

BROWN UNIVERSITY
Department of Computer Science
Master's Project
CS-89-M5

“Visualization of Disassemblies of Mechanical Objects”

by
Robert C. Zeleznik

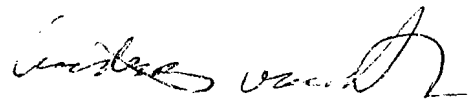
**Visualization of Disassemblies of
Mechanical Objects**

submitted by

Robert C. Zeleznik

in partial fulfillment of the requirements for the
Master of Science Degree
in Computer Science at Brown University

May 18, 1989

A handwritten signature in dark ink, appearing to read 'Andries van Dam', is written over a horizontal line.

Andries van Dam, *advisor*

Acknowledgements

I would like to thank Andries van Dam and Brown University for providing continuing support for students to pursue research in computer graphics. John Hughes was supportive and provided the initial directions for this project. Henry Kaufman was extremely helpful in assisting the development of the interactive software that was necessary for this project. Thanks also to Paul Strauss, Mike Natkin, and the rest of the Brown Computer Graphics Group for their efforts in building and supporting BAGS, and their thoughtful advice. Finally, thanks to Lisa Winterbottom for her support and understanding throughout this project.

This work has been sponsored in part through research contracts with International Business Machines Corporation, National Cash Register and Digital Equipment Corporation, and equipment grants from Sun Microsystems, and Hewlett-Packard.

Contents

	Introduction	1
1	Disassembly Descriptions	2
2	Existing Methods	3
2.1	Static Images	3
2.1.1	Graphic Styles	3
2.1.2	Detail Regulation	5
2.2	Video Based Images	10
3	Interactive Model-Driven Disassembly Systems	11
3.1	Interactive Animations	12
3.2	Dependency Encoding	14
4	Disassembly System Requirements	16
5	PUDGE - Disassembly Language	18
5.1	PUDGE Design	20
5.1.1	Part Objects	20
5.1.2	Actions Objects	21
5.1.3	Apply Objects	21
5.1.4	Context Objects	22
5.1.5	Depend Objects	23
5.1.6	Predicate Objects	23
5.2	PUDGE Examples	23
6	PUDGE Analysis	25
7	Future Directions	27
8	Conclusion	29
	References	31

INTRODUCTION

The degree to which our educational resources bear a strikingly similar form to those resources that existed a thousand years ago is somewhat disturbing. Indeed, our most vital educational device is the book, an effective but outdated tool. This is not a disparagement of books, but an observation that modern technology has made available more expressive and stimulating mechanisms for disseminating information. In response to this technological gap, the notion of electronic books has received increasingly more attention, and an ever growing body of literature is forming upon that topic, as represented, for example, in [YAN85].

In essence, an electronic book is the integration of various media and technology into a single interactive entity. Integration is accomplished by linking conceptually related blocks of information together in order to form webs of association, called hypertext and hypermedia. Thus, a reader of an electronic book is not restricted to viewing information in a linear manner as with standard books. Instead, with appropriate cues from the electronic book itself, a reader can view a page of the electronic book, and decide to follow any of a number of possible links. These links can be to video images, to interactive animations of pictorial concepts, or to further textual information and related material.

Electronic book technology is well suited to repair and instruction manuals as discussed in [FEIN82] and as evidenced by the *Movie Manual* described in [BRAN88]. The precise, goal-oriented nature of repair manuals presents a convenient metric for evaluating the effectiveness of the manual, by which current efforts have had only limited success. These efforts have provided only static drawings, or limited video animations, which cannot capture the temporal and visual nature of repair processes. In addition, it is very difficult to update standard repair manuals even if only slight structural modifications are made to the assembly.

A central aspect of maintenance and repair manuals is their focus on describing processes for how to disassemble components of an assembly, in order to repair, or replace them. Traditionally, these descriptions have been expressed through sets of line and tone art drawings with accompanying textual blurbs. Such descriptions need to be laboriously crafted by skilled artisans who precisely understand both the physical descriptions of the objects and the mechanics of the disassembly process. However, what is troublesome is not the effort expended by the artists, but rather the unnaturalness of the static, non-interactive media in which these artists must design process descriptions. Alternative means for describing disassemblies have been explored with the most promising being the application of laser technology. Video disk, CD-ROM, optical disk, and DVI technology provide the power to describe disassembly processes in a more natural way because of their ability to store movies of actual disassemblies. Although, laser disk technology presents a significant improvement

over the static line and tone art drawings, it still is not adequate, especially since it offers only limited user-interaction. That is, a canned image or animation can be interactively requested, but cannot be manipulated and altered. With model driven images and animations, a user can change characteristics of the model, change how the model is viewed, and generate animations that were never specifically defined by the author of the process model.

The purpose of this thesis is to present a general methodology for describing and interactively animating a subset of repair manuals - the disassembly processes of mechanical devices. This approach represents assemblies as pseudo-realistic 3D models and disassembly descriptions by dependency graphs that can be executed to produce animation scripts. Thus in theory, the user is able to configure and generate animations of whichever aspects of the disassembly process they find interesting.

This thesis will begin by exploring in detail the existing means for presenting disassembly descriptions. Each of the existing primary techniques used to convey disassembly information will be reviewed in order to elicit its shortcomings, and its advantages. After looking at these methods, this thesis will describe the need for, and the objectives of, an interactive disassembly animation system. Finally, this thesis will present and review an executable disassembly description language, PUDGE, that was developed to provide interactive disassembly animations.

1 Disassembly Descriptions

In considering how to describe disassemblies of mechanical objects, two critical issues must be addressed. First, what is the character of the people that the descriptions are aimed at, and second what is an acceptable medium for providing these descriptions? Clearly, the latter issue is influenced by the former.

Descriptions of mechanical disassemblies are often directed towards individuals with little or no prior knowledge of the devices that they are disassembling. This derives from the pervasive desire of individuals to, "fix it themselves", even if they have no training whatsoever in disassembling and repairing devices. However, such descriptions may also be used by more experienced persons who essentially know how to perform the disassembly, but who need to reference some information on some subset of the entire disassembly. In either case, designing a single description uncomplicated enough for a novice and yet sufficiently descriptive for an expert is a non-trivial task. Compound the disparity between novices and experts with the inherent differences between individuals, and it becomes clear that a single description of anything is not ideally suited for all people. Thus, the appropriate paradigm for describing disassemblies must consider the potentially vast differences between individual's

tastes and background knowledge, and allow for dynamic user-directed disassembly descriptions.

2 Existing Methods

The concern with formally describing the disassembly procedures for mechanical objects is by no means novel, although these descriptions have become increasingly important, given the greater complexity of modern object designs. The accepted methodology for describing these procedures has been to produce a manual that exhaustively describes all disassembly procedures with the primary explicative technique being a large collection of line and tone art drawings for each stage of the disassembly (see Figure 2.1). More recently, such traditional specifications have been augmented or replaced by video based descriptions. It is important, then, to explore the techniques used in each of these methodologies to determine to what degree the success of these methods is limited by their media.

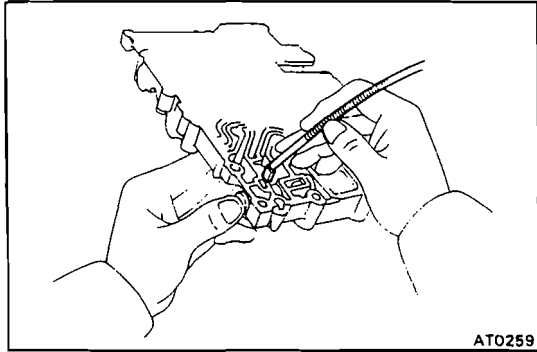
2.1 Static Images

Static images, with few exceptions, are the primary graphic illustrations in all mechanical disassembly descriptions. As such, a well-refined corpus of techniques exists for the purpose of making these images as clear as possible. One might expect that the more realistic media, such as photographs, would necessarily be better suited for technical illustrations. However, as will be discussed next, this has not been borne out in the case of disassembly descriptions, which are almost always characterized undetailed drawings.

2.1.1 Graphic Styles

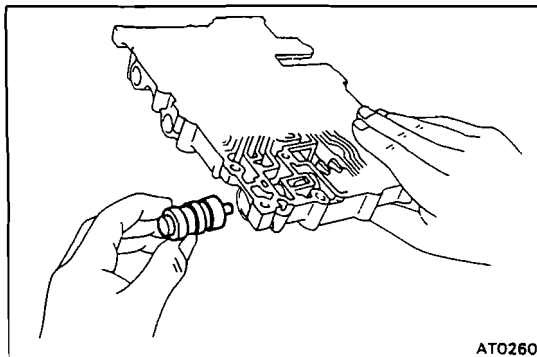
The three primary modes of graphic presentation in maintenance and repair manuals are photography, tone art, and line art. However, the most striking characteristic of the static images used in technical descriptions is that they are often composed solely of line art, which depicts only the visible peripheries of objects. Quite often, in instances where more detail is desirable, tone art, which shades objects in even gradations, is employed to enhance visual perception. In any case, exactly how much detail is optimal is often an irreconcilable issue because perception is a highly individualistic procedure [FRIE80]. The following is a summary of the virtues and deficiencies of each of the three primary graphic styles.

Although the natural expectation is that the realism of photographs would be a paramount advantage, the least realistic technique - line art - is in actuality the commonest. This arises from a number of factors, not the least of which is its economy [WIL89].

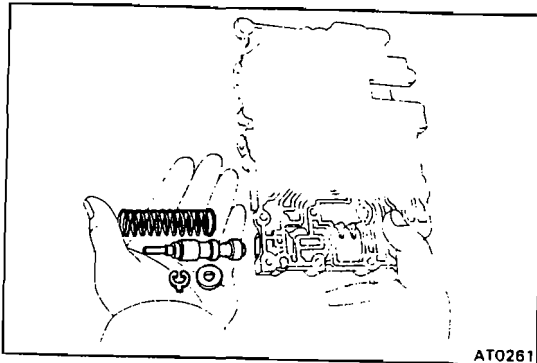


5. REMOVE PRIMARY REGULATOR VALVE

(a) Remove the retainer with a magnetic finger.



(b) Remove the sleeve and plunger.



(c) Remove the spring and valve.

NOTE: Note the number of adjusting rings.

Figure 2.1 Standard disassembly description [TOY86]. Note the use of elision.

However, cost aside, the simplicity of line art images itself is a principal cause in its extensive presence in repair manuals as the rendering method of choice. Line art drawings suffer only slightly from their lack of 3D cues and generally are no less expressive than more realistic pictures. Depth and shading cues can be effected by simply regulating the width and darkness of lines. Further contour cues can be provided by cross hatching surfaces with closely set lines.

Perhaps, the motivation for utilizing line art is precisely that it offers an elegantly unadorned means for expressing both detail and reference context information. This judiciously restrained view toward detail is appropriate, since what is being expressed is an object or a process and not an image. The merit of pictures is judged by how well they communicate and not by their artistic value. Thus, the picture's purpose is to help the reader to correlate the steps of an abstract process description with an actual mechanical object.

In comparing tone art drawings to photographs, it is evident that the visual complexity of the pictures greatly diminishes, but without a commensurate decrease in the amount of reference information that the pictures provide. In fact, the wealth of minutiae in photographs may be distracting, since important components cannot be suitably emphasized, nor can unimportant ones be effectively de-emphasized. In addition to detail, tone art provides the limited capability of artificially inducing translucency upon otherwise opaque surfaces (see Figure 2.1.1.1). As with line art, tone art affords the artist with the capacity to present unrealistic, yet expressive, axonometric and orthographic views of objects. Thus, tone art drawings represent a compromise between the fine detail of photographs and the utilitarianism of line art drawings.

Notwithstanding that the emphasis on process must be stressed over pictorial realism, photograph-based descriptions are often extremely beneficial. Not only do photographs overcome any problems with "artist's misconceptions", but they may also provide important surface, shadowing, texture, and 3D accuracy that is not readily available through other methods. Photographs are especially important for some intricate objects which may be confusing if rendered in a less precise fashion. In addition, exploitation of air-brush techniques can be very effective in suppressing excess detail without loss of realism.

2.1.2 Detail Regulation

It is clear that, although too much detail may be distracting, some level of detail is essential for an effective description of a mechanical object; otherwise, text could be used by itself. Therefore, it is critical that a description be able to provide an adequate means for accessing sufficient detail when necessary but not at the expense of providing too much detail

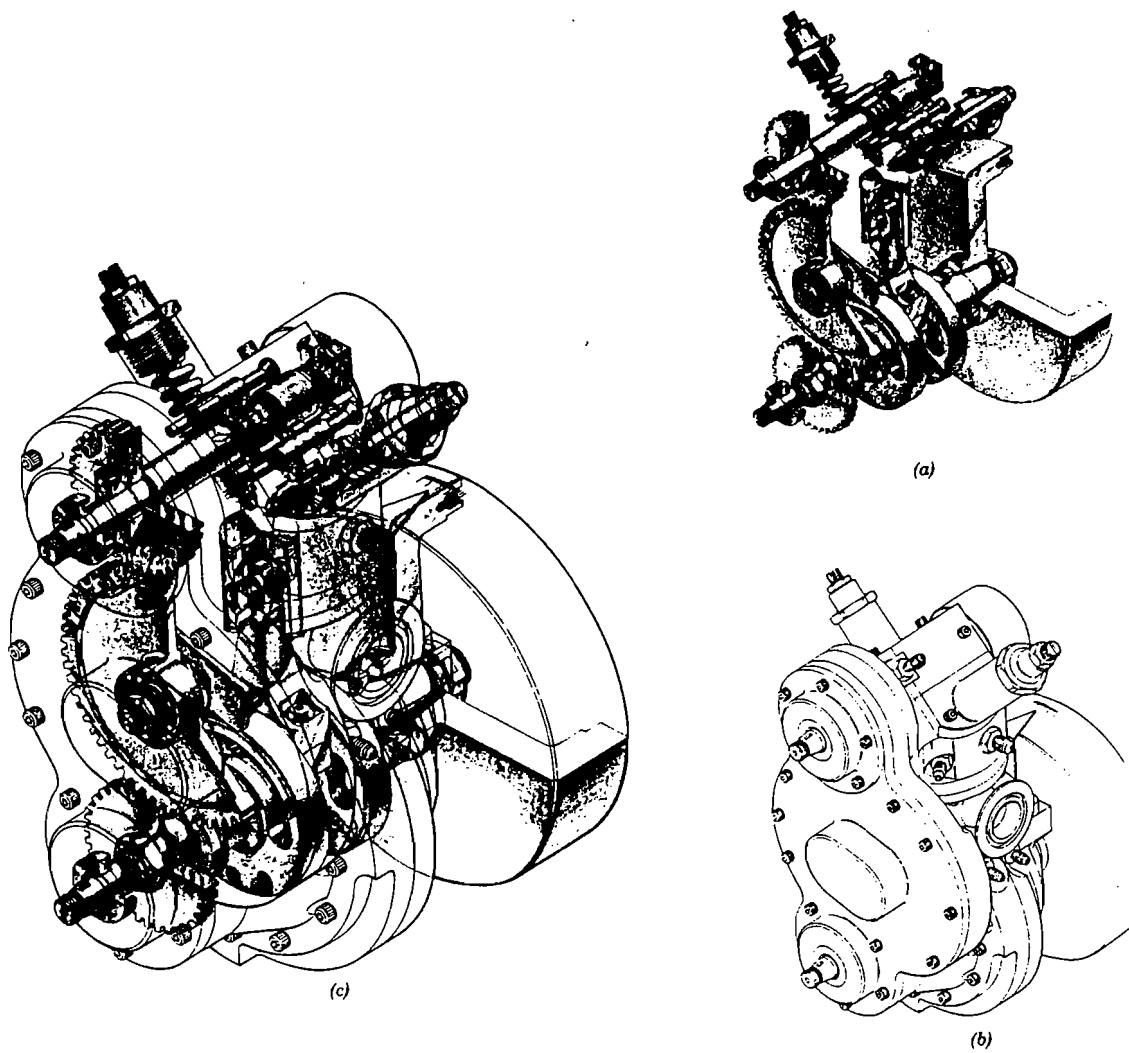


Figure 2.1.1.1

Standard rendering methods [MAGN70].

- a) tone art rendering of assembly
- b) line art rendering of assembly and its casing
- c) combination of tone and line art to render a partially transparent casing around the assembly

in each drawing. Clearly, it would be most convenient if readers were able to select the levels of detail with which they were most comfortable, preserving the option to switch immediately between more and less detail without needing to refer to another image. Standard static images are unable to accommodate such dynamic views by their nature, and so the point of how much detail to include in each drawing is an extremely important design criterion. In fact, a variety of techniques are commonly used just for the purpose of providing various levels of detail, and therefore, varying levels of context information [MAG70]. Among these techniques are *elisions*, *omissions*, *cut away* and *inset* views of objects.

The purpose of elisions in technical graphics is to assert that a spatially significant object exists, but is not of central importance to the drawing. Elisions usually come in the form of a de-emphasized partial outline of an object without any of its 3D detail - essentially a silhouette. Elisions are important since they often refer to components that are easily recognizable, but are not important to the process. Such components provide a spatial context merely by their presence, and thus, nearly all of the detail of these components can be eliminated without forfeiting any of their descriptive function.

Similarly, cut away views of an object are used when parts of interest are shielded from view by some opaque object. In this case, the shielding object is critically important to the picture; but it is not satisfactory either to omit this object or to allow it to conceal the visual description of the parts that are important to the process. In these situations, just enough of the shielding object is "cut away", with visual cues to express that the part has been cut (*e.g.* a ragged edge on a cut away surface), so that the objects of interest can be seen.

Omissions on the other hand are used to reduce visual detail by omitting objects that are not part of the process and that do not provide orientation cues. Although this is probably not a problem for experienced individuals, such omissions may confuse someone not well versed in technical explanations.

Inset views are another convenient way of providing adequate context information without admitting too much detail. Inset views usually assume the form of a drawing of an assembly with an inset picture of some small part of that assembly. Arrows or lines are used to indicate that portion of the full assembly to which the inset picture corresponds.

Since providing context information is so important, and yet, as much detail as possible is eliminated from each drawing in order to emphasize methodology, most disassembly descriptions offer another means of accessing context information. Where each drawing in a disassembly description can be thought of as a local road map, it is usually worthwhile to provide a more global road map in order to really understand what is happening in the larger scope. The technique for providing this global road map is accomplished through the

presentation of a *parts explosion*. In a parts explosion, all of the constituent parts of an object are depicted in such a way, that the most important spatial relationships between parts are preserved as seen in Figure 2.1.1.2. That is, each part is drawn so that its position relative to the parts adjacent to it in the assembly is preserved, and can be clearly identified. In this way, the reader can develop a general impression of which parts will depend upon and interact with which other parts during disassembly. Parts explosions are extremely important since, to the indoctrinated, they can often suffice as the only visual description of a mechanical object.

Incorporated into drawings of parts explosions and, more importantly, process descriptions are *meta-objects*[FEIN87]. These objects are the primary visual descriptions of temporal dependencies and procedures. Again, since what is being described are processes and not pictures, it is essential that a mechanism exists for describing the temporal relationships between process elements. Static drawings can only show the state of a system at discrete instants, such as before and after something has been moved. This is sufficient only for processes in which actions are simple enough that they can be inferred from a series of snapshots of the disassembly process. As actions become more complex, it becomes necessary to present visual descriptions of the actions, and perhaps the interdependence between actions, themselves. Meta-objects furnish an efficacious mechanism for describing processes, as they are a concise visual abstraction of the temporal nature of actions. The set of meta-objects commonly consists of arrows for indicating the direction in which an object should be pulled or twisted, and tools (or hands) for depicting how an action can be accomplished. Of course, text is also used in cases where visual descriptions are not amply illustrative.

Thus, the approach of centering disassembly descriptions about static images is primarily one of coercing an intrinsically temporal, 3D concept into a static, 2D paradigm. Extensive techniques have been developed which make the framework of drawn art and photography workable. Nevertheless, the fact remains that static 2D images are a sub-optimal environment in which to present disassembly descriptions.

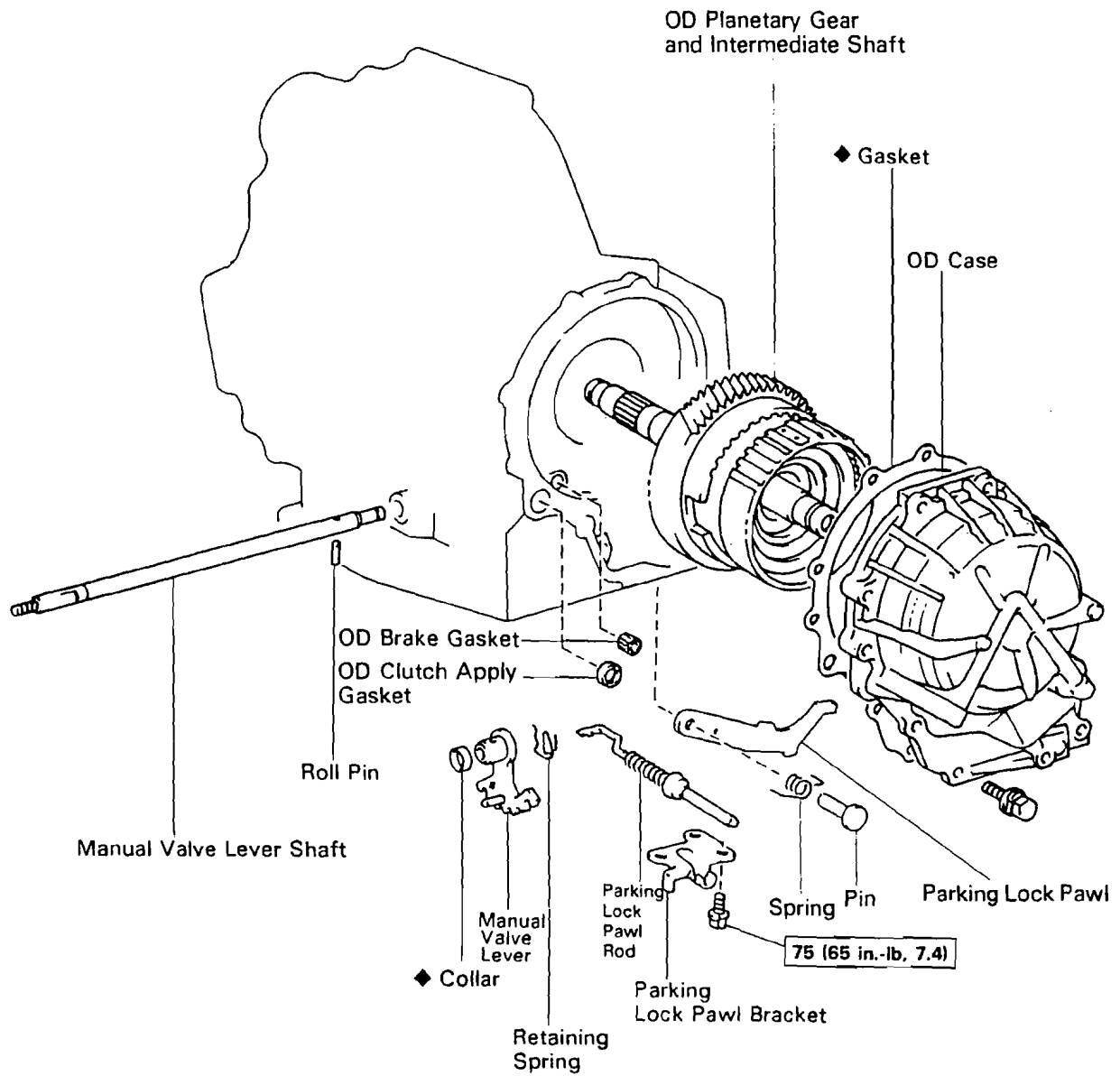


Figure 2.1.1.2 Standard “parts explosion” [TOY86].

2.2 Video-based Images

More recently, a considerable amount of attention has been placed upon video technology, especially that based on video disks [MCC84]. Historically, people have shied away from video based solutions because of both the cost of mastering, and the technological problems associated with retrieving information from vast video stores. However, with the development of increasingly cheaper CD-ROMs and near random-access to megabytes or even gigabytes of information, mass storage of realistic video footage has become a possibility. The feasibility issues of storing numerous video snippets of disassemblies and accessing them in real-time is no longer a primary concern. (Only the storage of realistic video will be considered, because storing animated video is merely a subset of the more general interactive disassembly descriptions that will be considered in section 3).

However, given the previous arguments against the inclusion of too much detail, one might wonder why realistic disassembly videos would even be considered as an improvement over more standard disassembly descriptions. In this regard, it should be readily apparent that videos overcome two of the major expressive impediments of static 2D art. First, videos are capable of explicitly capturing the essence of the steps of a disassembly, both in terms of how an action is accomplished and what the temporal relationships are between different actions. Second, realistic videos provide a natural means for capturing 3D information.

Being able to watch an actual person performing a disassembly of some object is appealing, because, with video, one would also make use of a corresponding sound track in order to offload and accentuate a portion of the process cues. The greater ease with which videos are able to capture 3D information should make disambiguation of parts in complex scenes a non-issue. However, these gains do not come without cost. As aforementioned, realistic levels of detail are not necessarily useful as they can detract from the processes that are being specified. In the case of videos, the detail problem is compounded since the tools, hands, and arms used to perform the disassembly cannot be elided or omitted. Achieving useful camera angles may be quite difficult under some circumstances without access to any methods of detail control.

Due to the information overloading associated with realistic images, one would expect drawings to still be used, especially as lead-ins to disassembly segments. DVI recognizes this need by providing hardware support for integrating computer graphics with live video in real-time. However, this support is primarily limited to overlaying annotations, arrows, textures and frames on top of video images. In general, though, more attention needs to be paid to producing animations from disassembly models than to the editing realistic video images. Certainly, a parts explosion of an intricate device provides more information at a glance than a

realistic picture of the same object would provide. This is because animated models allow concepts to be visually encoded in a way that a realistic video cannot. The very nature of realistic video makes it difficult to express a concept except through its application to a particular instance. That is, a realistic video expresses by example, whereas, animated models can express by example or by abstraction, since they are not rigidly bound to realism.

Finally, the notion of filming all possible disassemblies of some device seems as unacceptable as precomputing tables for arbitrary mathematical expressions. Any given system might have an exponential number of possible disassemblies (see Figure 2.2.1), and in those cases, it would hardly seem feasible to film videos of all such disassemblies. Also, as updates need to be made to the disassembly descriptions, previous filmed videos become useless. Such time and space profligacy would only be justifiable if more conservative measures were unavailable or were far less expressive.

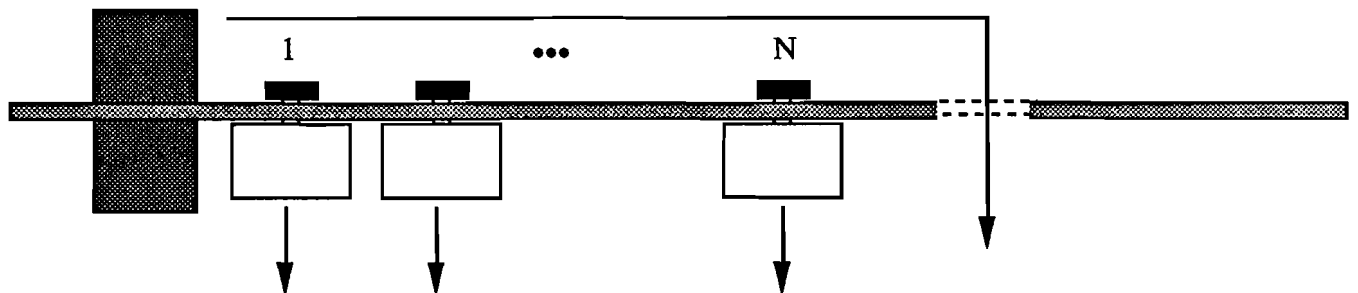


Figure 2.2.1 In this example, each of the objects from 1 to N can be independently removed by pulling them down. However, the block on the left can only be removed by pushing the remaining of the N blocks to the right. The number of animations is $O(2^N)$ since each block can be either present or removed.

3 Interactive Model-Driven Disassembly Systems

Given the deficiencies of existing methods for describing disassemblies, it is incumbent to find a more acceptable system for describing disassemblies. The most crucial elements of such a system are that the system be both interactive and able to vary the detail at which assemblies are displayed, in addition to being efficient. Unlike static systems which discourage variety of presentation, dynamic systems demand diversity as the basis for their appeal to the user's individuality. Furthermore, the system should expand the concept of parts disassemblies to include non-standard methods. That is, disassembly manuals generally depict only how one would disassemble a device if all the parts were working properly. If a bolt had rusted to a screw, it is left up to the reader to decide if there is some easier way to continue than by somehow breaking the screw. Although, it may not be possible or necessary to consider all defects in a model, commonly occurring defects should be added to the disassembly model.

3.1 Interactive Animations

The motivation for developing model-driven interactive animations is that a single description of a disassembly is not adequate for all individuals. Some people are better equipped to comprehend 3D detail than others. Others, having strong procedural and mechanical acuity, can develop comprehensive mental models of a disassembly from only a few examples. Interactive animations provide a means for generating multiple descriptions of a disassembly process in a natural and concise manner. A single disassembly description provides all the information necessary to describe disassemblies for the beginner as well as the expert. Conversely, avoiding multiplicity is extremely difficult when the descriptive medium is either hardcopy or video disk, because no interpretable model exists, only an image.

It should be clear, that the development of an interactive animation system depends upon the maintenance of a 3D model of the real assembly, in addition to a description of the disassembly process. These 3D models can be rendered in real-time on a high-performance workstation, especially those that provide hardware support for 3D rendering. By sustaining a 3D object model at all times, it becomes possible for the user to intervene at any point in an animation of a disassembly process. Thereby, the user can customize various components of the visual description, such as the camera position or the rendering method for objects. Because of the maintenance of an object model, moving the camera or switching between line art, flat shading, and more realistic lighting and shading models is potentially trivial. In this way, the essence of line art drawings can be maintained in concert with the techniques of more realistic video animation systems. In addition, if the structure of an assembly were to change, then only one 3D model would have to be revised, instead of redrawing all the images of a repair manual, or re-shooting all disassembly processes. Certainly, the true realism of realistic videos is lost, and the techniques of elisions, and cut away views are perhaps made more difficult to realize, yet it seems that no expressive power has really been conceded.

In fact, cut aways can be achieved to some degree by a straightforward application of clipping planes. This would not exactly duplicate to cut aways in line art, but the effect would be quite similar. On the other hand, because of the precise nature of rendering systems, elisions are more difficult to express dynamically. It is both hard to define and to render arbitrary levels of superficial detail for objects. Therefore, omissions or excess detail might have to suffice. However, in going to a 3D model in which it is expected that objects will be viewed in many ways, a new technique for controlling context can be utilized - that of *partial transparencies*. The use of partial transparencies allows for an object to be present in order to provide context cues, but not so as to obscure objects of interest. In static images and especially tone art, partial transparencies are much less effective as they tend to ambiguate

pictures, rendering them more difficult to understand (Compare the objects of Figure 2.1.1.1). However, in an interactive environment, partial transparencies can be controlled by the user, both in terms of what is transparent and the level of its opacity. In so doing, the ambiguities of static transparent images are clarified.

Another useful technique, that of inset views, takes on a greater power in an interactive disassembly system since inset views can be instantiated at user discretion. The user could elect to view a scene from a number of different camera positions, where each of these views can be mapped onto an inset portion of the one window or onto a different window altogether. Although these cameras would view the same assembly, they could be independently manipulated by the user.

The meta-objects utilized in static images can still be applied in the case of interactive animations but are of lesser significance. As was the case for video disk animations, the problem that meta-objects were primarily designed for had already been effectively relieved by the mere fact that animations are not static. That is, the description of how an object should be moved, which needs to be described abstractly with arrows in the case of static images, can be viewed explicitly in an animation. In addition, the use of hands and tools to show how parts should be manipulated is no longer necessary, since much of that information can be inferred from the actions themselves. It is still possible to incorporate arrows and tools (hands would of course be more difficult) into an interactive animation as needed.

Additional user interplay would typically be in the form of interacting with the system in some natural fashion in order to change attributes of the camera(s). With the ability to move the camera, the user can navigate through a device as much as is necessary in order to understand the context of the disassembly. If a disassembly involved small parts, the user could zoom into the assembly without losing any detail because of the existence of the underlying object model. Conversely, the user might zoom out in order to get a better idea of the larger scope in which the disassembly were occurring. Further customization would be available in the form of being able to omit objects by simply turning them off or by reviewing/replaying the animation at different speeds.

It is critical to note that perhaps the most expressive of the static image descriptions, the parts explosion, becomes even more expressive in the venue of animation. Instead of abiding by the lack of 3D information inherent to static parts explosions, this information becomes self-evident with the ability to move the camera. The relative spatial information of static images is made much more vivid through the ability to move about any given piece of the parts explosion.

Thus, by merely having the ability to interact with the system instead of being a

passive viewer, the user is afforded a much richer learning environment. The ability to customize the disassembly to personal tastes means that a single disassembly system can be used by people of vastly different backgrounds without sacrificing any expressive power or detail.

3.2 Dependency Encoding

Interactive model-driven animations are the first step toward truly modelling a disassembly. However, they still do not provide the user with the ability to specifically request a disassembly of interest. The user can customize existing animations to an arbitrary level of detail but cannot request to see how some particular part would be disassembled (unless there happened to be a pre-existing animation for just that particular disassembly). Neither could the user ask questions of the "What if?" format. The problem is that only the description of instances of a disassembly (*i.e.* disassembly scripts) have been stored but no description of the disassembly process itself (*i.e.* something to produce a disassembly script) has been modelled. Consequently, it is necessary to develop some method of encoding the information necessary to generate disassembly animations from a knowledge-based model of disassembly processes, as explored in the rudimentary work of Neiman's CAD animations [NEI82].

Modelling processes can be done by either modelling all of the possible instances of that model, which is akin to the video based methods, or by modelling the interrelationships between the movements of parts under various conditions of disassembly. As discussed, the former is insufficient and so this thesis concerns the latter.

The most important characteristic of modelling a disassembly process and not its instances, is that the user should not be given complete latitude to perform arbitrary actions upon any part. A disassembly system is not intended to demonstrate the operations of a device, nor is it a physical simulation. Such license would not facilitate understanding of how a device should be disassembled. Thus, the disassembly model should not center on general simulation abilities, but rather it should only permit of actions that are didactic and that lead toward a specific goal - the disassembly of some subset of an assembly. That is, it would not seem useful to allow the functioning of a Rubik's cube, as opposed to its disassembly, to be modelled in a disassembly system, unless achieving a certain configuration of the cube were necessary in order to disassemble it.

Therefore, it would not be adequate to encode a system that simply enforced the non-intersection of parts and forced the user to make all the decisions concerning which parts to move and how to move them. Further, it would be too naive for a system to blindly inspect the geometry of an assembly and thereby determine some means of removing its component parts.

This is because geometric solutions may not in fact correspond to viable real solutions. Consider, for example, that in a particular assembly, a screw might be removable by virtue of its geometric properties alone. Yet, such a removal might not be possible in the actual assembly, because no screw driver might be able to fit into the assembly in order to unfasten the screw (some other part would have to be removed first). Or perhaps it might be necessary that some set of parts might be required to be moved in specific order (perhaps for electrical or safety reasons) even though there may not be any geometric reason for that ordering. A number of other examples exist along these lines, and therefore, disassembly descriptions must include some abstract knowledge, unattainable through geometric reasoning, about the type of object that is being modelled.

It is possible that a geometric reasoning system could be used in unison with a knowledge base, such that, the knowledge base could request geometric information to evaluate conditional geometric assertions. Simple geometric reasoning, such as determining whether some part can move along one of its principal axes without intersecting any other part, is not particularly difficult and will be reviewed later as a possible means for automating the description of disassemblies. Unfortunately, non-trivial geometric reasoning would probably be an exceedingly difficult task that would not be realizable in real-time for a number of hardware generations. In any case, such a geometric reasoner would probably be most efficient if it cooperated with a knowledge base especially in order to describe correctly animations for parts such as screws and bolts. That is, it would be difficult to deduce how to rotate and translate a correctly modelled screw based on the geometry of its grooves. If the screw were not modelled correctly, then a geometric reasoner would not be able to discern the difference between a screw and a peg, even though both might have very different disassembly implications.

From the difficulties inherent in geometric reasoning, it seems that a far more tractable real-time solution would have all of the necessary motion and dependency information between parts explicitly described in some formal language. This formal description of the dependencies between parts, the disassembly process, would by nature be executable. Such a description would have to allow for the possibility that parts might be removed in different fashions depending upon the overall context of the assembly. Essentially, such a language would describe a dynamic dependency graph that would change whenever a part were moved, or whenever the user specified context information (*e.g.*, a bolt is rusted to a screw). It is possible that this formal description itself might be the output of an expert system, but that is a less crucial issue than the actual formal descriptions of disassemblies themselves.

The notion of storing disassembly information as a formal language description allows

an interactive system to interpret those rules and generate animations, perhaps exponential in number, from a compact rule set. Combined with the capabilities of an interactive disassembly system, such a dependency system could truly describe disassembly processes.

4 Disassembly System Requirements

Given the previous motivation for interactive model-based disassembly animations systems, it is necessary to outline the functionality that such a system must encompass. This can be most easily explained by looking at example assemblies and determining what is necessary to describe their disassembly processes.

Consider first the assembly of Figure 4.1. In this assembly, there is only one possible disassembly for any part. To remove the small cube, the lid and the screw must be removed first. Therefore, the formal description of the disassembly of this assembly is trivial. A simple static dependency graph could be constructed that would explicitly state the motion dependencies for each object. This graph would not need to consider any context-sensitive information since the topology of the dependency graph would remain constant no matter what had previously been disassembled.

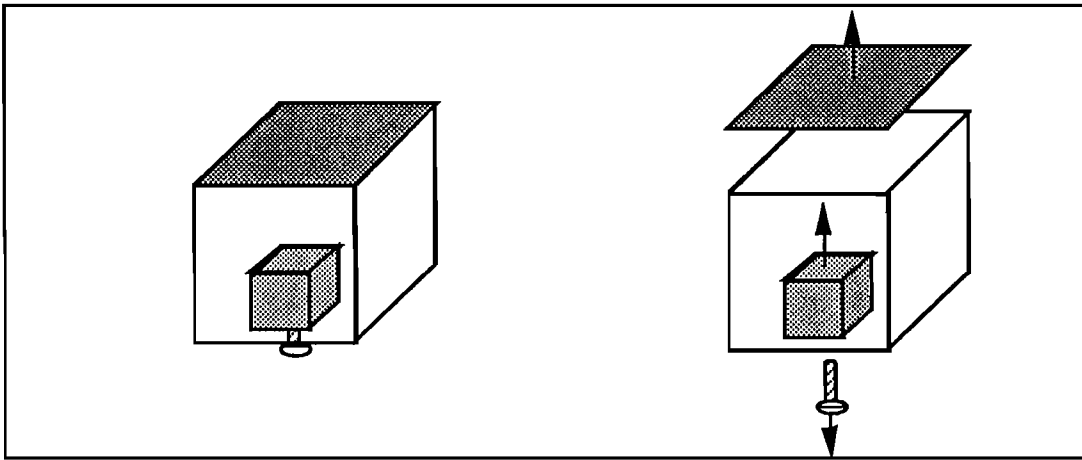


Figure 4.1 Context-free disassembly.

However, the assembly of Figure 4.1 is not representative of the majority of mechanical devices. Figure 4.2 depicts a more interesting assembly in which a single static dependency graph must allow for context-sensitive dependencies to be expressed. The removal of the cone from the system of Figure 4.2 can be exacted in 3 distinctly different ways. The standard method would be to open the latch, rotate the lid, and lift the cone. However, if the hinge had already been removed then the lid should not be rotated. Instead the latch should be opened, the lid should be lifted off, and then the cone should be lifted. Additionally, the user might specify that the latch does not open, and so a third disassembly

would remove the hinge, then slide the lid away from the latch, and then lift the cone. In this way, the context of the assembly defines what actions should actually be executed.

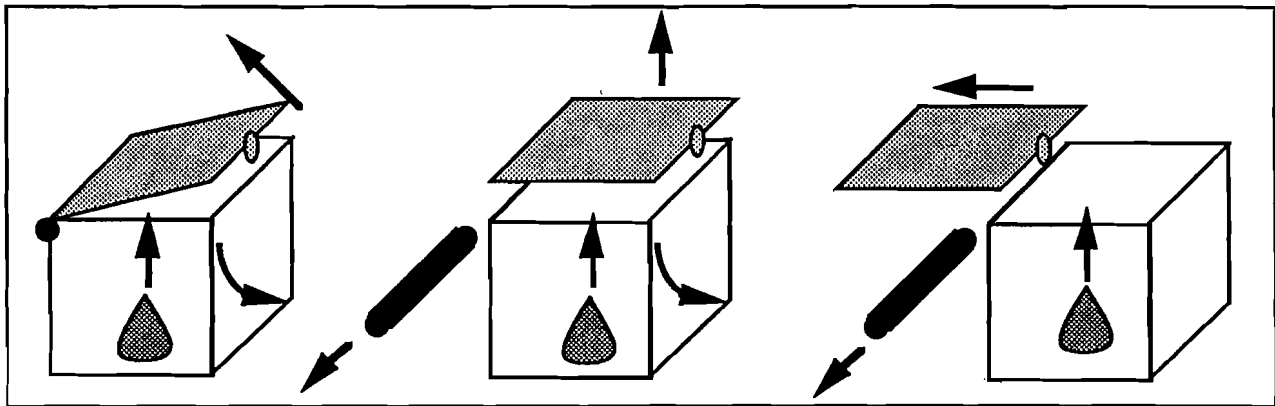


Figure 4.2 Context-sensitive disassembly.

Allowing for context-sensitive dependencies is a significant step forward, however it is still not adequate for some devices. Consider the system of Figure 4.3. In this case the device is quite simple, but what is interesting is how dependencies need to be dynamically transferred from one object to another. In this example, by definition any of the three disks can be removed by pulling it to the left, and then up. But if the plate is lifted up, then all the disks above it also must be lifted. Therefore, lifting any disk must depend upon lifting the disk above it. Although removing any disk can be accomplished without forcing any other actions to occur, the removal of a disk must cause the dependency graph to change so that the disk below the removed disk will then depend upon the disk above the removed disk. This system could be described with a static dependency graph, but it would then be necessary to consider all possible combinations of disks above the plate and encode specific dependencies for each of those combinations.

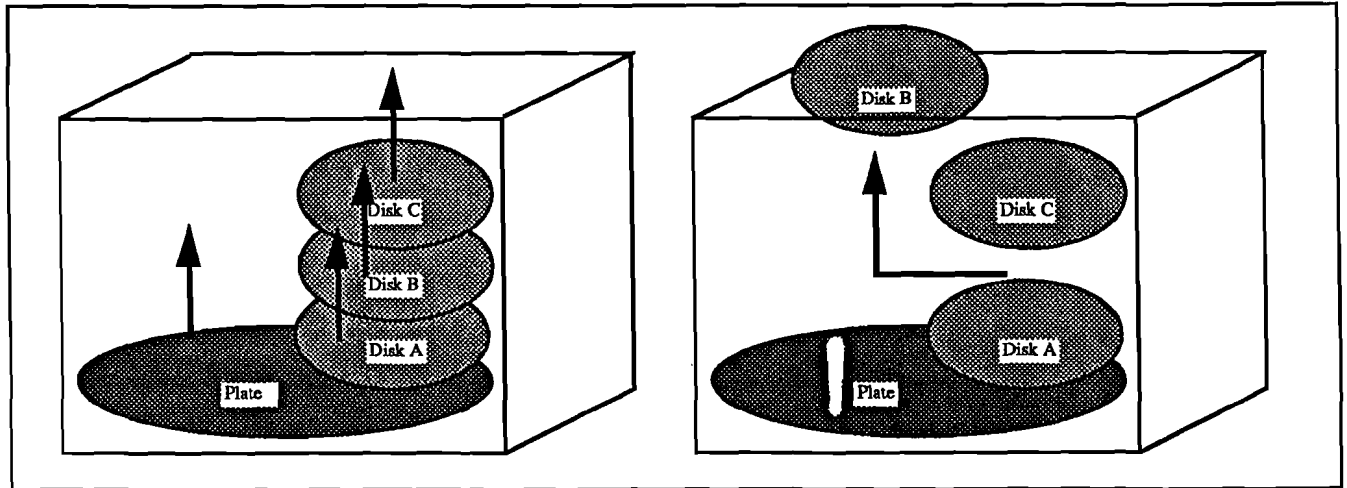


Figure 4.3 Dynamically changing disassembly.

5 PUDGE - Disassembly Language

With the preceding in mind, PUDGE, a language for describing disassemblies, was developed and, in fact, is executable. The general design of PUDGE centers on the notion of a dynamic dependency graph. As previously described, the problem with static dependency graphs is that they are only manageable when nodes of the graph depend, in different contexts, upon only a small number of other nodes. However, if the number of dependencies for a particular node is dependent upon the problem size, then a static graph must be completely enumerated, and can become exponential in size. Dynamic dependency graphs provide a more intuitive and space efficient way to encode such graphs. In a dynamic dependency graph, nodes correspond either to the application of an action to a part, or to some complex dependency. An action is only executed after all of its dependencies have been satisfied (executed). Since dependency references can be made through variables, the topology of the dependency graph can change dynamically, and, therefore, can be resolved in possibly a number of different ways depending on the overall status of the disassembly.

For example, consider the dependency graph of Figure 5.1. In this case, the dynamic nature of the dependency graph is evidenced only by the presence of context nodes. The dependency that follows a context node will only be executed when the associated context of the context node is True. Contexts can be arbitrary boolean expressions about the state of the assembly (e.g. *partB* moved and *partC* opened). Therefore, lifting *partA* depends upon lifting *partC* if the context node *Ccontext* is True. Similarly for shifting *partB* and lifting *partD*. If all of the contexts are True, then each dependency is resolved at the same time

(executed) before lifting *partA* (simultaneity can occur because all actions are buffered).

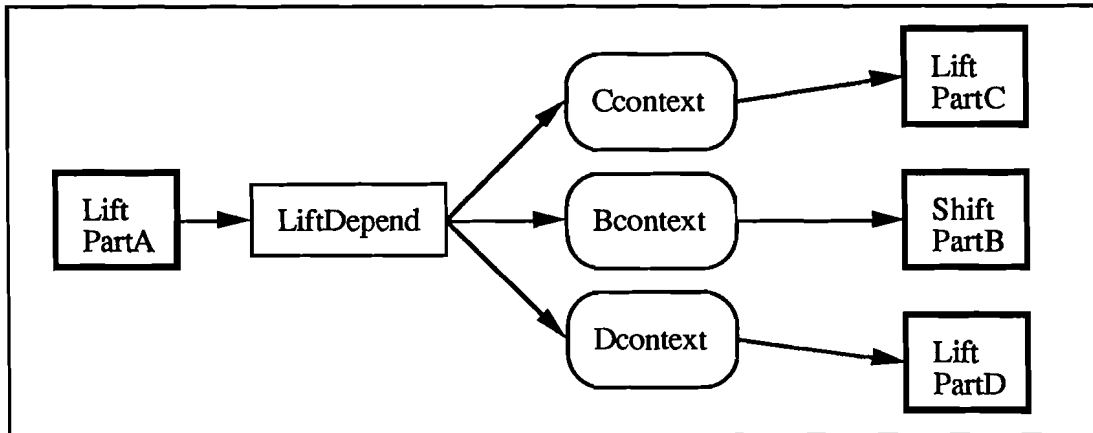


Figure 5.1 Thick squares are actions. Thin squares are dependency nodes. Rounded squares are context nodes. Arrows specify dependency links.

Now consider the dependency graph of Figure 5.2. In this example, lifting *partA*, is initially specified as depending upon shifting *partB* where reference is by a variable. At some time X, because of other disassembly events, the very same dependency for lifting *partA* might correspond to the dependency for lifting *partC* and *partD* simultaneously.

Transformation of the dependency graph can occur because a dependency's reference variables (essentially pointers to actions or dependencies), and these pointers can be modified at any point during the disassembly.

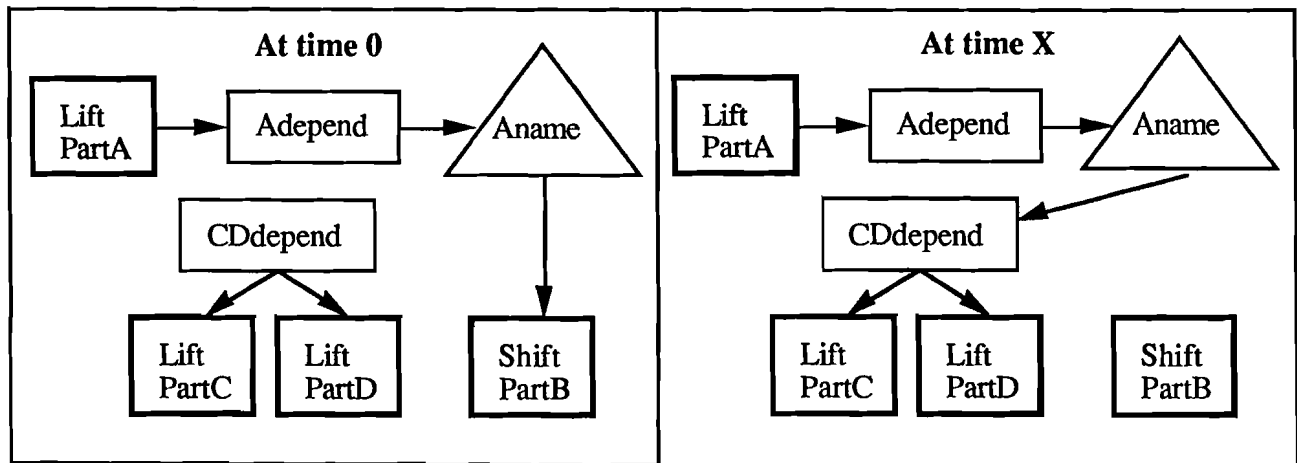


Figure 5.2 Thick squares are actions. Thin squares are dependency nodes. Triangles are variables. Arrows specify dependency links.

Therefore, one executes a disassembly by specifying a starting point, which is either an action node, or a dependency node (these nodes are roots of subgraphs of the dependency

graph and are seen as squares in Figures 5.1 and 5.2). PUDGE then traverses the graph by following dependency links that are appropriate to the current context. Whenever an action node is reached, it is not executed until after all of its dependencies have been resolved. In addition, when an action is executed, it may be necessary to update the dependency graph, since the state of the assembly has changed.

5.1 PUDGE Design

PUDGE identifies six basic entities as being important to the description of disassemblies: Parts, Actions, Applies, Contexts, Dependencies, and Predicates. In following an object-oriented design, each of these basic types is implemented as an object and will be referred to as such throughout this description. There are additional Variable objects which are used to maintain state information. These Variables are only declared within the scope of Part objects, and can be one of three types: *Real*, *String*, and *Ptr*. Real Variables correspond to C language floats, String Variables correspond to C language character arrays, and Ptr Variables are pointers to other Variables where reference is by name (a string). Evaluating a Ptr Variable returns the evaluation of the Variable that it points to. Evaluating a String or Real Variable simply returns its value. Thus, the nature of the dependency graph can change by using ptr Variables to reference different String Variables that contain names of other objects.

PUDGE is built on top of the BAGS animation system and utilizes its basic animation data structures. These data structures are managed by a package called SCENE which is a database for object shapes and animation definitions [STR88]. The PUDGE language is tightly integrated with the BAGS object modeler and uses its previewer as the means for interactively viewing animations and defining the parts that comprise assemblies. The BAGS previewer is designed to run on an HP835 Turbo SRX workstation using the Starbase display list package, and on the Stellar GX1000 workstation using the PHIGS+ display list standard. The BAGS object modeler has the ability to interactively make objects invisible, and will have the ability to change the transparency of objects and move clipping planes. In addition, the modeler provides the user with the ability to define up to 4 different camera views of an animation and to move any of these cameras while an animation occurs.

The following is a description of the PUDGE language that is used to define the disassembly processes and to generate animations of disassembly instances.

5.1.1 Part Objects

Part objects define names for parts in an assembly. Part objects can be animated only if there exists an object in the SCENE database with the same name as the Part object, but Parts

are not required to correspond to such a viewable SCENE object. Thus, the geometric definitions of parts are properly confined to the animation system. In addition, Variables can be defined within the scope of a Part. Thus, the definition for a Part and its Variables follows the syntax:

```
name    { <variable>:<type> [ = value]
        ... }
```

Each Variable is then accessed by specifying a Part as the scope (prefix) for the Variable, as in `Lid.some_var`. The optionally specified value is the initial value of the Variable and therefore must match the type of the defined Variable. If the Variable type is `Ptr`, then the value can only be the name of some other Variable.

5.1.2 Action Objects

Action objects define actions which can then be applied to parts. Actions are restricted to translations along an object axis, or rotations about an object axis. However, an Action can also correspond to an Action of the same name in the SCENE database. Thus, although it is expected that most Part motions can be described using only orthogonal motion descriptions, the more general concept of a SCENE Action is available for non-standard motion descriptions. The syntax for an Action definition is:

```
name          or
name { com = value ... } where   com is TX, TY, TZ, RX, RY, RZ, VIS
                                and T = translate, R = rotate, and X, Y, and Z are axes
```

If only an Action name is specified, then it must correspond to a SCENE Action, otherwise, the named action is constructed so that it will perform the specified set of rotations, translations, and visibility changes. The values for each of the actions can either be a constant real number, or it can be the name of a Variable which contains either a real number or recursively is a pointer to a real number. The coordinate system in which an Action transpires is the coordinate system specified by the Apply object that applies the Action.

5.1.3 Apply Objects

An Apply object applies a specified Action to a specified Part. If the Apply does not explicitly specify a coordinate system, then the Action is executed such that its coordinate system refers to the world coordinate system. However, an Apply can specify a coordinate system in which an Action should occur by specifying the name of some Part. That is, the coordinate system is set to be the object based coordinate system of the specified Part. In addition, an Apply can define a set of Variables that should be updated after every time that

Apply is executed. This is the method by which dependencies, contexts, and motion parameters can be dynamically changed. Finally, temporal dependencies can be defined for an Apply by defining an object that must be evaluated prior to evaluating the Apply. The dependency object can be another Apply, a Depend object, or a Variable that recursively points to an Apply or a Depend object. In any case, the Apply will not start its Action until all of the dependency object's Actions have terminated (if the dependency fails, then the Apply fails). The duration of an Action is optionally specified in parentheses after Action name. The syntax for an Apply is:

```
[?][$(partAname)]partBname:actionName [(time)] { <partC>.<variable> = value
... } [ <- [?]dependency_object ]
```

The value specified can be a Variable or an atomic value, with the only requirement being that the type of the Variable being updated must match the type of the value specified. if \$(partAname) is specified, then the Action is applied with respect to the object based coordinate system of partAname (\$ by itself indicates that the coordinate system should be that of the part being moved). If a '?' precedes the Apply definition or the dependency object, then the Apply or dependency object will be executed only once, succeeding thereafter, and returning the time at which it was first executed.

5.1.4 Context Objects

Context objects provide the mechanism for determining whether a dependency is applicable at some point during the disassembly. When a Context object is encountered during a disassembly, it returns either the *time* that the associated context became True, or it returns False. Contexts can be composed of any boolean combination of Variable comparisons. Precedence is from left to right except as specified by parentheses. Thus, the syntax for a Context object is:

```
name ((exp op exp) op exp...) where op is [&& | ||] and exp is:
[!] var [> | < | == | >= | <= | !=] value or
partlact or contextObj (the name of another Context object)
```

Values again can be either a constant or a Variable with the requirement that their type match the type of the Variable being tested. The valid Variable operations are >, <, ==, >=, <=, !=, and expression operations &&, ||, ! have meanings equivalent to the same symbols in the C language. In addition a special test is available to determine whether an Action has been applied to a Part. The test Part!Action succeeds if Action has been applied to Part and fails otherwise.

5.1.5 Depend Objects

Depend objects provide a general method for specifying dependencies between Applies and other Depend objects. A Depend object consists of a set of Context objects (or Variables that refer to Context objects). Associated with each Context object is a set of dependencies. Again, each dependency can be either an Action object, a Depend object, or a Variable that refers to an Action or Depend object. Whenever a Depend object is encountered during a disassembly, it tests each of its Context objects. For any Context object that has an associated context which is True, its set of associated dependencies are executed. Each dependency is executed simultaneously, allowing for the possibility of concurrent actions to take place. The syntax of a Depend object definition is:

```
name      { [?]dependency1 ... } <- context_object_1
          ... { [?]dependencyN ... } <- context_object_n
```

Any dependency that is preceded by a '?' will be executed only once. Thereafter, it will succeed and return the time it was first executed. A Depend object will fail if either no context objects are applicable, or any dependency fails when its associated Context object was True.

5.1.6 Predicate Objects

Predicate objects are the principal interface to the user of the disassembly system. A user invokes a Predicate object by selecting its name from a list of applicable Predicates with a graphical locator. Similarly, selecting the display of any Part implicitly generates an evaluation request for the extraction Predicate of that Part. Evaluation of a Predicate causes its associated dependency, normally a specification for the removal of some part, to be executed. Each Predicate object can be thought of as the root of a subgraph of the dependency graph. Thus, by invoking a Predicate, traversal of the dependency graph begins at the indicated root, and animation of the disassembly commences.

Predicates should be defined for any context changing events that the user is allowed to specify, in addition to defining extraction Predicates for each part in the assembly. The syntax of a Predicate definition is:

```
name { dependency } where dependency is either an Action or a Depend
```

5.2 PUDGE Examples

With the preceding in mind, it is appropriate show how PUDGE can be used to describe the disassemblies of the objects depicted in Figures 4.2 and 4.3.

Re-considering the disassembly of Figure 4.2, there are 3 possible disassemblies for

the cone. The PUDGE disassembly description defaults to the standard method of removing the cone unless either the hinge has already been removed or the user has specified that the latch does not work. The PUDGE disassembly description for this system only takes limited advantage of PUDGE's dynamic nature. All context-dependent dependencies are explicitly stated.

```

PARTS                                /* specify Parts and Variables */
Lid      { in_place:float = 0 }
Cone     {}
Box      {}
Hinge    {}
Latch    { rusty:float = 0 }

ACTIONS                             /* specify possible Actions */
rotate   {RZ = 90}
lift     {TY = 3}
open     {RZ = -90}
rust     {}

APPLYS                               /* specify Applies for actions */
?$Lid:rotate {Lid.in_place = 1} <- latch_open /* rotating lid depends on opening latch */
?$Lid:lift   {Lid.in_place = 1} <- latch_open /* lifting lid depends on opening latch */
?$Lid:slide  {Lid.in_place = 1}
?$Latch:rust {Latch.rusty = 1}              /* one way to make latch rusty */
?$Hinge:lift {}
?$Cone:lift  {} <- lid_in_way

CONTEXTS                             /* specify Contexts */
lid_in_place (Lid.in_place == 0)           /* True if the lid blocks the cone */
latch_ok     (Latch.rusty == 0)
latch_not_ok !(latch_ok)
lid_rotatable (!Hingelift && latch_ok && lid_in_place)
hinge_in_place !(Hingelift)

DEPENDS                               /* specify Dependencies */
lid_in_way { lid:rotate } <- lid_rotatable /* either try to rotate the lid */
           { lift_lid } <- lid_in_place     /* or remove it by some other means */
           {}
latch_open { Latch:open } <- latch_ok
lift_lid   { Lid:lift } <- latch_ok         /* either lift lid if the latch is working */
           { Lid:slide } <- latch_not_ok    /* or slide the lid if the latch is broken */

PREDICATES
remove_lid {lift_lid}
remove_cone {Cone:lift}
remove_hinge {Hinge:lift}
rust_latch {Latch:rust}

```

Re-considering the assembly of Figure 4.3, it is necessary to dynamically change the dependency graph whenever a disk is removed. The following is the PUDGE description for this system:

PARTS

```
DiskA      {prev:string  = Plate
            next:string  = DiskB
            up:string    = DiskAlift}
DiskB      {prev:string  = DiskA
            next:string  = DiskC
            up:string    = DiskBlift}
DiskC      {prev:string  = DiskB
            up:string    = DiskClift}
Plate      {next:string  = DiskA}
```

ACTIONS

```
up          { TY = 4 }
remove      { TX = -1  TY = 4  VIS = 0 }      /* move left, then up, then make invisible*/
```

APPLYS

```
Plate:up    { }
DiskA:up    { }
DiskB:up    { }
DiskC:up    { }
DiskA:remove { DiskA.prev@next = DiskA.next } /* previous disk depend on next disk */
DiskB:remove { DiskB.prev@next = DiskB.next }
DiskC:remove { DiskC.prev@next = "" }          /* previous disk depends on nothing */
```

CONTEXTS

DEPENDS

```
diskAlift   { DiskA:up DiskA.next@up }        /* lifting A depends on lifting what's above A */
diskBlift   { DiskB:up DiskB.next@up }        /* lifting B depends on lifting what's above B */
diskClift   { DiskC:up }
plateup     { Plate:up Plate.next@up }        /* lifting Plate depends on lifting what's above it */
```

PREDICATES

```
remove_diskA {diskA:remove }
remove_diskB {diskB:remove }
remove_diskC {diskC:remove }
remove_plate {plateup }
```

6 PUDGE Analysis

The current implementation of PUDGE successfully illustrates the power of an interactive, user-directed animation system. User interaction is intuitive and clearly specified, both in terms of generating disassembly animations and of regulating detail. Disassembly descriptions can be written for a wide range of mechanical devices and previewed in real-time. Indeed, in most all cases, the limiting factor for the domain of mechanical disassembly descriptions is the inability to produce a correct 3D model. Complex assemblies and objects comprised of elastic and inelastic tubes, intricate surfaces, and parts that can bend are all examples of devices that are not only difficult to model, but also are difficult to animate in real-time. As well as limiting the domain of describable assemblies, generation of 3D models is also the major temporal bottleneck in producing a disassembly animation.

Nonetheless, the limitations imposed by the modeling and animation system are not particularly astringent, since the class of assemblies composed of more simple rigid bodies is

still quite rich. In addition, The overwhelming majority of assemblies which can be modeled can also be described in a straightforward fashion in PUDGE. Those examples of devices that require complicated PUDGE descriptions are usually somewhat contrived; although, complexity also arises from trying to encode too many contingencies into a disassembly description. The complexity encountered when encoding contingencies must be viewed in light of the fact that such encoding is a new concept. Previously, describing contingencies would have been too complex because of its potentially unlimited scope. That is, when an assembly is not guaranteed to be in perfect working order, or when many possible similar disassemblies exist, there are too many conditions to anticipate to attempt to detail every one of them. Through the use of PUDGE, however, many of these contingencies can be condensed into just a few lines of a PUDGE description. The drawback, of course, is that the PUDGE descriptions tend to become more and more inscrutable.

However, the credit for the success of PUDGE must in large measure be attributed to its interactive nature. By itself, dynamically involving the viewer in the disassembly description is a powerful pedagogical technique [GAG87], but by further enabling a viewer to move about an object as it disassembles, 3D comprehension becomes effortless. The animation alone makes the process description self-evident, alleviating the need for arcane symbolism. Detail control, in the form of part visibilities and rendering methods, makes the system suitable to virtually any level of scrutiny. Although pseudo-realistic animations, incorporating techniques such as pattern mapping and reflections, are not currently available interactively, they can be produced on video tape. This can be accomplished by saving disassembly animations in SCEFO, and then rendering them by using other components within the BAGS environment [STR88].

Another attractive feature of PUDGE is the capability it provides for prototyping disassemblies. Currently, PUDGE descriptions can be loaded interactively, making it easy to test animations, then edit and reload the descriptions again. In addition, incomplete PUDGE descriptions can be written and executed for only subsets of an entire assembly, since the user selects which parts to view during animation. Given PUDGE's object oriented design, a natural future extension might be to allow for the interactive modification of individual PUDGE object descriptions.

As mentioned above, disassembly descriptions generally share a very standard structure, and in these cases PUDGE descriptions can be provided in a correspondingly natural manner. In reviewing automobile maintenance manuals and home repair guides, it seems that the vast majority of disassemblies can be decomposed into only a very few distinct motions. Parts are usually constrained to undergo an extremely restricted set of actions, even though

none of the part's degrees of freedom may actually be geometrically constrained. This set of actions, in nearly all cases, consists of orthogonal motions about one of the part's principal axes.

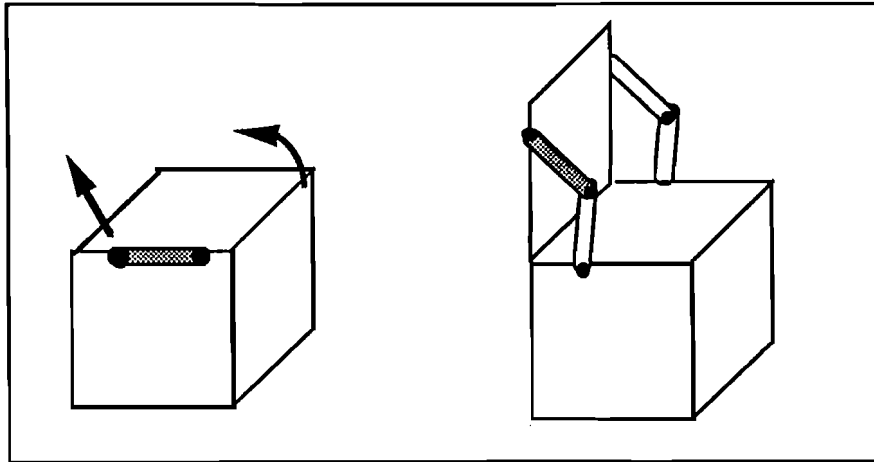


Figure 6.1 Dynamically constrained disassembly.

More complicated examples do exist, but usually are the result of dynamic motion constraints, as in the struts of Figure 3. Animation descriptions for such motions would be impracticable to produce in PUDGE, if they could be produced at all. The difficulty with constrained motion is really not a limitation of PUDGE, but of the underlying animation system. For example, opening the lid of the box in Figure 6, logically depends on the simultaneous constrained motion of the supporting struts on either side of the lid. The gray strut's movement must satisfy the constraints that the point of attachment between itself and the lid be maintained, and the point of attachment between itself and the strut below it be maintained. However, description of such constrained motion in PUDGE would not be natural or appropriate, unless constrained joint motion were an animation primitive, just as translations and rotations are. If constrained motion were primitive to the animation system, then opening the lid of the box, in Figure 6, could be described by a standard dependency and action.

7 Future Directions

In light of the expressive power of a modest language like PUDGE, it seems that future research in disassembly animation should be directed ostensibly towards the development of systems and techniques to automate the process of authoring disassembly descriptions. Such systems might utilize a variety of methods in order to reason about an assembly, and then, automatically generate a disassembly description. The resultant

disassembly description could be specified directly in a formal language like PUDGE, or might be expressed as a skeleton of procedural constructs that an animator would have to modify.

Perhaps the most tractable area for automation would be in the production of 3D exploded views of assemblies, as proposed by Feiner [FEIN82]. Automation of such views would not replace the need to automate the production of process descriptions, since exploded views may reveal very little about formal disassembly processes. However, the geometrical reasoning involved with parts explosions is much more pliant than the reasoning involved in process descriptions. This is because the animations of parts in 3D explosions would not be bound to the rigorous rules of non-intersection and correctness that process descriptions are. That is, an animation of a parts explosion does not endeavor to show how parts should actually be disassembled, but rather attempts to show the approximate spatial relationships between parts.

Consequently, an exploded view of an assembly might be generated automatically by first determining the center of the assembly. Then, starting with parts farthest from the center of the assembly, each part might move along one of its object axes that most closely approximated the ray from the center of the assembly to the part itself. Doing this in a hierarchical fashion would probably be necessary to produce appropriate results, and so it might be necessary to define sub-assemblies within the complete assembly. This would necessarily be a first approximation for animated parts explosions, because such an algorithm would not, for instance, sufficiently disambiguate parts that had occupied the same initial spatial locations (such as a screw and a washer). In addition, although it is acceptable for parts to intersect as they are being animated, it would not be tolerable for such conditions to exist after the animation had completed. Thus, additional geometric reasoning would be needed to insure that parts eventually became disambiguated. Furthermore, heuristics would have to be found to determine how far and how fast parts should move in order to produce aesthetically pleasing animations. It is possible that full automation of exploded views is not a realistic solution, but instead automation might be utilized as a tool to generate animation scripts which animators would then need to edit.

The automation of process descriptions is a far more difficult task. This arises from two points: first, that process descriptions include geometric reasoning that would be very difficult to infer from a 3D object model alone; and second, that process descriptions are not necessarily tightly coupled with the geometry of an assembly. For example, the motion of an object like a screw would seem quite difficult to infer from its geometric structure alone. The motion of other objects like snaps, would require knowledge about the degree to which objects could bend or twist. A complicated structure like a locomotive linkage might require that the

linkage be in a certain position before it could be disassembled. This would mean that the motion of the entire linkage would have to be inferred simply from its geometry. In addition to the geometric problems of automation, one must also provide a means for incorporating more arbitrary knowledge not based on geometry at all. For instance, it is conceivable that in order to take apart an object resembling a Rubik's cube, one might be required to solve the puzzle first. Such a disassembly would have no relation to the cube's geometry.

It seems evident from these arguments that automation of process descriptions cannot follow solely from the geometry of the assemblies. A reasonable solution, perhaps, might include a provision for geometric reasoning capabilities from within a knowledge based expert system. The more difficult and abstract process description knowledge could then be asserted by some knowledge base familiar with the functioning of the system to be disassembled. The knowledge base would be freed from needing explicit information about the mundane aspects of a disassembly, since they could be deduced by the geometrical reasoner. For example, the knowledge base might know only that a screw depended on there being no objects above it, along its principal axis. A geometrical reasoner would then have no problem returning the set of objects that were directly above the screw. The knowledge base would then decide for each object above the screw whether it knew how to remove those objects or whether it needed to gather more geometric information. This knowledge base might be of a structure similar to the PUDGE language description in this paper, although it would need to be equipped with a more sophisticated reasoning system.

8 Conclusion

The application of interactive 3D graphics techniques to disassembly descriptions promises a much more expressive paradigm for explaining and understanding mechanical assemblies. In a sense, such an interactive description captures the essence of bulky repair manuals in a single page of an electronic book. This paradigm is suitable for the gamut of users from novice to expert, but avoids problems with versioning, since the information flow between the system and the user is definable by the user, and stems from a single disassembly description. None of the descriptive techniques of static 2D images and realistic videos is necessarily sacrificed in advancing to 3D object models, and, in fact, new more expressive techniques become available. The methodology, presented in this thesis for describing interactive disassembly animations using PUDGE, is natural for both the viewer and for the problem.

However, the work required to produce 3D disassembly description models is somewhat more complicated than that required to produce static images and videos. Certainly,

the refinement of a language like PUDGE to include both geometric and more sophisticated language constructs would ease some of the burdensome aspects of creating process descriptions. Yet, it seems that more fruitful results will probably result from channeling future research towards the development of expert disassembly systems to automate the authoring of disassembly descriptions. Given the inaccessibility of purely geometric arguments to correct disassembly solutions, the extent to which real-time geometric reasoning can work in unison with these knowledge bases is unclear. Nonetheless, expert systems, with explicit access to some geometric reasoning and with sufficient information about a particular mechanical domain, should in principle be able to produce disassembly descriptions without human intervention.

References

- [BRAN88] Brand, Stewart. *The Media Lab*. Penguin Books, NY: 1988.
- [FEIN82] Feiner, S., Nagy, S., and van Dam, A. "An Experimental System for Creating and Presenting Interactive Graphical Documents." *ACM Transactions on Graphics*, Vol. 1, No. 1, January 1982, 59-77.
- [FEIN87] Feiner, S., *Computer Generation of Pictorial Explanations*. Ph.D, Thesis, Technical Report CS-87-30, Computer Science Dept., Brown University, Providence, RI, April 1987.
- [FRIE80] Friedman, S. and Stevenson, M. "Perception of Movement in Pictures.", In M. Hagen (ed.), *The Perception of Pictures* NY: Academic Press, 1980, 225-255.
- [GAG87] Gagne, R. *Instructional Technology: Foundations*. Lawrence Erlbaum Associates, Publishers, NJ: 1987.
- [MAG70] Magnan, G. *Using Technical Art: An Industry Guide*. Wiley-Interscience, NY: 1970.
- [MCC84] McCracken, D. and Akscyn, R. "Experience with the Zog Human Computer Interface System." *International Journal of Man-Machine Studies.*, 21:4, October 1984, 293-310.
- [NEI82] Neiman, D. "Graphical Animation from Knowledge." *Proceedings of the AAAI 82*, Pittsburgh, PA, August 18-20, 1982, 373-376.
- [STR88] Strauss, P., *BAGS: The Brown Animation Generation System*. Ph.D, Thesis, Technical Report CS-88-22, Computer Science Dept., Brown University, Providence, RI, April 1988.
- [TOY86] *1985 Toyota Camry Repair Manual*. Toyota Motor Corporation, Japan: 1986.
- [YAN85] Yankelovitch, N., Meyrowitz, N., and van Dam, A. "Reading and Writing the Electronic Book." *Computer*, October 1985.
- [WIL89] Wildbur, P. *Information Graphics*. Van Nostrand Reinhold Co, NY: 1989.