EUROGRAPHICS 2015 / O. Sorkine-Hornung and M. Wimmer
(Guest Editors)

Volume 34 (2015), Number 2

Interactive Disassembly Planning for Complex Objects

Bernhard Kerbl† Denis Kalkofen† Markus Steinberger† Dieter Schmalstieg†

Graz University of Technology, Austria

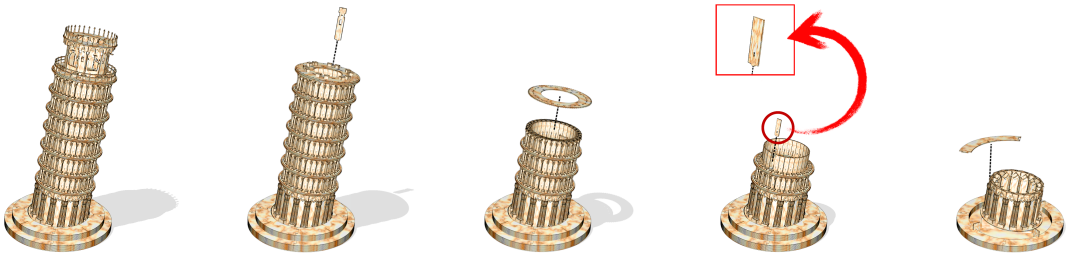


Figure 1: *Key frames from a disassembly sequence for the Leaning Tower CAD model. Our approach identifies reusable blocking information and evaluates the geometric feasibility of part removals in parallel. Thus, we are able to create disassembly sequences for models such as this 4,193 part assembly within minutes instead of days. Furthermore, unstable configurations are avoided by using structural constraints to elevate reproducibility on real-world objects. We facilitate the visual assessment of sequences by combining previewing techniques with visual cues to highlight smaller details.*

Abstract

We present an approach for the automatic generation, interactive exploration and real-time modification of disassembly procedures for complex, multipartite CAD data sets. In order to lift the performance barriers prohibiting interactive disassembly planning, we run a detailed analysis on the input model to identify recurring part constellations and efficiently determine blocked part motions in parallel on the GPU. Building on the extracted information, we present an interface for computing and editing extensive disassembly sequences in real-time while considering user-defined constraints and avoiding unstable configurations. To evaluate the performance of our C++/CUDA implementation, we use a variety of openly available CAD data sets, ranging from simple to highly complex. In contrast to previous approaches, our work enables interactive disassembly planning for objects which consist of several thousand parts and require cascaded translations during part removal.

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Computational Geometry & Object Modeling—Geometric algorithms, languages, and systems

1. Introduction

Disassembly planning describes the process of designing and editing a series of instructions on how to disassemble a given multipartite object. The desired result is a concise and reproducible disassembly sequence that describes the order and manner in which the parts of the object have to be removed to decompose it. A disassembly sequence can usually be inverted to yield a plan for assembly. Computer-aided disassembly planning systems aim to automatically identify feasible sequences to facilitate the design of disassembly and assembly plans for a wide range of applications [APH*03, LACS08, KTS09]. However, in the general case, finding a disassembly sequence has been shown to be NP-complete [KLW93], which is strongly reflected by the limitations of current approaches. Even moderately complex objects consisting of a few hundred parts are usually impossible to process at interactive rates. Furthermore, suitable editing mechanisms do not exist for large assemblies either. Consequently, the applicability of computer-aided disassembly planning is questionable, since existing methods cannot handle those cases where human individuals are most likely to seek assistance from automated systems.

† {kerbl,kalkofen,steinberger,schmalstieg}@icg.tugraz.at

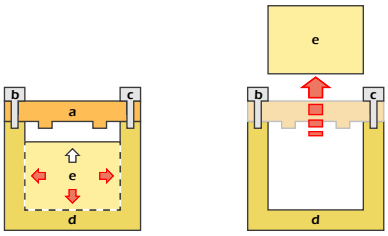


Figure 2: *Problems of conventional local interference testing. (left) Part* **e** *is to be extracted from a container. Since* **e** *touches* **d**, *local interference testing marks all directions indicated by red arrows as blocked. The interference of* **a** *with an upward motion of* **e** *is not detected. (right) The computed sequence suggests removing* **e** *through* **a**, *which is infeasible.*

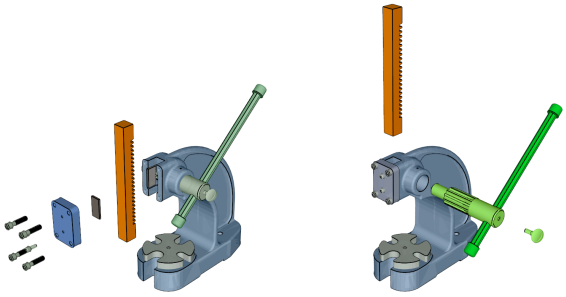


Figure 3: *Even for simple objects, multiple disassembly sequences exist for extracting the same parts. While the procedure on the left successfully extracts the ram from a press, it requires a hex driver that may be unavailable. In this case, the solution on the right is preferable.*

1.1. Problem

The major problem with disassembly planning for complex objects is the vast computational effort involved in *geometric reasoning*: Whenever a part is checked for removal, the number of paths this part can follow is infinite. When moving the part along a path, any other part could potentially block its motion. Thus, collisions between parts during removal need to be tested. To reduce the number of tests, previous approaches strongly limit the considered path motions and possible interferences [LACS08, GYL*13]. In doing so, they not only trade correctness for speed, but also limit themselves to disassembly sequences composed of very simple motions. Although these simplifications significantly reduce computational cost, they are still not sufficient to provide interactivity for large, complex objects. Like most existing work, we simplify the problem by focusing on translational motions only; part rotations will not be considered. By analyzing previous approaches in more detail, we identified four major challenges which an interactive disassembly planning approach for complex objects must overcome:

Geometric reasoning with many parts. When attempting to remove a part along a path, it is necessary to check if the movement is geometrically feasible, i.e., validating that there is no collision with other parts. While intersection tests with all other parts would actually be necessary to guarantee correctness, most approaches limit themselves to so-called local blocking relationships [Wil92]. A local blocking relationship $\Psi_L(A,B,\omega)$ determines whether applying a motion ω to a part A is directly prohibited by its contact with part B. The global blocking relationship $\Psi_G(A,B,\omega)$ considers the entire path of ω and detects any imminent collision of A and B during its execution. Although local testing has been shown to greatly improve runtime, handling highly complex assemblies such as the Leaning Tower model depicted in Figure 1 is still problematic, according to reported performance [RGGR95, LACS08, GYL*13]. Furthermore, ignoring potential interferences with more distant parts may result in solid objects passing through each other (see Figure 2).

Complex part motions. Creating feasible part motions is the basis for disassembly planning. Due to the possibly large number of parts, complicated geometric relations, and complex part geometries, an exhaustive evaluation of many different motions is generally not feasible. Thus, only subsets of possible motions are usually tested on a per part and path basis, i.e., whether the tested part can be removed along one of several given paths. To reduce computation cost, supported paths are usually restricted to straight-line translations [LW95, HLW00, GYL*13]. However, these simple motions are often insufficient for disassembling real-world objects.

Support for external constraints. Disassembly plans are generally created with a definite goal, for example, to extract a specific part. While the goal is usually unique, there may exist multiple different sequences to achieve it. In many cases, one specific sequence is preferable to another [KWJ*96], e.g., if a certain set of tools is not available (see Figure 3). In such a case, the user should be able to quickly search for alternative solutions. For large, complex objects, the computation of alternatives may involve a multitude of operations and thus very long waiting time. For an intentionally interactive process, such delays are not acceptable [JWC97].

Structural stability. Disassembly planning approaches usually do not consider the structural stability of the remaining assembly when removing parts [Wil92, RGGR95, APH*03, LACS08]. Global selective disassembly (GSD), for example, disassembles objects by iteratively removing unblocked parts, until all required parts have been extracted [SG02]. By selecting the next best removal, the remaining parts might be left disconnected or floating in midair. Especially for complex objects, such behavior must be avoided to provide the user with viable instructions.

The necessity to overcome these challenges is also reflected by our evaluation (Section 5). All but one input model in our test suite required global interference testing, complex part motions, and support for external constraints.

1.2. Contribution

In this paper, we tackle the aforementioned challenges by providing an approach that can process large, complex models with elaborate part movements, while enabling interactivity in disassembly planning. Our overall contributions are two-fold. First, we show how possible motions for thousands of parts can be efficiently evaluated. To this aim, we analyze input CAD models to identify reusable geometric information and provide a parallel approach to test many translating motions concurrently on the GPU. Second, we introduce new editing techniques tailored for effectively visualizing and altering the disassembly of large models in real-time.

2. Related work

Early disassembly planning systems focus exclusively on user interaction to relieve the underlying sequencing algorithms of their implied NP-completeness [DFW87]. Strategies for geometry-based disassembly planning have first been proposed by Homem de Mello and Sanderson [HdMS91]. Given that a method is provided for evaluating whether two parts can be separated, they generate an AND/OR graph that encodes all possible disassembly sequences. Wilson expands on this concept with GRASP, a fully automatic planner capable of decomposing simple objects [Wil92]. The AND/OR graph is generated from input geometry using a function that evaluates if parts can be separated using straight-line translations. Wilson originally states that computing the AND/OR graph becomes infeasible for assemblies with significantly more than 50 parts. Although several extensions have been proposed to make GRASP more versatile [HW95, LW95, HLW00], doing so additionally increases the computational demands and thus reduces the size of processible models even further. Instead of computing the complete AND/OR graph, Srinivasan and Gadh propose an algorithm that detects one sequence at a time to reduce runtime and memory consumption [SG02]. However, expensive geometric reasoning and unstable configurations are not addressed by their approach.

While geometric reasoning helps discard infeasible disassembly sequences, additional subjective real-world restrictions may be of relevance for object disassembly. In constraint-based disassembly planners, the application suggests a single sequence at a time. Kaufman et al. provide user interfaces for adding rules and constraints to modify the current suggestion until it satisfies the requirements of the user [KWJ*96]. Each modification requires the system to calculate a new, updated suggestion. While this approach is intended to be interactive, Jones et al. report that finding a constrained suggestion with conventional algorithms usually takes several minutes at each iteration for moderately complex objects, which significantly limits usability [JWC97].

Disassembly planning has several established applications in computer graphics. Agrawala et al. [APH*03] propose relevant criteria and explicit methods for automatically determining and visualizing comprehensible assembly sequences. Li et al. [LACS08] expand on this approach to create interactive 3D explosion diagrams from disassembly sequences. Similarly, Kalkofen et al. [KTS09] use disassembly planning to search for comprehensible explosion layouts. Depending on the goal of the current task, they present different search strategies to find Focus+Context part arrangements. Tatzgern et al. [TKS13] compute compact exploded views for small screen devices. Although all four systems restrict themselves to testing straight-line translations and local blocking relationships, the authors conclude that they are unable to handle complex objects (i.e., assemblies consisting of more than 100 parts) in an acceptable amount of time.

3. Efficiently extracting spatial information

At the core of our approach to interactive disassembly planning for complex objects are three concepts to speed up the evaluation of global blocking relationships. Blocking relationships are evaluated whenever a part should be removed from the assembly to determine if any other part blocks this movement. For complex objects with large numbers of parts, the evaluation of blocking relationships becomes the dominating factor. To provide high performance in our approach, we reduce the implied computational effort by identifying pairwise identical part constellations and storing reusable geometric information. Furthermore, we test a large number of straight-line translations in parallel using the GPU. Finally, we extend our approach to more complex, cascaded motions. We discuss these three concepts in the following subsections.

3.1. Generating reusable $\mathcal{C}$-Space obstacles

Our approach for part interference testing discards minuscule obstacles (e.g., bolt threading) via mesh erosion and employs the concept of configuration space ($\mathcal{C}$-Space) for collision detection [LP83]. $\mathcal{C}$-Space provides a suitable domain for testing the geometric feasibility of moving one object past another, based on the evaluation of their Minkowski difference. Inverting and sweeping A along the boundaries of B yields the Minkowski difference $A \ominus B$. In the case of polygon meshes, we obtain the 3D $\mathcal{C}$-Space obstacle $\mathcal{A}_B$ as shown in Figure 4. Iff a line from $\vec{0}$ in the direction of $\vec{d}$ intersects $\mathcal{A}_B$, translating A along $\vec{d}$ will cause it to collide with B. This information can be directly converted to provide global blocking relationships for parts in an assembly. For any two parts (A, B) and motion ω, $\Psi_G(A, B, \omega)$ is true, iff at least one of the line segments encoding the translations in ω intersects $\mathcal{A}_B$.

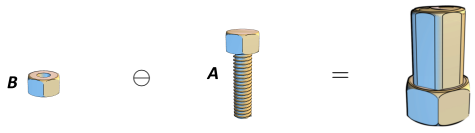
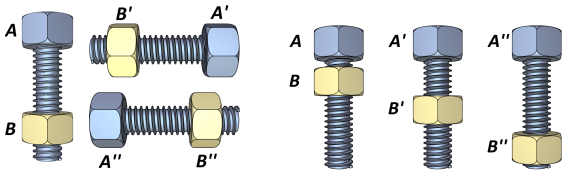


Figure 4: *Creation of a $\mathcal{C}$-Space obstacle. Calculating the Minkowski difference $B \ominus A$ yields the inflated obstacle $\mathcal{A}_B$.*

Given an assembly of N parts, testing all part pairings is achieved by generating and testing against $\mathcal{C}$-Space obstacles for all $\frac{N^2-N}{2}$ relevant combinations [Wil92]. However, doing so quickly becomes infeasible with increasing N, since even highly efficient methods for calculating the Minkowski difference of two polygon meshes are prohibitively expensive. To address this issue, we exploit the fact that most assemblies contain recurring relative part constellations and reuse the computed $\mathcal{C}$-Space obstacles whenever possible.

During our analysis, we create a catalog of unique part constellations. We iterate through all pairs of parts, and, before computing the $\mathcal{C}$-Space obstacle, we make sure that no identical constellation has been added to the catalog yet. In case an identical entry already exists, we can simply reference this entry. To identify duplicated part constellations, we search for parts with identical geometries and matching relative transformations. Let R_X, R_Y and $\vec{t}_X, \vec{t}_Y$ denote the rotation and translation of parts X and Y in the assembly. We define the relative rotation $R_{X\to Y}$ as $R_Y \cdot {R_X}^{-1}$ and the relative rotated translation $\vec{t}_{X\to Y}$ as $\vec{t}_Y - R_{X\to Y}\vec{t}_X$. If we find two part pairings with identical underlying part geometries (A,B) and (A',B') such that $R_{A\to B}$ equals $R_{A'\to B'}$, we can rotate $\mathcal{A}_B$ by $R_{A\to A'}$ and translate it by $(\vec{t}_{B\to B'} - \vec{t}_{A\to A'})$ to obtain ${\mathcal{A}'}_{B'}$. If $\mathcal{A}_B$ has already been generated and stored, the effort for computing a potentially complex $\mathcal{C}$-Space obstacle can thus be effectively reduced to a matrix transformation. The catalog entries consist of the two involved part geometries (A,B) and the relative rotation matrix $R_{A\to B}$, such that each part pairing in the assembly can be associated with one of these constellations. We then calculate representative $\mathcal{C}$-Space obstacles for each catalog entry. Examples of eligible constellations for reusing $\mathcal{C}$-Space obstacles are presented in Figure 5.



(a) Equal relative rotation (b) Translational part offset

Figure 5: *Reusability of $\mathcal{C}$-Space obstacles. (a) The relative rotation $R_{A\to B}$ is identical to $R_{A'\to B'}$ and $R_{A''\to B''}$. If we assume no part translations, rotating $\mathcal{A}_B$ by $R_{A\to A'}$ yields ${\mathcal{A}'}_{B'}$, while $R_{A\to A''}$ yields ${\mathcal{A}''}_{B''}$ (b) Translational differences in setups can be accounted for by shifting the rotated $\mathcal{A}_B$ by $(\vec{t}_{B\to B'} - \vec{t}_{A\to A'})$ and $(\vec{t}_{B\to B''} - \vec{t}_{A\to A''})$ respectively.*

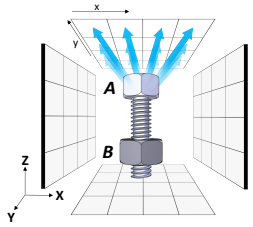
To measure the influence of reusable $\mathcal{C}$-Space obstacles, we define the $\mathcal{C}$-Space exhaustion for a given assembly as the ratio of necessary catalog entries to the number of obstacles generated with conventional testing. According to our experiments, reusing $\mathcal{C}$-Space obstacles can dramatically reduce computational effort for complex and repetitive assemblies, as will be shown in Section 5. During our assessment, we recorded $\mathcal{C}$-Space exhaustion rates as low as 1% and less.

3.2. Testing straight-line translations in parallel

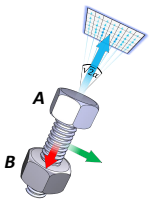
If a $\mathcal{C}$-Space obstacle $\mathcal{A}_B$ for a moving part A and a potentially blocking part B is available, checking for collisions when moving A along a straight line l can be reduced to intersecting l with $\mathcal{A}_B$. Let us now consider rendering a scene to a frame buffer as a collection of intersection tests: According to the OpenGL standard, a pixel is only drawn if at least one projected primitive covers the pixel center. After rendering, each updated pixel p thus encodes a positive result for an intersection test of the rendered scene with a line l_p running from the eye point through the pixel center on the view plane. If the camera is positioned at $\vec{0}$ and the rendered scene represents the $\mathcal{C}$-Space obstacle $\mathcal{A}_B$, p contains the result for a collision test of A and B when moving A along l_p. We exploit this fact and provide methods for using OpenGL rasterization to test many motions in parallel on the GPU. For each part, we combine broad sampling of the entire space of straight-line translations with dense sampling for those directions that are most likely used for its removal.

Broad sampling. The space of straight-line translations in 3D can be represented by mapping it to the surface of a convex polyhedron. A discrete evaluation of the complete space can be obtained by distributing samples on a unit cube and testing all vectors leading from its center to the sample points on its surface. We use cube maps to evaluate numerous discrete sample locations on the unit cube during rendering. We allocate six textures with resolution $r_x \times r_y$ and attach them to a framebuffer object (FBO) to store the result from rendering the $\mathcal{C}$-Space obstacles. Camera poses are chosen in such a way that after rasterization, the textures contain the six 2D projections for the sides of a unit cube centered at $\vec{0}$. This enables us to quickly extract the blocking relationships for a large number of pre-defined, discretely sampled straight-line translations (see Figure 6(a)). For instance, an updated pixel value at the normalized coordinate (x,y) in the texture containing the XY projection on the positive Z axis indicates that A is blocked by B in direction $[x,y,1]^T$. The granularity for broad sampling and thus the number of tested motions can be directly defined by choosing r_x and r_y accordingly.

Dense sampling for preferred directions. Based on the shape of specific parts, we can infer straight-line translations that have a high probability of being a viable removal action. For instance, a cylindrical body (e.g., a bolt or screw) is most likely extracted along its primary principal axis. We utilize these priors to put special focus on corresponding directions. Specifically, we consider the positive and negative extent of the three principal axes of each individual part. The axes are obtained by applying a principal component analysis (PCA) on a part's 3D vertex coordinates. For each specified candidate direction $\vec{v}$, we test a multitude of similar motions

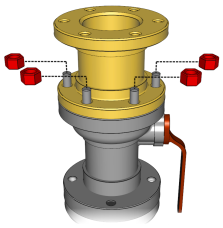


(a) Broad sampling

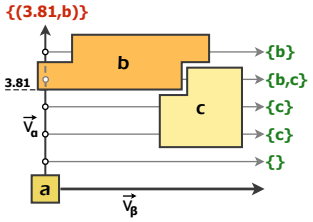


(b) Dense sampling

Figure 6: *Efficiently testing translating motions. (a) For any two parts A and B, we sample the space of straight-line translations in parallel by rendering $\mathcal{A}_B$ to a cube map. (b) We identify preferred directions for which we use dense sampling around the principal axes with angular tolerance α.*



(a) Wedged bolt nuts



(b) Testing dual translations

Figure 7: *Dual translating motions. (a) The red bolt nuts in the valve model require right-angular dual translations for disassembly. (b) Evaluating dual translation of part* **a** *with two other parts* **b** *and* **c**. *We show resulting primary blockers (red) and secondary blocking relationship sets (green).*

by densely sampling inside an enveloping cone. An angular value α can be chosen to define the maximum deviation from $\vec{v}$. Again, we use OpenGL to render $\mathcal{A}_B$ to an $r \times r$ texture that is attached to an FBO. The camera parameters are chosen such that the view vector coincides with $\vec{v}$, and the field-of-view is set to $\sqrt{2}\alpha$. After rasterization, each unmodified pixel value in the texture is not covered by the projection of $\mathcal{A}_B$ and thus corresponds to an unblocked direction $\vec{w}$, where $\angle(\vec{v},\vec{w}) \leq \alpha$. If such a $\vec{w}$ exists, A is considered to be unblocked by B along $\vec{v}$ with respect to the tolerance parameter α. By focusing on directions that are likely to represent intended part motions, dense sampling effectively complements general, broad sampling.

3.3. Testing dual translations

In many scenarios, testing straight-line translations is not sufficient for disassembly. Dual translations, i.e., motions that consist of two concatenated translations, are reasonably easy to perform and thus often implied for real-world objects like the Valve in Figure 7(a). We propose a parallel, discrete method for evaluating global blocking relationships of dual motions through rasterization of $\mathcal{C}$-Space obstacles.

A dual translation ω_2 is defined by two normalized vectors $\vec{v_\alpha}, \vec{v_\beta}$ representing the directions of the two translating motions and a scalar l_α indicating the extent of the first translation. Given a part's assembled position $\vec{p}$, the midpoint $\vec{m}$ of ω_2 which marks the end of the first translation is given by $\vec{m} = \vec{p} + l_\alpha \vec{v_\alpha}$. The second translation in the direction of $\vec{v_\beta}$ continues from $\vec{m}$ until the part is visibly separated from the remaining assembly. We focus exclusively on right-angular motions, which satisfy $\vec{v_\alpha} \perp \vec{v_\beta}$. To this aim, we evaluate all perpendicular combinations of directions from the part's principal axes as detected by PCA.

Given a dual translation ω_2 of a moving part A and a potential blocker B, we first determine the distance of the closest interference, if any, along $\vec{v_\alpha}$. In our implementation, this information is accessed by interpreting the contents of the depth buffer after testing straight-line translations. The closest interference and an identifier for the blocking part are then added to a set of *primary blockers* of ω_2. Next, we test for collisions of A with B during the second translation, which may start at any one $\vec{m}$ out of multiple options. We consider possible locations for $\vec{m}$ at N evenly spaced intervals along $\vec{v_\alpha}$ in the range $[0,d]$, where d is the diameter of the assembly's bounding sphere, and N is a granularity parameter. For each $\vec{m}$, we maintain a set of *secondary blocking relationships* for moving A to infinity in the direction of $\vec{v_\beta}$. Assuming that all references to parts are deleted upon their removal, a dual translation ω_2 is feasible as soon as an empty set of secondary blocking relationships exists for a midpoint that is closer than the nearest primary blocker. Figure 7(b) illustrates the principle of testing dual motions in an exemplary setup.

We can generate all secondary blocking relationships of A and B concurrently by rendering $\mathcal{A}_B$ with orthogonal projection along $\vec{v_\beta}$ and up-vector $\vec{v_\alpha}$ to a one dimensional FBO with N pixels. The interpretation of the buffer contents is analogous to the procedure described in Section 3.2, as each pixel encodes the influence of B on a blocking relationship set.

4. Interactive editing of disassembly sequences

Building on our techniques for fast evaluation of blocking relationships, we tackle the challenges of interactive editing of disassembly sequences. With our work, we aim to fulfill the following three requirements. First, computing and updating a sequence should be fast to avoid long waiting times, while ensuring constraints of reproducibility. Second, the user must be able to easily review and analyze suggested solutions to identify unwanted or suboptimal instructions. Third, necessary interactions with the application to change the current sequence should be easy and intuitive.

4.1. Fast generation of stable sequences

For our planning application, we use an approach that is similar to GSD, which allows us to produce and recompute results

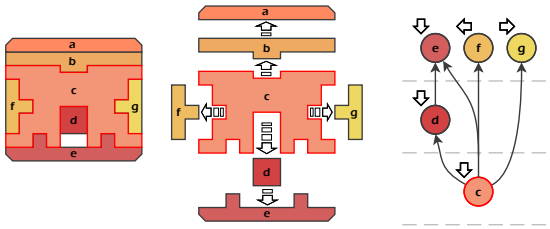


Figure 8: *Sequencing using GSD. (a) The assembled model with selected target part* **c**. *(b) Feasible removal actions for the individual parts as detected by GSD. (c) White arrows in the output arrangement indicate removal actions, black arrows mark blocking relationships. Parts* **a** *and* **b** *are pruned, since they do not block a downward extraction of* **c**.

at interactive rates [SG02]. Given a number of target parts, GSD uses the information available through global blocking relationships to iteratively "peel" all unblocked parts in the assembly until the targets have been extracted. Subsequently, blocking relationships of detected removal actions are analyzed to prune unnecessary removals. The output of GSD is a causally organized arrangement of required removal actions (see Figure 8).

While GSD does not consider contact coherence or structural stability, we believe these features are very important for real world applications. Thus, our approach encourages connected, stable configurations, as outlined in Figure 9.

To prevent situations where "floating" parts would be left hanging in the air, we create a *contact graph* of the assembly and monitor this graph during sequencing. The contact graph contains a node for each part of the assembly. Edges are inserted for parts that share at least one mutual contact surface in their assembled state. Floating parts can then be avoided by rejecting all part removals that leave the contact graph of the remaining assembly disconnected. If peeling all target parts succeeds under this constraint, at least one contact-coherent disassembly sequence can be found. In this case, we can compute a coherent and preferably stable disassembly sequence.

To determine whether a removal sequence threatens structural stability, we run the following algorithm. We first define $\mathbf{A}$ as the set of all parts in the entire assembly. Next, we find a reference part P_Ω by choosing either a random part that is still present after the last target has been peeled, or, if no others remain, the last target itself. After GSD has finished pruning the list of removed parts, we store them in a set $\mathbf{S}$. We then find the set of parts $\mathbf{C}$ that can be reached in the contact graph of $\mathbf{A} \setminus \mathbf{S}$ using P_Ω as a seed. If $\mathbf{C} \neq \mathbf{A} \setminus \mathbf{S}$, removing $\mathbf{S}$ leaves all parts in $\mathbf{A} \setminus (\mathbf{C} \cup \mathbf{S})$ disconnected or floating. Hence, we add them to the list of required target parts and recompute the corresponding disassembly sequence. Since the newly generated sequence may again lead to unstable configurations, we

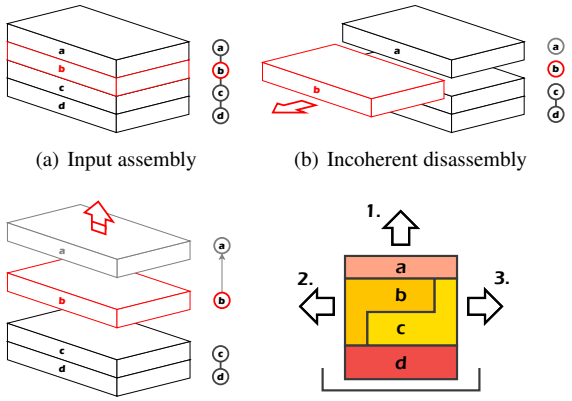
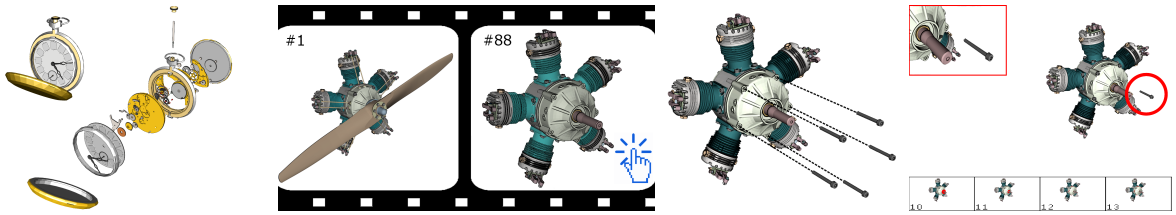


Figure 9: *Generating a stable disassembly sequence by considering the contact graph (shown next to the illustrations). (a) A simple brick assembly with the selected target in red. (b) GSD suggests a motion that leaves part* **a** *suspended in mid-air. The contact graph is clearly disconnected. (c) Our algorithm determines that the smaller subgraph comprised by* **a** *should also be removed to avoid floating parts. (d) If multiple options are available, parts are chosen in an order that minimizes instability during their removal.*

repeat this procedure until $\mathbf{C} = \mathbf{A} \setminus \mathbf{S}$ is fulfilled. In this case, the newly generated, pruned output arrangement G contains all necessary removal actions to create a sequence that avoids interferences as well as floating parts.

After determining that a structurally stable sequence can be generated with the identified removal actions, we need to order the instructions to create the actual sequence. This is achieved by interpreting disassembly as reverse assembly: Starting with the foundation of unremoved parts, we progressively attach parts from G until the object has been fully restored. We initialize the foundation $\mathbf{F}$ by adding all parts that are not referenced in G. Next, we copy all parts in G with no incoming edges to the set of candidate attachments $\mathbf{X}$. We select the part $x \in \mathbf{X}$ that shares a direct contact with a part in $\mathbf{F}$ and whose attachment is least likely to diminish the stability of the foundation. In our implementation, we estimate the introduced instability by the sum over the distances between the centroid of x and the centroids of the parts that are already in $\mathbf{F}$. This criterion causes the algorithm to prefer the attachment of parts close to the current center of gravity, which eventually leads to the behavior illustrated in Figure 9(d) during disassembly. The chosen part x is subsequently moved from $\mathbf{X}$ to $\mathbf{F}$. All outgoing edges of x are removed from G. If this causes another node $y \in G$ to lose its last incoming edge, y is subsequently added to $\mathbf{X}$. This procedure is repeated until $\mathbf{X} = \emptyset$. A stable and coherent disassembly sequence is created by removing parts in the inverse order in which they were added to $\mathbf{F}$.



(a) (b) (c) (d)

Figure 10: *We combine multiple visualization techniques to allow switching between views with coarse-to-fine navigation. (a) Explosion diagram for the Pocket Watch (b) Fast-forward animations showing a radial engine over time during disassembly. (c) Clicking an exploded part or stopping at a particular frame during fast-forward causes the application to load the corresponding action diagram. (d) Progressive animation for the current action diagram demonstrates each part motion separately.*

4.2. Detail-on-demand visualization

For quickly locating errors or unwanted instructions in the current disassembly sequence as well as for reviewing the final result, we provide a collection of visualization techniques. In order to prevent visual clutter and tedious navigation due to vast numbers of actions being displayed, our planning application allows for interactively exploring disassembly sequences on multiple levels of detail:

- **3D exploded views** provide an overview for assemblies that involve a limited number of instructions ($\lesssim$40) and removal directions ($\lesssim$6), such as the Pocket Watch shown in Figure 10(a). Using explosion diagrams for more extensive and varied disassembly sequences usually leads to visual clutter [LACS08, TKS10].
- **Fast-forward animations** allow to present previews of long and diverse sequences. Instead of spatially separating parts from each other, sequences are presented by displaying the structural development of the assembly over time using screen captures, as illustrated in Figure 10(b). The user may stop the fast-forward animation at any time to obtain further detail on individual steps.
- **Action diagrams** combine multiple instructions in a single illustration by offsetting a limited number of parts and adding guide lines according to their removal action. Figure 10(c) gives an example that provides overview for a small set of actions, while preventing visual clutter.
- **Progressive animations** offer the most detailed view on disassembly sequences by animating all part motions separately. The animation is looped continuously, until the user decides to either abort or advance to the next step. Visual cues are added for highlighting smaller parts. Additional means for quick navigation are provided by thumbnail previews at the bottom of the screen (Figure 10(d)).

We provide intuitive navigation between overview and detail visualizations by linking the individual techniques. Selecting a single part in a 3D exploded view or pausing the fast-forward animation automatically loads the corresponding action diagram. The user can then start the progressive animation for displaying the part removals one by one. Alternatively, she can navigate between different stages of the disassembly sequence by scrolling through the preview thumbnails at the bottom of the screen (see Figure 10(d)) and selecting a particular thumbnail to load the corresponding action diagram. The user can further choose whether computed sequences should be displayed as assembly or disassembly plans. The switch is easily executed by reversing the order and directions used for the action diagrams and animations. For a demonstration of interaction with our visualization suite, please refer to the accompanying video material.

4.3. Modifying disassembly sequences

In addition to the ensured constraints of basic reproducibility, the user may wish to define additional restrictions to adapt a disassembly sequence according to personal preferences. To enable editing of sequences within the boundaries of geometrical feasibility, the user sets constraints that are then considered during the computation of a modified solution.

Basic constraints. Our interactive planning application supports editing disassembly sequences by interactively setting constraints for individual parts or part groups. Specifically, we allow altering hierarchical and directional preferences.

Partitioning allows the user to fuse multiple parts or subassemblies to form a bigger subassembly. Conversely, collapsing a subassembly divides it into the fragments from which it was created. During disassembly sequencing, each subassembly is treated as an indivisible entity, until it has been separated from the remaining structure. Partitioning can thus be used to handle interlocking part groups and introduces hierarchical behavior to the sequence.

Designated removal actions can be selected for each individual part or partition. Our sequencing algorithm ensures that each part with a fixed removal action is removed in the specified way. The user is free to choose any removal action for which blocking relationships are already available. In addition, custom removal actions may be specified for individual parts. Our interface supports the definition of cascaded

translations of arbitrary length. The corresponding blocking relationships can be calculated on-the-fly.

Detecting recurring constellations. Real-world objects often contain identical structures in multiple places for reasons of functional or artistic design. In many cases, the user may be interested in locating other instances of a specific constellation, e.g., because all such part groups require replacement. However, the effort for doing so significantly increases with the complexity of the assembly. Our interface provides an original method for automatically detecting all part groups that resemble the current selection. Given a connected set of one or more parts $\mathbf{S}$, we store the selected part geometries and underlying contact graph C_S. We then locate every instance P_i of a random part $P \in \mathbf{S}$ and test if we can reproduce C_S using P_i as a seed node. Finally, we verify that all parts that were involved in reproducing C_S have the correct orientation with respect to P_i. If so, we add this group of parts to the list of detected recurrences. Usage of the *recurring constellations* operator is shown in Figure 11.

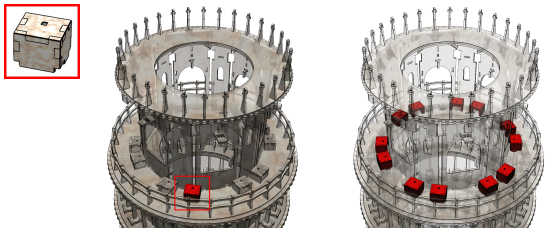


Figure 11: *(left) A group of four individual parts is selected and shown in red. (right) Other instances of the same part constellation are automatically detected and highlighted.*

Propagating constraints. Once a suitable constraint has been identified for a part or partition, the user may want to set it on other instances of those part groups as well. This may be done in order to achieve uniformity in the disassembly sequence or because the constraint is necessary and identical for all instances. We allow propagating basic constraints by finding similar part constellations for the current selection and applying the chosen modification to all detected recurrences of the selected part or partition. We have found that this option makes interaction with repetitive assemblies, such as the Leaning Tower, much faster and less tedious.

Splitting stack assemblies. Although basic partitioning enables hierachical structures during disassembly, it requires explicit, exhausting selection of all parts that should be grouped together. For stack-shaped models, we offer a facilitating mechanism that uses spatial analysis to quickly split a large assembly into multiple smaller subassemblies. The planning application enables the user to indicate parts that should act as separating elements for the new partitions. The operator first extracts the stacking axis by calculating a linear least squares fit through the centroids of the selected separators. The new partitioning is then obtained by identifying and greedily fusing all connected components that lie entirely between the axial extrema of two separators with regard to the stacking axis. Additional partitions are created from the remaining parts at the top and bottom of the stack. Figure 12 shows the Eiffel tower model being split into four subassemblies.

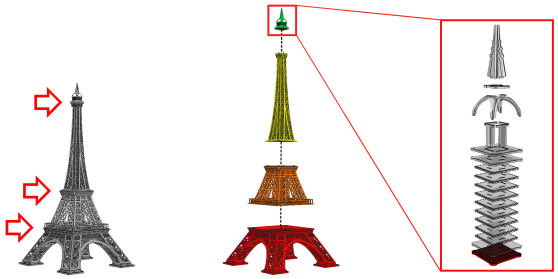


Figure 12: *Using the 'Split Stack' operator of our planning application with separators in the indicated places, a complex assembly can be quickly divided into smaller subassemblies.*

4.4. User controls

Our implementation allows the user to interact through the default interfaces of a personal computer. For spatial navigation, the camera can be dragged, rotated or zoomed in on particular sections of the scene. The user can select individual parts of the assembly, for which she may then either define a designated removal action, set a partitioning constraint (*Fuse*, *Collapse*) or apply the *Split Stack* operation. Furthemore, the most recently set constraint can be propagated throughout the entire assembly, if desired. All modifications can be performed via hotkeys or a concealable toolbar menu. At any point, the user is free to initiate the computation of a new disassembly sequence, with regard to all defined constraints, and view the current solution. Once a sequence has been computed, it can be explored in overview or detail, as described in Section 4.2. While assessing the visualizations, the user may notice the necessity for adding further constraints and resume editing until satisfied.

5. Evaluation

We evaluate our approach using a variety of CAD data sets that were obtained from an openly accessible online repository. The attributes, feature requirements, preprocessing and planning run times for each assessed model are listed in Table 1. The performance was recorded on a personal computer with an Intel i7-4771 CPU @ 3.50 Ghz, 16 GB RAM and an NVIDIA GeForce GTX 780 Ti with 3 GB graphics memory. We implemented the program logic in C++ and used OpenGL/CUDA for parallel and rendering-based tasks. Our implementation uses a method by Li and McMains [LM11] for calculating Minkowski differences of polygon meshes on the GPU to create renderable $\mathcal{C}$-Space obstacles. All assemblies except for the Leaning Tower were evaluated with

$r_x = r_y = 45$ for broad sampling. To reduce the amount of generated output data for the Leaning Tower assembly, we only stored computed blocking relationships for the principal directions of its parts. We chose $r = 255$ for dense sampling of principal directions in all assemblies with an angular tolerance of 2°. We used $N = 9$ midpoints for sampling dual translations in the Leaning Tower assembly and $N = 255$ midpoints for all others. Figure 14 demonstrates some of the results that were generated using our implementation.

We distinguish two separate processing steps in our evaluation. The preprocessing stage is concerned with the efficient computation of global blocking relationships. This step needs to be executed only once for each input assembly. The runtime for preprocessing, T_{prep}, thus presents a one-time-only effort. Once the information has been generated for a model, the user can start the interactive planning application. For this second step, we consider T_{plan}, which denotes the maximal time it takes to compute a constrained sequence. All of our examined data sets could be preprocessed in a couple of minutes. During planning, interaction with assemblies is highly responsive, with T_{plan} never exceeding three seconds.

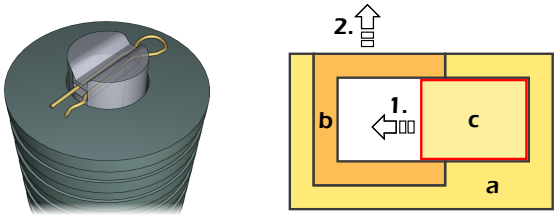
Although parallelization obviously grants our approach enhanced performance, it by itself does not suffice to enable processing of highly complex objects. Reusing $\mathcal{C}$-Space obstacles also plays an essential role. For the Leaning Tower model, a naïve, parallel evaluation of blocking relationships did not finish within 24 hours and was aborted. As can be read from Table 1, the $\mathcal{C}$-Space exhaustion appears to decline with increasing assembly size. Thus, it seems that large objects tend to show more recurring part constellations, increasing the value of reusing $\mathcal{C}$-Space obstacles. Combining parallelization and $\mathcal{C}$-Space obstacle reusability therefore yields best performance for complex objects.

During disassembly planning, we found that in all eight test case scenarios, using only local interference testing causes irreproducible sequences. As seen in Table 1, the number of parts that would erroneously be detected as removable with local blocking relationships varied between 17% and 57% among those seven scenarios. Consequently, it can be concluded that global testing is essential for disassembly planning. Also, we found that straight-line translations were not sufficient, but dual translations were required in seven out of eight scenarios. The number of parts requiring dual translations among those scenarios was between 1% and 36%. These results clearly underline the necessity to support more than straight-line translations. Dual translations were however sufficient in all scenarios. Finally, we want to note that in all but one scenario, some user modifications were required to create the disassembly plans shown in the paper, which underlines the importance of interactive editing capabilities.

To assess the proposed visualization and editing mechanisms, we asked five scientists with a computer graphics background to rate the usability for interactive disassembly planning on the largest assemblies listed in Table 1. After a brief introduction, all five users were able to quickly identify necessary steps for modifying sequences according to their wishes. We received positive feedback regarding the handling of our tool and the employed visualizations, which were described as both pleasing and helpful for navigating in disassembly sequences. The users also appreciated the almost instantaneous response time when adding constraints. Finally, they highly valued the propagation of constraints to reduce the time spent on editing. Suggested improvements were concerned with making thumbnails for progressive animation more expressive, e.g., by highlighting or focusing on regions of change. Furthermore, they recommended adding means for listing all user-added constraints and visually relating them to corresponding parts in the assembly.

6. Limitations

The outlined approach aims to extend the range of processible models by ensuring high fidelity in interference testing and considering removal actions beyond simple straight-line motions. However, considering translations alone is sometimes insufficient for detecting viable disassembly sequences in real-world use cases. A common example would be the twisting motions that are required to remove fixtures, such as nuts and bolts, from an assembly. Although these particular cases can be handled in our application by using mesh erosion to shrink away minor obstacles, other scenarios exist where our method fails to produce a solution. For instance, the algorithm is ignorant of part deformability. Wires, plastic or metal components may require some sort of compression or expansion to enable their removal. Figure 13(a) illustrates this case on a bendable metal pin that is used for keeping a cylinder cover in place. Furthermore, there is no automatic testing for temporary part displacements, which commonly occurs in geometric riddles or puzzle games (see Figure 13(b)). Efficiently handling those cases for complex objects remains a challenging task for future research.



(a) (b)

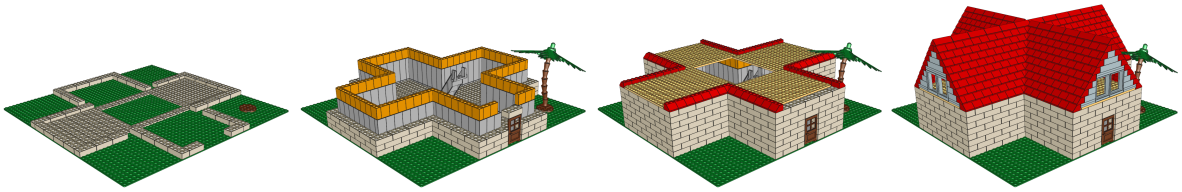
Figure 13: *(a) Assembly involving a deformable object. Lifting the cylinder cover upward is prohibited by a bendable metal pin, that needs to be compressed before it can be removed. (b) Multi-stage part removal of* **c**. *To solve this puzzle, part* **c** *needs to be moved to the left to an intermediate position. In this way,* **b** *and* **c** *can be extracted together as one.*

Table 1: *Attributes and collected results for all tested assemblies. We list the number of created representative $\mathcal{C}$-Space obstacle and compare it to conventional testing ($\mathcal{C}$-Space exhaustion). For each assembly, we list the number of parts that at some point during disassembly had no local, but global interference, as well as parts that required dual motion testing. We also give the number of necessary interactions. T_{prep} and T_{plan} list runtimes for preprocessing and sequencing assemblies, respectively.*

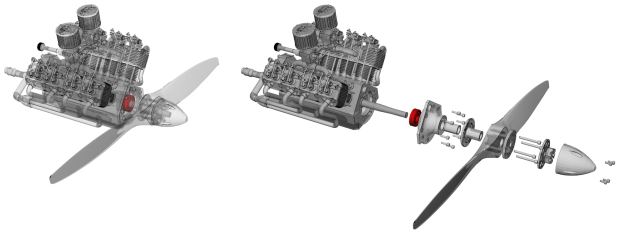
	Model attributes			Interference			Runtimes	
Assembly	Parts	Triangles	$\mathcal{C}$-Space exhaustion	Global	Dual	Interaction	T_{prep}	T_{plan}
Arbor Press, Fig. 3	22	30,857	150 (64.93%)	4	1	2	17s	<1s
Valve, Fig. 7(a)	38	84,614	601 (85.63%)	14	11	1	37s	<1s
Pocket Watch, Fig. 10(a)	132	133,268	3,655 (42.27%)	75	2	6	1m 45s	1.1s
Radial Engine, Fig. 10(b)	239	265,080	12,271 (43.14%)	64	23	12	2m 45s	1.7s
V8 Engine, Fig. 14(c)	512	537,385	19,675 (15.04%)	92	77	5	6m 37s	2.1s
Eiffel Tower, Fig. 12	735	167,408	31,420 (11.65%)	167	114	4 / 1†	5m 16s	2.5s
LEGO Mansion, Fig. 14(a)	1,840	910,400	849 (0.05%)	320	0	0	13m 40s	2.2s
Leaning Tower, Fig. 1	4,193	511,204	73,734 (0.83%)	1185	390	360 / 3‡	28m 53s	2.9s

† One-by-one disassembly requires partitioning 4 interlocking part groups. The scene in Figure 12 is created by one *Split Stack* operation.

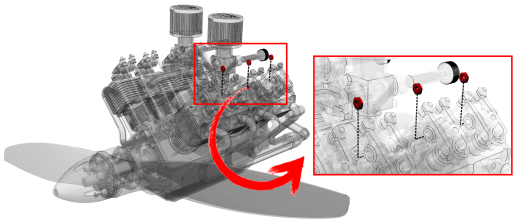
‡ The 360 interlocking partitions in the assembly can be fused using the *Propagate Constraints* operation on 3 representative part groups.



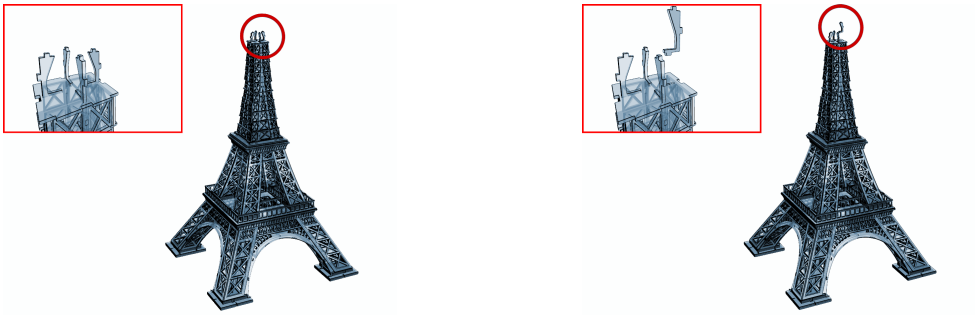
(a) Screen captures from the fast-forward preview animation for assembling the LEGO Mansion model.



(b) Explosion diagram for extracting the highlighted part from an engine.



(c) Action diagram of dual motions. Inset added by hand.



(d) Progressive animation with automatic close-up and highlight for removing a small part from the Eiffel Tower.

Figure 14: *Various examples demonstrating the results and visual output produced by our approach. (a) Fast-forward animation and (b) 3D exploded views provide an overview of complex as well as simple sequences. (c) Action diagrams enable to quickly review subsequences while (d) progressive animation enables inspecting every single instruction in full detail.*

7. Conclusions and future work

We have presented a complete method for planning assembly and disassembly sequences on highly complex objects. We have described algorithms for efficiently calculating blocking relationships of multiple straight-line and dual translating motions in parallel by exploiting GPU acceleration. We have provided a prototype implementation, which processes assemblies that require cascaded removal actions and contain vast numbers of parts in a matter of minutes. Compared to previous reports, the accumulated performance gain enables us to process assemblies that are more than 40 times larger. The unprecedented ability to handle such intricate data sets opens up novel insights regarding special requirements for planning and editing complex disassembly sequences. In order to address these challenges, we have outlined an interactive planning application that combines several visualization techniques to provide detail-on-demand exploration. Furthermore, we have introduced editing mechanics for conveniently performing repetitive modifications on large assemblies.

Our work aims to extend the range of processible data sets by enabling its application to highly complex objects. However, another very common reason for inapplicability are the structural problems that occur in amateur 3D designs, such as self-intersecting assemblies or degenerate geometries. Furthermore, elaborate removal actions that require rotation, twisting or compression of parts are currently not possible to evaluate efficiently for complex objects. In addition, consideration of physical attributes of parts could be incorporated to allow for highly sophisticated choices during the generation of stable sequences. These open issues provide ample potential for future work that will improve the user value of computer-assisted disassembly planning even more.

Acknowledgements

This work was funded by the Austrian Science Fund (FWF) under contract P-24021. All models were used with the consent of the corresponding CAD authors. We would like to thank: Jason Mitchell, Los Angeles Pierce College for the Arbor Press; Adrian Mazufri, Universidad Nacional de Río Cuarto for the Pocket Watch; Michael Hahn for the Radial Engine; Christopher Dabek for recreating the original V8 Engine design by Eric Whittle; Richard K. Lowe Jr of LoweandBehold Designs and DJFabricators Inc., as well as Pranav Panchal for their combined effort on creating the Eiffel Tower and Leaning Tower models.

References

[APH*03] AGRAWALA M., PHAN D., HEISER J., HAYMAKER J., KLINGNER J., HANRAHAN P., TVERSKY B.: Designing effective step-by-step assembly instructions. *ACM Trans. Graph. 22*, 3 (July 2003), 828–837. 1, 2, 3

[DFW87] DE FAZIO T., WHITNEY D.: Simplified generation of all mechanical assembly sequences. *Robotics and Automation, IEEE Journal of 3*, 6 (1987), 640–658. 3

[GYL*13] GUO J., YAN D.-M., LI E., DONG W., WONKA P., ZHANG X.: Illustrating the disassembly of 3D models. *Computers & Graphics 37*, 6 (2013), 574–581. 2

[HdMS91] HOMEM DE MELLO L., SANDERSON A. C.: A correct and complete algorithm for the generation of mechanical assembly sequences. *IEEE Trans Robotics and Automation 7*, 2 (1991), 228–240. 3

[HLW00] HALPERIN D., LATOMBE J.-C., WILSON R. H.: A general framework for assembly planning: The motion space approach. *Algorithmica 26*, 3-4 (2000), 577–601. 2, 3

[HW95] HALPERIN D., WILSON R. H.: Assembly partitioning along simple paths: the case of multiple translations. In *ICRA* (1995), IEEE Computer Society, pp. 1585–1592. 3

[JWC97] JONES R. E., WILSON R. H., CALTON T. L.: Constraint-based interactive assembly planning. In *IEEE Robotics and Automation* (1997), vol. 2, IEEE, pp. 913–920. 2, 3

[KLW93] KAVRAKI L., LATOMBE J.-C., WILSON R. H.: On the complexity of assembly partitioning. *Information Processing Letters 48*, 5 (1993), 229 – 235. 1

[KTS09] KALKOFEN D., TATZGERN M., SCHMALSTIEG D.: Explosion diagrams in augmented reality. In *VR* (2009), IEEE, pp. 71–78. 1, 3

[KWJ*96] KAUFMAN S. G., WILSON R. H., JONES R. E., CALTON T. L., AMES A. L.: The archimedes 2 mechanical assembly planning system. In *ICRA* (1996), pp. 3361–3368. 2, 3

[LACS08] LI W., AGRAWALA M., CURLESS B., SALESIN D.: Automated generation of interactive 3D exploded view diagrams. *ACM Trans. Graph. 27*, 3 (Aug. 2008), 101:1–101:7. 1, 2, 3, 7

[LM11] LI W., MCMAINS S.: Voxelized minkowski sum computation on the gpu with robust culling. *Comput. Aided Des. 43*, 10 (Oct. 2011), 1270–1283. 8

[LP83] LOZANO-PEREZ T.: Spatial planning: A configuration space approach. *IEEE Trans. Comput. 32*, 2 (Feb. 1983), 108–120. 3

[LW95] LATOMBE J.-C., WILSON R. H.: Assembly sequencing with toleranced parts. In *ACM Solid Modeling and Applications* (New York, NY, USA, 1995), SMA '95, ACM, pp. 83–94. 2, 3

[RGGR95] ROMNEY B., GODARD C., GOLDWASSER M., RAMKUMAR G.: An efficient system for geometric assembly sequence generation and evaluation. In *Proc. ASME Int. Computers in Engineering Conference* (1995), pp. 699–712. 2

[SG02] SRINIVASAN H., GADH R.: A non-interfering selective disassembly sequence for components with geometric constraints. *IIE Transactions 34*, 4 (2002), 349–361. 2, 3, 6

[TKS10] TATZGERN M., KALKOFEN D., SCHMALSTIEG D.: Compact explosion diagrams. In *ACM NPAR* (2010), pp. 17–26. 7

[TKS13] TATZGERN M., KALKOFEN D., SCHMALSTIEG D.: Dynamic compact visualizations for augmented reality. In *VR* (2013), IEEE, pp. 3–6. 3

[Wil92] WILSON R. H.: *On geometric assembly planning*. PhD thesis, Stanford University, 1992. 2, 3, 4