Designing Effective Step-By-Step Assembly Instructions

Maneesh Agrawala[*]
Microsoft Research

Doantam Phan
Stanford University

Julie Heiser
Stanford University

John Haymaker
Stanford University

Jeff Klingner
Stanford University

Pat Hanrahan
Stanford University

Barbara Tversky
Stanford University

Abstract

We present design principles for creating effective assembly instructions and a system that is based on these principles. The principles are drawn from cognitive psychology research which investigated people's conceptual models of assembly and effective methods to visually communicate assembly information. Our system is inspired by earlier work in robotics on assembly planning and in visualization on automated presentation design. Although other systems have considered presentation and planning independently, we believe it is necessary to address the two problems simultaneously in order to create effective assembly instructions. We describe the algorithmic techniques used to produce assembly instructions given object geometry, orientation, and optional grouping and ordering constraints on the object's parts. Our results demonstrate that it is possible to produce aesthetically pleasing and easy to follow instructions for many everyday objects.

Keywords: Visualization, Assembly Instructions

1 Introduction

Many everyday products, such as furniture, appliances, and toys, require assembly at home. Included with each product is a set of instructions showing how to put it together [Mijksenaar and Westendorp 1999]. For modular product lines, such as customizable office furniture, many different versions of the instructions are necessary. As the number of customizable products and demand for task-specific instructions increase, technology will be needed to produce instructions more cost effectively. Already there is a high incidence of poorly designed and out of date instructions.

The problem is that it is difficult and expensive to design assembly instructions that are easy to understand and follow. Since the instruction design process has not been systematized, skilled human designers are needed to produce good instructions. As a result, the process of producing instructions is time-consuming and labor-intensive. Computer support is currently limited to replacing low-level tools such as pen and paper. Most high-level design decisions are still made by human designers.

We have developed a system that provides higher-level tools for designing assembly instructions. Figure 1 depicts instructions produced with our system. A broader goal of our work is to understand how humans produce and use visual instructions. By codifying this design knowledge in computer programs, we can make it easier to
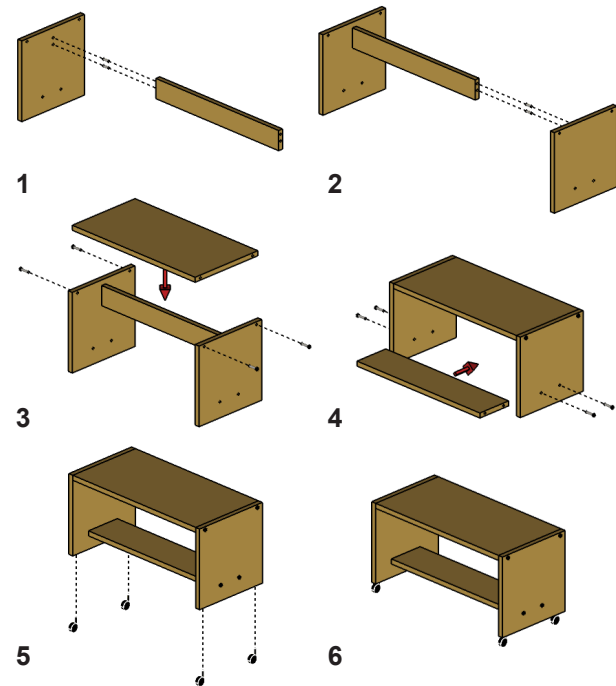

Figure 1: Assembly instructions for a TV stand. Our system plans the set of assembly operations to show in each diagram and then renders action diagrams which explicitly depict the operations required to attach each part.

produce clear drawings of 3D objects and more effective instructions [Tversky et al. Submitted].

The two primary tasks in designing assembly instructions are:

- **Planning:** Most objects can be assembled in a variety of ways. The challenge is to choose a sequence of assembly operations that will be easy for users to understand and follow.

- **Presentation:** There are many ways to depict assembly operations. The challenge is to convey the assembly operations clearly in a series of diagrams.

These tasks have been independently studied in the areas of robotics and visualization. Assembly planning is a classic problem in robotics [Wolter 1989; de Mello and Sanderson 1991; Wilson 1992; Romney et al. 1995]. Given the geometry of each part in the assembly, an assembly planner computes all geometrically feasible sequences of assembly operations. These plans are used by robotic machine tools for automated manufacturing and are not meant to be seen, understood, or carried out by humans. Most robotic assembly plans would seem unnatural to people assembling everyday objects.

In contrast, automated presentation design systems have been developed in the domain of visualization [Feiner 1985; Mackinlay 1986], with the goal of producing diagrams that are easy for humans to understand. These systems assume that the information to be portrayed is given as input and automatically design an effective diagram to convey that information. Although some of these automated presentation systems have been developed to illustrate 3D

objects and actions [Seligmann and Feiner 1991; Rist et al. 1994; Butz 1997; Strothotte 1998], their primary focus has been on showing the locations or physical properties of parts.

Our approach is inspired by a combination of ideas from these previous systems. However, we believe that decisions involved in planning and presentation are strongly intertwined. Therefore both issues must be considered simultaneously.

The contributions of our work include:

Cognitive design principles for effective assembly instructions: We performed cognitive psychology experiments to identify how people conceive of the assembly process and to characterize the properties of well-designed instructions. Based on the results of these experiments and prior cognitive psychology research, we identify design principles for effective assembly instructions. These principles connect people's conceptual model of the assembly task to the visual representation of that task.

A system instantiating these design principles: Our assembly instruction design system consists of two parts: a planner and a presenter. The planner searches the space of feasible assembly sequences to find one that best matches the cognitive design principles. To do this the planner must also consider many aspects of presentation. The presenter then renders a diagram for each step of the assembly sequence generated by the planner. The presenter also uses the design principles to determine where to place parts, guidelines and arrows. In particular, the presenter can generate action diagrams which use the conventions of exploded views to clearly depict the parts and operation required in each assembly step.

2 Design Principles for Assembly Instructions

Before we can develop automated tools for designing assembly instructions, we must understand how people think about and communicate the process of assembling an object. Cognitive psychologists have developed a variety of techniques to investigate how people mentally represent ideas and concepts. We recently performed human subject experiments based on these techniques to determine the mental representations underlying assembly [Heiser and Tversky 2002]. We briefly describe our experimental setup.

In the first experiment, we asked participants to assemble a TV stand, given only a photograph of the completed stand as a guide. After they assembled the TV stand, we asked them to create a set of instructions that would show another person how to assemble it. Examples of the diagrams they drew are shown in Figure 2. In the second experiment, we asked a new group of participants to rank the effectiveness of a subset of the instructions produced in the first experiment. Finally, the third experiment tested whether the highly ranked instructions were more effective. Yet another group of participants used instructions ranked in the second experiment to assemble the TV stand, while experimenters recorded task completion time and error rates. We found that in general the highly rated instructions were easier to understand and follow. Participants spent less time assembling the TV stand and made fewer errors.

Based on these experiments, as well as earlier cognitive research, we identify a set of design principles for creating assembly instructions that are easy to understand and follow.

Hierarchy and grouping of parts: People think of assemblies as a hierarchy of parts. At the base level, parts are segmented by perceptual salience indexed by contour discontinuity; that is, parts that are disjoint are more likely to be segmented. Typically, the disjoint parts are also grouped by different functions (e.g. the legs of a chair or the drawers of a desk) [Tversky and Hemenway 1984]. When possible, people prefer that parts within a group are added to the assembly at the same time, or in sequence one after another. The part groups are usually considered as hierarchical structures, which parallel the subassembly structure of the object.

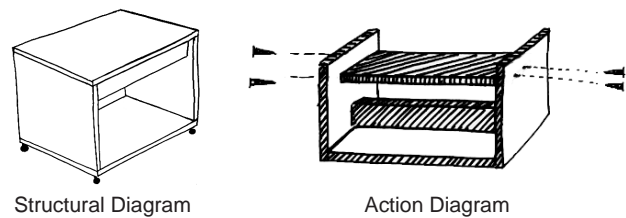Structural Diagram   Action Diagram

Figure 2: Hand-drawn assembly diagrams for the TV stand. The action diagram is preferable to the structural diagram because it depicts the operations required to attach each part. In this case the action diagram shows how the shelf is fastened by the screws.

Hierarchy of operations: People think of the attachment operations required to build an assembly as a hierarchy of actions on the parts [Zacks et al. 2001]. At the higher levels, people consider the operations required to combine separate subassemblies. Our experiments showed that as people work down the subassembly hierarchy, they eventually consider the operations required to join significant individual parts. At the lowest level of the hierarchy, people consider attaching smaller parts and fasteners to the more significant parts. The significance of a part depends on a number of factors including function, size, and symmetry.

While the hierarchy of operations may contain many levels for complicated objects with numerous subassemblies (e.g. a car engine), we have found that a two-level hierarchy (significant parts and less important parts + fasteners) is common for many build-at-home objects, including most furniture. In this paper we focus on design tools for these two levels.

Step-by-step instructions: Our experiments confirmed the results of Novick et al. [2000] showing that people prefer instructions that present the assembly operations across a sequence of diagrams rather than a single diagram showing all the operations. Moreover, if the assembly contains significant parts as well as less important parts, people generally prefer that each diagram show how to attach only one significant part at a time. However, each diagram will usually show multiple non-significant part attachments. In Figure 1, the non-significant parts include the fasteners and the wheels.

While it is essential that the assembly diagrams are clear and easy to read, each diagram should also present as much information as possible. If instructions are split across too many diagrams, they become tedious to use. Similarly, some assemblies require the same sequence of operations to be repeated many times. For example, when assembling a bookcase, each shelf is attached in exactly the same way. Depicting such repetitive operations in detail can make the instructions unnecessarily long and tiresome. A better approach is to skip repetitive operations after they have been presented in detail a few times.

Structural diagrams and action diagrams: Based on analysis of the hand-drawn instructions we collected in the first experiment, we define two types of assembly diagrams: structural diagrams and action diagrams (see Figure 2). Structural diagrams present all the parts of the assembly in their final assembled positions; users must compare two consecutive diagrams to infer which parts are to be attached. Action diagrams spatially separate the parts to be attached from the parts that are already attached and use guidelines to indicate where the new parts attach to the earlier parts.

We found that action diagrams are superior to structural diagrams for the TV stand assembly task. We believe that this is because action diagrams contain all the information in the structural diagrams and also explicitly depict the attachment operations required in each step. However, toys such as LEGO often use structural diagrams rather than action diagrams. Showing the attachment operations may be less important because most LEGO parts fasten in the same way.

Orientation: Most objects have a set of natural orientations or preferred views [Palmer et al. 1981; Blanz et al. 1999]. These orien-

tations maximize the number of important features that are visible. They minimize accidental alignments and facilitate object recognition. In assembly instructions, these natural orientations may conform to other objectives such as gravitational stability. Large bulky parts, like the frame of a bookcase, may be oriented to lie horizontally on the ground plane in the initial frame-building stages and later be reoriented to stand upright when the shelves are attached.

Visibility: Perhaps the strongest design principle is that all the new parts added in each step of the assembly must be visible. Users must be able to see the parts in order to attach them. However, one exception to this rule is that maintaining visibility for all parts in a symmetric group is less important. If the user is aware of the symmetry, it is usually enough that at least one part in the group is visible, since the others will attach in a similar way.

While the new parts have to be visible, the parts attached in earlier steps should also be visible to provide context for the new attachments. If only the new parts are visible, it may be unclear how the new parts are supposed to attach to the earlier parts. Note that it is not necessary for each of the earlier parts to be visible. Rather, a portion of the entire set of earlier parts should be visible.

3 System Overview

Our assembly instruction design system is divided into two components; a planner and a presenter. The input consists of:

- **Geometry:** The geometry of each part of the object in its final assembled position.

- **Orientations:** Default camera viewpoint and assembly orientation. A preferred orientation for each *significant part* may also be specified.

- **Groupings:** Labeling of parts that should be grouped together based on functional, semantic, or geometric properties. The grouping labels used by the system include *fasteners*, *significant parts*, *symmetry*, and *similar-action*.

- **Ordering Constraints:** Constraints on the order of assembly operations in the form **attach** part p_i **before** part p_j.

The minimum input information consists of the part geometry, a default camera viewpoint, and the object's orientation. All other information is optional, and the system can produce useful instructions without it. The additional grouping information and ordering constraints help the system to make even better design choices. The input may be generated automatically by other software or provided by the users. Regardless of how the input is first generated, users can work with the system, adding information such as ordering constraints or part groupings to produce the final set of instructions.

From this input the planner computes the set of operations to show in each assembly diagram. The presenter renders each assembly step as a diagram, and outputs the sequence of diagrams as the final set of assembly instructions. The planner and presenter are described in Sections 4 and 5. Before we present these components in detail, we summarize several of the basic, low-level computations used throughout our system.

3.1 Low-level Computations

Our system frequently needs to know if parts are in contact, whether parts block each other, and the visibility relationship between parts. We briefly describe how to compute this information.

Contacts: Given any set of parts P, we provide a function $Contacts(P)$ that returns the set of parts in contact with at least one part in P. To determine which parts are in contact we compute the shortest distance between each pair of parts [Lin and Canny 1991; Quinlan 1994]. If this shortest distance is 0.0, we mark the pair as being in contact. We also store the faces of each part that are within the contact distance of one another.
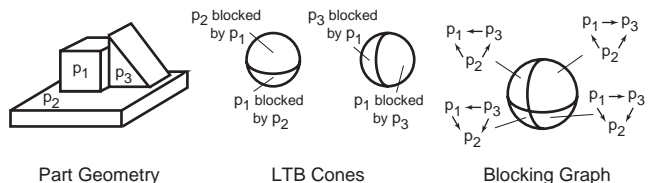

Figure 3: (left) A three part assembly. (middle) The LTB cone for p_1 and p_2 encodes that p_2 is blocked from moving in the upper hemisphere by p_1 and vice versa. Similarly, the LTB cone for p_1 and p_3 encodes the p_3 is blocked from moving in the left hemisphere by p_1. (right) The blocking graph is a partition of the sphere of translational motion directions, with a directed graph storing the blocking relationships for each partitioned region. A graph edge of the form $p_i \rightarrow p_j$ means that part p_i is blocked by p_j for all directions in the corresponding region of the sphere. The blocking graph combines all the information from the LTB cones into one structure. Figure adapted from Romney et al. [1995]

Blocking: For any two parts p_i and p_j, we provide a function $RemoveDirs(p_i, p_j)$ that returns the set of directions in which p_i can be removed from p_j without interference via a single translational motion. In all other directions p_i is blocked by p_j.

To compute these blocking relationships between all the parts in the assembly we follow the approach of Romney et al. [1995]. We first compute a local translational blocking (LTB) cone for each pair of parts p_i and p_j that are in contact. The LTB cone is the set of directions in which p_i blocks p_j from translational motion and can be computed directly from the geometry of the parts. The cone encodes local blocking relationships because it is only computed for parts in contact. As shown in Figure 3, we combine all the LTB cones into a blocking graph structure that encodes all the blocking relationships in the assembly. Although we consider only single step local translational motions in our blocking analysis, extension to global translations [Wilson 1992], multi-step translations and rotational motions [Guibas et al. 1995] is possible.

While Romney et al. store the blocking graph as an analytical partitioning of the translational motion sphere, for simplicity we discretize the sphere and store blocking relationships for each direction independently. For many assemblies only the 6 principal motion directions are of interest and we only need to compute and store blocking relationships for these directions. Following Wilson's [1992] terminology, we refer to the directed graph associated with each discrete direction as a *directional blocking graph*.

Using the blocking graph structure we can look up $RemoveDirs(p_i, p_j)$ for any pair of parts in the assembly. Similarly, we can compute the set of removable directions between two sets of parts P and Q as:

$$RemoveDirs(P,Q) = \bigcap_{\forall p_i \in P, q_j \in Q} RemoveDirs(p_i, q_j) \qquad (1)$$

Visibility: Given two sets of parts P and Q our visibility test $Vis(P,Q)$ computes the percentage of P that is visible with respect to Q. We use a two-pass rendering approach in which we first render the parts in P using a single color (green in our case). We count the green pixels to determine $Area(P)$, the projected area of P. Next, without clearing the framebuffer we render the parts in Q using a different color (red) and again count the number of green pixels that remain visible to determine $Area(P,Q)$, the visible projected area of P with respect to Q. Then the visibility percentage of P with respect to Q is given by:

$$Vis(P,Q) = \frac{Area(P,Q)}{Area(P)} \qquad (2)$$

4 Planner

As shown in Figure 4, our assembly planner computes a set of assembly operations to show in each diagram. The planner can

Planner

All parts

U_i = Unattached parts

Search R = subset of U_i

Evaluate

- **Interference**
- **Attachment**
- **Ordering**
- **Grouping**
- **Visibility**

Score for R

$U_{i+1} = U_i - Q_i$ Q_i = best R

Sequence Determination

$P_j, P_{j+1}, ..$

Reinsert Fasteners

Reorientation

$(P_j, \mathbf{V}_j, \mathbf{M}_j), (P_{j+1}, \mathbf{V}_{j+1}, \mathbf{M}_{j+1}) ...$
sequence of assembly steps for Q_i

Sequence of all assembly steps

top screws
pegs
a: top shelf
e: left side   b: support board   f: right side
c: bottom shelf
d: wheels
bottom screws

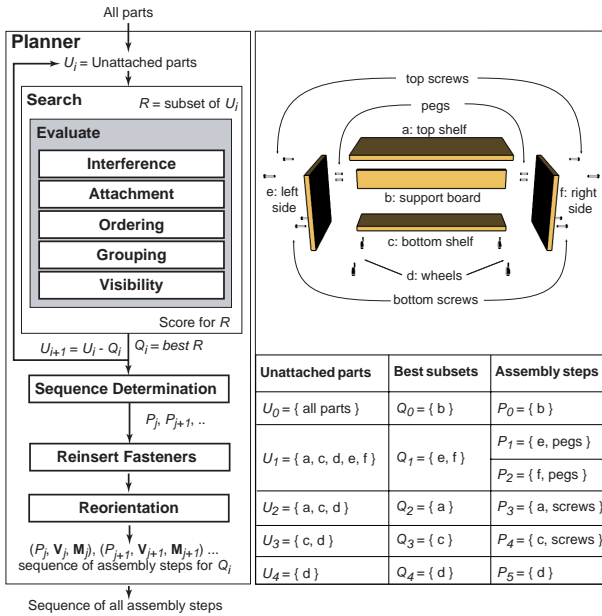| Unattached parts	Best subsets	Assembly steps
U_0 = { all parts }	Q_0 = { b }	P_0 = { b }
U_1 = { a, c, d, e, f }	Q_1 = { e, f }	P_1 = { e, pegs }
		P_2 = { f, pegs }
U_2 = { a, c, d }	Q_2 = { a }	P_3 = { a, screws }
U_3 = { c, d }	Q_3 = { c }	P_4 = { c, screws }
U_4 = { d }	Q_4 = { d }	P_5 = { d }

Figure 4: Block diagram of the planner and the plan it generates for the TV stand. The input required to generate this plan consists of: 1) geometry of each part in assembled configuration, 2) default camera and object orientations, 3) parts grouped by *symmetry* {e, f}, {4 *wheels*}, {4 *pegs*}, {4 *top screws*}, {4 *bottom screws*}, 4) *significant parts* {a, b, c, e, f}, and 5) *fasteners* {4 *pegs*, 8 *screws*}.

choose to attach a part to the assembly or to reorient the assembly. Attachment is the primary operation, and the planner searches over the space of all parts to find the best set of part attachments to depict in each diagram. In some situations the planner may also reorient the assembly. Such reorientations are either performed to improve the visibility of subsequent parts or based on orientation constraints that specify a natural orientation for the object.

Our notation is as follows. When designing assembly step i, the input to the search is U_i, the set of unattached parts. These are the parts that have not yet been added to the assembly. The search then considers each subset $R \subseteq U_i$ to find the best subset $Q_i = R_{best}$ to add to the assembly. Once a part is added to the assembly it is placed in the set of attached parts A_i. After choosing the best subset Q_i, we set $U_{i+1} = U_i - Q_i$ and $A_{i+1} = A_i + Q_i$. We iterate the search until U is empty and A contains all the parts.

As we saw in Section 2, people usually consider the process of assembly as a hierarchy of attachment operations (attaching subassemblies, attaching significant parts and attaching fasteners to the significant parts). Our planning search acts at the significant parts level and therefore only considers non-fastener parts as it is choosing which set of parts to attach next. Initially, U_0 is the set of all non-fasteners in the assembly.

After the search, if Q_i contains multiple significant parts, the sequence determination stage subdivides Q_i into subsets $P_j, P_{j+1}, ... \subseteq Q_i$, such that each one contains at most one significant part. This stage also decides whether to omit repetitive assembly operations after they have been presented in detail a few times. After the sequence determination stage we reinsert the fasteners connecting each P_j to the earlier parts back into P_j.

The planner outputs a sequence of assembly steps, one step per diagram. Each assembly step specifies a set of parts P to attach, a camera viewpoint $\mathbf{V}$, and an object orientation $\mathbf{M}$. Note that since the part sets can contain multiple parts each diagram may show multiple attachment operations. Part sets may also be empty. In such cases the step will contain a camera reorientation, an object reorientation, or both.

4.1 Search

As shown in Figure 4, we evaluate several constraints for each subset $R \subseteq U_i$. Most of these constraints are hard constraints that check the feasibility of attaching the parts in R to the earlier parts A_i. If the current subset R violates any of the hard constraints, it is immediately rejected by the system. The visibility constraint is the only soft constraint, or objective function, in the system. The search therefore looks for a subset of U_i that passes the feasibility constraints and maximizes the visibility score.

Interference: We can add the parts in R to the assembly only if all of them are removable from the assembly and do not block any other unattached part. We can determine if the parts in R interfere with one another, or the other unattached parts U_i, by checking that each unattached part $u_j \in U_i$ and its associated fasteners are removable from $A_i \bigcup R$, the union of the previously attached parts and the current subset. Note that U_i includes R when this test is performed. To test if u_j is removable from $A_i \bigcup R$, we simply check that $RemoveDirs(u_j, A_i \bigcup R)$ is not empty. We perform that same removability test on each part in $Fasteners(u_j, A_i \bigcup R)$, the set of fasteners connecting u_j to the parts in $A_i \bigcup R$.

Attachment: Parts should only be added to the assembly when they can be fastened to it. For example, suppose that in Figure 1, the support board was added to the assembly in the first step. The next step should not attach the top shelf. Even though the top shelf is supported by the support board, the top shelf does not directly fasten onto the support board. In general, if a part attaches onto the assembly by fasteners, it should be added to the assembly only after one or more of the parts it fastens onto is already attached. More precisely, for each $r_j \in R$ we check that $Fasteners(r_j, A_i)$, the set of fasteners connecting r_j to any other non-fastener part already attached in the assembly, is not empty. This constraint is not applied to the first diagram since A is empty at that point.

If the fasteners have not been labeled, or if a part r_j is not attached to the assembly by separate fasteners (for example the wheels of the TV stand), we simply check that the part is in contact with some part already in the assembly. This is a weaker form of the main attachment constraint. In our previous example, the top shelf of the TV stand would pass this weaker form of the constraint.

Ordering: To check that ordering constraints of the form **attach** p_k **before** p_l are satisfied we make sure that if p_l is in the current subset R then p_k is in the set of previously attached parts, A_i.

Grouping: Parts that are labeled as belonging to the same group should be added to the assembly at the same time. Our system checks this constraint for parts labeled as symmetric to one another. For each part $r_j \in R$ we first look up $Symm(r_j)$, the set of parts labeled by the user as being *symmetric* to r_j. We then check that R includes all parts in $Symm(r_j)$.

Visibility: For people to easily understand and follow an assembly diagram, it is important for all the parts being attached, R, to be visible. In addition, some portion of the parts attached previously, A_i, should be visible to provide context for the new attachments. Finally, the parts being attached in the current step should not significantly occlude parts that will be added in future steps, U_i. Therefore our visibility constraint evaluates three kinds of visibility: current parts visibility, previous parts visibility and future parts visibility.

The current parts visibility score computes a lower bound on the visibility of the parts in the current subset R with respect to the other parts in R as:

$$Score(R) = \min_{r_j \in R}(Vis(r_j, R - r_j)) \tag{3}$$

The previous parts visibility test computes the visibility of the previously attached parts A_i with respect to the current subset R:
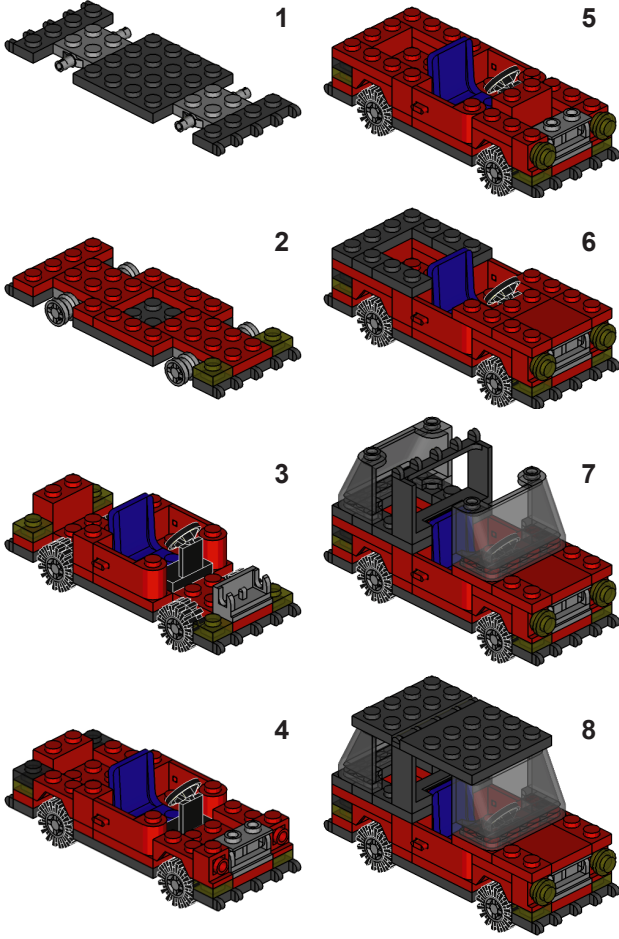
$$Score(A_i) = Vis(A_i, R) \tag{4}$$

Figure 5: Assembly instructions for LEGO car. The planner finds an assembly sequence that maintains good visibility for all the parts added in each step. Each step is presented as a structural diagram. Notice that two consecutive steps must be compared to discover which parts were added in each step.

Score(A_i)	= 0.37
Score(R)	= 0.72
Score(U_i-R)	= 0.82

Score(A_i)	= 0.21
Score(R)	= 0.42
Score(U_i-R)	= 0.83

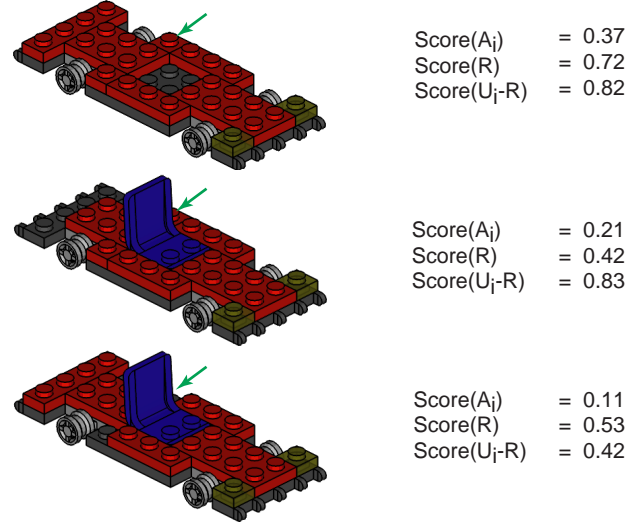| Score(A_i) | = 0.11 |
| Score(R) | = 0.53 |
| Score(U_i-R) | = 0.42 |

Figure 6: Visibility scores for three subsets R while generating step 2 of the LEGO car instructions. The first subset has relatively good visibility for the previous parts A_i, the current parts R and the future parts $U_i - R$. In the second subset, the seat replaces the red part at the very back of the car. The current parts score, $Score(R)$, is substantially lower than for the first subset. The part indicated by the green arrow is occluded by the seat and it sets the minimum visibility for the current parts to 0.42. In the third subset, the seat replaces the part indicated by the green arrow and the part symmetric to it, on the left side of the car. In this case the part indicated by the green arrow will be added in a future step and therefore it sets the future parts visibility score $Score(U_i - R)$ to 0.42. Eventually the search finds that the first subset of these three maximizes the visibility score and chooses it as the second step in the assembly instructions.

The future parts visibility test checks that current parts will not completely occlude any of the future parts $U_i - R$. This test is similar to the current parts test:

$$Score(U_i - R) = \min_{u_j \in U_i - R}(Vis(u_j, R)) \quad (5)$$

The total visibility score for R is the sum of these three scores. Examples of these scores are shown in Figure 6. Empirically we have found that visibility scores at or above 50% for minimum current part visibility, 10% for previous part visibility, and 25% for minimum future part visibility produce good results.

Maintaining the visibility of every part in a symmetric group is less important than ensuring that some portion of all the symmetric parts are visible. In step 3 of Figure 5, for example, it is less important that the wheels on the left side of the car are visible because the two symmetric wheels on the right side are visible. Therefore we modify the current and future parts visibility scores to group all symmetric parts together and treat them as one large part, rather than checking visibility for each symmetric part individually.

Optimizing the search: The search strategy outlined above evaluates all subsets of U_i. Therefore the size of the search space is 2^N where N is the number of parts in U_i. For large assemblies, the size of the search space can become prohibitive.

Yet most subsets of $R \subseteq U_i$ fail one or more of the hard constraints. For example, if a part $u_j \in U_i$ is not in contact with any part in A_i then any subset P that contains u_j will fail the attachment constraint. The same holds true if u_j fails the interference constraint or the ordering constraint. Therefore, we can conservatively cull parts from U_i that cannot possibly be added to the assembly. Starting with U_i we form a new set U_i' containing only the parts $u_j \in U_i$ that individually pass the interference, attachment and ordering constraints. Testing these constraints for single parts is fast because each one simply requires a few look-ups. We then search for the best subset of U_i' rather than U_i.

Since we are interested in the largest subsets of U_i' that pass all the constraints, we can further accelerate the search by processing the subsets in breadth-first order and using a heuristic to exit the search early. We first consider U_i' itself, then each subset of U_i' with one part removed, each subset with two parts removed and so on. We exit the search as soon as we examine an entire level of this subset tree and find at least one subset with visibility score greater than an empirically determined threshold. In practice we have found that this early exit significantly reduces the search time, and generates the same results as performing the full search.

As we form subsets of U_i' in breadth-first order we enforce the grouping constraint by treating all parts in a symmetric group as a single part. Suppose U_i' contains a symmetric group of parts $S = \{s_1, s_2, ...\}$. Instead of removing each s_j individually from U_i' to form the subsets, we remove the entire set S at once. Note that for the early exit test this subset $U_i' - S$, is treated as being one element smaller in size than U_i'. That is, $U_i' - S$ is at the same level of the subset tree as $U_i' - u_j$ where u_j is not part of a symmetric group.

4.2 Sequence Determination

While the search encodes many of the cognitive design principles for producing effective assembly instructions, it does not encode them all. We check two of the principles outside the main search to improve the running time of the algorithm. In particular, the search does not check that subsets $R \subseteq U_i$ contain at most one significant part and it does not check for repetitive steps. Instead, we allow the
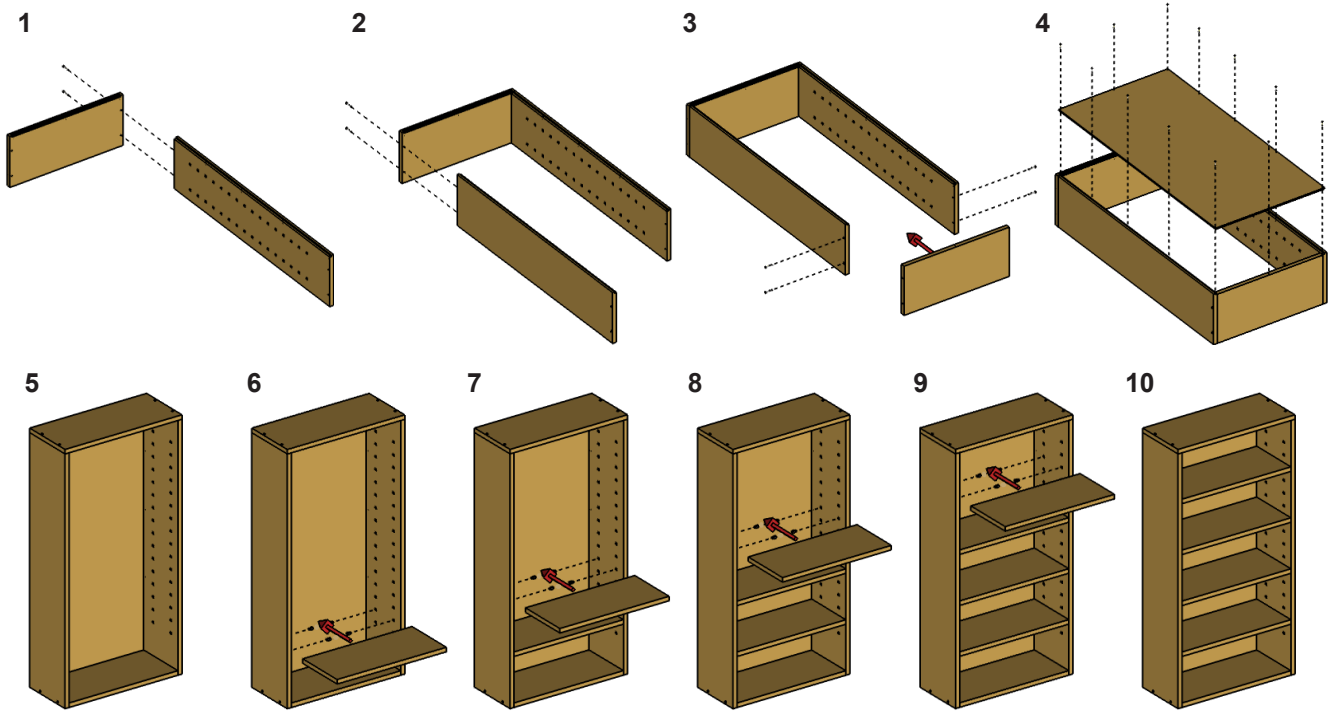
Figure 7: Instructions for building a bookcase. Because the back of the bookcase is functionally part of the frame, we specified an ordering constraint forcing the back to be attached before the shelves. **Sequencing significant parts:** After the back of the bookcase is attached in step 4, the planner search selects the set of four shelves as the best subset of unattached parts Q_i to show in the next diagram. However, the shelves are labeled as significant parts, so the sequencing stage splits Q_i into four separate steps. Each shelf yields the same visibility score, so they are sequenced in back-to-front order based on distance from the camera. **Omitting repetitive steps:** Each shelf is attached in exactly the same way. While we show all four shelf attachments here, if the shelf attachment operations are marked as *similar-actions* the system will omit the last two shelf attachments, steps 8 and 9, and jump directly to step 10 after step 7. **Reorientation:** Although the search decides that the shelves should be attached after step 4, the shelves would not be visible in the horizontal orientation. Therefore the reorientation stage finds a new orientation for the bookshelf that ensures the shelves are visible.

search to find a subset Q_i containing more than one significant part or repetitive steps and deal with them in the sequence determination stage, by further subdividing Q_i into smaller subsets as necessary.

4.2.1 Handling Significant Parts

As we saw in Section 2, people prefer that the significant parts in the assembly are added one by one in a sequence of separate diagrams. We maintain this design principle by splitting Q_i if it contains more than one significant part. To split Q_i we first compute the visibility $Vis(q_j, A_i \bigcup Q_i - q_j)$, for each part $q_j \in Q_i$, with respect to all the other parts in the assembly. The significant parts are added in order from least visible to most visible. Ties in this visibility score are resolved by adding the part furthest away from the viewer first. This distance is computed from the center of q_j to the current camera viewpoint. The camera viewpoint is usually the default, but may have been changed by the reorientation stage of the planner. An example of resolving such a tie is presented in Figure 7. If q_j is symmetric to other parts in Q_i, all of these symmetric parts are sequenced using the visibility/distance ordering, before any other parts in Q_i. This ensures that symmetric groups of parts are added in sequence one after another. Finally, all of the non-significant parts in Q_i are added to the assembly. This sequencing stage produces a sequence $P_j, P_{j+1}, ...$ of subsets of Q_i.

4.2.2 Omitting Repetitive Operations

Since repetitive operations can make instructions unnecessarily long and tedious, we also omit repetitive part attachment operations in the sequence determination stage. Omitting repetitive steps requires the user to label groups of parts that require similar attachment operations as *similar-action* groups. Given a set of parts P_j

we wish to attach, we consider each $p_k \in P_j$ and check how many parts in the similar-action group for p_k have already been attached. If some similar-action parts have already been depicted in full detail, we skip depicting the attachment of p_k by removing it from P_j. Users can decide how many similar-action attachments to show in full detail before they are skipped. By default only the first two such attachments are shown in detail.

4.3 Reinserting Fasteners

Once we have chosen the non-fastener parts P_j for each assembly step we reinsert the fasteners connecting P_j to the earlier parts in the assembly A_i. If F is the set of all *fasteners*, then we can find the set of fasteners connecting any set of parts P to any other part in the assembly as:

$$Fasteners(P) = Contacts(P) \bigcap F \qquad (6)$$

Similarly we can determine the fasteners connecting P_j to earlier parts A_i as:

$$Fasteners(P_j, A_i) = Fasteners(P_j) \bigcap Fasteners(A_i) \qquad (7)$$

We add these fasteners to P_j before passing it to the next stage.

4.4 Reorientation

For some objects the default camera and object orientations can be used for the entire set of assembly diagrams. In many cases however, the default orientations may not be appropriate for every assembly diagram. There are two primary reasons to reorient the

object or the camera: to put the object in a more natural orientation for the current assembly operation and to improve the visibility of parts. The reorientation stage handles both cases. If the object is reoriented, we maintain the reorientation for all subsequent diagrams in the sequence.

If the current assembly step contains a significant part for which an orientation preference has been specified, we first apply the preference. We then consider whether reorienting the camera could improve the visibility of the parts in the current step. To determine if camera reorientation is necessary, we compute $max_{p_k \in P_j}(Vis(p_k, A_i \bigcup Q_i - p_k))$, the maximum visibility percentage for each part in the current subset P_j with respect to all the other parts in the assembly. If this upper bound on part visibility is low (we have found that a threshold of 35% works well), we search for a new camera orientation that will increase visibility.

The camera reorientation search computes the minimum visibility of all parts in P_j from a small set of alternative viewpoints and chooses the viewpoint producing the largest minimum visibility. As Blanz et al. [1999] have empirically shown, people have a strong preference for viewing most objects from above and at oblique angles, rather than front or side views. We select the alternative viewpoints for our camera reorientation search based on these findings. Since the object orientation is known, we have a frame of reference and we can set the camera to look down at the object from either the left or right side.

5 Presenter

The presenter renders the sequence of assembly steps output by the planner as a series of either structural diagrams or action diagrams. We describe techniques for rendering both types of diagrams.

5.1 Structural Diagrams

Each structural diagram presents all parts attached in the current step j along with all the parts seen in earlier steps in their final assembled positions. The parts are rendered using the current camera viewpoint $\mathbf{V}_j$ and object orientation $\mathbf{M}_j$. Figure 5 shows an example of such structural diagrams as generated by our system. While such structural diagrams are typically provided with toys like LEGO, for many assemblies they can be difficult to follow because the user must compare before and after diagrams to figure out which parts were added in each step.

5.2 Action Diagrams

Action diagrams are generally easier to follow than structural diagrams because they spatially separate the parts being added in each step from the earlier parts. They also use diagrammatic elements to show how and where the new parts attach to the earlier parts.

To generate an action diagram for assembly step j, we first choose the direction in which we want to separate each of the parts in P_j from the earlier parts. We then set a separation distance for each part and finally place the guidelines between the parts. We consider each of these steps in detail.

5.2.1 Choosing the Separation Direction

Since the goal of an action diagram is to spatially separate the parts being attached in the current step from the earlier parts, the separation direction for each new part should be chosen to maximally separate the new part from the earlier parts. Moreover, the direction chosen for part $p_i \in P_j$ must be interference-free with respect to the parts attached earlier, A_j. That is, the separation direction must be one of the directions in $RemoveDirs(p_i, A_j)$. We choose the separation direction for p_i as the interference-free direction that allows p_i to escape the bounding box of earlier parts A_j as fast as possible.
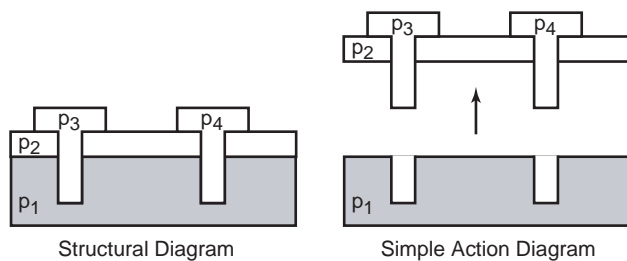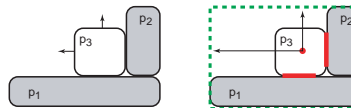

Figure 8: One approach to generating action diagrams is to translate the parts being attached in the current step p_2, p_3, p_4 away from the earlier parts p_1 by a fixed distance along the separation direction (*up* in this case). But because all the new parts are moved a fixed distance, this approach may not separate the new parts from one another.

In this example, suppose we are adding part p_3 to the assembly consisting of $A_j = \{p_1, p_2\}$. Of the four principal directions in 2D, the only free directions for p_3 are *up* and to the *left*. To choose between them we first look up $Contacts(p_3, A_j)$ and compute a bounding box for these contact parts, as shown in green. We also look up the faces of p_3 that are in contact with parts in A_j and compute the center of the bounding box for these contact faces, as shown in red. We then compute the distance from the contact faces' center to the contact parts' bounding box in each of the feasible removal directions. The direction yielding the shortest distance is chosen as the separation direction. Here the separation direction for p_3 is *up*, since this direction allows p_3 to exit the bounding box of the earlier parts p_1 and p_2 fastest. If there is a tie in the shortest distance, we pick the direction that is pointing towards the camera viewpoint.

5.2.2 Setting the Separation Distance

A simple technique for setting the separation distance for the action diagram is to translate each part in the current step $p_i \in P_j$ by a fixed distance in the separation direction. However, as shown in Figure 8, this simple approach does not always produce the desired results. While the new parts are separated from the earlier parts, they may not be separated from another. Notice that the parts sit one atop another in the separation direction. To properly set the separation distance for these parts we must first determine this stacking relationship between them.

Building Stacks We define a *stack* as a sequence of parts with three properties: 1) all the parts (except the first) share the same separation direction, 2) the parts share a stabbing line in the separation direction, and 3) each part is in contact with the next part in the sequence. The first part in the sequence is called the *base part*. It provides an anchor for the stack and may have a different separation direction than the other parts in the stack. Creating an action diagram requires building a tree of such stacks.

To build a tree of stacks from any set of parts P, we begin by grouping the parts in P by separation direction. Let G be one such group. We look up the directional blocking graph for the parts in G along the separation direction. Since the separation direction is interference-free for all parts in G, this directional blocking graph cannot contain cycles. However, the graph may be disjoint.

We iteratively build a tree of stacks from this blocking graph in a greedy manner. Starting from a part g_i at the root of the blocking graph, we form a stabbing line through the center of g_i and find the maximal sequence of parts in P that include g_i and meet the other stack conditions (same separation direction, sequential contact). We consider both directions of the stabbing line as we are building these stack sequences. The furthest part on the stabbing line in the negative separation direction is chosen as the base part for the stack and may have a different separation direction than the
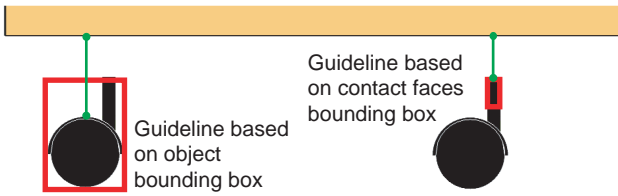
Figure 9: Side view of two wheels of the TV stand. (left) Guidelines (in green) generated from the center of the wheel bounding box (in red) do not pass through the stem of the wheel and look awkward. (right) Guidelines generated from the center of the contact faces' bounding box correctly pass through the stem of the wheel.

other parts of the sequence. Each part added to this stack sequence is removed from G. We then update the blocking graph for G and iterate the stack-building procedure until G is empty.

Expanding Stacks   Using these stack structures, we can properly separate all the parts in each stack. Given a stack expansion distance d, we start at the base of each stack and translate each subsequent part in the stack sequence by distance d from the previous part along the separation direction. In this way the total translational distance accumulates as we move down the stack sequence. If a non-base part p_i in the current stack is also a base part for another stack, we translate all parts the other stack by the same distance we translate p_i. Thus, translations propagate recursively through the stack tree.

To create action diagrams, we build stacks for the current set of parts P_j. We also combine the set of previously attached parts A_j and include the group as a single part with no separation directions in the stack building process. This grouped set of previous parts provides a good base part for all of the stacks built in this stage. Note that this stack building process can also be used to generate exploded views of the entire object. In this case, we build stacks from all of the parts in the assembly rather than just the parts in P_j.

Most previous systems for generating exploded views have required users to manually specify the stacking relationships and explosion directions for each pair of parts [Rist et al. 1994; Driskill and Cohen 1995]. A notable exception is Raab and Rüger's [1996] technique which uses non-linear 3D zooming to produce exploded views. However, their technique relies on the ability to non-linearly distort the space around each part.

5.2.3   Placing Guidelines

The stack structures allow the presenter to separate the new parts in each assembly step from the earlier parts. However, one drawback to separating these parts is that it can be difficult to figure out where the new parts are supposed to attach to the earlier parts. The alignment between the new parts and previous parts may be ambiguous. Guidelines running between the parts of the stack can remove this alignment ambiguity. But, given a stack of parts, the challenge in generating guidelines is choosing where to position the guideline endpoints. As shown in Figure 9, simply connecting the center of each part may not produce good results for some asymmetric parts.

Our approach to generating a guideline emanating from part p_i is to look up the set of contact faces between p_i and the next part in the stack sequence. We compute the center of the bounding box for these contact faces and shoot a ray in the positive and negative separation directions. We then set the guideline start point (end point) to the intersection point of the negative (positive) ray with the contact faces' bounding box. This approach properly handles many asymmetric parts without forcing users to manually specify the guideline endpoints. Currently we only support guidelines between parts aligned along a common vector.

If either endpoint of the guideline contains a fastener, we draw the guideline as a dotted line. If the part at the end point of a guideline is a significant part, we draw the guideline as an arrow in the negative separation direction to indicate how the significant part should be brought in and attached to the assembly.

Model		Execution Time			Subsets
Name	# Parts	Precomp.	Planner	Vis.	Scored/Total
Bookcase	9	20m 2s	47.70s	47.04s	13/885
Case27	25	24s	52.36s	45.88s	525/*
LEGO car	61	7m 23s	545.79s	477.06s	99/*
TV stand	9	33m 14s	25.19s	24.99s	12/882
80/20 Table	13	1m 27s	32.96s	27.59s	16/6840

Table 1: Performance of the planner running on a 500 Mhz SGI 320 Visual Workstation. The second column indicates the number of non-fastener parts in each model. The subsets column reports the number of part subsets for which the planner computed visibility scores as well as the total number of subsets an exhaustive search would have considered. Since the total number of subsets is exponential in the number of parts in the model, we use *'s to indicate that the number is extremely large. The search optimizations significantly reduce the number of subsets scored.

5.3   Rendering Style

We render the final assembly diagrams using the conventions of technical illustration [Martin 1989]. We use Gouraud shading and we render the edges of the parts in contrasting colors (black lines for the lighter parts and white lines for the darker parts). Outlining the edges in this manner helps to differentiate the parts from one another, especially if they are rendered in the same color.

6   Results

Several examples of assembly instructions generated with our system have already been presented (Figures 1, 5, 7). Two more examples are shown in Figures 10 and 11.

The TV stand shown in Figure 1 is based on the TV stand we used in our human-subject experiments. The action diagrams show how each part is to be attached to the earlier parts. Dotted guidelines indicate where the fasteners connect parts and red arrows show how significant parts can be moved into position for attachment. Because the left and right sides are symmetric, the planner adds them to the assembly in consecutive steps. The input required to generate these instructions is described in Figure 4. All four wheels are added to the assembly at the same time in step 5 because they are symmetric but are not specified to be significant parts.

To generate instructions for the LEGO car shown in Figure 5, the input consisted of the part geometry, default camera and object orientations, and part symmetries. The planner is able to find a sequence of steps that builds the model in natural layers from the bottom up while maintaining good visibility for all the parts added in each step. Symmetric parts are added together in the same step.

Like the LEGO car instructions, the instructions shown in Figure 10 were generated from minimal input: part geometry, default orientations and symmetries. Note that this object, which we refer to as *case27*, was developed as a test for our system and does not represent any real-world object. In Figure 12, we show an exploded view of this object, which is generated using the same stack-building machinery we developed to produce action diagrams.

The input required to generate the bookcase instructions shown in Figure 7 includes an ordering constraint forcing the frame to be attached before the shelves. Since the back of the bookcase is part of the frame, it is attached in step 4. The system then automatically reorients the camera to ensure that the shelves will be visible. The shelves are symmetric to one another and added in sequence. With additional input specifying that the shelves are attached via similar actions, the system would automatically omit steps 8 and 9.

The table shown in Figure 11 is built using a standardized set of industrial parts developed by 80/20 Inc. [2003]. The parts are modular and can be used to design many different types of assemblies; this table is just one example. In addition to the part geometry and default orientations, we specified a preferred orientation for showing the trays resting on the table. This forces the reorientation shown in step 13 of the instructions. We also labeled all the L-shaped brackets as requiring similar attachment operations. Thus,
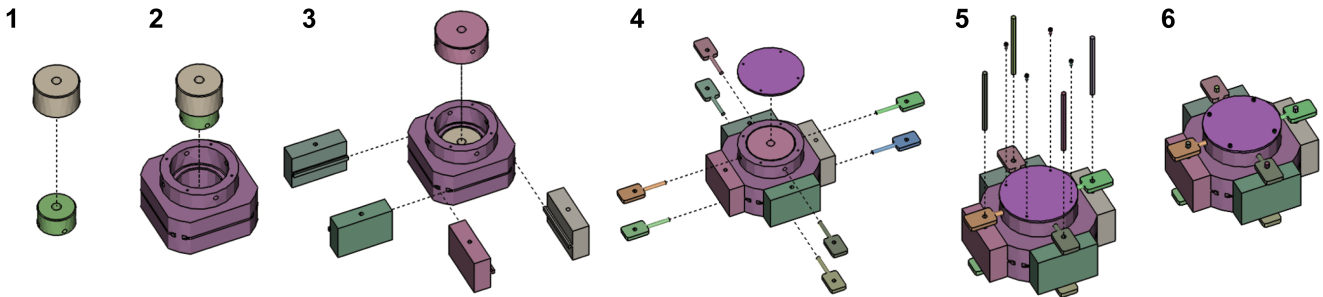
Figure 10: Assembly instructions for case27. The placement of the guidelines is automatically chosen to show how the parts attach to one another. The guidelines connect the centers of the bounding boxes of contact faces rather than connecting the centers of parts. In step 3 the guidelines properly show how the rectangular parts slide into the main hull.
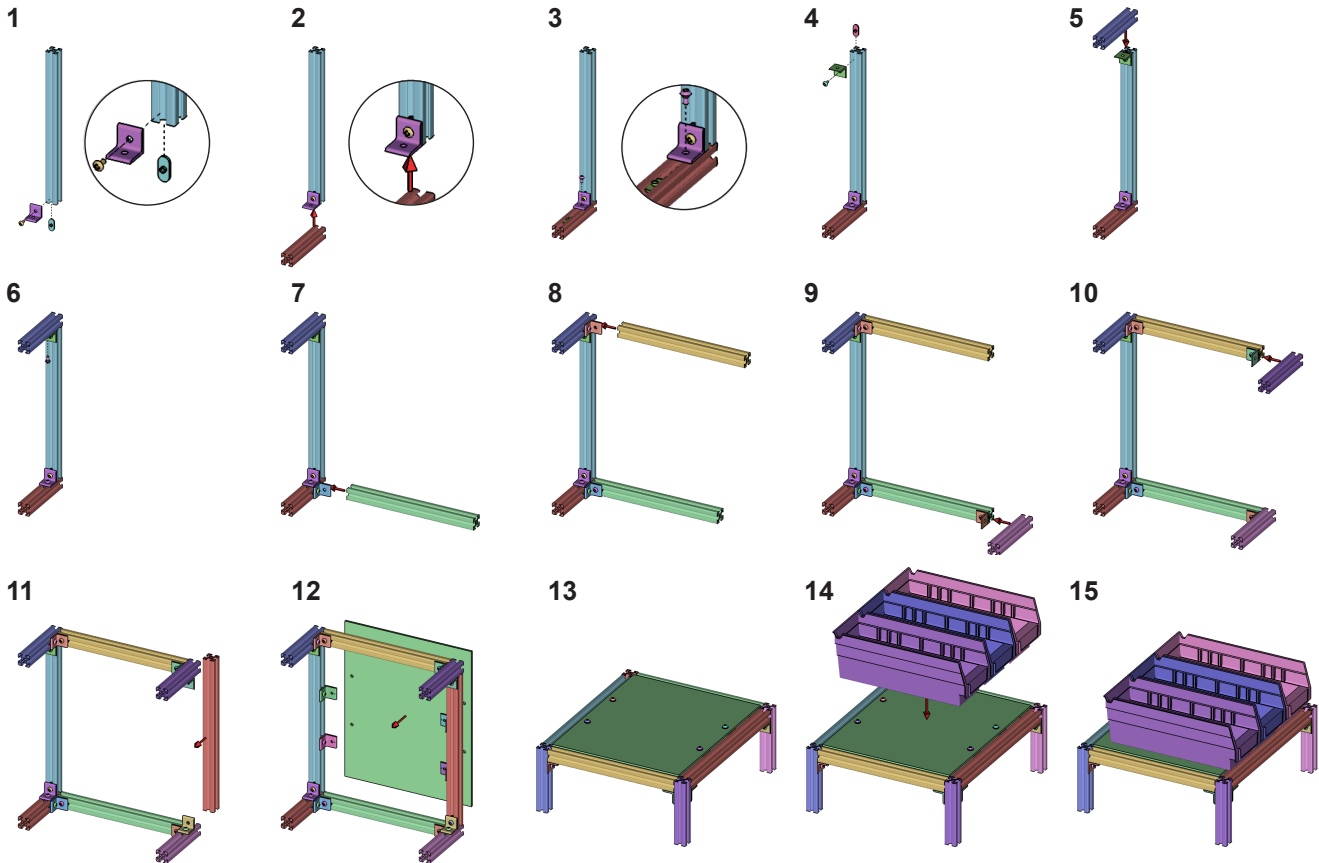

Figure 11: Assembly instructions for a table built from the 80/20 standardized parts. The first six steps detail how two of the L-shaped brackets are used to attach parts of the frame. The planner then omits these details in subsequent steps. Because the fasteners are much smaller than the other parts, they can be difficult to see in the original diagrams. We manually added the insets for the first 3 steps to make the fasteners easier to see.

only the first two bracket attachments (steps 1 through 6) are shown in full detail. In step 7 and beyond, the instructions no longer show the nuts and bolts required to secure the brackets to the frame. We manually added the insets for the first 3 steps to make the fasteners easier to see. Seligmann and Feiner [1991] have proposed an automated approach for designing such insets, and we are currently exploring the possibility of adding this technique to our system.

The performance of our system is presented in Table 1. The running time of the system is dominated by the low level geometric and visibility computations. We have not focused on optimizing these parts of the system and believe that more sophisticated low-level algorithms could increase the speed of that code. The final column of the table reports the number of subsets for which we compute the visibility score as well as the total number of subsets an exhaustive search would have to score. As Table 1 shows, our search optimizations significantly reduce the number of subsets scored.

7 Discussion

While our system can generate assembly instructions for a variety of objects, it also makes several basic assumptions that we hope to relax in future work.

Two-level hierarchy: The system operates on the two bottom levels of the hierarchy of operations (joining significant parts and attaching fasteners). Extending the system to handle subassemblies would allow for larger, more complicated assemblies.

Single-step translations along principal axes: We only consider single-step translational motions along the principal axes when computing the blocking relationships between parts. Using Guibas et al.'s [1995] approach to handle multi-step translations and rotations in all motion directions would increase the types of assemblies our system could handle.
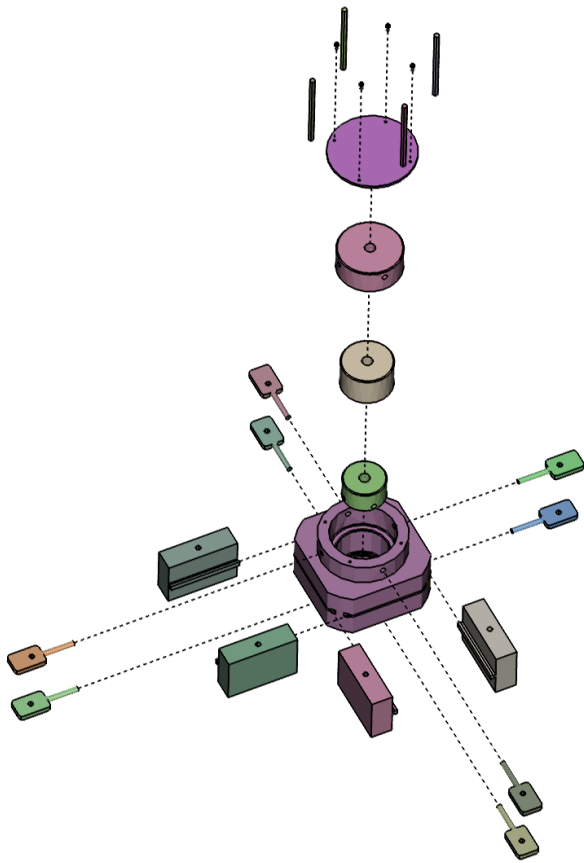
Figure 12: Exploded view of case27. Our stack-building procedure can be used on the entire assembly rather than individual steps to produce such exploded views. The algorithm properly handles building stacks for different separation directions. As shown in Figure 10, the four rods at the top of the assembly slide into the paddles and rectangular parts below. After expanding the stacks, the rods no longer align with the parts they slide into. Therefore, the system does not generate guidelines showing how the rods attach to the assembly.

Local interference: Blocking relationships are computed for local pairs of parts that are in contact with one another. However, it is possible that parts which are not in contact block one another. Therefore, global interference detection would impose stronger, more robust feasibility constraints on the assembly sequence. Wilson [1992] has proposed techniques for computing this type of global interference.

Input of semantic/functional knowledge: Our system is designed to use semantic and functional knowledge about the parts when it is provided. In practice we have supplied this information manually. However, it may be possible to infer some of these properties from the part geometry based on models of perception. For example, it may be possible to automatically group parts that are perceived as roughly symmetric.

Although these assumptions do limit the types of assemblies our system can handle, we believe that the overall framework of the system is sound. Relaxing any of these assumptions would require localized changes to modules within the framework rather changes to the framework itself.

8 Conclusions

We have described a set of design principles for designing effective assembly instructions that are easy to understand and follow. The principles are based on cognitive psychology research examining how people mentally represent and communicate the process of assembling an object. We have also demonstrated an automated system that instantiates these design principles and can substantially reduce the effort required to produce good assembly instructions.

Our key insight is that *planning* a sequence of assembly operations that is easy to understand and *presenting* those operations in a clear and concise manner are strongly interrelated problems. Our system is based on this idea and considers both problems in parallel as it is designing the instructions.

References

80/20 INC. 2003. www.8020.net.

BLANZ, V., TARR, M. J., AND BÜLTHOFF, H. H. 1999. What object attributes determine canonical views. *Perception 28*, 575–600.

BUTZ, A. 1997. Anymation with CATHI. In *Proceedings of AAAI/IAAI '97 in Providence / Rhode island*, AAAI Press, 957–962.

DE MELLO, L. S. H., AND SANDERSON, A. C. 1991. A correct and complete algorithm for the generation of mechanical assembly sequences. *IEEE Transactions on Robotics and Automation 7*, 2, 228–240.

DRISKILL, E., AND COHEN, E. 1995. Interactive design, analysis and illustration of assemblies. In *1995 Symposium on Interactive 3D Graphics*, ACM Press, 27–33.

FEINER, S. 1985. APEX An experiment in the automated creation of pictorial explanations. *IEEE Computer Graphics and Applications 5*, 11, 29–37.

GUIBAS, L. J., HALPERIN, D., HIRUKAWA, H., LATOMBE, J.-C., AND WILSON, R. H. 1995. A simple and efficient procedure for polyhedral assembly partitioning under infinitesimal motions. In *IEEE International Conference on Robotics and Automation*, IEEE, 2553–2560.

HEISER, J., AND TVERSKY, B. 2002. How to put things together. In *Poster presentation at the meeting of the Psychonomics Society*, Psychonomics Society.

LIN, M. C., AND CANNY, J. F. 1991. A fast algorithm for incremental distance calculation. In *IEEE Intern. Conf. on Robotics and Automation*, IEEE, 1008–1014.

MACKINLAY, J. 1986. Automating the design of graphical presentations of relational information. *ACM Transactions on Graphics 5*, 2, 110–141.

MARTIN, J. 1989. *High Tech Illustration*. North Light Books.

MIJKSENAAR, P., AND WESTENDORP, P. 1999. *Open Here: The Art of Instructional Design*. Joost Elffers Books, New York.

NOVICK, L. R., AND MORSE, D. L. 2000. Folding a fish, making a mushroom: The role of diagrams in executing assembly procedures. *Memory & Cognition 28*, 1242–1256.

PALMER, S., ROSCH, E., AND CHASE, P. 1981. Canonical perspective and the perception of objects. In *Attention and Performance IX*, 135–151.

QUINLAN, S. 1994. Efficient distance computation between non-convex objects. In *IEEE Intern. Conf. on Robotics and Automation*, IEEE, 3324–3329.

RAAB, A., AND RÜGER, M. 1996. 3D-ZOOM interactive visualization of structures and relations in complex graphics. In *3D Image Analysis and Synthesis*, 87–93.

RIST, T., KRÜGER, A., SCHNEIDER, G., AND ZIMMERMAN, D. 1994. AWI A workbench for semi-automated illustration design. In *Proc. of Advanced Visual Interfaces*, 59–68.

ROMNEY, B., GODARD, C., GOLDWASSER, M., AND RAMKUMAR, G. 1995. An efficient system for geometric assembly sequence generation. *Proc. ASME International Computers in Engineering Conference*, 699–712.

SELIGMANN, D. D., AND FEINER, S. 1991. Automated generation of intent-based 3D illustrations. In *Proceedings of SIGGRAPH 91*, 123–132.

STROTHOTTE, T. 1998. *Computational Visualization, Graphics, Abstraction and Interactivity*. Springer, ch. 13, 215–240.

TVERSKY, B., AND HEMENWAY, K. 1984. Objects, parts and categories. *Journal of Experimental Psychology: General 113*, 169–193.

TVERSKY, B., AGRAWALA, M., HEISER, J., LEE, P., HANRAHAN, P., STOLTE, C., AND DANIEL, M.-P. Submitted. Cognitive design principles for automated generation of visualizations.

WILSON, R. H. 1992. *On Geometric Assembly Planning*. PhD thesis, Stanford University.

WOLTER, J. D. 1989. On the automatic generation of assembly plans. In *Proc. IEEE International Conference on Robotics and Automation*, 62–68.

ZACKS, J., TVERSKY, B., AND IYER, G. 2001. Perceiving, remembering and communicating structure in events. *Journal of Experimental Psychology: General 136*, 29–58.