

# Illustrating How Mechanical Assemblies Work

By Niloy J. Mitra, Yong-Liang Yang, Dong-Ming Yan, Wilmot Li, and Maneesh Agrawala

## 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 the individual parts and the interactions across the 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 across parts. We demonstrate our system on a wide variety of assemblies.**

## 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*<sup>18</sup>)

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 a specific 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 such mechanical assemblies work.

There are two types of information that are crucial for understanding this transformation process: (i) the spatial configuration of the individual parts within the assembly and (ii) the causal chain of motions and mechanical interactions between the parts. While most technical illustrations effectively convey spatial relationships, only a much smaller subset of these visualizations is 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: (i) *motion arrows* indicate how individual parts move; (ii) *frame sequences* highlight key snapshots of complex motions and the sequence of interactions along the causal chain; and (iii) *animations* show the dynamic behavior of an assembly.

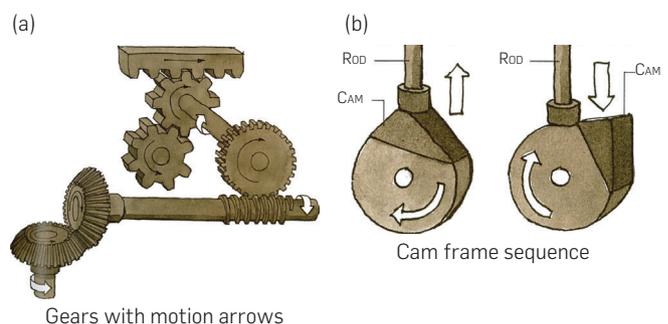
Creating effective how-things-work illustrations and animations by hand is difficult because a designer must understand how a complex assembly works and also have the skill to apply the appropriate visual techniques for emphasizing the motions and interactions between parts. As a result, well-designed 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 (see Figure 1).

In this paper, we present an automated approach for generating how-things-work visualizations of mechanical assemblies from 3D CAD models, thus facilitating the creation of both static illustrations and animations from any user-specified viewpoint (see Figure 2). Our work addresses two main challenges:

(i) **Motion and interaction analysis.** Most 3D models do not specify how their parts move or interact with one another. 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.

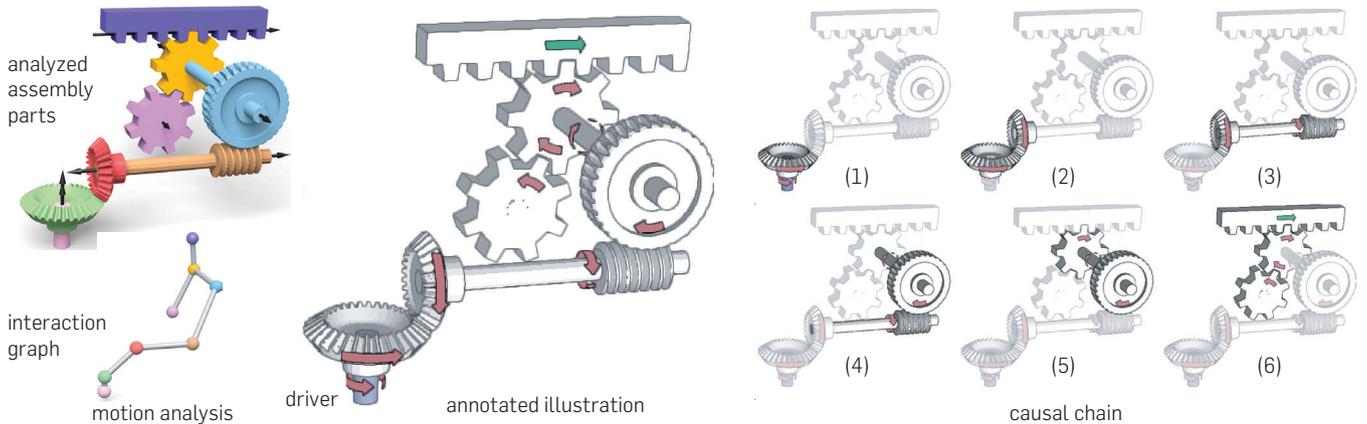
(ii) **Automatic visualization.** We present algorithms that use the motion and interaction information from our analysis to automatically generate a variety of how-things-work visualizations, including static illustrations with motion arrows, frame sequences to highlight key snapshots, and the

**Figure 1. 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.**



The original version of this paper was published in *ACM Transactions on Graphics (SIGGRAPH)*, July 2010.

**Figure 2.** We analyze a given geometric model of a mechanical assembly to infer how the individual parts move and interact with each other and encode this information as a time-varying interaction graph. Once the user indicates a driver part, we use the interaction graph to compute the motion of the assembly and generate an annotated illustration to depict how the assembly works. We also produce a corresponding causal chain sequence to help the viewer mentally animate the motion.



causal chain of mechanical interactions, as well as simple animations of the assembly in motion.

## 2. DESIGNING HOW-THINGS-WORK VISUALIZATIONS

Illustrators and engineers have produced a variety of books<sup>2, 3, 15, 18</sup> and websites (e.g., [howstuffworks.com](http://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 also studied how static and multimedia visualizations help people mentally represent and understand the function of mechanical assemblies.<sup>19</sup> For example, Narayanan and Hegarty<sup>24, 25</sup> propose a cognitive model for comprehension of mechanical assemblies from diagrammatic visualizations, which involves (i) constructing a spatial representation of the assembly and then (ii) producing the causal chain of motions and interactions between the parts. To facilitate these steps, they propose a set of high-level design guidelines to create *how-things-work* visualizations.

Researchers in computer graphics have concentrated on refining and implementing design guidelines that assist the first step of the comprehension process. Algorithms for creating exploded views,<sup>13, 16, 20</sup> cutaways,<sup>4, 17, 27</sup> and ghosted views<sup>6, 29</sup> of complex objects apply illustrative conventions to emphasize the spatial locations of the parts with respect to one another. In contrast, the problem of generating visualizations that facilitate the second step of the comprehension process remains largely unexplored within the graphics community. While some researchers have proposed methods for computing motion cues from animations<sup>26</sup> and videos,<sup>8</sup> these efforts do not consider how to depict the causal chain of motions and interactions between parts in mechanical assemblies.

### 2.1. Helping viewers construct the causal chain

In an influential treatise examining how people predict the behavior of mechanical assemblies from static

visualizations, Hegarty<sup>9</sup> 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. Although 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.

While 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.<sup>28</sup> 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:

(i) **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.<sup>10, 12</sup> Placing the arrows near contact points between parts that interact along the causal chain can help viewers better understand the causal relationships.

(ii) **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 keyframes that correspond to the sequence of steps in the chain. Each keyframe 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 improves comprehension compared to animations without such cues.<sup>11,14</sup>

**(iii) Highlight important keyframes 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 8 move the cylinder up and down during a single period of motion. To depict such complex motions, static illustrations sometimes include keyframes 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 keyframe between each pair of critical instances can help clarify how the parts move from one critical instance to the next.

### 3. 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.

### 4. MOTION + INTERACTION ANALYSIS

We analyze the input polyhedral model of an assembly to extract the degrees of freedom for each part, and also to understand how the parts move and interact with one another within the assembly. We encode the extracted information as an *interaction graph*  $G := (V, E)$  where, each node  $n_i \in V$  represents part  $P_i$  and each edge  $e_{ij} \in E$  represents a mechanical interaction between two touching parts ( $P_i, P_j$ ) (see Figure 2).

In order to construct this interaction graph, we rely on two high-level insights: first, the motion of many mechanical parts is related to their geometric properties, including self-similarity and symmetry; and second, the different types of mechanical interactions between parts 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 to construct the interaction graph:

In the *part analysis* stage, we analyze each individual part to determine its type (e.g., spur gear, bevel gear, and axle)

and relevant parameters (e.g., rotation axis, side profile, and radius) using existing shape analysis algorithms. We store the extracted information in the corresponding nodes of graph  $G$ .

In the *interaction analysis* stage, we analyze each pair of touching parts and classify the type of mechanical interaction based on their spatial relationships and part parameters. We store the information in the corresponding edges of graph  $G$ . Our system handles a variety of part types and interactions as shown in Figure 3.

#### 4.1. Part analysis

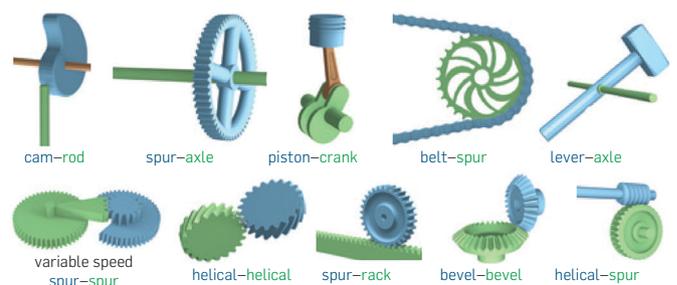
Our part analysis automatically classifies parts into the following common types: rotational gears (e.g., spur and bevel), helical gears, translational gears (i.e., racks in spur-rack mechanisms), axles, and fixed support structures (i.e., the stationary parts in an assembly that support and constrain the motions of other parts). The classifier is based on the geometric features of the parts. We rely on the user to manually classify parts that lack distinctive geometric characteristics such as cams, rods, cranks, pistons, levers, and belts. Figure 3 shows many of the moving parts handled by our system, and Figures 7a and 10b–c include some fixed support structures.

We also compute part parameters that inform the subsequent interaction analysis stage that determines how motion is transmitted across parts. For all gears and axles, we estimate the axis of rotational, helical, or translational motion. We compute teeth count and width for rotational and translational gears, and the pitch for helical gears. For rotational and helical gears, we also compute whether the part has a conical (e.g., bevel) or cylindrical (e.g., spur) side profile and its radii (e.g., inner, outer), as these properties influence the gear can interact with other gears. Since support structures often have housings or cutouts that constrain the rotational motion of other parts, we compute potential axes of rotation for these structures. Finally, we also compute potential rotation axes for user-classified cams, cranks, and levers.

To distinguish the different types of parts and estimate their parameters, we use the following shape analysis algorithms.

**Symmetry detection.** We assume that all gears and axles exhibit rotational, helical or translational symmetry and move based on their symmetry axes. We use a variant of the algorithm proposed by Mitra et al.<sup>22</sup> to detect such

**Figure 3. Typical part types and interactions encountered in mechanical assemblies and handled by our system. While we automatically detect most of these configurations, we require the user to manually classify cams, rod, cranks, pistons, levers, and belts.**



symmetries and infer part types and parameters based on the symmetry properties. If a part is rotationally symmetric, we mark it as either a rotational gear or axle, use the symmetry axis as the rotation axis, and use the order of symmetry to estimate teeth count and width. If a part is helically symmetric, we mark it as a helical gear, use the symmetry axis as the screw axis, and record the helix pitch. Finally, if a part has discrete translational symmetry, we mark it as a translational gear (i.e., rack), use its symmetry direction as the translation axis, and use the symmetry period to estimate the teeth count and width. Note that the symmetry detection method also handles partial symmetries that are present in parts like variable-speed gears (see Figure 3). If a part exhibits no symmetries and has not been classified by the user as a cam, rod, crank, piston, lever or belt, we assume it to be a fixed support structure.

**Cylinders versus cones.** Next, we analyze the side profiles of rotational and helical gears to determine whether they are cylindrical or conical, respectively. Specifically, we partition such gears into *cap* and *side* regions as follows. Let  $a_i$  denote the rotation/screw axis of part  $P_i$ .

We mark its  $j$ -th face as a cap face if its normal  $n_j$  is parallel to the part's rotational/screw axis, such that  $|n_j \cdot a_i| \approx 1$ , otherwise we mark it as a side face. We then build connected components of faces with the same labels, and discard components with only few faces as members (see Figure 4). Subsequently, we fit least squares cylinders and cones to the side regions and classify parts with low residual error as cylindrical or conical, respectively.

**Sharp edge loops.** Finally, we use *sharp edge loops*, which are 1D curves defined by sharp creases on a part, to determine additional part parameters for rotationally or helically symmetric parts, as well as cams, cranks, levers, and fixed support structures. We start by marking all mesh edges whose adjacent faces are close to orthogonal (i.e., dihedral angle in  $90^\circ \pm 30^\circ$  in our implementation) as *sharp* (see also Gal et al.<sup>7</sup> and Mehra et al.<sup>21</sup>). We then partition the mesh into segments separated by sharp edges, discard very small segments (less than 10 triangles in our tests), and label the boundary loops of the remaining segments as sharp edge loops. Next, we identify all the sharp edge loops that are (roughly) circular by fitting (in a least squares sense) circles to all the loops and selecting the ones with low residual errors. For rotationally and

helically symmetric parts, we use the minimum and maximum radii of the circular loops as estimates for the inner and outer radii of the parts (e.g., Figure 4-left shows the outer radius of a cylindrical gear).

For fixed support structures, a group of circular loops with a consistent orientation often indicates a potential axis of rotation for a part that docks with the fixed structure. Such clusters of loops also indicate potential rotation axes for cams, cranks, and levers. We cluster circular loops in two stages (see Figure 5): For each loop we compute the normal of the plane that contains the fitted circle, which we call the *circle axis*, and cluster loops with similar circle axes. Then, we partition each cluster based on the projection of the circle centers along a representative circle axis for that cluster. The resulting clusters represent groups of circular loops with roughly parallel circle axes that are close to one another. We record the representative circle axis for each cluster as a potential rotation axis.

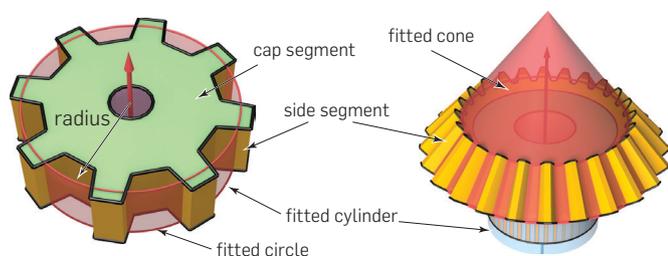
## 4.2. Interaction analysis

To build the edges of the interaction graph and estimate their parameters, we: (i) compute the topology of the interaction graph based on the contact relationships between part pairs; (ii) classify the type of mechanical interaction at each contact (i.e., how motion is transferred from one part to another); and (iii) compute the motion of the assembly.

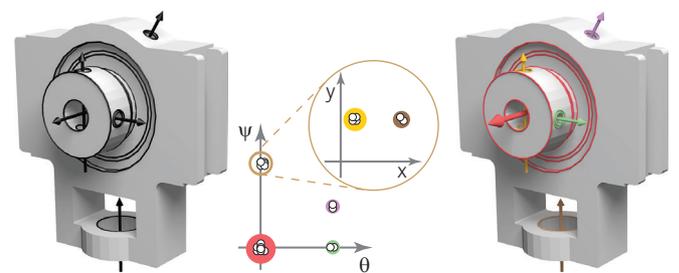
**(i) Contact detection.** We use the contact relationships between parts to determine the topology of the interaction graph. Following the approach of Agrawala et al.<sup>1</sup>, we consider each pair of parts ( $P_i, P_j$ ) in the assembly and compute the closest distance between them. If this distance is less than a threshold  $\alpha$ , we consider the parts to be in contact, and we add an edge  $e_{ij}$  between the nodes  $n_i$  and  $n_j$  in the interaction graph. We set  $\alpha$  to 0.1% of the diagonal of the assembly bounding box in our experiments.

As an assembly moves, its contact relationships evolve, that is, edges  $e_{ij}$  in the interaction graph may appear or disappear over time (see Figure 10c). We detect such contact changes using a space-time analysis. Suppose at time  $t$ , two parts  $P_i$  and  $P_j$  are in contact and we have identified their interaction type (see below). On the basis of this information, we estimate their relative motion parameters, compute their positions at subsequent times  $t + \Delta$ ,  $t + 2\Delta$ , etc., and

**Figure 4.** For rotationally and helically symmetric parts, we use their symmetry axes to partition the parts into cap- and side-regions. We fit cylinders or cones to the side regions to determine their profile types and extract sharp edge loops to estimate part attributes like radii.



**Figure 5.** We detect circular sharp edge feature loops on a part (left) and cluster the loops based on the orientations of their circle axes (middle) and the projections of the circle centers onto these axes (middle-inset). Such clusters represent groups of nearby loops with parallel axes, shown here in different colors (right).



compute the contact relationships at each time. We use a fixed sampling rate of  $\Delta = 0.1$  s with the default speed for the driver part set to an angular velocity of 0.1 radian/s or translational velocity of 0.1 unit/s, as applicable. Our method detects cases where two parts transition from touching to not touching over time. It also detects cases where two parts remain in contact but their contact region changes, which often corresponds to a change in the parameters of the mechanical interaction (e.g., the variable speed gear shown in Figure 3). For each set of detected contact relationships, we compute a new interaction graph. Note that we implicitly assume that part contacts change discretely over the motion cycle, which means that we cannot handle continuously evolving interactions, as in the case of elliptic gears. See original paper<sup>23</sup> for additional details.

**(ii) Interaction classification.** We classify the type of interaction for each pair of parts  $P_i$  and  $P_j$  that are in contact, using their relative spatial arrangement and the individual part attributes. Specifically, we classify interactions based on the positions and orientations of the part axes  $a_i$  and  $a_j$  along with the values of the relevant part parameters. For parts with multiple potential axes, we consider all pairs of axes.

*Parallel axes:* When the axes are nearly parallel, that is,  $|a_i \cdot a_j| \approx 1$ , we detect one of the following interactions: cylinder-on-cylinder (e.g., spur gears) or cylinder-in-cylinder (e.g., planetary gears). For cylinder-on-cylinder,  $r_i + r_j$  (roughly) equals the distance between the part axes. For cylinder-in-cylinder,  $|r_i - r_j|$  (roughly) equals the distance between the part axes. Note for cylinder-on-cylinder, the parts can rotate about their individual axes, while simultaneously one cylinder can rotate about the other one, for example, (subpart of) planetary configuration (see Figure 9).

*Coaxial:* When the axes are parallel and lie on a single line, we classify the interaction as coaxial (e.g., spur-axle and cam-axle).

*Orthogonal axes:* When the axes are nearly orthogonal, that is,  $a_i \cdot a_j \approx 0$ , we detect one of the following interactions: spur-rack, bevel-bevel, helical-helical, helical-spur. If one part is a rotational gear and the other is a translational gear with matching teeth widths, we detect a spur-rack interaction. If both parts are conical with cone angles summing up to  $90^\circ$ , we mark a bevel-bevel interaction. If both parts are cylindrical and helical, we mark a helical-helical interaction. If the parts are cylindrical but only one is helical, we mark a helical-spur interaction.

*Belt interactions:* Since belts do not have a single consistent axis of motion, we treat interactions with belts as a special case. If a cylindrical part touches a belt, we detect a cylinder-on-belt interaction.

These classification rules are carefully designed based on standard mechanical assemblies and successfully categorize most part interactions automatically. In our results, only the cam-rod and piston-crank interactions in the hammer (Figure 7a), piston engine (Figure 8), and the drum (Figure 10d) needed manual classification.

**(iii) Motion computation.** Mechanical assemblies are brought to life by an external force applied to a driver and propagated to other parts according to interaction 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 the relative speed at any node based on the interaction type with its parent<sup>5</sup>. For example, for a cylinder-on-cylinder interaction, if motion from a gear with radius  $r_i$  and angular velocity  $\omega_i$  is transmitted to another with radius  $r_j$ , then the imparted angular velocity  $\omega_j = \omega_i r_i / r_j$ . Our approach handles graphs with loops (e.g., planetary gears). Since we assume that 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, for example, a part is fixed or restricted to translate only along an axis, we impose the constraint in the forward-kinematics computation. Note that 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.

## 5. VISUALIZATION

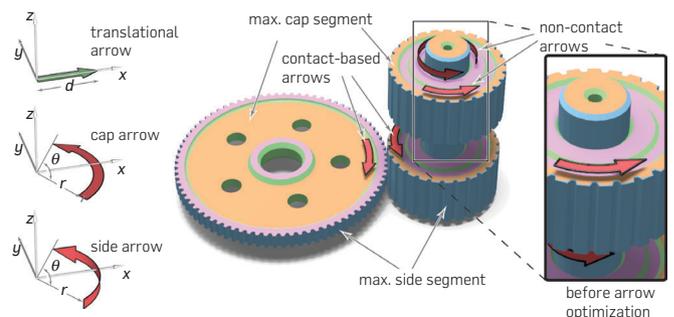
Using the computed interaction graph, our system automatically generates how-things-work visualizations based on the design guidelines discussed in Section 2. Here, we present algorithms for computing arrows, highlighting both the causal chain and important keyframes of motion, and generating exploded views.

### 5.1. Computing motion arrows

For static illustrations, our system automatically computes arrows from the user-specified viewpoint. We support three types of arrows (see Figure 6): *cap arrows*, *side arrows*, and *translational arrows* and generate them as follows: (i) determine how many arrows of each type to add; (ii) compute initial arrow placements; and (iii) refine arrow placements to improve their visibility.

For each non-coaxial part interaction encoded in the interaction graph, 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

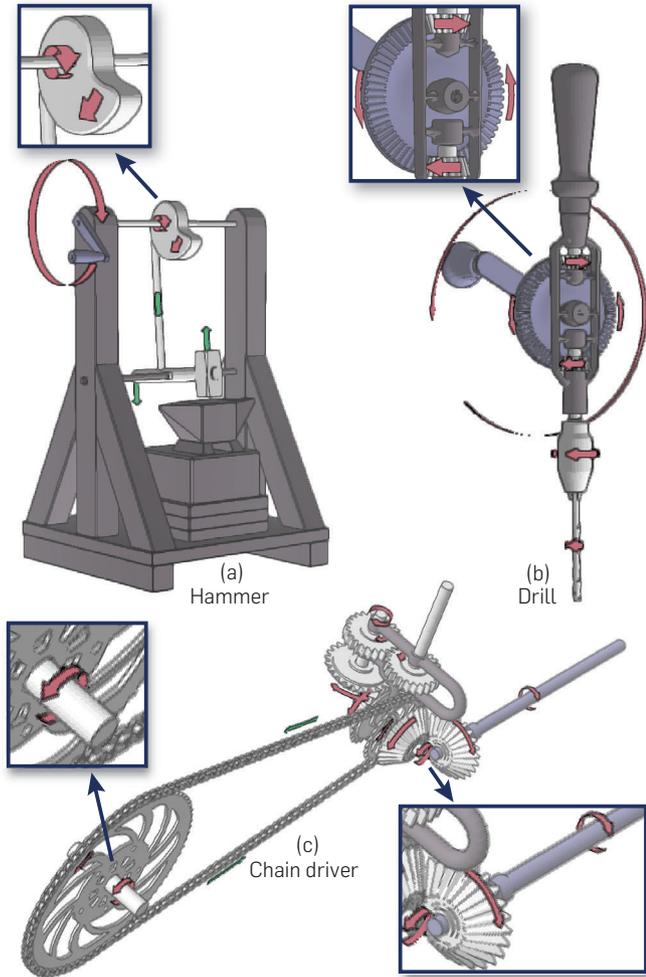
**Figure 6. Translational, cap, and side arrows (left). Arrows are first added based on the interaction graph edges, and then to any moving parts without an arrow assignment. The initial arrow placement can suffer from occlusion (right-inset), which is fixed using a refinement step (center).**



relations. We use the following rules to add contact arrows based on the type of part interaction:

- cylinder-on-cylinder*: add cap arrows on both parts;
- cylinder-in-cylinder*: add a cap arrow for the inner cylinder and a side arrow for the outer cylinder;

**Figure 7. Motion arrow results.** To convey how parts move, we automatically compute 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.



- spur-rack*: add a cap arrow to spur and translational arrow to rack;
- bevel-bevel*: add side arrows on both (conical) parts;
- helical-helical*: add a cap arrow on both parts;
- helical-spur*: add a cap arrow on the cylinder and a side arrow on the helical part; and
- cylinder-on-belt*: add a cap arrow on the cylinder and a translational arrow on the belt;

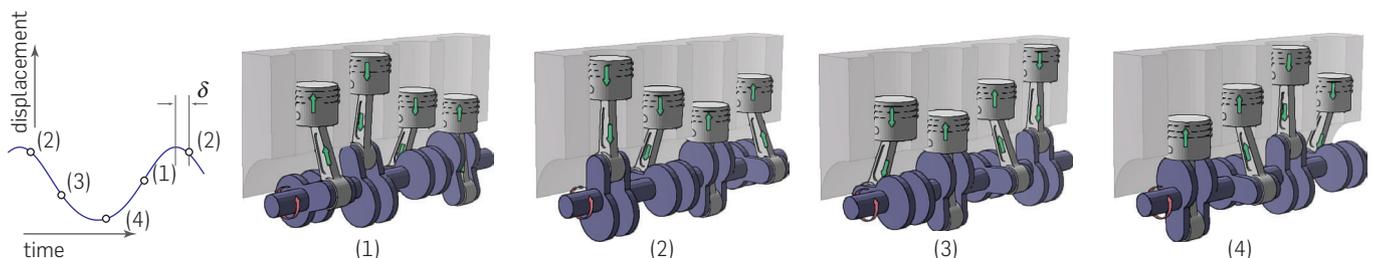
Note that these rules do not add arrows for certain types of part interactions (e.g., coaxial). For these interactions, we add a non-contact arrow to any part that does not already have an associated contact arrow. Furthermore, if a cylindrical part is long, a single arrow may not be sufficient to effectively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

After choosing the number of arrows to add and associating a part with each one, we next compute their initial positions using the cap and side segments for each part (see Section 4). We use the z-buffer to identify the cap and side face segments with the largest visible areas after accounting for occlusion from other parts as well as self-occlusion. 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 ( $d, r, \theta$  as in Figure 6) are derived in proportion to the part parameters like its axis, radius, and side/cap segment area. We position non-contact side arrows such that the viewer sees the arrow head face-on. Please refer to the original paper<sup>23</sup> for additional details.

## 5.2. 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

**Figure 8. Keyframes for depicting periodic motion of a piston engine.** For each of the pistons, we generate two keyframes based on its extremal positions (i.e., the top and bottom of its motion). We typically also add middle frames between these extrema-based keyframes, but due to the symmetry of the piston motion, the middle frames of each piston already exist as extrema-based keyframes of other pistons.

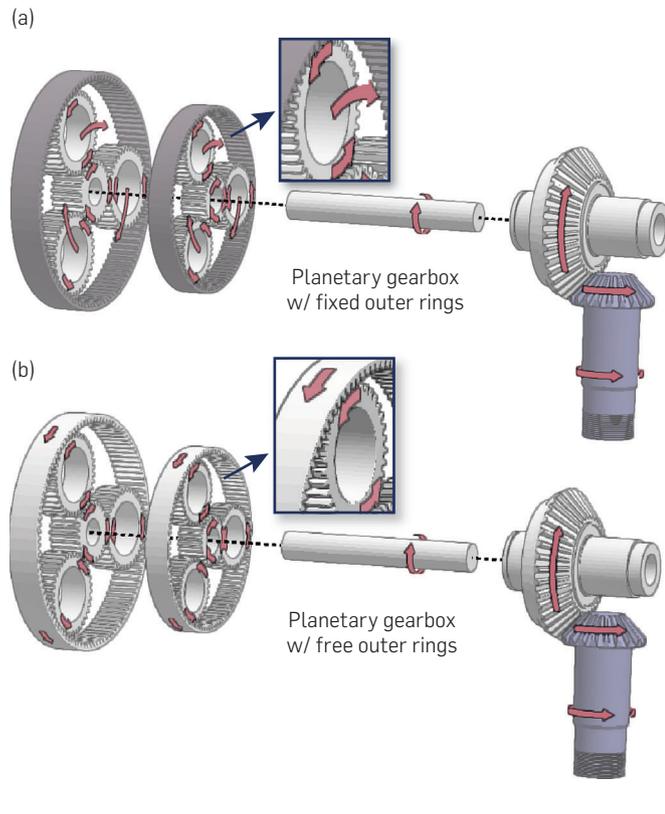


the rest of the assembly. Starting from the root of the interaction graph, we perform a breadth first traversal. At each traversal step, we compute a set of nodes  $S$  that includes the frontier of newly visited nodes, as well as any previously visited nodes that are in contact with this frontier. We 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 interactively step through the causal chain while the animation plays; at each step, we highlight parts and arrows as described above.

### 5.3. Highlighting keyframes of motion

As explained in Section 2, some assemblies contain parts that move in complex ways (e.g., the direction of motion changes periodically). Thus, static illustrations often include keyframes that help clarify such motions. We automatically

**Figure 9. Exploded view results.** Our system automatically generates exploded views that separate the assembly at coaxial part interactions 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 in blue, while fixed parts are in dark gray.



compute keyframes of motion by examining each translational part in the model: if the part changes direction, we add keyframes 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  $\delta$  after the critical time instances to determine which direction the part is moving in (see Figure 8). Additionally, for each part, we also add middle frames between extrema-based keyframes to help the viewer easily establish correspondence between moving parts. However, if such frames already exist as the extrema-based keyframes of other parts, we do not add the additional frames (see Figure 8).

We also generate a single frame sequence that highlights both the causal chain and important keyframes 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 keyframes to convey the motion of the part and then continue traversing the graph (see Figure 10c).

### 5.4. Exploded views

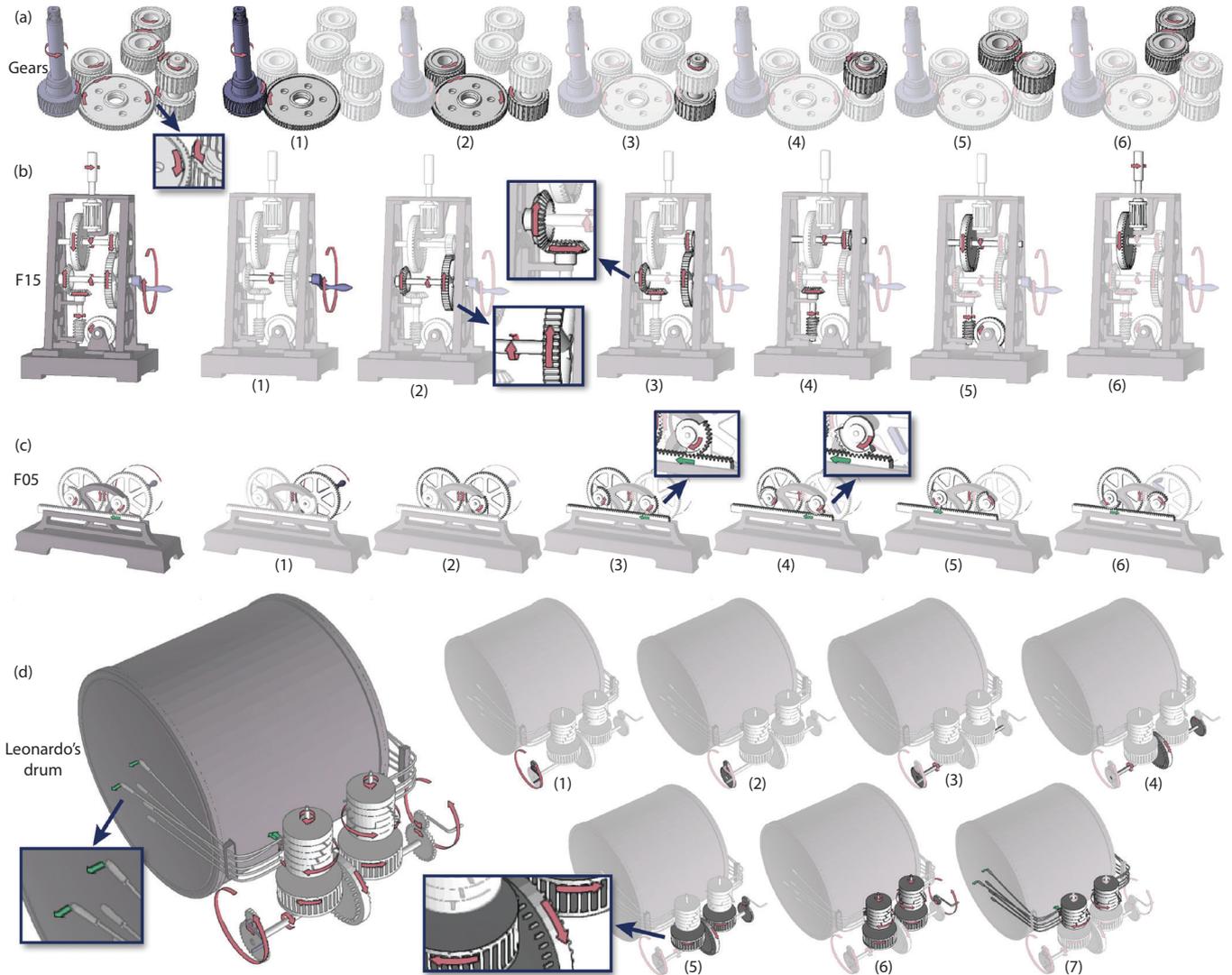
In some cases, occlusions between parts in the assembly make it difficult to see motion arrows and internal parts. To reduce occlusions, our system generates exploded views that separate portions of the assembly (see Figure 9). Typical exploded views separate all touching parts from one another to ensure 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 one another.

To address this problem, we only separate parts that are connected via a coaxial interaction; since such parts rotate rigidly around the same axis, we believe it is easier for viewers to understand their relative motion even when they are separated from one another. To implement this approach, our system first analyzes the interaction graph and then cuts coaxial edges. The connected components of the resulting graph correspond to sub-assemblies that can be separated from one another. We use the technique of Li et al.<sup>16</sup> to compute explosion directions and distances for these sub-assemblies.

## 6. 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, collected from various sources. Figures 2, 7–10 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, rod and belt parts, respectively. We also specified the crank and piston parts in the piston model. In all of our results, we colored the driver blue, fixed support structures dark gray, and all other parts light gray. We render translation arrows in green, and side and cap arrows in red. In all the examples, analysis takes 1–2 s, while visualization runs at interactive rates.

**Figure 10. 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.



Our results demonstrate how the visual techniques described in Section 2 help convey the causal chain of motions and interactions that characterize the operation of mechanical assemblies. For example, not only do the arrows in Figure 10a indicate the direction of rotation for each gear, but their placement near contact points also emphasizes the interactions between parts. The frame sequence in Figure 10b shows how the assembly transforms the rotation of the driving handle through a variety of gear configurations, while the sequence in Figure 10c conveys both the causal chain of interactions (frames 1–3) and 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 (which can be found at: [http://vecg.cs.ucl.ac.uk/Projects/SmartGeometry/how\\_things\\_work/](http://vecg.cs.ucl.ac.uk/Projects/SmartGeometry/how_things_work/)) show how sequential 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.

## 7. CONCLUSIONS AND FUTURE WORK

In this work, we 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.

We see several directions for extending our approach: (i) *Handling more complex models:* Analyzing and 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. (ii) *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 and hydraulic machines). One approach for supporting such assemblies would be to incorporate a fluid simulation into our analysis technique. (iii) *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 depicting such forces is an open research challenge. □

**References**

1. Agrawala, M., Phan, D., Heiser, J., Haymaker, J., Klingner, J., Hanrahan, P., Tversky, B. Designing effective step-by-step assembly instructions. In *Proceedings of ACM SIGGRAPH* (2003), 828–837.
2. Amerongen, C.V. *The Way Things Work: An Illustrated Encyclopedia of Technology*, Simon and Schuster, 1967.
3. Brain, M. *How Stuff Works, Hungry Minds*, New York, 2001.
4. Burns, M., Finkelstein, A. Adaptive cutaways for comprehensible rendering of polygonal scenes. In *ACM TOG (SIGGRAPH Asia)* (2008), 1–7.
5. Davidson, J.K., Hunt, K.H. *Robots and Screw Theory: Applications of Kinematics and Statics to Robotics*, Oxford University Press, 2004.
6. Feiner, S., Seligmann, D. Cutaways and ghosting: Satisfying visibility constraints in dynamic 3D illustrations. *Vis. Comput.* 8, 5 (1992), 292–302.
7. Gal, R., Sorkine, O., Mitra, N.J., Cohen-Or, D. iWIRES: An analyze-and-edit approach to shape manipulation. *ACM TOG (SIGGRAPH)* 28, 3:#33 (2009), 1–10.
8. Goldman, D.B., Curless, B., Salesin, D., Seitz, S.M. Schematic storyboarding for video visualization and editing. *ACM TOG (SIGGRAPH)* 25, 3 (2006), 862–871.
9. Hegarty, M. Mental animation: Inferring motion from static displays of mechanical systems. *J. Exp. Psychol. Learn. Mem. Cognit.* 18, 5 (1992), 1084–1102.
10. Hegarty, M. Capacity limits in diagrammatic reasoning. In *Theory and Application of Diagrams* (2000), 335–348.
11. Hegarty, M., Kriz, S., Cate, C. The roles of mental animations and external animations in understanding mechanical systems. *Cognit. Instruct.* 21, 4 (2003), 325–360.
12. Heiser, J., Tversky, B. Arrows in comprehending and producing mechanical diagrams. *Cognit. Sci.* 30 (2006), 581–592.
13. Karpenko, O., Li, W., Mitra, N., Agrawala, M. Exploded view diagrams of mathematical surfaces. *IEEE Vis.* 16, 6 (2010), 1311–1318.
14. Kriz, S., Hegarty, M. Top-down and bottom-up influences on learning from animations. *Int. J. Hum. Comput. Stud.* 65, 11 (2007), 911–930.
15. Langone, J. *National Geographic's How Things Work: Everyday Technology Explained*, National Geographic, 1999.
16. Li, W., Agrawala, M., Curless, B., Salesin, D. Automated generation of interactive 3d exploded view diagrams. *ACM TOG (SIGGRAPH)*, 27, 3 (2008).
17. Li, W., Ritter, L., Agrawala, M., Curless, B., Salesin, D. Interactive cutaway illustrations of complex 3d models. *ACM TOG (SIGGRAPH)*, 26, 3:#31 (2007), 1–11.
18. Macaulay, D. *The New Way Things Work*, Houghton Mifflin Books for Children, 1998.
19. Mayer, R. *Multimedia Learning*, Cambridge University Press, 2001.
20. McGuffin, M.J., Tancau, L., Balakrishnan, R. Using deformations for browsing volumetric data. In *IEEE Visualization* (2003).
21. Mehra, R., Zhou, Q., Long, J., Sheffer, A., Gooch, A., Mitra, N.J. Abstraction of man-made shapes. In *Proceedings of ACM TOG (SIGGRAPH Asia)* (2009), 1–10.
22. Mitra, N.J., Guibas, L., Pauly, M. Partial and approximate symmetry detection for 3D geometry. *ACM TOG (SIGGRAPH)* 25, 3 (2006), 560–568.
23. Mitra, N.J., Yang, Y.L., Yan, D.M., Li, W., Agrawala, M. Illustrating how mechanical assemblies work. *ACM TOG (SIGGRAPH)*, 29 (2010), 58:1–58:12.
24. Narayanan, N., Hegarty, M. On designing comprehensible interactive hypermedia manuals. *Int. J. Hum. Comput. Stud.* 48, 2 (1998), 267–301.
25. Narayanan, N., Hegarty, M. Multimedia design for communication of dynamic information. *Int. J. Hum. Comput. Stud.* 57, 4 (2002), 279–315.
26. Nienhaus, M., Döllner, J. Depicting dynamics using principles of visual art and narrations. *IEEE Comput. Graph. Appl.* 25, 3 (2005), 40–51.
27. Seligmann, D., Feiner, S. Automated generation of intent-based 3D illustrations. In *Proceedings of ACM SIGGRAPH* (1991), ACM, 132.
28. Tversky, B., Morrison, J.B., Betrancourt, M. Animation: Can it facilitate? *Int. J. Hum. Comput. Stud.* 5 (2002), 247–262.
29. Viola, I., Kanitsar, A., Gröller, M.E. Importance-driven volume rendering. In *IEEE Visualization* (2004), 139–145.

**Niloy J. Mitra** University College London.

**Wilmot Li** Adobe.

**Yong-Liang Yang, Dong-Ming Yan** King Abdullah University of Science and Technology (KAUST).

**Maneesh Agrawala** University of California, Berkeley.

Text excerpt and illustrations from *The Way Things Work* by David Macaulay. Compilation copyright (c) 1988, 1998 Dorling Kindersley, Ltd., London. Text copyright (c) 1988, 1998 David Macaulay, Neil Ardley. Illustrations copyright (c) 1988, 1998 David Macaulay. Used by permission of Houghton Mifflin Harcourt Publishing Company. All rights reserved.

© 2013 ACM 0001-0782/13/01

# World-Renowned Journals from ACM

ACM publishes over 50 magazines and journals that cover an array of established as well as emerging areas of the computing field. IT professionals worldwide depend on ACM's publications to keep them abreast of the latest technological developments and industry news in a timely, comprehensive manner of the highest quality and integrity. For a complete listing of ACM's leading magazines & journals, including our renowned Transaction Series, please visit the ACM publications homepage: [www.acm.org/pubs](http://www.acm.org/pubs).

## ACM Transactions on Interactive Intelligent Systems



ACM Transactions on Interactive Intelligent Systems (TIIS). This quarterly journal publishes papers on research encompassing the design, realization, or evaluation of interactive systems incorporating some form of machine intelligence.

## ACM Transactions on Computation Theory



ACM Transactions on Computation Theory (ToCT). This quarterly peer-reviewed journal has an emphasis on computational complexity, foundations of cryptography and other computation-based topics in theoretical computer science.

PLEASE CONTACT ACM MEMBER SERVICES TO PLACE AN ORDER  
 Phone: 1.800.342.6626 (U.S. and Canada)  
 +1.212.626.0500 (Global)  
 Fax: +1.212.944.1318  
 (Hours: 8:30am–4:30pm, Eastern Time)  
 Email: [acmhelp@acm.org](mailto:acmhelp@acm.org)  
 Mail: ACM Member Services  
 General Post Office  
 PO Box 30777  
 New York, NY 10087-0777 USA



Association for Computing Machinery

*Advancing Computing as a Science & Profession*

[www.acm.org/pubs](http://www.acm.org/pubs)