Propagating Topological Tolerances for Rapid Prototyping

T. J. Peters, S. A. Demurjian, D. M. Needham[1]
Department of Computer Science and Engineering
University of Connecticut, Storrs, CT 06269 - 3155
email:{tpeters,steve,needham}@eng2.uconn.edu

R. J. Peters
16 Nicholas Avenue
Groton, CT 06340

S. M. Dorney
Grand Valley State University
Allendale, MI 49401 - 9403

Abstract

Rapid prototyping technologies permit timely inclusion of experimental verification within an iterative design process. This work introduces *topological tolerances* as a tool for retaining desired design invariance during the production of successively refined prototypes. These topological tolerances are applied to the supporting geometric data representations for rapid prototyping.

At each successive local modification of the geometric data, these topological tolerances must be dynamically updated. These updates are modeled by object-oriented propagations, leading to investigation of efficient algorithms to re-compute these local topological tolerances. Relevant industrial examples will be discussed.

Software to compute topological tolerances has been fully implemented and tested, and one purpose of this paper is to report that development. Another emphasis of this paper is to report upon software prototyping of performance enhancements to adapt this code to support iterative re-design within a rapid prototyping environment. The final emphasis is to report upon our use of object-oriented propagations within our software engineering environment ADAM to model appropriate change notification upon a decision to alter the supporting geometry data structures for rapid prototyping.

[1]These authors, as well as S. M. Dorney, acknowledge, with appreciation, partial funding for this work received under National Science Foundation Grant Number MII-9308346. The views expressed herein are of these authors, not of the National Science Foundation

1 Motivation: Prototyping Leading to Redesign

To provide context for this work, industrial evidence is offered as to the role of stereolithography as a rapid prototyping technique [25] within iterative design cycles.

From Chrysler Corporation, "[Stereolithography] allows us to test many, many more iterations of a piece, with little to no effort, so we have more time for development or [the development] gets done faster." At Ford Motor Company, " ... engineers are much more creative ... because it's more of a tinker shop and their creativity goes way up."

For evaluating a design, the importance of experimental stress analysis upon physical prototypes has been documented for the compressor disk of a gas turbine [16]. Such experimental stress analysis can improve upon the nondefinitive stress analyses available from computer modeling [8, 16]. For this compressor disk evaluation, it was only the experimental method of three-dimensional (3D) photoelasticity that provided the surprising result that unacceptable stresses were introduced. This design flaw in the blade could have caused part failure and discovering it prior to production avoided significant economic loses.

Although this particular blade study predated the advent of rapid prototyping, the perspective it affords underscores the merits of rapid prototyping relative to time and material issues. First, significant improvements are possible upon the time frames of two to eight weeks for the creation of traditional experimental models [10, 16]. Second, recent results [10, 22] have demonstrated that stereolithography models permit inexpensive, yet effective, photoelasticity.

These themes were also echoed in a recent National Science Foundation Workshop [17]. Having established the importance of permitting the design process to be responsive to the results of experimental verification, the emphasis here is upon the inclusion of topological considerations as essential for design optimization.

2 Introduction: Topological Invariance

The desirability of topological invariance during iterative redesign was initially expressed relative to tolerance modeling [20, 21]. There, the treatment was largely informal, but mathematically rigorous foundations for preservation of topological form have subsequently been proposed [6, 7]. Consistent with these

cited works, the definition given here for *topological tolerance* is analogous to the typical use of dimensional tolerances, which imposes numeric limits upon a dimension.

Definition: A *topological tolerance* is a numeric limit specifying bounds within which the geometry of an object may be perturbed, while guaranteeing preservation of its topological form.

Hence, a topological tolerance may be considered as a constraint upon the freedom of geometric editing, and the terminologies of topological tolerances and topological constraints will be used synonymously throughout this paper. The definition of topological invariance used will be that two objects have the same topological form provided that there is a homeomorphism from R^3 onto R^3 which carries one object onto the other [6, 7, 23]. This is considerably stronger than the classical topological equivalence as defined merely by a homeomorphism between two objects [24]. The additional requirement of defining the homeomorphism over all of R^3 addresses equivalent embeddedness of the objects within R^3.

Our emphasis upon direct editing of .STL files is reflective of the state of the art that the .STL file is the *existent* data representation currently used to communicate the critical information *shared* between designer and fabricator. In practice, both the resultant .STL file and the original CAD file are saved by the designer. Hence, if changes are made to the triangulated geometry of the .STL it is important to update the corresponding free-form geometry of the CAD file. While the editing techniques discussed are currently restricted to triangulated geometry, this is consistent with the historical development of many CAD tools. The mathematics for preserving topological consistency is much more difficult for spline surfaces. However, the authors have begun efforts to develop that theory and view the present report as the initial step towards that envisaged extensibility. The theoretical mathematical basis for topological tolerances [1] is now rephased for the present application.

Theorem: Let S be a finite triangulated polyhedron within R^3 with a corresponding parameter ν defined to be the minimal separation between disjoint pairs of vertices, edges and faces. If for each j and each vertex p_j, the vertex perturbations δp_j are such that $\|\delta p_j\| < \nu/2$, then the perturbed object retains the same topological form as S.

The intuitive summary for present purposes is that this theorem provides a global upper bound to maintain topological form, allowing all geometric perturbations within the stipulated limit. Its advantage is that it is a single parameter, which

can be calculated once for each design. Its disadvantage is that it is somewhat conservative, because locally some greater perturbations might be possible [9].

Software to compute ν has been fully implemented [9]. Successful "proof of concept" studies have been done to adapt this code to the iterative modification of .STL files, using local tolerance techniques, which are not as restrictive as the global upper bound. Object-oriented propagations within our software engineering environment ADAM model appropriate change notification upon a decision to alter the geometry of the .STL file. Summarizing tersely, the software developed to compute ν follows the obvious quadratic algorithm, which would be quite acceptable in many contexts, but the large data magnitudes of .STL files renders this computation prohibitively slow for interactive design experiments. Thus, this paper also reports upon investigations for culling appropriately small data subsets from the .STL file as a pre-processing step to achieve acceptable performance.

3 Topological Integrity for Rapid Prototyping

The "... *de facto* solid freeform fabrication industry standard..." [3] is a data representation consisting of triangulations of the boundary surfaces of a solid. The emphasis of the work presented herein is to provide software tools (which could even be invoked over the World Wide Web) to preserve topological integrity of a .STL file when the part geometry is being modified.

The topological integrity of the data stored in an .STL file is crucial for reliable rapid prototyping[3]. Unresolved topological inconsistencies "... frequently cause problems during fabrication." Algorithms are presented to remedy topological deficiencies, so as to allow the manufacturing cycle to proceed with a solution that "... is numerically robust and ... is based on topological principals [sic]."

If geometric edits are applied without sufficient guidance, then unintended self-intersections might arise within an .STL file. For ease of illustration, a 2D example of such difficulties is depicted, but it should be noted that the software developed is fully supportive of 3D geometry. The left half of Figure 1 represents a simple polygon. The right half of Figure 1 represents how perturbing geometry in violation of topological constraints could yield an unintended self-intersection, leading to the creation of a non-manifold. Clearly, it is desirable to prevent such edits.

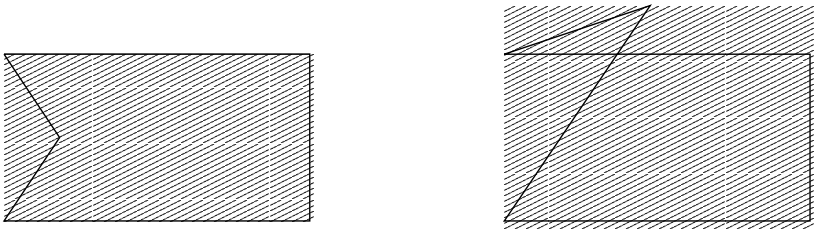

Figure 1: Creation of a non-manifold

4 Algorithm and Cullings for Local Edits

Our "proof of concept" software prototyping experiments utilized visual inspection of graphical displays of .STL files, followed by text editing to cull the .STL file. This has led to a general purpose algorithm, 'Local ν Algorithm', (given below) to support programmatic assistance for such culling. An example of those culling experiments is presented, with discussions for enhancements to the general 'Local ν Algorithm'. For interactive design experiments, the integration of appropriate change notification has been modeled by object-oriented propagations, as will be discussed in Section 5.

Local ν Algorithm

Purpose: Compute a local constraint for retaining topological form.
Input: A vertex to be perturbed, p.
Output: The numerical value of the local constraint.

Local_$\nu(p)$: Determine $C(p)$, the appropriate culling set for p.
for all geometric elements g in $C(p)$,
compute $d(g,p)$, where
d is the usual Euclidean distance function,
Let $\nu(g) = \min\, d(g,p)$ over all g.
end for loop.

The above algorithm outlines the essential procedure for Local_$\nu(p)$ for one vertex. Since the .STL file is entirely of triangulated data, in principle, any other desired edit can be achieved by finitely many calls of the above algorithm. However, the culling set must be integrated with the specific calls, requiring some finesse. Also, the above algorithm has no dependency upon direction, so it may still be somewhat conservative, in that even greater freedom for perturbation

may be possible if direction is specified. Supporting this additional freedom by means of propagations is discussed in Section 5.

Culling Example: A representative culling, executed upon an industrial .STL file from Pratt and Whitney Aircraft (P&WA), will be described. The overall design is known as a ball box (Please see Figure 2.) and included over 5,700 facets. Visual inspection was used to decide a candidate area on the inside of the ball box (Please see Figure 3.) for local editing and that subset is shown in Figure 4. As a first step in deriving the culling set, a particular vertex was chosen and designated as p, shown in Figure 4 as the vertex common to the 90 degree angles of the black and dark grey triangles at the center of the image. An initial candidate for the appropriate culled set was selected by visual inspection. These experiments led to discovery of efficient techniques to compute $C(p)$, as is addressed in the final paragraph of this section.

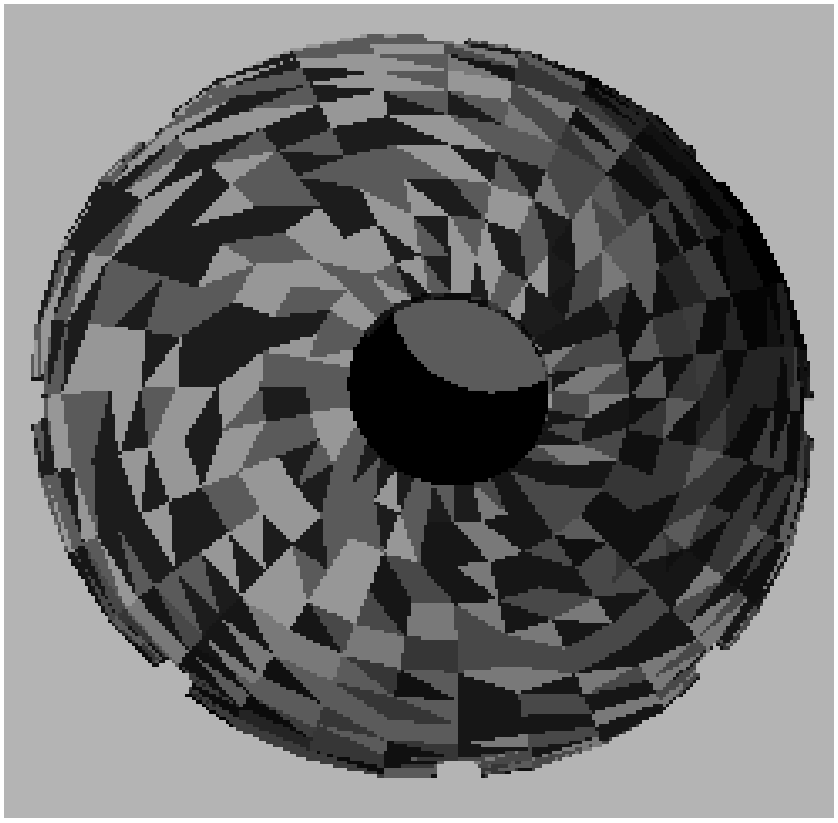

Figure 2: Ball Box

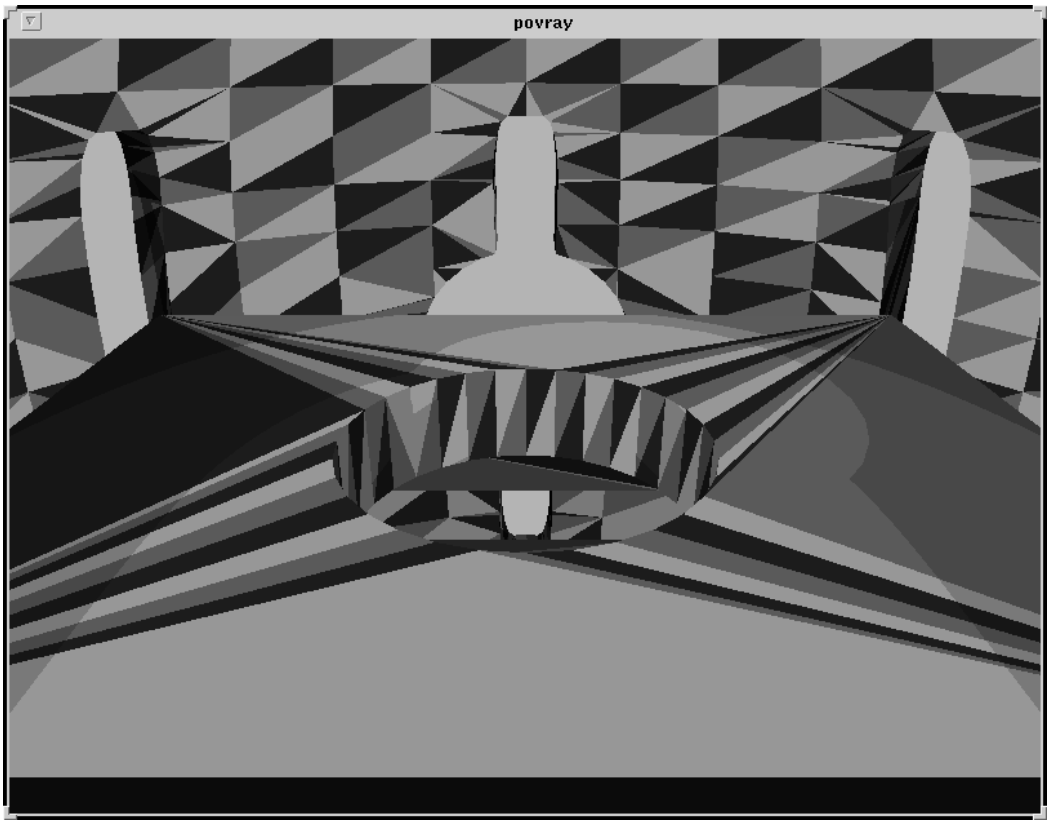

Figure 3: Internal View of Ball Box

The statement of the Local_$\nu(p)$ algorithm is offered to explicate the application of the existing mathematics to our situation, but, for application to interactive editing sessions the *explicit value* of a local_$\nu(p)$ may be less important to a user than a true/false response as to whether a contemplated edit is permissible. In reviewing our culling experiments, it becomes clear that a conservative set $C(p)$ could be determined upon the basis of a facet's intersection with a bounding box [11] sufficiently large to contain the intended perturbed geometry and its preimage. If multiple movements were required, then an obviously larger bounded set would be appropriate and could easily be created as a superset (possibly improper) of the union of the collection of bounding boxes. The performance trade-offs in creating these larger bounding sets bear some additional investigation. Using this culled set as input to the basic algorithm results in a directionally dependent local_ν to be compared to the contemplated movement. The use of such techniques for interactive editing is addressed by object-oriented

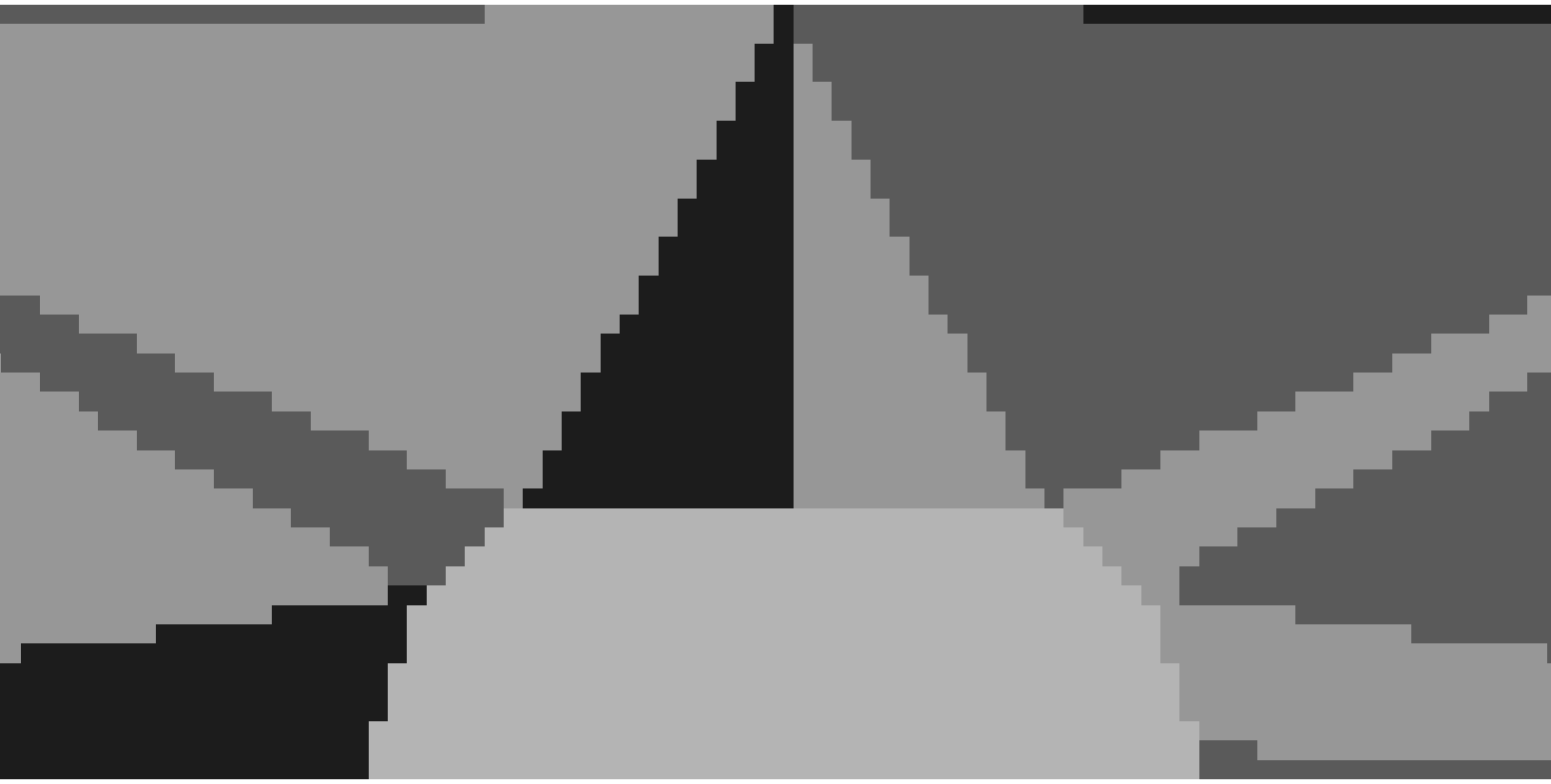

Figure 4: View of Selected Region from Interior of Ball Box

propagations within the next section.

5 Propagations for Local Edits

ADAM (short for Active Design and Analyses Modeling) is a University of Connecticut software engineering research environment, which supports language-independent design via graphical/textual user interface and automatically generates compilable code in multiple languages (C++, Ontos C++, Ada95, Ada83 and a dialect of LISP) [18] for entered designs that contain object types, inheritance, relationships, and propagations. A *propagation* is a design-level trigger for modeling dynamic behavior among different object types. ADAM provides support for the design-time specification of propagations by allowing the modeling of interdependencies between design objects that are not ancestrally related.

An initial "proof of concept" study applied propagations to mechanical design features [18] and an industrial application [4, 5] utilized four propagations. The present context of successive edits to .STL files will typically entail tens to hundreds of thousands of propagations–one for each vertex, thus providing a rich environment for performance issues upon significantly increased scale. Representative individual vertex propagations have been modeled in ADAM, as indicated, below.

Figure 5 is a bitmap of the ADAM environment modeling the interdependency relationship, *Update_Vertex*, between an object-type *Vertex* (which represents a particular vertex being moved) and *V_Culled_STL* (which represents the portion of the .STL file that is effected.) In Figure 5, *Update_Vertex* is an ADAM propagation modeling abstraction. Figure 5 indicates the simplest propagation circumstances, where a simple numeric comparison has determined that the total intended perturbation distance is strictly less than $\nu/2$, where ν is the global, conservative constraint, which is directionally independent and can be computed once, off-line.

The fundamental action that must be undertaken upon moving p less than $\nu/2$ is to update that subset of the .STL equal to star(p), where star(p) is defined as the union of all facets containing p as a vertex [2]. Note that the previously computed global ν will continue to suffice as a constraint with respect to the movement of *any other* vertices, as such stability was the intent in defining ν conservatively. However, if this same vertex were ever to be moved again, then a revised value of ν will be necessary and it is timely to compute this simultaneously with responding to the query of the movement of p. Note that both the revised star and the revised ν are computed via this propagation, but these are temporarily stored in separate data structures from the original .STL file and the original ν. The function of this propagation is completed upon returning those intended revisions.

The sequence of the propagation, embedded within the control logic of *Update_Vertex*, follows:

1. The *Vertex* instance provides the propagation entity (*Update_Vertex*) with new location as a result of the designer's edit request.

2. *Update_Vertex* calls upon *V_Culled_STL*, to determine an updated value for ν for the edited vertex, which is then passed back to *Update_Vertex*

3. *Update_Vertex* then computes the revised value for ν.

The resulting ADAM generated source code for the *Update_Vertex* propagation is given in an Appendix. Note that the propagation is fired by invocation of *Update_Vertex*'s `Trigger` method.

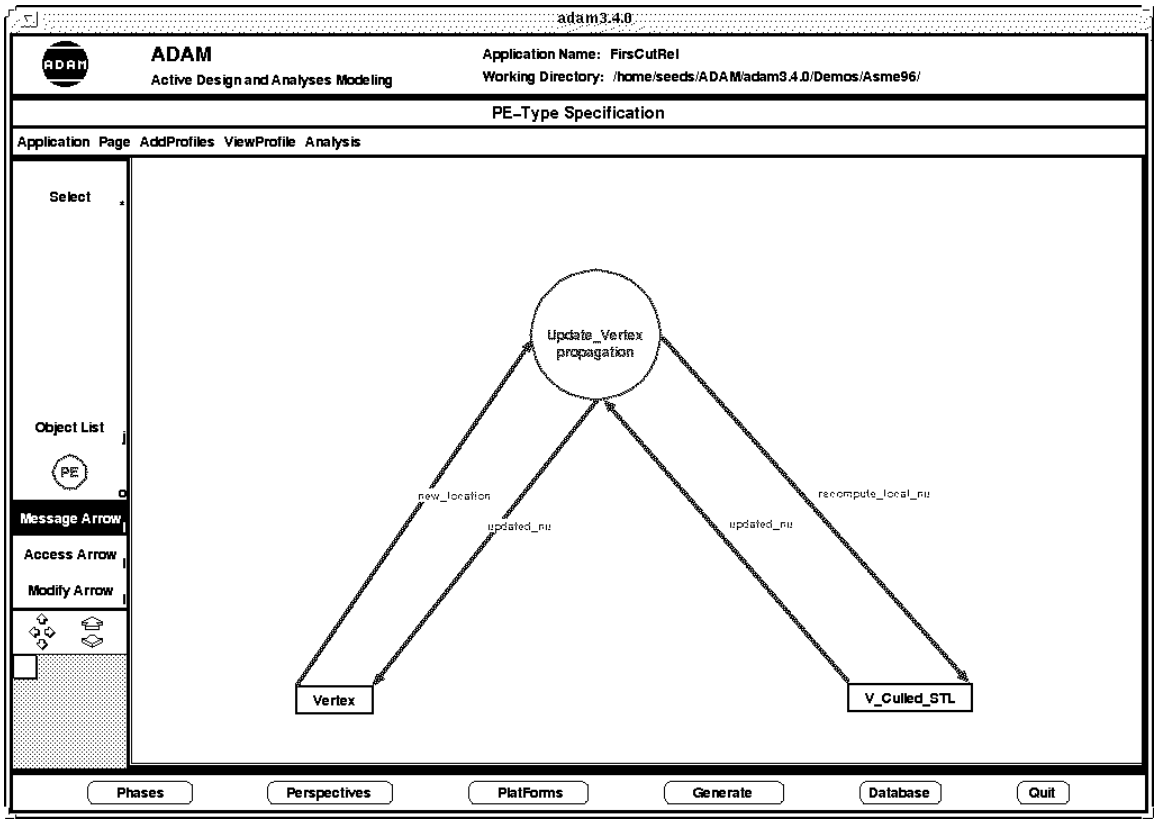

Figure 5: LESS_THAN_NU/2 Propagation

Figure 6 shows an ADAM model of the propagation occurring when the mechanical designer wishes to ascertain whether a vertex can be perturbed in a *specified direction* when it has already been determined that the distance is at least $\nu/2$. Determining whether the movement is allowable in a particular direction is the subject of a separate propagation entity, depicted in Figure 6 as *Probe_With_Vector*. The sequence of this propagation, embedded within the control logic of *Probe_With_Vector*, follows:

1. The *Vertex* instance provides the propagation entity (*Probe_With_Vector*) with its contemplated new location as a result of the designer's edit request.

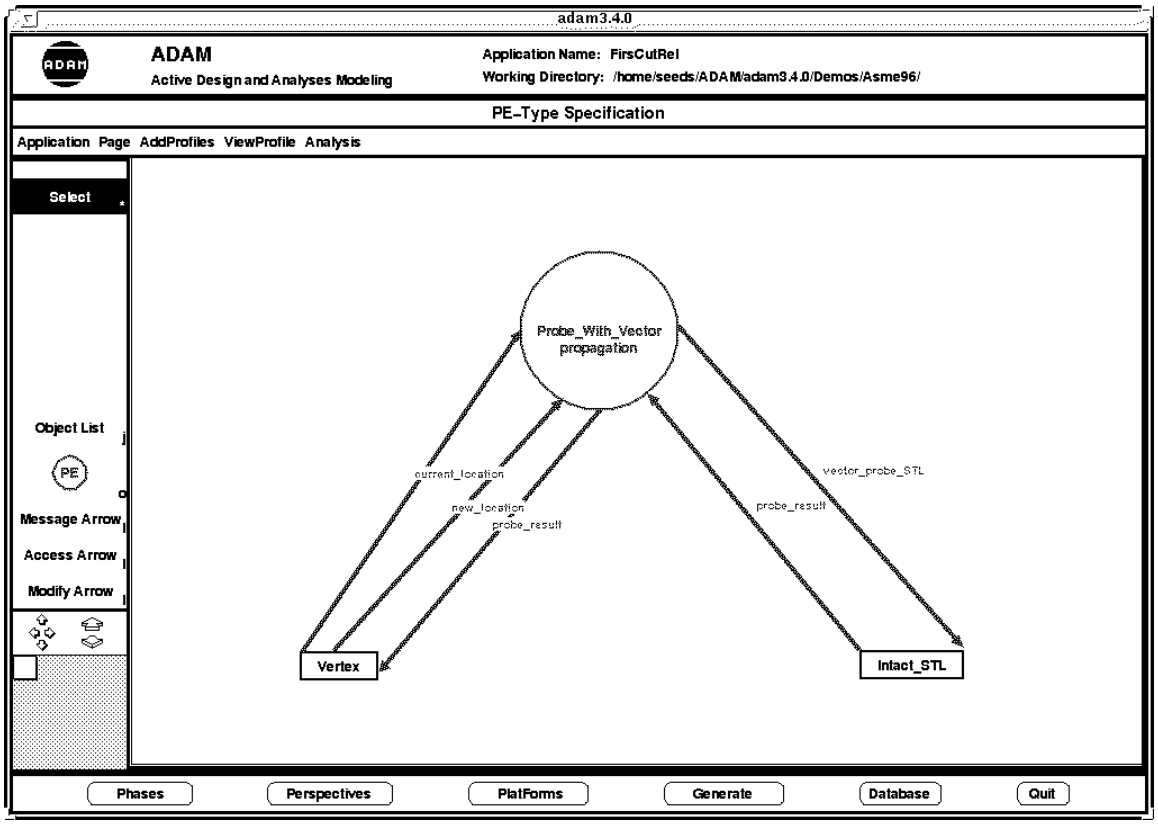

Figure 6: GREATER_THAN_EQUAL_NU/2 Propagation

2. *Probe_With_Vector* computes the vector of movement, based upon the current and new location of the vertex.

3. *Probe_With_Vector* calls upon the *Intact_STL* instance to determine if the edit remains within topological tolerance, given the direction of movement.

4. *Probe_With_Vector* is passed the boolean result of the probe, which is in turn passed back to *Vertex*.

Although not shown in this paper, the *Vertex* instance next either fires another propagation to update its ν (if the probe result was positive) or queries the mechanical designer using the application, requesting input as to whether the edit should be discarded or the topological tolerance violated. For the sake of brevity, the resulting ADAM generated source code is omitted, but its format is similar to that previously presented. Note that the relationship of the Vertex objects

to the Intact_STL object is many to one. These types of relationships pose a challenging new application for propagations, which have previously emphasized one to one relationships.

6 Conclusions and Future Challenges

This work reports upon several software tools (some fully developed and tested, with others at the prototype level) for preserving topological form within an iterative redesign process based upon successive rapid prototypes. These tools include global topological tolerances, more liberal local topological constraints, performance techniques, and object-oriented propagations – all applied to .STL files. A challenge remains to more rigorously test these software tools within a production environment, where it also remains to explore whether the best integration option might be to a CAD/CAM system or directly to the software systems processing the .STL files. For a production environment, it is likely that additional performance enhancements will be needed, particularly for interactive execution on typical .STL files, which might have tens or hundreds of thousands of facets. Software on data sets of this size must usually pay particular attention to the potential for combinatorial explosion, so these performance needs may warrant incorporation of techniques from artificial intelligence.

The supporting mathematical theory and the application discussed are now currently specialized for triangular faces. An alternative file format (known as .SLC) now exists for free-form objects, but has not yet been widely adopted [15]. Further extensions of the theory to free-form objects would be necessary to attempt applications to .SLC files, but these two activities could proceed concurrently.

Note that the topological constraints [1] provide theoretical limits and care must still be taken relative to the usual issues of topological degradation inherent in the use of finite-precision arithmetic. Nonetheless, these topological constraints do provide significant guidance.

The use of topological tolerances to provide interactive design advice is facilitated by integration with the software engineering techniques of object-oriented propagations. This is an intensive new environment for the deployment of propagations and its demands may cause re-design of the very notion of propagations, especially in regard to relations that are many to one, particularly when the set of many items has very large magnitude.

The Federated Modeling Architecture [12] indicates how a geometric-constraint

system can be integrated with conventional CAD modeling systems. This Federated Modeling Architecture appears readily extensible and merits investigation as a means to also integrate topological tolerances.

Acknowledgements: We express our appreciation to L-E. Andersson, T. Bardasz, T. Barnes, R. Disa, R. Garrett, D. C. Gossard, C. M. Hoffmann, J. W. Kane, E. J. Mintel and N.F. Steward for helpful comments and discussions. We thank J. Damon and R. Disa of Pratt and Whitney Aircraft for supplying us with examples of .STL data. However, any remaining errors or omissions are solely the responsibility of the authors.

References

[1] Andersson, L-E., Dorney, S. M., Peters, T. J., Stewart, N. F., *Polyhedral perturbations that preserve topological form*, CAGD, 12 (1995) 785 - 799.

[2] Bing, R. H. *The Geometric Topology of 3-Manifolds*, American Mathematical Society Colloquium Publications, Vol. 40, Providence, RI, 1983.

[3] Bohn, J. H., and Wozny, M. J., *A topology-based approach for shell-closure*, IFIP Transactions, Geometric Modeling for Product Realization, eds., P.R. Wilson, M.J. Wozny, and M.J. Pratt, August 1992, pp. 297 - 320.

[4] Brett, B., *Application of Propagation & Object-Oriented Design to a Feature-Based Mechanical Design Problem*, University of Connecticut, 1995.

[5] Brett, B., Peters, T. J., Demurjian, S. A., Needham, D. M., *Relations between features–an object-oriented industrial prototype*, pre-print.

[6] Boyer, M., Stewart, N. F., *Modelling spaces for toleranced objects*, Int. J. Robot. Res. 10, No. 5, 1991, pp. 570-582.

[7] Boyer, M., Stewart, N. F., *Imperfect-form tolerancing on manifold objects: a metric approach*, Int. J. Robot. Res. 11, No. 5, 1992, pp. 482-490.

[8] Cernosek, J., *Three-dimensional photoelasticity by stress freezing*, Experimental Mechanics, December 1980, pp. 417 - 426.

[9] Dorney, S. M., *Maintaining Equivalent Topological Form in Tolerance Modeling*, Doctoral Dissertation, University of Connecticut, 1994.

[10] Dornfeld, W. H., *Direct dynamic testing of stereolithographic models*, Proceedings of the International Gas Turbine and Aeroengine Congress and Exposition, The Hague, Netherlands, June 13 - 16, 1994, pp. 1 - 6.

[11] Foley, J. D., van Dam, A., Feiner, S. K., and Hughes, J. F., *Computer Graphics, Principles and Practice*, 2nd Edition, Addison-Wesley, Reading, MA, 1990.

[12] Hoffmann, C. M, and Juan, R., *High-level representation for geometric design and analysis*, IFIP Transactions, Geometric Modeling for Product Realization, eds., P.R. Wilson, M.J. Wozny, and M.J. Pratt, August 1992, pp. 129 - 164.

[13] Hoffmann, C. H., personal communication, September 10, 1994.

[14] Hoppe, H. et al, *Mesh optimization*, Computer Graphics ACM SIGGRAPH Proceedings, Aug. 1-6, 1993, Vol. 27, pp. 19 - 25.

[15] Jacobs, P., personal communication, October 3, 1994.

[16] Peters, R. J., and Kastel, D., *Three-dimensional photoelasticity methods complement finite element analysis modeling*, Technology Today, Textron Lycoming, Vol. II, No. 4, January 1988, pp. 1 - 4.

[17] *Workshop on Design Methodologies for Solid Freeform Fabrication*, NSF.

[18] Peters, T. J., Demurjian, S. A., Ting, T.C., and Glovin, S., *Feature-based modeling by object-oriented design with propagation*, Proceedings of 1994 International Conference on Data and Knowledge Systems for Manufacturing and Engineering, Hong Kong, May 1994, pp. 101-110.

[19] Peters, T. J., *Integration of features and design interdependencies within CAD/CAM*, National Science Foundation Design and Manufacturing Grantees Conference, January 4 - 6, 1995.

[20] Requicha, A. A. G., *Toward a theory of geometric tolerancing*, Int. J. Robot. Res., Vol. 2, No. 4, 1983, pp. 45 - 60.

[21] Requicha, A. A. G., *Representation of tolerances in solid modeling: issues and alternative approaches*, in Solid Modeling by Computers, eds., Pickett, M. S. and Boyse, J. W., Plenum, USA, 1984.

[22] Steinchen, W., Kramer, B., and Kupfer, G., *Photoelasticity cuts part-development costs*, Photonics Spectra, May 1994, pp. 157 - 162.

[23] Stewart, N. F., *Sufficient condition for correct topological form in tolerance specification*, Computer-Aided Design 25, No. 1, 1993, pp. 39-48.

[24] Willard, S., *General Topology*, Addison-Wesley, Reading, MA, 1970.

[25] 3-D Systems marketing/sales videotape, 'Stereolithography–the Competitive Edge', Leonard South Productions.

Appendix

```
// Update_Vertex.h:
// Interface for Update_Vertex Propagation Entity

// Other objects involved in the propagation:
class Vertex;
class V_Culled_STL;

class Update_Vertex  {
protected:
 Vertex* Vertex_ptr;
 V_Culled_STL* V_Culled_STL_ptr;
 position_ptr* new_location; // xyz coordinates for a vertex
 nu_ptr* updated_nu;         // updated nu for a vertex

public:
// System defined  CONSTRUCTOR & DESTRUCTOR:
 Update_Vertex();
 ~Update_Vertex();

 void SetVertex_ptr(Vertex_ptr* _Vertex_ptr);
 Vertex_ptr* GetVertex_ptr();

 void SetV_Culled_STL_ptr(V_Culled_STL_ptr* _V_Culled_STL_ptr);
 V_Culled_STL_ptr* GetV_Culled_STL_ptr();

 void Trigger();            //Begins propagation upon invocation
};
```

Sample code for the implementation of the user defined method *Trigger* is also given:

```
// Update_Vertex.c
// Definition for Update_Vertex Propagation Entity

Update_Vertex::Trigger(){
  new_location = GetVertex_ptr()->new_location();
  updated_nu   = GetV_Culled_STL_ptr->recompute_local_nu(new_location);
  GetVertex_ptr()->updated_nu(updated_nu);
};
```