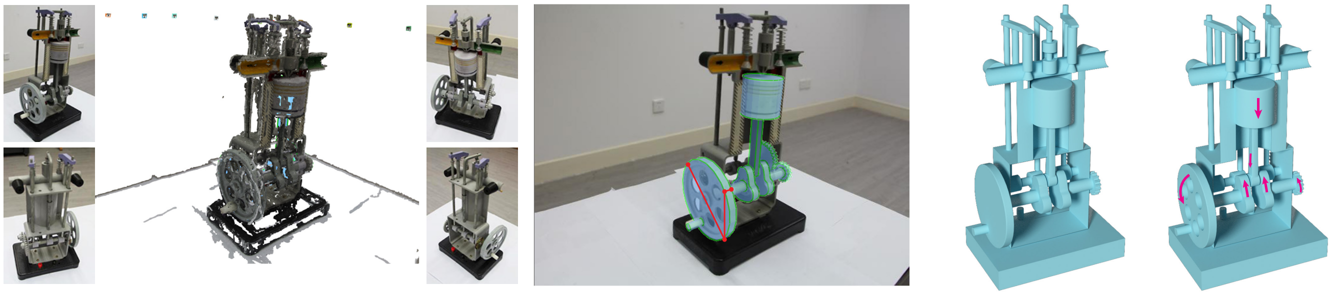
Interactive Mechanism Modeling from Multi-view Images



(a) Input multi-view data (b) Modeling (c) Interactive modeling (d) Refined result

Figure 1: *(a) An example of point cloud data reconstructed from multi-view images of a piston-engine model. (b) The extracted 3D part models are overlaid on the image with cyan color, and the three red strokes indicate the current part drawn by the user. The details added in the refined result in (d), i.e. the holes of the driving gear, are generated in Blender softwares based on the 3D modeling results using our approach in (c) within 3 minutes. The whole modeling process only takes around 25 minutes on an off-the-shelf computer. The red arrows visualize the allowed motion of its movable parts. Please see the accompanied video for the kinematic simulation of the reconstructed mechanism model.*

Abstract

In this paper we present an interactive mechanism modeling system from multi-view images. Its key feature is that the generated 3D mechanism models contain not only their geometry information but also their internal motion structure: they can be directly animated through kinematic simulation. Our system can be generally separated into two steps: interactive 3D modeling and stochastic motion parameter estimation. At the 3D modeling step, our system is designed to integrate the sparse 3D points reconstructed from multi-view images and a sketch interface to achieve accurate 3D modeling of a mechanism. To recover the motion parameters, we record a video clip of the motion of an input mechanism and adopt stochastic optimization to recover its motion parameters by matching the edge information in video frames and the projected 2D silhouettes of the 3D parts. Experimental results show that our system can be practically applied for the 3D modeling of a range of mechanisms from simple mechanical toys to complex mechanism objects.

CR Categories: I.6.8 [Simulation and Modeling]: Types of Simulation—Animation I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Curve, surface, solid, and object representations

Keywords: Interactive modeling, multi-view images, sketch-based modeling, motion structure optimization, mechanism modeling

1 Introduction

Mechanism modeling is one of the central tasks in commercial CAD softwares, such as Autodesk Inventor and Solidworks. A typical mechanism modeling procedure usually includes two steps: create 3D part models and design joint constraints between the parts so as to achieve the desirable motion. However, the state-of-the-art CAD software packages focus on providing professional tools for expert users to create mechanisms; it is a non-trivial task for an average user to learn the necessary knowledge to efficiently use such software packages to create complex mechanism models.

Motivated by the recent progress in image-based interactive modeling [Sinha et al. 2008; Chen et al. 2013], in this paper we propose a multi-view image-based mechanism modeling system for a variety of users who may only have preliminary knowledge in mechanism design. The system is designed to utilize the visible geometric relationship between parts in the images to guide users in complex mechanism modeling. Even though there exist a variety of 3D modeling algorithms from multi-view images [Hartley and Zisserman 2004; Seitz et al. 2006], it is technically challenging to straightforwardly extend these methodologies to model complex man-made objects such as high-quality, functioning 3D mechanism models. Its main reasons are: First, as a complex man-made object, a mechanism typically consists of many interlinked parts, which leads to severe occlusions among them in 2D images. Therefore, existing multi-view 3D modeling approaches would fall short to handle such cases. Second, its internal motion structure, including the motion constraints among parts, is essential to realize the functional design of a mechanism. Nevertheless, identifying such a motion structure is non-trivial, because this would require algorithms not only to model the 3D shape of each part accurately but also to robustly estimate its underlying motion constraints, i.e., the joints among different parts.

The major contribution of this paper is the development of a novel interactive mechanism modeling approach based on multi-view images, which tackles the above technical challenges on the accurate reconstruction of both part shapes and motion parameters of mechanisms:

- Inspired by [Chen et al. 2013], we extend the stroke-based sweep modeling interface to create 3D parts. Our interactive modeling interface not only enables users to create the 3D shape of each part from the view where it is mostly visible using generalized cylinders or cube shapes, but also assists users to determine the accurate shape and placement of each part in 3D through the integration of the sparse point cloud data reconstructed from them. Besides, new snapping operations are introduced to assist the estimation of correct geometric relationship between parts, and the part details can be added through parametric modeling. Furthermore, we also develop an alignment algorithm, a variant of the inter-part optimization in [Chen et al. 2013], to optimize the modeling result to enforce the alignment constraints among parts, such as parallelism, orthogonality and attachment. The algorithm is a single step optimization with 3D point cloud registration term and additional snapping constraints.

- In order to recover the motion structure of a mechanism, we record a video clip of its movements under a specific view. A stochastic motion parameter estimation algorithm is designed to estimate the types of joints between parts and the motion parameters of each part so that the motion of 2D part silhouettes matches the motion of edges detected in the acquired motion video. The joint types are treated as discrete variables, and their possible values are estimated through the geometric shape of their intersections according to the mechanism-specific domain knowledge [Slater 2011]. The formed discrete configuration space is searched using a tree-like representation of mechanism to avoid combinatorial explosion, under the assumption that the motion of the mechanism can be kinematically controlled. The kinematic loops are supported in our system.

We have tested our system on a variety of mechanical objects, ranging from mechanical toys to real mechanisms. The modeling time is around 10 to 30 minutes on an off-the-shelf computer, depending on the complexity of the mechanism. Our experimental results show that our system is capable of effectively reconstructing various mechanism objects from multi-view images, and the reconstructed 3D mechanism models can be directly used for a variety of animation and functional demonstration applications.

2 Related Work

Multi-view 3D modeling: The goal of multi-view 3D modeling is to create complete 3D models from the captured images, and its theory is rigorously formulated in the seminal textbook [Hartley and Zisserman 2004]. A comprehensive survey of multi-view 3D modeling research is given in [Seitz et al. 2006]. Generally, the pipeline of multi-view 3D modeling consists of three steps: the calibration of intrinsic and extrinsic camera parameters [Zhang 2000; Song et al. 2013], dense correspondences among multiple images [Furukawa and Ponce 2007; Valgaerts et al. 2012; Ahmadabadian et al. 2013], and stereo triangulation to compute 3D points. Multi-view 3D modeling has also been extended to handle large scale images from Internet, such as photo tourism [Snavely et al. 2006] and Rome city modeling [Agarwal et al. 2011].

Our work is mostly related to the interactive architecture modeling from a collection of photos in [Sinha et al. 2008]. However, different from the 3D plane construction in [Sinha et al. 2008], our work focuses on the shape of mechanical part primitives and their motion constraints to obtain both geometry and structure information of a mechanism object.

Sketch-based 3D modeling: Sketching interface, in contrast to WIMP (Windows, Icon, Menu, Pointer), is a metaphor of pen-ink user interaction, which is preferred by designers for the concept design of 3D products. It has been applied to a variety of applications, such as architecture modeling [Chen et al. 2008], volumetric modeling [Owada et al. 2007; Schmidt et al. 2007; Owada et al. 2004], the interpretation of concept sketches [Shao et al. 2013] and 3D shape retrieval [Eitz et al. 2012b]. Eitz et al. [2012a] also investigated how to recognize object categories from user sketches. A survey of sketch-based modeling techniques is given in [Olsen et al. 2009].

An essential step in sketch-based 3D modeling is to convert the 2D strokes drawn on the screen into 3D curves. Igarashi et al. [1999] create 3D objects using inflation, by assuming the user-drawn 2D freeform strokes are silhouettes of the 3D objects. In [Nealen et al. 2007], the drawn freeform curves can be further edited to control the final shape of the 3D objects. With the assistance of epipolar geometry or 3D scaffold, users can draw a freeform 3D curve network as a sparse representation of a 3D shape [Bae et al. 2008; Schmidt et al. 2009]. A recent contribution in [Xu et al. 2014] interprets 2D strokes as 3D curves through the integration of sketch fidelity and regularity constraints. The constructed 3D curve network can be converted into a surface model by interpreting the geometry of its closed curve patches as spline patches or quad meshes [Abbasinejad et al. 2011; Bessmeltsev et al. 2012; Sadri and Singh 2014].

Our work is inspired by a recent work on combining sketching interface with image edge information for sweeping-based modeling [Chen et al. 2013], named 3-sweep modeling. While the 3-sweep system focuses on 3D modeling from a single photo, the goal of our work is to model mechanism objects with complex structures, which is difficult to be fully captured in a single view due to unavoidable internal occlusions. Our algorithm also supports the integration of sparse point cloud information and image edges to get an illustrative 3D mechanism model to aid product design.

Multi-component 3D models: The key of the processing of a multi-component 3D model is how to maintain the proper constraints among its components. For example, the editing of a multi-component 3D mesh model emphasizes the importance of the motion, symmetry, and primitive shape preserving constraints for high-quality editing results [Xu et al. 2009; Gal et al. 2009; Zheng et al. 2011; Kraevoy et al. 2008]. Such constraints have also been considered in various research efforts on multi-component model processing, including hierarchy analysis [Wang et al. 2011], internal structure visualization [Li et al. 2008], image-based modeling [Xu et al. 2011; Zheng et al. 2012] and component-based geometry synthesis and fitting to point clouds [Kalogerakis et al. 2012; Xu et al. 2012; Shen et al. 2012]. A comprehensive survey of structure-aware processing of multi-component models can be found in [Mitra et al. 2013a].

For a mechanism with multiple parts, Niloy et al. [2013b] proposed a method to visualize its possible motion. Recently, several approaches have been presented to design and fabricate mechanism objects using 3D printing technique [Zhu et al. 2012; Bächer et al. 2012; Ceylan et al. 2013; Coros et al. 2013; Thomaszewski et al. 2014]. There also exist researches on modeling mechanism from point-cloud data. However, they mainly focus on extracting geometric primitives, such as cylinders and 3D parts, and did not attempt to recover the motion structure of mechanisms [Huang and Menq 2002; Bey et al. 2011]. The goal of our work is to facilitate the 3D modeling of functional mechanism objects, and the output of our system is multi-part mechanism models which are ready to be the input of the above editing, analysis and visualization algorithms.

3 Multi-view Modeling Preliminaries

Based on user strokes drawn on 2D images, we need to estimate their 3D positions in order to reconstruct 3D part shapes. In other words, given multi-view images as the input, for a 2D pixel we need to estimate its global Z value so as to decide its 3D position in the world coordinate system. In our approach, the Z values are treated as the optimization variables. After the Z values are obtained, the global X and Y coordinates for a pixel (x, y) can be determined using the projection matrix calculated in the construction of sparse point cloud data, described below.

Let us first denote the intrinsic and extrinsic matrices associated to a view i by $\mathbf{K}^i$ and $\mathbf{E}^i = \{\mathbf{R}^i, \mathbf{T}^i\}$, where the intrinsic matrix $\mathbf{K}^i$

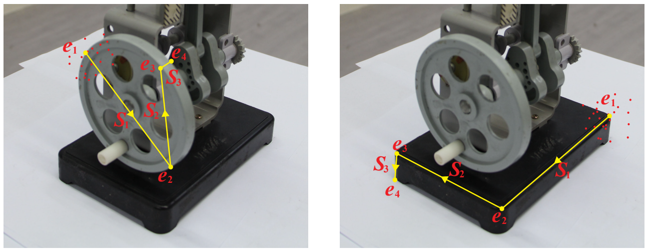


Figure 2: *Three strokes based single part modeling interface. The first two strokes represent the profile and the third stroke is for extruding the profile. The red dots indicate the sparse 3D point around the end points of the drawn strokes.*

is a 3×3 matrix, $\mathbf{R}^i$ a 3×3 rotation matrix, and $\mathbf{T}^i$ is a 3×1 translation vector. Given a 3D coordinate $\mathbf{V} = \{X, Y, Z\}$, the well-known projection formula can be written as:

$$\bar{\mathbf{v}}^i = P(X, Y, Z) = \mathbf{K}^i(\mathbf{R}^i\mathbf{V} + \mathbf{T}^i), \tag{1}$$

where $\bar{\mathbf{v}}^i = \{\bar{x}^i, \bar{y}^i, \bar{z}^i\}$ is the homogeneous coordinate of the projected pixel coordinate. Afterwards, the 2D pixel coordinate can be easily calculated by $\{x^i = \frac{\bar{x}^i}{\bar{z}^i}, y^i = \frac{\bar{y}^i}{\bar{z}^i}\}$. Given the pixel coordinate $\{x^i, y^i\}$ at a specific view and its global Z value, we can easily derive from Eq.1 that its global $\{X, Y\}$ coordinates as two linear functions of Z:

$$\begin{aligned} X &= \mathbf{f}_X(Z) = a_x Z + d_x \\ Y &= \mathbf{f}_Y(Z) = a_y Z + d_y \end{aligned} \tag{2}$$

where the coefficients, $\{a_x, a_y, d_x, d_y\}$, are computed through $\{x^i, y^i\}$ and the entries of $\mathbf{K}_i$ and $\mathbf{F}_i$. For the detailed derivation of Eq. (2), please refer to the appendix.

Fig. 1 illustrates the basic interface and pipeline of our system. The transparent cyan shapes are the created mechanical parts, while the three red strokes illustrate the basic stroke-based interface to create the next part. The 3D point cloud data is reconstructed from multi-view images using VisualSFM software [Wu 2015].

4 Single Part Modeling

In this section, we describe the interactive part modeling step including its user interface, depth value calculation and the parametric model of the gear tooth to add details to the generalized cylinder.

4.1 3D Part Generation Pipeline from Images

We use three steps to create the 3D shape of a single part. First, the user selects a proper view where the part to be constructed is mostly visible. The view selection can be done by selecting an image or navigating through the 3D point cloud data. Second, the user needs to decide whether the part should be constructed directly based on the sparse point cloud data, or constructed by snapping to the surface and/or the edges of already constructed parts. Third, the user draws the base profile and sweeping axis to construct the part shape.

Simple primitives: Simple primitives such as generalized cylinder and cube shapes are supported in our system. We adopt the stroke interface similar to [Chen et al. 2013], where the first two consecutive strokes are used to specify the base profile, and the third stroke serves as the main axis for the sweeping operation. The places to draw the strokes are selected from the silhouette of the part in the image. As shown in Fig. 2, the three straight strokes $\{S_1, S_2, S_3\}$ are connected to each other, and we denote the four end points of the three strokers as $\{e_1, e_2, e_3, e_4\}$.

We further allow the base profile to be a free-form, non-uniform B-Spline curve to model parts with complex shapes [Cao et al. 2014]. See Fig. 4 for an example of the modeling of a crank in the image. In this case, we allow the user to draw the first two stokes to specify the plane for the free-form base profile. There could also exist very thin parts in a mechanism, for example, the two thin rectangles besides the piston cylinder (see Fig. 1). Our system allows the user to only draw the base profile, and the third dimension is only extruded with a user-specified thickness parameter through the third stroke.

Some parts might be decomposed into several sweeping surfaces to reduce the modeling time. In that case, we allow user to group these decomposed sweeping surfaces into one part to ease the following motion parameter estimation.

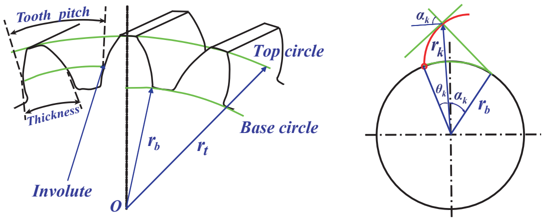
Derivation of Z values for stroke end points: The end points of the first two strokes define the base profile. For a cylinder, the first stroke is supposed to define the diameter of the base circle, and the end point e_3 of the second stroke indicates a point on the circle. We then have the perpendicular relationship between two edges $\overrightarrow{e_1e_3}$ and $\overrightarrow{e_2e_3}$ in 3D. If Z_1 and Z_2 for e_1 and e_2, respectively, are known, the Z_3 for e_3 can be determined via the following perpendicular condition:

$$\begin{aligned} \overrightarrow{e_1e_3} \cdot \overrightarrow{e_2e_3} =& (\mathbf{f}_X(Z_3) - \mathbf{f}_X(Z_1)) * (\mathbf{f}_X(Z_3) - \mathbf{f}_X(Z_1)) \\ &+ (\mathbf{f}_Y(Z_3) - \mathbf{f}_Y(Z_1)) * (\mathbf{f}_Y(Z_3) - \mathbf{f}_Y(Z_1)) \\ &+ (Z_3 - Z_1) * (Z_2 - Z_1) = 0 \end{aligned} \tag{3}$$

Eq. 3 is a quadratic function of Z_3, and our approach picks its solution closest to Z_1 in the local coordinate system of the selected view. For Z_1 and Z_2, we require the user to draw stroke end points on the corresponding points computed from multi-view images, where the Z values can be easily determined. In the case that only Z_1 is known, we assume Z_2 has the same value in the local coordinate system of the selected view (Fig. 2). The Z values of a cube can also be estimated through the perpendicular condition between the first and second strokes.

Snapping operations: Parts in a mechanism are often attached to each other. We thus introduce a new concept of *snapping operations* to exploit this relationship to obtain accurate placements of the parts, which would help to resolve the ill-conditioned inverse projection from 2D points in an image to 3D points. The 3D surface and/or edge for the snapping operations can be determined manually or automatically by checking whether the drawn base profile is inside the 2D region of the created parts. Our system supports the following two types of snapping operations.

- *3D object snapping*, defined to snap a part to the surface of already created 3D objects. When the user draws strokes to specify the base profile of a part, our system detects whether the end points of the strokes are on the surface of already drawn objects through the OpenGL selection scheme. If so, the depth values of the end points are set to be the depth values extracted from the Z-buffer at those points.
- *Silhouette snapping*, defined to snap a part to the Silhouette of created 3D objects. Our system detects whether the end points of the drawn strokes are sufficiently close (e.g., below a user-specified threshold) to the Silhouette of already drawn objects. If so, the depth values of the end points are set to the depth value of the point on the silhouette that is closest to the drawn end points. The silhouettes of the drawn 3D objects are determined through the edge detection from their OpenGL rendering result on the current view.



(a) Gear tooth (b) Involute (red line)

Figure 3: *Spur gear (a). The profile of its tooth is generated using the involute shown in (b).*

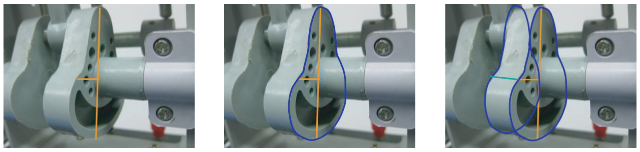


Figure 4: *Parts with a free-form base profile. (Left) The first two strokes are used to define the plane for the base profile. (Middle) A free-form profile is drawn on the plane. (Right) The third stroke is used to extrude the base profile to form the part.*

The above 3D object snapping operations are performed prior to the silhouette snapping operations in our system. Note that the user can manually turn on/off snapping operations when the system performs wrong snapping operations.

4.2 Part Details and View Selection

Part details: Although the combination of primitive shapes and the extrusion of free-form profile curves can cover most shapes of the mechanical parts in our experiments, it will be tedious for the user to model the details of mechanical parts, such as gear tooth, using such operations. To address this issue, our system supports the use of parametric models to efficiently add geometric details to these mechanical parts. Without the loss of generality, we take the generation of the tooth shape of spur gears as an example using the popular involute method in Mechanics [Slater 2011; Wikipedia 2015], as illustrated in Fig. 3(b). In our current implementation, the tooth number, hight and thickness ratio are manually specified. A general rule in mechanism is that the tooth size of two pairing gears should be roughly the same to smoothly transfer the motion. Thus, for two pairing gears, if the tooth number of one gear is determined, then that of the other can be determined accordingly (refer to [Slater 2011] for details). Our system adopts this rule to save the manual efforts in gear chains.

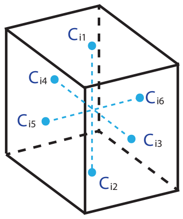
View selection: In the modeling of a mechanical part from multiview images, the user can select a specific view with the largest visibility of the part to handle occlusions. Besides directly selecting images, we also allow the user to rotate the object in 3D view; once a 3D view is selected, we can automatically determine its corresponding 2D image by selecting the captured image whose extrinsic matrix is closest to the current 3D view. After view selection, all the modeled parts are rendered into the current view to facilitate the above operations in part modeling, including stroke drawing and snapping operations.

5 Part Alignment Optimization

Mechanism objects are typical examples of man-made objects, and their parts typically conform to global alignment constraints, such as parallelism, orthogonality, and attachment constraints [Li et al. 2011]. In [Chen et al. 2013], these constraints are represented by the alignment constraints among the vectors formed by the anchor points of a primitive and optimized to improve the quality of the modeling results. We also adopt the anchor point idea for global alignment constraints but reinforce them to integrate the snapping constraints.

Moreover, since the 3D information required to derive the 3D positions of anchor points are nicely initialized by the multi-view reconstruction result, our optimization algorithm does not need to start with the guess of depth information as in [Chen et al. 2013]. It directly optimizes the 3D positions of anchor points according to various alignment constraints. In this section, we first briefly describe the selection of anchor points, and then details the snapping constraint and our optimization algorithm.

Anchor points representation: The anchor points $\{\mathbf{C}_{ij}, j = 1..m_i\}$ for a part $\mathbf{P}_i$ are usually selected to be the end points of its central axes. In this way, the parallelism constraint between parts can then be represented by the parallelism of the central axes of two parts. The anchor points are selected differently for different primitive shapes. For a generalized cylinder, we use two end points on its main axis as the optimization variables while keeping its radius constant. Thus, the value of m is 2 in this case. For a cube shape, we approximate its shape using six anchor points (i.e., $m = 6$) that stay on the three main axes passing through the cube center (see the right inset). Thus, the cube can be translated and scaled if the lengths of the main axes are changed while optimizing the positions of its anchor points.



The six types of constraints between the anchor points of mechanical parts used in [Chen et al. 2013], including parallelism, orthogonality, collinear axis endpoints, overlapping axis endpoints, coplanar axis endpoints, and coplanar axes, are also supported in our system.

Snapping constraint: It is used to snap an arbitrary point of a part to one plane or edge of another part, as illustrated in Fig. 5, which is called *point-on-line* and *point-on-plane* constraints in this paper. Note that the points to be snapped are not anchor points but usually the stroke endpoints drawn by the user in these two constraints.

The snapping constraint is integrated into the optimization by representing the position of a point, for a part, by the linear combination of its anchor points. Specifically, for any point $\mathbf{v}_i^k$ of a created 3D part $\mathbf{P}_i$, its 3D coordinates can be represented by the combination of its anchor points as $\mathbf{v}_i^k = \sum_j w_{ij}^k C_{ji}$. Arbitrary lines and planes can also be derived from the 3D positions of anchor points subsequently. The coefficients are computed after the single part modeling and kept constant in the alignment optimization.

Such a representation is suitable for generalized cubes with three pairs of anchor points, which can be viewed as the local coordinate system of the cube. However, it is not applicable to generalized cylinders with only one pair of anchor points. We thus only allow to apply the point-on-line or point-on-plane constraint at their anchor points, and only the top and bottom cap planes of a cylinder are allowed to be the planes in the point-on-plane constraint.

Alignment constraints optimization: The energy function is designed to maximize the projection accuracy of anchor points while preserving the identified constraints. Specifically, it can be formu-

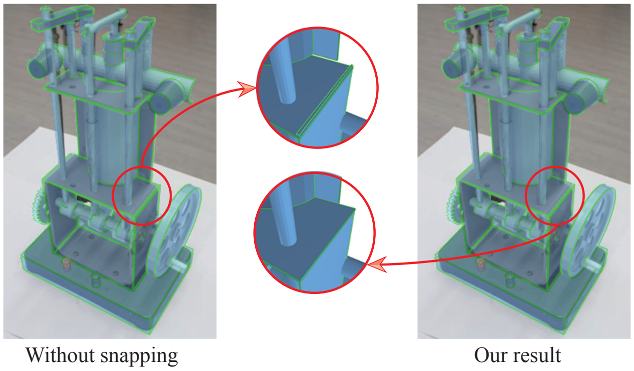


Figure 5: *Snapping constraint. Left: optimization without snapping constraint. Note the misalignment at the two cubes in the magnified view. Right: our optimization with snapping constraints. The boundary of two cubes are perfectly snapped.*

lated as follows:

$$
\begin{aligned}
E = \sum_{i,j} \|P(C_{ij}) * \bar{z}_{ij} - [x_{ij}, y_{ij}]\|^2 + w_d \sum_m \|dist(\mathbf{v}_m, \mathbf{e}_n)\|^2 \\
\text{Subject to}:\ S_{il}(C_{ij}, C_{ln}) = 0,\ i,l = 1 \cdots K \\
G_i(C_{ij}, C_{in}) = 0,\ j,n = 1 \cdots m_i
\end{aligned}
\tag{4}
$$

In the first term, $\bar{z}_{ij}$ represents the z component of the projected homogeneous coordinate, the projection function $P(.)$ is defined in Eq. 1, and $[x_{ij}, y_{ij}]$ is the 2D pixel coordinate of its projection on the view where the part is drawn. The values of $[x_{ij}, y_{ij}]$ are kept constant as the initial drawn positions in the optimization procedure. Therefore, our objective function is designed to minimize the variation between the initially drawn parts and the optimized parts: the anchor points are only allowed to move around the initial positions. S_{il} represents the identified semantic constraint between two parts $\mathbf{P}_i$ and $\mathbf{P}_l$, while G_i represents the orthographic constraint between a pair of anchor vectors of a cube part $\mathbf{P}_i$.

The second term is used to minimize the distance of sparse 3D points to the reconstructed model, where $\mathbf{e}_n$ is the closest part of a 3D point $\mathbf{v}_m$. The computation of distance function $dist$ is related to the part type. For a generalized cylinder, the distance is computed as the absolute difference between the closest distance from its central axis and a 3D point and the radius of the cylinder. The distance between a 3D point to a cube is just the closest distance from the point to any of its six planes. All the distance functions can be represented by the coordinates of the anchor points. If the closest distance between a part and a 3D point is within a threshold (default 0.5), or the closest point on a part according to the distance function is within the volume of the part, the 3D point is deemed to belong to the part. The weight w_d is set to be 10^{-4} in our implementation. It is small since we do not want the optimization result to be overweighted by the substantial noise in the reconstructed point cloud. The energy function in Eq.4 is optimized through augmented Lagrangian method [Jorge and Stephen 2006].

6 Motion Parameters Estimation

While we have the basic geometry of the parts in a mechanism after the interactive modeling step, it is still not enough to reveal its kinematic structure: the interaction between connected parts to realize the functional design (i.e., target motion) of the mechanical assembly. As shown in [Mitra et al. 2013b; Zhu et al. 2012], the motion of a mechanism is initialized at the driving part and transferred to other parts through its kinematic chain, i.e., the joint types between connected parts and the shape of each part. Therefore, we design a motion parameter estimation algorithm to determine these three kinds of information, namely, the motion of the driving part, the joint types between connected parts, and the shape parameters of each part. The necessity of re-estimating the shape parameters is due to that the shapes of certain occluded parts might be incorrect or they might not be sufficiently accurate to reproduce the expected motion. For example, as shown in Fig. 1, the slider connected to the piston is occluded; therefore, it is difficult to obtain its correct length at the interactive modeling step, which would result in inaccurate motion as illustrated in Fig. 7.

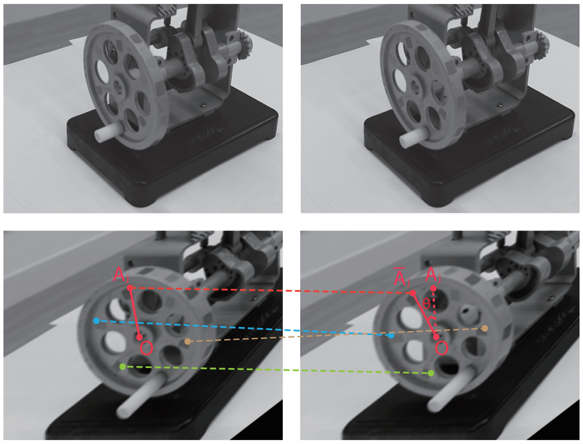


Figure 6: *The computation of rotation angle. Top row: The two frames of the driving gear in an input motion video. Bottom row: The two front views of the gear by homography transformation. The lines connect corresponding feature points, and the rotation angle can be estimates through dot product between the vectors formed by connecting the rotation center and the feature points.*

We thus use a pre-recorded video clip of the mechanism motion as the input of our motion parameters estimation algorithm, which is taken at one of the viewpoints used in the aforementioned multi-view modeling pipeline. Prior to the optimization, the moving parts are detected by checking whether there are optical flow information in their visible projected regions in the first video frame [Sun et al. 2010], or manually specified by the user. However, since optical flow algorithms rely on the accurate correspondence information between video frames, such correspondences may not be sufficiently robust to be used to recover the rigid motion of mechanism parts that typically do not have salient textures. To this end, in our approach we first adopt a feature tracking method to obtain the detailed motion of the driving part by adding a few artificial marks on it. Then, the motion parameters of the rest parts are optimized by checking whether their projected silhouettes sufficiently match the edges at each video frame. The user also need to manually specify the driving part to jumpstart the optimization process.

6.1 Joint Types

We treat a mechanism as a collection of rigid bodies inter-connected to transmit rigid motions, and a joint connects two parts to form a kinematic pair. Different joint types impose distinct motion constraints between two parts. Figure 8 summarizes the four main types of joints used in our mechanism modeling experiments: fixed or welded joints, revolute joints, gear-2-gear contact joints, and point-on-line joints.

Rules-based guess for joint types: As pointed out in [Mitra et al.

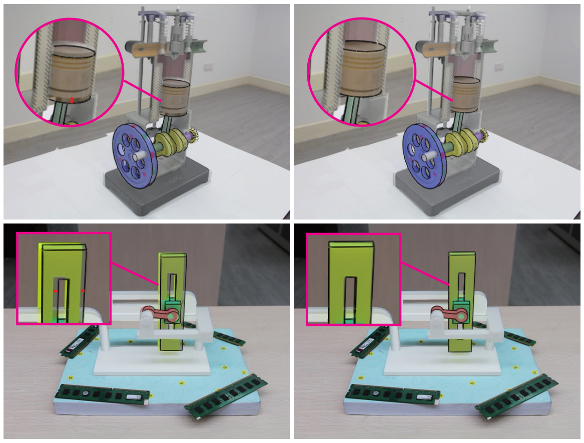


Figure 7: *The stochastic shape parameter optimization results. Left column: Although the part silhouettes usually match the edge information well in the first frame of the video clip of mechanism motion, the mismatching between the silhouettes and the edge information, indicated by short red line segments, still occur in the subsequent video frames due to the inaccurate shape parameters from modeling. Right column: Optimization results. The mismatching is corrected through shape parameter optimization.*

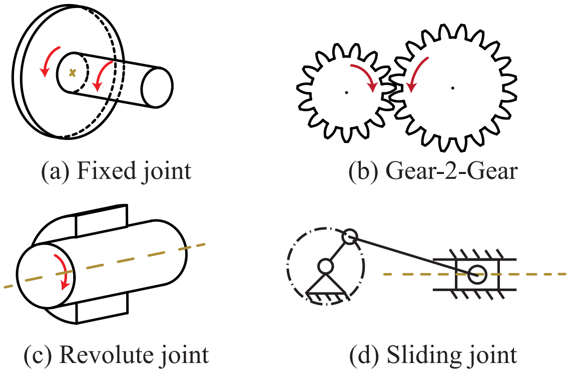
2013b; Xu et al. 2009], the relative motion of two parts connected via a joint should be a slippable motion that does not lead to interpenetration of parts for the smooth motion of both parts. It can be determined by the intersection surface. Particularly, the slippable motion is rotation for a cylindrical intersection surface, corresponding to revolute joint, and sliding motion for a planar intersection surface, called sliding joint in this paper. As such, we can categorize the joint types based on the slippable motion of the intersection surface, and further derive a set of rules to have a reasonable guess on the possible joint types as follows:

- If a generalized cylinder is connected to the surface of another part, we can soundly guess that the joint between them would be an either fixed or revolute joint, since the intersection surface in this case is either a generalized cylinder whose slippable motion is rotation or a generalized cylinder that can be a shaft to transfer the motion so it is fixed to that part.
- If a generalized cylinder or a slim cube serves as a connecting rod and it is connected to the interior of another part, we can guess that there is a fixed joint or sliding joint. For the sliding joint, its sliding direction is selected as the axis of the local coordinate system. Only the line direction that can keep the length of the connecting rod constant in motion is feasible, which can be tested in kinematic simulation.
- If two gears are in contact, it is unquestionable there exists a gear-2-gear contact joint to transfer the rotation motion from one gear to the other.
- If a cube is placed on a plane, we assume that there is a point-on-line joint, and the line directions parallel to the surface are two axes of the local coordinate system of the cube, since the slippable motion for a plane is sliding.

To the end, we can obtain a possible, small set of joint types for each pair of connected parts, i.e., they serve as the possible values of discrete joint types.

6.2 Motion Estimation of the Driving Part

It is critical to obtain the accurate motion of the driving part of a mechanism, since it actually determines the entire motion of the mechanism through its kinematic chain. In our algorithm, the motion of the driving part is referred to as how it moves in each frame of the recorded motion video. For example, the rotational motion of the driving part is calculated as the rotation angle at each frame with respect to a rotation axis.



(a) Fixed joint

(b) Gear-2-Gear

(c) Revolute joint

(d) Sliding joint

Figure 8: *Joints. (a) The two parts connected via fixed joint move in the same direction. (b) The gear-2-gear joint can transfer the rotational motion from one gear to another. (c) The relative motion of a revolute joint is rotation. (d) The sliding (point-on-line) joint indicates that the block can slide on the planar surface of another part.*

Since the viewpoint of the input video is usually selected to ensure parts with large motions are visible, the perspective effect poses difficulties in the computation of rotation angles. We choose to remove the perspective distortion of the primitive driving part by homography transformation and then calculate the rotation angle through feature matching [Lowe 2004]. As shown in Fig. 6, the circle of the driving gear in the recorded viewpoint is transformed to its front view at two consecutive frames i and $i+1$. With the corresponding feature points, we can calculate the rotation angle with respect to the rotation axis as follows:

$$\theta^{i,i+1} = \operatorname{avg} \sum_{j} \arccos \frac{\overrightarrow{OA_j} \cdot \overrightarrow{O\overline{A}_j}}{\left|\overrightarrow{OA_j}\right| \cdot \left|\overrightarrow{O\overline{A}_j}\right|} \tag{5}$$

where A_j and $\overline{A}_j$ are two corresponding feature points, and O is the rotation center which is computed by the projection of the 3D rotation center of the modeled driving gear. The homography transformation is computed by choosing four points on the 3D gear and their projections on the front view by setting up a virtual orthographic camera whose principal axis is the rotation axis of the gear.

The initial rotational motion calculated by Eq. 5 can be further optimized by checking how the image regions of the driving gear in two consecutive frames are matched. Specifically, we optimize the rotational motion using the following objective function:

$$E_{rot} = \sum_{m}^{W} \sum_{n}^{H} \left| I^{i}_{m,n} - I^{i+1}_{rot(m,n)} \right|^2 \tag{6}$$

where W and H are the dimension of the gear in the front view, and $rot(m,n)$ denotes the function to rotate the image coordinate $\{m,n\}$ at frame i to frame $i+1$. To reduce the influence of the background to the optimization, we first render the 3D gear at the starting video frame and only choose the pixels inside its projected region in the optimization. Thus, in Fig. 6, the pixels inside the holes of the gear are eliminated from the optimization.

If the driving motion is a linear translation, we can similarly compute its 3D motion through corresponding feature points. One extra

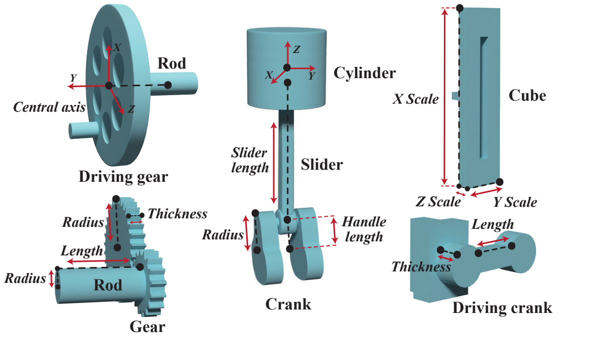


Figure 9: *Shape parameters. They are indicated by the texts, except two local coordinate systems and part names.*

computation is to transform the translation in pixel unit to the corresponding 3D motion on the translation axis based on the known projective information.

6.3 Stochastic Estimation Algorithm

If the projected silhouettes of each part are close to the matched edges in each video frame, we deem the current set of motion parameters are sufficiently accurate to reproduce the recorded motion. Therefore, the objective function in motion parameters estimation is designed to be the summed squared distance between the silhouette pixels and their corresponding edge pixels in video frames, denoted by *silhouette matching energy*. It can be formulated as follows:

$$E(\mathcal{J},\mathcal{P}) = \min_{\mathcal{J},\mathcal{P}} \sum_{i}^{n} \Big(\sum_{j}^{m^i} \|\tilde{e}_j^i(\mathcal{J},\mathcal{P}) - e_j^i\|^2 / m^i \Big) \qquad (7)$$

where $\mathcal{J}$ denotes the set of joint types and $\mathcal{P}$ denotes the set of part shape parameters. The pixel j at frame i on the 2D silhouettes of the 3D part is denoted by $\tilde{e}_i^j$, and it is computed by extracting the edge pixels from the rendered result of the 3D part at each frame with the optimized camera settings in multi-view reconstruction. It can be easily done by retrieving the frame buffer in OpenGL rendering pipeline. Also, e_j^i denotes the edge pixel at video frame i, corresponding to $\tilde{e}_i^j$. We determine e_j^i in the following two steps: (i) we first identify the closest edge pixel in the recorded video frame i; (ii) if the closest distance is below a user-specified threshold (10 pixels in our experiments) and the angle between the normals at these two pixels is below a threshold (45° in our experiments), the closest edge pixel is accepted as the e_j^i. If such a e_j^i cannot be identified using the above protocol, $\|\tilde{e}_j^i(\mathcal{J},\mathcal{P}) - e_j^i\|$ in the above Eq. 7 is set to a large penalty value (in our experiments it is set to 1000). The correspondences between edge pixels and silhouette pixels of all the frames except the first frame need to be re-computed once the parameters are updated in the optimization process. The first frame exception is due to the reason that the drawn parts usually well match the boundary edges at the first frame, where the mechanism is not in motion. We extract edges from the video frames using the classic canny edge detector [Canny 1986].

Stochastic optimization: Since it is technically infeasible to construct an analytical function to map the joint type and part shape parameters to the silhouettes of the parts in motion, we minimize Eq. 7 with stochastic optimization techniques, where joint types are discrete random variables and part shape parameters are continuous random variables. Furthermore, due to the reason that joint types can be initially guessed with sound prior knowledge in mechanism theory (Section 6.1), each joint has a small set of possible types. Therefore, we separate the optimization process into two stages: the joint types are first exhaustively searched, then, the set of joint types with smallest silhouette matching energies are retained for the subsequent shape parameter optimization. Specifically, we will keep two sets of joint types with two smallest silhouette matching energies and optimize the shape parameters for them, respectively, if the difference of their silhouette matching energies is below a threshold. Finally, the joint types and the shape parameters leading to the smallest matching energy are accepted as the optimal solution.

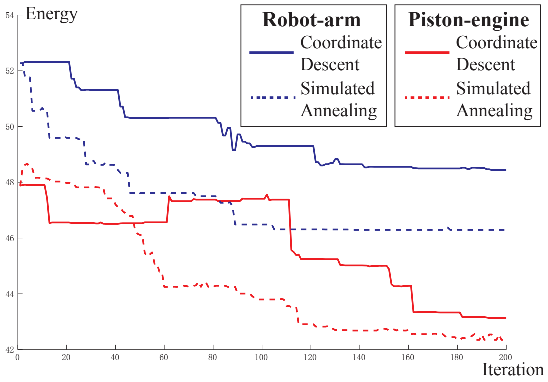
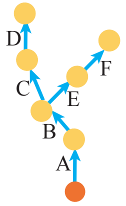


Figure 10: *The convergence curves of the simulated annealing algorithm in our approach for two test models: Piston-engine and robot arm.*

Joint type search: The search for joint types starts with organizing the connection graph, with possible loops, of parts into a tree-like representation as in [Zhu et al. 2012], where a node denotes a part and an edge denotes a joint. First, our algorithm traverses the graph in a breadth-first manner to form an array of the nodes, starting from the driving part. Second, we detect the shortest loop for the node with the smallest array index. The loop can pass driving parts, but the driving parts are not included in the formed loop. An edge can only appear in a loop once, and we group the parts and the joints in the loop into a virtual part. All the nodes in the virtual part should be marked as visited and ignored in the subsequent loop detection. However, the virtual part itself is put at the position of the node with the smallest index and other possible loops can be detected from the virtual part again. Third, if there is no possible loops in the graph, our algorithm organizes the virtual or real parts into a tree representation of the mechanism, choosing the driving part as the root node. For a mechanism that might have more than one driving parts, the driving parts are specially treated as one grouped driving part in current implementation to ease the tree representation.

Afterwards, our algorithm exhaustively searches for joint types on one path from the root to a leaf node, and then proceed to other paths in the tree while ignoring the joints already searched. As illustrated in the right inset, supposing each edge has two choices of joint type, the joint type search for the whole tree can be done in 20 times, since the shared edges only need to be searched once. For instance, after the path formed by edges $\{A, B, C, D\}$ is searched, later the shared two edges (i.e., A, B) do not need to be searched again. In this way, this search is more efficient than a purely combinatorial method, which is 64 times in this case. For a virtual part, its number of possible joint types is the product of the numbers of possible join types of all its grouped joints, and it is simulated by the joint constraint



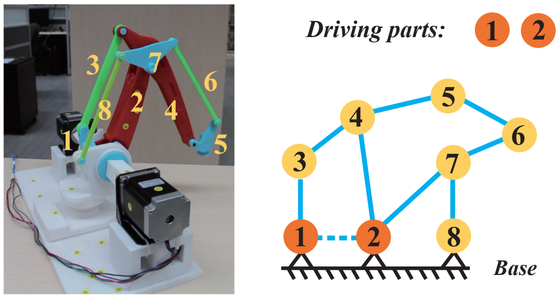
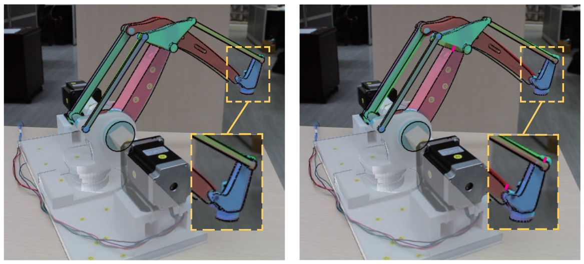


Figure 12: *Left: The robot arm mechanism. Right: Its connection graph. It parts are labeled by numbers. The dashsed line between two driving gears indicates that the two driving parts are interconnected in this mechanism.*



Simulated annealing Coordinate descent

Figure 13: *Left: the shape parameter optimization result by the simulated annealing algorithm in our method. Right: the optimization result by the coordinate descent method. The red lines indicate the uncorrected mismatches by the coordinate descent method.*

equations and rigidity constraint of each part as in [Coros et al. 2013]. In the worst case, all the parts might be grouped into one virtual part and our search will be degenerated into an exhaustive search method.

Shape parameter optimization:The shape parameters are optimized through the simulated annealing algorithm, inspired by the shape parameter optimization algorithm in [Zhu et al. 2012]. We minimize a Boltzmann-like objective function as follows:

$$f(x) = \exp\left(-\frac{\tilde{E}(\mathcal{P})}{T}\right) \tag{8}$$

where $\tilde{E}(\mathcal{P})$ is the cost function computed for $E(\mathcal{J}, \mathcal{P})$ by fixing the joint parameters found at the exhaustive joint type search step. A new state $\mathcal{P}'$ is proposed at each iteration and accepted with a probability calculated below.

$$\alpha(\mathcal{P}'|\mathcal{P}) = \min\left(1, \frac{f(\mathcal{P}')}{f(\mathcal{P})}\right). \tag{9}$$

The annealing parameter T is initialized to be 1, and its value is decreased by a factor of 0.9 every 50 iterations.

Our random shape parameter generation procedure to explore the configuration space $\mathcal{P}$ works in two steps. First, we pick a mechanical part and one of its shape parameters with a uniform distribution. Second, the new shape parameter is sampled from a Gaussian distribution $[\mathcal{N}(\mathbf{s}, \delta_s^2)]$, where $\mathbf{s}$ is the current value of the parameter, and the distribution variance, δ_s, is set to $\frac{1}{10}$ of its value after the modeling step, since we assume the modeling result after alignment optimization is reasonably close to the optimal shape parameters that influence the final motion.

The shape parameters for each part are illustrated in Fig.9. We allow the driving gear to translate to control the location of the kinematic chain in the mechanism, which is denoted by XYZ in Fig. 9. The shape parameters for a gear and a cylindrical rod are the same, where the length of the cylinder is denoted by thickness in the case of gear. For a crank part, its handle length indicates the distance between the slider center and the crank center in the local Z axis, which influences the movement of the slider. All the parts in our system has a local coordinate system, and we choose the Z axis to point to their child parts. Besides the shape parameters, as in [Zhu et al. 2012], all the parts are allowed to move along the local Z axis of their parent parts.

Discussion: We only optimize the motion parameters with respect to a video clip of the mechanism recorded from a single viewpoint. In general, the information from a single view point is insufficient to determine its 3D counterpart. However, in our case since the parts in the mechanism move with rigid motion, their motion information can be robustly recovered by the edge features from a single viewpoint as investigated in model-based object tracking [Chin and Dyer 1986; Kragic and Christensen 2003], given the accurately reconstructed camera and 3D shape information. The main reason is that the reconstructed part shapes already reasonably match the sparse 3D point cloud, and the shape parameters control the global shape of the parts, which means their changes directly influence how the rigid motion of the parts is matched with that in video.

7 Experimental Results

We have implemented the system on a desktop PC with Intel I5 CPU (2.67G HZ) and 8G memory. The modeling algorithm has been tested on six mechanism objects ranging from simple mechanical toys to small-scale real mechanisms. The statistics of each reconstructed mechanism are listed in Table 1 .

Interactive modeling: We tested our modeling system on four real mechanisms, the piston-engine in Fig. 1 and the other three mechanisms, simple three-gears, crank-block and robot arm, in the first three rows in Fig. 11. For the two relatively simple mechanisms (simple three-gears, and crank-block), their interactive modeling time is below 15 minutes. The piston-engine mechanism has 31 parts, and its crank part is of free-from profile curve. It is relatively complicated and its modeling time is around 25 minutes.

As illustrated in Fig. 12, the robot arm mechanism has 8 parts. A four-bar linkage is used to transfer the motion from the two driving gears to the longest arm on the top of the mechanism. We write a program to control the two electric motors, i.e. the two metal boxes in Fig. 12, to drive the two revolute joints of the four bar linkage. Therefore, the motion of its two driving parts can be directly determined; the motion estimation for the driving parts can be avoided for this specific case. The motors and the links between them and the driving parts are not included in the modeling results for the clarity purpose.

The last two rows of Fig. 11 illustrate the modeling results of two mechanical toys. The first mechanical toy has many parts, but most of them are generalized cylinder shapes. Thus, it is relatively simple to our modeling system to obtain the 3D modeling result. It takes only 20 minutes to obtain the modeling result shown in Fig. 11(b). Another mechanical toy modeling result is shown the last row of Fig. 11, where the wings of the windmill are modeled by flat cubes.

Stochastic optimization: Fig. 10 illustrates the convergence of the simulated annealing algorithm to search for the shape parameters. The number of iterations is approximately proportional to the num-

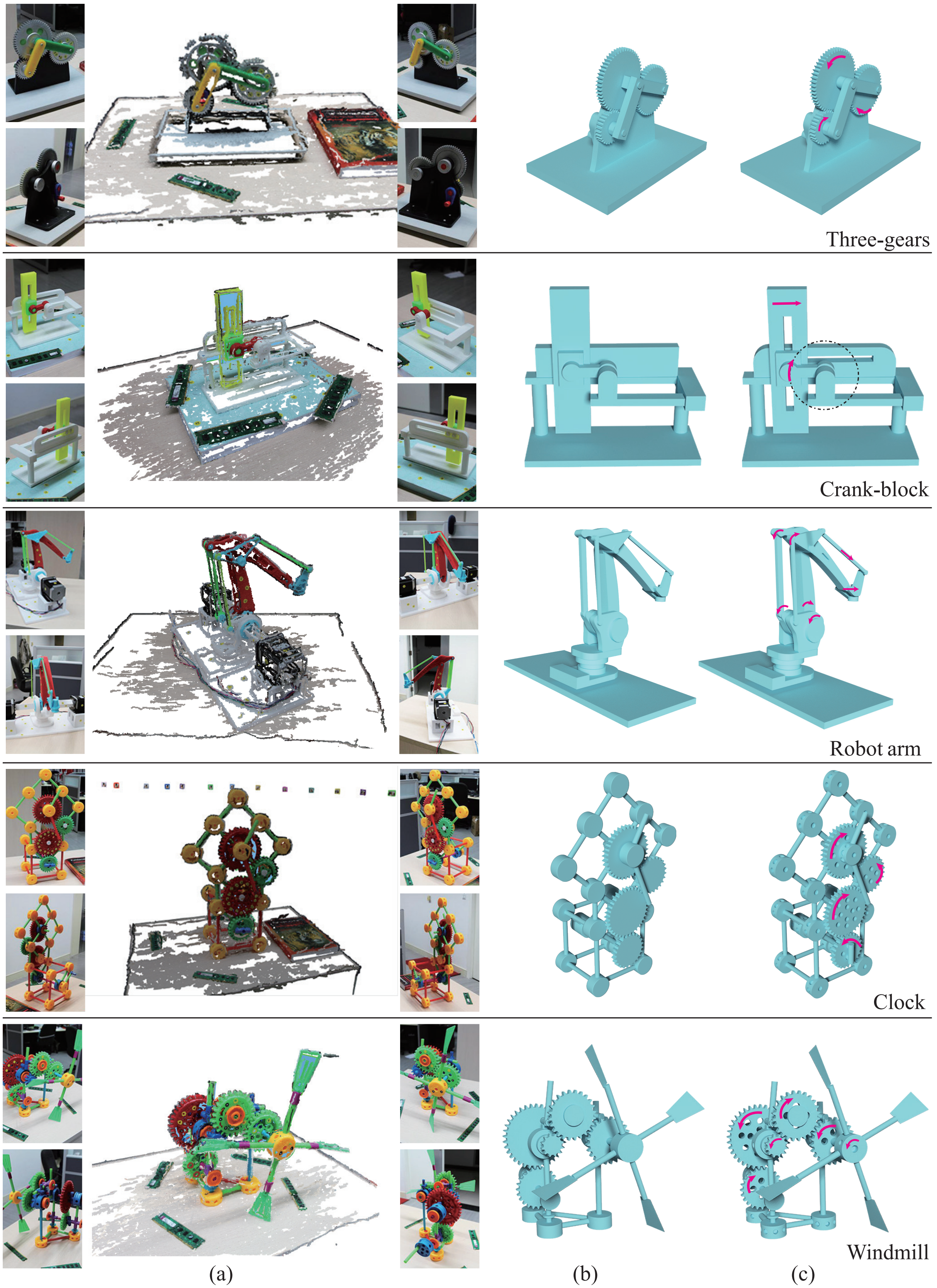
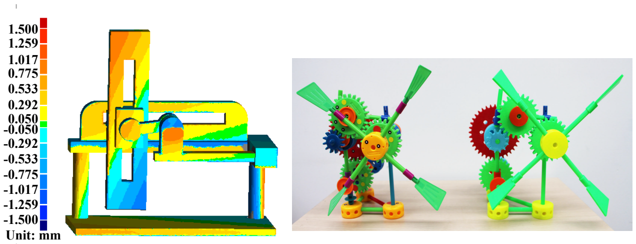


Figure 11: *Mechanism models created using our system. (a) Multi-view images with sparse point cloud data. (b) Interactive modeling results. (c) Refined results. The holes and bevels are generated in Blender software within 10 minutes.*

Name	#Parts	Parts in Motion	Modeling	Alignment	Joint guess	Stochastic optimization
Piston-engine	31	10	25m	1.38s	20	198.4s
Three-gears	13	8	12m	0.53s	16	180.3s
Crank-block	11	6	15m	0.39s	8	170.4s
Robot-arm	17	8	20m	0.59s	128	205.7s
Clock	49	12	20m	1.16s	34	228.2s
Windmill	57	15	30m	0.87s	134	210.6s

Table 1: *Statistics of the reconstructed models. #parts denotes the total number of the parts of the reconstructed mechanism. All the times were measured in seconds.*



(a) (b)

Figure 14: *Validation results. (a) The visualized distance errors for the crank-block model. (b) The real windmill toy (left) and its 3D printed replication (right). Please see the accompanying video for their animation comparison.*

ber of shape parameters to be optimized. Therefore, for the piston-engine model of 21 shape parameters, the number of its iterations, 150, before convergence is larger than that for the robot arm mechanism with 15 shape parameters. Fig. 7 illustrates the corrected mismatches between the part silhouettes and the part edges in one video frame after stochastic optimization.

We also compared the simulated annealing algorithm used in our method with coordinate descent optimization algorithm, where the gradient of each shape parameter is numerically computed. The reason we chose coordinate descent method over gradient decent method is that the geometric constraints between parts are mutually influenced and thus they are hard to maintain correctly if all the shape parameters are optimized simultaneously. Fig. 10 shows that the simulated annealing method can achieve lower silhouette matching energies than the coordinate descent method for this highly nonlinear mapping from part shape parameters to 2D silhouettes, due to its ability to avoid the local minimum. Also, the side-by-side comparison of the silhouette matching results of the robot arm model is shown in Fig. 13.

Table 1 lists the number of guesses in the exhaustive search of joint types. For simple toy examples whose joint types are dominated by gear-2-gear contact, the joint type choices for these examples are mainly the connection between gears to cylinders for the purpose of installation. The windmill toy has a long kinematic chain of 10 parts through the driving gear to the final motion of windmill, which contain 3 gear-2-gear contacts to transmit the motion. In our algorithm, the joint type search times for the windmill toy is 128 for the chain and 6 for the other 3 joints between gear and installation cylinders, totally 134, which can be finished in around 30 seconds on our quad-core computer. According to our loop detection algorithm, all the parts of the robot arm, except #8, are in one loop as illustrated in Fig. 12. Since the joints in the loop can only be fixed or revolute joints with respect to their cylindrical intersection, 128 times of joint type search is needed for this model.

Validation: The validation study, as shown in Fig. 14a, is to illustrate the quantitative error between our modeling result and the real mechanism shown in the second row of Fig. 11. The real mechanism was fabricated using a 3D printer, and then we took photos to reconstruct its 3D model. Thus, the reconstructed 3D model can be directly compared to the original model to study the modeling accuracy of our approach. The dimension of the real mechanism is $180mm \times 180mm \times 75mm$, and the surface approximation error as shown in Fig. 14a is small, usually below $1mm$. We also fabricated a mechanism toy model using a 3D printer to validate the motion parameters estimation result. The accompanying video shows that the estimated motion parameters are sufficiently accurate to reproduce very similar motion to the original model.

8 Discussion and Conclusion

In this paper, we present a modeling system to create a variety of 3D mechanisms from multi-view images. It consists of two main steps: The first step is a stroke-based interactive interface to create the 3D shapes of mechanism parts through the integration of the edge information in images and the sparse 3D point cloud reconstructed from the multi-view images. Using a pre-recorded video clip of the motion of a mechanism, the second step estimates the joint types between the parts and further optimizes their shape parameters to reproduce the mechanism motion recorded in the video.

Limitations and future work: The type of a joint or the relative motion between a pair of parts is constrained according to the part types and their intersection surface in mechanism theory. That is the reason we design the rules to guess joint types. However, the set of rules in our current system only supports the four main types of joints. Ideally, they can be further expanded to cover more types of joints in mechanism design, such as geneva and escape mechanisms. Also, the part type information is manually specified via our interactive user interface; an automatic or semi-automatic part type recognition algorithm can help to reduce such manual efforts.

The joint type search in our current method performs exhaustive search for every combination of joint types of a virtual part, which is less optimized, indeed. We plan to address this issue by developing pruning algorithms to remove infeasible combinations as soon as possible in the search process. For example, if one joint of a four-bar linkage is fixed, then the part rigid constraints are violated in motion. Thus, all the joint type combinations that have that fixed joint should be avoided. To this end, a smart algorithm to do early judges on whether a joint type configuration is feasible would help to improve the search efficiency.

In the future, we also plan to integrate sketch-based interface for the modeling of 3D curves and free-form shapes to create more complex mechanisms, breaking the modeling limitation of generalized cylinders or cubes. Furthermore, to improve the accuracy of the reconstructed mechanism models, we also plan to investigate how to apply our interface to reconstruct parts from dense 3D point clouds from a 3D scanner or depth data captured by off-the-shelf depth cameras.

Appendix

The Eq. 2 can be derived by the expansion of the Eq. 1. For the purpose of clarity, we drop the subscript i, the index of an image, in the derivation. Let us first define the entries for the intrinsic $\mathbf{K}$ and extrinsic rotation matrix $\mathbf{R}$ as:

$$\mathbf{K} = \begin{bmatrix} f_1 & 0 & c_x \\ 0 & f_2 & c_y \\ 0 & 0 & 1 \end{bmatrix}, \mathbf{R} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \tag{10}$$

while the translation is denoted by $\mathbf{T} = \{t_1, t_2, t_3\}^t$.

Given an input 3D point $\{X, Y, Z\}$, its 2D projection in homogeneous coordinate $\{\bar{x}, \bar{y}, \bar{z}\}$ can be derived according to Eq. 1:

$$\begin{bmatrix} \bar{x} \\ \bar{y} \\ \bar{z} \end{bmatrix} = \begin{bmatrix} m_{11}X + m_{12}Y + m_{13}Z + m_{14} \\ m_{21}X + m_{22}Y + m_{23}Z + m_{24} \\ r_{31}X + r_{32}Y + r_{33}Z + t_3 \end{bmatrix} \tag{11}$$

where:

$$\begin{aligned} m_{11} = f_1 r_{11} + c_x r_{31}, m_{12} = f_1 r_{12} + c_x r_{32} \\ m_{13} = f_1 r_{13} + c_x r_{33}, m_{14} = f_1 t_1 + c_x t_3 \\ m_{21} = f_2 r_{21} + c_y r_{31}, m_{22} = f_2 r_{22} + c_y r_{32} \\ m_{23} = f_2 r_{23} + c_y r_{33}, m_{24} = f_2 t_2 + c_y t_3 \end{aligned} \tag{12}$$

Since the pixel coordinate $\{x, y\}$ is related to the homogenous coordinate by $x\bar{z} = \bar{x}$ and $y\bar{z} = \bar{y}$, Eq. 11 implies two linear equations of $\{X, Y\}$, which can be easily solved to obtain:

$$\begin{aligned} X = \mathbf{f}_X(Z) = a_x Z + d_x \\ Y = \mathbf{f}_Y(Z) = a_y Z + d_y \end{aligned} \tag{13}$$

The coefficients in the equation is detailed as follows:

$$\begin{aligned} a_x = \frac{a_x^1 - a_x^2}{n_1 - n_2}, \;\; d_x = \frac{d_x^1 - d_x^2}{n_1 - n_2} \\ a_y = \frac{a_y^1 - a_y^2}{n_1 - n_2}, \;\; d_y = \frac{d_y^1 - d_y^2}{n_1 - n_2} \end{aligned} \tag{14}$$

where

$$\begin{aligned} a_x^1 = (m_{22} - r_{32}y)(m_{13} - r_{33}x), a_x^2 = (m_{12} - r_{32}x)(m_{23} - r_{33}y) \\ d_x^1 = (m_{22} - r_{32}y)(m_{14} - t_3x), d_x^2 = (m_{12} - r_{32}x)(m_{24} - t_3y) \\ a_y^1 = (m_{11} - r_{31}x)(m_{23} - r_{33}y), a_y^2 = (m_{13} - r_{33}x)(m_{21} - r_{31}y \\ d_y^1 = (m_{11} - r_{31}x)(m_{24} - t_3y), d_y^2 = (m_{14} - t_3x)(m_{21} - r_{31}y) \\ n_1 = (m_{11} - r_{31}x)(m_{22} - r_{32}y), n_2 = (m_{12} - r_{32}x)(m_{21} - r_{31}y) \end{aligned} \tag{15}$$

References

Abbasinejad, F., Joshi, P., and Amenta, N. 2011. Surface patches from unorganized space curves. *Computer Graphics Forum 30*, 5, 1379–1387.

Agarwal, S., Furukawa, Y., Snavely, N., Simon, I., Curless, B., Seitz, S. M., and Szeliski, R. 2011. Building rome in a day. *Commun. ACM 54*, 10 (Oct.), 105–112.

Ahmadabadian, A. H., Robson, S., Boehm, J., Shortis, M., Wenzel, K., and Fritsch, D. 2013. A comparison of dense matching algorithms for scaled surface reconstruction using stereo camera rigs. *ISPRS Journal of Photogrammetry and Remote Sensing 78*, 0, 157 – 167.

Bächer, M., Bickel, B., James, D. L., and Pfister, H. 2012. Fabricating articulated characters from skinned meshes. *ACM Trans. Graph. 31*, 4 (July), 47:1–47:9.

Bae, S.-H., Balakrishnan, R., and Singh, K. 2008. Ilovesketch: As-natural-as-possible sketching system for creating 3d curve models. In *UIST'08*, 151–160.

Bessmeltsev, M., Wang, C., Sheffer, A., and Singh, K. 2012. Design-driven quadrangulation of closed 3d curves. *ACM Trans. Graph. 31*, 6 (Nov.), 178:1–178:11.

Bey, A., Chaine, R., and Raphael, M. 2011. Reconstruction of consistent 3d cad models from point cloud data using a priori cad models. In *Proceedings of ISPRS*, vol. 12.

Canny, J. 1986. A computational approach to edge detection. *IEEE Trans. Pattern Anal. Mach. Intell. 8*, 6, 679–698.

Cao, Y.-P., Ju, T., Fu, Z., and Hu, S.-M. 2014. Interactive image-guided modeling of extruded shapes. *Comput. Graph. Forum 33*, 7, 101–110.

Ceylan, D., Li, W., Mitra, N. J., Agrawala, M., and Pauly, M. 2013. Designing and fabricating mechanical automata from mocap sequences. *ACM Trans. Graph. 32*, 6, 186.

Chen, X., Kang, S. B., Xu, Y.-Q., Dorsey, J., and Shum, H.-Y. 2008. Sketching reality: Realistic interpretation of architectural designs. *ACM Trans. Graph. 27*, 2 (May), 11:1–11:15.

Chen, T., Zhu, Z., Shamir, A., Hu, S.-M., and Cohen-Or, D. 2013. 3-sweep: extracting editable objects from a single photo. *ACM Trans. Graph. 32*, 6, 195.

Chin, R. T., and Dyer, C. R. 1986. Model-based recognition in robot vision. *ACM Comput. Surv. 18*, 1, 67–108.

Coros, S., Thomaszewski, B., Noris, G., Sueda, S., Forberg, M., Sumner, R. W., Matusik, W., and Bickel, B. 2013. Computational design of mechanical characters. *ACM Trans. Graph. 32*, 4 (July), 83:1–83:12.

Eitz, M., Hays, J., and Alexa, M. 2012. How do humans sketch objects? *ACM Trans. Graph. 31*, 4 (July), 44:1–44:10.

Eitz, M., Richter, R., Boubekeur, T., Hildebrand, K., and Alexa, M. 2012. Sketch-based shape retrieval. *ACM Trans. Graph. 31*, 4 (July), 31:1–31:10.

Furukawa, Y., and Ponce, J. 2007. Accurate, dense, and robust multi-view stereopsis. In *CVPR '07*, 1–8.

Gal, R., Sorkine, O., Mitra, N. J., and Cohen-Or, D. 2009. iwires: An analyze-and-edit approach to shape manipulation. *ACM Trans. Graph. 28*, 3 (July), 33:1–33:10.

Hartley, R. I., and Zisserman, A. 2004. *Multiple View Geometry in Computer Vision*, second ed. Cambridge University Press, ISBN: 0521540518.

Huang, J., and Menq, C.-H. 2002. Automatic cad model reconstruction from multiple point clouds for reverse engineering. *J. Comput. Inf. Sci. Eng. 2*, 3, 160–170.

Igarashi, T., Matsuoka, S., and Tanaka, H. 1999. Teddy: A sketching interface for 3d freeform design. In *Proc. of SIGGRAPH'99*, 409–416.

Jorge, N., and Stephen, W. 2006. *Numerical Optimization*. Springer-Verlag New York.

KALOGERAKIS, E., CHAUDHURI, S., KOLLER, D., AND KOLTUN, V. 2012. A probabilistic model for component-based shape synthesis. *ACM Trans. Graph. 31*, 4 (July), 55:1–55:11.

KRAEVOY, V., SHEFFER, A., SHAMIR, A., AND COHEN-OR, D. 2008. Non-homogeneous resizing of complex models. *ACM Trans. Graph. 27*, 5 (Dec.), 111:1–111:9.

KRAGIC, D., AND CHRISTENSEN, H. I. 2003. Confluence of parameters in model based tracking. In *ICRA*, IEEE, 3485–3490.

LI, W., AGRAWALA, M., CURLESS, B., AND SALESIN, D. 2008. Automated generation of interactive 3d exploded view diagrams. *ACM Trans. Graph. 27*, 3.

LI, Y., WU, X., CHRYSATHOU, Y., SHARF, A., COHEN-OR, D., AND MITRA, N. J. 2011. Globfit: Consistently fitting primitives by discovering global relations. *ACM Trans. Graph. 30*, 4 (July), 52:1–52:12.

LOWE, D. G. 2004. Distinctive image features from scale-invariant keypoints. *Int. J. Comput. Vision 60*, 2, 91–110.

MITRA, N., WAND, M., ZHANG, H. R., COHEN-OR, D., KIM, V., AND HUANG, Q.-X. 2013. Structure-aware shape processing. In *SIGGRAPH Asia 2013 Courses*, SA '13, 1:1–1:20.

MITRA, N. J., YANG, Y.-L., YAN, D.-M., LI, W., AND AGRAWALA, M. 2013. Illustrating how mechanical assemblies work. *Commun. ACM 56*, 1 (Jan.), 106–114.

NEALEN, A., IGARASHI, T., SORKINE, O., AND ALEXA, M. 2007. Fibermesh: Designing freeform surfaces with 3d curves. *ACM Trans. Graph. 26*, 3 (July).

OLSEN, L., SAMAVATI, F. F., SOUSA, M. C., AND JORGE, J. A. 2009. Sketch-based modeling: A survey. *Computers & Graphics 33*, 1, 85–103.

OWADA, S., NIELSEN, F., OKABE, M., AND IGARASHI, T. 2004. Volumetric illustration: Designing 3d models with internal textures. *ACM Trans. Graph. 23*, 3 (Aug.), 322–328.

OWADA, S., NIELSEN, F., NAKAZAWA, K., AND IGARASHI, T. 2007. A sketching interface for modeling the internal structures of 3d shapes. In *ACM SIGGRAPH 2007 Courses*, SIGGRAPH '07.

SADRI, B., AND SINGH, K. 2014. Flow-complex-based shape reconstruction from 3d curves. *ACM Trans. Graph. 33*, 2 (Apr.), 20:1–20:15.

SCHMIDT, R., WYVILL, B., SOUSA, M. C., AND JORGE, J. A. 2007. Shapeshop: Sketch-based solid modeling with blobtrees. In *ACM SIGGRAPH 2007 Courses*, SIGGRAPH '07.

SCHMIDT, R., KHAN, A., SINGH, K., AND KURTENBACH, G. 2009. Analytic drawing of 3d scaffolds. *ACM Trans. Graph. 28*, 5 (Dec.), 149:1–149:10.

SEITZ, S., CURLESS, B., DIEBEL, J., SCHARSTEIN, D., AND SZELISKI, R. 2006. A comparison and evaluation of multi-view stereo reconstruction algorithms. In *CVPR'06*, vol. 1, 519–528.

SHAO, T., LI, W., ZHOU, K., XU, W., GUO, B., AND MITRA, N. J. 2013. Interpreting concept sketches. *ACM Trans. Graph. 32*, 4, 56:1–56:10.

SHEN, C.-H., FU, H., CHEN, K., AND HU, S.-M. 2012. Structure recovery by part assembly. *ACM Trans. Graph. 31*, 6 (Nov.), 180:1–180:11.

SINHA, S. N., STEEDLY, D., SZELISKI, R., AGRAWALA, M., AND POLLEFEYS, M. 2008. Interactive 3d architectural modeling from unordered photo collections. *ACM Trans. Graph. 27*, 5 (Dec.), 159:1–159:10.

SLATER, N., 2011. Mechanisms and mechanical devices sourcebook. McGraw-Hill.

SNAVELY, N., SEITZ, S. M., AND SZELISKI, R. 2006. Photo tourism: exploring photo collections in 3d. *ACM Trans. Graph. 25*, 3, 835–846.

SONG, L., WU, W., GUO, J., AND LI, X. 2013. Survey on camera calibration technique. In *Intelligent Human-Machine Systems and Cybernetics (IHMSC), 2013 5th International Conference on*, vol. 2, 389–392.

SUN, D., ROTH, S., AND BLACK, M. J. 2010. Secrets of optical flow estimation and their principles. In *IEEE CVPR*, 2432–2439.

THOMASZEWSKI, B., COROS, S., GAUGE, D., MEGARO, V., GRINSPUN, E., AND GROSS, M. 2014. Computational design of linkage-based characters. *ACM Trans. Graph. 33*, 4 (July), 64:1–64:9.

VALGAERTS, L., BRUHN, A., MAINBERGER, M., AND WEICKERT, J. 2012. Dense versus sparse approaches for estimating the fundamental matrix. *International Journal of Computer Vision 96*, 2, 212–234.

WANG, Y., XU, K., LI, J., ZHANG, H., SHAMIR, A., LIU, L., CHENG, Z., AND XIONG, Y. 2011. Symmetry hierarchy of man-made objects. *Computer Graphics Forum 30*, 2, 287–296.

WIKIPEDIA, 2015. Involute. `http://en.wikipedia.org/wiki/Involute/`.

WU, C., 2015. Visualsfm: A visual structure from motion system. `http://ccwu.me/vsfm/`.

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*, 3.

XU, K., ZHENG, H., ZHANG, H., COHEN-OR, D., LIU, L., AND XIONG, Y. 2011. Photo-inspired model-driven 3d object modeling. *ACM Trans. Graph. 30*, 4 (July), 80:1–80:10.

XU, K., 0002, H. Z., COHEN-OR, D., AND CHEN, B. 2012. Fit and diverse: set evolution for inspiring 3d shape galleries. *ACM Trans. Graph. 31*, 4, 57.

XU, B., CHANG, W., SHEFFER, A., BOUSSEAU, A., MCCRAE, J., AND SINGH, K. 2014. True2form: 3d curve networks from 2d sketches via selective regularization. *ACM Trans. Graph. 33*, 4 (July), 131:1–131:13.

ZHANG, Z. 2000. A flexible new technique for camera calibration. *IEEE TPAMI 22*, 11, 1330–1334.

ZHENG, Y., FU, H., COHEN-OR, D., AU, O. K.-C., AND TAI, C.-L. 2011. Component-wise controllers for structure-preserving shape manipulation. *Comput. Graph. Forum 30*, 2, 563–572.

ZHENG, Y., CHEN, X., CHENG, M.-M., ZHOU, K., HU, S.-M., AND MITRA, N. J. 2012. Interactive images: Cuboid proxies for smart image manipulation. *ACM Trans. Graph. 31*, 4 (July), 99:1–99:11.

ZHU, L., XU, W., SNYDER, J., LIU, Y., WANG, G., AND GUO, B. 2012. Motion-guided mechanical toy modeling. *ACM Trans. Graph. 31*, 6 (Nov.), 127:1–127:10.