Supplementary Document for “Designing Chain Reaction Contraptions from Causal Graphs”

Anonymous Authors

1 PRIMITIVES

This section describes in more detail the primitives implemented in our system. Please refer to the code for a complete description.

1.1 Simple primitives

Simple primitives describe single solid objects. Each such object is instantiated with design parameters describing its dimensions (e.g., width, length, height, etc.). Additionally, arbitrary physical parameters can be provided (e.g., mass, friction, restitution, etc.). Since simple primitives are constructed by combining rigid body shapes from Bullet, please refer to the documentation [1] to find available parameters. Importantly, a simple primitive without mass will be static in the simulation (and conversely, defining a mass makes it dynamic).

Most simple primitives are 3D: `Ball`, `Box`, `OpenBox`, `Cylinder`, `Capsule`, `Goblet`, `Track`. There is a single 2D primitive, `Plane`, and a ‘0D’ primitive, `Empty`, whose role is to serve as parent of other primitives in the scene graph (similar to what exists in Blender).

1.2 Constraint primitives

Constraint primitives take one or two simple primitives and add a Bullet constraint. `Pivot`, as the name suggests, adds a pivot constraint (one degree of freedom), while `Fastener` glues objects together (0 degrees of freedom).

1.3 Complex primitives

Complex primitives are created by combining any number of the above primitives. For instance, `Lever` and `Pulley` respectively combine `Box` and `Cylinder` with a `Pivot`. Meanwhile, `DominoRun` provides an easier interface to define `Box`es aligned along a path. `TensionRope` combines several Bullet constraints with input primitives to emulate the behavior of a rope in tension (although the rope itself has no physical presence in the world, i.e., it is not involved in collisions). Lastly, `RopePulley` is an even more complex version combining input primitives with constraints and a callback function approximating the effect of a rope-pulley system. As with `TensionRope`, the rope is purely visual.

1.4 Primitives used in each scenario

The following simply provides the primitives involved in each of the scenarios presented in the evaluation. Please see the configuration files for a complete description.

- causalitySwitch: `Ball`, `Box`, `DominoRun`, `Plane`, `Track`
- ballRun: `Ball`, `Box`, `Goblet`, `Lever`, `Track`
- longChain: `Ball`, `Box`, `Cylinder`, `DominoRun`, `Fastener`, `Goblet`, `Lever`, `OpenBox`, `TensionRope`, `Track`
- teapotAdventure: `Ball`, `Box`, `Cylinder`, `Fastener`, `Goblet`, `Lever`, `Pivot`, `RopePulley`, `Track`

2 EVENTS

As described in the paper, the occurrence of events in the simulation is checked with specific conditions based on rigid bodies’ spatial transforms and their derivatives. `Toppling` simply requires one of the Euler angles to be greater than a threshold, and `NotMoving` checks that the position has not changed since the beginning. Meanwhile, `Falling`, `Pivoting`, `Rising` and `Stopping` all compare a component of the body’s linear or angular velocity with a given threshold. `Contact` and its opposite, `NoContact`, use the simulator’s internal collision check. `RollingOn` combines `Pivoting` with `Contact`. Lastly, `Inclusion` uses the simulator’s internal ray casting ability to check that one body is inside another.

3 LOCAL ROBUSTNESS

In the paper, we mention an evaluation dataset $X \subset D$ created for each scenario $\mathbb{S}$ with design space D, decomposed as $X = X^+ \cup X^- \cup X^\varnothing$ (respectively successes, failures and impossible instances). The local robustness $\rho_l : D \times [0,1] \to [0,1]$ is then defined as

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

where $\mathcal{B}_\epsilon(\mathbf{x})$ is the ball of radius ϵ centered at $\mathbf{x} \in D$.

In practice, the local robustness is computed differently depending on whether it is used as an objective function $\mathbf{x} \mapsto \rho_l(\mathbf{x}, 0.1)$ for baseline methods (B2) and (B3), or as a function of the error ϵ in Figures 15 and 16. In the former case, for each function call, d^2 physically valid points are uniformly sampled around $\mathbf{x}$ and simulated on the fly (d being the number of layout parameters). Therefore, the simulation budget B defined for the baselines directly translates to a maximum number of function evaluations $\lfloor B/d^2 \rfloor$. In the latter case, as explained in the paper, the evaluation dataset X is obtained by physically checking and simulating points for each scenario; specifically, 100K points for causalitySwitch, and 1M points for ballRun, longChain and teapotAdventure. These points are drawn from the quasi-random Sobol sequence [2]. However, as there is little chance to find the solution $\mathbf{x}^*$ of a method in the evaluation dataset X, the value of $\rho_l(\mathbf{x}^*, 0)$ is very likely to be 0, since the ball $\mathcal{B}_0(\mathbf{x}^*)$ is very likely to be empty. To counter this, we give a ‘thickness’ to the ball: i.e., we use a slightly modified local robustness $\epsilon \mapsto \rho_l(\mathbf{x}^*, \epsilon + \eta)$. Then, for each solution $\mathbf{x}^*$, we sample and simulate 100 points in the local neighborhood $\mathcal{B}_\eta(\mathbf{x}^*)$ and temporarily add them to X. In our experiments, we used a thickness $\eta = 0.1 l_\epsilon$, with l_ϵ the length of a plot step ($l_\epsilon = 1/30$).

REFERENCES

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

[2] 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