Designing Chain Reaction Contraptions from Causal Graphs

ROBIN ROUSSEL, University College London
MARIE-PAULE CANI, LIX, École Polytechnique
JEAN-CLAUDE LÉON, Univ. Grenoble Alpes, CNRS, Inria, Grenoble INP, LJK
NILOY J. MITRA, University College London

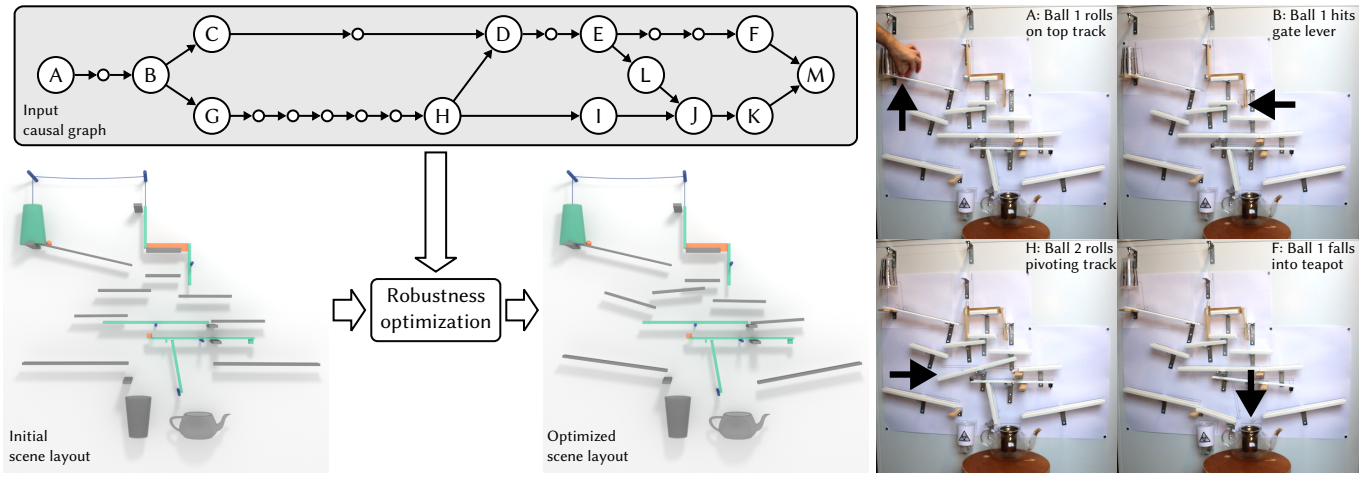

Fig. 1. Our system takes as input an initial scene layout associated with a causal graph of expected events. It then combines simulation, search and learning to build a success probability measure with respect to layout perturbations, and optimizes the layout for robustness against manual placement errors during assembly. The optimized layout is then exported as a guide sheet and used to successfully assemble complex chain reactions in the physical world.

Chain reaction contraptions, commonly referred to as Rube Goldberg machines, achieve simple tasks in an intentionally complex fashion via a cascading sequence of events. They are fun, engaging and satisfying to watch. Physically realizing them, however, involves hours or even days of manual trial-and-error effort. The main difficulties lie in predicting failure factors over long chains of events and robustly enforcing an expected causality between parallel chains, especially under perturbations of the layout. We present a computational framework to help design the layout of such contraptions by optimizing their robustness to possible assembly errors. Inspired by the active learning paradigm in machine learning, we propose a generic sampling-based method to progressively approximate the *success probability distribution* of a given scenario over the design space of possible scene layouts. The success or failure of any given simulation is determined from a user-specified causal graph enforcing a time ordering between expected events. Our method scales to complex causal graphs and high dimensional design spaces by dividing the graph and scene into simpler sub-scenarios. The aggregated success probability distribution is subsequently used to optimize the entire layout. We demonstrate the use of our framework through a range of real world examples of increasing complexity, and report significant improvements over alternative approaches.

CCS Concepts: • **Computing methodologies** → **Shape analysis**; *Model verification and validation*; Physical simulation.

Additional Key Words and Phrases: computational design, robust design, causal graphs, chain reactions, success probability distribution

ACM Reference Format:
Robin Roussel, Marie-Paule Cani, Jean-Claude Léon, and Niloy J. Mitra. 2019. Designing Chain Reaction Contraptions from Causal Graphs. *ACM Trans. Graph.* 38, 4, Article 43 (July 2019), 13 pages. https://doi.org/10.1145/3306346.3322977

1 INTRODUCTION

Chain reaction contraptions, also known as Rube Goldberg machines, achieve simple functions from intentionally complex sequences of events (see Figure 2). These machines sit at the intersection of entertainment, art and engineering; as such, they are featured in TV commercials [Honda 2003], exhibited as art pieces [Fischli and Weiss 1987], and used for educational purposes in classrooms and science fairs [Kim and Park 2012]. A particularly compelling aspect of these setups is the careful management of risk: a chain of events is all the more captivating when it looks like it could fail at multiple points. Contraption builders can spend days trying to assemble these sophisticated structures in a reasonably predictable way, relying on a rich community knowledge including rules of thumb and specific procedures to try to minimize risks of failure [Price 2017].

Authors' addresses: Robin Roussel, University College London, robin.roussel.15@ucl.ac.uk; Marie-Paule Cani, LIX, École Polytechnique, marie-paule.cani@polytechnique.edu; Jean-Claude Léon, Univ. Grenoble Alpes, CNRS, Inria, Grenoble INP, LJK, jean-claude.leon@ense3.grenoble-inp.fr; Niloy J. Mitra, University College London, n.mitra@ucl.ac.uk.

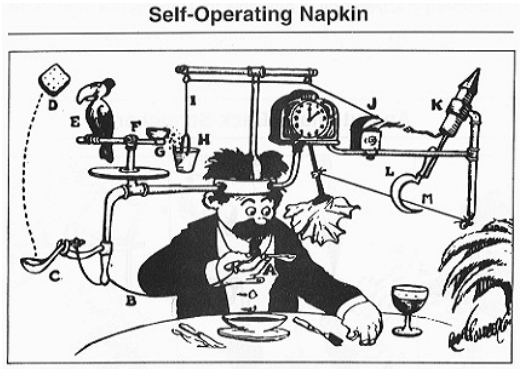

Fig. 2. In a now-famous cartoon series, Rube Goldberg invented complicated gadgets performing simple tasks in convoluted ways. These machines are fun and exciting as they delicately balance apparent unpredictability and careful risk management. In this paper, we focus on computationally optimizing the layout of real-world *Rube Goldberg machines*. "Professor Butts and the Self-Operating Napkin" is from Wikipedia (public domain).

Despite these efforts, physical realization of such chains of events remains a delicate art, involving a tedious and very time consuming trial-and-error design process.

Authors of chain reactions face two main challenges. First, small variations at one step may result in wider unintended deviations further down the line (aka the butterfly effect). Limitations in our spatial cognitive abilities prevent us from considering all possible outcomes of a sequence of physical events [Schwartz and Hegarty 1996]. As a consequence, long chains of events – even individually simple ones – can easily fail due to a single unwanted side effect. For instance, dominoes arranged along tight, highly curved paths can fall onto each other in an unexpected order. Moreover, orchestrating complex sequences may require carefully synchronizing several simpler sub-chains that run in parallel, or at least being able to robustly predict the completion order of these sub-chains. In other words, a target *causality* between events is often sought, such as a lid being removed from a cup *so that* a ball can fall in it. This kind of effect is essential to make contraptions more visually engaging, as they make potential failures points more obvious to the spectator.

Rube Goldberg machines are an example of real life designs where authoring and assembly are several orders of magnitude longer than the final execution. Hence, despite the efforts of many passionate practitioners, such contraptions are often limited to linear chains and lack non-trivial causal dependencies involving the synchronization of parallel branches. In this paper, we investigate the use of computational design to simplify and accelerate the realization of chain reaction contraptions, notably by making designs robust to modeling bias and perturbations induced by manual assembly.

Interestingly, developing a computational design tool for such machines is quite different from design problems already tackled in computer graphics: while the creation of objects and assemblies from target motion has already been investigated (see the survey by Bermano et al. [2017]), fabrication was done by connecting 3D printed or laser cut parts. By contrast, we face two extra challenges: first, Rube Goldberg machines are fully assembled by hand, and each placement error may jeopardize the whole execution. Second, only pre-existing and possibly imperfect real-world objects are used as components. The designed layout therefore needs to account for their variability and approximately known features. While the management of variability and errors is discussed in industrial contexts [Brewer et al. 2010], the prevalence of intentional risk-taking in Rube Goldberg machines makes them a challenging case study.

The key idea of this paper is to build a simulation-based *success probability distribution* (SPD) for the intended scenario conditioned on the layout parameters of the contraption assembly. The input design is subsequently optimized under this estimated probability to improve the robustness to perturbations of the machine layout. More precisely, we start with (i) an initial set of primitive objects (e.g., ball, track) arranged in a coarse scene layout provided by the user; (ii) a set of predefined events (e.g., 'rolling on', 'falling') arranged in a *causal graph* specifying their expected event order as in Figure 1; and (iii) a limit range for each layout parameter. Note that the initial layout does not need to yield a successful run; instead, we expect to find such successful layouts in the provided parameter space. Efficiently computing a probability of success from such input requires solving two challenges: first, exploring the potentially high-dimensional design space to find enough successful instances; and second, building an estimator that is accurate near the relevant regions of the design space.

We combine efficient search and machine learning techniques to address both issues. We tackle the first challenge using an adaptive sampling algorithm that progressively trades exploration of the design space for exploitation of the discovered successful regions. We formulate the second challenge as a binary success/failure classification task, where features are layout parameters and labels are derived from simulations run under the supervision of the causal graph. The success probability given the layout is therefore expressed as the probability of belonging to the 'success' class, as provided by the classifier; it is further refined with an *active learning* technique. Simulations are run with a fast rigid body engine [Coumans 2018], as we posit that a relatively coarse approximation of reality is sufficient to build a confidence metric. Additionally, we use sensitivity analysis to identify events holding a critical role in the sequence and map their individual probability of success to the relevant design parameters; this allows our method to scale to a high number of dimensions. Once a full SPD is built, we increase the robustness of the layout by identifying and optimizing weak points where the design is likely to fail. Note that our optimization takes place in a space with voids, i.e., containing physically impossible configurations preventing any meaningful measure of success.

We evaluate our framework on real-world contraption examples of increasing complexity, both quantitatively (by computing an integral sampling-based robustness metric and comparing the output of our method against several baselines) and qualitatively (by constructing these examples in real life). Our results demonstrate that we consistently generate robust designs even in high dimensional configuration spaces. In summary, our contributions are (i) a general methodology to optimize chain reaction contraptions; (ii) a general simulation-based measure of robustness to assembly errors; and (iii) a divide-and-conquer method to efficiently compute this function for complex sequences of events.

2 RELATED WORK

Fabrication-aware motion design. Real-life Rube Goldberg machines are a form of kinetic art. In the context of computational design and fabrication, researchers have investigated both kinematics and dynamics of machines from various functional considerations. Notable examples for kinematics include: periodic motion design for assemblies of gears and pulleys [Ceylan et al. 2013; Coros et al. 2013; Zhu et al. 2012] and linkages [Bächer et al. 2015; Thomaszewski et al. 2014], as well as folding [Li et al. 2015] and motion sequences [Garg et al. 2016]. Examples of dynamics-driven design include: adapting internal mass distribution to allow spinning [Bächer et al. 2014], or configuring parts for pneumatic [Ma et al. 2017] and elastic [Chen et al. 2017] objects as well as wind-up toys [Song et al. 2017]. Different from these examples, our contraptions are relatively large and their components may be loosely or even not constrained to each other: rather, collision and friction dynamics are predominant, and difficult to model accurately. Previous works in computational design generally neglect the sensitivity of solutions to modeling bias and fabrication errors, which can be significant when hand assembly with preexisting components is required. As Rube Goldberg machines typically reuse and subvert common objects, we consider the shape and physical properties of all components as given, and aim to obtain a robust design by only optimizing positions and orientations. The work of Furuta et al. [2010] appears closer to our goal, as the authors propose an interface to design kinetic art pieces; however, their tool only allows to previsualize the contraption's behavior, whereas ours actively optimizes the layout to improve the robustness of the chain reaction.

Simulation-based design functions. A growing number of works in computational design combine the parametrization of a 3D model with a black-box physically-based simulator to build functions directly in the *design space*. This approach is general, parallelizable, and can leverage efficient spatial data structures [Shugrina et al. 2015] and machine learning techniques [Umetani and Bickel 2018]. Such design functions are useful for exploration, allowing interactive visualization of complex physical phenomena including flight trajectory [Umetani et al. 2014], fluid flow [Umetani and Bickel 2018], and various material properties such as heat and stress distribution [Schulz et al. 2017]. More general measures of performance and validity can be computed as well to provide user guidance [Shugrina et al. 2015] or enable exploration of design trade-offs [Schulz et al. 2018]. Different from these works, we use simulations to build a measure of *success probability*. Since our goal is to find a design robust to modeling and assembly inaccuracies, we only need to predict the occurrence of functionally important events (e.g., collision, rolling on a track, falling into a container, etc.) and not the exact motion, which allows us to use a faster (albeit less realistic) rigid body simulator. Moreover, we can focus the computational effort around the discovered successful regions of the design space. Lastly, we leverage the causal graph to decompose the global success probability into simpler functions defined on subspaces of the design space, allowing to scale to a high number of dimensions.

Computational models of causality. Event graphs have been used for long to represent storyboards in narrative design. Pioneering work from Kalra and Barr [1992] used directed graphs to analyze and model time and events in computer animation. While our causal graph is inspired from their event graph, our goal is to exploit it to create a real world contraption. Chains of events were also studied for video games and computational narratology, e.g. with the goal of finding a consistent causal order among events [Riedl and Young 2006]. In our setting however, the causal chain is fully specified by the user. In the context of mechanical assemblies, researchers have investigated how representations can help analyze and understand causal relations in mechanisms [Mitra et al. 2010; Schwartz and Hegarty 1996], while more recently, functional graphs have also been used for reconstruction [Lin et al. 2018]. By contrast, our work uses the causal graph to build a measure of robustness that is subsequently used to optimize the design.

3 CONCEPTS AND DEFINITIONS

We start by introducing some key concepts with an illustrative example (see Figure 3). Let us consider the case of a ball initially at rest on a tilted plank. The ball starts rolling on this plank, gains momentum, leaves the plank, and hits the head of a row of dominoes, which all gradually topple until the last one finally comes to a halt.

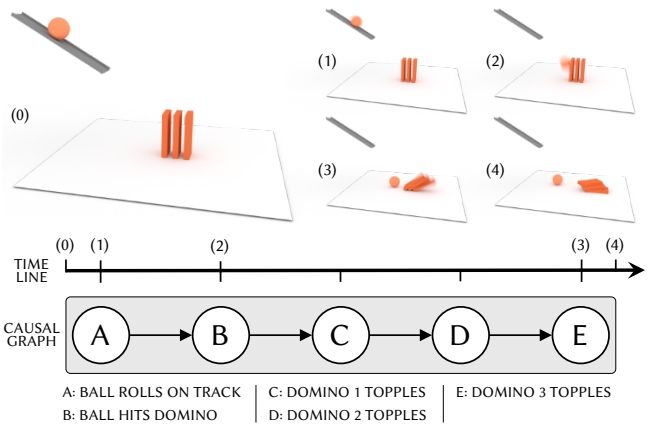

Fig. 3. **SIMPLE scenario.** A ball rolls on a track and triggers the fall of a sequence of dominoes. A succession of snapshots taken from the simulation (top) is matched with the events of this scenario's *causal graph* (bottom). (Ticks along the timeline are uniformly spread for clearer visualization.)

We have just described a *scenario*, the central component of our framework. A scenario is a triplet $\mathbb{S} = \{S, G, D\}$ consisting of a *scene* S, a *causal graph* G and a *design space* D.

Scene. A scene S is a collection of m 3D objects $\{o_i\}_{i=1}^{m}$ laid out in space and organized as a scene graph in which a child object's transform (i.e., position and orientation) is defined in the local frame of reference of its parent. This graph is useful for objects whose initial position is more intuitively described relative to others (e.g., a ball resting on a track). For the sake of convenience, we assume that each scene is made of a small number of *primitives* (in this example, ball, track, dominoes) arbitrarily repeated, combined, and constrained to form an initial setup. Figure 4 shows all the primitives implemented in our system. Each primitive object o_i is built from

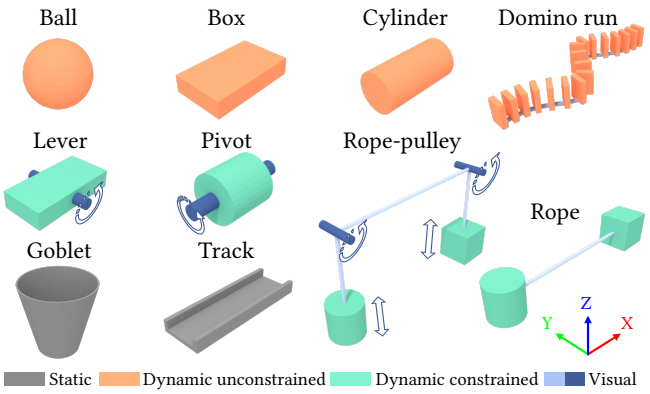

Fig. 4. **Primitive types.** The above primitive types are available to the user in our implementation. The color hues correspond to the different types of behavior in the physically-based simulation. Arrows indicate the motion type allowed by the constraint. Please see the supplementary for details.

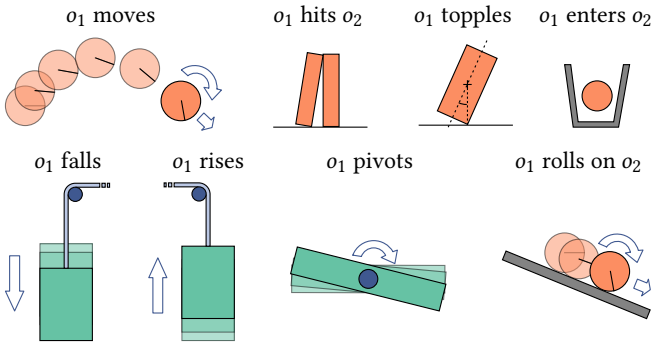

Fig. 5. **Event types.** The above events are supported in our implementation. Each event's condition $c_e(t)$ is a function of the spatial transform (and corresponding time derivative) of the target object(s) at time t. Events may also have a negated version (e.g., 'o_1 stops' being equivalent to 'o_1 does not move'). Please see the supplementary for details.

a set of predefined *design parameters* $\Theta_i = \{\theta_i^j\}$, including both geometric (e.g., length, width) and physical (e.g., mass) parameters.

Causal graph. A causal graph G organizes a collection of *events* expected to happen during the simulation. An event $e = (c_e, s_e)$ has a specific definition in our framework: it is an entity characterized by a *condition* c_e and a *state* s_e. The event condition is a Boolean function of time and one or more objects $c_e(t, o_i, \ldots)$ that evaluates one or more statements about the transform, velocity and/or geometric relationship of these objects at time t. The event state $s_e(t)$ is one of $\{\texttt{asleep}, \texttt{awake}, \texttt{success}, \texttt{failure}\}$. Let $(t_k)_{k\geq 0}$ be the sequence of simulation times. Any event but the first starts with $s_e(t_0) = \texttt{asleep}$, and is triggered awake at some time t_e by the success of all of its predecessor(s). The condition $c_e(t_k)$ is only evaluated while $s_e(t_{k-1}) = \texttt{awake}$. Since we cannot wait indefinitely for the event to happen, we introduce a timeout duration $t^{\max}$ such that

$$s_e(t_k) \leftarrow \begin{cases} \texttt{success} & \text{if } c_e(t_k) = 1 \text{ and } t_e \leq t_k < t_e + t^{\max}, \\ \texttt{failure} & \text{if } c_e(t_k) = 0 \text{ and } t_k \geq t_e + t^{\max}. \end{cases}$$

https://www.overleaf.com/project/5a436e7525871b781d2c84e8 The timeout $t^{\max}$ is manually set to match the longest expected time between two events (2s in our experiments). Figure 5 shows the events currently supported in our system. We note that our formulation induces a discrepancy with the intuitive definition of some events: the act of falling, for instance, is not instantaneous – it lasts a certain amount of time. In our system, however, the switch success or failure is immediate; hence, in such cases, success merely means that the event has started. In practice, we found this formulation sufficiently expressive for our needs.

Events are tied together as nodes of the causal graph, which is a directed acyclic graph with a single root node (i.e., only one starting event) and one or more terminal branches. Using a graph rather than a single timeline allows to account for events happening in parallel in more complex scenarios (see Figure 6). Each edge (e_i, e_j) enforces a temporal ordering of the two events it connects. Hence, for instance, if the ball was to fall on the last domino instead of the first, the causal graph would be violated because the intermediate expected events have not happened. We note that such a causal graph is not necessarily a tree: two branches may converge, signifying that *all* parent events need to happen *before* the current one. The scenario as a whole reaches its *termination condition* when either (i) the last event of each branch has been reached (global success), or (ii) at least one event has timed out (global failure).

Design space. The design space D allows us to explore different realizations of a scenario. While the previously defined primitive design parameters $\{\Theta_i\}$ are fixed, D is composed of the layout parameters of each object relative to its parent in the scene graph. Therefore in the general case $D = SE(3)^m$; in practice however,

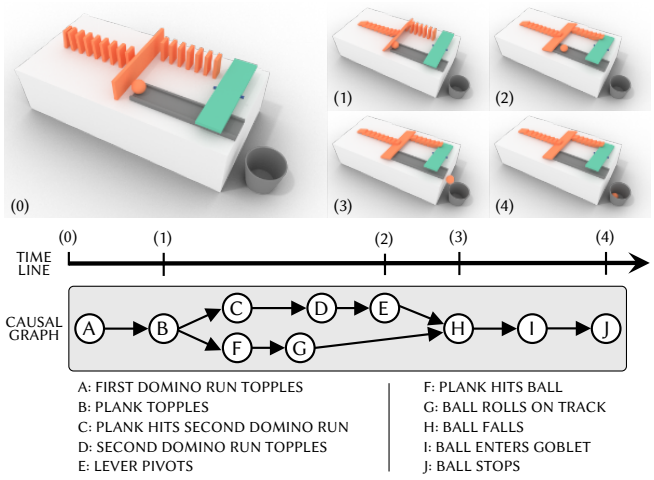

Fig. 6. **BRANCHING scenario.** A first domino run (top left of the view) topples and hits a plank, which in turns triggers two parallel branches: on one side, a second domino run topples and falls on a lever, which pivots; on the other, a ball rolls on a track, passes below the now-raised lever, and falls into a goblet. As in Figure 3, snapshots are matched with events from the causal graph. The two arrows pointing towards event H mean that both E and G need to have happened for H to happen; i.e., the ball can only fall if it started rolling and the lever was raised.

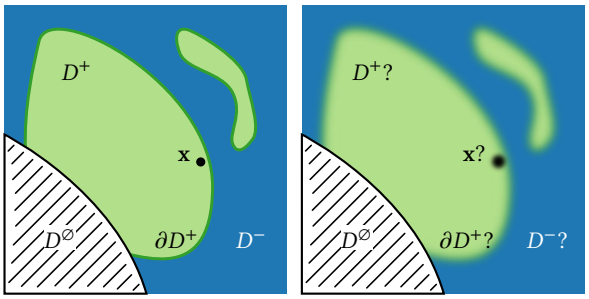

Fig. 7. **Design space.** Abstract representation of a slice of a design space D, divided into success (D^+), failure (D^-) and impossible ($D^\varnothing$) regions. On the left, an idealized view shows each region unambiguously defined, and a layout $\mathbf{x}$ clearly positioned. On the right, uncertainties are taken into account: the realization of a layout is now only close to $\mathbf{x}$ with a certain probability, while its success or failure also become probabilistic. In other words, points near the success boundary ∂D^+ are likely to fail in real life.

some layout parameters can be frozen (e.g., if an object is always in a given plane). This results in d free parameters, each in a predefined range $[a, b]$. For convenience, we assume that each parameter range is normalized so that $D = [0, 1]^d$.

This design space is further structured as follows (see Figure 7-left): first, some regions are forbidden a priori (i.e., before simulation) because they are not physically feasible: typically when two distinct rigid bodies intersect each other at t_0. They form the *impossible region* $D^\varnothing$. Second, the physically feasible space is divided between the *success region* D^+ and the *failure region* D^-, which have no explicit representation in the general case, but can be approximated by sampling scenario instances and simulating them under the supervision of the causal graph. Thus, $D = D^\varnothing \cup D^+ \cup D^-$, with some of these regions potentially disconnected. Note that although we are ultimately only interested in finding D^+, $D^\varnothing$ and D^- are kept distinct for reasons given in Section 7.

While such virtual regions are (implicitly) defined, many sources of bias and error can make the behavior of a virtual scenario diverge from its concrete realization. Coming back to our design space representation, a more telling picture is given Figure 7-top right: uncertainty affects not only the boundary ∂D^+, but also the layout of the scene, or in other words, the position $\mathbf{x}$ in the design space D. The former reflects a wide range of factors throughout the pipeline: it encompasses random measurement errors as well as calibration, modeling and simulation biases, and rounding errors. The latter can originate from random placement error by the user. While a lot of effort can go into reducing these biases and errors (e.g., by choosing a better simulator, being more careful during assembly, etc.), we take a different approach to lower the chance of failure.

4 OVERVIEW

4.1 User Experience

Our method workflow is divided in three steps: scenario definition, probability computation and optimization, and physical realization. Defining a scenario consists in specifying the scene, causal graph, and design space.

The user describes the scene by selecting the primitives, setting their geometric and physical parameters, organizing them as a scene graph (optional), and providing an initial layout (not necessarily a successful one). Setting the fixed parameters requires at least a few real-world measurements (e.g., size and weight). The user then indicates a causal graph by choosing events relative to one or several primitives from a preexisting library, and connecting them by directed edges. Lastly, the user specifies the design space in terms of ranges of values for the six transform parameters (position and orientation) of each primitive. Parameters with no range are locked to their initial values. We note that specifying a scene hierarchy in the first step can help avoid the exploration of large irrelevant portions of the design space; just like accurate models however, a sophisticated hierarchy is not strictly necessary. From this input, a simulation-based success probability is built, allowing to optimize the contraption layout to find a solution robust to errors. The solution is then exported as a printed outline to guide the user during assembly (see supplemental).

4.2 Algorithm Overview

The core of our method is the computation of a success probability distribution conditioned on the layout $\mathbf{x}$, modeled as the class probability output by a binary `success`/`failure` classifier trained on the simulated scenario instances. In Section 5, we propose algorithms to efficiently find successful points in the design space, train the classifier and improve its accuracy via *active learning*. Section 6 then demonstrates how this method can scale in high-dimensional design spaces using a divide-and-conquer method where the *global* probability of success is decomposed into conditional probabilities of success of *individual* events. Each of these components is restricted to the design parameters that really influence the corresponding event, thus reducing the dimensionality of their respective design subspace. Finally, in Section 7, we take the scenario instance with the highest success probability and refine it using an SPD-based global energy that we minimize under physical validity constraints.

5 SUCCESS PROBABILITY COMPUTATION

We consider a scenario $\mathbb{S}$ where objects are laid out according to a vector of parameters $\mathbf{x} \in D$, with $D = D^\varnothing \cup D^+ \cup D^-$. Our goal is to build an approximation of the success probability distribution (SPD) $\Pr(D^+|\mathbf{x})$ using the data provided by the simulator.

Our key observation is that we can avoid explicitly modeling all the sources of uncertainty listed in Section 3. As long as we have a synthetic approximation of the outcome function (or 'oracle') $h : D \setminus D^\varnothing \to \{\texttt{failure}, \texttt{success}\}$, we can build a probability distribution based solely on a learned approximation of the success region D^+. Maximizing such a probability allows to 'push' a point $\mathbf{x}$ deeper inside D^+, thus increasing *robustness* to bias and assembly errors. Indeed, moving away from the boundary reduces the dependence between the variance of the *input* (layout parameters $\mathbf{x}$) and the variance of the *output* (`success`/`failure` flag), which is a central principle of the Taguchi methods for robust design [Rao et al. 2008]. In short, the idea is to make the output *less sensitive* to parameter variations, thus avoiding the need to explicitly account for all possible sources of error.

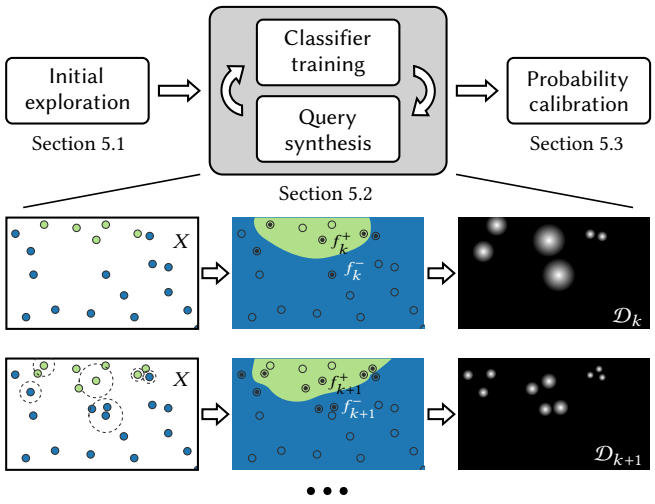

Fig. 8. **Building the SPD.** Top: main steps of our global SPD approximation method. Bottom: detail of the active learning loop: at iteration k, we use the current dataset X (left) to train an SVM (middle; classification shown as background colors). The new support vectors (black dots) are used to build a new distribution $\mathcal{D}_k$ (right; probability density shown in white) that encourages additional sampling where the classifier is the most uncertain (i.e., near the boundary).

The SPD is built by calibrating the score of a Support Vector Machine (SVM) classifier trained on simulated samples. We chose SVMs not only for their robustness to overfitting in high-dimensional spaces, but also because they mesh very well with our active learning strategy, as described next (see Section 8 for a comparison with baseline methods).

In this section, we consider a single SPD computed on the entire design space. Our method, as shown in Figure 8-top, starts with an *initial exploration* of the design space (Section 5.1) by adaptively sampling D and running simulations to find a minimal number of successful instances. The main body of the algorithm (Section 5.2) then follows an *active learning* strategy in two alternating steps: first, during *classifier training*, a non-linear kernel SVM is trained on the current dataset to approximate ∂D^+; second, during *query synthesis*, the decision function of the SVM helps identify uncertain regions of the design space which are then probed to augment the dataset. As a final step (Section 5.3), we apply a *probability calibration* to map the final SVM score to a class probability for `success`.

During the entire process, new candidate samples are filtered to discard the physically impossible ones (e.g., those where rigid bodies intersect). Physical validity does not need to be learned because it is enforced by constraints during layout optimization (see Section 7). Valid scenario instances are simulated under supervision of the causal graph, yielding a global `success` or `failure` label.

5.1 Initial Exploration by Adaptive Sampling

The goal of the exploration stage is to discover an initial number of successful instances N^+ (200 by default). Algorithm 1 details our adaptive sampling method. We iteratively grow a list of physically valid sample points $X = \bigcup_k X_k$, where X_k is the list of N_k^s points

Algorithm 1 Exploration by adaptive sampling.

```
X ← ∅
{n_i^+} ← ∅
D_0 ← SobolSequence()
k ← 0
enough ← false
while k ≤ k^e and not enough do
    X_k ← SamplePhysicallyValid(D_k, N_k^s)
    X ← X ∪ X_k
    // Simulate each sample to get their # of successful events.
    {n_i^+} ← {n_i^+} ∪ {GetNumSuccEvents(x) ∀x ∈ X_k}
    if |{i : n_i^+ = n}| ≥ N^+ then
        enough ← true          // Because x_i ∈ D^+ ⇔ n_i^+ = n.
    else
        I ← ArgNMax({n_i^+}, N^+)
        w ← {n_{I_i}^+ / Σ_{j∈I} n_j^+ ∀i ∈ [1 .. N^+]}
        D_{k+1} ← Σ_{i=1}^{N^+} w_i N(x_{I_i}, diag(σ|b − a|))
        k ← k + 1
    end if
end while
```

(10 by default) drawn from distribution $\mathcal{D}_k$ at step k, until either (i) the number of successful points $|\{\mathbf{x} \in X : \mathbf{x} \in D^+\}|$ reaches N^+, or (ii) after k^eiterations (500 by default, which was never reached in our experiments). The initial sampling X_0 is drawn from the quasi-random Sobol sequence [1967] ($N_0^s = 500$ by default). We use the causal graph G to orient the sampling towards the most relevant regions of the design space: for each new sample point $\mathbf{x}_i \in X$, we simulate the corresponding scenario instance and record the number of successful events n_i^+ (between 0 and n, where n is the total number of events); in other words, n_i^+ is the number of causal graph nodes whose state is `success` after simulation of a single instance $\mathbf{x}_i$. Then, we select the top N^+ values from $\{n_i^+\}$, and note I their indices. We use them to build a mixture of Gaussians

$$\mathcal{D}_k \sim \sum_{i=1}^{N^+} w_i \mathcal{N}(\mathbf{x}_{I_i}, \mathrm{diag}(\sigma)), \tag{1}$$

with a diagonal factor $\sigma = 0.01$ by default. There is one Gaussian per sample $\mathbf{x}_i$; their weight w_i reflects the relative success of $\mathbf{x}_i$ with $w_i = n_{I_i}^+ / \sum_{j \in I} n_j^+$. This formulation focuses exploration around the current best *partially* successful scenario instances, which effectively helps it reach regions containing full successes even in high-dimensional design spaces. As in reinforcement learning, we can tune the balance between *exploration* and *exploitation*: for instance, a higher σ favors exploration, as points are sampled further from the current best. Moreover, as new successful data points are found, only taking the top N^+ points at each step means that our method progressively favors exploitation of full successes over exploration of partial successes. Lastly, we note that exploration can be made easier by providing a more detailed causal graph, as it yields a finer-grained distinction between partially successful instances.

5.2 Classifier Training and Query Synthesis

The goal of this step is to obtain a classifier with sufficient accuracy (90% in our experiments). We iteratively train an SVM and query new design space points until we reach either the target accuracy

Algorithm 2 Classifier training and query synthesis.

$\mathbf{y} \leftarrow$ ComputeLabels(X) // Simulate each sample.
$k \leftarrow 0$
// Initial classifier training
$f_k, \{\hat{\mathbf{x}}_i\}_{i=1}^{v}, U, acc \leftarrow$ TrainEstimator($X, \mathbf{y}$)
while $k \leq k^{\mathrm{l}}$ **and** $acc \geq 0.9$ **do**
  // Query synthesis
  $\mathbf{w} \leftarrow \{|f_k(\hat{\mathbf{x}}_i)| \,/\, \sum_{j=1}^{v} |f_k(\hat{\mathbf{x}}_j)| \;\forall i \in [1\,..\,v]\}$
  $\mathcal{D}_k \leftarrow \sum_{i=1}^{v} w_i \mathcal{N}(\hat{\mathbf{x}}_i, |f_k(\hat{\mathbf{x}}_i)|U)$
  $X_k \leftarrow$ SamplePhysicallyValid($\mathcal{D}_k$, $10N^s$)
  $I \leftarrow$ ArgNMin($\{|f_k(\mathbf{x}_i)| \;\forall \mathbf{x}_i \in X_k\}$, N^s)
  $X'_k \leftarrow \{\mathbf{x}_i \in X_k : i \in I\}$
  $X \leftarrow X \cup X'_k$
  $\mathbf{y} \leftarrow \mathbf{y} \cup$ ComputeLabels(X'_k) // Simulate each new sample.
  // Classifier training
  $f_k, \{\hat{\mathbf{x}}_i\}_{i=1}^{v}, U, acc \leftarrow$ TrainEstimator($X, \mathbf{y}$)
  $k \leftarrow k + 1$
end while

or the maximal number of iterations k^{l} (5 by default), as illustrated in Figure 8-bottom and detailed in Algorithm 2. The list of samples is noted again $X = \bigcup_k X_k$, where X_0 is the set of sample points obtained from the initialization step.

Classifier training. Following common machine learning practices, the dataset is first standardized (i.e., transformed to zero mean and unit variance). The SVM classifier has two hyperparameters: C, the regularization parameter, and γ, the inverse radius of influence of each support vector. We automatically select their optimal value from a logarithmic range using stratified 3-fold cross-validation. The accuracy at step k is given by the cross-validation score.

Query synthesis. In active learning, a learner is able to improve its accuracy by querying an 'oracle' for data points that were not part of its original training set [Settles 2012]. The query, however, comes at a computational cost; the algorithm thus needs to choose its queries wisely in order to improve its performance. In our case, where the oracle is a simulator, any physically valid point in the design space can be queried to obtain `success`/`failure` label; this problem is called *query synthesis*. A common strategy consists in reducing estimator uncertainty by querying regions of which the learner is the least certain about. For an SVM, this region is easy to find: it lies near the classification boundary, where the current SVM *decision function* $f_k : \mathbb{R}^d \rightarrow \mathbb{R}$ is close to 0. This boundary is itself defined by the v support vectors $\{\hat{\mathbf{x}}_i\}_{i=1}^{v}$. Hence, after training the SVM at step k, we draw samples from the mixture of Gaussians

$$\mathcal{D}_k \sim \sum_{i=1}^{v} w_i \mathcal{N}(\hat{\mathbf{x}}_i, |f_k(\hat{\mathbf{x}}_i)|U), \tag{2}$$

where U is the inverse of the diagonal scaling matrix used for standardization. The Gaussians are weighted by $|f_k|$ with: $w_i = |f_k(\hat{\mathbf{x}}_i)|/\sum_j |f_k(\hat{\mathbf{x}}_j)|$, thus giving sampling priority to the farthest support vectors (i.e., where the boundary is most uncertain). The decision function f_k allows to scale the Gaussians to sample the appropriate neighborhood around each support vector $\hat{\mathbf{x}}_i$. However, since samples are taken in all directions around each $\hat{\mathbf{x}}_i$, it is unlikely that many of such samples will actually lie near the boundary. Therefore, we first sample (without simulating) $10N^s$ points using $\mathcal{D}_k$, and only keep the N^s ones having the smallest $|f_k(\mathbf{x})|$ value.

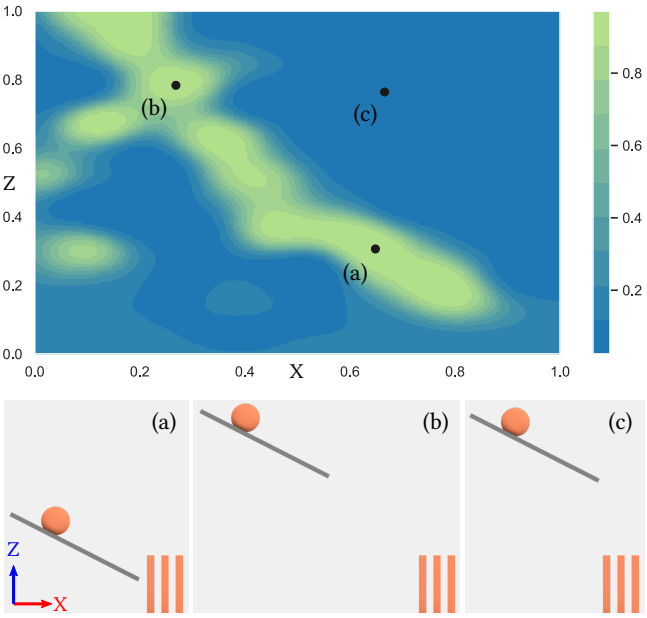

Fig. 9. **SPD Visualization.** A slice of our learned SPD approximation (top), with three instances (bottom) sampled from the design space of our simple scenario from Figure 3. The two dimensions of D shown here are the normalized X and Z coordinates of the track center (fixed slope). The color range is discretized for clearer visualization of the isolevels. We observe that both (a) and (b) belong to an elongated region where the ball hits the top left corner of the first domino, while in (c) it clearly misses the domino run.

5.3 Probability Calibration

While we do not strictly need to compute a probability when there is only one classifier (as the SVM decision function can be maximized directly), the divide-and-conquer method of Section 6 requires probabilities to be output by each classifier so that they can be meaningfully combined and/or compared. The decision function f, however, approximates a signed distance to a regularized boundary and not a probability. Nevertheless, the success probability distribution $\Pr(D^+|\mathbf{x})$ can be approximated by applying a continuous transformation to the decision function, following a method known as Platt Scaling [Platt 1999] that fits a logistic regression model to the classifier's scores. Specifically, a maximum likelihood optimization is performed to calibrate the coefficients $\alpha, \beta \in \mathbb{R}$ in

$$\Pr(D^+|\mathbf{x}) = (1 + \exp(\alpha f(\mathbf{x}) + \beta))^{-1}. \tag{3}$$

After this calibration, we can evaluate the SPD of a new scenario instance $\mathbf{x}$ by computing $\Pr(D^+|\mathbf{x})$ (see Figure 9).

6 EXTENSION TO COMPLEX CAUSAL CHAINS

Chain reactions of reasonable visual complexity can easily depend on several dozens of layout parameters. To help the SPD computation scale to such a high number of dimensions, we propose a

divide-and-conquer method where the global SPD for a scenario $\mathbb{S} = \{S, G, D\}$ is broken down into a set of success probabilities of simpler sub-scenarios $\mathbb{S}_i = \{S, G_i, D_i\}$, where G_i is a subgraph of G, and D_i is a subspace of D with dimension $d_i < d$. This inequality is key to the scalability of our method, as it reduces the combinatorial complexity of exploring D and approximating the SPD.

Before detailing our extended pipeline, let us demonstrate how to factorize $\Pr(D^+|\mathbf{x})$. By definition, a scenario is successful if and only if each event happens in the correct order; this is equivalent to each node of the causal graph reaching success after its parent(s) did the same. Formally, if we associate to each causal graph event e_i the random variable E_i giving the final state of this event after simulation, we are trying to decompose the joint probability

$$\Pr(D^+|\mathbf{x}) = \Pr\left(\{E_i\}_{i=1}^n = \texttt{success} \mid \mathbf{x}\right), \tag{4}$$

where n is the number of events, and '$\{\cdot\} = \texttt{success}$' means that all elements of the set are equal to success. To do so, let us consider the directed graphical model $\mathcal{G}$ obtained by substituting each node e_i in G by the corresponding E_i. By construction of the causal graph, the success of the parents is equivalent to the success of all ancestors; therefore the success of any E_i, given $\mathbf{x}$, only depends on its parents' success. In other words, $\mathcal{G}$ satisfies the local Markov property, expressed as conditional independence:

$$\forall E_i \in V(\mathcal{G}): \quad E_i \perp\!\!\!\perp \{\mathrm{nd}(E_i) \setminus \mathrm{pa}(E_i)\} \mid \mathrm{pa}(E_i), \mathbf{x},$$

where $V(\mathcal{G})$ is the set of vertices in $\mathcal{G}$, and $\mathrm{nd}(E_i)$ and $\mathrm{pa}(E_i)$ are respectively the set of non-descendants and parents of E_i in $\mathcal{G}$. It can be shown that for directed acyclic graphs, this property is notably equivalent to the factorization of joint probabilities on the graph nodes into conditional probabilities given the node's parents [Lauritzen 2001]. In particular, Eq. 4 yields the SPD factorization:

$$\Pr(D^+|\mathbf{x}) = \prod_{i=1}^{n} \Pr\left(E_i = \texttt{success} \mid \mathrm{pa}(E_i) = \texttt{success}, \mathbf{x}\right). \tag{5}$$

We call the i-th factor of the above product the E_i-CSPD (where 'C' stands for 'Conditional'). Of course, if we were to approximate each E_i-CSPD as we approximate the global SPD, the complexity would be multiplied by n, rather than decreased. To effectively reduce it, we make the key observation that given $\mathrm{pa}(E_i) = \texttt{success}$, having $E_i = \texttt{success}$ depends on few design parameters; in other words, the variance of each E_i-CSPD mostly only occurs in a relatively low-dimensional subspace $D_i \subset D$. Computing this *subspace mapping* (see second block in Figure 10) is described next.

First, as a pre-processing step, we identify which E_i are *quasi-deterministic*, i.e., nearly always successful when their parents are successful. Given X the sampling obtained after initial exploration (Section 5.1), which contains a minimum number of globally successful samples, we compute the success rate of each E_i,

$$\rho_i = N_i^+ / (N_i^+ + N_i^-),$$

where N_i^+ and N_i^- are respectively E_i's number of successes and failures in X, and assign $D_i = \emptyset$ to each E_i having $\rho_i \geq 0.95$. The corresponding E_i-CSPDs are subsequently set to 1. We measure correlations between the remaining E_i and the design parameters using mutual information [Cover and Thomas 2006], allowing us to discover linear as well as nonlinear relationships. In our experiments, parameters were selected if their mutual information with E_i was greater than 0.2.

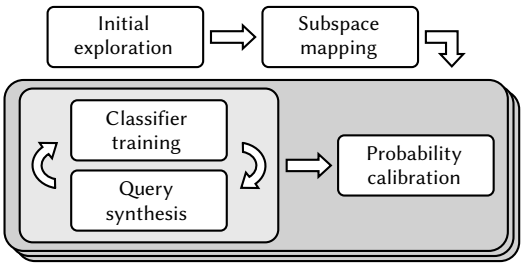

Fig. 10. **Extended SPD building pipeline.** We propose an extended version of our SPD building to scale with a high-dimensional design space.

To approximate the non-constant factors, we apply the following method, illustrated as the stack of blocks of the extended pipeline in Figure 10. For each E_i-CSPD, we consider the subgraph G_i containing only e_i and its ancestors. We run the training and boundary consolidation loop (Section 5.2) using G_i, with two slight modifications: (i) to satisfy the conditional probability in Eq. 5, we only keep the samples satisfying $\mathrm{pa}(E_i) = \texttt{success}$. We know that such samples exist in the initial set for each E_i, since some of the initial samples are globally successful over G. (ii) During boundary consolidation, we restrict sampling to the corresponding D_i by simply taking the indices of the parameters not in D_i, and setting their scale factor in U to 0: therefore, only the parameters in D_i have non-zero variance. Lastly, we calibrate each probability as in Section 5.3.

As a result of the above steps, the SPD approximation can be computed with a significant complexity reduction as

$$\Pr(D^+|\mathbf{x}) \approx \prod_{i=1}^{n} r_i(\mathbf{x}), \tag{6}$$

where $r_i(\mathbf{x}) = \Pr\left(E_i = \texttt{success} \mid \mathrm{pa}(E_i) = \texttt{success}, \phi_i(\mathbf{x})\right)$ and $\phi_i : D \to D_i$ is the subspace mapping.

7 LAYOUT OPTIMIZATION

Once the SPD has been computed, we take the sample point with the highest success probability as our most robust current solution. Although this design is indeed already quite robust, we further refine it by applying a nonlinear optimization. While the factorization could allow us, in theory, to optimize each E_i-CSPD separately, in general they are not separable because their subspaces D_i overlap. Instead, we aggregate all components into a global energy to find

$$\begin{aligned} \min_{\mathbf{x} \in D} \quad & E^{\mathrm{r}}(\mathbf{x}) \\ \text{subject to} \quad & \mathbf{C}^{\varnothing}(\mathbf{x}) \geq \mathbf{0} \end{aligned} \tag{7}$$

with

$$E^{\mathrm{r}}(\mathbf{x}) = -\mathcal{S}_\alpha \circ \mathbf{r}(\mathbf{x}), \tag{8}$$

where $r_i(\mathbf{x})$ is the E_i-CSPD approximation from Eq. 6, and the function $\mathcal{S}_\alpha : \mathbb{R}^n \to \mathbb{R}$ is the smooth minimum [Lange et al. 2014]: $\mathcal{S}_\alpha(\mathbf{z}) = \sum_i z_i e^{-\alpha z_i} / \sum_i e^{-\alpha z_i}$ with $\alpha \in \mathbb{R}^+$ controlling the importance of the smallest component of $\mathbf{z}$. The reason for this choice (rather than taking the product, as in Eq. 6) comes from the observation that a chain is only as strong as its weakest link. This entails

that priority should be given to maximizing the minimal E_i-CSPD value, rather than maximizing their product.

The constraint vector $\mathbf{C}^{\emptyset}$ ensures that the design stays physically valid. It aggregates (i) penetrations between distinct rigid bodies in the scene, and (ii) primitive-specific constraints, such as ensuring that the layout of a rope-pulley is compatible with the rope length. The former is easily obtained from the rigid body simulator, as penetrations are needed to compute the reaction force between colliding shapes [Coumans 2018]. While we could have learned invalid configurations when computing the SPD, thus integrating the constraint into the energy, we chose to explicitly enforce physical validity during optimization for two reasons: first, validity would not have been guaranteed since we only approximate the SPD, and second, *impossibility* and *failure* are two distinct concepts. Indeed, the probability of success does not necessarily decrease as a design $\mathbf{x} \in D^+$ is moved closer to $D^{\emptyset}$: for example, putting two successive dominoes in contact might robustly ensure that both topple.

As described earlier, the initial solution is the sample point with the highest SPD value. Assuming that this guess is close enough to the global minimum, we use Sequential Least-Squares Quadratic Programming [Kraft 1988] to find the optimal design.

8 RESULTS AND EVALUATION

8.1 Implementation

Our framework was implemented in Python 3. 3D models are generated by *OpenSCAD* and simulated using *Bullet Physics*. Most computations, including optimization, are done with *Numpy* and *Scipy* while *Scikit-learn* is used for the SVM classifier and the other machine learning tools. Our graphical interface (described next) was implemented with the *Panda3D* game engine. Code and additional details are included as supplementary material.

Interface. Our graphical interface allows users to define the scene and causal graph. They can instantiate the primitives described in Section 3 and define the initial layout. We provide specific tools to help designing the most complex primitives: for instance, domino runs can be generated by simply drawing a path and having dominoes automatically distributed along it. Simulation can be run in real time, providing visual feedback during design. Once the scene is complete, users define the causal graph by instantiating events, linking them to the objects in the scene and drawing directed edges between them, and finally specifying necessary parameters such as design space ranges. We note that the specific layout designed in the GUI does not need to be a fully successful one: all that matters is that there is a successful region somewhere in the design space.

Export. The final optimized layout is automatically exported as PDF sheets to be printed, in order to provide guidance during assembly. The outline is obtained by projecting the convex hull of each object onto a vertical or horizontal plane, depending on the user's need. A pattern of grey lines is added to the background to help join the paper sheets after printing. Guidance patterns for the examples presented in this paper are provided as supplementary material.

8.2 Qualitative Evaluation

We designed, implemented, and physically realized a number of scenarios to validate our pipeline. We present here a selection of four examples, focusing on those most challenging due to complex movements and/or event synchronizations. Note that simple domino runs following long low-curvature paths are easy to design (as commonly seen in online videos) and hence were avoided in these experiments. The four presented sequences are called BALLRUN, CAUSALITYSWITCH, LONGCHAIN, and TEAPOTADVENTURE, in increasing order of complexity. The first two resulted in a successful real-life run after a single try; the others, due to their higher complexity, required a more careful adjustment of the parts to the printed layout and succeeded after 4-5 trials. We note that this number is much lower that the dozens of trials usually shown in behind-the-scenes videos found online. In this section, we describe each scenario at a high level, while further details and video clips are provided in the supplemental material.

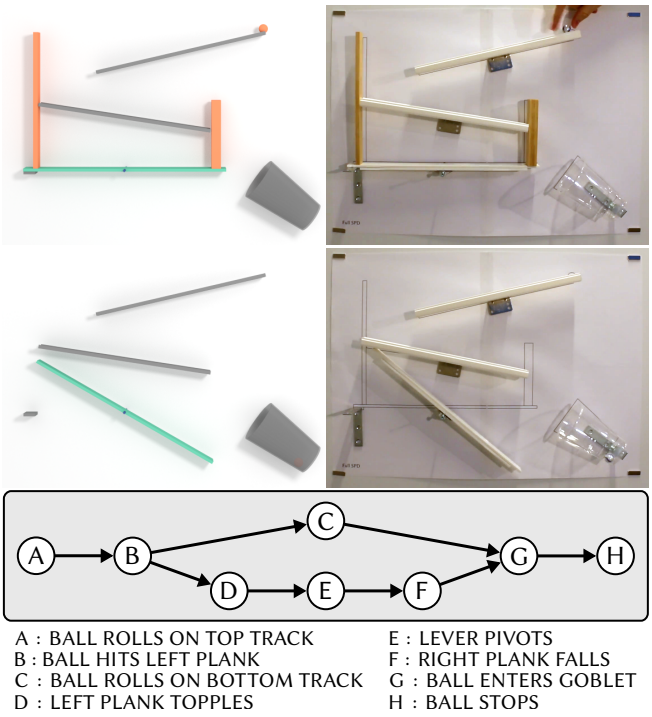

Fig. 11. **BALLRUN scenario.** A ball rolls on a sequence of tracks and falls into a goblet. To open the entrance, it needs to hit the left plank, to make it fall, triggering the fall of the right plank via the bottom lever. Timing control is needed to ensure that the right plank falls before the ball reaches the goblet. Top: initial state; bottom: final state. See supplemental video.

In BALLRUN, the goal is to get the ball to roll down the tracks and fall into the cup. However, a wooden plank blocks the entrance of the cup. Synchronization is needed along the causal graph to realize the following sequence: the ball hits the first wooden block to get the lower support rotating, but the ball has to travel slowly enough to allow the other wooden block to fall, thus opening the pathway to the target cup (see Figure 11 but best seen in the video).

The CAUSALITYSWITCH contains two longer chains running in parallel until a 'domino switch' allows only the fastest path to go through by blocking the way of the other. One path is a wave-like

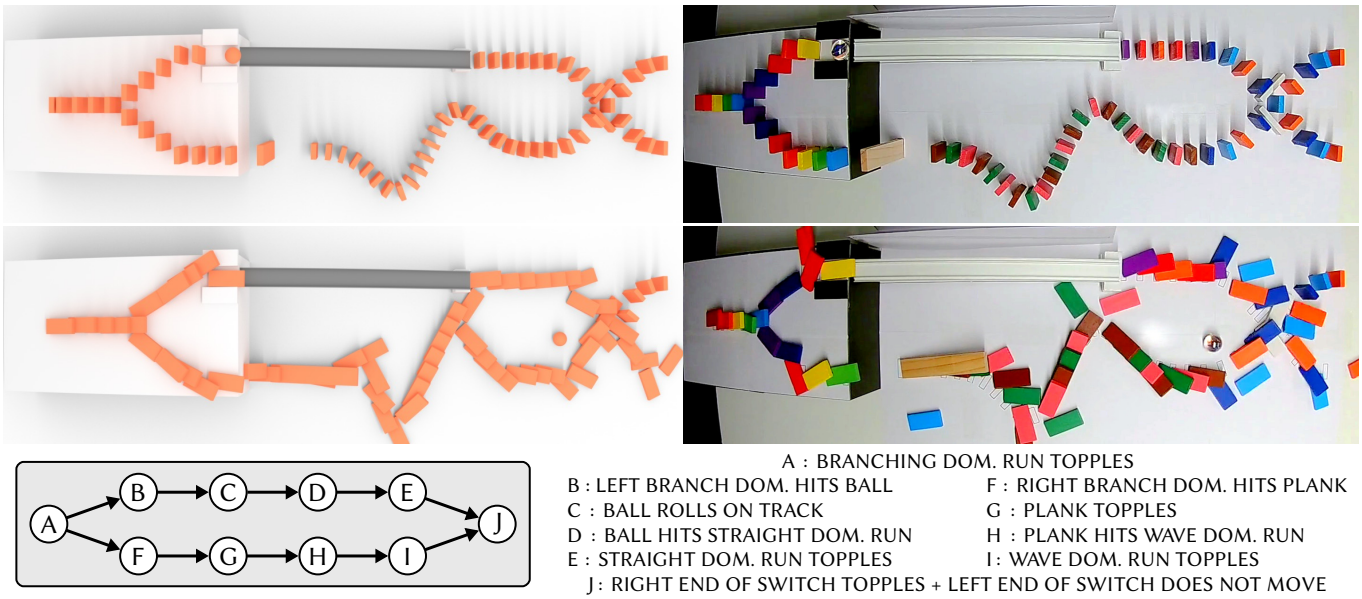

Fig. 12. **causalitySwitch scenario.** A branching domino run topples and triggers two parallel branches. On one side ('left'), a ball rolls down a track and topples a short straight domino run. On the 'right' side, a plank falls and topples a long, curved domino run. Whichever side is faster (in this figure: the left one) triggers the 'domino switch', closing the path of the other side.

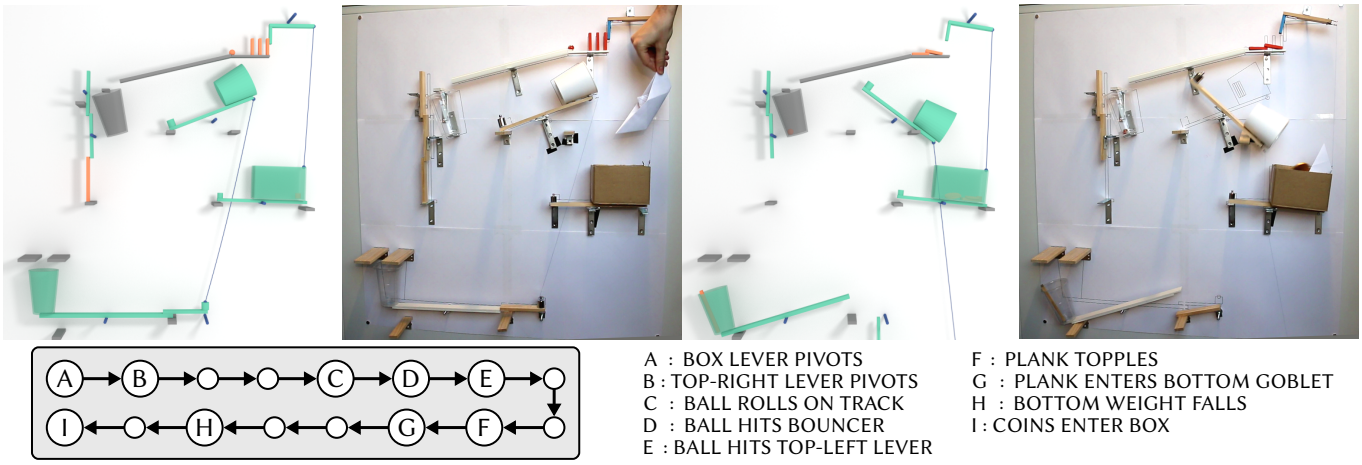

Fig. 13. **longChain scenario.** A long chain of events that is triggered by the box weight pivoting an initial lever. Dominoes topple and send a ball onto a track to go hit a lever, which makes a plank topple through a narrow entrance into a goblet. This triggers the fall of the bottom weight, and in turn, the central goblet pivots to let coins fall into the box. Please refer to the supplementary video.

chain of dominoes, while the other involves a ball rolling on a track. This experiment demonstrates that we can choose to optimize for either side to be the fastest by modifying the causal graph accordingly. Figure 12 shows the causal graph along with the final and initial state with the 'ball' side successfully reaching the switch first. Both versions endings are shown in the supplementary video.

The longChain is a long linear sequence of events. Under the weight of the box's contents, a lever pivots to topple the domino run that, in turn, nudges the ball onto the track. The ball bounces on a small platform and hits a lever, resulting in the orange plank toppling. The plank tumbles and falls through a narrow entrance into a goblet, moving a pivot that makes the bottom weight fall, tugging on the central goblet from which coins fall into the box. Figure 13 shows the causal graph along with initial and final states of the optimized layout, while the full run is shown in the video.

The teapotAdventure is the most complex example shown in this paper (see Figures 1 and 14). As seen from the causal graph, success for this contraption requires a very challenging synchronization between delicate event chains. In short, there are two balls involved (one initially free, and one in a cage), that need to escape

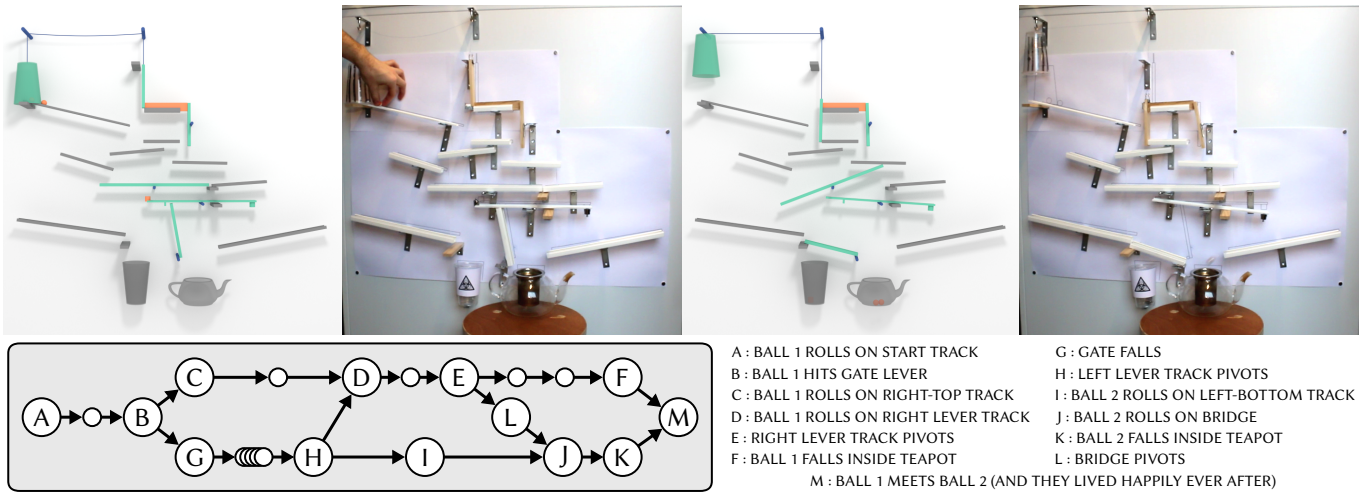

Fig. 14. **teapotAdventure scenario.** See Figure 1 for initial layout. This figure shows rendered versus assembled layouts for start and end frames of the optimized layout. Ball 1 triggers the fall of the middle gate, releasing ball 2. Ball 2 hits the gate and takes the left route, falling on a lever so that ball 1 can roll underneath. Ball 2 then makes the central weight fall, liberating the bottom bridge. Both balls then meet in the teapot. Please see the supplementary video.

the contraption and reach the teapot by opening each other's path along the way. Note that such a sequence is very difficult to manually author without computational guidance as proposed in this paper. On a lighter note, this kind of contraption illustrates the narrative power of Rube Goldberg machines, such as can be seen, e.g., in the Japanese show *PythagoraSwitch*.

8.3 Quantitative Evaluation

Local and global robustness. We compare the output of different methods with the following measures of robustness. Let $\mathbb{S}$ be a scenario with design space D, and $X \subset D$ a set of points decomposed as $X = X^+ \cup X^- \cup X^\emptyset$ (respectively successes, failures and impossible instances). The *local robustness* $\rho_l : D \times [0,1] \to [0,1]$ is defined as

$$\rho_l(\mathbf{x}, \epsilon) = \begin{cases} \frac{|\mathcal{B}_\epsilon(\mathbf{x}) \cap X^+|}{|\mathcal{B}_\epsilon(\mathbf{x}) \cap \{X^+ \cup X^-\}|} & \text{if } \mathbf{x} \in D \setminus D^\emptyset, \\ 0 & \text{otherwise,} \end{cases}$$

where $\mathcal{B}_\epsilon(\mathbf{x})$ is the ball of radius ϵ centered at $\mathbf{x} \in D$. In this formulation, ϵ represents a uniform error on the layout parameters, while ρ_l is the success rate in the neighborhood $\mathcal{B}_\epsilon(\mathbf{x})$. The global robustness $\rho_g : D \to [0,1]$ is then defined as

$$\rho_g(\mathbf{x}) = \int_0^1 \rho_l(\mathbf{x}, \epsilon) d\epsilon.$$

In practice, the evaluation dataset X is computed from a relatively dense sampling of the design space (with 100K points for causalitySwitch, and 1M points for the three others). Additional details on the computation of ρ_l are provided in the supplemental.

Comparison with baseline methods. We define three baseline methods to compare against our technique:

(B1) Uniformly sample and simulate points in D until a successful configuration is found.

(B2) Uniformly sample and simulate points in D, compute the local robustness $\mathbf{x} \mapsto \rho_l(\mathbf{x}, 0.1)$ for each, and take the best one.

(B3) Run a Bayesian Optimization with a Gaussian Process prior [Shahriari et al. 2016] (initialized with uniform sampling) using $\mathbf{x} \mapsto \rho_l(\mathbf{x}, 0.1)$ as an objective function.

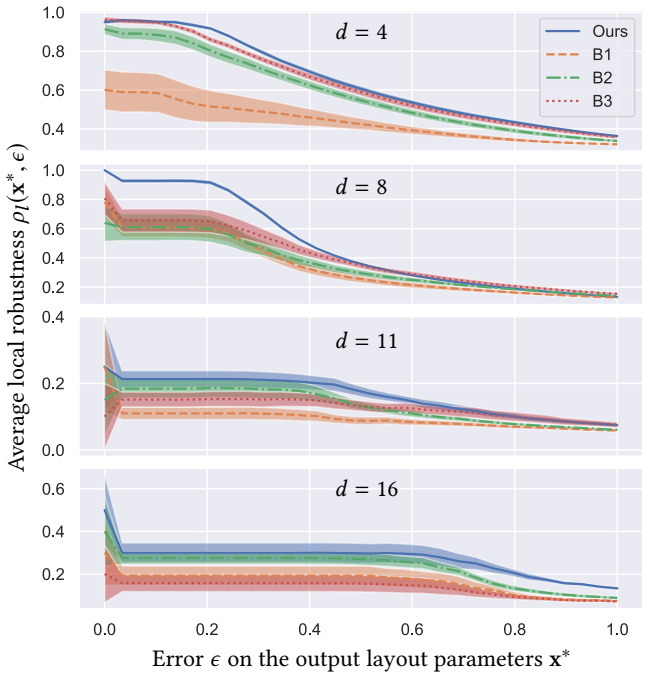

Fig. 15. **Local robustness plots.** Average local robustness $\rho_l(\mathbf{x}^*, \epsilon)$ as a function of the error ϵ on the layout parameters $\mathbf{x}^*$ output by each method, for problems of different dimension d. Each curve is surrounded by a standard error of the mean interval (with $T = 10$ runs per method). Scenarios are respectively causalitySwitch, ballRun, longChain and teapotAdventure.

Table 1. **Experiments statistics.** Both global robustness and processing time are averages of $T = 10$ random trials. The simulation step was 0.002s.

Scenario	Number of param. d	Max. simu. time (s)	Simu. budget B	Global robustness $\rho_g(\mathbf{x}^*)$				Processing time (s)			
				B1	B2	B3	Ours	B1	B2	B3	Ours
causalitySwitch	4	4	614	0.44	0.59	0.63	**0.65**	**263**	282	463	364
ballRun	8	4	1120	0.34	0.35	0.40	**0.48**	59	**52**	786	153
longChain	11	5	1915	0.09	0.13	0.13	**0.16**	1499	1597	1956	**636**
teapotAdventure	16	8	2420	0.16	0.23	0.14	**0.27**	**536**	**536**	988	779

Each baseline is given the same 'simulation budget' B computed from our own method: we first run our method T times with a different random generator seed each time (with $T = 10$ in our experiments), and compute B as the average number of simulations carried out. Each baseline is then also run T times until B is reached (or until a success is found for baseline B1). The final local robustness curve is the average curve of the different trials.

Results in Figure 15 and Table 1 show that our method outperforms the baselines for all four scenarios presented in Section 8.2. Interestingly, the Gaussian Process optimization (B3) appears to perform worse than the simpler uniform sampling (B2) in higher dimensions, which could be due to the presence of holes (i.e., components of $D^{\varnothing}$) in the optimization landscape. Additionally, we demonstrate the positive impact of SPD factorization in Figure 16.

8.4 Limitations and Future Work

Our method is a first step towards the robustification of machines involving complex sequences of events. It presents several limitations that open exciting avenues for future improvements.

First, while our method does increase robustness to modeling biases, some neglected physical effects can still significantly influence the final execution, such as vibrations created by collisions, or tracks sagging under the weight of balls. Using a more realistic physical simulator would allow to take these secondary effects into account, as well as support more advanced primitives and events (e.g., involving cloths, fluids or fire), at the price of an increase in simulation time. Second, it is assumed that the design space contains a continuous region of solutions, implying that the expected sequence of events is indeed possible. While it is unclear how this could be checked at design time, some heuristics could be developed to give feedback to the user and avoid the most obvious mistakes. Third, a better understanding of dysfunctional cases could help use simulations more efficiently, by stopping runs early if failure is expected. In the same vein, when building CSPD factors, further analysis could allow restarting simulations mid-run, i.e., from an initial state specific to each event, rather than starting all simulations from the same point. Lastly, the SPD computation could be improved in several ways. A clear next step is to try alternative classification models, such as neural networks, although SVMs present distinct advantages, such as the speed of training, the inherent ability to avoid overfitting, and the ease of sampling new points near the boundary for active learning. Performance could be improved by (i) using models supporting online training (i.e., allowing to train on new samples without retraining over the entire dataset), and (ii) using a more advanced stopping criterion for active learning.

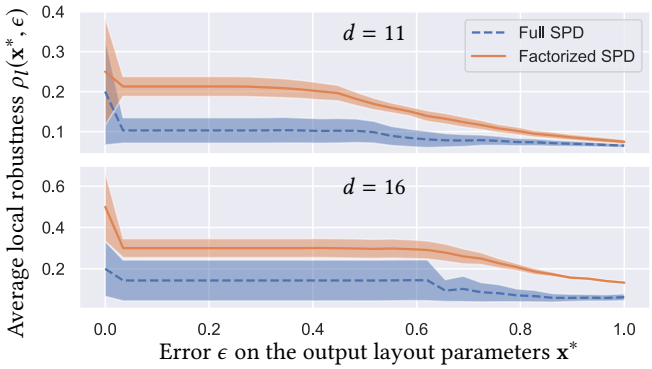

Fig. 16. **Full versus factorized SPD.** We use the same metric as in Figure 15 to compare two versions of our method for the longChain and teapotAdventure scenarios. 'Full' means that a single SVM was used over the entire design space, while 'Factorized' follows the method in Section 6.

Besides these technical points, our work may appear very specific both in scope and audience. On one hand, Rube Goldberg machines are indeed a niche application; however, our work could be applied to more useful real-world mechanisms involving sequences of events, such as locks, pop-up tents and ejection seats, to name a few. Indeed, while we chose to limit our design space to layout parameters (as components were assumed to be preexisting), nothing prevents the inclusion of other geometric and physical parameters (although additional physical validity checks may have to be performed). The part requiring an extension would be the causal graph itself: indeed, it would need to support additional features such as disjunctions (i.e., 'if/else' paths), 'or' preconditions (i.e., checking for an event condition if *any* of its parent events happened) and duration-based events (i.e., with a condition expected to be true for a given amount of time). However, this does not fundamentally change the method in Section 5; as for the factorization in Section 6, it only requires the graph to be directed and acyclic. On the other hand, regarding the audience, while it is true that the expected input (scene, causal graph and parameter ranges) is demanding for novice users, we envision our work as part of a larger workflow where such data could be automatically generated from a more intuitive user input; this problem is left as an exciting avenue for future research.

9 CONCLUSION

We presented a computational approach to help design chain reaction contraptions by optimizing the components' layout for a target

sequence of events specified as a causal graph. We specifically focused on robustifying the design against modeling bias and manual assembly errors. At the core of the method is the computation of a *success probability distribution (SPD)* that provides an approximate measure of robustness to uncertainties. We combined simulation-based search and machine learning techniques to build the SPD, before using it to obtain a robust design layout. We showed significant improvement over baseline methods across a wide range of dimensions, and validated our method by physically realizing a set of complex Rube Goldberg machines optimized with our technique.

ACKNOWLEDGEMENTS

We are grateful to the reviewers for their helpful feedback. We thank Dan Koschier for proofreading the paper, Tobias Ritschel and Simon Julier for their valuable comments and Mohamed Sayed for the video voiceover. This work was funded by an ERC Starting Grant (SmartGeometry StG-2013-335373), an ERC PoC Grant (SemanticCity), a Google Faculty Award, a Royal Society Advanced Newton Fellowship and gifts from Adobe.

REFERENCES

Moritz Bächer, Stelian Coros, and Bernhard Thomaszewski. 2015. LinkEdit: Interactive Linkage Editing Using Symbolic Kinematics. *ACM Trans. Graph.* 34, 4, Article 99 (July 2015), 8 pages. https://doi.org/10.1145/2766985

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 (2014), 1–96. https://doi.org/10.1145/2601097.2601157

Amit H. Bermano, Thomas Funkhouser, and Szymon Rusinkiewicz. 2017. State of the Art in Methods and Representations for Fabrication-Aware Design. *Computer Graphics Forum* 36, 2 (May 2017), 509–535. https://doi.org/10.1111/cgf.13146

K. Brewer, L. Carraway, and D. Ingram. 2010. *Forward Selection as a Candidate for Constructing Nonregular Robust Parameter Designs.* Technical Report. Arkansas State University.

Duygu Ceylan, Wilmot Li, Niloy J. Mitra, Maneesh Agrawala, and Mark Pauly. 2013. Designing and Fabricating Mechanical Automata from Mocap Sequences. *ACM SIGGRAPH Asia* 32, 6 (2013), 11.

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. https://doi.org/10.1145/3072959.3073669

Stelian Coros, Bernhard Thomaszewski, Gioacchino Noris, Shinjiro Sueda, Moira Forberg, Robert W. Sumner, Wojciech Matusik, and Bernd Bickel. 2013. Computational Design of Mechanical Characters. *ACM Trans. Graph.* 32, 4 (2013), 1–83. https://doi.org/10.1145/2461912.2461953

Erwan Coumans. 2018. Bullet Physics SDK. https://github.com/bulletphysics/bullet3. Accessed: 2018-01-01.

Thomas M. Cover and Joy A. Thomas. 2006. *Elements of Information Theory (Wiley Series in Telecommunications and Signal Processing).* Wiley-Interscience, New York, NY, USA.

Peter Fischli and David Weiss. 1987. The Way Things Go. Retrieved from https://www.youtube.com/watch?v=GXrRC3pfLnE. Accessed: 2018-01-01.

Yohsuke Furuta, Jun Mitani, Takeo Igarashi, and Yukio Fukui. 2010. Kinetic Art Design System Comprising Rigid Body Simulation. *Computer-Aided Design and Applications* 7, 4 (2010), 533–546. https://doi.org/10.3722/cadaps.2010.533-546

Akash Garg, Alec Jacobson, and Eitan Grinspun. 2016. Computational design of reconfigurables. *ACM Trans. Graph.* 35, 4 (2016), 1–14. https://doi.org/10.1145/2897824.2925900

Honda. 2003. Cog. Retrieved from https://www.youtube.com/watch?v=Z57kGB-mI54. Accessed: 2018-01-01.

Devendra Kalra and Alan H Barr. 1992. Modeling with Time and Events in Computer Animation. *Computer Graphics Forum* 11, 3 (may 1992), 45–58. https://doi.org/10.1111/1467-8659.1130045

Yilip Kim and Namje Park. 2012. Development and Application of STEAM Teaching Model Based on the Rube Goldberg's Invention. In *Computer Science and its Applications*, Sang-Soo Yeo, Yi Pan, Yang Sun Lee, and Hang Bae Chang (Eds.). Springer Netherlands, Dordrecht, 693–698.

Dieter Kraft. 1988. *A Software Package for Sequential Quadratic Programming.* Technical Report. Institut fuer Dynamik der Flugsysteme, Oberpfaffenhofen.

Mandy Lange, Dietlind Zühlke, Olaf Holz, and Thomas Villmann. 2014. Applications of lp-Norms and their Smooth Approximations for Gradient Based Learning Vector Quantization. In *ESANN*. Bruges, 271–276.

Steffen L Lauritzen. 2001. Causal inference from graphical models. *Complex stochastic systems* (2001), 63–107.

Honghua Li, Ruizhen Hu, Ibraheem Alhashim, and Hao Zhang. 2015. Foldabilizing Furniture. *ACM Trans. Graph.* 34, 4, Article 90 (July 2015), 12 pages. https://doi.org/10.1145/2766912

M. Lin, T. Shao, Y. Zheng, N. J. Mitra, and K. Zhou. 2018. Recovering Functional Mechanical Assemblies from Raw Scans. *IEEE Transactions on Visualization and Computer Graphics* 24, 3 (March 2018), 1354–1367. https://doi.org/10.1109/TVCG.2017.2662238

Li-Ke Ma, Yizhonc Zhang, Yang Liu, Kun Zhou, and Xin Tong. 2017. Computational Design and Fabrication of Soft Pneumatic Objects with Desired Deformations. *ACM Trans. Graph.* 36, 6, Article 239 (Nov. 2017), 12 pages. https://doi.org/10.1145/3130800.3130850

Niloy J. Mitra, Yong-Liang Yang, Dong-Ming Yan, Wilmot Li, and Maneesh Agrawala. 2010. Illustrating How Mechanical Assemblies Work. *ACM Trans. Graph.* 29, 4 (2010), 58:1–11. https://doi.org/10.1145/1833351.1778795

John C. Platt. 1999. Probabilistic Outputs for Support Vector Machines and Comparisons to Regularized Likelihood Methods. In *Advances in Large Margin Classifiers*. MIT Press, 61–74.

Steve Price. 2017. Top 10 Chain Reaction Tips | Rube Goldberg HowTo. Retrieved from https://www.youtube.com/watch?v=p8Wwq_B5S7I. Accessed: 2018-01-01.

Ravella Sreenivas Rao, C. Ganesh Kumar, R. Shetty Prakasham, and Phil J. Hobbs. 2008. The Taguchi methodology as a statistical tool for biotechnological applications: A critical appraisal. *Biotechnology Journal* 3, 4 (4 2008), 510–523. https://doi.org/10.1002/biot.200700201

M. O. Riedl and R. M. Young. 2006. From linear story generation to branching story graphs. *IEEE Computer Graphics and Applications* 26, 3 (May 2006), 23–31. https://doi.org/10.1109/MCG.2006.56

Adriana Schulz, Harrison Wang, Eitan Crinspun, Justin Solomon, and Wojciech Matusik. 2018. Interactive Exploration of Design Trade-offs. *ACM Trans. Graph.* 37, 4, Article 131 (July 2018), 14 pages. https://doi.org/10.1145/3197517.3201385

Adriana Schulz, Jie Xu, Bo Zhu, Changxi Zheng, Eitan Grinspun, and Wojciech Matusik. 2017. Interactive Design Space Exploration and Optimization for CAD Models. *ACM Trans. Graph.* 36, 4, Article 157 (July 2017), 14 pages. https://doi.org/10.1145/3072959.3073688

Daniel L. Schwartz and Mary Hegarty. 1996. Coordinating multiple representations for reasoning about mechanical devices. In *Proceedings of the AAAI Spring Symposium on Cognitive and Computational Models of Spatial Representation*. AAAI Press, Menlo Park, CA, 9.

Burr Settles. 2012. Active learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning* 6, 1 (2012), 1–114.

B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. de Freitas. 2016. Taking the Human Out of the Loop: A Review of Bayesian Optimization. *Proc. IEEE* 104, 1 (Jan 2016), 148–175. https://doi.org/10.1109/JPROC.2015.2494218

Maria Shugrina, Ariel Shamir, and Wojciech Matusik. 2015. Fab Forms: Customizable Objects for Fabrication with Validity and Geometry Caching. *ACM Trans. Graph.* 34, 4, Article 100 (July 2015), 12 pages. https://doi.org/10.1145/2766994

I.M Sobol'. 1967. On the distribution of points in a cube and the approximate evaluation of integrals. *U. S. S. R. Comput. Math. and Math. Phys.* 7, 4 (1967), 86–112. https://doi.org/10.1016/0041-5553(67)90144-9

Peng Song, Xiaofei Wang, Xiao Tang, Chi-Wing Fu, Hongfei Xu, Ligang Liu, and Niloy J. Mitra. 2017. Computational Design of Wind-up Toys. *ACM Trans. Graph.* 36, 6, Article 238 (Nov. 2017), 13 pages. https://doi.org/10.1145/3130800.3130808

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. https://doi.org/10.1145/2601097.2601143

Nobuyuki Umetani and Bernd Bickel. 2018. Learning Three-dimensional Flow for Interactive Aerodynamic Design. *ACM Trans. Graph.* 37, 4, Article 89 (July 2018), 10 pages. https://doi.org/10.1145/3197517.3201325

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 (2014), 1–10. https://doi.org/10.1145/2601097.2601129

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. https://doi.org/10.1145/2366145.2366146