Graz University of Technology

Institute for Computer Graphics and Vision

Master's Thesis

A Framework for Automatically Creating 3D Explosion Diagrams

Markus Tatzgern

Graz, Austria, October 13, 2008

Thesis supervisors

Univ.-Prof. Dipl.-Ing. Dr. techn. Dieter Schmalstieg

Dipl.-Ing. Denis Kalkofen

Abstract

This thesis presents a framework for automatically calculating explosion diagrams from pure geometrical data. Diagrams are created by utilizing data structures and algorithms from the assembly planning domain. Four parameters influencing the final layout of the exploded view are identified. Firstly, the partitioning of the assembly is responsible for disassembling the input model either by removing single parts or groups of parts. Secondly, relations between the exploded parts are established, ensuring that part movements are propagated between associated parts. Thirdly, separation directions and distances are selected. A fourth parameter is responsible for introducing layers into the explosion, thereby exploding certain parts at the same point in time. It is shown, how these parameters can be used to create symmetrical explosion diagrams for exploratory tasks, or an explosion focused on a certain part of interest. The relations between the parts are visualized by either simple guide lines or a static blur, stretching the exploded part to its original position and blurring it. To be able to follow the explosions, different types of animation strategies are provided by a force-based layout approach. These are stepwise explosions of single parts or groups of parts, a layered explosion removing parts having the same properties at once, or exploding all parts at once.

Keywords. explosion diagrams, assembly sequencing, layout, visualization aids

Kurzfassung

Diese Arbeit präsentiert System zur automatischen Berechnung von anwendungsabhängigen Explosionsdiagrammen aus rein geometrischen Daten. Die Diagramme werden unter Verwendung von Datenstrukturen und Algorithmen aus dem Fertigungsbereich erzeugt. Vier Parameter werden identifiziert, die das Layout des fertigen Explosionsdiagramms beeinflussen. Erstens, die Aufteilung der übergebenen Baugruppe bestimmt wie das Eingangsmodell zerlegt wird, indem entweder einzelne Teile oder ganze Gruppen von Teilen in einem Explosionsschritt entfernt werden. Zweitens werden Beziehungen zwischen den explodierten Teilen hergestellt, damit Bewegungen zwischen zusammenhängenden Teilen weitergegeben werden. Drittens müssen Entfernungsrichtungen und Distanzen berechnet werden. Ein vierter Parameter sammelt bestimmte Teile in einer gemeinsamen Schicht, sodass diese gleichzeitig explodiert werden können. Durch Anpassung dieser Parameter werden in dieser Arbeit symmetrische Explosionsdiagramme und Diagramme, die ein Fokusobjekt in den Vordergrund stellen, erzeugt. Die Beziehungen zwischen den Teilen werden durch einfache Verbindungslinien oder durch statische Unschärfen, bei denen ein Objekt in den Ursprung relativ zu einem Elternteil gestreckt und verschmiert wird, visualisiert. Um den Explosionen und somit den Änderungen des Layouts folgen zu können, werden verschiedene kraftbasierte Animationsstrategien verwendet. Teile können nacheinander als Einzelteile oder in Gruppen entfernt werden oder in Schichten von Teilen, die dieselben Eigenschaften aufweisen. Man kann auch alle Teile auf einen Schlag explodieren.

Schlagwörter. Explosionsdiagramme, Fertigungssequenzen, Layout, Visualisierungsmethoden

Contents

List of Figures

List of Tables

Chapter 1

Introduction

An explosion diagram disassembles a model consisting of several parts and explodes the objects along their insertion direction by a certain distance. The impact of an explosion diagram depends on the final arrangement of all parts, as well as the visualization of the part movements and relationships. Both support the user in reconstructing the model in its assembled state. These ideas will be explained in the following, using a bad and a good example of an explosion diagram, created by the implemented framework.

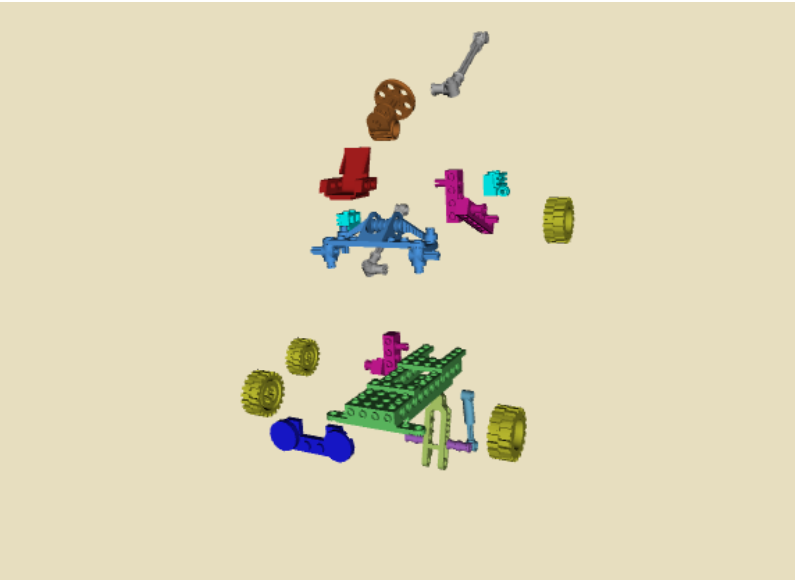


Figure 1.1: The parts of the assembly are removed one by one. The green base plate is kept static, the assembly is exploded by randomly removing parts or groups of parts. The hierarchy is not structured symmetrically and not even visualized.

Figure 1.1 depicts a bad example of an explosion. To create this diagram, the green base plate was kept static and random partitions were exploded away from it. No hierarchy of parts is created. For instance, symmetric parts like the gray axles are exploded in completely different ways. Furthermore, the relations among the parts are not visualized. Hence, the explosion diagram does not provide any hints to the user, how the original assembly can be reconstructed.

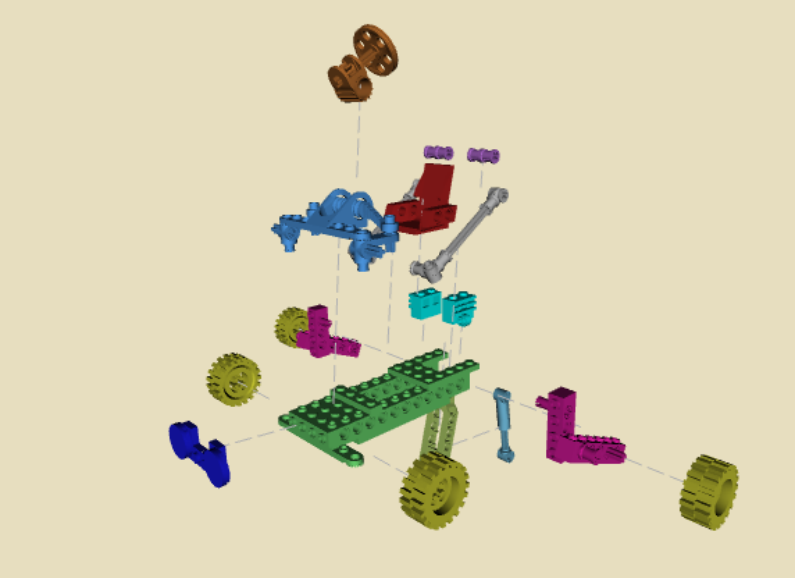


Figure 1.2: The explosion diagram is symmetric. Groups of parts having the same size are removed in a row and assigned to parents of the same symmetrical group. Hierarchy is visualized by guide lines.

On the other hand, figure 1.2 shows a full symmetric explosion of the same assembly. This arrangement was created according to symmetrical considerations outlined by Agrawala et al. in [2]. A hierarchy of part removal was employed, removing smaller parts before larger ones. This follows the reasoning that small parts tend to attach larger ones to the assembly. Furthermore, smaller parts are often perceived as less significant than larger ones. Hence, small objects are associated with larger ones, indicated by stippled guide lines. Creating such a hierarchy also prevents part collisions, which were a problem in the previous example. Now, the user is able to mentally reconstruct the assembly by moving the exploded parts into their initial position relative to a parent part. This way, the hierarchy is used to reassemble the model.

The thesis presents a framework, which is able to automatically create explosion diagrams for connected assemblies. Connectedness is an important criterion to reduce the

computational complexity for calculating an explosion diagram. Each part of the virtual model has to be in contact with at least one other part. The final layout of the explosion can be influenced by combining different types of styles, thus building a hierarchical layout. Additional information is provided by animating the part movements. Hence, the user is able to follow changes of the layout. Furthermore, the user is supported in mentally reconstructing the original assembly by visualizing the initial position of an object, relative to a parent part. For this purpose, two techniques were utilized, which are simple guide lines and static blurs. A static blur is a modified motion blur, where the blurring trail reaching to the initial position of a part is always visible. The perception of the different layouts and visualization methods is evaluated in a user study.

1.1 Explosion Framework

The framework implemented in this thesis is able to automatically create explosion diagram for mechanical assemblies. The arrangement of the objects is influenced by three parameters, each able to introduce hierarchical structures into the layout. These are the partitioning of the assembly into separable subassemblies and parts, the definition of parent-child (`PC`) relations among the parts and the selection of valid separation directions and distances respectively. Each of these parameters are explained further in the following.

The parameters can be used to introduce a hierarchy of parts into the arrangement of exploded objects. The partitioning of the whole assembly can be influenced by a classification of the single assembly parts. In general, only size and location of objects can be used to classify components according to certain criteria, because pure geometrical data is processed. The main criterion employed in this thesis is symmetry. For instance, a car assembly is symmetric along its length axis, thus having the same number of wheels on both sides. Functional groupings of objects into subassemblies cannot be performed automatically. For this purpose, additional information about the assembly would be required. A symmetric hierarchy can already be introduced when partitioning the assembly, by removing objects of similar size in a row, thus peeling the assembly from outside to inside.

The `PC` relations ensure that parts are moved relative to their parents, also preventing collisions with this parent part, when it is exploded. This thesis shows that the assignment of `PC` relations has a great impact on the arrangement of parts in an explosion diagram. Such relations can be used to limit the extents of an explosion, and to create a hierarchical grouping, where smaller parts are always moved relative to larger parents.

The separation directions have to be chosen in a way that the removed parts do not interfere with any other assembled geometry. A hierarchy is created, when the distances of the objects depend on their symmetrical grouping. For instance, similar parts are moved a similar distance. Each of these parameters can be exchanged independently by the user, to create an explosion diagram appropriate to the current requirements. The output depending on the selected parameter can also be influenced by a focus element.

The implemented system consists of two parts. In a first step the necessary configuration information is calculated in a preprocessing step. A valid partitioning, respectively explosion sequence, has to be determined. For this purpose, the framework calculates the range of all possible sequences of part removals. This way, a whole search space is created. A sequence is valid, if the parts are removed in a certain ordering, ensuring that they do not collide with any of the still assembled components. It is obvious, that the explosion of an assembly is the inverse of an assembly operation. Hence, algorithms from the assembly planning domain were utilized to create the mentioned search space, as well as appropriate separation directions.

The second part of the framework, the actual engine, is responsible for creating an explosion layout according to the requirements of the user. Therefore, the previously mentioned parameters are processed, consisting of a valid assembly sequence representing the partitioning of the assembly, appropriate `PC` relations and valid separation directions. An assembly can either be exploded fully, in layers or as single parts. Layers are for instance groups of symmetrical parts, or the least number of part removals to reveal a focus element. The ordering of single part removal is influenced by the chosen assembly sequence. The parameters influencing the final arrangement of the parts can be influenced by the user by combining so called explosion styles.

1.1.1 Configuration

Configuration is a preprocessing step and consists of several tasks. First of all, the geometrical information of the assembly has to be specified. The input assembly has to be separable into rigid, removable parts. This means, that at some point in the sequence a part should be removable as single object. Hence, interlocking parts must not be present. Nevertheless, the system is able to handle such cases. If parts interlock or are blocked otherwise, the least blocked direction is chosen, i.e. the one yielding the smallest collision area. Furthermore, the coordinates of each part need to be transformed into their final location in the assembly coordinate frame.

The assembly sequencing algorithm utilizes an undirected contact graph and global blocking constraints. The graph is calculated by determining part contacts. Global blocking constraints are computed pair wise and consist of valid separation directions for each pair of parts. A part is globally free to move into a certain direction relative to another object, if it can be translated to infinity collision-free. If a subassembly cannot be separated using global constraints, the test is repeated using a simple form of local blocking constraints. Instead of considering all valid direction pairs of a partitioning into two subassemblies, only the part pairs in contact are tested. This thesis limits the number of potential separation directions to the six main frame axis. If none of these directions is valid, the least blocking direction is chosen.

Utilizing the previously defined undirected contact graph and valid pairwise removal directions, the range of valid assembly sequences can be calculated. Using a depth first search (DFS) of the contact graph all partitions of the assembly are calculated. The valid removal directions are employed to test the separability of the partitions. Each partitioning passing this test is added to an AND/OR graph data structure, representing the explosion sequences of an assembly. To limit the computational complexity, only connected partitions are considered, as well as partitionings into at most two partitions. All created partitions are processed using this method, until only single part partitions are left.

After finishing the partitioning of the assembly, all potential `PC` relations are determined. These relations indicate which parts move relative to other parts. Having defined, which partitions are actually moved, the range of valid separation directions for each partitioning are determined. These three parameters, AND/OR graph, `PC` relations and separation directions span the search space for a single explosion sequence used to create the explosion diagram.

1.1.2 Engine

The engine is responsible for processing the user input, thus creating the desired explosion diagram. The user is able to influence the appearance of the diagram by specifying a focus element and combining different types of styles. A style is responsible for changing one of the previously mentioned parameters. Hence, there exist AND/OR graph, `PC` relation and separation direction styles. A fourth type is required to influence the ordering of part removal, because the AND/OR graph used to store the selected explosion sequence allows the parallel explosion of available partitions.

When performing an explosion, the exploded parts are not tested for collisions. Thus, the explosion sequence ensures only that the part removal is not blocked by the rest of the assembled partition. Nevertheless, this thesis developed a method to prevent collisions with exploding parts moving into the same as previously moved components. Therefore, a simple algorithm adapts the relevant `PC` relations.

The part arrangement itself is performed using a force-based approach. A spring-damper system was implemented which provides smooth animations when objects are exploded, or moved relative to a parent part. As mentioned earlier, the relationships among the exploded parts and partitions are visualized by either using simple guide lines or static blurs.

1.2 Structure of Thesis

This thesis has the following structure. In chapter 2 previous work dealing with the creation of different forms of explosion diagrams is outlined. Furthermore, a collection of assembly sequencing algorithms is described. The challenges of automatically generating explosion diagrams, as well as the general framework created in this thesis are presented in chapter 3. The actual implementation details including the utilized software libraries are specified in chapter 4. Chapter 5 discusses explosion layouts created by the different combinations of styles, developed in this thesis. The last chapter 6 summarizes the results of this thesis and provides an outlook on future work.

Chapter 2

Related Work

This thesis presents a framework for automatically creating explosion diagrams, which are used to reveal a focus element by still providing the full contextual information. Before being able to remove any parts and partitions, removable parts and valid separation directions have to be identified. This further involves the determination of a whole sequence of part removals. For instance, it is not allowed to explode an object, which is completely hidden by other parts, without first removing the occluding objects. The contextual information of the moved component would not be present, if it is simply moved out of the assembly. Hence, the user has no clue where to put the object in a mechanical assembly. Therefore, an assembly should be peeled layer by layer, removing either single parts or whole partitions, starting with he unblocked outermost parts and moving inwards.

Determining such a sequence is a task out of the assembly planning domain, which deals with algorithms and methods to automatically assemble or disassembly a product in a production environment. It is common practice to follow the approach of assembly-by-disassembly, which means that an assembly sequence is calculated by removing parts or whole subassemblies from the initially assembled model. This way, the search space of valid sequences is reduced, and thus also the computational complexity. This thesis utilizes techniques out of the assembly planning domain to create valid explosion sequences, which are the inverse of assembly sequences.

The chapter is split into two parts. Section 2.1 outlines a range of different approaches for creating explosion diagrams or representations similar to such diagrams. In section 2.2 an overview on the research related to assembly sequencing is provided.

2.1 Explosion Diagrams

Explosion diagrams can be found in different books, visualizing amongst others medical data or mechanical assemblies. In a typical diagram, the parts of an assembly or an anatomical model are arranged around an object of interest. This way additional contextual information is provided. For instance, in an anatomical explosion diagram a bone is exposed by removing the surrounding tissue consisting of muscles and fibers. This context providing objects are arranged around the focus element in a way that the user is able to reconstruct the initially unexploded state. Hence, a muscle situated on top of the bone should also be exploded upwards, not sideways or downwards through the bone. When exploding a mechanical assembly, directions are even more important, because they indicate valid insertion directions.

Explosion diagrams can be enhanced by using methods to visualize the relationships among the parts. This helps the user to understand the explosion hierarchy, prescribing which parts have to be removed before other parts. Often additional methods are utilized to visualize the different materials of the exploded model. Furthermore, labels containing additional information, can be provided. Such visualization aids are not part of this thesis. The work concentrates on automatically finding appealing arrangements of exploded parts, which may be optimized to reveal a focus element. Simple visualization techniques are employed, indicating the part hierarchy used to create the explosion.

Several systems supporting the user in the creation of explosion diagrams have been developed and interactive explosion diagrams have been used in several focus and context applications. This chapter presents an overview on previous work utilizing such diagrams to perform exploratory tasks in section 2.1.1 and for creating illustrations in section 2.1.3. Explosion diagrams are also used in the assembly planning domain to support the design of new mechanical assemblies. Such approaches are described in section 2.1.2. The last section 2.1.4 presents zooming techniques, able to create explosion diagrams, or similar representations.

2.1.1 Interactive Exploration

Preim, Ritter et al. [29, 32] present a 3D puzzle, which helps the user learning spatial relations of a model. First the object is segmented into parts, which are connected by manually defined docking points. After this preparation the puzzle is solved by bringing all objects together. To get a better insight into the fully or partially assembled model, an explosion diagram can be generated by scaling down all parts in their original position.

In this presentation, the docking points are connected by a line as shown in figure 2.1. Although they work on medical data, their approach can also be applied on mechanical assemblies.

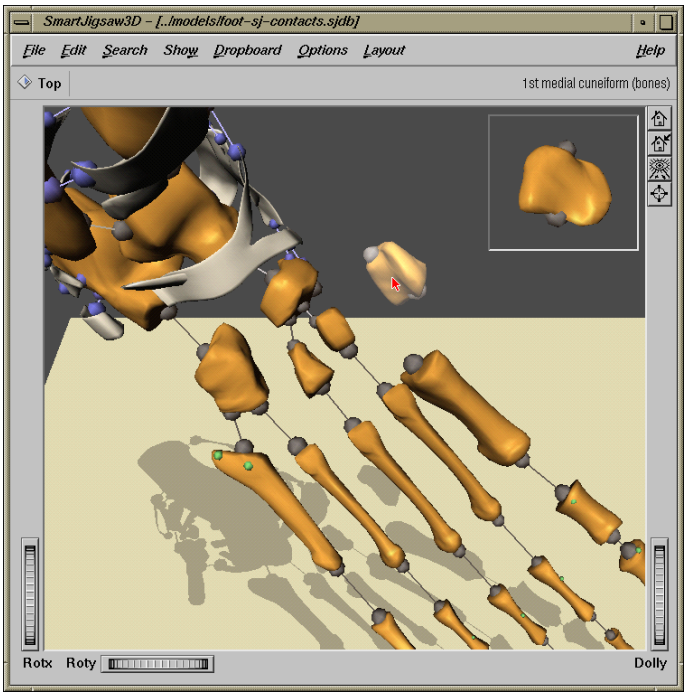
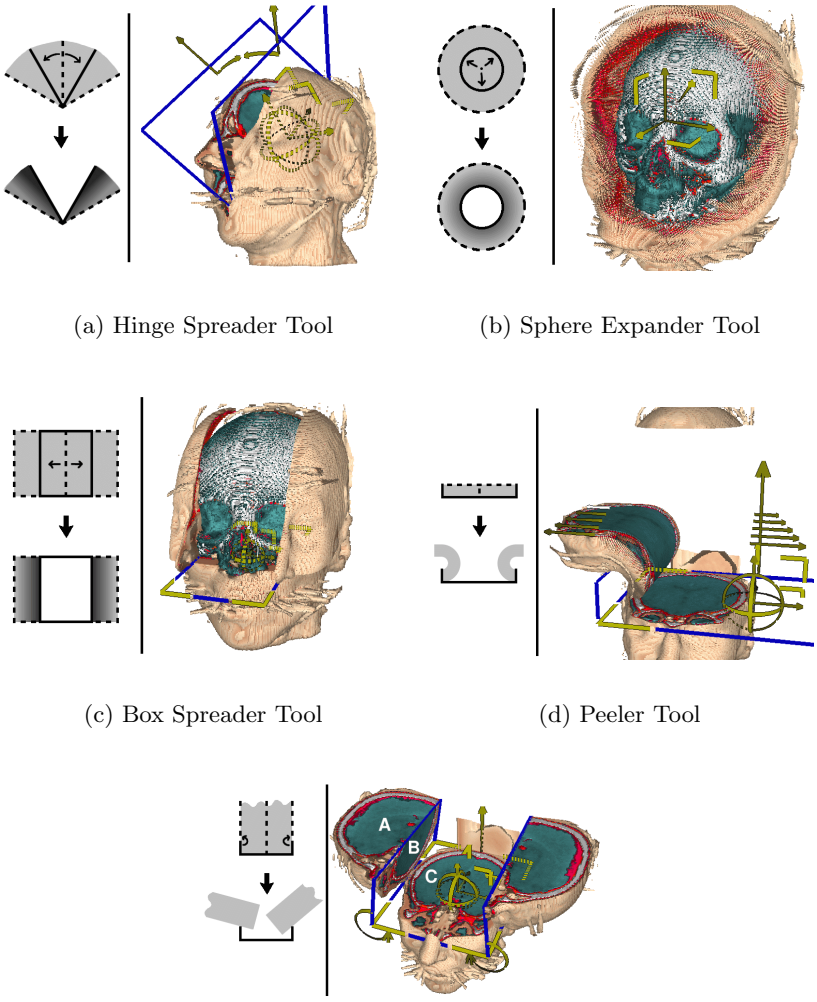


Figure 2.1: Explosion diagram of the 3D puzzle showing leader lines between manually defined connection points. Shadows improve the depth perception (adopted from [32]).

McGuffin et al. [23] present a prototype for interactively exploring volume data using deformation tools. This approach was motivated by the drawback of loosing contextual information, when insight is gained by modifying the transparency of objects or by creating cutaway views. Instead of removing data, manually defined layers of voxels are translated and deformed in their context. For this purpose different tools are used, some of which produce results, similar to explosion diagrams.

Each tool may act on one single layer or several layers at once. In the following, short descriptions of the incorporated tools are given. A hinge spread tool opens layers like a book, compressing the affected voxels. A sphere expander drives voxels out of the center of the sphere into its bordering region, thus producing holes. Analogously a box spreader shifts data to the sides of a box. The leafer has the form of a tray and hinges open the selected layers into two parts, rotating around the opposing edges of a tray. Additionally, after hinging open a set of layers, they can be fanned out. The peeler works similar to

the leafer, but peels the layers by applying a non-rigid, curvilinear transformation, instead of rigidly rotating them around fixed axis. Figure 2.2 shows how the tools are used to build up an exploded look of a human head. To reduce the prototyping time, each voxel is rendered as simple GL_POINT, thus producing only low-quality renderings.



(a) Hinge Spreader Tool

(b) Sphere Expander Tool

(c) Box Spreader Tool

(d) Peeler Tool

(e) Leafer Tool

Figure 2.2: Deformation tools as implemented by McGuffin et al. The gray images depict the form of the tool, as well as the deformation pattern. Darker shade means more compression of data (pictures adopted from [23]).

Sonnet et al. [37] utilize a visual distortion technique originally developed for browsing 3D graphs [5, 6]. A 3D probe having the form of a cube is presented, which enables the user

to move a viewing ray attached to it. All parts within a circular region around this line of sight are pushed out of the way. Smaller parts or parts contained in others are revealed by scaling the scope of the probe depending on the size of the affected objects. Thus, larger components are moved a greater distance than smaller ones. The translation distance of an object also depends on its initial distance to the ray and the drop off characteristics of the scope of the cube. By providing dynamically expanding annotations for the components, additional context information is added. Figure 2.3 depicts the application of the probe and expanding labels.

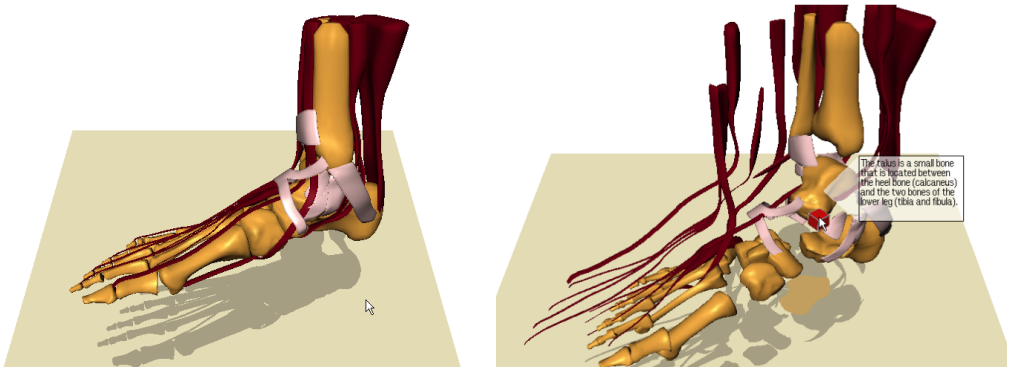


Figure 2.3: Explosion diagram created using the 3D probe, also showing expanding labels (adopted from [37]).

Bruckner et Al. [3] implement an engine to create explosion diagrams for volume data, splitting it into several pieces and arranging them automatically using a force-directed approach. In the beginning the user has to define the data he is interested in, which then stays static. The rest of the object comprises the moving geometry, which is further subdivided into convex regions of interest represented by bounding polygonal meshes. Hence the regions do not fit the real shape of the object.

Three splitting techniques are employed to interactively define the subdivision. An axis splitter divides the geometry according to a plane lying perpendicular to the projection plane and either normal to its vertical or horizontal axis. A split can also be performed by shooting a ray into the scene, which at some point may hit the part geometry. The model is then separated along a plane having the same same depth values as the intersected point. A third tool enables the user to print a line onto the screen, which is then projected into the scene. This projection results in a plane perpendicular to the projection plane, thereby dividing the geometry.

To create a force-directed layout return, explosion, viewing and spacing forces are

defined. Return forces try to move the part geometry back to its origin, while the explosion forces drive it away from the selection. The power of the explosion is influenced by a degree-of-explosion, which is zero in the unexploded state. To prevent clustering of parts a spacing force pushing part geometry away from each other is applied. A fourth force, the viewing force is responsible for providing an unobstructed view on the focus object by pushing parts in the line of sight out of the way as described in [5, 6] presented in subsection 2.1.4. The arrangement can be restricted using different types of constraints, limiting movement to certain directions or rotational motions as shown in 2.4.

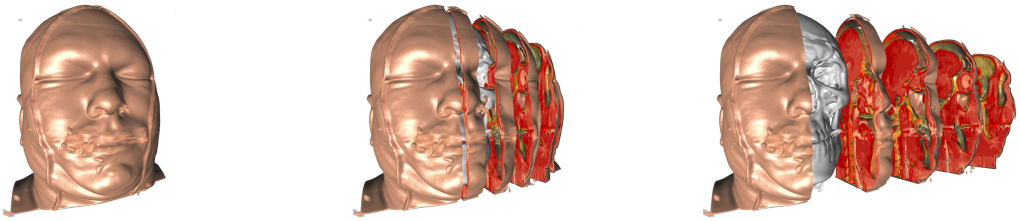


Figure 2.4: A series of pictures depicting an explosion diagram created by splitting a volume data set. The arrangement is achieved by restricting the movement to one axis and assigning different weights to the parts (adopted from [3]).

2.1.2 Assembly Presentation

Finding valid assembly sequences is one of the main problems in the assembly planning domain. Planning also includes the consideration of the environment and physical constraints, e.g. the center of weight of assembled parts. Since the creation of explosion diagrams requires the inverse operation of an assembly task, it suggests to itself taking a closer look at algorithms determining this information automatically. The results of this research are presented in 2.2. This section introduces developments out of this area, used for presenting assembly data, rather than calculating orderings of assembly operations.

Mohammad and Kroll [24] present a method for determining such relationships, rather than describing an engine for producing the explosion. Nevertheless, their work fits more likely into the context of explosion diagram, than into assembly sequencing as outlined in section 2.2. They developed a method for the "Automatic generation of Exploded Views by Graph Transformation", thus producing explosion diagrams of assemblies along the three major axes. Based on a high-level representation scheme for assembly data, a directed graph is created for each potential explosion direction. Such graphs contain blocking constraints defined by contacting features of the parts. The features, as well

as the relationships between the parts are described in a high-level data structure. All graphs are transformed into directed explosion graphs, resolving conflicting situations like multiple parents for a child. Because a high-level data structure is needed, this approach will not be discussed further.

Explosion diagrams may be used to support the design of assemblies. A system pursuing this goal is presented in [8]. Using this software the user has the choice to either build an assembly using a top-down or a bottom-up approach. The assembly is represented in a tree structure, where the leafs contain the single parts, which can be inserted into subassemblies, being the nodes. Geometry as well as features, resembling joining information are defined for each single part. A feature provides information on which other attachment types it can mate with, and further its position in the part it is associated with. The defined components have to be brought together manually to form subassemblies and the final assembly. Although no automatic support is given for the assembly task, the user receives feedback if parts do not fit together, hence enabling him to catch design errors. In addition the degrees-of-freedom of installed parts can be checked.

After creating the assembly, an explosion diagram can be generated automatically, by processing the manually defined information about removal directions and assembly sequences. The minimum separation distance is calculated by first determining the bounding box of the moved part and the one of the subassembly it is removed from, and then translating the exploded part until the bounding boxes do not intersect anymore. For the final diagram a constant is added to this distance. To create the explosion diagram, the assembly is split into its subassemblies, which are then exploded recursively, until only single parts are left. The user can influence the appearance by deciding on which subassemblies are exploded, by changing the distance or enlarging small parts and subassemblies, or even making them invisible. The context of the parts is established by drawing guidelines between the matching features of the components and by annotating them with labels. Figure 2.5 shows the part of the software responsible for creating the explosion diagram.

2.1.3 Illustration Design

Research on the automatic generation of explosion diagrams has been done in the area of illustration design. The AWI (A Workbench for semi-automated Illustration Design) [31] helps the user creating aesthetical illustrations by applying certain criteria, like for example part visibility. It can be used in two ways, either creating an image automatically, which is then adapted by the user, or refining the one built by the user. Amongst illustration

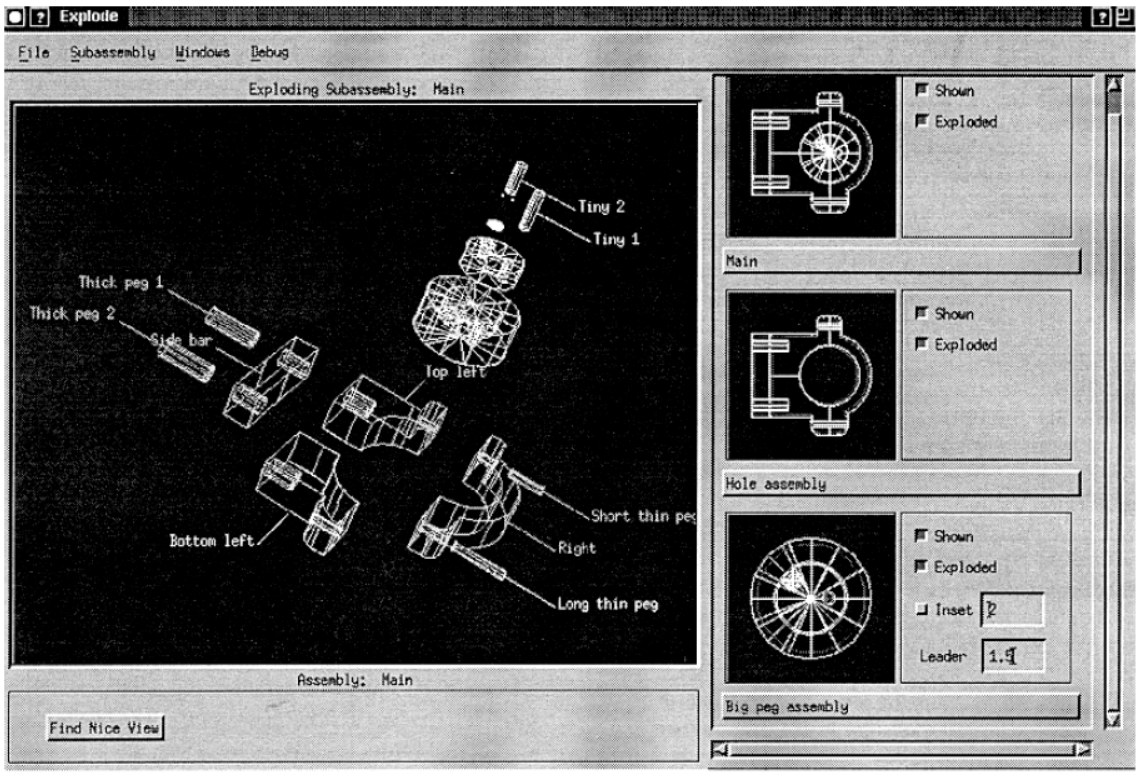


Figure 2.5: The part of a system supporting designers in the creation of assemblies, which is used to create explosion diagrams (adopted from [8]).

techniques like cutaways or geometrical abstraction, explosion diagrams are supported.

An explosion diagram is generated by selecting a group of objects to be exploded. The system is provided with a geometrical description of the assembly, hence no information about assembly sequences or blocking relationships are provided. Instead of calculating all relations, the user is urged to define which one of two connected parts is the base part, thus the parent part. Evaluating this information, AWI determines the invalid separation directions and an assembly hierarchy automatically. When exploding the assembly, this hierarchy is processed to identify an explosion sequence, explosion axes and separation distances. These are chosen in a way that the exploded parts do not intefere and certain criteria like part visibility are enforced. After the explosion is finished the user may move the parts along their explosion axes to improve the final arrangement. The source does not describe how the relevant data is calculated automatically.

While AWI [31] presented a semi-automated approach for creating explosion diagrams, which could be used to build assembly instructions, Agrawala et al. [2] demonstrate how

they can be created fully automatically from a geometrical description of an assembly. The output consists of a set of images depicting step-by-step installation instructions. Hence the diagram does not necessarily contain all parts, but only those relevant for assembling a product. During a preprocessing step contact information including the contacting faces between the parts, as well as directional blocking graph (DBG)s [40] are calculated. The DBG [34] is a technique out of the assembly planning domain and will be explained in more detail in section 2.2. A DBG is determined for a distinct direction and contains nodes representing the parts of the assembly. Directed arcs denote that the local movement of a component is blocked by the part of the target node when moving into this direction. The authors state, that they reduced the number of possible motions to the six major axis, which seem to be enough for most assemblies.

In the following planning step, out of the input consisting of the previously described data, as well as default part orientations and a default viewing direction, a valid assembly sequence is calculated. To receive better results grouping and ordering constraints may be defined. Grouping is important to let the algorithm decide on which parts are added next. Furthermore, it divides the set of parts into significant ones and fasteners. Additional grouping types are symmetry, stating that parts can be added at the same time, and similar-action, describing components which are attached using the same operation. The implemented sequencing algorithm selects parts from the set of unattached significant components and adds it to the assembly until all parts are processed. During the calculation, several criteria, such as part visibility and symmetry are evaluated. They incorporate design choices for creating efficient and understandable instructions. Each time a significant part is attached, the necessary fasteners are inserted too. The constraint similar-action reduces the number of illustrations, by removing those depicting the same operation for other parts in the group.

After determining an appropriate sequence, the user can select between a presentation as structural diagram or action diagram. In a structural diagram each step depicts the newly added components already in their final position, giving not clue about insertion directions. Action diagrams explode the parts to be attached along their assembly direction. Leader lines between the contacting faces of the involved assembly parts visualize these directions. Hence, an action diagram is similar to an explosion diagram. Stacks support the generation of such diagrams. All parts of a stack, but its base part, have the same separation direction and axis, and are in contact with each other. Using this data structure, the placement of parts is propagated to the ones associated with it.

Separation directions are determined by calculating the bounding box surrounding the parts neighboring the moving one, and the center of the bounding box of the contact faces of the moving part. Then the part is moved into the one major axis direction, into which it escapes the neighbor bounding box most quickly. Assembly instructions created with the presented software are shown in 2.6.

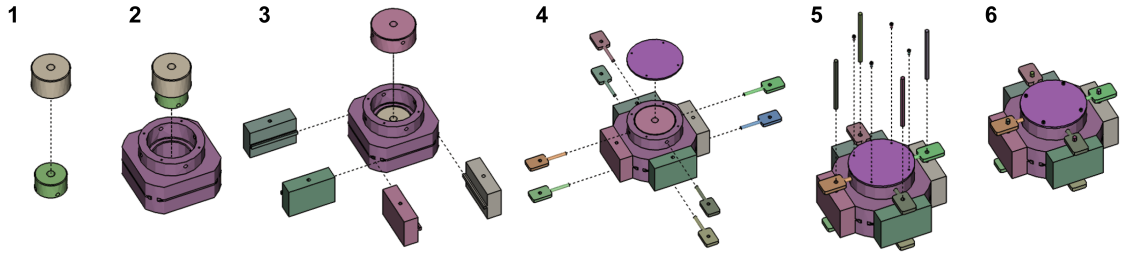


Figure 2.6: A set of assembly instruction depicting how to assemble an object. Parts defined as symmetric are attached in the same picture (adopted from [2]).

Another approach for creating illustrations of mechanical assemblies is presented in [22], where explosion diagrams are produced by processing 2D images of the parts of an assembly. Semi-automated tools are provided to define the relationships between the parts. Furthermore the tools are used to fragment images and to assign depth values, thus creating 2.5D representations of the input pictures. Depth information has to be added so that parts fitting into each other are occluded correctly. Explosion sequences and relationships among the parts are established manually by drawing an explosion axis through a group of parts, which then form a stack and thus a removal ordering. Each stack consists of a root part, which can also be root to other stacks, and a collection of components on top it. Components can only be assigned to a single stack. The used data structure is borrowed from [2] presented earlier. Important data like initial unexploded and final exploded position can be changed by the user. Additional information can be assigned by creating annotations for single parts, which are only visible if the manually defined anchoring point of the annotation is not occluded. After the user has chosen an anchor point relative to the commented part, the assigned label moves statically with the part. To further improve the contextual information, leader lines can be drawn manually between the parts. The process of creating the explosion diagram is depicted in figure 2.7.

The final illustration can be changed interactively by either fully exploding or collapsing the assembly, or searching for a certain part, which is then selectively exploded with its

surroundings to provide contextual information. Users are also able to manipulate the parts directly by dragging them along their explosion axis. If the maximum explosion distance is reached, the previous components in the same stack are expanded. Parts are immovable, until all of its offsprings are either fully exploded or in their original position.

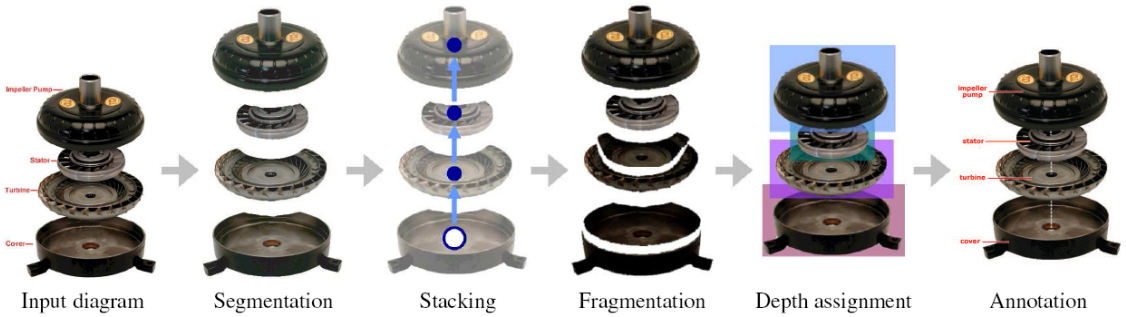


Figure 2.7: A series of pictures depicting the process of creating a 2.5D explosion diagram (adopted from [22]).

In a very recent paper [21], the previously described interaction techniques of the 2.5D image approach [22] for creating illustrations and the automatic creation of assembly sequences [2] were joined and improved to automatically generate interactive explosion diagrams for 3D data. The presented software also supports the splitting of containers surrounding components of interest, but this is out of the scope of this thesis and will not be described in more detail. Input to the system consists of a 3D geometrical description of the single parts. Furthermore, optional subassemblies may be defined manually. By default, the part movement is constraint to the coordinate axes of the entire model, hence to six directions, but additional ones may be specified. Before an explosion diagram can be created, the directional blocking relationships of parts in contact have to be determined and stored in an `DBG` as described in [2].

The algorithm for calculating an explosion ordering is similar to the one of Agrawala et al. [2]. After calculating the blocking relationships, an iterative algorithm loops through the set of unblocked parts and adds the one, which can escape the bounding box of the rest of the assembly most quickly to a directed acyclic explosion graph. Each node represents a single part or a subassembly. Arcs denote, that a part may only be moved if all descendants have been expanded. Subassemblies have their own explosion graph and are treated as a single part in the graph they are contained in. Special care has to be taken, when subassemblies interlock with the assembly they are removed from. If such a case occurs

the largest possible partition is moved. When all parts have been processed the algorithm terminates. Separation directions and distances of the components are also stored in this graph, as well as the original position relative to the parent. A parent is the largest part, which is blocked by the exploding part. Figure 2.8 shows an illustration created by this system.

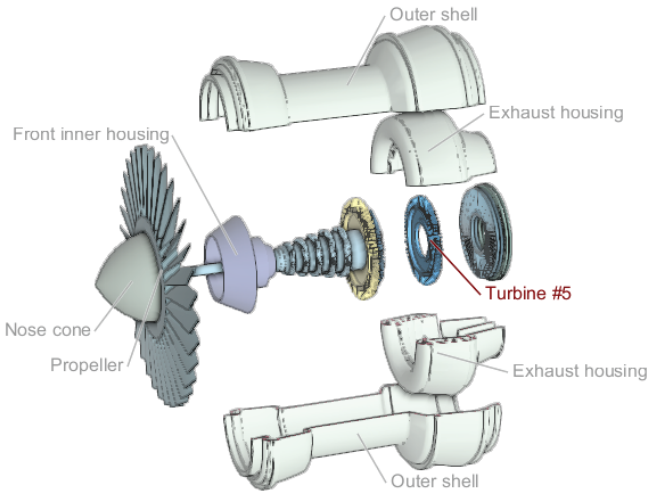


Figure 2.8: An automatically generated explosion diagram by Li et al. (adopted from [21]).

Several different viewing tools are implemented. A fully exploded view is generated by translating the parts along their separation direction, starting with the outer-most parts. Furthermore, it is possible to drag components directly along their explosion axis, propagating the movement through to its ancestors. The approach of [22] was improved by incorporating blocking relationships, making it impossible to move parts, until the blocking parts have been expanded. Another technique is riffling, which acts similar to a 3D fisheye zoom by exploding parts away from the one under the mouse cursor. Expanded parts collapse back into place when the mouse does not hover over the focus element anymore. The created view can also be locked by setting the focus to one or more parts. Compared to a fisheye view, the illustration is not distorted and the movement is restricted to the explosion axes. Users are also able to search for certain parts. After selecting target parts, the explosion distances are adapted by evaluating a set of criteria, which ensure that the target parts are not hidden by others and are expanded to set them apart from the rest of the assembly.

Approaches presented earlier in this chapter dealt with the creation of explosion di-

agrams for technical illustrations of assemblies or for exploring data sets. But explosion diagrams are also very informative in the architectural domain. Niederauer et al. [28] present a technique for automatically creating such diagrams from 3D data of buildings. Because of the appearance of a typical construction, it is not necessary to find complex separation directions and sequences. The problem is reduced to dividing the building into its stories, which are created by splitting the model a small distance beneath the ceilings. Potential ceiling polygons are those having a normal direction opposite to a defined up-vector. Summing up the areas of polygons having at the same level result in a table, which is then used to determine where to split the model. While the splits are performed automatically at the heights with the largest area, the user still has do define the number of stories. To compensate small distances between the stories, a minimum offset between adjacent ceilings can be specified. An exploded look is then achieved by translating the parts of the model along the up-vector by a user defined value.

2.1.4 Zooming Techniques

Zooming techniques presented here, also create noteworthy exploded representations. Like explosion diagrams, zooming may reveal internals otherwise not visible by still preserving the context.

Carpendale et al. [5, 6] tried to map 2D distortion and scale techniques for browsing 2D graphs into the third dimension. Evaluating the results they concluded that direct application does not produce satisfying results. Generally, the created 3D graph distortions did not reveal the focus object if it was occluded by other nodes. Therefore, a visual access distortion technique was developed, which explodes nodes out of the line of sight pointing from the viewpoint to the focus objects as shown in figure 2.9. Providing a free view onto parts of interest is an important criterion when creating explosion diagrams. Bruckner et al. [3] applied this approach to volume data sets, Sonnet et al. [37] a modified version to 3D bounding meshes.

A 3D zooming technique able to produce an explosion diagram is presented by Raab [30]. It is based on the division of the space into intervals bordering the objects. Changes in the size of an interval are propagated through neighboring ones and result in a zoom action, enlarging a focused part and scaling down others. Limiting the scaling operations to changing the interval sizes instead of the geometry itself, leads to an arrangement where objects are separated from each other, hence an rudimentary explosion diagram. Because it may not be intended to split apart a whole model, when the user is only interested

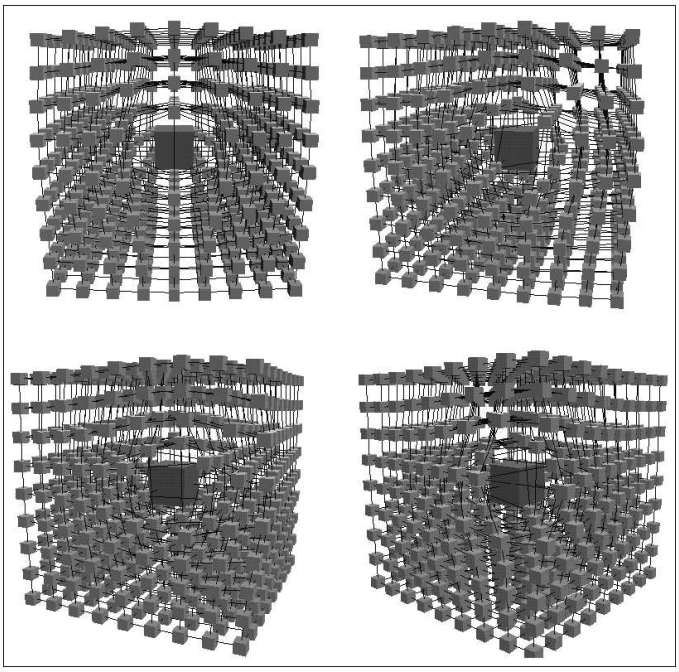


Figure 2.9: The visual access distortion applied on a 3D graph. (adopted from [5]).

in a single part, simple rules are defined, which prevent objects from being separated. Applying these constraints produces a zoom, where only the focused component is exposed. An interesting side effect is that these rules can also be used to improve the previously created diagram, lacking restrictions on how to separate parts from one another. Now directions are given implicitly by the information introduced by the rules. If two objects touch each other by having the same border, they are moved away from this border, if a part lies inside another part, the containing one is moved relative to the center of the contained one. To further improve the explosion diagram parts may be grouped to create explosion hierarchies.

2.1.5 Discussion

In the previous sections of this chapter a collection of approaches able to create explosion diagrams for different purposes and types of input data was presented. Although there exist CAD applications able to create explosion diagrams, they will not be described in this thesis. Such software often builds on information only available because of a high-level description of an assembly, as used in [24] and [8] presented earlier. This work focuses on the automated generation of explosion diagrams with only geometrical information

available. Now follows a discussion of these techniques outlining interesting aspects, but also trying to pull them together with CAD data of mechanical assemblies. The structure of the following section is derived from the requirements of creating an appealing explosion diagram for assemblies.

One basic task is the determination of valid separation directions and the selection of an appropriate distance. But directions alone are not sufficient. An explosion sequence has to be found, which may also resemble one for assembling the model. Finding an explosion sequence is easier than determining assembly sequences, because it is not constraint by considerations like part accessibility or stability concerning a center of gravity. Furthermore an explosion hierarchy has to be defined, containing information about associated parts, so that they are able to move relatively to each other. Additional data about subassemblies may also be contained. An assembly hierarchy in general is much more complex consisting of data about functional and semantic groupings of parts as stated in [2]. Such knowledge is hard to retrieve automatically.

Another challenge is the arrangement of parts after executing the explosion. A layout should be chosen so that objects of interest are visible and do not interfere with already exploded parts. When moving components, different visualization methods can be used to enhance the user's ability to reconstruct the exploded assembly.

Cluttering of parts is prevented by implementing methods to reduce the number of necessary explosion operations. Hence, instead of exploding components, objects are deformed or cut open to reveal certain parts.

Following this train of thoughts leads to the individual discussion of explosion directions and distances, sequences and hierarchies, the arrangement of expanded parts, means for providing contextual information and a separate section about splitting and deformation techniques.

Separation Direction and Part Distance

Parallels can be drawn between the sphere expander tool of McGuffin et al. [23] and the 3D probe developed by Sonnet et al. [37]. The sphere applied to a volume data set pushes voxels out of its center towards the border, to increase the distance between the data points. This simulates a kind of transparency and reveals the selection. If used on 3D objects without settings static focus geometry, all parts would be cluttered in the outer regions of a sphere. The 3D probe also moves objects away from a center, but the translation distance is depending on the size of the object, their initial position relative

to the focus point and a radial drop off function. But the parts are not constrained to be removed along valid assembly directions, thus loosing contextual information.

In [2] and [21] an object is moved until it exits the bounding box of the parts it is in contact with. While this is an improvement over the 3D probe, it requires knowledge about contacts and the blocking relationships between the parts, defining which directions are blocked by other components. Both are calculated automatically using techniques out of the assembly planning domain. Furthermore, situations where the walls of the container are separate objects not in touch with the part to be expanded, result in a distance too small. Driskill et al. [8] move a component until it exits the bounding box of the rest of the assembly, thereby solving this problem. To achieve the exploded look a constant factor is added to the distance.

Raab [30] incorporates contact relationships between objects in adjacent intervals into a 3D zooming technique. These should not be broken when enlarging a focus. By inverting these restrictions, separation directions were generated, leading to an explosion diagram. Because objects are contained in disjoint regions, which are separated from others by boundaries, this approach may not be easily applied on objects like fasteners. In addition, there is no data on how exact the algorithm would identify valid insertion directions.

Bruckner et al. [3] reveal a static selection by segmenting the volume data set and translating the created parts. In general the distance depends on an user defined degree-of-explosion. Directions are given implicitly by calculating the explosion forces applied to each movable object. Its power and direction are determined automatically by taking into account the position of the focus voxels, and the nearest vertex of each hull containing the unselected data. Changing the physical weights of the different parts, changes the explosion distance. Applying a force-based approach on mechanical assemblies is promising and part of this thesis. As proposed, part movement can be constraint to precalculated separation distances and directions, which creates an explosion diagram depicting valid insertion directions. In addition unwanted torques are avoided. Nevertheless, before mapping this approach on CAD data, several problems have to be solved. To provide interactivity there must be the possibility to move parts separately. Furthermore, expanded objects have to retain their relative position to the part, they are associated with.

Sequences and Hierarchies

Most of the presented literature does not automatically determine valid assembly sequences, either because it is not necessary to create an explosion diagram using the de-

scribed technique ([29, 32], [23], [5, 6], [30], [28]), or a manual solution is implemented ([8], [22]).

The 3D probe [37] implicitly defines an ordering by affecting larger parts stronger than smaller ones. It is interesting to note, that this way some sort of size-based hierarchy is created.

Bruckner et al. [3] only allow two types of objects in their explosion diagram. A static selection and the possibly segmented rest, which is either fully expanded or collapsed. Hence, finding a sequence is not necessary. Nevertheless, a form of hierarchy can be introduced by assigning different physical weights to the moving parts. It can be enforced even further using movement constraints, which may also improve the context, if parts are not allowed to be translated away from the selection too far as shown in figure 2.4.

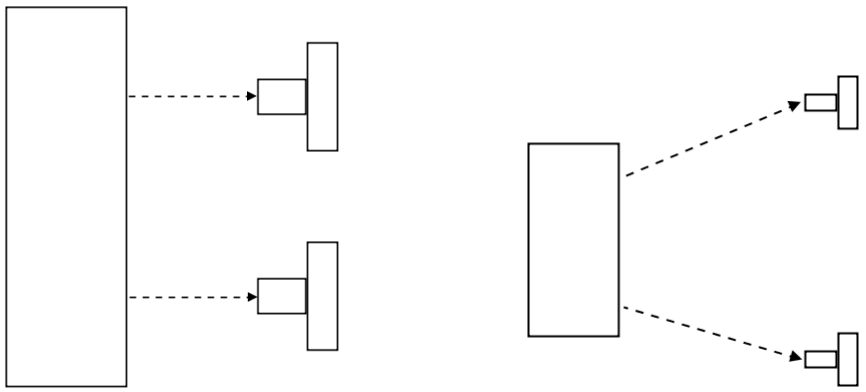
Agrawala et al. [2] determine the assembly sequence automatically. An interesting development is the manually defined grouping according to semantical and functional criteria. Automatically generated explosion diagrams are greatly enhanced, when employing such constraints, leading to a true mapping of the assembly structure onto the explosion hierarchy. In addition a stack data structure is introduced, providing the necessary hierarchy for moving expanded parts relatively to their base parts. In [22] the same structure is used, although the necessary sequences are defined manually.

Li et al. [21] merge a valid explosion sequence, separation directions and part hierarchies into an explosion graph, which is used to automatically create explosion diagrams. Such a data structure offers enough flexibility to be able to interactively browse through the parts of a model. By removing only one part at once, a natural ordering is generated where fasteners are removed before the object, they are attached to. Nevertheless, it contains only a single explosion sequence and the number of removed components to quickly reveal the one a user is interested in may not be optimal. This thesis proposes a data structure and a system allowing the user to influence the outcome of the sequence finding algorithm as well as the explosion hierarchy to create an application specific explosion diagram. A data structure is presented, which is able to store several explosion sequences at the same time.

Arrangement of Expanded Parts

Simply scaling down the objects in their position as done by the 3D puzzle [29, 32] quickly reaches its limits, when it comes to CAD data. Fasteners attaching objects to each other would not be positioned along the vector of their insertion direction (see figure 2.10).

Thus, important context information is lost. Some information can be retained by drawing leader lines between the insertion point and the removed parts. But for complex models this results in a great number of lines pointing into different directions, probably making it hard for the user to read the diagram. In addition it is not possible to partially expand a model, without changing the relative size of the parts.



(a) (b)

Figure 2.10: (a) A correct explosion diagram is shown and (b) a diagram where all parts are scaled down in their position . Leader lines do not point into a valid insertion direction.

The underlying physical framework used by Bruckner et al. [3] allows parts to be arranged automatically. In addition, smooth animations are provided, until all forces reach equilibrium. New forces leading to a different layout can be implemented rather quickly. However, forces like the described spacing force act between each object of the simulation and lead to a runtime of $O(n^2)$. Hence, if applied to a large number of objects, optimization strategies have to be employed. Using a spring may also pose a problem, if forces disappear instead of being reduced step-by-step. Objects would start to oscillate because of their inertia. Hence, a spring-damper system could be employed.

Force-directed layouts are also used by the 3D probe [37] and the visual access distortion [5, 6]. There, no physical framework is used, thus removing the need to compensate inertia.

Li et al. [21] influence the final position of focused parts by visibility considerations. To get a clearer view on target components, several visibility criteria are defined. Objects are rearranged until these constraints are fulfilled. Hence, not the sequence itself is affected, but its outcome. A hierarchical layout is achieved by making the parts biggest in size to

parent parts. This may pose a problem when screws fit smaller objects on top of bigger ones. When exploding the small part, the already removed screws associated with the bigger parent are static. Thus the small part moves through the screws. This thesis presents a simple algorithm, preventing such collisions by modifying the relationships among the parts.

The interval zooming technique [30] automatically arranges parts, but may violate the relative scaling of parts as well as the scale of the parts themselves.

None of the presented related work deals with the problem of parts interfering already exploded ones or at least their explosion vector. Such a situation can occur, if two movement directions are oriented normally to each other and intersecting. Although discussing this problem in 3, this thesis does not handle cases like this either.

Enhancement of Contextual Information

Contextual information can be added by drawing guidelines between associated parts as done by the 3D puzzle [29, 32]. One drawback of the presented approach is the manual specification of contacting faces, which are then connected. Furthermore, because objects are scaled by their technique of creating an explosion diagram, leader lines are not equal to separation directions. Li et al. [22] let the user manually assign anchor points which are updated as a part moves. Hence the created guidelines always map to the explosion directions. Agrawala et al. [2] present an automatic technique, where lines are drawn between real contacting faces of the base part and the expanding one. The development of this method was motivated by asymmetric parts, having only a small contacting area. Nevertheless, to be able to create ideal guidelines, the real attachment features of an assembly have to be identified.

When annotating parts with labels, one has to deal with the problem of arranging them in a way, they do not occlude important areas. Aside of arranging them manually ([22], [8]), algorithms have been developed performing this task automatically. Li et al. [21] incorporate such a technique. In conjunction with the 3D probe, Sonnet et al. [37] introduced dynamically expanding text labels. Depending on the position of the mouse, either part or all of the label is visible.

Navigating and orienting in explosion diagrams may be rather difficult. For instance, a user could have problems determining the distance between the single parts. Providing artificial depth cues greatly enhances a 3D representation. In [29, 32] parts are projected onto the ground plane to create flat shadows. In addition, the whole shadow volume can

be rendered, making it easier to associate them with the correct objects. Volume shadows are restricted to a selected component, thus preventing cluttering.

By employing movement constraints implemented in the used physical framework, Bruckner et al. [3] are able to restrict the movement of parts along certain directions and to certain joint types. Therefore, the arrangement can be influenced to provide more contextual information. The tools for browsing volume data sets [23] also create constraint visualizations.

Deformation and Cutting

McGuffin et al. [23] implemented different tools to gain insight into a volume data set by deformation. Most of the tools, except leafer and peeler, compress the rest of the data according to a certain pattern to reveal a selected object. Applied to CAD data there has to be decided if parts are split if requested by the tool or moved as a whole. Another important decision is, if parts stay rigid or may be compressed. Compression could lead to a loss of context when changing size relationships. Leafer and peeler may be applied to CAD data to open hulls without removing information as cutaways do. Assembly parts then may be exploded through the created hole. Creating cutaway views and expanding objects through the removed hull was utilized by Li et al.[21].

2.2 Automated Assembly Sequencing

Assembly sequencing is part of the assembly planning problem and describes the ordering of operations necessary to assemble an object, including assembly directions and a description of the mating subassemblies. In addition to finding such a sequence, assembly planning also incorporates considerations about the production environment, such as available workers and tools [40]. In the following, algorithms for determining simple valid assembly sequences are presented. Such a sequence consists of the selection of the parts to be mated and the corresponding motion along a single vector.

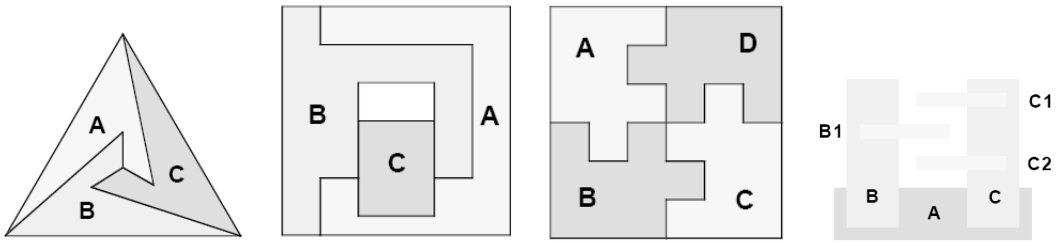
While there exists research also dealing with part tolerances [19], fixture design and assembly stability [33], or complex insertion directions [1, 13, 35] along multiple paths, it is not considered further, because of the following reasons. Using only 3D representations of assemblies to create an explosion diagram, there is no need to handle part tolerances. In addition, fixtures and stability concerns are not of interest when exploring a computer generated view of a mechanical assembly. Complex insertion directions may consist of

several translational motions, or a combined rotational and translational motion, typically applied when removing screws. As stated in [21], screws and other fasteners are often represented without thread geometry, thus removing the requirement to rotate the component. Nevertheless, the direction finding algorithm used in this thesis is able to deal with such cases. In general, determining an assembly direction consisting of several different movement vectors, is a very complex task and out of the scope of this thesis. Additionally, if multiple translations would be necessary, other visualization techniques than an explosion diagram can be chosen, such as cutaway and splitting techniques ([21]).

Before outlining different assembly sequencing algorithms in section 2.2.2, some terms are defined in section 2.2.1, thus allowing a classification of the methods. In section 2.2.3 follows a discussion of the outlined techniques.

2.2.1 Definition of Terms

Research on assembly sequence planning often restricts itself to certain problem cases, thus limiting the search space, which otherwise would grow too big. Amongst others, such constraints describe the number of part attachments performed in each step or whether assembled components are inserted into their final position or parked in an intermediate position. The following definitions used to characterize the application cases of a certain sequencing algorithm are borrowed from [40]. Furthermore, the term assembly-by-disassembly is declared, which is present throughout the assembly planning literature. It additionally reduces the computational effort of finding assembly sequences.



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

Figure 2.11: Examples of assemblies for which there does not exist a binary (a), monotone (b) or linear (c) assembly sequence. The assembly in (d) cannot be partitioned into connected subassemblies (pictures adopted from [41]).

Number of Hands

The number of hands defines how many parts are joined in a single assembly step, and refers to the number of robotic arms needed for executing the operation. This number should be as small as possible to reduce computational complexity of a sequencing algorithm and manufacturing costs. Merging only two parts in each assembly step, creates a binary or two-handed sequence. Most assemblies can be built using binary sequences, but there may be cases, where more hands are needed. Figure 2.11(a) depicts an assembly, which needs at least three robotic arms to be assembled. If such a situation comes up, the assembly is very likely to be redesigned [40].

Monotonicity

A plan is monotone, if all parts are inserted into their final relative positions in a single operation. Monotone sequences are preferred, because no intermediate positions have to be calculated, thus reducing the computational complexity. When considering monotonicity, the number of hands also has to be taken into account. For example, the assembly in figure 2.11(b) can be created by applying a binary non-monotone sequence placing part C at an intermediate position in B, before attaching A. Using three hands, the executed sequence is monotone.

Linearity

A linear sequence simply inserts a single component into the rest of the subassembly, thus relieving an algorithm from finding valid movable subassemblies. Figure 2.11(c) shows an example, which cannot be assembled by a binary, linear sequence. It is always necessary to move two subassemblies containing two components to create the final assembly.

Connectedness

An assembly is connected if all components are in contact with at least one other part of the same assembly. Allowing only connected subassemblies to be inserted into an assembly, greatly reduces the search space for identifying such partitions. If an assembly sequence fulfills this constraint, it is also called contact-coherent. Figure 2.11(d) illustrates an assembly, which cannot be built using a binary, contact-coherent sequence, where the insertion operations are restricted to single translations. Mating B and A before C, renders it impossible to insert C, and vice versa. The operation is only successful, when moving

B and C at the same time. But then an unconnected partition is formed.

Local Freedom

If a subassembly or part can be moved an arbitrarily small distance relative to the rest of the assembly, without colliding with another component, it is locally free.

Global Freedom

Global freedom extends the definition of local freedom to the case of infinite motion. A subassembly is globally free if it can be moved to infinity without interfering with other objects.

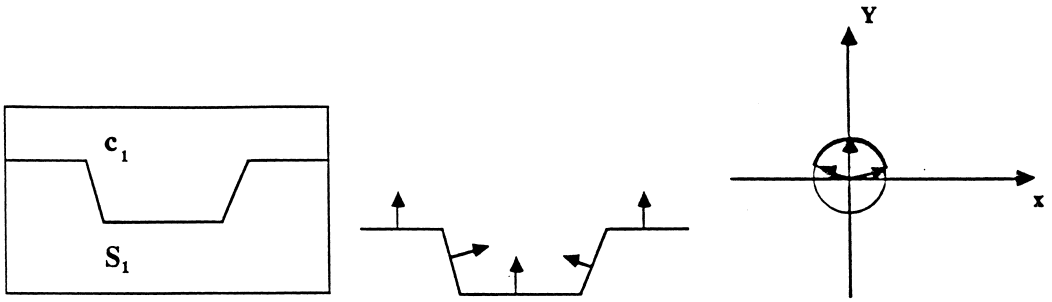
Assembly-by-Disassembly

Finding a valid assembly sequence based on the initially unassembled state of a product results in a great number of possible combinations of parts, many of which have to be discarded because created subassemblies cannot be joined. Hence, most algorithms perform an inverse search, starting from the completely assembled product and removing parts until it is completely disassembled. To be able to apply this approach to assembly planning, a disassembly operation must have an assembly equivalent [14].

2.2.2 Sequencing Algorithms

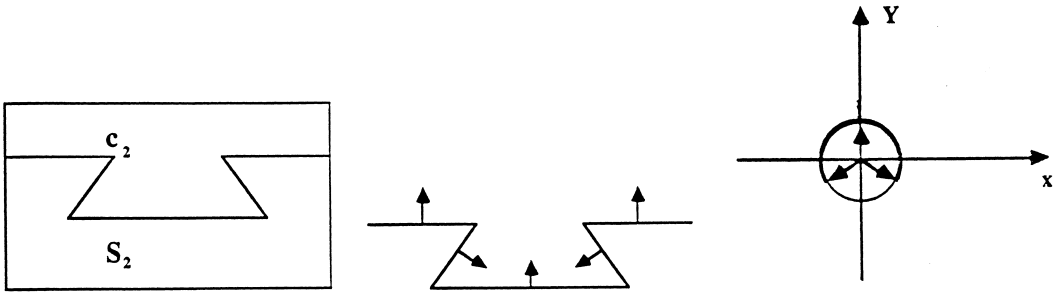
Because most of the presented algorithms follow the approach of assembly-by-disassembly, terms like disassemble and remove are used instead of assemble and insert.

Woo et al. [42] describe an algorithm able to determine a valid monotone, binary and linear assembly sequence of a totally ordered assembly from its geometrical description. Totally ordered means, that there has to be removed at most one part to be able to disassemble another component. Before determining the actual sequence, an undirected face adjacency graph (`FAG`) is created in a preprocessing step. In this graph, each face of a part is represented by a node and may either be a mating face between two components, or a boundary face. Arcs denote neighboring faces. Furthermore, a method to test separability has to be introduced. A part is disassemblable if all of its mating face normals lie in the same half-sphere, which is referred to as monotonicity. More formally, a set of points is created on a unit sphere, by adding the normalized normals to its origin. If the convex hull incorporating this points lies in the same hemisphere, the normals are monotone. Figures 2.12 and 2.13 illustrate this test for the 2D case.



(a) (b) (c)

Figure 2.12: The normals of the contacting faces lie in the same hemisphere, hence the parts are separable (pictures adopted from [42]).



(a) (b) (c)

Figure 2.13: The normals of the contacting faces lie in both hemispheres, hence the parts are not separable (pictures adopted from [42]).

The sequencing algorithm then checks all parts having boundary faces for separability. Each removable part is then added as a new leaf node to the root of an initially empty disassembly tree (DT) and their mating faces are changed to boundary faces. In the following, the neighbors of each leaf node are tested for removability and added to the corresponding node. If a neighbor was also added to one of the previous nodes, it is not considered any further. This is done until all parts have been processed. Traversing the final DT in-order produces a disassembly sequence, while post- and pre-order traversal lead to an assembly ordering.

Wolter [41] presents the XAP/1 assembly planning system, which determines a monotone, linear, but not contact coherent assembly sequence, by evaluating manually defined precedence constraints. Unlike other techniques, this one does not follow the approach of

assembly-by-disassembly. Input to the system consists of direction proposals, and directional blocking and precedence constraints, stating that a part can only be inserted into the assembly along a certain direction, if it precedes another component. XAP/1 does not calculate this information automatically. All possible assembly sequences are generated by evaluating these constraints.

To reduce the search space, hence the computational complexity, optimization is performed on-the-fly during the processing of the input constraints. The given optimization criteria are directionality, fixture complexity and manipulability. Directionality allows only a minimal number of assembly directions, thus decreasing the potential reorientations when assembling the model, fixture complexity optimizes the number of fixtures required to hold the assembly during production and manipulability selects those parts to be moved in an assembly step, which can be handled easier than the rest. Out of the list of currently unrefined assembly sequences, the optimizer always chooses the one yielding the best result when evaluating the previously described criteria. The selected unfinished sequence is then continued using two very different branchings, so that the difference between both evaluated optimization criteria is as big as possible. Hence the one with the worse optimization result, will very likely be not refined further, thus reducing the search space. This is done, until a sequence has been found.

The calculated ordering is represented in a tree, where leaf nodes resemble the single parts. Instead of considering general mating operations like other data structures, the tree stores insertion operations. Nodes correspond to subassemblies created according to an ordered sequence of insertions of its children. Because the subassemblies are held in place and other components are inserted into them, this data structure contains information about the required number of fixtures, and which parts are actually moving. Since only linear sequences are calculated, this tree does not contain subassemblies and has a depth of one.

Homem de Mello et al. [15] developed an algorithm yielding a binary, monotone and contact-coherent assembly sequence. It is calculated by splitting the assembly into all possible combinations of two partitions containing one or more parts. For each of the resulting partitionings a number of criteria are evaluated, e.g. if it is geometrically separable or not. If a partitioning passes this test, it is added to a data structure called AND/OR graph, which stores information about the possible assembly sequences. In the following, the used data structures and the algorithm are described in more detail.

The mechanical assembly is represented by a two-level data structure. In a higher

layer, the parts and their contact relationships to other components are described. Possible contacts types are plane-plane contacts, or cylinder-bolt and cylinder-hole contacts. In addition, an attachment may be associated with a contact, thus defining the type of connection. Attachment types could for example be pressure fits or screws. The lower layer consists of the geometrical description of the assembly. Figure 2.15 depicts the relational model of the assembly in figure 2.14.

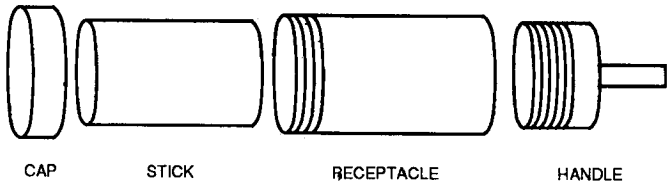


Figure 2.14: An assembly consisting of a stick, fitting into a receptacle. A cap and a handle are screwed onto the receptacle (picture adopted from [15]).

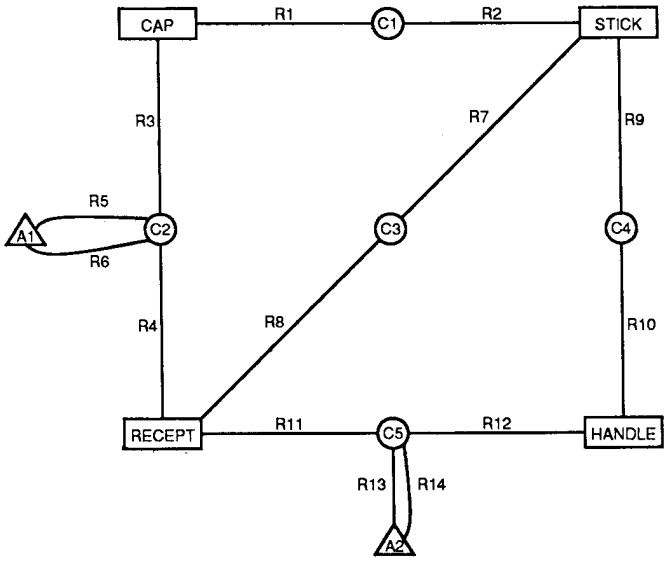


Figure 2.15: The relational model of an assembly, defining all parts, including contact relationships between the parts. Attachments assigned to contacts further refine the type of contact (picture adopted from [15]).

As stated earlier, assembly sequences are stored in an AND/OR graph, which can be understood better, when visualizing it as a tree, where the finally assembled product comprises the root. Each node contains a subassembly of the whole product. Splitting a subassembly into two partitions results in the same number of nodes connected to the one holding the original subassembly via a hyper-arc. Hence, an AND-branching is created.

If a node can be split into several combinations of two subassemblies, each partitioning is associated with the corresponding node. Nodes containing only a single part are the leafs, and cannot be split further. The graph does not only store a single assembly sequence, but all possible ones. It also enables a planner to identify assembly tasks, which can be processed in parallel.

To generate the AND/OR graph of a mechanical assembly, it is decomposed starting with the fully assembled state. Using the information contained in the high-level data structure, an undirected contact graph is built. Then all possible partitions of this graph, resulting in two connected sub-graphs, are calculated and tested for local freedom. If they are separable, the partitions are added to the AND/OR graph as nodes and are connected via a hyperarc. This is done recursively, until all subassemblies have been tested. The full AND/OR graph of the assembly shown in figure 2.14 is depicted in figure 2.16.

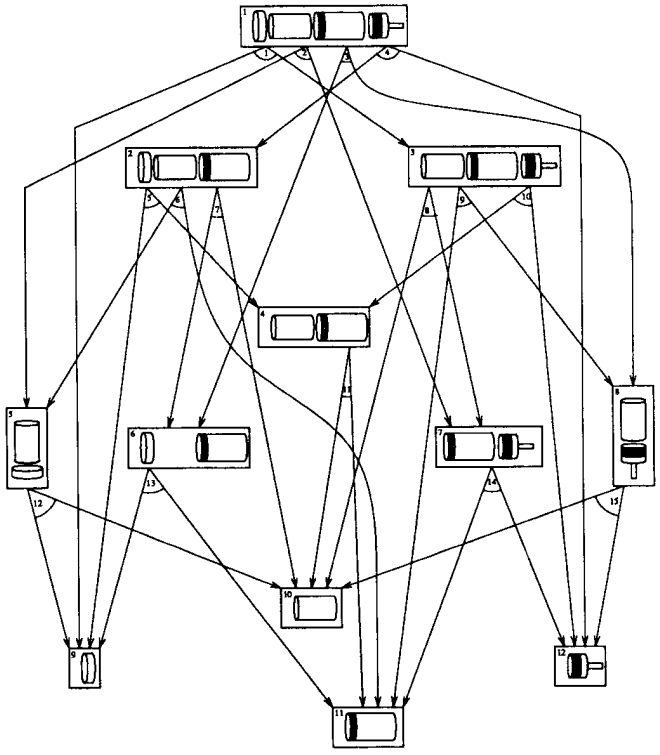


Figure 2.16: The complete AND/OR graph of the assembly of figure 2.14 (picture adopted from [15]).

To determine local freedom, the strongest contacts between the two subassemblies are considered. A cylinder-bolt and cylinder-hole type contact for example restricts the movement of a part to a single direction. If only plane-plane contacts are present, the test

is more complex and basically the same as incorporated by Woo et al. in [42]. Instead of testing for monotonicity, the solution to a series of inequalities of the form $n_c x \geq 0$ has to be found, where n_c is a normal of a plane contact. The solution x will be a range of directions which can be represented as a polyhedral convex cone as described in [9]. It is a very common approach to determine local freedom and is for example also used by Wilson [40] or Romney et al. [34]. The latter refer to this cone as local translational freedom (`LTF`) cone. Both will be outlined later on. The system of Agrawala et al. [2] used for automatically creating assembly instructions (see section 2.1.3) called this an local translational blocking cone.

Depending on the size of the processed assembly, an AND/OR graph may contain a great number of possible assembly sequences. Therefore, aside from local freedom, additional criteria can be evaluated in the decomposition step, reducing the search space for the final assembly sequence. An AND/OR graph can also be converted into a Petri Net (`PN`), assigning weights to transitions and nodes, to be able to apply efficient `PN` evaluation methods [4, 26, 27]. But optimization is out of the scope of this thesis, and will not be discussed further.

Wilson [39] presents a semi-automated technique for determining assembly sequences. Thereby the user is queried, if a situation cannot be resolved. The system calculates an AND/OR graph and evaluates local and global freedom, if a subassembly is to be partitioned. Local freedom is checked using the technique presented in [40], which will be described later on. To test if a partition is globally free, it is swept to infinity. If screw attachments are involved, the translation is combined with a rotation. But the partition is not rotated along the movement vector, but the rotational volume is created and moved to infinity. When the freedom tests fail, the user is queried. He then may state, that the subassemblies are movable or not movable. If something is declared to be not movable, a constraining set of parts can be specified, which prevent the parts from being separated. The system stores the answers and resolves future conflicts involving those parts automatically, thus reducing the number of user queries.

Mok et al. [25] create a monotone, binary and linear assembly sequence by parsing the model of a mechanical assembly stored in the CAD data format Standard for the Exchange of Product Model Data (`STEP`). `STEP` has a hierarchical structure, describing an object as a closed shell consisting of advanced faces in the highest level, and as basic primitives in the lowest. Basic primitives are for instance planes, circles or vertices. The complete hierarchy is illustrated in figure 2.17.

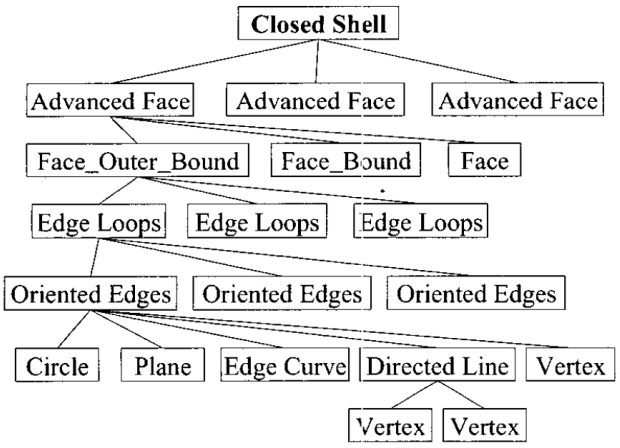


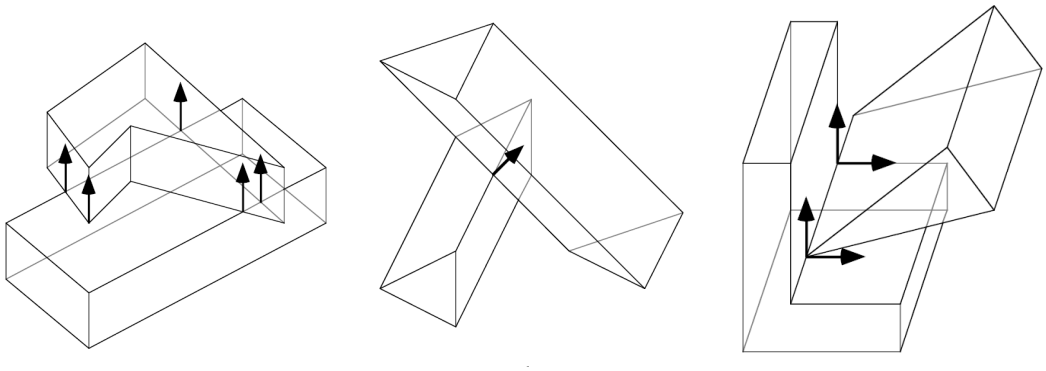
Figure 2.17: The hierarchy of a file stored in the STEP format (picture adopted from [25]).

A sequence is generated by first transforming all parts into the same coordinate frame and then calculating their bounding boxes. The one having the lowest z-value is chosen to be the root object. The following component is the one having the next smallest z-value. If the z-values of parts overlap a different criterion is applied. Out of the collection of objects overlapping with the root element, the one having the lowest maximum z-value is selected. This is done until all parts are processed.

The proposed technique is also able to identify features such as planes, holes, protrusions and grooves. This information, combined with the generated assembly sequence, is used to create a set of assembly commands.

In [40], Wilson presents a data structure, implicitly encoding the blocking relationships between the parts of an assembly for each possible removal direction, derived from a pure geometrical description of the product. Using this data structure, generated partitions are already separable and do not have to be tested any further. The presented technique is able to produce binary, monotone, and contact coherent assembly sequences.

Starting with a geometrical description of the product, different types of contacts between the parts are determined. For example, two contacting planes of an assembly can be reduced to the four points forming the convex hull of the faces in contact. These contacts are reduced to a sufficient number of point-plane contacts, equivalently describing the original contact. Figure 2.18 illustrates three examples of contacts between two parts, and how they are reduced to a number of point-plane contacts. The resulting point-plane constraints are then used to determine if a part is locally free in respect to other components. As done by Homem de Mello et al. [15] outlined earlier in this section, a



(a) face-face contact (b) non-aligned convex edge contact (c) edge-face contact

Figure 2.18: An example of contacts between parts, which are transformed to point-plane contacts indicated by the arrows, representing the normal of this constraint (picture adopted from [40]).

range of linear inequalities is solved, incorporating all outward facing normals of the static part. A solution to $n_c x \geq 0$ has to be found, where n_c is the normal of the constraint and the solution x comprises all valid directions for this constraint.

Wilson defines two data structures. A directed `DBG` storing blocking relationships between the parts of the assembly for a single direction and an non-directional blocking graph (`NDBG`) holding information to be able to create `DBG`s for each possible direction. In the `DBG` each part of the assembly is represented by a node. A directed arc from one node to the other represents a blocked motion into the direction, for which this graph was built. This graph is strongly connected, if there exist at least two nodes being blocked by each other, hence both have arcs ending in the other node. Subgraphs being strongly connected are referred to as strong components. Figure 2.19 shows the `DBG` for a crate covered by a lid and containing a box.

`DBG`s are constructed out of an `NDBG`. Considering the case of three dimensional motion directions, a normal of a single contact constraint defines a plane cutting a unit sphere S^2 in two halves. A point on the unit sphere represents a motion direction. If a direction lies on one of the two sides of the created circle, the movement is either blocked or not. A motion vector situated on the circle itself results in a sliding motion of the involved part on the otherwise blocking component. The set of constraint between all parts divides the sphere into cells surrounded by arcs, which are part of the created circles. Cells are also bordered by the intersections of these arcs, or circles respectively.

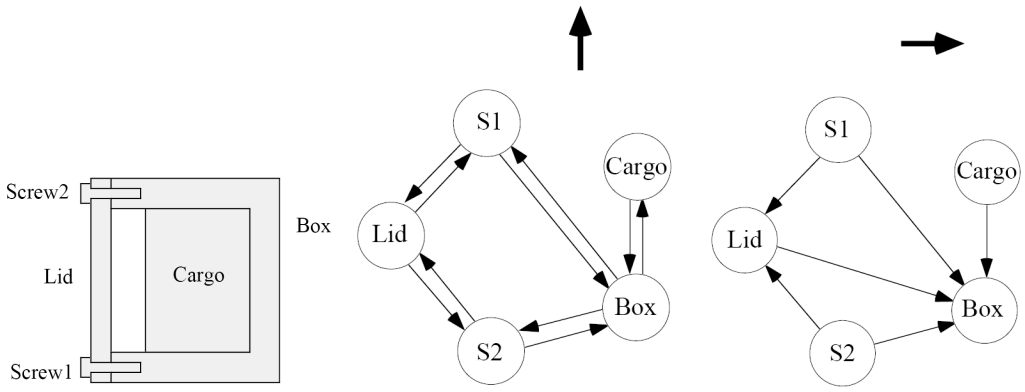


Figure 2.19: Two examples of DBGs for the assembly on the left. The arrows indicate the directions for which the graphs were created. For the up-motion the complete graph is strongly connected, hence the assembly cannot be partitioned for this direction (pictures adopted from [40]).

The blocking relationships in the interior of a cell, are the same, hence result in the same DBG, if calculated for one of the associated directions. Moving into a neighboring cell either breaks or creates a new blocking relationship and result in a new DBG. Hence, if a DBG has been calculated, the next one can be calculated incrementally by adapting the existing one. A direction situated on one of the arcs results in a sliding motion of the parts associated with this constraint and removes a blocking relationship from the DBGs of the adjacent cells, if it was present. A simple example of an NDBG and some of the calculated DBGs is illustrated in figure 2.20. It visualizes the intersection of the unit sphere S^2 by the planes defined by point-plane constraints.

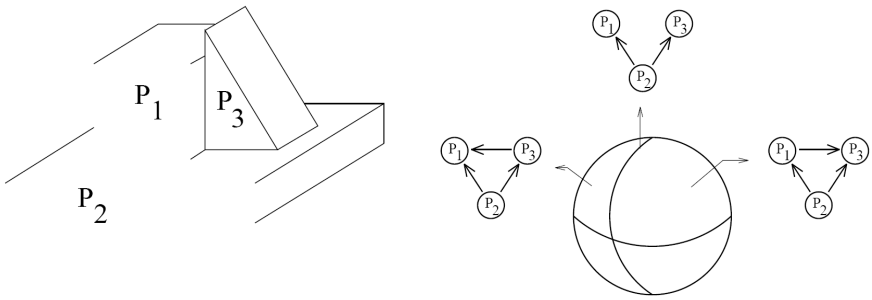


Figure 2.20: An example NDBG for a simple assembly. DBGs for three directions are also illustrated (pictures adopted from [34]).

Having defined this data structures, the partitioning algorithm can be outlined. Like the previously described approaches, the calculated partitions are stored in an AND/OR

graph. Hence the partitioning algorithm separates an assembly into two connected subassemblies. As stated earlier, the partitions are not tested for separability, but chosen to be already separable. This is done by considering connected subgraphs of the DBGs. A DBG contains contact information like an undirected contact graph, but also directed arcs denoting whether a part can be moved into a certain direction or not. Hence, following the directions of arcs creates separable partitions. An algorithm starts at some random node and adds all successor nodes including the starting node to one partition. A second partition is determined by adding all ancestor nodes of a neighbor of the starting point including the neighbor. Both partitions are now built recursively by adding neighboring nodes to the partitions, until the whole assembly is separated. Applying the method to all DBGs generates all separable connected partitionings. Because the direction lying on a circle defined by a constraint results in a sliding motion, the intersection points of the circles are the least blocking points. Therefor it is sufficient to consider only DBGs for this intersections.

The data structures are also able to hold blocking relationships for rotational motions. The solutions to the system of inequalities would then contain six variables, instead of three, a single translational motion and rotations around each axis. An NDBG incorporating this solutions then is a unit sphere S^5, and the constraints are represented by 6D hyperplanes.

The previously described NDBG is used to determine local freedom of parts. Wilson also describes a method to generate an extended NDBG, incorporating blocking relationships of translations to infinity. For this purpose, the Minkowski difference for each part pairing is calculated, resulting in a so called configuraton space (c-space) obstacle. The Minkowski difference is built by subtracting each point of one part, from each point of a second part. Moving the parts also changes the position of the obstacle in c-space. When such an obstacle intersects the origin of this space, the associated parts collide. Using Minkowski difference is a common technique for motion planning of robots in production environments [20] or as used here, for assembly sequence planning. It can be used to find valid separation path using multiple directions as discussed in [13], instead of a just single translation.

The extended NDBG is created by projecting the c-space obstacles onto the unit sphere S^2, thus creating non-convex cells representing blocking constraints. This approach is intuitive, because the direction of motion starting in the origin of the c-space is either blocked by an obstacle in this direction, or not (see figure 2.21). The resulting DBGs cannot be used to determine all separable partitionings directly, because when considering

global motion it does not map simple contact information anymore.

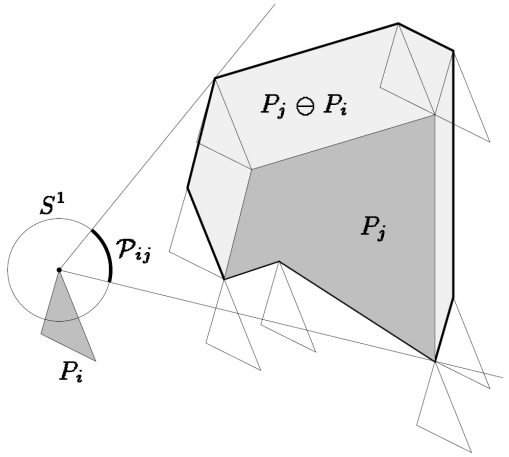


Figure 2.21: P_i is the moving part, P_j the obstacle. The corresponding c-space obstacle is the polygon surrounded by the bold contours. The center of the c-space is surrounded by a unit sphere S^1, onto which the c-space obstacle is projected. Extended motions of P_i into this direction are blocked (picture adopted from [12]).

Guibas et al [11] present an improvement to the previously described data structures for testing local freedom, thus reducing the number of DBGs which have to be considered when calculating all partitionings. In the following, the technique is discussed for the case of single translations, although it also maps to rotational motions.

Instead of applying all contact constraints directly on the sphere S^2, the constraints for each part pairing are collected into a set to create a single area on the sphere. The intersection of all this areas form cells. The interior of such a cell basically is the solution set described by a convex polyhedral cone and represents valid movement directions. When moving from one cell to a neighboring one, without leaving the area, which is associated with the original cell, a constraint is removed. Hence, the DBG for the new cell contains the same partitions as the one of the original cell. This observation leads to the definition of maximally covered cells, which are associated with a larger number of valid areas than their neighboring cells. Figure 2.22 shows examples of maximally covered cells.

Romney et al. [34] utilize DBGs and NDBGs in an assembly planning system called Stanford Assembly Analysis Tool (STAAT). As done by Guibas et al. [11], areas representing valid motion directions are stored in the NDBG, hence reducing its complexity. The stored information is referred to as LTF, which is the LTB used by Agrawala et al [2] described in 2.1.3.

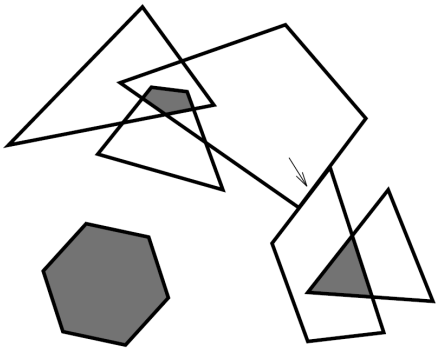


Figure 2.22: Examples of maximally covered cells. The hexagon is covered by only one area, the triangle by two and the pentagon by three. The arrow indicates a one-dimensional maximally covered cell, where two cells intersect only at their borders (picture adopted from [11]).

Kaufman et al. [16, 17] implemented an assembly planning system incorporating DBGs and NDBGs to determine local freedom. In addition, a method to check global motions is presented, which takes advantage of the z-buffer of graphics hardware. When splitting the assembly into two locally free subassemblies, the translation to infinity into the same direction also has to be tested. After setting the viewing direction to be the separation direction, the moving partition is rendered in black, writing its depth values into the z-buffer. The second subassembly is rendered in white, setting the depth function to greater-than. Hence, this partition is only rendered in places, where it blocks the movement of the translated subassembly. A collision occurs, if white pixels are detected.

Thomas et al. [38] create binary, monotone and contact-coherent assembly sequences, following an approach very similar to the one used for constructing the NDBG for translations to infinity [40]. There, c-space obstacles were projected onto the unit sphere S^2 to create cells of valid motion directions. To improve the handling of the NDBG, both hemispheres were stereographically projected onto parallel planes, tangent to the poles. The presented technique utilizes graphics hardware to be able to quickly create the stereographic projections of c-space obstacles and presents an efficient method for determining the separability of partitions.

To determine if a part pairing is separable, the c-space obstacle is created by calculating the Minkowski difference. As stated earlier, the obstacles are projected onto the unit sphere S^2 and the hemispheres are then projected stereographically on two parallel planes. Hence the 3D representation of the sphere is reduced to two 2D planes. A point on the unit sphere can now be calculated by using simple 2D coordinates. A third value is associated with

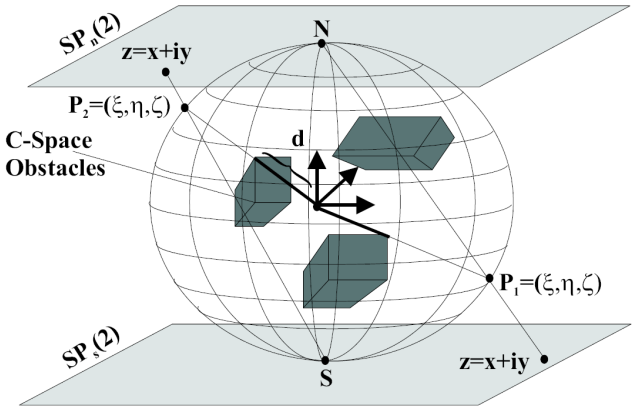


Figure 2.23: C-space obstacles are projected stereographically onto two parallel planes (picture adopted from [38]).

each plane coordinate, storing the distance of the c-space obstacle associated with this point from the origin of the c-space. This describes a vector of motion for the moving part, which does not cause interference with the static object. If it moves further into the same direction than the stored distance, the objects collide. Figure 2.23 illustrates the concept of the algorithm.

To retrieve valid separation directions for a pair of parts, the maximum distance values stored in both planes are determined and set to one. Each point containing a lower distance is set to zero. This leads to binary stereographic projections, consisting of areas of ones representing valid directions. To select a separation direction, it is sufficient to consider only one point out of each area. The outlined creation of stereographic projections can be performed using graphic hardware, where the planes are rendered into textures.

Before creating the assembly sequence, all stereographic projections of each part pairing are generated. An undirected contact graph is then calculated, by determining all distance values, falling below a certain threshold. If such a case is found in one of the projections, a connection between the two associated parts is established. All possible connected partitionings into two subassemblies are then created by performing a DFS on the contact graph, starting with each part in the assembly. The DFS only guarantees, that the enumerated partition is connected, hence the connectivity of the second one has to be checked separately. Separability is tested by performing an AND-conjunction of the binary stereographic projections of each part pairing between the two partitions (see figure 2.24). If the resulting projection still contains ones, then the partitions are separable and added to and AND/OR graph. The algorithm splits all subassemblies recursively, until

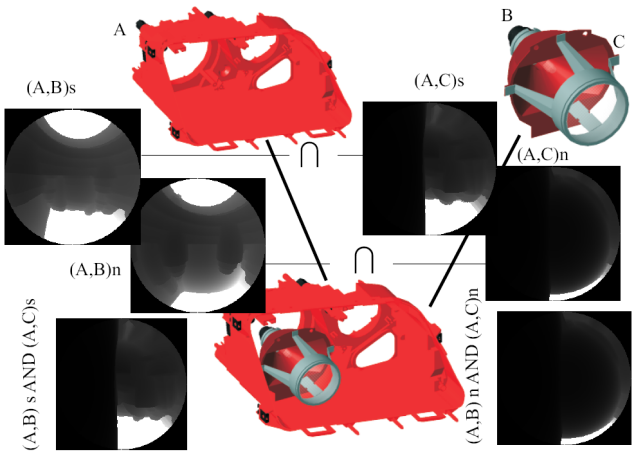


Figure 2.24: An example of an AND-conjunction of binary stereographic projections of two partitions (picture adopted from [38]).

only single parts or inseparable partitions are left.

2.2.3 Discussion

Examining the previous description of different assembly sequencing algorithms, some main tasks can be identified. Removable subassemblies have to be identified, as well as assembly directions. Both objectives are closely related to each other. After finding all valid subassemblies, a sequence is determined out of this collection. In assembly planning such a sequence may be optimized to create only stable subassemblies, where parts hold themselves together. Another criterion may be used to create orderings, which are ideal for executing several assembly tasks in parallel.

In the following, the creation of partitions and the determination of valid assembly directions are discussed in separate sections. Then the sequencing algorithms themselves will be investigated. When appropriate, connections to the presentation of different methods for creating explosion diagrams will be considered (see section 2.1).

Partitions

For linear sequences, where only single components are added into a subassembly [25, 41, 42], a partition consists only of one part. When subassemblies containing more than one part are mated, the task of finding such partitions becomes more expensive. To reduce the computational complexity, the partitioning algorithms are restricted to produce only

two valid partitions, hence allowing only binary or two-handed assembly sequences, e.g. as in [15, 40].

An additional restriction reducing the complexity of partitioning algorithms is the creation of contact coherent subassemblies, i.e. the parts of the generated partitions are in contact with other parts of the same partition. In practice, if a partitioning consists of several groups of parts, which are not in contact, an assembly task would become more complicated. Although, all groups follow the same insertion direction, a robot worker would have to move the partitions separately and a greater number of gripping points has to be found [40]. In connection with explosion diagrams, allowing such partitions, would result in the movement of groups of parts, which are not necessarily related to each other in any way. In general, there is no need to create unconnected partitions, because if a partition contains disconnected groups of parts, which are all removable along the same motion vector, there can also be found separate contact-coherent partitions, containing only one of this groups (see figure 2.25). Thus contact-coherence is a very useful constraint to reduce the complexity of partitioning. In the following, only binary and contact-coherent partitionings are discussed.

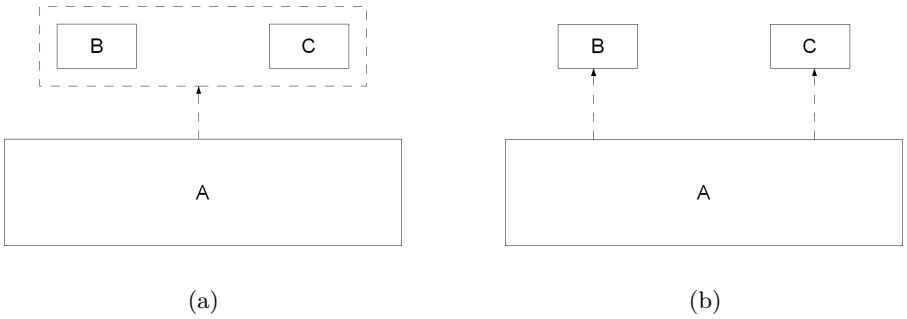


Figure 2.25: (a) An unconnected partition containing parts B and C. (b) The same parts can be moved in separate connected partitions.

Homem de Mello et al. [15] creates an undirected contact graph out of information of a high-level representation of an assembly. This graph is input to a partitioning algorithm, which tests the found partitions for local separability. Using directed DBGs as proposed by Wilson et al. [40], this test is not necessary anymore. As described earlier, a DBG is a more advanced version of a contact graph, where the directed arcs represent blocking relationships. An algorithm following this arcs, directly creates separable connected partitions. This technique cannot be used, when the DBG is built for extended translations, because it does not map the connection relationships anymore. The partitions again have

to be calculated out of a contact graph, as done by Thomas et al. in [38].

To generate an assembly sequence, a product is partitioned recursively, until all subassemblies have been separated. The evaluation of partitions can be stopped very early, when certain optimization criteria do not hold, e.g. in [15] a criterion is the stability of the created subassembly. On the one hand, this reduces the number of partitions which have to be tested, on the other hand, also the search space of the assembly sequencing algorithm.

As mentioned before, the great advantage of a DBG is, that separable partitions can be created directly from the graph, thus removing the requirement of testing for separability. Li et al. [21] and Agrawala et al. [2], outlined in section 2.1, use DBGs to test blocking relationships of parts and to build a valid explosion sequence. Both described methods do not fully benefit from this data structure, because only partitions containing one part are allowed.

Although this thesis allows assemblies to be separated into subassemblies containing several objects, the partitioning algorithm works on an undirected contact graph, instead of a DBG. Partitioning has to be performed in a pre-processing step, thus is not executed in real-time. Efficiency is not of great importance. Nevertheless, a method was implemented creating partitions on-the-fly, when required. In this case, only one partitioning is chosen out of the whole list of partitions of a subassembly to be separated. Therefore, only a certain path through an AND/OR graph is calculated. To further improve the efficiency of the partitioning algorithm, DBGs could be used. Using the current information, DBGs can theoretically be generated. Part contacts are calculated and blocking relationships are determined for extended motions between all pairs of parts. Combining this information blocking relationships for part neighbors can be transformed into a DBG. Nevertheless, without identifying real contact constraints like Wilson [40], an NDBG cannot be built.

Assembly Directions

After determining parts, which may be removed, the separability has to be tested. In [41] the directions are given by manually defined precedence relationships for certain motion vectors. These constraints are evaluated to find an assembly sequence. Mok et al. [25], only provide a solution for building stack assemblies, hence there has to be considered only one insertion direction. Woo et al. [42] use the normals of the contact faces of the parts to test for separability. If all normals lie in the same hemisphere of the unit sphere S^2, a part is meant to be locally free. They refer to this check as monotonicity check.

Homem de Mello et al. [15] take a similar approach for testing such planar contacts, but also support other contacts. The assembly is described by a relational model, containing information about the types of contacts between the parts. The strongest contact between parts, is used for determining local freedom. For example, a pin in a hole restricts the motion to one direction, while planar contacts allow translations spanning a whole half sphere. In this case, the normals of the face contacts between the parts to be separated are used in a system of linear inequalities $n_c x \geq 0$. If there is a non-zero solution, the objects are locally free. The solution space contains all valid separation directions.

While Woo et al. [42] only consider planar contacts, and Homem et al. [15] retrieve contact information from a high-level description of the assembly, Wilson [40] derives local freedom constraints from a pure geometrical description of a mechanical assembly. A thorough discussion of geometrical contacts is given, not only dealing with planar contacts, but amongst others also including the description of edge-edge contacts or edge-face contacts. A contact is then expressed by a sufficient number of point-plane constraints, each having a normal associated with it. Using all constraints an `NDBG` is built, which in fact contains the solutions of the previously mentioned system of inequalities. Wilson also describes how the linear inequalities can be used to determine valid rotational motions. The `NDBG` contains information on all possible removal directions.

Until now, only directions valid for infinitesimal motions were discussed. The problem with local freedom is, that parts can still interfere with other objects, when moved into a locally free direction (see figure 2.26). Hence other tests have to be applied. After having determined a locally free subassembly, global separability can be checked by sweeping the parts to infinity and checking for interferences. Kaufman et al. [16, 17] describe such an approach utilizing graphics hardware, in detail. Wilson [40] introduces an extension to `NDBG`s, able to handle global translations. Instead of using contact constraints, so called c-space obstacles are projected onto the sphere describing the `NDBG`. A part is free to move, if the point on the sphere, representing the motion direction, is not contained in such a projection. Because c-space obstacles restrict only translational motions, rotations cannot be derived from the `NDBG` anymore. Thomas et al. [38] present an efficient technique for calculating the information contained in an `NDBG`. Graphics hardware is used to build the projections of the obstacles.

Li et al. [21] and Agrawala et al. [2] outlined in section 2.1 use `DBG`s to identify local blocking relationships. Li et al. [21] state, that the implemented approach has problems identifying blocking relationships of complex objects containing a lot of normals showing

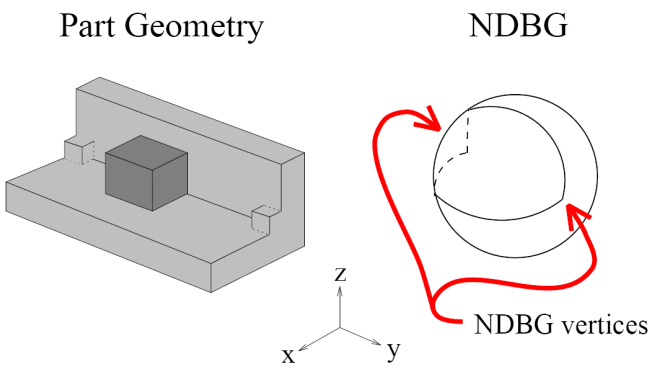


Figure 2.26: Although the part is free to move locally into the two tested directions given by the NDBG, the translation to infinity is blocked (picture adopted from [34]).

into different directions. The issue arose when computing the constraint motions for a spring. Such situations can be resolved using the more involved method of projecting c-space obstacles [38, 40], or the simple sweeping proposed by Kaufman et al. [16, 17].

Furthermore, Li et al. [21] and Agrawala et al. [2] allow only manually defined motion directions, thus only a very limited number of DBGs have to be calculated and there is no need for an NDBG.

This thesis tests extended translations using the sweeping technique presented by Kaufman et al. [16, 17]. Blocking relationships for predefined directions are calculated for each part pairing. A partition then moves along a motion vector, which is not blocked for any contained part.

Sequencing Algorithms

Linear sequences are very restrictive, allowing only a single part to be inserted in each assembly step. Woo et al. [42], restrict the presented algorithm even more by considering only totally ordered assemblies. Totally ordered means, that each part can be separated, by removing at most one other part. Hence, the applications of such an algorithm are quite limited. Wolter [41], does not follow the approach of assembly-by-disassembly. An assembly sequence is created by evaluating precedence constraints, which state, that a part can be inserted before another part, when moving along a certain motion vector. Directions and constraints have to be defined manually in this algorithm. Mok et al [25] also generate a linear assembly sequence, but only for a single motion direction. Furthermore, to be able to apply the proposed method, the assembly has to resemble a stack structure.

The other presented methods [15, 38, 40] use AND/OR graphs to define a search space for assembly sequences. One path through the graph represents a single sequence. Weights assigned to the arcs of the graph can represent certain criteria like stability or manufacturing costs. An optimizer may evaluate these factors during the creation of an AND/OR graph and let the partitioning algorithm drop certain subassemblies, which are not optimal. This is more efficient, than building the whole graph of geometrically separable partitions and then finding a path through the graph.

This thesis uses an AND/OR graphs to represent all possible explosion sequences. In a first implementation the whole search space was calculated in a preprocessing step. Depending on the size of the assembly, the search space can grow rather big. Hence, another approach was used, where the partitions of a certain subassembly are calculated on request. This way, only those subassemblies are further refined, which are currently of interest to the application. In addition, the preprocessing step is reduced to the calculation of blocking relationships and contact information. A performance penalty has to be taken into account, when calculating the partitions online. But this only is visible, when initializing a new style. Depending on the complexity of a model, the user may prefer one approach or the other.

Chapter 3

Concept

This thesis presents a flexible system for creating explosion diagrams in augmented reality. Explosion orderings, as well as separation directions are calculated automatically. Certain parameters can be changed by using combinations of different types of styles, enabling the user to change the presentation of the diagram. Each type of style influences a distinct aspect of the explosion, like the ordering of the part removal. In the following, a subassembly is called a partition, which contains one or more parts. Hence also single parts may be referred to as partitions. The explosion input consists of a 3D model of the assembly, where each component is represented as closed 3D polygonal mesh.

Because the input consists of models of assemblies, it is obvious to apply algorithms out of the assembly planning domain to find appropriate sequences. This leads to appropriate disassembly orderings, as well as separation directions, into which removed parts do not collide with the rest of the assembly. The output of such an assembly sequencing algorithm can be influenced by user defined optimization criteria. Hence, very different explosion sequences can be generated, resulting in a variety of explosion diagrams. The used sequencing method requires the input model to be connected. Each part of the assembly has to be in contact with at least one other component.

In the following, the terms explosion and assembly sequences are used interchangeably, although sequences for exploding a model have to fulfill different requirements than those for assembling it. For instance, it is very important for an assembly sequence to create stable subassemblies, which do not break apart in a real world environment. For an explosion diagram a prime target is to reveal those parts, which occlude certain internals. In addition, the parts are not translated to infinity, as done by many sequencing algorithms, but have to be arranged relative to other objects. This introduces the need to find a

separation distance and to implement animations moving components into their exploded position. In this thesis, the arrangement of parts and the corresponding movement animations are realized by employing a physical layout approach using the Open Dynamics Engine (ODE) physics engine[1].

User interaction consists of navigational tasks and the possibility to either explode all parts or certain layers. Single parts can be removed in an ordering defined by an assembly sequence. As stated before, the explosion ordering can be influenced by switching through predefined styles. To enhance the contextual information of an explosion and to facilitate the orientation of the user, some simple visualization methods are utilized.

This chapter is organized into five sections. In section 3.1 the challenges arising when calculating an explosion diagram of a typical mechanical assembly are described. Section 3.2 presents the used sequencing algorithm, which solves some of the problems described in the previous section. The general approach for creating an explosion and the possible user interaction is outlined in 3.3. In 3.4 the different types of styles and their influence on the explosion are specified in more detail. Visualization methods enhancing the ability of the user to reconstruct the exploded model are described in 3.5.

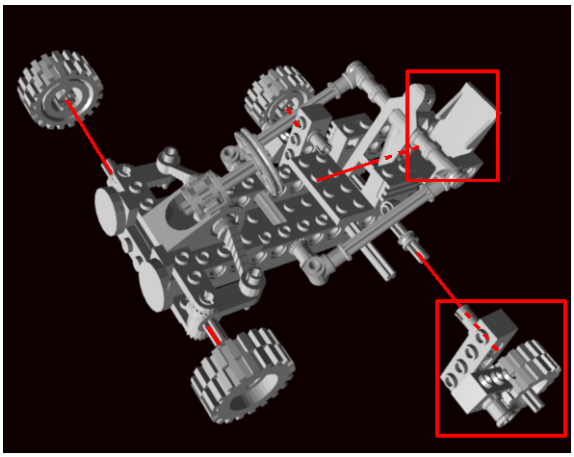


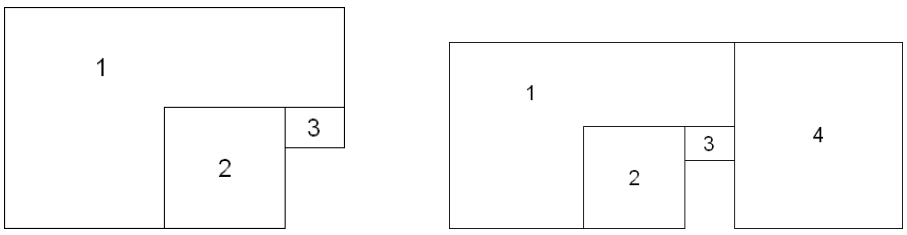
Figure 3.1: An artificially created erroneous explosion. The seat interferes with the back geometry, the wheels are not moved the same distance and one of them collides with parts exploded afterwards.

[1] http://www.ode.org/, last visited October 13, 2008

3.1 Challenges of Creating Explosion Diagrams

When creating an explosion diagram automatically, several problems come up. Not only explosion sequences and directions have to be calculated, but also the distance has to be determined appropriately to be able to arrange the parts. In addition, already exploded objects should move relative to other objects, which are exploded afterwards. Figure 3.1 shows an artificially generated erroneous explosion of one of the models used in this thesis. It illustrates some of the mentioned problems. Because, no explosion sequence was determined, the seat is moved through the back geometry of the car. But in this explosion state, it should still be blocked. All of the wheels of the car are exploded, but no to the same distance. One of them even collides with parts, translated afterwards, because the wheel did move far enough.

In the following, such challenges are described in more detail. Most of the examples use the assembly depicted in 3.2, if not outlined otherwise.



(a) (b)

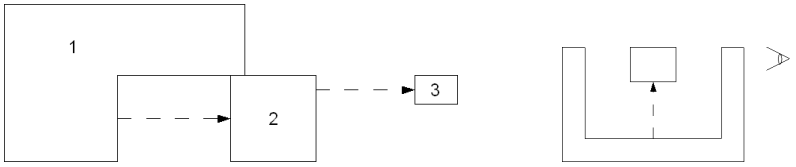
Figure 3.2: This assemblies are used for outlining the challenges of creating an explosion diagram. Part 3 may vary in size.

3.1.1 Explosion Sequence and Directions

Explosion sequences and directions have to be calculated in a way that the removed parts can be transformed collision-free into their exploded state. Hence, an exploding partition must not collide with the rest of the unexploded assembly. In addition, in order to truly separate parts from each other, the separation distance has to be chosen appropriately. Figure 3.3(a) shows an example where the distance is too small. Part 2 still is in contact with part 1. But the requirement to break contact between all parts is not sufficient. There can occur situations, where parts do not touch the partition they are exploded away from anymore, but still are not moved far enough. Figure 3.3(b) depicts a case, where part 2

is originally contained in the boxlike part 1. Although both parts are not in contact, the translated part is not fully visible, because it is surrounded by another part.

If many parts are involved in an explosion, the explosion diagram can grow rather big. Each part has to be translated away from the rest of the assembly. Thus a mechanism to limit the range of the explosion is required. When performing an explosion in augmented reality (AR), where a 3D model maybe is placed on top of a real world object, an additional criterion may be that certain explosion directions and distances are blocked. Exploded components should not be allowed to penetrate certain obstacles in the scene, like a ground floor or walls.



(a) Break contact

(b) Containment situation

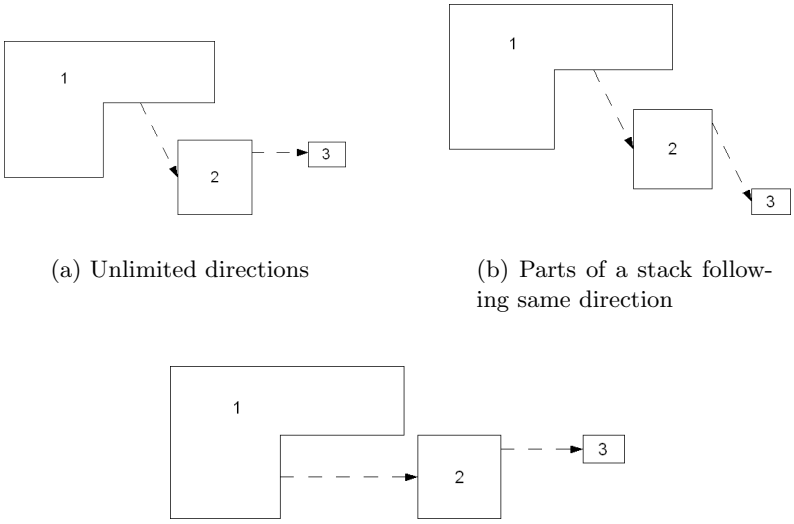
Figure 3.3: The explosion distance should be chosen in a way so that the exploded parts are displaced from the rest of the assembly. (a) Part 2 should be moved further to break the contact with part 1. (b) Ideally part 2 leaves the surrounding box.

3.1.2 Limiting the Number of Directions

The more directions are used to create an explosion diagram, the harder it could be for a user to identify parts associated with each other. Sonnet et al. [37] do not restrict the possible directions in any way. Hence, each part can move along its own explosion vector. To make the explosion diagram more homogenous, stacks can be used. A stack consists of a set of parts, which follow the same explosion direction [2, 34]. Hence, two neighboring objects follow the same motion vector. The application of a stack is depicted in figure 3.4. Parts 2 and 3 are free to move along a wide range of motion vectors. A stack can be formed incorporating these parts, which on the one hand reduces the number of directions and on the other hand creates a more homogenous explosion.

Utilizing stacks may not reduce the explosion directions to ideal ones. In figure 3.4(b) parts 2 and 3 are exploded diagonally, which may be confusing for an observer. Therefore the number of possible explosion directions has to be limited further to a few reasonable

ones. This can be done in advance by only allowing a small subset of motion vectors. This approach leads to cases where parts cannot be separated from each other, because their assembly direction is not specified. Another method is to find the minimal number of separation directions, which enable a sequencing algorithm to fully disassembly the product.



(a) Unlimited directions

(b) Parts of a stack following same direction

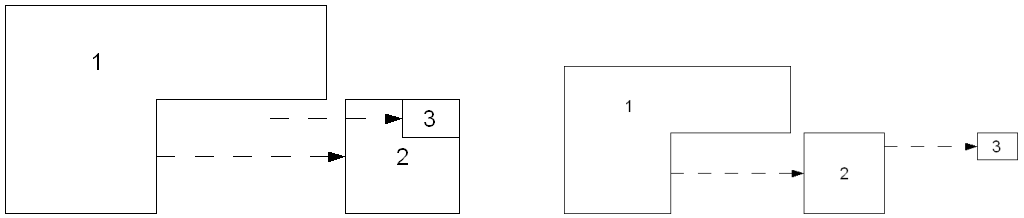
(c) Limited directions and stack

Figure 3.4: Identifying stacks of parts, to group them together and give them the same explosion direction

3.1.3 Moving Already Exploded Parts

Exploding parts have to be prevented from intersecting already exploded ones. This is illustrated in figure 3.5. Part 2 is moved and ends up colliding with the previously translated part 3. Hence, when a part is exploded, the explosion motion has to be propagated to other objects, which either were exploded before, or are otherwise associated with this part, maybe because they are in the same partition. Bruckner et al. [3] utilize spacing forces, pushing parts away from each other. Although translated parts do not collide anymore, the objects are not moved relative to each other. For this purpose, a `PC` relation has to be formed, where parts are attached to a parent part. This is a 1-n relationship, one parent for a set of n children. It is obvious, that a child can only have one parent part. Otherwise, it would not be clear which part movement is affecting it. Nevertheless, a parent may have more than one child.

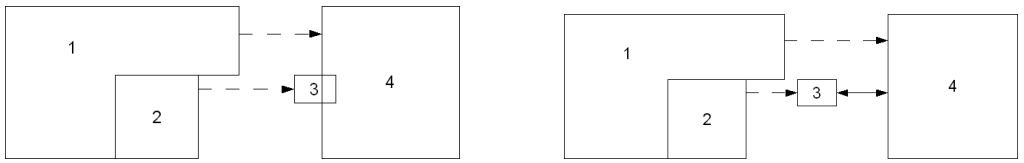
Another restriction to a relation is, that the involved parts have to be in contact. In an assembly, one of the direct neighbors is very likely to be a contextually associated component. Moving it relative to a part at the other end of the model, would be misleading. Thus, it makes perfect sense to only search the neighbors of a part for a parent.



(a) Part 2 collides with part 3

(b) Part 3 moved with parent part 2

Figure 3.5: Parts have to be moved preserving their explosion direction and relative to their parent, when the parent is exploded



(a) Part 3 collides with exploded part 4

(b) Part 3 pushes part 4

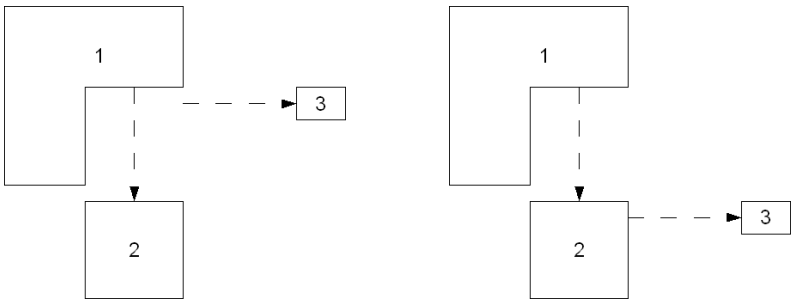
Figure 3.6: To prevent a collision, part 4 has to be moved along its explosion vector to make room for the new part.

After establishing a `PC` relation, the movement of parts, not associated with already exploded objects, may still lead to collisions. Figure 3.6 illustrates such a case. Part 4 has no relation to part 3. Hence, after exploding part 3, it collides with part 4. Already exploded objects constraining the translation of other parts should be moved out of the way, while changing the original explosion directions only marginally. Another method may change the `PC` relations directly, so that part 4 moves relative to part 3.

3.1.4 Parent-Child Relations

Having introduced the notion of `PC` relations in 3.1.3, the new challenge is to determine appropriate ones. Finding such a relationship is easy, when a component is only in contact with one other part (see figure 3.7). If a part has more than one direct neighbor, the

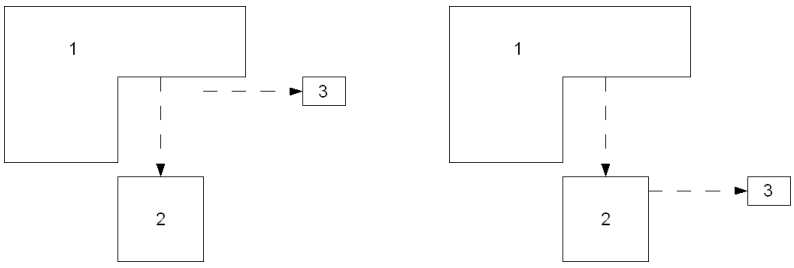
resulting n-1 relationship has to be reduced to a single 1-1 relationship (see figure 3.8)



(a) No relationship established

(b) Relationship formed between part 2 and 3

Figure 3.7: Part 3 only is in contact with part 2 making it easy to determine a parent-childrelationship.



(a) Part 1 is parent of part 3

(b) Part 2 is parent of part 3

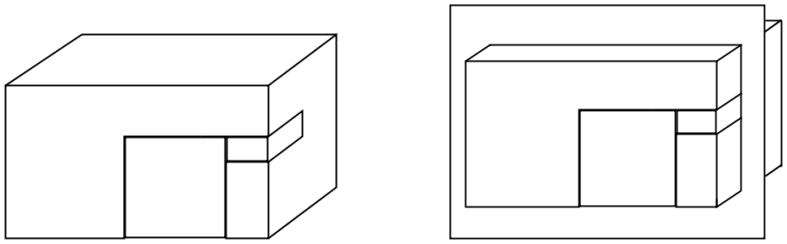
Figure 3.8: The n-1 relationship of part 3 has to be broken down to a 1-1 relationship.

The following examples are outlined using a cut through the assembly of figure 3.9. Resolving an n-1 relationship between two parts can lead to the case, where a parent moves its child as intended, but the motion vector of the child interferes with another part as illustrated in figure 3.10. There the association with part 2 has to be removed or it should not be established in the first place. This can be done by checking if the unexploded children of the parent to be exploded collide with any unexploded other geometry. In the depicted case the tested partition would consist of parts 2 and 3. A collision is detected between parts 3 and 1, hence part 1 becomes the new parent of part 3.

The algorithm cannot resolve the same problem if part 3 is smaller and thus only in contact with part 1. No other parent can be assigned to it. In this case, the requirement

of parents having to be direct neighbors of their children has to be removed, or at least softened to allow parts within a certain radius to be parents.

But still there can occur cases, where this algorithm does not produce correct results, even when the contact requirement for parents is removed. Figure 3.11 depicts a situation, where the motion vector of the exploded part 3 interferes with part 1, although the separation test involving the partition containing parts 2 and 3 was passed. To solve this problem, the partition also has to be tested in its exploded state. The moved parts are swept to create a volume starting at their original location relative to the parent, and ending at their finally exploded position. Then the collision test is performed for this volume.



(a) Example assembly

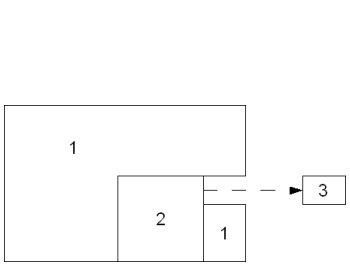
(b) Cutting plane through assembly

Figure 3.9: (a) The assembly consists of three parts. (b) The cutting plane to create a 2D image of the assembly.

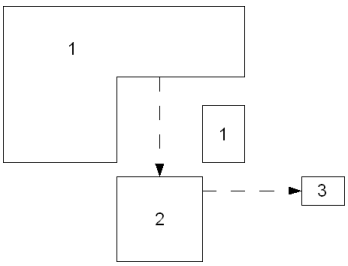
3.2 Assembly Sequence Generation

An assembly sequence defines the ordering of bringing parts and partitions together without colliding with already inserted objects to form an assembly. Furthermore, correct assembly motions are calculated. Most assembly sequencing algorithms follow the approach of assembly-by-disassembly to reduce the computational complexity. Because an explosion diagram resembles the disassembly of a product, an assembly sequence determined this way, can also be used to generate an explosions. But instead of removing parts to infinity as done in assembly sequencing, they are only offset from the rest of the assembly by a certain distance.

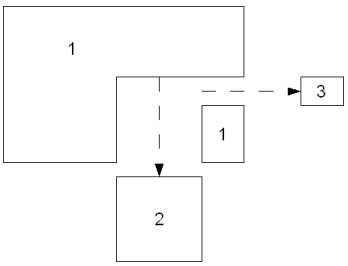
A data structure out of the assembly planning domain is the AND/OR graph. It can store all possible assembly sequences of a product. Therefore, it is very practical to use this



(a) Explosion of part 3

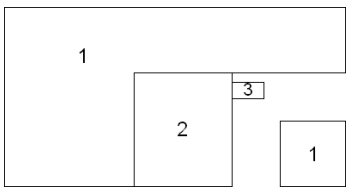


(b) Motion vector interferes

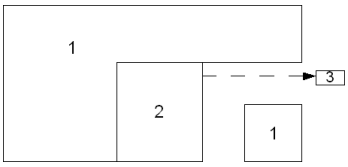


(c) Interference prevented

Figure 3.10: (a) The explosion of part 3, having part 2 as parent. (b) The motion vector of 3 interferes with another object, when moved relative to its parent part 2. (c) This conflict is resolved by assigning it to part 1.



(a) Unexploded state



(b) Explosion of part 3

Figure 3.11: (a) The unexploded assembly consisting of three parts. Part 3 is associated with part 2, because it was only checked if the unexploded parts interfere with any other part. (b) After exploding part 3, the motion vector again interferes with part one

data structure as foundation of a flexible system for creating different explosion diagrams of a single model. Depending on the current requirements defined by the user, a certain path through the AND/OR graph can be chosen.

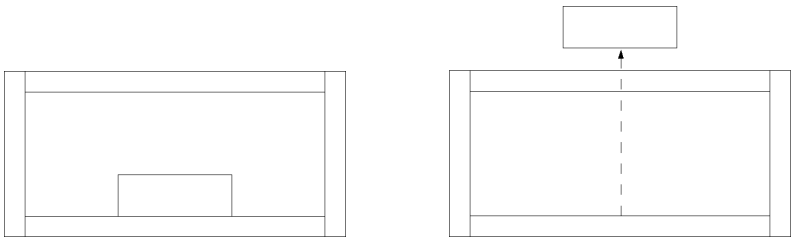
Following this reasoning, the thesis utilizes an assembly sequencing algorithm to create the set of all possible explosion sequences, which are stored in an AND/OR graph. To reduce the computational complexity of partitioning the assembly, only binary and contact-coherent sequences are considered. Furthermore, the separation motions are limited to translations along one out of a set of predefined vectors. Hence, translations along multiple paths and rotations are ruled out. As stated by Agrawala et al [2] and Li et al [21], for most assemblies it is sufficient to consider only the six major axes directions of the coordinate frame of the model. This way the number of potential explosion directions is reduced and the final explosion diagram is more homogeneous. If partitions cannot be split any further using one of the motion vectors, maybe because parts are interlocking, different visualization methods than an explosion diagram can be considered. For example, Li et al. [21] use splitting or cutaway techniques. The system implemented in this thesis allows the manual definition of explosion directions. Furthermore the least blocking direction is chosen as separation direction, if all vectors are blocked. Hence, interlocking parts can be handled.

In the following the necessary preprocessing steps for creating the AND/OR graph are described in section 3.2.1. It involves the calculation of contact information and valid separation directions of parts. Processing this information, an AND/OR graph is determined as described in section 3.2.2.

3.2.1 Preprocessing Steps

An undirected contact graph is created out of the geometrical description of the model. Each part is represented by a node, an arc between two nodes defines a contact between the involved parts. The graph is used for partitioning the assembly and its subassemblies into two connected partitions, which are then added to the AND/OR graph representing the explosion sequences.

Furthermore, the blocking relationships between the parts of the assembly have to be determined to be able to test the separability of certain parts or partitions. Most assemblies allow only a limited number of assembly directions, as stated in [2]. Thus, by default only the removability along the six major axes directions is considered. Nevertheless, the tested directions can be chosen by the user, if necessary.



(a) (b)

Figure 3.12: (a) The rectangular part is completely surrounded by other components. (b) Using only local blocking constraints, the part is only blocked by one part and is exploded updwards.

If only local blocking relationships for partitions are calculated, a part may be locally free to move, but in fact is blocked when moving further along the same explosion vector. Hence it is not globally free. This situation occurs, when blocking components are not in contact with the tested part. Therefore, this thesis considers global blocking constraints. Another advantage of global constraints is, that an assembly is truly exploded from the outside to the inside, covering objects are removed before exposing internals. Figure 3.12 illustrates an erroneous explosion using only local blocking constraints. The contained rectangular part is exploded although it is still blocked by another part.

To determine valid separation directions, all pairs of parts are considered. One part of such a pair is chosen to stay static or passive, while the second one is moved along each of the given motion vectors, hence is active. If it interferes with the static part, the direction is blocked. All valid separation directions for a part pairing are stored. This approach was adopted from [38].

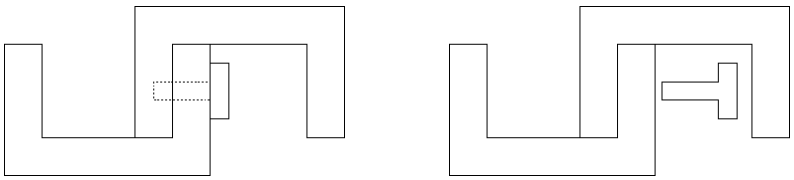
3.2.2 AND/OR Graphs

An explosion ordering is determined by recursively dividing the assembly into disjunct partitions, until all of the created subassemblies have been disassembled. The output of the sequencer is stored in an AND/OR graph. The root of this graph contains the completely assembled model, which is partitioned into all possible combinations of two connected subassemblies. This derives from the previously defined requirement of calculating only two-handed and contact-coherent assembly sequences. Each of the found partitionings is connected to the root node via a hyper-arc, which is referred to as AND-branch. A sequence can choose one out of the set of all AND-branches of a node. Thus, this AND-

branches are collected in an OR-branch. The previously determined subassemblies are further subdivided in the same way, until the leaf nodes contain only single components, or the partitions cannot be separated anymore.

The pre-computed undirected contact graph is used to calculate all possible partitions of a subassembly. As done by Thomas et al. [38], a DFS is performed on the contact graph. The DFS starts at a node and selects one of the connected edges to reach the next node. This is done on each visited node, until the whole graph has been processed. The algorithm then steps back to the previous node and selects a different branch to an unvisited neighbor. This recursion creates connected subgraphs g_i of the contact graph G. If both g_i and $G \backslash g_i$ are connected, two potential partitions of the assembly have been found.

Before adding the connected partitions of an assembly to the AND/OR graph, the separability has to be tested. In the preprocessing step, the valid explosion directions of each pair of parts were calculated. Considering two partitions P_i and P_j, containing the parts p_i and p_j, the set of common directions of all pairs (p_i, p_j) is created. If the set is empty, the partitions are not separable and thus not added to the AND/OR graph. This way global separability is tested.



(a) (b)

Figure 3.13: (a) No part is free to move to infinity into any direction. (b) Using local blocking constraints the screw can be removed

There can occur situations, where a subassembly cannot be separated at all using global blocking constraints. Such an assembly is depicted in figure 3.13. After moving the locally free screw, the other parts are globally separable. Hence, if a partition is not separable using global blocking constraints, the partitioning algorithm should fall back to local blocking constraints. Furthermore, many of the created explosion diagrams depend on separable single part partitions. Hence, if no globally separable single part partitions are found, locally separable single part partitions are determined. The currently implemented system follows this hybrid approach, but does not consider true local blocking constraints.

A true local constraint is determined by checking if an object is free to move into a certain direction without colliding with contacting *faces*. The local separability test performed in this thesis, checks if a part can be exploded without colliding with any of the contacting *parts*. This way, local constraints can be derived from the global blocking constraints and no additional configuration information has to be calculated. Nevertheless, a flaw of this approach is that global and local blocking constraints are the same, if all parts of an assembly are in contact. For instance, applying this algorithm to the assembly of figure 3.13 would not lead to a partitioning, because the screw is in contact with all other parts. Although this is a limitation to using true local blocking constraints, the discussion will provide an example assembly, which is only separable when applying this hybrid approach.

3.3 Creating Explosion Diagrams

An AND/OR graph defines the search space for retrieving a single instance of an explosion sequence, which is a certain path through the graph leading to the complete disassembly of the product. But still, additional information for creating an explosion diagram is necessary. Up to this point, there is information available about the explosion ordering and the partitioning of the assembly, as well as a range of valid removal directions for each partition. To create an explosion, one separation direction has to be chosen for each part, as well as relationships between the objects so that translations can be propagated through to associated parts. The relations, which have to be established, are the `PC` relations, mentioned in 3.1.

Hence, the data used to create explosion diagrams consists of a path trough the AND/OR graph, appropriate separation directions and distances for each partition, as well as `PC` relations to be able to move parts relative to other objects. The appearance of the explosion changes with the input data. A sequence removing whole subassemblies is different from a sequence exploding only single parts. The selection of `PC` relations also has an impact on the explosion. This notion will be explained later on, when `PC` relations are discussed in more detail in 3.3.1. Hence, providing the user with the possibility of changing the required data enables him to create different explosion diagrams. For this purpose, styles are defined, which can be exchanged independent from each other. Aside from the three styles, required to influence sequence, separation directions and `PC` relations, an additional ordering style is defined. This is necessary, because the representation of the sequence as AND/OR graph does not contain information on which partition is

to be exploded next, if several subassemblies can be separated in parallel. Furthermore, styles may be influenced by a user defined focus assembly.

The explosion itself is then performed using a force-based layout approach similar to the one presented in [3]. There springs are used to hold objects in place, while explosion forces try to move them away. In this thesis a spring-damper system is utilized to explode the parts. This is necessary, because a real physical framework is used, where mass inertia is present and has to be compensated.

In the following, the basics of finding valid PC relations are presented in section 3.3.1. Then, the general force-based approach to create the explosion diagram is described in section 3.3.3. At last, the possible user interactions, including the general idea of using styles are outlined in section 3.3.4. Styles by themselves are explained in more detail in section 3.4.

3.3.1 Determining Parent-Child Relations

As discussed in 3.1, PC relations are required to propagate the translation of a part to other associated objects. In the following, only general considerations related to the creation of such a relation are presented. A style is responsible for the selection of a PC relation and has to fulfill the defined requirements.

A PC relation is a 1-n relationship. Hence, one part can be parent to n children. On the other hand, a child can only have one parent it can move relative to. Establishing such a relationship by investigating the unassembled components of the product is a complex task. An algorithm has to be found, which fulfills the above requirement and additionally prevents circular dependencies. Such a dependency occurs, when the PC relations form a loop, where the first parent is child of the last children. This leads to an unstable system.

Using the AND/OR graph, proper relationships can be established quite easily. For each partitioning, one part of a partition is the parent, another component of the second partition the child. Hence, no loop can occur anymore, because when the subassemblies are separated further, only parts of their own partition are considered when establishing a PC relation. Their own parent part is not present. This leads to a 1-1 connection between all exploded partitions. The relationships can be calculated recursively, until partitions cannot be further exploded.

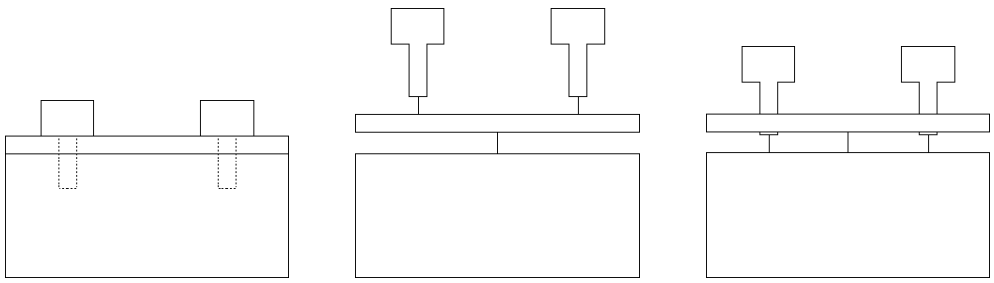
The presented requirement is necessary for generating valid PC relations, but not sufficient. Loops are prevented, but it is possible that an exploded child is exploded again. This happens, when a partition already containing a child is split and the newly established

PC relation creates a second child in the same partition. The partition then is influenced by two explosion operations, which is not tolerable. Therefore, the partition containing the child always has to contain the next parent part, thus being the next parent partition.

An algorithm has to ensure that only 1-1 connections between partitions are established, and that child partitions contain only one child. This boils down to one general condition, which always has to be true for all partitions. All partitions, except the first parent partition, must contain exactly one child. Which parts and partitions are used to establish a PC relation depends on the corresponding style selected by the user. A style is free to determine the parent and child partitions of an AND-branch, until the framework suggests them to enforce the previously defined condition. The selection of single parent and child parts is always performed by the style.

3.3.2 Parent-Child Relations and Collisions

The used sequencing algorithm ensures that removed parts do not collide with any of the still assembled components. But there is no mechanism preventing moving objects from interfering with each other. Such a situation is depicted in figure 3.6, where part 3 collides with the already exploded part 4. The situation can be handled by moving part 4 further along its explosion vector, but this requires the implementation of a collision detection system. Another approach is to prevent the collision at all by adapting the PC relations themselves, thus moving part 4 relative to part 3.



(a) (b) (c)

Figure 3.14: (a) The flat part is attached to the bigger one by two screws. (b) This is the way the explosion should look like. (c) A style may create relations resulting in the collision of the exploded parts.

This thesis utilizes the method of changing PC relations to prevent collisions with already exploded objects. As stated earlier, the relations are output of a PC relation

style selected by the user. After determining all relations, the output is modified, thus overwriting some of the choices of the style. The implemented algorithm prevents collisions of parts, which are exploded into the same direction as already translated objects. This way, key situations like the one in figure 3.14 can be handled. There, a flat part is attached to a bigger one and held in place by two screws. If the `PC` relations are not set appropriately, the small object and the screws will collide.

The used algorithm is limited to handle only single part partitions. This is a simplification, which comes from the implemented AND/OR graph styles. Most of them are peeling styles, which partition the assembly into single parts in each explosion step. In general, the algorithm processes the whole AND/OR graph considering each AND-branch containing a single part partition. Each of these parts is a potential candidate for a parent reassignment. Looking at the previously mentioned example, a screw is such a candidate.

For each candidate, the partitionings of the remaining partition are retrieved from the graph. The partitionings are filtered according to the following criteria. Only those single parts partitions are selected, where the single part is in contact with the candidate part. Mapped to the example, the partitioning into the two rectangular shapes and the second screw is dropped, because the second screw is not in contact with the candidate screw. But the partitioning of the rectangles into single part partitions is considered further. After finding all of these partitionings, they are sorted in descending order according to the number of parts of the remaining partition.

Up to this point, all single part partitionings of the remaining partition not containing the candidate element have been determined. Further, they fulfill the requirement that the single part is in contact with the candidate part. The partitionings are sorted in descending order, depending on the number of parts of the remaining partition. This partitioning list is now processed in-order, applying two simple rules, which check if the single part should be the parent of the candidate part. First, if the single part partition is moving and follows the same explosion direction as the candidate part, it is the new parent. If this criterion does not hold, the second one is applied. If the single part partition stays static and the remaining partition does not move into the same direction as the candidate part, the single part is the new parent. In both cases, the chosen parent part has to be in contact with the moving object. The algorithm processes each partitioning of the list, until a valid parent is found.

The two presented rules are able to handle all of the example cases depicted in figure 3.15. The screw is the candidate part, the remaining partition consists of several parts.

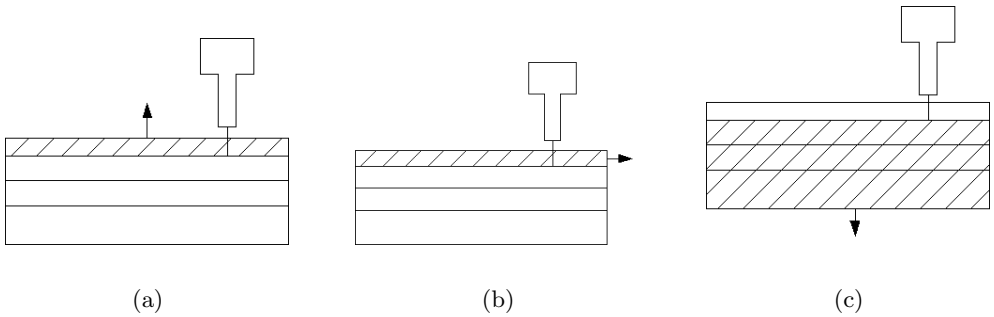


Figure 3.15: The hatched objects are moved. The arrow indicates the movement direction. The screw is the candidate part for a PC relation modification. (a) The exploded part moves into the same direction as the screw. To prevent a collision the PC relation is modified. (b) The exploded part does not move into the same direction as the screw. The PC relation stays the same. (c) The single part stays passive, while the rest is exploded into the opposite direction of the screw. The PC relation is modified to prevent a collision. The screw stays with the passive part.

The partition is split into the single part partition being the top of the part stack and the rest of the subassembly. The hatched components are the moving objects. The arrow indicates the movement direction. In figures 3.15(a) and 3.15(b) the potential new parent moves, while the rest remains static. In the first case the part moves into the same direction as the candidate part. To prevent a collision, the PC relation has to be modified. Hence, the active part is the new parent. In the second case, the active part moves into a different direction, not leading to a collision with the candidate part. The PC relation is maintained. Figure 3.15(c) moves the rest of the partition away from a single static part. If the PC relation stays the same, the candidate part would move with the active partition and thus through the passive part. Therefore, the PC relation is changed and the single passive part is the new parent.

3.3.3 Force-Based Approach

Using a force-based approach for creating explosion diagrams has the advantage, that animations can be created by setting appropriate explosion forces. In addition, part arrangement can be influenced rather easily using additional forces, like a spacing or a viewing force as described in [3].

The focus of this thesis lies on the generation of different representations of an explosion diagram by changing its fundamental parameters. These are the partitioning of

the assembly, the established PC relations, and the separation directions and distances. Altering this parameters leads to a variety of explosions having completely different structures and appearances. The mentioned viewing and spacing forces merely modify the final layout of the diagram. Hence, there are only those forces present, which are required to create a spring-damper system for exploding partitions. Additional forces, like the viewing force, are not implemented. In fact, by using styles it is possible to greatly enhance the explosion diagram in similar ways, without making any changes to the existing explosion engine. In the following, the application of the spring-damping system for creating explosion diagrams is explained.

A spring force tries to move an object back into its original position. It grows stronger, the farther a part is moved. This is known as Hooke's law. The force can be written as

$$\overrightarrow{F_s} = -k\overrightarrow{x},$$

where k is a constant and $\overrightarrow{x}$ denotes a vector from the initial position to the current position of a part. The negative sign orients the force so that the displaced object is moved back to the origin.

By utilizing this force, an explosion can be created. $\overrightarrow{F_s}$ holds the components of the model in their assembled position. When a part is to be exploded, an appropriate explosion force moves it out of the initial position, until both forces are in equilibrium. The problem with this simple approach is that the distance of the explosion cannot be influenced directly, but only by calculating an appropriately powerful explosion force. Doing so requires additional computation time, which can be avoided when letting the spring force act a little bit different. In the unexploded state, a part is forced back into its originally assembled position, just as before. But after the explosion, the spring force keeps the object in the final explosion location determined by the separation direction and distance. In both cases, the current position of the parent part has to be known, so that the exploded part can be held in the same relative location, when the parent itself is translated.

Following this approach also removes the need for an explosion force. To create an explosion only the spring position has to be updated. Nevertheless, another force has to be introduced. Because the whole explosion arrangement is calculated using a physical framework, each part has a mass. When parts are translated to reach their exploded or unexploded positions, this leads to inertia and therefore an infinite oscillation of the part around its final resting place. The solution to this problem is a damping force,

supplementing the spring force. This damping force is defined as

$$\overrightarrow{F_d} = -d\overrightarrow{v},$$

where d denotes a damping constant and $\overrightarrow{v}$ represents the current velocity of the part. By negating this velocity, the movement of the component is damped. Thus, the bigger the velocity, the greater the damping force, slowing the object down. The complete spring-damper system now can be described as

$$\overrightarrow{F} = \overrightarrow{F_s} + \overrightarrow{F_d} = -k\overrightarrow{x} - d\overrightarrow{v}.$$

Care has to be taken when choosing spring and damping constants. The described system is a harmonic oscillator, which can be solved using a differential equation [18]. The solution yields

$$\delta = \frac{d}{2\sqrt{(km)}},$$

which can be used to determine appropriate spring and damping constants. An optimal system is critically damped ($\delta = 1$), which means, that the object is moved back into its original position as fast as possible. In an over-damped system ($\delta > 1$) the objects may be translated very slowly, while under-damping ($\delta < 1$) results in an oscillation of the objects around their resting place. This thesis uses a critically damped system, to speed up the arrangement of the objects.

3.3.4 Interaction and Explosion Styles

The user has the possibility to explode all parts at once, a sequence of layers or just a single part. Furthermore, the creation of the final explosion diagram can be influenced by combining four types of styles. An AND/OR graph style selects a distinct partitioning of the whole assembly, a `PC` relation style determines appropriate `PC` relations, a separation direction style select an explosion vector for each part out of the range of its valid removal directions and a last style influences the ordering of the part removal. This style is also responsible for determining the parts to be removed next, when the user chooses to explode single parts. In addition, it defines the layers used for a layered explosion of the assembly. All styles can be influenced by a user defined focus part.

Styles are a flexible way to adapt the explosion diagram to the current needs. Depending on the focus, an appropriate diagram can be calculated. Section 3.4 further describes

the different styles and presents a range of potentially useful styles.

3.4 Explosion Styles

Four types of styles can be combined to influence the explosion diagram. An AND/OR graph style defines, how the assembly is partitioned. Its output is a distinct path through the AND/OR graph. After having determined all partitions of the model, the `PC` style calculates appropriate `PC` relations. In 3.3.1 was defined, that all partitions except the first one have to contain a child. To ensure that this requirement is preserved, appropriate parent and child partitions are suggested to the style. If none are provided, the style is free to define parent and child partition on its own. Afterwards, the relation is refined to a selection of a parent and a child part out of the two corresponding partitions. The final AND/OR graph and `PC` relations are input to a style calculating separation directions. This ordering of styles is necessary, because it is not possible to determine separation directions without knowledge about parent and child parts. A parent stays static, while the child moves. Similarly `PC` relations cannot be created without knowing the partitioning of the assembly. The final `PC` relation output of all of these three styles is modified according to the collision prevention algorithm described in section 3.3.2.

The fourth style defines an ordering of exploding the partitions. The AND/OR graph contains information on which parts have to be removed before other parts, but it prescribes no ordering of explosion operations. In an AND/OR graph, partitionings of separate AND-branches can be exploded in parallel, which is perfectly suited for an all-part explosion. But when the user wants to explode single parts or layers, there has to be information on which parts out of the set of removable components are exploded next. In case of a degenerate AND/OR graph these considerations are not necessary. A graph is degenerate, if in each step only single parts are removed from the assembly. Hence, each AND-branch consists of a partitioning into a partition of $n \geq 1$ parts and a second partition containing only one part. The ordering is inherently given by walking through the graph, until the last AND-branch contains only single part partitions.

In the following, the styles are presented in the ordering of how the data is calculated. It may become obvious, that different combinations of the styles may create similar explosion diagrams. For instance, an AND/OR graph style creating symmetric explosions, where the same types of parts are removed in a row could also be realized by an explosion ordering style, sorting a previously calculated AND/OR graph. Which style is implemented depends on the complexity of this task. On the other hand, both styles are

implemented if they also can be used to create completely different explosions, aside from the similar output in certain cases. In this example, already sorting in the AND/OR graph style is easier, than doing it afterwards in the explosion ordering style. Nevertheless, such an explosion ordering style may also be applied on a non-degenerate AND/OR graph to explode symmetric parts one after one another if possible, as well as for defining layers containing only symmetric parts. Hence, both styles are interesting.

It should also be noted, that not all of the outlined styles have been implemented. Nevertheless, to provide a thorough description of the potential of the different styles, also the unimplemented ones are mentioned.

3.4.1 AND/OR Graph

An AND/OR graph style selects a path out of the precalculated AND/OR graph spanning the search space of all explosion sequences. The resulting graph prescribes a certain partitioning of the assembly, as well as the ordering of part removal. The ordering is very strict. Starting with the fully assembled model, only one possible partitioning is possible. The two resulting partitions are also partitioned and can be exploded in parallel. A single part partition cannot be partitioned further. Therefore, if the assembly is partitioned into single part partitions, as done by the peeling styles, the resulting graph is degenerate. In each explosion step only one possible partitioning is available. In this case, the ordering is fixed. If the graph is not degenerate, an ordering style can influence the sequence of part removal by selecting one of the AND-branches, which can be processed in parallel.

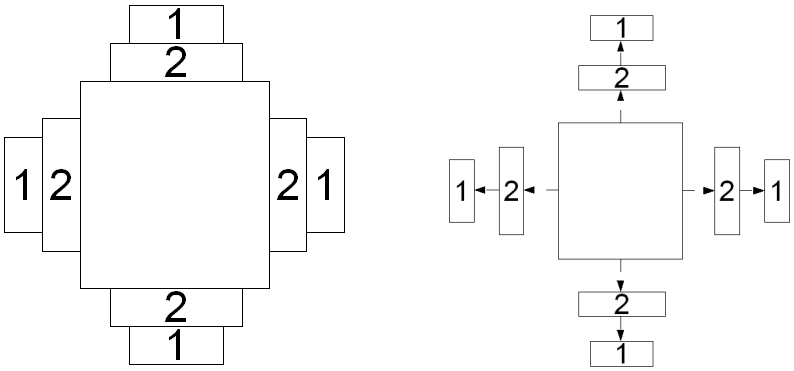
General Peeling

An explosion could peel an assembly by first exploding all of the outermost parts and then working its way inwards. Although an AND/OR graph stores only two-handed partitionings of an assembly, the necessary information is available. A style has to determine all parts, which are removable from the partition. Then the AND-branch removing the first part is retrieved and added to the styles AND/OR graph. In the next step, the second object is removed. Because it was separable from the original partition, it is also removable from the rest of the partition not holding the first part anymore. But the corresponding AND-branch does not necessarily have to exist. There can occur a situation where the removal of a part prevents the separation of a component of the same single part layer, because an invalid unconnected partition would be created. This is the case when a previously removed part and the part to be exploded next formed a connection

between other components. Exploding the first part is valid, the partition is connected by the still unexploded single component. But when this single component also has to be moved, there is no connection anymore. An AND-branch creating such a partition does not exist, hence the object cannot be removed anymore. Taking care of such cases, all objects of a layer can be processed and the next layer of removable parts is determined. This is done until the assembly has been completely disassembled.

Peeling could remove either single parts or whole collections of parts. But the more parts have to be removed, the more complicated is the peeling algorithm. For instance, if sets of two parts are to be exploded away from a partition, there have to be found a maximum number of such disjunct sets for each peeled layer. Figure 3.16 illustrates the idea of peeling single parts.

An appropriate `PC` style can be used so that parts are moved away from the rest of the partition containing $n \geq 1$ objects.

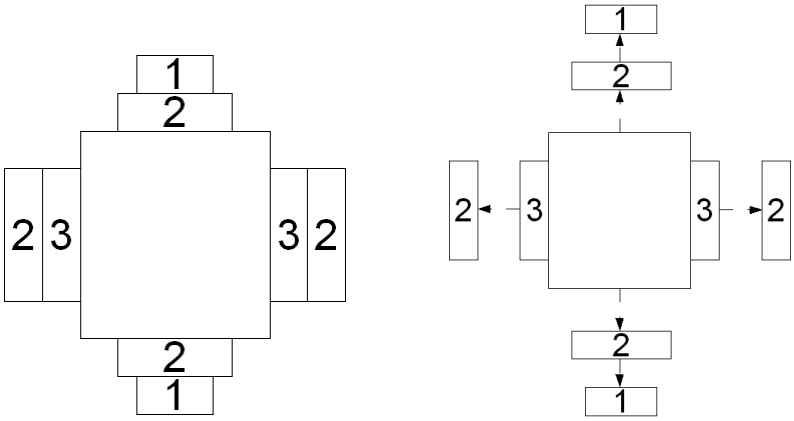


(a) General Peeling (b) Full Explosion

Figure 3.16: (a) Using general peeling, the parts marked with 1 can be removed in the same partitioning step. Afterwards the same is true for parts 2. (b) The full explosion of the assembly. Parts of the same size are moved the same distance.

Symmetric Peeling

The general peeling algorithm, removing all unblocked parts of a partition can be refined to explode only those parts at once, which are of similar size. This way, symmetry can be introduced into an explosion diagram, using only the available geometric data. For instance, a typical car is symmetric. Hence, the wheels can be removed at the same time and are of the same size. This information is sufficient for identifying the wheels



(a) Symmetric Peeling Sequence

(b) Partial Symmetric Explosion

Figure 3.17: The parts are peeled from outside to inside. (a) All parts having the same size are removed in the same step, if they are not blocked. In this case, the smallest parts are preferred. (b) A partial symmetric explosion. Moving parts of the same size the same distance, supports the symmetric appearance.

as symmetric parts. Figure 3.17 depicts symmetric peeling. In this case, the algorithm decided to remove the smallest parts first, although this may not be important. Another implementation could explode the first part it finds. If it has no symmetric counterparts, then only this single component is removed. The symmetric parts are then identified in a later iteration. To support the symmetric appearance, parts of the same symmetric groups should be moved the same distance. Furthermore, the same `PC` relations should be established. This is discussed further when presenting the corresponding styles.

A refinement of the size-based approach is to also consider the number and size of the direct neighbors of all parts, identified using the previous approach. This may prevent wrong identification of parts as being symmetric, which by chance are of similar size.

A similar explosion diagram can be created using an explosion ordering style, sorting the nodes using the same criteria.

Focused Peeling

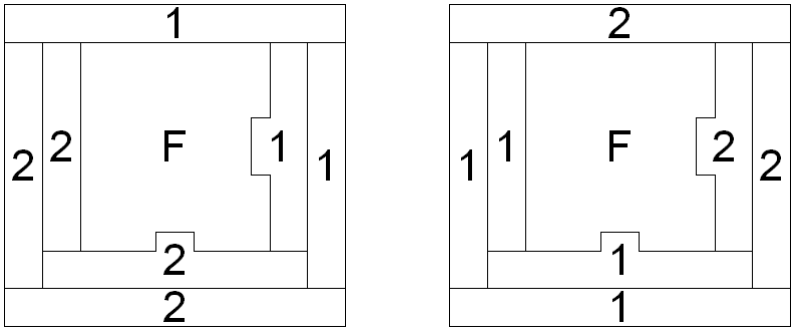
To create a focused peeling, only the parts blocking the focus object are removed, instead of all objects of an outer layer. The necessary information is already stored in the pre-calculated AND/OR graph. It contains only those subassemblies, which can be separated

into any direction. Hence, the path to the partition consisting of the greatest number of parts and containing the unblocked focus element corresponds to the shortest path to reveal this focus (see figure 3.18(a)). The partitionings along this path do not necessarily contain only single parts, but the algorithm can be restricted to consider only single part removal. In this case, depending on the occurrence of interlocking parts, more than one path may have to be examined.

Another approach is to search the AND/OR graph for the partition containing the most parts but **not** the focus as depicted in figure 3.20(b). The found AND-branch is added to the AND/OR graph of the style, the rest of the focus partition is processed further in the same way. This leads to a minimal number of partitions which have to be separated until a partition contains only the focus assembly. In fact, this is the inverse to the operation described in the beginning of this section. As presented in figure 3.18, the first method determines the biggest partition from which the focus element is removable and calculates the path to it. Because the algorithm has to take into account each partitioning of the AND/OR graph to find the required partition, information about the whole graph has to be present. On the other hand, the second algorithm determines the biggest partition, which can be removed from the focus element and steps through the AND/OR graph to create other removable partitions. No knowledge about other partitionings, than the following ones is required. Hence, the second method achieves similar results and is easier to implement, because no search operation has to be applied to the graph. Furthermore, the AND/OR graph is searched top-down, thus is not needed to be generated as a whole.

When calculating the partitioning using the previously described method, still a range of valid separation directions for each part is available. Although the AND/OR graph style cannot directly select an explosion direction for the freed focus part, it is able to restrict the selection by choosing an appropriate partitioning. For instance, if the focus should be removed upwards, the partitioning is performed in a way, that the parts on top of the focus part are removed first.

A simpler method for revealing the focus is to remove those single parts first, which are nearest to the focus part. But this does not guarantee to remove only blocking parts. When the focus element is removable, it is nearest to itself and thus put into a single part partition. The rest of the assembly is processed using the same method, while assuming that the focus element is still unexploded. This is illustrated in figure 3.19.



(a) Shortest-path (b) Biggest partitions

Figure 3.18: (a) The cube marked with F is the focus element. Using shortest path peeling not considering only single parts, the partition marked 1 is removed and unblocks the focus. (b) To free the focus element F, 2 partitions containing the most parts are created. After removing the first one, F still is not removable

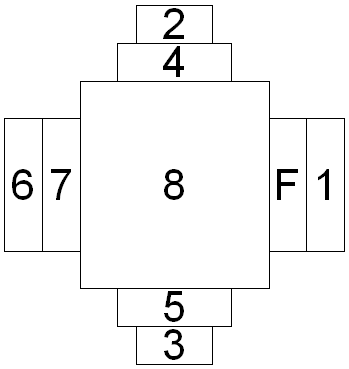


Figure 3.19: The part marked with F is the focus element. The other parts are removed from the focus starting with the nearest first. After removing part 1 the focus is free to move. Then, the rest of the assembly is enumerated using the same method. The decision to remove part 2 before part 3, as well as 4 before 5 is ambiguous.

3.4.2 Parent-Child Relations

Input to the calculation of `PC` relations is the final AND/OR graph calculated in the corresponding style. Creating `PC` relations for parts of an assembly without considering partitioning information could very likely lead to cases, where a partition contains more than one children. As outlined in 3.3.1, this is not permissible. A `PC` relation is established between the two partitions of an AND-branch. To ensure that each partition only contains one child, the explosion framework suggests appropriate parent and child partitions to the

PC style if necessary. If there is no suggestion, the style is free to determine both partition using its internal criteria.

Determining a PC relation consists of two separate tasks. After having established a PC relation between two partitions, the same has to be done on the part level. Hence, this style has works on a partition and a part layer and therefore can independently apply different criteria to both of them. For example, on the partition layer the partition biggest in size is the parent partition, while on the assembly layer, the connection is established between the smallest, contacting assembly parts of both partitions. In general, the PC style is allowed to freely select the parent and child partitions for the first AND-branch and later on only for the partition containing no child, which is the first created parent partition. Nevertheless, the free partition selection can greatly influence the appearance of the explosion diagram.

As mentioned in section 3.3.2, the final PC relation may be overwritten to prevent collisions of already exploded parts. The used algorithm only changes the part level relations. Hence, parent and child partitions will stay the same.

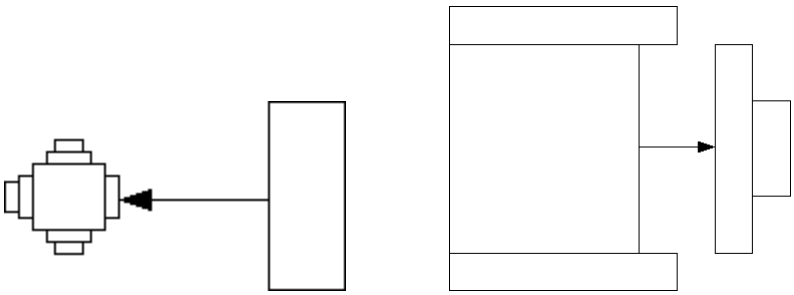
Selecting Partitions

A PC style may select the partition to be the parent, which contains the most number of parts. Hence, partitions containing fewer parts are exploded away from partitions containing more parts. But considering the number of parts may lead to cases, where a subassembly, which is small in size, but contains many parts, stays static, while the huge rest of the assembly is moved. This may not be intended by the user. Therefore, the choice could depend on the spatial extension of partitions (see figure 3.14).

Size-Based Part Level Selection

On the part level, a criterion for a PC relation selection can be the size of parent and child parts. A child may either be assigned to the biggest parent part, or the smallest one, or vice versa. If the input consists of a degenerate AND/OR graph, depending on the chosen parent and child partitions, there only has to be considered a single parent, or a single child part.

If both partitions contain more than one part, this choice becomes more complex. The selection can be limited to only those parts of both partitions, which are in contact in the unexploded state. Now an algorithm can first determine a parent part, or a child part. Or the selection is influenced by both.



(a) Biggest parent partition (b) Biggest parent part

Figure 3.20: (a) The partition bigger in size is chosen as parent part. If the decision would depend on the number of parts, the smaller assembly is the parent, which may be confusing. (b) The child was in contact with three assembly parts, but selected the biggest one to be its parent.

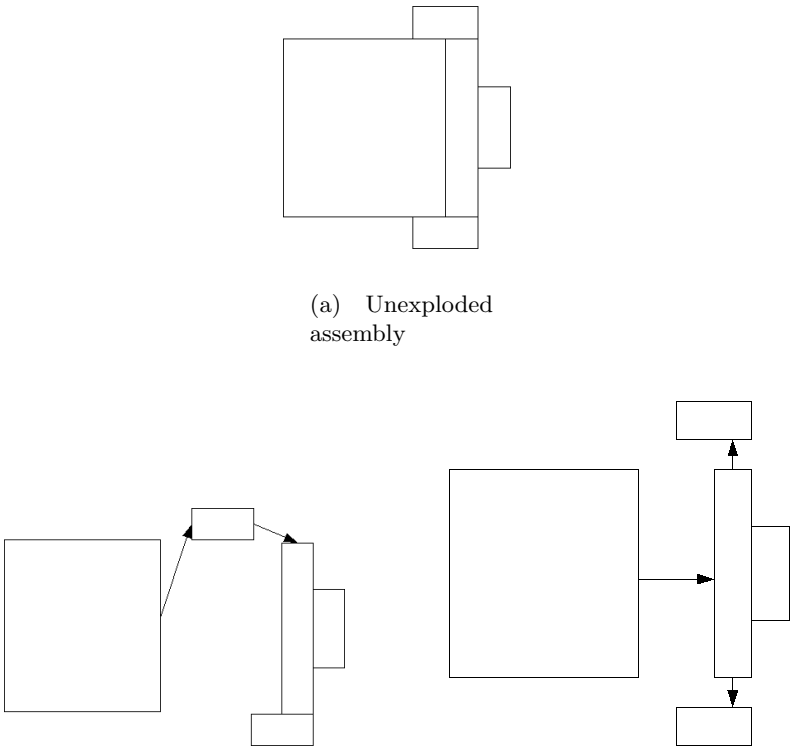
Symmetric Relations

In an explosion diagram of a symmetric assembly, parts having the same context should also have the same parent. For example, if the front wheel on one side of the car is child of a neighboring axle, the parent of the counterpart on the other side should also be an axle. If the wheels are fixed to the same axle, it should be parent to both of them. Therefore, a `PC` style should be able to create such symmetrical relations.

Determining an appropriate relation is a local problem, when reducing the search space to parts in contact. But, the `PC` relation style has to take into account the partitioning of the assembly, output by the previously executed AND/OR graph style. Therefore, the parent could already have been exploded out of the partition. This situation can be prevented by creating an appropriate AND/OR graph, for instance by using symmetric peeling as described in 3.4.1. In this case, the parent part will be contained in each partition until all symmetric parts have been removed. Establishing these preconditions leads to local properties, which are the same for each symmetric part. Hence by applying the same criteria on each of this parts, the same parent can be identified.

Finding symmetric `PC` relations for a degenerate AND/OR graph is different than for a partitioning where both partitions contain several parts. The difficulty is to create an explosion, where the parts of the same symmetric group always move. Selecting wrong relations can result in some symmetric parts moving away from their parent part, and others exploding their parent part. This leads to an asymmetric explosion diagram (see figure 3.21). In the degenerate case, this is not a problem, because only single symmetric

parts are exploded. Hence no other parts can be removed from these components. But if both partitions contain several parts, the PC style algorithm is more complex. It has to select a child part, which allows all symmetric parts to be exploded away from the child partition. If it fails to do so, the partition may explode around a symmetric part. To resolve this situation, there has to be found an asymmetric part in the child partition, resembling the child in the PC relation.



(a) Unexploded assembly

(b) Asymmetric explosion

(c) Symmetric explosion

Figure 3.21: (a) The unexploded assembly. (b) A symmetric part is parent of the other parts of the partition. Therefore, parts are exploded away from it. (c) An asymmetric part was detected and becomes parent part. Symmetric parts are now exploded.

Considering Focus

All of the previously described PC styles, can be influenced by the focus element. For instance, the selection of the parent partition may depend on whether it contains the focus element or not. Because parent partitions stay static, this ensures, that the focus part is never moved. A different style keeps the focus static until it can be separated from

the rest of the assembly, thus exploding it out of its assembled position.

Solving Motion Vector Interferences

The `PC` relation style is the place, where the problems discussed in section 3.1.4 can be solved. As described, the situation in figure 3.10 is resolved by recursively returning all children of the moving part back into the assembled position. The resulting subassembly is tested for interference with the rest of the assembly. If this happens, another parent is selected.

For the case illustrated in figure 3.11, in addition to the created subassembly, the motion vectors of the exploded version have to be tested for collision with the rest of the assembly.

Such `PC` styles are not implemented in this thesis.

3.4.3 Separation Directions

The separation direction style receives information about the whole partitioning of the assembly, as well as the created `PC` relations. These relations tell the style, which objects actually move, and which ones stay static. Without knowledge about this relations, the separation directions and distances cannot be chosen appropriately. A larger part, should be moved farther than a smaller component. Furthermore, partitioning information is equally important. A partition containing several parts should be exploded a greater distance than the single child of the partition would be. On the one hand, the parts exploded away from the partition have more room and may not interfere, with the other partitions. On the other hand, it is necessary to move larger parts far enough, to provide a free look onto parts, which were obstructed in the assembled state.

Separating Parts

The distance should be chosen in a way to separate the exploding part from the rest of the assembly. Thus, after the explosion, the moved partition must not be in contact with the remaining subassembly. To achieve this, the best separation direction is the one, which does not result in a sliding motion of the exploded part on the subassembly it is removed from.

In addition, partitions have to be translated far enough to make room for other parts exploded afterwards. Consider the assembly in figure 3.22, where a partition consisting of a box with a cover is exploded away from a ground plate. Using only a small distance,

the partition quickly escapes the base plate, but in the next step the box is exploded downwards towards base plate, which could lead to a collision. It is clear, that this situation can be prevented by choosing another `PC` style, but it can also be resolved using a separation direction style. Making the explosion distance dependent on the volume of the moving part or another measure related to its size may solve the problem. The partition is translated far enough so that the box does not interfere with the base plate anymore. But because box and cover nearly have the same dimensions, both will also be move a similar distance. The box would nearly be exploded back into its original position. Furthermore, if other parts are attached to the box, an explosion towards the base would lead to a collision. Parts moving upwards from the base will also collide with other exploded parts.

A solution to this problem is to select the explosion distance in a way, that no parts of the moved partition and the static partition collide in their fully exploded state.

(a) (b)

Figure 3.22: A box and its cover are exploded away from a base part. If the `PC` relation is not set appropriately, the box would be moved back, maybe colliding with the base. Hence the distance has to be chosen carefully.

Revealing Hidden Components

As mentioned before, occluded parts should be revealed when partitions are exploded away from the assembly. Because larger parts tend to block the sight more than smaller parts, it is logical to move them a greater distance.

Considering only the volume of a part may not be sufficient. In figure 3.23, the parts on top of the base part have the same size, but are oriented differently. In one case, the part stands on the base, in the other it lies on it. If the distance would depend only on the volume, both objects are translated the same distance, but it is obvious, that the standing part does not occlude the base part by the same amount as the lying one. Hence, the separation distance does not need to be the same.

Occlusion also depends on the position of the viewer. Parts may either occlude more

or less of an object depending on the angle of viewing. Hence the separation distance can be adapted to get a clear view on all parts as done in [21]. For the extreme case, when the assembly of figure 3.23 is viewed from above, changing the distance does not reveal the base part anymore. Hence, the separation direction itself has to be adapted.

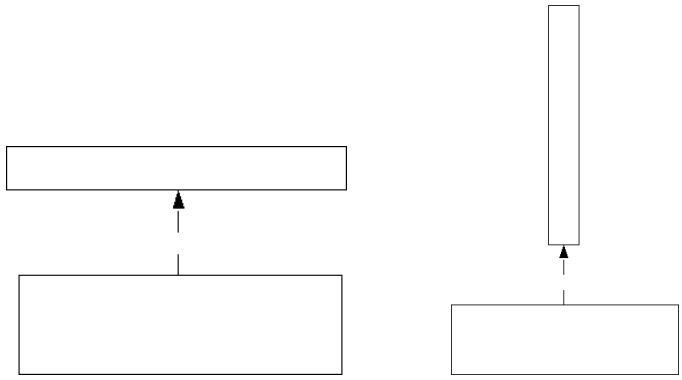


Figure 3.23: (a) Although not required, the standing object is translated by the same distance as the (b) lying cover. Ideally, the distance and direction are chosen to completely reveal the base part, when viewing the assembly from a certain angle.

Introducing Borders

A separation direction style can also be used to limit the extends of the explosion diagram by preventing partitions from being exploded beyond a certain border. Such a style may conflict with the requirement of separating the parts or revealing hidden components.

In AR a user may want to introduce blocking elements into the scene. Hence explosions into the floor or into walls could be prevented. If the style detects such a collision, separation direction and/or distance have to be adapted.

Preventing Sliding Motions

If planar contacts between parts are present, a style can be used to prevent sliding motions of exploding components on the static ones. Considering the case where one part is put on top of another one, a part slides on the static object when the movement vector lies in the contact plane. In the case where only the six main axes are considered as separation directions, four explosion vectors result in a sliding motion, while the fifth separates the

parts. An algorithm can identify this direction as not lying in the plane, spanned by the four other valid motion vectors. Using only three vectors to define the plane is ambiguous, because the required separation direction and two plane vectors, as well as the three plane vectors each span a plane.

3.4.4 Explosion Ordering

The user can either explode the whole assembly at once, only layers of parts or the single parts in a certain ordering. An explosion ordering style is responsible for defining the appropriate layers, and the ordering of how partitions are removed. For a degenerate AND/OR graph an ordering is implicitly given, but layers can still be layers introduced.

Parallel Explosion

If an AND/OR graph is not degenerate, a style could create an ordering which explodes all partitions that can be separated in parallel successively. Furthermore, such partitions can be used to form explosion layers.

Symmetric Layers

Having available the same information as the AND/OR graph style creating a symmetric AND/OR graph, an ordering style can introduce a separation into layers of symmetric parts. The ordering is already given when the graph is degenerate.

Focus Layers

A layer could be defined as the selection of the shortest explosion path until the focus assembly is revealed. Such an algorithm works on each AND/OR graph, degenerate or not.

3.5 Visualization Techniques

An explosion diagram could be hard to read if it contains a great number of parts. Without additional clues, the user may not be able to perceive the spatial extends of the diagram. In addition, it is hard to understand the relationships between the exploded parts. Therefore, the contextual information has to be enhanced. In this thesis the user can choose between two different visualizations, simple leader lines or a static blur, which is derived from a motion blur. Both approaches are described in the following.

Leader lines are drawn between two parts, which are connected to each other via a `PC` relation. The line is drawn from the original position of the child to its current position, instead of connecting it directly to the parent part. If the current position changes, the line is updated appropriately. Because the original location is associated with a parent object, it is also adapted, when the parent moves. Hence the line always follows the movement of parent and child part.

To visualize the movement of an object a motion blur can be used. There, an object is drawn slightly stretched and blurred into the motion direction [10]. This thesis uses a modified version of this technique resulting in a static blur. The trailing blur of the part keeps visible when the motion stops, instead of disappearing completely. Printing such a trail from original position to the finally exploded location can substitute leader lines. Because a static blur illustrates the volume swept by the moved partition, it contains more information about the original location of each contained part, than a simple line.

An additional technique is used to enhance the representation of the assembly in AR. Because the virtual model can be projected over the real world object, it is possible to use this image information to texture the 3D model. Thus, there is no need to manually texture the model.

Chapter 4

Implementation

The system implemented in this thesis utilizes Studierstube [1] to create an explosion diagram in AR. The Studierstube framework employs the Coin3D [2] scene graph library to render 3D scenes. Coin3D is fully compliant to Open Inventor created by NVIDIA. A powerful feature of this library is the possibility to extend its functionality by adding customized scene graph nodes. Hence, the explosion engine is implemented as Coin3D node. In addition, the configuration information required for the generation of an explosion diagram is also determined using scene graph nodes. Taking this approach increased the prototyping time during development, because new configuration methods could be added easily by just exchanging certain nodes of the graph. Furthermore, the rendering capabilities and data structures used by Coin3D could be used to support the configuration step.

User input is handed over to the application by using OpenTracker, which is part of Studierstube. The user is able to explode an assembly as a whole, in layers or as single parts. Furthermore, different types of styles can be combined to change the appearance of the explosion. These styles can be influenced by a user-defined focus part of the assembly. There exist four types of styles, each influencing a different parameter of the explosion. The implemented system is able to exchange each style independent from others.

The parts of the assembly are loaded into the application using a Coin3D wrapper for `trimesh2` [3], which is a C++ library, providing functionality for reading and writing 3D triangle meshes. Each single component has to be located in its final position relative to other parts in the coordinate frame of the whole assembly. The resulting assembly

[1] `http://studierstube.icg.tu-graz.ac.at/`, last visited October 13, 2008
[2] http://www.coin3d.org/, last visited October 13, 2008
[3] http://www.cs.princeton.edu/gfx/proj/trimesh2/, last visited October 13, 2008

must be connected, thus each part has to be in contact with at least one other part. Otherwise, the partitioning algorithm used to create the AND/OR graph would have to be computationally more complex.

The force-based layout is realized by employing the Inventor Physics Simulation API (IPSA) [4], which is a wrapper for the rigid body physical simulation engine `ODE` [5]. Using IPSA, the physical simulation can be built directly into the scene graph. Nevertheless, a few adaptations of the wrapper were necessary to be able to dynamically add and remove objects from the physical world, and to enable the AR tracking of a simulation.

This chapter provides a short description of the physics wrapper IPSA and the changes made to it in section 4.1. Then section 4.2 describes the configuration framework and the methods for automatically generating the required data. The explosion engine is described in section 4.3.

4.1 Physical Simulation

The `ODE` wrapper IPSA is used to simulate a physical world for arranging the exploded parts of the assembly. Before describing the basics of IPSA, a short overview over `ODE` is given. The following information is taken from the official `ODE` user guide [36].

`ODE` creates a physical world, which can contain three different types of objects. A physical object is used to represent an object of the simulation. It is influenced by forces, has a mass assigned to it and holds information about the distribution of its mass around the center of mass. If collision testing is required, a geometrical object (geom) has to be assigned to the physical objects. Such a geom can consists of different predefined geometrical objects, e.g. boxes, spheres or triangle meshes. The third type of component of the physical simulation is the joint type. A joint limits the freedom of a physical object relative to other physical objects. For instance, a hinge joint may be used to create a swinging door.

All of these objects are added to a physical world, which executes update cycles in certain time intervals. The physical objects of the world are influenced by forces, which can be created manually or are the result of collisions or joint constraints. Manipulating a physical object directly is not advisable, because the physical simulation may become unstable.

[4] `http://www.umi.cs.tu-bs.de/full/education/practical/ss2004/vmp/ipsa/docs/index.html`, last visited October 13, 2008

[5] `http://www.ode.org/`, last visited October 13, 2008

IPSA wraps the three objects and the physical world so that a physical simulation can be created easily in an Open Inventor scene graph. In the following, the basic classes are described in section 4.1.1. Then some changes made to the wrapper, as well as the addition of a new joint type for creating the force-based layout are outlined in section 4.1.2.

4.1.1 Basic Classes

Each of the objects wrapped by IPSA is added to an instance of the separator `SoWorldPhysics`, representing the world. Objects in the same world are able to interact with each other. It is responsible for calling the initialization methods of each object so that a physical counterpart in the `ODE` world is created. This is done by starting the simulation with the method `startPhysicWorld`.

When the simulation is running, the world calls the update methods of the contained objects, thus propagating changes further to `ODE`. Such a change may be a force, applied to a physical object, or the change of the world gravity. Although the world class communicates with the `ODE` world, it is not responsible for updating the current positions of the wrapped physical objects. This is done by engines held by the physical objects themselves.

Physical Objects

Physical objects are wrapped by the base kit `SoPhysics` depicted in figure 4.1. Creating an instance of `SoPhysics` also instantiates an engine `SoPhysicsEngine`. The engine is responsible for retrieving the physical position from the `ODE` world and setting the transformation of the corresponding `SoPhysics` object. Therefor, the matrix `ODETransform` is updated at certain time intervals.

To ensure, that the object is only influenced by the `ODE` transformation, all other transformations of the scene graph are negated by the `reset` node of type `SoResetTransform`. It sets the model view matrix back to the identity matrix. Nevertheless, the initial position of the physical object is still retrieved from the scene graph, when the physical world is started. Therefore, a scene can be built by adding `SoPhysics` objects like any other node, but after starting the simulation, transformations contained in the scene graph are not considered anymore. The location of an object is only retrieved from `ODE`.

To be able to actually move geometry using output from the physical simulation, it has to be associated with the physical object. This is done by assigning it to the node

`object`. The node `collisionShape` defines the geom for this object.

Other physical objects are derived from `SoPhysics`. These are `SoPhysicsBox`, `SoPhysicsCappedCylinder`, `SoPhysicsCylinder`, `SoPhysicsSphere` and `SoPhysicsTorus`. The only difference between them is the calculation of the inertia. It is appropriate to the represented form and set in the `ODE` world.

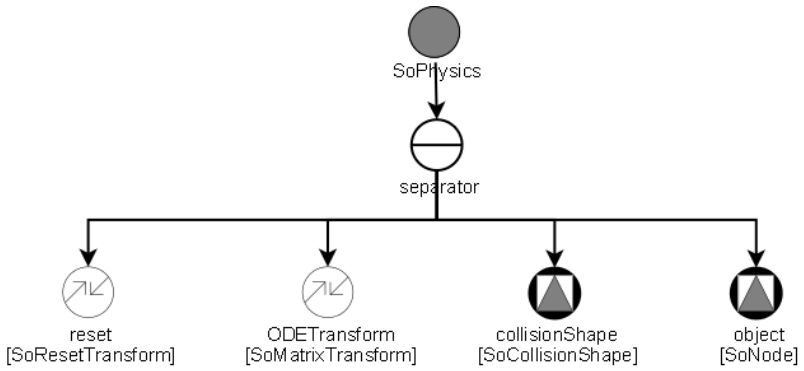


Figure 4.1: Scene diagram of `SoPhysics`

Geometry Objects

As mentioned earlier geoms are required to enable collision testing between physical objects. In IPSA the class `SoCollisionShape` is the base class for all geom encapsulations. By default, the base class creates a cube with side length one, but its subclasses can implement different shapes. Only three of the collision shapes provided by `ODE` are supported by IPSA. These are box (`SoCollisionShapeBox`), sphere (`SoCollisionShapeSphere`)and cylinder (`SoCollisionShapeCylinder`). While the default shape of (`SoCollisionShape`) cannot be changed, the shapes of the subclasses are influenced by user-defined parameters, like the radius of the sphere.

Joints

Joints are used to constrain the movement of two physical objects relative to each other. Hence, to create a joint, two objects of type `SoPhysics` are required. All joint types are derived from the base class `SoJoint`. IPSA supports the `ODE` constraints slider (`SoJointSlider`), universal (`SoJointUniversal`), angular motor (`SoJointAMotor`), ball (`SoJointBall`), fixed (`SoJointFixed`), hinge (`SoJointHinge`) and hinge2 (`SoJointHinge2`). Each subclass implements an interface enabling the user to define appropriate constraint parameters. A hinge may for example be restricted to open only by a certain angle. For

more information on the different constraint types refer to the ODE documentation.

4.1.2 Modifications

To be able to use IPSA in augmented reality the geometry of the physical simulation somehow had to be influenced by tracking information. In the original implementation this was not possible, because of the `reset` nodes of the `SoPhysics` objects. Furthermore, a spring joint was implemented, which is required to arrange the parts of an explosion. The changes made to the original system are outlined in the following.

Tracking

IPSA had to be extended to enable tracking. Applying the transformation directly to objects in the physical simulation was not an option. As mentioned earlier influencing physical objects directly can lead to an unstable physical simulation. Translating the tracking information to forces, influencing the physical world is too complicated. In addition, taking this approach it can take some time until the physical world is arranged according to the tracking data.

In the implemented solution, the tracking information influences only the geometrical positions of the objects and has nothing to do with the physical simulation. IPSA was modified in two steps. First, the new physical world `SoTrackedWorldPhysics` was implemented, which can be connected to a transformation matrix. It holds an instance of the original `SoWorldPhysics` class, responsible for the physical simulation. Furthermore, the class `SoTrackedPhysics` a subclass of `SoPhysics` was created. It contains a matrix, which can be set to the tracking transformation. When added to an instance of `SoTrackedWorldPhysics`, this matrix is connected to the tracking transformation of the world.

Spring Joint

According to the ODE documentation it is possible to create springs using a slider constraint with appropriate parameters. But using ODE joints is not applicable for creating the required force-based layout. Each joint type connects the involved objects together, thus forming a larger object. The problem is that also the masses of the involved parts are added. Assuming that the masses of all objects are equal, moving a single part requires less force, than moving connected objects. Using the same force powers for both cases, would result in different animation speeds when the parts are arranged. While it is possible to

adapt the power of the force according to the collective mass, it is easier to implement a simple spring-damper system as described in 3.3.3.

Following this approach lead to the new joint `SoJointSpring`, which preserves the relative positions of the objects without adding their masses. The mass of each object is set to one. Hence, using equal spring and damper constants results in the same animation speed for each part. The spring receives two objects of type `SoPhysics` as input and keeps the second object, which is meant to be the child, relative to the first one. The positions of the objects are retrieved directly from the physical world. Thus, tracking information and other transformations are not considered.

The spring holds the child either in its initial or in the exploded position. The exploded position is calculated using a given vector and length. The vector has to be supplied in the physical coordinate frame.

To get the spring and damping forces into the simulation, both are set directly at the corresponding `SoPhysics` objects. During an update cycle of `SoWorldPhysics`, the forces are sent to the `ODE` world. To be able to update the spring-damper forces according the position changes of the objects, an update method for each spring has to be called. Because the updates for the built-in joint are handled by `ODE`, there was no update cycle for joints available in `SoWorldPhysics`. Hence, a minor change had to be made to the world class and the `SoJoint` base class. Now update methods for joints are called in each update cycle, thus also enabling the recalculation of the spring forces.

Adding Physical Objects at Run-Time

Creating new objects is essential for the implemented system. Each time, a style is changed, all physical objects are created anew to incorporate the new partitionings and connections, as well as directions. In the original `SoWorldPhysics` implementation the addition of physical objects to a running world was not supported. Although, the new objects were initialized correctly, at some point in time the frame-rate broke in. The reason for this was the creation of a new update timers each time the world was restarted. An update timer is respondible for triggering the update cycle of the world. The problem was solved by instantiating only one timer.

4.2 Configuration

The configuration consists of two parts. In the first part, geometrical information is specified. It consists of the geometry of the single assembly parts, contact information and valid directions for each pair of parts. The second part consists of the AND/OR graph containing all possible partitionings of the input assembly, as well as the associated `PC` relations and separation directions and distances for each partition.

The configuration framework is implemented as scene graph, thus utilizing Open Inventor functionality. This approach has the advantage that a configuration file can be written in a readable format using the existing Open Inventor syntax. In addition, functionality for reading and writing scene graph files is already present. One implementation of a configuration step can be exchanged rather easily by inserting a different scene graph node.

The whole configuration is collected in the base kit `SoExplosionConfiguration`, which provides an interface returning the abstract configuration interfaces for retrieving geometrical information (`SoAbstractGeometricalInfo`), as well as the AND/OR graph information (`SoAbstractAndOrGraphConfiguration`). These interfaces by themselves return abstract interfaces for the different components of the configuration, namely `SoAbstractContacts`, `SoAbstractValidDirs`, `SoAbstractAndOrGraph`, `SoAbstractParentChildRelations` and `SoAbstractSeparationDirections`. The specifics of the configuration algorithms are left to the implementations of these abstract interfaces. An implementation could allow manual definitions, while others calculate information automatically. The current framework supports both approaches. In a first phase of development, methods for the manual definition of the required information were implemented, thus also creating a basic configuration structure for contact information, valid directions pairs, AND/OR graph, `PC` relations and separation directions. During the coarse of the development, the manual specifications were gradually replaced by automatic calculations, building on top of the structure of the manual definition.

In general, the configuration is performed in the `GLRender` methods of the nodes, which are called when the scene graph is traversed. To reduce the size of the scene graph in memory, already initialized configuration nodes are removed from the graph and stored in the corresponding nodes. Without taking this approach, the performance of the scene graph traversal would depend on the amount of the configuration information. For instance, an AND/OR graph of a 19 part model could massively reduce the frames per second (`FPS`).

Listing 4.1: Example of SoExplosionConfiguration

```
#Inventor V2.1 ascii
SoExplosionConfiguration {
  writeConfiguration TRUE
  geometricalInfo
  DEF GEOMETRICAL_INFO SoGeometricalInfo {

    geometry DEF INPUT_GEOMETRY
    NodeKitListPart { # definition of geometry # }

    validDirections DEF DIRECTIONS
    So3DOFCalculatorValidDirs {
      testDirections [ 0 0 1,
          0 0 -1,
          0 1 0,
          0 -1 0,
          1 0 0,
          -1 0 0 ]
      geometry
      USE INPUT_GEOMETRY
    }

    contacts DEF CONTACTS
    SoGeometricalContacts {
      geometry
      USE INPUT_GEOMETRY
    }
  }
  andOrGraph SoAndOrGraphConfiguration {
    andOrGraph DEF AND_OR_GRAPH SoCalcOnRequestAndOrGraphDFSSplitter
    {
      geometricalInfo USE GEOMETRICAL_INFO
    }
    parentChildRelations SoCalcOnRequestParentChildRelations
    {
      contacts USE CONTACTS
      andOrGraph USE AND_OR_GRAPH
    }
    separationDirections SoCalcOnRequestSeparationDirections
    {
      contacts USE CONTACTS
      validDirections USE DIRECTIONS
      andOrGraph USE AND_OR_GRAPH
    }
  }
}
```

Listing 4.1 shows an example of an explosion configuration file, where all data is calculated. The geometrical definitions are left out to reduce the size of the shown file. The listing illustrates the input required for each configuration step. The calculation of valid directions and contact information only depends on the geometry of the assembly, while the AND/OR graph uses the whole geometrical information. The calculation of `PC` relations and separation directions finally need the determined AND/OR graph and either the contact information or the valid direction pairs, respectively. While traversing the scene graph, all configuration nodes are initialized, depending on their implementation. If a node is input to another one, it has to be initialized before it can be used. Hence, several iterations of the scene may be required.

The field `writeConfiguration` is set to `TRUE`, so the final configuration is written to a file. In this example case only the geometrical information is calculated and written to file. The AND/OR graph configuration is determined in the running program, when requested by the application. Calculating certain information only when it is actually needed reduces the computational time of the configuration step, as well as the configuration file size. This notion will be discussed in more detail, in the following sections and relates to the usage of the different types of explosion styles.

The description of the configuration implementation is separated into two parts. In section 4.2.1, the specification of the assembly parts, the contact information and valid direction pairs is outlined. Section 4.2.2 describes the configuration interface used to retrieve and calculate the AND/OR graph, `PC` relations and separation directions. Some performance measures of the configuration step are provided in section 4.2.3, while section 4.2.4 presents data about the configuration file size.

4.2.1 Geometrical Information

The abstract interface `SoAbstractGeometricalInfo` is implemented by `SoGeometricalInfo`. It contains the geometrical definition of the assembly, the contact information and the valid direction pairs in the nodes `geometry` of type `SoNodeKitListPart`, `contacts` of type `SoAbstractContacts` and `validDirections` of type `SoAbstractValidDirs`, respectively.

Assigning geometry to an `SoNodeKitListPart` is straight-forward, and will not be considered further. As mentioned before, the geometry has to be split into the single parts, which can move separately. Furthermore, their coordinates have to be transformed into the final absolute positions in the coordinate frame of the assembly. Listing

Listing 4.2: Definition of geometrical objects

```
1  geometricalInfo
2  DEF GEOMETRICAL_INFO+1 SoGeometricalInfo {
3    geometry
4    DEF INPUT_GEOMETRY NodeKitListPart {
5      containerTypeName "Separator"
6      childTypeNames "SoAssemblyGeometry"
7      containerNode
8      Separator {
9
10       DEF floorPlate+3 SoAssemblyGeometry {
11         filename "floorPlate.obj" =
12         Concatenate {
13           type "MFString"
14           input0 "floorPlate.obj"
15
16         }          . output
17         meshtype TRIMESH
18         faceBBox TRUE
19         name "floorPlate"
20       }
21       DEF steeringWheel+4 SoAssemblyGeometry {
22         filename "steeringWheel.obj" =
23         Concatenate {
24           type "MFString"
25           input0 "steeringWheel.obj"
26
27         }          . output
28         meshtype TRIMESH
29         faceBBox TRUE
30         name "steeringWheel"
31       }
```

4.2 shows part of the definition of the geometrical parts as `SoAssemblyGeometry` nodes. `SoAssemblyGeometry` derives from a class wrapping the `trimesh2` data structure, so that it can be used in Open Inventor. In fact, the wrapper is only used for finding contacts between the parts. It cannot be used for rendering, because a bug in `trimesh2` causes a wrong calculation of normals. Therefore, the rendered geometry is loaded from an Open Inventor file containing the correct normals.

As mentioned before, the implementations of the configuration nodes used to define contact information and valid direction pairs follow the same approach. First, a node allowing the manual assignment of the required data was developed. The defined structure then was utilized by nodes, which automatically calculate the relevant information. The final geometrical information is written to file, with the rest of the data.

Listing 4.3: Manual contact definition

```
1  DEF CONTACTS+10 SoManualContacts {
2    contacts
3    NodeKitListPart {
4      containerTypeName "Separator"
5      childTypeNames "SoContact"
6      containerNode
7      Separator {
8        SoContact {
9          partOne
10         USE floorPlate+3
11         partTwo
12         USE steeringAxle+5
13       }
14       SoContact {
15         partOne
16         USE floorPlate+3
17         partTwo
18         USE frontLights+6
19       }
```

If some of the calculated values do not fit the requirements, they can be changed in the final configuration file. This is perhaps necessary, because different explosion directions than the calculated ones should be chosen for certain pairs of parts. After changing any geometrical information, the whole AND/OR graph configuration has to be recalculated, because the partitioning of the assembly depends on the contact information, as well as the valid directions of the pairs.

Contact Information

All configuration nodes used to retrieve contact information must implement the abstract interface `SoAbstractContacts`. Using instances of `SoAssemblyGeometry`, the interface can be queried for contact information about a certain pair of parts. In addition, all contacts can be returned at once. For this purpose, an `SoContact` node was defined, which references two assembly parts in contact. Hence, for each pair in contact a separate `SoContact` node is created. Furthermore, the interface returns an undirected contact graph, which can be used by the partitioning algorithm of the AND/OR graph.

The class `SoManualContacts` is a simple implementation of `SoAbstractContacts`, which allows a manual definition of the contact information. The user manually creates `SoContact` nodes for each part pair in contact and adds them to an `SoNodeKitListPart`. Listing 4.3 shows an example of such a manual definition. To reduce the file size, it is

sufficient to define a contact only for one combination of a part pairing. The inverse is determined by inverting the contact containing both parts. This is done in the initialization phase of the contact information.

Using the `trimesh2` data structure, contact information is calculated automatically in the node `SoGeometricalContacts`, a subclass of `SoManualContacts`. Each pair of parts is tested for a possible contact. Because the contact for the inverse pair can be created by inverting the contact of the original pair, only one combination of part pairs is considered. Before actually testing for a contact, a simple check determines whether a contact is possible or not. This test utilizes the bounding boxes of the parts. Only if the bounding boxes interfere or touch, the parts are considered further. In this case, each face bounding box of one part is tested for a collision with a face bounding box of the other part. The face bounding box is part of the `trimesh2` library. When a collision is detected, a contact is established by creating an instance of `SoContact` referencing the involved parts.

Valid Direction Pairs

The abstract interface `SoAbstractValidDirs` returns all valid direction pairs for a single part, or only the valid directions for a certain pairing. The returned data structure is `SoValidDirectionPair`, which contains references to the two involved assembly parts and a range of valid directions. One part of the pair is the active one, hence the moved one, while the second one stays passive, or static.

Ideally, the active part is allowed to move to infinity into each of the provided directions, without colliding with the passive part. But there exist cases where a part is blocked completely, maybe because it is elastic and has to be squeezed to be removed. Because this thesis assumes only rigid parts, such cases have to be handled by providing a direction, although parts will collide. A part also cannot be removed collision free, if its assembly requires more than a single translation or even rotations.

Analogous to the contact configuration, the valid direction pairs can be specified manually using the node `SoManualValidDirs`. Again, only one part pair has to be specified. The data for the inverse pair is determined by inverting the corresponding `SoValidDirection-Pair`. Listing 4.4 shows an excerpt of a manual definition.

The direction pairs are calculated automatically by node `So3DOFCalcualtorValidDirs` derived from `SoManualValidDirs`. It employs a method presented in [17], which makes use of graphics hardware to check global blocking constraints. The algorithm the uses

Listing 4.4: Manual valid direction pair definition

```
1   DEF DIRECTIONS+9 SoManualValidDirs {
2     validDirectionPairs
3     NodeKitListPart {
4       containerTypeName "Separator"
5       childTypeNames "SoValidDirectionPair"
6       containerNode
7       Separator {
8         SoValidDirectionPair {
9           directions [ 0 0 -1,
10                0 1 0,
11                0 -1 0,
12                -1 0 0 ]
13          activePart
14          USE floorPlate+3
15          passivePart
16          USE steeringWheel+4
17        }
18        SoValidDirectionPair {
19          directions -1 0 0
20          activePart
21          USE floorPlate+3
22          passivePart
23          USE steeringAxle+5
24        }
```

z-buffer to detect collisions when moving the active part into a certain direction. First of all, the z-buffer is cleared to 1 and the viewpoint is changed to view into the direction of part removal. The depth function is set to *less-than* and the active part is rendered in red. Then, the depth function is set to *greater-than* and the passive part is rendered in a different color, in green. Using the *greater-than* function, the passive part is only rendered in places, where it lies before the active object. Hence, if the number of green pixels is greater than zero, a collision occurs in that particular direction.

The original method was enhanced by using back face culling, to handle 3D assemblies, where the single parts overlap by a small distance. In this case, only the front facing geometry of both active and passive part are rendered. When parts overlap, the back facing geometry of the passive part lies in front of the front facing geometry of the active part. This situation is handled, thus the active part is not blocked anymore. The render test was implemented using `SoOffscreenRenderer`.

By default the six main frame directions are tested, but the user can change the checked directions in the configuration file. Testing only a limited number of directions raises the problem that parts cannot be removed at all, because they are blocked in each

direction. This problem became apparent when exploding the LEGO© Technic car. Although parts are assembled using mainly one of the six main axes directions, not all parts could be removed anymore. This came from the small rotation of the final assembly. For instance, the front lights could not be removed anymore in a straight forward motion, because some of the pins characteristic to LEGO© Technic parts blocked its motion. To ensure, that at least one valid direction is returned by the calculation, the one having the least number of green pixels, indicating a collision with the passive part, is chosen.

Taking this approach, the least blocked direction is always chosen, which in case of the car worked in most cases. Only steering wheel and steering axle yielded a wrong valid direction. The geometry of the wheel is fully blocked by the steering axle, which comes from the pre-partitioning into groups of parts. Nevertheless, even removing the corresponding geometry as single parts does not work because multiple translations would be required to remove the axle from the steering wheel. Hence, the resulting valid direction was corrected manually in the final configuration file, so that the steering wheel exploded upwards instead of sideways.

4.2.2 AND/OR Graph

Similar to `SoAbstractGeometricalInfo`, the interface `SoAbstractAndOrGraph-Configuration` returns abstract interfaces to access the configuration information of the AND/OR graph. These are `SoAbstractAndOrGraph` used to query the defined AND/OR graph containing the partitioning of the assembly, `SoAbstractParentChildRelations` returning `PC` relations for each partitioning and `SoAbstractSeparationDirections` holding information about the possible separation directions. `SoAndOrGraphConfiguration` implements `SoAbstractAndOrGraphConfiguration` and defines public nodes holding the implementations of the interfaces.

Again, the interfaces are implemented gradually, starting with manual implementations, which are the foundation for nodes performing the configuration automatically. While the smallest part holding geometry was `SoAssemblyGeometry` during the geometrical configuration described previously, now it is `SoPartition`. `SoPartition` is an `SoNodeKitListPart` being able to hold several references to `SoAssemblyGeometry` instances. Furthermore it provides convenience methods for checking if a certain assembly part is contained and if the partition is geometrically equal to another partition.

AND/OR Graph

`SoAbstractAndOrGraph` is used to query the AND/OR graph containing the partitioning of the assembly. The interface returns all possible partitionings of a partition as a set of `SoAndBranch` nodes contained in an `SoOrBranch` node. Although the used AND-branch data structure can contain several partitions at once, the current implementation of the system only handles partitionings into two connected partitions. As mentioned earlier in this thesis, this is a necessary requirement to reduce the computational complexity of splitting an assembly. In addition, all `SoOrBranch` nodes of the graph can be returned at once, to be able to search for certain partitions. Nevertheless, there exists an implementation, which does not support this functionality. It will be outlined later on. The root of the AND/OR graph is the `SoOrBranch` containing the partitioning information of the partition holding the whole assembly geometry. It can be queried separately.

Listing 4.5: Example of manual AND/OR graph definition

```
1  DEF +11 SoAndOrGraphConfiguration {
2    andOrGraph
3    DEF AND_OR_GRAPH+12 SoManualAndOrGraph {
4      orBranches
5      NodeKitListPart {
6        containerTypeName "Separator"
7        childTypeNames "SoOrBranch"
8        containerNode
9        Separator {
10
11         SoOrBranch {
12           mainPartition
13           SoPartition {
14             containerTypeName "Separator"
15             childTypeNames "SoAssemblyGeometry"
16             containerNode
17             Separator {
18
19               USE floorPlate+3
20               USE steeringWheel+4
21               USE steeringAxle+5
22               USE frontLights+6
23               USE tyreRF+7
24               USE tyreLF+8
25             }
```

```
26
27              }
28              andBranches
29              NodeKitListPart {
30                containerTypeName "Separator"
31                childTypeNames "SoAndBranch"
32                containerNode
33                Separator {
34
35                  SoAndBranch {
36                    containerTypeName "Separator"
37                    childTypeNames "SoPartition"
38                    containerNode
39                    Separator {
40                      DEF +13 SoPartition {
41                        containerTypeName "Separator"
42                        childTypeNames "SoAssemblyGeometry"
43                        containerNode
44                        Separator {
45                          USE floorPlate+3
46                          USE frontLights+6
47                          USE tyreRF+7
48                          USE steeringWheel+4
49                          USE steeringAxle+5
50                        }
51                      }
52                      DEF +14 SoPartition {
53                        containerTypeName "Separator"
54                        childTypeNames "SoAssemblyGeometry"
55                        containerNode
56                        Separator {
57                          USE tyreLF+8
58                        }
59                      }
60                    }
61                  }
62                  SoAndBranch {
```

`SoManualAndOrGraph` implements `SoAbstractAndOrGraph` and allows a manual definition of the whole AND/OR graph. Defining the whole AND/OR graph of an assembly manually is a very time-consuming and error prone task. Therefore, the manual implementation is rather an encapsulation of the relevant data structures, which can be reused

by subclasses, which calculate the graph. Listing 4.5 shows a small part of an example configuration of the AND/OR graph. It also visualizes the `SoOrBranch` and `SoAndBranch` nodes, which are returned by the interface.

When calculating the AND/OR graph, two steps have to be performed. First, all possible connected partitions of an input partition have to be found. Second, the partitions have to be tested for separability. Each partitioning passing this test is added to the AND/OR graph. `SoAbstractCalculateAndOrGraph` is a subclass of `SoManualAndOrGraph` and provides general functionality for calculating the whole graph. When implementing this class only the definition of the algorithm returning all possible connected partitions of a graph is necessary. `SoAbstractCalculateAndOrGraph` checks the global separability of the partitions by finding valid directions common to all parts of a partition. If a partition is not separable, local separability is tested by considering only the valid direction pairs for the parts in contact between both partitions. As described in section 3.2.1, this form of local freedom does not equal true local freedom. To reduce the required memory and the file size, as well as to prevent ambiguities, geometrically equal partitions are identified and represented by the same `SoPartition` instance.

`SoCalculateAndOrGraphDFSSplitter` derives from `SoAbstractCalculateAndOrGraph` and implements a DFS on the undirected contact graph to find all connected partitions. The same approach was taken by Thomas et al. [38]. The splitter uses the undirected DFS contained in the Boost graph library. The library provides means to specify an own method, which is called each time a new node is visited. This facility is used to check if the unvisited graph is connected. If this is true, the partition is separated into the currently visited nodes and the rest of the graph, and added to the possible partitionings. It is sufficient to check if the rest is connected, because the search itself already created a connected graph. In fact, the DFS does not find all possible partitionings, because it does not reconsider any already visited nodes. Hence, to increase the number of found partitionings, the DFS is called with each node as starting point. This approach ensures, that if at least one of the parts of a partition is separable, it will be found. Theoretically, it could happen that the algorithm will not find any valid partitionings, if the parts of the assembly interlock and can be removed only as groups.

Finding all partitionings may be very time consuming depending on the number of parts of the assembly. In addition, only a small part of the possible partitionings will be used by the final explosion styles. Hence, if a small lag in the running application is acceptable, the partitioning of a certain subassembly can be done, when it is requested by

the application. For this purpose, SoCalcOnRequestAndOrGraphDFSSplitter was implemented. Instead of creating the whole AND/OR graph in the configuration step, it only determines all possible partitionings of the partition, for which the SoOrBranch was requested at runtime. Because no AND/OR graph is calculated, it is not possible to request all SoOrBranch nodes at once.

Parent-Child Relations

The interface SoAbstractParentChildRelations returns all possible PC relations of a certain partitioning, which is identified by the corresponding SoAndBranch. All potential PC relations between two partitions are the set of parts establishing a contact between the two partitions, in the unexploded state. Depending on the number of partitions in each SoAndBranch, a certain number of SoParentChildRelation nodes encapsulating the PC relations is returned. This data structure holds information on which of the two partitions is the parent, and which one the child. Because either of the partitions can be the parent, two SoParentChildRelation nodes exist for each AND-branch. The node stores all contacts between the partitions in SoPCElement nodes, containing references to the corresponding SoAssemblyGeometry nodes. The user can refine the query for a PC relation by specifying a parent partition. In this case only one reference to an SoParentChildRelation is returned.

SoManualParentChildRelations implements the abstract interface and allows the manual definition of PC relation according to the previously described scheme. The helper class SoAndBranchParentChildRelations was implemented, which holds all SoParentChildRelation nodes of a certain AND-branch. The configuration references to SoAndBranch and SoPartition nodes of the calculated AND/OR graph. It is sufficient to define the PC relation only for one combination of parent and child partitions. The inverse combination is calculated from the specified information. Listing 4.6 visualizes the usage of the data structures.

The node SoCalculateParentChildRelations derives from SoManual-ParentChildRelations and calculates the PC relations using the AND-branches defined by the AND/OR graph configuration and the contact information of the geometrical configuration. An algorithm simply determines the parts in contact between the two partitions. To speed up the configuration and to reduce the file size, SoCalcOnRequestParentChildRelations was implemented. It creates the PC relations on request. This involves only a simple search operation in the contact information.

Listing 4.6: Example of manual `PC` relation definition

```
SoManualParentChildRelations {
  parentChildRelations
  NodeKitListPart {
    containerTypeName "Separator"
    childTypeNames "SoAndBranchParentChildRelations"
    containerNode
    Separator {

      SoAndBranchParentChildRelations {
        andBranch
        USE AND_BRANCH_1
        parentChildRelations
        NodeKitListPart {
          containerTypeName "Separator"
          childTypeNames "SoParentChildRelations"
          containerNode
          Separator {

            SoParentChildRelations {
              parentPartition USE PARTITION_1
              childPartition  USE PARTITION_2
              pcElements
              NodeKitListPart {
                containerTypeName "Separator"
                childTypeNames "SoPCElement"
                containerNode
                Separator {

                  SoPCElement {
                    parentAssembly
                    USE valve_bolt_fixture+8
                    childAssembly
                    USE valve_basebolt_right+6
                  }
                  SoPCElement {
                    parentAssembly
                    USE valve_stabilizer_plate+14
                    childAssembly
                    USE valve_basebolt_right+6
                  }
```

Separation Directions

The separation direction interface `SoAbstractSeparationDirections` returns the possible separation directions of a partitioning as `SoSeparationDirections` nodes. These nodes contain information on which partition is active, thus moving, and which one is passive, or static. Furthermore, all valid explosion directions of the active part are provided,

Listing 4.7: Example of manual separation direction definition

```
SoCalculateSeparationDirections {
 separationDirections
  NodeKitListPart {
    containerTypeName "Separator"
    childTypeNames "SoAndBranchSeparationDirections"
    containerNode
    Separator {

      SoAndBranchSeparationDirections {
        andBranch
        USE ANDBRANCH_1
        separationDirections
        NodeKitListPart {
          containerTypeName "Separator"
          childTypeNames "SoSeparationDirections"
          containerNode
          Separator {

            SoSeparationDirections {
              defaultDirection 0
              length 0.33508578
              directions -1 0 0
              activePartition USE PARTITION_1
              passivePartition USE PARTITON_2
            }
          }
        }
      }
```

as well as a default direction and distance. Because only two partitions are contained in each `SoAndBranch`, there exist only two corresponding `SoSeparationDirections` nodes for each branch.

`SoManualSeparationDirections` implements this interface and allows the manual specification of separation directions. As with the other configuration nodes, it is sufficient to define only one instance of `SoSeparationDirections` for each pair of active and passive partitions. The information for the inverse pair is created by inverting the corresponding node. Similar to `SoManualParentChildRelations`, a helper class is implemented, which holds all `SoSeparationDirections` instances for a certain AND-branch. Listing 4.7 presents an example for defining separation directions. The `SoPartition` and `SoAndBranch` nodes are referenced from the previously determined AND/OR graph.

The separation directions can be calculated using the information of the previously calculated AND/OR graph and the valid pair directions of the geometrical information,

Configuration Step	LEGO© Technic car $n = 19$, $v = 44428$	valve $n = 14$, $v = 31435$
Contacts	27 sec	17 sec
Valid Directions	104 sec	53 sec
AND/OR graph	705 sec	38 sec
`PC` Relations	81 sec	8 sec
Separation Directions	224 sec	15 sec

Table 4.1: The table contains the time required for each configuration step. n is the number of parts, v the number of vertices.

as done in the configuration node `SoCalculateSeparationDirections`. It derives from `SoManualSeparationDirections` to be able to use its data structure. Separation directions for each partition pair of an `SoAndBranch` are determined by finding the common directions of each part of the active partition. For this purpose, each pair of parts formed by objects of both partitions is considered, thus retrieving the corresponding valid direction pairs from the geometrical information. The common directions of all direction pairs of the partitions are the valid separation directions. The separation directions can be created on request using the configuration node `SoCalcOnRequestSeparationDirections`. Hence, the final configuration file size and the required memory is reduced.

4.2.3 Performance

Before discussing the performance of the configuration step it should be noted that the algorithms are not optimized, but rather straightforward implementations. Optimization was not that important, because calculating configuration information is a preprocessing step, which does not have any influence on the running system. Nevertheless, moving some of the previously mentioned configuration steps into the running program actually has an impact on the system. But it is negligible for the used test assemblies. Using assemblies consisting of more parts, this issue has to be reinvestigated.

Table 4.1, shows the time required for each configuration step. The used computer contained an Intel Pentium 4 CPU, 1`GB` RAM and an NVIDIA GeForce 7800 GS with 256`MB`. The application was run on Windows XP with Service Pack 3 installed. The calculation of contacts and valid direction pairs has a runtime of $O(\frac{n(n+1)}{2})$ for n parts, because the utilized algorithms only check one combination of a pair. Hence, after considering part pair (p_i, p_j), the data for the inverse pair (p_j, p_i) is created by inverting the information of the original pair.

The contact finding algorithm incorporates an early termination algorithm. A first

check discards pairs from further testing, if the bounding boxes of the pair of parts do not interfere. Otherwise, each face bounding box f_i of one part is checked for a collision with a face bounding box f_j of the second part. This leads to a runtime of $O(f_i f_j)$. After detecting the first contact, the algorithm moves on to the next pair. The overall runtime for the contact finding algorithm is $O(n^2 f^2)$.

Valid pair directions are determined by checking if two parts collide for any of the defined potential separation vectors d. If a collision occurs, the motion of a part of the pair is blocked by the other part. For each tested direction, all blocked pixel have to be counted. Hence, the performance also depends on the number of pixels p. The test is performed for each direction, resulting in a runtime of $O(n^2 dp)$.

The AND/OR graph is calculated using the DFS of the Boost graph library on the undirected contact graph. Hence the performance of the partitioning algorithm is proportional to the number of parts n and the number of contacts c. The DFS is executed with each possible node as starting point, to find all partitions. It is known from literature [7] that the runtime of a DFS is $O(n + c)$, because each node as well as each edge is visited. Calling it n times then yields $O((n + c)n)$.

The algorithm calculating the `PC` relations for each partitioning, finds the contact for each pair (p_i, p_j) of the two involved partitions. When determining common separation directions of the parts of a partition, again each pair has to be considered, as well as each of the valid pair directions. The number of `PC` relations and separation directions depends on the size of the AND/OR graph. For each AND-branch there has to be at least one `PC` relation and one separation direction.

4.2.4 File Size

Open Inventor nodes were implemented, which performed the necessary configuration calculations. In an early stage of the development, the whole configuration data was written to hard disk, which could lead to a file of considerable size. Reading and writing such a file took several minutes. Therefore, only contact information, pairwise separation directions and the AND/OR graph were calculated and stored. `PC` relations and separation directions of partitions were determined if requested by the corresponding style. Doing so did not have any recognizable performance impact in the running program. The configuration file size was reduced further by also calculating the partitionings of the AND/OR graph only if required. Determining this information on-the-fly brought a noticeable runtime performance reduction. Depending on the used AND/OR graph style, which sends partitioning

Configuration Step	LEGO© Technic car $n = 19$	valve $n = 14$
All data	162 MB	23.5 MB
All data except PC relations and separation directions	26 MB	4.2 MB
Only contact data and valid direction pairs	84 KB	54 KB

Table 4.2: The table contains the file sizes for calculating either all configuration data, all except PC relations and separation directions, or only the geometrical information consisting of contacts and valid direction pairs.

requests to the configuration, the final initialization of a distinct path through the graph could take several seconds.

In the following, the impact of the different approaches used to reduce the file size is outlined, utilizing the LEGO© Technic car assembly, consisting of 19 parts. Storing the whole configuration information, the file size climbed above 100 MB. Loading a file of this size into the scene graph took several minutes. By calculating PC relations and separation directions on request the size was reduced to 23MB, resulting in a load time of approximately 10 seconds. Additionally finding partitionings on request reduced the size further to only 89KB. Requesting a path through the whole AND/OR graph took 1-5 seconds, depending on the used AND/OR graph style. The file sizes for two example assemblies are summarized in table 4.2.

4.3 Explosion Engine

The explosion engine is responsible for executing the commands of the user, thus exploding or unexploding assembly. Before starting the explosion, configuration information has to be calculated using the nodes described in 4.2, or loaded from a file. The appearance of the explosion depends on the used path through the AND/OR graph, the PC relations and the separation directions.

This section describes functionality common to all explosion engines in 4.3.1. The developed framework for switching between different styles is described in section 4.3.2. In section 4.3.3 the implemented engines are presented.

4.3.1 General Functionality

To be able to quickly create different explosion engines, an abstract base kit `SoAbstractExplodedView` was implemented. It enforces a general structure, each engine has to follow. The node `configuration` holds a node of type `SoExplosionConfiguration`, which is initialized while traversing the scene graph. If a configuration was already written to a file, a file can be specified using the field `configurationFile`.

User input is received by a node `SoUserInput` stored in node `userInput`. This node detects changes to certain fields representing the user input and sets an appropriate state in `ExplosionStateMachine`. During each scene graph traversal the engine queries the current user request and executes an appropriate method. Such a request can be to explode or unexplode all, only a layer or single part object. For each of these commands a separate method is called. The actual explosion functionality then has to be defined by the implementation of `SoAbstractExplodedView`.

Because a force-based layout approach is implemented, the abstract class already defines a node to hold the physical world. To enable tracking of physical objects, `SoTrackedWorldPhysics` is used. Furthermore, different visualization methods can be chosen by the user at startup by setting appropriate values for the fields `staticVisualization`. The actual implementation of the visualization is left to subclasses of `SoAbstractExplodedView`, because the abstract engine has no information about the explosion structure defined by the implementation.

Listing 4.8 shows the scene graph creating a running system. To limit the size of the listing, the tracking setup is left out. The engine presented in the listing extends the abstract engine by a node `branchSelector`, which is responsible for managing the path taken through the AND/OR graph, as well as the current explosion state. The idea of using such a branch selector will be discussed in the following.

4.3.2 Explosion Styles

Before describing the implemented engines, the notion of using a branch selector has to be explained. The engines are responsible for creating an appropriate explosion structure, so that parts can be arranged and the different visualization methods can be applied. Furthermore, each engine requires information on how to partition the assembly, and which parts are moved relative to others, as well as separation directions and distances. This information is provided by the branch selector `SoBranchSelectorExplosionStyle`, which answers requests of the engine on which partition is separated or merged next.

Listing 4.8: Example of explosion engine definition

```
SoExplodedViewPrecalculatedGraphAllParts {
    # NO_STATIC_VIS, LINES, STATIC_BLUR
    staticVisualization NO_STATIC_VIS

    configurationFile SoFile { name "explosion_configuration.iv" }
    userInput SoUserInput
    {
      explodeSingleButton = USE keysSingle.button0
      unexplodeSingleButton = USE keysSingle.button1
      explodeLayerButton = USE keysLayer.button0
      unexplodeLayerButton = USE keysLayer.button1
      explodeAllButton = USE keysAll.button0
      unexplodeAllButton = USE keysAll.button1
    }

    branchSelector SoBranchSelectorExplosionStyle {
      focusAssemblySelector SoFocusAssemblySelector {
        translation = USE keysSingle.translation
      }
      explosionStyleSelector SoExplosionStyleSelector {
        translation = USE keysSingle.translation
      }
    }

    trackedWorldPhysics SoTrackedWorldPhysics
    {
      trackerMatrix = USE MODEL_STATE.modelViewMatrix
      worldPhysics SoWorldPhysics { }
    }
}
```

Hence, the selector also has to keep track of the current explosion state. This is done by internally maintaining a stack data structure, which initially is empty. The information about each exploded partition is put on top of the stack. By removing the elements from the stack, the correct unexplosion sequence is created. This is important, because certain partitions have to be merged before others.

The branch selector itself receives information about the partitioning to be used by querying the node `SoBranchSelectorExplosionStyle` for an `SoAbstractAndOrGraph-Configuration` created by utilizing a certain combination of styles. The returned configuration only contains a distinct path through the AND/OR graph and the relevant `PC` relations and separation directions, as required by the engine.

The user is free to select styles, appropriate to the current needs. In section 3.4, five different types of styles were presented, each influencing another aspect of the final explosion

diagram, as well as a style for determining a certain ordering of a non-degenerate AND/OR graph partitioning. These are AND/OR graph styles, partition and part level PC relation styles, separation direction styles and ordering styles. For the generation of a distinct SoAndOrGraphConfiguration, an AND/OR graph has to be related to appropriate PC information and separation directions. Therefore, the corresponding styles are combined into a class ExplosionStyleGeneral, which utilizes the strategy design pattern to be able to exchange the algorithms calculating the different parameters. To implement a certain style, one of the abstract classes AbstractStyleAndOrGraph, AbstractStylePCRelations and AbstractStyleSepDirs has to be implemented. In fact, there exists only one class implementing AbstractStylePCRelations, which is StylePCRelationsStrategyPattern. This style is itself split into two parts using the strategy pattern. Implementations of AbstractStrategyPCRelationsParentPartition are used to calculate partition level PC relations, while AbstractStrategyPCRelationsParentAssembly determines part level relations.

The calculation of the configuration for an explosion follows the structure outlined in section 3.4. After determining an AND/OR graph, PC relations are calculated and at last the separation directions. Furthermore, according to the defined restrictions to prevent the explosion of already exploded parts, parent and child partitions are suggested to the PC relations style, if appropriate. The created part level PC relations are modified according to the collision prevention algorithm presented in section 3.3.2. The ordering style is not part of the returned SoAndOrGraphConfiguration. Hence, it is used outside of ExplosionStyleGeneral.

The configuration of the AND/OR graph information used for the explosion can be influenced by a focus assembly, which is selected by the user utilizing the node SoFocusAssemblySelector. The focus assembly is then input to the style calculation.

4.3.3 Engines

Two engines were implemented, both following the same basic idea of using a branch selector. In general, the engines require a distinct path through the AND/OR graph created during the configuration step, as well as corresponding PC relations and separation directions. Using this information, the physical simulation of the explosion is initialized, creating instances of SoExplosionPartPrecalculatedGraph for each single geometrical object specified in the geometrical information. These objects are connected by SoJointSpring nodes resembling PC relations and added to the physical world. If a parent is

moved, all attached parts are also translated.

Hence, the whole explosion structure is initialized before executing an explosion. The difference between the engines is the point in time, the actual visible geometry is added to the physical objects. `SoExplodedViewPrecalculateGraph` assigns the partition geometry of the current AND-branch to be exploded physical objects just before executing an explosion. In the beginning only one physical object holds the whole visible geometry as a single partition. After the first explosion, the geometry of the parent partition is assigned to the physical object being the parent of the employed `PC` relation. The corresponding child receives the geometry of the child partition. Exploding the child then separates both partitions. When partitions are unexploded again, the partitions have to be merged and assigned to the parent part.

On the other hand, the engine `SoExplodedViewPrecalculateGraphAllParts` assigns the visible single part geometry to each physical object before performing the explosion. This simplifies the engine, because partition geometry does not have to be assigned or reassigned. A side effect of this approach is, that the child part of a moved partition drags the other single parts with it, because the used `SoJointSpring` requires some time to force parts back into their relative position. Hence, the partition does not stay rigid as in the first case, but the single parts of a partition become visible.

4.4 Visualization Methods

The user can decide to visualize the original location of an object by either using simple line stipples or a static blur. Both implementations are described in the following. An additional visualization method textures the virtual model using the AR video stream.

4.4.1 Line Stipples

To activate the drawing of lines, nodes of type `SoSpringStippledLine` have to be added to physical world of `trackedWorldPhysics`, which is done by the implemented engines. Then, a simple stippled line is drawn between the two parts of a `PC` relation, which is represented by a spring. To determine information about the current locations of the objects, the appropriate `SoJointSpring` is handed over to a stippling node. The spring is used to retrieve the current physical positions of the involved parts. It also contains information about the initially unexploded position of the child relative to its parent. Utilizing this information a line can be drawn between the original location of the center

of the bounding box of the child part, and its current center. The stipples themselves are realized using simple OpenGL `GL_LINE_STIPPLE`s.

It is sufficient to retrieve the physical location of the objects, because the lines are transformed into the tracked position by the used `SoTrackedWorldPhysics` instance.

4.4.2 Static Blur

The static blur is implemented by modifying a technique for producing a motion blur presented in [10]. It involves shader programming, which was done using the OpenGL Shading Language (GLSL). To be able to utilize customized shader code in Open Inventor a shading framework, provided by Studierstube was employed. In the real world, a fast moving object is blurred in the direction of motion. This effect can be created artificially in computer graphics to enhance the realism of the rendered scene. If the blur trail does not disappear, when the object stops its motion, a static blur is created. In this thesis, such blurs substitute the simple guide lines, to provide hints on the parent of a part. Hence, the `PC` relations are visualized. Before describing the modification to produce a static blur, the method for creating the motion blur is explained.

The motion blur technique is a post processing effect. First, the whole scene is rendered to texture. In a second pass, the scene is rendered again thus calculating a velocity texture, used to blur the moving object. The velocity texture contains a non-zero vector in each pixel, the moving object went through. This vector points from the previous position represented by the velocity pixel it is associated with, to the current position of the same pixel in the scene. To create the motion blur, each pixel of the scene having an associated velocity value is blended with a color value, created by sampling along this velocity vector. The general idea becomes clearer when looking at the vertex shader code for creating the velocity texture in listing 4.10. The velocity texture created in this shader actually does not store the offset to the current pixel, but the location of the current pixel itself. The velocity is calculated in the fragment shader, when sampling the color values. The corresponding fragment shader code is shown in listing 4.10. In the following, the creation of the texture is described in more detail.

To generate the velocity texture, the geometry of the model is stretched in the direction of motion. This is done by specifying the previous model view matrix of the object. The vertices having a normal pointing into the direction of motion are transformed using the current model view matrix, while those having a normal directed against the motion vector are transformed by the previous model view matrix. The direction is determined by

Listing 4.9: Vertex shader code creating the motion blur velocity texture

```
#extension GL_ARB_texture_rectangle: enable
uniform mat4 prevModelView;

varying vec3 normal_objSpace;
// original unwarped position
varying vec3 realPosition;

void main()
{
  vec4 newPos = gl_ModelViewMatrix * gl_Vertex;
  vec4 oldPos = prevModelView * gl_Vertex;

  vec3 normal = normalize(gl_NormalMatrix * gl_Normal);
  vec3 velocity = (newPos - oldPos).xyz;

  float warpFactor = dot(normal, normalize(velocity));
  vec4 warpedPos;
  if(warpFactor < 0)
    warpedPos = oldPos;
  else
    warpedPos = newPos;

  newPos = gl_ProjectionMatrix * newPos;
  warpedPos = gl_ProjectionMatrix * warpedPos;

  vec3 ndcNewPos = newPos.xyz/newPos.w;

  realPosition =  ndcNewPos * 0.5 + vec3(0.5, 0.5, 0.5);

  normal_objSpace = normal;
  gl_Position = warpedPos;
}
```

Listing 4.10: Fragment shader code sampling along the velocity vector

```
  vec2 pixelVelocity = originalSceneCoord - gl_FragCoord.xy;
  int SAMPLES = 10;
  float blurFactor = 1.0/float(SAMPLES);

  vec3 accumColor = vec3(0.0, 0.0, 0.0);
  for(int scale = 1; scale <= SAMPLES; scale++)
  {
    vec2 scaledOffset = float(scale) * blurFactor * pixelVelocity;
    accumColor += getComposedColor(gl_FragCoord.xy + scaledOffset) *
        blurFactor;
  }
```

calculating the dot product between the motion vector and the vertex normal. If the dot product is greater than or equal to zero the current model view matrix is used, otherwise the previous one. This stretching assures that the velocity values are also stored for positions lying outside of the actual object. As noted before, in the current implementation the velocity is not stored in the texture, but the current position of the corresponding pixel value. Then this position is interpolated across each triangle and written to the velocity texture in the fragment shader.

To create a motion blur, the previous model view matrix must be set to a previous transformation of the object in certain time intervals. On the other hand, the previous model view matrix of the static blur is always a transformation to the initial position of a part relative to its parent part. Hence, the blur stretches from initial position to the currently exploded position. As in the case of using guide lines, the relevant part relationships are given by the `SoJointSpring` connection between the physical parts.

Chapter 5

Discussion and Examples

To create an explosion diagram of an assembly, the user first has to perform a preprocessing step. In this step, basic configuration data consisting of contact information and valid pair directions is calculated and written to file. In addition, the user can decide to either calculate AND/OR graph, PC relations and separation directions during the running program, or also in this preprocessing step. The more configuration data is determined, the larger the file size and thus the load time into the running program. In general, the more information is determined beforehand, the bigger the file size and the smaller the performance impact of calculating the required information in the application.

Utilizing the configuration, an explosion can be generated. The general appearance of the created diagram varies with the user-defined combination of different types of styles. Depending on the application requirement, the user has the choice to create pure symmetric explosions, or to additionally specify a focus part. A symmetric explosion may be used to inspect the assembly as a whole, while a focused explosion can be used to investigate only a single part and its surroundings. A possible application of explosion diagrams is to make the selection of certain parts easier. Then, it is sufficient to translate the parts only by a small distance.

The movement of the partitions is animated. Hence, objects do not jump into their final position, but the user can follow its motion. Furthermore, two different visualization methods are utilized, to improve the recognition of the spatial arrangement of parts and the ability of the user to reconstruct the exploded assembly. Either simple guide lines reaching from parent to child parts can be chosen by the user, or a static blur. Similar to a motion blur, there the object is blurred along the motion vector. But instead of removing the trailing blur when the object stops, it keeps visible.

In section 5.1 the implemented styles are presented and a naming convention is introduced to be able to distinguish between them. Section 5.2 presents the models used in the discussion. The following section 5.3 discusses combinations of the styles, starting with a random explosion. Section 5.4 demonstrates the usage of guide lines and static blurs.

5.1 Implemented Styles

This section summarizes and explains the implemented styles. In addition unique names are assigned to them, so that they can be referred to in the discussion of the combinations of styles. Descriptions overlapping with the introduction of the styles in section 3.4 are kept short. The `PC` relations are restricted to encompass only parts in contact between two unexploded partitions.

The names of the styles derive from the titles of their corresponding sections. A short version of the title of such a section is the first part of the name of the style. "AND/OR graph" is the title of the section describing AND/OR graph styles and is shortened to *AND/OR*. Respectively, "separation direction" is shortened to *SepDir*. The second part of the name is the title of the paragraph describing a specific implementation of a style. For instance, an AND/OR graph style outlined in paragraph "First Hit" is named *AND/OR First Hit*.

`PC` relation styles are a special case, because there exist separate styles working on either partition or part level. Hence, these subdivision defines own styles, which described separately. *PCP* is the shortcut for a `PC` style dealing with partitions, *PCA* for one determining the relations between assembly parts. The title of the section describing ordering styles is "Ordering". It is not abbreviated, thus these styles start with *Ordering*.

If not stated otherwise, the size of a part or partition is calculated by determining the volume of the corresponding bounding box.

5.1.1 AND/OR Graph

First Hit: Without considering any criteria, simply the first of all possible AND-branches in the AND/OR graph is selected for each partition. This may result in a kind of single part peeling, splitting the assembly into several big partitions or in a combination of both. The outcome depends on the partitioning algorithm used to build the overall AND/OR graph of the configuration.

General Single Part: A general peeling style removing all of the outermost single parts before considering the next layer of separable parts. As outlined in section 3.4.1, determining all removable single parts of a subassembly in one step can lead to unconnected partitions. This implementation handles such cases by dropping single parts, violating the connectedness of the remaining partition. If all parts have been processed or dropped, the next single part layer of the rest of the assembly is calculated.

Smallest First: Instead of considering all single removable parts of an assembly like in the general peeling style, only separable components of the same size are considered in each iteration. This is an implementation of a symmetric style. In this case the smallest parts are exploded first. Using floating point precision the size of the same type of parts may vary. Hence, a certain tolerance is introduced into the comparison.

Biggest First: The implementation is similar to the previous symmetric style, but it removes biggest parts first.

Focus Nearest First: This style reveals the user-defined focus part by removing the nearest parts until it can be exploded. Unlike the peeling styles described before, not all removable parts are determined at once, but the nearest component is always recalculated after removing an object. Hence, no layers of parts having the same nearest distance are defined. If the focus is free to move, it is nearest to itself. At this point, the assembly is partitioned into the rest and a partition holding only the focus part.

Then, the rest of the assembly not containing the focus element is processed using the same algorithm. During the selection of a path through the AND/OR graph, the assembly is unexploded. Hence, the focus element is still in its original location and the distance to each part of the following partitions still is the nearest distance of the original assembly (see figure 3.19).

Focus Most Parts Partitions: In section 3.4 two different algorithms removing whole partitions to reveal a focus part were described. One required data about all partitionings of the assembly, hence the whole AND/OR graph, while the other could utilize an iteratively built graph. The second version was implemented to take advantage of the configuration speedup, gained by calculating the graph on request. Implementing the first version would require the calculating of great parts of the whole AND/OR graph, hence the advantage would be lost.

Starting with the fully assembled product, the partition consisting of the greatest number of parts not containing the focus part is determined and removed. After finding all partitions using this approach, the last partitioning consists of the rest of the assembly and the single focus object. The following partitions not containing the focus element are then partitioned by removing the smallest parts first, thus creating a symmetric partitioning. Unlike done in style *AND/OR Smallest First*, not all removable parts of the same size are considered at once. This style follows the approach of *AND/OR Focus Nearest First*, where the whole range of separable parts are always reconsidered after exploding a part.

5.1.2 Parent-Child Partition

First Hit: The first partition of an AND-branch is chosen to be the parent partition. No other criteria are applied.

Biggest Parent: The partition of greatest size is the parent partition.

Most Parts Parent: The partition containing the greatest number of parts is the parent partition. When receiving the last partitioning as input, consisting only of two single part partitions, it randomly selects one of those as parent partition. The style is meant to be used in cases, where a partition containing a great number of parts is smaller than the second partition, but the larger partition should be moved.

Movable Focus Biggest Parent: The biggest partition always is the parent partition, except when the focus assembly can be removed as a single part. In this case the partition not containing the focus is the parent and the focus moves.

Movable Focus Most Parts Parent: This style is the same as the previous one, but instead of making the biggest partition to the parent, the partition containing the most parts is chosen. Again, the focus part is the movable child if it is the only part of a partition.

Static Focus: To keep the focus part static, the partition containing the focus is always the parent partition. Even if a *PCA* style does not care about the focus, it will never move, because the focus part can never be selected as a child part.

5.1.3 Parent-Child Part

First Hit: The first out of the set of potential `PC` relations between two partitions is selected.

Smallest Parent: This style is meant to be used when the AND/OR graph is degenerate. It is sufficient to determine one out of the set of parent parts, because only a single child exists. In this case the smallest object is the parent.

Biggest Parent: Similar to the previous style, but the biggest component is the parent part. Exploded parts are always associated with the biggest parts of the rest of the assembly.

Biggest Parent/Child: When both, parent and child partition, consist of more than one object, the selection has to be extended to encompass the range of possible child parts. In this style, the biggest parent part is determined first. The parts of the other partition, which are in contact with this parent are potential children. The biggest child is chosen to finalize the `PC` relation.

Biggest Parent/Child Prefer Focus: A refinement of the style *PCA Biggest Parent/Child* is to prefer the focus element as parent part. As will be seen later on, this is a useful requirement when performing a focused explosion.

Biggest Asymmetric Parent/Child Prefer Focus: If partitions containing several parts have to be separated from each other, both may contain symmetric parts. As outlined in 3.4 it can happen that parts of the same symmetrical group are exploded while others stay static. This style implements a countermeasure by extending the style *PCA Biggest Parent/Child Prefer Focus*. The potential child parts of the focus element are searched for asymmetric parts, one of which is then used to establish the `PC` relation. If the parent partition contains no focus element, the biggest asymmetric parent part is chosen.

5.1.4 Separation Direction

Size Distance: The separation direction found first is used as explosion direction. The distance depends on the size of the moved partition. The bigger the partition, the farther it moves. The size actually is the length of the diagonal of the parts bounding box.

Ten Percent Distance: The distance of moving parts is reduced to ten percent of the length of the bounding box diagonal.

Ten Percent Distance Except Focus: All parts are moved by only ten percent of the length of their bounding box diagonal, except the partition containing the focus element. This partition is moved to its full extend.

Ten Percent Distance Except Focus Contacts: The focus element and the partitions being in direct contact with the focus are moved to the full distance. All other parts are only exploded to ten percent of the length of their bounding box diagonal.

5.1.5 Ordering

Reveal Focus: This ordering style reveals the focus when the user decides to perform a layered explosion.

Symmetric Layers: All parts of a symmetric group are exploded in each layer.

5.2 Discussed Models

Two example assemblies were used to test the creation of explosion diagrams. General information like the number of parts, as well as problems, which came up when exploding the models are described in the following. Parts of similar size and shape are colored in the same way.

5.2.1 Car Assembly

In this discussion the LEGO© Technic car depicted in figure 5.1 is used. The car consists of 19 parts and is symmetric along its length axis. Originally, it consisted of more than 19 parts, but to reduce the computational effort the assembly was prepartitioned into several multi part partitions, which can be seen as a groupings into subassemblies. Furthermore, a non-rigid hose part was removed from the model, because only rigid objects are allowed. The model itself still contains several non-rigid parts. The flexible technical pins of LEGO© Technic snap into place when they are attached. When removing them, they are compressed a little bit. The implemented direction finding algorithm identifies correct separation directions for such cases.

A problem with this model is, that the steering wheel is attached to a bent steering axle. To remove the wheel, it has to follow a path consisting of multiple translations, but the implemented algorithm simply checks for single translations to infinity. Because no direction is valid, the least blocking direction is chosen as separation direction. Hence, the steering wheel is exploded sideways away from the axle, which in fact is a part of the correct assembly motion. Nevertheless, the direction is manually corrected so that the steering wheel is exploded upwards, which was found to look better.

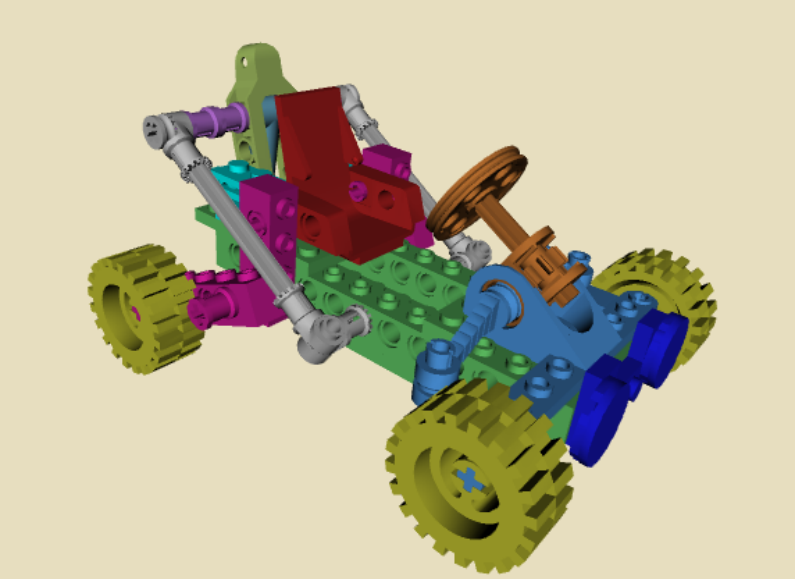


Figure 5.1: The assembled LEGO© Technic car model consisting of 19 parts

5.2.2 Valve

The valve depicted in figure 5.2, consists of 14 rigid parts. This model was the reason for developing the collision prevention algorithm presented in 3.3.2, as well as the extension to simple local freedom tests in the partitioning step of the AND/OR graph described in section 3.2.1.

Considering only global blocking constraints, the screwnuts attached to the green bolts could not be removed in single part partitions when assembled with the nearly black part, thus blocking the partitioning algorithms of a great part of the implemented styles. Testing the removal of the screwnuts relative to the parts in contact, solved this problem. If the blocked screwnuts would have been in direct contact with the black object, the partition could not be split. The global and the local blocking constraints would have been the

same.

Another problem with this model were collisions between exploding objects. For instance, the screwnuts in contact with the black and the red object are associated with the red object, if the *PCA* style selects the biggest part to be the parent. When removing the black object, it collides with the screwnuts. The collision prevention algorithm detects such cases and assigns the black part to be the new parent.

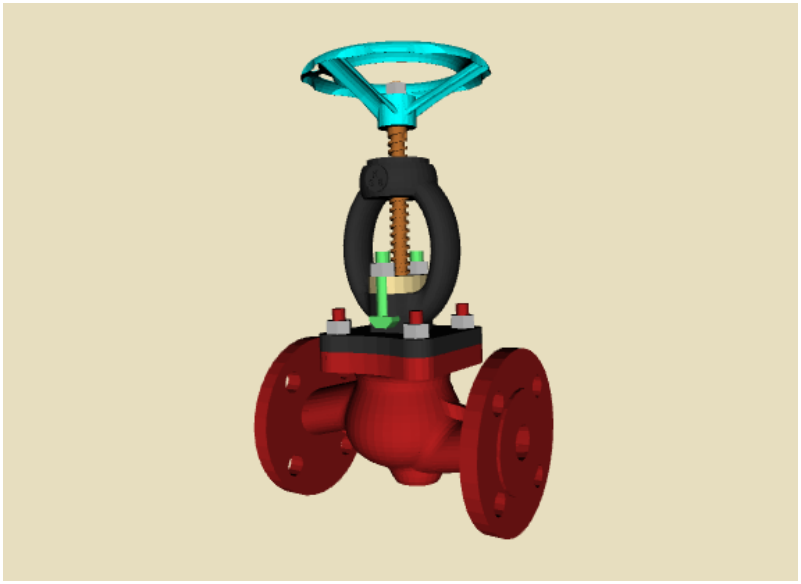


Figure 5.2: The assembled valve model consisting of 14 parts.

5.3 Combining Styles

In this section different combinations of the style implementations are investigated. Before starting the discussion, the requirements to create an appealing explosion diagram are examined. Agrawala et al. [2] automatically generate illustrations of assembly instructions and performed a study on the design principles of such instructions. They came to the conclusion that a user perceives an assembly as hierarchy of parts, where the components are grouped into subassemblies. Furthermore, the single parts are arranged in functional groupings, like for instance the wheels of a car. In the context of assembly instructions, users prefer illustrations where parts of the same grouping are assembled in the same image, or at least in a sequence.

Agrawala et al. [2] also identify a hierarchy of operations. When assembling a product, users tend to identify significant parts, which can be added next. These parts are then

attached to the assembly using less significant parts. They state that the significance of objects relates to their functionality, as well as to their size and symmetrical considerations.

This knowledge can be mapped to explosion diagrams, because an explosion resembles the inverse of assembly instructions. Thus, a hierarchy should be identified, so that parts of the same functional group are removed at the same time, resulting in a symmetric explosion. Furthermore, because smaller parts are perceived as less significant objects attaching larger parts, it may be better to remove smaller parts from an assembly first. This requirement leads to the assumption that moving small parts relative to bigger objects is perhaps more intuitive for a user. In the following is shown, that following this considerations leads to well-structured, symmetric and homogenous diagrams.

Symmetrical properties of the assembly are identified by investigating its size and the position relative to other parts. Without defining additional information, the real functional groupings cannot be determined. In addition, the grouping of parts into meaningful subassemblies, e.g. the motor of a car, is not possible.

Starting with a bad example of an explosion diagram created by using the first hit of each parameter, it is shown how the result can be improved by changing only the `PC` relation styles. Nevertheless, this approach has its limitations, because the existing AND/OR graph is very unstructured. Using other AND/OR graph and `PC` relation styles overcomes many problems and well-structured explosion diagrams can be created. Additionally considering a focus part when generating an AND/OR graph and corresponding `PC` relations, generates an explosion diagram, revealing the focus element by exploding only relevant parts or partitions of the assembly. An explosion diagram can be used to loosen the surroundings of a part, the user is interested in, thus revealing more of the potential selection.

In the following, not all possible combinations of the styles are considered, because of the great number of combinations. In addition, the outcome of certain styles is very similar. For instance, exploding the smallest or the biggest single parts of an assembly changes the ordering of part removal, but the symmetric considerations are the same. For `PC` relations it will be shown, that moving smaller parts relative to larger ones should be preferred over moving parts of larger extents relative to small parents. This supports the hierarchy considerations described earlier in this section.

In general, the parameters influencing the appearance of the explosion most are AND/OR graph and `PC` relations. Separation directions are secondary if the range of potential assembly directions is already limited to a few. Using a

LEGO© Technic model of a car as example assembly, the motion vectors are very constrained, because most parts can be plugged together only along a single direction. Hence, the separation direction style for each combination is *SepDir Size Distance*, if not stated otherwise. Examining more general assemblies allowing planar contacts between parts, introduces the need to prevent sliding motions of parts. In section 3.4, an algorithm solving this problem was described. Currently, if a separation direction for a planar contact is not to the liking of the user, it can be changed in the configuration file for each relevant part pairing. When discussing explosions considering a focus element, other separation direction styles are utilized. The distance has to be big enough when creating general explosion diagrams, to really separate the parts. If focus parts become more relevant, this requirement is weakened.

Ordering styles are only used to allow a definition of layers the user can explode. Additionally, they can describe a part ordering in case of a non-degenerate AND/OR graph. Because they contribute nothing to the appearance of the explosion diagram when applied on degenerate graphs, their influence can be neglected for most AND/OR graph styles.

The discussion is supported by images of the explosions. The explosion directions are visualized by guide lines. Using this information makes it easier to visualize the position of the exploded parts in the room and to reconstruct the assembly.

The following sections are structured according to the used AND/OR graph styles and describe the influence of different `PC` relation styles on the explosion partitioning created by the AND/OR graph. Section 5.3.1 describes the random explosion style *AND/OR First Hit*. Section 5.3.2 presents results using the style *AND/OR General Single Part*. Section 5.3.5 demonstrates the application of the symmetric style *AND/OR Smallest First*, while section 5.3.4 outlines the usage of the focus styles *AND/OR Focus Nearest First* and *AND/OR Focus Most Parts Partitions*. Section 5.3.5 gives a summary of the analysis of the different styles.

5.3.1 Random Style

Using only first hit AND/OR graph and `PC` relation styles, namely *AND/OR First Hit*, *PCP First Hit* and *PCA First Hit*, creates a somewhat random explosion diagram. The partitionings as well as the relations do not follow any conventions. Hence the explosion can quickly expand into any direction, as seen in figure 5.3. The image shows that partitions consisting of a great number of parts can be moved relative to single part partitions, which

lead to the massive expansion to the right.

Influencing the explosion using different `PC` relation styles improve its appearance, but the results are not satisfying. Figure 5.4 shows the outcome when using *PCP Biggest Parent* in combination with the previously used *AND/OR First Hit* and *PCA First Hit*. Although the extents of the explosion are now limited, the part level `PC` relations are not optimal. For instance, the gray parts from the rear geometry are associated with different parent parts. Such assignments may be prevented by further using the part level style *PCA Biggest Parent/Child*. In this case, out of the range of possible parents and children between two partitions, the biggest ones are chosen. As depicted in figure 5.5 the `PC` relations still are not assigned properly. Utilizing the mentioned *PCP* and *PCA* styles should lead to the assignment of the parents out of the same symmetrical group to symmetric children. For instance, the gray parts should both be assigned to the green floorplate. The `PC` relation styles do not work correctly, because the structure of the AND/OR graph, hence the partitioning of the assembly itself, is completely random. In this example, one of the gray objects is exploded in a single part partition, while the other one is moved in a partition containing amongst other parts the seat.

The combination of the styles *AND/OR First Hit*, *PCP First Hit* and *PCA First Hit* applied to the valve example results in the explosion shown in figure 5.6. First of all, the explosion ordering is completely random. For instance, partitions containing a great number of parts are moved relative to single parts. Hence, the blue hand wheel is moved back to its original location on top of the bolt. Nevertheless, the explosion shows symmetric properties. By coincidence, the screwnuts fixing the black object to the red one are all child parts and thus exploded. This supports the symmetry. In addition, the collision prevention algorithm described in 3.3.2 corrects the relations of the said screwnuts so that they are assigned to the black object. All in all, the symmetrical structure comes from this algorithm, the small number of parts and pure coincidence.

The random AND/OR graph structure becomes apparent in figure 5.7, where the symmetric `PC` relation styles *PCP Biggest Parent* and *PCA Biggest Parent/Child* are used. The marked area shows a collision of a part with the green bolt. The collision prevention algorithm described in section 3.3.2 cannot work properly, because the green bolt was moved in a partition with the attached screwnut, thus not as single part partition as required by the algorithm. By coincidence the random partitioning of the valve leads to the same moved partitions, but still the explosion ordering is completely random. Symmetric parts are not exploded at the same point in time.

Hence, PC relations can be used to greatly improve an even unstructured AND/OR graph partitioning by reducing its extents. But the results are not very satisfying. Partitions are removed in any order and size. Therefore, PC relations are often assigned in ways, resulting in rather confusing part movement of symmetric parts. For instance in figure 5.5, a rear wheel is moved to the bottom left, while its counterpart on the other side of the car is exploded to the top right. The best method to enhance the explosion diagram is to create an AND/OR graph following certain conventions, like symmetry or revealing a focus element.

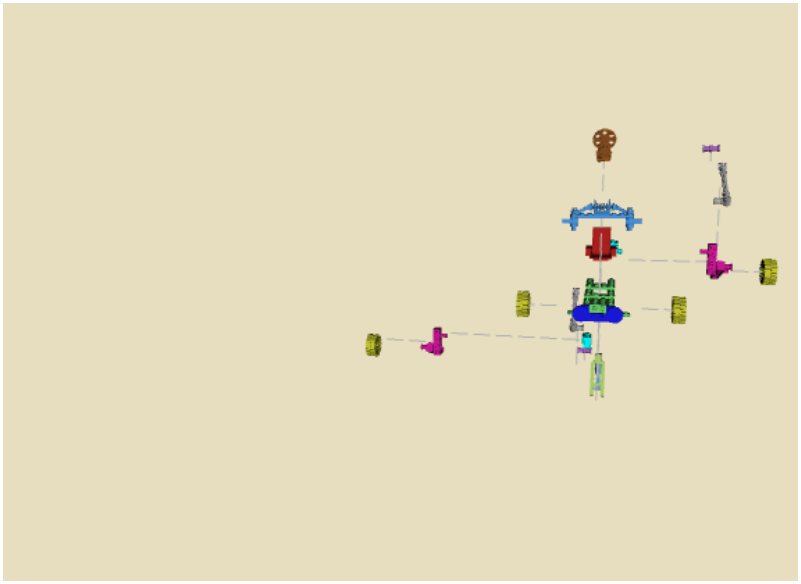


Figure 5.3: A random explosion, which quickly became unstructured and extended in size

5.3.2 General Peeling

A simple structure is introduced by using a general peeling style like *AND/OR General Single Part*. The partitioning algorithm creates single part partitions, where the outermost objects are exploded first. Even without using a special PC relation style the explosion is more hierarchical than the random version (see figure 5.8). This comes from the implicitly correct assignment of PC relations. Because single parts of the same removable layer are detected, also groups of symmetric parts are identified. Hence, in addition to other parts, symmetric parts are exploded in the same layer. If they are in contact with only one parent part and the whole assembly is symmetric each part implicitly has the same parent. This is the case in the example.

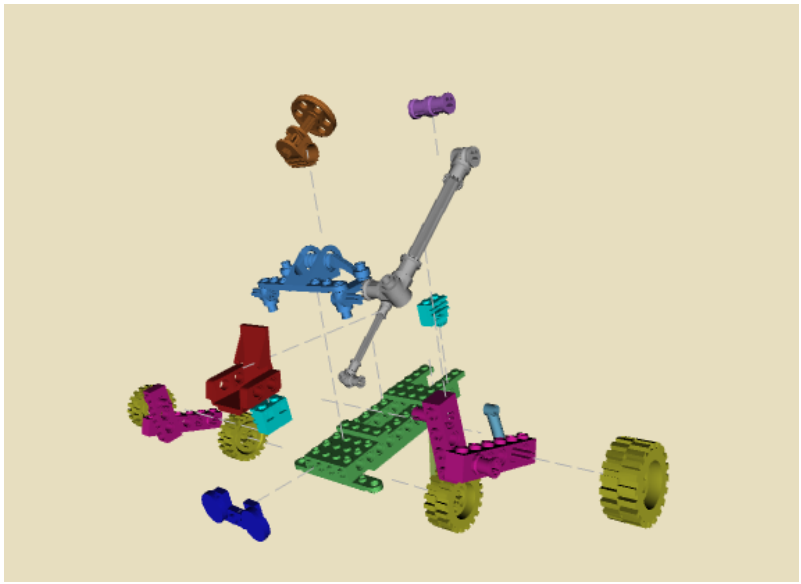


Figure 5.4: A partition level `PC` style (*PCP Biggest Parent*) is applied to a randomly selected AND/OR graph. The assignments of parent and child partitions slightly improve the diagram by reducing its extents, but this is not sufficient.

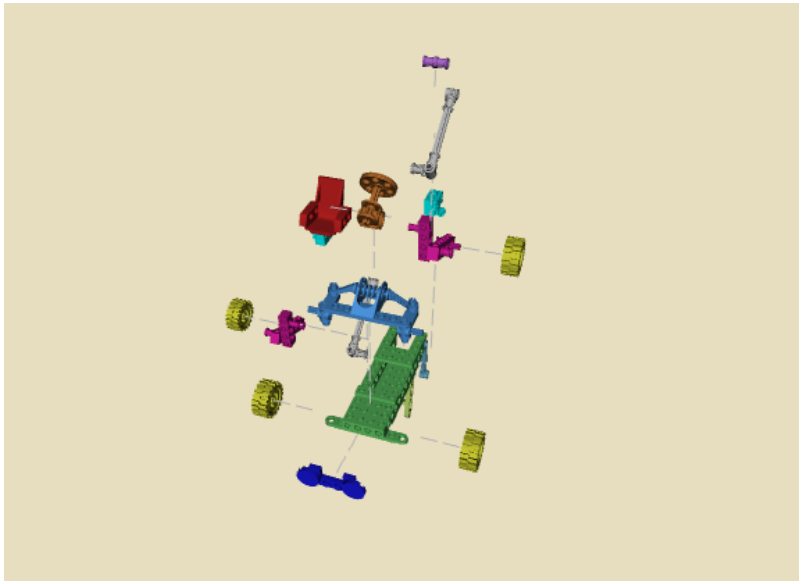


Figure 5.5: Partition and part level styles (*PCP Biggest Parent*, *PCA Biggest Parent/Child*) are employed on a random explosion. The explosion is more symmetrical, but the partitioning still is random.

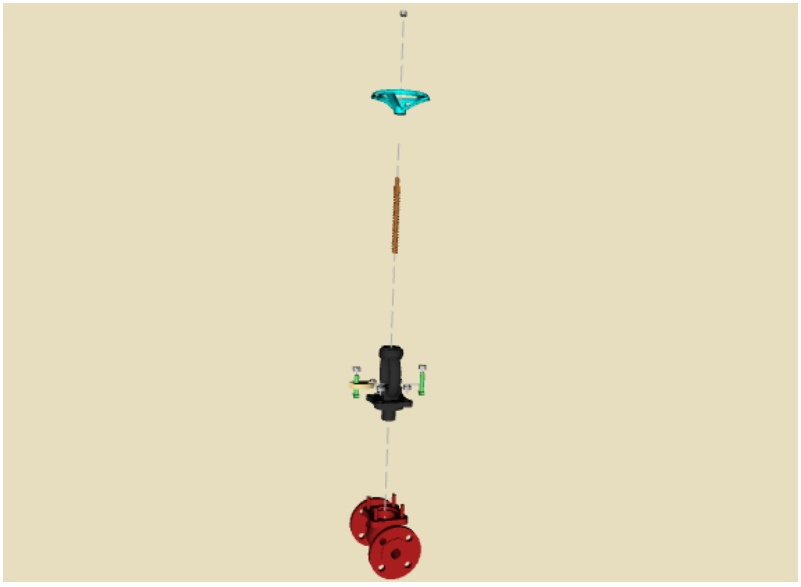


Figure 5.6: The random styles *AND/OR First Hit*, *PCP First Hit* and *PCA First Hit* are combined. The explosion is symmetric, which is on the one hand coincidence, on the other hand comes from the structure of the model and the collision prevention algorithm.

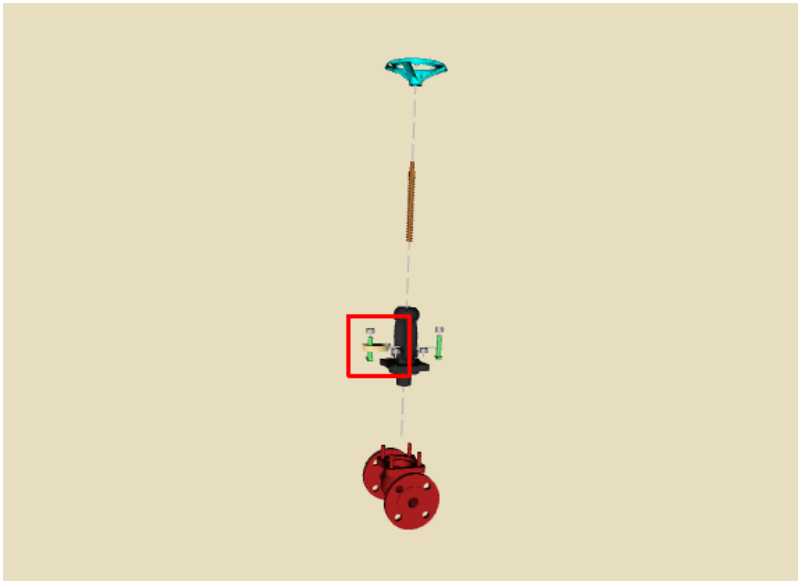


Figure 5.7: Partition and part level styles *PCP Biggest Parent* and *PCA Biggest Parent/Child* are employed on a random AND/OR graph created by *AND/OR First Hit*. Because not only single part partitions are removed, a collision occurs in the marked area.

But the assignment of symmetric PC relations is random. Hence, partitions containing a great number of parts are moved, leading to the great extents of the explosion. Such cases are marked in the figure by red lines. Another problem is, that the ordering of the parts does not follow any convention and cannot be influenced by an ordering style because the AND/OR graph is degenerate. Hence, the implicit symmetry is only visible in the fully exploded state. Exploding the assembly step-by-step would be rather confusing, because parts are removed in any order.

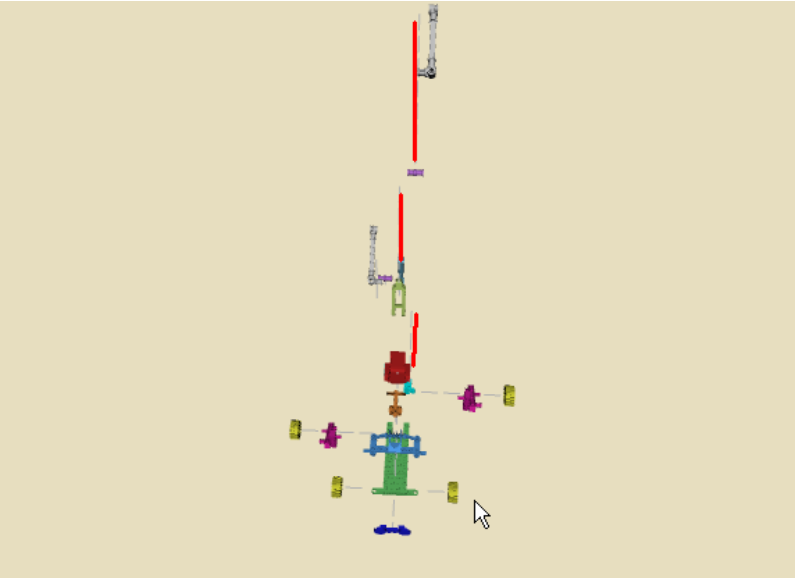


Figure 5.8: Using a general peeling algorithm, the explosion diagram appears to be well structured, even without using any additional PC relation styles. This comes from implicitly created symmetry. But the explosion is very extended. The red lines mark movements of a multi part partition relative to a single object.

Employing symmetric PC relation styles solves the problem of moving big partitions or partitions containing more than one part and reduces the extents of the explosion. Additionally, the appearance of the implicit symmetry is also improved, as depicted in 5.9. There the style *PCP Biggest Parent* was used. The explosion is more centered around the largest part and the single parts are removed from the model, instead of moving the rest of the assembly. But single parts of the same symmetrical group are not assigned to the same symmetrical parents. In the example case, the turquoise geometry on the right of the car has a different parent than the same geometry of the let side. In fact, a multi part partition was moved and one of the turquoise parts is the parent of the seat. Therefore, the *PCP* style should be supported by a *PCA* style. But before utilizing such a style, the

style *PCP Most Parts Parent* is discussed.

In another example, the style *PCP Most Parts Parent* was employed and the whole exploded assembly was moved relative to a single part. This situation occurred, because the last partition consisting of only two parts also contained the first parent of all parts. Partitioning this subassembly, the parent-child partition (*PCP*) relation algorithm randomly decided to make this parent to a child. To prevent such an assignment, the algorithm can be modified to always make the parent of all parts to the parent of the last part. Nevertheless, if the biggest parent partition is always chosen using *PCP Biggest Parent*, it is very likely, but not assured, that the last partitioning will move the part being parent to a smaller number of children than the other component. For the assembly used in this discussion, this assumption held, because all parts were fit onto a base plate, bigger than any other object.

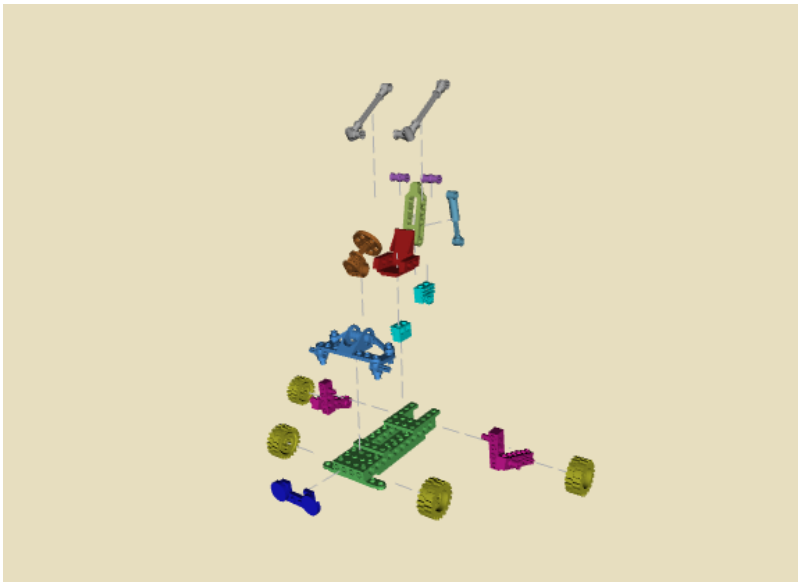


Figure 5.9: Using the style *PCP Biggest Parent* to support the general style *AND/OR General Single Part*, the explosion becomes more structured. Bigger partitions are not moved away from smaller ones anymore.

The symmetrical hierarchy created by the styles *AND/OR General Single Part* and *PCA Biggest Parent* can be improved further by using a part level `PC` relation style, for example *PCA Smallest Parent*, as depicted in figure 5.10. The resulting explosion extends upwards. This extension of the explosion comes from the assignment of bigger parts to smaller ones, as well as the single part explosion algorithm of the AND/OR graph style, removing each separable single part. After freeing the floor plate of the car model, the

rear geometry containing the seat and other small parts ended up in the same partition. This small parts already were parents to other large parts, which were exploded upwards. Utilizing *PCP Biggest Parent* the partition was exploded away from the bigger floor plate, thus exploding the larger parts further upwards. A countermeasure to this problem is to use *PCA Biggest Parent* or *PCA Biggest Parent/Child*.

The second technique is more promising, because it also handles possible moving multiple part partitions like the one containing the seat and the turquoise side parts. In figure 5.10 is visible that one turquoise part out of the multiple part partition is chosen as parent part, when using the style *PCA Smallest Parent*. By using the style *PCA Biggest Parent/Child*, the `PC` relation is established between the biggest parts of the partitions, thus the smallest parts are not selected as parents, but the seat is chosen. It is true that this method only works if the part biggest in size always is the asymmetric part of the multiple part partition. Therefore, a more exact algorithm like the one used in *PCA Biggest Asymmetric Parent/Child Prefer Focus* would identify asymmetric parts in both partitions to prevent such cases. This style will be discussed in section 5.3.4, when focused explosions are created. In general it may be a good idea to move smaller parts relative to bigger ones. It is more intuitive for a user, that a screw is associated with the object it is attached to and not vice versa. This also follows the hierarchy considerations mentioned in the beginning of this section (see 5.3).

Figure 5.11 depicts an explosion using the styles *AND/OR General Single Part*, *PCP Biggest Parent* and *PCA Biggest Parent/Child*. The *PCA* style selects the parent, which is biggest in size. Because most of the parts are in contact with the central green base plate, it becomes the main parent. Hence, the extents of the explosion are reduced. The AND/OR graph structure created by the general peeling algorithm is enough for the valve model to be exploded symmetrically, as seen in figure 5.12. The hybrid blocking constraint approach taken by the partitioning algorithm becomes visible. The green bolt on the right side is exploded to the right, while the sane bolt on the left side is exploded downwards. In this case a globally blocked partition was separated using local blocking constraints. After removing the first green bolt, a partition consisting of the other bolt, a screwnut and the yellow plate was moved relative to the black fixture. The outcome may not be symmetrical, but the hybrid approach is a necessary modification to be able to separate this assembly. Although the final appearance of the created symmetric explosion diagrams may be appealing, the information gained from interactively exploding the assembly suffers from the unordered removal of parts. Hence, symmetry has to be considered further in

a separate AND/OR graph style, introducing a partitioning where parts of the same symmetric group are exploded in a sequence.

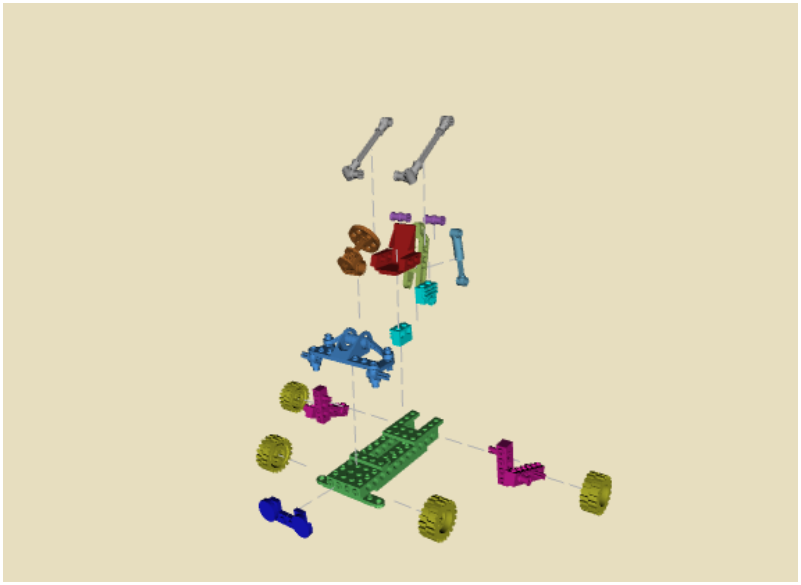


Figure 5.10: Using the styles *PCP Biggest Parent* and *PCA Smallest Parent* to support the general style *AND/OR General Single Part*, the final explosion diagram is very symmetric.

5.3.3 Symmetric Peeling

Both symmetric styles, *AND/OR Biggest First* and *AND/OR Smallest First* in combination with the `PC` relation styles *PCP First Hit* and *PCA First Hit* run into the problems outlined in section 5.3.2 when discussing the general peeling style. Although the explosion is ordered, the `PC` assignments are random, for example resulting in the movement of the whole exploded assembly. Therefore, combinations using random `PC` relation styles will not be discussed further.

As outlined earlier it may be better to explode smaller parts before larger ones. Small parts are more likely to be components fixing other parts together, like for instance screws. Hence, the styles *AND/OR Smallest First*, *PCP Biggest Parent* and *PCA Biggest Parent/Child* were utilized to create an symmetric explosion. Although the final explosion depicted in 5.13 is very similar to the one in figure 5.11 using a general peeling approach, the generated AND/OR graph in the background is very different. The parts are now grouped depending on their size, enabling the user to sequentially remove the same type of parts, before other objects are exploded. Additionally, by using the ordering style *Or-*

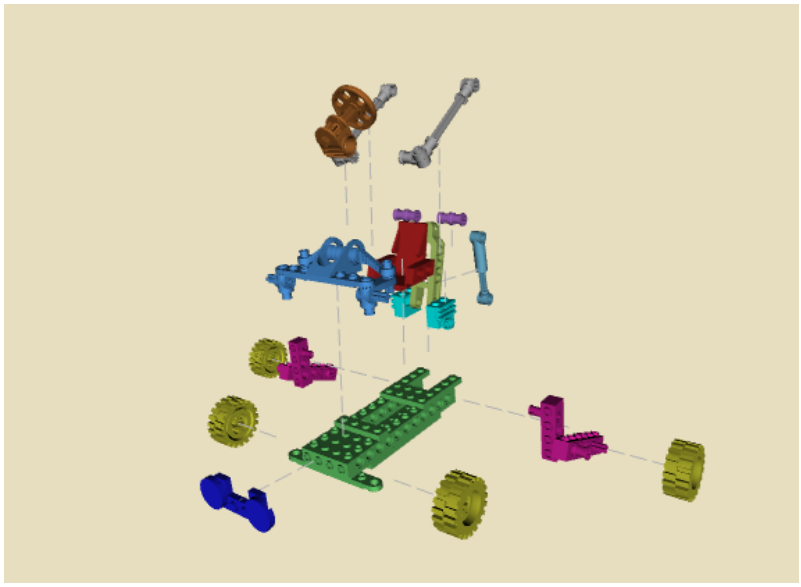


Figure 5.11: Using the styles *PCP Biggest Parent* and *PCA Biggest Parent/Child* to support the general style *AND/OR General Single Part*, the final explosion diagram is very symmetric. The ordering of part removal is random. The green base plate is the main parent, thus the parts are not separated very far from each other.

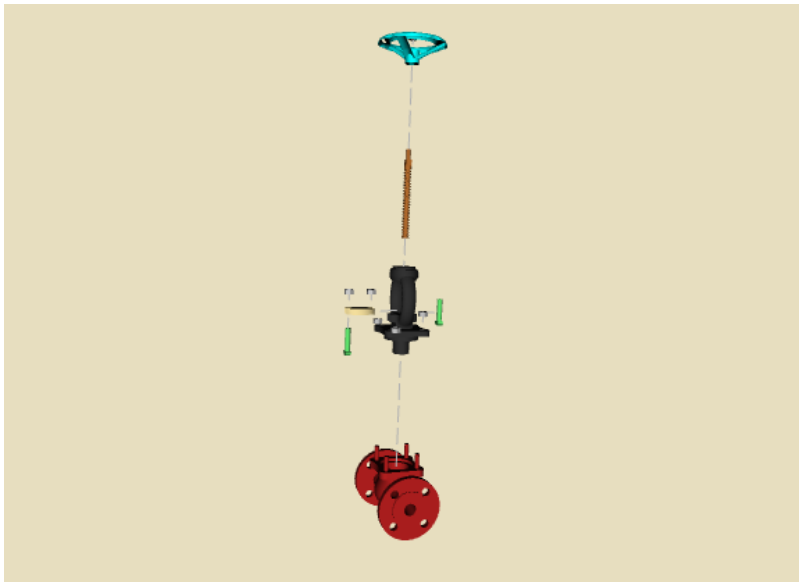


Figure 5.12: Using the styles *PCP Biggest Parent* and *PCA Biggest Parent/Child* to support the general style *AND/OR General Single Part*, the final explosion diagram of the valve is completely symmetric, but the ordering of part removal is random.

dering Symmetric Layers, the components of the same symmetric group can be exploded in the same layer. Figure 5.15 shows the same combination of styles applied to the valve model. The exploded arrangement was completely symmetrical when using the general peeling style, but removing the smallest part first also introduced symmetry into the interactive part removal performed by the user. Furthermore, removing smallest parts first in combination with the biggest parent partition `PC` relation style assures that the bigger parts are really moved after smaller ones, if at all. This was not the case when using the general peeling style *AND/OR General Single Part*.

When discussing *AND/OR General Single Part* in combination with styles *PCP Biggest Parent* and *PCA Smallest Parent* the extents of the explosion increased. When using *PCA Smallest Parent* in combination with *AND/OR Smallest First*, the extents are still limited. This is a logical consequence of such a combination. Removing single, smallest parts first, they cannot be used in parent assignments for larger parts anymore. Figure 5.14 shows such an explosion. A modification by the collision prevention algorithm is also visible. Using the smallest parents, the front wheels are originally assigned to the blue axle. The algorithm modifies the relation so that the floor plate is the new parent.

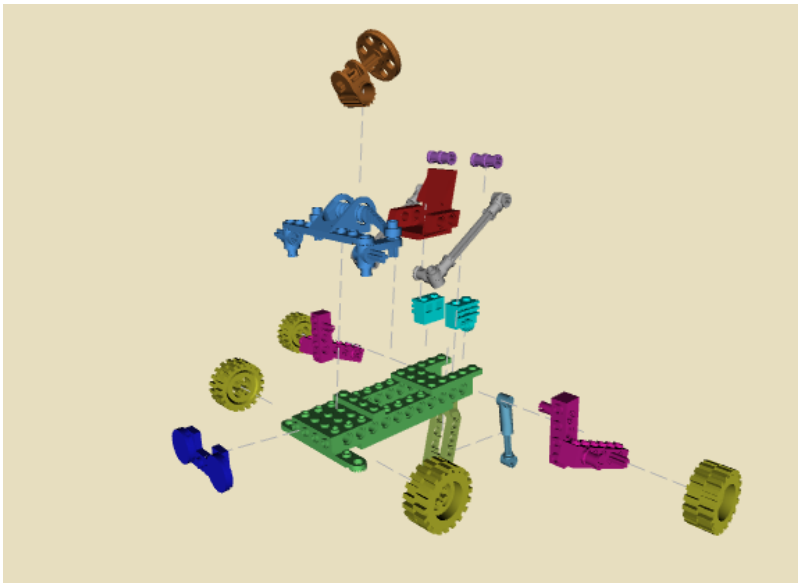


Figure 5.13: A symmetric explosion resulting from the styles *AND/OR Smallest First*, *PCP Biggest Parent* and *PCA Biggest Parent/Child*. The final explosion diagram is symmetric and the part removal is ordered. Groups of parts having the same size are removed in a row. Using *Ordering Symmetric Layers*, these parts can be exploded in layers.

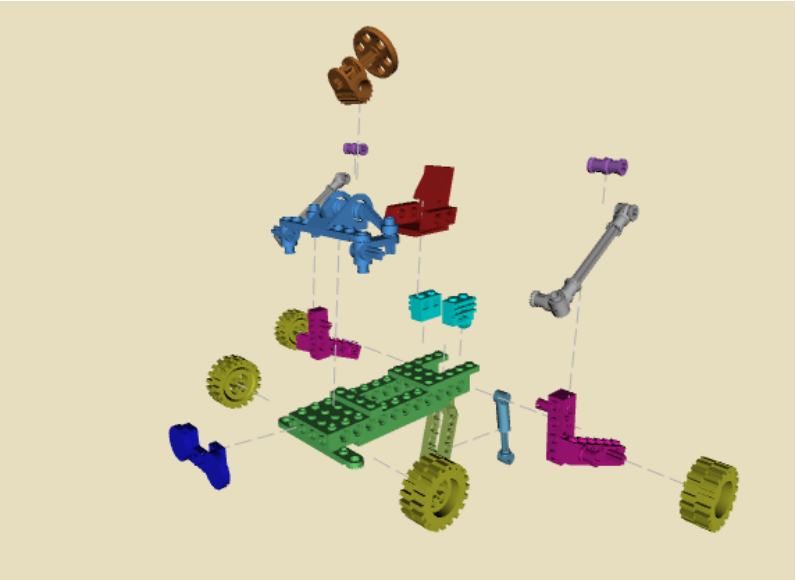


Figure 5.14: A symmetric explosion resulting from the styles *AND/OR Smallest First*, *PCP Biggest Parent* and *PCA Smallest Parent*. The rear geometry moves with smaller parts and is not assigned to the green base plate anymore.

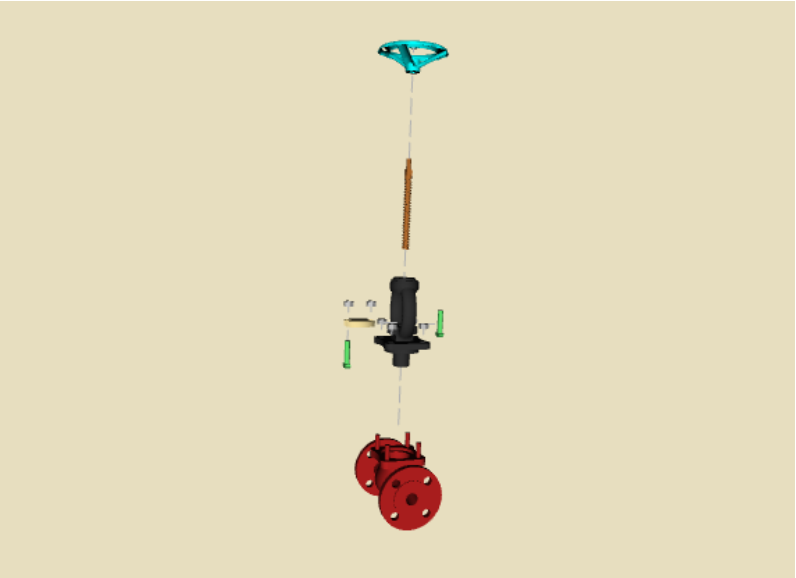


Figure 5.15: A symmetric explosion resulting from the styles *AND/OR Smallest First*, *PCP Biggest Parent* and *PCA Biggest Parent/Child*. The final explosion as well as the step-wise part removal is ordered symmetrically.

5.3.4 Revealing a Focus Element

To quickly reveal a focus element, the style *Ordering Reveal Focus* is used for all of the following combinations. This allows the user to jump to an exploded state where the focus element is partitioned into a single part partition.

Before describing AND/OR graph styles optimizing the graph to reveal a focus element, the previously discussed styles are reconsidered using the style *PCP Movable Focus Biggest Parent* and *PCA Biggest Parent/Child*. It is shown, how these styles behave when a focus element is considered in the `PC` relation creation. The examples depict the minimal explosion state revealing the focus element, which is the seat of the car, if not stated otherwise.

Figure 5.16 illustrates the application of the style *AND/OR First Hit*. The explosion is completely chaotic, hence the context of the seat can hardly be reconstructed. Creating an explosion diagram using the style *AND/OR General Single Part*, the explosion diagram is well structured, but the worst case has happened. All parts must be exploded to move the seat, because the ordering of moving parts is prescribed by the precalculated AND/OR graph (see figure 5.17). The situation is better for the style *AND/OR Smallest First*, as shown in 5.18, but still objects having no contextual connection to the seat are exploded. In addition, the large number of parts hide the focus element and the worst case where all parts have to be exploded to reveal a focus is not ruled out. Therefore, a new structure for the AND/OR graph has to be found.

It should be noted that although the AND/OR graph prescribes the ordering of the automatic part removal, the data structure can be transformed into a more general one, which allows the search of the minimal number of part movements to reveal the focus element, similar to [21]. For this purpose, only `PC` relations, separation directions and the blocking relationships have to be considered. Nevertheless, the structure of the resulting explosion is still influenced by the underlying AND/OR graph. It describes the possible partitionings and is the foundation for the calculation of the other data. Hence, the AND/OR graph itself has to be optimized to reveal the focus element.

Removing Nearest Parts

The style *AND/OR Focus Nearest First* only removes those parts, which are nearest to the focus element. After removing a nearest object, the newly created subassembly holding the focus is reconsidered as a whole. Although this method is not explicitly used to detect parts symmetric to a central part, implicit symmetrical explosions can be created. As

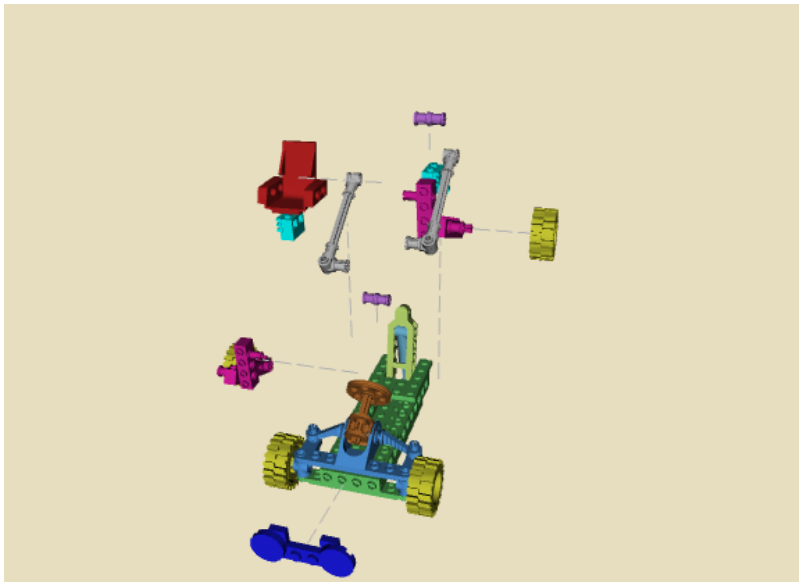


Figure 5.16: A random explosion resulting from the styles *AND/OR First Hit*, *PCP Movable Focus Biggest Parent* and *PCA Biggest Parent/Child*. This is the explosion state revealing the seat element. The explosion is completely chaotic.

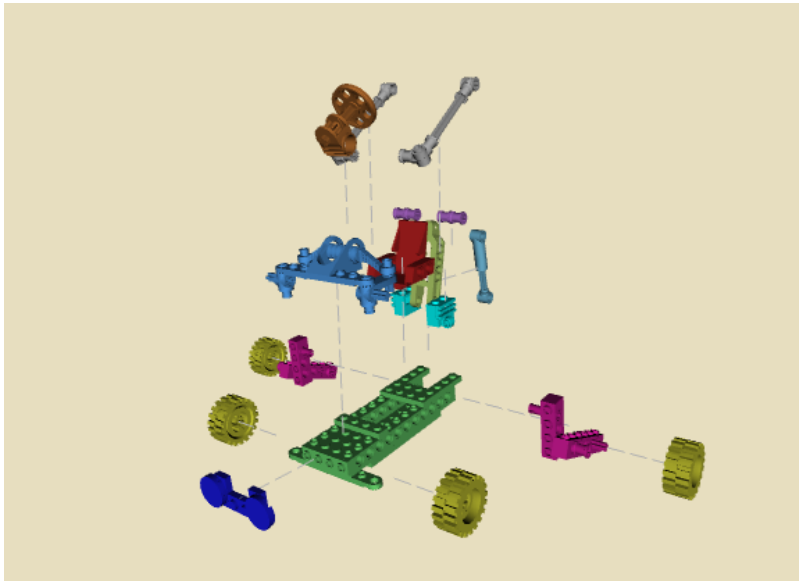


Figure 5.17: A general explosion resulting from the styles *AND/OR General Single Part*, *PCP Movable Focus Biggest Parent* and *PCA Biggest Parent/Child*. The minimal explosion state revealing the seat is shown. The worst case has happened. All parts are exploded before the focus element.

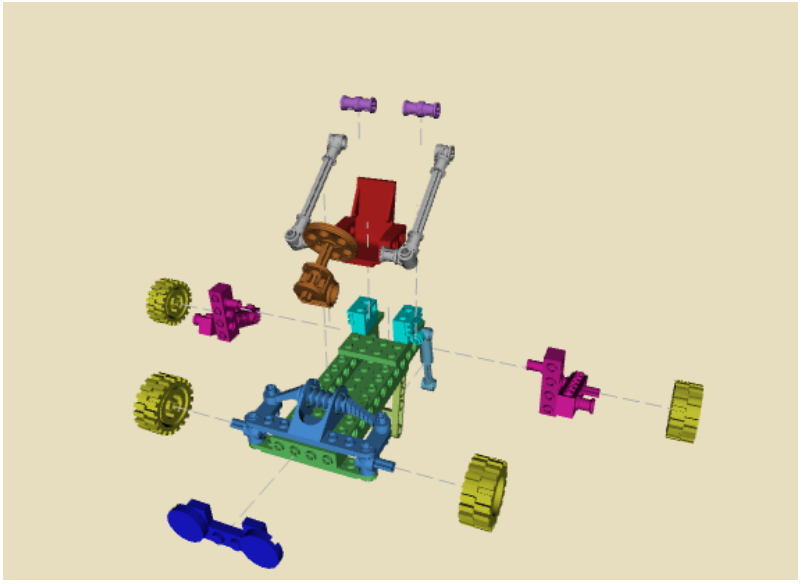


Figure 5.18: A symmetric explosion resulting from the styles *AND/OR Smallest First*, *PCP Movable Focus Biggest Parent* and *PCA Biggest Parent/Child*. The minimal explosion state revealing the seat is shown. The explosion is symmetric, but many parts having a different context than the seat are exploded.

seen in figure 5.19 the explosion is symmetric, although the parts of the same symmetric group are not removed sequentially. But more importantly the focus element is revealed by a smaller number of explosions than in the previously cases. But a problem of the used algorithm is visible. Still parts having no contextual or blocking relation to the focus element are exploded. The figure shows the explosion of the steering wheel, which does not block the seat in any way. In addition, exploding single parts may not necessary to reveal a focus element. The user might be more interested in the overall context, but not in details of how the surroundings are assembled. Nevertheless, the used AND/OR graph style shows, that the distance to a central part of a symmetric assembly can be used as measure for determining symmetric parts. Components of the same symmetric group will very likely have the same distance to a central part of the assembly.

Substituting the partition level `PC` relation style for *PCP Biggest Parent Prefer Static Focus* leads to an explosion of great extents. The rest of the unexploded assembly can be rather big in size and therefore is exploded a great distance away from the focus element as seen in figure 5.20. In general, the exploded rest of the assembly is moved away from the static focus, which may not be very appealing. Because the user is only interested in a certain part of the assembly, there is no need anymore to separate the parts by a large

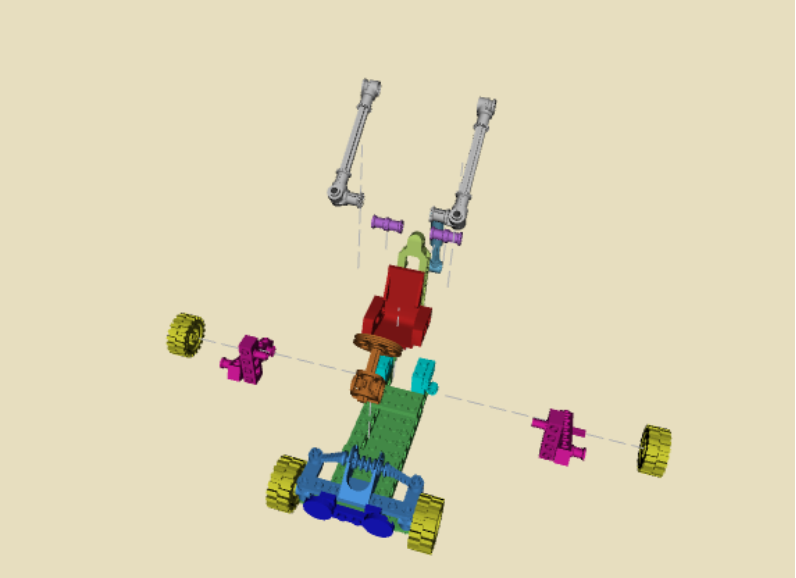


Figure 5.19: A focused explosion resulting from the styles *AND/OR Focus Nearest First*, *PCP Movable Focus Biggest Parent* and *PCA Biggest Parent/Child*. The smallest number of explosions revealing the seat is shown. The diagram is symmetric, but parts having a different context are exploded.

distance. Hence, a different separation direction style can be utilized to limit the extents of the explosion.

In figure 5.21 the previous separation direction style was exchanged for the style *SepDir Ten Percent Distance Except Focus Contacts* and the partition level `PC` style was switched back to *PCP Movable Focus Biggest Parent*. Now only parts in contact with the focus element, as well as the focus element itself are exploded to the full distance. The steering wheel still is moved, but this is not that visible. The user clearly sees, which parts were in contact with the focus element. When utilizing the style *SepDir Ten Percent Distance Except Focus*, all parts except the focus element are translated only a small distance. This loosens the surroundings of the focus element a little bit. Further removing the focus element allows an investigation of the elements surrounding it (see figure 5.22).

This loosening can also be achieved when the focus part can be removed in the first explosion step. In this case, the layered explosion stops, but the user can manually explode single parts further. The underlying algorithm still removes the elements nearest to the focus element first (see figure 5.23). If the assembly is fully exploded using this combination of styles, the complete model is loosened, with a focus element standing out. Exchanging the AND/OR graph style for the symmetric styles, or the general peeling style, the same

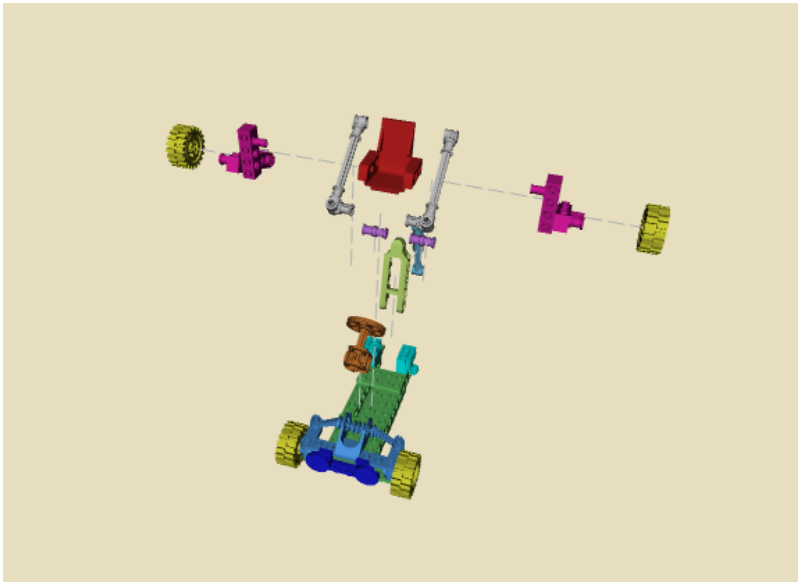


Figure 5.20: A focused explosion resulting from the styles *AND/OR Focus Nearest First*, *PCP Biggest Parent Prefer Static Focus* and *PCA Biggest Parent/Child*. The focused seat, stays static, the rest of the assembly is moved. Some context of the seat is removed, because the assembly is exploded a great distance.

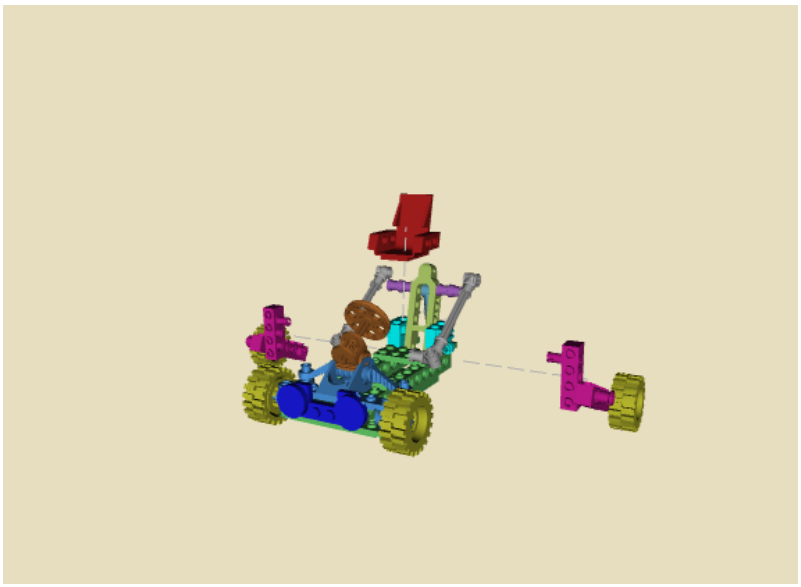


Figure 5.21: A focused explosion resulting from the styles *AND/OR Focus Nearest First*, *PCP Movable Focus Biggest Parent*, *PCA Biggest Parent/Child* and *SepDir Ten Percent Distance Except Focus Contacts*. The focus element and the parts in contact with the focus are fully exploded. Moving the rest only a small distance, the context becomes clearer.

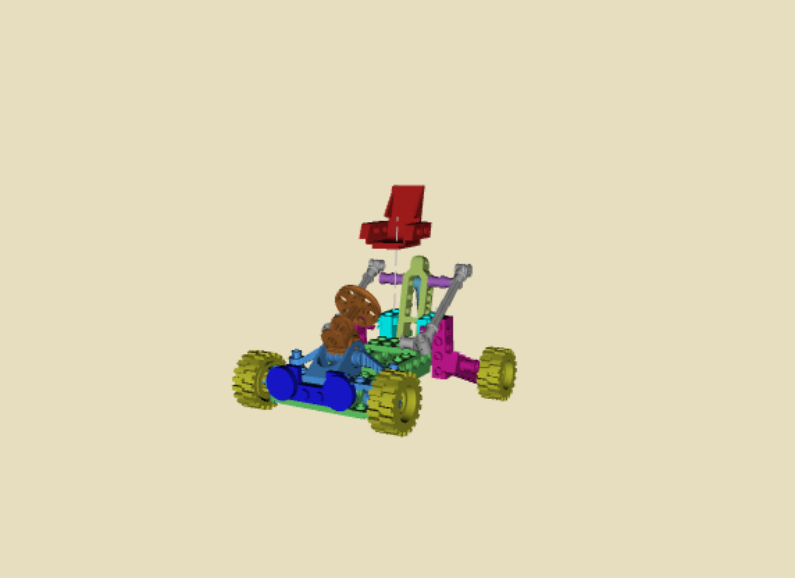


Figure 5.22: A focused explosion resulting from the styles *AND/OR Focus Nearest First*, *PCP Movable Focus Biggest Parent*, *PCA Biggest Parent/Child* and *SepDir Ten Percent Distance Except Focus*. Only the focus element is fully exploded. The rest is moved a small distance and loosens the surrounding assembly.

result can be achieved. Only the ordering of the part removal differs.

The previously described combination of *AND/OR Focus Nearest First*, *PCP Movable Focus Biggest Parent*, *PCA Biggest Parent/Child* and *SepDir Ten Percent Distance Except Focus* also works well when exploding the valve assembly, as seen in figure 5.24. The nearest first algorithm of the AND/OR graph style removes the green bolts and some screwnuts, which are not blocking the focus at all. But this is not that visible, because the distance of all non-focus parts is reduced. This approach leads to the problem that the bolt is not clearly separated from the blue wheel. Hence, the separation direction style *SepDir Ten Percent Distance Except Focus Contacts* should be used, leading to the explosion depicted in figure 5.25.

Reducing the distance of all parts except the focus element to zero results in an explosion of only the focus element. But if the focus element is parent to other parts, these parts still would move relative to the focus, thus hiding certain parts of it. Considering the explosion in figure 5.26, where the focus is set to the green base plate of the car, the focus would be completely hidden. Even in the depicted case, where the parts are moved by ten percent of their original distance, the focus is not really revealed. Using the style *SepDir Ten Percent Distance Except Focus Contacts* removes the parts hiding the floor

plate (see figure 5.27). But nearly all parts are exploded before the focus is revealed. In this figure, a problem of the used collision prevention algorithm is visible. The gray components are associated with the small purple components, because they originally moved into the same direction. In a later step these purple objects were exploded downwards with another component. To prevent such movements, stacks should be identified.

Changing the AND/OR graph style to *AND/OR Smallest First*, explodes even more parts than the previous style, as seen in figure 5.28. But by removing smaller parts before larger ones, the large gray objects become parents of the small purple parts. Thus, they are not moved downwards. If exploding a lot of parts is not appropriate to create a focused explosion diagram, it is better to consider a completely new AND/OR graph style, which will very likely reveal each hidden part, without splitting the model into small pieces. Hence, the style *AND/OR Focus Most Parts Partitions* is discussed next.

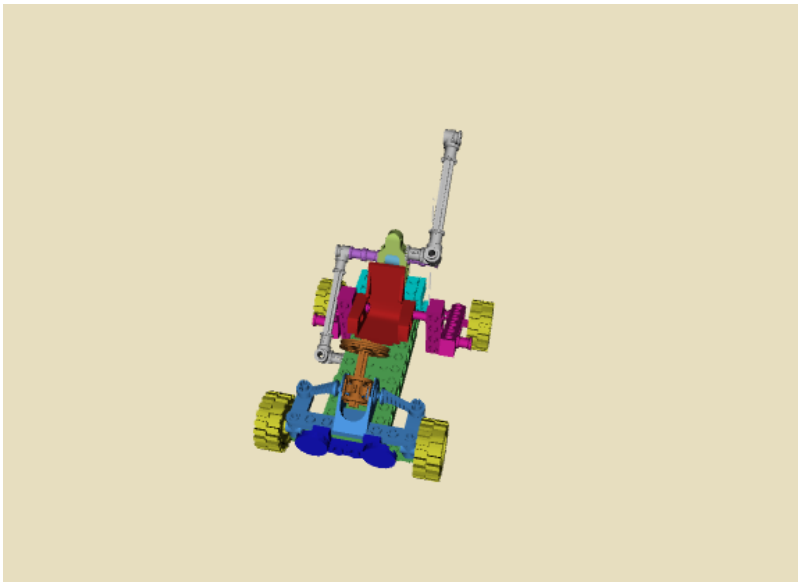


Figure 5.23: A focused explosion resulting from the styles *AND/OR Focus Nearest First*, *PCP Movable Focus Biggest Parent*, *PCA Biggest Parent/Child* and *SepDir Ten Percent Distance Except Focus*. The gray focus element is fully exploded in the first step, using the layered explosion. The rest can still be loosened by exploding single parts or all of it manually.

Most Parts Partitions

The style *AND/OR Focus Most Parts Partitions* creates partitionings into the biggest possible partitions, not containing the focus element. Hence, the focus is completely freed

Figure 5.24: A focused explosion resulting from the styles *AND/OR Focus Nearest First*, *PCP Movable Focus Biggest Parent*, *PCA Biggest Parent/Child* and *SepDir Ten Percent Distance Except Focus*. The bronze-colored focus bolt, which is moved out of the assembly. The removal of the green parts and some screwnuts is not visible because of the reduced distance. But the focus element is not clearly separated from the blue wheel.

Figure 5.25: A focused explosion resulting from the styles *AND/OR Focus Nearest First*, *PCP Movable Focus Biggest Parent*, *PCA Biggest Parent/Child* and *SepDir Ten Percent Distance Except Focus Contacts*. The bronze-colored focus bolt is clearly separated from the rest, because the contacting parts are moved the full distance.

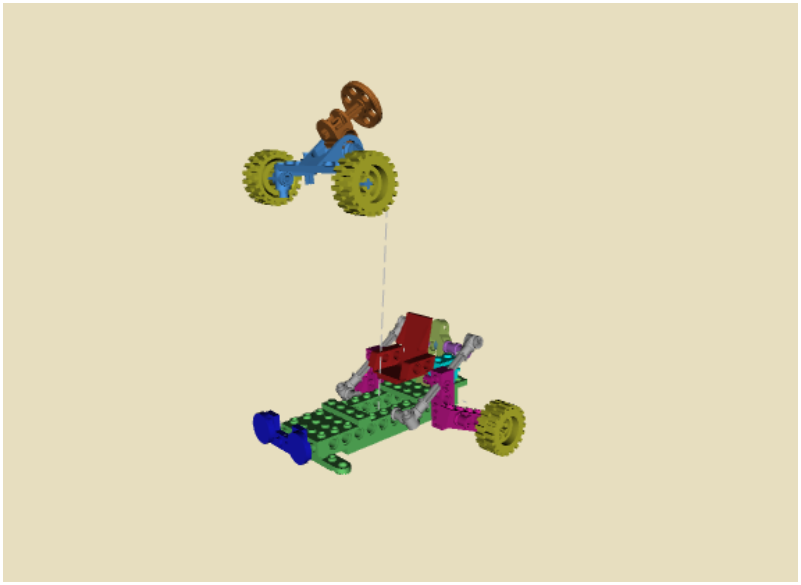


Figure 5.26: A focused explosion resulting from the styles *AND/OR Focus Nearest First*, *PCP Movable Focus Biggest Parent*, *PCA Biggest Parent/Child* and *SepDir Ten Percent Distance Except Focus*. The focused floor plate, is fully exploded, but it is still hidden by other components.

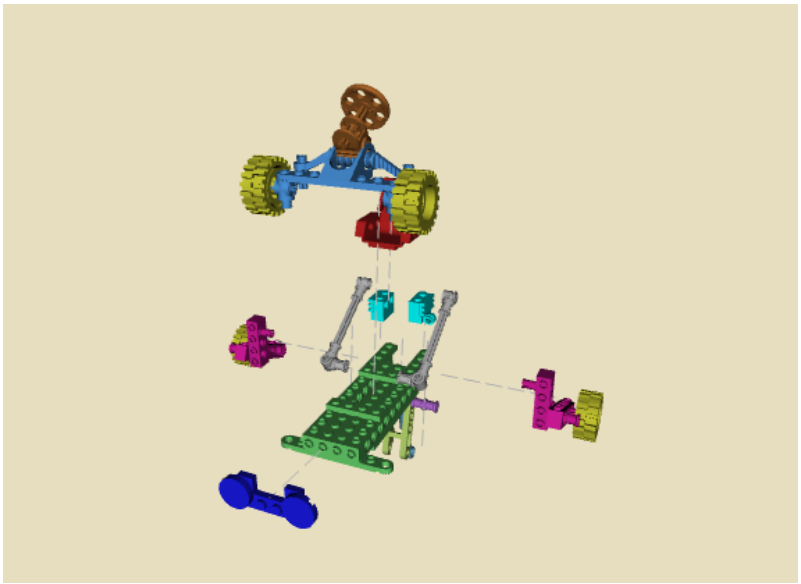


Figure 5.27: A focused explosion resulting from the styles *AND/OR Focus Nearest First*, *PCP Movable Focus Biggest Parent*, *PCA Biggest Parent/Child* and *SepDir Ten Percent Distance Except Focus Contacts*. The focused floor plate, and its direct neighbors are fully exploded. The green plate is fully visible from certain angles, but the model has to be fully exploded.

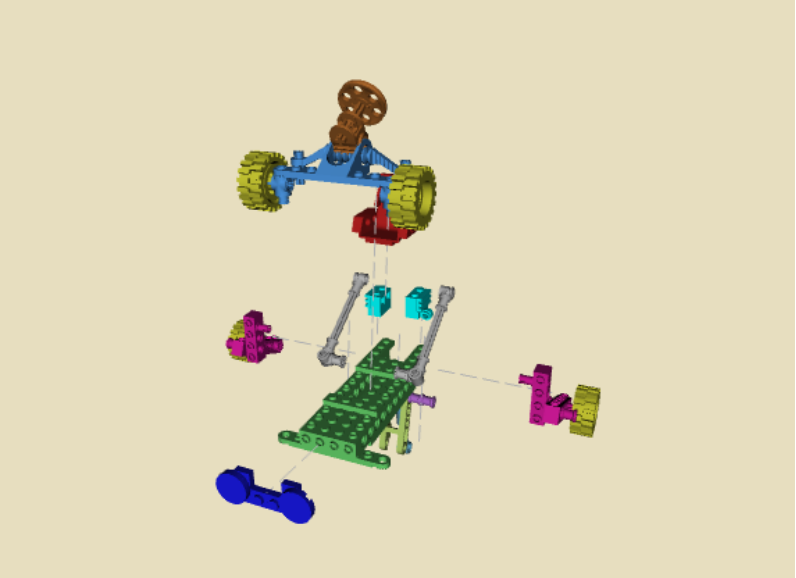


Figure 5.28: A symmetric explosion resulting from the styles *AND/OR Smallest First*, *PCP Movable Focus Biggest Parent*, *PCA Biggest Parent/Child* and *SepDir Ten Percent Distance Except Focus Contacts*. The focused floor plate, and its direct neighbors are fully exploded. Again, all parts are exploded to reveal the focus.

as fast as possible. The created partitions not containing the focus are then separated into single parts starting with the smallest. Combining the style with the partition level style *PCP Biggest Parent*, the part level style *PCA Biggest Parent/Child* and the separation direction style *SepDir Size Distance*, the focus part is separated from its surroundings, but it is not clear if it stays static or is exploded. Using the style *PCP Movable Focus Biggest Parent* assures that the focus is moved in each explosion. Nevertheless, changing the location of the focus may be confusing for the user, if he concentrates on it and wants to investigate it in more detail. Hence, the style *PCP Biggest Parent Prefer Static Focus* is used to keep the focus static.

Using the style *AND/OR Focus Most Parts Partitions* combination with *PCP Biggest Parent Prefer Static Focus*, *PCA Biggest Parent/Child* and *SepDir Size Distance* generates the explosion depicted in figure 5.29. Unlike using style *AND/OR Focus Nearest First*, the focus, in this case the floor plate, is clearly separated from the rest of the assembly. By changing the point of view, the user is able to investigate the focus.

Nevertheless, this combination has some flaws. Exploding the rear of the car creates the diagram illustrated in figure 5.30. The explosion is asymmetric, because a symmetric part of the exploded partition was chosen as child. Other objects are exploded away from

this child. Another problem can be seen in figure 5.31. Not all partitions are associated with the focus part. Hence, one of it moves relative to another partition, leading to another asymmetric explosion.

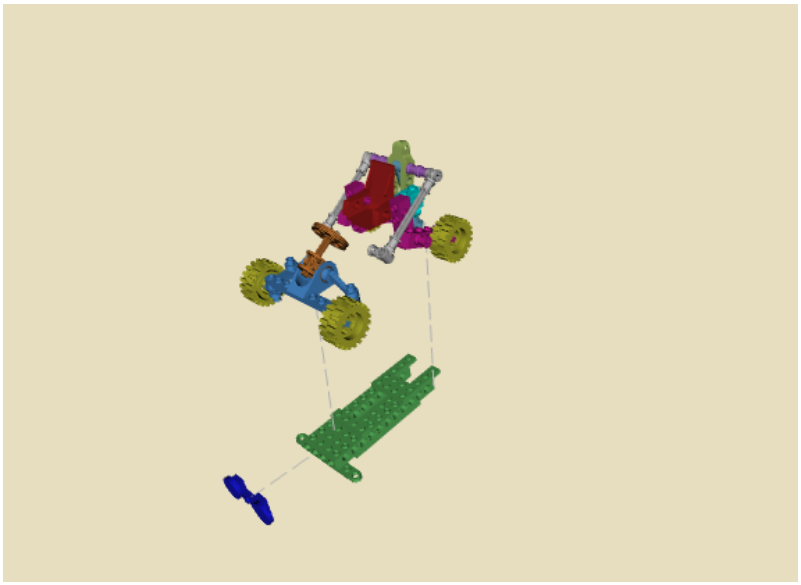


Figure 5.29: A focused explosion resulting from the styles *AND/OR Focus Most Parts Partitions*, *PCP Biggest Parent Prefer Static Focus*, *PCA Biggest Parent/Child* and *SepDir Size Distance*. The focused base plate, stays static and is revealed by removing the biggest possible partitions. Changing the point of view, the user is able to inspect the focus element.

Both problems are solved using the part level `PC` relation style *PCA Biggest Asymmetric Parent/Child Prefer Focus*. As the name implies the focus element is preferred when present in the parent partition. Therefore, all partitions are exploded away from the focus, not from other partitions. Additionally, if asymmetric parts are present in the child partition, the biggest one is selected to be the child of the whole partition. If the parent partition does not contain a focus element, the biggest asymmetric parent is chosen. Utilizing this style for the floor plate focus creates the explosion shown in figure 5.32. The rear of the car is exploded symmetrically. Focusing on the seat yields the diagram depicted in figure 5.33. Now the seat really is the center of the explosion.

Applying the combination of *AND/OR Focus Most Parts Partitions*, *PCA Biggest Asymmetric Parent/Child Prefer Focus*, *PCA Biggest Parent/Child* and *SepDir Size Distance* to the valve assembly with the bronze-colored bolt as focus element leads to the explosion diagram shown in figure 5.34. Unlike in figure 5.25 where all nearest parts were

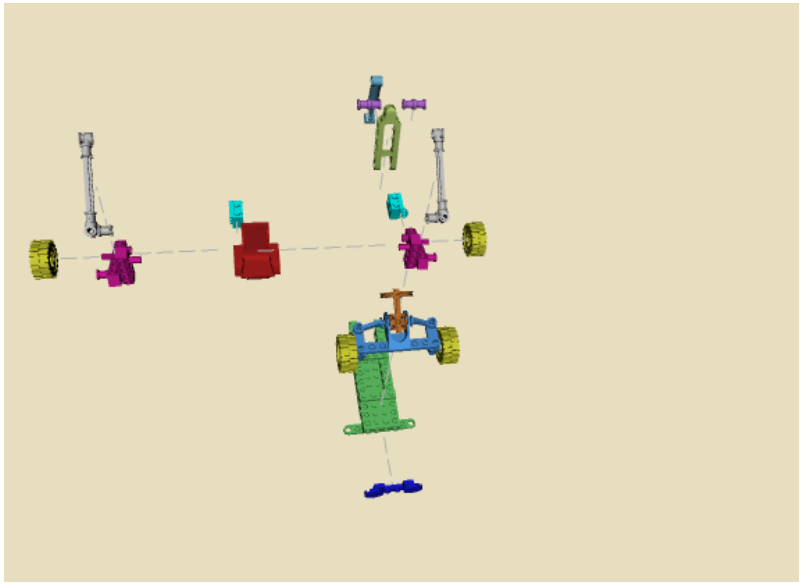


Figure 5.30: A symmetric explosion resulting from the styles *AND/OR Focus Most Parts Partitions*, *PCP Biggest Parent Prefer Static Focus*, *PCA Biggest Parent/Child* and *SepDir Size Distance*. The focused base plate, stays static. Exploding the rear of the car reveals that the `PC` relations are asymmetric.

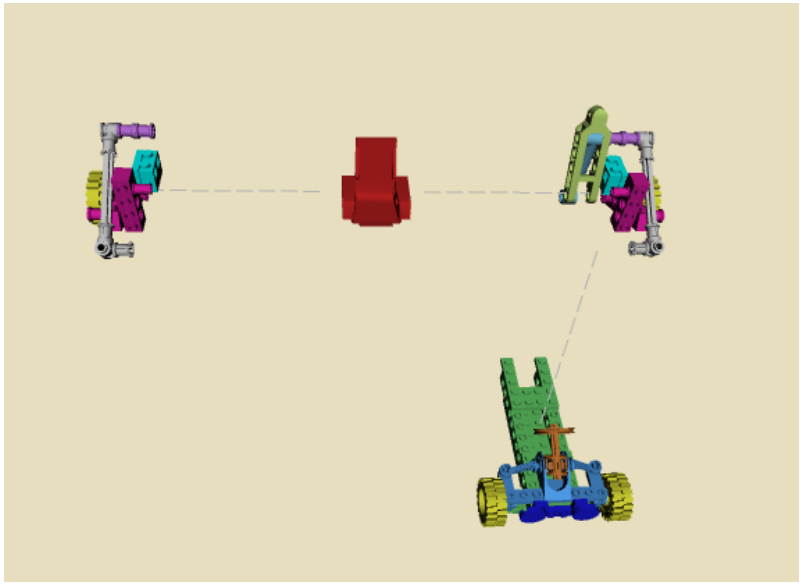


Figure 5.31: A focused explosion resulting from the styles *AND/OR Focus Most Parts Partitions*, *PCP Biggest Parent Prefer Static Focus*, *PCA Biggest Parent/Child* and *SepDir Size Distance*. The seat stays static. The partitions are associated with one another instead of the focus element, creating an asymmetric explosion.

exploded before the bolt was removed, now only the relevant partitions are expanded. Changing the focus element to a more central part, namely the black fixture, creates the explosion diagram depicted in figure 5.35. Although, the whole model appears to be exploded, only the minimal number of parts and partitions has been removed.

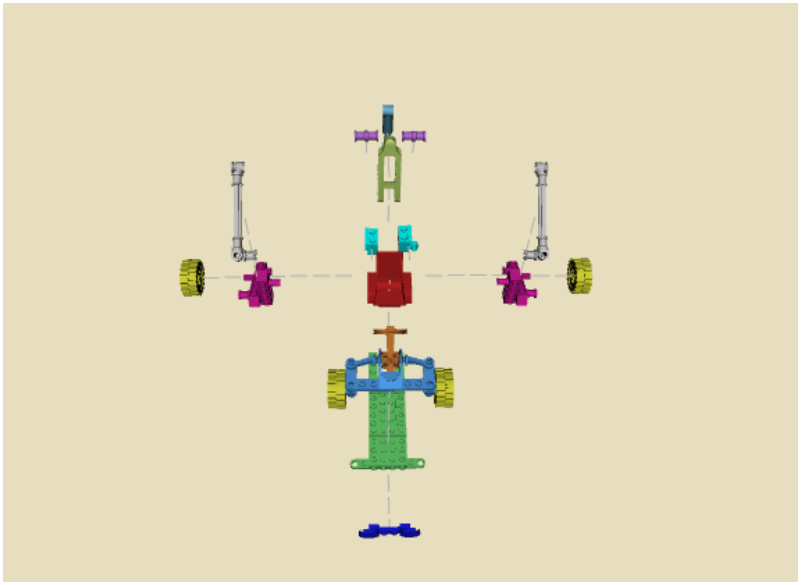


Figure 5.32: A focused explosion resulting from the styles *AND/OR Focus Most Parts Partitions*, *PCP Biggest Parent Prefer Static Focus*, *PCA Biggest Asymmetric Parent/Child Prefer Focus* and *SepDir Size Distance*. The focused base plate stays static. The rear of the car is exploded in a symmetric way.

5.3.5 Summary

Until now each AND/OR graph style was refined by combining it with different other styles. The resulting explosion diagrams were investigated applying the symmetrical criteria mentioned in the introduction to this section (see 5.3). This section presents combinations of certain styles to achieve a specific result.

In general, an explosion should be structured according to symmetrical criteria, which additionally reduces the extents of the explosion. Therefore, the symmetric AND/OR graph style *AND/OR Smallest First* should be used, which first of all introduces symmetry into the AND/OR graph of the explosion diagram. But symmetry also heavily depends on the employed `PC` relations. They ensure that symmetric parts are assigned to similar parent parts. In addition, either all objects of the same symmetric group move, or stay static. Hence, the styles *PCP Biggest Parent* should be combined with the part level style

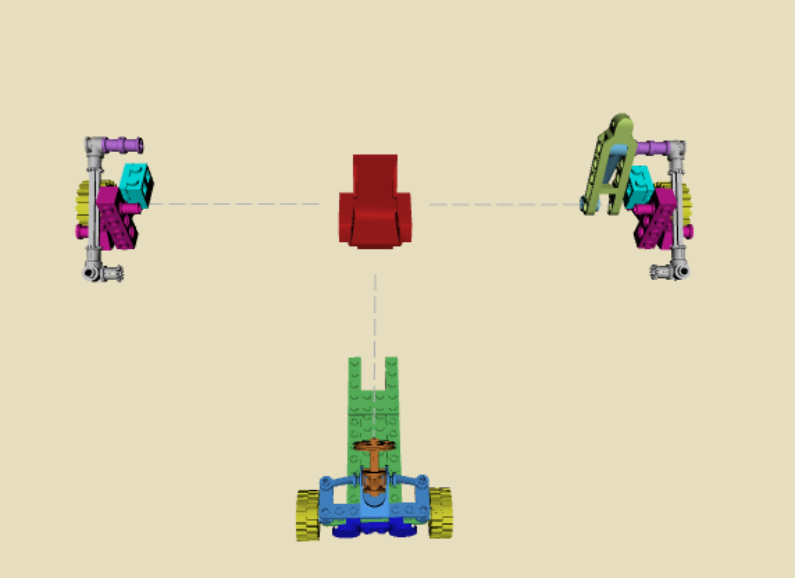


Figure 5.33: A focused explosion resulting from the styles *AND/OR Focus Most Parts Partitions*, *PCP Biggest Parent Prefer Static Focus*, *PCA Biggest Asymmetric Parent/Child Prefer Focus* and *SepDir Size Distance*. The focused seat stays static and is the center of the explosion.

Figure 5.34: A focused explosion resulting from the styles *AND/OR Focus Most Parts Partitions*, *PCP Biggest Parent Prefer Static Focus*, *PCA Biggest Asymmetric Parent/Child Prefer Focus* and *SepDir Size Distance*. The focused bronze-colored bolt stays static and is the center of the explosion.

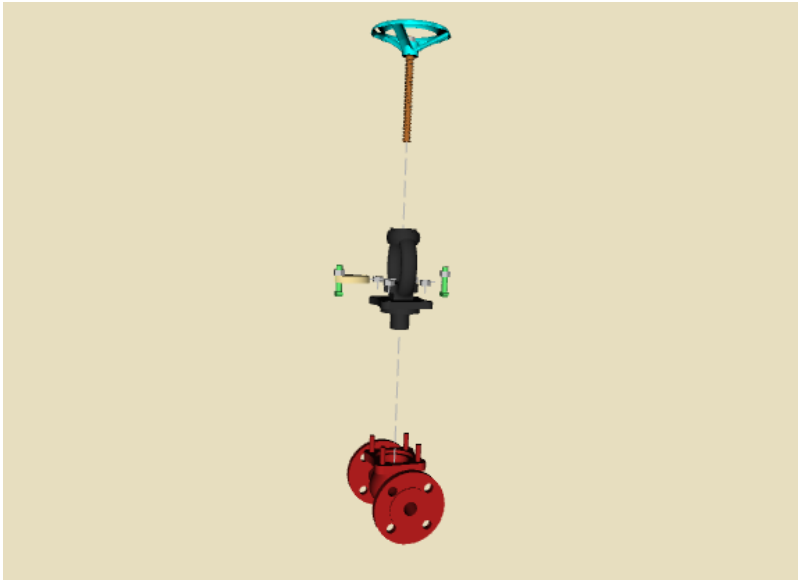


Figure 5.35: A focused explosion resulting from the styles *AND/OR Focus Most Parts Partitions*, *PCP Biggest Parent Prefer Static Focus*, *PCA Biggest Asymmetric Parent/Child Prefer Focus* and *SepDir Size Distance*. The focused black fixture stays static and is the center of the explosion. Nearly the whole model is exploded.

PCA Biggest Parent/Child. This combination is the preferred choice, because it follows the requirement, that smaller and thus less significant parts move relative to larger parts. The style *Ordering Symmetric Layers* provides the user with the ability to explode the symmetrical parts layer-by-layer. Nevertheless, because symmetry depends on the size of the objects, asymmetric parts of similar size may be exploded in the same layer. Using the separation direction style *SepDir Size Distance* creates an explosion, where parts are clearly removed from each other. To achieve only a loosening of the assembly, maybe to make part selection easier, the separation direction style *SepDir Ten Percent Distance* can be used.

Using the part level `PC` relation style *PCA Biggest Parent* follows the same approach as *PCA Biggest Parent/Child*, but does not ensure that the smallest child is chosen, if the child partition consists of more than one part. In fact, the style *PCA Biggest Parent/Child* should be enhanced to include the consideration of asymmetric parts as in *PCA Biggest Asymmetric Parent/Child Prefer Focus*. Otherwise, it is not assured that only asymmetric parts are selected as child parts out of child partitions consisting of more than one component. By coincidence, finding asymmetric child parts was not required for the used examples, because exploding smallest single parts with *AND/OR Smallest First*

and the combination with the style *PCP Biggest Parent*, merely produced multiple part child partitions. Therefore, each exploded symmetrical single part partition was assigned as child to different symmetrical parent parts or the same asymmetric parent part.

If the explosion is focused on a certain object, it is advisable to use the ordering style *Ordering Reveal Focus*, which explodes the assembly until the focus part is freed from its surroundings. In general, there exist three applicable structures for the AND/OR graph, hence also three corresponding AND/OR graph styles. First, a pure symmetric structure created by *AND/OR Smallest First* can be used, in combination with *PCP Movable Focus Biggest Parent*, *PCA Biggest Parent/Child* and *SepDir Ten Percent Distance Except Focus Contacts*. When the assembly is fully exploded, this combination ensures a symmetric explosion, where the focus part is moved away from its parent part and its children are translated away from the focus. The rest of the assembly is only exploded by a small amount. The focus element can be investigated thoroughly without loosing its surrounding context.

Another focused explosion can be created by using the focused AND/OR graph structure of *AND/OR Focus Nearest First* combined with *PCP Movable Focus Biggest Parent*, *PCA Biggest Parent/Child* and *SepDir Ten Percent Distance Except Focus*. Although it has been shown, that the resulting explosion diagram can be symmetric using this AND/OR graph style, it is not assured. Nevertheless, when performing a focused explosion, a user might not really be interested in the rest of the assembly, but rather in the surroundings of the focus element and the focus itself. Utilizing this combination offsets the focus element from the rest of the assembly, while loosening the surroundings. This is similar to the previous approach, but in this case only a small part of parts is exploded to reveal the focus element.

If a user is not at all interested in the elements surrounding the focus, it can be completely freed by utilizing *AND/OR Focus Most Parts Partitions*, while keeping the focus itself static using *PCP Biggest Parent Prefer Static Focus* and *PCA Biggest Asymmetric Parent/Child Prefer Focus*. The resulting explosion is centered around the focus element, which is parent to all moved partitions. Using the separation direction style *SepDir Ten Percent Distance Except Focus Contacts* the focus element is clearly separated. At least if all partitions are of appropriate size. Because the separation distance is proportional to the diagonal of the moved partition, it can happen that small partitions are not translated far enough to provide a clear view on the focus. To handle such cases, a minimal distance could be introduced.

5.4 Visualization Methods

To support the user in following the progress of the explosion, animations are employed. In addition, lines and static blurs were used to mark the parents of the exploded assembly parts, thereby constructing a hierarchy and helping the user to mentally reconstruct the assembly. Projecting the 3D model on its real world object counterpart allows an enhancement of the visualization by assigning the real world texture to the virtual objects. In the following animations and the approach of using guide lines and static blurs are discussed in more detail.

5.4.1 Animation

Providing animated transitions of an object from its initially unexploded into the exploded state, helps the user to follow the motion of the object. According to [23], using animations reduces the time a user requires to adapt to the new state.

Aside from using animations, another approach is taken to improve the users' understanding of the relationships among the parts and the impact of the different style combinations. The model has to be brought back into its initially assembled state, before applying a new style.

5.4.2 Lines and Static Blur

The implemented methods for visualizing part movements and part relations use the calculated `PC` relations to decide which parts should be connected. Figure 5.36 shows an example, where relationships are not outlined. It is not clear, which parts are associated with each other and where they came from. Therefore, this thesis implements simple guide lines, reaching from the center of the bounding box of each parent to the center of its children. Employing this lines into the previous explosion greatly enhances the ability to reconstruct the assembly (see figure 5.37). There, a problem of this simple approach becomes apparent. The guide line connecting the seat to the floor plate penetrates the seat to reach the bounding box center. In addition, the recognition of parents is ambiguous, as in the marked case. The small part is exploded downwards from its original location, which is indicated by the line. But it is not clear, which one of the neighboring parts is the parent. Nevertheless, is may be more important to visualize the original location, relative to other parts, than the actual parent of the explosion.

Using static blurs instead of lines can further enhance the ability to reconstruct an

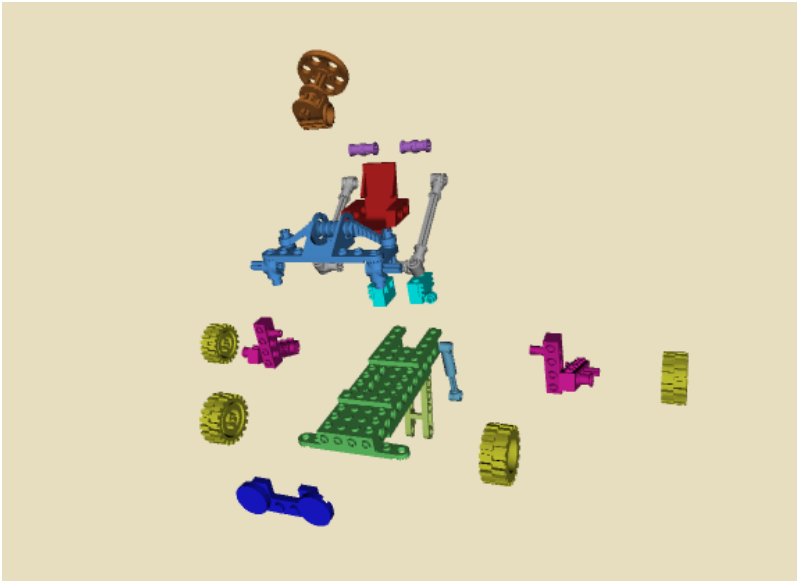


Figure 5.36: A symmetric explosion resulting from the styles *AND/OR Smallest First*, *PCP Biggest Parent*, *PCA Biggest Parent/Child* and *SepDir Size Distance*. Without guidelines the relationships between the parts and the original position are hard to reconstruct.

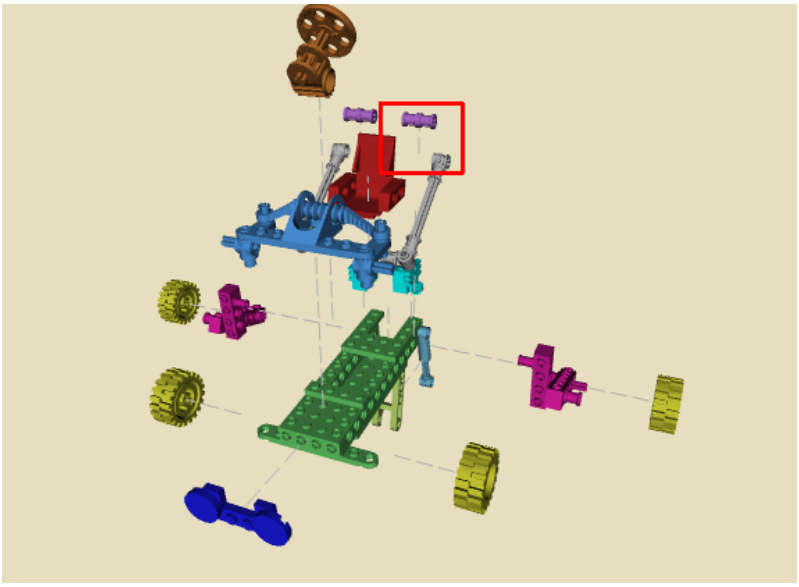


Figure 5.37: A symmetric explosion resulting from the styles *AND/OR Smallest First*, *PCP Biggest Parent*, *PCA Biggest Parent/Child* and *SepDir Size Distance*. Guidelines show the original location of the part relative to its parent part. But it may not be clear, which one is the parent.

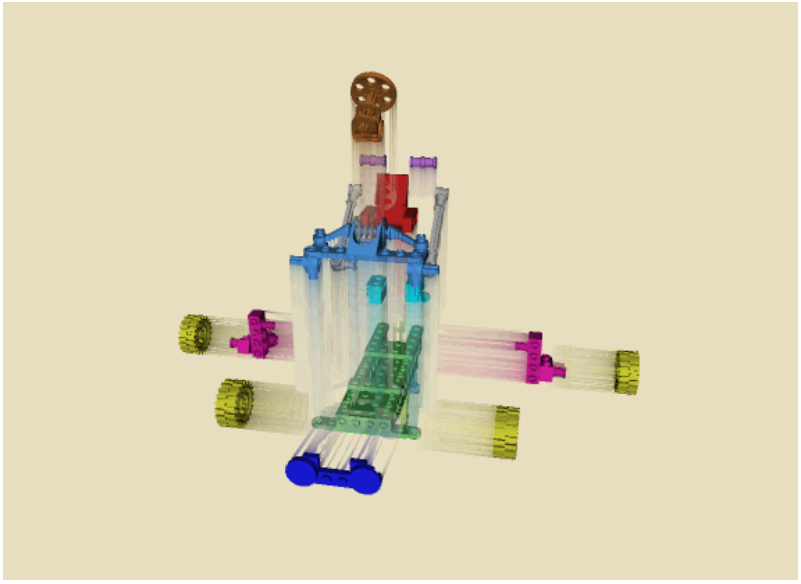


Figure 5.38: A symmetric explosion resulting from the styles *AND/OR Smallest First*, *PCP Biggest Parent*, *PCA Biggest Parent/Child* and *SepDir Size Distance*. Static blurs indicate the part movement relative to parent parts.

exploded assembly. Figure 5.38 shows the same explosion, but now the motion is visualized by stretching the object to its original location and blurring it. The blur clearly marks the path, an object would take to reach its original location. Additionally, perspective information is contained in the blur trail. If an object moves to the back, the blur gets thinner until it reaches the object. In general, when viewing a blur trail, users are able to draw conclusions about the associated object or at least its dimensions. A problem with using blurs is that objects located behind such the trail also appear blurry. Furthermore, when blur trails overlap, it may be hard to relate them to the correct objects.

Another advantage of blurs becomes visible when changing the point of view and reducing the explosion distance to a small one. The spot marked in figure 5.40, shows the blur trail of an object. Hence, the image contains information about an object which was situated there before the explosion. Using guide lines this information is not present, as seen in figure 5.39. There can be seen, that for this small distance, the lines are not even shown. But even if they would, there is no clue about the removed geometry at the back, because the center of the bounding box was not visible.

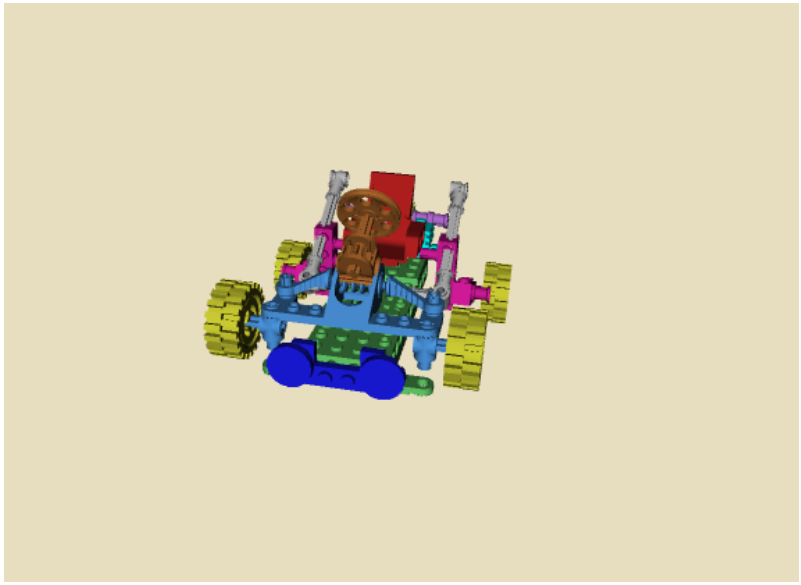


Figure 5.39: The seat is the focus. For small distances guide lines are not visible.

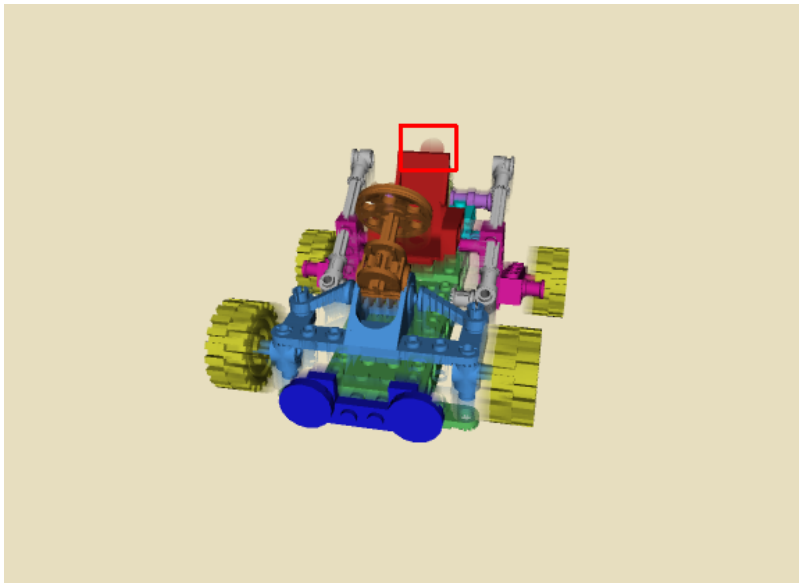


Figure 5.40: The seat is the focus. The marked static blur shows that there was an object behind the seat.

Chapter 6

Conclusion

In this thesis, a general framework for automatically creating explosion diagrams from a 3D model was presented. Four parameters influencing the interactive presentation, as well as the layout of the explosion were identified. First, the underlying explosion sequence defining the partitioning of the assembly can be selected appropriately to follow for example symmetrical considerations, thus removing parts of the same type one after another. Second, the determined sequence is supported by `PC` relations, which enable the creation of a size-based hierarchy where small parts are moved relative to bigger ones. Furthermore, a focus element can either be moved or kept static. Third, separation directions and distances may also be influenced by symmetrical considerations, thus moving less significant parts a smaller distance than more significant ones. Taking into account a focus element, its direct neighbors may move a larger distance, than parts not being in contact. This way, the context of the focus can be made more visible. A fourth parameter is able to introduce layers into the interactive explosion. Hence, a user is able to explode all parts until a focus element is revealed, or parts of the same symmetrical group are exploded in the same layer. In general, the interactive system allows the explosion of single parts in an ordering prescribed by the explosion sequence, a layered explosion or the creation of a fully exploded view.

To support the ability of the user to mentally reconstruct the exploded assembly, several visualization aids are implemented. By providing animations, the user can follow the explosion of the parts, making it easier to adapt to the explosion layout. The relationships among the parts are visualized by either simple guide lines or a static blur, where the part is stretched to its original location relative to the parent and then blurred.

The system can be enhanced in different ways. The hybrid approach for testing the

separability of partitions presented in section 3.2.2 should use true local blocking constraints instead of simplified global ones. In the current implementation, there can occur situations where the used local and global blocking constraints are the same, thus preventing the separation of two partitions. Another enhancement is the investigation of contextual symmetry, thus also taking into account the parts neighboring potential symmetric objects. Currently, all parts of similar size, which are removable in the same step are defined to be symmetrical. Therefore, it may happen that parts having by coincidence the same size are identified as symmetrical parts. Contextual symmetry could improve the identification of size-based symmetry by taking into account the number of neighbors and their sizes.

This thesis presented two example applications for the framework. Parameters were identified leading to a symmetrical explosion diagram, as well as an explosion focused on a single part. For more sophisticated applications, rules should be identified, allowing the creation of task-dependent layouts. The created styles should be evaluated in a user study to support the symmetrical layout considerations described in this thesis. Furthermore, the impact of the static blur should be part of this evaluation.

Appendix A

Acronyms and Symbols

List of Acronyms

AND/OR	AND/OR graph
API	Application Programming Interface
AR	augmented reality
CAD	computer aided design
CMG	component mating graph
CPU	central processing unit
c-space	configuraton space
DFS	depth first search
dof	degrees of freedom
DBG	directional blocking graph
DT	disassembly tree
FAG	face adjacency graph
FPS	frames per second
GB	Giga Byte
geom	geometrical object
GLSL	OpenGL Shading Language
IPSA	Inventor Physics Simulation API
KB	Kilo Byte
LTB	local translational blocking
LTF	local translational freedom
MB	Mega Byte

NDBG	non-directional blocking graph
PN	Petri Net
PC	parent-child
PCA	parent-child assembly part
PCP	parent-child partition
ODE	Open Dynamics Engine
RAM	Random Access Memory
SepDir	separation direction
STAAT	Stanford Assembly Analysis Tool
STEP	Standard for the Exchange of Product Model Data

Appendix B

Style Combinations

The table contained in this appendix shows the used combinations of AND/OR graph and *PCP*, as well as *PCA* styles. Because separation direction and ordering styles do not change the underlying explosion hierarchy created by the previous styles, they are not mentioned. The names of the styles are the same as defined in chapter 5.

The tables B.3, B.4, B.5, B.6 and B.7 describe the possible combinations of the different AND/OR graph styles with all *PCP* and *PCA* styles. To get all `PC` relation style names into one table, only acronyms are used. Table B.1 shows the acronyms for the *PCP* styles, while table B.2 shows the ones for the *PCA* styles.

PCP styles	
Full name	Acronym
PCP First Hit	*FH*
PCP Biggest Parent	*BP*
PCP Most Parts Parent	*MPP*
PCP Movable Focus Biggest Parent	*MFBP*
PCP Movable Focus Most Parts Parent	*MFMPP*
PCP Biggest Parent Prefer Static Focus	*BPPSF*

Table B.1: The *PCP* style names are shortened to their acronyms.

PCA styles	
Full name	Acronym
PCA First Hit	*FH*
PCA Smallest Parent	*SP*
PCA Biggest Parent	*BP*
PCA Biggest Parent/Child	*BPC*
PCA Biggest Parent/Child Prefer Focus	*BPCPF*
PCA Biggest Asymmetric Parent/Child Prefer Focus	*BAPCPF*

Table B.2: The *PCA* style names are shortened to their acronyms.

AND/OR First Hit						
PCA / *PCP*	*FH*	*SP*	*BP*	*BPC*	*BPCPF*	*BAPCPF*
FH	5.3,5.6					
BP	5.4			5.5, 5.7		
MPP						
MFBP				5.16		
MFMPP						
BPPSF						

Table B.3: The table shows all possible combinations of *AND/OR First Hit* with *PCP* and *PCA* styles. The numbers indicate the figures in this thesis.

AND/OR General Single Part						
PCA / *PCP*	*FH*	*SP*	*BP*	*BPC*	*BPCPF*	*BAPCPF*
FH	5.8					
BP	5.9	5.10		5.11, 5.12		
MPP						
MFBP				5.17		
MFMPP						
BPPSF						

Table B.4: The table shows all possible combinations of *AND/OR General Single Part* with *PCP* and *PCA* styles. The numbers indicate the figures in this thesis.

AND/OR Smallest First						
PCA / *PCP*	*FH*	*SP*	*BP*	*BPC*	*BPCPF*	*BAPCPF*
FH						
BP		5.14		5.13, 5.15, 5.37, 5.38		
MPP						
MFBP				5.18, 5.28		
MFMPP						
BPPSF						

Table B.5: The table shows all possible combinations of *AND/OR Smallest First* with *PCP* and *PCA* styles. The numbers indicate the figures in this thesis.

AND/OR Focus Nearest First						
PCA / *PCP*	*FH*	*SP*	*BP*	*BPC*	*BPCPF*	*BAPCPF*
FH						
BP						
MPP						
MFBP				5.19, 5.21, 5.22, 5.23, 5.24, 5.25, 5.26, 5.27		
MFMPP						
BPPSF				5.20		

Table B.6: The table shows all possible combinations of *AND/OR Focus Nearest First* with *PCP* and *PCA* styles. The numbers indicate the figures in this thesis.

AND/OR Focus Most Parts Partitions						
PCA / *PCP*	*FH*	*SP*	*BP*	*BPC*	*BPCPF*	*BAPCPF*
FH						
BP						
MPP						
MFBP						
MFMPP						
BPPSF				5.29, 5.30, 5.31		5.32, 5.33, 5.34, 5.35

Table B.7: The table shows all possible combinations of *AND/OR Focus Most Parts Partitions* with *PCP* and *PCA* styles. The numbers indicate the figures in this thesis.

Bibliography

[1] Agarwal, P. K., de Berg, M., Halperin, D., and Sharir, M. (1996). Efficient generation of k-directional assembly sequences. In *Symposium on Discrete Algorithms*, pages 122–131, Philadelphia, PA, USA. Society for Industrial and Applied Mathematics.

[2] Agrawala, M., Phan, D., Heiser, J., Haymaker, J., Klingner, J., Hanrahan, P., and Tversky, B. (2003). Designing effective step-by-step assembly instructions. *ACM Trans. Graph.*, 22(3):828–837.

[3] Bruckner, S. and Gröller, M. E. (2006). Exploded Views for Volume Data. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):1077–1084.

[4] Cao, T. and Sanderson, A. (1991). Task Sequence Planning in a Robot Workcell using AND/OR Nets. In *IEEE International Symposium on Intelligent Control*, pages 239–244.

[5] Carpendale, M. S. T., Cowperthwaite, D. J., and Fracchia, F. D. (1996). Distortion Viewing Techniques for 3-Dimensional Data. In *IEEE Symposium on Information Visualization*, pages 46–53,119.

[6] Carpendale, M. S. T., Cowperthwaite, D. J., and Fracchia, F. D. (1997). Extending Distortion Viewing from 2D to 3D. *IEEE Computer Graphics and Applications*, 17(4):42–51.

[7] Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2001). *Introduction to Algorithms*. MIT Press.

[8] Driskill, E. and Cohen, E. (1995). Interactive Design, Analysis, and Illustration of Assemblies. In *Symposium on Interactive 3D Graphics*, pages 27–34, New York, NY, USA. ACM Press.

[9] Goldman, A. J. and Tucker, A. W. (1956). Polyhedral Convex Cones. In Kuhn, H. W. and Tucker, A. W., editors, *Linear Inequalities and Related Systems*, pages 19 – 40. Princeton University Press.

[10] Green, S. (2003). Stupid OpenGL Shader Tricks. In *Game Development Conference 2003*.

[11] Guibas, L., Halperin, D., Hirukawa, H., Latombe, J.-C., and Wilson, R. (1995). A Simple and Efficient Procedure for Polyhedral Assembly Partitioning under Infinitesimal Motions. In *IEEE International Conference on Robotics and Automation*, volume 3, pages 2553–2560.

[12] Halperin, D., Latombe, J.-C., and Wilson, R. H. (1998). A General Framework for Assembly Planning: The Motion Space Approach. In *Symposium on Computational Geometry*, pages 9–18.

[13] Halperin, D. and Wilson, R. H. (1995). Assembly Partitioning along Simple Paths: the Case of Multiple Translations. In b, editor, *IEEE International Conference on Robotlcs and Automation*, pages 1585–1592.

[14] Homem de Mello, L. and Sanderson, A. (1986). AND/OR Graph Representation of Assembly Plans. Technical Report CMU-RI-TR-86-08, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA.

[15] Homem de Mello, L. and Sanderson, A. (1991). A Correct and Complete Algorithm for the Generation of Mechanical Assembly Sequences. In *IEEE Transaction on Robotics and Automation*, volume 7, pages 228–240.

[16] Kaufman, S., Wilson, R., Jones, R., Calton, T., and Ames, A. (1996a). LDRD Final Report - Automated Planning and Programming of Assembly of Fully 3D Mechanisms. Technical report, Sandia National Laboratories.

[17] Kaufman, S., Wilson, R., Jones, R., Calton, T., and Ames, A. (1996b). The Archimedes 2 Mechanical Assembly Planning System. In *IEEE International Conference on Robotics and Automation*, volume 4, pages 3361–3368.

[18] Kurzweil, P., Frenzel, B., and Gebhard, F. (2008). *Physik Formelsammlung*. Vieweg.

[19] Latombe, J. and Wilson, R. H. (1995). Assembly Sequencing with Toleranced Parts. In *Third Symposium on Solid Modeling and Applications*, pages 83–94.

[20] Latombe, J.-C. (1991). *Robot Motion Planning*. Kluwer Academic Publishers, Norwell, MA, USA.

[21] Li, W., Agrawala, M., Curless, B., and Salesin, D. (2008). Automated Generation of Interactive 3D Exploded View Diagrams. In *SIGGRAPH 2008*.

[22] Li, W., Agrawala, M., and Salesin, D. (2004). Interactive Image-Based Exploded View Diagrams. In *Graphics Interface*, pages 203–212.

[23] McGuffin, M., Tancau, L., and Balakrishnan, R. (2003). Using Deformations for Browsing Volumetric Data. In *Visualization, 2003. VIS 2003. IEEE*, pages 401–408.

[24] Mohammad, R. and Kroll, E. (1993). Automatic Generation of Exploded Views by Graph Transformation. In *Artificial Intelligence for Applications, 1993. Proceedings., Ninth Conference on*, pages 368–374.

[25] Mok, S., Ong, K., and Wu, C.-H. (2001). Automatic Generation of Assembly Instructions using STEP. In *Proc. ICRA Robotics and Automation IEEE International Conference on*, volume 1, pages 313–318 vol.1.

[26] Moore, K., Gungor, A., and Gupta, S. (1998a). Disassembly Petri Net Generation in the Presence of XOR Precedence Relationships. In *IEEE International Conference on Systems, Man, and Cybernetics*, volume 1, pages 13–18vol.1.

[27] Moore, K., Gungor, A., and Gupta, S. (1998b). Disassembly Process Planning using Petri Nets. In *IEEE International Symposium on Electronics and the Environment*, pages 88–93.

[28] Niederauer, C., Houston, M., Agrawala, M., and Humphreys, G. (2003). Non-Invasive Interactive Visualization of Dynamic Architectural Environments. In *Symposium on Interactive 3D Graphics*, pages 55–58, New York, NY, USA. ACM Press.

[29] Preim, B., Deussen, O., and Ritter, F. (1999). Interaktive Zusammensetzung von 3D-Modellen zur Unterstützung des räumlichen Verständnisses. In O. Deussen, V. Hinz, P. L., editor, *Simulation und Visualisierung*, pages 109–126. In German.

[30] Raab, A. (1998). *Techniken zur Interaktion mit und Visualisierung von geometrischen Modellen*. PhD thesis, Otto-von-Guericke Universität Magdeburg, Germany. In German.

[31] Rist, T., Krüger, A., Schneider, G., and Zimmermann, D. (1994). AWI: A Workbench for Semi-Automated Illustration Design. In *Advanced Visual Interfaces*, pages 59–68, New York, NY, USA. ACM Press.

[32] Ritter, F., Preim, B., Deussen, O., and Strothotte, T. (2000). Using a 3D Puzzle as a Metaphor for Learning Spatial Relations. In *Graphics Interface*, pages 171–178. Morgan Kaufmann Publishers.

[33] Romney, B. (1997). *On the Concurrent Design of Assembly Sequences and Fixture*. PhD thesis, Stanford University, Stanford, CA, USA.

[34] Romney, B., Godard, C., Goldwasser, M., and Ramkumar, G. (1995). An Efficient System for Geometric Assembly Sequence Generation and Evaluation. In *ASME Int. Computers in Engineering Conference*, pages 699–712.

[35] Schwarzer, F. (2000). *Geometric Reasoning About Translational Motions*. PhD thesis, Technische Universität München.

[36] Smith, R. (2006). Open Dynamics Engine V0.5 User Guide. Technical report.

[37] Sonnet, H., Carpendale, S., and Strothotte, T. (2004). Integrating Expanding Annotations with a 3D Explosion Probe. In *Advanced Visual Interfaces*, pages 63–70, New York, NY, USA. ACM Press.

[38] Thomas, U., Barrenscheen, M., and Wahl, F. (2003). Efficient Assembly Sequence Planning Using Stereographical Projections of C-Space Obstacles. In *IEEE International Symposium on Assembly and Task Planning*, pages 96–102.

[39] Wilson, R. (1993). Minimizing User Queries in Interactive Assembly Planning. In *IEEE International Conference on Robotics and Automation*, pages 322–327.

[40] Wilson, R. H. (1992). *On Geometric Assembly Planning*. PhD thesis, Stanford University, Stanford, California.

[41] Wolter, J. D. (1989). On the Automatic Generation of Assembly Plans. In *IEEE International Conference on Robotics and Automation*, volume 1, page pp. 6268.

[42] Woo, T. C. and Dutta, D. (1990). Automatic Disassembly and Total Ordering in Three Dimensions. Technical report, University of Michigan.