SANDIA REPORT

SAND97–3141 • UC–705
Unlimited Release
Printed December 1997

Solid Model Design Simplification

Arlo L. Ames, J. Jill Rivera, Annie J. Webb, David M. Hensinger

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under Contract DE-AC04-94AL85000.

Sandia National Laboratories

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

Available to DOE and DOE contractors from
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Prices available from (615) 576-8401, FTS 626-8401

Available to the public from
National Technical Information Service
U.S. Department of Commerce
5285 Port Royal Rd
Springfield, VA 22161

NTIS price codes
Printed copy: A03
Microfiche copy: A01

SAND 97-3141
Unlimited Release
Printed December 1997

Distribution
Category UC-705

Solid Model Design Simplification

Arlo L. Ames
J. Jill Rivera
Advanced Engineering and Manufacturing
Software Development
Intelligent Systems and Robotics Center

Annie J. Webb
Special Projects Department II
Aerospace Systems Development Center

David M. Hensinger
Thermal Sciences
Engineering Sciences Center

Sandia National Laboratories
P. O. Box 5800
Albuquerque, NM 87185-1010

Abstract

This paper documents an investigation of approaches to improving the quality of Pro/Engineer-created solid model data for use by downstream applications. The investigation identified a number of sources of problems caused by deficiencies in Pro/Engineer's geometric engine, and developed prototype software capable of detecting many of these problems and guiding users towards simplified, usable models. The prototype software was tested using Sandia production solid models, and provided significant leverage in attacking the simplification problem.

Acknowledgments

Thanks to David Saylors, Rick Eisler, and Pat Xavier for serving as willing reviewers and Pat Wheeler for help in document preparation.

Contents

Figures

Introduction

This report documents the results and recommendations of the Solid Model Design Simplification LDRD project.

The principal accomplishments of this project are: an understanding of the variety of problems encountered in downstream applications caused by design data, understanding of the sources of these problems, a variety of approaches to attacking the simplification problem, and a prototype system for automatically detecting and semi-automatically eliminating design data problems within and outside of a feature-based framework.

Problem Statement

Technical Problem

In developing automated analysis and manufacturing applications that use solid model descriptions of part geometry, we are confronted with a curious dilemma: designers are rewarded for including as much geometric detail as possible, and that detail causes great difficulty for automated analysis and manufacturing applications. Designs are created to enable parts to be made, not necessarily to enable analysis to be performed. Extra detail at best causes the algorithms to require far more time and space to execute; at worst, the application completely fails. In many cases design data is so overly complex that the cost of manually simplifying the model far exceeds the cost of re-creating the geometry for analysis. In one recent example, over 99 percent of a design was unnecessary for analysis purposes, and ferreting out the important geometry was hopelessly impractical. Less time was required to create the analysis model from scratch than to *load* the design model of the assembly into the CAD system.

Even in cases where the designer has constructed models tailored for analysis, we still encounter problems. "Dirty Geometry" is a term coined to describe models that, for a variety of unexpected (and frequently unknown) reasons, are unusable for the analysis or manufacturing task at hand. A slip of a mouse can create invisible topology that can cause effects ranging from sluggish algorithm performance to complete failure of the application. Failures caused by dirty geometry tend to be frustrating "show-stoppers" -- the application fails to work for unexplained reasons, with no suggestion of how to proceed.

In an ideal world, designers would model geometry in a manner that would easily permit part and assembly features to be trivially suppressed. In practice, it is difficult to predict what portions of the model will be interesting. We thus have the problem of how to traverse and simplify Pro/Engineer [1] design data in an easy, efficient manner.

Technical Approach

Our approach to simplifying Pro/Engineer designs is conceptually quite simple: recognize which elements of a design are unnecessary for analysis, and suppress them. Practically, this requires significant effort. Recognizing which parts of a design are unnecessary is dependent on the analysis being performed, and requires a variety of geometric and non-geometric (e.g. functional) information. Selection of suppressible geometry may even be dependent on physics information (e.g. removing fillets where stress gradients are low). Eliminating superfluous detail is greatly simplified through the possibility of suppressing the form features that spawned the offending geometry and regenerating new geometry, but is not trivial because a feature may create any number of faces, some of which we do not wish to remove.

Technical Issues

Algorithms for recognizing suppressible geometry involve a range of technologies. The simplest algorithm involves searching for features of a certain type (e.g. suppress all holes) or with some specific parameter (e.g. suppress all features with "R1" less than .25). Geometric queries, such as finding all very small faces, or small radius fillets, are more difficult. In addition to suppressing features of a part, it is necessary to suppress parts in an assembly (e.g. find all parts visible from outside, or all parts in the use-control system).

Algorithms for suppressing features to eliminate geometry are not straightforward. A feature may produce some faces we wish to eliminate and others which we wish to keep. A feature we wish to suppress might be a parent to other features, so other features have to be suppressed or redefined to permit the suppression to be performed.

A user interface that enables recognition requests to be easily controlled by both naive and sophisticated analysts is required. The interface must allow composition of complex recognition queries (e.g. find all long, expensive, steel parts) easily.

Quality Metric

The ultimate success of this work is measured in terms of our confidence in our ability to deal with new design data in production downstream applications. At the beginning of the work, we had very low confidence in our ability to perform meaningful quadrilateral surface meshing or hexahedral meshing of $2\frac{1}{2}D$[1] geometries produced by Pro/Engineer.

[1] $2\frac{1}{2}D$ geometry is geometry that can be described by a 2 D cross-section and a sweep. Rotational and linear swept objects are examples of $2\frac{1}{2}D$ geometry.

Representations

The fundamental representations used by modern CAD systems are feature-based and boundary representations. Since this work is intimately tied to operations on these representations, we provide a description of these models. These representations are present in two solid modeling packages in production use at Sandia, Pro/Engineer[1] and ACIS [3].

Boundary Representation

Boundary representation is a fundamental means of representing solid objects. It is one of a family of geometric representations (including, for example, constructive solid geometry and spatial enumerations), and is used in design systems because it is an evaluated representation that conveniently represents precise surface information.

A minimal three-dimensional boundary representation (as present in Pro/Engineer) consists of faces, contours (also called loops) and edges[2]. *Faces* are bounded portions of *surfaces*. A face is bounded by one or more *contours*. A contour is a list of directed *edges*. For a given face, two notions of "interior" are necessary. The first tells which side of the unbounded surface constitutes the outside of the face; a flag is used to define whether the normal of the surface or it's complement is outside. The second notion of interior tells which side of each contour is in the bounded region of the face. The direction of traversal of edges, crossed with the direction of surface normal at a point in question, tells which side of each edge is on the interior of the face. An edge is a bounded portion of a *curve*; two *points* are used to describe the beginning and ending of the interior of the edge.

Feature-Based Modeling

Feature-based design systems, such as Pro/Engineer, model designs in terms of an ordered list of *features* (such as holes, slots, and pockets). The feature representation is a constructive representation; each feature adds to the description of the model by adding or removing material from the object.

Each feature has *defining geometry* and *references*. Defining geometry establishes the shape of the feature; references relate the defining geometry of a feature to the geometry of the part, for placement and sizing. References include *dimensions* and *alignments*. Dimensions establish length and angular positional relationships. Dimensions have *parameters* which establish the numeric values of lengths and angles, and which have names (e.g. d5 = 4.0). Alignments explicitly relate endpoints of edges to curves/surfaces of existing geometry. Dimensions and alignments refer to sketch and part geometry to establish a relationship defining the shape and position of the feature. *Relations* define relationships between the values of parameters. The Pro/Engineer model of relations partitions parameters into sets of dependent and independent parameters. The values of dependent parameters are derived by evaluating relations (e.g. d1 = d2 * d3, where d1 is a dependent parameter). Independent parameters do not depend on any relations; their values are user-supplied.

A feature-based model is *evaluated* to produce a boundary representation. The Pro/Engineer term *"regeneration"* is synonymous with the term *"evaluation"*. In a typical modification, one or more parameters are selected and changed, either through relations or

---

[2] Boundary representations can be much richer than Pro/Engineer's. ACIS represents bodies, lumps, shells, faces, loops, coedges, edges and vertices.

through selecting the parameters of interesting dimensions of affected features. Pro/Engineer reevaluates the feature list from the first changed feature through the end of the list to produce a modified version of the part. If any geometric relationship cannot be evaluated under the modified set of parameters, the regeneration fails.

From a data content standpoint, feature-based models are a richer source of information than evaluated models. Feature-based models contain construction geometry (e.g. datums) and a wealth of relationship information regarding how the part can be modified. Evaluated models can be inferred from feature-based models, but the feature-based models cannot be uniquely determined from evaluated models.

Parts and Assemblies

A design may represent a *part* or an *assembly*. A part is the smallest chunk of mass that is individually considered in mechanical design, and can be of arbitrary geometric complexity. Assemblies are collections of references to parts and other assemblies; each reference has an associated transformation for placing the referenced part or assembly within the coordinate frame of the containing assembly. An assembly may contain more than one instance of any given part. Assemblies describe groups of parts that work together to perform a mechanical function.

There are both feature-based and evaluated versions of assembly representations. The feature-based version defines a set of features that place parts in the assembly using positional relationships between them (e.g. contact and alignment relationships). The evaluated version of an assembly representation is a list of part/assembly references, each having an associated transformation matrix to position the instance within the assembly's coordinate frame.

Range of Downstream Applications

The downstream applications we considered span a range of geometric complexity, which plays a critical role in the sensitivity of the application to the details in the input. The following categorization of algorithms will serve to define sensitivity to input problems: topological query, geometric query, geometry modification, and geometric interaction.

Topological Query

Topological query algorithms traverse the topology of the model, querying the information found. An example algorithm of this type counts the number of faces in the model, or searches for an attribute attached to an edge. Query algorithms are very insensitive to problems in input data, as they are merely traversing pointers. They expect the topology to be well-defined.

Geometric Query

Geometric query algorithms query the geometric definitions of points, curves, and surfaces to determine shape information. Examples might discretize an edge, compute surface normals, or find surface area. Besides requiring well-defined topology, algorithms of this kind expect a certain "well-formedness" in the surface equations; that is, that each surface and curve is continuous and differentiable everywhere on its interior.

Geometry Modification

Geometry modification algorithms modify the geometry and topology of the solid. Boolean operations and sweep algorithms are examples from this class. Such algorithms expect not only well-formed topology and geometry, but that adjacent entities match each other well. Edges must lie on faces and points must lie on edges.

Geometric Interaction

Geometric interaction algorithms operate on the shape of geometric interactions between non-connected objects. Example algorithms compute adjacency between objects in an assembly, or the shape of the space between two holes in an object. Algorithms of this kind require topological and geometric well-formedness, and may or may not require close entity matches as in geometry modification. They require well-formedness in the interactions between objects that are being compared; if two objects are supposed to touch, then the faces that touch should be close to each other with tangent surface representations.

Identifying Problem Areas in Parts

Historical experience in attempting to use design data for downstream analysis indicated that many of the problem areas were invisible to the people defining and attempting to use the data. Our intent in this project was to provide algorithms that would search the design information, using a variety of metrics, to locate problem areas so they could be dealt with.

In order to be able to develop algorithms to search for suppressible geometry, it is necessary to identify sources of difficulty. We identified problems by collecting examples of processing failures in CUBIT, Archimedes, Cosmos/M, and other applications. The following areas have been encountered in attempts to use Pro/Engineer design data for analysis purposes: numerical inaccuracy, degenerate surfaces, unnecessary topology, unanticipated topologies, extraneous detail, and difficult interactions in the object interior. Similar problem areas are expected, to varying degrees, with any design system.

Diagnosis of problems can be very difficult. Problems are typically manifested by downstream applications breaking, abruptly and without any clue as to how to proceed. In instances where the application has been developed in-house, time spent with a debugger can produce a reasonable diagnosis; in the case of commercial software, diagnosis requires either involving the vendor, or dissecting the model to locate the problem area.

This list can be viewed as both a list of areas to search for in simplification, and as a list of items to be wary of in developing robust downstream applications. For our purpose, the former view is taken.

Numerical Inaccuracy

All modern CAD systems approximate a variety of numerical parameters to some degree. The fundamental architecture of computers involves finite precision (typically 32 or 64 bit floating point). Finite precision arithmetic guarantees truncation. Intersection curves are frequently heavily simplified. A NURBS-NURBS[3] intersection is theoretically a degree 54 polynomial which most CAD systems approximate with a cubic spline curve (degree 3).

Pro/Engineer has a parameter in the Setup menu of Sketch, Part, and Assembly modes that defines what accuracy should be used in modeling. Adjusting that parameter can be difficult -- it is possible that a part will regenerate correctly at a low accuracy, then fail when higher accuracy is requested, because of difficulty in evaluating geometric relationships precisely. This parameter defaults to low accuracy which improves modeler speed, sacrificing accuracy.

Inaccuracies are typically discovered as gaps between points and the curves they are supposed to limit, between curves and the surfaces they limit, and as gaps and overlaps between surfaces of parts that are expected to be adjacent. In production Pro/Engineer data, gaps as large as 10^{-2} have been observed between surfaces and the curves that limit them, on objects of roughly unit size. Gaps at the highest accuracy setting tend to be roughly 10^{-4}. Surface equations tend to be much more precise than edge equations; the surfaces are extrusions and sweeps of curves that have been explicitly drawn and dimensioned, while many of the curves are intersections of surfaces, which are approximated for speed and representational convenience.

Accuracy tends to have largest affect on geometric intersection-based calculations, such as boolean operations, where algorithms attempt to resolve model topology to the shape of the

[3] NURBS is an acronym for nonuniform rational B-spline surface.

geometric intersections. Query-based functions, such as surface-area calculations, are entirely unaffected, as the positions of each entity are treated as independent parameters. Finite element meshers are typically unaffected by inaccuracies, but decompositions applied to geometry in preparation for meshing can be very sensitive to gaps.

Degenerate Surfaces

Degenerate surfaces are surfaces whose equations are defined in a manner that permits more than one point in parametric space to occupy a single point in geometric space. Examples are toroidal surfaces with zero major radius (a sphere), and NURBS surfaces with a set of knots at a single point in space, such as along a single edge.

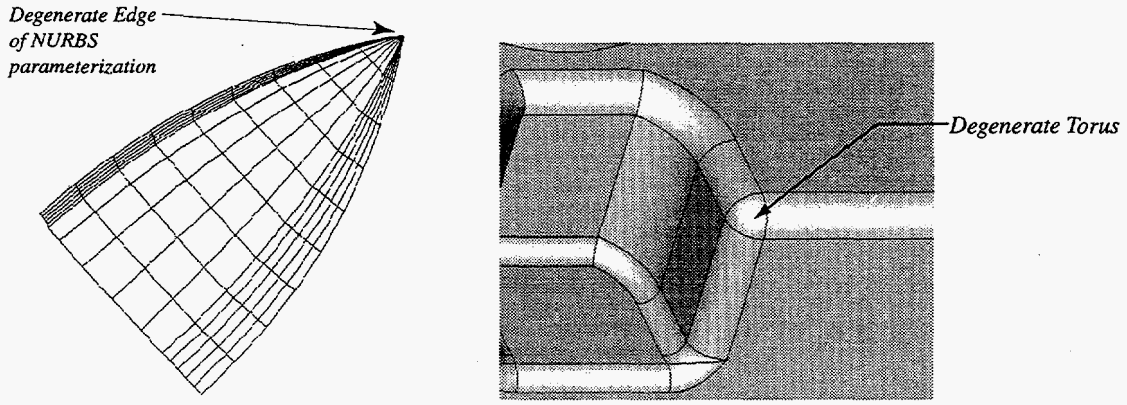


Figure 1: Degenerate NURBS surface and a torus with zero major radius.

Pro/Engineer has proven to generate a variety of degenerate surfaces. Filleting algorithms frequently generate degenerate tori, and blending functions can produce surfaces with zero-length edges in their boundaries if blends are connecting boundaries with differing numbers of edges. Blending from a curve to a point always generates a degenerate surface.

Surface degeneracies produce erratic behavior in algorithms that expect uniformly parameterized surface equations. Surface normal computations are extremely erratic at points of degeneracy, such as cusps. Algorithm behavior can vary from producing subtle numeric noise (in the case of querying algorithms, such as surface area), to algorithm failure (e.g. the NO_SURF_PERP error in ACIS causing a core dump).

Degenerate curves are also theoretically possible; they have not yet been observed from any CAD system, perhaps because of the nature of construction algorithms. Intersections of degenerate surfaces could produce them, but the affect of the degenerate surface is likely much more pronounced than the affect of a degenerate curve.

One known, expected, degenerate curve exists at the apex of rotationally swept curves, such as the apex of a cone.

Unnecessary Topology

In defining topology to represent a geometric model, some topology occurs at arbitrarily chosen locations to satisfy modeler limitations. For example, every edge might be required to have a start and end vertex. For a circular edge, a minimal topology would not require start and end vertices, yet current CAD systems always insert the vertices so that all edges

have a consistent topology. Another example of unnecessary topology is a seam along a cylindrical face; the seam is not necessary, but it connects the boundary of the face into a single loop.

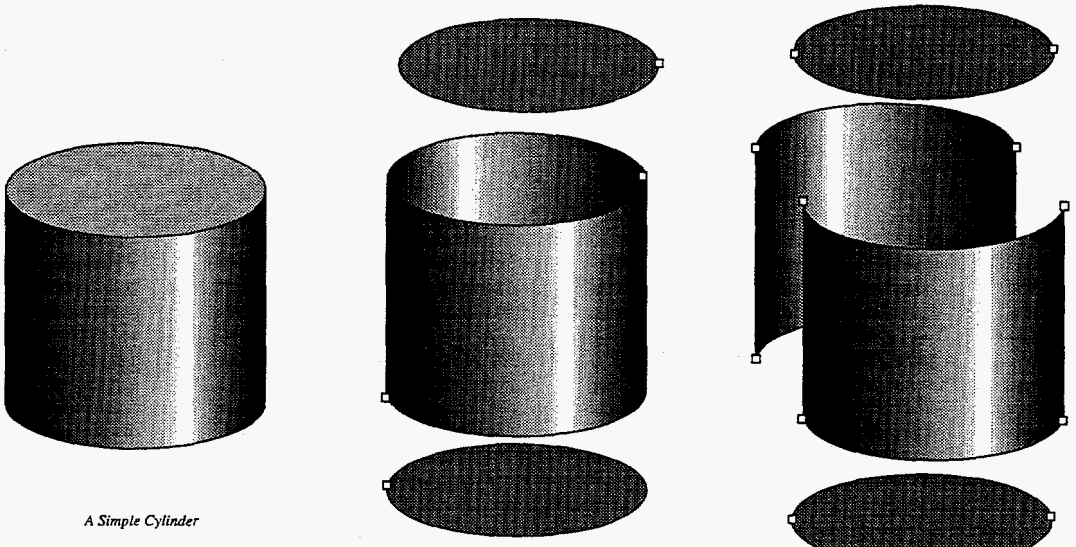


Figure 2: Minimal and non-minimal topologies.

All modelers produce some unnecessary topology. Figure 2 illustrates minimal and non-minimal topology for a simple cylinder.

For most applications, arbitrary topology has little or no effect besides speed degradation. Speed degradation occurs when extra unnecessary topology is introduced to algorithms having super-linear (O(n log n) or worse, where n is a count of topological entities, such as faces) performance; the extra entities slow processing down. Faceting and finite element meshing algorithms have additional difficulties, because they produce geometry that is required to match the topology of the given model. Unnecessary topology can produce very short edges in the model, which force the faceting algorithm to produce facets with very short edges, either resulting in waste of facet budget, or worse, in poorly shaped finite elements. Extreme cases of short edges can cause complete failure of meshing algorithms.

It is possible to eliminate the effects of unnecessary topology automatically, either by filtering it out or by developing algorithms that account for its presence. Current downstream applications frequently lack such a capability.

Unnecessary topology presents difficulties for a simplifier, though, as it is necessary to distinguish between short edges that occur because of the presence of some feature and short edges that occur due to unnecessary topology created by the geometry engine.

Unanticipated Topologies

This problem case involves topologies that are more complex than the application developer expected. In an example, a model of a silo was presented to a commercial tetrahedral mesher. The silo was essentially a dome-topped cylinder with windows cut through the walls. The mesher performed well until a simplification occurred that removed all the

windows. At that point, the mesher failed. When the windows were removed, the object's faces were disconnected into two separate shells; the mesher filled the exterior shell to completion, then tried to fill the "interior" of the interior shell, effectively attempting to mesh the universe. Failure occurred when the mesher ran out of memory.

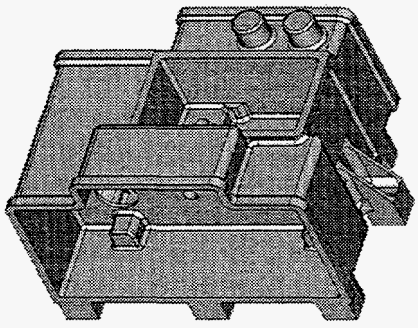
In another example, a standard approach to representing the presence of the apex of a cone (a degeneracy) as part of a face boundary is to use a zero-length edge having no curve representation. Files containing such a representation caused a meshing package to fail when attempting to discretize the edge. The developers of the mesher assumed that every edge could be discretized into multiple, distinct points.

Dealing with unnecessary topology is certainly the job of the application developer. The "simplification" tool can, however, provide warnings about atypical topologies (e.g. disconnected shells, disconnected faces, nonmanifold topologies) so that the user has a suggestion as to how to proceed until downstream applications can be corrected.

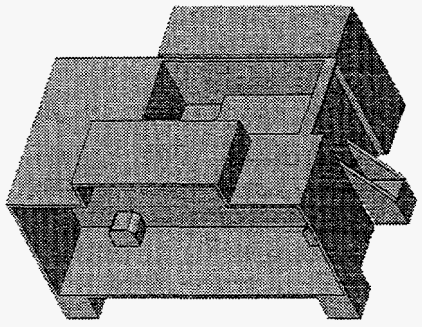
Excessive Detail

One of the fundamental reasons for producing design data is to produce parts. The design data is a statement of every detail that is expected to be in the final parts, so it is used in defining contractual agreements.

Including details involves a price in the size and complexity of the data. In examining typical design data, we find that the majority of the geometry present is in the form of details. Comparing a simple cube to a filleted version of the same, the first has only 6 planar faces, while the filleted object has 26 faces. All of the intersection curves in the simpler object are lines, while the filleted object has arcs as well. If variable radius fillets were used, the surface complexity increases to NURBS (with significant increase in representational cost[4]).



Features: 202
B-Rep: 4.45 MB
Facets: 8 MB



Features: 47
B-Rep: 507 KB
Facets: 214 KB

Figure 3: Detailed and simplified representations of a part, with comparative sizes.

[4] Higher order surfaces, such as NURBS, incur significant computational cost because of the amount of data involved in their definition, as well as the algorithms involved in their evaluation. While a quadric surface involves a small amount of parameters (6 numbers define a plane), a NURBS is defined by a grid of points, which can have arbitrary size. The algorithms to evaluate NURBS are frequently iterative, while quadrics can be evaluated in closed form.

In Figure 3, we see a detailed part, and a simplified version of the same. The simplification is typical of analysis needs, but does not represent any specific analysis. Various representations of these objects show differences in data size; the largest variation shown is for a faceted representation, where more than an order of magnitude difference in data size is observed.

Details can cause great difficulty in downstream applications (e.g. finite element analysis). In the simplest case, the downstream application takes longer to run. Since geometic applications have worse than linear performance ($O(n^2)$ and $O(n^3)$ are common), detailed models can take far too long to run. Designers concerned about long run times can speed up their process considerably by judicious (temporary) simplification.

Frequently details can be ignored by downstream applications. Many applications facet or mesh the object, ignoring much of the local surface detail. Depending on the resolution of the analysis model being constructed, the details often have only negligible affect on the solution, and can cause incredible cost in runtime.

Controlling details can have a significant effect on the speed of solid modeling operations during designs. Feature addition algorithms are implemented in terms of Boolean operations or Euler operators[5], both of which have superlinear runtimes (Booleans can exhibit $O(n^3 \log n)$ behavior).

Interactions in the Object Interior

Possibly the most difficult problem encountered in design data involves interactions in the interior of objects. We are concerned here not with the shape of the exterior surface, but the shape of the space between features that are close to each other (where the wall thickness is small, or changes shape in a complex way). While many algorithms operate on the object's boundary, there are a few that perform operations on the shape of the interior space. One such application is hexahedral finite element meshing, a current research topic.

Interactions occur in the object's interior for a variety of reasons. Some are for functional reasons, but many are unexpected. Designers are typically working with $2\frac{1}{2}$D features that are being used to create functional surfaces; non-$2\frac{1}{2}$D interactions can occur in regions that are not of functional concern.

Most simple downstream applications are unaffected by the shape of the interior of the object. For applications such as hexahedral meshing, such interactions can, however, cause complete algorithm failure, or at least require inordinate time to deal with.

Identifying Problem Areas in Assemblies

In addition to the problem areas described above, assembly models present an additional set of difficulties in the area of part/part interaction, including positioning errors and assembly definition errors.

---

[5] Euler operators are operations that preserve the Euler characteristic of an object -- that is, they maintain a required consistency in the number of topological entities required to represent a valid solid object. Such topologically localized operators are not guaranteed to preserve geometric integrity of an object.

Positioning Errors

Recall that parts and assemblies are positioned within their containing assemblies using transformation matrices. When parts occur in subassemblies of assemblies, matrices are concatenated to arrive at a matrix that will place parts in the global coordinate frame. Matrix concatenation produces roundoff, resulting in positioning errors in placing parts.

When mixed-unit assemblies are defined, the units conversion is embedded in the transformation matrix as a scale factor. If the transformation is applied to the geometric definitions in a boundary representation (scaling the object up or down), the scaling also is applied to gaps in the geometry (opening or closing them). The same effects as numerical inaccuracy occur here, but in a nonuniform way; the part processes well by itself, but fails in the context of an assembly.

Translations in matrices can produce yet another difficulty. Most modern CAD systems use floating point representation for numbers. Floating point numbers degrade in accuracy as they grow increasingly large. The result of applying a translation to an object that moves it away from the origin is that the parameters in its representation are represented with less accuracy. This reduction in accuracy produces closure errors that can grow larger than the size of the part. In such a case, the object's geometric definition is useless.

All of these positioning errors are subtle. Wherever possible, parts should be defined in a consistent set of units, and operated on in the original coordinate system as long as possible.

Assembly Definition Errors

Relative to parts, assembly information tends to be significantly less accurate. A common practice is to define a rough model of the assembly in order to divide up the space, then allow component designers to develop their designs within the space limitations they are given. Over time, they produce increasingly detailed part definition, perhaps changing the shape of their space (by trading space with others) until the detailed definition is complete. Since attention is focused on the details, the assembly may not be updated (or even loaded, since it is so complex relative to parts). Thus, natural work habits can result in an assembly definition that really doesn't match the parts in the assembly. Overlaps and gaps are very common in such a scenario.

Pro/Engineer really provides little in the way of tools to guarantee that the assembly is as intended. Parts and assemblies must be carefully scrutinized after regeneration to ensure that design intent hasn't been compromised.

Approaches to Part Simplification

There are a variety of approaches to removing portions of part geometry to achieve simpler models, including feature suppression, boundary representation editing, and model re-creation. These approaches are presented in order from maximal to minimal use of the original design geometry.

Feature Suppression

Modern CAD systems tend to be based on a constructive model, such as a feature-based representation. The explicit geometry produced by such models is generated by evaluating the generative model, and reevaluation (regeneration) is frequently performed to achieve some editing need.

Since feature inclusion/suppression is a fundamental part of feature-based CAD interfaces, suppression of this kind is as available as the CAD system itself. Suppression activities operate in the originating CAD environment on the feature-based representation that is evaluated to produce the geometry, so every downstream application has access to models that have been changed in this fashion.

Producing simplified models in such representations appears to be very straightforward: merely suppress certain details and regenerate. When such a scheme works, it is very easy. Suppression-based simplification becomes difficult in cases of inconvenient parent/child and geometric relationships between features.

Parent/Child Relationships

As mentioned above, feature definitions require definition of geometry and references. References relate shape and position of the feature's geometry to other geometry of the part. These references mean that any given feature can depend on the geometric definition of any previously defined feature. This dependency is called a *parent/child relationship.*

Under ordinary feature suppression, Pro/Engineer skips the evaluation of the geometry of any suppressed features. Since some geometry has not been evaluated, any feature that depends on that geometry is lacking definition and cannot itself be evaluated. In such a case, Pro/Engineer presents a menu permitting child features to be *rerouted* (have new definitions provided), or suppressed as well.

Suppressing child features can provide the desired simplification, but this is a binary decision. If any geometry of the child features is desired in the final model, this route cannot be taken.

Rerouting involves defining new references for the dimensions and alignments of a feature. Manually rerouting a feature can be very difficult, as the original definition of the feature may be forgotten, and there may be no convenient geometry to use in the rerouting.

A different approach to feature suppression in Pro/Engineer is the creation of a *simplified rep*. The simplified rep finesses the parent/child relationship problem to a degree by providing a mechanism for child features to reference geometry in a *suppressed* feature. The effect of a simplified rep is that a complete model of the part is generated, and the simplified rep is permitted to reference geometry in that representation to evaluate features in the simplified rep.

Simplified reps would appear to finesse the problem of simplifying by feature suppression. It is not a complete solution, because of difficulties related to feature usage (see next section). Downstream applications must be aware of simplified reps in order to be able to use them. Pro/Develop applications cannot, at this time, create or access definitions of simplified reps. The geometry of a simplified rep can be accessed by a Pro/Develop application if the simplified rep is the active representation of the object.

Inconvenient Geometric Relationships

More difficult than parent/child relationships are inconvenient geometric relationships. Any given part geometry can be created in any number of ways. Any choice implies certain kinds of editability, and precludes others.

A very simple example is the creation of a hole pattern versus creation of individual holes. If the hole pattern is created, it can be edited as a pattern. If individual holes are created, they can be suppressed individually, but cannot be moved as a pattern.

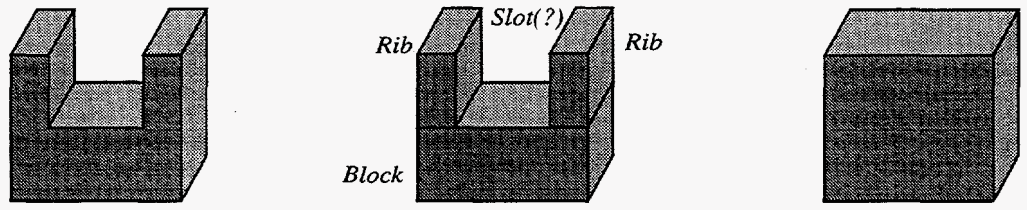


Figure 4: A block with ribs cannot be conveniently simplified to remove the slot.

In another example (Figure 4), two ribs are added to a block. The region between the ribs can be considered a "slot." This slot cannot be suppressed, however, to form a larger block, because it was not constructed by removing material to form the slot, but by adding material around the space of the slot. The slot can only be removed by constructing an object that represents the slot's geometry, then performing material addition.

More subtle geometric interactions can occur, preventing the part from geometrically resolving to a closed solid. Such difficulties represent a fundamental limitation of suppression-based approaches.

Editing the Boundary Representation

Directly editing the boundary representation is another approach to achieving simplification. Every known mechanism for editing boundary representation models can be applied in simplifying the geometry directly, including Boolean operations, imprinting, slicing, and local topology editing. A common thread throughout is that any new geometry required (e.g. a primitive to unite with the object to remove a hole) must be created and aligned to the object being modified.

Some CAD systems (notable Pro/Engineer) do not permit the boundary representation to be directly edited. In such a case, boundary editing can only be performed by exporting to another geometry engine (e.g. ACIS). Direct editing of the boundary representation overcomes difficulties inherent in feature suppression, as the representation that defines the limitations is not present.

While directly editing a boundary representation eliminates the possibility of parent/child interactions and bad feature choices becoming a limiting factor, this approach is not without difficulty. In the feature suppression approach, a well-formed set of semantics are available -- every feature can be present or absent. In direct boundary-representation editing, no such semantics are present. You cannot simply "remove a hole" because no "holes" are present. The "holes" are concave cylindrical faces. The model has to be treated as unadorned solid geometry (faces, edges, and vertices) unless feature information is available from another source or feature recognition (e.g. [4]) is used. Additionally, construction geometry is frequently present in the feature-based representation that can ease the simplification process.

Constructing New Models

Despite our best efforts to modify design models, it may be most expeditious to build new models from scratch with the application's needs in mind. Two approaches are: creating only the topology (referencing the geometry), or creating the entire model.

Creating Only Topology

Frequently, the geometric information (surfaces, curves, and points) is sufficiently well formed to be usable by the application. Problems might be principally topological, involving too many details, unnecessary topology, and the like. A reasonable approach would be to construct a new topological model that refers to the original geometry, but provides a different, well-formed topology.

The TrueGrid [5] product uses this approach: meshable topology is imposed on product geometry, ignoring the original solid topology.

Complete Reconstruction

Obviously, if the correct model is not available, it is always possible to build the requisite model from scratch. If the CAD system is carefully driven, it is possible that building simple models directly can produce usable results.

Such an approach is not directly connected to the "design" geometry, so changes in the design do not propagate easily if analysis models have been built from scratch. Additionally, errors may occur if the original model is not precisely followed in constructing the analysis model. The cost of re-creating models can be high. However, in cases where very small models are desired and only very large, complex models exist, this approach may be the most practical.

Redefining the Feature Evaluator

In a related project, it has been necessary to produce geometry with a specific kind of meshable topology *independent of the geometric ramifications*. A custom feature evaluator was developed that guaranteed the necessary topology even if it entailed significant geometric modification (i.e. telling geometric falsehoods to create a usable topology). Development of such an approach is impractical with Pro/Engineer due to the closed nature of its interface, but is possible in cases where a push-button solution is desired and the domain of product definition is sufficiently simple. The approach has been very successful in a real-world production setting.

Design Rules

Moving further upstream, it is possible to develop a set of design rules that define constraints on how parts are designed (in terms of possible shape or in terms of feature

usage) to ensure the usability of design definition throughout the range of applicable applications.

Designers tend to display a variety of responses when asked to conform to a set of design rules. Rules limit creativity, and add burdens to the difficulty of creating a useful design. Design rules tend to evolve over time as applications become increasingly sophisticated, requiring periodic retraining. Additionally, design requirements for different disciplines can conflict, adding to the designer's burden. Designers must see an immediate benefit to design rules, and their impact on practices must be balanced with their value, or the rules will not be adopted.

Approaches to Assembly Simplification

Part Elimination

Suppressing parts in a feature-based model can involve the same difficulties as present in the part-editing paradigm. Parent/Child relationships can make suppression of parts a serious problem. Simplified rep, with its caveats, can improve this situation to a degree.

Removing part references from the evaluated representation of the assembly is very simple. In that representation, elements of the assembly are simply references to parts and assemblies, accompanied by transformation matrices. Removal of a part from the assembly in this representation is as simple as removing a reference. This approach is not available to applications within the Pro/Engineer framework, but works well if data has been exported.

Providing Matching Parts

Inaccuracies in evaluation can produce gaps and overlaps between parts in an assembly. Gaps in "contact" regions can occur for valid design reasons as well; for example, the relationship between parts might be a running or a press fit, for which the geometry of the real parts will involve such gaps and overlaps. For a number of assembly-level applications (e.g. finite element meshing, assembly planning), it is desirable to provide precisely matching geometry in contact regions.

We have used two approaches: modifying the geometry and explicitly representing the relationship as a feature.

Modifying the geometry involves part editing techniques similar to those outlined above. Parameters may be changed and geometry moved to close gaps. A function to find and measure near misses between parts is useful in order to permit the editing to occur.

An alternative approach that was implemented in the Archimedes assembly planner is to model the relationship as a feature. A press fit relationship can be explicitly modeled in the application ([6]) or in the CAD system and translated to the application ([7]). Either approach yields data that can support reasoning about the relationship without relying on the contact.

Combining Parts

Excessive fidelity in an assembly model results in there being more parts present than the analyst wishes to resolve. In such cases, it is necessary to treat multiple parts as a single object. The multiple parts can be combined by boolean union, or replaced by newly-modeled geometry representing the lumped mass of the objects being combined. Combining the objects has the advantage of requiring no geometry creation, but can produce rather complex models, due to interactions between the parts (e.g. gaps, tangencies, etc.). The boolean union can tend to preserve significant geometric detail, while modeling from scratch produces only the details desired.

In complex assemblies, it is sometimes appropriate to perform radical simplification, perhaps replacing objects with simple lumped-mass models. Solid modeling engines are very adept at producing mass and moment-of-inertia calculations for such simplifications, so this approach to simplification is not further treated here.

Modeling Missing Geometry

While models of parts are generally too detailed, insufficiently detailed models also occur. The missing information can be in the form of missing parts or missing details.

At Sandia, a common example of this occurs in modeling encapsulants and foams that protect other parts. Since Pro/Engineer lacks functionality for filling voids between parts, creating models of such parts would involve defining features representing the void space around the parts that have already been defined and maintaining both part and void space geometries over design changes. This represents significant work, and the shapes of these objects can be inferred from the objects around them, so the foam/encapsulant is not modeled.

Frequently, analysis work requires that the encapsulants be modeled in order to achieve a complete model. Boolean operations can produce models of many missing objects in a direct way. Manual construction is also possible.

Pro/Engineer Based Simplification

Our investigation of simplification by feature suppression within the Pro/Engineer framework focused on the requirements of radar cross-section analysis. The algorithms used in our radar cross-section analysis require well-formed triangular facets. Ordinary faceting algorithms don't provide adequate control of the shape of the facets, so the geometry is surface meshed with CUBIT to produce quadrilaterals, which are subdivided to produce the requisite geometry.

The application was chosen because it involves translation of the geometry to an application outside of the Pro/Engineer framework, and the application was experiencing significant difficulty with the data being produced by Pro/Engineer.

The work focused on simplification within Pro/Engineer using feature suppression, as that approach maximizes the number of downstream applications that can benefit from the simplification activity. Pro/Develop prevents direct editing of the boundary representation in Pro/Engineer, so a feature-based approach is the only approach feasible. We identified applications that were unable to process the Pro/Engineer "simplified rep," so we limited our inquiry to feature suppression.

The algorithms developed in this work were not intended to provide complete coverage; rather, to provide a necessary set of tools for a particular kind of analysis, and to measure the value of such tools in a production environment.

Degenerate Surface Detection

In our attempts to use Pro/Engineer-generated data downstream, degenerate surface detection received a high priority, as degenerate surfaces caused some of the most difficult problems. Crashes were frequent in CUBIT, occurring whenever the surface equations were queried in the neighborhood of a degeneracy. The ACIS modeler complained of inability to compute surface normals at points of degeneracy, causing an inability to find closest points on surface, thus preventing meshing.

Degenerate surface recognition algorithms address degeneracies in the following kinds of surfaces: spline face, conic face, NURBS, ruled surface and torus. Spline and NURBS surfaces are checked to see if defining knots were too close to each other (within a tolerance). The parameters of conic face and torus are checked to find degenerate angles and radii.

The surfaces checked by our algorithms have been shown to exhibit degeneracy rather frequently, due to peculiarities of feature construction algorithms in Pro/Engineer (e.g. blend surface and fillet). Most of the remaining surface types are simple extensions of the algorithms already developed. The "foreign surface" in Pro/Engineer is difficult to check for degeneracy, as its definition is provided by an application programmer, and complete access to the surface definition is not guaranteed.

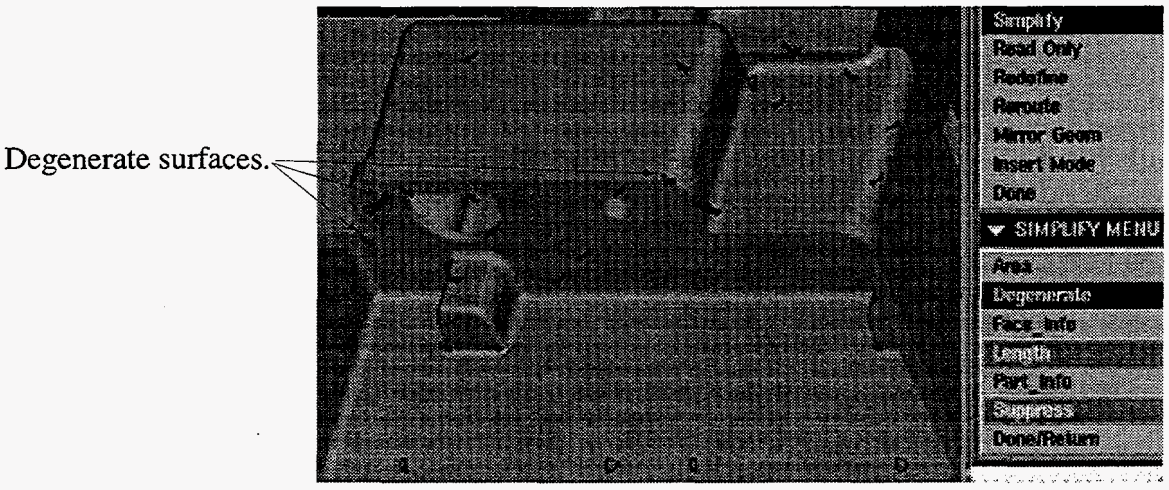


Figure 5: Automatic detection of degenerate surfaces.

Detection of Details

Algorithms for detecting details and redundant topology were developed. The faces of part models are traversed, and measured in terms of surface area; edges are likewise measured in terms of length. Filters for small and large-valued objects are provided, to enable visualization of both important and unimportant geometry.

Algorithms for traversing the assembly to locate details were developed. The assembly can be searched to identify and highlight piece parts of specific materials, to satisfy a need for removing materials that were unimportant to analysis.

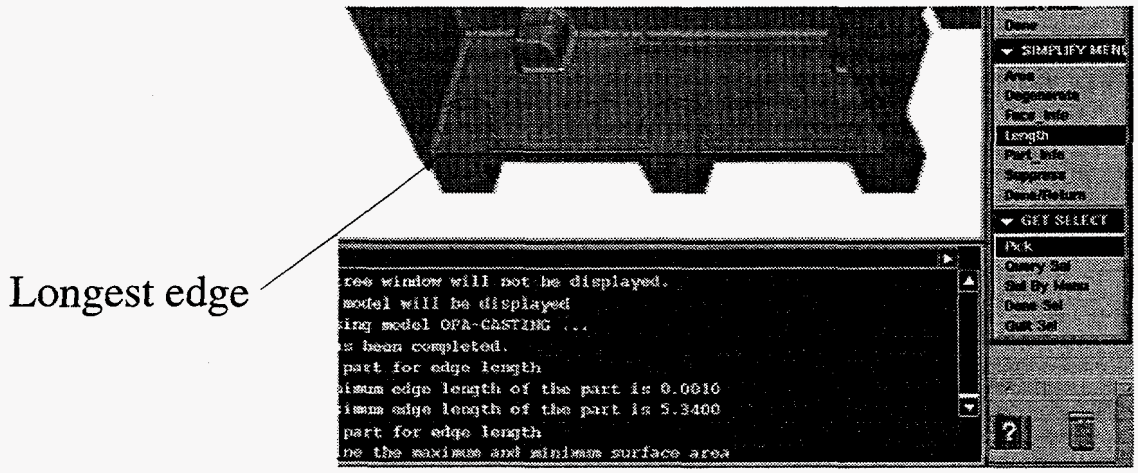


Figure 6: Size-based geometric search.

Simplification Algorithms

In the domain of part simplification, we developed an algorithm for traversing feature definitions to determine the ramifications of parent/child relationships, automatically suppressing the feature if possible. In order to attack parent/child relationships as much as possible, we worked to develop an algorithm for automatic feature rerouting (see below).

In the domain of assembly simplification, we responded to a request from Design Services to provide a means for suppressing/resuming all features in named layers of parts and subassemblies in an assembly. The algorithm traverses the assembly data structure, traverses layer information, and constructs a list of features to suppress. The part suppression algorithm mentioned above is exercised for the features to be suppressed in each part. The algorithm is complete; testing is underway.

Another assembly-level operation merges piece parts into a single part by simply selecting them. Merging is not equivalent to boolean union, as all of the original faces of the merged parts are left intact. Boolean operations can, however, be performed outside of the Pro/Engineer framework, if such part combination is required. For certain applications it is sufficient to consider the combined parts as a single object.

Inquiry and Summarizing

An almost surprising result of our work in providing algorithms for automatic search was the requirement for summary information, and the ability to interactively apply evaluation functions to anything the user selected. This summary information provides a rapid way to assess how much simplification might be required before any significant time or effort is invested.

Summary information for parts includes the number of surface finishes on the part, and a table listing the number of faces having each type of surface (planes, cylinders, cones, tori, surfaces of revolution, tabular cylinders, ruled surfaces, coons patches, fillet surfaces, spline surfaces, NURBS, cylindrical splines and foreign surfaces). Summary information available for a selected face includes surface type, number of bounding edges, and counts of each type of edge(line, arc, and spline edges).

Application-Driven Simplification

After developing a number of recognition algorithms within the Pro/Engineer framework, it became obvious that some applications are effective at determining which kinds of geometry and topology they are not prepared to process, and that the algorithms for detection of inappropriate geometry are best developed in the application. Thus, we developed an algorithm for performing simplification within the Pro/Engineer simplification framework, but driven by the external application.

Requisite for application-driven simplification is a means of stating which faces and edges are not desired, in a manner that survives starting and stopping of Pro/Engineer. Pro/Engineer defines three mechanisms (pointers, ID's and user attributes) that can be used to refer to faces and edges. Both ID's and user attributes survive regeneration and restarting of the modeler, so are appropriate. User attributes increase data size, so ID's are the labeling method of choice.

When geometry is transferred to the application, it is necessary to label faces and edges using ID's. In the case of translating to another geometric model (ACIS in our case), attributes can be used to contain the ID information.

When the external application encounters inappropriate faces or edges, it needs to build a list of the offending entities. This list is then processed by a Pro/Develop application to locate the features that contain the offending geometry. The features are searched for children and suppressed if appropriate/possible. After regeneration, the data can be transferred to the application again for another attempt at processing.

Feature Rerouting

Work was begun on an algorithm for rerouting troublesome features. The work was halted due to a lack of fundamental functionality in Pro/Develop. The functionality was promised in Pro/Toolkit, but was unavailable in time to permit inclusion before project funding terminated. The algorithm was designed and some coding begun; an outline of the necessary algorithm follows.

Given that we desire to reroute a child feature in the absence of the parent, it is necessary to change external references from objects in the feature being removed to other references in the geometry. These references are of two forms: dimensions and alignments. The referenced geometry in dimensions and alignments are any of the kinds of geometry supported in Pro/Engineer: datums (points, axes, planes, curves and surfaces), and part boundaries (faces, edges, and points). Features can be rerouted by finding or creating geometry that is equivalent to the referent that is being eliminated, then changing the reference and regenerating the feature.

Proof of geometric equivalence is rather straightforward, merely involving checking point, edge, or surface definitions for coincidence. It is possible for equivalent curves and surfaces to have different definitions, so algorithms must be sensitive to this possibility. Most solid modelers have such algorithms embedded in them to permit merging of equivalent geometry during boolean operations. Access to such algorithms is typically missing from programmer's interfaces, however.

For rerouting purposes, the definition of equivalence must be expanded to permit equivalence relative to the dimension or alignment being rerouted. In the case of a linear dimension, for example, it is possible for a point and a line to be geometrically equivalent, if the point lies on the line and the line is perpendicular to the direction of the dimension. The referents must possess equivalent position in their projection onto the dimension.

If the search for geometrically equivalent geometry fails, two choices are possible for dimension references: constructing equivalent geometry, or searching for non-equivalent geometry that can be used in conjunction with a change in dimension value. If equivalent geometry is constructed, a datum is adequate. Construction of arbitrary datums is greatly simplified if the datum is referenced to other datums, possibly to the default coordinate system. Algorithms for creating arbitrarily oriented datums in Pro/Engineer have been developed.

For alignment references, geometrically equivalent references are required. They must either be located in previously existing geometry or constructed.

The algorithms for rerouting features were designed and partially constructed. Incomplete programmer interface to Pro/Develop prevented their completion. The newly released Pro/Toolkit promises the possibility of performing feature rerouting automatically, but is currently unproven in our environment.

ACIS Based Simplification

In attempting to create hexahedral finite element meshes of complex $2\frac{1}{2}$D assemblies, we have encountered a variety of assembly-related difficulties. A suite of tools was developed to provide a means of closing gaps, modeling foam, performing imprinting, and otherwise modifying the geometry of these assemblies to produce meshable geometry. The tools were developed as small, simple, standalone modules that can be executed from the UNIX command line. Each file processed by the tools is assumed to contain the definition of one

or more solid objects, which are operated on and written to an output file. This approach was taken following programming frustration in working with large packages such as Pro/Engineer and CUBIT, where significant time is spent compiling and linking, and dealing with the complexity and limitations of the tool. ACIS was used as the geometry engine due to ease of programming the tools. It is interesting to note that these tools were developed by an analyst to meet his needs, rather than being developed by a geometry or software development specialist.

Assembly Level Tools

Certain of these tools are limited to $2\frac{1}{2}$D objects. Equivalent functionality will be required in three dimensions.

This set of tools arose out of a need to perform geometric manipulations on a large number of ACIS bodies. These tools take as arguments the names of ACIS '.sat' files and in some cases an axis designation and or numerical constant. The '.sat' file can contain any number of ACIS bodies. Each of the geometry tools operates on every body in every target file. Operations can be performed on a single body by isolating that body in a '.sat' file.

The output file name may be the same as any of the input file names. Very little error checking is done within these tools. Some of the tools (`tool_maker`, `cube_maker`, `imprinter`, `leveler`) are designed to operate on a file containing $2\frac{1}{2}$D geometry aligned on the y axis.

`acis_checker target_file`

`acis_checker` checks every body in the target file for correctness and closure by invoking the ACIS `api_check` function. This check is a useful but not all inclusive check of the validity of the ACIS entities in the target file. Output includes statements regarding peculiar topologies (e.g. backwards coedges, backpointers to parent entities that don't claim an entity as a child), and geometric gaps. Degenerate surfaces are *not* included in this report. The functionality relies on built-in ACIS functionality; results are far more comprehensive if recent versions of ACIS (i.e. 3.0) are used.

`boolean tool_file target_file output_file`

Boolean decomposes (subdivides into small, simpler pieces) all bodies in the target file with all bodies in the tool file. Any body in the target file which is overlapped by a body in the tool file will be cut into separate bodies along the surfaces defined by the tool body (one of the bodies from the tool file).

`cube_maker target_file output_file`

`cube_maker` creates large cubes (hard wired for size currently in x and z direction) which have top and bottom faces (y axis) that correspond to the planar (constant y value) faces in the target file. The cubes created by `cube_maker` are useful for decomposing geometry. If `boolean` is invoked with a tool file of cubes and a target file from which the cubes were made, the bodies in the target file will be decomposed by the y axis aligned planar faces in the target file. On $2\frac{1}{2}$D geometry this leaves all of the resulting geometry meshable by a pave and sweep operation.

`extractor tool_file target_file output_file`

Extractor removes every body from the target file which intersects with any body in the tool file and places them in the output file. The target file is changed in this case. This tool can

be useful for segregating components which intersect before operating on them using boolean or subtractor.

imprint_files tool_file target_files output_file

`imprint_files` imprints every body in the target file with every body in the tool file and places them in the output file. When two bodies are imprinted, each body is modified to include an image of the other body where they touch or overlap. Such functionality permits each body to explicitly model its contact interactions with other bodies, and is necessary for producing contiguous meshes across contacting objects.

imprinter target_file output_file

`imprinter` imprints every body in the target file with every other body in the target file having its centroid at the same y location (this is very useful with David Saylors' `decomp` tool). This imprinting operation ensures contiguity of mesh between bodies on the same extrusion level in a model decomposed using `decomp`. The contiguity of mesh in the extrusion direction is guaranteed by the algorithms of `decomp`. Imprinting only within the same extrusion level greatly reduces imprinting time.

imprinter_all target_file output_file

`imprinter_all` imprints every body in the target file with every other body in the target file and places it in the output file. This imprint operation is more general than the `imprinter` tool and can take a significantly longer time.

joiner target_files. . . . output file

Joiner places all the bodies from the input files into the output file (wildcards can be used for the target files). This tool is extremely useful for building models out of separate files of geometric parts.

inquire target_file

`inquire` writes to standard output the centroids and moments of bodies and lumps and the areas and moments of faces in the model.

leveler target_file output_file tolerance

`leveler` operates on all of the faces in the input file which are normal to the y axis and are planar. These faces are moved on the bodies so that their centroids are at their original location rounded to the tolerance value supplied. WARNING: `leveler` will probably perform erratically on files containing non-$2\frac{1}{2}$D geometry or on files containing $2\frac{1}{2}$D geometry which is not aligned with the y axis. leveler is useful for removing small gaps which can exist between components.

plane_align_bodies target_file output_file {x|y|z}

`plane_align_bodies` moves each body in the target file so that its minimum bounding box[6] location aligns with the x, y, or z plane. This is useful for removing small alignment offsets between groups of bodies.

plane_align_file target_file output_file {x|y|z}

`plane_align_file` moves each the bodies in the target file so that the minimum bounding box around all the bodies is aligned with its minimum location on the x, y, or z

[6] Minimum bounding box is an axis-aligned box computed by functionality provided within ACIS.

plane. This is similar to `plane_align_bodies` except that it moves all the bodies in the target file as a group.

`rotator target_file output_file {x|y|z} angle (degrees)`

`rotator` rotates all the bodies in the target file [angle] degrees about the given axis.

`scaler target_file output_file scale_factor`

`scaler` scales all bodies in the target file by the scale factor and saves them in the output_file .

`self_boolean target_file output_file`

`self_boolean` decomposes every body in the target file with every other body in the target file and places the result in the output file. This ensures that none of the bodies in the output file share volume.

`shell_filler target_file output_file`

`shell_filler` creates a new body for every void or solid within the main body. It blindly creates a new volume for all but the first shell in the target file. The first shell of an ACIS body is by convention its external boundary. This is very useful for creating solid bodies to represent material filling the voids within components and parts. The foam within a closed volume can be created using `shell_filler`. The volumes occupied by bodies within the foam can then be formed by subtracting (using `subtractor`) the surrounded bodies from the foam.

`splitter target_file output_root_name`

`splitter` puts every body in the target file in a separate file with the base name `output_root_name`. This will be a more useful function when it is enhanced to name the resulting files according to attributes attached to the bodies in the target file.

`subtractor tool_file target_file output_file`

`subtractor` subtracts every body in the tool file from every body in the target file and puts the result (if any) in the output file. `subtractor` is very useful in producing models of encapsulants such as foam and potting material.

`tool_maker target_file output_file`

`tool_maker` creates extruded bodies from every planar (constant y) face in the target file. Size of bodies is currently hardwired. `tool_maker` combined with `cube_maker` and `boolean` can be used to fully decompose a set of $2\frac{1}{2}$D bodies.

`translator target_file output_file {x|y|z} distance`

`translator` translates all bodies in target file a distance along the chosen axis.

`uniter target_file output_file`

`uniter` unites (combines into a single ACIS body) all the bodies in the target file.

While any individual tool might seem almost trivial, the combination of tools is quite powerful. For instance, foam can be modeled by performing unites of objects to create a shell, which can be converted to a solid with shell_filler. The voids for components within the foam can then be created using subtractor. See Figure 7.

The availability of a robust translator and a suite of solid model manipulation tools is essential if solid models are to be used for the generation of spatial discretizations suitable for finite element analysis. Most solid modeling applications focus on providing the ability to add features to a solid in the process of defining a part. Their ability to manipulate multiple parts into an assembly appears to be largely an afterthought. The command line based geometry tools allow the user to manipulate components and assemblies on an equal basis. These manipulations can be scripted to allow replication of components after the original model parts have been modified.

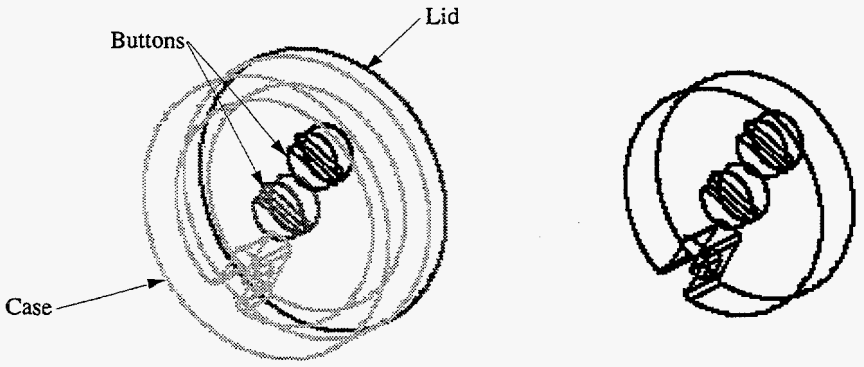


Figure 7: An example of constructing a model of encapsulant from the surrounding geometry. The left image shows a case, lid, and two buttons. The right image shows a solid body created by filling the void between the case and lid and subtracting the two buttons.

Part Simplification

Regeneration based feature abstraction is a strategy for transforming a feature-based part representation into a collection of solids each of which corresponds to a feature on the part. The strategy exploits Pro/E's sequential regeneration of each feature of the part and depends on a reliable translation of the Pro/E part into a solid model after each regeneration step. If a correct solid model of the part can be produced after the regeneration of each feature, and if these solid models are sufficiently precise to allow Booleans between them, then Booleans performed between the solids of each regeneration will produce solids representing only the features added during the generation.

If a part consists of just two features, a protrusion and a cut, then subtracting the solid containing the second feature from the solid corresponding to the regeneration before the cut was added will create a solid part corresponding to the cut. If the second feature is a protrusion, then subtracting the first part from the second part will produce a part corresponding to the protrusion feature. The procedure is executed for the entire feature sequence to produce "delta volumes", which are volumetric models of the material added or removed with each feature.

This procedure can be automated. It removes all parent child relationships. The delta volumes can be joined back onto or subtracted from the base part to produce a part with any combination of the original parts' features. Features can be combined with each other with boolean operations to produce models of feature interactions, and permit portions of

features to be applied to a model. The original feature definitions still limit the kinds of simplifications that can be performed, but the ability to recombine features in arbitrary ways strongly reduces the affect of feature definitions. All these operations depend on the successful operation of the translator and the Boolean operations.

The approach was tested with a small number of examples, with Pro/Engineer as the generator of data and ACIS as the external geometry engine. It worked on our test cases, one of which is shown in Figures 8 (the original feature history) and 9 (the automatically derived delta volumes). The approach was expected to have robustness problems when faced with large inaccuracies in the data because it relies on boolean operations. We have not yet encountered accuracy-related difficulties, possibly because the bulk of the geometry is precisely coincident, without the grazing cases that commonly plague Booleans. Further experimentation is warranted.

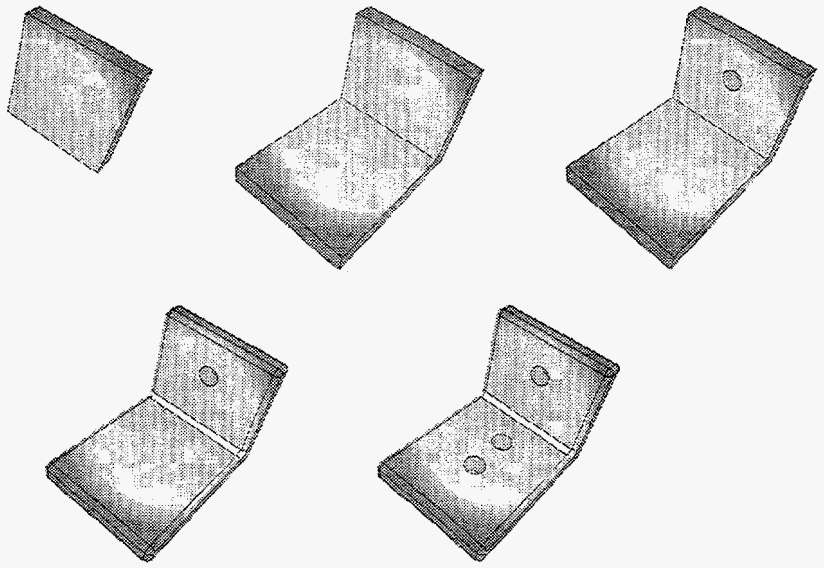


Figure 8: The feature history of a part.

It is possible to construct a variant of boolean operations that would be significantly faster and more robust for generating delta volumes. The approach exploits the fact that the models we compare to produce delta volumes are precisely equivalent except where the new feature has modified the geometry. A fast search can determine what geometry and topology is identical in two models being compared, and construct the difference graph. This difference graph can be operated on with simple Euler operations to produce the desired delta solids, without the attendant computational cost or numerical problems associated with boolean operations.

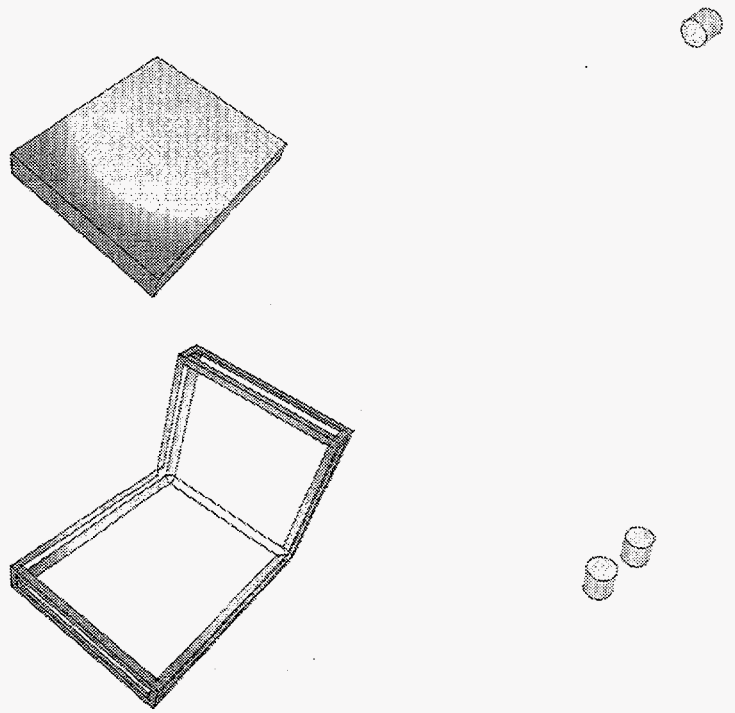


Figure 9: Automatically derived delta volumes representing changes induced with each feature. These delta volumes, combined with the base feature, can produce the original part and a much wider variety of variants than feature suppression.

Design Rules

We developed a set of design "suggestions" that would improve the ease of simplifying Pro/Engineer models for use in radar and finite element activities. The design rules included:

- Defining part features using datums first, followed by main shape defining features, followed by details. Such a feature ordering ensures that main shape doesn't depend on details.
- Ensuring that parts can be regenerated at highest accuracy. This can be done by either always using a high accuracy, or by periodically regenerating with accuracy high to ensure that feature geometry is resolvable at high accuracy.
- Dimensioning features from datums, rather than other part geometry. This is consistent with geometric dimensioning and tolerancing recommendations. Align references are nearly impossible to define from datums; we believe align references to be a "crutch" that supports inadequate geometric reasoning in the feature engine, and recommend investigating feature-based approaches that don't require align references.
- Avoiding blends to a point or blends between contours with different numbers of edges.

- Design in $2\frac{1}{2}$D wherever possible, as fully 3D geometry is profoundly more difficult to mesh with hexahedral elements.

We presented these suggestions to a group of designers, and they met a mixed response. Designers have a difficult job to perform, and our recommendations were viewed as potentially adding yet more difficulties and restrictions to the job.

Results

Both the Pro/Engineer- and ACIS-based simplification approaches were tested in production-like settings.

The Pro/Engineer-based tools were used to prepare models for surface meshing with CUBIT. When we began using CUBIT in this fashion, failures were much more common than successes. Production use of the tool was impossible because it was impossible to estimate how long a given job would take. There were simply too many show-stopping errors in the data (especially surface degeneracies and geometric closure problems), and no mechanisms for diagnosing problems.

Experience with Pro/Engineer-based tools shows that the most useful tools were those that detected problems with the data. Degeneracies and invisible topologies become easy to detect. After detection, eradication is ordinarily rather straightforward. A seasoned Pro/Engineer user was ordinarily capable of eliminating most causes of problems, and the process of performing the simplification was an excellent way to educate the designer in the problems that are created by certain design practices. Figure 10 shows a manually-performed simplification that eliminated a previously show-stopping degeneracy. Figure 100 shows a model that was radically simplified with the help of automated parent/child inferencing.

The ability to see and eliminate problem areas has increased the analysis capability to approach production-worthiness. In terms of our quality metric, we have made great strides in increasing our ability to deal with new, unknown data. The prototype is still incomplete, but it addresses an important class of problems encountered in real data.

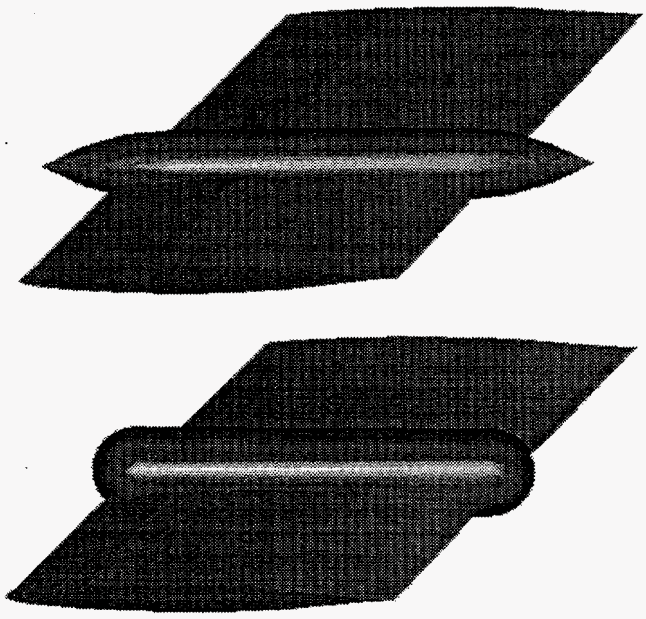


Figure 10: Manual simplification to remove automatically detected degeneracies at pointed ends.

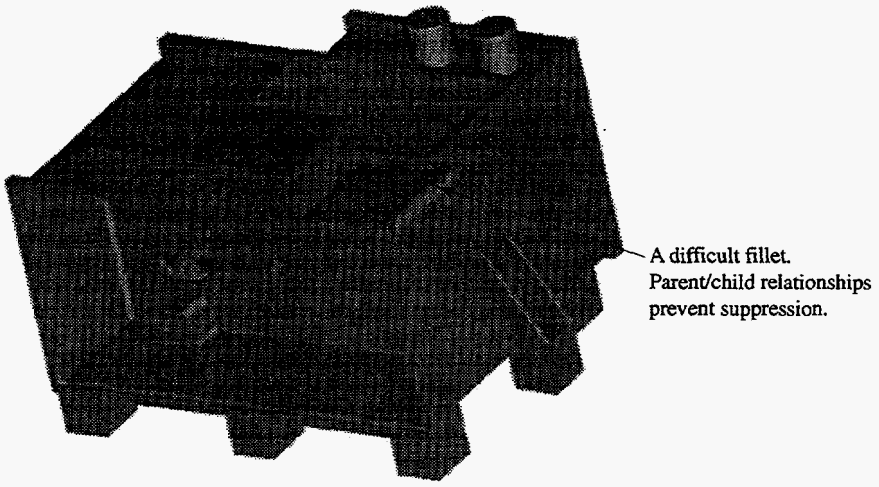


Figure 11: A part simplified using the Pro/Engineer-based tools. Only a few stubborn details remain, and the part is proven free of degenerate surfaces.

The ACIS based tools were used to operate on a complex $2\frac{1}{2}$D assembly in an attempt to use CUBIT to produce a mesh. Figure 12 shows a simple example of geometry processed by these tools to produce decomposable, meshable geometry. The tools significantly increased the degree of success in meshing the assembly, but were somewhat incomplete. Simultaneous with this activity, a more manual approach was being attempted. The manual approach was chosen over the automated route because many of the automatic tools were under development. However, our ability to attack complex data sets grew significantly.

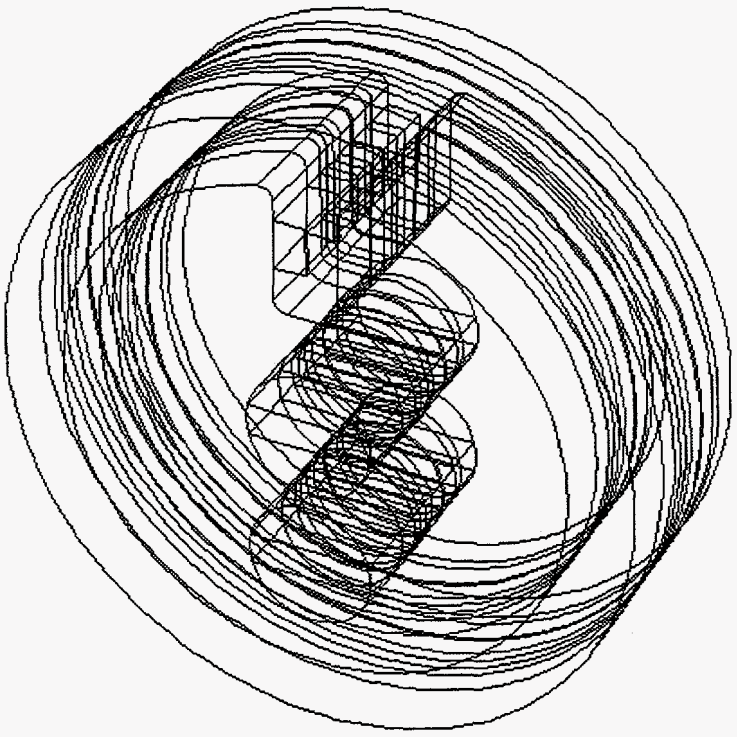


Figure 12: A decomposed, meshable assembly. Foam was produced and assembly gaps were corrected automatically.

Our most effective development activities involved a reasonably tight coupling between developers and analysts. The ACIS-based code was, in fact, developed by an analyst (the "user" was the "developer"). Close communication between analyst and developer ensures that the right modules are built, and with a reasonable priority. Testing the results on real production problems is the most effective way to guarantee that the tools will be genuinely effective.

Future Directions

Both toolsets would be very useful in a production environment. In order to become production worthy, a number of enhancements are required, as described below.

We note that the prototype software needs to be more robust. All of the modules built for this work were developed to test theories of simplification, so are rather lacking in the software engineering required of production code.

Pro/Engineer-Based Simplification Tools

The Pro/Engineer geometry search capability should be expanded to provide more thorough search coverage. More extensive attribute-driven search and search for features and parts based on visibility are priorities.

The ability to reroute features is important to any suppression-based scheme. The Pro/Toolkit environment claims to provide the requisite functionality, so should be investigated.

A diagram showing feature precedence (parent/child interactions) would be very useful in planning a simplification strategy, and could help designers prevent awkward feature definitions. The model tree window, new in Pro/Engineer 18, can provide an excellent basis for such functionality.

ACIS-Based Tools

The `acis_checker` module should be enhanced to check for degenerate surfaces, and for assembly-level problems, including gaps between parts.

Software to recognize that parts are meshable by currently available techniques (e.g. pave-and-sweep) would be a significant help in finding peculiar geometries.

To extend the work from $2\frac{1}{2}$D to 3D, it is necessary to deal with gaps between parts. One approach recognizes gaps and overlaps and adjusts the solids for precise coincidence. A different approach would modify the imprinting algorithm to imprint based on imprecise coincidence, and adjust mesh coordinates to achieve closure. The first approach is much more difficult to implement, but is more general in supporting ancillary meshing requirements, such as decomposition.

Further testing of regeneration based feature abstraction is warranted. Our testing was limited to rather simple parts, with no NURBS surfaces. Automation of the process would permit significantly improved testing, and would provide a basis for a production capability. The translation infrastructure (modeler accuracy, Pro/E to ACIS translation) must be enhanced to create a production capability.

References

[1] Pro/Engineer Fundamentals, Release 16, Parametric Technology Corporation, Waltham, MA, 1995.

[2] Pro/Develop Reference Guide, Release 16, Parametric Technology Corporation, Waltham, MA, 1995.

[3] ACIS Geometric Modeler Application Guide, Version 2.0, Spatial Technology Corporation, Boulder, CO, 1996.

[4] Recognizing Shape Features in Solid Models", H. Sakurai and D. Gossard, IEEE Computer Graphics and Applications, Sept. 1990.

[5] TrueGrid Manual Version 1.4.0 XYZ Scientific Applications, Inc., 1997

[6] Jones, R.E., et al, "Constraint-based Interactive Assembly Planning", *Proceedings IEEE International Conference on Robotics and Automation*, 1997.

[7] Ames, A.L., et al, "Liaison Based Assembly Design", Sandia Report SAND96-3004, 1996.

Distribution

Internal Distribution:

1	MS0151	G. Yonas, 9000	
1	MS1165	J. Polito, 9300	
1	MS 1002	P. J. Eicker, 9600	
1	MS 1010	M. E. Olson, 9622	
10	MS 1010	A. L. Ames, 9622	
1	MS 1010	J. J. Rivera, 9622	
1	MS 0105	R. E. Asher, 2435	
1	MS 0312	A. Webb, 2435	
1	MS 0835	T. C. Bickel, 9113	
1	MS 0835	D. M. Hensinger, 9113	
1	MS 0316	P. F. Chavez, 9204	
1	MS 0441	T. Tautges, 9226	
1	MS 0625	L. Grube, 9783	
1	MS 0625	E. Eager, 9783	
1	MS 0624	C. Neugebauer, 9784	
1	MS 0624	M. R. Ashby, 9784	
1	MS 0188	LDRD Office, 4523	
1	MS 9018	Central Technical Files, 8940-2	
5	MS 0899	Technical Library, 4916	
2	MS 0619	Review & Approval Desk, 12690	For DOE/OSTI