SANDIA REPORT

SAND99–0326
Unlimited Release
Printed February 1999

Immersive CAD

Arlo L. Ames

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 (703) 605-6000
Web site: http://www.ntis.gov/ordering.htm

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

DISCLAIMER

Portions of this document may be illegible in electronic image products. Images are produced from the best available original document.

SAND 99-0326
Unlimited Release
Printed February 1999

Immersive CAD

Arlo L. Ames
Engineering and Manufacturing Software
Intelligent Systems and Robotics Center
Sandia National Laboratories
P. O. Box 5800
Albuquerque, NM 87185-1010

Abstract

This paper documents development of a capability for performing shape-changing editing operations on solid model representations in an immersive environment. The capability includes part- and assembly-level operations, with part modeling supporting topology-invariant and topology-changing modifications. A discussion of various design considerations in developing an immersive capability is included, along with discussion of a prototype implementation we have developed and explored. The project investigated approaches to providing both topology-invariant and topology-changing editing. A prototype environment was developed to test the approaches and determine the usefulness of immersive editing. The prototype showed exciting potential in redefining the CAD interface. It is fun to use. Editing is much faster and friendlier than traditional feature-based CAD software. The prototype algorithms did not reliably provide a sufficient frame rate for complex geometries, but has provided the necessary roadmap for development of a production capability.

Acknowledgments

Thanks to the following for contributing to the development of the Immersive CAD prototype environment: Ed Pena, James Brazee, David Hickerson, Charles Little, Chris Wilson. A special thanks to Pat Xavier for the C-Space Toolkit. Thanks to Craig Peterson and the staff of MUSE Technologies for timely development of device drivers and other software supporting our work.

Thanks to David Hensinger and Deepesh Kholwadwala 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 Immersive CAD LDRD project. The project investigated approaches to providing immersive editing of CAD geometry at VR framerates, including both topology-invariant and topology-changing editing. A prototype environment was developed to test the approaches and determine the usefulness of immersive editing. The prototype showed exciting potential in redefining the CAD interface. It is fun to use. Editing is much faster and friendlier than traditional feature-based CAD software. The prototype algorithms did not reliably provide a sufficient frame rate for complex geometries, but has provided the necessary roadmap for development of a production capability.

Modern feature-based modelers have provided an excellent mechanism for capturing detailed design intent in a form suitable for establishing contractual agreements for making parts. Their solid model underpinnings provide exact, unambiguous, three-dimensional geometry necessary for supporting downstream processes. Product data can be attached anywhere in the model (on a face, edge, part, or assembly) in the form of attributes. Parametric and variational capabilities permit studies of design space. Dimensions and tolerances can be applied (to the degree we understand them). Their dialog with users is in familiar engineering terms (e.g. hole, slot, and pocket).

These modelers are the first practical approach to delivering solid modeling capability to production design environments. Despite their success, current solid modeling tools fail to provide a friendly conceptual design environment. They are very details-oriented, requiring complete specification of every detail of how each feature relates to the part. If a detail, such as a rotational degree of freedom, is left out, it cannot be added later without serious model reconstruction, and cannot be varied without potentially serious model redefinition.

Capture of editing intent seems to be a principal focus, even to the exclusion of capturing design and manufacturing intent. Frequently, no provision is made for stating the functionality of various features, or constraints such as wall thickness that would guarantee manufacturability or part functionality, and the order of feature creation and feature interactions dictates future editability.

Feature based modelers, while faster and easier than previous solid modeling tools, can be terribly slow. Practical experience has shown models that require hours to load, and minutes or hours to modify, depending on which feature is being edited. Time required for regeneration tends to be proportional to the size of the feature list, rather than on the degree of change in the final model.

Operators of CAD systems find the software interfaces disruptive. The designer decides on the kind of edit to be performed and has to navigate both menus and complex dimensioning models to arrive at the right series of mouse clicks to perform a simple action – nothing as simple as grabbing the object to be moved and moving it. Operators have been observed "wiggling" the geometry back and forth to achieve a more complete spatial understanding of the shape of the object being edited.

In the quest for speed, some feature-based modelers have compromised model accuracy. Compromising accuracy can produce significant problems for using the geometry in downstream applications, especially in performing geometry slicing and Boolean operations.

Paper and pencil (e.g. writing on the back of an envelope) continues to be the conceptual modeler of choice. CAD tools tend to be used to capture the final design and make small tweaks to shape, but engineers still draw on the back of a napkin to "get their ideas on paper."

Problem Statement

Given the wide variety of problems experienced with current modeling systems, our problem is to investigate techniques for providing a responsive, intuitive conceptual modeling tool, and develop a prototype for testing these techniques.

A basic requirement for intuitive editing is the capability for grabbing an object and dragging it, without resorting to menus, and having that object move in "real time." Real time, in this context, means that, as far as possible, every operation works within virtual reality frame rate – 15 frames per second or faster. It is desired that such speed occur on contemporary hardware with a small number of processors (hopefully one).

Simultaneously, objects should behave in engineering terms – a hole should be able to act as a hole, not permitting itself to interact with other objects in ways that compromise its integrity to contain, attach, and be sufficiently strong, unless the operator so wishes.

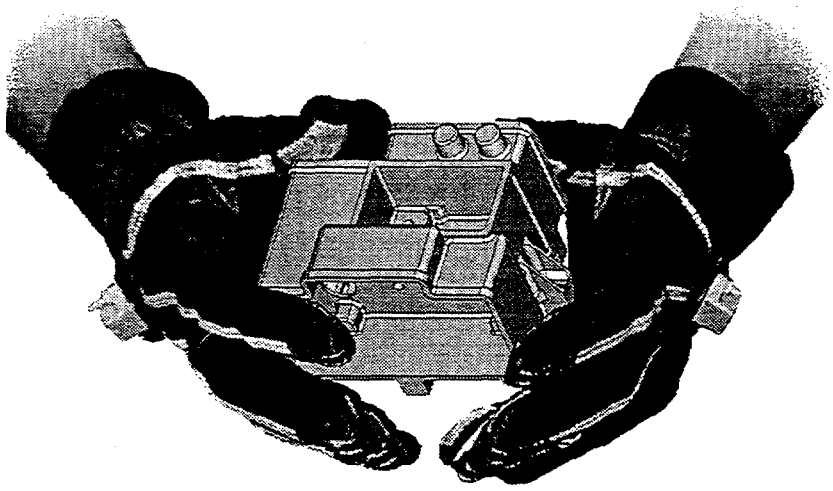

Figure 1: "Hands-on" modeling concept.

Ideally, this fast, responsive modeler should be built upon a very numerically precise representation, with no compromise of accuracy in the quest for speed.

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 solid modeling representations are presently in production use at Sandia, in Pro/Engineer [15], and ACIS [1].

Constructive Representations

A constructive representation is a list of operations to be performed in creating a shape. Examples include Constructive Solid Geometry (CSG) and Feature Based Modeling (FBM). Such representations permit easy change to geometry Shape-defining parameters are explicit, so can be changed directly. These representations are not directly usable for many applications, as the interactions between elements of the design (primitives, features, etc) must be computed –the representation must be evaluated.

Evaluated Representations

An evaluated representation is a representation of geometry in which the shape of the object is explicitly evaluated -- the object is expressed as what it is, not as a collection of operations that would create the shape. Boundary representation and spatial enumeration are examples of evaluated representations. These representations can be directly edited, but only in fundamental terms (e.g. move a face). They are directly usable by most downstream applications.

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. They 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.

It is possible for a feature-based model to be a hybrid between constructive and evaluated models. At each step during regeneration, an evaluated model exists. The hybridization occurs if features are permitted to reference geometry in evaluated geometry in the regeneration of previous features.

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

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.

Boundary Representation

Boundary representation (b-rep) 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[1]. *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 its complement is outside. The second notion of interior tells which side of each contour is in the bounded region of the face. The tangent direction of traversal of edges, crossed with the direction of surface normal at a point in question, determines 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.

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 constructive and evaluated versions of assembly representations. A constructive representation might define 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.

Facets

A faceted representation of a solid is a kind of boundary representation, where the bounding loops have simple topologies (e.g. triangular or quadrilateral) and surface complexity is minimal (e.g. planar or bilinear surface). A faceted solid has many more

---

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

faces than an "exact" model of the same object, but the faces have lower geometric complexity. Such representations involve much simpler algorithms for manipulation, so are more amenable to hardware optimization.

"Exact" solids can be converted to a faceted representation through a process called "facetization." Each face of the exact solid is triangulated; the runtime of such a process is proportional to the number of faces, and the geometric and topological complexity of each face.

Faceted representations are used in this work for rendering (using GL rendering hardware), and for fast, approximate computation of geometric properties such as distance between objects.

Part Modeling

Part modeling is the creation and modification of the shape of an individual part. Part modeling involves changing the fundamental geometry and topology of the object. Such editing operations are technically challenging from a solid modeling standpoint. Our work in this area focuses primarily on developing and experimenting with approaches to exploiting invariance in editing operations.

Previous Work

Historically, CAD systems offered solid model editing directly in terms of solid operations. In constructive solid geometry, such editing operations were limited to editing the CSG tree and modifying parameters associated with sizing and orienting primitives. In boundary representation, the operations involved creating primitive shapes and applying Euler operators and Boolean operations. Such editing operations never achieved wide acceptance in the design community, ostensibly because such operations bear little resemblance to the manner in which designers think about editing geometry.

More recently, feature based modeling systems (e.g. Pro/Engineer) have developed. These systems have become a preferred approach for editing solid geometry because the human/machine interface is represented in natural terms (hole, pocket, and slot). Such systems maintain an ordered list of feature-based operations (e.g. "add hole") whose evaluation produces the part's geometry. Each feature is a recipe for performing a single editing operation. The feature contains the geometry and parameters required to size and shape the feature, along with dimensions for positioning the feature. Evaluating the feature involves creating its geometry (e.g. by sweep, primitive instantiation, etc.) and computing the interaction between the feature and the rest of the model. Regeneration is the process of producing the final geometry by evaluating each feature in turn, with each operation potentially modifying all prior geometry. After regeneration, the geometry is rendered. For wireframe display, edges are discretized and the resulting lines are drawn. For hidden line display, unoccluded portions of edges must be computed, followed by discretization and drawing as for wireframes. For shaded images, each face of the solid is faceted and the facets are passed to a polygon rendering algorithm (frequently implemented in hardware).

Gadh, et. al. [4, 5, 6] has performed work in immersive modeling with solid models. The work has focused on topology-invariant modeling and on interface techniques. This work differs in attacking the problem of topology change and detecting when topology change occurs.

Complexity of Modeling Operations

In practical experience, model regeneration can take from seconds to hours, depending on the complexity of the model. Particularly annoying is the fact that small edits can be either very fast or incredibly slow, depending on where in the feature list the edit occurs. In order to determine feasible approaches to providing fast, grab-the-object-and-move-it operations, we examined the complexity of performing editing operations in feature-based CAD systems.

The regeneration process currently implemented in CAD systems treats geometry generation as a monolithic process. The shape of each feature and its interactions with other features in the model are computed from scratch each time the model is changed. Computation of the feature's geometry is a function of the geometric complexity of the object (frequently linear, but possibly more costly, as in computing fillets). Computation

of the feature's interaction with the remainder of the model is, topologically, related to the number of faces in the feature, the number of faces in the object being interacted with, and the complexity of each face. This topological complexity is roughly O (n^3log n), where n relates to the number of faces. This complexity results from the size of the potential output; the number of intersections present in the output can far outweigh the size of the input in pathological cases. The complexity of each face/face comparison is related to the pairwise complexity of the surface definitions involved (plane/plane intersection costs linear time evaluation of closed-form equations, while NURBS/NURBS intersection costs quadratic time in the complexity of the surface patches, computing iteratively to convergence).

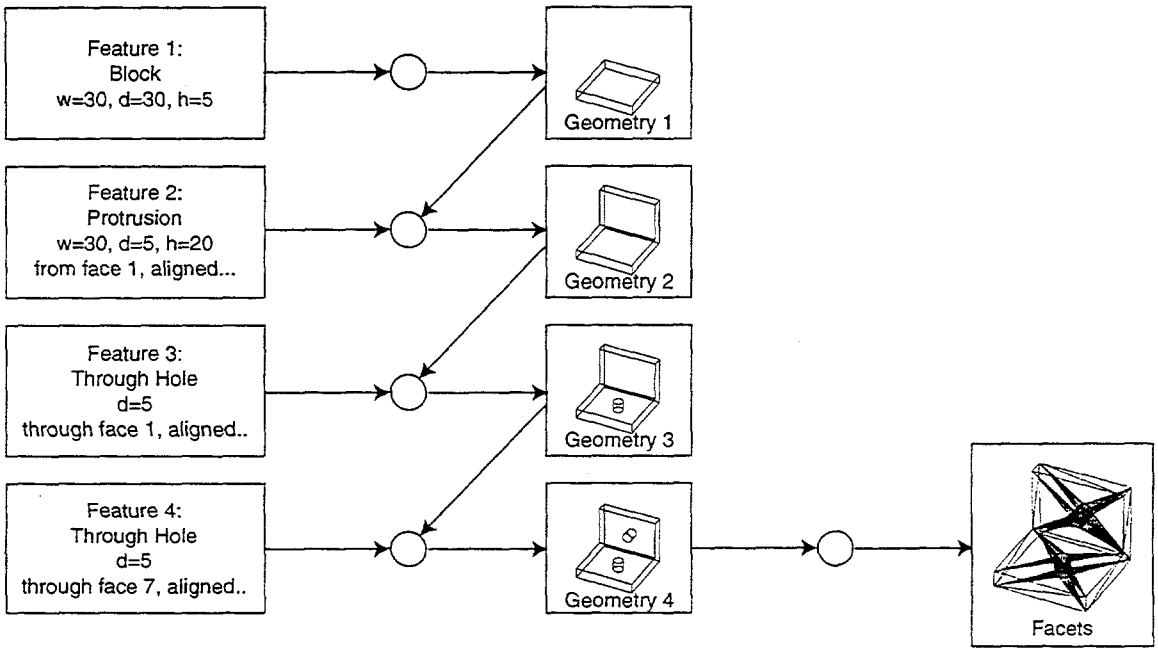

Figure 2: The regeneration process.

The complexity of faceting is proportional to the number of faces to be faceted, the geometric complexity of the surface, and the number and geometric complexity of the edges bounding the face. The runtime for rendering facets in hardware is linear in the size of the faceted representation.

Simplified Representations

Since the complexity of editing operations is a polynomial function of the complexity of the model, it is clear that operating on simplified models can provide significant speed improvements. Note that simplification recommendations are only recommendations, as it is necessary to provide the designer with an adequate representation for the design process to proceed – overly simplified models are useless.

The topology of models is a primary factor in the speed of operations. Adding features costs roughly O (n^3log n) time, so reducing the number of faces in the model being operated on is a significant consideration. Some features, especially fillets, by their very nature add a significant number of faces to the model. Filleting a simple block triples the number of faces in the model. Other features (e.g. holes, slots and pockets) tend to have significantly less interaction most times, so cause roughly linear growth. It is important to keep the model small by limiting the number of interactions wherever possible. "Feature suppression" is available in production modelers, and if carefully used can provide a mechanism for leaving out complex geometry most of the time.

Geometric locality can significantly decrease the expense of feature editing. Every boundary-representation system has some kind of boxing technique for determining whether faces interact – if the boxes don't interact, the faces don't. In cases where faces have small geometric extent relative to the size of the model, a significant reduction in computational effort is realized. The value of exploiting geometric locality is dependent on the design being created; this is a function of the object being designed (and the designer's thought process).

The geometry of faces can have significant affect on the speed of operations. Quadric surfaces can be intersected in constant time, while NURBS surfaces involve iterative solution over pairs of patches. Many of the world's engineered artifacts can be described using quadrics – those that cannot are surfaces found in consumer products (e.g. telephones), automobile bodies, and aircraft surfaces. Blend and variable radius fillets create NURBS surfaces; they can also be created to represent surfaces that might otherwise be represented using a lower-order representation (e.g. torus). Again, avoiding or suppressing features that produce complex surfaces can keep speed high.

Curved surfaces tend to cause large numbers of facets to be required for rendering. Facetization parameters can be adjusted to limit the number of facets required to render the object, but have only a limited effect on the facetization. Curved geometry tends to place a lower bound on the number of facets required to adequately render a shape. Adjacency between curved faces and the requirement for facet contiguity can cause some faces to display quadratic growth in facet count. Adjusting facet parameters produces, at best, a linear decrease in the number of facets on the body, so are somewhat ineffectual in reducing facet count. Suppressing complex surfaces altogether can strongly reduce the number of curved surfaces and face/face facet interactions, dramatically reducing facet count.

The discipline involved in producing representations that simplify easily (either through clever modeler construction or clever model construction) is more important in an immersive environment than in traditional CAD. In traditional CAD, slowness is annoying. The user asks for something and waits. In immersive CAD, the rapidly generated images provide a critical link in the feedback loop between man and machine. Sluggish response destroys the illusion of being immersed.

A side benefit of producing simplifyable models is that simplification is useful for enabling and accellerating downstream applications, especially finite element meshing [3].

An approach to exploiting simplification during part editing is to regenerate only that portion of the model that can be conveniently computed within the frame rate's limitations, then display that model. It is necessary that the modified geometry can be included in that computation, but all other geometry is potentially optional.

Partitioning the model into degrees of detail is very effective in exploiting simplification. If the model is partitioned into those features that define major shape, followed by features of increasingly smaller details, it is straightforward to enable an algorithm to use that partitioning to organize work. Partitioning can be attempted algorithmically (e.g. based on size of faces, feature dependency, and geometric location), but performing the partitioning as part of design permits maximal user control, and is suggestive of design importance to a degree not likely obtainable by automatic recognition.

Choice of Representation

Given that we have a variety of representations (features, exact b-rep, and facets), we have an opportunity to choose which representations to edit. Maximal engineering content occurs within the feature-based model. It is expressed in engineering terms, and carries any engineering constraint information. The feature-based model cannot be viewed, however, as it is a constructive model. It must be evaluated. The exact b-rep contains less engineering information than the feature-based model, and more than the faceted model. It is an "exact" model of the engineered artifact. The exact b-rep can be rendered using ray tracing, but this is too slow to achieve the requisite frame rate. The exact b-rep is available for import from other CAD systems, though, while feature-based models can only be translated between CAD systems through major custom development (essentially reimplementing the generating CAD system's feature evaluator in the receiving system). The faceted representation is almost devoid of engineering content (it being an approximation of the shape of the object, with no explicit engineering information), but is fastest to render and easiest to modify. Faceted representations are trivial to translate between systems.

In order to provide maximum operational flexibility, we desire a system that is capable of importing data from any source and editing it with maximal engineering fidelity. This translates into a requirement to import whatever data is available, and represent multiple models internally. Selection occurs on the lowest-level entities in the system (i.e. facets). Those entities point to the entities that created them (i.e. exact faces) if present. Those entities point to the entities that produced them (i.e. features). Constraints on editing operations are implemented at the highest level available. Thus, if only facets are present, editing operations move nodes of facets. If faces are present, selection of a facet actually selects that face, and all the facets on the face move. If a feature is present, selection proceeds to select the feature, so all the faces produced by the feature move.

Opportunities for Exploiting Invariance

Typical design editing operations include adding, deleting, moving and resizing individual features. Operations such as inserting a hole, moving a boss, or widening a slot typically involve relatively few faces out of the entire model. Moreover, such operations rarely introduce changes in the interactions between features – if a hole begins to intersect another hole, it's design integrity might be compromised. If most of the model is unchanged during an editing operation, most of the computation involved can be avoided, speeding up interaction dramatically with no compromise in accuracy. There are various degrees of invariance: script invariance, topology invariance, geometry invariance, and facet topology invariance.

The Small Change Assumption

Solid modeling editing operations tend to change small portions of the model at a time (e.g. move a hole or a single face), rather than moving every face simultaneously. Even though certain editing operations might suggest global changes, changes are most easily understood if they are somewhat localized. This tendency toward localized editing operations suggests that the number of faces that move during editing will be small compared to the overall model.

While the small change assumption isn't an invariance that can be directly implemented into algorithms, it is necessary to note that any attempt at real-time solid modeling, without complete model regeneration, is dependent on being able to strongly limit the number of modeling operations required to compute the changed model. Without the assumption that edits are small compared to the overall model, this work is for naught.

Script Invariance

Whenever a feature is modified, it is easy to notice that all of the features before it in a feature list have not changed. Thus, any work involving generating geometry to that point in the script can be avoided if the results of that work are cached. Such a cache can be easily implemented (at significant memory cost) by making copies of the geometry at some interval (e.g. every 5 features) during the regeneration process. Any time regeneration occurs, locate the cached geometry before the earliest modified feature, and regenerate from the cache. This technique is used in commercial CAD systems, but it alone cannot provide enough reduction in computation to achieve VR frame rates, because the list of features that must be reevaluated will still be long (possibly involving every feature in the list).

Taking advantage of script invariance *after* the modified feature is more difficult. In cases where the modified feature affects later features, it is necessary to regenerate the affected features, just as in traditional regeneration. In cases where the modified feature doesn't affect the modified feature, it is necessary to re-play the topological and geometric modifications to the model, without incurring significant computational cost. Research in persistent data structures is promising as a mechanism for minimizing the cost of replaying operations while minimizing computation.

Exploitation of simplified models has an effect on exploiting script invariance. If the model is constructed as a progression from most significant to least significant features, features of similar significance will be evaluated at the same time in model construction. Thus if any portions of the model cannot be visited in the frame rate, they will at least be the least significant features, and the most significant portions of the model will be maximally available for display during editing.

Topology Invariance

Most design changes that don't involve adding new geometry are topologically invariant. The editing operations that don't involve adding new geometry are changing the size or location of various features. Any new interactions between features that might occur (e.g. a hole intersects another hole) are likely to change the ability of the features to achieve their designed function.

In any editing scenario, if it can be shown that the topology of the model is invariant, then the only required computation for an edit is computing the equations of the curves and surfaces that move. The interrelationships between entities (the number and connectivity of faces, edges and so on) remain unchanged.

Geometry Invariance

Even in cases where the topology of the model is changing, the geometry of the model is changing very little. Tracking editing operations as they are performed guarantees that the changes between any two frames are small. For small motions, the new interactions will occur in the geometric neighborhood of the objects being moved. It is unnecessary to compute the potential interactions between all faces in the model – it is sufficient to compute the interactions between moving faces and faces in their local geometric neighborhood, if such faces can be located easily.

Facet Invariance

For any editing operation, a significant number of faces will be unchanged. For such faces, the facets are invariant. In cases where faces have only been rigidly transformed without any change in their bounding edges, it is possible to apply rigid motion to facets without performing any refaceting.

In cases of faces that have changed geometry nonrigidly but are topologically invariant, it is possible that the topology of the facetization can be reused. Whether this is true depends on whether facets have become degenerate or inverted. It is possible to relax facet vertices to the changed boundaries, then check to determine whether any affected facets have become degenerate. Degenerate facets can be removed and an incremental faceting algorithm invoked to close the holes in the faceting. Updating, checking, and correction can occur in roughly linear time, and much faster than refaceting every changed face.

An Editing Algorithm

Having identified opportunities for exploiting invariance, we are faced with producing an algorithm suitable for maximally exploiting invariance during editing operations. We assume that a small number of faces of the model are being modified at any given time.

Facets

It is desired to have the ability to view the geometry using any viewing model (wireframe, hidden line, smooth shaded) during the editing process. Viewing as wireframe requires that the edges of the model be discretized; viewing (and selecting) as a shaded image requires that the faces be discretized (faceted). Faceting faces is more computationally expensive than discretizing edges, so the following discussion will focus on faceting as the limiting case.

The simplest exploitation of facet invariance is to only refacet those faces that were changed, simply redisplaying any faces that were left unchanged. The modified faces (moving faces, faces previously adjacent to moving faces and newly adjacent faces) have their facets deleted and recreated with the new geometric model.

A more complex model for exploiting facet invariance is to locally edit the facets of a face to match the changed geometry. This means local modification to facet geometry and topology. Results in finite element adaptivity and rezoning might provide excellent resources in this area.

Since faceting is roughly linear in time with model complexity and refaceting is easy to implement, we implemented refaceting. Time constraints prevented investigation of local facet modification. We suspect, without proof, that facet modification is of limited significance in increasing editing speed. We focused on more significant efficiency issues.

Detecting Topology Invariance

Once a face is selected, it can be moved. During most motion, the topology of the model will not be changing. Occasionally, the face will either become too close to a face it was not adjacent to or will move too far from a face it was adjacent to, resulting in a topology change. Since topology changing potentially requires Boolean operations and topology-invariant editing does not, we must be able to distinguish between the two possibilities.

An obvious approach for determining the potential for topology change is to compute distances between faces in the model. In order for such a distance measure to work, it must be sufficiently fast to work within the frame rate, leaving time for the requisite modeling operations to be performed. Our real-time distance computation algorithm of choice is from [20]. This algorithm is derived from Gilbert's algorithm [11], which provides very fast distance computation between convex objects by noting that, for convex objects, search for closest points is essentially an unconstrained global optimization with no local optima. Search for closest point pairs is limited to search through entities adjacent to the last closest pair before the motion took place, and is roughly O (1) in time. The C-Space Toolkit (CSTk) [22] extends Gilbert's algorithm to nonconvex objects by producing

nested geometric hierarchies that bound the nonconvex objects, which limit the number of comparisons to be performed. It also performs distance calculations between translational and rotational sweeps of objects (rotational sweeps are less exact than translational sweeps). The algorithms are capable of applying affine transformations to the geometries being compared, permitting the compared objects to be scaled. Distance updates are very fast; building the hierarchies can be somewhat slow. The CSTk assumes that objects being managed are topologically invariant, and are only geometrically modified by application of affine transformations; under these constraints, hierarchies need only be built once. Unfortunately, our application permits topology to change and allows arbitrary geometric operations.

An alternative approach for predicting topology change is invocation of a topology-invariant geometry-updating algorithm, followed by checking for errors and self-intersections. The error check is simple; the self-intersection test is rather expensive.

Performing topology change tests via distance computation can provide benefits beyond the simple topology check, so it is preferred. The distance computation involves a certain amount of overhead, but avoids the self-intersection test.

A remarkable benefit of providing algorithms for computing proximity is the ability to implement a wall thickness constraint. Contrary to popular naive expectation, modern CAD systems have no significant capability for guaranteeing the manufacturability or functionality of parts being designed[2]. They are simply generators of geometry from a script of manufacturing-like instructions. With a wall thickness constraint, it is possible to globally apply a new kind of constraint that guarantees that every feature is sufficiently bounded by material to be strong enough to manufacture and use. This capability is achieved by simply modifying the tolerance on the proximity algorithm used to detect topology changes – the tolerance corresponds to the desired wall thickness. This tolerance can be applied globally, or on a face-wise basis, or on the basis of any spatial function, as desired.

Choice of Representation for Determining Topology Change

In order to compute distances between moving faces and stationary faces, we must represent the shape of these objects (we will call this representation the dc-representation, or dc-rep). This is more difficult than it seems, as the faces can change shape as they are moved. There is some latitude in choosing distance-calculation representation: representing the face's feature as an object, attempting to compute the modified shape of the face, or approximating the face by transforming the unmodified face's facets. Additionally, there is some latitude in deciding whether to use the exact face or some other representation (e.g. a skeletal representation).

In the case where a face has a defining feature, it is possible to define the shape of its interaction volume by using the feature to define the shape. For example, for a blind hole, a cylindrical volume could be used. This approach has the advantage that for many feature-based operations, the interaction volume will have a constant topology and will be edited by applying only affine operations, so the dc-rep need only be computed once for use with the CSTk. The approach is deficient for features whose shape is defined by geometric context,

---

[2] Some CAD systems do permit constraints to be applied between parameter values. Implementing a wall thickness constraint is very cumbersome using such a capability, however, because of the need to constrain every face-face pair, and the trigonometric nature of the constraint. For n faces, a complete wall thickness constraint would require n^2 nonlinear constraints to be created and maintained.

such as blends, fillets, and "through-next" holes (whose depth depends on what they interact with), since the extent of the geometry is unclear.

It is possible to compute the modified shape of a moved face by reintersecting the surfaces used for edge definitions, based on the new location of the moved face. This approach would fail if the faces cannot be intersected, but such failure is expected in cases of topology change anyway. This approach is quite general, but essentially involves the same work as performing the topology-invariant edit, even if it won't work. The approach does require recomputation of dc-reps.

Approximating a face by simply transforming its facets is a fast, simple approach to approximating the shape of a moved face. It is technically incorrect, but works well under the assumption that faces are moving reasonably small distances. Remembering that the purpose of computing distance is to avoid using topology-changing algorithms, it is only necessary that the distance calculation is conservative: that it doesn't fail to invoke the topology-changing algorithm when it is needed. An appropriate tolerance (roughly equivalent to the size of the expected error) can be applied to make the algorithm conservative. Dc-reps must be recomputed using this approach.

Representing stationary faces involves roughly the same mechanics as representing moving faces. The faces adjacent to moving faces require a bit of special handling. During a move, moving faces are constrained to not contact objects they weren't originally in contact with. For non-adjacent faces, this is simple face/face contact. For adjacent faces, this is edge/face or vertex/face contact between edges and vertices of the adjacent stationary face and the moving face. This special case was encountered in moving a face of a tetrahedron. Without a vertex/face constraint, there are no non-adjacent faces to prevent a moving face from causing a topological change that eliminates the model entirely.

After faces are modified, it is necessary to recompute the faceted representation of the model, both for rendering and for creating appropriate dc-reps. These computations must be performed both for moved faces and for faces adjacent to the moving faces, since the boundaries between the faces can change shape. Discretization of shared edges is not a parallel activity (although it is quick). Facetization of each face is a parallel activity, and behaves as a nearly linear-time algorithm. Dc-rep computation from scratch (with current CSTk algorithms) is significantly slower than facetization. We are investigating approaches to updating dc-rep hierarchies without complete recomputation.

Computing Geometry for Invariant Topology

For invariant topology, it is necessary to compute the geometry of the changed model. Topology invariance requires the recomputation of the surfaces of moved faces, followed by recomputation of the curves and points of intersections between the surfaces of the moved faces.

We initially developed a set of algorithms for moving surfaces and computing curve and point geometry for moved surfaces, but found that a robust capability was available for ACIS in the form of the Local Operations Husk [3].

Computing Topology Changes

Topology change requires a well-formed set of semantics. It is necessary for the nature of the interactions between modified faces (both moving and stationary) to be understandable if appropriate behavior is to occur. In the simplest example, a through hole moves off the edge of a part completely. The hole has no bottom. Now the hole is separate from the part. If the hole's geometry impinges on the face of the part and the hole had no bottom,

any solid modeling operation between the disjoint hole and the part will result in an incomplete solid. While it is possible that the semantics of topology change can be inferred for certain special cases, it is possible to guarantee model semantics by requiring a feature based model to be available; by definition, the feature-based model describes the semantics of feature-part interaction.

The simplest implementation of topology change involves traditional regeneration to resolve topology-changing operations. While regeneration is slow, by definition, it should not be overlooked. The frequency of topology changing operations during editing is small compared with topology invariant operations. Even if regeneration is slow, it is comparatively rare during editing. Regeneration will result in a pause during editing, while the new topology is computed.

Caching geometry at various points during regeneration can be used to exploit feature-based invariance. Regeneration proceeds from the last cache before the feature in question.

Taking advantage of invariance after the feature being modified is more difficult. For small edits, the bulk of the geometry and topology after the edited feature is invariant. Each feature after the modified feature that can be shown to be invariant with respect to the changed geometry is invariant. If the feature is not invariant, it must be replayed, and added to the list of modified features.

Replaying invariant features rapidly requires a mechanism for adding them to the model with minimal cost. Two approaches have been considered. In the first, the geometric results of each feature are cached; these results include both the geometry of the feature and where that geometry stitches to the model. Replay requires copying the cached feature geometry and mapping the stitching references to the new model after regeneration. Constructing the mapping could require nonlinear time.

An alternative mechanism for rapidly replaying features involves the use of persistent data structures [9]. This approach requires modification to the geometry engine. Each pointer in the model has associated "time" information associated with it; the state of the model after each feature operation can be queried by providing the appropriate "time" information in the query. Construction and maintenance of the "time" information is reasonably well-formed, with predictable cost in time and space. The current state of research in persistent data structures is limited to tree-based data structures, however. The graph-based data structures required in boundary representation are not currently supported.

We developed the simple regeneration-based approach. Work on replaying invariant features has begun, but remains unfinished due to time constraints.

Assembly Modeling

Assembly modeling operations involve changing the relative spatial relationships between collections of otherwise rigid objects (parts). Since shape of individual objects is invariant in assembly modeling, the operations on objects are limited to modification of transformation matrices (or other part placement representations) that position parts within the assembly. From a geometry standpoint, assembly modeling is significantly less challenging than part modeling. Since assemblies consist of any number of parts and assemblies, which individually can be rather large, assembly modeling is still quite challenging.

Previous Work

Current feature-based CAD systems approach assembly modeling in the same manner as part modeling: with a feature tree that is evaluated monolithically. Each feature represents positional relationships between parts. A part can be placed in an assembly when enough relationships (dimensions) have been provided to constrain 6 degrees of freedom.

Previous work in virtual reality has concentrated on facet reduction to maintain frame rates in large models. Such VR techniques, coupled with appropriate part simplification, would seem to address the problem of rendering speed adequately, so we defer its consideration here.

Proximity-Based Approach to Constraint Definition

A number of our production designers have expressed frustration at modeling assemblies using the 6DOF constraint-based feature modeling approach. They have requested a modeling approach in which the parts of the assembly can be treated like toy blocks: stacked and unstacked, rotated and moved until the assembly meets their needs.

We investigated an approach to assembly modeling in which the proximity between moving and stationary parts is monitored. Whenever faces are in reasonably close proximity, it is possible for a mating constraint to occur between them. Once a mating constraint is defined (at user option), it is maintained until broken (at user option). Any further motion of the part is constrained to keep contact faces together.

The same fast distance function used for topology invariance determination is used to compute distance. Once close pairs of faces are found, it is necessary to query the user about desired constraint. If a constraint is required, it is created. If not, it is necessary to remember that the constraint was not made (until the proximity condition is not met), so as to avoid querying about constraint at every frame. The constraint is simple to implement; any portion of a motion that has a component in the direction normal to the surfaces in contact is subtracted from the input motion. The remaining components can be applied. Thus, if two planes are in contact, the part can still be rotated about an axis in the normal direction of the planes, and translated in the planes.

The Prototype

A prototype immersive CAD system was developed to test the ideas developed here. The prototype implemented in ACIS [1], Sandia's C-Space Toolkit [22], the MuSE [13] immersive environment, and OpenGL. The ACIS representation is an exact boundary representation; no modifications were made nor operations performed that would compromise the accuracy of ACIS. MuSE supplied the immersive environment: navigation capabilities, positioning, head tracking, plus interfaces to virtual reality hardware, including Crystal Eyes [7], Fake Space Pinch Gloves [10], Polhemus FASTRAK trackers [23], Spaceball [18] and mouse. ACIS provided the representation for exact boundary-representation geometry and the ability to facet that geometry. OpenGL provided the interface to draw ACIS-provided facets in the MuSE environment. The C-Space Toolkit provided distance calculations to enable decisions about topological invariance in modeling operations.

For prototype development, the hardware devices were used as follows: the mouse was used by the right hand for navigation through MuSE's existing interface. Crystal Eyes were available selectively for stereo viewing. A Pinch Glove was worn on the left hand and used for choosing what modeling operations to perform. A Polhemus tracker was worn on the back of the glove to permit motion tracking of the hand for input of translation

and rotation. A tracker was available for head tracking. A 195 MHz single processor SGI Octane with 384 MB of memory performed all of the computation. Sufficient memory was available to prevent any paging activity from slowing down the process.
The basic algorithm is as follows: for each frame, the pinch glove is queried to determine whether a command has been given. Valid commands include selection, translation, surface offset, and scaling functions for motion sensitivity. If a translation or scaling operation is selected, the position of a motion tracker is noted, compared to position in the prior frame, and scaled to determine how faces are moving. Then motion is applied to the CSTk representations of the moving faces and their distances to each other and to nonmoving faces are computed. If no topology change occurs, ACIS local operations are used to compute the new geometry. If a topology change has occurred, the feature tree, if present, is replayed. For models lacking a feature-based representation, topology changes are only permitted to the degree that the ACIS local operations functionality permits (i.e. a face may cease to exist if it can be simply removed by either loop removal or extension of adjacent faces). After the geometry for the current frame has been produced, any new or modified geometry is faceted. Then all the facets are presented to GL for rendering.

Speed tests were performed for four models of increasing complexity. For each test, frame rate (in frames per second) was measured for simple flythrough, then for moving a single face. The number of faces adjacent to the moved face was noted, as the price of the edit is proportional to the number of modified faces. Since frame rate can vary with the size of the image on the screen, each object was rendered using roughly half of the screen width. For one test case, the block with holes, a feature-based model was present and complete regeneration was performed whenever topology change occurred.

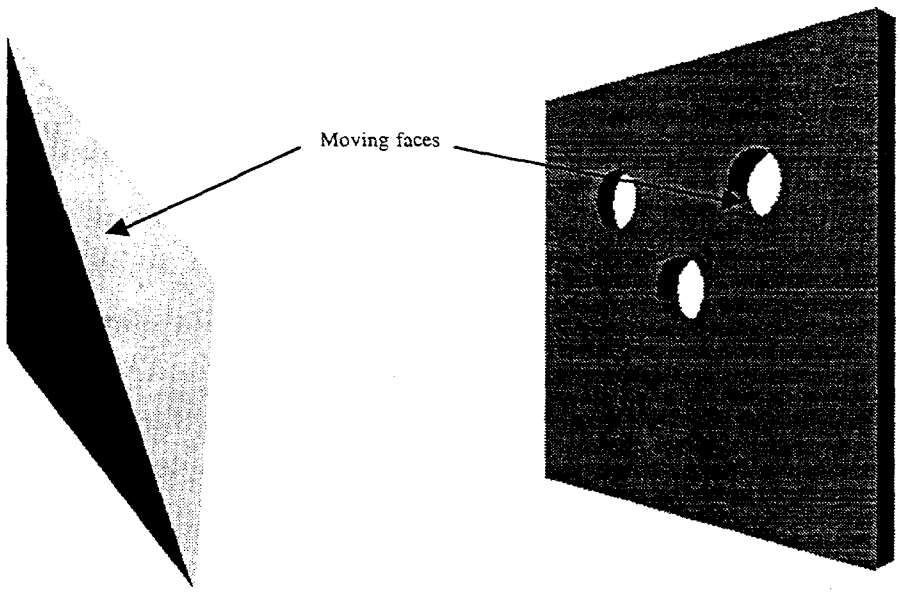

Figure 3: Tetrahedron and Block with Holes test cases.

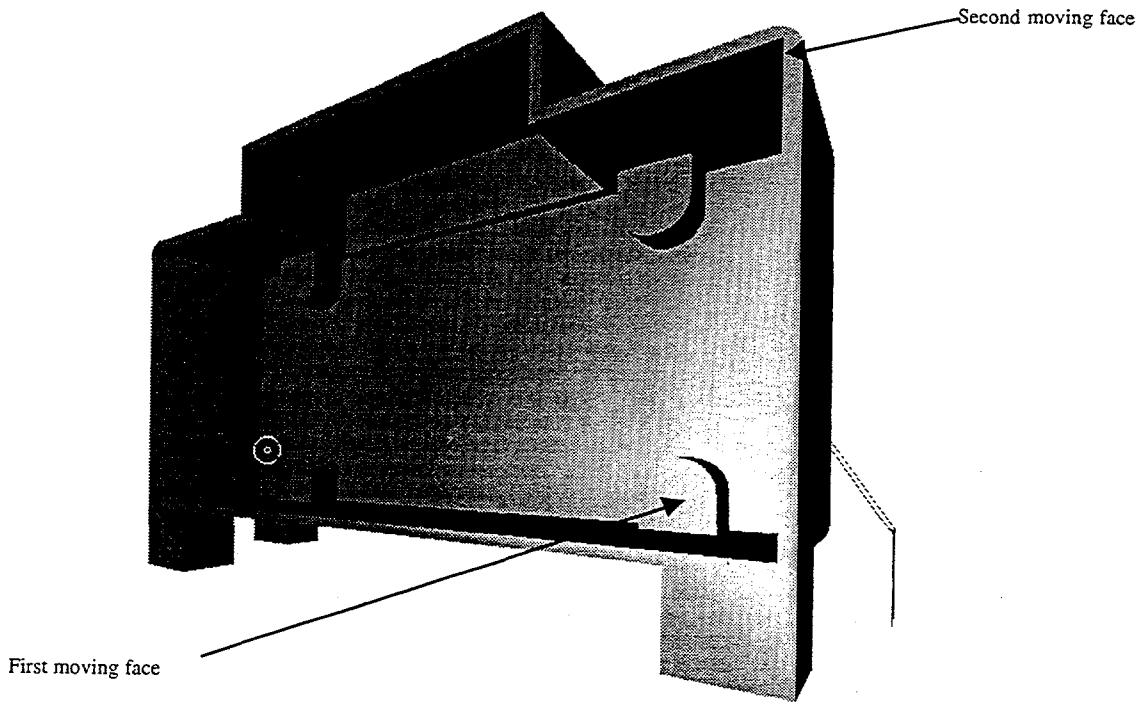

Figure 4: Simplified Casting test cases.

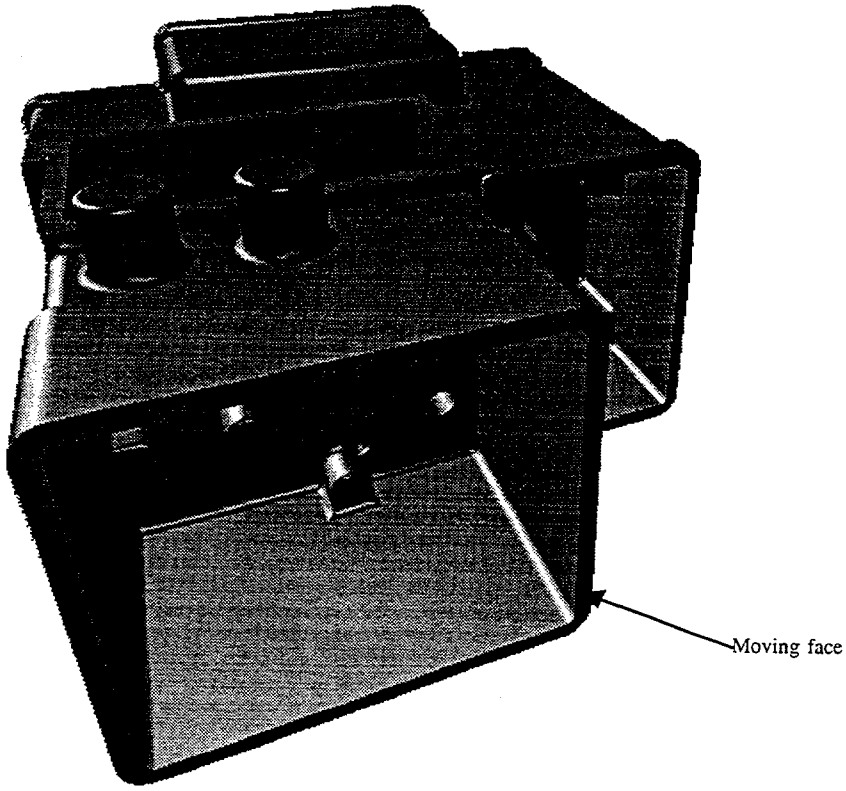

Figure 5: Casting test case.

	Number of Faces in Model	Number of Facets in Model	Faces Affected by Edit	Facets Affected by Edit	Flythrough Speed (fps)	Face Moving Speed (fps)	Face Moving Without Distance Check (fps)	Topology Change Speed (fps)
Tetrahedron	4	4	4	4	72	60	60	N/A
Block with 3 Holes	15	312	3	208	38	12	13	4
Block with 10 Holes	17	964	3	546	38	2	8	<1
Block with 20 Holes	27	1964	3	1066	38	2	5	N/A
Block with 30 Holes	37	2964	3	1586	36	1	3	N/A
Block with 40 Holes	47	3964	3	2106	25	<1	2	N/A
Simplified Casting	94	590	7	42	36	6	20	N/A
	94	590	23	151	36	2	7	N/A
Complex Casting	766	21752	17	182	9	<1	4	N/A

Figure 6: Editing speeds for various objects.

Typical practice in virtual reality dictates that frame rates be at least 15 frames per second. This requirement stems from the need to have the environment respond as quickly as a user is providing input. In use, the 6 frames per second rate was sufficiently fast to support editing, while 2 was noticeably jerky. For edits with distance checking enabled, only the simplest of edits (tetrahedron and block with holes) qualifies as being sufficiently fast to support the traditionally required frame rate. With distance checking disabled, the frame rates for more complex models became more tolerable. Recall that distance computation is required to ensure that topology changes are not occurring unchecked – they should not be disabled in normal editing operations.

The prototype represents our first attempt to create an immersive environment. Our intent was to produce a geometrically correct implementation with no compromises in functionality so that the practicality of the approach could be measured. For sufficiently small models, the immersive behavior is precisely what is desired – the model changes at the nudge of a hand, in a very responsive way. For larger models, more speed is necessary to move this from research to a production tool.

Despite any disappointment in the speed of the prototype, it must be noted that current CAD systems tend to require seconds, minutes, or hours to perform editing tasks. For all of the cases shown, the immersive frame rate is one or more orders of magnitude faster that the edit rate of traditional CAD systems. The feedback provided during a given editing operation is significant – rather than waiting to see whether an edit worked, the model

changes dynamically, providing real-time feedback on what is happening in the model. Despite any current limitations, Immersive CAD is fun to use.

Future Directions

Faster Response

There are a variety of approaches to improving the speed of the algorithms presented. The most obvious is to buy a faster machine. In every case presented here, CPU speed, not graphics pipeline speed, was the limiting factor. With no improvements in the basic algorithm, sufficiently fast machines should be available in roughly five years.

Related to simple speed improvements is the possibility of parallelizing the algorithms. Parallelizing the faceting and distance-representation computations will provide large returns, as they represent much more than half the computation effort, and are inherently parallel. Except for data storage concerns, the algorithms are completely independent, and tens or hundreds of processors can be applied easily to the problem.

Computation of dc-reps is likely much slower than necessary. The current algorithms being used are optimized for off-line computation of the dc-reps. They produce very fast distance computation at runtime, but perform the dc-rep computation offline. Our application requires dc-reps to be updated for small changes, which will require further development of the C-Space Toolkit.

Further investigation of different choices of dc-reps is warranted. Skeletal representations of certain features, e.g. holes, can profoundly reduce the geometry being searched during distance calculations. Additionally, features can be defined in partial terms (e.g. the through-next hole feature, whose limit is the first face touched). Such partial feature definitions are likely to require semi-infinite (partially unbounded) dc-representations.

Incremental Faceter

As shown by the test case of a block with 40 holes, it is possible for a single face to be very complex, and to be involved in any edit involving any other face. In such a case, the geometric operations can be trivial, while refaceting is very expensive. The faceted model is likely to change very little, except in the geometric neighborhood of the change, so an incremental faceting algorithm is necessary.

An incremental faceter would require knowledge of which facets are adjacent to moving geometry. For a given edit, only the facets adjacent to moving geometry would be modified. Their geometry might be locally changed, or in extreme cases, deleted and the faceter directed to fill in the missing facets. The faceting algorithm would be analogous to the topology-invariant and topology-changing algorithms used for maintaining the exact solid.

Incremental Topology-Changing Algorithms

Due to time limitations, we were unable to complete work in prototyping fast topology-changing algorithms. Development had begun on algorithms that would propagate topology changes through a feature tree using fast distance computation and persistent data structures to localize the cost of feature modification. This effort should be completed to enable effective study of improved feature-based modeling speed, independent of whether the modeler is traditional or immersive.

Expanded Feature-Based Model

Our prototype was limited to block and hole features to enable the study of immersive editing. The domain of discourse of the feature-based model needs to be significantly expanded to enable more complete evaluation of this technology in production CAD terms.

Incremental Analysis and Planning Algorithms

The immediate feedback provided by incremental geometry evaluation suggests the possibility of expanding beyond geometry into analysis and planning algorithms. Many downstream algorithms (e.g. kinematic and dynamic simulation, finite element analysis, etc), while computationally expensive, might lend themselves to incremental update. As such, they provide an opportunity to observe the effect of a design change in new terms: not as a monolithically computed answer representing a discrete point in design space, but as a continuum of answers showing downstream effects of every nudge of a design feature.

References

[1] ACIS 4.0 Online Help CD, Spatial Technology Corporation, Boulder, Colorado, 1998.

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

[3] Ames, A. L., J. Jill Rivera, Annie J. Webb, David M. Hensinger, Solid Model Design Simplification, Sandia Report SAND97-3141, 1997.

[4] Chu, C. P., Dani, T., Gadh, R., "Evaluation of Virtual Reality Interface for Product Shape Design", IIE Transactions, Special Issue on Virtual Manufacturing, February 20, 1998.

[5] Chu, C.C. P., Dani T., Gadh R., "Multimodal Interface for a Virtual Reality Based Computer Aided Design System", Proc. IEEE Int'l Conf. on Robotics and Automation, 1997.

[6] Chu, C. P., Dani, T., Gadh, R., "The use of virtual reality for creating solid model shapes", Proceedings of IMAGINA Workshops, Monte Carlo, February 1996.

[7] CrystalEyes 2 Stereo Eyewear User's Manual, StereoGraphics Corporation, 1997.

[8] Dani, T., Gadh, R., "COVIRDS: A New Approach to Concept Shape Modeling via a Virtual Environment", 1997 High Performance Computing Conference, 1997.

[9] Driscoll, J. R., Sarnak, N., Sleator, D., and Tarjan, R. E., ``Making Data Structures Persistent'', Proc. of the 18th Annual ACM Symposium on Theory of Computing, 109-121, 1986

[10] Fakespace Pinch™ Glove System, Installation Guide and User Handbook Document GL-9001, Revision B, Fakespace, Inc., CA, 1997.

[11] Gilbert, E. G., Johnson, D. W., and Keerthi, S. S., "A Fast Procedure for Computing the Distance between Complex Objects in Three-Dimensional Space, *IEEE Journal of Robotics and Automation*, 4(2), April 1988.

[12] Hoschek, J., and Dankwort, W., Parametric and Variational Design, B. G. Teubner, 1994.

[13] μuSE Application Programmer's Reference, Muse Technologies, Inc., Albuquerque, NM, 1996.

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

[15] Pro/HELP, Release 20, Parametric Technology Corporation, Waltham, MA, 1998.

[16] Requicha, A.A.G., "Representations for rigid solids: Theory, methods, and systems", ACM Computing Surveys, 12(4): 437-464, December 1980.

[17] Shapiro, V., "Maintenance of geometric representations through space decompositions", International Journal of Computational Geometry and Applications, 1994.

[18] Spaceball 2003B SpaceWare® Version 7.3, V 1.0, Installation & User's Manual, Spacetec IMC Corporation, 1997.

[19] Springer, S., Gadh, R., "Haptic Feedback for Virtual Reality Computer Aided Design", ASME International Mechanical Engineering Congress and Exposition, 1997.

[20] Xavier, P. G., "Fast Swept-Volume Distance for Robust Collision Detection", Proceedings 1997 IEEE Int'l Conf. On Robotics and Automation, 1997.

[21] Xavier, P. G., "A Generic Algorithm for Constructing Hierarchical Representations of Geometric Object", Proceedings 1996 IEEE Int'l Conf. On Robotics and Automation, 1996.

[22] Xavier, P. G., Lafarge, R. A., A Configuration Space Toolkit for Automated Spatial Reasoning: Technical Results and LDRD Project Final Report, Sandia Report SAND97-0366, 1997.

[23] 3SPACE® FASTRAK® User's Manual, Revision F, Polhemus Incorporated, VT, 1993.

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	
20	MS 1010	A. L. Ames, 9622	
1	MS 1010	D. M. Hensinger, 9622	
1	MS 1010	D. K. Kholwadwala, 9622	
1	MS 1008	P. G. Xavier, 9621	
1	MS 0316	P. F. Chavez, 9204	
1	MS 0188	LDRD Office, 4523	
1	MS 9018	Central Technical Files, 8940-2	
2	MS 0899	Technical Library, 4916	
1	MS 0619	Review & Approval Desk, 15102	For DOE/OSTI