricated physically, i.e., the “brachiosaurus” in Figure 52, the “bird swing” in Figure 63.

Trusscillator allowed a team of two to design, cut, drill, assemble, weld, and paint each model in 2-3 days.

5.5.1 SIMULATION ACCURACY

We conducted a technical evaluation validating the accuracy of Trusscillator’s simulation, in which we compared the acceleration response measured for our “brachiosaurus” device with the acceleration response predicted by our simulation.

Figure 69 (left) shows the evaluation setup. Three IMU loggers (G-Sensor Logger [127]) 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 69b shows frequency spectra measured and simulated. We obtain them by applying FFT on the acceleration data from the IMU data from the real model (green line), and on the simulation data of the respective node (orange line). We observe that the simulation matches the real-world observations closely.

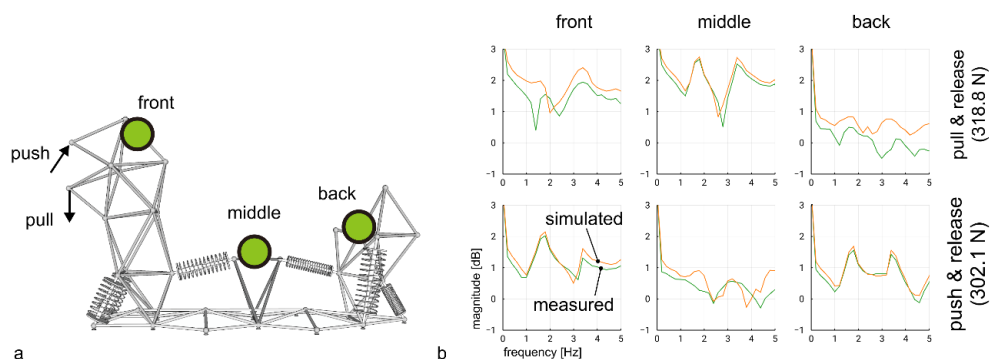


Figure 69: (a) The measurement points indicated on the real and virtual model. (b) Frequency response comparison of the push and pull experiments.

The slight differences between the real and the simulated model we interpret by increased friction and slack in the joints, caused by the imprecision of the fabrication, causing 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 fabrication quality, materials used, lubrication, etc.

5.5.2 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 61c), the bird-swing (Figure 63), and the brachiosaurus (Figure 52).

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 3, all the simulations run under 1 second—appropriate for a turn-taking interaction.

| 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 |

Table 3: Simulation benchmark results

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 (the 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 5.4.1).

5.6 CONTRIBUTION, BENEFITS, AND LIMITATIONS

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. 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 the structures.

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

On the implementation side, we contribute with a continuous-time, extensible, high-fidelity simulator with strong robustness for variable-geometry spring-damper truss structures. In contrast to many physically-based spring damper simulators that are used in computer graphics, our simulator is not prone to change the energy in the system, even for high-frequency applications.

On the hardware level, Trusscillator contributes with a series of novel hardware tools that support the fabrication of the steel truss structures, such as the drilling stencils and a temporary connector system that supports welding.

We have validated our system by designing novel pieces of playground equipment, workout devices, and interactive kinetic installations, two of which we manufactured end-to-end, and by evaluating the technical aspects, such as simulation time and accuracy.

We note that before devices designed using Trusscillator can be deployed, additional safety checks need to be considered, according to the applicable local regulatory requirements for playground equipment, such as DIN EN 1176 [125].

Zooming out, we think of Trusscillator as a tool that pushes research on large-scale personal fabrication in two ways. First, it goes to the next logical 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.

6

CONCLUSION

In this chapter, we expand upon the insights of the individual projects and draw conclusions about human-scale personal fabrication on a broader scale. We discuss how this work might impact fabrication technology and summarize our main contribution before we close by discussing long-term future directions.

6.1 SUMMARY OF CONTRIBUTIONS

Creating large-scale objects and mechanisms has so far been mainly the privilege of engineers and industry. Not only because of the high price of heavy-duty fabrication machinery, but also because of the lack of engineering know-how to create the right structure that can withstand large forces.

Our first attempt to explore human-scale personal fabrication was on the *shape* level by our Protopiper device. However, in this thesis, we strive to go beyond shape and help engineering objects for real-life *forces*. As a result, we provide blueprints of three end-to-end software systems that embody the required *domain knowledge* to enable non-professional individuals to design and fabricate human-scale objects and mechanisms that involve human-scale forces.

The presented fabrication systems are concerned with three aspects beyond the shape. These we classify in Table 4: (1) TrussFab – creating

static load-bearing structures, (2) TrussFormer — creating kinematic mechanisms, and (3) Trusscillator — creating mechanisms with dynamic movement. These aspects also correspond to three major fields of mechanical engineering: statics, kinematics, and dynamics.

| | | |
|------------------|-----------------------------|-----------------------------|
| motion | TrussFormer (kinematics) | Trusscillator (dynamics) |
| no motion | Protopiper (shape) | TrussFab (statics) |
| | no force | force |

Table 4: Classification of the presented systems by the domains they cover.

These end-to-end fabrication systems assist the creation process by: (1) providing custom editor tools for designing load-bearing trusses and mechanisms, (2) verifying the structural integrity and behavior of the resulting structures, and (3) streamlining the building process by generating the necessary parts lists, aids, and instructions. With this, our systems help users to focus on high-level design objectives without being concerned about engineering aspects of the design. Figure 70 summarizes these aspects for the three respective systems.

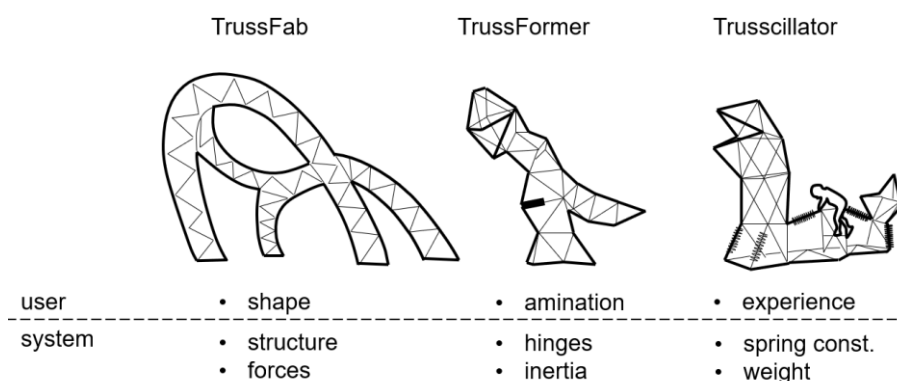


Figure 70: Our systems allow users to focus on the high-level design objectives, while the software takes care of the specific underlying engineering aspects.

While good design is certainly a crucial aspect when creating a new object, the fabrication phase is equally important in order to materialize a functional artifact. Both of these phases require different skills; therefore, they are often not carried out by the same individuals. Our systems resolve this by uniting these two phases into one integrated workflow, where the design is readily supported by a fabrication pipeline. This way single individuals are able to carry out both the design (with engineering behind the scenes) and the fabrication tasks as well. This empowers individuals to create human-scale objects using commonly available household fabrication equipment. Our contribution is this set of integrated end-to-end fabrication systems.

6.2 IMPACT

To foster this benefit of our solutions, we have been sharing our software systems freely among individuals, schools, fablabs, researchers, architects, and designers, who have been using them for a variety of hobby, art, and research projects, some of them shown in Figure 71. This general interest and the positive feedback suggest that the development of such end-to-end systems is relevant direction in personal fabrication.



Figure 71: Furniture created by tech-enthusiasts* using the TrussFab system.

* Images from Instagram.com #trussfab (by curtesy of the authors).

<https://www.instagram.com/explore/tags/trussfab>

6.3 LIMITATIONS AND FUTURE CHALLENGES

Personal fabrication of large-scale objects opens up a range of new challenges. Unlike designing desktop-scale objects, software systems for large-scale need to consider how to assure structural integrity and to guarantee safety.

One of the key challenges when designing end-to-end fabrication systems is to balance ease-of-use and expressiveness. In our systems, we have always aimed to create the simplest, yet most expressive workflow within the given fabrication constraints. We acknowledge that this might be frustrating for experienced users, who seek more design freedom.

With larger objects comes greater responsibility as well. This means that safety needs to be taken seriously when fabricating at human-scale. Safety considerations should start already at the software implementation and also be taken into account when fabricating the objects following safety standards. Since our research projects were mostly concerned with the design and fabrication aspects, rather than implementing safety regulations, this remains a task for future research and development.

While our systems peek into the three main aspects of mechanical engineering (statics, kinematics, dynamics), they are far from providing complete solutions for all challenges of human-scale fabrication. However, they might serve as inspiration for systems concerning other specific application domains.

6.4 FINAL REMARKS

Our key takeaway from exploring the topic of human-scale fabrication is that building larger objects is not only about achieving the scale by inventing specialized machinery, but more importantly to engineer for disproportionately increasing forces. This insight has driven our vision to create software systems that encapsulate pieces of engineering domain

knowledge that help users focus on high-level design objectives. We believe this principle can be applied to many other technical domains, all with the aim to *democratize engineering*.

7

REFERENCES

1. Muhammad Abdullah, Martin Taraz, Yannis Kommana, Shohei Katakura, Robert Kovacs, Jotaro Shigeyama, Thijs Roumen, and Patrick Baudisch. 2021. FastForce: Real-Time Reinforcement of Laser-Cut Structures. *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*. Association for Computing Machinery, New York, NY, USA, Article 673, 1–12. DOI: <https://doi.org/10.1145/3411764.3445466>
2. Harshit Agrawal, Udayan Umapathi, Robert Kovacs, Johannes Frohnhofen, Hsiang-Ting Chen, Stefanie Mueller, and Patrick Baudisch. 2015. Protopiper: Physically Sketching Room-Sized Objects at Actual Scale. In *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology (UIST'15)*, ACM, New York, NY, USA, 427–436. DOI: <http://doi.org/10.1145/2807442.2807505>
3. Rahul Arora, Alec Jacobson, Timothy R. Langlois, Yijiang Huang, Caitlin Mueller, Wojciech Matusik, Ariel Shamir, Karan Singh, and David I. W. Levin. 2019. Volumetric Michell trusses for parametric design & fabrication. In *Proceedings of the ACM Symposium on Computational Fabrication (SCF '19)*. Association for Computing Machinery, New York, NY, USA, Article 6, 1–13. DOI: <https://doi.org/10.1145/3328939.3328999>
4. V. Arun, Charles F. Reinholtz, and Layne Terry Watson. 1990. Enumeration and analysis of variable geometry truss manipulators. Department of Computer Science, Virginia Polytechnic Institute and State University.
5. Daniel Ashbrook, Shitao Guo, and Alan Lambie. 2016. Towards Augmented Fabrication: Combining Fabricated and Existing Objects. In

- Proceedings of the 2016 CHI Conference Extended Abstracts on Human Factors in Computing Systems (CHI EA '16), ACM, New York, NY, USA, 1510–1518. DOI: <http://doi.org/10.1145/2851581.2892509>
6. 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>
 7. 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>
 8. Patrick Baudisch and Stefanie Mueller. Personal Fabrication. *Foundations and Trends® in Human-Computer Interaction* Vol. 10: No. 3–4, 165-293. 2017.
 9. Patrick Baudisch, Arthur Silber, Yannis Kommana, Milan Gruner, Ludwig Wall, Kevin Reuss, Lukas Heilman, Robert Kovacs, Daniel Rechlitz, and Thijs Roumen. 2019. Kyub: A 3D Editor for Modeling Sturdy Laser-Cut Objects. *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. Association for Computing Machinery, New York, NY, USA, Paper 566, 1–12. DOI: <https://doi.org/10.1145/3290605.3300796>
 10. Jeff Bezanson, Stefan Karpinski, Viral B. Shah, Alan Edelma. 2012. Julia: A fast dynamic language for technical computing." arXiv preprint arXiv:1209.5145.
 11. Gaurav Bharaj, Stelian Coros, Bernhard Thomaszewski, James Tompkin, Bernd Bickel, and Hanspeter Pfister. 2015. Computational design of walking automata. In *Proceedings of the 14th ACM SIGGRAPH / Eurographics Symposium on Computer Animation (SCA '15)*. ACM, New York, NY, USA, 93-100. DOI: <https://doi.org/10.1145/2786784.2786803>
 12. Bernd Bickel, Moritz Bächer, Miguel A. Otaduy, Hyunho Richard Lee, Hanspeter Pfister, Markus Gross, and Wojciech Matusik. 2010. Design and fabrication of materials with desired deformation behavior. *ACM Trans. Graph.* 29, 4, Article 63 (July 2010), 10 pages. DOI: <https://doi.org/10.1145/1778765.1778800>

13. Paul Bosscher, Imme Ebert-Uphoff. 2003. A novel mechanism for implementing multiple collocated spherical joints. In *Robotics and Automation, Proceedings. ICRA'03. IEEE International Conference on* (Vol. 1, pp. 336-341). IEEE.
14. 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>
15. 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>
16. Xiang “Anthony” Chen, Stelian Coros, Jennifer Mankoff, and Scott E. Hudson. 2015. Encore: 3D Printed Augmentation of Everyday Objects with Printed-Over, Affixed and Interlocked Attachments. In *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology (UIST'15)*, ACM, New York, NY, USA, 73–82. DOI: <http://doi.org/10.1145/2807442.2807498>
17. Lung-Pan Cheng, Thijs Roumen, Hannes Rantzsch, Sven Köhler, Patrick Schmidt, Robert Kovacs, Johannes Jasper, Jonas Kemper, and Patrick Baudisch. 2015. TurkDeck: Physical Virtual Reality Based on People. In *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology (UIST '15)*. Association for Computing Machinery, New York, NY, USA, 417–426. DOI: <https://doi.org/10.1145/2807442.2807463>
18. Foster Collins, and Mark Yim. 2016. Design of a spherical robot arm with the spiral zipper prismatic joint. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2137-2143.
19. 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 Transactions on Graphics* 32, 4: 1. DOI: <http://doi.org/10.1145/2461912.2461953>

20. Laura Devendorf and Kimiko Ryokai. 2015. Being the Machine: Reconfiguring Agency and Control in Hybrid Fabrication. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems (CHI '15)*, ACM, New York, NY, USA, 2477–2486. DOI: <http://doi.org/10.1145/2702123.2702547>
21. Paul H. Dietz and Catherine Dietz. 2007. The animatronics workshop. In *ACM SIGGRAPH 2007 educators program (SIGGRAPH '07)*. ACM, New York, NY, USA, Article 36 . DOI: <https://doi.org/10.1145/1282040.1282078>
22. Gábor Erdős - Linkage Designer. Retrieved on Nov. 15, 2021, from <http://www.linkagedesigner.com/>
23. 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.
24. Jacob Fish and Ted Belytschko. 2007. *A first course in finite elements*. Wiley New York.
25. 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.
26. Ollé Gellért. Print To Build, 3D printed joint collection. Retrieved on Nov. 15, 2021 from <https://www.behance.net/gallery/27812109/Print-To-Build-3D-printed-joint-collection>
27. Gregory J. Hamlin and Arthur C. Sanderson. 2013. Tetrobot: A Modular Approach to Reconfigurable Parallel Robotics. In Volume 423 of *The Springer International Series in Engineering and Computer Science*, Springer Science & Business Media. ISBN: 1461554713, 9781461554714
28. Zachary M. Hammond, Nathan S. Usevitch, Elliot W. Hawkes, and Sean Follmer. 2017. Pneumatic Reel Actuator: Design, modeling, and implementation. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 626-633.
29. 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>

30. 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>
31. 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>
32. Scott E. Hudson. 2014. Printing Teddy Bears. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '14)*, ACM, New York, NY, USA, 459–468. <http://doi.org/10.1145/2556288.2557338>
33. Yuki Igarashi, Takeo Igarashi, and Jun Mitani. 2012. Beady: interactive beadwork design and construction. *ACM Transactions on Graphics (TOG)* 31, c: 49. DOI: <http://doi.org/10.1145/2185520.2185545>
34. 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>
35. 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>
36. 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>
37. Sasa Jokic and Petar Novikov. Mataerial - A Radical New 3D Printing Method. Retrieved on Nov. 15, 2021 from <http://www.mataerial.com/>

38. Sasa Jokic, Petr Novikov, Shihui Jin, Stuart Maggs, Cristina Nan, and Dori Sadan. Minibuilders: Robots for 3D printing in construction and design. Retrieved on Nov. 15, 2021 from <http://robots.iaac.net/>
39. 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.
40. Behrokh Khoshnevis. 2004. Automated Construction by Contour Crafting—Related Robotics and Information Technologies. *Automation in Construction* 13, 1: 5–19. DOI: <http://doi.org/10.1016/j.autcon.2003.08.012>
41. Konstantin Klamka, Raimund Dachsel, and Jürgen Steimle. 2020. Rapid Iron-On User Interfaces: Hands-on Fabrication of Interactive Textile Prototypes. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems (CHI '20)*. Association for Computing Machinery, New York, NY, USA, 1–14. DOI: <https://doi.org/10.1145/3313831.3376220>
42. 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>
43. 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>
44. Robert Kovacs, Lukas Rambold, Lukas Fritzsche, Dominik Meier, Jotaro Shigeyama, Shohei Katakura, Ran Zhang, Patrick Baudisch. 2021. Trusscillator: a System for Fabricating Human-Scale Human-Powered Oscillating Devices. To appear in *Proceedings of the 34th Annual ACM Symposium on User Interface Software and Technology (UIST '21)*. Association for Computing Machinery, New York, NY, USA, 1074–1088. DOI: <https://doi.org/10.1145/3472749.3474807>

45. Benjamin Lafreniere, Marcelo H. Coelho, Nicholas Cote, Steven Li, Andy Nogueira, Long Nguyen, Tobias Schwinn, James Stoddart, David Thomasson, Ray Wang, Thomas White, Tovi Grossman, David Benjamin, Maurice Conti, Achim Menges, George Fitzmaurice, Fraser Anderson, Justin Matejka, Heather Kerrick, Danil Nagy, Lauren Vasey, Evan Atherton, and Nicholas Beirne. 2016. Crowdsourced Fabrication. In *Proceedings of the 29th Annual ACM Symposium on User Interface Software & Technology (UIST '16)*, 15–28. <http://doi.org/10.1145/2984511.2984553>
46. Tien T. Lan. 2005. *Structural Engineering Handbook - Space Frame Structures*. CRC Press.
47. Manfred Lau, Akira Ohgawara, Jun Mitani, and Takeo Igarashi. 2011. Converting 3D furniture models to fabricatable parts and connectors. *ACM Trans. Graph.* 30, 4, Article 85 (July 2011), 6 pages. DOI: <https://doi.org/10.1145/2010324.1964980>
48. Jiahao Li, Jeeun 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>
49. Michael Lindner, Lucas Lincoln, Fenja Drauschke, Julia Monika Koulen, Hans Würfel, Anton Plietzsch, and Frank Hellmann. 2021. NetworkDynamics.jl—Composing and simulating complex networks in Julia. - *Chaos*, 31, 6, 063133. DOI: <https://doi.org/10.1063/5.0051387>.
50. Wenjuan Lu, Lijie Zhang, Yitong Zhang, Yalei Ma, Xiaoxu Cui. 2014. Modified Formula of Mobility for Mechanisms. In *Proceedings of the International Conference on Intelligent Robotics and Applications (ICIRA'14)*. Springer, Cham, Germany, 2014, 535-545. DOI : https://doi.org/10.1007/978-3-319-13963-0_54
51. Linjie Luo, Ilya Baran, Szymon Rusinkiewicz, and Wojciech Matusik. 2012. Chopper: Partitioning Models into 3D-Printable Parts. *ACM Transactions on Graphics* 31, 6: 1. DOI: <http://doi.org/10.1145/2366145.2366148>
52. Michael Makris, David Gerber, Anders Carlson, and Doug Noble. 2013. Informing Design through Parametric Integrated Structural Simulation.

In eCAADe 2013: Computation and Performance—Proceedings of the 31st International Conference on Education and research in Computer Aided Architectural Design in Europe, Delft University of Technology, 69–77.

53. Asok Kumar Mallik, Amitabha Gosh, Günter Dittrich. 1994. Kinematic analysis and synthesis of mechanisms. CRC Press.
54. Stefan Marti and Chris Schmandt. 2005. Physical embodiments for mobile communication agents. In *Proceedings of the 18th Annual ACM Symposium on User Interface Software and Technology (UIST'05)*. ACM, New York, NY, USA, 231-240. DOI: <http://dx.doi.org/10.1145/1095034.1095073>
55. Alexander R. McN. 1989. Elastic mechanisms in the locomotion of vertebrates. *Netherlands Journal of Zoology* 40, no. 1-2(1989): 93-105.
56. 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.
57. 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>
58. James L. Meriam and L. Glenn Kraige. 2012. *Engineering mechanics: dynamics*. Vol. 2. John Wiley & Sons.
59. Niloy J Mitra and Mark Pauly. 2009. Shadow art. *ACM Transactions on Graphics* 28, 5: 1. DOI: <http://doi.org/10.1145/1618452.1618502>
60. Stefanie Mueller, Sangha Im, Serafima Gurevich, Alexander Teibrich, Lisa Pfisterer, François Guimbretière, and Patrick Baudisch. 2014. WirePrint: 3D Printed Previews for Fast Prototyping. In *Proceedings of the 27th Annual ACM Symposium on User Interface Software & Technology (UIST '14)*, ACM, New York, NY, USA, 273–280. DOI: <http://doi.org/10.1145/2642918.2647359>
61. Stefanie Mueller, Tobias Mohr, Kerstin Guenther, Johannes Frohnhofen, and Patrick Baudisch. 2014. FaBrickation: fast 3D printing of functional objects by integrating construction kit building blocks. In *CHI '14 Extended Abstracts on Human Factors in Computing Systems (CHI EA '14)*.

- Association for Computing Machinery, New York, NY, USA, 187–188.
DOI: <https://doi.org/10.1145/2559206.2582209>
62. Takumi Murayama, Junichi Yamaoka, and Yasuaki Kakehi. 2020. Reflatables: A Tube-based Reconfigurable Fabrication of Inflatable 3D Objects. In *Extended Abstracts of the 2020 CHI Conference on Human Factors in Computing Systems (CHI EA '20)*. Association for Computing Machinery, New York, NY, USA, 1–8. DOI: <https://doi.org/10.1145/3334480.3382904>
 63. Simon Olberding, Sergio Soto Ortega, Klaus Hildebrandt, and Jürgen Steimle. 2015. Foldio: Digital Fabrication of Interactive and Shape-Changing Objects With Foldable Printed Electronics. In *Proceedings of the 28th Annual ACM Symposium on User Interface and Software Technology (UIST '15)*. Association for Computing Machinery, New York, NY, USA, 223–232. DOI: <https://doi.org/10.1145/2807442.2807494>
 64. 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.
 65. Joseph Reuben Harry Otter, Alfred Carlo Cassell, and Roger Edwin Hobbs. 1966. Dynamic Relaxation. In *Proceedings of the Institution of Civil Engineers* 35, 4: 633–656. DOI: <http://doi.org/10.1680/iicep.1966.8604>
 66. Michael J. D. Powell. 1964. An Efficient Method for Finding the Minimum of a Function of Several Variables without Calculating Derivatives. *The computer journal*: 155–162. <http://doi.org/10.1093/comjnl/7.2.155>
 67. Clemens Preisinger. Karamba3D - Parametric Structural Modeling. Retrieved on Nov. 15, 2021 from <http://www.karamba3d.com/>
 68. Romain Prévost, Emily Whiting, Sylvain Lefebvre, and Olga Sorkine-Hornung. 2013. Make it stand: balancing shapes for 3D fabrication. *ACM Trans. Graph.* 32, 4, Article 81 (July 2013), 10 pages. DOI: <https://doi.org/10.1145/2461912.2461957>
 69. Jie Qi and Leah Buechley. 2012. Animating paper using shape memory alloys. In *Proceedings of the 2012 ACM annual conference on Human Factors in Computing Systems (CHI '12)*. DOI: <http://doi.org/10.1145/2207676.2207783>

70. 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).
71. Marvin D. Rhodes, and Martin M. Mikulas. *Deployable controllable geometry truss beam*. 1985 National Aeronautics and Space Administration, Scientific and Technical Information Branch (Technical report No. NASA-TM-86366. 1985).
72. Tiago Ribeiro and Ana Paiva. 2012. The illusion of robotic life: principles and practices of animation for robots. In *Proceedings of the seventh annual ACM/IEEE international conference on Human-Robot Interaction (HRI '12)*. ACM, New York, NY, USA, 383-390. DOI: <https://doi.org/10.1145/2157689.2157814>
73. Ronald Richter and Marc Alexa. 2015. Beam Meshes. *Computers & Graphics*, 1: 1–8. <http://doi.org/10.1016/j.cag.2015.08.007>
74. David W. Rosen. 2007. Computer-Aided Design for Additive Manufacturing of Cellular Structures. *Computer-Aided Design and Applications*, 4:5, 585-594, DOI: 10.1080/16864360.2007.10738493
75. Thijs Roumen, Willi Mueller and Patrick Baudisch. 2018. Grafter: Remixing 3D Printed Machines. In *Proceedings of the 36th Annual ACM Conference on Human Factors in Computing Systems (CHI '18)*. ACM, New York, USA DOI: <https://doi.org/10.1145/3173574.3173637>
76. 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>
77. B. Roth. 1981. Rigid and Flexible Frameworks. *Mathematical Association of America* 88, 1: 6–21.
78. Mose Sakashita, Tatsuya Minagawa, Amy Koike, Ippei Suzuki, Keisuke Kawahara, and Yoichi Ochiai. 2017. You as a Puppet: Evaluation of Telepresence User Interface for Puppetry. In *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology (UIST'17)*. ACM, New York, NY, USA, 217-228. DOI: <https://doi.org/10.1145/3126594.3126608>

79. Harpreet Sareen, Udayan Umapathi, Patrick Shin, Yasuaki Kakehi, Jifei Ou, Hiroshi Ishii, and Pattie Maes. 2017. Printflatables: Printing Human-Scale, Functional and Dynamic Inflatable Objects. *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*. Association for Computing Machinery, New York, NY, USA, 3669–3680. DOI: <https://doi.org/10.1145/3025453.3025898>
80. Hiroki Sato, Young ah Seong, Ryosuke Yamamura, Hiromasa Hayashi, Katsuhiko Hata, Hisato Ogata, Ryuma Niiyama, and Yoshihiro Kawahara. 2020. Soft yet Strong Inflatable Structures for a Foldable and Portable Mobility. In *Extended Abstracts of the 2020 CHI Conference on Human Factors in Computing Systems (CHI EA '20)*. Association for Computing Machinery, New York, NY, USA, 1–4. DOI: <https://doi.org/10.1145/3334480.3383147>
81. Greg Saul, Manfred Lau, Jun Mitani, and Takeo Igarashi. 2011. 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)*, ACM, New York, NY, USA, 73. <http://doi.org/10.1145/1935701.1935717>
82. 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>
83. Rita Shewbridge, Amy Hurst, and Shaun K. Kane. 2014. Everyday Making. In *Proceedings of the 2014 conference on Designing interactive systems (DIS '14)*, ACM, New York, NY, USA, 815–824. DOI: <http://doi.org/10.1145/2598510.2598544>
84. Melina Skouras, Stelian Coros, Eitan Grinspun, and Bernhard Thomaszewski. 2015. Interactive Surface Design with Interlocking Elements. *ACM Transactions on Graphics* 34, 6: 224. DOI: <http://doi.org/10.1145/2816795.2818128>
85. Jeffrey Smith, Jessica Hodgins, Irving Oppenheim, and Andrew Witkin. 2002. Creating Models of Truss Structures with Optimization. *ACM Transactions on Graphics.* 21, 3: 295–301. DOI: <http://doi.org/10.1145/566654.566580>

86. A. Y. N. Sofla, D. M. Elzey, and H. N. G. Wadley. 2009. Shape morphing hinged truss structures. *Smart Materials and Structures* 18: 65012. DOI: <http://doi.org/10.1088/0964-1726/18/6/065012>
87. Alexander Spinos, Devin Carroll, Terry Kientz, and Mark Yim. 2017. Variable topology truss: Design and analysis. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Vancouver, BC, 2017, 2717-2722. DOI: 10.1109/IROS.2017.8206098
88. Alexander Spinos and Mark Yim. 2017. Towards a variable topology truss for shoring. 2017. In *Proceedings of the 14th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI)*, 244-249. DOI: <http://doi.org/10.1109/URAI.2017.7992723>
89. Doug Stewart. 1965. A platform with six degrees of freedom. In *Proceedings of the institution of mechanical engineers* 180.1, 371-386.
90. Devika Subramanian. 1995. Kinematic synthesis with configuration spaces. In *Research in Engineering Design* 7, no. 3, 193-213.
91. Ryo Suzuki, Ryosuke Nakayama, Dan Liu, Yasuaki Kakehi, Mark D. Gross, and Daniel Leithinger. 2020. LiftTiles: Constructive Building Blocks for Prototyping Room-scale Shape-changing Interfaces. In *Proceedings of the Fourteenth International Conference on Tangible, Embedded, and Embodied Interaction (TEI '20)*. Association for Computing Machinery, New York, NY, USA, 143-151. DOI: <https://doi.org/10.1145/3374920.3374941>
92. Saiganesh Swaminathan, Michael Rivera, Runchang Kang, Zheng Luo, Kadri Bugra Ozutemiz, and Scott E. Hudson. 2019. Input, Output and Construction Methods for Custom Fabrication of Room-Scale Deployable Pneumatic Structures. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* 3, 2, Article 62 (June 2019), 17 pages. DOI: <https://doi.org/10.1145/3328933>
93. Anton Synytsia. *MSPhysics* physics simulation. Retrieved on Nov. 15, 2021 from <https://extensions.sketchup.com/en/content/msphysics>
94. 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.

95. Joshua G. Tanenbaum, Amanda M. Williams, Audrey Desjardins, and Karen Tanenbaum. 2013. Democratizing Technology: Pleasure, Utility and Expressiveness in DIY and Maker Practice. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '13)*: 2603–2612. DOI: <http://doi.org/10.1145/2470654.2481360>
96. 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>
97. Alexander Teibrich, Stefanie Mueller, François Guimbretière, Robert Kovacs, Stefan Neubert, and Patrick Baudisch. 2015. Patching Physical Objects. In *Proceedings of the 28th Annual ACM Symposium on User Interface Software Technology (UIST '15)*. Association for Computing Machinery, New York, NY, USA, 83–91. DOI: <https://doi.org/10.1145/2807442.2807467>
98. Emil Teutan, S.D. Stan., D. Verdes, R. Balan. 2009. Virtual Reality Simulation of Tetrobot Parallel Robot for Medical Applications. In *Proceedings of International Conference on Advancements of Medicine and Health Care through Technology*, Vol 26. pp. 177-180.
99. 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>
100. Nobuyuki Umetani, Takeo Igarashi, and Niloy J. Mitra. 2012. Guided exploration of physically valid shapes for furniture design. *ACM Transactions on Graphics* 31, 4: 1–11. DOI: <http://doi.org/10.1145/2185520.2335437>
101. 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>
102. Weiming Wang, Tuanfeng Y. Wang, Zhouwang Yang, Ligang Liu, Xin Tong, Weihua Tong, Jiansong Deng, Falai Chen, and Xiuping Liu. 2013. Cost-effective Printing of 3D Objects with Skin-frame Structures. *ACM*

Transactions on Graphics 32, 6: 1–10.
DOI: <http://doi.org/10.1145/2508363.2508382>

103. Christian Weichel, Manfred Lau, David Kim, Nicolas Villar, Hans W. Gellersen, Christian Weichel, Manfred Lau, David Kim, Nicolas Villar, and Hans W. Gellersen. 2014. MixFab: A Mixed-reality Environment for Personal Fabrication. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '14)*, 3855–3864. DOI: <http://doi.org/10.1145/2556288.2557090>
104. David Weintrop, Afsoon Afzal, Jean Salac, Patrick Francis, Boyang Li, David C. Shepherd, and Diana Franklin. 2018. Evaluating CoBlox: A Comparative Study of Robotics Programming Environments for Adult Novices. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems (CHI '18)*. ACM, New York, NY, USA. DOI: <https://doi.org/10.1145/3173574.3173940>
105. Emily Whiting, John Ochsendorf, and Frédo Durand. 2009. Procedural modeling of structurally-sound masonry buildings. In *ACM SIGGRAPH Asia 2009 papers (SIGGRAPH Asia '09)*. Association for Computing Machinery, New York, NY, USA, Article 112, 1–9. DOI: <https://doi.org/10.1145/1661412.1618458>
106. Emily Whiting, Nada Ouf, Liane Makatura, Christos Mousas, Zhenyu Shu, and Ladislav Kavan. 2017. Environment-Scale Fabrication: Replicating Outdoor Climbing Experiences. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems (CHI '17)*. Association for Computing Machinery, New York, NY, USA, 1794–1804. DOI: <https://doi.org/10.1145/3025453.3025465>
107. Karl Willis, Eric Brockmeyer, Scott Hudson, and Ivan Poupyrev. 2012. Printed Optics. In *Proceedings of the 25th annual ACM symposium on User interface software and technology (UIST '12)*, 589. DOI: <http://doi.org/10.1145/2380116.2380190>
108. Jan Willmann, Federico Augugliaro, Thomas Cadalbert, Raffaello D'Andrea, Fabio Gramazio, and Matthias Kohler. 2012. Aerial Robotic Construction Towards a New Field of Architectural Research. *International Journal of Architectural Computing* 10, 3: 439–460. DOI: <http://doi.org/10.1260/1478-0771.10.3.439>

109. 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>
110. Suguru Yamada, Hironao Morishige, Hiroki Nozaki, Masaki Ogawa, Takuro Yonezawa, and Hideyuki Tokuda. 2016. ReFabricator: Integrating Everyday Objects for Digital Fabrication. In *Proceedings of the 2016 CHI Conference Extended Abstracts on Human Factors in Computing Systems (CHI EA '16)*, ACM, New York, NY, USA, 3804–3807. DOI: <http://doi.org/10.1145/2851581.2890237>
111. Hironori Yoshida, Syunsuke Igarashi, Takeo Igarashi, Yusuke Obuchi, Yosuke Takami, Jun Sato, Mika Araki, Masaaki Miki, Kosuke Nagata, Kazuhide Sakai, Kyungeun Sung, and Tim Cooper. 2015. Architecture-scale Human-assisted Additive Manufacturing. *ACM Transactions on Graphics* 34, 4: 88:1-88:8. DOI: <http://doi.org/10.1145/2766951>
112. Yonghao Yue, Yuki Koyama, Issei Sato, and Takeo Igarashi. 2021. User interfaces for high-dimensional design problems: from theories to implementations. In *ACM SIGGRAPH 2021 Courses (SIGGRAPH '21)*. Association for Computing Machinery, New York, NY, USA, Article 11, 1–34. DOI: <https://doi.org/10.1145/3450508.3464551>
113. Henrik Zimmer and Leif Kobbelt. 2014. Zometool Rationalization of Freeform Surfaces. *IEEE Transactions on Visualization and Computer Graphics* 20, 10: 1461–1473. DOI: <http://doi.org/10.1109/TVCG.2014.2307885>
114. 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.
115. Lifeng Zhu, Weiwei Xu, John Snyder, Yang Liu, Guoping Wang, and Baining Guo. 2012. Motion-guided mechanical toy modeling. *ACM Trans. Graph.* 31, 6, Article 127 (Nov. 2012), 10 pages. DOI: <http://dx.doi.org/10.1145/2366145.2366146>
116. Sasa Zivkovic, and Christopher Battaglia. 2017. Open source factory: democratizing large-scale fabrication systems. *Proceedings of the 37th Annual Conference of the Association for Computer Aided Design in Architecture (ACADIA)* ISBN 978-0-692-96506-1] Cambridge, MA 2-4

November, 2017), pp. 660- 669.
DOI: <https://doi.org/10.52842/conf.acadia.2017.660>

117. ABB RobotStudio. Retrieved on Nov. 15, 2021 from <https://new.abb.com/products/robotics/robotstudio>
118. Adobe Premier's keyframe editor. Retrieved on Nov. 15, 2021, from: <https://helpx.adobe.com/premiere-pro/using/adding-navigating-setting-keyframes.html>
119. Algoryx Momentum. Retrieved on Nov. 15, 2021, from: <https://www.algoryx.se/momentum/>
120. Autodesk Fusion 360. Retrieved Nov. 15, 2021 from: <https://www.autodesk.com/products/fusion-360>
121. Autodesk MeshMixer. Retrieved Nov. 15, 2021 from: <http://meshmixer.com>
122. Autodesk Revit. Retrieved Nov. 15, 2021 from: <https://www.autodesk.com/products/revit/>
123. Blender's Keyframe editor, <https://docs.blender.org/manual/en/dev/animation/keyframes/editing.html>, last accessed at 31/03/2018.
124. Crayon Physics. Retrieved on Nov. 15, 2021 from: <http://crayonphysics.com/>
125. DIN EN 1176 - Safety requirements and test methods for playground equipment. Retrieved on Nov. 15, 2021 from: <https://www.din.de/de/meta/suche/62730!search?query=DIN+EN+1176&submit-btn=Submit>
126. freeCAD. Retrieved on April 3, 2018 from: <http://www.freecad.com/freecad/>
127. G-Sensor Logger. Retrieved on Nov. 15, 2021, from: <https://play.google.com/store/apps/details?id=com.peterhohsy.gsensorddebug&hl=en>
128. GlobalTruss TRUSSTOOL. Retrieved on Nov. 15, 2021 from: <https://trusstool.com/>
129. Gutekunst Federn. Retrieved on Nov. 15, 2021, from: <https://www.federnshop.com/de/produkte/druckfedern.html>

130. Homes Made from: Plastic Bottles. Retrieved on Nov. 15, 2021 from:
<http://www.inspirationgreen.com/plastic-bottle-homes>
131. KUKA.sim. Retrieved on Nov. 15, 2021 from:
https://www.kuka.com/en-de/products/robot-systems/software/planning-project-engineering-service-safety/kuka_sim
132. KUKA | prc. Retrieved on Nov. 15, 2021 from:
<http://www.robotsinarchitecture.org/kuka-prc>
133. Mathworks Simscape. Retrieved on Nov. 15, 2021, from:
<https://www.mathworks.com/products/simscape.html>
134. MeshLab. Retrieved on Nov. 15, 2021 from:.
<http://meshlab.sourceforge.net>
135. MiTek-PAMIR Software. Retrieved on Nov. 15, 2021 from:
<http://www.mitek.co.uk/PAMIR/>
136. Newton Dynamics. Retrieved on Nov. 15, 2021 from:
<http://newtondynamics.com/forum/newton.php>
137. OpenSCAD parametric solid modeling software. Retrieved on Nov. 15, 2021, from: <https://www.openscad.org/>
138. React. Retrieved on Nov. 15, 2021 from: <https://reactjs.org/SketchUp>
139. RhinoVAULT by Matthias Rippmann. Retrieved on Nov. 15, 2021 from: <https://www.food4rhino.com/en/app/rhinovault>
140. ROS – Robot Operating System. Retrieved on Nov. 15, 2021 from:
<https://www.ros.org/>
141. SkyCiv cloud engineering software. Retrieved on Nov. 15, 2021 from:
<https://skyciv.com/>
142. Trimble SketchUp. Retrieved Nov. 15, 2021 from:
<http://www.sketchup.com/>
143. Vortex Studio. CM Labs. Retrieved on Nov. 15, 2021, from:
<https://www.cm-labs.com/vortex-studio>