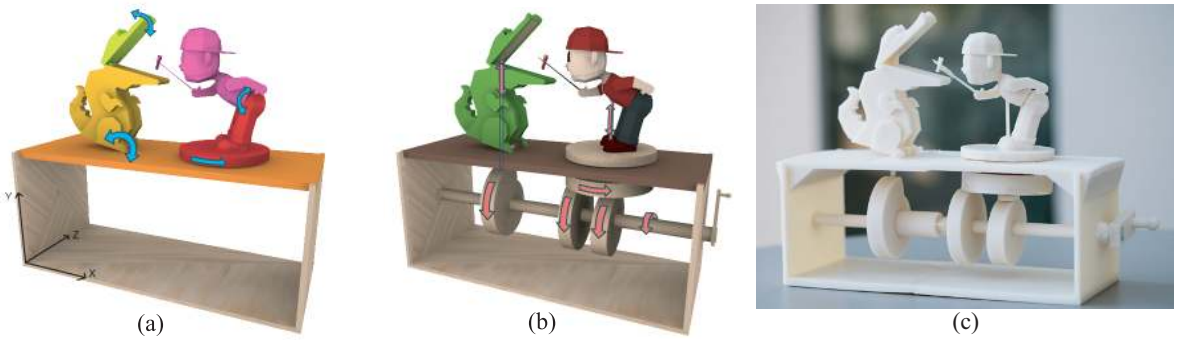
Motion-Guided Mechanical Toy Modeling

Lifeng Zhu* Weiwei Xu†‡ John Snyder § Yang Liu ‡ Guoping Wang* Baining Guo‡
Peking University* Hangzhou Normal University† Microsoft Research Asia‡ Microsoft Research§



(a) (b) (c)

Figure 1: *Example mechanical toy: Crocodile Feeding. (a) Input. The designer specifies the geometry and motion of the toy's features, in this case a boy and a crocodile object, forming two kinematic chains and four color-coded feature components. The feature base is colored orange. (b) Mechanical assembly synthesized by our system to generate the target motion. (c) Fabricated result. Overlayed arrows illustrate the motion, both input for features in (a) and output for the synthesized mechanism in (b), via the rules in [Mitra et al. 2010]. The canonical local coordinate system for the mechanical assembly is shown in (a). Please see the accompanying video for the full animation.*

Abstract

We introduce a new method to synthesize mechanical toys solely from the motion of their features. The designer specifies the geometry and a time-varying rotation and translation of each rigid feature component. Our algorithm automatically generates a mechanism assembly located in a box below the feature base that produces the specified motion. Parts in the assembly are selected from a parameterized set including belt-pulleys, gears, crank-sliders, quick-returns, and various cams (snail, ellipse, and double-ellipse). Positions and parameters for these parts are optimized to generate the specified motion, minimize a simple measure of complexity, and yield a well-distributed layout of parts over the driving axes. Our solution uses a special initialization procedure followed by simulated annealing to efficiently search the complex configuration space for an optimal assembly.

Keywords: forward and inverse kinematics, MCAD, mechanism synthesis, simulated annealing.

Links: DL PDF

*lfzhulf@gmail.com, wgp@pku.edu.cn, Key Lab of Machine Perception(MOE). This work was done when Lifeng Zhu was an intern at Microsoft Research Asia.

†weiwei.xu.g@gmail.com, corresponding author

‡YangLiu@microsoft.com, bainguo@microsoft.com

§johnsny@microsoft.com

ACM Reference Format
Zhu, L., Xu, W., Snyder, J., Liu, Y., Wang, G., Guo, B. 2012. Motion-Guided Mechanical Toy Modeling. *ACM Trans. Graph. 31* 6, Article 127 (November 2012), 10 pages. DOI = 10.1145/2366145.2366146 http://doi.acm.org/10.1145/2366145.2366146.

1 Introduction

Mechanical toy automata produce fascinating motion. Their history dates back to at least 400BC with the flying pigeon of Archytas of Tarentum. Modern versions range from large exhibits such as the Cabaret Mechanical Theater in the United Kingdom and the Swiss Jolly Ball at the Museum of Science and Industry in Chicago to playthings used by millions of children. Mechanisms in these automata drive the motion of the toy's whimsical features, and form an interesting attribute in their own right. Today, mechanical automata represent a popular topic as witnessed by the many books and DVDs covering their design and fabrication (e.g., [Neufeld 2003; Peppe 2005; Frost 2007]).

Mechanical toy design integrates a range of skills from art and craftsmanship to mechanics and mathematics. The designer typically begins by sketching the shapes and motions of the toy's features. He then plans a mechanical assembly to generate this motion. Inevitably, the design is iterated by refining the parameters of its different parts such as cams, gears, cranks, etc., to tune the motion and optimize other relevant factors such as manufacturing cost. The process is difficult and time-consuming, and demands an understanding of the complicated behavior of mechanisms, as well as a balancing of different design desiderata.

We automate design for mechanical toys such as the one in Figure 1. Our larger goal is to integrate kinematic simulation of mechanical assemblies into 3D modeling. Integrated simulation allows a motion-to-form mapping: conversion of user-specified motion into a physical, functioning mechanism. A toy designed in our system can be directly fabricated into a real object using a 3D printer, as shown in Figure 1c. We think custom mechanical design for ordinary users and for recreational and educational purposes will become increasingly important with the advent of inexpensive 3D printing. We also believe our approach can be generalized to other types of high-level mechanism design.

Our algorithm's input is the 3D geometry and motion of the toy's top-level features. These consist of rigid *feature components* connected by joints, to form one or more kinematically-linked chains. Its output is a *mechanical assembly* located below the base of features. The assembly consists of individual *parts*, such as cams,

quick-returns, and crank-sliders, connected to a manually-rotated *driving axis*. The free end or *handle* of each part connects to a specified feature component and drives it to generate the specified motion. Our system supports automatic splitting of the main driving axis into separate axes connected to the main axis by belts.

Current MCAD software such as Autodesk Open Inventor and Solidworks models mechanical assemblies through the powerful tool of geometric constraint maintenance. Editing performed on one mechanical part automatically propagates to others in the assembly. Our algorithm more fully automates design by seeking parameters based directly on the assembly's final function: how it moves the toy's feature components.

Automatic synthesis of a 3D mechanical assembly from its function is nontrivial, even for a mechanical toy. It must determine both the types and parameters of the parts as well as a non-colliding, physically-realizable layout. Though our system currently uses only a few basic part types, their combination in an assembly leads to an exponential growth of the possibilities. We handle this problem by exploiting knowledge about motion transmission in mechanisms. We first categorize input motions into a few standard types (e.g., circle, line, arc, and helix), each of which determines an initial subassembly. We then incrementally optimize this initial configuration using simulated annealing, exploiting kinematic relationships among shape parameters of a part to speed up convergence.

2 Related Work

Assembly Modeling/Mechanism Synthesis: Assembly modeling determines the position and orientation of parts according to constraints governing their mating or alignment. Symbolic, rule, and graph-based approaches construct the geometry constraint equations which are then solved via numerical optimization [Verroust et al. 1992; Gao and Chou 1998; Kondo 1992; Kim et al. 2000; Li et al. 2002; Peng et al. 2006]. Other research generates conceptual assemblies from a functional specification [Gui and Mntyl 1994; Roy et al. 2001; Chiou and Sridhar 1999]. Data structures such as the *bond graph* characterize standard parts in a database, which can be searched for a combination to fulfill a function [Hoover and Rinderle 1989; Finger and Rinderle 1989; Chiou and Sridhar 1999]. However, conceptual design is abstract and ignores physical properties such as the size and position of parts in a non-interpenetrating assembly.

Our algorithm generates a physically-realizable assembly from a description of the target motion. Unlike previous work, we use a higher-level abstraction that avoids specifying any information about the shape, parameters, or motion of parts in the assembly. Instead, we automatically select them from a small parameterized set of part types according to *a priori* knowledge of motion transmission in mechanisms, and refine the initial selection by optimizing over both discrete and continuous variables.

Stochastic Optimization: Markov chain Monte Carlo (MCMC) methods, such as Metropolis-Hastings, are widely used in computer graphics. Veach and Guibas [Veach and Guibas 1997] applied the Metropolis-Hastings algorithm to light transport. MCMC has been used to find plausible animations of passive rigid objects satisfying user constraints [Chenney and Forsyth 2000]. Recently, MCMC was applied to grammar-based procedural modeling [Talton et al. 2011], to search for a 3D model that conforms to a user's specification from the grammar's production space. MCMC-based optimization has been used in automatic residential building and furniture layout [Merrell et al. 2010; Merrell et al. 2011]. The related method of simulated annealing has also been used for automatic furniture layout [Yu et al. 2011].

We apply simulated annealing to the motion-to-form mechanical modeling problem, since it allows more flexible random walks through our constrained configuration space. We optimize over the shape space of the assembly's mechanical parts, considering its function (i.e., how well does it reproduce the desired motion?), layout (i.e., how evenly does it distribute mechanical parts along the driving axis?), and cost/complexity (i.e., how many parts and types of parts does it need?). We exploit relationships between part parameters to accelerate the search.

Motion Analysis: Given the geometry of a mechanical assembly, interactions and motions of mechanical parts can be inferred. Xu et al. [2009] performed slippage analysis over contact surfaces to categorize joints in man-made objects, and used them for subsequent animation. Mitra et al. [2010] analyzed interactions between mechanical parts from contact detection and relations between part axes. This data was then used to visualize how each part moves. Mechanical toy modeling can be viewed as the inverse problem, where geometry of mechanical parts is derived from user-specified motion.

Fabrication: Fabrication of physical objects has received significant research attention in CG. Objects with desired subsurface scattering and deformation properties can be fabricated using 3D printing [Hašan et al. 2010; Dong et al. 2010; Bickel et al. 2010]. Methods have also been developed to facilitate fabrication, by converting a 3D shape into planar slices [McCrae et al. 2011; Hildebrand et al. 2012], or optimizing it to improve its stability [Stava et al. 2012]. Another research trend is custom design and creation of objects by non-professional users. Xin et al. [2011] proposed fabrication of 3D burr puzzles. Mori et al. [2007] integrated simulation and modeling in the design of plush toys. Lau et al. [2011] converted an existing 3D furniture model into parts and connectors that can be fabricated and assembled by the user. Our work extends non-professional design and fabrication to a class of animated mechanisms.

3 Algorithm Overview

We synthesize mechanical toys that convert the rotary motion of driving axes to the periodic animation of feature components. An example is shown in Figure 1. The input to our algorithm consists of three pieces of information: the 3D geometry and motion of the toy's feature objects, the dimensions of an underlying box, called the *assembly box*, in which to install the mechanical assembly producing that motion, and the location on feature objects where they connect to synthesized mechanical parts.

We assume that the feature objects have been separated into rigid components, with specified joint constraints between them. The specification of geometry, motion, and joints are created with 3D modeling software, such as Maya or 3DS Max. The toy's features thus consist of multiple, kinematically-linked chains. The motion created by the designer is then recorded to yield a time-varying translation and rotation for each feature component, which is input to our assembly generation algorithm.

For the purpose of motion generation, the designer also needs to specify which feature components should be driven by (i.e., attached to) the free end point or *handle* of a part in the assembly. Two general connection types are supported: *rigid*, in which the feature is completely controlled by the part handle, and *translational*, in which the feature's translation is controlled while its orientation is left free. A translational connection can be realized by a ball joint at the handle. Our system also supports a special third type of *slotted* connection when a part that generates linear (reciprocating) motion is attached to a feature component which lacks sufficient degrees of freedom to be connected in either of the previous two

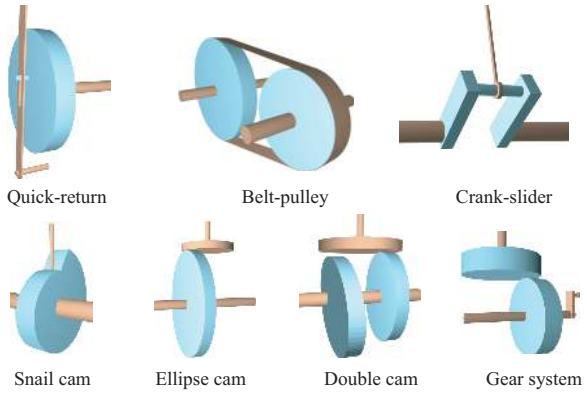


Figure 2: *Mechanical parts used in our system.*

ways (i.e., a single-component feature chain containing only one 2D hinge joint). The slot allows the handle to move freely and clear the feature geometry; see Figure 4b.

The initial position of the handle on each driven feature component is also specified as an input, but we allow it to move during optimization to provide extra flexibility in the layout.[1] This reflects the fact that we can easily add a rigid link to the end of a part after fabrication, to extend it and allow its attachment at a fairly arbitrary point on the feature component. The designer can constrain the space of possible handle locations within a specified constraint volume. If unspecified, we use the driven feature component's oriented bounding box.

With the above inputs, our system searches for a good configuration of an assembly to realize the desired motion. Our algorithm first guesses the type and shape of mechanical parts according to the input motion, and then optimizes the assembly using simulated annealing. The optimization objective integrates the approximation quality of the generated motion and the layout quality and complexity of the assembly.

At each step of assembly optimization, we simulate the mechanism over one animation cycle using forward kinematics. The motion of the assembly's handles then determines the motion of its feature components through inverse kinematics. We finally measure how similar the features' simulated motion is to the specified target motion. Use of inverse kinematics may not seem appropriate since it assumes systems in which each joint can be driven individually, such as a robotic arm, while joints in the feature chains of our toys are passive and driven only through the motion of part handles. But it works as long as the motion of features is uniquely determined by the motion of handles, with no extra degrees of freedom left over. This property is not hard to achieve in the toys we target: systems of 2D revolute or hinge joints with angular limits imposed by the constraint of non-colliding geometry (e.g., the foot of the crocodile in Figure 1 can not penetrate the floor).

4 Mechanical Toy Representation

A mechanical toy consists of two parts: its top features and an underlying assembly. Its kinematic simulation is also separated into two modules. The positions and orientations of the handles are driven by a forward kinematics simulation of the mechanism. These handles in turn drive the features, based on inverse kinematics.

The following introduces the mechanical part types we use, and then briefly describes how we perform their kinematic simulation.

[1]Handle position varies as the part moves and so is obviously time-dependent. "Initial" here refers to the optimization sequence, not animation time. Each initial handle position is specified at the beginning of the animation cycle.

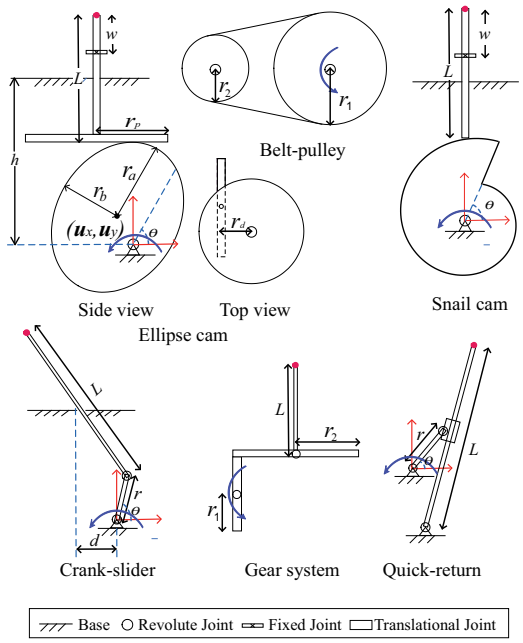


Figure 3: *Schematics for mechanical parts in Figure 2. Based on the shape parameters shown for each part type, the motion of the part's free end point (indicated by a red dot) is transmitted to a connected feature component.*

4.1 Representation and Motion of Mechanical Parts

Figure 2 illustrates the mechanical parts our system uses. Parts convert rotary motion on the driving axes into other types of motion, such as linear (moving backwards and forwards continuously in a straight line), oscillating (moving backwards and forwards along an arc), or helical (moving up and down along a helix). Motions generated by the various mechanical parts are listed in Table 1.

Figure 3 illustrates the *shape parameters* for each mechanical part type, which determine its motion in an assembly. Dark blue arrows in the figure indicate the direction of driving rotary motion and the position where the part connects to the driving axis. Each part's handle (free end point) is indicated by a red dot, and connects to the feature component it drives. The "belt-pulley" part does not directly connect to features but instead transmits motion between driving axes. Each part type's shape parameters, the constraints between them, and its kinematic behavior are fully described in supplementary material. The double cam part is omitted from the figure but is similar to a (single) ellipse cam and included in the supplement.

The lobe in the snail cam shown in Figure 3 produces a slow lift and sudden drop motion. Multiple lobes are also allowed in our system to generate a series of such motions in one rotational period (see e.g. the rightmost part in Figure 7). Details are included in supplementary material.

4.2 Forward Kinematic Simulation of Assembly

Kinematic simulation of a mechanical assembly is a well studied problem [Uicker 2010]. Our implementation is a simplified version, in which a hand-controlled driving axis originates the

Output Motion (plane/axis)	Part	Geometry Primitive
rotary (xz)	gear	circle (2D)
rotary (yz)	crank-slider,belt-pulley	circle (2D)
linear (y)	ellipse cam, snail cam double cam	line (2D)
linear (z)	quick-return	line (2D)
oscillating (xy)	ellipse cam, snail cam	arc (2D)
oscillating (yz)	ellipse cam, snail cam quick-return	arc (2D)
elliptical (yz)	crank-slider	ellipse (2D)
helical (y)	ellipse cam, double cam	helix (3D)

Table 1: *Converting input rotary motion to desired motion of a handle. Planes or axes are represented using the coordinate frame shown in Figure 1a. Note that the trajectory generated by the crank-slider is only approximately elliptical.*

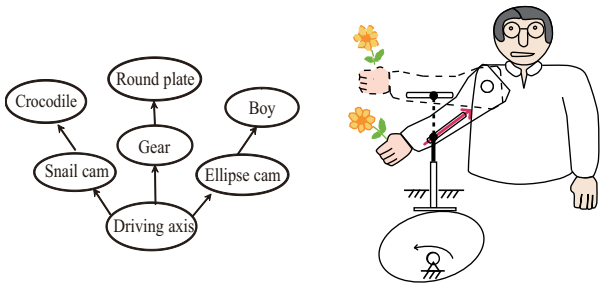


Figure 4: *(a) Graphical representation of crocodile feeding example. (b) Sliding slot connection for the ellipse cam part. The red arrow indicates the line constraint target for IK motion simulation.*

rest of the motion. First, a graph representing the assembly is built [Mitra et al. 2010]. Graph nodes represent mechanical parts. Parts are connected through a directed edge representing how motion is transmitted in the assembly. Each edge also encodes the coordinate frame of a child part relative to its parent. A node with no incident edges is called the root node, and represents the primary driving axis in our system. Figure 4a shows an example graph.

Kinematic simulation performs a breadth-first graph traversal which transmits the motion from the driving axis through each part to the handles. The detailed mathematics necessary to kinematically simulate each mechanical part type is contained in the supplementary material.

4.3 Inverse Kinematic Simulation of Features

Inverse kinematics (IK) determines how feature components move, given the position of handles computed from the forward simulation of the assembly in the previous subsection. Note that the motion of a single handle drives the motion of all linked components on each kinematic feature chain. The position (for a translational connection) or position and orientation (for a rigid connection) of the handle are first set as the target position/orientation of the driven feature component. IK then solves for the translation and rotation of every component on the kinematic chain needed to reach this target. The damped, pseudo-inverse Jacobian method is used to compute the search direction in each IK iteration [Wampler 1986].

When connecting a part that realizes linear (also called reciprocating) handle motion to a feature component containing just one hinge joint, a slot connection is necessary. Figure 4b illustrates the situation for the ellipse cam. In this case, the cam's free end point moves on a straight line, but its trajectory in the local coordinate system of the "pinned" feature component changes distance with respect to the fixed hinge, making a slot necessary. For a slotted connection, the IK algorithm's target constrains the free end point to lie on a line representing the sliding slot. Further details are included in supplementary material. Our system supports slotted connections for the ellipse cam and quick-return part types.

5 Assembly Generation

We synthesize an assembly by searching within the mechanism's configuration space for an optimal result that produces the specified motion. This configuration space includes discrete variables, representing the type of each mechanical part, and continuous variables, representing the parameters of each mechanical part. The relation between this configuration space and the final motion of the toy's features is complex and not expressible in closed form. The following describes how we solve this optimization problem.

Notation: We denote the translation and orientation, respectively, of feature components as $\mathbf{t}_{ij}$ and $\mathbf{q}_{ij}$, where i is the component index and j ranges over the N frames of an animation cycle. Quaternions represent orientation. These unhatted vectors represent each component's rigid motion as driven by the current assembly. Hatted versions, $\hat{\mathbf{t}}_{ij}$ and $\hat{\mathbf{q}}_{ij}$, represent their corresponding target rigid motions, as specified by the designer.

Recall that handle position is updated in the optimization. We use the notation $\mathbf{p}_i^*$ for the initial handle position specified by the user, and $\mathbf{p}_i$ for the actual handle position in the current optimization iteration, where i is the handle index. Handles also move over an animation cycle. We use doubly-indexed notation to refer to handle trajectories, e.g., $\mathbf{p}_{ij}$, where the second index j represents animation time. A singly-indexed vector refers to the handle position at the first frame of the animation cycle. As with the rigid motions, these vectors represent motion driven by the current assembly. Placing a hat on them denotes their (desired or target) trajectories when driven by the feature component to which they are attached, using the input motion for that component as specified by the designer rather than its motion as generated by the assembly.

5.1 Simulated Annealing

Denote the configuration space of the assembly by $\mathbb{X}$, and a state in this space $x \in \mathbb{X}$. As in furniture layout optimization [Yu et al. 2011], we adopt simulated annealing to search for a solution state. The search minimizes a Boltzmann-like objective

$$f(x) = \exp\left(-\frac{C(x)}{T}\right) \tag{1}$$

where $C(x)$ is a positive definite cost function. A new state x' is proposed at each iteration and accepted with probability

$$\alpha(x'|x) = \min\left(1, \frac{f(x')}{f(x)}\right). \tag{2}$$

The annealing parameter T is initialized to be 10, and its value is decreased by a factor of 0.9 every 300 iterations. The next two subsections describe our initialization procedure that determines a starting point, our cost function, and the parameter settings used in optimization. The same parameter settings were used to synthesize all toys in the paper.

5.2 Initialization

A naively-chosen starting state x is likely to be far from optimal, and leads to slow convergence or overall failure of the optimization. We

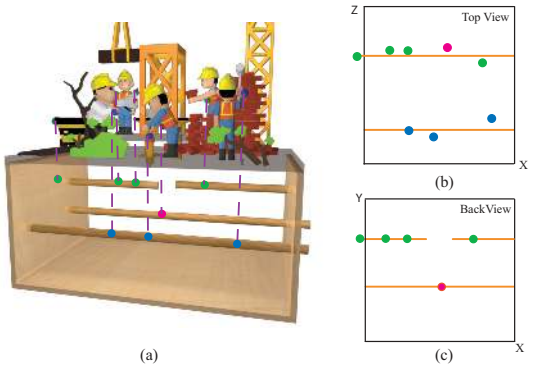


Figure 5: *Driving axis initialization. A front view of the toy is shown in (a). Dots indicate the projected centers of initial handle trajectories; color indicates their period (green = whole animation cycle, red is 1/2, blue is 1/4). Two initial driving axes, corresponding to clusters in z-coordinate of the handle trajectory centers, are shown in (b). Differences in motion period cause the back axis in (b) to be split into three, as shown in (c).*

propose an initialization procedure based on mechanism theory to choose a good initial state. The following details the two stages in this initialization: for driving axes and for parts.

Driving Axes Initialization: We initialize the number and position of driving axes in the assembly, based on the distribution of initial handles and their animation periods. The following steps are used:

1. Compute the centers $\mathbf{m}_i = 1/N_H \sum_{j=1}^{N_H} \hat{\mathbf{p}}_{ij}^*$ of the trajectories of each of the N_H initial handles. Cluster these by their z coordinate using the mean shift method in [Comaniciu et al. 2002]. The radius parameter in the mean shift is set to 4% of the diagonal length of the assembly box.

2. Initialize a driving axis for each cluster. Its z coordinate is just the mean z coordinate of $\mathbf{m}_i$ over cluster members; its y coordinate is set at 1/2 the height of the assembly box.

3. Perform period analysis of the trajectory of initial handles in each cluster. If there are several different periods in one cluster, split the initial driving axis. The driving axis with the lowest period remains the initial axis; the others are moved above it, in the y direction. The axis length is set to the maximal interval in x coordinate of consecutive centers with the same period. At least one end of the axis must extend all the way to an assembly box wall. We extend the end closest to a wall or extend both ways if both ends are close.

Our system randomly picks one driving axis as the main driving axis which is manually cranked by the user. It must be an axis that extends completely through the assembly box. For partial axes, as in the Construction Site example shown in Figure 5, auxiliary walls are added when fabricating the toy to support any "hanging" ends.

We use the fast Fourier transform (FFT) to extract period values. Specifically, we sample the trajectory $\{\hat{\mathbf{p}}_{ij}^*, j = 1 \ldots N\}$ of an initial handle i. The FFT is then applied to each of the three (x,y,z) coordinates of this sequence, and the period derived from the frequency of maximum modulus over all three coordinates. Figure 5 illustrates the result for an example.

If the initial driving axis is split, our algorithm automatically creates belt-pulleys to connect the initial and split axes and to realize the required conversion of rotational speed. A gear system is also an option; our system currently supports only belts.

Parts Initialization: Table 1 shows that categorizing an initial handle's motion type roughly indicates how to choose an associated mechanical part type. This idea motivates the following two-step procedure. First, we apply a principle component analysis to the trajectory of each handle to categorize it as 2D (planar) or 3D (space curve). We then choose the best-fitting geometric primitive to approximate this trajectory. A trajectory classified as 2D is fit with either a circle, line, arc, or ellipse, while a 3D trajectory is restricted to a helix. For rotary or helical motion, the trajectory evaluated exactly at the rotation center is fixed or on a straight line. So placing the handle there misclassifies its desired motion in initialization. We instead classify based on the trajectory of a slightly perturbed version of the input handle. A candidate part that can generate the appropriate motion category is selected from the list in the table; the one whose initialized parameters best approximate the input handle trajectory is selected.

Our current implementation only supports helical 3D motion, which can be realized with an ellipse cam or double cam where r_d is not 0.

Parameters for each mechanical part chosen must also be initialized. The required procedure for each part type is described in the supplementary material. The initialized parameters are then further refined to minimize error between the handle's actual and specified trajectories, via the objective

$$E_i = \sum_{j=1}^{N} ||\mathbf{p}_{ij} - \hat{\mathbf{p}}_{ij}^*||^2 \tag{3}$$

Constraints that keep the parts functioning properly must be maintained and are also documented in the supplementary material.

Motion Decomposition: An individual part in Table 1 generates a relatively simple motion, but more complex motion can be realized by connecting multiple parts to the same feature chain. Our system supports this by decomposing the complex motion into components, each controlled by a separate part. To work correctly, this *motion decomposition* assumes that the planes or axes of the two motion components are orthogonal. Suitable motions include translation along and rotation around the same axial line (motion of upper body of the boy in Figure 1), or translation along two orthogonal axes (motion of drawing character in Figure 9).

Motion decomposition is invoked if the fitting error from Eq. 3 exceeds a threshold, $\sqrt{E_i} > N\,\delta_f$, on a part that drives a feature chain containing multiple components. δ_f is set at 5% of the diagonal length of the assembly box. The procedure tries using two parts to separately drive the motions of the end component and its parent component. Specifically, we initialize a part connected to the original feature component by representing its motion in the local frame of its parent component. The part for the parent component is initialized as usual. In the crocodile feeding example, the boy's arm is decomposed into rotary motion of the plate (parent component) and up/down motion of the arm. The two parts are then initialized independently. The designer can also explicitly indicate the parent and child components to be driven. Collisions between the two parts initialized for motion decomposition are ignored in both the initialization and optimization.

5.3 Cost Function

The cost function characterizes the mechanical assembly in terms of its generated motion fidelity and aspects of its layout and complexity. It is defined as:

$$C(x) = C_a(x) + C_l(x) + C_t(x)$$
$$\text{Subject to : no mechanical part collision.} \tag{4}$$

The term $C_a(x)$ measures motion similarity, $C_l(x)$ measures layout quality, and $C_t(x)$ measures topological complexity. These terms are detailed below.

Non-collision is a hard constraint. Our current implementation uses the OBB-tree algorithm [Gottschalk et al. 1996] to detect collisions between different parts. Collisions between intermeshed components on a single part, such as the cam and follower of an ellipse cam, are ignored; these are handled by imposing constraints on the shape parameters. We also ignore collisions between feature objects and instead rely on the motion similarity metric, since motion that causes interpenetration will not match the non-colliding motion specified by the designer. Finally, we maintain IK feasibility as a hard constraint. During optimization, any change to the mechanism yielding handle positions that violate the kinematic constraints in any feature chain is rejected.

If any pair of parts collide immediately after initialization, we invoke a special procedure to avoid getting stuck in a colliding state. It replaces the collision hard constraint with an additional objective term, $C_c(x) = \exp(\alpha \sum_{i=1}^{M} D_i)$, where M is the number of collisions, D_i is the depth of collision i, and α is a large value equal to 10^6 in our current implementation. This penalty allows the algorithm to gradually reach a non-colliding state. After removing all collisions with this procedure, we revert to the normal, three-term objective and enforce non-collision in subsequent states by rejecting any state that yields a collision. This can greatly improve efficiency in optimization, because recovery from a colliding state with soft constraint is expensive and slows convergence.

Motion Similarity Objective: Given the current state of the assembly, we measure the difference between the actual and specified motions for each initial handle and for each feature component. Denote the number of frames in one cycle of the animation as N, the number of handles as N_H, and number of feature components as N_C. Motion similarity is defined as

$$C_a(x) = w_p \sum_{i=1}^{N_H} \sum_{j=1}^{N} \|\mathbf{p}^*_{ij} - \hat{\mathbf{p}}^*_{ij}\|^2 + \sum_{i=1}^{N_C} \sum_{j=1}^{N} \|\mathbf{t}_{ij} - \hat{\mathbf{t}}_{ij}\|^2 + \sum_{i=1}^{N_C} \sum_{j=1}^{N} \left\| \log(\mathbf{q}_{ij}^{-1}\, \hat{\mathbf{q}}_{ij}) \right\|^2 \quad (5)$$

where i indexes over handles or components, and j indexes over animation time. The weight w_p is set to 40 in our experiments.

The target trajectory of the initial handle, $\hat{\mathbf{p}}^*$, is determined by the specified motion of the feature component to which the handle is attached. Using the initial rather than current handle avoids discontinuous change to the cost function as the optimization proceeds.

Layout Objective: We optimize the assembly's layout through two adjustments: a) movement of the driving axis within the assembly box, and b) movement of parts along the driving axis. The main goal of this term is to avoid part collisions, to guarantee that the toy can be fabricated and functions properly. Another goal is to evenly distribute the parts to make their behavior easily visible.

Denoting as n the number of driving axes, and m_i the number of parts on the i-th driving axis, the layout term is defined as

$$C_l(x) = w_1 \sum_{i=1}^{n} \|\mathbf{c}_i - \mathbf{c}^*_i\|^2 + w_2 \sum_{i=1}^{n} \sum_{j=1}^{m_i} (l^i_{j+1} + l^i_{j-1} - 2l^i_j)^2 + w_3 \sum_{i=1}^{n} \sum_{j=1}^{m_i} s(|b^{i,r}_{j-1} - b^{i,l}_j|/\delta) \exp\left(\frac{\delta}{\|b^{i,r}_{j-1} - b^{i,l}_j\|} \right) + w_4 \sum_{i}^{N_H} \|\mathbf{p}_i - \mathbf{p}^*_i\|^2 \quad (6)$$

Note that the total number of parts is larger than or equal to the total number of handles. Belt-pulleys are parts but are not connected to handles.

The first term penalizes movement of the driving axis i away from its initial position $\mathbf{c}^*_i$ to a new position $\mathbf{c}_i$. To evaluate the next two terms, parts are first ordered by their position along their respective driving axis, by x coordinate. The second term characterizes uniformity of the distribution of parts based on their bounding boxes, where l_i is the x coordinate of the center of part i's axis-aligned bounding box. For indices j beyond the first or last part in the list, we use the wall position instead. The third term penalizes proximity of adjacent pairs of bounding boxes, and forces their separation to be greater than a specified threshold δ. b^r_{i-1} denotes the x position of the right side of bounding box $i-1$, while b^l_i denotes the x position of the left side of i. The term assumes we maintain non-overlapping bounding boxes, i.e., $b^r_{i-1} < b^l_i$. The function s is defined as a sigmoid function $s(d) = 1/(1+e^{8d})$. The final term penalizes movement of the handle away from its initially specified location. Weights w_1, w_2, w_3 and w_4 are set to 1.0, 5.0, 10.0 and 5.0, respectively. The threshold δ in the third term is set to 1/100 of the diagonal length of the assembly box.

Topology Objective: We prefer mechanisms which reduce the number of parts, N_P, and the number of different part types, N_T, via the objective

$$C_t(x) = w_T N_T + w_P N_P. \quad (7)$$

The weights w_T and w_N are set to 10 in our implementation.

5.4 Proposal Moves

Proposal moves adjust the current assembly to efficiently explore its configuration space. They comprise shape parameter moves, layout moves, and topology moves. The probability density function used to select moves is defined hierarchically. We first pick the overall move type based on a discrete probability distribution: 0.7 for a shape parameter move, 0.2 for a layout move, and 0.1 for a topology move. Within the selected move type, we then choose an appropriate element using a uniform distribution. For example, for a shape parameter move we randomly choose a single part parameter to adjust. The operation of each move type is described below.

Shape Parameter Move: This move locally adjusts the shape parameters of each mechanical part. We randomly choose a mechanical part and one of its parameters and replace it with a new value from a Gaussian distribution $[\mathcal{N}(\mathbf{s}, \delta_s^2)]$. The distribution variance, δ_s, varies depending on the type of the shape parameter. We set it to 15° for angle parameters and 1/20 of the diagonal length of the assembly box for length parameters.

Layout Move: This move adjusts the position of an entire driving axis or the position of a single mechanical part along it. The two operations are randomly chosen with the same probability. A uniform distribution also governs which mechanical part to move. A layout move that leads to a collision is rejected.

The first type of layout move randomly slides one driving axis into a new position, $\mathbf{c}$. The new value is chosen from the Gaussian distribution $[\mathcal{N}(\mathbf{c}, \delta_c^2)]$. Positions are 2D projections (in yz) onto the assembly box wall. Before evaluating the objective function, all parts connected to the driving axis must be translated accordingly. This can be done given the graph representation of the assembly.

The second type of layout move translates one selected mechanical part along its driving axis. We use a 1D parameter to represent the position of each mechanical part along its axis. This move also translates the free end of the mechanical part which attaches to a feature component. If it moves outside the feature's handle constraint volume, we reject the layout move.

Topology Move: We allow two types of topology moves tailored to the mechanical toy: type change and split/merge. The two types are randomly selected with equal probability. A type change replaces one type of mechanical part with a different, compatible type. Compatibility is listed in Table 1. The split or merge operation applies to two nearby gears with opposite rotational direction. Figure 7cd shows an example, in which two gears in the middle of the figure are merged. In general, two gear pairs (four gears total) can be merged into a single gear that drives two others (three gears total), or the reverse split operation performed on systems of three gears. The topology term prefers a merge, which reduces the number of mechanical parts, but a split can be accepted if it reduces the motion approximation error enough.

One problem with the above method is that a type change move is almost always rejected, because its motion is based on roughly initialized shape parameters. We thus adopt the delayed rejection technique to improve the move's acceptance probability [Tierney and Mira 1999; Talton et al. 2011]. Specifically, we try 150 iterations of shape parameter moves after changing a part type but before deciding whether to accept the type change.

5.5 Convergence Acceleration

Two methods accelerate convergence of the optimization algorithm. Figure 6 quantifies their benefit on the example toy in Figure 7.

Constrained Sampling: We observe that each mechanical part is relatively independent in approximating the input motion, since it controls only one particular component on the feature object. This motivates the following sub-optimization procedure, which is invoked after a shape parameter move. We fix the single parameter adjusted in this move, and then optimize the rest of the parameters pertaining to the same part, in order to better match the specified motion at the current handle position. This is identical to minimizing Equation 3 in part initialization, except that the actual handle, $\hat{\mathbf{p}}_{ij}$, replaces the initial one $\hat{\mathbf{p}}_{ij}^*$, to account for subsequent movement of the handle's attachment point on the feature component during optimization. As before, $\mathbf{p}_{ij}$ is the motion of the free end point of the part simulated from its current shape parameters. This strategy considers the global relations between shape parameters of a single mechanical part, and greatly speeds up convergence.

For gear and belt systems, a simple rule that preserves radius ratio is enforced when adjusting their radius parameters. For a slotted connection, the part's handle floats over the driven component; we ignore the constrained sampling step for such parts.

Covariance Matrix Adaptation: Covariance matrix adaptation (CMA) iteratively adapts a Gaussian distribution over parameter vectors. It has been proven effective in speeding up stochastic optimization algorithms [Wampler and Popović 2009; Wang et al. 2009]. However, its direct application to our problem is complicated by topology moves, which change the dimensionality of the configuration space. We thus limit CMA to the set of shape parameters for an individual mechanical part. In our implementation, each parameter's Gaussian distribution is re-estimated after 300 iterations. The top 30 percent samples ranked by cost function value determine the updated distribution.

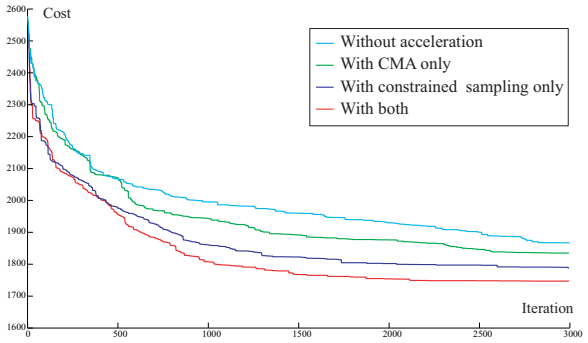


Figure 6: *Optimization acceleration.*

6 Experimental Results

Our system runs on a 2.99GHz Intel Quad Core PC with 4GB of RAM. Automatically generated results are shown in Figures 1, 7, 9, and 11. Table 2 documents statistics for these examples. Please see the accompanying video for the animations.

Assembly Optimization: We tested how various objective terms affect assembly generation on the "New Year" toy example. The initial assembly is shown in Figure 7b. It produces a rough approximation to the motion but does not prohibit collisions between feature objects. Collisions between parts in the assembly also occur because we initialize each part independently based only on its own target handle trajectory. Optimizing the shape parameters of all parts with the motion similarity objective alone removes the collision between feature objects, as shown in Figure 7b. Adding the collision-free constraint and the layout objective removes collisions between parts and more uniformly distributes them along the driving axes. Finally, adding the topology objective term merges two close-by gears to reduce the total number of parts in the assembly.

More results from the optimization algorithm are shown in Figure 11. The Robot DJ contains an example of a helical motion target on the stand with two notes, realized by an ellipse cam. The Construction Site example targets motions with different periods. Workers in the first line share one common period; the trolley and two workers in the second line share a second one, while the rising pad in the scaffold exhibits a third. Our algorithm automatically clusters features sharing a common period and generates child axes to drive them, using belt-pulleys to convert the rotational speed of the main user-cranked axis to the one needed at each child axis.

Motion Decomposition: Two examples use motion decomposition: Crocodile Feeding in Figure 1 and Lazy Drawer in Figure 9. In the crocodile feeding toy, the overall motion of the feature component representing the boy's upper body is decomposed into the rotary motion of its (parent) rotating base plate and the oscillating motion of the (child) upper body part itself. This decomposition removes the parent's rotary motion and so simplifies the oscillatory motion that must be matched in the child, allowing the initialization procedure to find a good part type (ellipse cam) to drive it.

The target motion in the Lazy Drawer toy in Figure 9 can not be approximated by any individual part type we support. But it can be decomposed into two linear motions: horizontal motion (in $\mathbf{Z}$)

(a) Initial assembly (b) Motion optimization (c) Motion+layout (d) Motion+layout+topology

Figure 7: *Effect of the cost function in the New Year example. (a) Initialized assembly. Collisions between feature objects and parts occur, indicated within red rectangles. (b) Optimizing the motion similarity objective by itself removes the collision between feature objects. (c) Adding the layout objective removes collisions between parts. (d) Adding the topology objective merges two nearby gears.*

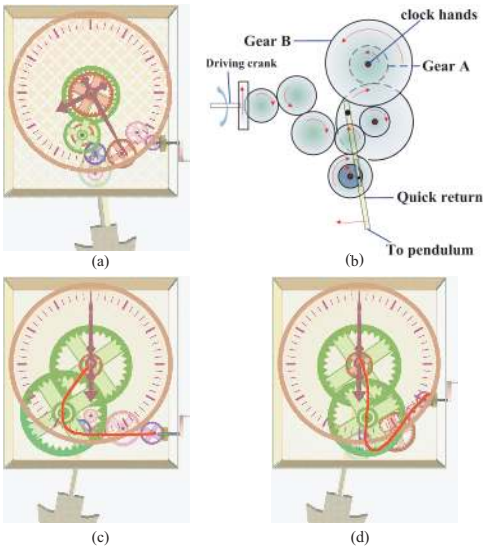


Figure 8: *Applying our algorithm to long kinematic chains. (a) Clock toy with an initial gear chain. The minute hand does not point to 12 when the hour hand points to 8. (b) Schematic view of the gear chain. (c,d) Optimization results with two different user-specified strokes guiding the gear layout.*

Robot DJ Construction Site

Figure 11: *Robot DJ and Construction Site mechanical toys.*

of the overall scaffold, and vertical motion (in **Y**) of the character's seat inside the scaffold. Note that the part driving the horizontal scaffold motion is connected behind the wall on which the drawing is traced. Our algorithm automatically generates a quick-return (for the horizontal motion) and various cam types (for the vertical motion) to generate the decomposed target motion.

Long Chains: To demonstrate the ability of our approach to handle longer kinematic chains, we used it to design a mechanical toy clock containing a 9-gear chain. This chain transmits rotational motion from the driving crank to its hour and minute hands (Figure 8). The number of gears is specified manually. Gears A and B in Figure 8 are placed so that their centers correspond to the rotation center of the clock hands. The other gears can be initialized arbitrarily; the system asks the user to specify their contact relationship.

However, a roughly initialized chain does not function properly. The hour and minute hands fail to rotate synchronously as shown in Figure 8a. Optimization to synchronize the hour and minute hands takes 10 seconds in our current implementation. Optimization randomly choose one gear, samples a new radius or position for it, and then performs constrained sampling to maintain contact between gears. The error in ratio between hand periods after optimization (compared to perfect synchronization) is below 10^{-8}. We let the user draw a stroke to guide gear layout. The distance between the center of the gear and the stroke is introduced as an additional energy term. Figure 8c and d show the results of stroke-based editing.

Motion Editing: Figure 9 also shows how changing the target curve the drawer traces out affects the assembly generated. Blue curves in the figure show the actual trajectory as generated by the assembly. Note our algorithm's selection of different cam types to realize the desired horizontal motion.

Our system's ability to automatically guess part types and their parameters makes it useful as an interactive, high-level mechanical design tool. We allow editing of the period, range, and type of motion for each feature object component. Figure 10 illustrates such editing in the robot DJ example. The compound gear in the middle appears when the period of the two bottles is edited, since the note object and bottle objects on the same driving axis no longer share a common period. We also substitute a disco ball with rotary motion for a note object with helical motion. Please see the supplementary video for the editing operations.

Validation: To validate the created mechanical toy, we fabricated the Crocodile Feeding example using a UPrint Plus 3D printer, shown in Figure 1c. The accompanying video shows that the actual motion generated by the fabricated toy is very similar to its simulated motion.

Limitations: Our system supports only the parts and motion types listed in Table 1. Success of the optimization relies on a reasonable feature layout. The algorithm may fail if the handles on different feature objects are placed too close together. In this case the user should rearrange features or enlarge their handle boxes to give the algorithm more freedom. Input motion should also be specified with some care. Feature objects should be largely non-colliding, and should move smoothly so that the time sampling is adequate. In our experience, this is not difficult for the motion types we demonstrate. The user simply reviews the animation and adjusts the size or position of features to avoid deeply interpenetrating collisions.

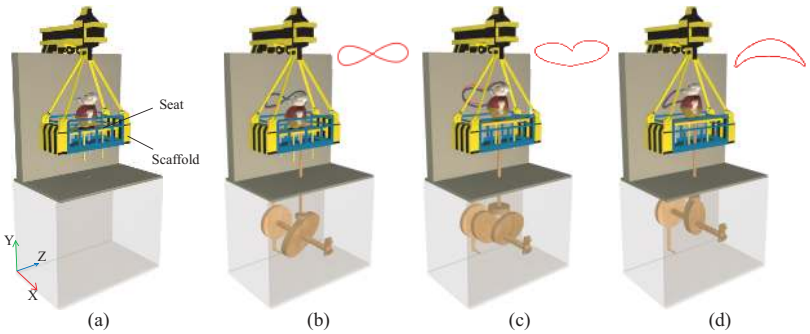


Figure 9: *Effect of motion editing on the generated assembly. (a) Input. (b,c,d) Assemblies generated for different target trajectories (red curves). In each image, the blue curve on the wall shows the trajectory actually traced out by the character in the scaffold, as driven by the assembly, to match this target.*

Figure 10: *Editing on the Robot DJ toy. Part's drawn in brown were updated according to the user's edit. Please see the accompanying video for the entire editing session.*

Name	Figure	# Feature Chains	# Feature Components	# Frames	Parts in Final Assembly	Time (seconds)
Robot DJ	Fig. 11	5	7	50	4 ellipse cams, 1 gear system, 1 belt-pulley	52s
Crocodile Feeding	Fig. 1	2	4	50	1 ellipse cam, 1 snail cam, 1 gear system	36s
Construction Site	Fig. 11	7	11	100	5 ellipse cams, 1 double cam, 1 crank-slider 1 quick-return, 3 belt-pulleys	135s
New Year	Fig. 7	12	26	50	1 ellipse cams, 2 snail cams, 2 gear systems 2 crank-sliders, 2 belt-pulleys	96s
Lazy Drawer 1	Fig. 9a	2	3	100	1 ellipse cam, 1 quick-return	62s
Lazy Drawer 2	Fig. 9b	2	3	50	1 double cam, 1 quick-return	37s
Lazy Drawer 3	Fig. 9c	2	3	50	1 ellipse cam, 1 quick-return	33s

Table 2: *Statistics for examples. Total time required to synthesize each toy example is reported in seconds in the rightmost column. 3000 iterations of the optimization algorithm were used in each example.*

7 Conclusion and Future Work

We pose a new motion-to-form problem in mechanical design and propose an algorithm to solve it that mimics the iteration of a human designer. It initializes the assembly using knowledge about each part's motion transformation and then refines it using simulated annealing, automatically adjusting each part's shape parameters to generate the specified motion of the mechanism's features.

Our specific goal is to allow people unfamiliar with mechanisms to design and fabricate their own mechanical toys. Our long-term aim is to integrate mechanism simulation into CG modeling to achieve a high-level (e.g. motion-to-form) specification and create a physical object. Generating mechanisms automatically from a specification of their function has occupied the attention of mathematicians and engineers from Archytas and Archimedes through Euler to the modern researchers we cite. Though we focus on toy design to demonstrate this contribution in a nontrivial application, our method can be generalized to other types of mechanisms by modifying the assembly's kinematic parameterization model. Our work advances the state-of-the-art in mechanism design which is currently difficult even for experts, and is based on low-level manipulation of parts governed by user-specified geometric constraints. We believe our approach might be useful in other applications, such as in shape reconstruction and reverse engineering of mechanical objects.

In future work, we plan to investigate new types of mechanical parts and new layout objectives. Our approach's division of the toy into driven features above and driving mechanism below a plane is also limited and could be improved to more freely combine mechanism and features within the same 3D space. Our system supports only a naive notion of fabrication cost in the topology term of its objective function. It also supports only kinematic simulation. Many mechanical toys require dynamics. For example, string often connects mechanical parts to feature objects; when the string is relaxed, object motions are not kinematically determined but behave in response to gravity and inertia. Dynamic behavior can also increase motion realism even in kinematically-linked toys. Finally, our system could be augmented with suggestions to help the user design motions that are approximated well by its set of mechanical parts.

Acknowledgements Guoping Wang is partially supported by the 973 program of China (No. 2009CB320801) and NSFC (No. 60925007, 60833007 and 61121002), and Weiwei Xu is partially supported by NSFC (No. 61272392). We would like to thank Yeying Chen for video production, and the SIGGRAPH reviewers for their constructive comments.

References

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. *ACM Trans. Graph. 29*, 4 (July), 63:1–63:10.

CHENNEY, S., AND FORSYTH, D. A. 2000. Sampling plausible solutions to multi-body constraint problems. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, SIGGRAPH '00, 219–228.

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

COMANICIU, D., MEER, P., AND MEMBER, S. 2002. Mean shift: A robust approach toward feature space analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence 24*, 603–619.

DONG, Y., WANG, J., PELLACINI, F., TONG, X., AND GUO, B. 2010. Fabricating spatially-varying subsurface scattering. *ACM Trans. Graph. 29*, 4 (July), 62:1–62:10.

FINGER, S., AND RINDERLE, J. 1989. *A Transformational Approach to Mechanical Design using a Bond Graph Grammar*, vol. 17. ASME, 107–116.

FROST, R. 2007. *Making Mad Toys & Mechanical Marvels in Wood*. Sterling.

GAO, X., AND CHOU, S. 1998. Solving geometric constraint systems. ii. a symbolic approach and decision of rc-constructibility. *Computer-Aided Design 30*, 115–122.

GOTTSCHALK, S., LIN, M. C., AND MANOCHA, D. 1996. Obbtree: a hierarchical structure for rapid interference detection. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, ACM, SIGGRAPH '96, 171–180.

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

HAŠAN, M., FUCHS, M., MATUSIK, W., PFISTER, H., AND RUSINKIEWICZ, S. 2010. Physical reproduction of materials with specified subsurface scattering. *ACM Trans. Graph. 29* (July), 61:1–61:10.

HILDEBRAND, K., BICKEL, B., AND ALEXA, M. 2012. crdbrd: Shape fabrication by sliding planar slices. In *to appear: Computer Graphics Forum (Eurographics 2012)*, vol. 31.

HOOVER, S. P., AND RINDERLE, J. R. 1989. A synthesis strategy for mechanical devices. *Research in Engineering Design 1*, 87–103.

KIM, J., KIM, K., CHOI, K., AND LEE, J. 2000. Solving 3d geometric constraints for assembly modelling. *The International Journal of Advanced Manufacturing Technology 16*, 843–849. 10.1007/s001700070019.

KONDO, K. 1992. Algebraic method for manipulation of dimensional relationships in geometric models. *Computer-Aided Design 24*, 3, 141 – 147.

LAU, M., OHGAWARA, A., MITANI, J., AND IGARASHI, T. 2011. Converting 3d furniture models to fabricatable parts and connectors. *ACM Trans. Graph. 30*, 4 (Aug.), 85:1–85:6.

LI, Y.-T., HU, S.-M., AND SUN, J.-G. 2002. A constructive approach to solving 3-d geometric constraint systems using dependence analysis. *Computer-Aided Design 30*, 3, 97–108.

MCCRAE, J., SINGH, K., AND MITRA, N. J. 2011. Slices: a shape-proxy based on planar sections. *ACM Trans. Graph. 30*, 6 (Dec.), 168:1–168:12.

MERRELL, P., SCHKUFZA, E., AND KOLTUN, V. 2010. Computer-generated residential building layouts. *ACM Trans. Graph. 29* (December), 181:1–181:12.

MERRELL, P., SCHKUFZA, E., LI, Z., AGRAWALA, M., AND KOLTUN, V. 2011. Interactive furniture layout using interior design guidelines. *ACM Trans. Graph. 30* (Aug.), 87:1–87:10.

MITRA, N. J., YANG, Y.-L., YAN, D.-M., LI, W., AND AGRAWALA, M. 2010. Illustrating how mechanical assemblies work. *ACM Trans. Graph. 29* (July), 58:1–58:12.

MORI, Y., AND IGARASHI, T. 2007. Plushie: an interactive design system for plush toys. *ACM Trans. Graph. 26*, 3 (July).

NEUFELD, L. 2003. *Making Toys That Teach: With Step-by-Step Instructions and Plans*. Taunton Press.

PENG, X., LEE, K., AND CHEN, L. 2006. A geometric constraint solver for 3-d assembly modeling. *The International Journal of Advanced Manufacturing Technology 28*, 561–570. 10.1007/s00170-004-2391-1.

PEPPE, R. 2005. *Making Mechanical Toys*. Crowood Press.

ROY, U., PRAMANIK, N., SUDARSAN, R., SRIRAM, R., AND LYONS, K. 2001. Function-to-form mapping: model, representation and applications in design synthesis. *Computer-Aided Design 33*, 10, 699 – 719.

STAVA, O., VANEK, J., CARR, N., AND MECH, R. 2012. Stress relief: Improving structural strength of 3d printable objects. In *to appear: Proceedings of SIGGRAPH 2012*.

TALTON, J. O., LOU, Y., LESSER, S., DUKE, J., MĚCH, R., AND KOLTUN, V. 2011. Metropolis procedural modeling. *ACM Trans. Graph. 30* (Apr.), 11:1–11:14.

TIERNEY, L., AND MIRA, A. 1999. Some adaptive monte carlo methods for bayesian inference. *Statistics in Medicine 18*, 2507–2515.

UICKER, J. 2010. *Theory of Machines and Mechanisms*. Oxford University Press.

VEACH, E., AND GUIBAS, L. J. 1997. Metropolis light transport. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, SIGGRAPH '97, 65–76.

VERROUST, A., SCHONEK, F., AND ROLLER, D. 1992. Rule-oriented method for parameterized computer-aided design. *Computer-Aided Design 24*, 10, 531 – 540.

WAMPLER, K., AND POPOVIĆ, Z. 2009. Optimal gait and form for animal locomotion. *ACM Trans. Graph. 28* (July), 60:1–60:8.

WAMPLER, II, C. W. 1986. Manipulator inverse kinematic solutions based on vector formulations and damped least-squares methods. *IEEE Trans. Syst. Man Cybern. 16* (January), 93–101.

WANG, J. M., FLEET, D. J., AND HERTZMANN, A. 2009. Optimizing walking controllers. *ACM Trans. Graph. 28* (December), 168:1–168:8.

XIN, S., LAI, C.-F., FU, C.-W., WONG, T.-T., HE, Y., AND COHEN-OR, D. 2011. Making burr puzzles from 3d models. *ACM Trans. Graph. 30*, 4 (Aug.), 97:1–97:8.

XU, W., WANG, J., YIN, K., ZHOU, K., VAN DE PANNE, M., CHEN, F., AND GUO, B. 2009. Joint-aware manipulation of deformable models. *ACM Trans. Graph. 28* (July), 35:1–35:9.

YU, L.-F., YEUNG, S.-K., TANG, C.-K., TERZOPOULOS, D., CHAN, T. F., AND OSHER, S. J. 2011. Make it home: automatic optimization of furniture arrangement. *ACM Trans. Graph. 30* (Aug.), 86:1–86:12.