Interactive Image-Based Exploded View Diagrams

Wilmot Li
University of Washington

Maneesh Agrawala
Microsoft Research

David Salesin
Microsoft Research
University of Washington

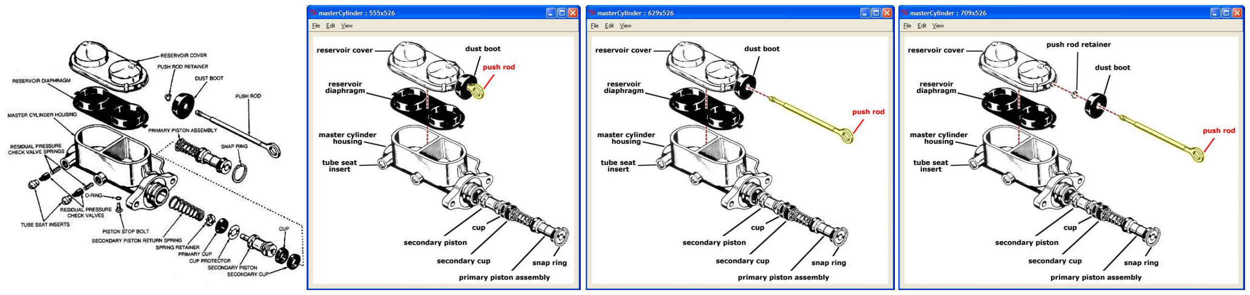

Figure 1: A static 2D exploded view diagram of a master cylinder (left). We construct an interactive version of the diagram using our system. These three frames show the user interactively expanding a portion of the object to examine it in more detail (right).

Abstract

We present a system for creating interactive exploded view diagrams using 2D images as input. This image-based approach enables us to directly support arbitrary rendering styles, eliminates the need for building 3D models, and allows us to leverage the abundance of existing static diagrams of complex objects. We have developed a set of semi-automatic authoring tools for quickly creating layered diagrams that allow the user to specify how the parts of an object expand, collapse, and occlude one another. We also present a viewing system that lets users dynamically filter the information presented in the diagram by directly expanding and collapsing the exploded view and searching for individual parts. Our results demonstrate that a simple 2.5D diagram representation is powerful enough to enable a useful set of interactions and that, with the right authoring tools, effective interactive diagrams in this format can be created from existing static illustrations with a small amount of effort.

Key words: Interactive diagrams, Image-based rendering, Exploded views

1 Introduction

Diagrams are essential for communicating the structure of complex 3D objects that are composed of many subparts, such as mechanical assemblies, architectural environments and biological organisms [5, 8, 15, 19, 25, 26]. To elucidate the composite structure of such objects, illustrators commonly use diagrammatic techniques such as exploded views and cutaways that reduce or eliminate occlusion and expose internal parts. In this work, we focus on exploded view diagrams, which simultaneously convey the global structure of the depicted object, the details of individual components, and the local relationships among them.

However, because exploded views are usually designed as static illustrations for print publications, they often suffer from two important drawbacks:

- ***Ambiguous spatial relationships.*** A static diagram can only show a fixed set of spatial relationships between parts. For complex objects, it may not be clear from a static exploded view how all the parts fit together, interact with, and constrain one another.
- ***Visual clutter.*** Static diagrams are usually designed to include all the information the viewer might need about the object. As a result, they are often visually cluttered, making it difficult to extract specific information about a particular part or subset of parts without carefully perusing the entire illustration.

In contrast, exploded view diagrams viewed through a computer can alleviate both of these problems by allowing viewers to interactively manipulate the parts and thereby dynamically filter the information presented in the diagram. For example, a viewer might interactively expand and collapse only the wheel assembly of a car diagram to better understand how the parts of that assembly interact with one another. On the other hand, a static, general-purpose car diagram would have to show all of the parts in an exploded state, making it difficult to focus on the wheel assembly. In general, we believe that interactive diagrams can be far more clear, informative, and compelling than their static counterparts.

In this paper, we present a novel framework for creating and viewing interactive exploded view diagrams of complex mechanical assemblies. As an example, Figure 1 shows a dynamic illustration that was authored and rendered using our system. Rather than using 3D models as input, our approach is to construct dynamic illustrations from 2D images, resulting in a layered 2.5D diagram representation. Although the lack of explicit 3D information puts some limits on the viewing experience (e.g., we cannot directly support arbitrary changes in viewpoint), this image-based strategy has several key benefits: it makes it easy to support arbitrary rendering styles (we just have to find or create pictures of each part of the object in the desired style); it obviates the need for 3D models, which are in general much more difficult to acquire or build than images; and, finally, using 2D images allows us to leverage the abundance of existing static exploded views commonly found in textbooks, repair manuals, and other educational material. As a result, we believe our image-based approach makes it possible to create effective interactive diagrams far more easily than with a 3D method.

The contributions of our work fall into two categories:

Semi-automatic authoring tools. The primary challenge in turning a 2D image into an interactive exploded view diagram is specifying how parts interact with one another. We provide a suite of semi-automatic tools for constraining the motion of parts as they expand and collapse, and for layering parts so that they properly occlude one another as they move. Our tools allow users to quickly create compelling interactive diagrams via simple, sketch-based interactions.

Interactive viewing interface. To help viewers dynamically filter the information presented in a diagram, we provide a viewing system that supports a number of useful interactions. Specifically, our interface allows the user to directly expand and collapse the exploded view, and search for individual parts. In our experience, these interactions help the viewer understand the spatial relationships between parts and the overall structure of the object.

2 Related work

Our work builds on two main areas of computer graphics research: automated design of technical illustrations, and 2.5D layer-based diagram representations. We consider related work in each of these areas.

Automated design of technical illustrations. A number of researchers have investigated the problem of automatically generating explanatory technical illustrations of 3D objects. Seligmann and Feiner [23] and Rist et al. [21] focus on generating a set of images to show the location or physical properties of a particular part within a 3D object. Butz [7] extends these techniques to automatically generate an illustrative animation rather than a set of images. Whereas these systems are aimed at completely automating all design decisions and thereby eliminating the need for a human designer, our work provides semi-automatic high-level interactive design tools that enable human designers to quickly produce the desired illustration. In addition, the previous systems do not produce interactive illustrations that allow users to directly manipulate the parts of the diagram. Several groups have also explored techniques for generating exploded views that reveal the complete structure of complex mechanical assemblies [1, 10, 16, 20], architectural environments [18], and anatomy [14, 22]. However, all of these systems rely on complete 3D representations of the object, whereas we use 2D images as input.

2.5D layer-based diagram representations. One of the main features of our image-based approach is a 2.5D representation for interactive diagrams that consists of layers of images. To facilitate the creation of diagrams in this format, we provide a set of 2.5D authoring tools. Although layer-based representations are not new in computer graphics [11, 13, 24], most of this previous work on 2.5D authoring has focused primarily on creating layered animations. Recently, Barett and Cheney introduced tools for selecting, bending, and even deleting entire objects rather than pixels in digital photographs [2]. In contrast to these general-purpose systems, we focus on the specific authoring issues involved in creating interactive image-based exploded view diagrams.

3 Authoring

Several steps are involved in creating an interactive image-based diagram (Figure 2). As input, our system accepts either a single image of an object with all of its constituent pieces visible (i.e., in a fully exploded state), or a set of images, one per piece. We assume that the object is rendered using an orthographic projection, as is typical in technical illustrations[1]. In the case where a single image is used as input, the static diagram is first segmented into *parts* corresponding to the constituent pieces of the depicted object. Next, these parts are organized into *stacks* that define how parts move relative to one another as the object is expanded and collapsed. The parts are then layered to produce the correct occlusion relationships between them. As we show later, this layering step often involves breaking parts into smaller *fragments* before as-

[1]With perspective projections, parts may not fit together properly when the exploded view is collapsed.

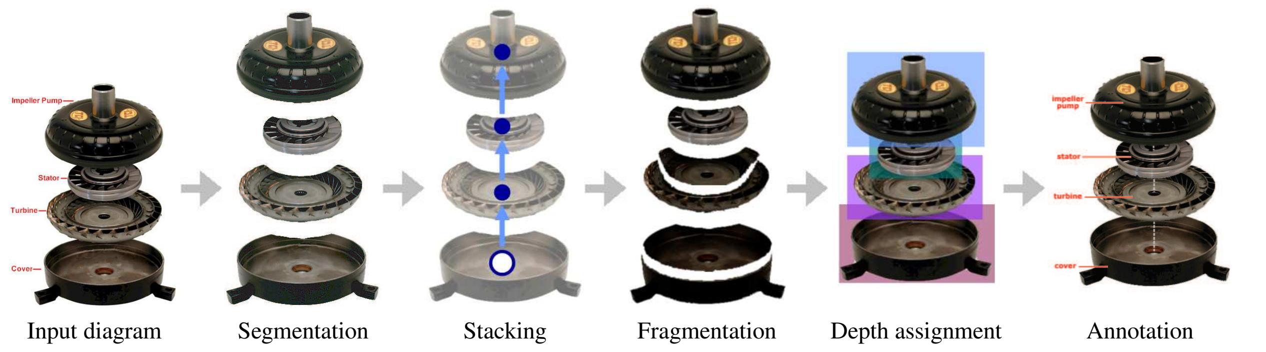

Figure 2: The flowchart for converting a static 2D exploded view diagram into an interactive diagram. We segment the input diagram into parts, organize the parts into stacks, break the parts into fragments, layer the parts and fragments so that they properly occlude one another, and finally, add desired annotations, such as labels and guidelines.

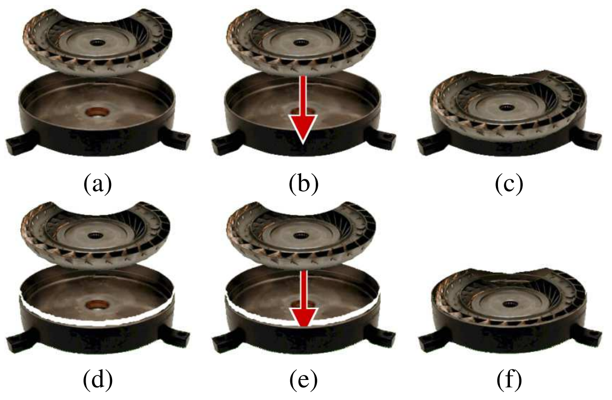

Figure 3: The occlusion relationship between the turbine (on top) and the bottom cover of the catalytic converter shown in Figure 2. If the turbine and bottom cover are each given a single depth value, the turbine incorrectly occludes the outer lip of the cover (a–c). Instead if we split the cover into two fragments and then layer the turbine between them we can produce the proper occlusion relationship (d–f).

signing depth values to each piece in the diagram. Finally, the diagram can be annotated with labels and guidelines. The remainder of this section outlines the stages of this pipeline in greater detail.

3.1 Diagram representation

A diagram in our system consists of parts and stacks. Each part includes an image of its corresponding component, as well as an alpha mask that defines its bounding silhouette. To achieve the correct impression of relative depth between the various portions of the object, parts are also assigned depth values that determine how they are layered. When two or more parts interlock such that they cannot be correctly rendered using the "painter's algorithm," [12] it is insufficient to assign a single depth value to each part (Figure 3). To solve this problem, parts can be divided into *fragments*. By specifying the appropriate depth value for each fragment, we can achieve the correct occlusion relationship between parts that overlap in complex ways.

To enable parts to expand and collapse dynamically, they are organized into stacks that define how the parts are allowed to move in relation to one another. More precisely, a stack is an ordered sequence of parts that share the same *explosion axis* (as defined by Agrawala et al. [1]). The explosion axis is a vector that specifies the line along which stack parts can move. We refer to the first part in a stack as its *root*. In our diagrams we enforce the restriction that each part can be a non-root member of only one stack. However, the same part can be the root for any number of stacks. Thus, a collection of stacks always forms a tree, as shown in Figure 4.

For each of its constituent parts, a stack stores three parameters. The *initial position* specifies the position of a part in its fully collapsed state with respect to its predecessor, the *current offset* keeps track of the part's current displacement from its initial position, and the *maximum offset* indicates how far a part can possibly move away from the preceding part in the stack. Given these stack parameters, the position of each part depends only on the position of its predecessor.

3.2 Creating parts

To help the user segment a single static illustration into parts, the authoring system includes an Intelligent Scissors (I-Scissors) tool [17] that makes it easy to cut out the individual components of the depicted object. The user simply loads the input image into the interface and then oversketches the appropriate part boundaries using I-scissors. In some cases, a component that is partially occluded in the input illustration might have holes in it that

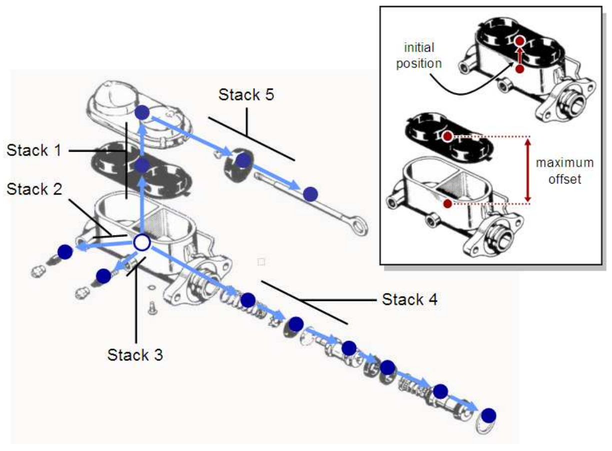

Figure 4: The stack hierarchy for the master cylinder. The arrows indicate the ordering of parts within each stack. The stack parameters consist of the initial position *of each part with respect to its predecessor and the* maximum offset, *which is the furthest distance a part can move with respect to its predecessor (inset).*

need to be filled. This can either be done manually using Adobe Photoshop, or via automatic hole-filling techniques [4, 9][2].

3.3 Creating stacks

After the parts have been created, they can be organized into stacks via a simple, sketch-based interaction (Figure 5). To create a new stack, the user connects the appropriate set of parts by drawing a free-form stroke. These components are then organized into a stack, preserving the part order defined by the stroke. The system assumes that the specified parts are currently in their fully exploded configuration and then infers an explosion axis, initial positions, and maximum offsets for the new stack that are consistent with this layout.

To determine the explosion axis, the system connects the bounding box centers of the first and last stack components with a straight line. The initial position for each part is set by default to be a small offset from its predecessor along the explosion axis. Since the parts start out in their fully exploded layout, the system sets the maximum offset for each part to be the distance from the part's initial position to its current, fully exploded position.

The user can manually tweak the stack parameters once the new stack is created via a number of simple direct-manipulation operations. To modify the explosion axis, the user drags out a line anchored at the stack's root, and then adjusts this vector to the desired direction. The

[2]Currently, our system does not include automatic hole-filling tools.

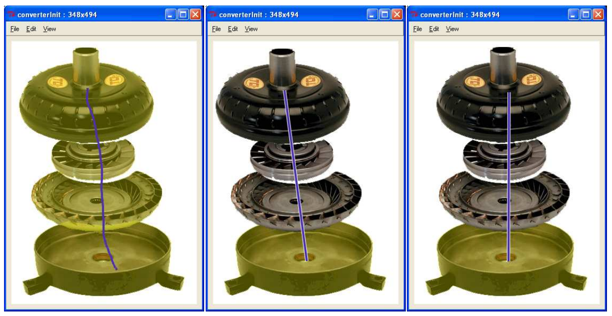

Figure 5: Users draw a free-form stroke to organize a set of parts into a stack (left). The stroke directly indicates the order of the parts in the stack as well as the explosion axis (middle). Users can interactively adjust the explosion axis if necessary (right).

stack's axis updates interactively during this operation so that the user can easily see how the parts line up. To change a part's initial position and maximum offset, the user switches to a "stack manipulation" mode, and then drags the component to its appropriate fully collapsed and expanded positions.

3.4 Layering

After all of the stacks have been created, parts are fragmented if necessary and then layered to produce the correct impression of relative depth between them. The user can manually partition a part into fragments with I-Scissors, and then explicitly assign a depth value to each part or fragment in the diagram. However, for objects with more than a few components, this type of manual layer specification can be tedious. To reduce the authoring burden, our system provides semi-automatic fragmentation and depth assignment tools that can be used for a large class of interlocking parts.

Semi-automatic fragmentation

Typically when two parts interlock, one component fits roughly inside the other (e.g., the two parts in Figure 3). In this case, the correct layering can usually be achieved by splitting the outer part into front and back fragments, and then layering the inner part to pass between them. To fragment the outer part, the user oversketches (with the help of I-Scissors) the closed boundary of the cavity or opening that encloses the inner piece. As shown in Figure 6, we refer to the 3D boundary of the cavity as B. The curve that the user draws, C, is B's projection onto the image plane. Given C, the system computes the occluding portion of this curve, C_O, where the inner part passes behind the outer part, and then uses it to divide the enclosing component into two fragments.

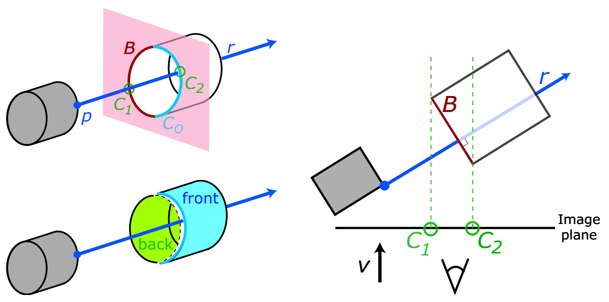

Figure 6: A 3D point p that passes through the opening defined by B while traveling in the explosion direction r (left top). Since the fragmentation assumptions are satisfied, the system computes the correct front and back fragments (left bottom). In the same scene viewed from above, we can clearly see that p passes in front of B at C_1 and behind B at C_2 (right).

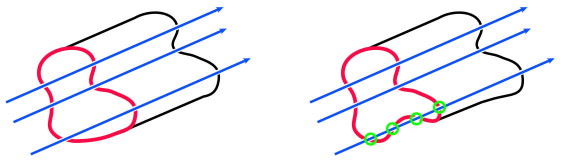

Figure 7: The cavity depicted on the left meets our constraints on the shape of C because a vector in the explosion direction intersects the red curve at most twice. The cavity on the right does not satisfy our assumptions because the bottom vector intersects the red curve in more than two places.

The system extracts C_O by determining, for any 3D point p that goes through the opening, where p passes behind B (i.e., out of the viewer's sight). Since parts are constrained to move within their stacks, we consider only points that go through the opening while traveling in the explosion direction r (Figure 6). The system assumes that C does not self-intersect, and that any line parallel to the explosion axis intersects C no more than twice (Figure 7). Given these restrictions on the shape of C and ignoring the tangent cases, the projection of r onto the image plane will intersect C exactly twice (at C_1 and C_2). Let C_1 be the first intersection point as we follow r away from the viewer, as shown in Figure 6. By default, we assume that p passes in front of B at C_1 and behind B at C_2, which corresponds to the common case in which p enters the opening defined by B as it moves away from the viewer.

Given this assumption, Figure 8 depicts the basic steps for computing C_O. The user must specify which end of the explosion axis points away from the viewer. We consider the path of every point that passes through B, by casting a ray from every pixel on C in the explosion direction. If the ray intersects C again, we add the pixel corresponding to this second intersection point to C_O. Once we are done processing the curve, we extrude C_O by rasterizing a line of pixels (using Bresenham's algorithm) in the explosion direction, starting from each pixel on the boundary. Every pixel that we encounter is added to the part's front fragment, and all remaining pixels comprise the back fragment. We stop the extrusion once we reach the boundary of the image.

Note that our assumptions produce correct fragmentations for a whole class of enclosing cavities of different shapes and orientations. Specifically, the assumptions do not restrict B to lie in a plane that is orthogonal to the explosion direction. For example, the notched object shown in Figure 9 contains a cavity with a non-planar opening. In this situation, the system is able to compute the correct fragmentation because all of the assumptions hold. Of course, there are situations that violate one or more of our assumptions. In Figure 10, B is oriented such that p emerges from behind C_1 and passes in front of C_2. In this case, the user can tell the system to invert the fragmentation algorithm by reversing the explosion direction.

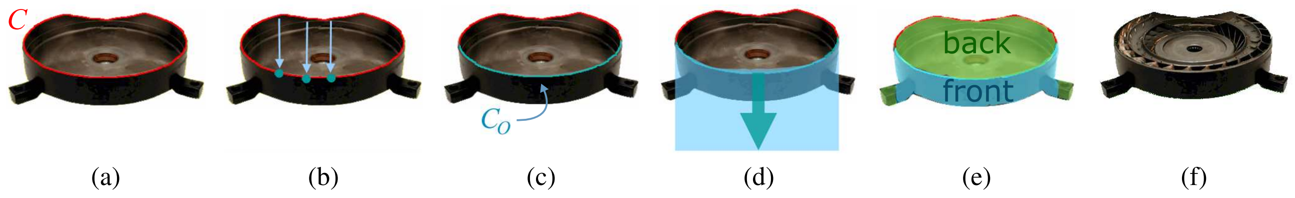

Figure 8: Semi-automatic fragmentation of the bottom cover. The user sketches the boundary of the cavity C (a). The system casts a ray from each pixel on the curve in the direction of the explosion axis pointing away from the viewer (b). For all rays that intersect C twice, the second point of intersection is added to the occluding portion of the curve C_O (c). The system extrudes C_O along the explosion axis (d). All pixels lying within the extruded region are classified as the front fragment and the remaining pixels are classified as the back fragment (e). The system can now set the depth value of the turbine to lie between the depth values of the front and back fragments to produce the correct occlusion relationship (f).

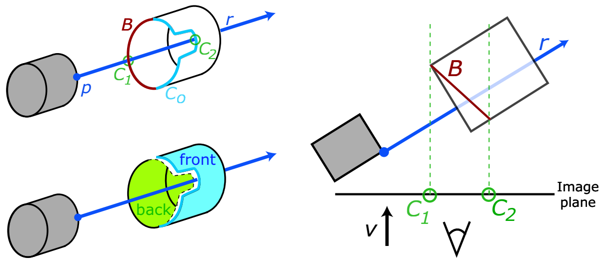

Figure 9: A cavity with a non-planar opening. The opening defined by B has a notch in it that causes B to be non-planar (left top). However, the system is still able to compute the correct layering shown here (left bottom). As long as p always passes in front of B at C_1 for all pairs of intersection points C_1 and C_2, the fragmentation algorithm obtains the correct result.

This inverted computation obtains the correct fragmentation result. In practice, however, we have found our default fragmentation assumptions to be valid for a large class of interlocking parts.

Semi-automatic depth assignment

Once all of the appropriate parts have been fragmented, the user can ask the system to infer part layers. We use a simple set of heuristics to compute a plausible layering for a diagram. For non-interlocking parts within a stack, we assume that their depth values are either strictly increasing or decreasing when we consider them in stacking order. For interlocking parts, we assume that the inner part must be layered between the outer part's front and back fragments. To compute a layering, the system first infers which parts interlock and then determines an assignment of depth values that satisfies these rules.

To determine whether two parts interlock, we check if the cross-section of one part (with respect to the explosion direction) fits within the cross-section of the curve that defines the cavity opening (if there is one) in the other part. If so, then the system assumes that the first part fits inside the second. Otherwise, the parts are assumed not to interlock. Although this heuristic works in many cases, there are situations in which it will fail, as shown in Figure 11. To handle these cases, the user can manually fragment a part so that the cross-section assumption holds for the fragment that actually fits into the enclosing component.

In general, non-adjacent parts in the stack can overlap, such as in Figure 12. As a result, we cannot simply propagate depth values outwards from the root in a single pass. Instead, we cast depth assignment as a constraint satisfaction problem. For any two non-interlocking parts in the same stack, we add a constraint that the part closer to the

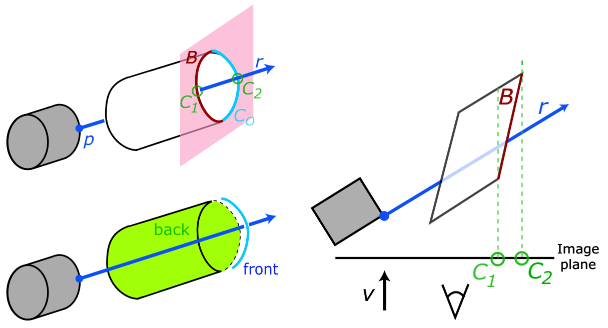

Figure 10: In this case, B is oriented such that the fragmentation assumptions do not hold (left top). Without any user intervention, the system computes an incorrect fragmentation (left bottom). This top-down view of the scene clearly illustrates that B is in front of r at C_1(right). To obtain the correct result, the user can tell the system to invert the fragmentation computation.

near end of the stack be layered in front of the other. For two parts that do interlock, we add a constraint that the inner part be layered in front of the back fragment and behind the front fragment of the outer part. If the interlocking relationships are consistent, we solve this system of inequality constraints using local constraint propagation techniques [6]. Otherwise, the constraint solver informs the user of the inconsistency.

One feature of our authoring framework is that it gracefully handles cases in which one or more of our assumptions are violated. Since the system is organized as a collection of semi-automatic tools, it does not force the

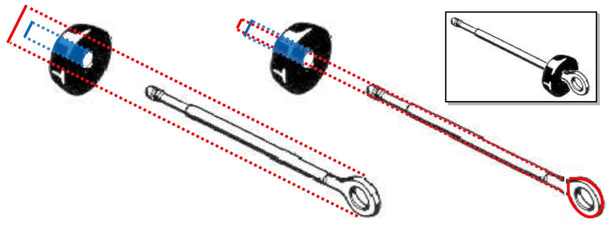

Figure 11: A failure case for the cross-section heuristic that tests whether or not two parts interlock. The cross-section of the push rod shown in red does not fit within the cross-section of the hole in the dust boot shown in blue (left). In general the heuristic fails when the inner part contains a bulbous end that does not fit within the outer part. We can resolve this case by manually fragmenting the push rod (right). Now the cross-section of the thin stem of the rod fits within the cross-section of the hole in the dust boot (inset).

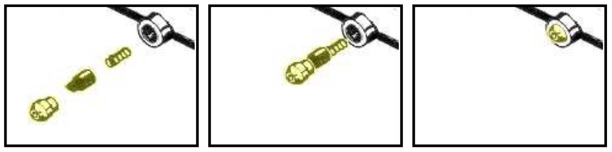

Figure 12: Multiple parts in a stack can interlock. Here all three highlighted parts interlock with the hole. Our semi-automatic depth assignment algorithm uses constraint propagation to automatically choose depth values for all of these parts so that they are properly layered between the front and back fragments of the hole.

user to choose either a fully automatic process or a completely manual interaction. Instead, the system can fluidly accept manual guidance at any stage in the authoring process. For instance, if the system is unable to find the proper fragments because one of the fragmentation assumptions is invalid, the user can manually divide a part into front and back pieces and then use the automatic depth assignment tool to infer a layering. Similarly, if the system guesses incorrectly whether or not two parts fit together, the user can first explicitly specify the correct interlocking relationship and then use the system's constraint solver to assign depth values.

3.5 Adding annotations

As an optional final step, the user can annotate individual parts with labels and add guidelines that indicate explicitly how parts move in relation to one another. For each part that requires a label, the user specifies the appropriate label text. To indicate where a label should be placed, the user clicks to set an anchor point (typically on or near the part being labelled), and then drags the label to the desired position (Figure 13). When the diagram is laid out, the system keeps the offset between the label and its anchor constant so that the label moves rigidly with its corresponding part. To make the association between a part and its label explicit, we render a line from the center of the label to its anchor point. To create a guideline, the user selects two parts and then drags out a line that connects them in the desired fashion. Each endpoint of the line is treated as an anchor that sticks to its corresponding part. As a result, the guideline adapts appropriately as the parts move. By default, guidelines are rendered as dotted lines.

4 Viewing

To display dynamic exploded view illustrations, we have developed software that supports a number of useful interactions to help the human viewer extract information from the diagram.

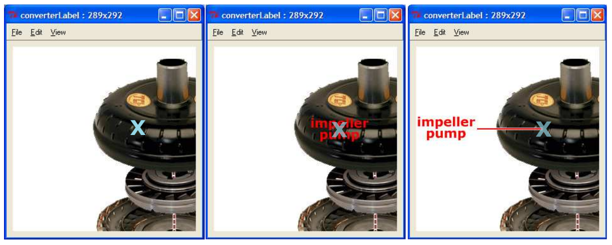

Figure 13: To position a label, the user clicks to set an anchor point (left). By default, the system centers the label on the anchor (middle). The user then drags the label to the desired position (right).

4.1 Layout

To lay out the parts in a diagram, we again use a local propagation algorithm [6] that works by traversing the stack hierarchy in topological order, successively computing and updating the position of each part based on its predecessor and current offset. Although local propagation cannot handle cycles, this is not an issue because our stack hierarchies form a tree, as mentioned in Section 3.1. Once all part positions have been calculated, the system renders each part and its fragments at their specified depths. The user can also enable label and guideline rendering in the viewing interface to display annotations. To prevent visual clutter, the system only renders labels and guidelines whose anchor points are unoccluded by other parts.

4.2 Animated expand/collapse

The viewing program supports a simple but useful interaction that allows the viewer to expand or collapse the entire diagram with the click of a button. To produce the desired animation, the system smoothly interpolates the current offset of each part either to its fully expanded or fully collapsed state, depending on which animation the user selects.

4.3 Direct manipulation

To enable the human viewer to focus on the interactions and spatial relationships between a specific set of parts without seeing all of an object's components in exploded form, our system allows the user to selectively expand and collapse portions of the diagram via constrained direct manipulation. After selecting a component, the viewer can interactively modify its current offset by dragging the part outwards or inwards within its stack. The manipulation is constrained because a part can only move along its explosion axis, no matter where the user drags.

When the user initiates this interaction, the system first records where the selected part is grabbed, as shown in Figure 14. As the user drags, we compute the projection

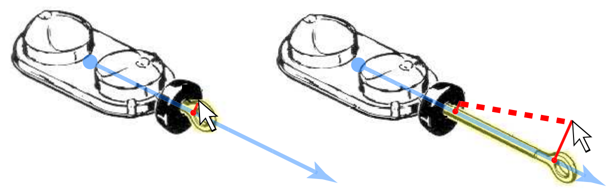

Figure 14: The user clicks and drags to directly expand the stack. The system projects the initial click point onto the explosion axis (left). As the user drags, the tip of the cursor is projected onto the explosion axis, and the current offset of the part is adjusted so that it lies on the projected point (right).

of the current mouse location onto the explosion axis. The system then sets the current offset of the selected part such that the grabbed point slides towards this projected point. If the user drags a part beyond its fully collapsed or expanded state, the system tries to modify the current offsets of the predecessor parts to accommodate the interaction. We consider each predecessor in order until we get to the root, so that a part will only move if all of its descendants (up to the manipulated component) are either fully collapsed or expanded. Thus, the user can effectively push a set of parts together or pull them apart one by one. Three frames of interactive dragging are shown in Figure 1.

4.4 Part search

In some cases, the viewer may want to locate a part that is hidden from view when the object is in its fully collapsed state. Instead of expanding the entire diagram and then searching visually for the component in question, our system allows the user to search for a part by looking up its name or picture in a list of all the object's components (Figure 15). The viewing software expands the object to reveal the requested part as well as the parts immediately surrounding it in the stack in order to provide the appropriate context.

5 Results

We have already presented example diagrams of a master cylinder (Figure 1), a converter (Figure 2), and a phone (Figure 15) that were created using our system. Another example depicting a car (Figure 16) demonstrates how our image-based framework can easily support arbitrary rendering styles. In this case, the car is rendered using markers.

Not surprisingly, we found that the time required to author a diagram depends on the number of parts in the object. The car and converter objects contain just a few parts, and as a result, each of those examples took approx-

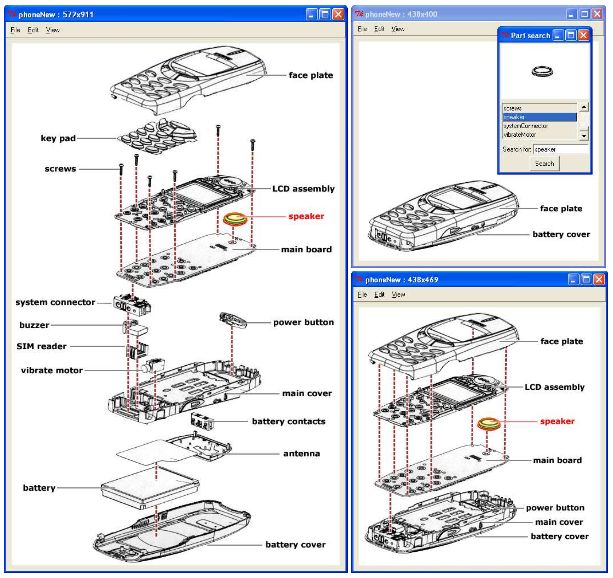

Figure 15: Visually searching for a particular part can be difficult if the object contains lots of parts (left). Here the viewer finally finds and selects the speaker for closer examination. In its fully collapsed state, the diagram is less cluttered, but it is impossible to see internal parts (right top). We provide an alternative search dialog that allows viewers to search for any part by name or by picture. Here, the viewing software expands the phone to reveal the speaker as well as the parts immediately surrounding it (right bottom).

imately five minutes to create. For the master cylinder and phone, we spent roughly fifteen minutes segmenting the original diagrams into parts. Although both of these objects contain many parts that interlock in complex ways, we manually fragmented only two parts for the master cylinder and eight parts for the phone, six of which were screws. In both cases, we used semi-automatic fragmentation to resolve the rest of the interlocking relationships. Once we obtained the correct fragmentation, depth assignment was completely automatic for the master cylinder. For the phone we had to manually assign a depth value for one of the fragments. Fragmentation and layering took roughly thirty minutes for the master cylinder and forty minutes for the phone. Most of this time was spent sketching cavity curves and providing manual guidance when necessary.

When viewing the objects, we found the direct manipulation interface to be extremely effective for conveying the spatial relationships between components. This was especially true for the master cylinder and phone because they contain many parts. While the layering contributes

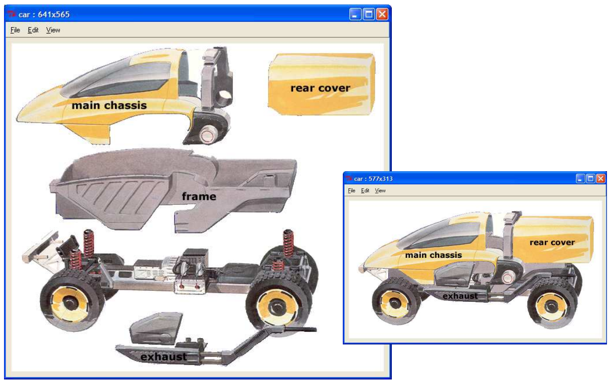

Figure 16: Interactive car diagram in its fully expanded (left) and fully collapsed configurations (right).

to the illusion of depth, the act of manipulation itself produces a sensation of physically pushing components together or pulling them apart one by one. This direct interaction clarifies and reinforces the relationships between parts. Furthermore, we found the ability to pull apart specific sections of each illustration very useful for browsing a localized set of components. In addition to reducing the amount of visual clutter, the ability to expand selectively makes it possible to view a diagram in a small window. For instance, the static phone illustration does not fit on a 1024x768 laptop screen when viewed at full resolution because the vertical stack is too tall in its fully exploded state. However, in our interactive phone diagram, we can easily view the phone on this display by expanding only a subset of the object's parts.

6 Future work and conclusion

There are many opportunities for future work in the domain of interactive diagrams. Here, we mention a few that seem particularly interesting:

Arbitrary explosion paths. To achieve a more compact exploded view layout, illustrators sometimes arrange parts using non-linear explosion paths that are often indicated with guidelines. With our system's constraint-based layout framework, it would be a relatively simple extension to support arbitrary, user-specified explosion paths.

Dynamic annotations. Although our system currently supports labels and guidelines, the way in which these annotations are laid out (i.e., as constant offsets from specific parts) is not very sophisticated. The main challenge here is determining how to arrange this meta-information dynamically to take into account the changing layout of an interactive diagram.

Emphasis. It might be useful to provide diagram authors with image-based tools for emphasizing and deemphasizing particular parts of the depicted object. These tools might be something like intelligent filters that take into account the perceptual effect of performing particular image transformations. Emphasis operations could also be used at display time to highlight important parts.

Semantic zooming [3]. For extremely complicated objects, it could be useful to introduce multiple levels of detail that would allow the viewer to interactively control how much information is presented for particular portions of the subject matter.

Depth cues. We have noticed that interactive diagrams created from 2D images can sometimes have a "flattened" appearance where layers overlap. It might be possible to automatically render simple depth cues (e.g., drop shadows) when viewing the diagram to clarify the spatial relationships between these layers.

Exploded views are crucial for explaining the internal structure of complicated objects. Interactive digital diagrams are especially important because they allow the viewer to extract specific information from an illustration by dynamically modifying the way in which the subject matter is presented. In this paper, we have described a novel framework for creating and viewing interactive exploded view diagrams using static images as input. Specifically, we presented a set of authoring tools that facilitates the task of creating such diagrams, and we described a viewing program that enables users to better understand spatial relationships between parts and the overall structure of the object.

References

[1] Maneesh Agrawala, Doantam Phan, Julie Heiser, John Haymaker, and Jeff Klingner. Designing effective step-by-step assembly instructions. In *Proceedings of SIGGRAPH 03*, 2003.

[2] William A. Barrett and Alan S. Cheney. Object-based image editing. In *Proceedings of SIGGRAPH 02*, 2002.

[3] Benjamin B. Bederson and James D. Hollan. Pad++: A zoomable graphical interface system. In *ACM User Interface Software and Technology Conference Proceedings*, 1994.

[4] Marcelo Bertalmio, Guillermo Sapiro, Vicent Caselles, and Coloma Ballester. Image inpainting. In *Proceedings of SIGGRAPH 00*, 2000.

[5] Stephen Biesty and Richard Platt. *Incredible Body*. Dorling Kindersley, 1998.

[6] Alan Borning and Richard Anderson. Indigo: A local propagation algorithm for inequality constraints. In *ACM User Interface Software and Technology Conference Proceedings*, 1996.

[7] Andreas Butz. Anymation with CATHI. In *Proceedings of AAAI/IAAI '97 in Providence / Rhode island*, pages 957–962. AAAI Press, 1997.

[8] Francis D. K. Ching. *Design Drawing*. John Wiley and Sons, 1997.

[9] Antonio Criminisi, Patrick Perez, and Kentaro Toyama. Object removal by exemplar-based inpainting. In *Proceedings of IEEE CVPR 03*, 2003.

[10] Elaine Driskill and Elaine Cohen. Interactive design, analysis and illustration of assemblies. In *1995 Symposium on Interactive 3D Graphics*, pages 27–33. ACM Press, 1995.

[11] Jean-Daniel Fekete, Erick Bizouam, Eric Cournarie, Thierry Gafas, and Frederic Taillefer. TicTacToon: A paperless system for professional 2D animation. In *Proceedings of SIGGRAPH 95*, 1995.

[12] James D. Foley, Andries van Dam, Stephen K. Feiner, and John F. Hughes. *Computer Graphics. Principles and Practice*. Addison-Wesley, 1990.

[13] P. Litwinowicz. INKWELL: A 2.5D animation system. *Computer Graphics*, 25, 1991.

[14] Michael McGuffin, Liviu Tancau, and Ravin Balakrishnan. Using deformations for browsing volumetric data. In *IEEE Visualization*, pages 401–408, 2003.

[15] Paul Mijksenaar and Piet Westendorp. *Open Here: The Art of Instructional Design*. Joost Elffers Books, New York, 1999.

[16] Riaz Mohammad and Ehud Kroll. Automatic generation of exploded views by graph transformation. In *Proceedings of IEEE AI for Applications*, pages 368–374, 1993.

[17] E. Mortensen and W. Barrett. Interactive segmentation with intelligent scissors. *Graphical Models and Image Processing*, 1998.

[18] Chris Niederauer, Mike Houston, Maneesh Agrawala, and Greg Humphreys. Non-invasive interactive visualization of dyanamic architectural environments. In *Proceedings of SIGGRAPH I3D 03*, 2003.

[19] Richard Orr and Moira Butterfield. *Nature Cross-Sections*. Dorling Kindersley, 1998.

[20] A. Raab and M. Rüger. 3D-ZOOM interactive visualization of structures and relations in complex graphics. In *3D Image Analysis and Synthesis*, pages 87–93, 1996.

[21] Thomas Rist, Antonio Krüger, Georg Schneider, and Detlev Zimmerman. AWI A workbench for semi-automated illustration design. In *Proc. of Advanced Visual Interfaces*, pages 59–68, 1994.

[22] Felix Ritter, Bernhard Preim, Oliver Deussen, and Thomas Strothotte. Using a 3d puzzle as a metaphor for learning spatial relations. In *Graphics Interface*, pages 171–178, 2000.

[23] Dorée Duncan Seligmann and Steven Feiner. Automated generation of intent-based 3D illustrations. In *Proceedings of SIGGRAPH 91*, pages 123–132, July 1991.

[24] Michael A. Shantzis. A model for efficient and flexible image computing. In *Proceedings of SIGGRAPH 94*, Computer Graphics Proceedings, Annual Conference Series, pages 147–154, July 1994.

[25] T. A. Thomas. *Technical Illustration 3rd Edition*. McGraw Hill, 1978.

[26] Edward Tufte. *Visual Explanations*. Connecticut: Graphics Press, 1997.