Computational Tools for Assembly Oriented Design

by

Stephen J. Rhee

B.S. Engineering and Applied Sciences
California Institute of Technology, 1994

M.S. Mechanical Engineering
Massachusetts Institute of Technology, 1996

SUBMITTED TO THE DEPARTMENT OF MECHANICAL ENGINEERING IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF MECHANICAL ENGINEER at the MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 1999

© 1999 Massachusetts Institute of Technology
All rights reserved

Signature of Author ____________________________
Department of Mechanical Engineering
August 23, 1999

Certified by ____________________________
Daniel E. Whitney
Senior Research Scientist
Center for Technology, Policy, and Industrial Development
Lecturer, Department of Mechanical Engineering
Thesis Supervisor

Accepted by ____________________________
Ain A. Sonin
Chairman, Departmental Committee on Graduate Students
Computational Tools for Assembly Oriented Design

by

Stephen J. Rhee

Submitted to the Department of Mechanical Engineering on August 23, 1999 in Partial Fulfillment of the Requirements for the Degree of Mechanical Engineer

ABSTRACT

It has been shown that it is necessary to consider manufacturing and assembly issues during the design phase of product development for numerous reasons. Doing so will allow for a better designed product in terms of ease and cost of manufacture and assembly in addition to a more robust product that best satisfies its intended functionality. It is the goal of assembly oriented design (AOD) methods to aid the designer in taking assembly issues into consideration during the design process to produce a design that can be better manufactured and assembled. The AOD methodology also allows for the analysis of existing designs in addition to being able to indicate possible areas of improvements in a design.

The goal of this research is to develop an integrated suite of computational tools for assembly oriented design. This involves first studying the flow of design information in the AOD approach. Then, a formal design process can be outlined. Previously developed theory and software tools and methods exist for conducting assembly oriented design. This includes the datum flow chain assembly model for use in the design of assemblies and software tools to generate geometric precedence constraints on assembly sequences (SPAS), determine all feasible assembly sequences given such constraints (LSG), interactively edit assembly sequences (EDIT), check for proper constraint of assemblies (MLA), and determine the tolerances for an assembly (Tolerance Analysis). The roles of these tools in the AOD process has been examined. A new software tool, the Assembly Designer, has been developed that incorporates the theory to promote this AOD approach, providing a designer an integrated environment in which to develop assembly designs with seamless interfaces with the other existing software tools. The development of additional software tools such as DFCPR, a tool that automatically generates the precedence constraints on assembly sequences that result from the selected datum flow chain model of an assembly, and the modification of existing tools such as SPAS and MLA (Constraint Analysis) were also necessary to facilitate the proper integration of the software and theory so as to guide a designer through the AOD process.

The design process was then applied to an actual assembly from industry making full use of the software tools and methods developed. This case study demonstrated the benefits of using the AOD approach and the abilities of the software tools.

Thesis Supervisor: Dr. Daniel E. Whitney
Senior Research Scientist
Center for Technology, Policy, and Industrial Development
Lecturer, Department of Mechanical Engineering
Acknowledgments

First, I would like to thank my advisor, Dr. Daniel E. Whitney, for giving me the opportunity to conduct this research. I appreciate all the guidance he has offered me over the years, pertaining to research and other matters as well. I would also like to acknowledge Dr. Thomas L. De Fazio, Dr. Krish Mantripragada, and Jeffrey D. Adams for all their valuable assistance to me in this research.

I would also like to show my appreciation to all the friends I have made during my time at MIT. They may never know the impact they have made on my life, and I can only hope that I might have similarly affected them even if only in some small way. From them I have learned many things that can only be learned outside of the classroom but no less valuable in life. I would especially like to thank Sun for giving me all her support over the years.

Finally, I would like to thank my family for always being there for me. Their constant support and faith in me have made my studies possible.

This material is based upon work supported by the National Science Foundation under Grant No. DMI 9610163. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the National Science Foundation.
# Contents

1. **INTRODUCTION** ................................................................. 13
   1.1 Motivation ........................................................................... 13
   1.2 Goal of Research ............................................................... 14
   1.3 Thesis Overview ................................................................. 14

2. **PRIOR WORK** ..................................................................... 17
   2.1 Assembly Sequence Analysis ................................................ 17
      2.1.1 Precedence Relation Generation ....................................... 17
      2.1.2 Assembly Sequence Representation .................................... 18
      2.1.3 Assembly Sequence Evaluation ......................................... 18
   2.2 Datum Flow Chain ................................................................ 19
      2.2.1 Concept ........................................................................ 19
      2.2.2 Definition and Properties ................................................ 20
      2.2.3 Mates and Contacts ........................................................ 20
   2.3 Constraint Analysis ............................................................. 21
      2.3.1 Feature Library ................................................................ 21
      2.3.2 Algorithm .................................................................... 22
   2.4 Summary ............................................................................ 22

3. **ASSEMBLY ORIENTED DESIGN** ........................................ 23
   3.1 AOD Approach ................................................................. 23
   3.2 Design Process ................................................................. 24
   3.3 Summary ........................................................................... 27

4. **SOFTWARE IMPLEMENTATION** ....................................... 29
   4.1 Software Layout ............................................................... 29
   4.2 Assembly Designer ............................................................ 31
      4.2.1 Datum Flow Chain Design ............................................... 32
      4.2.2 Feature Selection ........................................................... 35
      4.2.3 Precedence Relation Generation ....................................... 38
      4.2.4 Constraint Analysis ....................................................... 44
      4.2.5 Assembly Sequence Editing ............................................. 44
   4.3 Datum Flow Chain Precedence Relation ................................ 45
   4.4 Summary ........................................................................... 50
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5. CASE STUDIES</td>
<td>5.1 Throttle Body</td>
<td>51</td>
</tr>
<tr>
<td>5.2 Summary</td>
<td>63</td>
<td></td>
</tr>
<tr>
<td>6. CONCLUSIONS</td>
<td>65</td>
<td></td>
</tr>
<tr>
<td>6.1 Conclusions</td>
<td>65</td>
<td></td>
</tr>
<tr>
<td>6.2 Future Work</td>
<td>65</td>
<td></td>
</tr>
<tr>
<td>BIBLIOGRAPHY</td>
<td>69</td>
<td></td>
</tr>
<tr>
<td>APPENDIX A</td>
<td>71</td>
<td></td>
</tr>
<tr>
<td>A.1 DFCPR</td>
<td>71</td>
<td></td>
</tr>
<tr>
<td>A.2 Assembly Designer</td>
<td>73</td>
<td></td>
</tr>
</tbody>
</table>
List of Figures

2.1 An example DFC ................................................................. 19
3.1 Top-down AOD approach .................................................. 24
3.2 Design process flowchart ................................................... 25
4.1 Software map ..................................................................... 30
4.2 Assembly Designer main window ......................................... 31
4.3 Add Part window ............................................................... 33
4.4 Edit Part window .............................................................. 34
4.5 Edit Liaison window ........................................................... 35
4.6 Add Feature window .......................................................... 36
4.7 Feature Coordinate Transform window ............................... 36
4.8 Feature Information window .............................................. 37
4.9 Edit Feature window ......................................................... 38
4.10 Select Feature window ..................................................... 38
4.11 SPAS window ................................................................. 39
4.12 Prismatic peg in prismatic hole assembly and feature definition 41
4.13 Alternative DFC’s for prismatic peg in hole assembly ........... 41
4.14 EDIT window ................................................................. 45
4.15 Liaison graph of an assembly in graphical and matrix form .... 46
4.16 DFC of an assembly in graphical and matrix form ................ 47
4.17 Alternative DFC for the assembly ....................................... 49
4.18 Another alternative DFC .................................................. 50
5.1 Throttle body assembled ..................................................... 51
5.2 Throttle body disassembled ............................................... 52
5.3 Throttle body liaison diagram ............................................ 52
5.4 Global coordinate frame shown on bore ............................. 53
5.5 Throttle body datum flow chain ......................................... 54
5.6 Geometrically feasible assembly sequence graph ................ 57
5.7 Revised throttle body DFC ................................................ 59
5.8 Geometrically feasible assembly sequence graph for revised DFC 60
5.9 Assembly sequence graph for revised DFC .......................... 61
5.10 Design process flow of throttle body assembly .................... 63
List of Tables

2.1 Feature types ................................................................. 21
4.1 Software tools and their functions ........................................... 30
4.2 Filename extensions .......................................................... 32
4.3 Disassembly axis vectors ...................................................... 40
5.1 Throttle body features ......................................................... 55
5.2 Revised throttle body features ............................................... 59
CHAPTER 1

Introduction

1.1 Motivation

It has been shown that it is necessary to consider manufacturing and assembly issues during the design phase of product development for numerous reasons. Doing so will allow for a better designed product in terms of ease and cost of manufacture and assembly in addition to a more robust product that best satisfies its intended functionality. Traditionally, product development began with the design process where a design engineer or design team would be responsible for producing a detailed design from which a given product would be manufactured. This design process consists primarily of product specification, preliminary design, and detail design. After the design has been finalized, product development enters the manufacturing phase where it is up to the manufacturing engineer or team to determine how best to manufacture and assemble the given design. However, since the design has already been fixed, much of the freedom in how the product can be assembled has been removed, often leading to difficult and costly assembly. It is the goal of assembly oriented design (AOD) methods to aid the designer in taking assembly issues into consideration during the design process to produce a design that can be better manufactured. This is done by taking advantage of the freedoms that exist during the design phase to optimize the design for assembly. The AOD methodology also allows for the analysis of existing designs in addition to being able to indicate possible areas of improvements in a design.
Past research, as detailed in Chapter 2, has resulted in theory, methods, and computational tools that support the top-down structure of the AOD methodology. However, as the research efforts were conducted by a number of individuals over a considerable period of time, there is a noted lack of integration in the methods and computational tools that have resulted.

1.2 Goal of Research

The ultimate goal of this research is to develop an integrated suite of computational tools for assembly oriented design. The first step in satisfying the goals of this research is to then study the design process itself and the steps involved in the design process. After having outlined the design process, it is then necessary to study the flow of design information in the AOD approach. Thus, the first goal of this research is to develop a model for the flow and analysis of design data that support the AOD framework. After having accomplished this, it is then possible to examine currently existing software tools and determine their roles in the design process making any necessary modifications in addition to developing new software tools that realize the theories and methods in the top-down AOD approach. The final step is to provide the designer an integrated user interface to the available software tools that aid in the AOD process. This involves the development of a front end design environment that interfaces with existing and newly developed software modules, as many of the existing software tools were developed independently and lack a coherent user interface.

1.3 Thesis Overview

Chapter 2 presents a brief review of previous research, including methods and computational tools, that provide the foundation for the integration and development of the computational tools for assembly oriented design in this research. Chapter 3 presents the
AOD process and the flow of information within. Chapter 4 discusses the actual software implementation. Chapter 5 gives case studies of actual assemblies and the application of the software tools. Finally, Chapter 6 presents overall conclusions and suggested directions for future work.
CHAPTER 2

Prior Work

This chapter gives a brief overview of previous work which is the foundation upon which this research is based. Previous research has been categorized into three subject areas: Assembly Sequence Analysis (ASA), Datum Flow Chain (DFC), and Constraint Analysis.

2.1 Assembly Sequence Analysis

De Fazio and Whitney [1] describes Assembly Sequence Analysis (ASA) as the methodology in which all mechanically feasible sequences are first generated, then they are edited based on given criteria, and finally compared on an economic basis.

2.1.1 Precedence Relation Generation

Bourjault [2] originally used a graph of contacts or liaisons between parts named a "liaison diagram" to model an assembly where a node represents a part and an arc between nodes represent a connection between two parts. He then developed an algorithm that was capable of generating all possible assembly sequences based on a series of yes-no questions based on part mates. After receiving user-input answers to these questions, the computer would then generate a series of constraints referred to as precedence relations. The form of precedence relations used in this thesis is as follows:
\[ L_i \& \ldots \& L_j \geq L_m \& \ldots \& L_n \]

which is read as all liaisons in the set of liaisons \( \{ L_i, \ldots, L_j \} \) must be completed previous to or simultaneously with the completion of all liaisons in the set of liaisons \( \{ L_m, \ldots, L_n \} \).

Subsequently, De Fazio and Whitney extended this method making it more efficient by adding new rules to the automatic reasoning that resulted in less questions to be answered by the user. Independent of this work, Homem De Mello [3] approached the task of precedence constraint generation using a different set of questions. Whipple [4] developed another method known as the "onion-skin" method for generating all valid assembly sequences on a process of questions that likens disassembly to the peeling of an onion. Baldwin [5] subsequently took these methods and incorporated them into a software tool called SPAS that generates the precedence relations by asking the necessary questions to the user.

2.1.2 Assembly Sequence Representation

After having determined the precedence relations required to generate all possible assembly sequences, it is necessary to represent the assembly sequences. In his assembly sequence generation software [6], Bourjault uses a parts tree to represent assembly sequences which is not compact but is easy to comprehend as it somewhat mimics a physical assembly line. Homem De Mello developed an And-Or Graph representation [7] to represent assembly sequences which is a compact representation but difficult to use to see how an assembly progresses. De Fazio and Whitney developed the Liaison Sequence Diagram based on a directed graph to offer a compact representation of assembly sequences that also offers more information on an assembly sequence's state by state progress.

2.1.3 Assembly Sequence Evaluation
Using the Liaison Sequence Diagram, Lui [8] developed a program, SED (Sequence Edit and Display) that would create the Liaison Sequence Diagram and allow for editing of the sequence diagram. Abell [9] expanded on this and Whipple's stability analysis creating a fully user interactive software tool, EDIT, for editing and evaluating assembly sequences.

2.2 Datum Flow Chain

Mantripragada and Whitney [10, 11] propose the concept of the Datum Flow Chain (DFC) for assembly modeling and the design of assemblies as a method of capturing the locational and dimensional constraint plan inherent in an assembly design. Figure 2.1 shows an example of a DFC.

![Figure 2.1 An example DFC](image)

2.2.1 Concept

Assembly requirements can be identified from top level customer requirements down to the manufacture and assembly of individual parts using a method called Key Characteristics (KC’s) [12, 13]. These KC’s capture the top level customer requirements as assembly level dimensions that relate a datum or feature on one part to another part in an
assembly. Thus, the DFC is used to model a given assembly with the intent of fulfilling the requirements of the KC’s of the assembly.

2.2.2 Definition and Properties

A DFC is an acyclic directed graph connective model of an assembly that defines the relationships between parts represented as nodes in the graph. A DFC identifies the part mates that convey dimensional control and identifies the hierarchy that determines which parts or fixtures define the location of other parts. The graph representation of a DFC is a subset of the liaison diagram where the arcs have direction indicating how a part defines the location of another part and a weight or label that indicates the number of degrees of freedom that are located. Note that in Figure 2.1, the numbers associated with the arcs do not represent the number of degrees of freedom located but simply enumerate the arcs themselves.

There are several properties that result from the definition of the DFC. There can be only a single root part that only has outgoing arcs. In addition, loops or cycles are not allowed as this would mean that a part would essentially be defining its own location. Finally, in a properly constructed DFC, each part should be constrained in all 6 degrees of freedom, unless there are degrees of freedom that are left unconstrained for functional reasons.

2.2.3 Mates and Contacts

In addition to the “mates,” represented by directed arcs, which transfer locational and dimensional constraint, there may be “contacts” between parts, represented by dashed lines, that transfer partial constraint or provide reinforcement without any locational constraint. This distinction implies that contacts between parts can not be established until the parts involved have been fully constrained by their mates.
2.3 Constraint Analysis

Often designers make constraint mistakes, where parts may be over-constrained or even under-constrained. Adams and Whitney [14, 15] have defined an algorithmic constraint analysis procedure making use of screw theory [16, 17] to examine combinations of elementary features in order to determine proper constraint of parts.

<table>
<thead>
<tr>
<th>Feature Type</th>
<th>Degrees of Freedom Constrained</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prismatic Peg / Prismatic Hole</td>
<td>6 X Y Z Tx Ty Tz</td>
</tr>
<tr>
<td>Plate Pin in Through Hole</td>
<td>5 X Y Z Tx Ty</td>
</tr>
<tr>
<td>Prismatic Peg / Prismatic Slot</td>
<td>5 X Z Tx Ty Tz</td>
</tr>
<tr>
<td>Plate Pin in Slotted Hole</td>
<td>4 X Z Tx Ty</td>
</tr>
<tr>
<td>Round Peg / Prismatic Slot</td>
<td>4 X Z Tx Ty</td>
</tr>
<tr>
<td>Round Peg / Through or Blind Hole</td>
<td>4 X Y Tx Ty</td>
</tr>
<tr>
<td>Threaded Joint</td>
<td>4 X Y Tx Ty</td>
</tr>
<tr>
<td>Elliptical Ball and Socket</td>
<td>4 X Y Z Tx</td>
</tr>
<tr>
<td>Plate-Plate Lap Joint</td>
<td>3 Z Tx Ty</td>
</tr>
<tr>
<td>Spherical Joint</td>
<td>3 X Y Z</td>
</tr>
<tr>
<td>Plate Pin in Oversize Hole</td>
<td>3 Z Tx Ty</td>
</tr>
<tr>
<td>Elliptical Ball in Cylindrical Trough</td>
<td>3 X Z Ty</td>
</tr>
<tr>
<td>Thin Rib / Plane Surface</td>
<td>2 Z Ty</td>
</tr>
<tr>
<td>Ellipsoid on Plane Surface</td>
<td>2 Z Tx</td>
</tr>
<tr>
<td>Spherical Ball in Cylindrical Trough</td>
<td>2 X Z</td>
</tr>
<tr>
<td>Peg in Slotted Hole</td>
<td>2 X Ty</td>
</tr>
<tr>
<td>Spheroid on Plane Surface</td>
<td>1 Z</td>
</tr>
</tbody>
</table>

Table 2.1 Feature types

2.3.1 Feature Library

Adams defines 17 types of assembly features as listed in Table 2.1 along with the degrees of freedom constrained in their nominal orientation. These 17 feature types are not meant to be a complete representation of all possible assembly features, but span the set of possible combinations of degree of freedom constraint of rigid body objects. These assembly features are modeled as kinematic joints allowing the use of a twistmatrix representation that describes the relative freedom in motion that a feature allows.
2.3.2 Algorithm

The constraint analysis algorithm determines whether a set of features acting on a part fully constrain that part. A wrenchmatrix reciprocal to a twistmatrix describes the set of forces and torques that can be transmitted by the joint described by the twistmatrix. First, the twistmatrices for each of the features acting on a part are concatenated into a single twistmatrix. Then, the reciprocal wrenchmatrix of the resulting single twistmatrix is calculated. This wrenchmatrix represents the logical intersection of the wrenchmatrices of the individual features. Thus, if this wrenchmatrix is not empty, each row of the matrix indicates that the features on the part are trying to constrain that same degree of freedom of that part, the definition of overconstraint. Adams has implemented this in the software tool, MLA.

2.4 Summary

In this chapter previously developed theory and software tools and methods that form the basis for the AOD process described in the next chapter have been presented. In addition, the modification and interfacing of these existing software tools and methods with newly developed software tools that will provide a designer with an integrated suite of software tools that aid in the AOD process is covered in Chapter 4.
CHAPTER 3
Assembly Oriented Design

In this chapter, the assembly oriented design (AOD) approach is discussed. The general design process is described, in addition to the flow of information during the design process.

3.1 AOD Approach

The AOD approach is a top-down design approach to modeling and analyzing assemblies and their assembly processes. The key goal of the AOD approach, as made obvious by its title, is to keep assembly issues in the foreground during the design process. As a result, it is desired that certain aspects of assembly analysis can be performed at an early stage when detailed part geometry may not be available. Thus, in this manner it is possible to examine how design changes affect the overall assembly. The end result should be an assembly design that satisfies both given design and assembly requirements. Figure 3.1 gives a broad overview of the AOD approach. Initially, the design process begins with concept design. A product architecture is defined from which Key Characteristics (KC’s) can be extracted. These are then modeled into a datum flow chain (DFC) for the assembly design. Next, features can be attributed to the DFC that satisfy the dimensional control plan laid out by the DFC. Using this featureized DFC, assembly analyses such as generation of tolerances and assembly sequences can then be carried out during this design stage. If those are found acceptable, work on detailed part design can then proceed.
3.2 Design Process

It is then necessary to outline a design process that follows the desired AOD approach. Figure 3.2 shows a flowchart for the design process as supported by the developed software tools and methods. This flowchart gives the nominal order of steps in the design process from beginning to end. However, the design process does not always follow such a linear order. Often, the design process is an iterative process where as more information becomes available, more details can be added to the design. This is critical to the idea of concurrent engineering where it is necessary to see how changes in different areas of the design affect the overall assembly. In addition, the design of an assembly may
not always start from scratch as much of design that occurs in industry supports the idea of reuse where it is desirable that existing parts of previous designs are to be incorporated with some amount of redesign. Therefore, the starting point of the design process is not always predetermined as all the necessary information may not initially be available. Furthermore, it is often necessary to backtrack in the design process as new details of the design require changes in earlier parts of the design in order to support such changes. The loops in the flowchart are an attempt to capture these complexities of the design process. The incoming arrows from the left in Figure 3.2 show what information needs to be provided by the designer at each stage of the design process. All other information created during the design session by the designer and generated by the software tools flows down along the design process.

![Design process flowchart](image)

Figure 3.2 Design process flowchart
Initially, given some notion of the parts, e.g. parts sketches, and some notion of the connections between parts, a liaison diagram can be constructed. Then, using the key characteristics (KC's) of the assembly that need to be satisfied, a datum flow chain (DFC) can be determined. Next, specific features can be assigned to liaisons that satisfy the determined datum flow. Once having done this, the constraints of the assembly can be automatically generated. This will give information about whether certain parts are overconstrained, underconstrained, or sufficiently constrained. If these constraints are not satisfactory, it may be necessary to modify the design of the DFC or of the specific features. To generate the geometric precedence relations, more detailed information of the geometry of the parts is required. It is still not necessary to specify the exact geometries of the parts, but rather just the relations between parts and how they interfere with each other during assembly. In addition, precedence relations derived from the DFC are automatically generated. Using these precedence relations, the family of feasible assembly sequences is generated and can then be edited according to more specific criteria. If the resulting sequences are not satisfactory, it is necessary to backtrack in the design process. One may go back as far in the design process as one desires to affect changes in the possible assembly sequences. This is due to the fact that the precedence relations that limit the feasible assembly sequences come from both the geometries of the parts, of which features play a limiting role in determining escape directions and the liaisons between parts, and the datum flow chain, which is coupled with the design of the features. This tight coupling of the parts, the features, and the DFC is what makes it difficult to specify a linear sequence for the design process as significant changes in one may require the others to be modified to support such a change. After having determined that the remaining assembly sequences are satisfactory, the tolerances can be determined. This requires the building of the tolerance chain for the specific design which in large part is specified by the DFC. If the resulting tolerances are not satisfactory, it will be necessary to return to the design of the DFC. Once the tolerances have been determined, the remaining assembly sequences are
examined to find an acceptable final assembly sequence. Thus, in the nominal flow of the
design process, the designer starts with a rough notion of the parts and connections
between parts in addition to the KC’s of the assembly. During the design process, the
details of the assembly become further specified, and at the end, the assembly sequence,
the tolerances, and the features have all been specified. In the case that the designer wishes
to redesign or perform an analysis on a existing design, it can be seen that the design
process just begins with much of the design already specified and the existing process can
still be applied.

3.3 Summary

In this chapter, the idea of the assembly oriented design approach has been
presented. In addition, a design process, as formalized in flowchart format, that supports
the AOD approach with the developed and existing software tools has been developed.
This required the determination of what information must be presented by the user at each
step in the design process in addition to what information can flow down from one step to
the next. The next chapter will briefly present the previously existing software tools and
describe in detail the newly developed software tools that support this AOD design
approach.
CHAPTER 4
Software Implementation

This chapter presents the software tools developed that aid a designer through the AOD process. In addition, the integration of the newly developed software tools with existing software tools is discussed in detail. The Assembly Designer (AD) is the main software tool that provides the user a central front end to the AOD methodology in addition to the necessary interfaces with existing software tools and methods. The Datum Flow Chain Precedence Relation (DFCPR) module is a faceless background module that was developed to provide for the algorithmic generation of precedence relations from a datum flow chain.

4.1 Software Layout

In Chapter 3, the flowchart of the design process in an AOD framework was presented. Figure 4.1 shows the layout and flow of the software tools that support the design process as given in Figure 3.2. In the software map, boxes represent software modules. Arrows between boxes represent the information that is passed between two software modules and the labels denote the exact data that is passed. Input arrows with italicized labels represent input from the user. The greyed out portions represent areas of future research that have not yet been implemented. Table 4.1 lists the major software tools and gives a brief description of each.
Figure 4.1 Software map

<table>
<thead>
<tr>
<th>Software Module</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assembly Designer</td>
<td>front end user interface for designing the DFC along with selecting the features</td>
</tr>
<tr>
<td>Constraint Checker (MLA)</td>
<td>checks for constraints given the DFC and the features using constraint analysis</td>
</tr>
<tr>
<td>SPAS</td>
<td>derives the geometric precedence relations through a series of yes/no questions</td>
</tr>
<tr>
<td>DFCPR</td>
<td>derives the precedence relations that result from the DFC</td>
</tr>
<tr>
<td>PRED</td>
<td>translates the precedence relations into C code</td>
</tr>
<tr>
<td>LSG</td>
<td>determines all feasible assembly sequences given the precedence relations</td>
</tr>
<tr>
<td>EDIT</td>
<td>allows for interactive editing of sequences</td>
</tr>
<tr>
<td>Tolerance Analysis</td>
<td>determines tolerances for a specific assembly sequence</td>
</tr>
</tbody>
</table>

Table 4.1 Software tools and their functions
4.2 Assembly Designer

The Assembly Designer (AD) software tool presents a designer with an environment in which to design an assembly, including all the necessary interfaces to other software modules in order to provide the designer a seamless, integrated design session. The software was written in C++ using X11 and Motif libraries to handle the graphics and is currently running on a UNIX workstation. Figure 4.2 shows the main window of AD. The main window consists of a menubar, a toolbox, the Assembly Window, the Part Window for displaying parts, and the Text Window for general information.

![Assembly Designer main window](image)

Figure 4.2 Assembly Designer main window

Table 4.2 lists the file extensions that AD uses for importing and exporting data and a brief explanation of each file extension.
<table>
<thead>
<tr>
<th>File Extension</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>.par</td>
<td>names of parts along with part id numbers and graphical positions in the assembly window</td>
</tr>
<tr>
<td>.lia</td>
<td>liaisons along with connected parts' id numbers and type of connection (mate with dof's, contact, unspecified)</td>
</tr>
<tr>
<td>.ptr</td>
<td>part coordinate transforms</td>
</tr>
<tr>
<td>.fea</td>
<td>feature type, attributes, and coordinate transforms</td>
</tr>
<tr>
<td>.ldm</td>
<td>liaison diagram incidence matrix</td>
</tr>
<tr>
<td>.dfc</td>
<td>datum flow chain incidence matrix</td>
</tr>
<tr>
<td>.dof</td>
<td>degrees of freedom (for dfc) incidence matrix</td>
</tr>
<tr>
<td>.adj</td>
<td>liaison diagram adjacency matrix</td>
</tr>
<tr>
<td>.nam</td>
<td>part names</td>
</tr>
<tr>
<td>.ld</td>
<td>liaison diagram (liaisons and names of connected parts)</td>
</tr>
<tr>
<td>.rc</td>
<td>escape directions for parts of each liaison</td>
</tr>
<tr>
<td>.pr</td>
<td>geometric precedence relations</td>
</tr>
<tr>
<td>.prd</td>
<td>datum flow chain precedence relations</td>
</tr>
<tr>
<td>.pra</td>
<td>all precedence relations</td>
</tr>
<tr>
<td>.in</td>
<td>input to MLA (includes part coordinate transforms, feature types, attributes, and coordinate transforms)</td>
</tr>
<tr>
<td>.zap</td>
<td>feasible assembly sequences</td>
</tr>
<tr>
<td>.expn</td>
<td>input to EDIT (includes part names and liaison diagram)</td>
</tr>
</tbody>
</table>

Table 4.2 Filename extensions

### 4.2.1 Datum Flow Chain Design

The design of a datum flow chain consists of three functional elements: parts, liaisons (which are specified as mates or contacts, or left unspecified in the intermediate stages of the design process), and features. The toolbar on the left of the Assembly Window, as can be seen in Figure 4.2, allows for the design and editing of DFC’s with the following self-explanatory tools: Part, Liaison, Mate, Contact, Feature, Show, Edit, and Delete. The first step of the design process requires building a connective model of the assembly. Using the Part tool, the designer specifies nodes in the Assembly Window for each part. When a new part is created, the program asks the designer to enter a name for the part and the part coordinate transform from the global coordinate frame as shown in Figure 4.3. At this point, this information, especially the part coordinate transform, is
optional until later needed when checking for proper constraint of the assembly, which is discussed in section 4.2.4.

Now, the connections between parts can be modeled. This is done by using the Liaison tool. By clicking the pointer on a part and dragging to another part, the designer can specify liaisons between parts. Note that at this point, very little knowledge of the assembly needs to be known, i.e. what parts exist and what connections exist between parts. With only this initial information, it is possible to determine the geometric precedence relations as covered in section 4.2.3 and generate all geometrically feasible assembly sequences as shown in section 4.2.5.

The next step in building a DFC for an assembly is to decide which liaisons act as mates, and which act as contacts. Recall that mates provide locational constraint whereas contacts merely provide support. The Mate and Contact tools are used to specify each of the liaisons in the assembly. This can be done by selecting an existing liaison with either tool or using the tool to draw a new mate or contact between two parts as with the Liaison tool. A contact is represented graphically in the Assembly Window as a dashed line and a mate as an arrow. When specifying a liaison as a contact, no further information is necessary. When creating a mate, the designer needs to specify how many degrees of
freedom are constrained in addition to selecting the proper direction in which that constraint is provided. The number of degrees of freedom constrained by a mate is shown graphically in the Assembly Window by a number next to the head of the arrow of the mate as can be seen in Figure 4.2. After completing the DFC, the precedence relations that are imposed by the DFC can be generated as shown in section 4.2.3 and covered in detail in section 4.3. Now, the family of feasible assembly sequences that satisfy the constraints imposed by the DFC can be examined (see section 4.2.5).

![Edit Part window]

Figure 4.4 Edit Part window

The DFC can also be edited and examined using the Show, Edit, and Delete tools. Using the Show tool, the designer can select a part or a liaison and have the part or parts connected by a liaison displayed in the Part Window if part drawings are available and textual information about the part or liaison displayed in the Text Window. The Edit tool allows the designer to change a part's name or modify the part coordinate transform of a part if a part is selected, as in Figure 4.4. If a liaison is selected, the designer can specify whether the liaison is a mate, contact, or unspecified, and if the liaison is a mate, modify
the number of degrees of freedom constrained and the direction of the mate, as shown in Figure 4.5.

![Edit Liaison window](image)

Figure 4.5 Edit Liaison window

Finally, the Delete tool is used to remove unwanted parts or liaisons. In addition, if a part is deleted, all liaisons involving that part are also deleted.

### 4.2.2 Feature Selection

Once a DFC has been created to model an assembly, it is then necessary to specify the features on the parts that achieve the liaisons. This is done using the Feature tool. Selecting a liaison using the Feature tool presents the designer with a feature selection window, as shown in Figure 4.6. The window presents the designer with icons for each of the 17 features listed in Table 4.1. When a feature is selected, a more detailed view of the feature is displayed along with the textual description from the table.
After the feature type is selected, the designer is prompted to enter the part to feature coordinate transform and also the specific attributes of the selected feature as shown in Figures 4.7 and 4.8.
After a feature has been fully specified, it is displayed graphically on the liaison in the Assembly Window as a circle containing the letter 'f' and a number. The number represents the number of features associated with that particular liaison. The Show, Edit, and Delete tools can also be used to edit and examine features. By selecting a feature with the Show tool, the parts connected by the feature are displayed in the Part Window, and the feature type, coordinate transform, and attributes are displayed in the Text Window. The Edit tool can be used to modify the coordinate transform and attributes of a feature as shown in Figure 4.9. If more than one feature exists, the designer is prompted to select which feature to modify among those present as seen in Figure 4.10. The Delete tool simply deletes a selected feature, and if more than one feature exists, again the designer is prompted for which feature to delete.
4.2.3 Precedence Relation Generation

Two types of precedence relations can be generated that impose constraints on the feasible assembly sequences for a given assembly. The first are those due to geometric interference and the second are those due to the theory of the datum flow chain. By using the menubar under the Modules menu, the designer can select the appropriate software tool.
for generating precedence relations, SPAS or DFCPR, and have the results displayed in the Text Window.

SPAS, originally written by Baldwin and given a new user interface by the author, is used to generate geometric precedence relations. This is done by asking the designer a series of questions in order to determine the geometric interferences that exist as discussed in section 2.1.1. Figure 4.11 shows an example session with SPAS.

![SPAS window](image)

Figure 4.11 SPAS window

In addition, SPAS can use disassembly axis vectors, or escape directions, to automatically determine certain cases of geometric interference. If the designer has already attributed features onto the DFC of an assembly, escape directions are automatically generated and supplied to SPAS to reduce the amount of user input required in determining the geometric precedence relations. A disassembly axis vector is the vector along which a part must be moved in order to remove it from another part. Thus, the disassembly axis vector is an attribute of a part and an associated liaison and is a function of the features
involved in that liaison. For each of the 17 features in the feature library, Adams has defined the feature coordinate frames such that either disassembly (and thus assembly) must occur along the z-axis, or it may be ambiguous as there might exist multiple possible paths for disassembly. Using Adams’ feature library, it is then possible to assign a disassembly axis to each feature; however, the direction of the disassembly axis vector for a part involved in that feature can be either positive or negative in the feature’s coordinate frame as there are two parts, one which has an escape direction in the +z direction and the other in the -z direction. This problem can be solved by utilizing the definition of the base part in each of Adams’ features. When the user enters the feature coordinate transform for a feature, the software asks for the coordinate transform relative to the coordinate frame of a part. This part is assumed to be the base part in the feature definition so that the disassembly axis vector for the base part associated with the feature can then be taken to be in the positive direction by the software when calculating the escape directions. Table 4.3 lists all 17 features and the disassembly axis vectors for the base parts of each feature. Note that a vector of (0, 0, 0) denotes that the disassembly axis vector is ambiguous.

<table>
<thead>
<tr>
<th>Feature Type</th>
<th>Disassembly Axis Vector</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prismatic Peg / Prismatic Hole</td>
<td>(0, 0, 1)</td>
</tr>
<tr>
<td>Plate Pin in Through Hole</td>
<td>(0, 0, 1)</td>
</tr>
<tr>
<td>Prismatic Peg / Prismatic Slot</td>
<td>(0, 0, 0)</td>
</tr>
<tr>
<td>Plate Pin in Slotted Hole</td>
<td>(0, 0, 1)</td>
</tr>
<tr>
<td>Round Peg / Prismatic Slot</td>
<td>(0, 0, 0)</td>
</tr>
<tr>
<td>Round Peg / Through or Blind Hole</td>
<td>(0, 0, 1)</td>
</tr>
<tr>
<td>Threaded Joint</td>
<td>(0, 0, 1)</td>
</tr>
<tr>
<td>Elliptical Ball and Socket</td>
<td>(0, 0, 1)</td>
</tr>
<tr>
<td>Plate-Plate Lap Joint</td>
<td>(0, 0, 0)</td>
</tr>
<tr>
<td>Spherical Joint</td>
<td>(0, 0, 1)</td>
</tr>
<tr>
<td>Plate Pin in Oversize Hole</td>
<td>(0, 0, 1)</td>
</tr>
<tr>
<td>Elliptical Ball in Cylindrical Trough</td>
<td>(0, 0, 0)</td>
</tr>
<tr>
<td>Thin Rib / Plane Surface</td>
<td>(0, 0, 0)</td>
</tr>
<tr>
<td>Ellipsoid on Plane Surface</td>
<td>(0, 0, 0)</td>
</tr>
<tr>
<td>Spherical Ball in Cylindrical Trough</td>
<td>(0, 0, 0)</td>
</tr>
<tr>
<td>Peg in Slotted Hole</td>
<td>(0, 0, 1)</td>
</tr>
<tr>
<td>Spheroid on Plane Surface</td>
<td>(0, 0, 0)</td>
</tr>
</tbody>
</table>

Table 4.3 Disassembly axis vectors
Thus, the disassembly axis vectors listed in Table 4.3 are for the base part in Adams’ feature definition. For example, consider the simple two-part assembly consisting of a prismatic peg (part A) in a prismatic hole (part B) and their respective part coordinate frames shown in Figure 4.12(a). The corresponding feature definition and the feature coordinate frame is shown in Figure 4.12(b). If given the DFC shown in Figure 4.13(a), part B is the source part in the DFC and the software will ask the user for the feature coordinate transform relative to part B’s coordinate frame. As the prismatic hole is the base part in the feature definition, the software uses Table 4.3 correctly to determine that part B’s disassembly axis vector is (0, 0, 1), i.e. +z direction, in the feature’s coordinate frame.
However, consider if the DFC is as shown in Figure 4.13(b). Here, the source part in the DFC is part A. However, according to the geometry of parts A and B, one might be inclined to associate part A, the peg, with the peg in Adams' feature definition and hence give the feature to part A coordinate transform as a 180 degree rotation around the y-axis. However, this would be incorrect as it would map part A's escape direction into the -z direction in its part coordinate frame. Since part A is the source part, it should be associated with the base part in the feature definition and the feature to part A coordinate transform should be the identity transform, such that the software can correctly identify its disassembly axis vector in the +z direction in the feature coordinate frame and hence +z direction in part A's coordinate frame. This is due to the fact that the software does not distinguish which part is the peg or hole, only which part is the base part in determining the disassembly axis vector.

In summary, only a disassembly axis can be associated to a feature definition, not the actual direction vector. However, the actual vector can be associated with the base part in the feature definition. In addition, the software has no knowledge of the actual feature geometry of the parts, e.g. which part has the peg and which has the hole. The software must then assume that the source part in the DFC is the base part in the feature definition. The software is then capable of accurately determining the disassembly axis vectors of each of the parts using the information from Table 4.3 provided that the user enters the feature to part transform correctly such that the source part in the DFC is associated with the base part in the feature definition regardless of the actual feature geometry on the parts as was seen with the example DFC in Figure 4.13(b).

It has been acknowledged that this may not be the most intuitive manner for the user to assign features to liaisons, i.e. to define the feature to part coordinate transform such that the base part from the feature definition is associated with the source part in the DFC disregarding the actual feature geometry on the parts. This method has been chosen so as to limit the amount of explicit geometric information the user has to provide and allow the
software to still be able to calculate the disassembly axis vectors. For example, it may be known that a certain feature is to be used to join two parts, e.g. peg and slot, but the actual feature geometry on the parts may not yet have been decided upon, e.g. which part has a peg and which has a slot. Yet, the disassembly axis vectors should still be able to be determined. The alternative approach then is to take as input from the user the explicit feature geometry and allow the user to input the feature to part coordinate transform such that the geometry from the feature definition matches the geometry of the features on the parts. For example, when assigning the peg and slot feature to a liaison between two parts, the user would also input which part has the peg. In this method, the software would not have to assume that the source part in the DFC is the base part in the feature definition, for it would now explicitly know according to the user input. The software would then be capable of correctly determining the disassembly axis vectors of each part given the feature to part coordinate transform that matches up the geometry of the feature definition to the geometry of the features on the parts.

Once the correct disassembly axis vectors are known for each feature, the disassembly axis vector of a part involved in a liaison with multiple features can be determined. That is, for each liaison, the disassembly axis vector for each part can then be automatically generated. This is done by taking the disassembly axis vector for each feature attributed to the liaison and using the feature coordinate transform and part coordinate transform to determine the vector in the global coordinate frame. Next, the vectors for all features are compared. If they are all identical, then the disassembly axis vector for that part and that liaison is simply that vector. Otherwise, the disassembly axis vector is taken to be uncertain.

DFCPR is used to automatically generate the precedence relations that result from constraints imposed by the datum flow chain upon the feasible assembly sequences. DFCPR is covered in detail in section 4.3.
4.2.4 Constraint Analysis

Once the DFC for an assembly has been created and features have been added, it is possible to check for proper constraint of the assembly. This is done by using the menubar to access the Constraint Analysis module. Constraint analysis is one of the functions of Adams’ MLA software tool. The Assembly Designer bypasses MLA’s text based user interface for entry of part coordinate transforms and feature selection (including feature type, coordinate transform, and attributes). Thus, AD is in effect providing a graphical user interface for feature selection and using MLA solely as a faceless background module to check for proper constraint. Results are returned in the Text Window of AD.

4.2.5 Assembly Sequence Editing

The final module in the menubar calls Abell’s EDIT software tool. There are two options for running this module. The designer may wish to generate the complete tree of geometrically feasible assembly sequences using only the geometric precedence relations from SPAS or just the family of assembly sequences that result after imposing the constraints due to the datum flow chain. Figure 4.14 shows an example session with EDIT, with a new graphical user interface written by Mantripragada.
4.3 Datum Flow Chain Precedence Relation

An assembly can be graphically represented as a liaison graph where nodes represent parts or subassemblies and liaisons represent contacts or mates between parts. This liaison graph can also be stored in a matrix to facilitate computer manipulation where rows represent parts and columns represent liaisons. This specific matrix is referred to as an incidence matrix. Figure 4.15 depicts the relation between graphical and matrix representation. To create this matrix, rows are created for all parts in the assembly. Then, for each liaison in numerically sequential order, a column is created with a '1' in the rows of the two parts the liaison connects and a '0' otherwise. Thus, it follows that for each column there must be exactly two elements with a value of '1'. For example, liaison 7
connects parts S3 and AS. Therefore, in its column, there are '1's in the rows for S3 and AS and '0's elsewhere.

Figure 4.15 Liaison graph of an assembly in graphical and matrix form

The datum flow chain (DFC) is graphically represented using an ordered liaison graph. An ordered liaison graph is similar to a liaison graph except that liaisons now have a direction from one node to another. The DFC can also be represented by a matrix similar to one used for a liaison graph. In order to capture the direction of a liaison, a '-1' is used to indicate that a liaison points to a part and a '1' is used to indicate that a liaison originates from a part. Thus, for a DFC matrix, every column must have exactly one '-1' element and one '1' element. Figure 4.16 illustrates the DFC and accompanying matrix representation. Dashed lines represent liaisons that are contacts and thus not part of the datum flow chain.
Figure 4.16 DFC of an assembly in graphical and matrix form

A given DFC layout imposes assembly constraints in addition to those due to the geometric relations between parts. According to Mantripragada, these constraints can be summarized by the following two rules:

1. There can be no subassemblies with only contacts between parts.
2. There can be no subassemblies with incompletely constrained parts.

These rules are referred to as the contact rule and the constraint rule, respectively. These statements can then be translated into a computer algorithm operating on the matrix forms of the DFC and liaison graph to allow for automatic generation of precedence relations similar to those generated from geometric interferences between parts. Given the two rules listed above, there are two types of precedence relations generated.

To eliminate the possibility of subassemblies with only contacts between parts (contact rule), the liaison graph matrix and the DFC matrix are compared to determine which liaisons are contacts. Then, for each contact, a precedence relation stating that all
mates in the DFC pointing to the parts the contact connects must be completed before the contact can be completed is generated. For example, in Figure 4.16, liaison 3 joining parts PC and S3 is a contact. Incoming mates to parts PC and S3 include liaisons 2 and 4. Thus, liaisons 2 and 4 must be completed prior to or simultaneous with liaison 3 (2 & 4 ≥ 3). This type of precedence relation generation will ensure that subassemblies with only contacts between parts will not be allowed.

To ensure that subassemblies with incompletely constrained parts are not allowed (constraint rule), each row in the DFC matrix is examined one at a time. If a part (row) has more than one incoming mate (element with value '-1'), then all incoming mates must be simultaneously completed to ensure that the part be fully constrained when assembled. For example, looking at the first row of the DFC matrix in Figure 4.16, part PC has two incoming mates, liaisons 2 and 4. Thus, liaisons 2 and 4 must be completed simultaneously (2 ≥ 4 and 4 ≥ 2).

The following is the list of precedence relations generated for the datum flow chain given in Figure 4.16:

\[
\begin{align*}
2 & \land 4 & \land 6 & \geq 1 \\
2 & \land 4 & \geq 3 \\
2 & \land 4 & \land 9 & \geq 5 \\
2 & \geq 4 \\
4 & \geq 2 
\end{align*}
\]

Note that the first three precedence relations satisfy the contact rule discussed above while the last two relations satisfy the constraint rule.
Figure 4.17 depicts an alternative DFC layout for the assembly shown in Figure 4.15. Its accompanying precedence relations are as follows:

- \(3 \& 7 \geq 2\)
- \(3 \& 8 \geq 4\)
- \(1 \geq 6\)
- \(6 \geq 1\)
- \(5 \geq 9\)
- \(9 \geq 5\)

Figure 4.18 depicts yet another alternative DFC layout with the following precedence relations:

- \(1 \& 2 \geq 6\)
- \(2 \& 3 \geq 7\)
- \(3 \& 4 \geq 8\)
- \(4 \& 5 \geq 9\)

For this particular DFC, all precedence relations generated are due to the contact rule to ensure that there are no subassemblies with only contacts.
4.4 Summary

In this chapter, an overview of the software tools was presented in addition to a detailed account of the two newly developed software tools, AD and DFCPR. In the next chapter, these tools will be applied to the AOD process of real assemblies.
CHAPTER 5

Case Studies

This chapter examines real assemblies using the developed software tools in an AOD framework.

5.1 Throttle Body

In our original assembly model, fasteners have largely been ignored as they have been considered to be more a part of the assembly process than actual parts in the assembly itself. However, in examining the throttle body, shown assembled in Figure 5.1 and disassembled into component parts and subassemblies in Figure 5.2, it has been determined that taking fasteners into consideration as part of the assembly model can provide useful insight.

Figure 5.1 Throttle body assembled
First, it is necessary to build an assembly model of the throttle body. The assembly consists of four parts: bore, shaft, disk, and screws. The parts are entered into the Assembly Designer software along with the connections that exist between the parts to arrive at the liaison diagram shown in Figure 5.3. For each part, the coordinate transform...
from the part coordinate frame to the global coordinate frame is simply the identity transform, i.e. all part coordinate frames are coincident with the global coordinate frame with the origin located in the center of the bore on the axis of the shaft. Figure 5.4 shows the global coordinate frame with respect to the bore.

Figure 5.4 Global coordinate frame shown on bore

Once the liaison diagram has been created for the assembly, the next step is to build the datum flow chain model of the assembly. This involves determining which liaisons act as mates and which act as contacts and deciding upon the number of degrees of freedom that each mate constrains. In this assembly, the bore acts as the base part for the entire assembly. It in turn locates both the shaft and the disk. However, the shaft is not fully constrained by the bore. The shaft is free move along its axis when inserted into the bore. The shaft’s degree of freedom that allows it to rotate on its axis has been ignored in this model of the assembly and is thus considered fixed. Thus, the bore locates the shaft in 5 degrees of freedom, i.e. all those except translation along the x axis. The shaft fully locates the screws in all 6 degrees of freedom. Finally, the bore, shaft, and screws act together to fully constrain the disk in all 6 degrees of freedom. The bore locates the disk in the x and y direction. The shaft locates the disk in the z direction and constrains rotations around the x and y axes. Lastly, the screws fix the disk’s rotation around the z axis.
Figure 5.5 Throttle body datum flow chain

Figure 5.5 shows the datum flow chain (DFC) for the throttle body assembly. Note that this DFC model of the assembly consists entirely of mates as there are no contacts. Now it is necessary to select the features that realize the mates in the DFC. The "prismatic peg/prismatic slot" feature is used to model the mate between the bore and the shaft in order to model the shaft's single degree of freedom of translation along its axis. The mate between the bore and the disk is modeled by the "spherical ball in cylindrical trough" feature as the bore is basically a cylindrical trough and the disk has the same degrees of freedom as a spherical ball in a cylindrical trough. The disk is mounted in a groove in the shaft with some clearance and thus the "plate pin in oversized hole" feature is used to model the mate. The mate representing the screws in the shaft is accurately modeled by using two "plate pin in through hole" features, one for each of the two screws. Note that this choice of features does not explicitly fully constrain the screws in all 6 degrees of freedom as each feature only constrains 5 degrees of freedom, i.e. all but rotation around the z-axis; however, Constraint Analysis will show that rotation around the z-axis is properly constrained by the combination of the two features as it should be. The final mate between the screws and the disk is modeled using two "plate pin in oversize
hole” features, capturing the interaction between each of the individual screws and the disk. These feature choices are summarized in Table 5.1.

<table>
<thead>
<tr>
<th>Part 1</th>
<th>Part 2</th>
<th>Feature</th>
<th>Transform</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bore</td>
<td>Shaft</td>
<td>Prismatic Peg / Prismatic Slot</td>
<td>(0 0 0 0 90)</td>
</tr>
<tr>
<td>Bore</td>
<td>Disk</td>
<td>Spherical Ball in Cylindrical Trough</td>
<td>(0 0 90 0 0)</td>
</tr>
<tr>
<td>Shaft</td>
<td>Disk</td>
<td>Plate Pin in Oversize Hole</td>
<td>(0 0 0 0 0)</td>
</tr>
<tr>
<td>Shaft</td>
<td>Screws</td>
<td>Plate Pin in Through Hole</td>
<td>(-1.21 0 0 0 0)</td>
</tr>
<tr>
<td>Screws</td>
<td>Disk</td>
<td>Plate Pin in Oversize Hole</td>
<td>(1.21 0 0 0 0)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Plate Pin in Oversize Hole</td>
<td>(1.21 0 0 0 0)</td>
</tr>
</tbody>
</table>

Table 5.1 Throttle body features

After having completed the feature selection, it is then possible to use Constraint Analysis to check for proper constraint. Constraint Analysis reports that the bore and screws are fully constrained but that the shaft and disk are not fully constrained. Examining the DFC, it can be seen that the shaft has only 5 degrees of freedom constrained by the bore. In the actual assembly, the disk provides constraint in the final degree of freedom, translation along the shaft’s axis. However, in the DFC model, the shaft is already locating the disk in 3 degrees of freedom, and the definition of the DFC states that it must be an acyclic graph of the assembly, i.e. loops or cycles are not allowed; otherwise, a part may in effect be locating itself. However, in the case of the throttle body, the shaft is locating the disk in translation along the z axis and rotations around the x and y axes, and the disk is really locating the shaft in translation along the x axis forming a loop in the current concept of the DFC. Yet since the shaft is locating the disk in different degrees of freedom than those in which it is being located by the disk, the shaft is not locating itself in any degree of freedom. This should then be considered a valid locating scheme. This leads to the conclusion that a DFC may have to allow for cycles or loops in cases where the degrees of freedom constrained are independent. In other words, it may be necessary to
build a DFC model of an assembly for each individual degree of freedom to correctly model an assembly.

However, this may be difficult in cases where part coordinate frames are such that it is not possible to determine for all parts the degrees of freedom constrained using the same six independent degrees of freedom in global coordinates. For example, imagine a part whose coordinate frame is offset from a second part's coordinate frame by a 45 degree rotation around the z axis. Then, constraint in the x degree of freedom in the first part's coordinate frame represents a combination of constraint in the second part's coordinate frame in the x and y directions. It would then be highly difficult to draw a DFC for the assembly in just a single degree of freedom. Another solution may be to allow for loops or cycles in the DFC, but check using part coordinate transforms to ensure that there are no loops in a single degree of freedom in the global coordinate frame. For now, we know that the shaft in the assembly is properly constrained and leave the extension of the concept of the DFC as an area for future research.

The DFC model of the throttle body seeks to have the disk properly constrained in all 6 degrees of freedom. However, it turns out that the features used in the mate between the screws and disk do not constrain the proper degrees of freedom as intended by the DFC model. The screws are physically pins in oversize holes in the disk, justifying the feature selection. Because the holes in the disk are oversized, the screws are incapable of properly locating the disk in the rotational degree of freedom around the z axis. Despite this lack of proper constraint, let us continue to study this assembly model to see what results are obtained before presenting a more suitable model of the assembly.

Using SPAS, geometric constraints generate the following precedence relations:

\[
1 \& 2 \geq 3
\]
\[
1 \& 3 \geq 2
\]
\[
2 \& 3 \& 5 \geq 1 \& 4
\]

These simply state that the shaft must be in place with the bore before the disk can be added to either the shaft or the bore and also that the disk and shaft must be assembled together.
before the screws can be assembled to the shaft. The graph of geometrically feasible assembly sequences obtained from EDIT is shown in Figure 5.6.

Figure 5.6 Geometrically feasible assembly sequence graph

In addition, the following precedence relations due to constraints imposed by the DFC are generated using DFCPR:

\[
\begin{align*}
2 & \geq 3 \\
2 & \geq 5 \\
3 & \geq 2 \\
3 & \geq 5 \\
5 & \geq 2 \\
5 & \geq 3 \\
\end{align*}
\]

These precedence relations state that the disk can only be added to the final assembly after the bore, shaft, and screws have been assembled, or more simply, the disk must be the last part to be assembled in the assembly sequence. This condition results from applying the constraint rule which seeks to avoid underconstrained subassemblies.\(^1\) However, this leads to an interesting contradiction. Given the precedence relations generated from

\(^1\) The DFC assumes the disk will be fully constrained if it is installed last, even though we know that the features selected can not achieve this.
geometric constraints (namely the precedence that the disk and shaft be assembled together before the screws can be assembled) and those generated from the DFC (the precedence that the disk must be the last part to be assembled), it is impossible to find an assembly sequence that satisfies both these precedence relations.

This contradiction can be explained by the conclusion that there is a fault either in the theory or in the application. Upon closer inspection of the actual assembly and the actual assembly process required to physically assemble the throttle body, it has been found that the datum flow chain is not so simple as previously presented. Recall that in the Constraint Analysis of the previous DFC model, the disk turned out to be incompletely constrained. Rather, in manual assembly, the hand serves as a temporary fixture during the assembly process. Figure 5.7 shows a revised DFC that includes the temporary fixture and corresponding feature selections in Table 5.2. This DFC indicates that the temporary fixture serves to help locate the disk along with the bore and shaft instead of the screws as previously modeled. By freeing the screws of this responsibility, the liaison between the screws and the disk is now only a contact indicating that the connection between the screws and the disk merely provides support. The feature used to realize the mate between the fixture and the disk is the “thin rib/plane surface” from the feature library. Although this feature does not exactly model the geometry of the interaction between the fixture and the disk, it does capture the constraint that the fixture imposes upon the disk, namely the rotation around the z axis. Constraint Analysis now returns the proper results showing that the disk is fully constrained by its mates. In addition, with this revised DFC, there is no conflict between the precedence relations generated from geometric constraints and from the DFC that would leave the assembly without any possible assembly sequences.
Figure 5.7 Revised throttle body DFC

<table>
<thead>
<tr>
<th>Part 1</th>
<th>Part 2</th>
<th>Feature</th>
<th>Transform (X Y Z θx θy θz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bore</td>
<td>Shaft</td>
<td>Prismatic Peg / Prismatic Slot</td>
<td>0 0 0 0 0 90</td>
</tr>
<tr>
<td>Bore</td>
<td>Disk</td>
<td>Spherical Ball in Cylindrical Trough</td>
<td>0 0 0 90 0 0</td>
</tr>
<tr>
<td>Shaft</td>
<td>Disk</td>
<td>Plate Pin in Oversize Hole</td>
<td>0 0 0 0 0 0</td>
</tr>
<tr>
<td>Shaft</td>
<td>Screws</td>
<td>Plate Pin in Through Hole</td>
<td>-1.21 0 0 0 0 0</td>
</tr>
<tr>
<td>Screws</td>
<td>Disk</td>
<td>Plate Pin in Oversize Hole</td>
<td>1.21 0 0 0 0 0</td>
</tr>
<tr>
<td>Fixture</td>
<td>Bore</td>
<td>Prismatic Peg / Prismatic Hole</td>
<td>0 0 0 0 0 0</td>
</tr>
<tr>
<td>Fixture</td>
<td>Disk</td>
<td>Thin Rib / Plane Surface</td>
<td>0 0 0 90 0 0</td>
</tr>
</tbody>
</table>

Table 5.2 Revised throttle body features

The precedence relations that result from geometric constraints as generated by SPAS are as follows:

1 & 2 ≥ 3
1 & 3 ≥ 2
2 & 3 & 5 ≥ 1 & 4
2 & 3 & 6 ≥ 1 & 7

These are the same as for the assembly without the fixture with the addition of the last precedence relation which basically adds the constraint that the bore must be placed within
the fixture and the shaft into the bore before placing the disk in the fixture. Thus, the possible assembly sequences considering only geometric constraints remains the same as before also with the addition of starting with the fixture as the first step in an assembly sequence. The graph of geometrically feasible assembly sequences obtained from EDIT is shown in Figure 5.8.

![Geometrically feasible assembly sequence graph for revised DFC](image)

Figure 5.8 Geometrically feasible assembly sequence graph for revised DFC

In this case, the precedence relations due to the DFC generated by DFCPR are as follows:

\[
\begin{align*}
2 & \geq 3 \\
2 & \geq 7 \\
3 & \geq 2 \\
3 & \geq 7
\end{align*}
\]
With this DFC, not all assembly sequences are eliminated as they were for the assembly without the fixture. Figure 5.9 shows the graph of assembly sequences containing only two possible assembly sequences after all precedence relations have been applied.

Given that the assembly sequence start with the fixture, the following assembly sequence satisfies all precedence relations and would mostly likely serve as the final assembly sequence:

```
Fixture - Bore - Shaft - Disk - Screws
```
Thus, it has been shown that the theory holds up even with the inclusion of fasteners. In fact, the theory can be used to suggest possible errors in modeling an assembly as shown here in the case of the DFC of the throttle body. Without the inclusion of fasteners in the assembly model, possible errors may even be neglected. In modeling the throttle body, the fact that during manual assembly the hand serves as a temporary fixture was overlooked at first. However, the notification that the disk was not fully constrained and later the fact that no possible assembly sequences resulted provided quick notification that our model of the assembly might not be entirely correct. Figure 5.10 shows the process flow in our analysis of the throttle body design. An initial DFC model was constructed with the intended constraint scheme. Features were then selected to realize the desired constraints. Constraint Analysis showed that the disk was not fully constrained by the features selected. At this point, it would have then been advisable to return to the design of the DFC or the selection of the features for a more accurate model of the assembly with proper constraint. Continuing on with the improperly constrained model of the assembly resulted in no possible assembly sequences as a result of conflicts between the DFC sequence constraints and the geometric sequence constraints. Thus, a new DFC model was constructed that included a fixture. Constraint Analysis showed that this assembly model was properly constrained and finally assembly sequences were generated. Note that this process flow confirms the design process laid out in Chapter 3 with all the possible iterations during the design process.
5.2 Summary

In this chapter, an example assembly has been examined in detail. Using the design process outlined in Chapter 3 and the software tools and methods presented in Chapter 4, an analysis of a throttle body design was conducted. The importance of proper constraint and the ability to detect for it was shown. In addition, the iterative nature of the design process was confirmed by the steps taken in accurately modeling the throttle body assembly.
CHAPTER 6
Conclusions

In this chapter the major conclusions of the research are summarized. In addition, possible areas of future research are presented.

6.1 Conclusions

Previously developed theory and software tools and methods exist for conducting assembly oriented design as presented in Chapter 2. In this research, an assembly oriented design process that makes use of these tools and methods has been formalized. In doing so, the flow of information necessary to support such a design approach has been outlined so as to determine the proper sequence of the AOD process. A new software tool, the Assembly Designer, has been developed that incorporates the theory to promote this AOD approach. This software tool provides the designer with the proper workspace in which to design a model of an assembly. In addition, AD offers the user an integrated front-end user interface to the other software tools as necessary during the steps in the AOD process. This involved the development of additional software tools such as DFCPR and the modification of existing tools such as SPAS and MLA (Constraint Analysis) in order to facilitate the proper integration of the software so as to guide the designer through the AOD process.

6.2 Future Work
Although the AOD process does not rely on having detailed part geometry throughout much of the design process, it would be of great benefit to link to existing CAD software. This would allow the designer to further design the parts in tandem with the modeling of the assembly. In addition, by having a CAD system available, information such as part coordinate transforms could swiftly be transferred from the CAD system to the Assembly Designer without the need for tedious manual entry by the user. This CAD system would also allow for more complicated disassembly axis vectors to be calculated further reducing the need for user interaction during the generation of the precedence relations that exist due to geometric interference between parts.

As mentioned in Chapter 4, further work on making the user interface more intuitive during assembly modeling may be conducted. The user has to be cautious when entering the feature coordinate transforms to ensure the correct calculation of disassembly axis vectors. However, perhaps by having the designer explicitly defining the geometry of feature on parts, the software will then have the necessary knowledge in order to automatically account for the correct direction of the direction of the parts’ disassembly axis vectors along the axis that is associated with a specific feature definition. In addition, have the software linked with a CAD system may also facilitate this process.

In Chapter 5, some possible shortcomings of the current concept of the DFC were introduced. This involved the need to be able to examine the flow of locational responsibility in individual degrees of freedom. This may be accomplished by allowing cycles in the DFC but having a rigorous analysis ensuring that no part is in effect locating itself in a particular degree of freedom. Note that this does not invalidate any of the theory of the DFC or the benefits that come from using the DFC as a model of an assembly. This enriching of the DFC would only make it possible to more accurately model assemblies with more complicated constraint schemes.

Future directions for research also include further integration of existing software tools and the need for new tools. Mantripragada developed software that applies optimal
control theory to calculate optimal variation propagation through tolerance analysis, making use of motion limit vectors calculated using Adams' Motion Limit Analysis software. Further work then may be done to integrate and take advantage of analysis tools that exist in Tolerance Analysis and the MLA package. This concept may also contribute to the possibility of developing new software tools for the automatic selection of features that offer optimal assembly.
Bibliography


[18] M. B. Wall, “Array.h” C++ array class written February 9, 1996, MIT. Email: mbwall@mit.edu.
APPENDIX A

Source Code

This appendix contains the source code for the two software tools developed in this research, Assembly Designer (AD) and DFCPR. DFCPR consists of a single C file, DFCPR.c. AD consists of four header files -- Mpart.h, Mliaison.h, Mfeature.h, and Mtransform.h -- and the main C++ file, main.C. In addition, AD uses Matthew Wall’s Array.h class [18].

A.1 DFCPR

DFCPR.c

/* DFC -> PR generator */
/* Stephen J. Rhee */
/* Revision History */
/* ---------------- */
/* 13 Jul 97 1.0 initial implementation */
/* 16 Jul 97 1.1 fixed bug where if DFC#col!=LIM#col, wrong # used in PRs */
/* 16 Jul 97 1.2 created a function to get liai # from the LIM using DFC */
/* 23 Jul 97 1.3 added an option for generating PRs for multiple SA's */
/* 04 Aug 97 1.4 changed PR generation and removed multi SA option */
/* 27 Sep 97 1.5 changed allowable command line calls */
/* 27 Jul 99 1.6 changed read matrix to use fgets instead of fgets due to */
/* some strange seg fault for no reason; in addition fgets */
/* returns the CRP strangely so had to use char typecasting */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#define MAXFILEEN 50
#define MAXLINEEN 200
#define MAXRC 15
#define MAXCOL 30

FILE *g;
char assem_file[MAXFILEEN];
char *file, *dfile;
char liai_file[MAXFILEEN];
char dfc_file[MAXFILEEN];
char llim_file[MAXFILEEN];
int liai_num[MAXCOL+1][MAXCOL+1];
int dfc_num[MAXCOL+1][MAXCOL+1];
int i, j, k, l, m, found, match;
void read_matrix(char *filen, int mat[MAXCOL+1][MAXCOL+1]);
void print_matrix(int mat[MAXCOL+1][MAXCOL+1]);
void print_liai(int liai_num[]);
char *fgets2(char *s, int n, FILE *f);
main (int argc, char *argv[]) {
 /*
 */
if (argc > 2) {
    printf("Usage: TCPR <assembly name>\n");
    exit(1);
} 
else if (argc == 2) {
    strcpy(liai_file, argv[1]);
    strcat(liai_file, "ldn");
    strcpy(dfc_file, argv[1]);
    strcat(dfc_file, "dfc");
    strcpy(prct_file, argv[1]);
    strcat(prct_file, "prd");
} 
else {
    printf("Enter assembly name:\n");
    gets(assembly);
    if (assembly[0] == '0')
        exit(1);
    strcpy(liai_file, assembly);
    strcat(liai_file, "ldn");
    strcpy(dfc_file, assembly);
    strcat(dfc_file, "dfc");
    strcpy(prct_file, assembly);
    strcat(prct_file, "prd")
} 
read_matrix(liai_file, liai_mat);
read_matrix(dfc_file, dfc_mat);
g = fopen(prct_file, "w");
for (i=1; i<=dfc_mat[0][1]; i++)
    for (j=1; j<=dfc_mat[1][0]; j++)
        if (dfc_mat[i][j] == -1)
            for (k=1; k<=dfc_mat[1][0]; k++)
                if (dfc_mat[i][k] == -1) {
                    fprintf(g, " %d \n", liai_mat[j][0]);
                    fprintf(g, " %d \n", liai_mat[i][0]);
                }
        if (liai_mat[i][0] == liai_mat[0][1])
            if (liai_mat[0][1] == liai_mat[0][0])
                match = 1;
        for (k=1; k<=liai_mat[0][1]; k++)
            if (liai_mat[k][0] == liai_mat[0][0])
                match = 0;
    break;
}
if (match == 1) {
    found = 1;
    break;
}
else {
    found = 0;
    if (match == 0) {
        if (liai_mat[0][1] == liai_mat[0][0])
            if (liai_mat[0][0] == liai_mat[0][0])
                match = 1;
        for (k=1; k<=liai_mat[0][1]; k++)
            if (liai_mat[k][0] == liai_mat[0][0])
                match = 0;
        break;
    }
    if (match == 1) {
        found = 1;
        break;
    }
}
if (found == 1)
    return i;
else
    return 0;
}

int liai_mat(int i, int j)
{
    int i, j;
    int found, match;
    found = 0;
    for (i=1; i<=liai_mat[0][1]; i++)
        for (j=1; j<=liai_mat[1][0]; j++)
            if (liai_mat[i][j] == liai_mat[i][j][DFC_liason])
                match = 1;
        if (match == 1) {
            found = 1;
            break;
        }
    if (found == 1)
        return i;
    else
return 0;
}

void print_dfcprs()
{
    FILE *f;
    int c;
    if ((f = fopen(prctfile, "r")) == NULL)
        printf("There are no precedence relations imposed by the DFC.\n");
    else {
        printf("The following precedence relations are imposed by the DFC:\n");
        while ((c = getc(f)) != EOF)
            putchar(c, stdout);
        fclose(f);
    }
}

void read_matrix(char *filen, int mat[MAXRN+1][MAXCL+1])
{
    FILE *f;
    char c;
    int i, j, k;
    if ((f = fopen(filen, "r")) == NULL) {
        printf("error: can't open %s\n", filen);
        exit(1);
    }
    i = 1;
    j = 0;
    while (((char)(c=fgetc(f))) != (char) (EOF)) {
        if (c == '"') {
            mat[i][j] = 0;
        } else if (c == '"') {
            mat[i][j] = 1;
        } else if (c == '-') {
            mat[i][j] = -1;
        } else if (c == '[') {
            if (j == 0) {
                j = 1;
                mat[0][k] = j;
            }
        } else if (c == ']') {
            if (j == 0) {
                j = 1;
                mat[i][0] = j;
            }
        }
    }
    void print_matrix(int mat[MAXRN+1][MAXCL+1])
    {
        int i, j;
        for (i=1, i=mat[0][1]; i++) {
            for (j=1; j=mat[1][0]; j++)
                printf("%d\t", mat[i][j]);
            printf("\n\n");
        }
    }

A.2 Assembly Designer

Mpart.h

#include <string.h>
#include "Asmasy.h"
#include "Mtransfon.h"

class Part{
public:
    char name[40];
    int id;
    int x, y;
    double gpl[6], pvl[6];
    Transform ptrans;
    // ***********************************************************************
    // Costructors and Destructors
    //Default Constructor
    Part(){
        strcpy(name, "default");
        id = 0;
        for(i=0; i<6; i++)
            gpl[i] = 0.0;
        pvl[1] = 0.0;
    }
    //Parameter Constructor
}
const char rxn[]{
    strcpy(name, nin);
    id = 0;
    for (i=0;i<6;i++) {
        ppl[i] = 0.0;
        pil[i] = 0.0;
    }
} //Destructor

//Member Functions

int get_id(){
    return this->id;
}

friend ostream& operator<<(ostream& os, Part& part){
    part.print_part();
    return os;
}

void print_part(){
    int i;
    cout << endl << "Part Name: " << name << endl;
    cout << "id#: " << id << endl;
    cout << endl;
}

class Liaison {
public:
    int id;
    int p1, p2;
    int type;
    int numf;
    Array<Feature> F;

    // **********************
    // Constructors and Destructors
    // Default Constructor
    Liaison(){
        id = 0;
        type = 0;
        numf = 0;
    }

    // Parameter Constructor
    Liaison(int i, int o1, int o2, int t){
        id = i;
        p1 = o1;
        p2 = o2;
        type = t;
        numf = 0;
    }

    // Destructor
    ~Liaison();

    // **********************
    // Member Functions
    int get_id(){
        return this->id;
    }

    friend ostream& operator<<(ostream& os, Liaison& liaison){
        liaison.print_liaison();
        return os;
    }

    void print_liaison(){
        int i;
        cout << endl << "Liaision: " << id << endl;
        cout << "parts: p1, p2, type: " << type << endl;
        cout << "number of features: " << numf << endl;
        for (i=0; i<numf; i++) {
            cout << "Feature # " << i << ':' << endl;
            F[i].print_ft_all();
        }
        return;
    }

};
Mfeature.h

```cpp
#include "Mtransform.h"
#include <math.h>

class Feature{
public: //Member Variables
Transform ft;
int type;
int id;
double atts[12];
int numa;

//Default Constructor
Feature()
{
    type = -1;
id = 0;
numa = 0;
for(i=0;i<12;i++)
atts[i] = 0.0;
}

//Destructor
virtual ~Feature() {}

//******************************************************************
//Member Functions
friend ostream& operator<< (ostream& os, Feature& feature){
    feature.print_ft_all(os);
    return os;
}

void print_ft_all()
{
    cout << "type = " << type << endl;
    cout << "Transform = " << endl;
    ft.print_tfnall();
}
};
```

Mtransform.h

```cpp
#include <math.h>

class Transform {
public: //Member Variables
    double x, y, z, tx, ty, tz;

//******************************************************************
//Constructors and Destructors

//Default Constructor
Transform()
{
x = 0; y = 0; z = 0; tx = 0; ty = 0; tz = 0;
}

//Parameter Constructor: pass a pointer to an array of 16 data // specifying the members of f. Column Wise.
Transform(double data[]){
x = data[0];
y = data[1];
z = data[2];
tx = data[3];
ty = data[4];
tz = data[5];
}

//Angle and Displacement Constructor
//takes as arguments the rotation angles in degrees. // and displacement vector for a homogeneous transform and returns //transform object
Transform(double XO, double YO, double ZO, double tx0, double ty0, double tz0) {
x = XO;
y = YO;
z = ZO;
tx = tx0;
ty = ty0;
tz = tz0;
}

//Destructor
~Transform() {}

//******************************************************************
//Member Functions
void print_tfm_f(){
    cout << endl << "X: " << x << endl;
    cout << "Y: " << y << endl;
    cout << "Z: " << z << endl;
    cout << "TX: " << tx << endl;
    cout << "TY: " << ty << endl;
    cout << "TZ: " << tz << endl;
    cout << endl;
}
};
```
main.C

/* standard header files */
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <ctype.h>
#include <limits.h>
#include <string.h>
#include <v racially.h>

/* Notify header files */
#include <win32.h>
#include <winuser.h>
#include <winlabel.h>
#include <winmain.h>
#include <winprotos.h>
#include <winutil.h>
#include <winblend.h>
#include <winlib.h>
#include <winresource.h>
#include <winresource.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
#include <winuser.h>
String types[] = {
  "Prametric Peg / Prismatic Hole (6 dof)",
  "Prismatic Peg / Prismatic Slot (5 dof)",
  "Prismatic Peg / Prismatic Slot (5 dof)",
  "Round Peg / Prismatic Slot (4 dof)",
  "Round Peg / Prismatic Slot (4 dof)",
  "Screwed Joint (4 dof)",
  "Elliptical Ball and Socket (4 dof)",
  "Plane-Plate Lap Joint (3 dof)",
  "Spherical Point (3 dof)",
  "Plate in Convex Hole (3 dof)",
  "Elliptical Ball in Cylindrical Trough (3 dof)",
  "Thin Rib / Plane Surface (2 dof)",
  "Ellipsoid on Plane Surface (2 dof)",
  "Spherical Ball in Cylindrical Trough (2 dof)",
  "Peg in Slotted Hole (2 dof)",
  "Spheroid on Plane Surface (1 dof)",
  "User Defined (2 dof)"
};
int n; i;

XMLEnterProc(NULL, NULL, NULL);

XMLExitProc();

top = XMHandInitialize(Lapp, "AC", NULL, 0, Large, 0,
FailbackResources, Null harassedResponse, NULL);

main_w = XMCreateMainWindow(top, "main_w", NULL, 0);
XMManageChild(main_w);

gcv.foreground = WhitePixelOfScreen(XBScreen(main_w));
gc = XMCreateGC(XDisplay(main_w),
          RootWindowOfScreen(XBScreen(main_w)), GCForeground, gcv);
gcv2.foreground = WhitePixelOfScreen(XBScreen(main_w));
gc2 = XMCreateGC(XDisplay(main_w),
          RootWindowOfScreen(XBScreen(main_w)), GCForeground, gcv2);

file = XMStringCreateLocalized("File");
modules = XMStringCreateLocalized("Modules");
member = XMCreateSimpleMenuBar(main_w, member,
               XMASCADE_MENUITEM, file, "F",
               XMASCADE_MENUITEM, modules, "M", NULL);

XMStringFree(file);
XMStringFree(modules);

open = XMStringCreateLocalized("Open...");
save = XMStringCreateLocalized("Save...");
quit = XMStringCreateLocalized("Quit");
SetCreateSimplePullDownMenu(member, "File menu", 0, file_cb,
               XMASCADE_BUTTON, save, "S", NULL, NULL,
               XMASCADE_BUTTON, quit, "Q", NULL, NULL,
               XMASCADE_MENUITEM, NULL, NULL, NULL);

XMStringFree(open);
XMStringFree(save);
XMStringFree(quit);

SPAS = XMStringCreateLocalized("SPAS");
IPUR = XMStringCreateLocalized("IPUR");
EXIT = XMStringCreateLocalized("EXIT");

COMMENT = XMStringCreateLocalized("Comment Analyzer");
menu = XMCreateSimplePullDownMenu(member, "Menu", menu,
               XMASCADE_BUTTON, SPAS, "S", NULL, NULL,
               XMASCADE_BUTTON, IPUR, "I", NULL, NULL,
               XMASCADE_BUTTON, COMMENT, "C", NULL, NULL,
               XMASCADE_BUTTON, EXIT, "E", NULL, NULL,
               XMASCADE_MENUITEM, NULL, NULL, NULL);

XMStringFree(SPAS);
XMStringFree(IPUR);
XMStringFree(EXIT);
XMStringFree(COMMENT);

COMMENTS = XMStringCreateLocalized("Comments");
FANAS = XMStringCreateLocalized("Fanas");
edit_menu = XMCreateSimplePullDownMenu(menu, "Edit menu", 3, edit_cb,
               XMASCADE_BUTTON, COMMENTS, "C", NULL, NULL,
               XMASCADE_BUTTON, FANAS, "F", NULL, NULL,
               XMASCADE_MENUITEM, NULL, NULL, NULL);

XMStringFree(COMMENTS);
XMStringFree(FANAS);
XMManageChild(member);

main_w = XMCreateFrame(main_w, "main_w", NULL, 0);
XMManageChild(main_w);

/* Create SPAS pane */
dfc_fr = XMCreateManagedWidget("dfc_fr", xsmFrameWidgetClass, main_f,
               XMHighlightAttachment, XmTackFORM,
               XMHighlightOffset, 0,
               XMHighlightAttachment, XmTackFORM,
               XMHighlightOffset, 0,
               XMHighlightAttachment, XmTackFORM,
               XMHighlightOffset, 0,
               XMHighlightAttachment, XmTackFORM,
               XMHighlightOffset, 0,
               XMHighlightAttachment, XMHIGHLIGHT_BK, NULL);

dfc_pane = XMCreateWidget("dfc_pane", xsmFrameShellWidgetClass, dfc_fr,
               XMHighlightWidth, 1, XMHighlightHeight, 1, NULL);

temp_text = XMStringCreateLocalized("Assembly Window");
dfc_label = XMCreateWidget("dfc_label", xsmLabelWidgetClass, dfc_pane,
               XMLabelString, temp_text,
               XMLabelAttachment, XmTackFORM,
               XMLabelOffset, 0,
               XMLabelAttachment, XmTackFORM,
               XMLabelOffset, 0,
               NULL);

XMStringFree(temp_text);
XMManageChild(dfc_label);

dfc_form = XMCreateManagedWidget("dfc_form", xsmFormWidgetClass, dfc_pane,
               NULL);

dfc_pane2 = XMCreateWidget("dfc_pane2", xsmFrameShellWidgetClass, dfc_form,
               XMHighlightWidth, 1, XMHighlightHeight, 1, NULL);

for (i=0; i<8; i++)
{  
dfc_p[i] = XMCreateManagedWidget("tool"[i], xsmPushButtonWidgetClass,
               dfc_pane2,
               XMHighlightAttachment, XmTackFORM,
               NULL);
    XAddCallback(dfc_p[i], XmActivateCallback, set_tool, tools[i]);
}
void file_cb(XWidget widget, XPointer client_data, XPointer call_data)
{
    int item_no = (int) client_data;
    XWidget dialog;
    XString t;

    switch(item_no) {
    case 0:
        t = XStringCreateLocalized("Enter name of assembly:");
        n = 0;
        XSetArg(arg[n], XSelectionLabelString, t); n++;
        XSetArg(arg[n], XSelectionTokenMessage, False); n++;
        dialog = XCreatePromptDialog(top, "Open Assembly", arg, n);
        XVaListValues(dialog, XNDialogStyle, XVADialog_FULL_APPLICATION_MODAL, NULL);
        XStringFree(t);
        break;
    case 1:
        t = XStringCreateLocalized("Enter name of assembly:");
        n = 0;
        XSetArg(arg[n], XSelectionLabelString, t); n++;
        XSetArg(arg[n], XSelectionTokenMessage, False); n++;
        dialog = XCreatePromptDialog(top, "Open Assembly", arg, n);
        XVaListValues(dialog, XNDialogStyle, XVADialog_FULL_APPLICATION_MODAL, NULL);
        XStringFree(t);
        break;
    case 2:
        sys.exit(0);
        break;
    }
}

void modules_cb(XWidget widget, XPointer client_data, XPointer call_data)
{
    int item_no = (int) client_data;
    char* command[80];
    switch(item_no) {
    case 0:
        save_files();
        save_src_dir();
        sprintf(command, "xterm -e /afs/users/asn/ASA/SPAV/SysGet/spec %s", asy_name);
        system(command);
        break;
    case 1:
        save_files();
        sprintf(command, "xterm -e /afs/users/asn/ASA/DFCH/DFCH %s", asy_name);
        system(command);
        break;
    }
}

80
fprint(computer);
break;
case 2:
if (check_features[i] == 1) {
    save_file(s);
sprintf(computer, "%s/users/mystery/ASA/ASA/ala %i < %s.in > %s.out",
    save_name, save_name, save_name, save_name);
system(computer);
sprintf(computer, "%s.out", save_name);
    fprint2(computer);
}
else
    sprintf("ERROR: not all liaisons have been assigned features!",
    computer);
break;
}
}

void edit_click(Widget widget, XPointer client_data, XPointer call_data)
{
    int item_no = (int) client_data;
    char *computer[60];

    switch(item_no) {
    case 0:
        save_files();
sprintf(computer, "op %s pr %s.pra", save_name, save_name);
system(computer);
sprintf(computer, "%s/users/mystery/ASA/ASA/ala/prd/prd-kb", save_name);
system(computer);
sprintf(computer, "%s/users/mystery/ASA/ASA/ala/mevalpr	kb", save_name);
system(computer);
    xterm = "%s/users/mystery/ASA/Edit/Editor %s.zap %s.exp",
    save_name, save_name);
system(computer);
    break;
    case 1:
        save_files();
sprintf(computer, "cat %s prd %s pr > %s.pra",
    save_name, save_name, save_name);
system(computer);
sprintf(computer, "%s/users/mystery/ASA/ASA/ala/prd/prd-kb", save_name);
system(computer);
sprintf(computer, "%s/users/mystery/ASA/ASA/ala/mevalpr	kb", save_name);
system(computer);
sprintf(computer, "%s/users/mystery/ASA/Edit/Editor %s.zap %s.exp",
    save_name, save_name);
system(computer);
    break;
    }
}

void dfo_callback(Widget widget, XPointer client_data, XPointer call_data)
{
    static Position x, y;
    static int movepart;
    XW pogningRealCallbackStruct *ths =
    ((XW pogningRealCallbackStruct *) call_data);
    XEvent *event = csp->event;
    if (csp->reason == XENTRY_INPUT)
        switch(current_tool) {
    case 0: /* add part */
        if (event->type == ButtonPress) {
            double x, y;
            int obj = -1;
            int lapart = -2;
            x = event->xbutton.x;
            y = event->xbutton.y;
            movepart = -1;
            which = select_obj(x, y);
            if (which == -1) {
                lapart = 1;
                else
                    lapart = 0;
            }
            if (lapart == 1)
                movepart = obj;
            else
                movepart = -2;
        }
        else if (event->type == ButtonRelease) {
        if (movepart == -1) {
            Widget dialog;
            XtStr = XtStringCreateLocalized("Enter name of part: ");
            Arg arg[5];
            int n = 0;
            x = event->xbutton.x;
            y = event->xbutton.y;
            tempstr[0] = x; tempstr[1] = y;
            XSetArg [arg, n];
            XtSelectionLabelString (t, n);
            XSetArg [arg, n];
            XtWindowString (t, n);
            dialog = XtCreatePopupDialog (dialog, "Add Part", arg, n);
            XtAddValues (dialog, XtEdit, "Add Part", 0, XtEditValues
dialog, XtDialogStyle, XtDIALOG_USED, XtDIALOG_APPLICATION_MODE,
            NLL);
            XtAddCallback (dialog, XtOkCallback, read_name, NLL);
            XtAddCallback (dialog, XtCancelCallback, read_name, NLL);
            XtAddCallback (dialog, XtOkCallback, read_name, NLL);
            XtAddCallback (dialog, XtCancelCallback, read_name, NLL);
            XtAddCallback (dialog, XtOkCallback, read_name, NLL);
        }
if (Which == (int)(which)) == 0)
    insert = 3;
else
    insert = ((int)(Which - int(which)));
obj = (int)(Which);
if (insert == 1)
    edit_part(obj);
else if (insert == 0)
    edit_iבן(obj);
else if (insert == -1)
    delete_feature(obj);

draw_dfl();
}
break;
case 7: /* delete */
if (event->type == ButtonRelease)
    double which;
    int obj = -3;
    int insert = -2;
    x = event->x;
    y = event->y;
    which = select_obj(x, y);
    if (which == -3)
        if (Which == (int)(which)) == 0)
            insert = 1;
        else
            insert = ((int)(Which - int(which)));
    else
        delete_part(obj);
    else if (insert == 0)
        delete_iבן(obj);
    else if (insert == -1)
        delete_feature(obj);

draw_dfl();
}
break;
default:
    break;
}
}

void edit_info(Widget widget, XPointer client_data, XPointer call_data)
{
    int i;
    double temptrans[4];
    int pmem = int(client_data);
    char *value = XtTransFieldGetString(tempw[6]);
    XString pmem
    for (i=0; i<6; i++)
        if (pmem == XtTransFieldGetString(tempw[i]))
            pmem
        if (pmem == XtTransFieldGetString(tempw[i]))
            pmem
    XtTransFieldGetString(tempw[6]);
    XDestroyWidget(GetSuccessful(widget));

draw_dfl();
}

void edit_part(int pmem)
{
    Widget dialog, pane, form, xwp, rowcol, tfom, label;
    Dimension dimension;
    int n = 0;
    char tampers[20];
    char *labels = (char *)malloc(20);
    char *labels[6] = { "Xi", "Yi", "T1", "T3", "TX", "TY" ];
    dialog = XCreatePopupShell("Edit Part",
        widgets, widgetClass,
        top,
        XtDeleteResponse, Xnone,
        XNone);
    XPutValues(dialog,
        XtChild, XtChild,
        XTACIÓN, XTACIÓN,
        XNone);
    pane = XCreateDialog("pane", widgets, widgetClass, dialog,
        BaseDirection, 1,
        BaseDirection, 1,
        NULL);
    form = XCreateDialog("form", widgets, widgetClass, pane, NULL);
    rowcol = XCreateDialog("rowcol", widgets, widgetClass, form, NULL);
    tfom = XCreateDialog("form", widgets, widgetClass, rowcol,
        AllocationRegion, 10, NULL);
    XCreateManageDialog(label, labels, widgetClass, tfom,
        XtAppAttach, XVIEW_FORM,
        XtAppAttach, XVIEW_FORM,
        XtAppAttach, XVIEW_FORM,
        XtAppAttach, XVIEW_POSITION,
        XtAppAttach, XVIEW_POSITION,
        XtAppAttach, XVIEW_POSITION,
        Xalign, NULL,
        Xalign, NULL,
        Xalign, NULL,
        Xalign, NULL,
        NULL,
null;

getText(6) = X11CreateManagedWidget("text", xmTextFieldWidgetClass,
传导, XmValue, PartArray[par].name,
XmHAlign, True, XmLeftPosition, 4,
XmNLLI);

XManageChild(form);
XManageChild(rowcol);
XManageChild(form);

form = X11CreateWidget("form", XmFormWidgetClass, par, NULL);

String tempstr = XmStringCreateLocalized("Transform: ");
text = X11CreateManagedWidget("text", XmTextFieldWidgetClass, form,
XmValue, tempstr, XmHAlign, XmALIGN XmLEFTBOTTOM, 
XmNLeftAttachment, XmALIGN XmTOPFORM,
XmNHeightAttachment, XmALIGN XmFORM,
XmNHeightPosition, 5, 
XmNLeftAttachment, XmALIGN XmFORM,
XmNLeftOffset, 5, 
XmNLLI);

XStringFree(tempstr);

XManageChild(label);
rowcol = X11CreateWidget("rowcol", XmFormColumnWidgetClass, form,
XmNRowAttachment, XmALIGN XmLEFTFORM,
XmNHeightAttachment, XmALIGN XmFORM,
XmNHeightPosition, 3, 
XmNAlign, XmALIGN XmLEFTFORM,
XmNLLI);

for (i = 0; i < NumberOfLabels; i++) {
    form = X11CreateManagedWidget("form", XmFormWidgetClass, rowcol,
XmHPosition, i, 
XManageChildManagedWidget(labels[i], labelWidgetClass, form,
XmNRowAttachment, XmALIGN XmLEFTFORM,
XmNHeightAttachment, XmALIGN XmFORM,
XmNHeightPosition, 3, 
XmNLeftAttachment, XmALIGN XmFORM,
XmNLeftPosition, 3, 
XmNAlign, XmALIGN XmLEFTFORM,
XmNLLI);

    switch (i) {
    case 0:
        printf(tempstr, "%d", PartArray[par].gpmems.x);
        break;
    case 1:
        printf(tempstr, "%d", PartArray[par].gpmems.y);
        break;
    case 2:
        printf(tempstr, "%d", PartArray[par].gpmems.x);
        break;
    case 3:
        printf(tempstr, "%d", PartArray[par].gpmems.x);
        break;
    case 4:
        printf(tempstr, "%d", PartArray[par].gpmems.x);
        break;
    case 5:
        printf(tempstr, "%d", PartArray[par].gpmems.x);
        break;
    }
    text[i] = X11CreateManagedWidget("text", XmTextFieldWidgetClass,
传导, XmValue, tempstr,
XmHAlign, True, 
XmLeftPosition, 4,
XmNLLI);

    XManageChild(form);
}

XManageChild(rowcol);
XManageChild(form);

form = X11CreateWidget("form", XmFormWidgetClass, par, NULL);

widx = X11CreateManagedWidget("OK", XmPushButtonWidgetClass, form,
XmValue, NULL, 
XmHAlign, True, 
XmLeftPosition, 4,
XmNLLI);

XAddCallback(widx, XmNactivateCallback, edit, info, (XPointer)par);
XAddValues(form, XmNdefaultActive, widx);

widx = X11CreateManagedWidget("Cancel", 
XmPushButtonWidgetClass, form,
XmValue, NULL, 
XmHAlign, True, 
XmLeftPosition, 4,
XmNLLI);

XAddCallback(widx, XmNactivateCallback, DestroyShell, dialog);

XManageChild(form);

XGetValues(widx, XmNheight, h, NULL);
XGetValues(form, XmPerimeterMin, h, XmPerimeterMin, b, NULL);

for (i = 0; i < NumberOfLabels; i++) {
    for (i = 0; i < NumberOfLabels; i++)
}
void edit_liaison(int limit) {
    Widget dialog, pane, form, widg, rowcol, tf, label, radio_box;
    Dimension d;
    int i, initype;
    char tempstr[20];
    char label[] = "Direction";
    XString one, two, thr;

    dialog = XCreatePopupShell("Edit Liaison",
        XWMShellOpaqueClass, top,
        XWMDeleteResponse, XDELETE, NULL);
    XCopyValues(dialog,
        WMShellStyle, XMALCOLL_FULL_APPLICATION_MODAL,
        NULL);
    pane = XCreateWidget("pane", xswmShellWindowClass, dialog,
        borderWidth, 1,
        backgroundPixel, 1,
        NULL);
    form = XCreateWidget("form", xswmFormClass, pane, NULL);
    XString tempstr = XStringCreateLocalized("Liaison Type: ");
    label = XCreateWidget("label", xswmLabelClass, form,
        XLabelString, tempstr,
        XLabelAlignment, XmALIGNMENT_BEGINNING,
        XLabelAttachment, XmATTACH_FORM,
        XmLabelWidth, 5,
        XmLabelHeight, 5,
        XmLabelOffset, 5,
        NULL);
    XStringFree(tempstr);
    XManageChild(form);
    radio_box = XCreateLabel("radio_box", xswmRadioBoxClass, form,
        XmNpacking, XmPACK_WIDGET,
        XmNlabel, tempstr,
        XmNlabelPosition, 5,
        XmNlabelWidth, 5,
        XmNlabelHeight, 5,
        XmNlabelOffset, 5,
        NULL);
    XStringFree(tempstr);

    if (LiaArray[im].type == 0) {
        initype = 2;
        tempstr[1] = initype;
        thr = XStringCreateLocalized("Thrust");
        one = XStringCreateLocalized("One");
        two = XStringCreateLocalized("Two");
        radio_box = XCreateRadioButtonShell(top, "radio_box", initype, edit_type,
            XmNshortcuts, xswmShellShortcuts, one, NULL, NULL, NULL,
            xswmShellShortcuts, two, NULL, NULL, NULL,
            xswmShellShortcuts, thr, NULL, NULL, NULL);
        XStringFree(tempstr);
    } else if (LiaArray[im].type == 1) {
        initype = 1;
        tempstr[1] = initype;
        one = XStringCreateLocalized("Liaison");
        two = XStringCreateLocalized("Two");
        radio_box = XCreateRadioButtonShell(top, "radio_box", initype, edit_type,
            XmNshortcuts, xswmShellShortcuts, one, NULL, NULL, NULL,
            xswmShellShortcuts, two, NULL, NULL, NULL);
        XStringFree(tempstr);
    } else {
        initype = 0;
        tempstr[1] = initype;
        one = XStringCreateLocalized("One");
        two = XStringCreateLocalized("Two");
        radio_box = XCreateRadioButtonShell(top, "radio_box", initype, edit_type,
            XmNshortcuts, xswmShellShortcuts, one, NULL, NULL, NULL,
            xswmShellShortcuts, two, NULL, NULL, NULL);
        XStringFree(tempstr);
    }

    if (LiaArray[im].type == 0) {
        sprintf(tempstr, "%d", LiaArray[im].type);
    } else {
        sprintf(tempstr, "-%d", LiaArray[im].type);
    }

    form = XCreateWidget("form", xswmFormClass, pane, NULL);
    tempstr = XStringCreateLocalized("Direction: ");
    label = XCreateWidget("label", xswmLabelClass, form,
        XLabelString, tempstr,
        XLabelAlignment, XmALIGNMENT_BEGINNING,
        XLabelAttachment, XmATTACH_FORM,
        XmLabelWidth, 5,
        XmLabelHeight, 5,
        XmLabelOffset, 5,
        NULL);
    XStringFree(tempstr);
    XManageChild(form);
    XStringFree(label);
}

87
```c
form = XCreateManagedWidget("form", MainFormWidgetClass, rowcol);
int type = 0;
temp[2] = intype;

88
```
void edit_feature(int *int)
{
    if (int == 1) {
        edit_feature(1, 0);
    } else {
        char feat[20][120];
        for (i = 0; i < len; i++) {
            sprintf(feat[i], *text, *int);
            *feat[i] = 't';
        }
    }
    show_image(int);
    Widget dialog;
    XString t, *str, *text;
    int j;
    str = (char *)malloc(len * sizeof(char *));
    for (i = 0; i < len; i++) {
        str[i] = XStringCreateLocalized(feat[i]);
    }
    return (strlen(feat) == 0);
}
void edit_feature(char client_data, XPointer call_data)
{
    char *value;
    int dialog_type, from;
    int len = (int)client_data;
    XSelectionGetCommonCallbackStruct *data = (XSelectionGetCommonCallbackStruct *)call_data;
    switch (data->reason) {
        case NO_OK:
            dialog_type = XEMAIL.BUTTON;
            SetSelectionData (-1, value, XEMAIL.BUTTON, NULL);
            return (value, "ok", &from);
        case value:
            ask_edit_feature(len, from-1);
            break;
        default:
            return (NULL);
    }
    XPopdown(client_data);
    XDestroyWidget(widget);
}
void ask_edit_feature(int *text, int *text)
{
    char test[256][256];
    int i;
    int obj = len;
    tempstr[0] = len, from, tempstr[1], label1, text;
    switch (len, len, from, tempstr[1], label1, text) {
        case 0:
            char *label1 = (""");
            tempstr[1] = 0;
            ask_edit_findividual(len, from, tempstr[1], label1, text);
            break;
        case 1:
            char *label1 = ("Positive Limit:");
            tempstr[1] = XEMAIL.BUTTON;
            if (text) *text = tempstr[1], label1, text;
            ask_edit_findividual(len, from, tempstr[1], label1, text);
            break;
        case 2:
            char *label1 = ("Negative Limit:");
            tempstr[1] = XEMAIL.BUTTON;
            if (text) *text = tempstr[1], label1, text;
            ask_edit_findividual(len, from, tempstr[1], label1, text);
            break;
    }
}
{| widget dialog, pane, form, width, rowcol, tform, label; |
| Dimension B; |
| int n = 0; |
| int i; |
| char tempat[20]; |
| char *label[] = { "X", "Y", "Z", "AX", "AY", "AZ" }; |
| dialog = XvCreatePopUpShell("Edit Feature", |
| xdialogShellClass, top, XWMHintToResponse, XWMHint, NULL); |
| XvSetClass(dialog, |
| XWMStyle, XGALO_ALL_APPLICATION_MENU, NULL); |
| pane = XvCreateWidget("pane", xwindowShellEdgeClass, dialog, |
| XWMNormalWidth, 1, XWMNormalHeight, 1, NULL); |
| form = XvCreateWidget("form", xformWidgetClass, pane, NULL); |
| Xstring tempat = XstringCreateLocalized("Transform in"); |
| tlabel = XvCreateWidget("tlabel", xwindowShellEdgeClass, form, |
| XWMLabelString, tempat, |
| XWMAlignment, XALIGNMENT_CENTER, |
| XWMStringAttachment, XSTRINGFORM, XWMSpacing5, 5, |
| XWMStringAttachment, XSTRINGFORM, |
| XWMStringOffset, XSTRINGFORM, XWMSpacing5, 5, NULL); |
| XStringFree(tempat); |
| XManageChild(tlabel); |
| rowcol = XvCreateWidget("rowcol", xwindowShellEdgeClass, form, |
| XWMStringAttachment, XSTRINGFORM, XWMString, XWMStringOffset, 5, NULL); |
| for (i=0; i<NumberLabels); i++) { |
| tform = XvCreateWidget("form", xformWidgetClass, rowcol, |
| XWMNormalWidth, 10, NULL); |
| XvCreateManagedWidget(labels[i], xwindowShellEdgeClass, tform, |
| XWMStringAttachment, XSTRINGFORM, |
| XWMStringAttachment, XSTRINGFORM, |
| XWMStringAttachment, XSTRINGFORM, XWMStringPosition, 3, |
| XWMAlignment, XALIGNMENT_END, NULL); |
| switch (i) { |
| case 0: |
| sprintf(tempat, "%.f", xArray[labels[i]].F[frames].frames.x); |
| break; |
| case 1: |
| sprintf(tempat, "%.f", xArray[labels[i]].F[frames].frames.y); |
| break; |
| case 2: |
| sprintf(tempat, "%.f", xArray[labels[i]].F[frames].frames.z); |
| break; |
| case 3: |
| sprintf(tempat, "%.f", XArray[labels[i]].F[frames].frames.xd); |
| break; |
| case 4: |
| sprintf(tempat, "%.f", XArray[labels[i]].F[frames].frames.yd); |
| break; |
| case 5: |
| sprintf(tempat, "%.f", XArray[labels[i]].F[frames].frames.zd); |
| break; |
| } |
| ttextarea[i] = XvCreateManagedWidget("textarea", xwindowShellEdgeClass, |
| form, XWMValue, tempat, |
| XWMStringAttachment, XSTRINGFORM, |
| XWMStringAttachment, XSTRINGFORM, XWMStringPosition, 4, NULL); |
| XManageChild(form); |
| XManageChild(rowcol); |
| XManageChild(tlabel); |
| if (xArray[labels[i]].F[frames].type == 0) { |
| form = XvCreateWidget("form", xformWidgetClass, pane, NULL); |
| XString tempat = XstringCreateLocalized("Attributes in"); |
| tlabel = XvCreateWidget("tlabel", xwindowShellEdgeClass, form, |
| XWMLabelString, tempat, |
| XWMAlignment, XALIGNMENT_CENTER, |
| XWMStringAttachment, XSTRINGFORM, XWMSpacing5, 5, |
| XWMStringAttachment, XSTRINGFORM, XWMStringPosition, 5, |
| XWMStringAttachment, XSTRINGFORM, XWMSpacing5, 5, NULL); |
| XStringFree(tempat); |
| XManageChild(tlabel); |
| rowcol = XvCreateWidget("rowcol", xwindowShellEdgeClass, form, |
| XWMStringAttachment, XSTRINGFORM, XWMString, XWMStringOffset, 5, NULL); |
| for (i=0; i<NumberLabels; i++) { |
| tform = XvCreateWidget("form", xformWidgetClass, rowcol, |
```c
row = (int)((y)/(PIX/4));
col = (int)((x)/(PIX/4));
current_feed = col * row;
draw_feed(x, y, current_feed);
}

void draw_feed(x, y, current_feed)
{
    int row, col;
    PIX = 72;
    curr_feed = 0;
    pixmap = XCreatePixmap(display, canvas, width, height, 0);
    XtPointer window, w, c;
    x = 0;
    y = 0;
    x = XGetPixmapPtr(c, pixmap, pixmap, PIX, PIX, 0, 0, 0, 0, 0, 0);
    if (x) {
        fprintf(stderr, "cannot get pixmap!");
        return;
    }
    for (row = 0; row < PIX; row++) {
        for (col = 0; col < PIX; col++) {
            switch (current_feed) {
                case 0:
                    printf("a Prismatic Peg / Prismatic Hole 6 X Y Z T1 T2 T3 T4 \nbreak; case 1:
                    printf("b Plate Pin in Through Hole 5 X Y Z T1 \nbreak; case 2:
                    printf("c Prismatic Peg / Prismatic Slot 5 X Z T1 T2 \nbreak; case 3:
                    printf("d Plate Pin in Slotted Hole 4 X Z T1 T2 \nbreak; case 4:
                    printf("e Round Peg / Prismatic Slot 4 X Z T1 T2 \nbreak; case 5:
                    printf("f Round Peg / Through or Blind Hole 4 X Y T1 T2 \nbreak; case 6:
                    printf("g Threaded Joint 4 X Y T1 T2 \nbreak; case 7:
                    printf("h Elliptical Ball and Socket 4 X Y T1 T2 \nbreak; case 8:
                    printf("i Plate-Plate Lap Joint 3 Z T1 T2 \nbreak; case 9:
                    printf("j Spherical Joint 3 X Y \nbreak; case 10:
                    printf("k Parallel Joint 2 X Y T1 T2 \nbreak;
            default:
                printf("unknown feed \nbreak; break;
            }
        }
    }
}
```

The provided code snippet is a part of a C program that appears to be generating a display or a graph based on various feed options. The code includes a function `draw_feed` that takes coordinates `x`, `y`, and a current feed `current_feed` as inputs to draw different types of feed options on a pixmap. It uses a switch statement to determine the type of feed and print the corresponding message.
<table>
<thead>
<tr>
<th>Case</th>
<th>Feature Description</th>
<th>Dimensions</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Elliptical Ball in Cylindrical Trough</td>
<td>3 x 2 x 3</td>
</tr>
<tr>
<td>2</td>
<td>Elliptical Ball on Plane Surface</td>
<td>2 x 2 x 3</td>
</tr>
<tr>
<td>3</td>
<td>Elliptical Ball on Plane Surface</td>
<td>1 x 2 x 3</td>
</tr>
<tr>
<td>4</td>
<td>Spherical Ball in Cylindrical Trough</td>
<td>2 x 2 x 2</td>
</tr>
<tr>
<td>5</td>
<td>Peg in Slotted Hole</td>
<td>2 x 3 x 2</td>
</tr>
<tr>
<td>6</td>
<td>Spheroid on Plane Surface</td>
<td>1 x 2 x 3</td>
</tr>
<tr>
<td>7</td>
<td>User Defined</td>
<td>1 x 2 x 3</td>
</tr>
</tbody>
</table>

C14

/* The procedure GetTopShell was
 * Written by Dan Heller and Paula Ferguson.
 * Copyright 1994, O'Sullivan & Associates, Inc.
 */

void GetTopShell(Widgeet widget, XPointer client_data, XPointer call_data)
{
    XCopyAreaDisplay(screen, display, spixmap, csa->window, fgid1, 0, 0, sz1, 0, 0);
}

/* The procedure DestroyShell was
 * Written by Dan Heller and Paula Ferguson.
 * Copyright 1994, O'Sullivan & Associates, Inc.
 */

void DestroyShell(Widgeet widget, XPointer client_data, XPointer call_data)
{
    Widget shell = (Widget)client_data;
    XtDestroyWidget(shell);
}

void get_feature(Widgeet widget, XPointer client_data, XPointer call_data)
{
    int j = tempint[1];
    int i;
    int obj = (int)client_data;
    for (i=0; i<4; i++)
    {
        char *value = XStringFieldGetString(text[i][1]);
        Xt XmString(value);
    }
    free(tempint[1]);
    XtDestroyWidget(GetTopShell(widget));
}

/* The procedure ask_info was
 * Written by Dan Heller and Paula Ferguson.
 * Copyright 1994, O'Sullivan & Associates, Inc.
 */

void ask_info(Widgeet widget, XPointer client_data, XPointer call_data, char *labels[], char textlab[256])
{
    Widget dialog, pane, form, widg, rowcol, tfbox, label;
    Dimension h;
    int n = 0;
    int i;
    XtString strlab;

    /* code here */
}
switch(current_feat) {
  case 0:
    ListArray[list].f.addP0();
    ListArray[list].f(ListArray[list].numf).type = current_feat;
    ListArray[list].f(ListArray[list].numf).refr = current_feat;
    ListArray[list].f(ListArray[list].numf).torq = current_feat;
    ListArray[list].f(ListArray[list].numf).m1 = 0;
    ListArray[list].f(ListArray[list].numf).m2 = 0;
    ListArray[list].f(ListArray[list].numf).id = ListArray[list].numf;
    ListArray[list].numf++;
    draw_dcf();
    break;
  case 1:
    char *label1[] = { "Positive Limit:\", "Negative Limit:\"};
    tempint1 = XNumber(label1);
    sprintf(textlab, "Enter the angular limits (in degrees) on rotation about the pin axis (TE1\')\";
    ask_info_widget( client_data, call_data, label1, textlab);
    break;
  case 2:
    char *label2[] = { "Positive Limit:\", "Negative Limit:\"};
    tempint1 = XNumber(label2);
    sprintf(textlab, "Enter the positive and negative limits on Y motion\";
    ask_info_widget( client_data, call_data, label2, textlab);
    break;
  case 3:
    char *label1[] = { "L1:\", "d1:\"};
    tempint1 = XNumber(label1);
    sprintf(textlab, "Enter the length (l) of the slot and the diameter (d) of the peg\";
    ask_info_widget( client_data, call_data, label1, textlab);
    break;
  case 4:
    char *label4[] = { "Py:\", "Ny:\", "Pz:\", "Nz:\"};
    tempint1 = XNumber(label4);
    sprintf(textlab, "Enter the positive and negative limits on Y motion and the angular limits (in degrees) on rotation about the peg axis (TE1\')\";
    ask_info_widget( client_data, call_data, label4, textlab);
    break;
  case 5:
    char *label5[] = { "Pz:\", "Nz:\", "Pm:\", "Nm:\"};
    tempint1 = XNumber(label5);
    sprintf(textlab, "Enter the angular limits (in degrees) on rotation about the Y and Z axes (TE1\')\";
    ask_info_widget( client_data, call_data, label5, textlab);
    break;
  case 6:
    char *label6[] = { "p1:\", "Pz:\", "Nz:\"};
    tempint1 = XNumber(label6);
    sprintf(textlab, "Enter the pitch (p) of the threads in units of rad\";
    ask_info_widget( client_data, call_data, label6, textlab);
    break;
  case 7:
    char *label7[] = { "Py:\", "Ny:\", "Pz:\", "Nz:\"};
    tempint1 = XNumber(label7);
    sprintf(textlab, "Enter the positive and negative limits on Z motion and the angular limits (in degrees) on rotation about the peg axis (TE1\')\";
    ask_info_widget( client_data, call_data, label7, textlab);
    break;
  case 8:
    char *label8[] = { "Pm:\", "Nm:\", "Pn:\", "Nn:\"};
    tempint1 = XNumber(label8);
    sprintf(textlab, "Enter the angular limits (in degrees) on rotation about the Y and Z axes (TE1\')\";
    ask_info_widget( client_data, call_data, label8, textlab);
    break;
  case 9:
    char *label9[] = { "Pz:\", "Nz:\", "Py:\", "Ny:\", "Pm:\", "Nm:\"};
    tempint1 = XNumber(label9);
    sprintf(textlab, "Enter the angular limits (in degrees) on rotation about the X, Y, and Z axes (TE1\')\";
    ask_info_widget( client_data, call_data, label9, textlab);
    break;
  case 10:
    char *label10[] = { "ch:\", "dpin:\"};
    tempint1 = XNumber(label10);
    sprintf(textlab, "Enter the diameter of the oversize hole (ch) and pin (dpin)\";
    ask_info_widget( client_data, call_data, label10, textlab);
    break;
  case 11:
    char *label11[] = { "Py:\", "Ny:\", "Pz:\", "Nz:\", "Pm:\", "Nm:\"};
    tempint1 = XNumber(label11);
    sprintf(textlab, "Enter the limits on Y axis motion and the angular limits (in degrees) on rotation about the X and Z axes (TE1\')\";
    ask_info_widget( client_data, call_data, label11, textlab);
    break;
  case 12:
    char *label12[] = { "Pz:\", "Nz:\", "Py:\", "Ny:\", "Pm:\", "Nm:\", "Pn:\", "Nn:\"};
    tempint1 = XNumber(label12);
    sprintf(textlab, "Enter the limits on X and Y axis motion and the angular limits (in degrees) on rotation about the X and Z axes (TE1\')\";
    ask_info_widget( client_data, call_data, label12, textlab);
    break;
  case 13:
    char *label13[] = { "Py:\", "Ny:\", "Pm:\", "Nm:\", "Pn:\", "Nn:\"};

96
void get_num_features (Widget widget, XPointer client_data, XPointer call_data)
{
    char *name = (char *)client_data;
    int obj = NULL;
    char *test = NULL;
    int nf;

    obj = tempint[0];
    t = obj_value;
    strprinting(t, "", &test);
    savef(test, "W", &test);
    XPFree(test);
}

void add_liaison(int pl, int p2, int type)
{
    int i;
    int exist = 0;

    for (i = 0; i < num_liai; i++)
        if ((liaiArray[i].gl == pl) & (liaiArray[i].gp == p2))
            exist = 1;

    if (exist == -1) {
        LIAIArray[exist].id = exist;
        LIAIArray[exist].gl = pl;
        LIAIArray[exist].gp = p2;
        LIAIArray[exist].type = type;
        LIAIArray[exist].numf = 0;
    }
}

void delete_feature (int lines)
{
    int i;

    if (LIAIArray[lines].numf == 1) {
        LIAIArray[lines].numf--;
        LIAIArray[lines].P.remove(LIAIArray[lines].numf);
    } else {
        for (i = 0; i < LIAIArray[lines].numf; i++) {
            printf("%s", "%d", i + 1);
            struct(tests[i], types[LIAIArray[lines].P[i].type]);
        }
    }
}
void delete鳀e_ch (Widget widget, XPointer client_data, XPointer call_data)
{
    char *value = NULL;
    int dialog_type, from;
    int line = (int) client_data;
    XSelectionBoxCallBackStruct *cbms =
        (XSelectionBoxCallBackStruct *) call_data;
    switch (cbms->reason) {
        case XmNOK:
            XmStringGetLineLabel (value, cbms->value, XmSTRING_DEFAULT_WIDTH, value);
            saved_value = value;
            XmFree (value);
            delete鳀e_ch (line, from);
            break;
        default: /* nothing selected and OK pushed */
            break;
    }
    XmPopups (XPointer (widget));
    XmDestroyWidget (widget);
}

void delete鳀e (int line, int from)
{
    int i;
    LineArray [line]. cnamogenic;
    LineArray [line]. F.remove (from);
    for (i = 0; i < LineArray [line]. nmax; i++)
        LineArray [line]. F[i]. id = i;
    draw_adj(l);
}

void delete_liaison (int line)
{
    int i;
    mm_list--;
    LiaiArray.remove (line);
    for (i = 0; i < mm_list; i++)
        LiaiArray[i]. id = i;
}

void delete_parts (int parts)
{
    int i;
    nm_list--;
    PartsArray.remove (parts);
    for (i = nm_list; i < nm_list; i++)
        if (LinesArray [i] . gl = parts || LinesArray [i] . g2 = parts)
            delete_liaison(i);
        else
            if (LiaiArray [i] . g1 > parts)
                LiaiArray [i] . g1 = LiaiArray [i] . g2 - 1;
            if (LiaiArray [i] . g2 > parts)
                LiaiArray [i] . g2 = LiaiArray [i] . g1 + 1;
    for (i = nm_list; i < nm_list; i++)
        PartsArray [i] . id = i;
}

void show_liaison (int line)
{
    char message [32];
    char tempw [256];
    char full [60];

double selectobj(int x, int y)
{
    int found = -1;
    int ispart = -2;
    int i;
    double r, d, a, olda = INT_MIN;
    double xl, yl, x2, y2;
    for (i=0; i<nparts; i++)
    {
        r = sqrt((LiaiArray[i][1].x-LiaiArray[i][0].x) * (LiaiArray[i][1].x-LiaiArray[i][0].x) + (LiaiArray[i][1].y-LiaiArray[i][0].y) * (LiaiArray[i][1].y-LiaiArray[i][0].y));
        d = sqrt((x-LiaiArray[i][0].x) * (x-LiaiArray[i][0].x) + (y-LiaiArray[i][0].y) * (y-LiaiArray[i][0].y)) + sqrt((x-LiaiArray[i][1].x) * (x-LiaiArray[i][1].x) + (y-LiaiArray[i][1].y) * (y-LiaiArray[i][1].y));
        a = M_PI*(d/2) * (sqrt(d*d-r*r) /2);
        if ((a < olda) && (a < (10*d)))
        { 
            olda = a;
            found = i;
            ispart = 0;
            if (LiaiArray[i].nanf != 0)
            {
                x1 = LiaiArray[i].p1.x;
                y1 = LiaiArray[i].p1.y;
                x2 = LiaiArray[i].p2.x;
                y2 = LiaiArray[i].p2.y;
                r = 10;
                d = sqrt((x-(x1+x2)/2)*(x-(x1+x2)/2) + (y-(y1+y2)/2) * (y-(y1+y2)/2));
                if (d < r)
                { 
                    a = M_PI*r*r;
                    olda = a;
                    found = i;
                    ispart = -1;
                }
            }
        }
        else if (x < 0)
        { 
            x = x1 - x;
            y = y1 - y;
            r = sqrt(x*x + y*y);
            x = (10*(double)x)/r;
            y = (10*(double)y)/r;
            x3 = x1;
            y3 = y1;
            x4 = x3 + x;
            y4 = y3 + y;
            while (!done)
            { 
                if (x < 0) 
                { 
                    if (x < x2) 
                    { 
                        x4 = x2;
                        y4 = y2;
                        done = 1;
                    }
                    else if (x > 0) 
                    { 
                        if (x > x2) 
                        { 
                            x4 = x2;
                            y4 = y2;
                            done = 1;
                        }
                        else 
                        { 
                            x4 = x2;
                            y4 = y2;
                            done = 1;
                        }
                    }
                }
            }
        }
    }
}
}
XDrawLine(disp, d, gc, x3, y3, x4, y4);
x3 = x3 + x;
y3 = y3 + y;
x4 = x3 + x;
y4 = y3 + y;
}
else {
    if (y < 0) {
        if (y4 < y2) {
            x4 = x2;
y4 = y2;
done = 1;
        }
    }
    if (y3 > y2) {
        XDrawLine(disp, d, gc, x3, y3, x4, y4);
x3 = x4 + x;
y3 = y4 + y;
x4 = x3 + x;
y4 = y3 + y;
    }
} else done = 1;
}
}

/* This algorithm taken from Friehs Menckinagara's DEFEDITOR */
void drawArrow(Display *disp, Drawable d, GC gc, int x1, int y1, int x2, int y2, int r1)
{
    double headK, headW, arrowangle, wingangle, int frontTipT, frontTipY, rightTipX, rightTipY,
        leftTipX, leftTipY, leftT1, arrowlength;
xPoint p1, p2, p3, p4;
xPoint points[4];

    if (x2 == x1) {
        if (y2 > y1)
            arrowangle = 3 * PI / 2;
        else
            arrowangle = M_PI / 2;
    }
    else
        arrowangle = atan2((double)x2 - (double)x1,
            (double)x2 - (double)x1) + M_PI;

    wingangle = atan(ARROW_WIDTH/ARROW_TIP_LENGTH);
    arrowlength = sqrt(pow(ARROW_WIDTH, 2) + pow(ARROW_TIP_LENGTH, 2));
    headK = x2 + r * cos(arrowangle);
    headW = y2 + r * sin(arrowangle);

    /* The front tip of the arrow head pointing the child */
    frontTipT = headK;
    frontTipY = headW;

    /* The rear tip of the arrow */
    rearTipT = headK + ARROW_LENGTH*cos(arrowangle);
    rearTipY = headW + ARROW_LENGTH*sin(arrowangle);

    /* The right() tip of the arrow wing. */
    rightTipX = headK + winglength*cos(arrowangle - wingangle);
    rightTipY = headW + winglength*sin(arrowangle - wingangle);

    /* The left() tip of the arrow wing. */
    leftTipX = headK + winglength*cos(arrowangle + wingangle);
    leftTipY = headW + winglength*sin(arrowangle + wingangle);

    p1.x = frontTipT; p1.y = frontTipY;
p2.x = rightTipX; p2.y = rightTipY;
p3.x = rearTipT; p3.y = rearTipY;
p4.x = leftTipX; p4.y = leftTipY;

    XFillPolygon(disp, d, gc, points, 4, Convex, CoordModeOrigin);
}
void read_masters/Midget widget, Minitester client_data, XPointer call_data)
{
    SelectionsCallbackStruct *data = XGetSelectionsCallbackStruct (*call_data);
    int x1, y1, x2, y2;
    int x, y;
in x, y1, n1, n2;
double r;
    XString s;
    genericist pointlist;
    char *text;
    int dot;

    x1 = tempint[0]; y1 = tempint[1];
x2 = tempint[2]; y2 = tempint[3];
n1 = tempint[4]; n2 = tempint[5];

    /*
     * The right() tip of the arrow wing.
     */
    rightTipX = headK + winglength*cos(arrowangle - wingangle);
    rightTipY = headW + winglength*sin(arrowangle - wingangle);

    /* The left() tip of the arrow wing. */
    leftTipX = headK + winglength*cos(arrowangle + wingangle);
    leftTipY = headW + winglength*sin(arrowangle + wingangle);

    p1.x = frontTipT; p1.y = frontTipY;
p2.x = rightTipX; p2.y = rightTipY;
p3.x = rearTipT; p3.y = rearTipY;
p4.x = leftTipX; p4.y = leftTipY;

    XFillPolygon(disp, d, gc, points, 4, Convex, CoordModeOrigin);
}
t = cbs->value;
XrrStringGetLtoR(t, "", &text);
sscanf(text, "%d", &dof);
XtFree(text);
aicjliaism(nl, n2, df);
drawdfc 0;
XtDestroyWidget(widget);

void get_pinfo (Widget widget, XtPointer client-data, XtPointer call data)
{
int i;
double terptrans[6];
i = (int)client-data;
for (i=0; i<6; i++)
{
char *value = MruTempielcdetString(ttextw[i]);
sscanf(value, "%lf", &terptrana[i]);
XtFree(value);
}
PartArray[ (gnm-l].ptrans = terptrans;
XtDestroyidget(GetTrpShell (widget));
}

void read_name (Widget widget, Widget client_data, Widget call_data)
{
SelextionBoxCallbackStruct *cbs =
(SelextionBoxCallbackStruct *) call_data;
int x, y;
XrrString t;
TempList formlist;
char *text;
if (fcountList == NULL)
{
return;
}

for (i=0; i<XtNumber(labels); i++)
{
tform = XtVaCreateWidget("form", xfForWidgetClass, tform, XnNfracticrnase, 10, NULL);
XtVaCreateManagecdidget (labels [i], anLabelGadgetClass, tform,
XaNtopAttachment, XnATI _ FORM,
MrbotcrAttachent, lnATIMCHFtRM,
XntleftAttachnent, XnATIOLFORM,
XrightAttachrent, bnALIRPOSTICN,
XrNrightPositicm, 3,
Xnfalignt, XnALIQ4ENTFlD,
NLL);
ttextw[i] = XtVaCreateAanagedidget("ttectw", :arIlextFieldidgetClass,
tform, XnNtraversalCn, True,
MarrightAttachnnt, >nATIFtRM,
XatleftAttachient, XmATIh _POSITICN,
XMrNleftPositicm, 4,
JLXL);
XtManageChild (tform);
}
XManageChild(tform);

form = XtVaCreateWidget("form2", xfForirWidgetClass, form, MntkightAttachment, True,
XtNleftAttachment, XMALIGHT_ATTACHMENT,
XrightAttachment, XMATCH_POSITION,
XrightPosition, 3,
Xlignment, XMALIGNMENT_H,
NLL);
ttextw[i] = XtVaCreateManagedidget("ttect", xfForixedFieldidgetClass,
form, XAPHimportation, True,
XrightAttachment, XMATCH_FORMAT,
XleftAttachment, XMATCH_FORMAT,
XrightPosition, 4,
NLL);
XManageChild(form);

XManageChild(tform);

form = XtVaCreateWidget("form", xfForomiidgetClass, pane, XrightFractionBase, 3, NULL);

widg = XtVaCreateManagedidget("OV", xfForixedFieldidgetClass, form,
XAPHimportation, True,
XrightAttachment, XMATCH_FORMAT,
XleftAttachment, XMATCH_FORMAT,
XrightPosition, 1,
XleftAttachment, XMATCH_POSITION,
NLL);}
```c
#include <stdio.h>
#include <stdlib.h>

int i, j;

int main() {
    int array[10];
    for (i = 0; i < 10; i++) {
        array[i] = i * i;
    }
    return 0;
}
```
double transform_x(Transform trans, double x, double y, double z)
{
    double cx, cy, cz, sx, sy, sz, x0, y0, z0;
    double result;
    cx = cos(trans.tx*M_PI/180);
    cy = cos(trans.ty*M_PI/180);
    cz = cos(trans.tz*M_PI/180);
    sx = sin(trans.tx*M_PI/180);
    sy = sin(trans.ty*M_PI/180);
    sz = sin(trans.tz*M_PI/180);
    x0 = 0;
    y0 = 0;
    z0 = 0;
    result = cy*cy*(x-x0) + (sx*cy*cy+cz)*y0 + (sx*cy*sz-cz)*z0;
    return result;
}

double transform_y(Transform trans, double x, double y, double z)
{
    double cx, cy, cz, sx, sy, sz, x0, y0, z0;
    double result;
    cx = cos((trans.tx*M_PI/180));
    cy = cos((trans.ty*M_PI/180));
    cz = cos((trans.tz*M_PI/180));
    sx = sin((trans.tx*M_PI/180));
    sy = sin((trans.ty*M_PI/180));
    sz = sin((trans.tz*M_PI/180));
    x0 = 0;
    y0 = 0;
    z0 = 0;
    result = -cy*cy*(y-y0) + (-sx*cy+cz)*x0 + (sx*cy*sz-cz)*z0;
    return result;
}

double transform_z(Transform trans, double x, double y, double z)
{
    double cx, cy, cz, sx, sy, sz, x0, y0, z0;
    double result;
    cx = cos((trans.tx*M_PI/180));
    cy = cos((trans.ty*M_PI/180));
    cz = cos((trans.tz*M_PI/180));
    sx = sin((trans.tx*M_PI/180));
    sy = sin((trans.ty*M_PI/180));
    sz = sin((trans.tz*M_PI/180));
    x0 = 0;
    y0 = 0;
    z0 = 0;
    result = sy*sy*(z-z0) + (-sx*sy)*x0 + (sy*cy)*z0;
    return result;
}

void ewb_esc_chars()
{
    int i, j, k, l;
    double x, y, z;
    int dx[99][199][2];
    double led[99][3];
    double tled[99][3];
    double fed[18][3] = {
        {0, 0, 1},
        {0, 0, 1},
        {0, 0, 0},
        {0, 0, 1},
        {0, 1, 1},
        {0, 0, 1},
        {0, 0, 0},
        {0, 0, 1},
        {0, 0, 1},
        {0, 0, 1},
        {0, 0, 1},
        {0, 0, 0},
        {0, 0, 0},
        {0, 0, 0},
        {0, 0, 0},
        {0, 0, 0},
        {0, 0, 0},
        {0, 0, 0},
        {0, 0, 0},
        {0, 0, 0},
    };
    char mc_file[40];
    FILE *f;
    strcpy(mc_file, ewb_name);
    strstream_file = "mc";
    if (check_features() == 1) {
        for (i=0; i<var_list; i++) {
            x = 0;
            for (j=0; j<lineArray[i].ref; j++) {
                tled[0][0] = fed[lineArray[i].f][j].type[0];
                tled[1][1] = fed[lineArray[i].f][j].type[1];
                tled[2][2] = fed[lineArray[i].f][j].type[2];
                x = tled[0][0];
                y = tled[1][1];
                z = tled[2][2];
                if ((x == 0) && (y == 0) && (z == 0))
                    k = 1;
                tled[0][0] = transform_x(lineArray[i].f[j].x, x, y, z);
                tled[1][1] = transform_y(lineArray[i].f[j].x, x, y, z);
                tled[2][2] = transform_z(lineArray[i].f[j].x, x, y, z);
            }
        }
    }
}
ted[1][2] = transform_s(LiaiArray[i].Fl[1].Ftrans, x, y, z);

x = ted[1][0];
y = ted[1][1];
z = ted[1][2];

ted[1][0] = transform_s(PartArray[LiaiArray[i].p1].ptrans, x, y, z);

if (k == 1) {
    led[i][0] = 0;
    led[i][1] = 0;
    led[i][2] = 0;
}
else if (LiaiArray[i].numf == 1) {
    led[i][0] = ted[0][0];
    led[i][1] = ted[0][1];
    led[i][2] = ted[0][2];
}
elser = 1.
for (pcl = LiaiArray[i].numf; j++)
    if ((fabs(ted[k][0] - ted[0][0]) > 0.001) ||
        (fabs(ted[k][1] - ted[0][1]) > 0.001) ||
        (fabs(ted[k][2] - ted[0][2]) > 0.001))
        k = 0;
    if (k == 1) {
        led[i][0] = ted[0][0];
        led[i][1] = ted[0][1];
        led[i][2] = ted[0][2];
    } else {
        led[i][0] = 0;
        led[i][1] = 0;
        led[i][2] = 0;
    }
}

for (i = 0; i < runliai; i++)
    for (j = 0; j < 99; j++) {
        dv[i][j][0] = 0;
        dv[i][j][1] = 0;
    }
f = fopen(file, "w");

x = 0;
for (i = 0; inxnliai; i++)
    for (j = 0; j < runliai; j++)
        fprintf(f, "%f %f %f
", led[i][0], led[i][1], led[i][2]);

for (i = 0; i < runliai; i++)
    for (j = 0; j < runliai; j++)
        fprintf(f, "0 0 0

");

for (i = 0; i < runliai; i++)
    for (j = 0; j < runliai; j++)
        fprintf(f, "%d,%d
", dv[i][j][0], dv[i][j][1]);
printf(f1, "%d", LialArr)(1).type);
}
else {
    printf(f1, "? ");
    if (LialArr)(1).type > 0 {  
        printf(f2, "%d");
        printf(f3, "%s");
    }
}
}

printf(f4, "\n", PartArr)(1).name);
}
close(f1);
close(f2);
close(f3);
close(f4);
close(f5);

void set_color(Wlndex client_data, XPointer call_data)
{
    String color = (String) client_data;
    Display *dpy = XDisplay(wlndex);
    Colormap cmap = DefaultColormap(Display(XScreen(dpy, widget)));
    Color col, used;

    if (!AllocNamedColor(dpy, cmap, color, &col, &used)) {
        buf[10] = "Can't alloc \n" color;
        XWarning(buf);
        return;
    }
    XSetForeground(dpy, gc, col.pixel);
}

void redraw_dCC(Wlndex client_data, XPointer call_data)
{
    DrawableAreaCallbackStruct *cb =
        (DrawableAreaCallbackStruct *) call_data;
    XCopyArea(dpy, source_area, expose, display, pixmap, dce, window, gc,
              0, 0, width, height, 0, 0);
}

void draw_dCC()
{
    int i;
    int x, y, c, xl, x2, y2;
    char str[10];
    char strr[10];
    XtSetArg(  
    XtSetClilist fromlist;
    XtSetForeground(XDisplay(dce_area), gc,
        XWtWindw(XDisplay(dce_area), DCE, XScreen(dce_area)));
    XFillRectangular(dpy, dce_area, pixmap, gc, 0, 0, width, height);
    XtSetForeground(XDisplay(dce_area), gc,
        XWtWindow(XDisplay(dce_area), DCE, XScreen(dce_area)));
    for (i = 0; i <= parts; i++) {
        set_color(dce_area, "Black", NELL);
        XFillRectangles(dpy, dce_area, XWtWindow(dce_area), gc,
            PartArr)(1).x-5, PartArr)(1).x+5, 10, 10, 0, 360, 04);
        XFillRectangles(dpy, dce_area, pixmap, gc,
            PartArr)(1).x-5, PartArr)(1).x+5, 10, 10, 0, 360, 04);
        t = StringCreateLocalized(PartArr)(1).name);
        XWtStringValues(dpy, gc[0], XtStringList, fromlist, NELL);
        XWtStringDraw(dpy, dce_area, XWtWindow(dce_area), fromlist, t, gc,
            PartArr)(1).x-5, PartArr)(1).x+5, 10, 10, 0, 360, 04);
        XWtStringDraw(dpy, dce_area, pixmap, fromlist, t, gc,
            PartArr)(1).x-5, PartArr)(1).x+5, 10, 10, 0, 360, 04);
        XtTextStringFree(t);
    }
    for (i = 0; i <= lines; i++) {
        set_color(dce_area, "Black", NELL);
        if (LineArr)(1).num == 0) {
            XWtStringDraw(dpy, dce_area, XWtWindow(dce_area), gc,
                PartArr)(1).x-20, PartArr)(1).x+20, 10, 10, 0, 360, 04);
            XWtStringDraw(dpy, dce_area, pixmap, gc,
                PartArr)(1).x-20, PartArr)(1).x+20, 10, 10, 0, 360, 04);
            printf(f1, "%s", LineArr)(1).name);
        t = StringCreateLocalized(From);
        XWtStringValues(dpy, gc[0], XtStringList, fromlist, NELL);
        XWtStringDraw(dpy, dce_area, XWtWindow(dce_area), fromlist, t, gc,
            PartArr)(1).x-10, PartArr)(1).x+10, 10, 10, 0, 360, 04);
        XWtStringDraw(dpy, dce_area, pixmap, fromlist, t, gc,
            PartArr)(1).x-10, PartArr)(1).x+10, 10, 10, 0, 360,
        XtTextStringFree(t);
    }
    if (LineArr)(1).type == 0) {
        XWtStringDraw(dpy, dce_area, XWtWindow(dce_area), gc,
PartArray[LiaiArray[i] .pl] .x, PartArray[LiaiArray[i] .pl] .y,
XDrawLine(XtDisplay(dfcarea), pixmap, gc,
PartArray[LiaiArray[i] .pl] .x, PartArray[LiaiArray[i] .pl] .y,
PartArray[LiaiArray[i] .pl] .x, PartArray[LiaiArray[i] .pl] .y);
}
else if (LiaiArray[i].type == -1)
{
    drawdashedline (XtDisplay(dfcarea), XtWindow(dfcarea), gc,
    PartArray[LiaiArray[i] .pl] .x, PartArray[LiaiArray[i] .pl] .y,
    draw_dashed_line (XtDisplay(dfcarea), pixnup, gc,
    x1 = PartArray[LiaiArray[i] .pl] .x;
y1 = PartArray[LiaiArray[i] .pl] .y;
x2 = PartArray[LiaiArray[i] .p2] .x;
y2 = PartArray[LiaiArray[i] .p2] .y;
    XDrwie (XtDisplay(dfcarea), XtWindow(dfcarea), gc,
    x1, y1, x2, y2);
    draw arrow (XtDisplay(dfcarea), XtWindow(dfcarea), gc,
    x1, y1, x2, y2, 5);
    XDrawLine (XtDisplay(dfcarea), pixmap, gc,
    x1, y1, x2, y2);
    draw arrow (XtDisplay(dfcarea), pixmap, gc,
    x1, y1, x2, y2, 5);
    x = x2 - x1;
y = y2 - y1;
t = sqrt(x*x + y*y);
x = (int) (30*x/t);
y = (int) (30*y/t);
    sprintf(str, "%d", LiaiArray[i].type);
    t = XStringCreateLocalized(data);
    XStringDraw(XtDisplay(dfcarea), XtWindow(dfcarea), fontlist, t, gc,
    x1-b, y1-y);
    XStringDrawWidth(fontlist, t, XMALIGNED_RIGHT, 0, NULL);
    XStringDraw(XtDisplay(dfcarea), pixmap, fontlist, t, gc,
    x1-b, y1-y);
    XStringDrawWidth(fontlist, t, XMALIGNED_RIGHT, 0, NULL);
    XStringFree (t);
}
void redraw_graph (Widget widget, XtPointer client_data, XtPointer calldata)
{
    XOPYArea(cbs->event->compose.display, pixmap2, cbs->window, gc2,
    0, 0, width2, height2,
    0, 0);
}
void wprint (char msgbuf[256])
{
    static XmTextPosition wpr_position;
    wpr_position = 0;
    XmTextSetString(textarea, msgbuf);
    wpr_position = wpr_position + strlen(msgbuf);
    XShowValues (textarea, XmTextMaxPosition, wpr_position, NULL);
    XmTextShowPosition (textarea, wpr_position);
}
void wprint2 (char msgbuf[256], int i)
{
    static XmTextPosition wpr_position;
    if (i = 1)
    {wprint (textarea, wpr_position, msgbuf);
    else {
        wpr_position = 0;
        XmTextSetString(textarea, msgbuf);
    }
    wpr_position = wpr_position + strlen(msgbuf);
    XShowValues (textarea, XmTextMaxPosition, wpr_position, NULL);
    XmTextShowPosition (textarea, wpr_position);
}
void wprintf (char msgbuf[256])
{
    static XmTextPosition wpr_position;
    wpr_position = 0;
    XmTextSetString(textarea, msgbuf);
    wpr_position = wpr_position + strlen(msgbuf);
    XShowValues (textarea, XmTextMaxPosition, wpr_position, NULL);
    XmTextShowPosition (textarea, wpr_position);
}
void fprint (char filename[80])
{
    static XmTextPosition wpr_position;
    FILE *f;
    char c[2];
    if ((f = fopen(filename, "r")) == NULL) {
        wpr_position = 0;
    }
while ((scanf(f, 'c', c) != EOF) {
    c[1] = '0';
    if (wp->position == 0)
        XTextSetString(text_area, c);
    else
        XTextInsert(text_area, wp->position, c);
    wp->position = wp->position + strlen(c);
    XTextSetValues(text_area, WFontColorPosition, wp->position, MUL);
    XTextShowPosition (text_area, wp->position);
} fclose(f);
}

void printf(char filename[80])
{
static intFontPosition wp->position;
FILE *f;
char c[2];
char *tempstr;
int go = 1;
if ((f = fopen(filename, "r")) != NULL) {
    wp->position = 0;
    while (go = 1) {
        fscanf(f, 'c', tempstr);
        if (strcmp(tempstr, "analyze") == 0)
            go = 0;
    } while ((fscanf(f, 'c', c) != EOF) {
    c[1] = '0';
    if (wp->position == 0)
        XTextSetString(text_area, c);
    else
        XTextInsert(text_area, wp->position, c);
    wp->position = wp->position + strlen(c);
    XTextSetValues(text_area, WFontColorPosition, wp->position, MUL);
    XTextShowPosition (text_area, wp->position);
} fclose(f);
}

void set_tool(Wid gletet, XPointer client_data, XPointer call_data)
{
    String tool = (String) client_data;
    int i;
    for (i=0; i<4; i++)
        if (strcmp(tool, tools[i]) == 0) {
            XTextSetValues(dfc_p[0][current_tool], XPushButtonEnabled, True, MUL);
            XTextSetValues(dfc_p[0][current_tool], XPushSensitive, True, MUL);
            current_tool = i;
            XTextSetValues(dfc_p[1][current_tool], XPushButtonEnabled, False, MUL);
            XTextSetValues(dfc_p[1][current_tool], XPushSensitive, False, MUL);
        }
}