Computational Design of Mechanical Characters

Stelian Coros*[1] Bernhard Thomaszewski*[1] Gioacchino Noris[1] Shinjiro Sueda[2] Moira Forberg[2]
Robert W. Sumner[1] Wojciech Matusik[3] Bernd Bickel[1]

[1]Disney Research Zurich [2]Disney Research Boston [3]MIT CSAIL

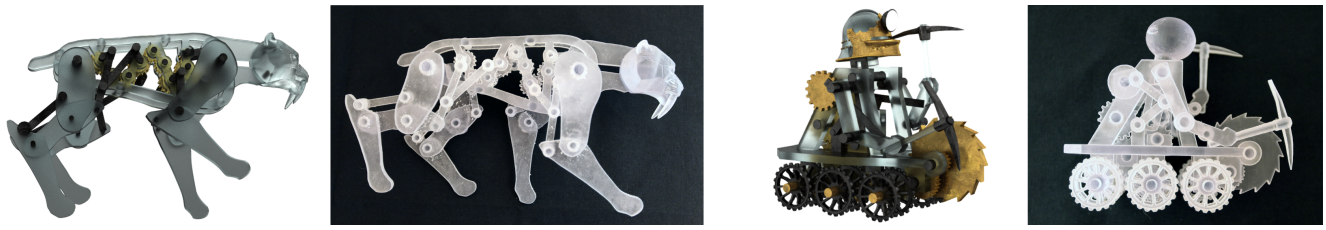


Figure 1: *The interactive design system we introduce allows non-expert users to create complex, animated mechanical characters.*

Abstract

We present an interactive design system that allows non-expert users to create animated mechanical characters. Given an articulated character as input, the user iteratively creates an animation by sketching motion curves indicating how different parts of the character should move. For each motion curve, our framework creates an optimized mechanism that reproduces it as closely as possible. The resulting mechanisms are attached to the character and then connected to each other using gear trains, which are created in a semi-automated fashion. The mechanical assemblies generated with our system can be driven with a single input driver, such as a hand-operated crank or an electric motor, and they can be fabricated using rapid prototyping devices. We demonstrate the versatility of our approach by designing a wide range of mechanical characters, several of which we manufactured using 3D printing. While our pipeline is designed for characters driven by planar mechanisms, significant parts of it extend directly to non-planar mechanisms, allowing us to create characters with compelling 3D motions.

CR Categories: I.3.6 [Computer Graphics]: Methodology and Techniques—Interaction Techniques; I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Physically based modeling

Keywords: mechanical characters, animation, fabrication, interactive design

Links: DL PDF

*The first two authors have contributed equally to this work.

1 Introduction

Character animation allows artists to bring fictional characters to life as virtual actors in animated movies, video games, and live-action films. Well-established software packages assist artists in realizing their creative vision, making almost any digital character and movement possible. In the physical world, animatronic figures play an equivalent role in theme parks and as special effects in movies and television. While these sophisticated robots are far from becoming household items, toys that exhibit mechanical movement are extremely popular as consumer products. However, unlike virtual characters, creating complex and detailed movement for *mechanical characters*, whose motion is determined by physical assemblies of gears and linkages, remains an elusive and challenging task. Although mechanical characters have been part of the toy industry since the nineteenth century [Peppe 2002], design technology for these characters has changed little and is limited to expert designers and engineers. Even for them, the design process is largely trial and error, with many iterations needed to produce an acceptable result. Since iteration times increase greatly as the complexity of the design space increases, mechanical characters are limited in scope and complexity, which in turn limits the range of possible movement and the creative freedom of the designers.

We present a computational design system that allows non-expert users to design and fabricate complex animated mechanical characters (Fig. 1). Our system automates tedious and difficult steps in the design process, and the resulting mechanical characters can be fabricated using rapid manufacturing methods such as 3D printing. Interactivity is a core design principle of our system, allowing users to quickly explore many different mechanical design options, as the motion of the characters is iteratively created.

In order to make the computational design problem tractable, we limit the scope of this work to characters that perform cyclic motions and that do not need to sense or respond to the external environment. However, within these restrictions, we wish to support a wide range of complex, user-defined motions. In order to accomplish this goal, we begin with a library of parameterized mechanical assembly types. Our system first pre-computes a sparse sampling of their parameter spaces, resulting in a representative set of motions for each type of mechanical assembly. After this precomputation step has been completed, our interactive design pipeline proceeds as follows (see Fig. 2 for a visual summary):

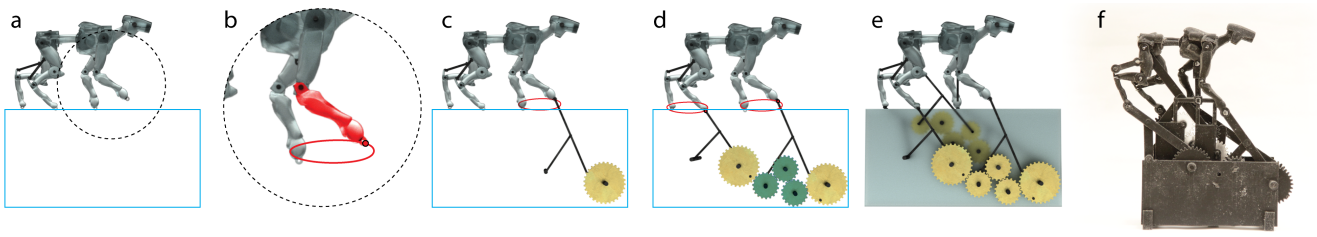


Figure 2: *Overview of our interactive design system:* a) *an input character and a gear box are provided;* b) *the desired motion of the character is specified by sketching motion curves;* c) *optimized driving mechanisms are generated by our framework;* d) *gear trains are created to connect all the driving mechanisms to one actuator;* e) *support structure is generated;* f) *the mechanical character is 3D-printed.*

1. Given an articulated character as an input, the user selects a set of actuation points on the character and sketches associated motion curves.
2. For each input curve, the precomputed database is queried to find an assembly that matches the motion specified by the user. This gives us both the type of mechanical assembly that is best suited for each motion curve, and a good initial set of parameter values (Sec. 4).
3. Appropriate driving mechanisms are instantiated, connected to the character, and their parameter values are further optimized using a gradient-based formulation (Sec. 4.2).
4. Once all driving mechanisms are set up, the gears that actuate them are connected to each other through a gear train in a semi-automatic fashion (Sec. 6.1).
5. For planar characters, our system then repositions the mechanical components to ensure that no collisions occur during operation (Sec. 6.2).
6. Finally, support structures are generated to hold the components of the assembly in place (Sec. 6.3). The complete character model and the mechanical assembly driving it are then ready for fabrication (Sec. 6.4).

2 Related Work

Mechanical Assemblies have tremendously changed our way of life since the industrial revolution — for a comprehensive introduction into mechanisms and mechanical devices we refer the interested reader to the textbook of Sclater and Chironis [2001]. The potential to use specialized software for analysis and design of mechanisms has been recognized since the early days of computers [Freudenstein 1954]. Consequently, a variety of approaches for representing mechanical structures have been presented. Some modeling systems, for instance, implement function-oriented and shape-oriented approaches, where objects, assemblies and positional relationships are represented as nodes in a graph structure, and an explicit or procedural representation is used for shape primitives [Wesley et al. 1980; Gui and Mäntylä 1994]. Similar techniques are used to implement basic functionality for commercially available CAD/CAM tools. Despite the significant benefits presented by such tools, creating complex mechanisms, such as those needed to animate mechanical characters, currently requires expert designers.

Kinematic Synthesis. Several existing methods aim at automating the design of devices that transform a specified input motion into a specified output motion. Chiou and Sridhar [1999], for instance, use symbolic matrices representing a library of mechanism as basic building blocks. Starting with an intended function, the system recursively decomposes it into simpler sub-functions until a match is found in the database. Similarly, Subramanian and Wang [1995] define valid configuration spaces for building blocks and use an iterative deepening search to generate mechanism topologies and geometries that satisfy given motion constraints. Alternatively, genetic algorithms or other stochastic search methods can be applied [Cabrera et al. 2002]. Recently, Zhu et al. [2012] suggested selecting a mechanism from a parameterized set according to a priori knowledge of its motion. The selected mechanism is then refined by optimizing over both discrete and continuous variables using simulated annealing. While this approach can effectively handle, e.g., linear, ellipsoidal or circular motions, creating mechanical characters exhibiting complex motions requires a more general formulation. In our work, we use a sampling-based approach to generate a sparse but representative set of motions that different types of mechanisms can generate. This allows us to determine which of the available mechanism types are well-suited to create a desired motion, and we use a gradient-based method in order to further optimize the mechanisms generated by our framework.

Motion and Interaction Analysis. There exists a variety of methods for analyzing the geometry of mechanisms in order to extract the kinematic constraints that define their motions. For instance, Mitra et al. [2010] present a semi-automatic technique for depicting the operation of mechanical assemblies. Their method determines the causal relationship between different parts of an assembly based solely on geometry. Such processes are also investigated for reverse engineering of scanned devices [Demarsin et al. 2007]. Recent work also explores the creation of a printable articulated model from input geometry by analyzing a skinned mesh [Bächer et al. 2012] or offering an intuitive user interface to control the placement and range of motion of joints [Calì et al. 2012]. While the resulting characters can be posed, these methods do not address the challenge of animating them. In principle, however, the characters designed using these methods could be used as input for our framework.

Design Systems and Personalized Fabrication are quickly gaining interest in the computer graphics community. Interactive systems enable non-expert users to design, for example, their own plush toys [Mori and Igarashi 2007] or furniture [Lau et al. 2011; Umetani et al. 2012]. Rapid prototyping devices allow for the fabrication of physical objects with desired properties based on custom microgemetry [Weyrich et al. 2009], reflectance functions [Malzbender et al. 2012], subsurface scattering [Hasan et al. 2010; Dong et al. 2010] and deformation behavior [Bickel et al. 2010]. Ensuring that printed objects are structurally stable, Stava et al. [2012] proposed to relieve stress by automatically detecting and correcting problematic parts of the models. In our work, we focus on the problem of designing mechanical assemblies. Inspired

Figure 3: *The driving mechanisms we use as building blocks can generate a continuous space of widely different motions. To generate optimized assemblies that best match an input motion, we first perform a coarse exploration of the space of possible motions for each mechanism, followed by a continuous parameter optimization.*

by recent work in sketch-based interfaces [Eitz et al. 2012] and design galleries [Marks et al. 1997], we present an interactive design system that allows non-expert users to intuitively create complex, animated mechanical characters.

3 Assembly simulation

The recent method proposed by Zhu et al. [2012] simulates mechanical assemblies in a two-stage process. First, the configuration of the driving assembly is computed using forward kinematics: by assuming a tree-based representation of the mechanism [Mitra et al. 2010], the configuration of a component fully determines the configuration of all other components that are connected to it. Second, the motion of the toys is computed using inverse kinematics. In contrast, we employ a constraint-based simulation of mechanical assemblies. This allows us to reconstruct the motion of the character and the driving assemblies in a unified manner, and it enables us to simulate a much broader range of mechanisms, such as assemblies with kinematic loops, which cannot be simulated using the forward kinematics approach employed by [Zhu et al. 2012]. As discussed shortly, this approach also allows us to efficiently compute gradients needed to optimize individual sub-assemblies.

3.1 Components and Connections

The mechanical assemblies we simulate consist of rigid components that are connected to each other. The world-coordinates $\overline{\mathbf{x}}$ of a point $\mathbf{x}$ expressed in the local coordinates of component i are given by $\overline{\mathbf{x}}(\mathbf{x})_i = \mathbf{T}_i + R_{\gamma_i} R_{\beta_i} R_{\alpha_i} \mathbf{x}$. Similarly, the world-coordinates $\overline{\mathbf{v}}$ of a vector $\mathbf{v}$ expressed in the local coordinates of component i are $\overline{\mathbf{v}}(\mathbf{v})_i = R_{\gamma_i} R_{\beta_i} R_{\alpha_i} \mathbf{v}$. The 6-dimensional state of the component is defined as $\mathbf{s}_i = \{\mathbf{T}_i^T, \gamma_i, \beta_i, \alpha_i\}^T$, where $\mathbf{T}_i \in \mathbb{R}^3$ denotes the global position of the component, $\gamma_i, \beta_i, \alpha_i$ are the Euler angles parameterizing the component's orientation and $R_{\gamma_i}, R_{\beta_i}$ and R_{α_i} are the corresponding rotation matrices. The rotation axes $\mathbf{n}_\gamma$ and $\mathbf{n}_\beta$ are initially set to the x and y-axes respectively. To avoid numerical problems due to the gimbal lock, they (together with the corresponding rotational degrees of freedom γ and β) can be recomputed by decomposing the component's net orientation whenever it approaches a singularity. The axis of rotation for R_{α_i} is set to the component's local z-axis, and we refer to the rotation angle α as the *phase* of the component.

The components of an assembly are linked to each other through different types of connections. Each connection c outputs a set of scalar constraints $\mathbf{C}_c$. We have implemented four types of connections that, when combined, allow us to simulate a wide variety of mechanical assemblies:

Pin Connections are used to model hinge joints between pairs of components i and j. The constraints output by this type of connection take the form $\mathbf{C}_c = \{(\overline{\mathbf{x}}(\mathbf{x}_i)_i - \overline{\mathbf{x}}(\mathbf{x}_j)_j)^T \quad (\overline{\mathbf{v}}(\mathbf{v}_i)_i - \overline{\mathbf{v}}(\mathbf{v}_j)_j)^T\}^T$. The parameters $\mathbf{x}_i, \mathbf{x}_j, \mathbf{v}_i$ and $\mathbf{v}_j$ define the position and rotation axis of the pin joint. Note that while the Pin Connections output six constraints, they only remove five degrees of freedom from the system: the two connected components can still rotate freely with respect to each other about the axis $\overline{\mathbf{v}}(\mathbf{v}_i)_i$ (or equivalently $\overline{\mathbf{v}}(\mathbf{v}_j)_j$). Attaching two pin connections between the same pair of components results in them being welded to each other.

Point On Line Connections ensure that a pin on component i always lies on a line defined on component j. This type of connection outputs constraints $\mathbf{C}_c = \overline{\mathbf{v}}(\mathbf{v}_j)_j \times (\overline{\mathbf{x}}(\mathbf{x}_i)_i - \overline{\mathbf{x}}(\mathbf{x}_j)_j)$, where $\mathbf{x}_i$ represents the local coordinates of the pin on component i, and the vector $\mathbf{v}_j$ and point $\mathbf{x}_j$ define the line that the pin should lie on. Using two point on line connections between the same pair of components allows us to model prismatic joints. As for the Pin Connections, we can create three additional scalar constraints to ensure that the two components can only rotate with respect to each other about one axis.

Phase Connections are typically used to connect gears to each other, and they directly constrain the phase of the affected components. In its most general form, the scalar constraint output by these connections has the form: $C_c = \alpha_i - f(\alpha_j)$. The function f depends on the types of gears that are connected. For spur and bevel gears, $f(\alpha) = -r * \alpha$, where r is the ratio of the number of teeth of the two gears. For non-circular gears, f is slightly more involved, as described in Sec. 4.3. As a special case of the phase connection, we allow gears to be directly connected to a virtual actuator, which we call the Input Driver.

Fixed State Connections affect individual components and outputs six constraints: $\mathbf{C}_c = \mathbf{s} - \mathbf{s}_d$, where $\mathbf{s}$ is the state of the affected component, and $\mathbf{s}_d$ is the desired state. The phase variable can be optionally omitted from the list of constraints, allowing components to rotate freely about one axis. These connections are used to simulate support structures such as walls or shafts, and to allow users to temporarily freeze parts of the characters.

3.2 Simulation

To run a simulation step, we first advance the phase of the Input Driver forward in time. We then solve the following optimization

problem in order to compute component state values that satisfy all the constraints:

$$\min_{\mathbf{s}} \frac{1}{2}\mathbf{C}^T\mathbf{C}, \tag{1}$$

where $\mathbf{C}$ is a vector collecting the scalar constraints output by all the connections in the assembly. We use a standard Newton-Raphson method to solve Eq. (1), assuming that the phase of the Input Driver fully determines the configuration of the entire assembly. The mechanical assemblies that we simulate, however, can be under or over constrained, if the user employs too few or too many driving mechanisms. Our framework detects these problems (if the system is over-constrained, Eq. (1) has a non-zero minimum; if it is under-constrained, the hessian of $\frac{1}{2}\mathbf{C}^T\mathbf{C}$ is singular) and allows the user to edit the assembly as required.

4 Mechanism Design

Our interactive design system allows users to sketch a set of curves that specify the motion of different parts of the input character. For each of the input curves, our system outputs optimized driving mechanisms that are attached to the character and control its motion. Two problems need to be addressed in order to achieve this. First, out of a library of input driving mechanism types, our system must choose an appropriate one. Second, the parameters of the selected mechanism must be optimized in order to best match the motion specified through the input animation curves.

For simple motion trajectories specified by circular, elliptical or straight-line curves, the map to an appropriate driving mechanism can be manually specified [Zhu et al. 2012]. However, as the complexity of the driving mechanisms increases, the space of possible output motions grows significantly (see, e.g., Fig. 3), and this approach very quickly becomes impractical. Furthermore, in order to accommodate arbitrary, black-box mechanisms, we do not assume that there exist analytic expressions describing the relationship between a given set of parameters and the motion they lead do. The methods we describe in this section allow us to address the challenges of working with complex driving mechanisms.

Overview Our system takes as input a library of parameterized mechanism types as illustrated in Fig. 4. Each of these mechanisms contains at least one driving gear that is either connected directly to the Input Driver or to other gears. Once instantiated, the assemblies are attached to the character, typically through pin connections. While the locations of the pin joints on the input character are generally user-specified, the attachment points on the driving mechanisms are parameterized. For each type of driving mechanism, the locations of the pin and point-on-line connections between the individual components as well as the positions of the support shafts are also parameterized. As an example, the mechanism types shown in Fig. 4 have between 6 and 12 parameters.

To address the challenge of determining which assembly type is best suited for a desired animation, we begin by performing a coarse exploration of their parameter spaces through sampling (Sec. 4.1). This pre-processing step results in a database that stores a representative set of possible motions for each class of mechanism, as illustrated in Fig. 3. Rather than recording the motion trajectories of the individual gears and linkages comprising the mechanisms, however, the database represents a map between specific points in an assembly's parameter space and the motion curves traced out by the mechanism's attachment point, since these trajectories fully define the motion of the input character.

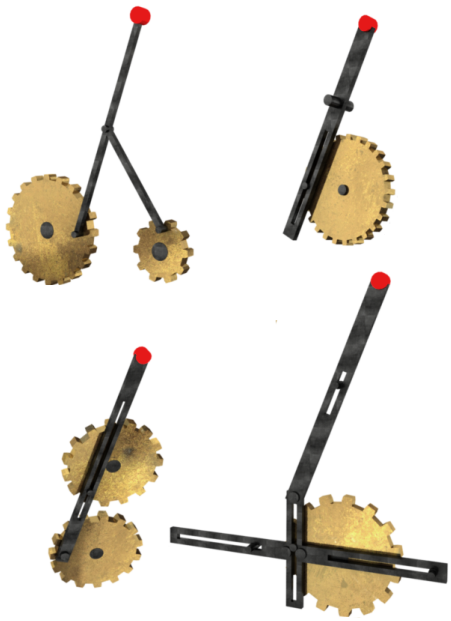


Figure 4: *We assume as input a library of parameterized driving mechanisms such as the ones illustrated here. These building blocks are instantiated by our framework, optimized and attached to the input character, driving its motion. The locations of the attachment points are highlighted.*

The input motion curves provided to our system directly specify the trajectories that the attachment point of a driving mechanism should trace out. To determine which type of mechanism is capable of best matching an input curve, it is therefore sufficient to search the precomputed database using the curve similarity metric described in Sec. 5. Once a mechanism is selected from the database, it is first scaled, rotated and translated using the transformation that best aligns the curve traced out by its attachment point to the input curve (see Sec. 5). If a designated gear box is provided as input, then we must ensure that the bounding box of the chosen mechanism fits inside. If this is not the case, we discard the mechanism and proceed to the next best match.

As the database we pre-compute represents only a sparse sampling of the parameter space for each assembly type, the selected mechanisms need to be further optimized, as described in Sec. 4.2. We note that the driving mechanisms obtained by querying the database typically provide a good starting point for the continuous optimization method we use. This significantly improves convergence rates and reduces the likelihood of encountering bad local-minima, which is a very real possibility given the highly non-linear relationship between assembly parameters and the resulting motions.

4.1 Parameter Space Exploration

We approximate the space of motions that a given type of mechanism can generate by sampling its parameter domain. Our goal in this context is to obtain a good coverage of the space of possible motion curves with a sparse set of samples. The latter point is important since it determines the response time for curve retrieval during interactive design. In order to generate a sparse set of samples that cover a large variety of motion curves, we use a Poisson-disk sampling scheme in metric space. This amounts to requiring that for any pair of parameter-space samples, the distance between the two corresponding curves must not be lower than a given thresh-

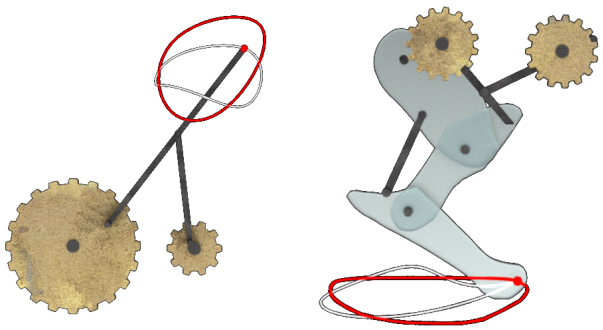


Figure 5: *The trajectory (gray polyline) traced out by a marker point (red dot) to approximate a target curve (red polyline). A marker point can either be located on the driving mechanism (*left*) or on a part of the character that is driven by a mechanism (*right*).*

old. Unfortunately, fast Poisson-disk sampling methods [Bridson 2007] cannot be applied in our setting: while it easy to satisfy the minimum distance criterion in parameter space, this is not the case in metric space since the mapping between the parameters of the mechanism and its generated motion curve is non-trivial.

We therefore generate the samples using a simple recursive process. For each accepted sample, we generate a number of new samples by uniformly probing the parameter space around the current sample's location. For each generated sample, we compute the corresponding curve, and using the distance metric described in Sec. 5, we check whether it is too close to any of the existing samples in the database. If this is the case, we reject the sample, otherwise it is added to the database. We note that it is possible that some regions in the parameter space lead to infeasible configurations. As a result, although our sampling strategy is very effective for exploring connected regions of a mechanism's parameter space, we have no guarantees that it exhaustively explores the entire space of feasible parameters.

4.2 Continuous parameter optimization

After selecting an appropriate driving mechanism from the database, its parameters need to be further fine-tuned, and we use a continuous optimization method for this purpose. We assume that a *marker* point specified on the character should follow a user-provided curve as closely as possible. The marker can either coincide with the attachment point of the driving mechanism, or it can be located at an arbitrary point on the character, as illustrated in Fig. 5. To optimize the resulting motion, we minimize the following objective as a function of the mechanisms's parameters $\mathbf{p}$:

$$F = \frac{1}{2}\int_{t=0}^{1} (\mathbf{x}(\mathbf{p},\mathbf{s}_t) - \hat{\mathbf{x}}_t)^T (\mathbf{x}(\mathbf{p},\mathbf{s}_t) - \hat{\mathbf{x}}_t) dt, \qquad (2)$$

where t is the phase of the Input Driver, such that when $t = 1$ a full cycle of the animation has been completed. $\mathbf{x}(\mathbf{p},\mathbf{s}_t)$ and $\hat{\mathbf{x}}_t$ denote the position of the marker point and its target position at phase t. The state of the assembly at phase t, $\mathbf{s}_t$, is computed by minimizing Eq. 1 after instantiating the mechanism with the current set of parameters $\mathbf{p}$. As the parameters of the mechanisms directly affect the connections between the different components, for the remainder of this section we write the system constraints as an explicit function of the parameters and the state variables i.e. $\mathbf{C}(\mathbf{p},\mathbf{s}_t)$.

Note that in Eq. (2), we use a point-based metric to measure the similarity between the input and output motion curves, rather than the more involved metric described in Sec. 5. Because the curve traced out by the marker point is already aligned with the input curve, and their shapes are relatively similar, we have found this to be sufficient. For any phase t, the gradient of the matching objective $\frac{\partial F_t}{\partial \mathbf{p}}$ is given by:

$$\frac{\partial F_t}{\partial \mathbf{p}} = (\frac{\partial \mathbf{x}}{\partial \mathbf{p}} + \frac{\partial \mathbf{x}}{\partial \mathbf{s}_t}\frac{\partial \mathbf{s}_t}{\partial \mathbf{p}})(\mathbf{x}(\mathbf{p},\mathbf{s}_t) - \hat{\mathbf{x}}_t), \qquad (3)$$

where all the partial derivatives are evaluated at $(\mathbf{p},\mathbf{s}_t)$. In the equation above, $\frac{\partial \mathbf{x}}{\partial \mathbf{p}}$ represents the change in the marker's position when the state of the sub-assembly remains unaltered. For instance, changing the location of the attachment point in the linkage structure shown in Fig 5 (left) does not affect the evolution of the state of the sub-assembly, but results in a different curve being traced out. The term $\frac{\partial \mathbf{x}}{\partial \mathbf{s}_t}$ is evaluated analytically, and the remaining term, $\frac{\partial \mathbf{s}_t}{\partial \mathbf{p}}$, represents the change in the state of the assembly as its parameters change. This term cannot be evaluated analytically because the sub-assemblies we optimize can be arbitrarily complex, and in general there is no closed-form solution describing their motion. Finite differences can be used to estimate the derivatives, but this requires Eq. (1) to be solved $O(|p|)$ times to a high degree of accuracy, which is too computationally demanding for interactive applications. Instead, we exploit the implicit relationship between the parameters of the sub-assembly and the state configuration that results in all constraints being satisfied, i.e., $C(\mathbf{p},\mathbf{s}_t(\mathbf{p})) = 0$. According to the *Implicit Function Theorem*,

$$\frac{\partial \mathbf{s}_t}{\partial \mathbf{p}} = -\frac{\partial C}{\partial \mathbf{s}_t}^{-1}\frac{\partial C}{\partial \mathbf{p}}. \qquad (4)$$

We estimate $\frac{\partial C}{\partial \mathbf{p}}$ using finite differences. This requires us to instantiate sub-assemblies with different parameters $\widetilde{\mathbf{p}}$ and compute the value of the constraints $C(\widetilde{\mathbf{p}},\mathbf{s}_t(\mathbf{p}))$. We note that this does not require Eq. (1) to be minimized, and its computational cost is therefore negligible. The matrix $\frac{\partial C}{\partial \mathbf{s}_t}$ is computed analytically. As noted in Sec. 3, our mechanical assemblies can have more constraints than degrees-of-freedom. Since the additional constraints are redundant however, the matrix $\frac{\partial C}{\partial \mathbf{s}_t}$ still has full column rank, and in such cases, we instead use the pseudo-inverse $\frac{\partial C}{\partial \mathbf{s}_t}^{+} = (\frac{\partial C}{\partial \mathbf{s}_t}^T \frac{\partial C}{\partial \mathbf{s}_t})^{-1}\frac{\partial C}{\partial \mathbf{s}_t}^T$.

We minimize Eq. (2) by evaluating the matching objective and its gradient at a discrete number of points (i.e., t values). To improve convergence rates we use the BFGS quasi-Newton method to estimate the approximate Hessian $\frac{\partial}{\partial \mathbf{p}}\frac{\partial F}{\partial \mathbf{p}}$ [Nocedal and Wright 2006].

4.3 Timing control via non-circular gears

Controlling the timing of motions is an integral part of character animation. We therefore also allow users to explicitly control the timing of the characters' motions. This is accomplished through the use of non-circular gears, which are always used in pairs: a constant angular velocity input signal applied to the first gear gets transformed to variable angular velocity for the second gear. The phase-dependent ratio of angular velocities of the two gears depends on the shape of their profiles, and in particular, on the ratio of the gears' radii at the point where they mesh. Asking users to directly provide the phase-dependent ratio of angular velocities, however, is not intuitive, as the integral of this ratio needs to remain constant (otherwise one gear rotates more than the other over a full cycle). Instead, we allow users to define the relative phase profile between the two gears, as illustrated in Fig 6 (left). This is accomplished by allowing the user to set samples $(\alpha_{1i},\alpha_{2i})$ that

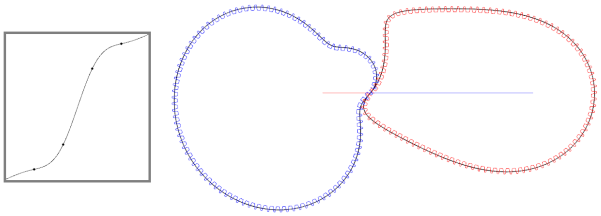


Figure 6: *Non-circular gears control the timing of the motion.*

specify the phase of the second gear as a function of the phase of the first gear. Two additional sample points, $(0,0)$ and $(1,1)$ are automatically added to ensure that a full revolution of the first gear corresponds to a full revolution of the second gear. These sample points define the function $f(\alpha)$ used by the Phase Connection, as described in Sec. 3. In particular, we compute $f(\alpha)$ using Radial Basis Function interpolation, such that $f(\alpha_{1i}) = \alpha_{2i}, \forall i$ with the additional constraints $f'(0) = f'(1), f''(0) = f''(1)$. These constraints ensure that the ratio of angular velocities varies smoothly. The interpolation uses thin plate splines as basis functions, and a low degree polynomial to accommodate the additional constraints.

The phase profile function $f(\alpha)$ is used to simulate the mechanical assemblies, and also to generate the geometry of the non-circular gears. Let $r_1(\alpha)$ and $r_2(f(\alpha))$ represent the non-constant radii that define the pitch surfaces of the two non-circular gears. Since the two gears are assumed not to move relative to each other, $r_1(\alpha) + r_2(f(\alpha)) = a$, where a is the distance between the two gears. In addition, the relative angular velocity $\frac{\partial f}{\partial \alpha}(\alpha)$ of the two gears is given by the ratio of the radii at the meshing point $r_1(\alpha)/r_2(f(\alpha))$. These two equations are sufficient to compute the radii of the two gears for any α, and therefore their pitch surfaces. The teeth of the gears are added procedurally, orthogonal to the pitch surfaces.

5 Curve Metric

The driving mechanisms described in Sec. 4 trace out planar, closed curves that we represent by polygons. In order to find the driving mechanism from our library that can best approximate a user-provided input curve, we must measure the distance, or similarity, between such curves. As an obvious candidate, the Hausdorff distance is simple to compute, but it does not consider the course of the curve, which can lead to unexpected similarities. A metric that is frequently used in the context of handwriting recognition is the Fréchet distance, which is also referred to as dog-walking metric [Alt and Godau 1995]: it is the length of the shortest leash necessary to connect a dogwalker and its dog as they progress along the two curves, possibly at different rates but without backtracking. While the discrete Fréchet distance [Eiter and Mannila 1994] can be computed in (sub-)quadratic time with respect to the number of points in the polygons, our experiments showed that the resulting matching quality is not always satisfying (see Sec. 7.1).

It is generally unlikely that a single parameter metric is able to reflect the similarity as perceived by the user across a broad range of curves. On the other hand, manually selecting coefficients for multi-parameter metrics is very difficult. We therefore take a different, more user-centered approach: we start by converting curves into feature vectors that capture salient characteristics such as length, curvature, area and so forth. We then formulate our metric as a bilinear form on differences in feature vectors and optimize its coefficients using user-generated data.

Let $\mathbf{c}_i$ and $\mathbf{c}_j$ denote closed, planar polygonal curves and let $\mathbf{f}_i$ and $\mathbf{f}_j$ denote their corresponding feature vectors. We define the distance metric as

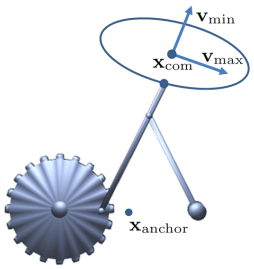
$$d(\mathbf{c}_i, \mathbf{c}_j) = ||\mathbf{f}_i - \mathbf{f}_j||_A = \sqrt{(\mathbf{f}_i - \mathbf{f}_j)^T \mathbf{A} (\mathbf{f}_i - \mathbf{f}_j)}\,,$$

In order for d to be a proper metric, i.e., satisfying positivity and the triangle-inequality, the matrix $\mathbf{A}$ has to be positive semi-definite. For simplicity, we consider only diagonal $\mathbf{A}$ such that this condition reduces to $\mathbf{A}_{ii} \geq 0$.

5.1 Curve Features

We do not want the metric to be sensitive to absolute position, size, and orientation of the curve in world-space since we can always translate, scale, and rotate the entire sub-assembly as required. However, we must capture these quantities relative to the sub-assembly, since design constraints such as the location of the gear box dictate how the sub-assembly has to be positioned, scaled and oriented relative to an input sketch of the user.

Before computing the actual features of a curve, we perform a number of pre-processing and normalization steps. We start by re-sampling the curve using constant length strides along its original segments. Timing information is discarded at this point, since we can explicitly control the velocity profile as a post process using non-circular gears (see Sec. 4.3). In order to normalize the curve with respect to global transformations, we translate it such that the anchor $\mathbf{x}_{\text{anchor}}$ of its generating mechanism lies at the origin. Next, we compute the two unit-length principal directions $\mathbf{v}_{\max}$ and $\mathbf{v}_{\min}$ of its points using principal component analysis. Finally, we extract the lengths $l_{\max}$ and $l_{\min}$ of the curve along the principal directions and normalize its scale by multiplying by $1/l_{\max}$.



We compute all features of the curve in this normalized setting. Let $\mathbf{c}_{i,j}$ denote the n_p vertices of $\mathbf{c}_i$ and define edges vectors and directors as $\mathbf{e}_j = \mathbf{c}_{i,j} - \mathbf{c}_{i,j-1}$, respectively $\mathbf{t}_j = \mathbf{e}_j / ||\mathbf{e}_j||$. With the first three features, we capture the geometric characteristics of the curve: its length ($\mathbf{f}_{i,1} = \sum_j^{n_p} ||\mathbf{e}_j||$), area ($\mathbf{f}_{i,2} = \frac{1}{2}\sum_j^{n_p} \mathbf{c}_{i,j} \times \mathbf{c}_{i,j-1}$), and ellipticity ($\mathbf{f}_{i,3} = l_{\min}/l_{\max}$). We furthermore extract the position of the curve relative to its generating mechanism as $\mathbf{f}_{i,4} = ||\mathbf{d}||$, where $\mathbf{d} = \mathbf{x}_{\text{anchor}} - \mathbf{x}_{\text{com}}$ is the vector between the anchor of the driving mechanism and the center of mass of the curve. The relative orientation of the curve is stored as $\mathbf{f}_{i,5} = \arcsin(||\mathbf{d}/||\mathbf{d}|| \times \mathbf{v}_{\max}||)$. Finally, we use the number of intersections between non-neighbor segments as feature $\mathbf{f}_{i,6}$.

While the above features capture the geometric characteristics of a given curve in an integral sense, we would also like to distinguish curves based on local variations. To this end, we use two additional indicators that are computed for pairs of curves: the L_2 norm of the vertex-wise difference in position and curvature. Since we cannot assume that the starting points of the two curves are in correspondence, we use the minimum distance values obtained by offsetting the starting index of one of the curves and define

$$d^2_{\text{pos}}(\mathbf{c}_i, \mathbf{c}_j) = \frac{1}{n_p} \min_l \sum_k ||\mathbf{c}_{i,k} - \mathbf{c}_{j,k+l}||^2 \qquad (5)$$

$$d^2_{\text{angle}}(\mathbf{c}_i, \mathbf{c}_j) = \min_l \sum_k (\kappa_{i,k} - \kappa_{j,k+l})^2\,, \qquad (6)$$

where $\kappa_{i,k} = \frac{2\mathbf{t}_{j-1} \times \mathbf{t}_j}{1+\mathbf{t}_j \cdot \mathbf{t}_{j-1}}$ is the discrete curvature at vertex k (see [Bergou et al. 2010]). We evaluate these indicators whenever the distance between two curves has to be determined and add their square roots as additional entries to $(\mathbf{f}_i - \mathbf{f}_j)$. Note that, unlike the other features, (5) and (6) require the curves to have the same number of vertices. If this is not the case, we upsample the coarser one.

5.2 Metric Training

It is not obvious how to choose the coefficients of the feature-based curve metric in order to best reflect a desired notion of similarity. We therefore automatically compute the coefficients that best explain a given set of interactively generated training data. The training data consists of a set $\mathbf{S}$ of pairs of similar curves as well as a set $\mathbf{D}$ of dissimilar curves. Using the formulation of Xing et al. [2002] as a basis, we compute the coefficients by solving the following constrained optimization problem

$$\min_{\mathbf{A}} \sum_{(i,j)\in\mathbf{S}} ||\mathbf{f}_i - \mathbf{f}_j||_A^2 + k_{\text{reg}} \sum \mathbf{A}_i \quad (7)$$

$$\text{s.t. } ||\mathbf{f}_k - \mathbf{f}_l||_A \geq \gamma, \quad \forall (k,l) \in \mathbf{D} \quad \text{and} \quad \mathbf{A}_i \geq 0 \quad (8)$$

The objective term (7) asks for small coefficients $\mathbf{A}_i$ (k_{reg} is a small regularizing scalar) that minimize the distance between similar curves. The inequality constraints (8) require dissimilar pairs to have a distance greater than a given threshold value γ, which for all our experiments is set to 1.

The choice of training data determines the coefficients and thus the behavior of the metric. Since it is difficult for the user to select good data without getting feedback on its quality, we use an iterative training scheme. During an initial stage, we ask the user to select a few (e.g., five) similar and dissimilar pairs from sets of nine randomly generated curves. We then compute a first guess for the coefficients from this training data by solving the optimization problem (7-8) using sequential quadratic programming (SQP). In the second stage of the training process, we randomly generate pairs of curves and categorize them into four similarity classes ('similar', 'probably similar', 'probably different' and 'different') according to their current distance. The pairs are then presented to the user three at a time with a color-coding indicating their similarity class. Apart from providing feedback on the current behavior of the metric, we also allow the user to flag any of the pairs as 'similar' or 'different', which leads to pairs being added to the list of objective terms, respectively constraints. The metric coefficients are then recomputed using the updated lists. In addition to adding new pairs, we also ask the user to reconfirm or dismiss pairs from the lists that belong to the two uncertain categories.

The system's response to user input is virtually instantaneous as solving the SQP takes only a few milliseconds even with more than a hundred pairs of curves in the lists. In our experiments, we noticed that the training scheme converges quite fast: after 30-40 iterations, the metric typically produces stable results that are consistent with the perceived similarity of the user. We provide an analysis of the matching quality for our metric in Sec. 7.1.

6 Character Finishing

Once the driving mechanisms that generate the desired motion of the character are designed, three more steps have to be performed before the character can be manufactured: the mechanisms have to be connected to an input driver using intermediate gears (Sec. 6.1), the assembly needs to be edited to ensure there are no collisions between the various moving parts (Sec. 6.2), and a support structure that holds all components in place has to be designed (Sec. 6.3).

the mechanisms have to be connected to an input driver using intermediate gears (Sec. 6.1), the assembly needs to be edited to ensure there are no collisions between the various moving parts (Sec. 6.2), and a support structure that holds all components in place has to be designed (Sec. 6.3).

6.1 Gear Optimization

Each of the driving mechanisms of the assembly has to be connected to an input driver, which can be a gear connected to a servomotor, a handle operated by the user, or a wheel in contact with the ground. Each type of driving mechanism has one or more driving gears. The location and angular velocity of these driving gears are fixed during the mechanism design step, but other parameters such as their radii can vary. For simplicity, we assume that the driving gears as well as the input driver have the same axis of rotation, which essentially reduces the problem to 2.5 dimensions. When this assumption does not hold, we split the path between driving gears and input drivers with bevel gears. Given this input data, we seek to find a set of intermediate gears that connect pairs of existing gears, as selected by the user. This problem consists of two parts: first, we have to determine the gear layout, i.e., the number of gears and how they connect to each other. Second, we must compute the parameters of the gears that satisfy a set of constraints (meshing of connected gears, desired angular velocities, no intersections), while minimizing an objective function that measures the quality of the resulting gear train.

Gear Layout Generation Automatically computing a gear layout subject to given optimality conditions (e.g., minimum number of gears or shafts) is difficult due the combinatorial nature of the problem. Each time such a layout is generated, it has to be tested using continuous optimization and it is unclear how to exchange feedback between these disconnected parts. Fortunately, it is a rather simple matter for the user to create the gear layout interactively: we let the user select pairs of gears that should be connected to each other. Depending on the relative location of the gears (in one plane, on the same axis or neither), we generate parallel and sequential gear-to-gear connections (Fig. 7). Gears on the same axis are connected in parallel, gears in the same plane are connected with an automatically determined number of in-between gears depending on the signs of the angular velocities, the distance between them and their radii. The third case is treated with both a parallel and a sequential connection. These connections are then transformed to a set of constraints and objectives for the continuous optimization scheme. The result of the optimization is added to the mechanical assembly, and the parameters of all the gears that were involved in the optimization process are treated as hard constraints for subsequent steps. See also the accompanying video for a visual demonstration of this process. Although this way of iteratively connecting pairs of gears can be more restrictive than connecting all the existing gears at once, our experiments show that it is a very intuitive process that leads to fast results.

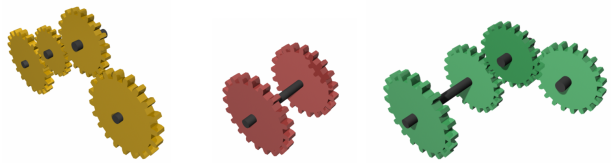


Figure 7: *Types of connections generated during interactive gear layout: sequential (*yellow*), parallel (*red*), and combined (*green*).*

Gear Parameter Optimization The gear layout step provides us with a number of n_g interconnected gears, each of which we assign five degrees of freedom $\mathbf{p}_i = (x, y, z, r, \omega)^T$, where x, y and z are the gear's coordinates, r is its radius, and ω its angular velocity. Without loss of generality, we assume that the z-axis is normal to the surface of the gears. We convert the connections between the gears into a set of equality constraints $\mathbf{C}_i$ and inequality constraints $\mathbf{I}_i$, as well as a number of n_g objective terms $\mathbf{F}_i$.

A sequential connection between two gears g_i and g_j is defined by three constraints that ensure the gears mesh properly:

$$\begin{aligned} C^{\text{depth}}_{\text{seq}}(\mathbf{p}_i, \mathbf{p}_j) &= z_i - z_j \\ C^{\omega}_{\text{seq}}(\mathbf{p}_i, \mathbf{p}_j) &= r_i\omega_i + r_j\omega_j \\ C^{\text{mesh}}_{\text{seq}}(\mathbf{p}_i, \mathbf{p}_j) &= \sqrt{(x_j - x_i)^2 + (y_j - y_i)^2} - (r_i + r_j)\ . \end{aligned}$$

The first constraint requires the gears to lie in the same plane (one could also take into account the thickness of the gears by formulating corresponding inequality constraints instead). The second constraint requires that the relative angular velocities of the two gears is correct, whereas the third one ensures that the distance between them is equal to the sum of their radii. We model parallel gear connections with four constraints,

$$\begin{aligned} C^{y}_{\text{par}}(\mathbf{p}_i, \mathbf{p}_j) &= y_i - y_j\ , \quad C^{x}_{\text{par}}(\mathbf{p}_i, \mathbf{p}_j) = x_i - x_j \\ C^{\omega}_{\text{par}}(\mathbf{p}_i, \mathbf{p}_j) &= \omega_i - \omega_j \\ I^{\text{depth}}_{\text{par}}(\mathbf{p}_i, \mathbf{p}_j) &= z_j - z_i - h \end{aligned}$$

The first two ensure that the gears are aligned correctly, the third one asks that they have the same angular velocity, since both gears are welded to the same support shaft, and the last constraint requires that the gears should not be closer than the average gear thickness h. In addition to these connections, we create non-intersection constraints for all pairs of gears that are not connected. To this end, we convert the distance constraint from above to inequality form and add a small negative safety threshold in order to prevent gears from being too close to each other. We switch these constraints on whenever two gears are closer than the gear thickness in the z direction. Finally, we introduce inequality constraints in order to enforce a minimum admissible gear radius as well as unary equality constraints that fix the parameters of gears that have been optimized by previous user operations.

Given these constraints, we compute optimal gear parameters $\mathbf{p}$ by solving the constraint optimization problem

$$\mathbf{p} = \arg\min_{\tilde{\mathbf{p}}} \sum_{i}^{n_g} ((\tilde{r}_i - r_i^{\text{target}})^2 + k_{\text{reg}}||\tilde{\mathbf{p}}_i - \bar{\mathbf{p}}_i||^2) \qquad (9)$$
$$\text{s.t.} \quad \mathbf{C}_i(\tilde{\mathbf{p}}) = 0 \quad \text{and} \quad \mathbf{I}_i(\tilde{\mathbf{p}}) \geq 0\ ,$$

where r_i^{target} denotes a user-provided (or otherwise automatically inferred) desired radius for gear i, k_{reg} is a small regularizer coefficient, and $\bar{\mathbf{p}}$ represents the set of initial parameters. Since some of the constraints are nonlinear, we solve (9) using a standard sequential quadratic programming solver.

6.2 Collision-free Layering for Planar Motions

Up to this point, no measures were taken to prevent different moving parts of the assembly from intersecting with each other. Our framework allows users to intuitively edit the assembly by independently moving the driving mechanisms if intersections are detected. However, it would be desirable to also provide an automatic solution to this problem. For general three-dimensional motions, this is very challenging, and it would require a close integration with the mechanism design phase. While conceptually clean, this approach would prevent us from treating sub-assemblies in isolation, thus rendering the optimization process significantly more complex. For assemblies where all the components move along parallel planes, and many of our results fall in this category, we can however address the intersection problem automatically by offsetting each component along the direction normal to this motion plane.

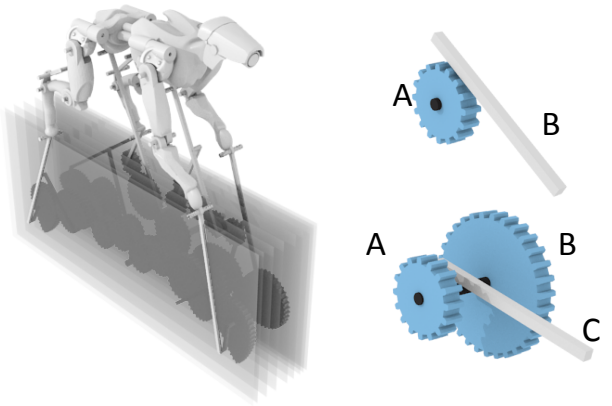


Figure 8: Left*: component placement in different planes to remove collisions.* Right*: types of collisions that can occur between components.*

Overview We aim to place each component in a different plane, such that as the animation is playing, the assembly remains collision-free. We discretize the 3D space occupied by the assembly into layers that are parallel to the plane along which the components are moving (Fig. 8, left), and we assign each component a layer (or plane) index. We start the process by checking for collisions between each pair of components (gears, linkages, and body parts of the character) that were assigned the same layer index. Note that we do not need to deal with gear-gear collisions, since we apply non-intersection constraints to all pairs of gears in the gear parameter optimization stage of the pipeline (Sec. 6.1). We perform standard collision checks between the triangle meshes of the components, after projecting them onto the motion plane. The two basic types of collisions that may occur are shown in Fig. 8 (right): on the top, components A and B are detected as colliding; on the bottom, component C is detected as colliding with a pin that connects components A and B. For the first example, components A and B cannot be assigned to the same layer, while for the second example, component C cannot lie in a layer between those assigned to A and B. We solve the layer assignment problem using boolean optimization, a variant of discrete constrained optimization, with the non-intersection conditions as constraints.

Algorithm With n components and m layers, there are m^n possible layer assignments, since each component can be placed in any of the layers. Even for relatively small problems, brute force approaches are intractable. One option for solving this problem is with integer programming, which can work quite well especially if the problem is linear. Some of our constraints, however, are non-linear. The non-linearity is due to the second type of constraint shown in Fig. 8, which states that a variable cannot lie between two other variables, a disjunctive constraint expressed as $(C < A) \vee (B < C)$. We therefore pose this as a constraint satisfaction problem (CSP) for which there are efficient, robust solvers for the scale of problems that we are interested in.

The CSP formulation assumes that we already know the number of discrete layers m. Since this number is typically not known *a priori*, we seek to find the minimum number of layers such that all components can be laid out without collisions. We start by setting $m = 4$ and increment until we obtain feasible solutions. Usually, there are multiple solutions once a feasible m is found. By repeatedly running the constraint solver, we obtain a set of feasible solutions, and select the one that minimizes the total length of the pins connecting the components of the assembly. We then estimate the thickness of each layer as the maximum thickness of the components assigned to it and offset each component to the center of the layer it was assigned to.

The worst-case running time for the CSP solver is exponential. However, even for our largest examples, which consists of close to 100 components and more than 10 layers, an off-the-shelf CSP solver (Walksat [Selman et al. 1993]) using default parameters was able to generate 1000 random solutions in less than 30 seconds. For comparison, a naive random search was unable to find a single solution within a reasonable amount of time (several hours).

6.3 Support Structure

At this point in the design process, the components of the assembly are interconnected and non-intersecting, but they are still floating in space. As a last step, a support structure is designed to ensure that the support shafts needed by the various components in the assembly are held in place. In many cases, the support structure is subject to aesthetic considerations and is thus best designed by the user. Whenever this is not the case, we use a simple algorithm to assist the user in the design of the support.

We start by computing a bounding box that contains all components of the assembly except for the character. We voxelize this volume on a regular grid and flag as occupied all voxels that lie inside the volume swept by the individual components during a full cycle of animation. The result of this process is a set of non-occupied voxels that define regions where supporting shafts can be inserted and grounded without interfering with the other moving objects. This information is then presented to the user, who can select planar slices from the non-occupied voxel grid in order to instantiate support walls. Once such a wall is created, we automatically connect to it all the support shafts that can reach it without passing through occupied voxels of the grid. During this process, we provide feedback to the user regarding which shafts still require support. Once all shafts are connected, the assembly is ready for manufacturing.

6.4 Fabrication

We use rapid manufacturing technology to create physical prototypes of our mechanical characters. Many examples exhibit a layered structure that allows easy decomposition of the mechanical components. In such cases, it is most efficient in terms of material cost and fabrication time to fabricate all components individually and assemble the character afterwards. Another advantage of this layer-wise fabrication is that it lends itself to less expensive manufacturing devices such as laser cutters or services such as *Shapeways*. Characters with internal driving mechanisms can be quite cumbersome to assemble, which is why we manufacture these examples in one piece using a 3D printer.

7 Discussion and Results

To demonstrate the versatility of our framework, we designed ten animated mechanical characters, many of which we also manufactured. These results are presented in detail in the accompanying video, and they are summarized in Table 1. Before presenting our results, we first validate some important design choices and discuss alternative approaches.

7.1 Validation

Metric Comparison In order to compare the matching quality of our metric to alternative measures, we created a set of hand-drawn curves and retrieved the closest one for each of them from a database containing 1000 randomly generated samples each for a four- and a five-bar linkage. Fig. 9 shows the best matching curves (in red) for three different metrics: our feature-based metric with optimized coefficients (first row), the discrete Fréchet distance [Eiter and Mannila 1994] (second row) and our feature-based metric with all coefficients set to 1.0. While the two alternatives found reasonable curves in many cases, some of the matches (e.g., row 2, column 2 or row 3, column 4) are clearly off. Although ranking the perceived similarity is difficult, our optimized metric yields well-matching curves in all cases.

Driving Mechanisms Generation and Optimization We use four main types of driving mechanisms, as illustrated in Fig. 4. The two mechanisms on the left columns are driven by two gears each. Altering the relative angular velocity of the two gears also affects the output motions. To ensure that the period of the output motion is the same as for the input driver, we create four variations of each of these assemblies, where the ratio of the angular velocities of the two gears are fixed to $-2, -1, 1$ and 2 respectively. The two mechanism types on the right are much simpler, and it is therefore not required to perform the parameter exploration for them. Rather, when needed, we proceed straight to the continuous optimization stage. Consequently, our motion database is built using 8 distinct mechanism types. For every type of mechanism we store up to 3000 representative motions. The parameter space exploration took between 0.5 and 2.5 hours until no further samples could be found. Even with a simple linear search, the database retrieval time is negligible. Using the approach discussed in Sec. 4.2 to compute gradients is 20 to 30 times faster than using finite differences, depending on the number of parameters of the assembly. As a result, the continuous optimization process is interactive, especially when using the initial parameter values obtained by querying the database.

7.2 Summary of Results

Designing the motions and the corresponding driving mechanisms for the mechanical characters presented in this paper took less than half an hour on average. The characters that were 3D printed in one piece (*Cyber Tiger, Clocky, DrillR*) were ready for fabrication within one day, with most of the additional time being spent on generating the support structures. 3D Printing took up to 42 hours, and removing the support material up to two hours. The characters that were created component-by-component (*Pushing Man, EMA, Bernie*), required some additional manual work to prepare for manufacturing and to assemble, but printing times were significantly lower.

The simplest of our examples is *Pushing Man* (Fig. 10), a character inspired by the *Sisyphus* automata model [Johnson 2010]. Its motion is generated by three driving mechanisms, one for each leg and one for the chest. Despite the simplicity of the design and the relatively small number of driving mechanisms, the resulting motion is fluid and compelling.

Our design interface makes it easy for non-expert users to create mechanical characters of higher complexity. For instance, the *EMA* characters show an example of different animations being created

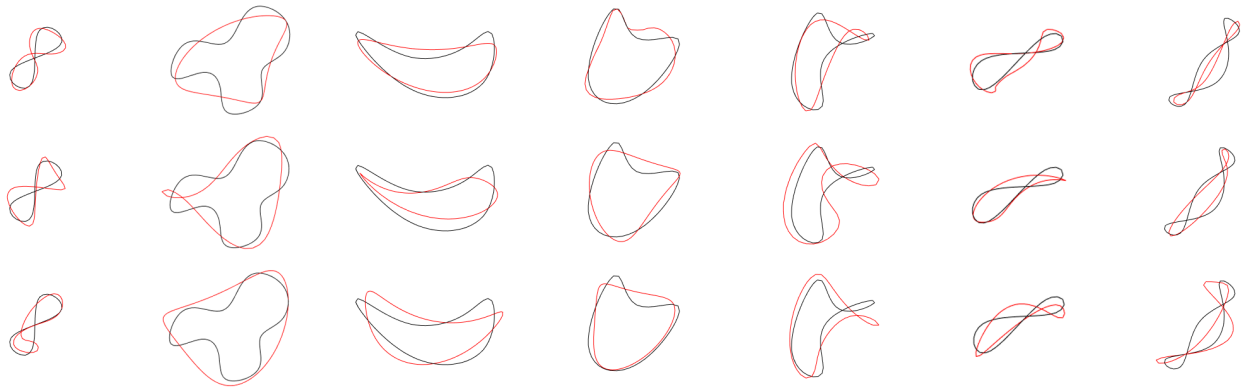


Figure 9: *Comparing the quality of curve matching for different metrics on a set of examples curves (black).* Top row*: feature-based metric with optimized coefficients,* middle row*: discrete Fréchet distance,* bottom row*: feature-based metric with all coefficients set to 1.0.*

for the same character. Our interface supports this style of motion design by providing the user with intuitive ways to select driving mechanisms that trace out desired motion curves and to adjust the velocity profile along these curves. The latter is essential for creating the characteristic motion of *EMA gallop*, which our framework implements using non-circular gears that directly control the timing of the foot-falls.

Even when the individual driving mechanisms are restricted to planar trajectories, users can create compelling 3D motions by combining components that operate in different planes. An example of this can be seen in the side-to-side tail and body sway of the *TRex* character shown in Fig. 11. Moreover, if the types of driving mechanisms available in the input library are capable of generating non-planar motions, they can also be used within our framework. However, to fully exploit the pipeline we present, two changes would be required: the curve distance metric (Sec. 5) should be extended to operate on 3D curves, and the automatic method we use to ensure that components do not collide with each other (Sec. 6.2) should be revised to handle arbitrary 3D motions. The rest of our pipeline does not need to be altered, and we used non-planar driving mechanisms to create the *Scorpio* character (Fig. 15). For this example we optimized the dimensions of the leg concurrently with the other parameters of the driving mechanism in order to obtain a desired, non-planar trajectory. This mechanism has 23 parameters in total, but optimizing it took only about 2 minutes. We instantiated the optimized leg six times and created a walking animation using physics-based simulation on the resulting mechanical assembly (see accompanying video).

Many of our characters such as *EMA*, *Pushing Man* and *TRex* have external gear boxes that are located below the characters. However, our framework also allows the user to design mechanical assemblies in which the gear boxes are internal to the character, as evidenced by the *Froggy* and *Clocky* (Fig. 14, right) examples. Integrating the gear box within the character allowed us to design free-roaming examples such as *Bernie* (Fig. 12) and *DrillR* (Fig. 1, right).

8 Limitations and Future work

We have presented an interactive design system that allows non-expert users to create animated mechanical characters. Our aim was to automate this design process as much as possible, while giving users sufficient artistic freedom. As evidenced by the wide range of characters that we designed, our method achieves this goal. Nevertheless, many exciting avenues for future work remain. First, the quality of the motions we generate directly depends on the types of

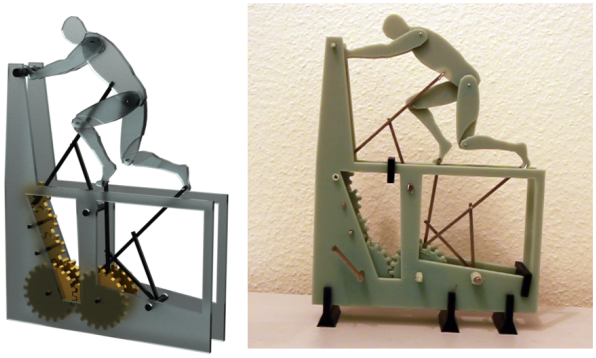


Figure 10: *PushingMan.*

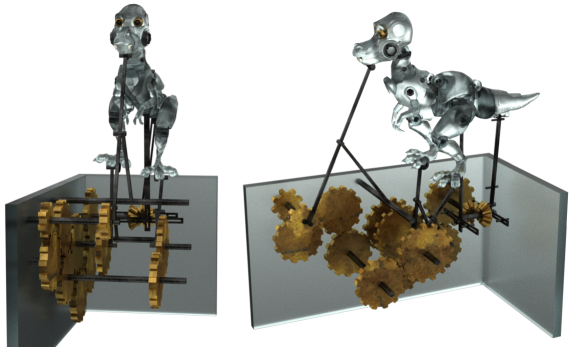


Figure 11: *TRex: By combining planar mechanisms that operate in different planes, we can create compelling 3D character motions.*

Figure 12: *Bernie. A DC motor and a battery pack were added to the body to make the character walk on its own (with strings attached for support).*

Figure 13: *EMA Walk and EMA Gallop. For the galloping motion we use non-circular gears to control the timing of the foot-falls.*

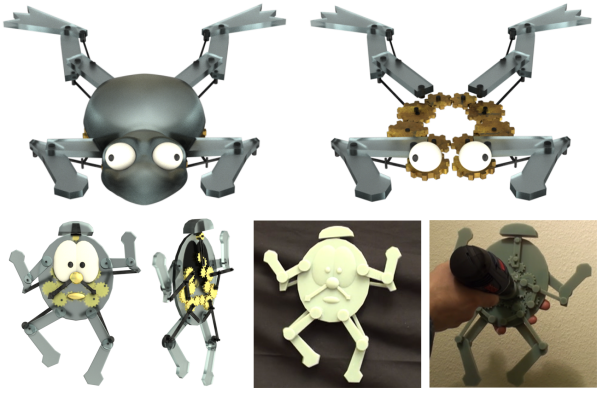


Figure 14: *Froggy and Clocky.*

Model	#Driving Mechanisms	#Components	#Connections
Pushing Man	3	24	37
EMA Walk	5	59	74
EMA Gallop	6	93	122
Bernie	4	40	60
CyberTooth	5	54	73
Drill-R	3	32	45
Clocky	5	32	48
Froggy	4	43	47
T-Rex	5	60	67
Scorpio	8	111	117

Table 1: *Statistics for the presented examples.*

Figure 15: *Scorpio character, showing 3D motion.* Left*: Leg and driving mechanism in isolation.* Right*: Simulated character walking.*

mechanisms that are available for use. Although we were able to generate a range of compelling examples, the types of mechanisms in our library ultimately limit the space of possible motions. In the future, we plan to investigate methods that automatically extract these building blocks from existing mechanical assemblies.

We plan to also incorporate structural analysis when designing mechanical characters in order to ensure that the resulting assemblies are lightweight, yet robust. To physically manufacture the characters that are shown in this paper, we used an empirical process to determine, for instance, the radius of the support shafts or the thickness of the linkages, such that they deformed as little as possible as the animation was played out. We believe that this process could be automated. In addition, for this work, we employed non-circular gears to explicitly control the timing of the animations. In the future we plan to also use them to reduce torque fluctuations when driving the assemblies [Yao and Yan 2003]. Relatedly, we would like to incorporate additional optimization objectives that minimize the net force required to drive the assemblies.

Our system detects and reports failures at various stages in the design pipeline, for example, if the assembly is over or under constrained, or if there are collisions between the components of a non-planar assembly as the animation is playing. The user is currently expected to manually edit the assembly in order to fix these problems. An interesting direction for future work is to address such problems automatically, or to provide helpful hints to guide the user in addressing them in an optimal manner.

One of the most basic limitations of our research is the restriction to cyclic motions, which leads to another interesting area for future work. By employing physical systems that can switch between multiple mechanism sets or different drivers, we believe we can generate motions that do not simply repeat every cycle. Finally, although our mechanical characters operate without environmental awareness, our research brings us one step closer to the rapid design

and manufacturing of customized robots that sense and interact with their environments in order to carry out complex tasks.

Acknowledgments

We thank the anonymous reviewers for their helpful comments and Maurizio Nitti for modeling many of the characters we used for this work.

References

ALT, H., AND GODAU, M. 1995. Computing the fréchet distance between two polygonal curves. *International Journal of Computational Geometry & Applications 5*, 01 & 02, 75–91.

BÄCHER, M., BICKEL, B., JAMES, D. L., AND PFISTER, H. 2012. Fabricating articulated characters from skinned meshes. In *Proc. of ACM SIGGRAPH '12*.

BERGOU, M., AUDOLY, B., VOUGA, E., WARDETZKY, M., AND GRINSPUN, E. 2010. Discrete viscous threads. In *Proc. of ACM SIGGRAPH '10*.

BICKEL, B., BÄCHER, M., OTADUY, M. A., LEE, H. R., PFISTER, H., GROSS, M., AND MATUSIK, W. 2010. Design and fabrication of materials with desired deformation behavior. In *Proc. of ACM SIGGRAPH '10*.

BRIDSON, R. 2007. Fast poisson disk sampling in arbitrary dimensions. In *Proc. of ACM SIGGRAPH '07*.

CABRERA, J., SIMON, A., AND PRADO, M. 2002. Optimal synthesis of mechanisms with genetic algorithms. *Mechanism and machine theory 37*, 10, 1165–1177.

CALÌ, J., CALIAN, D., AMATI, C., KLEINBERGER, R., STEED, A., KAUTZ, J., AND WEYRICH, T. 2012. 3D-printing of non-assembly, articulated models. In *Proc. of ACM SIGGRAPH Asia '12*.

CHIOU, S., AND SRIDHAR, K. 1999. Automated conceptual design of mechanisms. *Mechanism and Machine Theory 34*, 3, 467–495.

DEMARSIN, K., VANDERSTRAETEN, D., VOLODINE, T., AND ROOSE, D. 2007. Detection of closed sharp edges in point clouds using normal estimation and graph theory. *Computer-Aided Design 39*, 4, 276–283.

DONG, Y., WANG, J., PELLACINI, F., TONG, X., AND GUO, B. 2010. Fabricating spatially-varying subsurface scattering. In *Proc. of ACM SIGGRAPH '10*.

EITER, T., AND MANNILA, H. 1994. Computing discrete fréchet distance. Tech. Rep. CD-TR 94/64, Christian Doppler Labor für Expertensyteme, TU Wien.

EITZ, M., RICHTER, R., BOUBEKEUR, T., HILDEBRAND, K., AND ALEXA, M. 2012. Sketch-based shape retrieval. In *Proc. of ACM SIGGRAPH '12*.

FREUDENSTEIN, F. 1954. *Design of Four-link Mechanisms*. Ph. D. Thesis, Columbia University, USA.

GUI, J., AND MÄNTYLÄ, M. 1994. Functional understanding of assembly modelling. *Computer-Aided Design 26*, 6, 435–451.

HASAN, M., FUCHS, M., MATUSIK, W., PFISTER, H., AND RUSINKIEWICZ, S. 2010. Physical reproduction of materials with specified subsurface scattering. In *Proc. of ACM SIGGRAPH '10*.

JOHNSON, D., 2010. Sisyphus testing shoes. http://www.youtube.com/watch?v=Rh-4zSbmhFU (Accessed on April 8, 2013).

LAU, M., OHGAWARA, A., MITANI, J., AND IGARASHI, T. 2011. Converting 3D furniture models to fabricatable parts and connectors. In *Proc. of ACM SIGGRAPH '11*.

MALZBENDER, T., SAMADANI, R., SCHER, S., CRUME, A., DUNN, D., AND DAVIS, J. 2012. Printing reflectance functions. *ACM Trans. Graph. 31*, 3 (June), 20:1–20:11.

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., AND SHIEBER, S. 1997. Design galleries: A general approach to setting parameters for computer graphics and animation. In *Proc. of ACM SIGGRAPH '97*, 389–400.

MITRA, N. J., YANG, Y.-L., YAN, D.-M., LI, W., AND AGRAWALA, M. 2010. Illustrating how mechanical assemblies work. In *Proc. of ACM SIGGRAPH '10*.

MORI, Y., AND IGARASHI, T. 2007. Plushie: An interactive design system for plush toys. In *Proc. of ACM SIGGRAPH '07*.

NOCEDAL, J., AND WRIGHT, S. J. 2006. *Numerical Optimization*. Springer.

PEPPE, R. 2002. *Automata and Mechanical Toys*. Crowood Press.

SCLATER, N., AND CHIRONIS, N. 2001. *Mechanisms and mechanical devices sourcebook*. McGraw-Hill.

SELMAN, B., KAUTZ, H., COHEN, B., ET AL. 1993. Local search strategies for satisfiability testing. *Cliques, coloring, and satisfiability: Second DIMACS implementation challenge 26*, 521–532.

STAVA, O., VANEK, J., BENES, B., CARR, N., AND MĚCH, R. 2012. Stress relief: improving structural strength of 3d printable objects. In *Proc. of ACM SIGGRAPH '12*.

SUBRAMANIAN, D., AND WANG, C. 1995. Kinematic synthesis with configuration spaces. *Research in Engineering Design 7*, 3, 193–213.

UMETANI, N., IGARASHI, T., AND MITRA, N. J. 2012. Guided exploration of physically valid shapes for furniture design. In *Proc. of ACM SIGGRAPH '12*.

WESLEY, M., LOZANO-PEREZ, T., LIEBERMAN, L., LAVIN, M., AND GROSSMAN, D. 1980. A geometric modeling system for automated mechanical assembly. *IBM Journal of Research and Development 24*, 1, 64–74.

WEYRICH, T., PEERS, P., MATUSIK, W., AND RUSINKIEWICZ, S. 2009. Fabricating microgeometry for custom surface reflectance. In *Proc. of ACM SIGGRAPH '09*.

XING, E., NG, A., JORDAN, M., AND RUSSELL, S. 2002. Distance metric learning, with application to clustering with side-information. *Advances in neural information processing systems 15*, 505–512.

YAO, Y., AND YAN, H. 2003. A new method for torque balancing of planar linkages using non-circular gears. *Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science 217*, 5, 495–503.

ZHU, L., XU, W., SNYDER, J., LIU, Y., WANG, G., AND GUO, B. 2012. Motion-guided mechanical toy modeling. In *Proc. of ACM SIGGRAPH Asia '12*.