EUROGRAPHICS 2013 / I. Navazo, P. Poulin
(Guest Editors)

Volume 32 (2013), Number 2

Physics Storyboards

Sehoon Ha[12] Jim McCann[1] C. Karen Liu[2] Jovan Popović[1]

[1] Adobe Systems Incorporated
[2] Georgia Institute of Technology

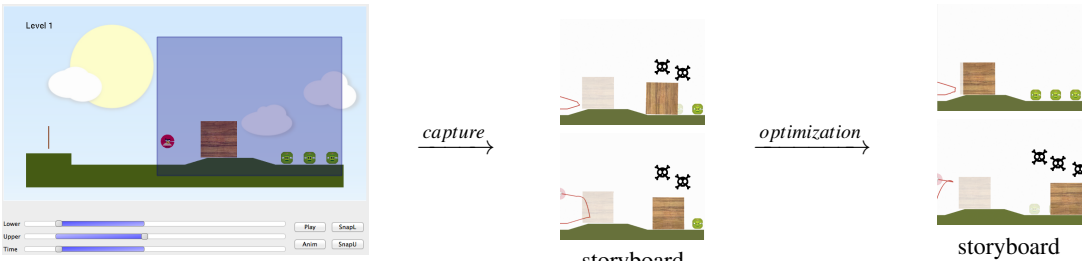

Figure 1: *Physics storyboards consist of snapshots of an interactive animation as captured by a designer. These snapshots provide both an at-a-glance overview of the consequences of parameter changes and an way to specify objectives for automatic parameter tuning. In this case, the parameters have been adjusted so that hitting the top of the box kills the pigs, but hitting the bottom does not.*

Abstract

Physical simulation and other procedural methods are increasingly popular tools in interactive applications because they generate complex and reactive behaviors given only a few parameter settings. This automation accelerates initial implementation, but also introduces a need to tune the available parameters until the desired behaviors emerge. These adjustments are typically performed iteratively, with the designer repeatedly running—and interacting with—the procedural animation with different parameter settings. Such a process is inaccurate, time consuming, and requires deep understanding and intuition, as parameters often have complex, nonlinear effects.

Instead, we propose that designers construct physics storyboards *to accelerate the process of tuning interactive, procedural animations. Physics storyboards are collections of space-time snapshots that highlight critical events and outcomes. They can be used to summarize the effects of parameter changes (without requiring the designer to perform extensive play-testing); and—when augmented with designer-provided evaluation functions—allow automatic parameter selection. We describe our implementation of this method, including how we use sampling to ensure that our automatically-selected parameters generalize, and how we time-warp user input to adapt it to changing parameters. We validate our implementation by using it to perform various design tasks in three example games.*

Categories and Subject Descriptors (according to ACM CCS): I.3.6 [Computer Graphics]: Methodology and Techniques—Interaction techniques

1. Introduction

Procedural methods are increasingly popular tools in interactive applications because they generate complex and reactive behaviors from just a few parameters. Applications range from the simple inertia, friction, and spring force models used for flick- and bounce- scrolling in modern touch UIs to

the particle systems, rigid-body dynamics, and discrete time simulators used to bring complex environments and reactive characters to life in computer games. Procedural methods are especially attractive for applications with user interaction because they offer a modular, systematic, and general approach for responding to varied user input.

Selecting the proper parameters for these simulations is critical. Mobile UIs should feel snappy and controllable, not sluggish or twitchy. Games need to be approachable and challenging, not counterintuitive, impossible, or trivial. However, the task of translating these ideas into good parameter settings for the underlying procedural animation engine is difficult.

Consider a designer building a physics-based game such as "Angry Birds" [Rov] where the goal is to destroy designated targets by launching projectiles onto a wide array of assembled structures. As all motions are simulated, the designer must assign specific physical values – density, friction, and restitution – for every material. A designer seeking to fix an undesirable interaction – say, a tower which topples too easily – has a wide range of possible parameters to choose from. Adjusting any given parameter can have a complex effect not only on the target event, but also on many other interactions. For instance, reducing the density of the projectile could cause another level to become impossible to complete. This is further complicated by the unpredictability of user interaction: verifying the effect of each parameter change requires (1) replaying (potentially, a large portion of) the game and (2) repeating that process a number of times to ensure that desired outcomes remain stable with minor variations of player input. Thus, setting the parameters to generate a specific outcome often requires luck, understanding, and extensive testing.

In this paper, we describe a methodology for parameter tuning of (pseudo-)physics-driven interactions which seeks to avoid some of these pitfalls. We build our method around the observation that physics *is not* path-dependant. Thus, any behavior a designer encounters while testing can be stored as a snapshot of initial conditions and player inputs, to be replayed later (under potentially different parameter settings). By accumulating such snapshots, a designer creates a "physics storyboard" (Figure 1), which allows at-a-glance review of the current parameter settings. Thus, designers need not replay the game, as the assembled storyboard visualizes any parameter change automatically.

Physics storyboards become even more useful when designers specify the desired outcome for each snapshot. This evolves the storyboard into a specification that our system can use to select parameters automatically. A snapshot for the "Angry Birds" game, for example, may indicate that a given target should be harder to destroy, or that a certain collapsing structure looks too dull, or both – the target should not be destroyed and the collapse should be more energetic.

In this paper, we apply physics storyboards to parameter tuning in three games. The first, an "Angry Birds" clone, demonstrates the basic operation of the system. The second, a racing game, demonstrates the need to consider how players will adapt their input in response to changing parameters. And the third, a real-time strategy game, shows that physics storyboards also work on "pseudo-physical" systems.

The optimizations in all three games are difficult due to a high-dimensional search space and non-smooth landscape. Nevertheless, we show how to formulate the problem for several difficult tasks – tasks that we could not accomplish with manual tuning alone. Furthermore, we show how to discourage overfitting so that optimization converges on parameters that yield the behavior not only on chosen snapshots but also on nearby variations.

2. Related Work

In computer graphics, early procedural animation techniques employed simulation to improve keyframe animation. The goal of these systems is to output motion that simultaneously achieves both the specified outcomes and the laws of physics [IC87, BB88, KWT88, BN88]. Trajectory optimizations, in particular, showed that realistic effects, such as anticipation and squash-and-stretch, emerged automatically from optimizations that minimize the overall control effort. Spurred on by these findings, subsequent years witnessed an explosion of related optimizations that improve output quality and extend practical application [Coh92, LGC94, FP03, LHP05]. In this project, we focus on *behaviors* instead of *trajectories*: while the storyboard does provides constraints, our system seeks a general (interactive) solution instead of a single (static) path that matches these constraints.

In that regard, our approach is more analogous to control of interactive behaviors such as bounding [RH91], crawling [Sim94], walking [LvdPF96, YLvdP07], and other forms of animal locomotion [HWBO95, GT95]. Instead of searching for individual motion trajectories, these approaches seek a feedback policy that will produce desired behavior in spite of external disturbances. Tellingly, these methods – like parameter tuning for interactive applications – draw on deep insights and understanding to complete this search successfully. For example, a principal contribution of the SImple BIped CONcontroller [YLvdP07] was the continuous adaption of control according to both position and velocity of the center of mass. This led to dramatic improvements over previous works which relied on feedback policies according to velocity alone. Rather than tuning controllers inside a specific simulation, our work focuses on tuning the simulations themselves.

This new domain was first proposed by Barzel and colleagues [BHW96] in a paper that advocated the notion of physically *plausible* simulation instead of the established physically *valid* simulation. This suggested that except in scientific applications, computer simulations could choose

the particular stochastic parameters that favor a specific outcome. Subsequent approaches proposed different methods for finding such parameters including gradient-based search [PSE*00, MTPS04] and stochastic sampling [CF00, TJ07]. Our paper explores a different approach with Covariance Matrix Adaptation (CMA) [Han06], which has been successful for trajectory optimization [WP09] and locomotion control [WFH10]. [TWSM*11] explores parameter estimation for image segmentation. Our domain introduces at least two new challenges including the need to express diverse set of design goals with the objective function and the need to adapt user input (i.e player behavior) so that resulting simulation achieve desirable outcomes in a stable/general manner.

A common approach in parameter selection interfaces is to present the parameter space (or a collection of samples thereof) in an explorable way, through 2D layout of results [MAB*97]; careful selection of sliders [OLGM11, LBH12]; or (in a physics context) direct manipulation [PSE*00] or in-situ visualization [TJ07]. These methods help a designer understand the effects of parameter variation on a single set of initial conditions. Both [WFR*10, WRF*11] represent simulations as tracks (or word lines) where each parameter change corresponds to branches that spawn new tracks, and [WFR*10] describes how to incorporate uncertain parameter values into this explorable visualization. Meanwhile [BM10] clusters outcomes from different timelines to suppress minor variations and to highlight entire outcome categories. As with other papers in this paragraph, our work is complementary as we can imagine deploying any of these techniques to specify and explore acceptable outcomes, while leaving our system to find consistent parameters over all the snapshots.

3. Physics Storyboard

Our physics storyboard system allows a designer to interact with a simulation, to record different snapshots, to specify desired outcomes for each snapshot, and to run an optimization to select parameters that satisfy these outcomes. Together this collection of events and outcomes enables automatic search for the parameters that meet the design goals.

3.1. What is a Simulation?

The storyboard approach can be applied to physically based simulation or other pseudo-physical discrete-event interactive procedural animation. Let $\mathcal{X}$ be a set of world states, $\mathcal{P}$ be a set of parameters, and $\mathcal{C}$ be a set of user controls. Each simulation defines a family of behaviors[†] parameterized by $p \in \mathcal{P}$ and induced by function $f_p : \mathcal{X} \times \mathcal{C} \to \mathcal{X}$, which advances the state of the system from x_t to x_{t+1} according to user input c_t:

$$x_{t+1} \leftarrow f_p(x_t, c_t) \tag{1}$$

We assume that parameters p are constant throughout simulation.

3.2. Snapshots

Each snapshot specifies desired behavior by recording a relevant example encountered by the designer during interaction with the simulation. Snapshots may highlight both behaviors that need to change and behaviors that should be preserved. In our prototype, a designer collect a snapshot by playing the game, pausing, and selecting a spatial and temporal region of interest (Figure 1, left). Note that the spatial extent of this region has no bearing on any underlying computations, but is helpful for visualization.

Under the hood, we augment the simulation to record the control c_t for all time steps and to occasionally checkpoint the state x_t. When the designer scrubs to time i we reconstruct x_i from the nearest prior x_t and subsequent user input $c_t, \ldots, c_{i-1}$. The system records the snapshot by storing the initial state x_i along with the user input $\mathbf{c} = (c_i, \ldots, c_j)$ for the duration of chosen time interval $[i, j]$. This representation allows the system to reconstruct states $x'_i, \ldots, x'_j$ for any set of parameters p' by re-simulating with $f_{p'}$.

3.3. Evaluation Function

Evaluation functions score each snapshot so that the system can optimize simulation parameters for designers. In our system, designers express these functions in the scripting language Lua [Lua], using standard mathematical operators and the following primitives:

General Terms	
`lt`(a,b)	zero if $a < b$, a large penalty otherwise
`AV`()	total angular velocity
`LV`()	total linear velocity
`Fr`(*Material*), `Rs`(*Material*), `Ds`(*Material*), `Size`(*Obstacle*), ...	various parameter values
Angry Birds Clone	
`DeadPig`()	number dead pigs at end of snapshot
Racing Game	
`Skid`()	number of skidded frames
`Crash`()	nonzero if car crashed
`Travel`()	arc-length of car trajectory
`PosDir`(*Pt*,*Dir*)	amount of travel in a given direction
Strategy Game	
`Totalhp`(*Team*)	total HP of team at end of snapshot

Of these operators, the `lt`() operator is especially useful, as it allows the specification of hard constraints. It is these

[†] We use the term "behavior" because it affects the manner in which the simulation responds to user input.

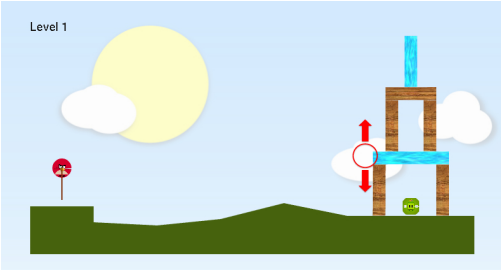

Figure 2: *A tower of wooden and glass blocks. Our system can be used to control the weak point of the tower.*

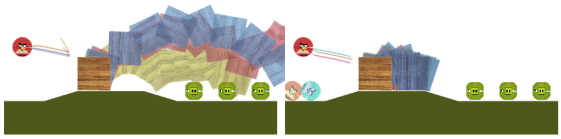

Figure 3: *Comparing a "lucky" trajectory and a stable behavior. The objective function is set to "save all pigs". Left, an unstable solution (in blue) involves a box with low mass that just barely manages to miss the pigs given exactly the initial conditions in the snapshot; randomly selected neighboring initial conditions (yellow,red) kill at least one pig. Right, a stable behavior with a heavy box, blocks the bird both for the initial conditions from the snapshot and randomly selected neighbors.*

constraints that we refer to when we talk about a snapshot's objective function being "satisfied".

Mathematically, this process augments each snapshot with evaluation function $e : \mathcal{X} \times \ldots \times \mathcal{X} \to \mathbb{R}$, which assigns a scalar value to the sequence of states $x_i, \ldots, x_j$ in the snapshot.

3.4. Optimization

The optimization tunes the parameters by maximizing the sum of evaluation functions for each snapshot $(x, \mathbf{c}, e) \in \mathcal{S}$:

$$p = \underset{p}{\operatorname{argmax}} \sum_{(x,\mathbf{c},e)\in\mathcal{S}} e(x_i, \ldots, x_j) \quad \text{where } x_{t+1} \equiv f_p(x_t, c_t) \tag{2}$$

We optimize with the Covariance Matrix Adaptation Evolution Strategy (CMA-ES), which populates successive generations of samples according to an evolving multi-variate distribution [Han06, IGHM08], where the mean and covariance of this distribution update according to evaluation of each generation of samples.

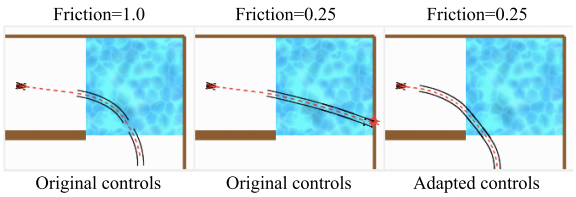

Figure 4: *Repeating the same user control on different surfaces leads to unrealistic model of player behavior. Our system modifies user input automatically to ensure that similar behavior is still feasible on new surface.*

3.4.1. Stability

On difficult non-smooth landscapes like this one, all optimizations are susceptible to getting stuck in local maxima. In our case, this manifests itself with unstable parameter settings that satisfy constraints in all snapshots but hinge on some irrelevant particulars within each snapshot. For example, a projectile which misses a target, as requested, because it bounces just barely high enough to pass over it (Figure 3). While trajectory optimizers might be satisfied with (or, indeed, seek out) such "lucky solutions" [BHW96, PSE*00, CF00, TJ07], they are unwelcome in our scenario, as they are unlikely to match the designer's intent. For the results of the optimization to be meaningful, our solutions must also generalize to similar situations.

We guide our optimization towards stable solutions by evaluating each sample multiple times. Whenever a sample satisfies all snapshot constraints, we also evaluate it on 10 random perturbations of the initial conditions. The perturbation is game-specific; we perturb the position and velocity of the bird in our Angry Birds clone, and the formation in our strategy game. We do not use perturbations in the racing game. The final value of the sample is the average of all jittered evaluations.

3.4.2. User Input

The system evaluates each parameter change by resimulating each snapshot, but this requires new user input. The simple solution of reusing the input $c_i, \ldots, c_j$ stored with the snapshot is inadequate because players may react differently for each parameter change. For instance, in a racing game, the same set of keystrokes may lead to a crash when the surface becomes more slippery (Figure 4). This does not mean that players cannot drive on the slippery surface: they may simply need to hold down the turning button longer, or accelerate less. Our system models these adaptations by time-warping user input.

To keep the optimization tractable, we warp with a monotonically increasing piecewise-linear function with 3 degrees of freedom; this also has the advantage of not deviating too

much from the original strategy recorded in the snapshot.

$$w(t) = \begin{cases} 4(p_1 - p_0)(t - \frac{0}{4}) + p_0 & \text{if } t \le \frac{1}{4} \\ 4(p_2 - p_1)(t - \frac{1}{4}) + p_1 & \text{if } \frac{1}{4} < t \le \frac{2}{4} \\ 4(p_3 - p_2)(t - \frac{2}{4}) + p_2 & \text{if } \frac{2}{4} < t \le \frac{3}{4} \\ 4(p_4 - p_3)(t - \frac{3}{4}) + p_3 & \text{if } \frac{3}{4} < t \end{cases} \quad (3)$$
$$\text{where } 0 = p_0 < p_1 < p_2 < p_3 < p_4 = 1$$

For each sample evaluation, the time-warp optimization maximizes the evaluation function for the snapshot to identify the time warp w_p for the current parameters p:

$$\begin{aligned} w_p &\equiv \underset{w}{\text{argmax}}\, e(x_i, \ldots, x_j) \\ &\text{where } \hat{\mathbf{c}} \equiv w(\mathbf{c}) \\ &\text{and } x_{t+1} \equiv f_p(x_t, \hat{c}_t) \end{aligned} \quad (4)$$

And the optimization over all snapshots depends on the optimally warped control for each snapshot:

$$\begin{aligned} p &= \underset{p}{\text{argmax}} \sum_{(x,\mathbf{c},e,w_p)\in\mathcal{S}} e(x_i, \ldots, x_j) \\ &\text{where } \hat{\mathbf{c}} \equiv w_p(\mathbf{c}) \\ &\text{and } x_{t+1} \equiv f_p(x_t, \hat{c}_t) \end{aligned} \quad (5)$$

Our experiments show that time-warping is a tractable and adequate model for user behavior in some situations. Just as jittering guides the optimization to behaviors that generalize beyond captured initial conditions, input warping helps the optimization explore behaviors that venture outside the captured control.

4. Results

	#Snaps	#Frames	#Params	μ	λ	#Iter	Time
Task1	2	702	6	50	100	6	~5 min
Task2	8	2940	12	50	100	6	~25 min
Task3	1	486	4	20	40	5	~15min
Task4	4	857	9	20	40	5	~50min
Task5	4	1954	12	50	150	6	~30min
Task6	4	1597	12	50	150	6	~30min

Table 1: *Storyboard and optimization parameters*

We built and tuned three games using our system. All parameters of storyboards and optimization are listed on the Table 1.

4.1. Angry Birds Clone

Our first example game is a clone of Angry Birds [Rov]. We choose this as an example domain because it contains complicated physical interaction, but limited user input. The objective of the game is to launch birds (effectively cannonballs) in order to knock down structures and crush pigs.

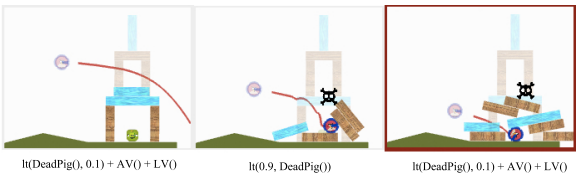

Figure 5: *Snapshots and their objective functions for the weak-point-setting task. Only the bird hitting the tower in the middle should topple it, killing the pig. The designer also chose to increase linear and angular velocity to make misses visually exciting.*

The structures are made of blocks of three different materials: wood, ice, and stone, each of which has different density, friction, and restitution coefficients. The interactions between birds, blocks, and pigs are computed with rigid-body simulation using Box2D [Cat].

In this setup, the parameter space, $\mathcal{P}$ is 12-dimensional, and contains the density, friction, and restitution coefficients of the wood, ice, and stone materials, as well as the bird. The space of control inputs, $\mathcal{C}$, contains only the launch velocity of the bird, and is ignored on all but the initial frame. The world state, $\mathcal{X}$, is the current simulation state, including positions and velocities as well as other internal state (e.g. persistent contact manifolds) of the simulator. Finally, the simulation function, f_p, is rigid body dynamics as implemented by Box2D (along with a bit of game-specific logic to handle bird launching).

This game is an illustrative target for our system because making the game compelling requires a careful selection of parameters. If the interactions in the game are too lively, structures will be unstable and the game will be too easy. However, if the interactions are too damped, the game will be visually boring and (in the extreme) it will be impossible to complete levels.

For our first tuning task we consider a single level containing a tower:

> **Task 1: Tuning a Tower's Weak Point.** Consider the tower shown in Figure 2; for a given parameter setting, the tower may topple onto the pig – or not – depending on the point of impact of the bird. Determine a parameter setting such that the pig is safe except when the tower is hit at the circled "weak point".

This task is difficult to accomplish by hand both because there is no straightforward relationship between the parameter settings and the tower's weak point(s) and because evaluating a given parameter setting requires the designer to play the game repeatedly to test various points of impact.

To perform this task using our system, the designer first captures snapshots of bird impacts above, below, and at the desired weak point. The designer then augments these snap-

shots with objective functions specifying that the pig should only be killed when the tower is struck at the weak point (Figure 5). The designer also adds a term to maximize energy even when the tower isn't toppled, in order to make misses more exciting.

This storyboard specifies all that's needed to find the required the parameters automatically. Figure 6 visualizes the changes made to each parameter.

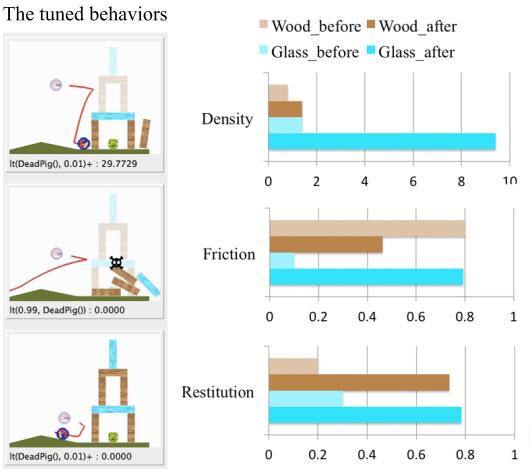

Figure 6: *After our system's automatic parameter tuning, the tower topples only in the middle snapshot, satisfying the designer's objective function. The chart shows the parameters before and after tuning.*

A word about stability. Each snapshot records one bird trajectory. However, if our system optimizes over only these trajectories, it can find unstable or "lucky" solutions that meet the constraints but don't generalize well to nearby trajectories. Therefore, as we discussed in Section 3.4.1, our system actually evaluates the objective function over families of nearby trajectories (Figure 3).

Our second design task is more holistic, encompassing three levels with shared physical properties.

> **Task 2: Changing the difficulty of the game** Produce both easy and challenging versions of the levels shown in Figure 7 by adjusting the underlying material properties.

To complete this task in our system, the game designer collects snapshots of trajectories illustrating different strategies. By adjusting the objective functions, the designer can select which strategies will work for the easy parameters and which (smaller) set will work for the challenging parameters. The snapshots after automatic tuning are shown in Figure 8.

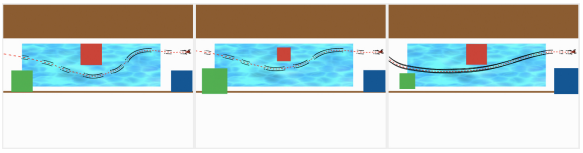

Figure 9: *Our system is used to automatically tune a chicane. Before tuning (left), the ice has relatively high friction and the obstacles are of uniform size. Tuning without input adaptation (center), forces the red obstacle to shrink so that the car does not crash. Tuning with input adaptation (right) can achieve lower friction while increasing the overall obstacle size.*

4.2. Racing Game

Our next example design tasks concern a 2D top-down-view racing game. The objective of this game is to drive a car around a track as quickly as possible. In our game, the player accelerates, breaks, and turns the car by pressing arrow keys.

We use Box2D as our simulator and emulate the physics of wheels by removing lateral velocities with impulses. By setting a maximum for these lateral impulses, and applying some angular damping, our vehicle can be made to skid on the surfaces with low friction. This method is commonly used in 2D racing games [top].

For this game the parameter space, $\mathcal{P}$, includes friction of the track, ice, grass, and dirt; track-specific parameters related to obstacle placement; and parameters for the driving model – skid-ness (a parameter that modulates both maximum lateral impulse and angular damping) and engine power. The space of control inputs, $\mathcal{C}$, consists of a steering signal in $\{left, center, right\}$ and an acceleration signal in $\{accelrate, coast, brake\}$. As in the previous example the world state, $\mathcal{X}$, is the simulator state, and the simulation function, f_p, is rigid body dynamics as implemented by Box2D (in this case, adapted for car physics).

> **Task 3: Tuning the difficulty of a chicane.** Consider the chicane shown in Figure 9; tune the friction of the icy ground and the size of the barriers to make this chicane as difficult as possible.

For this task, the designer uses a single snapshot, consisting of a successful run through the chicane. They augment the snapshot with an objective function which reflects the task: lower the friction, and increase barrier size, but don't allow the car to crash. The result after automatic tuning is applied to a four-parameter space consisting of the friction of the ice and the three barrier sizes is shown in Figure 9 (right). This figure also shows the result when user input adaptation (as described in Section 3.4.2) is disabled.

> **Task 4: Balancing the difficulty of the race track.** Produce a more balanced race track by adjusting the terrain and vehicle properties as shown in Figure 10.

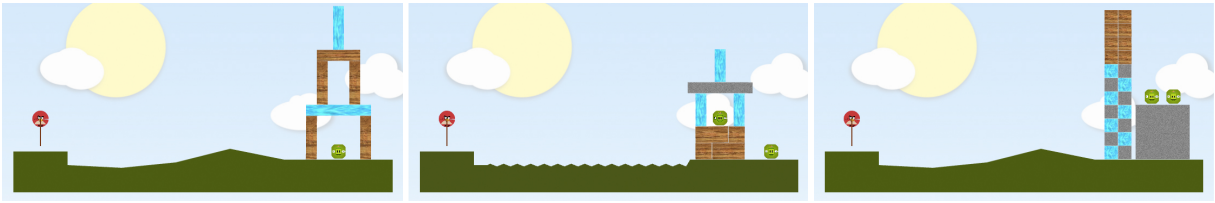

Figure 7: *In our second example task, our system is used to adjust physical properties shared by these three levels in order to produce both easy and challenging versions.*

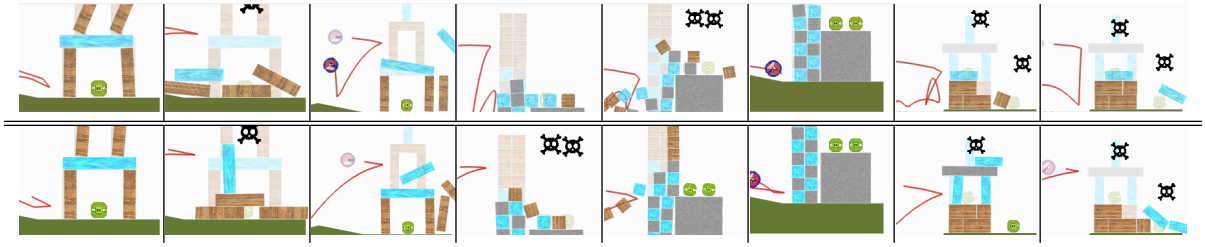

Figure 8: *The designer tunes the difficulty of three levels by using our system to automatically adjust the material parameters (shared between levels). The snapshots after tuning are shown. For the easy version of the parameters (top), the designer constrains the number of pigs to be killed to 0,1,0, 2,2,0, 2, and 2. For the challenging version of the parameters (bottom), the designer uses 0,1,0, 2,0,0, 1, and 2.*

In the next task, the track contains four main features: a wall that blocks passage unless the car is skidding, an s-curve, a corner turn, and a chicane. The default material parameters allow the vehicle to grip through all three turns easily, but they also make it impossible to skid through the blocking wall. The designer creates a storyboard with four snapshots to adjust the behavior of all four features: the three turns should have lower friction to make the turning more difficult but possible; the chicane should have larger obstacles; and the wall should have higher friction to make skidding more difficult but possible.

The optimizer runs on a 9-parameter space: the four ground frictions, three obstacle sizes, and the car's torque and skid-ness settings. Satisfying all the snapshots requires carefully balancing the car settings and the ground fictions. Striking this balance is delicate and showed up in the longer optimization time of 50 minutes. But, whereas we couldn't find appropriate parameters manually, the optimization achieved all four tasks. Playtesting the final behavior confirmed that all four features became passable while each turn required a more skillful control to avoid collisions with the wall.

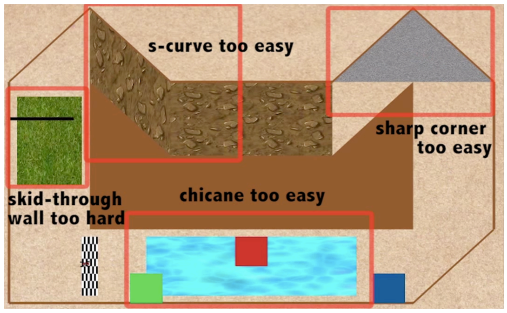

Figure 10: *The default friction parameters make all three turns too easy, but make passage through the skid-through wall impossible. The designer can use one snapshot to ensure that wall passage becomes possible but not too easy (by favoring higher friction), and three other snapshots to make each turn more challenging (by reducing the friction).*

4.3. Strategy Game

Our final design tasks concern a real-time strategy game; we chose this domain because it illustrates using our system in a non-physics setting. In a real-time strategy game, players attempt to destroy an opponent's assets and/or control territory by constructing and maneuvering various units. One can view a real-time strategy game as having two aspects: first, an economic aspect, in which players build infrastructure to effectively transform resources into units; and, second, a combat aspect, in which individual groups of units skirmish over territory.

For our real-time strategy game tuning tasks, we focus on the combat aspect. We implemented a basic combat pseudo-physics with three unit types, each with five parameters.

Rather than building a whole strategy game, we allow the designer to explicitly construct battles by adding units to both sides. These battles serve as the snapshots.

The parameter space for this game consists of the attack power, attack range, attack speed, movement speed, and hit points of each unit type. The control input is empty (as if the player had simply set the units to attack). The step function is basic combat pseudo-physics: units move toward the nearest enemy unit if not in range, and attack it if able.

We start out by to duplicating two common dynamics found in real-time strategy games.

> **Task 5: Basic game dynamics.** Tune the three unit types to exhibit "rock-paper-scissors" and "diminishing-returns" behavior.

These are both classic behaviors for RTS games. The "rock-paper-scissors" behavior means for groups A, B, and C – in this case: five pawns, three knights, and four rooks – group A beats C, C beats B, and B beats A. This is desirable, because it forces players to adapt their strategy based on that of their opponent (rather than building one "best unit").

The "diminishing-returns" behavior refers to how unit effectiveness scales with army size. In this case, four rooks lose to five pawns, but eight rooks beat ten pawns; the pawns do not work as well in a large group as the pawns. Such nonlinear scaling behavior provides an incentive for players to adapt their army composition as its size grows, or to seek out battles of an appropriate size for their troop type.

In this case, the designer can express these behaviors in just four snapshots, as shown in Figure 11. The adjusted unit parameters (Figure 12) exhibit rock-paper-scissors behavior as follows: the pawns are short-range but speedy units, allowing them to quickly overrun rooks, despite the rook's superior range; the knights are short-range, slow, heavy-hitting units with plenty of health, allowing them to beat the pawns with brute strength; and the rook's superior range means that they are able to kill the knights before the knights get close enough to deal too much damage. The range of the rooks also enables larger groups of rooks to be more effective than pawns, as pawns that are stuck behind their peers are unable to attack, despite coming under fire from multiple ranks of rooks.

Another desirable feature of RTS games are control- and position- dependent battle outcomes; that is, a player should be able to influence the outcome of a close battle by attacking from a good angle, or by carefully controlling their units during the battle. This forces players to divide their attentions between strategy and tactics, increasing the excitement in the game.

> **Task 6: Position- and control-dependence.** Make the outcome of rook-vs-pawn battles position-dependent and the outcome of rook-vs-knight battles control dependent. Specifically, three rooks should lose to six pawns when surrounded, but win when battling in tight groups; and three rooks should normally lose to three knights, but should win when their AI is modified to step backward after firing[‡].

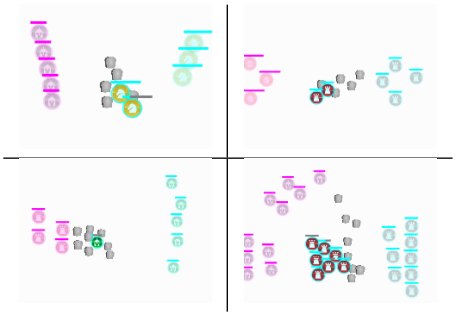

Figure 11: *Four snapshots which the designer uses to specify "rock-paper-scissors" and "diminishing-returns" behavior. Initial and final positions of the units are shown; these are the results after tuning, showing that the designer's constraints (that the team on the left lose) have been satisfied.*

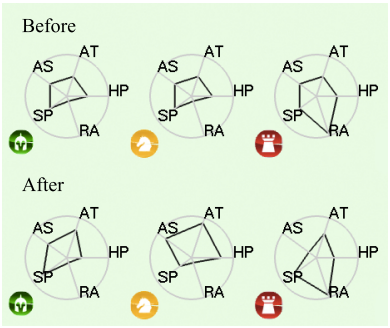

Figure 12: *Before tuning, the units have equal stats, though the pawns and knights are set as close range units while the rooks are long range. After tuning, the pawns are fast melee units (making them more effective against the rooks), while the knights have more health and attack power (giving them the edge against the pawns), and the rooks have a higher attack speed (allowing them to defeat the knights before they get too close).*

For this task, the designer uses four snapshots (Figure 13). Figure 14 shows the parameters before and after tuning. With the starting parameters, the pawns are too strong when attacking head-on, and the knights are too weak to defeat even uncontrolled rooks. After tuning, the pawns become slower, making them less efficient in a crowd, where they get in each-others way; and the knights become stronger, helping them to deal with the (uncontrolled) rooks. The rooks be-

‡ This simulates a player performing a control technique called "stutter-stepping," which is popular in Starcraft [Bli]

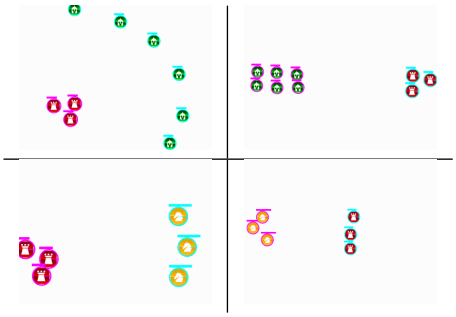

Figure 13: *Snapshots to make battle outcomes position-dependent (top row), and control-dependent (bottom row). In each case the designer wants the right team to win. The rooks in the bottom-right snapshot are controlled to step away from the knights when not firing.*

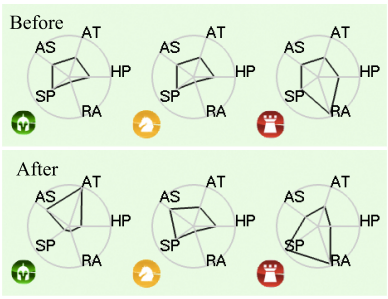

Figure 14: *The optimized unit parameters for position- and control- dependence. The pawns become slower, thus less efficient in crowds, but also more powerful (more deadly once they close in). The knights become relatively stronger (faster attack speed and more hit points), but the rooks get faster, making their stutter-stepping AI more effective.*

come faster, which allows them to gain more distance from the knights when controlled.

5. Conclusion

In this paper, we presented *physics storyboards*, a new methodology for tuning interactive procedural animations. This method provides designers with at-a-glance overview of the results of their parameter tuning, and can relieve them of burdensome playtesting. Further, we showed how physics storyboards augmented with objective functions can be used for automatic parameter tuning, and how care must be taken to ensure generality and to adapt recorded user input. Our experiments reveal that these storyboards can be helpful for several tuning tasks, and highlight some open problems for future investigation.

Our handling of user input raises two concerns. First, the racing prototype warps the stored user input with only a 3-dof curve. Although this may not suffice in other applications, more general warps could be used in our framework. We believe that warping will remain a suitable method for generalizing user input without abandoning its intent, similar to its success in other graphical applications [WP95, HPP05, MPS06], but more work will be needed to avoid the associated increase in optimization complexity. Second, the search for the optimal time-warp currently conflates two separate goals: user intent and designer intent. In a game, for example, a player wants to win by racing around the track as fast as possible, while a designer wants to create a track that forces the player to slow down. This tension implies that goals for time-warping, which determine player input, should be permitted to differ from evaluation functions used for parameter tuning. A more general treatment would separate these two concerns and ask designers to specify two sets of functions.

Optimization may converge to solutions that surprise designers. Consider the task of making a corner less slippery while ensuring that the car does not crash. The optimization may converge to a solution where the user input has been warped such that the car performs a u-turn, avoiding the slippery surface of the corner entirely. In our prototypes, the designer must resolve this ambiguity with another objective term such as maximizing the distance travelled along the track, or with weights to encourage/discourage certain behavior. Or, they may need to create new snapshots. Because longer optimization times can make such adjustments frustrating, more studies are needed to discover how to help designers formulate their goals for different types of applications.

When the optimization fails to find a satisfying solution, it often provides useful statistics that can deepen the understanding of the designer. For example, a designer could surmise which snapshots are difficult to achieve based on the number of successful samples. Also, when there are two snapshots suggesting that one knight should defeat two pawns and one pawn should defeat two knights, a designer can infer that they are in conflict because the optimization found plenty solutions for each snapshot, but no solution for both. These and other statistics can inform and improve design so improved browsing and visualization of possible outcomes is another open area for future work.

References

[BB88] BARZEL R., BARR A. H.: A modeling system based on dynamic constraints. In *Computer Graphics (Proceedings of SIGGRAPH 88)* (Aug. 1988), Annual Conference Series, ACM SIGGRAPH, pp. 179–188. 2

[BHW96] BARZEL R., HUGHES J. F., WOOD D. N.: Plausible motion simulation for computer graphics animation. In *Computer Animation and Simulation '96* (Poitiers, France, Sept. 1996), Proceedings of the Eurographics Workshop, pp. 184–197. 2, 4

[Bli] BLIZZARD: *Starcraft, http://us.blizzard.com/en-us/games/sc/*. 8

[BM10] BRUCKNER S., MOLLER T.: Result-driven exploration

of simulation parameter spaces for visual effects design. *Visualization and Computer Graphics, IEEE Transactions on 16*, 6 (nov.-dec. 2010), 1468 –1476. 3

[BN88] BROTMAN L. S., NETRAVALI A. N.: Motion interpolation by optimal control. In *Computer Graphics (Proceedings of SIGGRAPH 88)* (Aug. 1988), pp. 309–315. 2

[Cat] CATTO E.: *Box2D, http://box2d.org/*. 5

[CF00] CHENNEY S., FORSYTH D. A.: Sampling plausible solutions to multi-body constraint problems. In *Proceedings of ACM SIGGRAPH 2000* (July 2000), Annual Conference Series, pp. 219–228. 3, 4

[Coh92] COHEN M. F.: Interactive spacetime control for animation. In *Computer Graphics (Proceedings of SIGGRAPH 92)* (July 1992), vol. 26, pp. 293–302. 2

[FP03] FANG A. C., POLLARD N. S.: Efficient synthesis of physically valid human motion. *ACM Transactions on Graphics 22*, 3 (July 2003), 417–426. 2

[GT95] GRZESZCZUK R., TERZOPOULOS D.: Automated learning of muscle-actuated locomotion through control abstraction. In *Proceedings of SIGGRAPH 95* (Aug. 1995), Annual Conference Series, pp. 63–70. 2

[Han06] HANSEN N.: *Towards a New Evolutionary Computation. Advances on Estimation of Distribution Algorithms*. Springer, 2006, ch. The CMA evolution strategy: A comparing review, pp. 75–102. 3, 4

[HPP05] HSU E., PULLI K., POPOVIĆ J.: Style translation for human motion. *ACM Transactions on Graphics 24*, 3 (Aug. 2005), 1082–1089. 9

[HWBO95] HODGINS J. K., WOOTEN W. L., BROGAN D. C., O'BRIEN J. F.: Animating human athletics. In *Proceedings of ACM SIGGRAPH 95* (Aug. 1995), Annual Conference Series, pp. 71–78. 2

[IC87] ISAACS P. M., COHEN M. F.: Controlling dynamic simulation with kinematic constraints, behavior functions and inverse dynamics. In *Computer Graphics (Proceedings of SIGGRAPH 87)* (July 1987), Annual Conference Series, ACM SIGGRAPH, pp. 215–224. 2

[IGHM08] IGEL C., GLASMACHERS T., HEIDRICH-MEISNER V.: Shark. *Journal of Machine Learning Research 9* (2008), 993–996. 4

[KWT88] KASS M., WITKIN A., TERZOPOULOS D.: Snakes: Active contour models. *International Journal of Computer Vision 1* (1988), 321–331. 2

[LBH12] LINDOW N., BAUM D., HEGE H.-C.: Perceptually Linear Parameter Variations. *EUROGRAPHICS 2012 31*, 2 (2012). 3

[LGC94] LIU Z., GORTLER S. J., COHEN M. F.: Hierarchical spacetime control. In *Proceedings of SIGGRAPH 94* (July 1994), Annual Conference Series, pp. 35–42. 2

[LHP05] LIU C. K., HERTZMANN A., POPOVIĆ Z.: Learning physics-based motion style with nonlinear inverse optimization. *ACM Transactions on Graphics 24*, 3 (Aug. 2005), 1071–1081. 2

[Lua] : *Lua, http://http://www.lua.org/*. 3

[LvdPF96] LASZLO J. F., VAN DE PANNE M., FIUME E. L.: Limit cycle control and its application to the animation of balancing and walking. In *Proceedings of SIGGRAPH 96* (Aug. 1996), Annual Conference Series, pp. 155–162. 2

[MAB*97] MARKS J., ANDALMAN B., BEARDSLEY P. A., FREEMAN W., GIBSON S., HODGINS J. K., KANG T., MIRTICH B., PFISTER H., RUML W., RYALL K., SEIMS J., SHIEBER S.: Design galleries: A general approach to setting parameters for computer graphics and animation. In *Proceedings of SIGGRAPH 97* (Aug. 1997), Computer Graphics Proceedings, Annual Conference Series, pp. 389–400. 3

[MPS06] MCCANN J., POLLARD N. S., SRINIVASA S.: Physics-based motion retiming. In *Symposium on Computer Animation (SCA)* (Sept. 2006), ACM SIGGRAPH/Eurographics, pp. 205–214. 9

[MTPS04] MCNAMARA A., TREUILLE A., POPOVIĆ Z., STAM J.: Fluid control using the adjoint method. *ACM Transactions on Graphics 23*, 3 (Aug. 2004), 449–456. 3

[OLGM11] OVSJANIKOV M., LI W., GUIBAS L., MITRA N. J.: Exploration of continuous variability in collections of 3d shapes. *ACM Trans. Graph. 30*, 4 (July 2011), 33:1–33:10. 3

[PSE*00] POPOVIĆ J., SEITZ S. M., ERDMANN M., POPOVIĆ Z., WITKIN A.: Interactive manipulation of rigid body simulations. In *Computer Graphics (Proceedings of SIGGRAPH 2000)* (July 2000), Annual Conference Series, ACM SIGGRAPH, pp. 209–218. 3, 4

[RH91] RAIBERT M. H., HODGINS J. K.: Animation of dynamic legged locomotion. In *Computer Graphics (Proceedings of SIGGRAPH 91)* (July 1991), Annual Conference Series, ACM SIGGRAPH, pp. 349–358. 2

[Rov] ROVIO: *Angry Bird, http://www.rovio.com/en/our-work/games/view/1/angry-birds*. 2, 5

[Sim94] SIMS K.: Evolving virtual creatures. In *Proceedings of SIGGRAPH 94* (July 1994), Computer Graphics Proceedings, Annual Conference Series, pp. 15–22. 2

[TJ07] TWIGG C., JAMES D. L.: Many-worlds browsing for control of multibody dynamics. *ACM Transactions on Graphics (TOG)* (2007). 3, 4

[top] : *Top-down car physics, http://www.iforce2d.net/b2dtut/top-down-car*. 6

[TWSM*11] TORSNEY-WEIR T., SAAD A., MOLLER T., HEGE H., WEBER B., VERBAVATZ J., BERGNER S.: Tuner: Principled parameter finding for image segmentation algorithms using visual response surface exploration. *Visualization and Computer Graphics, IEEE Transactions on 17*, 12 (dec. 2011), 1892 –1901. 3

[WFH10] WANG J. M., FLEET D. J., HERTZMANN A.: Optimizing walking controllers for uncertain inputs and environments. *ACM Transactions on Graphics 29*, 4 (July 2010), 73:1–73:8. 3

[WFR*10] WASER J., FUCHS R., RIBICIC H., HIRSCH C., SCHINDLER B., BLOSCHL G., GROLLER M.: World lines. *Visualization and Computer Graphics, IEEE Transactions on 16*, 6 (nov.-dec. 2010), 1458 –1467. 3

[WP95] WITKIN A., POPOVIĆ Z.: Motion warping. In *Computer Graphics (Proceedings of SIGGRAPH 95)* (Aug. 1995), Annual Conference Series, ACM SIGGRAPH, pp. 105–108. 9

[WP09] WAMPLER K., POPOVIĆ Z.: Optimal gait and form for animal locomotion. *ACM Transactions on Graphics 28*, 3 (July 2009), 60:1–60:8. 3

[WRF*11] WASER J., RIBICIC H., FUCHS R., HIRSCH C., SCHINDLER B., BLOSCHL G., GROLLER M.: Nodes on ropes: A comprehensive data and control flow for steering ensemble simulations. *Visualization and Computer Graphics, IEEE Transactions on 17*, 12 (dec. 2011), 1872 –1881. 3

[YLvdP07] YIN K., LOKEN K., VAN DE PANNE M.: SIMBICON: Simple biped locomotion control. *ACM Transactions on Graphics 26*, 3 (July 2007), 105:1–105:10. 2