Illustrating How Mechanical Assemblies Work

Niloy J. Mitra[1,2] Yong-Liang Yang[1] Dong-Ming Yan[1,3] Wilmot Li[4] Maneesh Agrawala[5]

[1] KAUST [2] IIT Delhi [3] Univ. of Hong Kong [4] Adobe Systems [5] Univ. of California, Berkeley

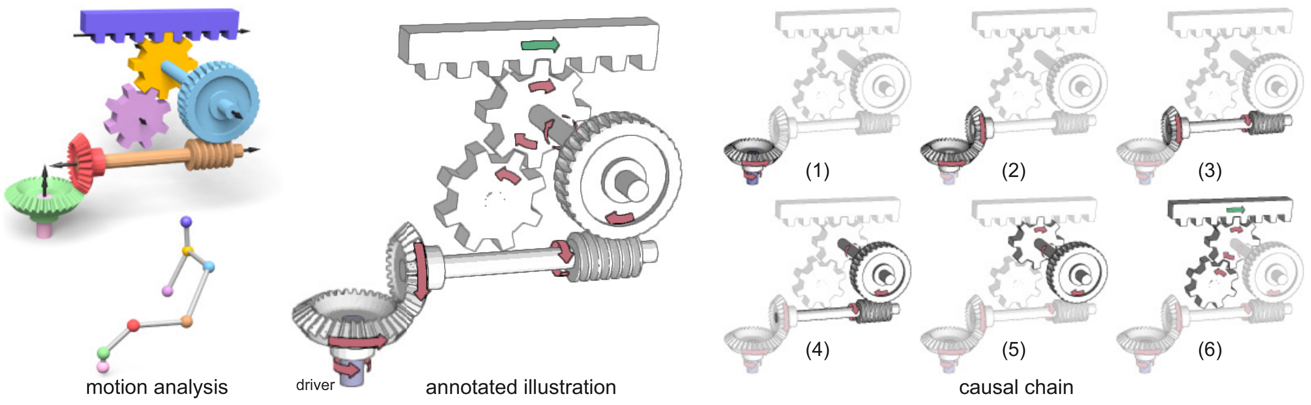

Figure 1: *Given a geometric model of a mechanical assembly, we analyze it to infer how the individual parts move and interact with each other. The relations and motion parameters are encoded as a time-varying interaction graph. Once the driver is indicated by the user, we compute the motion of the assembly and use it to generate an annotated illustration to depict how the assembly works. We also produce a corresponding causal chain sequence to help the viewer better mentally animate the motion.*

Abstract

How things work visualizations use a variety of visual techniques to depict the operation of complex mechanical assemblies. We present an automated approach for generating such visualizations. Starting with a 3D CAD model of an assembly, we first infer the motions of individual parts and the interactions between parts based on their geometry and a few user specified constraints. We then use this information to generate visualizations that incorporate motion arrows, frame sequences and animation to convey the causal chain of motions and mechanical interactions between parts. We present results for a wide variety of assemblies.

Keywords: mechanical assembly, motion depiction, visualization, shape analysis, causal chaining

1 Introduction

. . . all machines that use mechanical parts are built with the same single aim: to ensure that exactly the right amount of force produces just the right amount of movement precisely where it is needed.

(David Macaulay, *The New Way Things Work* [1998])

Mechanical assemblies are collections of interconnected parts such as gears, cams and levers that move in conjunction to achieve a specific functional goal. As Macaulay points out, attaining this goal usually requires the assembly to transform a driving force into movement. For example, the gearbox in a car is a collection of interlocking gears with different ratios that transforms rotational force from the engine into the appropriate revolution speed for the wheels. Understanding how the parts interact to transform the driving force into motion is often the key to understanding how mechanical assemblies work.

There are two types of information that are crucial for understanding this transformation process: 1) the spatial configuration of parts within the assembly and 2) the causal chain of motions and mechanical interactions between parts. While most technical illustrations effectively convey spatial relationships, only a much smaller subset of these visualizations are designed to emphasize how parts move and interact with one another. Analyzing this subset of *how things work* illustrations and prior cognitive psychology research on how people understand mechanical motions suggests several visual techniques for conveying the movement and interactions of parts within a mechanical assembly:

Motion arrows indicate how individual parts move.

Frame sequences show key snapshots of complex motions and highlight the sequence of interactions along the causal chain.

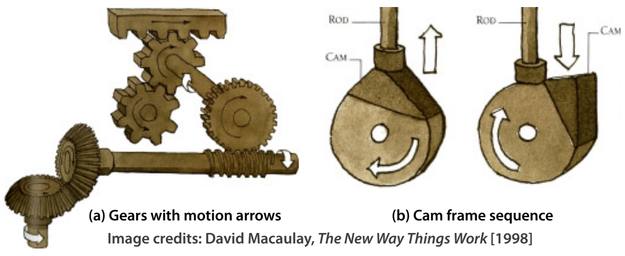
Rod
Cam
Rod
Cam
(a) Gears with motion arrows
(b) Cam frame sequence
Image credits: David Macaulay, The New Way Things Work [1998]

Figure 2: *Hand designed illustrations. These examples show how motion arrows (a) and sequences of frames (b) can help convey the motion and interactions of parts within mechanical assemblies. © Houghton Mifflin Company.*

Animations are sometimes used to show the dynamic behavior of an assembly.

Creating effective how things work illustrations and animations by hand is difficult because a designer must understand how a complex assembly works and also have the skill to apply the appropriate visual techniques for emphasizing the motions and interactions between parts. As a result, well-designed illustrations and animations are relatively uncommon, and the few examples that do exist (e.g., in popular educational books and learning aids for mechanical engineers) are infrequently updated or revised. Furthermore, most illustrations are static and thus do not allow the viewer to inspect an assembly from multiple viewpoints.

In this paper, we present an automated approach for generating how things work visualizations of mechanical assemblies from 3D CAD models. Our approach facilitates the creation of static illustrations and animations from any user-specified viewpoint. We address two main challenges:

Motion and interaction analysis. Most 3D models do not specify how their parts move or interact with each other. Yet, this information is essential for creating visualizations that convey how the assembly works. We present a semi-automatic technique that determines the motions of parts and their causal relationships based on their geometry. With a small amount of user assistance, our approach can successfully analyze a wide range of CAD models.

Automatic visualization. We present algorithms that use the motion and interaction information from our analysis to automatically generate a variety of how things work visualizations, including static illustrations with motion arrows, frame sequences that highlight key snapshots and the causal chain of mechanical interactions, as well as simple animations of the assembly in motion. Figure 1 shows examples of static illustrations generated automatically using our system. Figure 2 shows similar hand-designed illustrations that incorporate motion arrows and frame sequences.

The contributions of our work include 1) a set of design guidelines for generating motion arrows and frame sequences that effectively convey the causal chain of motions and mechanical interactions between parts, 2) an analysis technique that extracts the relevant motion and interaction information from an input 3D model, and 3) automated visualization algorithms that apply our design guidelines based on the results of the analysis.

2 Related Work

Our work builds on three main areas of related work.

Illustrating Motion. Depicting motion in a still image is a challenging task that requires mapping changes in time to locations in image space. Illustrators and artists use a variety of cues to depict motion, including sequences of key poses, stroboscopic effects, motion blur, affine shear (or forward lean), action lines, and arrows [McCloud 1993; Cutting 2002]. Researchers have developed algorithms for adding such motion cues to computer-generated animations of 2D and 3D geometric scenes [Masuch et al. 1999; Kawagishi et al. 2003; Nienhaus and Döllner 2005], time-varying volumetric data [Joshi and Rheingans 2005], video [Collomosse et al. 2005; Kim and Essa 2005; Dony et al. 2005; Goldman et al. 2006] and skeletal motion capture data [Assa et al. 2005; Bouvier-Zappa et al. 2007]. All these techniques assume that the input data directly contains some representation of the motions that must be visualized. For example, Nienhaus and Döllner [2005] illustrate the motion of 3D animations, based on an analysis of specialized scene graphs that encode the structure and motion of the animated scene. Similarly the techniques designed to illustrate volumetric data, video and motion capture, assume that the data itself is time-varying. A key feature of our approach is that it does not require such a representation of the motion as part of the input. Instead we analyze a purely geometric representation of a mechanical assembly to extract the relevant kinematic motions of its parts.

Researchers have also developed a variety of techniques for interpreting and animating 2D sketches of complex physical systems, including mechanical devices, analog circuits, chemical structures. See Davis [2007] for a survey of these techniques.

Shape Analysis. Early research efforts at Felix Klein's celebrated Erlanger Program [1893] and later by Thompson [1917] established the importance of geometry to the study of form and structure. The ideal goal is to extract high level shape semantics and relations between the various parts of shapes by working with their geometric descriptions alone. For general shapes, this remains a difficult, and often an unrealistic task.

However, significant progress has been made in the case of engineered objects. Such objects are designed and manufactured using well established processes and often share distinctive properties. Researchers in the CAD/CAM community have long explored such characteristics to facilitate reconstruction, segmentation and denoising of shapes for reverse engineering [Benkö et al. 2001; Demarsin et al. 2007]. In addition, man-made objects are well suited for a variety of shape analysis techniques including slippage-analysis [Gelfand and Guibas 2004], symmetry detection [Mitra et al. 2006], up-right positioning [Fu et al. 2008], abstraction [Mehra et al. 2009], and structural feasibility [Whiting et al. 2009]. Recently, Xu et al. [2009] employed slippage analysis to segment and categorize joints in man-made models, and used the information for interactive volumetric-cell-based space deformation. Concurrently, Gal et al. [2009] demonstrated that working with a set of 1D feature curves extracted from engineered objects, and preserving their intra- and inter-relations while deforming the shapes lead to an intuitive manipulation framework. This research suggests that some of the characteristic properties of man-made shapes may be closely related to their geometry. In this work, we make use of recent advances in geometric shape analysis to infer relevant attributes of individual parts and their mutual relations for typical mechanical assemblies. We propose a light weight shape analysis system that uses a small set of assumptions and user specifications to automatically infer the motion of such assemblies.

Mechanical Animation. Although most CAD models do not include information about how their parts move and interact, a few commercial CAD packages provide tools that help users create mechanical animations in order to evaluate functional aspects of an assembly design (e.g., *SolidWorks Motion, Solid Edge Motion Simulation*). However, most of these tools still require the user to manually specify information for all (or many) of the assembly parts, including motion parameters and interaction constraints between the parts. In contrast, our approach infers such information directly from geometry with far less user assistance. Furthermore, even after motion parameters have been specified, existing CAD tools do not automatically produce the types of *how things work* visualizations that are the focus of our work.

3 Designing How Things Work Visualizations

Illustrators and engineers have produced a variety of books [Amerongen 1967; Macaulay 1998; Langone 1999; Brain 2001] and websites (e.g. howstuffworks.com) that are designed to show how complex mechanical assemblies work. These illustrations use a number of diagrammatic conventions to highlight the motions and mechanical interactions of parts in the assembly.

Cognitive psychologists have studied how static and multimedia visualizations help people mentally represent and understand the function of mechanical assemblies [Mayer 2001]. For example, Narayanan and Hegarty [1998; 2002] propose a cognitive model for comprehension of mechanical assemblies from diagrammatic visualizations that involves 1) constructing a spatial representation of the assembly and then 2) constructing a model of the causal chain of motions and interactions between the parts. They also suggest a set of high-level design guidelines for creating *how things work* visualizations that facilitate these two steps.

Researchers in computer graphics have concentrated on refining and implementing many of the design guidelines aimed at assisting the first step of the comprehension process. Algorithms for creating exploded views [McGuffin et al. 2003; Bruckner and Groller 2006; Li et al. 2008], cutaways [Seligmann and Feiner 1991; Li et al. 2007; Burns and Finkelstein 2008] and ghosted views [Feiner and Seligmann 1992; Viola et al. 2004] of complex objects apply illustrative conventions to emphasize the spatial locations of the parts with respect to one another. Since our focus in this work is on depicting motions and mechanical interactions between parts, we concentrate the following discussion on visual techniques that facilitate the second step of the comprehension process.

Helping Viewers Construct the Causal Chain

In an influential treatise examining how people predict the behavior of mechanical assemblies from static visualizations, Hegarty [1992] found that people reason in a step-by-step manner, starting from an initial *driver* part and tracing forward through each subsequent part along a causal chain of interactions. At each step, people infer how the relevant parts move with respect to one another and then determine the subsequent action(s) in the causal chain. Even though all parts may be moving at once in real-world operation of the assembly, people mentally animate the motions of parts one at a time in causal order.

Although animation might seem like a natural approach for visualizing mechanical motions, in a meta-analysis of previous studies comparing animations to informationally equivalent sequences of static visualizations, Tversky et al. [2002] found no benefit for animation. Our work does not seek to engage in this debate between static versus animated illustrations. Instead we aim to support both types of visualizations with our tools. We consider both static and animated visualizations in our analysis of design guidelines.

Effective how things work illustrations use a number of visual techniques to help viewers mentally animate an assembly.

Use arrows to indicate motions of parts. Many illustrations include arrows that indicate how each part in the assembly moves. In addition to conveying the motion of individual parts, such arrows can also help viewers understand the specific functional relationships between parts [Hegarty 2000; Heiser and Tversky 2006]. Placing the arrows near contact points between parts that interact along the causal chain can help viewers better understand the causal relationships.

Highlight causal chain step-by-step. In both static and animated illustrations, highlighting each step in the causal chain of actions helps viewers mentally animate the assembly by explicitly indicating the sequence of interactions between parts. Static illustrations often depict the causal chain using a sequence of key frames that correspond to the sequence of steps in the chain. Each key frame highlights the transfer of movement between a set of touching parts, typically by rendering those parts in a different style from the rest of the assembly. In the context of animated visualizations researchers have shown that adding signaling cues that sequentially highlight the steps of the causal chain improve comprehension compared to animations that do not include such cues [Hegarty et al. 2003; Kriz and Hegarty 2007].

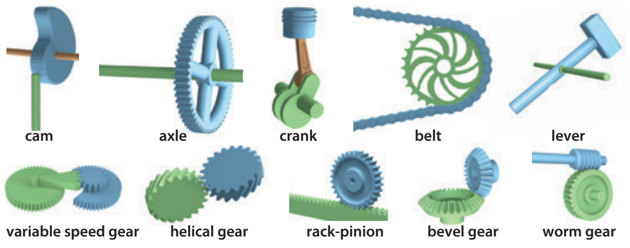

Figure 3: *Typical joints and gear configurations encountered in mechanical assemblies, and handled in our system. While most types we can automatically detect and handle, we require the user to mark some parts, for example a lever (see accompanying video).*

Highlight important key frames of motions. The motions of most parts in mechanical assemblies are periodic. However, in some of these motions, the angular or linear velocity of a part may change during a single period. For example, the pistons in the assembly shown in Figure 12 move up and down the cylinder during a single period of motion. To depict such complex motions, static illustrations sometimes include key frames that show the configuration of parts at the critical instances in time when the angular or linear velocity of a part changes. Inserting one additional key frame between each pair of critical instances can help clarify how the parts move from one critical instance to the next.

4 System Overview

We present an automated system for generating how things work visualizations that incorporate the visual techniques described in the previous section. The input to our system is a polygonal model of a mechanical assembly that has been partitioned into individual parts. Our system deletes hanging edges and vertices as necessary to make each part 2-manifold. We assume that touching parts are modeled correctly, with no self-intersections beyond a small tolerance. As a first step, we perform an automated motion and interaction analysis of the model geometry to determine the relevant motion parameters of each part, as well as the causal chain of interactions between parts. This step requires the user to specify the driver part for the assembly and the direction in which the driver moves. Using the results of the analysis, our system allows users to generate a variety of static and animated visualizations of the input assembly from any viewpoint. The next two sections present our analysis and visualization algorithms in detail.

5 Motion and Interaction Analysis

The analysis phase computes the type of each part and how the parts move and interact with each other within the assembly. We encode this information as an *interaction graph* (see Figure 4) in which nodes represent parts, and edges represent mechanical interactions between touching parts. Each node stores parameters that define the type (e.g. axle, gear, rack, etc.) and motion attributes (e.g., axis of rotation or translation, range of possible motion) of the corresponding part. We use shape and symmetry information to infer both the part type and part motion directly from the 3D model. Each edge in the graph is also typed to indicate the kind of mechanical interaction between the relevant parts. Our system handles all of the common joint and gear configurations shown in Figure 3, including cam and crank mechanisms, axles, belts, and a variety of gear interactions.

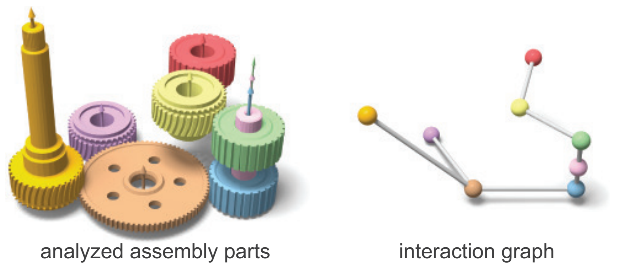

Figure 4: *(Left) Parts of an assembly with automatically detected axes of rotation or symmetry, and (right) the corresponding interaction graph. Graph edges encode the types of joints.*

To construct the interaction graph, we rely on two high-level insights: 1) the motions of many mechanical parts are related to their geometric properties, including self-similarity, symmetry; 2) the different types of joint and gear configurations are often characterized by the specific spatial relationships and geometric attributes of the relevant parts. Based on these insights, we propose a two-stage process for constructing the interaction graph. First, in the *part analysis* stage, we examine each part and compute a set of candidate rotation and translation axes based on the self-similarity and symmetry of the part. We also compute key geometric attributes, including radius, type of side profile (cylindrical, conical), pitch, teeth count, as applicable. Complex parts are segmented into simpler subparts, each with its associated radius, side profile type, etc. Then, in the *interaction analysis* stage, we analyze the axes and attributes of each pair of touching parts and classify the type of joint or gear configuration between those parts. Based on this classification, we create the interaction graph nodes, store any relevant motion parameters, and link touching parts with the appropriate type of edge.

5.1 Part Analysis

For each part, we estimate its (potential) axes, along with attributes like side profile, radius, teeth count, and pitch as applicable. Based on these attributes we classify the type of the part. In Section 5.2, we explain how we use these attributes to determine how motion is transmitted across parts in contact with one another. In this interaction analysis phase we also use the motion information to refine our classification of the part type.

Assumptions. Inferring inter-relations between parts and their motion directly from the geometry of a static configuration of a mechanical assembly can be ambiguous. However, making simple assumptions about the assembly allows us to semi-automatically rig up its motion. We assume that parts usually rotate about a symmetry or dominant axis, translate along translational symmetry directions, or engage in screw motion along a helical axis, as applicable. Unsymmetric parts are assumed to remain fixed. Such parts typically constitute the support structures of mechanical assemblies. Levers, belts and chains have few dominating geometric characteristics. Thus, we expect the user to identify such parts. Our system automatically identifies all the other part types shown in Figure 3.

Symmetry detection. Most machine parts are regularly structured and we use symmetry analysis to identify translational or rotational axes of such parts as well as their degrees of freedom. A part P is said to be symmetric if it remains invariant under some transformation T_j, i.e., $T_j(P) = P$. For mechanical assemblies we are only interested in the family of rigid transformations $\{T_j\}$, i.e., translations, rotations, and their combinations. We use a simplified variant of Hough transform based voting scheme [Mitra et al. 2006] to look for translational and rotational symmetries, while ignoring reflective ones. The algorithm also handles partial symmetries, which are present in certain parts like variable-speed gears (see Figure 3). For rotationally symmetric parts, we obtain their rotational axis $\mathbf{a}$ and teeth count based on the order of their symmetry. Similarly, for parts with discrete translational symmetry, we obtain their movement direction $\mathbf{t}$ and teeth width, e.g., for a gear rack; while for helical parts we compute the axis direction $\mathbf{t}$ along with the pitch of the screw motion.

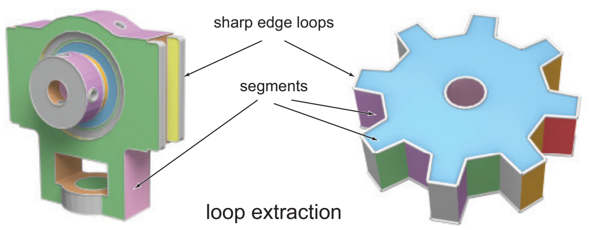

Figure 5: *Parts are segmented using edges with high dihedral angles across adjacent faces to form sharp edge loops.*

Sharp edge loops. Parts with insignificant symmetry often provide important constraints for the other parts, facilitating or restricting their movements. Even for (partially) symmetric parts we must estimate attributes like gear radius, range of motion, etc. We extract such properties by working with sharp edge loops or creases, that are common 1D feature curves characterizing machine parts (see also [Gal et al. 2009]).

We employ a simple strategy to extract sharp edge loops from polygonal models of mechanical parts. First all the edges with dihedral angle between adjacent faces exceeding threshold θ_s ($40°$ in our implementation), are marked as *sharp*. Then starting from a random seed triangle as a segment, we greedily gather neighboring triangles into the segment using a floodfill algorithm, while making sure not to cross any sharp edge. We repeat this segmentation process until all triangles have been partitioned into segments separated by sharp edges. Segments with only few sharp edges (less than 10 in our experiments) are discarded as insignificant. Boundary loops of the remaining segments are used as sharp edge feature loops for the remaining analysis (see Figure 5). Note that an edge can belong to up to two loops. This simple loop extraction procedure is sufficient to handle clean input geometry of mechanical parts with sharp features like in CAD models. However, for parts with few sharp edges, boundaries of proxy segments computed using variational shape approximation can be used as feature loops [Cohen-Steiner et al. 2004; Mehra et al. 2009].

Next we fit least squares (parts of) circles to the sharp edge loops and based on the residual error we identify circular loops. For each circle loop l_i, besides its center $\mathbf{c}_i$ and radius r_i, we also obtain its (canonical) axis direction $\mathbf{a}_i$ as the normal to its containing plane.

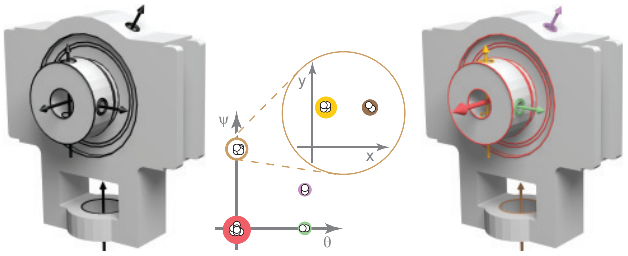

Figure 6: *(Left) Circles fitted to sharp edge feature loops of a part. (Middle) The circles are first grouped based on their axes. Clusters in this space denote parallel axes, which are in turn grouped based on their (projected) centers. (Right) Detected axes are rated based on the corresponding cluster size (denoted by axis thickness).*

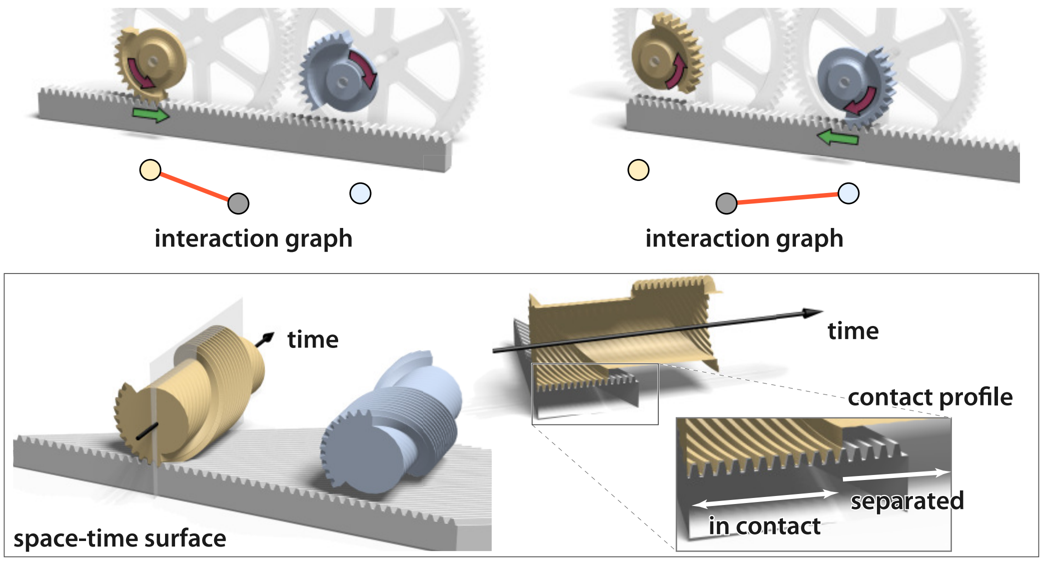

Figure 7: *A time-varying interaction graph. (Top) Graph edges for the rack-and-pinion configuration are active at different times during the motion cycle. (Bottom) We use a space-time construction to find the time interval for each edge. When the contact surfaces separate or self-intersect, we identify end of a cycle. We show the space-time surface analysis for the top-left configuration.*

For each part, all its estimated circle loops $\{l_i\}$ are clustered to extract its potential axes of rotation. First we group loops $\{l_i\}$ with similar axis direction, i.e., we cluster the axes $\{\mathbf{a}_i\}$ in the line space parameterized by their altitude-azimuthal angles. For each detected group, we further partition its elements based on the projection of their centers in a direction along the cluster representative axis, chosen as the centroid of the cluster in the line space. We use a radius of 5 degrees for clustering axes.

This simple grouping scheme works well because mechanical models are largely designed from canonical geometrical primitives, and have their underlying axes of the circle loops well aligned. We consider the clustered directions to be potential axes of a part, if their cluster sizes are significant. We rank the directions based on the number of elements in their corresponding clusters (see Figure 6). These axes directions, both for moving or static parts, often contain important alignment cues for neighboring parts, e.g., as potential shaft locations for axles.

Cylindrical or conical parts have similar rotational symmetry and similar sharp edge loop clusters. To differentiate between them we consider their (approximate) side profiles. We partition rotationally symmetric parts into *cap* and *side* regions. Let $\mathbf{a}_i$ denote the rotational axis of part P_i. Each face $t_j \in P_i$ is classified as a cap face if its normal $\mathbf{n}_j$ is such that $|\mathbf{n}_j \cdot \mathbf{a}_i| \approx 1$, otherwise it is marked as a side face. We then build connected components of faces with the same labels, and discard components that have only few triangle faces

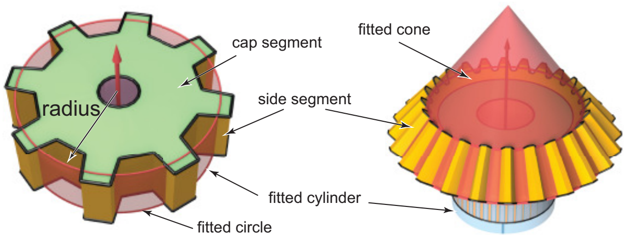

Figure 8: *Sharp edge loops are fitted with circles, and regular loops identified. The dominant part axis is used to partition the part into cap- and side-regions, which are then tested for cylindrical or cone fits for determining region types.*

as members (see Figure 8). Finally, we fit least squares cylinders and cones to the side regions, using the rotational axis and respective loop radius to initialize the non-linear fitting. Detected cylinder or conical side profiles are later used to classify joint types, e.g., cylinder-on-plane, cone-on-cone (bevel), etc.

We obtain the remaining relevant part attributes from the sharp edge loops. Their radii give us estimates for inner and outer radii of parts. For parts with cylindrical, conical, or helical regions, we count the intersection of the sharp edge loops with respect to their fitted circles to get the teeth count (see Figure 8). To rule out outlier parts like rectangular bars, we consider rotationally symmetric parts to be gears if they exhibit n-fold symmetry with $n > 4$.

Finally, for levers, chains and belts, which are essentially 1D structures, we use local principal component analysis to determine the dominant direction for the part.

5.2 Interaction Analysis

To build the interaction graph and estimate its parameters, we proceed in three steps: we compute the topology of the interaction graph based on the contact relationships between parts; we then classify the type of interaction at each edge (i.e., the type of joint or gear configuration); finally, we compute the motion of each part in the assembly.

Contact detection. We use the contact relationships between parts to determine the topology of the interaction graph. Following the approach of Agrawala et al. [2003], we consider each pair of parts in the assembly and compute the closest distance between them. If this distance is less than a threshold α, we consider the parts to be in contact, and we add an edge between their corresponding nodes in the interaction graph. We set α to be 0.1% of the diagonal of the assembly bounding box in our experiments.

As assemblies move, their contact relationships evolve. Edges e_{ij} in the interaction graph may appear or disappear over time. To compute the evolution of the interaction graph we establish contact relations using a space-time construction. Suppose at time t, two parts P_i and P_j are in contact and we have identified their joint-type (see later). Based on the joint-type we estimate their relative

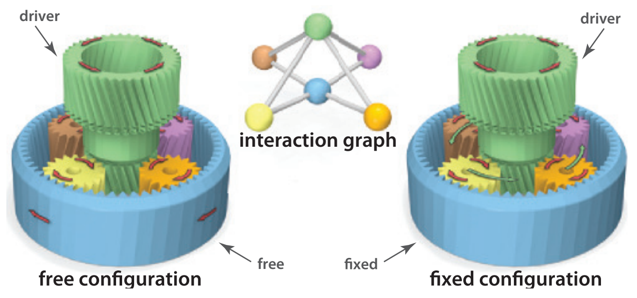

Figure 9: *Planetary gear arrangement. The same arrangement of mechanical parts can have largely different motion depending on the chosen driver and the part constraints. The blue gear can be free (left), or can be marked to be fixed (right).*

motion parameters and compute their positions at subsequent times $t + \Delta t, t + 2\Delta t$, etc. We stack the surfaces in 4D using time as the fourth dimension and connect corresponding points across adjacent time slices.

Often, due to symmetry considerations, it is sufficient to work with 2D cross sectional curves of parts and construct the space-time surface in 3D (see Figure 7). By extracting contact relations between the space-time surfaces (as an instance of local shape matching), we infer the time interval $[t, t + n\Delta t]$ when the parts P_i and P_j remain in contact, and hence the connection e_{ij} survives. The same method applies when the contact remains valid, but the junction parameters change, e.g., variable speed gear (see Figure 3). In this case, the space time surfaces become separated or start to intersect, as the junction parameter changes. Afterwards, we look for new contacts at the new event time, and continue building the graph. Note, we implicitly assume that the junction part contacts and parameters change discretely over the motion cycle allowing us to perform such an event based estimation. Hence we cannot handle continuously evolving junctions that may occur for parts like elliptic gears. Assuming that the relative speeds between parts of an assembly are reasonable, we used a fixed sampling $\Delta t = 0.1$sec with the default speed for the driver part set to angular velocity of .1 radian/sec, or translational velocity of 0.1 unit/sec, as applicable.

Interaction classification. Having established the connectivity of the graph, we now determine the relevant attributes for each edge (i.e., we classify the corresponding junction and estimate the motion parameters for the relevant parts). We categorize a junction between nodes P_1 and P_2 using their relative spatial arrangement and the individual part attributes. Specifically, we classify junctions based on the relation between the part axes $\mathbf{a}_1$ and $\mathbf{a}_2$, and then check if the part attributes agree at the junctions. For parts with multiple potential axes, we consider all pairs of axes.

Parallel axes: When the axes are nearly parallel, i.e., $|\mathbf{a}_1 \cdot \mathbf{a}_2| \approx 1$, the junction can be cylinder-on-cylinder (e.g. yellow and green gears in Figure 9), or cylinder-in-cylinder type (e.g. yellow and blue gears in Figure 9). For the former $r_1 + r_2$ (roughly) equals the distance between the axes, e.g., spur gears, helical gear; while for the latter $|r_1 - r_2|$ (roughly) equals the distance between the axes, e.g., inner ring gears, (belong to) planetary gears. Note for cylinder-on-cylinder, the cylinders can rotate about their individual axes, while simultaneously one cylinder can rotate about the other one, e.g., (subpart of) planetary configuration (see Figure 9).

Coaxial: As a special case of parallel axes, when both the axes are also contained by a single line, the junction is marked coaxial. This junction type is commonly encountered in wheels, cams, cranks, axles, etc.

Orthogonal axes: When the axes are nearly orthogonal, i.e., $\mathbf{a}_1 \cdot \mathbf{a}_2 \approx 0$, the junction can be a cylinder-on-plane, cylinder-on-line, rack-and-pinion, worm gear, bevel gear, helical gear. For cylinder-on-plane, one part is cylindrical with radius matching the distance between the cylinder axis and the plane. For cylinder-on-line, e.g., belts, pulley ropes, the principal direction of the 1D part is tangent to the cylinder. For a worm gear, one part is helical and the other cylindrical. If both parts are conical with their cone angles summing up to 90 degrees, we mark a bevel gear. When one part exhibits rotational symmetry, and the other translational symmetry, and their teeth width match, we flag a rack-and-pinion arrangement.

Our junction classification rules are carefully chosen and designed based on standard mechanical assemblies and can successfully categorize most joints automatically, as found in our extensive experimentation. However, our system also allows the user to intervene and rectify misclassifications, as shown in the supplementary video.

Compute motion. Mechanical assemblies are brought to life by an external force applied to a driver and propagated to other parts according to junction types and part attributes. In our system, once the user indicates the driver, motion is transferred to the other connected parts through a breadth-first graph traversal of the interaction graph G, starting with the driver-node as the root. We employ simple forward-kinematics to compute relative speed at any node based on the joint type with its parent [Davidson and Hunt 2004]. For example, for a cylinder-on-cylinder joint, if motion from a cylinder with radius r_1 and angular velocity ω_1 is transmitted to another with radius r_2, then the imparted angular velocity ω_2 is $\omega_1 r_1 / r_2$. Our approach handles graphs with loops (e.g., planetary gears). Since we assume our input models are consistent assemblies, even when multiple paths exist between a root node and another node, the final motion of the node does not depend on the graph traversal path. When we have an additional constraint at a node, e.g., a node is fixed or restricted to translate only along an axis, we perform a constrained optimization to find a solution. For example in Figure 9-right, when the green part is the driver, and the blue part is fixed, the intermediate nodes rotate both about their individual axes and also about the green cylinder to satisfy constraints on both their edges. Since we assume that the input assembly is a valid one and does not self-penetrate during its motion cycle, we do not perform any collision detection in our system. If the detected motion is wrong, the user can intervene and correct misclassifications. Such intervention was rarely needed for the large set of assemblies we tried.

6 Visualization

Using the computed interaction graph, our system automatically generates how things work visualizations based on the design guidelines discussed in Section 3. Here, we present algorithms for computing arrows, highlighting both the causal chain and important key frames of motion, and generating exploded views.

6.1 Computing Motion Arrows

For static illustrations, our system automatically computes arrows from the user-specified viewpoint. We support three types of arrows (see Figure 10): *cap arrows, side arrows,* and *translational arrows*. Our algorithm for generating these arrows consists of three steps: 1) determine how many arrows of each type to add, 2) compute initial arrow placements, and 3) refine arrow placements to improve their visibility.

For each (non-coaxial) edge in the interaction graph, based on the junction type, we create two arrows, one associated with each node connected by the graph edge. We refer to such arrows as contact-based arrows, as they highlight contact relations. We add contact arrows using the following rules:

cylinder-on-cylinder joints: we add cap arrows on both parts;

cylinder-in-cylinder joints: we add a cap arrow for the part inside and a side arrow for the one outside;

cylinder-on-plane joints: we add a cap arrow on the cylinder and a translational arrow for the translational part;

bevel gears: we add side arrows on both (conical) parts;

worm-gear: we add a cap arrow on the cylinder and a side arrow on the helical part.

Note that these rules do not add arrows for certain junction types (e.g., coaxial joints). Thus, after applying the rules, we add a non-contact arrow to any part that does not already have an associated contact arrow. For example, we place a side arrow around a coaxial joint. Furthermore, if a cylindrical part is long, a single arrow may not be sufficient to effectively convey the movement of the part. In this case we add an additional non-contact side arrow to the part. Thus, a part may have multiple arrows assigned to it.

Having decided how many arrows to add and their part associations, we compute their initial positions as follows. During the part analysis, we partitioned each part into cap and side segments (see Section 5). Using the z-buffer, we identify the cap and side face segments with the largest visible areas under self-occlusion and also under occlusion due to other parts. These segments serve as candidate locations for arrow placement; we place side arrows at the middle of the side segment with maximal score (computed as a combination of visibility and length of the side segment) and cap arrows right above the cap segment with maximal visibility. For contact-based side and cap arrows, we move the arrow within the chosen segment as close as possible to the corresponding contact point. Non-contact translational arrows are placed midway along the translational axis with arrow heads facing the viewer. The local coordinate frame of the arrows are determined based on the directional attributes of the corresponding parts, while the arrow widths are set to a default value. The remaining parameters of the arrows (d, r, θ in Figure 10) are derived in proportion to the part parameters like its axis, radius, side/cap segment area. We position non-contact side arrows such that the viewer sees the arrow head face-on.

Our initial arrow placement algorithm puts each arrow on a maximally visible cap or side segment. However, we can still improve arrow visibility by optimizing arrow placement within each segment. For cap arrows, we allow freedom to increase or decrease the radius for placement; while for side arrows we allow the freedom to translate along and rotate about the part axis. Optimization proceeds greedily, simultaneously exploring both directions, by taking small steps proportional to respective arrow thickness. We terminate the process when the arrow head is fully visible and the visibility of the arrow crosses a prescribed threshold (50% in our examples). Cap arrows viewed at oblique angles can be difficult to interpret. In such cases (angles greater than 70 degrees in our system), we switch cap arrows to side arrows, and locally adjust their parameters for better visibility.

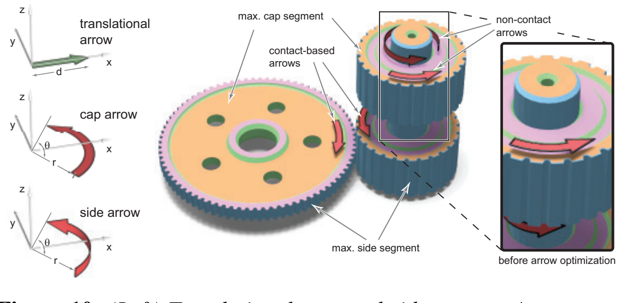

Figure 10: *(Left) Translational, cap and side arrows. Arrows are first added based on the interaction graph edges, and then to the moving parts without (sufficient) arrow assignment. The initial arrow placement can suffer from occlusion (right inset), which is fixed using a refinement step (center).*

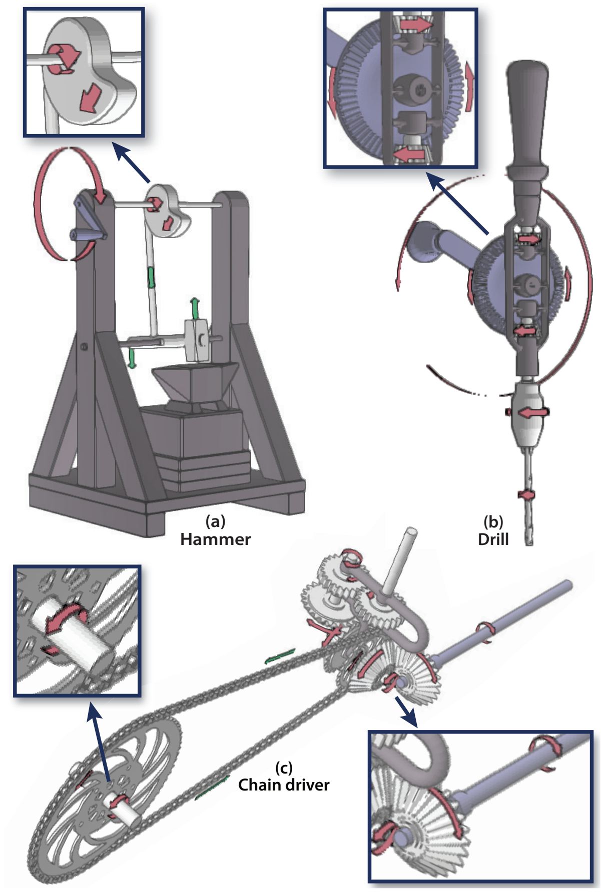

Figure 11: *Motion arrow results. To convey how parts move, our system automatically computes motion arrows from the user-specified viewpoint. Here, we manually specified the lever in the hammer model (a) and the belt in the chain driver model (c); our system automatically identifies the types of all the other parts.*

6.2 Highlighting the Causal Chain

To emphasize the causal chain of actions, our system generates a sequence of frames that highlights the propagation of motions and interactions from the driver throughout the rest of the assembly. Starting from the root of the interaction graph, we perform a breadth first traversal. At each traversal step, we compute a set of nodes S that includes the frontier of newly visited nodes, as well as any previously visited nodes that are in contact with this frontier. We then generate a frame that highlights S by rendering all other parts in a desaturated manner. To emphasize the motion of highlighted parts, each frame includes any non-contact arrow whose parent part is highlighted, as well as any contact-based arrow whose two associated parts are both highlighted. If a highlighted part only has contact arrows and none of them are included based on this rule, we add the part's longest contact arrow to the frame to ensure that every highlighted part has at least one arrow. In addition, arrows associated with previously visited parts are rendered in a desaturated manner. For animated visualizations, we allow the user to interac-

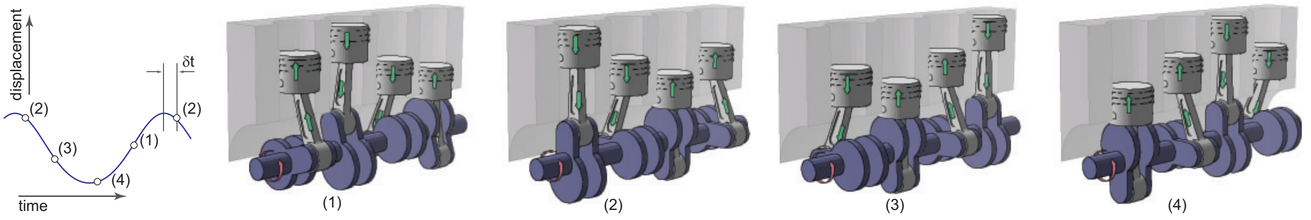

Figure 12: *Keyframes for depicting periodic motion of a piston-engine. Because of symmetry across parts and across motion (periodic), snapshot times are decided based on the positional extremes of the piston tops.*

tively step through the causal chain while the animation plays; at each step, we highlight parts and arrows as described above.

6.3 Highlighting Important Key Frames of Motion

As explained in Section 3, some assemblies contain parts that move in complex ways (e.g., the direction of motion changes periodically). Thus, static illustrations often include key frames that help clarify such motions. We automatically compute key frames of motion by examining each translational part in the model; if the part changes direction, we add key frames at the critical times when the part is at its extremal positions. However, since the instantaneous direction of motion for a part is undefined exactly at these critical times, we canonically freeze time δt *after* the critical time instances to determine which direction the part is moving in (see Figure 12). Additionally, for each part, we also add middle frames between extrema-based keyframes to help the viewer easily establish correspondence between moving parts (see F05 model example in the accompanying video). However, if such frames already exist as extrema-based keyframes of other parts, we do not add the additional frames, e.g., in the piston-engine example (see Figure 12).

Our system can also generate a single frame sequence that highlights both the causal chain and important key frames of motion. As we traverse the interaction graph to construct the causal chain frame sequence, we check whether any newly highlighted part exhibits complex motion. If so, we insert key frames to convey the motion of the part and then continue traversing the graph (see Figure 14c).

6.4 Exploded views

In some cases, occlusions between parts in the assembly make it difficult to see motion arrows and internal parts. To reduce occlusions, our system generates exploded views that separate portions of the assembly (see Figure 13). Typical exploded views separate all touching parts from one another to ensure that each part is visually isolated. However, using this approach in how things work illustrations can make it difficult for viewers to see which parts interact and how they move in relation to each other.

To address this problem, we only separate parts that are connected via a coaxial junction; since such parts move rigidly together (i.e., they rotate in the same direction around the same axis), we believe it is easier for viewers to understand their relative motion even when they are separated from each other. To implement this approach, our system first analyzes the interaction graph and cuts coaxial edges. The connected components of the resulting graph correspond to sub-assemblies that can be separated from each other. We use the technique of Li et al. [2008] to compute explosion directions and distances for these sub-assemblies.

7 Results

We used our system to generate both static and animated how things work visualizations for ten different input models, each of which contains from 7 to 27 parts (see Table 1). The models come from a variety of sources; for details, please refer to the Acknowledgements section. Figures 1, 11–14 show static illustrations of all ten models. Other than specifying the driver part and its direction of motion, no additional user assistance was required to compute the interaction graph for seven of the models. For the drum, hammer and chain driver models, we manually specified lever, cam and chain parts, respectively. In all of our results, we color the driver blue, fixed parts dark grey, and all other parts light grey. We render translation arrows in green, and side and cap arrows in red.

Our results demonstrate how the visual techniques described in Section 3 help convey the causal chain of motions and interactions that characterize the operation of mechanical assemblies. For example, the arrows in Figure 14a not only indicate the direction of rotation for each gear, their placement near contact points also emphasizes the interactions between parts. The frame sequence in Figure 14b shows how the assembly transforms the rotation of the driving handle through a variety of gear configurations, while the sequence in Figure 14c conveys both the causal chain of interactions (frames 1–3) as well as the back-and-forth motion of the horizontal rack (frames 3–6) as it engages alternately with the two circular gears. Finally, our animated results (see video) show how sequential

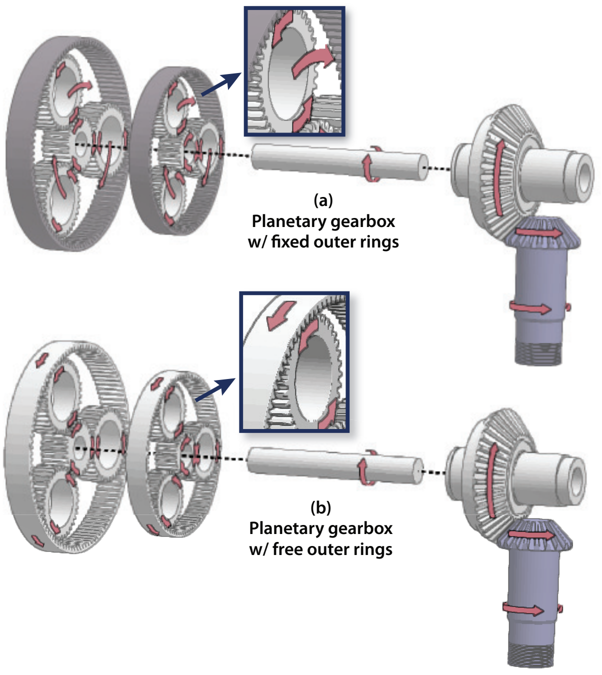

Figure 13: *Exploded view results. Our system automatically generates exploded views that separate the assembly at co-axial junctions to reduce occlusions. These two illustrations show two different configurations for the planetary gearbox: one with fixed outer rings (a), and one with free outer rings (b). The driver part is shown in blue, and fixed parts are shown in dark grey.*

highlighting of parts along the causal chain can help convey how motions and interactions propagate from the driver throughout the assembly while the animation plays.

model	tris.	# parts	# loops	R / T / H	time(sec)	# arrows	time(sec)
gears	23k	8	125	8 / - / -	0.44	12	0.96
F15	92k	19	126	9 / - / 1	1.65	11	1.10
planetary gearbox	49k	13	110	13 / - / -	1.42	28	1.85
Macaulay gears	32k	8	40	6 / 1 / 1	0.52	10	0.95
Leonardo's drum	26k	27	442	12 / - / -	0.99	22	1.00
Leonardo's hammer	3k	27	39	1 / - / -	0.16	6	0.31
piston-engine	18k	14	145	4 / - / -	0.67	9	0.06
F05	60k	9	30	2 / 1 / -	0.91	6	0.64
chain driver	61k	15	116	11 / 1 / -	1.96	14	0.93
hand drill	44k	7	51	4 / - / -	0.67	7	0.26

Table 1: *Performance statistics along with detected symmetry types of the parts, i.e., rotational (R), translational (T), and helical (H).*

8 Conclusions and future work

In this work, we have presented an automated approach for generating *how things work* visualizations from 3D CAD models. Our results demonstrate that combining shape analysis techniques with visualization algorithms can produce effective depictions for a variety of mechanical assemblies. Thus, we believe our work has useful applications for the creation of both static and animated visualizations in technical documentation and educational materials. To conclude, we present several areas of future work:

Automating more parts and configurations. The rules that we use in our analysis technique to identify different types of joint configurations handle a range of common assembly types. However, there are some parts and configurations that we cannot automatically recognize (e.g., the lever and belt configurations in Figure 11). New shape analysis methods may be necessary to automatically handle such parts.

Handling more complex models. We have tested our approach on several input assemblies of moderate complexity. Visualizing significantly more complex models (with hundreds or even thousands of parts) introduces additional challenges, including the possibility of excess visual clutter and large numbers of occluded parts. Although exploded views can help reduce occlusions, our approach of only separating parts that move together rigidly may be overly restrictive for assemblies with densely packed parts.

Handling fluids. While the parts in most mechanical assemblies interact directly with one another via contact relationships, some assemblies use fluid interactions to transform a driving force into movement (e.g., pumps, hydraulic machines). One approach for supporting such assemblies would be to incorporate a fluid simulation into our analysis technique.

Visualizing forces. In addition to visualizing motion, some *how things work* illustrations also depict the direction and magnitude of physical forces, such as friction, torque and pressure, that act on various parts within the assembly. Automatically detecting and visualizing such forces is an open direction for future work.

Acknowledgements

We thank Helmut Pottmann, Pat Hanrahan and Carlo Sequin for interesting discussions and suggestions, and the anonymous reviewers for their useful comments. This work was partially supported by a Microsoft young faculty fellowship and National Science Foundation grant CCF-0643552. We thank Bin Chan for helping with the video, Jonathan Balzer for the voice over, and Zhengzheng Kuang and Ruihua Ye for creating the Macaulay gear model. The hand-drill and chain-driver models are obtained from the Google 3D Warehouse, F15 and F05 from the "Antique Mechanism Teaching Models Digital Museum", Leonardo's drum and hammer models from Exchange3D, and the piston-engine model from Design-simulation Technologies Inc.

References

AGRAWALA, M., PHAN, D., HEISER, J., HAYMAKER, J., KLINGNER, J., HANRAHAN, P., AND TVERSKY, B. 2003. Designing effective step-by-step assembly instructions. *Proc. ACM SIGGRAPH*, 828–837.

AMERONGEN, C. V. 1967. *The Way Things Work: An Illustrated Encyclopedia of Technology*. Simon and Schuster.

ASSA, J., CASPI, Y., AND COHEN-OR, D. 2005. Action synopsis: pose selection and illustration. *ACM Trans. on Graphics (Proc. SIGGRAPH) 24*, 3, 667–676.

BENKÖ, P., MARTIN, R. R., AND VÁRADY, T. 2001. Algorithms for reverse engineering boundary representation models. *Computer Aided Design 33*, 11, 839–851.

BOUVIER-ZAPPA, S., OSTROMOUKHOV, V., AND POULIN, P. 2007. Motion cues for illustration of skeletal motion capture data. In *Proceedings of the 5th international symposium on Non-photorealistic animation and rendering*, 140.

BRAIN, M. 2001. *How stuff works*. Hungry Minds New York.

BRUCKNER, S., AND GROLLER, M. E. 2006. Exploded views for volume data. *IEEE Transactions on Visualization and Computer Graphics 12*, 5, 1077–1084.

BURNS, M., AND FINKELSTEIN, A. 2008. Adaptive cutaways for comprehensible rendering of polygonal scenes. In *SIGGRAPH Asia '08: ACM SIGGRAPH Asia 2008 papers*, ACM, New York, NY, USA, 1–7.

COHEN-STEINER, D., ALLIEZ, P., AND DESBRUN, M. 2004. Variational shape approximation. In *Proc. ACM SIGGRAPH*, 905–914.

COLLOMOSSE, J., ROWNTREE, D., AND HALL, P. 2005. Rendering cartoon-style motion cues in post-production video. *Graphical Models 67*, 549–564.

CUTTING, J. E. 2002. Representing motion in a static image: constraints and parallels in art, science, and popular culture. *Perception 31*, 1165–1193.

DAVIDSON, J. K., AND HUNT, K. H. 2004. *Robots and Screw Theory: Applications of Kinematics and Statics to Robotics*. Oxford University Press.

DAVIS, R. 2007. Magic paper: Sketch-understanding research. *Computer 40*, 9, 34–41.

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

DONY, R., MATEER, J., ROBINSON, J., AND DAY, M. 2005. Iconic versus naturalistic motion cues in automated reverse storyboarding. In *Conf. on Visual Media Production*, 17–25.

FEINER, S., AND SELIGMANN, D. 1992. Cutaways and ghosting: satisfying visibility constraints in dynamic 3D illustrations. *The Visual Computer 8*, 5, 292–302.

FU, H., COHEN-OR, D., DROR, G., AND SHEFFER, A. 2008. Upright orientation of man-made objects. In *ACM Trans. on Graphics (Proc. SIGGRAPH)*, 1–7.

GAL, R., SORKINE, O., MITRA, N. J., AND COHEN-OR, D. 2009. iWIRES: An analyze-and-edit approach to shape manipulation. *ACM Trans. on Graphics (Proc. SIGGRAPH) 28*, 3, #33, 1–10.

GELFAND, N., AND GUIBAS, L. J. 2004. Shape segmentation using local slippage analysis. In *Proc. of Symp. of Geometry Processing*, 214–223.

GOLDMAN, D. B., CURLESS, B., SALESIN, D., AND SEITZ, S. M. 2006. Schematic storyboarding for video visualization and editing. *ACM Trans. on Graphics (Proc. SIGGRAPH) 25*, 3, 862–871.

HEGARTY, M., KRIZ, S., AND CATE, C. 2003. The roles of mental animations and external animations in understanding mechanical systems. *Cognition and Instruction 21*, 4, 325–360.

HEGARTY, M. 1992. Mental animation: Inferring motion from static displays of mechanical systems. *Journal of Experimental Psychology: Learning, Memory, and Cognition 18*, 5, 1084–1102.

HEGARTY, M. 2000. Capacity limits in diagrammatic reasoning. In *Theory and Application of Diagrams*. 335–348.

HEISER, J., AND TVERSKY, B. 2006. Arrows in comprehending and producing mechanical diagrams. *Cognitive Science 30*, 581–592.

JOSHI, A., AND RHEINGANS, P. 2005. Illustration-inspired techniques for visualizing time-varying data. In *IEEE Visualization*, 679–686.

KAWAGISHI, Y., HATSUYAMA, K., AND KONDO, K. 2003. Cartoon blur: Non-photorealistic motion blur. In *Proc. of Computer Graphics International*, 276–281.

KIM, B., AND ESSA, I. 2005. Video-based nonphotorealistic and expressive illustration of motion. *Proceedings of Computer Graphics International (CGI 05)*, 32–35.

KLEIN, F., AND (TRANSLATOR), M. W. H. 1893. A comparative review of recent researches in geometry. *Bull. New York Math. Soc.*, 215–249.

KRIZ, S., AND HEGARTY, M. 2007. Top-down and bottom-up influences on learning from animations. *International Journal of Human-Computer Studies 65*, 11, 911–930.

LANGONE, J. 1999. *National Geographic's how things work: everyday technology explained*. National Geographic.

LI, W., RITTER, L., AGRAWALA, M., CURLESS, B., AND SALESIN, D. 2007. Interactive cutaway illustrations of complex 3d models. *ACM Trans. on Graphics (Proc. SIGGRAPH) 26*, 3, #31, 1–11.

LI, W., AGRAWALA, M., CURLESS, B., AND SALESIN, D. 2008. Automated generation of interactive 3d exploded view diagrams. *ACM Trans. on Graphics (Proc. SIGGRAPH) 27*, 3.

MACAULAY, D. 1998. *The New Way Things Work*.

MASUCH, M., SCHLECHTWEG, S., AND SCHULZ, R. 1999. Speedlines: depicting motion in motionless pictures. In *SIGGRAPH Conference abstracts*.

MAYER, R. 2001. *Multimedia learning*. Cambridge Univ Pr.

MCCLOUD, S., 1993. Understanding Comics. 1993.

MCGUFFIN, M. J., TANCAU, L., AND BALAKRISHNAN, R. 2003. Using deformations for browsing volumetric data. In *Proceedings of the 14th IEEE Visualization*, 53.

MEHRA, R., ZHOU, Q., LONG, J., SHEFFER, A., GOOCH, A., AND MITRA, N. J. 2009. Abstraction of man-made shapes. In *ACM Trans. on Graphics (Proc. SIGGRAPH Asia)*, 1–10.

MITRA, N. J., GUIBAS, L., AND PAULY, M. 2006. Partial and approximate symmetry detection for 3D geometry. 560–568.

NARAYANAN, N., AND HEGARTY, M. 1998. On designing comprehensible interactive hypermedia manuals. *International Journal of Human-Computers Studies 48*, 2, 267–301.

NARAYANAN, N., AND HEGARTY, M. 2002. Multimedia design for communication of dynamic information. *International journal of human-computer studies 57*, 4, 279–315.

NIENHAUS, M., AND DÖLLNER, J. 2005. Depicting dynamics using principles of visual art and narrations. *IEEE Comput. Graph. Appl. 25*, 3, 40–51.

SELIGMANN, D., AND FEINER, S. 1991. Automated generation of intent-based 3D illustrations. In *Proc. ACM SIGGRAPH*, ACM, 132.

THOMPSON, D. W. 1917. *On Growth and Form*. Dover Publications.

TVERSKY, B., MORRISON, J. B., AND BETRANCOURT, M. 2002. Animation: Can it facilitate? *International Journal of Human Computer Studies 5*, 247–262.

VIOLA, I., KANITSAR, A., AND GRÖLLER, M. E. 2004. Importance-driven volume rendering. In *Proceedings of IEEE Visualization 2004*, H. Rushmeier, G. Turk, J. van Wijk, 139–145.

WHITING, E., OCHSENDORF, J., AND DURAND, F. 2009. Procedural modeling of structurally-sound masonry buildings. *ACM Transactions on Graphics 28*, 5, 112.

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. on Graphics (Proc. SIGGRAPH) 28*, 3, #33, 1–10.

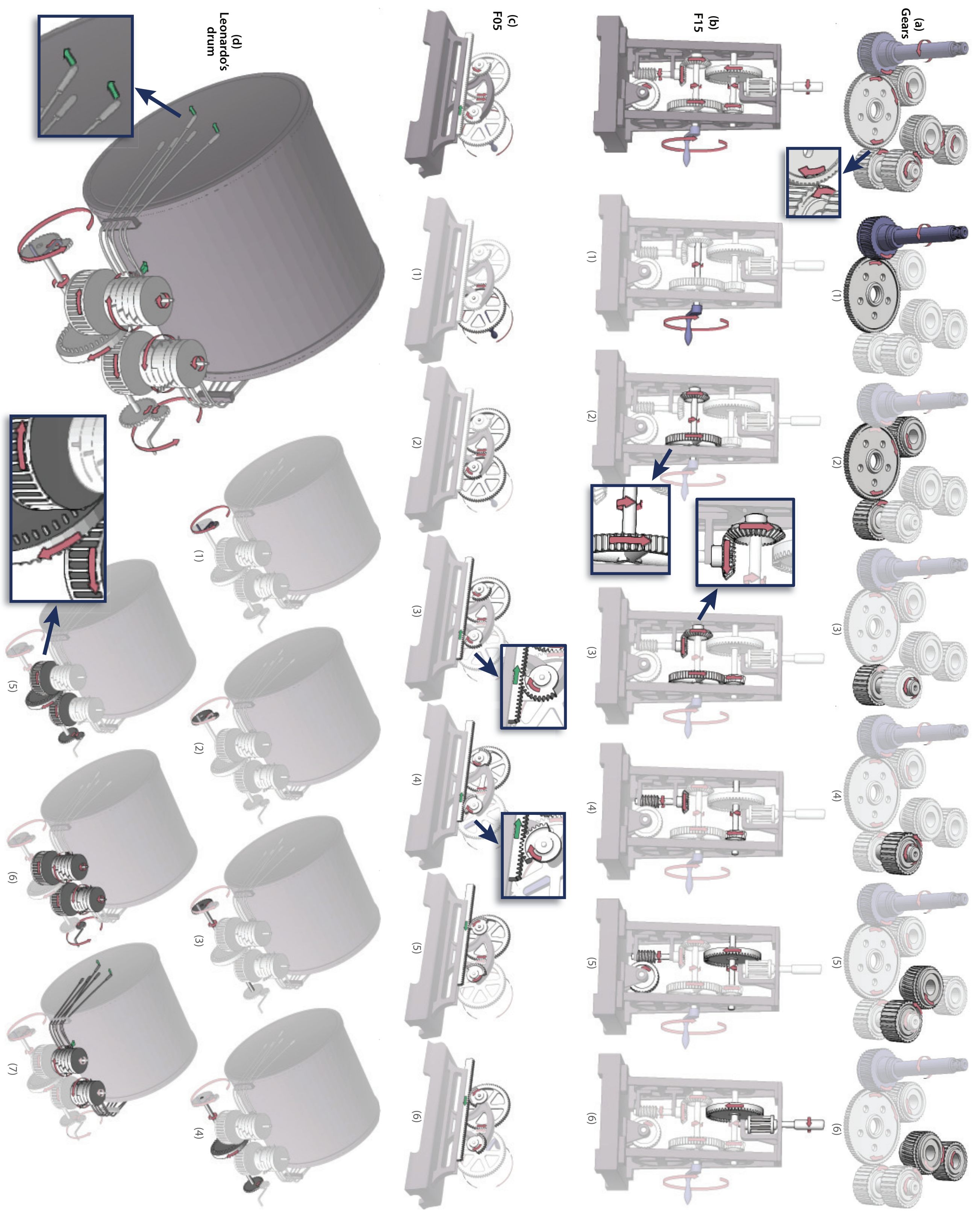

Figure 14: *Illustration results. We used our system to generate these* how things work *illustrations from 3D input models. For each model, we specified the driver part and its direction of motion. In addition, we manually specified the levers in the drum (c). From this input, our system automatically computes the motions and interactions of all assembly parts and generates motion arrows and frame sequences. We created the zoomed-in insets by hand.*